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

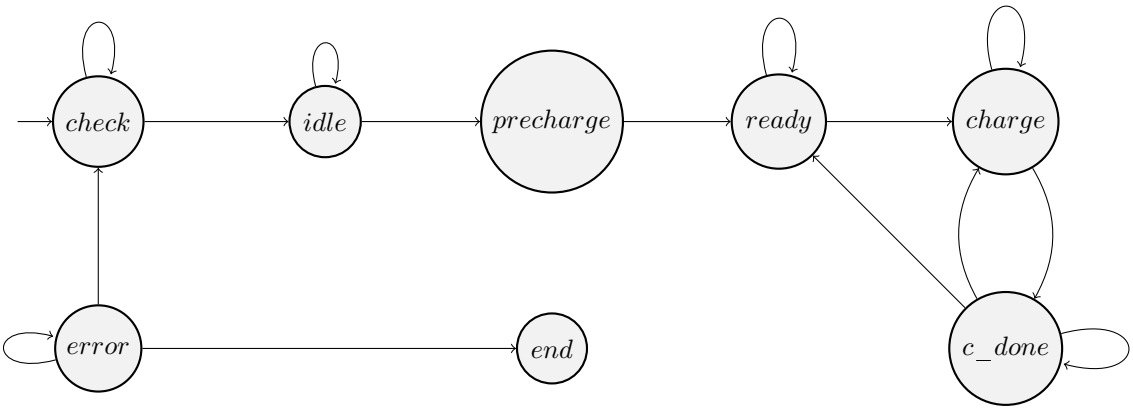

Figure 2.1: Back-end state machine

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 based on Qt, written in python. It has the purpose to act as a interface for the user to manage the charge process

Chapter 4

Deamon gRPC server

4.1 General description

The server is needed from the back-end and the front-end to communicate. It is based on gRPC and protocol Buffers