CONVERTING AN 07 CAR TO A REMOTE CONTROLLED EV USING OPEN SOURCE SOFTWARE

Marc Lainez

The team

Marc

Loïc

Thibault

2013: Spin42

↳ 2016-2023: Ibanity (Sold it in 2017)

↳ 2024: Taking a sabbatical break, playing with cars...

Why?

The transport industry is rather siloed and closed

The software reliability and safety of vehicles is rather "opaque" and not necessarily reassuring

There is no real "aftermarket software" for cars

Parts from different brands do not work together, too much vendor lock-in

What does it mean to “upgrade” a vehicle?

- Bringing it on par with environmental requirements
  - Engine swap
  - EV Retrofit
- Adding the features we expect in today’s cars
  - Infotainment system
  - Assisted driving/autonomous driving
  - Remote control

Small disclaimer

NONE OF WHAT YOU WILL SEE IS ROAD CERTIFIED 😱

Why not try to upgrade a 2007 Polo
using only open source software ?

The donor car...

2007 Polo Bluemotion

What upgrades have we done to it?

Mechanical work

- Renovate both drivetrain
- Change brake pads
- Modify body and chassis to support the new motor and battery packs
- Install Nissan Leaf motor and fabricate connection pieces
- Swap the brake system with a Tesla iBooster module
- Install a modified steering column
- 3D printed countless pieces
- Add (many) new wires...
- ...

Hardware and Software work

- Add a custom made infotainment touchscreen
- Create the interface for the Nissan motor (ignition, throttle, RPMs, gear selector, ...)
- Control the steering pump independently
- Create the interface with the battery management system and charger (wip)
- Build new brains for the car in order to make all new parts communicate together
- Create Mavlink bridge
- Create ROS2 bridge (wip)
- Build basic perception layer for future autonomous experiments (wip)
- ...

What does it look like now?

All thanks to open source projects

Elixir

All non-arduino components

Nerves

Firmware builder based on buildroot

All non-arduino components

Vue.js

Frontend web

VMS

Flutter

Frontend embedded

Infotainment

Phoenix

Backend API

VMS + Infotainment

Why Elixir

- Dynamic, functional language running on the BEAM (Erlang VM)
- Made to build scalable and highly available systems
- Uses pattern matching, making parsing bytes or messages quite straightforward

```
<<id::little-integer-size(16),
  _::binary-size(2),
  byte_number::little-integer-size(8),
  _::binary-size(3),
  raw_data::binary-size(byte_number),
  _::binary>> = raw_frame
```

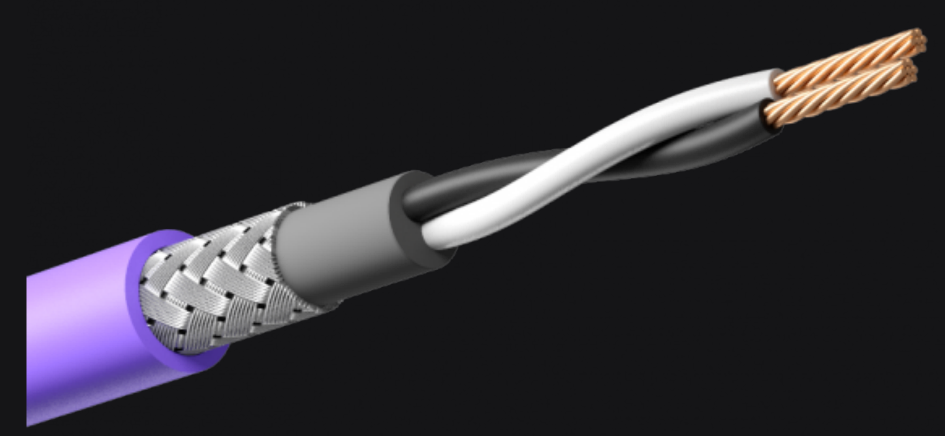
Why Nerves

- Quickly build firmware based on buildroot
- Boots straight into the Beam (Erlang VM)
- Packages Elixir code and runs it on several off-the-shelves boards
- Deals with partition redundancy, OTA updates, and all the firmware development/deployment cycle
- Leverages the power and flexibility of buildroot

https://nerves-project.org/

Understanding the car’s language

CAN communication bus



CAN bus (Controller Area Network) is where all car components talk together

Standard protocol in automotive, aeronautics, industrial machinery, ...

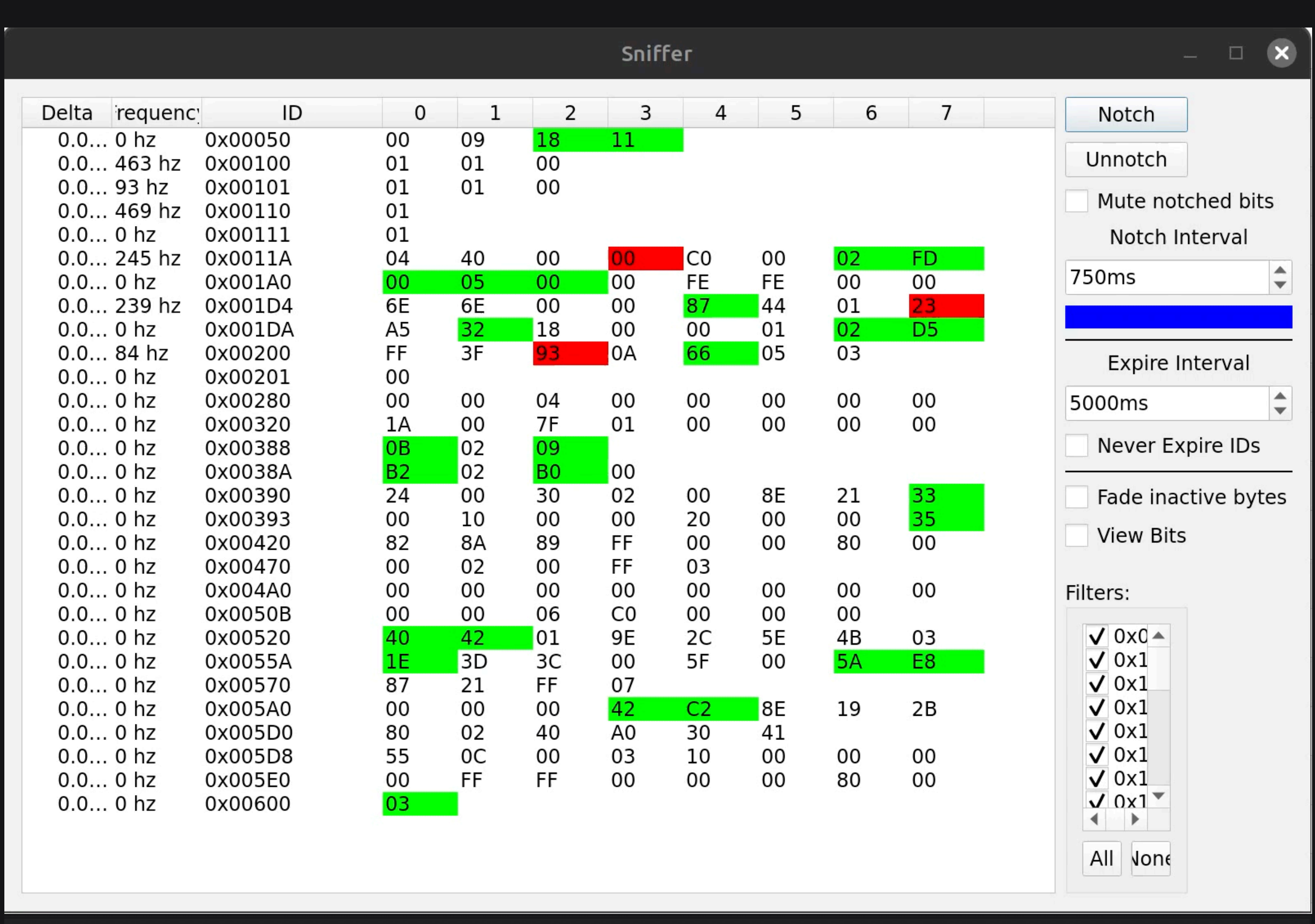
Although CAN is standard, the messages you transfer through it are not

CAN communication bus

Data exchanged on CAN is represented as a series of bytes

A “frame” with a specific ID is published periodically on the CAN

https://github.com/commaai/opendbc

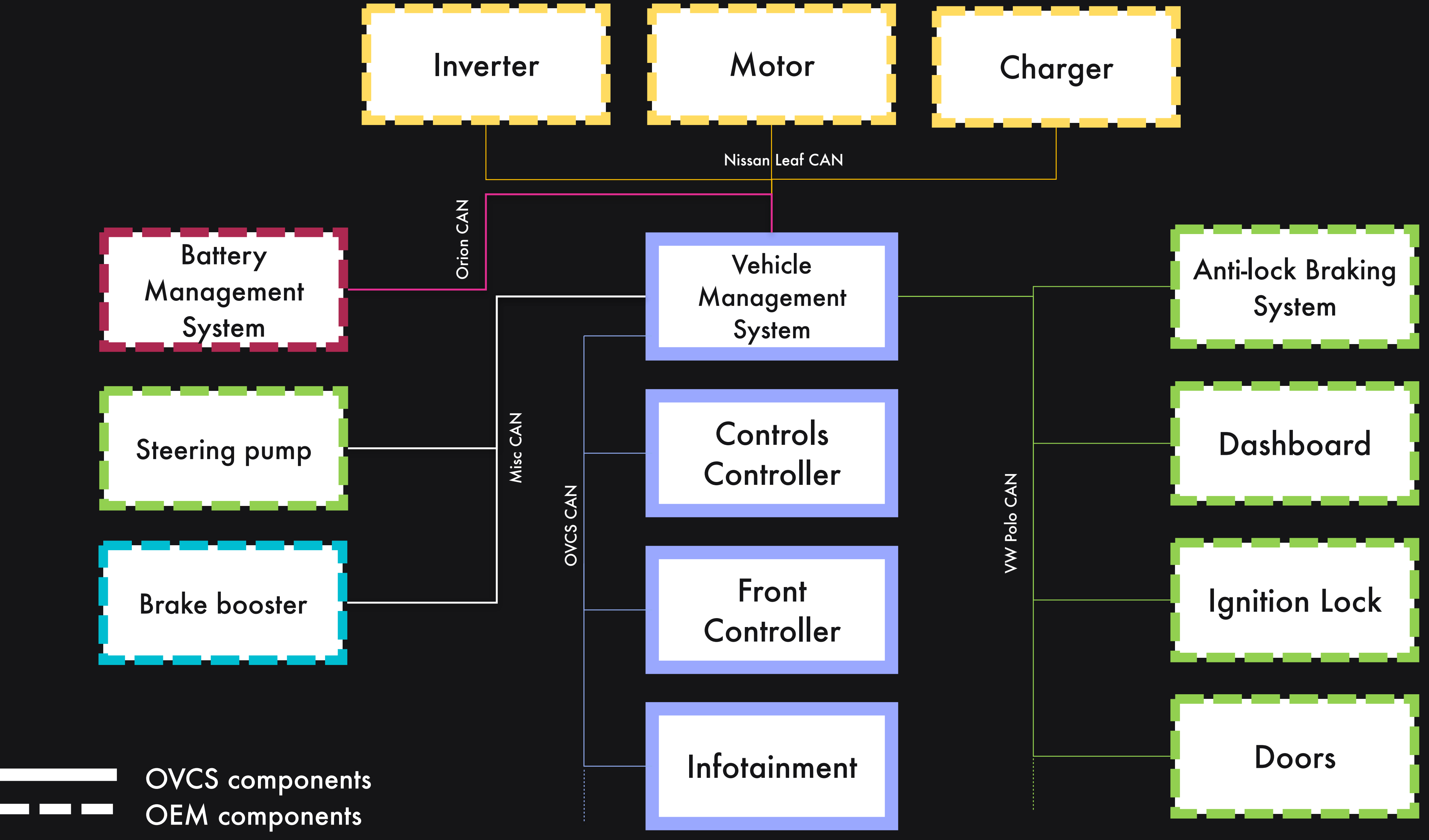


Cantastic

- Building our own (open source) Elixir CAN library
- On top of Erlang sockets
- Uses yaml instead of dbc files
- Takes advantage of the functional power of Elixir

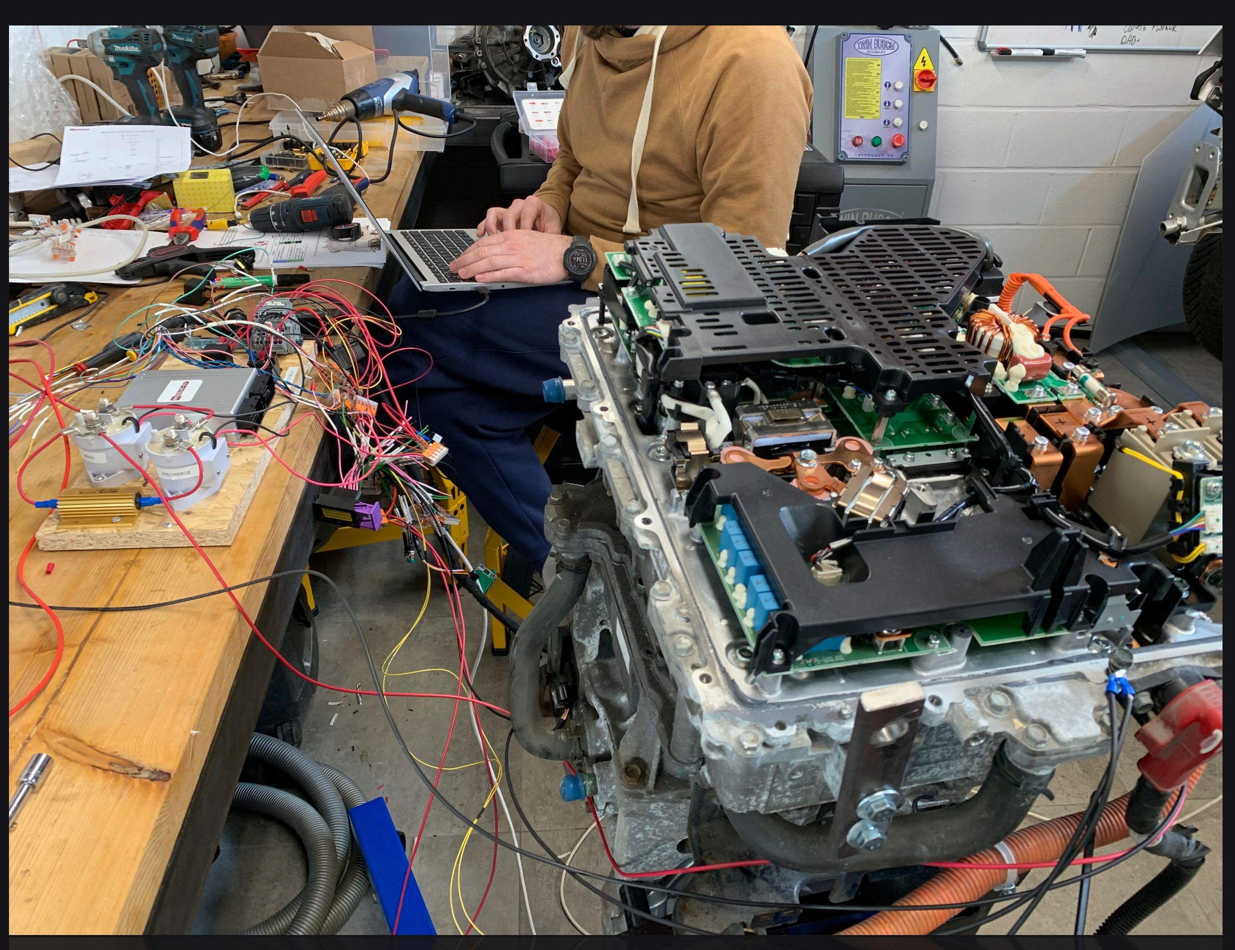
```yaml
name: wheels_speed
id: 0x4A0
signals:
  - name: front_left_wheel_speed
    unit: km/h
    value_start: 0
    value_length: 16
    scale: "0.005836"
  - name: front_right_wheel_speed
    unit: km/h
    value_start: 16
    value_length: 16
    scale: "0.005836"
  - name: rear_left_wheel_speed
    unit: km/h
    value_start: 32
    value_length: 16
    scale: "0.005836"
```

https://github.com/open-vehicle-control-system/cantastic


Inverter
Motor
Charger
Nissan Leaf CAN
Orion CAN
Battery Management System
Vehicle Management System
Anti-lock Braking System
Steering pump
Misc CAN
OVCS CAN
Controls Controller
VW Polo CAN
Dashboard
Brake booster
Front Controller
Ignition Lock
Infotainment
Doors
OVCS components
OEM components

Getting the leaf motor to spin

Reverse engineering the Leaf motor



We needed to find the right CAN messages to power up the motor

We used DBC files we could find online to figure them out

Information found mostly on auto enthusiast forums and by observing the CAN

Using the Polo gas pedal to control the motor

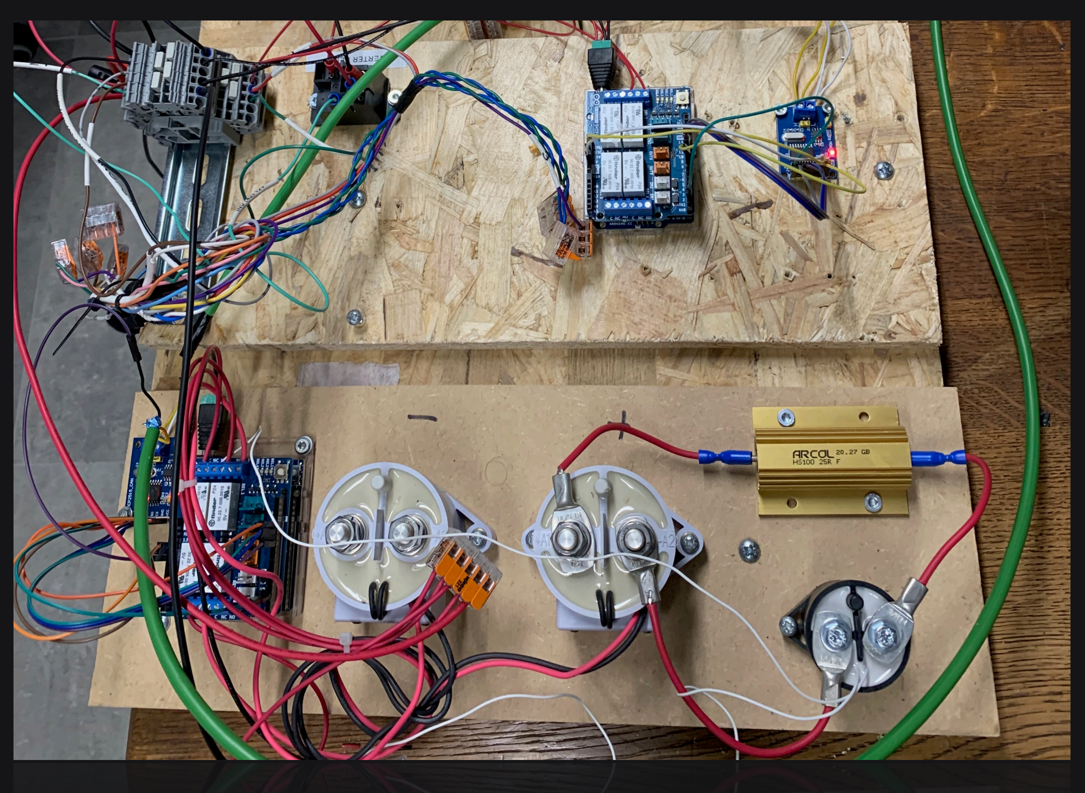
Connecting the pedal to the CAN

The pedal is a simple potentiometer (2 actually...)

You can see two signals, one is used to give the pedal position and the other one is a control value

We connected it to an arduino with a CAN module over SPI

Deal with motor contactors



Several relays need to be activated in a specific order

Adding relays to arduino was quite straightforward

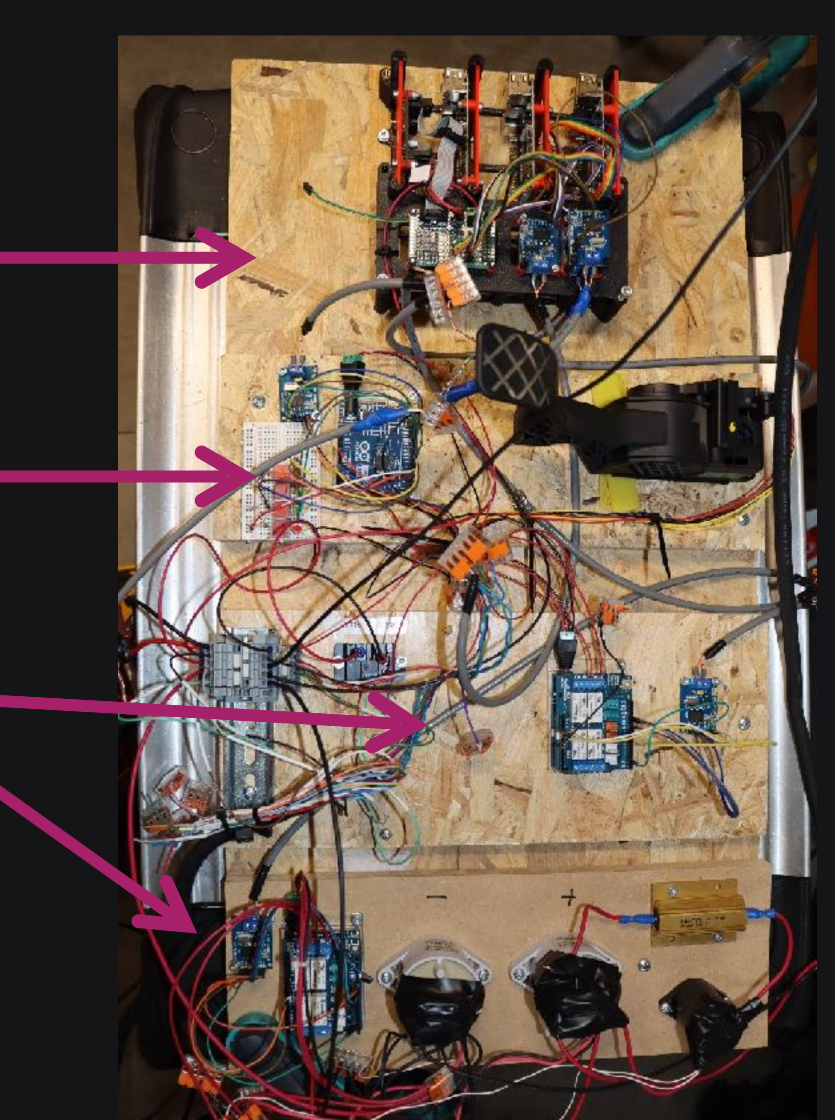
The arduinos are connected to the CAN network with CAN SPI modules

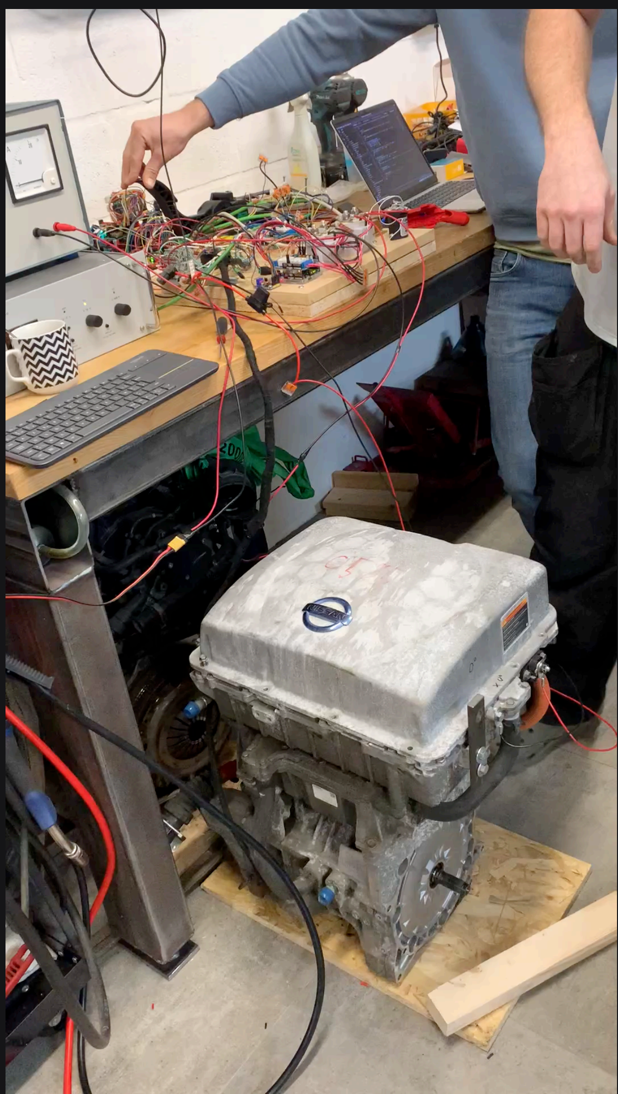
The first "end-to-end" prototype

Vehicle Management System

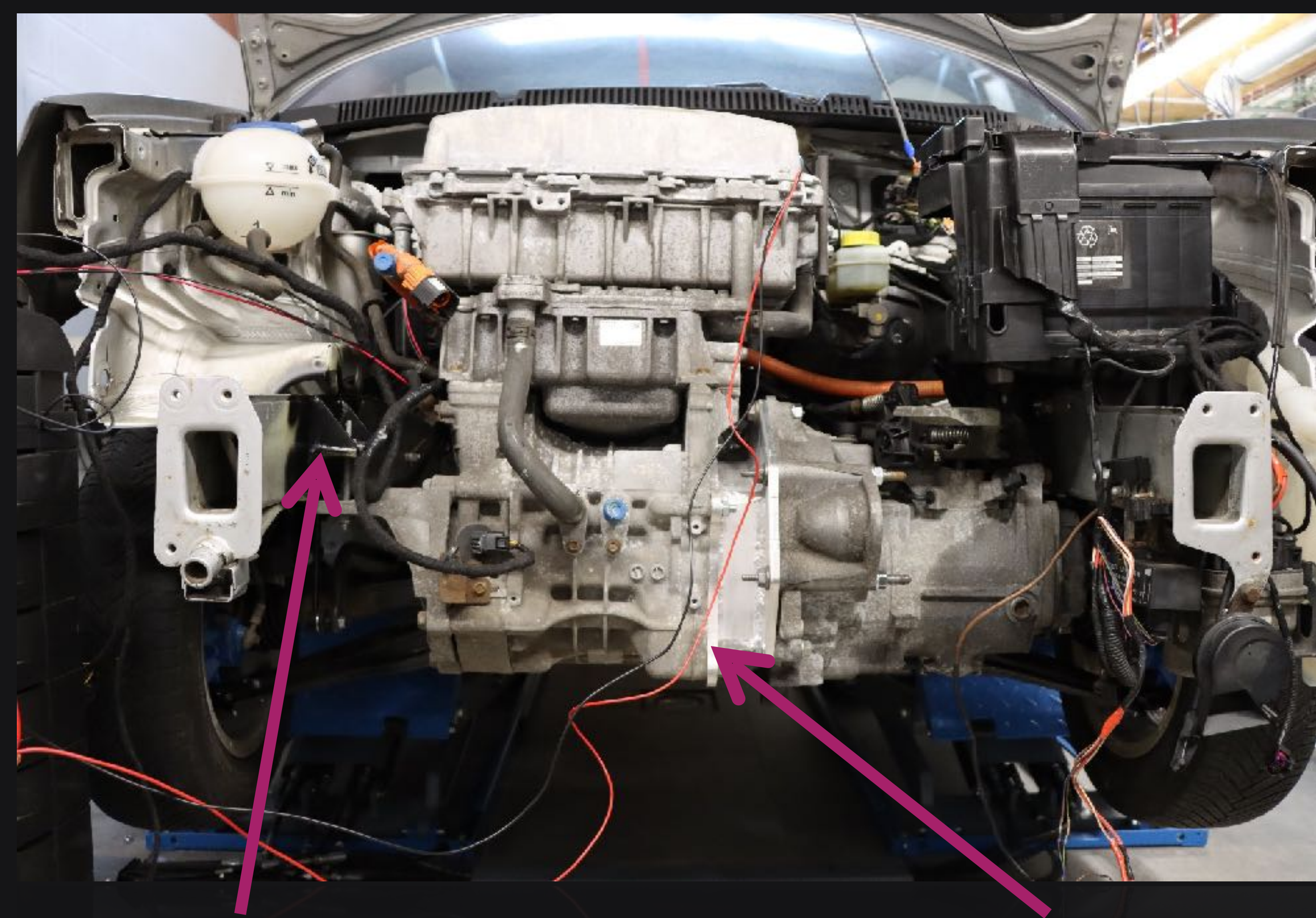
Car controls controller

Contactors controller





Putting the motor in the Polo

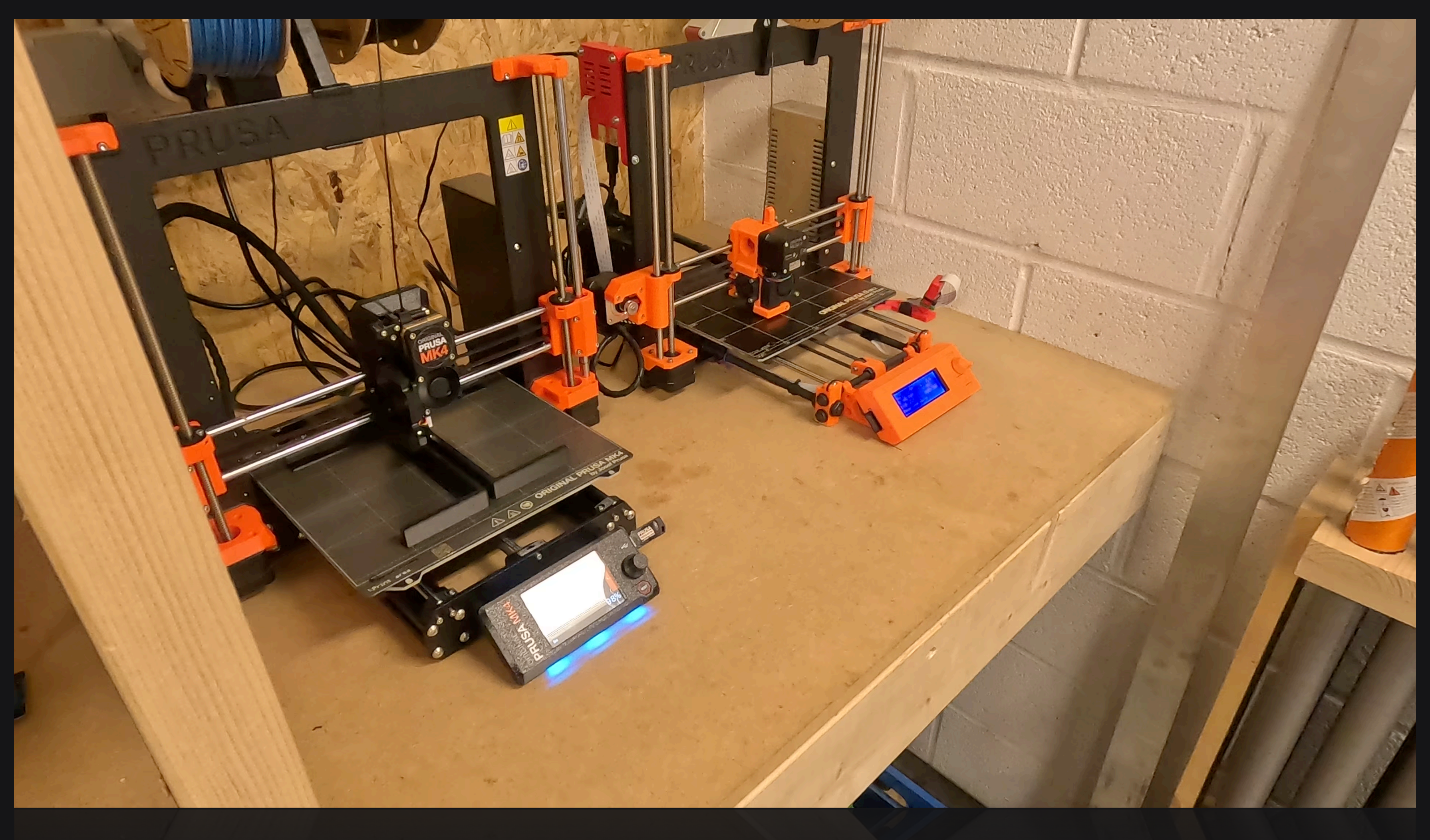


New motor support welded

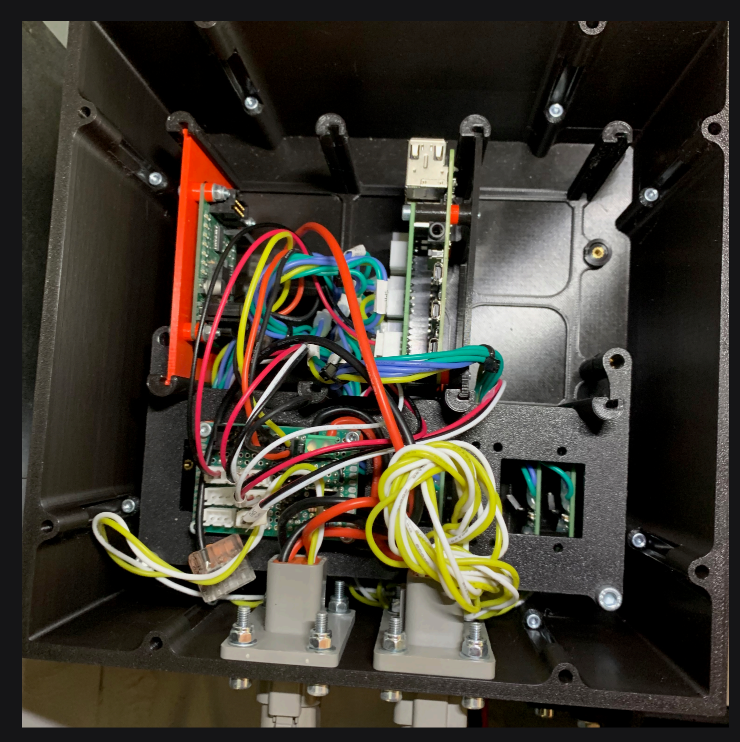
Connection plates CNC'd and welded

Custom junction piece to connect the motor to the gearbox

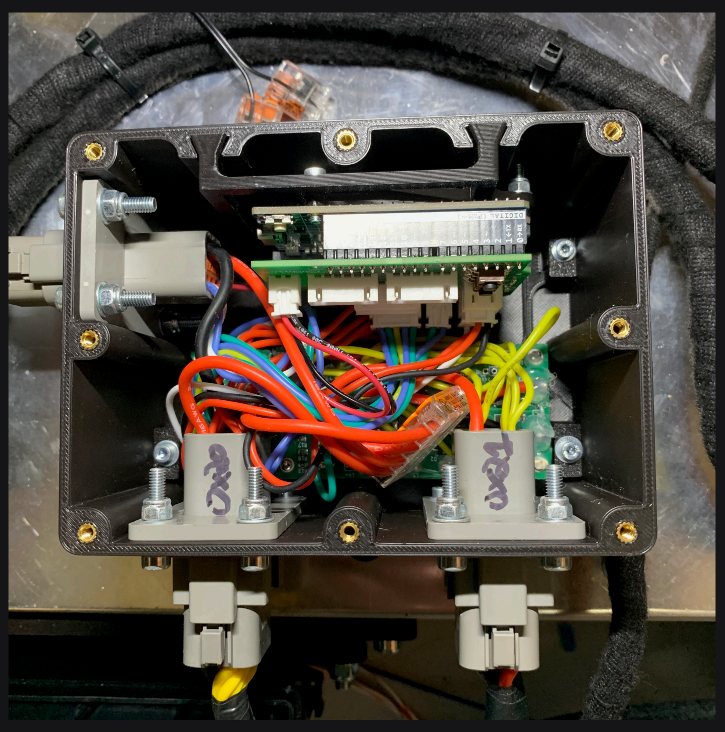
3D printing



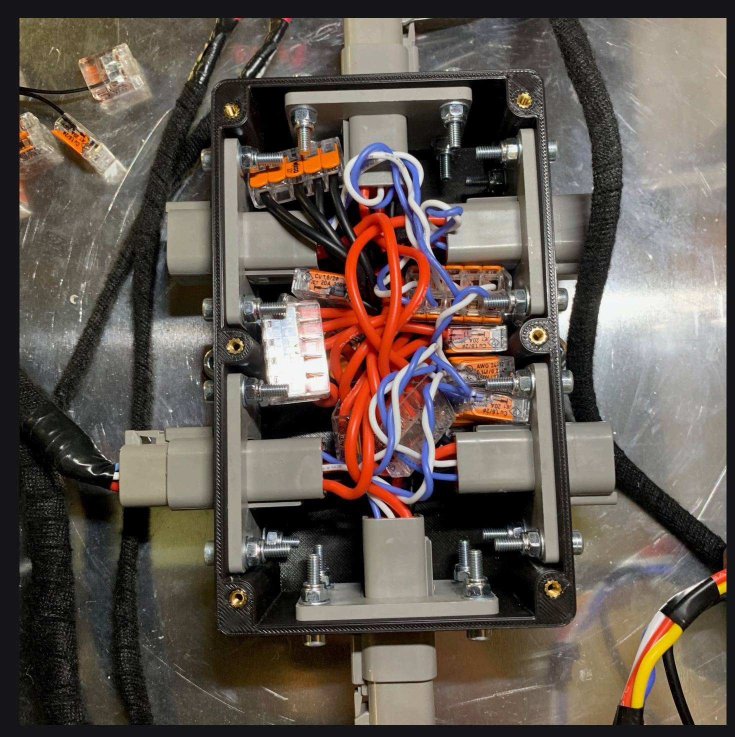
Iterating on the “plank” prototype



VMS (x1)
RPI4
Custom SPI hat
5xMCP2517FD



Generic controller (x3)
Arduino R4 Minima
Custom SPI hat
MCP2517FD



OVCS Canhub (x3)

(Just cables 😄)

VMS in more details

Dynamic Web frontend in Vue.js

Backend Core and API in Elixir + Phoenix

Nerves running the Erlang VM

Buildroot

VMS vehicle configuration

Vehicle composer

```
# VwPolo
{Polo9N.Dashboard, %{
  contact_source: Polo9N.IgnitionLock,
  rotation_per_minute_source: LeafZE0.Inverter
}},
{Polo9N.ABS, %{
  contact_source: Polo9N.IgnitionLock,
  rotation_per_minute_source: LeafZE0.Inverter
}},
{Polo9N.PassengerCompartment, []},
{Polo9N.IgnitionLock, []},
{Polo9N.PowerSteeringPump, %{
  selected_gear_source: Managers.Gear
}},

# NissanLeaf
{LeafZE0.Inverter, %{
  selected_control_level_source: Managers.ControlLevel,
  selected_gear_source: Managers.Gear,
  contact_source: Polo9N.IgnitionLock,
  controller: OVCS1.FrontController,
  power_relay_pin: 3
}},
```

Dashboard composer

```
def definition(order: order) do
  %{
    name: "Dashboard",
    icon: "HomeIcon",
    order: order,
    blocks: %{
      "vehicle-information" => %{
        order: 0,
        name: "Vehicle Information",
        type: "table",
        rows: [
          %{type: :metric, name: "Control Level", module: Managers.
          ControlLevel, key: :selected_control_level},
          %{type: :metric, name: "Manual Control forced", module:
          Managers.ControlLevel, key: :forced_to_manual},
          %{type: :metric, name: "Selected Gear", module: Managers.Gear,
          key: :selected_gear},
          %{type: :metric, name: "Key Status", module: Polo9N.
          IgnitionLock, key: :contact},
          %{type: :metric, name: "Speed", module: Polo9N.ABS, key:
          :speed, unit: "kph"},
          %{type: :metric, name: "RPM", module: LeafZE0.Inverter, key:
          :rotation_per_minute},
          %{type: :metric, name: "Output Voltage", module: LeafZE0.
          Inverter, key: :inverter_output_voltage, unit: "V"},
```

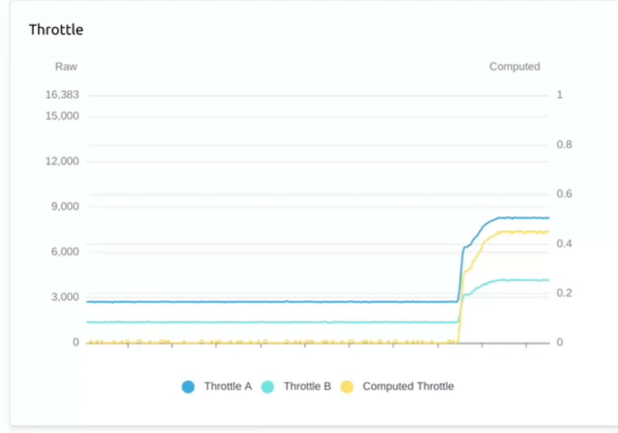
The car's new "brains"

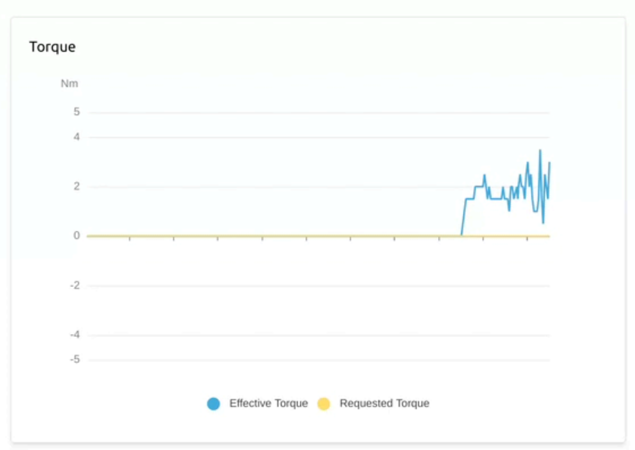
OVCS VMS

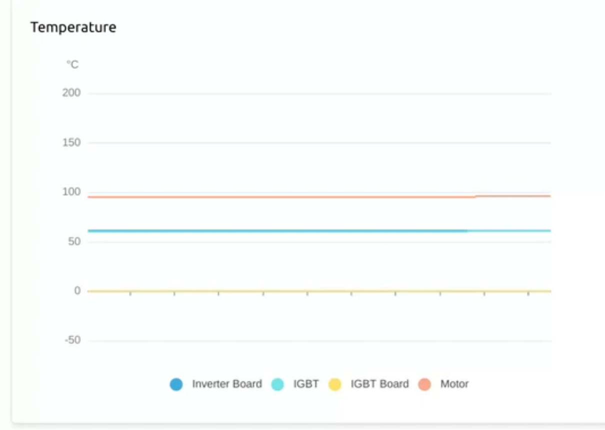
- Dashboard
- Network
- Car Controls

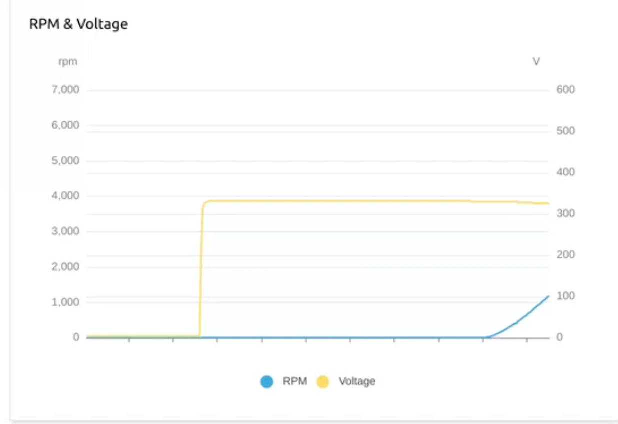
Vehicle information

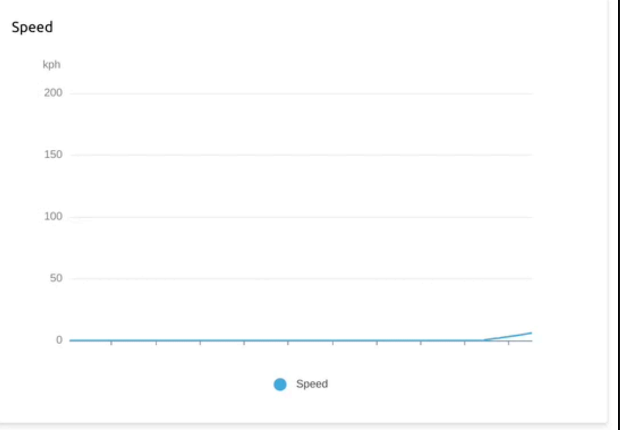
Selected Gear	parking
Key status	contact_on
Speed	6.11 kph
RPM	1183
Output voltage	326V
Motor temperature	96C











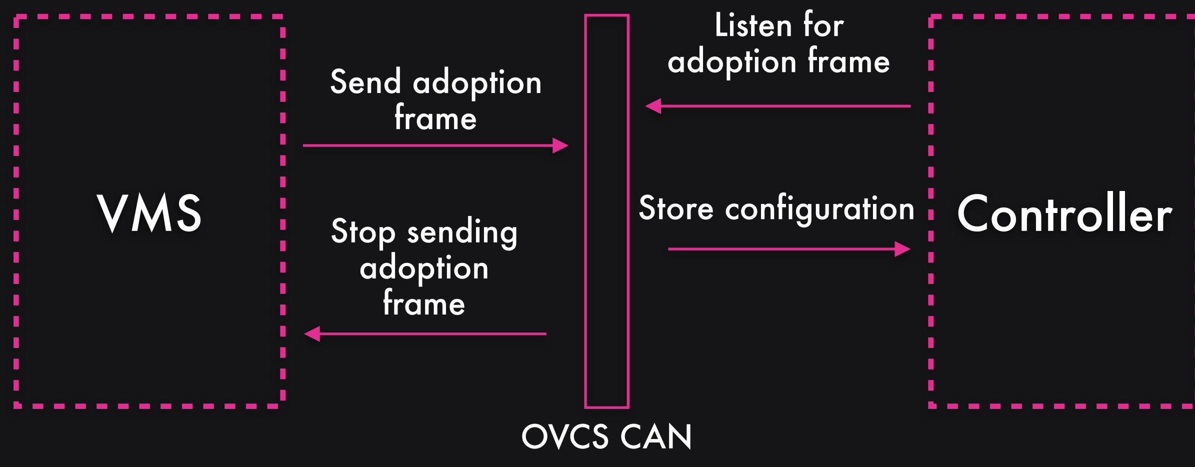
Generic controller in more details

OVCS Function	Physical Pin	OVCS Pin
UART Receive	D0	
UART Transmit	D1	
Adopt button	D2	
SPI CAN Int	D3	
Digital	D4	0
Software PWM	D5	0
Software PWM	D6	1
Digital	D7	1
Digital	D8	2
Software PWM	D9	2
SPI CAN CS	D10	
SPI CAN COPI	D11	
SPI CAN CIPO	D12	
SPI CAN SCK	D13	
DAC	A0	0
Analog In	A1	0
Analog In	A2	1
Analog In	A3	2
I2C SDA - MOSFET	A4	
I2C SCL - MOSFET	A5	
Digital	MOSFET0-0 -> 7	3 -> 10
Digital	MOSFET1-0 -> 7	11 -> 18
Hardware PWM	PiC32 over UART	0 -> 3

All controllers run the same code on Arduinos now (we dropped the specific controller code)

Their function is determined by the VMS during adoption

A button on the controller makes it go into adoption mode



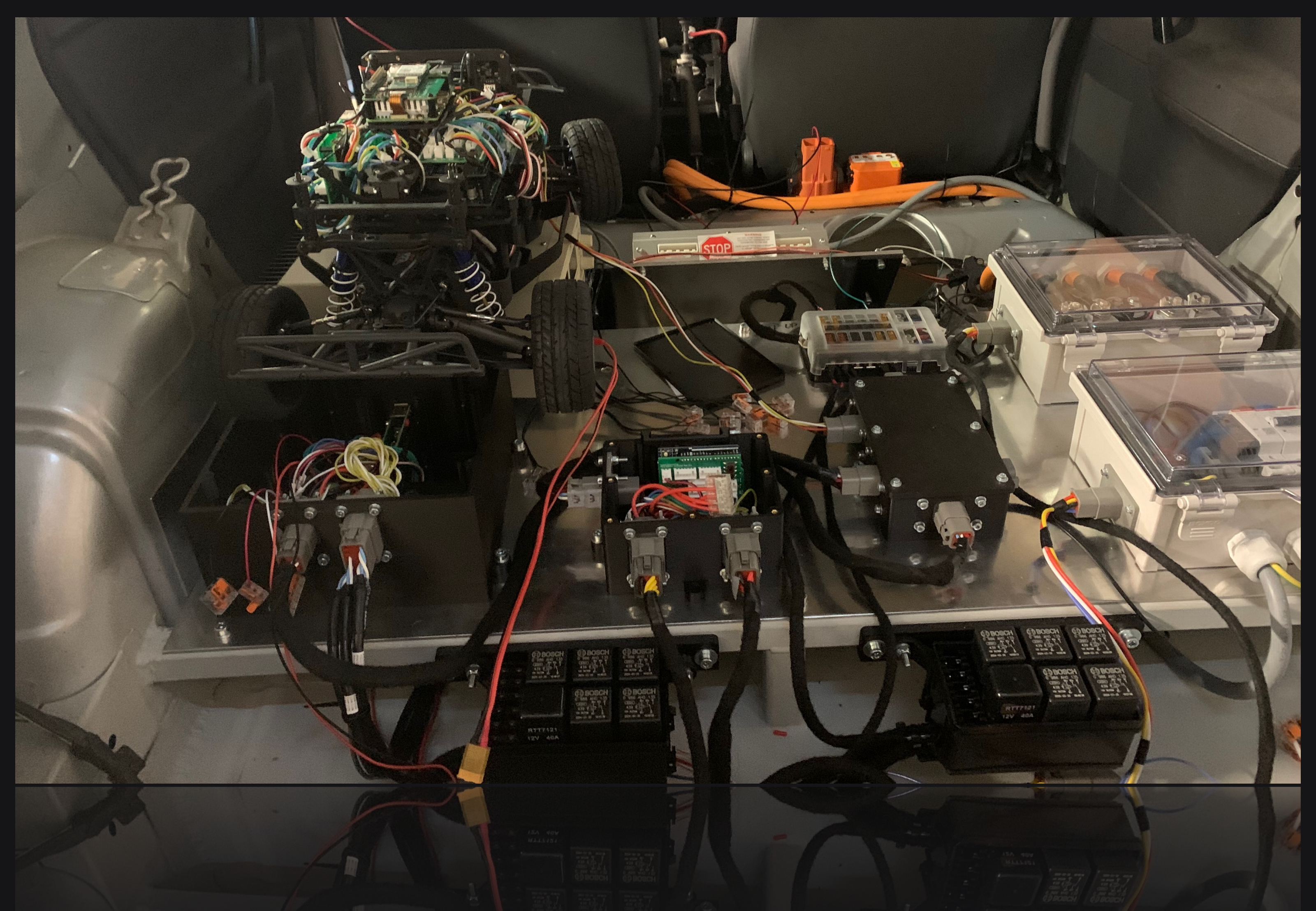
The infotainment

Gives information and diagnostics about car features

Replaces the gear selector (PRND)

Built on top of Nerves + flutterpi

Placing the components in the car



Adapting the car

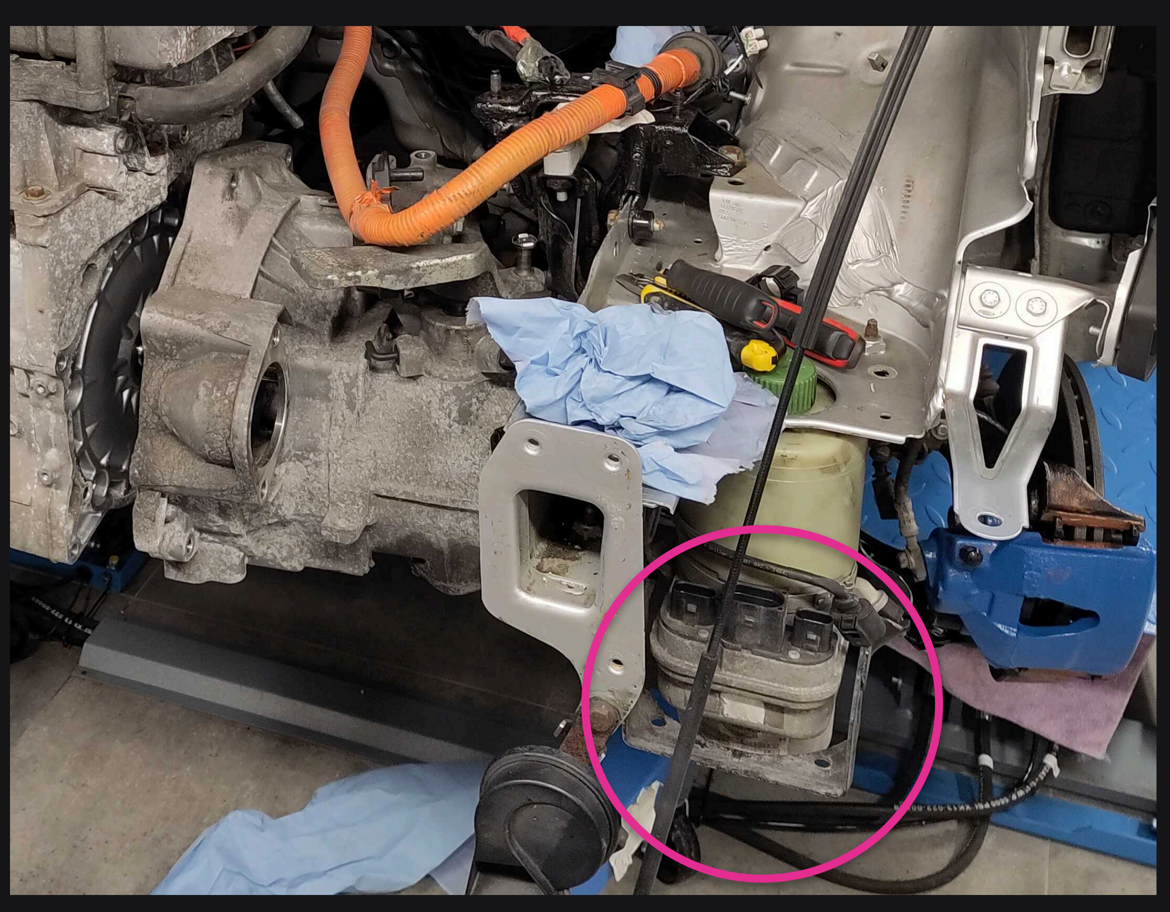
Changing the servo-brakes

The polo had a servo-brake that used the depression from the thermal engine to provide brake assistance

Tesla's "brake boosters" are popular in old car renovations

We simply installed a gen2 Tesla iBooster to solve this issue

Controlling the steering hydraulic pump



The pump starts when the thermal engine is started

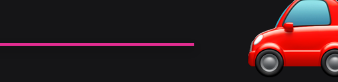
It knows it's started when the RPMs on the CAN are the "idle RPM" of the thermal engine...

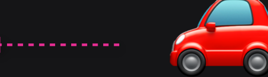
We are controlling it separately through the VMS by faking the engine presence and RPM

Building the battery from used cells ☠️

What if we transformed our Polo EV into an RC and autonomous vehicle?

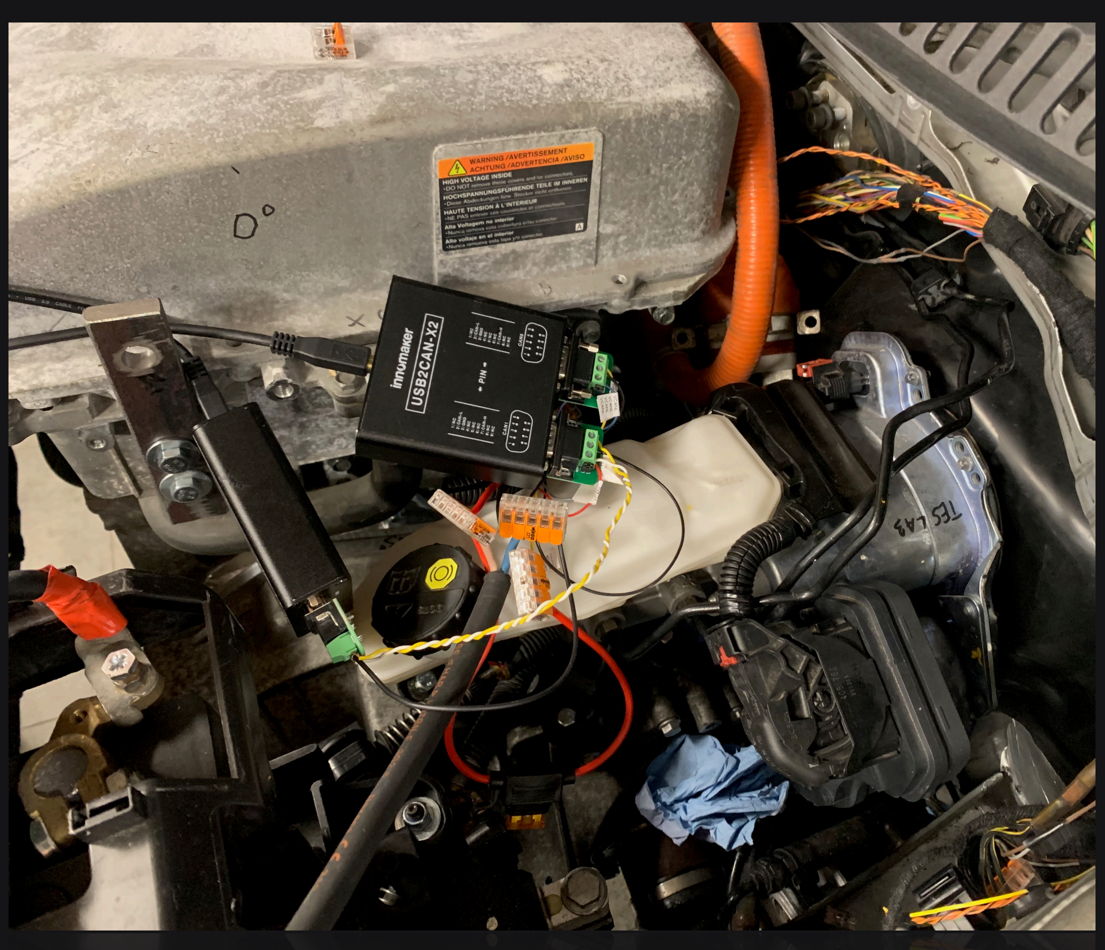
Controlling the Polo

Acceleration 

Braking 

Steering

Braking



Tesla's brake boosters can be controlled via CAN

The CAN messages allow to control the rate of fluid going through the booster

From gen1 DBC files and some CAN traces we found, we were able to reverse the right CAN messages

https://github.com/open-vehicle-control-system/dbc/tree/main/ibooster

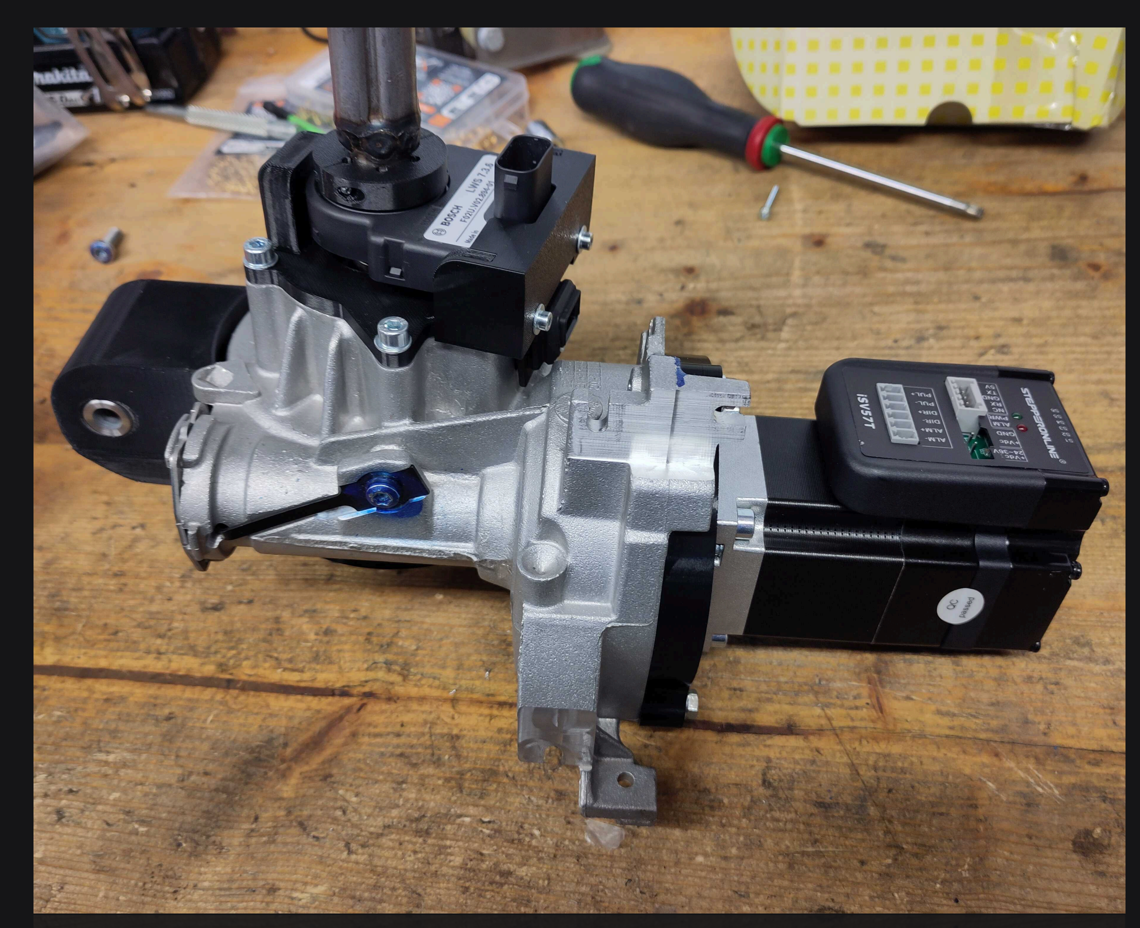
Braking ✅

Steering

The original steering column is not motorised

We tried reversing a 2019 Polo steering column (2Q1909144) with no success

We stripped the 2Q1 of it's ECU and motor and simply connected another servo and angle sensor



Steering ✅

A Mavlink bridge for OVCS

"Micro Air Vehicle Link", mostly used in aerial drones

Also supports "rover" types of drones

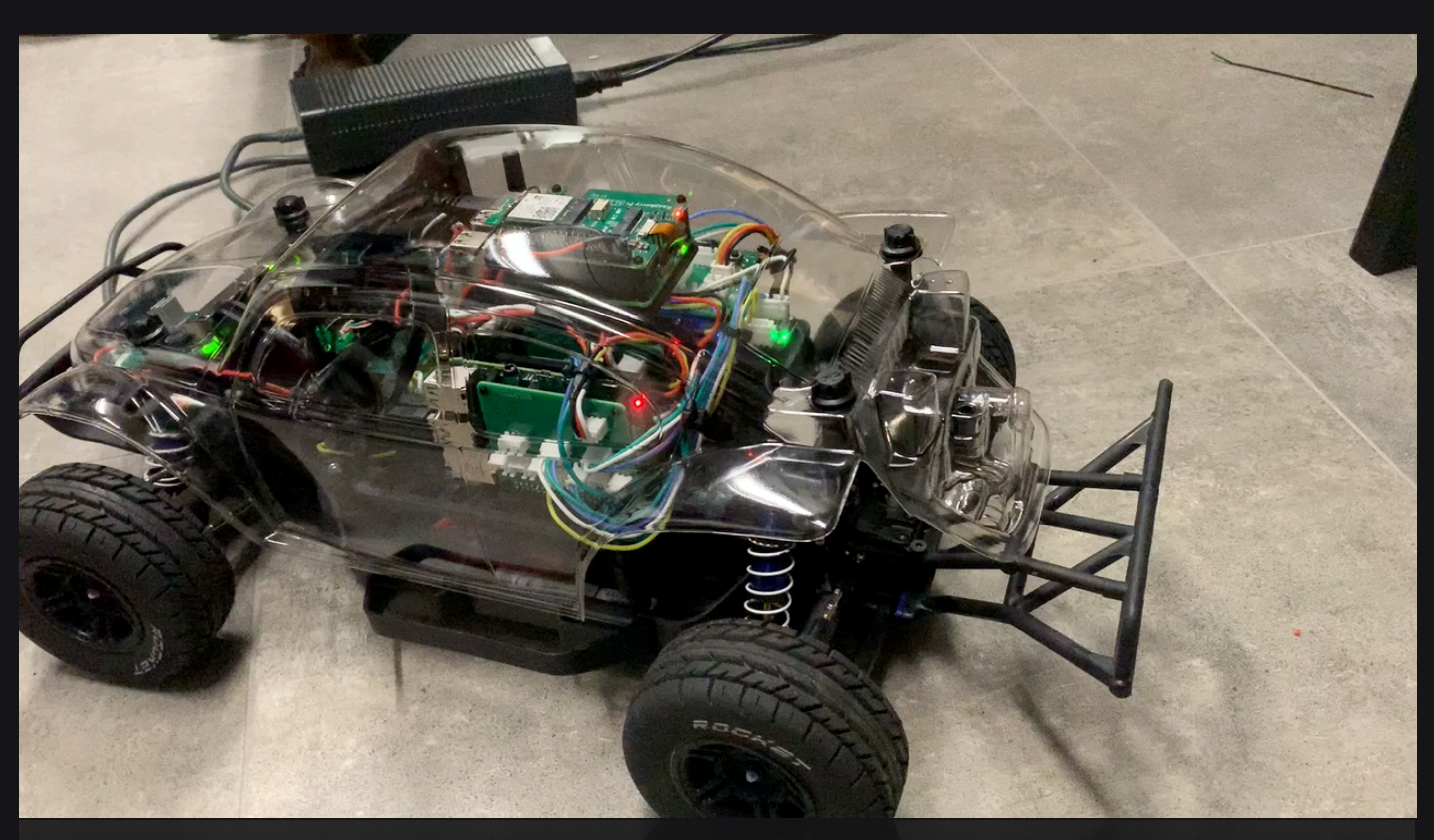
Open protocol which can be extended with our own messages

Supported by several controllers, libraries and tools

RC control of OVCS1

What **now**?

OVCS Mini, because testing on a real size car can be... dangerous... ☠️



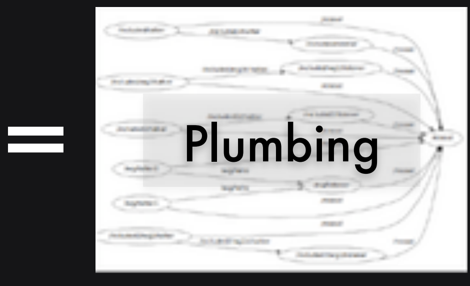
Same hw/sw stack as the full size car

Also using CAN as a communication bus

Will allow us to test features in a safer way

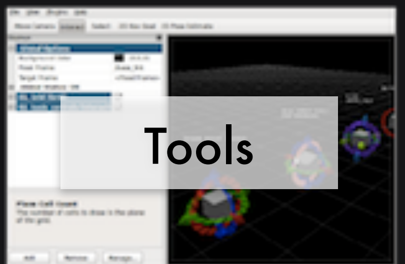
A ROS2 bridge for OVCS (wip)

=



+

+



+

Robot Operating System

Using Rclex, an Elixir ROS2 client working with Nerves

No need to run Ubuntu 🎉, it runs on Buildroot through Nerves

Perception stack (wip)

- Uses open source AI models for object detection and segmentation
- Sends detected bounding boxes and classes through ROS2 topics

The OVCS remote (wip)

Multi protocol remote
(Mavlink, ROS2, ?)

Allows us to test new features that are not supported by off-the-shelve transmitters

And… it's just cool to build one 😎

Small reminder

NONE OF THIS IS ROAD CERTIFIED

Maybe one day...

That's it!
Any ideas, suggestions, questions?
info@spin42.com
https://github.com/open-vehicle-control-system
NERVES
elixir
OVCS1