震江電力科技股份有限公司

PD-Survival 移植文件

文件編號：

版次：3.0

文件日期：2020 年 04 月 22 日

機密等級：震江內部使用

文件制/修訂紀錄

版次	制/修訂日期	制/修訂人員
1.0	2020.04.01	張岳傳
2.0	2020.04.08	張岳傳
3.0	2020.04.22	張岳傳

目錄

壹、目的

一、PD-survival

在 pd-simply 硬體置入演算法，透過趨勢與圖譜進行放電類型輔助判斷。

貳、圖譜辨識

功能說明: 輸入圖片特徵向量，輸出辨識結果 0, 1, 2, 3

使用文件:

svc_simplified.c

Train_simply 文字檔(3072 個值)

輸入:

圖譜格式: 80x180 RGB, 除過 255

3072 一維向量，代表一張圖 放在 train 文件檔

原始碼: svc_simplified.c

編譯：

$ gcc svc_simplified.c -std=c99 -lm -o svc_simplified

測試硬體:

```
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 42
Model name:            Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz
Stepping:              7
CPU MHz:               1643.923
CPU max MHz:           3200.0000
CPU min MHz:           800.0000
BogoMIPS:              4983.73
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              3072K
NUMA node0 CPU(s):     0-3
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe s
yscall nx rdtscp lm constant_tsc arch_perfmon pebs bts nopl xtopology no
nstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx e
st tm2 ssse3 cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic popcnt tsc_deadlin
e_timer aes xsave avx lahf_lm epb pti ssbd ibrs ibpb stibp tpr_shadow vn
mi flexpriority ept vpid xsaveopt dtherm ida arat pln pts md_clear flush
_l1d
```

使用:

$ ./svc_simplified

輸出結果:

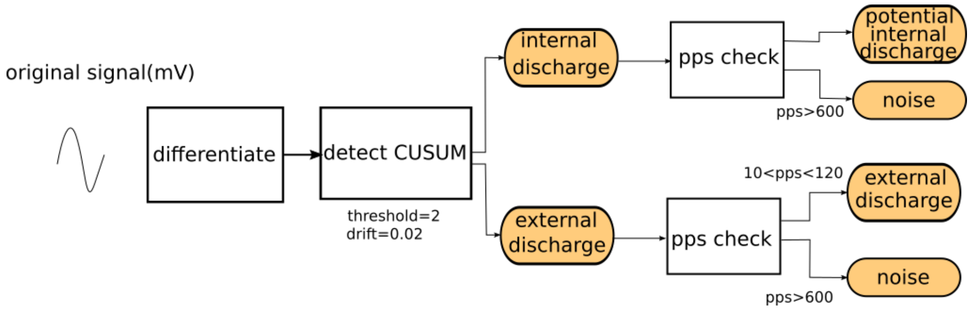
```
last number = 1.0000000000000002
features number = 3072
2
```

2 為分類結果(礙子放電)

叁、趨勢辨識

功能說明: 輸入 120 個 float，輸出辨識結果 0, 1, 2, 3

流程圖:



使用文件與原始碼:

main.c : 主程式
list.h list.c : 實做 linked list
diff.h diff.c : differentiate
detect_cusum.h detect_cusum.c : detect CUSUM
read_data.h read_data.c: 讀入測試 data_Mv.txt data_Pps.txt
convolve.h convolve.c: convolution
Makefile: 編譯用
data_Mv.txt 120 點(2 小時內)電壓值
data_Pps.txt 120 點(2 小時內)Pps

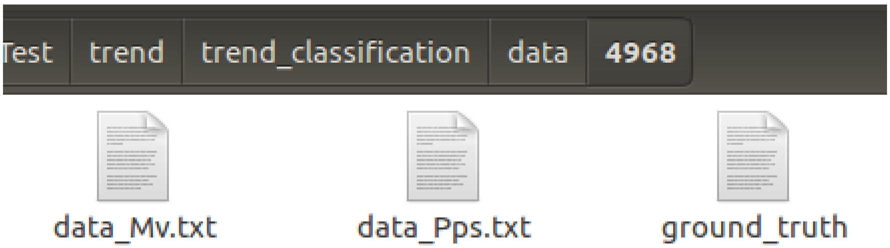
測試資料: data 資料夾(Golden sample)

真值(ground truth): data 資料夾下 ground truth

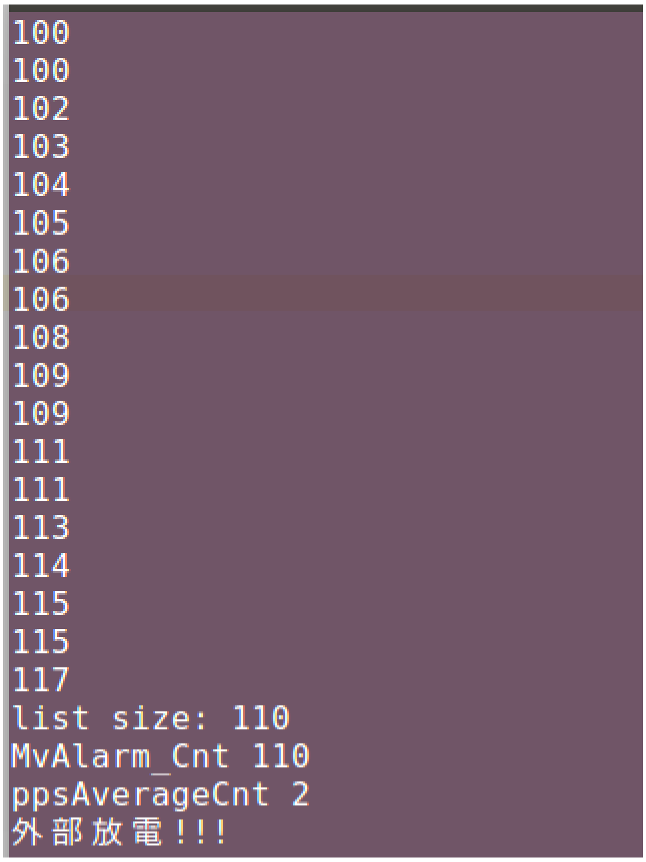
編譯:
make

測試:
將 data 資料夾下，比如 4968 資料夾中 ***data_Mv.txt, data_Pps.txt*** 丟入主程式同一層資料夾,
接著執行
./main
ground_truth 為測試正確結果

比如



測試畫面截圖:

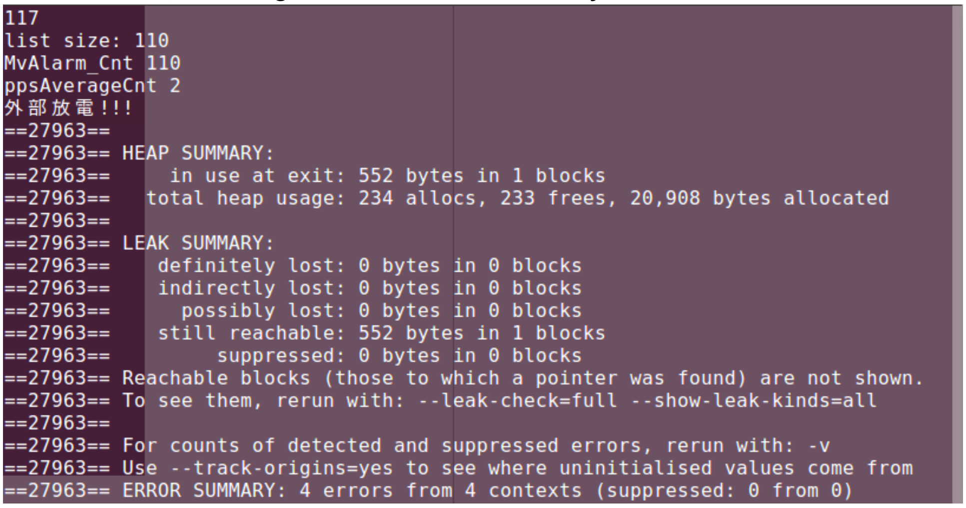


```
100
100
102
103
104
105
106
106
108
109
109
111
111
113
114
115
115
117
list size: 110
MvAlarm_Cnt 110
ppsAverageCnt 2
外部放電!!!
```

其他說明:

為了降低執行時期記憶體開銷，透過 calloc 動態配置記憶體。

執行 **make leaks,** 使用 Valgrind 檢查記憶體 memory leaks



```
117
list size: 110
MvAlarm_Cnt 110
ppsAverageCnt 2
外部放電!!!
==27963==
==27963== HEAP SUMMARY:
==27963==     in use at exit: 552 bytes in 1 blocks
==27963==   total heap usage: 234 allocs, 233 frees, 20,908 bytes allocated
==27963==
==27963== LEAK SUMMARY:
==27963==    definitely lost: 0 bytes in 0 blocks
==27963==    indirectly lost: 0 bytes in 0 blocks
==27963==      possibly lost: 0 bytes in 0 blocks
==27963==    still reachable: 552 bytes in 1 blocks
==27963==         suppressed: 0 bytes in 0 blocks
==27963== Reachable blocks (those to which a pointer was found) are not shown.
==27963== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==27963==
==27963== For counts of detected and suppressed errors, rerun with: -v
==27963== Use --track-origins=yes to see where uninitialised values come from
==27963== ERROR SUMMARY: 4 errors from 4 contexts (suppressed: 0 from 0)
```

執行 **make profile** 使用 Valgrind **massif 進行 process 記憶體分析如附錄(文件 massif.txt)**

http://valgrind.org/docs/manual/ms-manual.html

在原本 pdsimply box 或可做**動態記憶體回收**進行記憶體最佳化。

原理 ：

Cumulative sum algorithm (CUSUM) to detect abrupt changes in data

Implenmentation Reference

https://elf11.github.io/2018/09/20/data-science-anomaly-detection.html

https://github.com/BMClab/BMC/blob/master/functions/detect_cusum.py

Adaptive Filtering and Change Detection

https://eden.dei.uc.pt/~tbohnert/math/ap-cp.pdf　P.66

Algorithm 3.2 CUSUM

$$g_t = g_{t-1} + s_t - \nu \tag{3.27}$$

$$g_t = 0, \text{ and } \hat{k} = t \text{ if } g_t < 0 \tag{3.28}$$

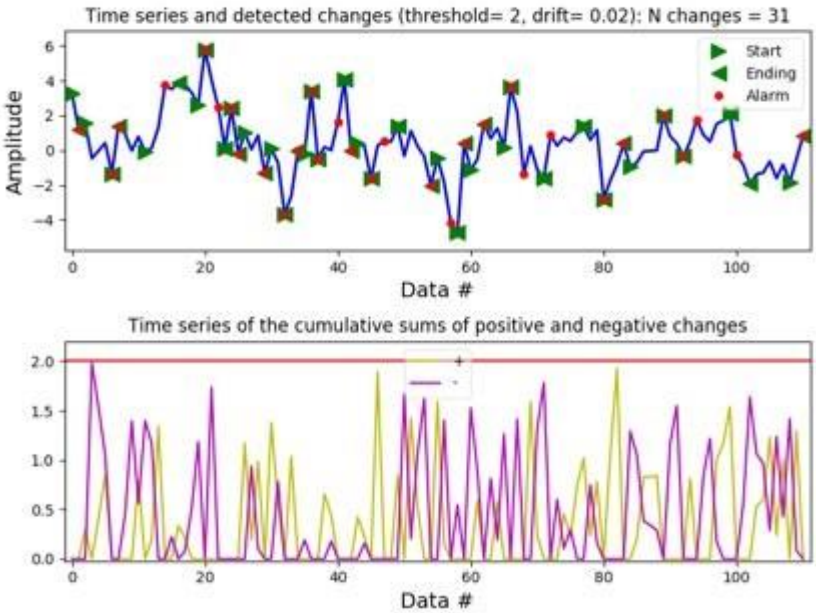
$$g_t = 0, \text{ and } t_a = t \text{ and alarm if } g_t > h > 0. \tag{3.29}$$

Design parameters: Drift ν and threshold h.
Output: Alarm time(s) t_a and estimated change time $\hat{k}$.

Both algorithms were originally derived in the context of quality control (Page, 1954). A more recent reference with a few variations analyzed is Malladi and Speyer (1999).

In both SPRT and CUSUM, the alarm time t_a is the primary output, and the drift should be chosen as half of the critical level that must not be exceeded by the physical variable θ_t. A non-standard, but very simple, suggestion for how to estimate the change time is included in the parameter $\hat{k}$, but remember that the change is not necessarily abrupt when using stopping rules in general. The estimate of the change time is logical for the following reason (although the change location problem does not seem to be dealt with in literature in this context). When $\theta_t = 0$ the test statistic will be set to zero at almost every time instant (depending on the noise level and if $a < -\nu$ is used). After a change to $\theta_t > \nu$, g_t will start to grow and will not be reset until the alarm comes, in which case $\hat{k}$ is close to the correct change time. As a rule of thumb, the drift should be chosen as one half of the expected change magnitude.

Robustness and decreased false alarm rate may be achieved by requiring several $g_t > h$. This is in quality control called a *run test*.



輸入: 輸入 120 個 float，輸出辨識結果 0, 1, 2, 3
輸入說明警報發起 2 小時內的電壓-時間，PPS-時間。各 120 點。

CUSUM: 計算變化點，正上升與反向下降次數。
參數 threshold, drift (使用 CUSUM 前半部)

肆、交付文件與驗收