Battery Monitoring App on a Raspberry PI

– Project Module –

Student:	Matthias Korf
Project:	Electrical Engineering SS18
Professor:	Ferdi Hermanns

Krefeld, 20/07/2018

Table of Contents

1 Introduction

This Web-App is based on the Python-Dash by Plotly Framework[1]. The purpose of this Web-App is to enable monitoring of battery cells in a battery pack and to provide a controller panel to load battery packs. The app is running on multiple threads. The first thread reads in the Can-Bus data from the PiCan2 Board into a global queue. The second thread takes values from the global queue and filters the CAN messages into different local queues. The third thread runs the Dash-Web-App with multiple callbacks updating the values from local queues into live plotting visualizations.

2 Raspberry Pi Installation Guide

2.1 OS Installation

Go to the official Raspberry Pi Webpage (RPI), download the Noobs Image and load it onto a SD-Card. Check out the official instructions in the link below:

https://www.raspberrypi.org/documentation/installation/noobs.md

After the installation, you should be able to boot up the Pi for the first time.

2.2 PiCAN2-Board Setup

The PiCAN2-Board (see Figure 1) is an adapter board which allows CAN bus messaging on a RPI. To setup the PiCAN2-Board check out the official instructions in the link below:
http://skpang.co.uk/catalog/pican2-canbus-board-for-raspberry-pi-23-p-1475.html

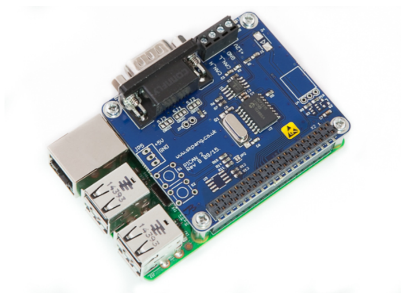

Figure 1: RPI with PiCAN2-Board
Source: Skpang[2]

[1]https://plot.ly/products/dash/

[2]http://skpang.co.uk/catalog/pican2-canbus-board-for-raspberry-pi-23-p-1475.html

Once, you completed the full installation, you should be able to ramp up the CAN bus interface on your RPI and send and receive messages via either a C or Python API. In this project we will make use of the Python API only.

Use this code *CanBus_Testcode.py* (see Listing 1) to test your CAN bus setup. As a result, you should be able to see incoming messages from the CAN bus.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
Decription: Test PiCAN 2 Interface
Version: 4/2018 Roboball (MattK.)
"""
import can
import os
import time
from collections import deque

#### init can bus globals ####
freq = 0.5 # set receiver frequency, e.g. 0.5
num_sens = 2 # number of sensor data
cells = 6 # number of cells per battery pack (accus)
# init buffer
a1_V_list = [] # accu 1 voltage
a1_T_list = [] # accu 1 temperature
a2_V_list = [] # accu 2 voltage
a2_T_list = [] # accu 2 temperature
for cell in range(cells):
	a1_V_list.append(deque(maxlen=20))
	a1_T_list.append(deque(maxlen=20))
	a2_V_list.append(deque(maxlen=20))
	a2_T_list.append(deque(maxlen=20))

#### Bring up the can0 Interface ####
# Close can0 if still open
os.system("sudo /sbin/ip link set can0 down")
# Bring up can0 interface at 500kbps
os.system("sudo /sbin/ip link set can0 up type can bitrate 500000")
time.sleep(0.1)
# Connect to can0 interface
bus = can.interface.Bus(channel='can0', bustype='socketcan_native')
print('connected to can0 interface')
print('ready to send/receive can messages')
```

```
def filter_buffer(byte_list, msg_id):
	''' a filter for CAN messages to sort into diff. buffer'''
	byte2 = byte_list[1] + byte_list[2]
	#print(byte2)
	# convert from hex to dec
	val_dec = int(byte2, 16) / 100
	print(val_dec)

	# filter into buffer (temperature and voltages)
	for pos2 in range(1,num_sens+1):
		for pos3 in range(1,cells+1):
			if byte_list[0] == str(pos2) + str(pos3):
				print('sorting byte '+ str(pos2) + str(pos3) + '!!')
				# sort for ID: 600 (Accu 1)
				if msg_id == 600:
					# sort for temperatures
					if pos2 == 1:
						#print(a1_T_list[pos3-1])
						a1_T_list[pos3-1].append(val_dec)
						#print(a1_T_list[pos3-1])
					# sort for voltages
					if pos2 == 2:
						#print(a1_V_list[pos3-1])
						a1_V_list[pos3-1].append(val_dec)
						#print(a1_V_list[pos3-1])
				# sort for ID: 602 (Accu 2)
				if msg_id == 602:
					# sort for temperatures
					if pos2 == 1:
						#print(a2_T_list[pos3-1])
						a2_T_list[pos3-1].append(val_dec)
						#print(a2_T_list[pos3-1])
					# sort for voltages
					if pos2 == 2:
						print(a2_V_list[pos3-1])
						a2_V_list[pos3-1].append(val_dec)
						print(a2_V_list[pos3-1])


if __name__ == '__main__':
	while (True):
		#print(list(a1_V_list[0]))
		# receive CAN messages
		msg = bus.recv()
		len_msg = len(msg.data)
		#print(vars(msg))
		#print(msg)
		#print(msg.arbitration_id) # 1536= ID 600, 1537= ID 601 (Error Akku1),
                msg_id = int('{0:x} '.format(msg.arbitration_id))
                #print(msg_id)
                #print(msg.timestamp)
                #print(msg.data)
                # check correct msg length
                if len_msg == 3:
                        byte_list = [] # init empty bytestring
                        for pos in range(len_msg-1,-1,-1):
                                byte = '{0:x}'.format(msg.data[pos])
                                print(byte)
                                if len(byte) == 1:
                                        byte = '0' + byte
                                        #print(byte)
                                        #print('Length ',len(byte))
                                byte_list.append(byte)
                                #print(pos)
                        print(byte_list)
                        # call filtering for values
                        filter_buffer(byte_list, msg_id)
                # update frequency
                time.sleep(freq)
```

Listing 1: Python CanBus_Testcode.py

3 Python Installation Guide

The good news: *All you need is Python (and everything in one file)!!*

This means the webserver backend, the web-frontend, the plotting library, the Can-Bus interface and all data buffers can be imported by just using various Python libraries. Additionally, CSS-Styles and Javascript Script can be imported from the web or stored locally. To get your Python Code running we need to install all dependencies first.

3.1 Dependencies

Go to my github and clone the repo:

https://github.com/dialogchat/dash_monitoring_app

Then enter the following command into your command line:

```
pip3 install -r requirements.txt
```

3.2 Demo Mode

Once you installed all dependencies (see 3.1 above), you should be ready to test the demo. The demo does not use the PI-Can2 interface, therefore you can test it by just using a standalone PC or RPI without PiCan2 Shield.

Before running the demo, go to the end of script in `demo.py` and enter the IP-address of your host PC or RPI at the line `IP = '192.168.200.1'`.

Then save the script and run the following command in your terminal:

```
python3 demo.py
```

Go to your favorite web-browser and enter the IP you just entered in your script followed by a your port number (e.g. `192.168.200.1:9999)`.

The demo should show up showing random bar-charts.

3.3 Full Mode

In full mode, you need to install the PiCan2 Shield on the RPI first. To check if everything works, you may test your installation with Listing 1.

Then analogous to 3.2, go to the end of script in `demo.py` and enter the IP-address of your RPI at the line `IP = '192.168.200.1'`.

Then save the script and run the following command in your terminal:

```
python3 run_app.py
```

Notice, dependent on the amount of your data, the PI might have performance issues to load the app fast. However, PCs in your network should be able to load the page without problems (The bottleneck are the update times of the callbacks. In case of problems, try to change them).

4 Layout Design

The app layout is build as Single Page Application (SPA). On the left side, there is a Control Panel with various buttons e.g. to charge or stop charging the battery pack. On the right side, you will find visualizations for cell temperatures and voltages.

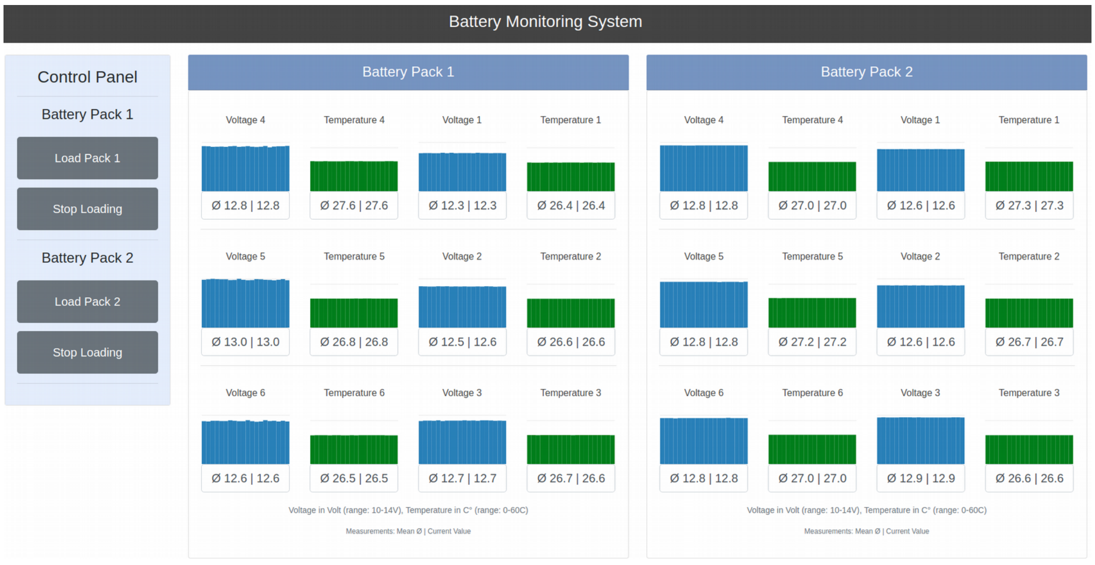

Figure 2: Web-App-Layout

The x-axis of each plot measures 20 data points where the most-right value represents the most recent value received by the CanBus. More recent values will stay in the buffer and will travel from the right to the left. In case of Voltage, the y-axis measures voltages from 10 to 14 Volt. In case of Temperature, the y-axis measures temperatures from 0 to 60 Degree Celsius. In case a value lies above or beneath the range of the y-axis, the color of the bar-plot will turn into a red warning-color.

The fields below each bar-plot contain two values. The left values represents the mean value of the last 20 values and the right value represents the most recent value received by the CanBus.

5 Hardware Setup

In Figure 3, you can see a complete hardware setup of the system. There are two battery packs equipped with 6 battery cells each and a Battery Monitoring System (The green BMS are located at the top right yellow part of each pack next to the orange power connectors). Each BMS sends out Voltages and Temperatures via CAN bus to the RPi (The blue PI CAN board with the green RPI in the middle of the Picture).

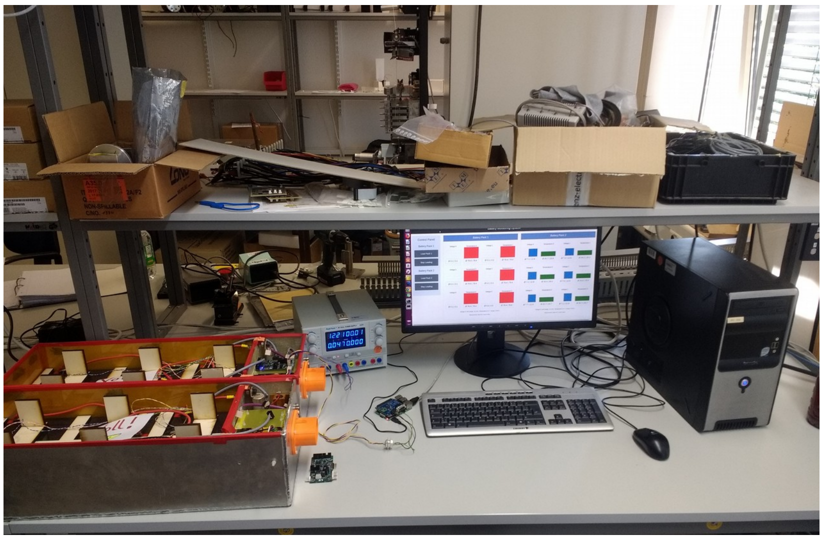

Figure 3: Full Hardware Setup

The Pi is connected to the internet (here gray cable: ethernet, alternatively use wlan) and the Battery Monitoring Web App is displayed in the browser of the host PC (screen on the right side). You can see live plotting of Voltages and Temperatures of the right Battery Pack (blue and green) while the left Battery Pack is currently disconnected (Error, red Values).

6 Technical Information

6.1 Can-ID's

Can-ID	Description	Additional Info
600	Accu 1, Voltages and Temperatures	Contains MSGs from 6 Cells
601	Accu 2, Voltages and Temperatures	Contains MSGs from 6 Cells
602	Accu 1, Error	
603	Accu 2, Error	

Table 1: Can-IDs
Source: Own creation