SaltSpreader_R4

A clean, modular Arduino Uno R4 firmware for BLE provisioning + Wi-Fi HTTP control with simulated telemetry and real motor/GNSS interfaces. Networking and control paths are production-grade; telemetry values are synthesized unless GNSS fixes are available.

Build and Run

- Board: Arduino Uno R4 WiFi
- Libraries: ArduinoBLE, ArduinoJson, WiFiS3 (core), EEPROM (core)
- Sketch: SaltSpreader_R4.ino

Configuration Summary

Constant	Value	Notes
kHttpPort	80	HTTP server port
kTcpPort	8080	TCP telemetry port
kWifiConnectTimeoutMs	25000	Wi-Fi association timeout (ms)
kTelemetryIntervalMs	800	Telemetry update cadence (ms)
kStatusLogIntervalMs	5000	Serial status log cadence (ms)
kUseStaticIp	false	Static IP support (when true)
kStaticIp	192.168.137.42	Used only when static IP enabled
kStaticGw	192.168.137.1	Used only when static IP enabled
kStaticSubnet	255.255.255.0	Used only when static IP enabled
kStaticDns	8.8.8.8	Used only when static IP enabled
kBleName	SALTBOT-R4	BLE local name
kBleServiceUuid	0000fd11-0000-1000-8000-00805f9b34fb	BLE service UUID
kBleRxUuid	0000fd11-0001-1000-8000-00805f9b34fb	BLE RX characteristic

Constant	Value	Notes
kBleTxUuid	0000fd11-0002-1000-8000-00805f9b34fb	BLE TX characteristic
kProfileAddr	0	EEPROM base for Wi-Fi profile
kRobotIdAddr	192	EEPROM base for robot ID
kProfileMagic	0x52503431	Magic tag for EEPROM validity

EEPROM Layout

Region	Base Address	Fields	Notes
Wi-Fi profile	0	magic, ssid[33], password[65], hidden, mode	Valid if `magic == kProfileMagic`
Robot ID	192	magic, robot_id[33]	Auto-generated from MAC if blank

Runtime Contracts

HTTP Endpoint Matrix

Method	Path	Purpose	Notes
GET	/status	Telemetry + subsystem state	Always 200 with JSON
GET	/diag	Motor controller diagnostics	Always 200 with JSON
POST	/control	Drive command or legacy command	Returns 200; errors in payload
POST	/mission	Mission and map updates	Returns 200; validates map_lines
GET	/path	Plan preview (first 120 steps)	503 if pathfinding unavailable
GET/POST	/motor_test	Runs motor test pattern	503 if motor unavailable

Base URL

The app connects to the Base URL returned by BLE provisioning and `/status`.

GET /status

Returns 200 OK with JSON including:

- `robot_id`
- `server_url`
- `current_pose.lat`, `current_pose.lng`
- `battery_pct`, `salt_pct`
- `robot_mode`
- `subsystems` object with class health/status strings
- `drive` object with last command state

Status payload fields (selected):

Field	Type	Source
type	string	Constant `STATUS`
robot_id	string	`ensureRobotId()` (EEPROM or MAC derived)
server_url	string	Built from local IP + port
timestamp_ms	uint32	`millis()` at build time
battery_pct	int	Telemetry model or external source
salt_pct	int	Telemetry model
gps_fix	string	`NONE` or `GPS`
gps_confidence	float	Derived from horizontal accuracy
current_pose.lat/lng	float	GNSS or synthetic drift
current_pose.heading_deg	float	GNSS or synthetic drift
subsystems.*	bool/string	Wi-Fi/BLE/TCP + module status strings
drive.left_motor/right_motor	int16	Last received drive command
drive.salt_spreader_*	bool	Last received spreader state

Field	Type	Source
nav.state	string	`NO_MAP`, `PLANNING`, `READY`, `ERROR`
nav.plan_length	uint32	Number of planned steps
nav.obstacle	string	`NONE/LEFT/CENTER/RIGHT/UNKNOWN`
nav.drive_hint	string	`NONE/LEFT/RIGHT/STOP`

GET /diag

Returns 200 OK with diagnostics for the last command, motor controller, and test pattern state.

Key fields:

Field	Type	Notes
drive_last_received.*	object	Mirror of `DriveState` + `last_received_ms`
motor.status	string	`INIT`, `READY`, `ACTIVE`, or `TEST`
motor.last_applied_*	int/bool	Last command applied to hardware
motor.last_cmd_ms	uint32	Time of last command change
motor.last_apply_ms	uint32	Time of last hardware update
motor.test_active	bool	Test pattern enabled
motor.test_phase	uint8	1=forward, 2=reverse, 3=stop
motor.controller_version	string	Basicmicro version string (if valid)
motor.controller_error	uint32	Basicmicro error code (if valid)
motor.main_batt_tenths_v	uint16	Battery in 0.1V units (if valid)

POST /control

Supports two modes:

1. Drive mode (new app contract):

```
{
  "type": "DRIVE_CONTROL",
  "left_motor": -10000..10000,
  "right_motor": -10000..10000,
  "salt_spreader_motor": true|false,
  "salt_spreader_servo": true|false
}
```

Response:

```
{ "success": true, "message": "Drive command accepted" }
```

Drive control validation:

Field	Type	Range	Notes
left_motor	int	-10000..10000	Required for DRIVE_CONTROL
right_motor	int	-10000..10000	Required for DRIVE_CONTROL
salt_spreader_motor	bool	true/false	Optional, defaults false
salt_spreader_servo	bool	true/false	Optional, defaults false

If `type` is missing or not `DRIVE_CONTROL`, the handler falls back to `command` (legacy mode).

2. Legacy command mode:

```
{ "command": "START" }
```

POST /mission

Accepts polygon missions and responds with `success` and `mission_id`.

Navigation updates are optional and can be included in the same payload:

```
{
  "mission_id": "lot-001",
  "map_lines": [
    "..####..",
    "..#..#..",
    "..#..#..",
    "..####.."
  ],
  "resolution_m": 0.25,
  "origin_gps": { "lat_e7": 420000000, "lon_e7": -930000000 },
  "resolution_e7": 1,
  "start": { "x": 1, "y": 1 },
  "goal": { "x": 6, "y": 2 },
  "autonomy": false
}
```

Notes:

- `map_lines` uses `.` for free space and `#` for walls. All lines must be equal length.
- `resolution_m` is the map grid resolution in meters per cell.
- `origin_gps` sets the top-left origin in `lat_e7`/`lon_e7`.
- `resolution_e7` sets degrees per cell for GPS mapping (default `1`).
- `start_gps`/`goal_gps` can be provided instead of `start`/`goal` to use GPS mapping.
- `start`/`goal` may also be `[x, y]` arrays.
- `autonomy` enables obstacle-driven steering overrides (see Obstacle Signal).

Mission update fields (parsed by firmware):

Field	**Type**	**Notes**
mission_id	string	Optional echo in response
map_lines	string[]	Either top-level or map.lines
map.resolution_m	float	Overrides `resolution_m`
origin_gps	object	`lat_e7`, `lon_e7` (top-left)
resolution_e7	int	Defaults to 1 if missing
start	[x,y] or object	Grid coordinates
goal	[x,y] or object	Grid coordinates
start_gps	object	`lat_e7`, `lon_e7` mapped to grid

Field	Type	Notes
goal_gps	object	`lat_e7`, `lon_e7` mapped to grid
autonomy	bool	Enables obstacle override logic

GET /path

Returns a preview of the current planned coverage path. Response includes:

- `plan_length` total steps
- `steps` list (first 120 entries)
- `truncated` boolean when more data is available

Action codes in `steps`:

- `0` START
- `1` SALTING
- `2` TRANSIT
- `3` RETURN_TO_BASE
- `4` COMPLETE

Action code table:

Code	Label	Meaning
0	START	Begin plan at base cell
1	SALTING	Coverage pass while spreading
2	TRANSIT	Travel to next coverage region
3	RETURN	Return to base for refill/recharge
4	COMPLETE	Final step at base

TCP telemetry also pushes `PATH` messages with:

- `plan_length`, `plan_seq`, `truncated`
- `steps` (first 120 entries, same action codes)

BLE Provisioning Expectations

- BLE name prefix: `SALTBOT`
- Service UUID: `0000fd11-0000-1000-8000-00805f9b34fb`
- RX char UUID: `0000fd11-0001-1000-8000-00805f9b34fb`
- TX char UUID: `0000fd11-0002-1000-8000-00805f9b34fb`
- Messages are newline-delimited JSON.
- BLE sends `ROBOT_HELLO` on connect and `WIFI_RESULT` on provisioning completion.

BLE Message Types

Direction	type	Required Fields	Notes
App -> Robot	PHONE_NETWORK_CONTEXT	wifi_type	Must arrive before WIFI_PROFILE
App -> Robot	WIFI_PROFILE	ssid, password, mode, hidden	mode: DIRECT_JOIN, HOTSPOT, ROBOT_AP, BLUETOOTH_ONLY
Robot -> App	ROBOT_HELLO	robot_id, fw_version, capabilities	Sent on BLE connect
Robot -> App	PROVISIONING_STATUS	step	Steps like ASSOCIATING, AUTHENTICATING, DHCP
Robot -> App	WIFI_RESULT	status, ip, server_port, tcp_port	On success
Robot -> App	WIFI_RESULT	status, reason_code	On failure
Robot -> App	LOG	message	Buffered logs flushed on connect

Provisioning failure reasons (non-exhaustive):

reason_code	Cause
SSID_NOT_FOUND	Network not visible
BAD_PASSWORD	Auth failure
DHCP_TIMEOUT	IP not acquired
ENTERPRISE_UNSUPPORTED	Enterprise network not supported

reason_code	Cause
AUTH_METHOD_UNSUPPORTED	BLUETOOTH_ONLY mode not supported

TCP Telemetry Stream

TCP uses newline-delimited JSON on port `kTcpPort`.

type	Direction	Payload	Notes
ROBOT_HELLO	Robot -> App	robot_id, tcp_port	Sent on connect
TELEMETRY	Robot -> App	See `/status` telemetry fields	Periodic (`kTelemetryIntervalMs`)
PATH	Robot -> App	plan_length, plan_seq, steps[], truncated	Sent on plan changes

Module Responsibilities and Cross-Talk

Networking

- `ble_provisioning`: Handles BLE provisioning state, accepts phone network context and Wi-Fi profile, sends provisioning status and result.
- `wifi_manager`: Performs Wi-Fi connection and failure reason mapping.
- `http_server`: Exposes REST endpoints and integrates with drive and telemetry state.
- `tcp_stream`: Streams telemetry and `ROBOT_HELLO` over TCP.

Sensors

- `telemetry`: Periodic pose/battery/salt/fault synthesis for UI testing; overridden by GNSS fixes when available.
- `gps`: GNSS module (SoftwareSerial + u-blox). Auto-retry init, publishes fix, heading, accuracy, and fix type.

Control

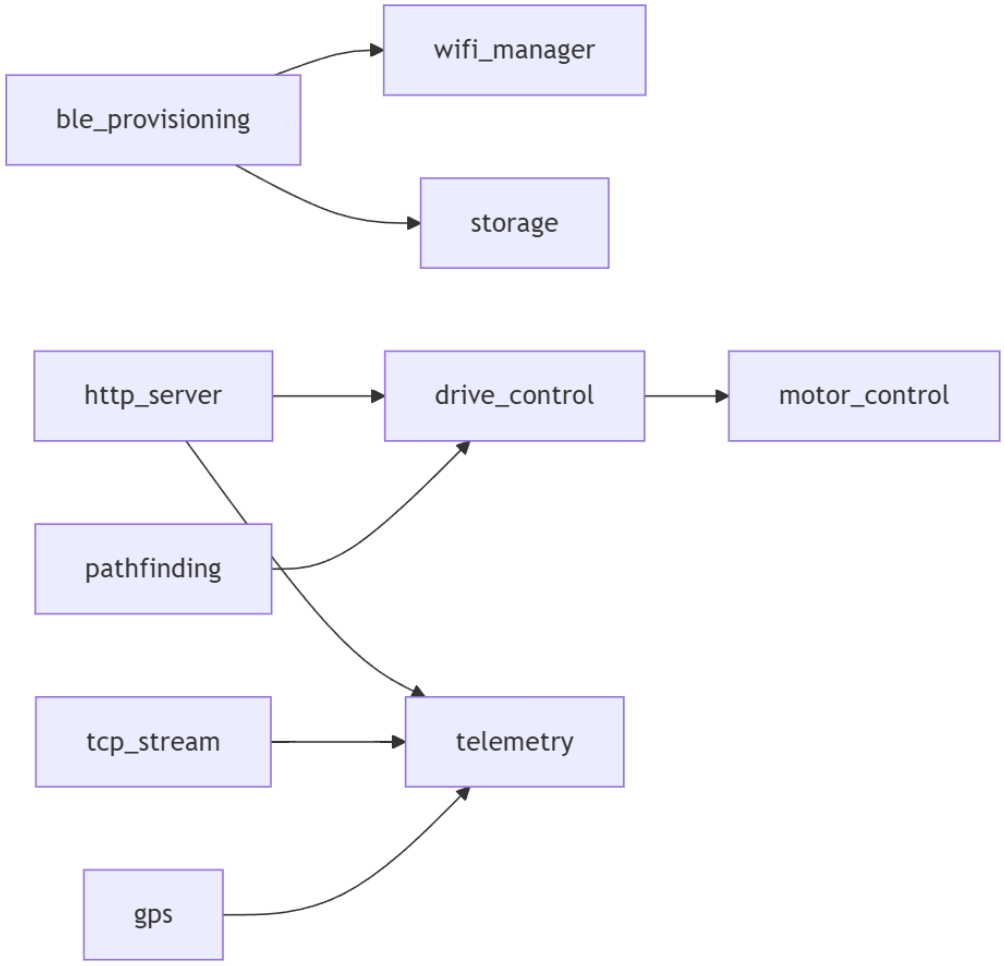
- `drive_control`: Stores last drive command (persistent booleans).
- `motor_control`: Orchestrates the Basicmicro motor controller and the salt spreader, supports a built-in test pattern and last-command tracking.

- `pathfinding`: Coverage planning + obstacle hinting (map/mission driven) with optional autonomy overrides.

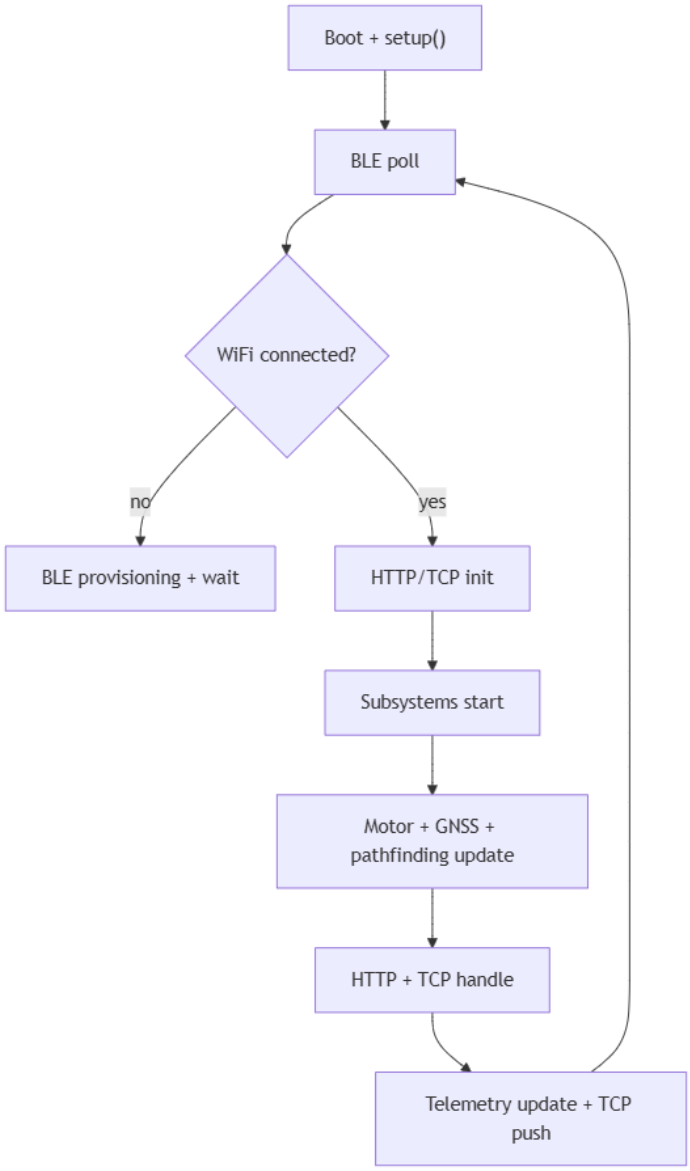
Cross-Talk

- `ble_provisioning` -> `wifi_manager` (connect attempt)
- `ble_provisioning` -> `storage` (save profile)
- `http_server` -> `drive_control` (apply commands)
- `http_server` -> `telemetry` (status response)
- `tcp_stream` -> `telemetry` (telemetry push)
- `motor_control` <- `drive_control` (consumes latest command)
- `gps` -> `telemetry` (updates pose when a fix is present)
- `pathfinding` -> `drive_control` (autonomy overrides when enabled and app input is stale)

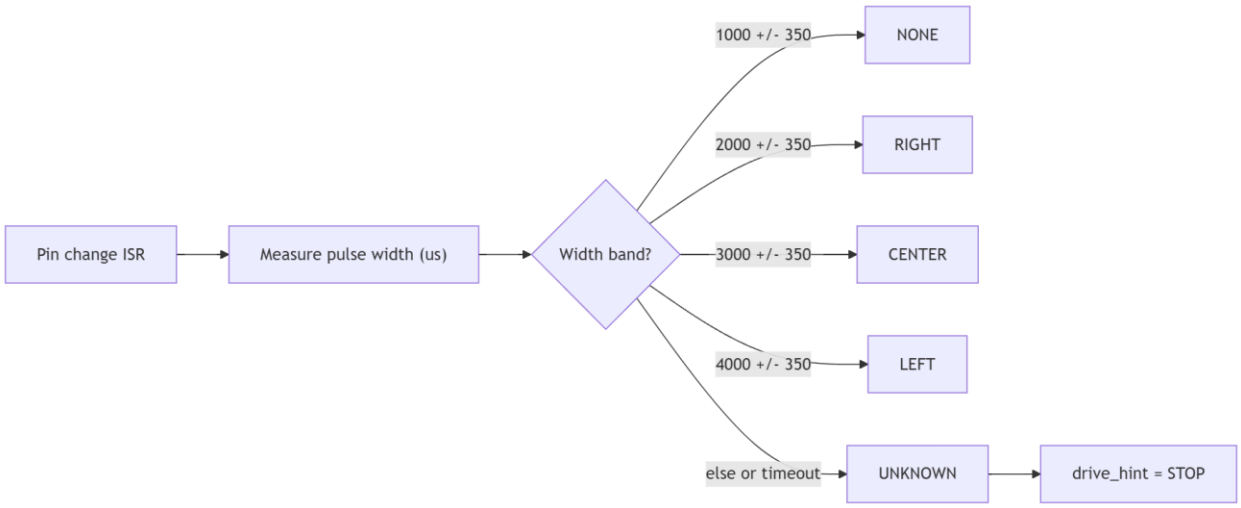
Diagram



Main Loop Flowchart

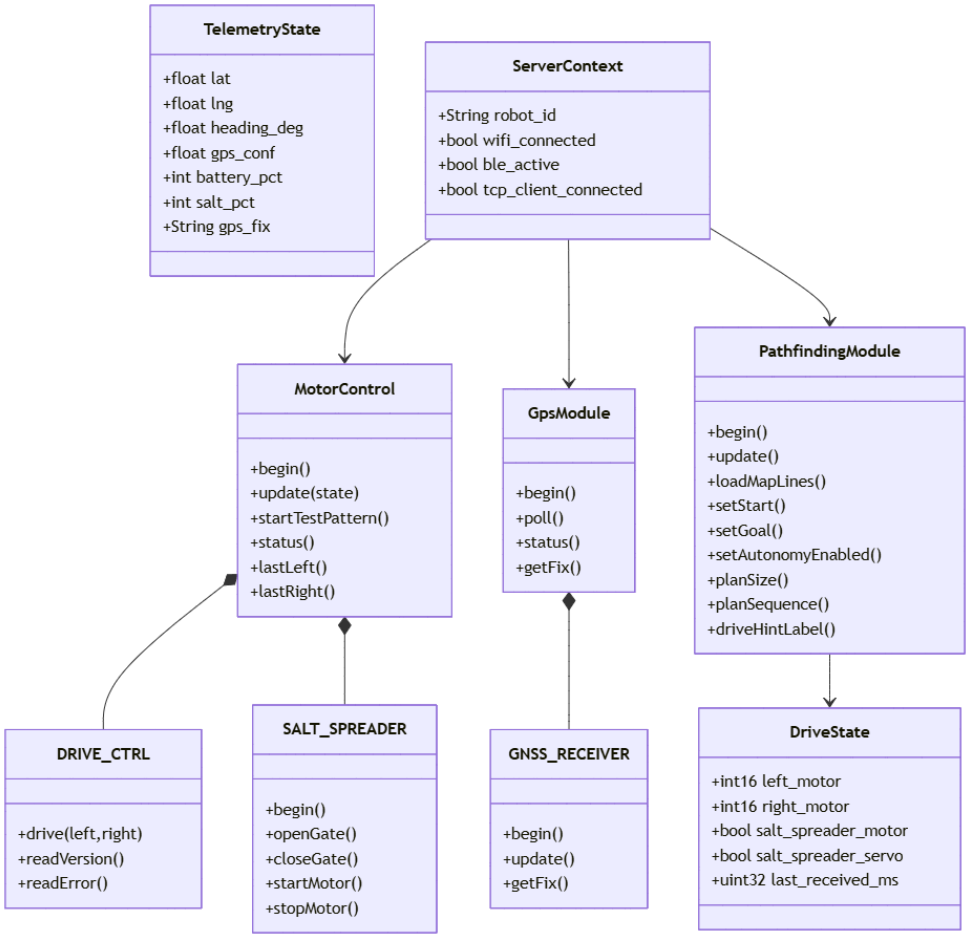


Obstacle Decode Flowchart

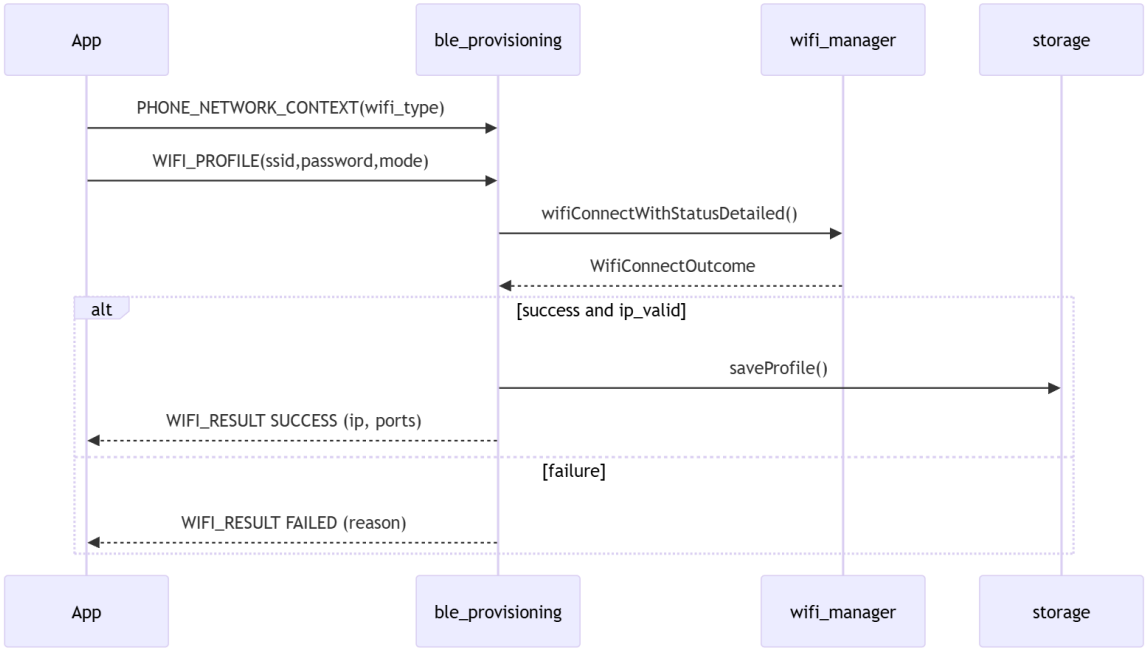


UML Diagrams

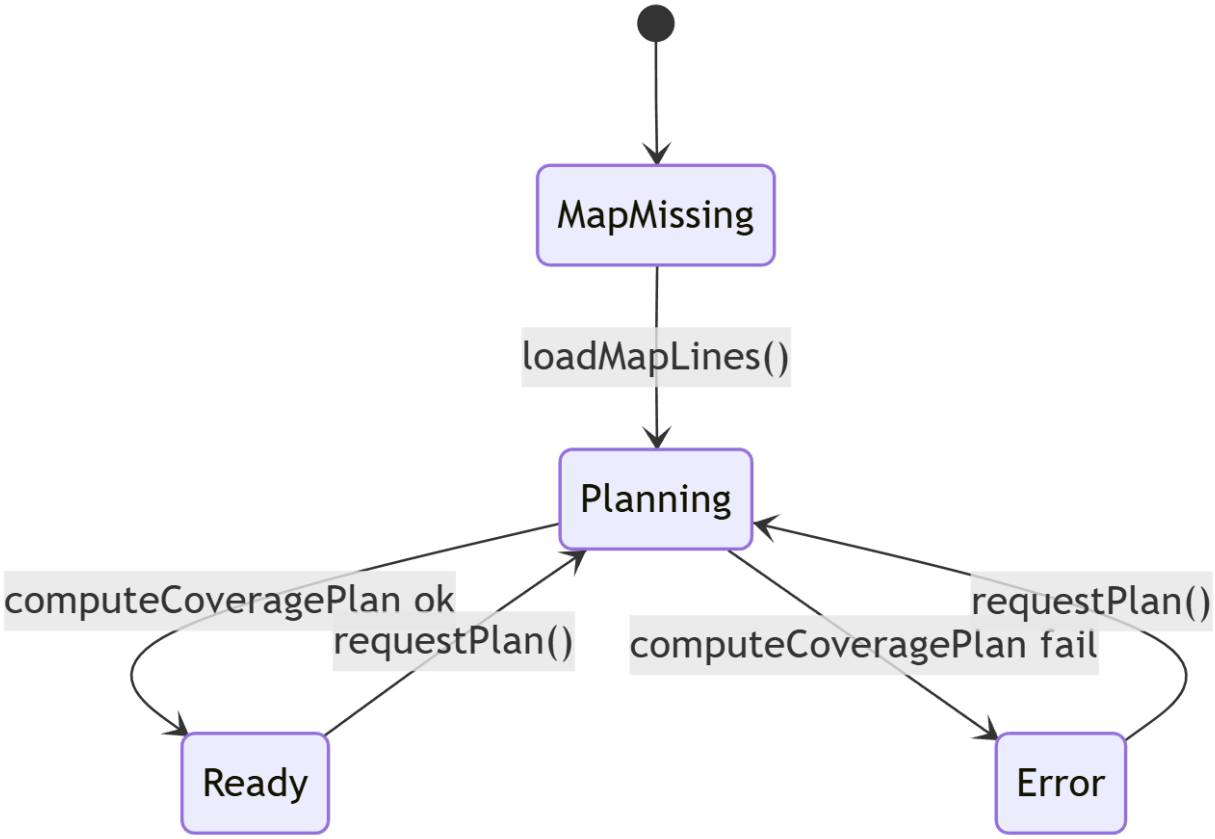
Class model (key runtime objects):



BLE provisioning sequence:



Pathfinding state model:



Sensor and Actuator Integration Notes

Integrated from `sensors_testing_copy-1` (as-is behavior, with Wi-Fi/NTRIP disabled):

- Basicmicro motor controller (`driveControl.h`)
- Salt spreader servo + motor (`saltSpreader.h`)
- GNSS receiver (`gnssReceiver.h`)

Additional library dependencies:

- Basicmicro (packet serial motor controller)
- Servo (core)
- SparkFun u-blox GNSS v3
- Base64 (only if enabling NTRIP later)

Conflicts and deviations:

- Original GNSS code used `WiFi.h` and NTRIP credentials; Uno R4 uses `WiFiS3`, so NTRIP is disabled by default (`GNSS_ENABLE_NTRIP=0`).
- Original GNSS code included `secrets.h`; this clean project omits it to avoid Wi-Fi credentials in firmware.
- GNSS uses `SoftwareSerial` on pins 2/3; if R4 core does not support `SoftwareSerial`, switch to a hardware UART and update `pin_assignments.h` + `gps.*`.

Motor control details:

- `driveControl.h` wraps Basicmicro packet serial, maps -10000..10000 commands into PPS.
- `motor_control.cpp` gates test overrides, logs changes, and applies spreader state.

Telemetry details:

- `telemetry.cpp` updates at `kTelemetryIntervalMs`, decays battery/salt, and injects faults.
- `applyGnssFix()` overrides pose and sets `gps_fix`/confidence based on accuracy.

Synthetic telemetry model (when no GNSS fix):

Behavior	Value	Notes
Position drift	1e-5 deg / update	Adds to lat, subtracts from lon
Heading drift	+0.2 deg / update	Wraps at 360
Battery decay	-1% every 600000 ms	Only if > 0
Salt decay	-1% every 600000 ms	Only if > 0
Fault toggle	150000..300000 ms	Toggles `fault_active`

Navigation + Obstacle Integration

Map Format (map_lines)

- `map_lines` is an array of strings (rows).
- Each row must be equal length.
- `.` = free cell, `#` = blocked cell.
- Origin is the top-left (`x=0`, `y=0`).
- `resolution_m` sets meters-per-cell for the coverage planner.

GPS Mapping (1e-7 degree precision)

When mapping GPS to grid coordinates, the firmware uses integer degrees in 1e-7 units:

- `origin_gps`: `lat_e7`/`lon_e7` define the top-left grid origin.
- `resolution_e7`: degrees per grid cell (default `1`, meaning 1×10^{-7} deg per cell).
- `start_gps`/`goal_gps`: `lat_e7`/`lon_e7` are converted into grid `x`/`y` using:
  - `x = (lon_e7 - origin_lon_e7) / resolution_e7`
  - `y = (origin_lat_e7 - lat_e7) / resolution_e7`

Obstacle Signal (ESP32S3 -> R4)

The ESP32S3 sends obstacle states to the R4 using a single digital pin with pulse widths:

Pulse Width (us)	Meaning
1000 +/- 350	NONE
2000 +/- 350	RIGHT

Pulse Width (us)	Meaning
3000 +/- 350	CENTER
4000 +/- 350	LEFT

Implementation details:

- Pin: `OBSTACLE_SIGNAL_PIN` in `pin_assignments.h`.
- Interrupt: `attachInterrupt(..., CHANGE)` captures pulse width with `micros()`.
- Timeout: If no valid pulse within ~800 ms, obstacle state becomes `UNKNOWN`.
- `drive_hint` in `/status` exposes LEFT/RIGHT/STOP guidance.
- If `autonomy` is enabled and the app has not sent a recent drive command (~800 ms), the firmware will issue a steering override based on the obstacle hint.

Pathfinding Output

- `nav` object in `/status` includes `state`, map dimensions, `plan_length`, `obstacle`, and `drive_hint`.
- `GET /path` returns a preview of the planned coverage path (first 120 steps).
- TCP telemetry pushes `PATH` messages whenever the plan updates.

Pathfinding Parameters

Parameter	Value	Notes
resolution_m_	0.2	Default meters per cell if not provided
resolution_e7_	1	Default GPS degrees per cell
max_salt_lbs_	8.0	Salt budget before return-to-base
max_battery_ah_	10.0	Battery budget before return-to-base
volts_per_cell_	0.02	Battery drain per step
spread_x_cells_	2	Coverage spray half-width in X
spread_y_cells_	3	Coverage spray half-height in Y
robot_x_cells_	2	Robot footprint in X (cells)
robot_y_cells_	2	Robot footprint in Y (cells)
kAutonomyStaleMs	800	Autonomy kicks in if app is silent

Parameter	Value	Notes
kAutonomyTurnSpeed	4000	Turn-in-place speed (cmd units)
kAutonomyForwardSpeed	5000	Forward speed when clear

Mathematical Notes

Drive Command to Motor PPS

The Basicmicro controller expects pulses-per-second (PPS). The firmware maps normalized commands as:

$$pps = (speed/10000) * 1320$$

Where `speed` is in [-10000, 10000] and 1320 is the configured top speed in PPS.

Command mapping examples:

speed	PPS	Meaning
-10000	-1320	Full reverse
-5000	-660	Half reverse
0	0	Stop
5000	660	Half forward
10000	1320	Full forward

GPS to Grid Mapping

GPS coordinates are stored as 1e-7 degrees (`lat_e7`, `lon_e7`). Grid conversion uses:

$x = (lon_e7 - origin_lon_e7) / resolution_e7$

$y = (origin_lat_e7 - lat_e7) / resolution_e7$

This preserves integer math while allowing meter-scale resolutions.

Distance and Heading (GEO_POINT)

The equirectangular distance approximation and bearing are:

$$d = R\sqrt{(\Delta\lambda cos\,((\phi_1 + \phi_2)/2))^2 + (\Delta\phi)^2}$$

$$\theta = \text{atan2}\,(sin\,(\Delta\lambda)cos\,(\phi_2), cos\,(\phi_1)sin\,(\phi_2) - sin\,(\phi_1)cos\,(\phi_2)cos\,(\Delta\lambda))$$

Where $R = 6371000$ meters, ϕ is latitude (rad), and λ is longitude (rad).

GNSS Accuracy to Confidence

Telemetry confidence is derived from horizontal accuracy:

$conf = \operatorname{clamp}(1 - accuracy_m / 5, 0.1, 1.0)$

Pathfinding Cost Model

The A* planner uses 8-connectivity with integer costs:

- Orthogonal step cost: 10
- Diagonal step cost: 14 (approx $10\sqrt{2}$)

Salt and Battery Consumption Heuristic

The coverage planner estimates per-cell salt use as:

$lbs_per_cell = \max(0.001, resolution_m^2 * 0.005)$

Battery reserve threshold:

$safeBattery = volts_per_cell * (width + height) * 0.5$

Obstacle Pulse Classification

Pulse widths are compared with a tolerance band. Example for NONE:

$width_us \in [1000 - 350, 1000 + 350]$

RIGHT/CENTER/LEFT use 2000/3000/4000 us with the same +/- 350 us tolerance.

Real-Time Architecture

Loop Phases (SaltSpreader_R4_Clean.ino)

1. BLE provisioning poll (non-blocking)
2. Wi-Fi server bring-up (if newly connected)
3. Subsystem updates (motor, GPS, pathfinding)
4. HTTP + TCP servicing (network I/O)
5. Telemetry update + TCP push

Flags & State

- `g_wifiConnected`: Wi-Fi link state (set by BLE provisioning or boot connect)
- `g_serversStarted`: HTTP/TCP servers initialized
- `g_subsystemsStarted`: motor/GPS/pathfinding ready
- `PathfindingModule::plan_dirty_`: re-plan on new map/mission/obstacle
- `PathfindingModule::obstacle_state_`: decoded obstacle state from ISR
- `PathfindingModule::drive_hint_`: LEFT/RIGHT/STOP decision
- `PathfindingModule::plan_sequence_`: incremented on each new plan (TCP push trigger)

BLE provisioning state (in `ble_provisioning.*`):

- `s_bleProvisioning`: BLE stack active / advertising
- `s_bleConnected`: phone connected over BLE
- `s_phoneContextSeen`: `PHONE_NETWORK_CONTEXT` received
- `s_wifiType`: phone network type (DIRECT/ENTERPRISE/etc)

TCP telemetry state (in `tcp_stream.*`):

- `s_clientActive`: TCP client connected
- `TcpStreamStatus::client_connected`: exposed via `/status`

Interrupts

- Obstacle input uses a CHANGE interrupt on `OBSTACLE_SIGNAL_PIN`.
- ISR captures pulse width only; decoding occurs in `PathfindingModule::update()`.
- No other interrupts are configured by this firmware.

Timing Constants

- `kWifiConnectTimeoutMs`: overall Wi-Fi connect timeout
- `kTelemetryIntervalMs`: telemetry update cadence
- `kStatusLogIntervalMs`: Serial health log cadence
- Obstacle pulse timeout: ~800 ms without a valid pulse -> `UNKNOWN`

Timing table:

Timer	Value	Purpose
kWifiConnectTimeoutMs	25000 ms	Wi-Fi association timeout
kTelemetryIntervalMs	800 ms	Telemetry update cadence
kStatusLogIntervalMs	5000 ms	Serial status cadence
GNSS retry	5000 ms	Re-init GNSS if not detected
Obstacle timeout	800 ms	No valid pulse -> UNKNOWN
Autonomy stale	800 ms	No recent drive cmd -> autonomy
Motor test forward	2000 ms	Test pattern phase 1
Motor test reverse	2000 ms	Test pattern phase 2
Motor test stop	500 ms	Test pattern phase 3

Pin Assignments

Subsystem	Pin	Notes
Serial Monitor	USB	`Serial` @ 115200

Subsystem	Pin	Notes
Motor Controller	Serial1	`driveCtrlSerial` @ 38400, addr 0x80
GNSS RX	D2	`GNSS_SERIAL_RX_PIN`
GNSS TX	D3	`GNSS_SERIAL_TX_PIN`
Salt Motor	D4	`SALT_MOTOR_PIN`
Salt Servo	D5	`SALT_SERVO_PIN`
Bumper Left	D6	`BUMPER_LEFT_PIN`
Bumper Right	D7	`BUMPER_RIGHT_PIN`
CV Left	D8	`CV_LEFT_PIN`
CV Center	D9	`CV_CENTER_PIN`
CV Right	D10	`CV_RIGHT_PIN`
Obstacle Signal	D7	`OBSTACLE_SIGNAL_PIN`

Note: D7 is assigned to both `OBSTACLE_SIGNAL_PIN` and `BUMPER_RIGHT_PIN`. Reassign one of them if you need both inputs.

Redundancy and Fault Detection

- `telemetry` injects random faults and tracks `fault_count` for UI testing.
- `subsystems` section in `/status` is the primary health surface; each class reports its own status.
- Cross-verification (planned):
  - `gps` vs `pathfinding` for route consistency
  - `motor_control` feedback vs `drive_control` command
  - `ble_provisioning` vs `wifi_manager` for provisioning outcomes

Debugging and Observability

- Serial logs all BLE, HTTP, and TCP traffic.
- `/status` includes `server_url` and per-class status strings for the app monitor page.
- `/status` includes `nav` details for pathfinding and obstacle status.

Files

- `SaltSpreader_R4_Clean.ino`: Main wiring
- `config.*`: Constants and UUIDs
- `storage.*`: EEPROM profile + robot ID
- `wifi_manager.*`: Wi-Fi connect helper
- `ble_provisioning.*`: BLE provisioning flow
- `http_server.*`: REST server
- `tcp_stream.*`: TCP telemetry stream
- `telemetry.*`: Mock telemetry
- `drive_control.*`: Drive command state
- `motor_control.*`: Motor + spreader orchestration, test pattern, status tracking
- `gps.*`: GNSS interface + fix handling
- `pathfinding.*`: Coverage planning + obstacle integration