Getting Started

Prerequistites

> MacOS installation steps were not tested, proceed at your own risk!

Rust

Install Rust via rustup — the official Rust toolchain installer.

All platforms: go to https://rustup.rs and follow the instructions, or run:

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

After installation, restart your terminal and verify:

```
rustc --version
cargo --version
```

Bun

Bun is the JavaScript runtime and package manager used for the frontend.

> Using npm or yarn will **not** work with the current setup, there are scripts that specifically call for bun.

Linux / macOS:

```
curl -fsSL https://bun.sh/install | bash
```

Windows: download the installer from https://bun.sh

Verify:

```
bun --version
```

Tauri CLI prerequisites

Tauri requires some OS-level dependencies in addition to Rust.

Linux (Ubuntu/Debian):

```
sudo apt update
sudo apt install libwebkit2gtk-4.1-dev libgtk-3-dev \
  libayatana-appindicator3-dev librsvg2-dev patchelf
```

Windows: install the Microsoft C++ Build Tools and WebView2 (usually already present on Windows 10/11).

macOS:

```
xcode-select --install
```

GStreamer

GStreamer handles video decoding and streaming. Version **1.22.x** is required to match the Rust crate versions (`gstreamer = "0.22"` maps to GStreamer 1.22).

Linux (Ubuntu/Debian):

```
sudo apt install \
  libgstreamer1.0-dev \
  libgstreamer-plugins-base1.0-dev \
  libgstreamer-plugins-bad1.0-dev \
  gstreamer1.0-plugins-base \
  gstreamer1.0-plugins-good \
  gstreamer1.0-plugins-bad \
  gstreamer1.0-plugins-ugly \
  gstreamer1.0-libav \
  gstreamer1.0-tools
```

Verify:

```
gst-launch-1.0 --version
```

Windows:

1. Download the **MSVC 64-bit** installer for GStreamer 1.22.x from https://gstreamer.freedesktop.org/download/
2. Download both the **runtime** and **development** installers
3. Install both to the default path: `C:\gstreamer\1.0\msvc_x86_64\`
4. Add the GStreamer bin directory to your system PATH:

```
C:\gstreamer\1.0\msvc_x86_64\bin
```

Verify in a new terminal:

```
gst-launch-1.0 --version
```

> The `GST_PLUGIN_PATH` environment variable is set automatically by the app at runtime (in `lib.rs`), so you do not need to set it manually.

macOS:

```
brew install gstreamer gst-plugins-base gst-plugins-good \
  gst-plugins-bad gst-plugins-ugly gst-libav
```

Cloning the Repository

> **Before** cloning the repository, install the following tools on your machine.

The repository contains a Git submodule for the protobuf definitions. You must clone with `--recurse-submodules` or the `src-tauri/proto/` directory will be empty and the build will fail.

```
git clone --recurse-submodules https://github.com/RoboTeamTwente/erc-software-basestation.git
cd erc-software-basestation
```

If you already cloned without the flag, initialise the submodule manually:

```
git submodule update --init --recursive
```

Keeping the submodule up to date

When pulling changes that include submodule updates, always run:

```
git pull
git submodule update --recursive
```

If the proto definitions change and your build starts failing with protobuf-related errors, this is almost always the cause.

Installing Frontend Dependencies

```
bun install
```

This reads `package.json` and installs all SvelteKit, Threlte, and other frontend dependencies into `node_modules/`. Run this once after cloning and again whenever `package.json` changes.

Common Operations

Running in Development

```
bun run tauri dev
```

This command does the following in parallel:

- Starts the Vite/SvelteKit dev server on `http://localhost:1420`
- Compiles the Rust backend (first run takes several minutes)
- Opens the Tauri application window

The frontend supports hot module replacement — changes to `.svelte` and `.ts` files appear immediately without restarting. Rust changes require a recompile, which Tauri handles automatically but takes longer.

> **First build warning:** the initial `cargo build` downloads and compiles all Rust dependencies including GStreamer bindings. This can take 5–15 minutes depending on your machine. Subsequent builds are fast due to incremental compilation.

Building for Production

> One of Tauri's dependencies, libc (the C standard library), is forward compatible but not backward compatible

The demo laptop has an old verison of linux so you need to use docker to build the app if you want to use it in it.
The first time you build you have to build docker first:

```
sudo docker build -t tauri-ubuntu2204 .
```

For building the app in linux the subsequent times for the demo laptop use the command:

```
sudo docker run --rm \
    -v $(pwd):/app \
    -v tauri-cargo-cache:/root/.cargo/registry \
    tauri-ubuntu2204
```

>

> You don't have to docker build every time, just the first time and if you change anything in the dockerfile

Testing Without Rover Hardware

You do not need a physical rover to develop or test the UI. The backend includes a full simulator.

Video Feeds

Fake camera

`fake_camera_gstreamer/` — a GStreamer-based test source that sends H.264 RTP streams on the expected UDP ports (4500, 4501, 4502). It can be run from by running the following command from `erc-software-basestation/fake_camera_gstreamer/`

```
cargo run --bin fake_camera_gstreamer
```

Stream from Webcam

To test the video feed with your webcam open a terminal and use the following command.

Linux

```
gst-launch-1.0 v4l2src ! videoconvert ! x264enc tune=zerolatency bitrate=800 speed-preset=ultrafast ! rtph264pay ! udpsink host=127.0.0.1 port=4500
```

Windows

```
gst-launch-1.0 ksvideosrc ! videoconvert ! x264enc tune=zerolatency bitrate=800 speed-preset=ultrafast ! rtph264pay ! udpsink host=127.0.0.1 port=4500
```

Dummy data streams (rover telemetry)

Once the app is running, go to `/settings` and use the simulator controls:

- **Start dummy general stream** — starts the full multi-stream simulator sending fake IMU, GPS, arm, drive, and sensor data to the app over UDP. Use this to test all telemetry UI at once.
- **Start dummy IMU stream** — starts an IMU-only stream with no jitter or packet loss. Use this for isolated IMU component testing.
- **Stop dummy general/IMU stream** — stops whichever simulator is running.

The simulator runs inside the Rust backend so it works regardless of whether a rover is connected.

Connecting to the Rover

> **TODO**

The base station listens for incoming UDP packets on the address set on lib.rs, must set static address in laptop settings

Common Issues

Build fails with "No valid proto files found under components/" The proto submodule is not initialised. Run `git submodule update --init --recursive`.

GStreamer errors at startup on Linux Verify the plugin path exists: `/usr/lib/x86_64-linux-gnu/gstreamer-1.0`. If your system uses a different architecture or distro the path in `lib.rs` may need updating.

GStreamer errors at startup on Windows Verify GStreamer is installed to exactly `C:\gstreamer\1.0\msvc_x86_64\` and that the `bin` directory is in your PATH.

Video feeds show "SIGNAL LOST" No camera is sending data on the expected UDP ports. Start `fake_camera_gstreamer` or connect the rover to get video.

White/blank window on launch The SvelteKit dev server may not have started yet. Wait a few seconds and the window should load. If it persists, check the terminal for Vite errors.

Model shows "Failed to load 3D model" In development, models are loaded from `src-tauri/models/`. Verify that `chibiRover.glb` (or your model file) exists there. In production builds, run `debug_resource_dir` from the Settings page to check where Tauri is looking.

"[Error] Unhandled Promise Rejection: ReferenceError: Cannot access uninitialized variable." Seems like a circular import error, check the name of the files, some require odd names such as `detected_objects.svelte.ts`. If you remove the .svelte it will error out oddly.