Handcart for Fenice

Matteo Bitussi - Lorenzo Rossi
E-Agle TRT

2020-2021

Contents

Introduction

Cose

Chapter 1

General view

Chapter 2

Back-end

The back-end is thought to act as the controller to start and stop the charge, to handle the fans and to act as an intermedery between the BMS and the BRUSA.

2.1 How it works

- The back-end sends a can message to the BMS, with **Cut-off voltage**, **type of charge** (fast or normal)
- The accumulator decides what charging curve to follow, he sends charging messages to the back-end that forward them to the BRUSA
- The charge can be interupped by the back-end itself, or finished/interrupped by the BMS

Note that the accumulator has a parallel state-machine (apart the normal one) when is charging

2.2 Settings

There are various settings to be choosen in the back-end

- **Charging speed**: back-end can ask the BMS to use a particular charging curve (**fast** or **normal**) default is normal.
- **Current drawn from the outlet**: back-end can ask BRUSA to set a maximum current to drawn from the outlet, useful when using standard home outlet
- **Fan profile**: back-end can use a fixed profile for the fans (i.e 90), by default it uses a fan curve
- **Choose Accumulator**: this setting is obligatory, the back-end has to know what car's accumulator is attached

2.3 The state machine

The main.py is based on a state machine, which states are these

2.3.1 STATE: check

Checks for the presence of the accumulator and BRUSA

2.3.2 STATE: idle

Accumulator and BRUSA are connected, wait for user input to start precharge

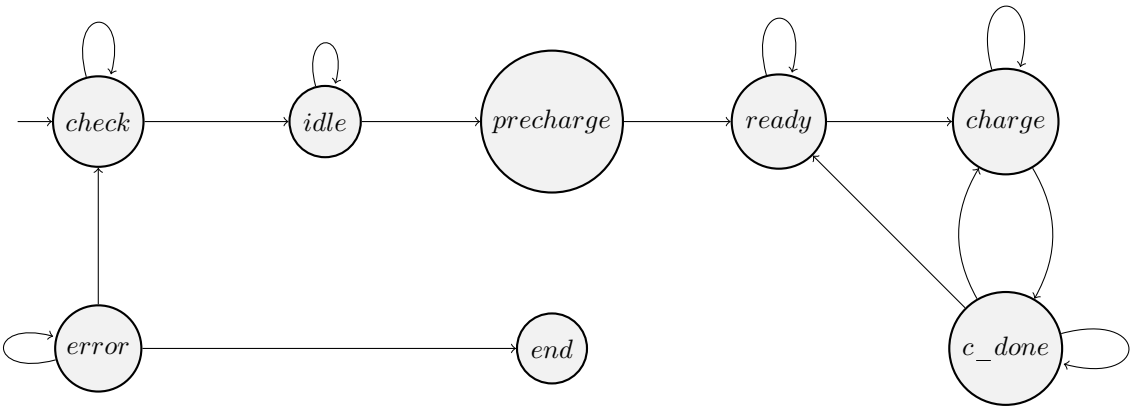

Figure 2.1: Back-end state machine

2.3.3 STATE: precharge

Ask BMS TS_{ON}

2.3.4 STATE: ready

Accumulator has closed airs, and is ready to be charged. Waiting for user input

2.3.5 STATE: charge

Charging, until bms stop or user stop

2.3.6 STATE: c_done

Looping waiting for user input

2.3.7 STATE: error

An error has ben detected, tell to user and restart program or exit

2.4 The can listener

The object istantiated by the class CanListener stores all the info received by can messages, its called every time a new message arrives, processes it and save the result in itself. This object (can_read) is thought to be an interface to read all the useful info from the can, so dont get and process can messages outside this object (please).

Chapter 3

Front-end

3.1 General description

The front-end GUI is a webapp written in JS, it communicates to the HTTP server with RESTFUL requests

Chapter 4

Flask HTTP server

4.1 General description

The server is needed from the back-end and the front-end to communicate. It is a flask server written in python. It serves both the http pages to clients and also the RESTFUL requests from the js webapp

Chapter 5

BRUSA NLG5 charger

Prepare yourself for the better part of the documentation, the brusa part.

5.1 connect in serial

With serial communication you can upload charging profiles via ChargeStar software (see proper section) and monitor some serial-related infos
To connect to the BRUSA with serial you have to use an USB to serial adapter. The pin has to be mapped from the brusa as told in the brusa's manual (PINs 11,12,14). There are the settings for the connection that in german are called anschlusseinstellungen (i have stepped in it while searching for the parameters ;)):

1. Baudrate: 19200
2. Data bits (Datenbits): 8
3. parity; none
4. stopbits: 1
5. protocol: Xon/Xoff

If you are on windows, let the serial COM settings as default, but edit the putty settings, i got some problems otherwise.
Very important: i don't know why, but if you are going to use ChargeStar software you need to change the COM port to COM1, otherwise the brusa will not be recognized by the program.
Note that the serial works only if the board on the brusa is fed with 12 volts via the proper pin 2 AUX or with the main power.
Note that sometimes the serial will freeze or not respond, try to restart the brusa.

5.1.1 serial monitor

to use the serial monitor you have to properly connect and setup the serial, then, use putty on windows or minicom on linux to connect to it. You will asked with a pasword, which is m̈onitor

5.2 ChargeStar software

With the ChargeStar software you can program a charging profile, set various parameters and change some configuration of the brusa, see the brusa's manual for all the infos. Via ChargeStar you can also set the mode to CAN, very useful to control the charge via can. The ChargeStar software will run only on Windows XP or Windows Vista, obviously we'll chose XP, you can run a virtual machine on virtualbox and do the USB-passthrough of the serial to USB adapter. I read that sombody had issues with ChargeStar using the 64 bit version of windows, but for me worked fine. Note that i ran in some problems uploading a custom setting to the brusa: sometimes when the settings are uploaded, the brusa gives an NVSRAM CRC error, the only possible fix is reupload the settings to brusa changing some parameters. I'm still not sure which

parameter is causing problems, so change them randomly a bit and it should work after some tries. If you see, some input fields don't accept values with the :ñot sure why.

5.3 connecting with CAN

Connecting with can allows to monitor the message outputed by the brusa and (if properly configured) to set some parameters for charging. The CAN connection has to end with a 120 Ohm resistor, otherwise the messages will keep bouncing (kinda), trust me, it is necessary.
See the full CAN matrix at the file or manual.
By default the CAN is at 500kbps, unless differently specified in config file with ChargeStar. As i saw, brusa send messages just when the PON pin is set to HIGH (>5V). To set and enable the charge via can you have to send periodically a can message named NLG5_CTL see details on the can matrix.
Note that the endianess is **big (motorola)**