Frontend — Basics

Location: src/ and src/routes/

The frontend is a SvelteKit TypeScript application. It uses Svelte 5's runes-based reactivity (*$effect, $props, $state*) throughout. All communication with the Rust backend goes through Tauri's *invoke()* function. Incoming rover data arrives as Tauri events listened to with *listen().*

routes/+layout.svelte — Navigation Bar

The layout wraps every page in the application. It renders the persistent navigation bar at the top and then the current page's content via `{@render children()}`.

Navigation bar

The navbar contains the following controls, always visible regardless of which route is active:

Task dropdown — lists the four task routes (Science, Navigation, Maintenance, Probing). Selecting one navigates to that route and updates the displayed task name.

Drive Control Mode dropdown — toggles between Manual and Automatic drive. Calls `set_state` with `DriveManual` to sync the mode to the backend.

Arm Control Mode dropdown — toggles between Manual and Automatic arm control. Calls `set_state` with `ArmManual` to sync the mode to the backend.

Start / Pause / Resume Task button — controls a task timer. Displays the elapsed time next to the label. The button label cycles through `▶ Start Task` → `❚❚ Pause <task>` → `▶ Resume <task>` depending on state.

Mode icon — a centred icon that shows either a driving icon or an arm icon depending on the current `pickup_mode` state. Polled from the backend every 250ms.

END TASK button — stops the timer and saves a task result file. Prompts the operator for confirmation before ending. The saved JSON file includes the task name, completion time, finish timestamp, and all attached samples. The filename is auto-generated as `{NNNN}_{task_name}.json` where `NNNN` is an incrementing zero-padded number.

Settings / Home icons — navigation shortcuts in the top right.

Task file naming

When a task ends, the layout reads the existing task files and finds the highest existing prefix number for that task type, then increments it. This ensures task files are always uniquely numbered in order (e.g. `0000_science.json`, `0001_science.json`).

Camera health

`initCameraHealthListener()` from `state.svelte.js` is called directly in the layout's `<script>` (outside `onMount`) so it is initialised as early as possible.

routes/ — Pages

/ — Dashboard (Home)

File: `routes/+page.svelte`

The main operator screen. A CSS grid layout combining six components into a single view.

Components: `Double_Video`, `Map`, `NavigationPlan`, `TaskCompletion`, `IMU`, `Model` (3D scene).

/navigation — Navigation

File: `routes/navigation/+page.svelte`

The navigation task screen. Focuses on map-based rover guidance.

Layout: a 2×2 grid with the map top-left, navigation plan and IMU bottom-left, double video feed top-right, and costmap bottom-right.

Components: `Map`, `NavigationPlan`, `IMU`, `DoubleVideo` (depth + front cameras), `Costmap`.

/maintenance — Maintenance

File: `routes/maintenance/+page.svelte`

The maintenance task screen. Focuses on the rover arm and diagnostics.

Layout: left side has two video feeds (arm camera and depth camera) each with a side panel for arm feedback and arm position data. Right side shows the maintenance task list.

Components: `Video` (arm camera), `Video` (depth camera), `MaintenanceTasks`.

/probing — Probing

File: `routes/probing/+page.svelte`

The probing task screen. Used when the rover is searching for and picking up probes.

Layout: map top-left, interest locations and probes list top-right, double video bottom-left, pickup mode toggle and IMU bottom-right.

The page reads and syncs `pickup_mode` from the backend on mount, and toggles it via the Drive/Pick-up Mode button, which also switches camera 1 between the depth camera and arm camera.

Components: `Map`, `InterestLocations`, `Probes`, `DoubleVideo`, `IMU`.

/science — Science

File: `routes/science/+page.svelte`

The science task screen. Used when the rover is collecting and analysing rock or soil samples.

Layout: left side has a map and double video feed with small side panels for locations of interest and the pickup mode toggle. Right side is fully occupied by the `SamplingLocations` component.

Like the probing route, it syncs `pickup_mode` on mount and switches camera 1 between depth and arm camera accordingly.

Components: `Map`, `DoubleVideo`, `SamplingLocations`.

/settings — Settings

File: `routes/settings/+page.svelte`

The settings page. Contains developer and diagnostic utilities, not used during rover operation.

Diagnostic controls: Ping Rust, Clear Cache, Ping UDP, Ping GPS, Ping pH — these call the corresponding backend commands and are used to verify the Tauri bridge and UDP connection are working.

Dummy stream controls: Start/Stop dummy IMU stream, Start/Stop full dummy stream — start or stop the backend simulator for development without hardware.

File management: List, view, and delete files from the `tasks/`, `images/`, and `maps/` storage directories. Files can be clicked to view their contents inline.

Snapshot: Saves a single JPEG frame from port 5000 as a test image.

Model debug: Calls `debug_resource_dir` and prints the result to the console — useful for verifying model bundling in a production build.

IP check: Fetches the operator laptop's public IP from `api.ipify.org` and displays it. Useful for network configuration when connecting to the rover.

types.ts — Types

Shared TypeScript types used across the frontend.

`Sample` — represents a single science sample collected by the rover. Contains location name, coordinates, before/after image paths, measurement, weight, and a set of boolean `_check` flags tracking which fields have been filled in. The `all_check` flag is true when all required fields are complete.

`Waypoint` — a map waypoint with an `id`, `lat`, and `lng`. Used for start point, end point, and the waypoints list in the map store.

`Probe` — a probe location with an `id`, `lat`, `lng`, and `timestamp`. Used in the probing task to record where soil probes were taken.

state.svelte.js — Global State

`state.svelte.js` is the single source of truth for shared reactive state that needs to be accessible across multiple components and routes. Currently it manages the three camera objects.

Camera objects

Three camera state objects are exported as Svelte 5 `$state` runes:

Export	Port	Camera
`depthCamera`	5000	Depth / front-facing camera
`frontCamera`	5001	Secondary front camera
`armCamera`	5002	Arm-mounted camera

Each object has three fields: `name` (display string), `port` (full `http://localhost:PORT` URL used as the `<img>` src), and `stale` (boolean set to `true` when the backend reports no frames for 2+ seconds).

Camera health listener — initCameraHealthListener()

This function is called once from `+layout.svelte` on app startup. It listens for the `camera-feed-status` Tauri event emitted by the GStreamer health watcher in the backend, and updates the `stale` flag on the matching camera object. A 500ms startup delay is included to ensure the Tauri bridge is ready before the listener is attached.

Components that display video can read the `stale` flag to show a warning overlay when a feed is lost.