FTF FREESCALE TECHNOLOGY FORUM 2015

Hands-On Workshop: Sensor Data Collection and Mining: **Intelligent Data Loggers**, Part 1 of 3

FTF-INS-F1220

Rod Borras | Freescale AMR FAE

Michael Steffen | Freescale AMR FAE

JUNE.2015

freescale™

External Use

Agenda

- Overview of Problems related to Long-Term Datalogging
- Our Methodology: brute force followed by local analysis
- Lessons Learned
- Hands-On: Phase 1 - Decimating the data
- Hands-On: Phase 2 - Running algorithms on huge data files
- Hands-On: Phase 3 - Narrowing down the algorithms
- Hands-On: Phase 4 - Putting it all together

Hands-on: Processing huge data files, and running algorithms, with the custom Java GUI

- ✓ Lab objective 1 – Decimating the data
- ✓ Lab objective 2 – Running algorithms on huge data files
- ✓ Lab objective 3 – Narrowing down algorithms for a specific need
- ✓ Lab objective 4 – Putting it all together

Overview of Problems related to Long-Term Datalogging

File Size: 64 bytes per sample at 100 samples/s for 48 hours = 1.03GB

File Analysis: Excel cannot open up a file with more than 32k entries; the above file has 17.3 million entries!

Battery Life: SD card writes + all sensors on for 48 hours create a power hungry system!

Where to store the data: pretty much has to be an SD card. Bluetooth, Wi-Fi, UART, USB are unavailable due to transit

File system: cannot use FAT32 for that file size; also too risky

Our methodology: Brute force followed by local analysis

- **Brute force:**
  - We use the microcontroller board as a dumb datalogger
  - We sample all sensors 100x per second, which results in 1GB+ for a 48 hour log

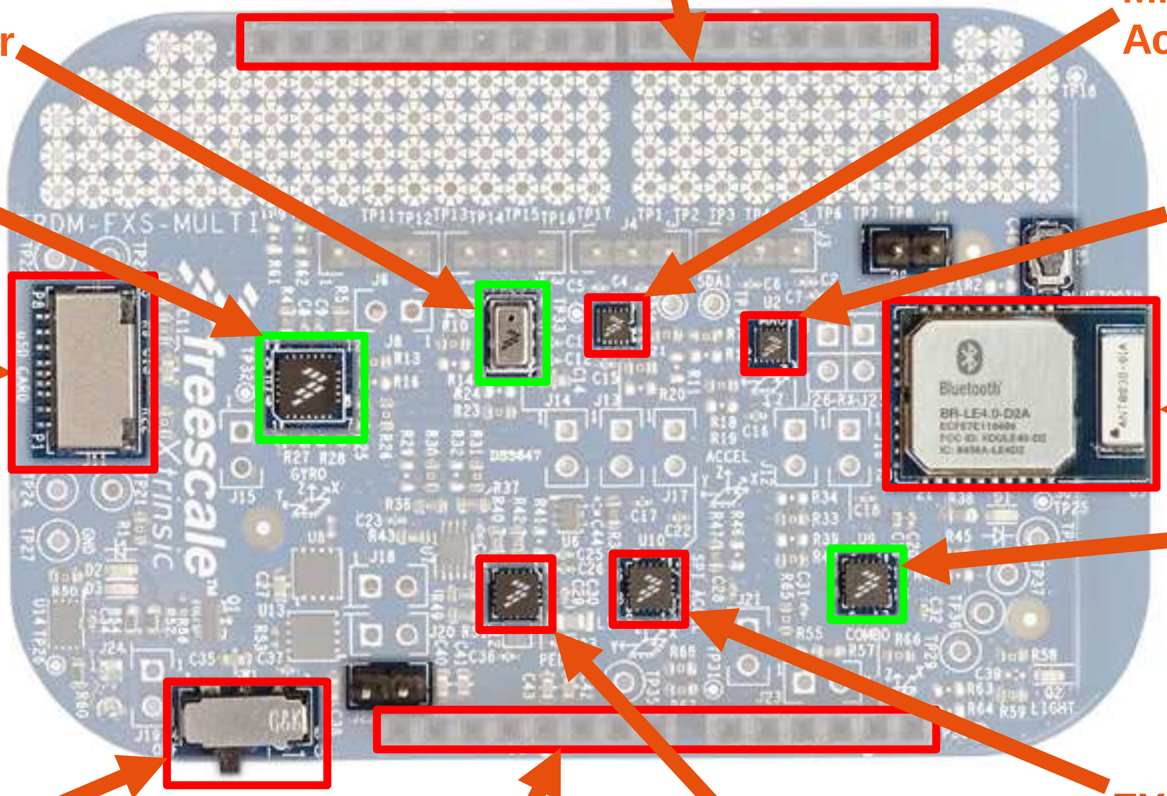
- **Our Local analysis:**
  - We store the huge files on the PC
  - We use a custom program to analyze the huge files
  - We then try multiple decimation techniques
    - Our first approach was true decimation: use 1 out of N entries
    - This simple approach lost lots of information that occurred in between decimated readings. We then moved over to a min/max/ave approach.
  - We narrow down the algorithms, we then have the ideal datalogger
  - We then port this optimized datalogger onto the microcontroller board

Our Lessons Learned

- Generating a 48-hour datalog is easier said than done
- SD cards do not write very fast in SPI mode
- We drained small batteries. No Li-Ion due to planes.
- During shipment, board can reset, SD card write can be interrupted, jumpers can temporarily disconnect, etc.
- Had to create contingencies for all the above
- Had to move away from FAT32 due to write being interrupted: used raw SD card access which required special tools to extract data
- Had to add markers to know when data stopped
- Process was very time consuming because if a mistake was made, large amounts of data had to be analyzed to find the error

FRDM-FXS-MULTI-B Hardware Overview – Used to collect long term large (.BIN) data files

FRDM-FXS-MULTI-B: Freedom Development Platform for Xtrinsic Sensors



MPL3115A2
Pressure Sensor

Arduino Expansion Header

MMA8652FC
Accelerometer

FXAS21002CQ
Gyroscope

MAG3110
Magnetometer

MicroSD Card
(Do not use this for the labs)

BT Module

FXOS8700CQ
Accelerometer +
Magnetometer

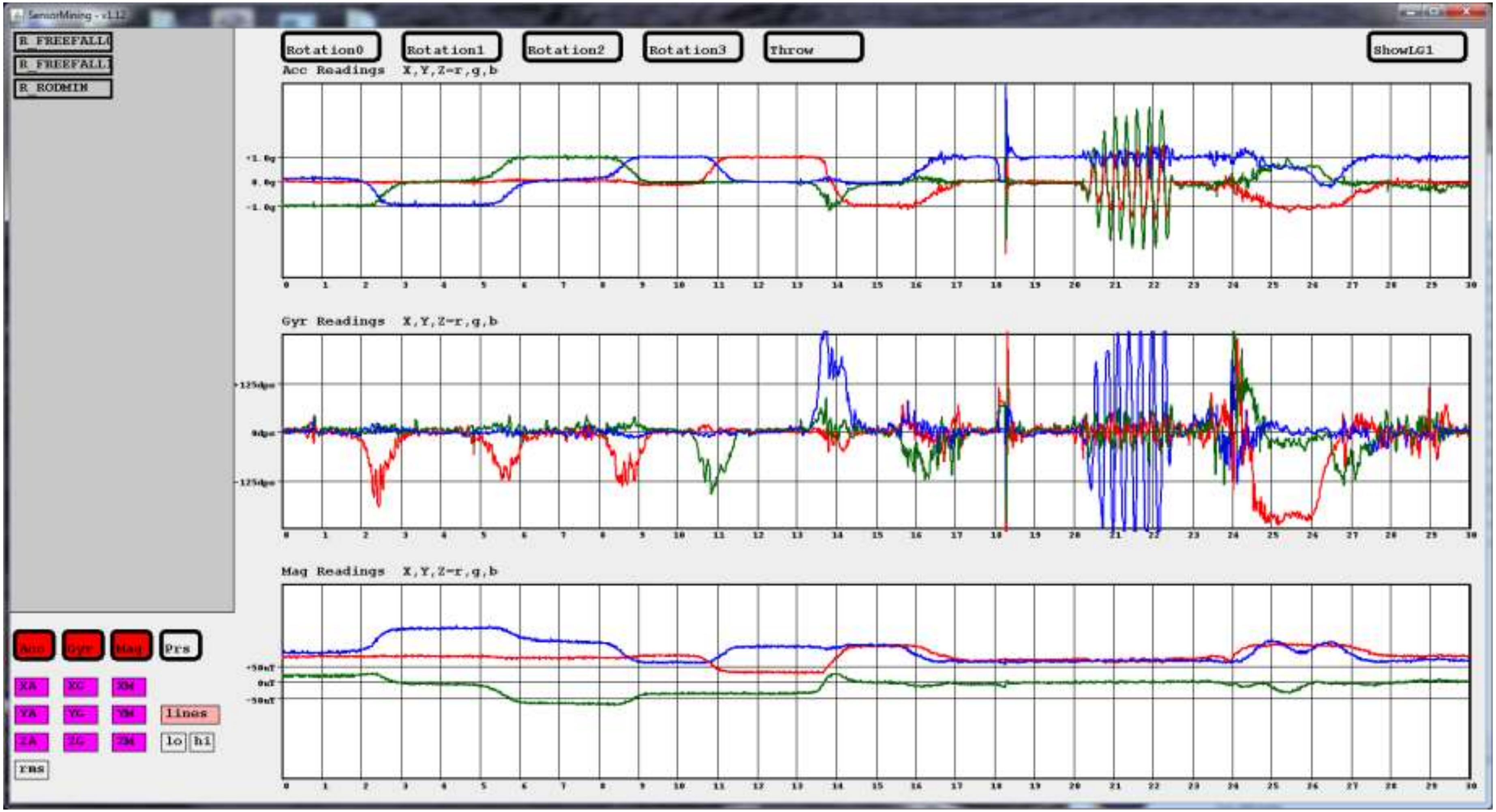
Power Switch

FXLS8471Q
SPI Accel

Arduino Expansion Header

MMA9553L
Pedometer

Sensors 101 – Explanation of Sensors used in class



Understanding the File Size

- 100 samples/s

- each sample produces 64 bytes
  - Time: 8 bytes
  - Accelerometer, Magnetometer, Gyro: 12 bytes each
  - Pressure: 8 bytes
  - Sample Number: 8 bytes
  - Log ID + line end: 4 bytes

- a 48 hour file is 100 x 64B x 3,600 x 48 = 1.03GB
  - **If printed, this represents 216,000 pages = 1.08 tons of paper!**

Why are we using a Java GUI approach????

- Traditional approach for algorithm development
  - Use the microcontroller board as a dumb datalogger, and sample all sensors 100x/per second, which results in large 1GB+ files for a 48 hour log!!

Sample data at 100x/sec for all sensors!

Generate LARGE data files 1GB+ files

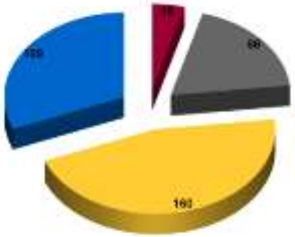
Process LARGE data files

- Our approach to algorithm development
  - Decimate the data, narrow down algorithms
  - Use Java GUI to develop algorithms and try out multiple algorithms
  - When satisfied with algorithm, download back into board

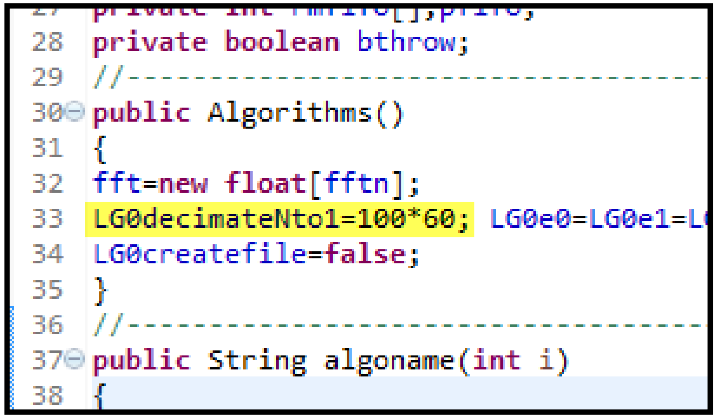
Decimate data and narrow down algorithms

Process smaller files

- Better battery life
- Easier to manage
- Traditional tools can view/analyze
- Much less time consuming

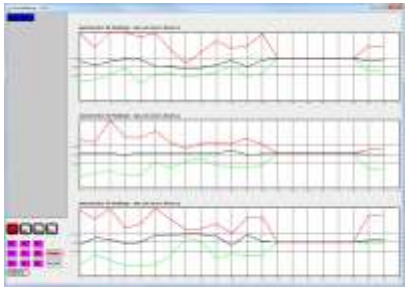


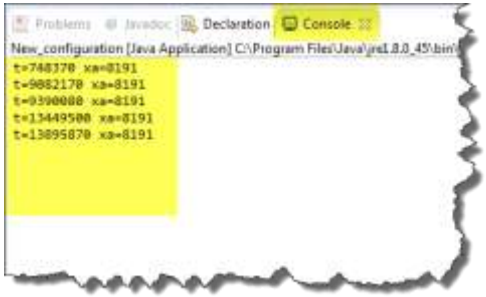
Labs - General Flow of the Labs – All labs performed in the Java based GUI – NO HARDWARE USED



```
private boolean bthrow;
//--------------------------------------
public Algorithms()
{
fft=new float[fftn];
LG0decimateNto1=100*60; LG0e0=LG0e1=L
LG0createfile=false;
}
//--------------------------------------
public String algoname(int i)
{
```

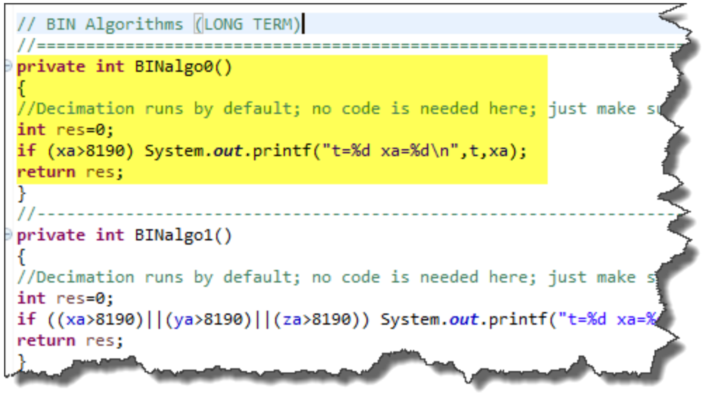
1. Decimate the data





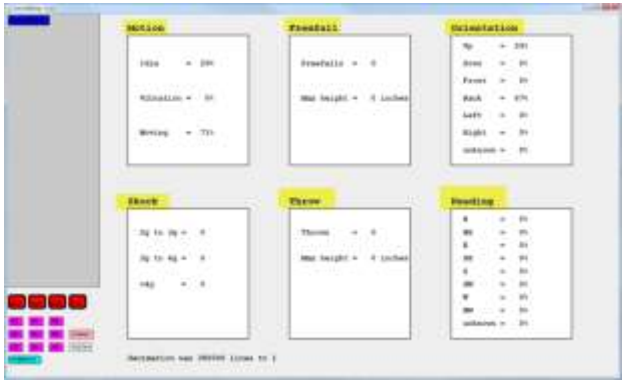
```
New_configuration [Java Application] C:\Program Files\Java\jre1.8.0_45\bin\
t=748370 xa=8191
t=9882170 xa=8191
t=9390088 xa=8191
t=13449500 xa=8191
t=13895870 xa=8191
```

2. Analyze your results



```
// BIN Algorithms (LONG TERM)
//=================================================================
private int BINalgo0()
{
//Decimation runs by default; no code is needed here; just make su
int res=0;
if (xa>8190) System.out.printf("t=%d xa=%d\n",t,xa);
return res;
}
//-----------------------------------------------------------------
private int BINalgo1()
{
//Decimation runs by default; no code is needed here; just make s
int res=0;
if ((xa>8190)||(ya>8190)||(za>8190)) System.out.printf("t=%d xa=%
return res;
}
```

3. Write a specific algorithm

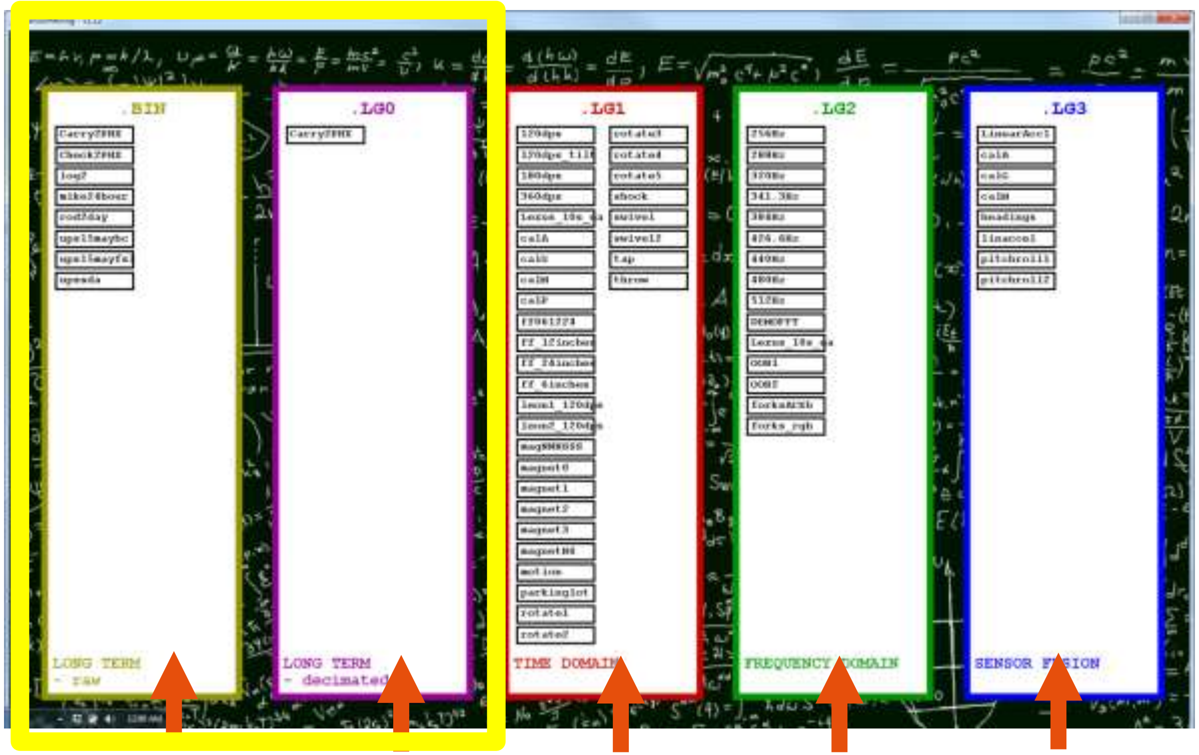


4. Check you results

SOFTWARE Description for the labs

In all the labs we will use a custom Java based GUI tool that will be referred to as “Sensor Mining Java GUI”.

A description of the opening screen:



Long Term Datalogger files Raw Values Used in Class 1 of 3

Long Term Datalogger files Pre-filtered values Used in Class 1 of 3

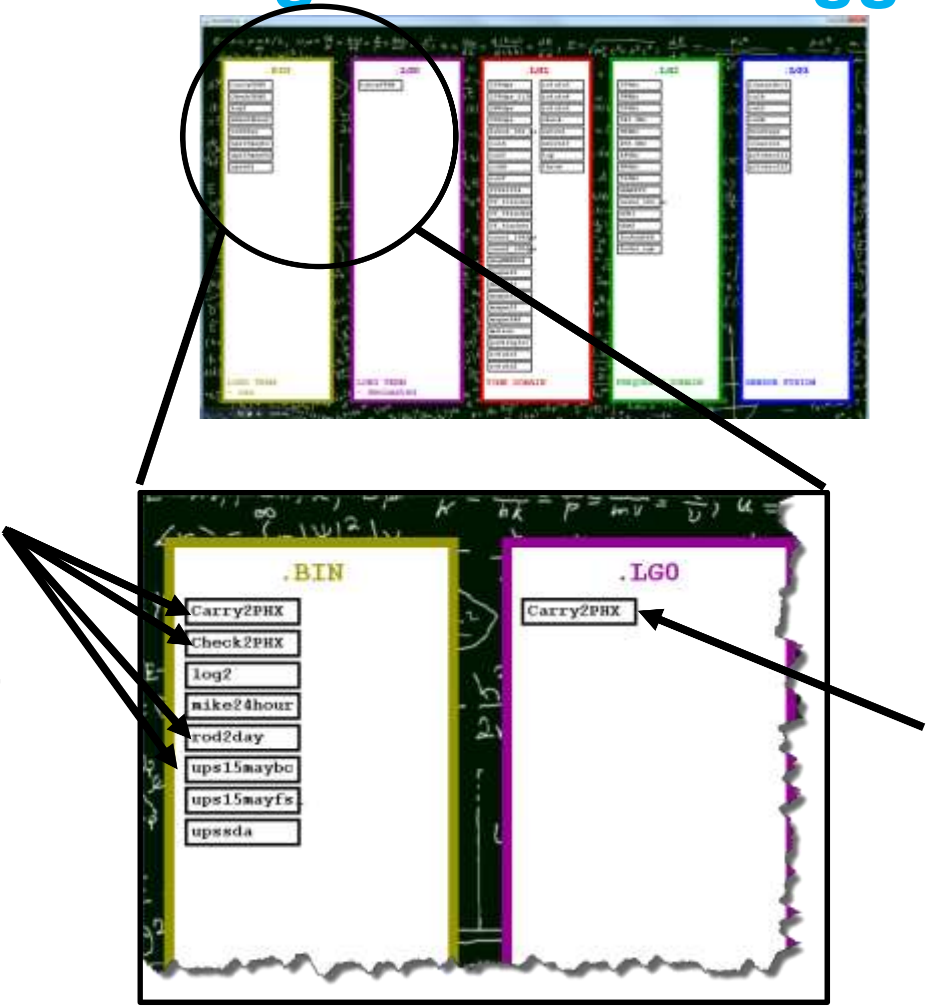
Time Domain Datalogger files Used in Class 2 of 3

Frequency Domain Datalogger files Used in Class 3 of 3

Sensor Fusion Datalogger files Used in Class 1 of 3

SOFTWARE Description for the labs

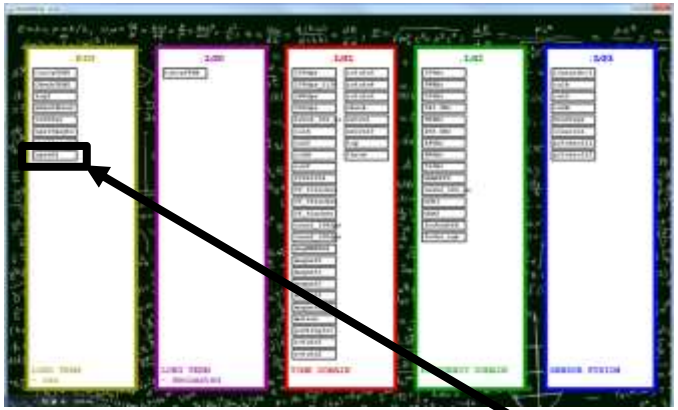
Sensor Mining Java GUI
“Long Term Data Logging”



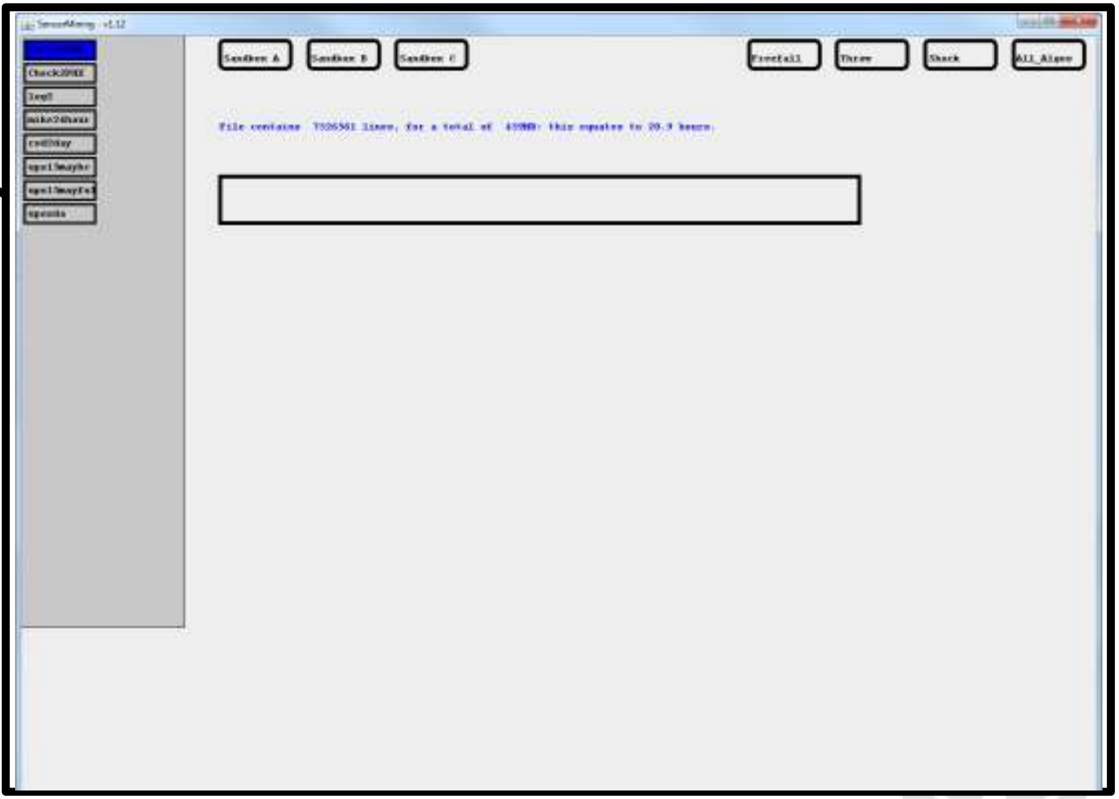
These are **large** .BIN files collected by the instructors for attendees to use in the labs.

These are **decimated files** converted in class.

SOFTWARE Description for the labs

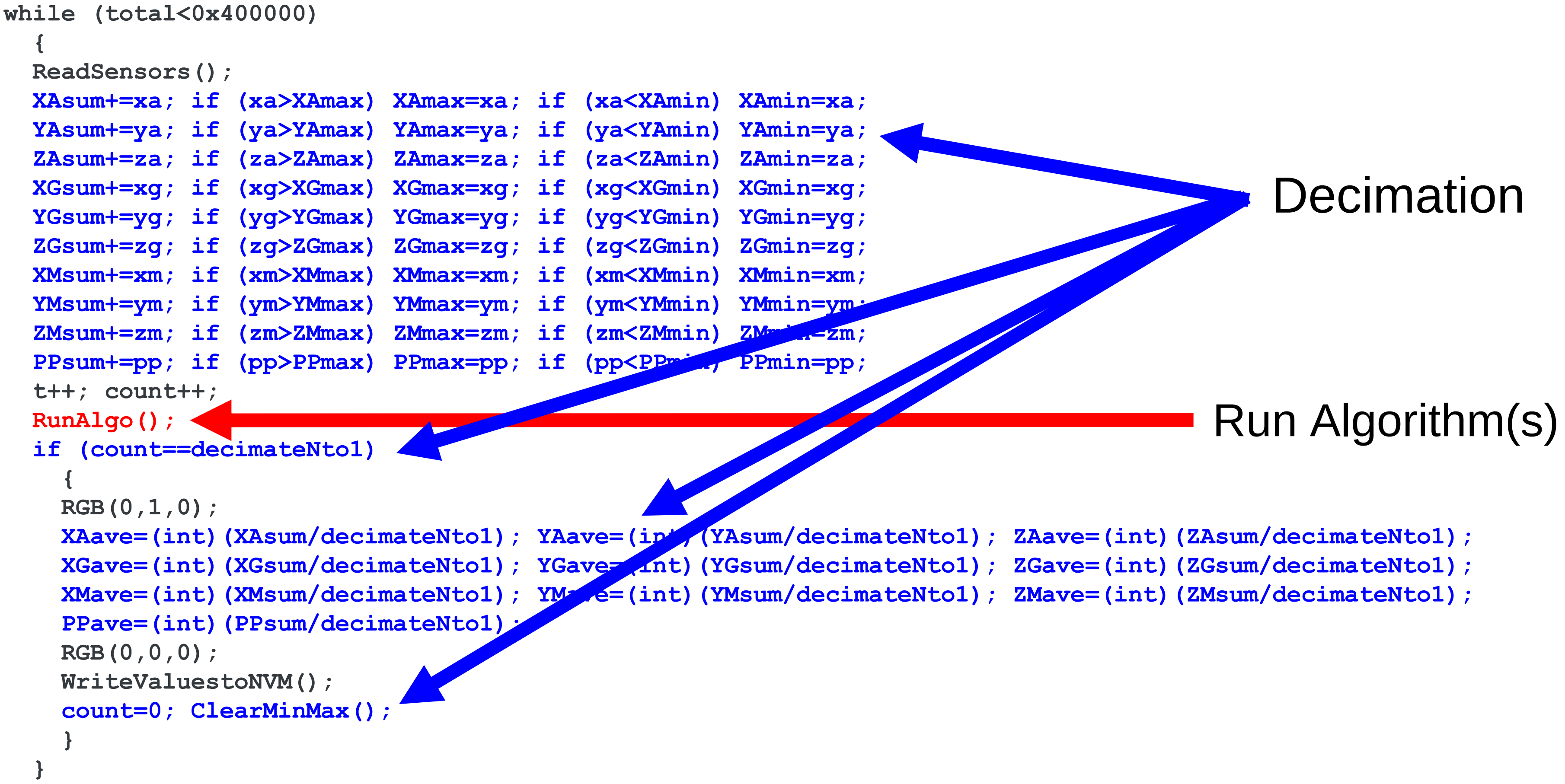


When you click on a .bin file, you open another window to execute the decimation for large .bin file.



File contains 7526561 lines, for a total of 459MB: this equates to 20.9 hours.

Let’s take a step back into the Microcontroller Embedded Code, this is the main.c code that Reads the Sensors, does the Decimation, Runs the Algorithm(s), and Programs the NVM



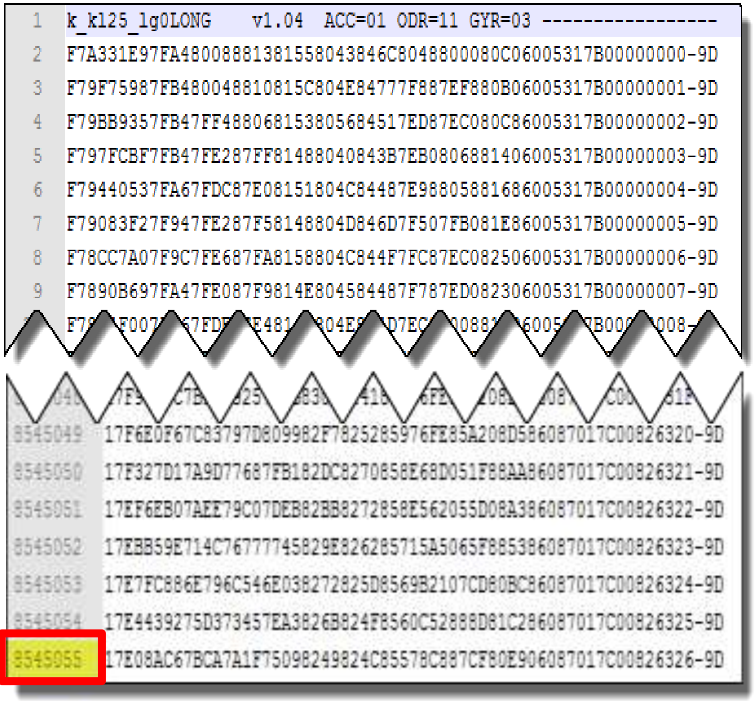
```
while (total<0x400000)
  {
  ReadSensors();
  XAsum+=xa; if (xa>XAmax) XAmax=xa; if (xa<XAmin) XAmin=xa;
  YAsum+=ya; if (ya>YAmax) YAmax=ya; if (ya<YAmin) YAmin=ya;
  ZAsum+=za; if (za>ZAmax) ZAmax=za; if (za<ZAmin) ZAmin=za;
  XGsum+=xg; if (xg>XGmax) XGmax=xg; if (xg<XGmin) XGmin=xg;
  YGsum+=yg; if (yg>YGmax) YGmax=yg; if (yg<YGmin) YGmin=yg;
  ZGsum+=zg; if (zg>ZGmax) ZGmax=zg; if (zg<ZGmin) ZGmin=zg;
  XMsum+=xm; if (xm>XMmax) XMmax=xm; if (xm<XMmin) XMmin=xm;
  YMsum+=ym; if (ym>YMmax) YMmax=ym; if (ym<YMmin) YMmin=ym;
  ZMsum+=zm; if (zm>ZMmax) ZMmax=zm; if (zm<ZMmin) ZMmin=zm;
  PPsum+=pp; if (pp>PPmax) PPmax=pp; if (pp<PPmin) PPmin=pp;
  t++; count++;
  RunAlgo();
  if (count==decimateNto1)
    {
    RGB(0,1,0);
    XAave=(int)(XAsum/decimateNto1); YAave=(int)(YAsum/decimateNto1); ZAave=(int)(ZAsum/decimateNto1);
    XGave=(int)(XGsum/decimateNto1); YGave=(int)(YGsum/decimateNto1); ZGave=(int)(ZGsum/decimateNto1);
    XMave=(int)(XMsum/decimateNto1); YMave=(int)(YMsum/decimateNto1); ZMave=(int)(ZMsum/decimateNto1);
    PPave=(int)(PPsum/decimateNto1);
    RGB(0,0,0);
    WriteValuestoNVM();
    count=0; ClearMinMax();
    }
  }
```

The code from the previous slide generates our BIN file.

Embedded Microcontroller Code from the MK64F MCU. This is the code used to try your algorithm in the Java GUI on **REAL** hardware.

```
while (total<0x400000)
  {
  ReadSensors();
  XAsum+=xa; if (xa>XAmax) XAmax=xa; if (xa<XAmin) XAmin=xa;
  YAsum+=ya; if (ya>YAmax) YAmax=ya; if (ya<YAmin) YAmin=ya;
  ZAsum+=za; if (za>ZAmax) ZAmax=za; if (za<ZAmin) ZAmin=za;
  XGsum+=xg; if (xg>XGmax) XGmax=xg; if (xg<XGmin) XGmin=xg;
  YGsum+=yg; if (yg>YGmax) YGmax=yg; if (yg<YGmin) YGmin=yg;
  ZGsum+=zg; if (zg>ZGmax) ZGmax=zg; if (zg<ZGmin) ZGmin=zg;
  XMsum+=xm; if (xm>XMmax) XMmax=xm; if (xm<XMmin) XMmin=xm;
  YMsum+=ym; if (ym>YMmax) YMmax=ym; if (ym<YMmin) YMmin=ym;
  ZMsum+=zm; if (zm>ZMmax) ZMmax=zm; if (zm<ZMmin) ZMmin=zm;
  PPsum+=pp; if (pp>PPmax) PPmax=pp; if (pp<PPmin) PPmin=pp;
  t++; count++;
  RunAlgo();
  if (count==decimateNto1)
    {
    RGB(0,1,0);
    XAave=(int)(XAsum/decimateNto1); YAave=(int)(YAsum/decimateNto1); ZAave=(int)(ZAsum/decimateNto1);
    XGave=(int)(XGsum/decimateNto1); YGave=(int)(YGsum/decimateNto1); ZGave=(int)(ZGsum/decimateNto1);
    XMave=(int)(XMsum/decimateNto1); YMave=(int)(YMsum/decimateNto1); ZMave=(int)(ZMsum/decimateNto1);
    PPave=(int)(PPsum/decimateNto1);
    RGB(0,0,0);
    WriteValuestoNVM();
    count=0; ClearMinMax();
    }
  }
```

Captured Raw Datalog
from .BIN files

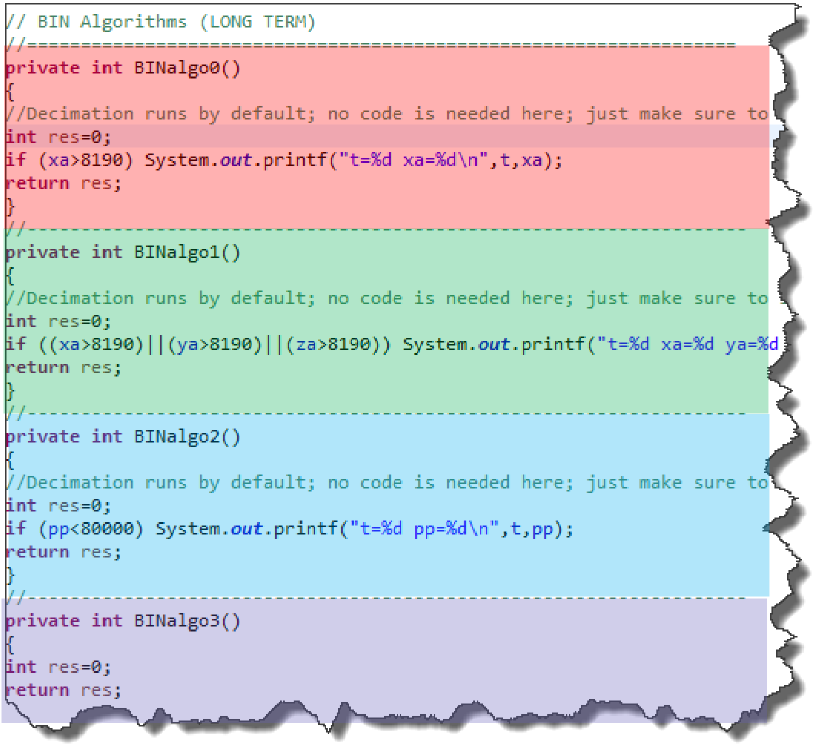


```
      1  k_k125_lg0LONG    v1.04  ACC=01 ODR=11 GYR=03 -----------------
      2  F7A331E97FA48008881381558043846C8048800080C06005317B00000000-9D
      3  F79F75987FB480048810815C804E84777F887EF880B06005317B00000001-9D
      4  F79BB9357FB47FF4880681538056845 17ED87EC080C86005317B00000002-9D
      5  F797FCBF7FB47FE287FF81488040843B7EB0806881406005317B00000003-9D
      6  F79440537FA67FDC87E08151804C84487E98805881686005317B00000004-9D
      7  F79083F27F947FE287F58148804D846D7F507FB081E86005317B00000005-9D
      8  F78CC7A07F9C7FE687FA8158804C844F7FC87EC08250600531 7B00000006-9D
      9  F7890B697FA47FE087F9814E804584487F787ED08230600531 7B00000007-9D
         F78 F007 67FDF E481 804E8 D7EC 00881 6005 7B000 008-
         046 7F9 C7B 825 B83 418 6FE 2081 208 C00 81F
8545049  17F6E0F67C83797D809982F782528597 6FE85A208D58608701 7C00826320-9D
8545050  17F327D17A9D77687FB182DC8270858E68D051F88AA86087017C00826321-9D
8545051  17EF6EB07AEE79C07DEB82BB8272858E562055D08A386087017C00826322-9D
8545052  17EBB59E714C76777745829E8262857 15A5065F88538608701 7C00826323-9D
8545053  17E7FC886E796C546E038272825D8569B2107CD80BC86087017C00826324-9D
8545054  17E4439275D373457EA3826B824F8560C52888D81C286087017C00826325-9D
8545055  17E08AC67BCA7A1F75098249824C85578C887CF80E906087017C00826326-9D
```

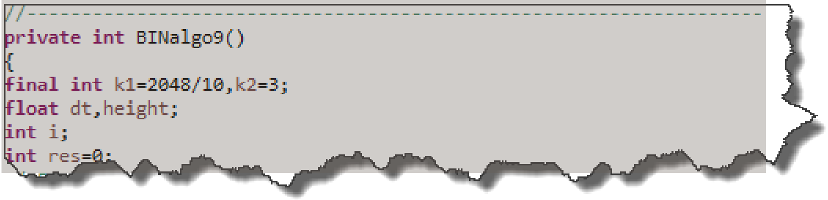
Brute force method of just capturing all sensors at 100x/second for an overnight datalog event yielded a HUGE file with over a million lines of code!!!!!! Some files over 1GB+ in size, traditional editors (Notepad++, Excel) would not even open the files!!

8,545,055 ENTRIES!

Let’s jump back into the Java GUI algorithms…



```
// BIN Algorithms (LONG TERM)
//=====================================================================
private int BINalgo0()
{
//Decimation runs by default; no code is needed here; just make sure to
int res=0;
if (xa>8190) System.out.printf("t=%d xa=%d\n",t,xa);
return res;
}
//--------------------------------------------------------------------
private int BINalgo1()
{
//Decimation runs by default; no code is needed here; just make sure to
int res=0;
if ((xa>8190)||(ya>8190)||(za>8190)) System.out.printf("t=%d xa=%d ya=%d
return res;
}
//--------------------------------------------------------------------
private int BINalgo2()
{
//Decimation runs by default; no code is needed here; just make sure to
int res=0;
if (pp<80000) System.out.printf("t=%d pp=%d\n",t,pp);
return res;
}
//--------------------------------------------------------------------
private int BINalgo3()
{
int res=0;
return res;
```



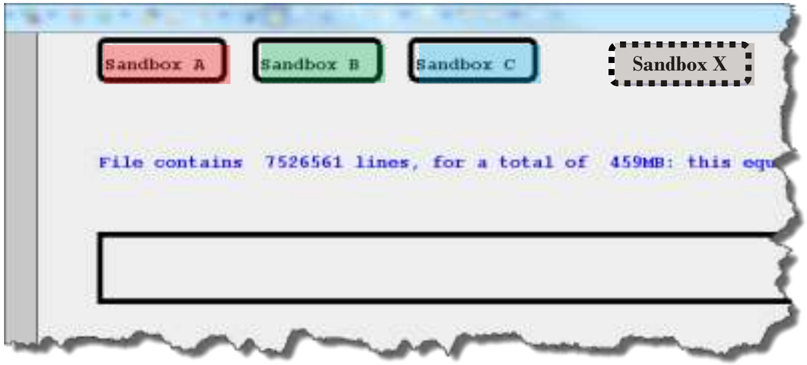
```
//--------------------------------------------------------------------
private int BINalgo9()
{
final int k1=2048/10,k2=3;
float dt,height;
int i;
int res=0;
```

In the labs, when you write your Algorithms, you will get to assign the button names for up to ten buttons (ten algorithms), in the examples below:

BINalgo0() is given the button name “Sandbox A”
BINalgo1() is given the button name “Sandbox B”
BINalgo2() is given the button name “Sandbox C”

⋮

BINalgo9() is given the button name “Sandbox X”

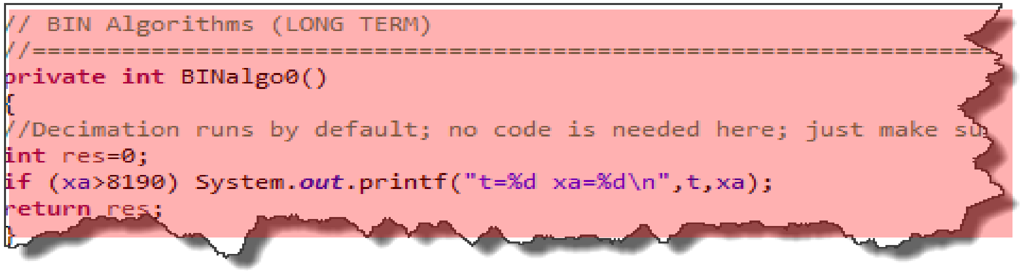


File contains 7526561 lines, for a total of 459MB: this equ

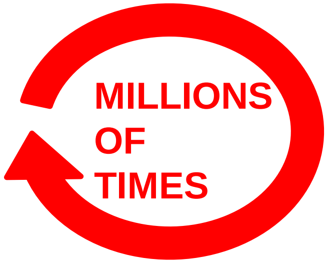
How does the Java Algorithm run on the Sensor Mining GUI?

- Here is the most important part of understanding the Sensor Mining GUI…

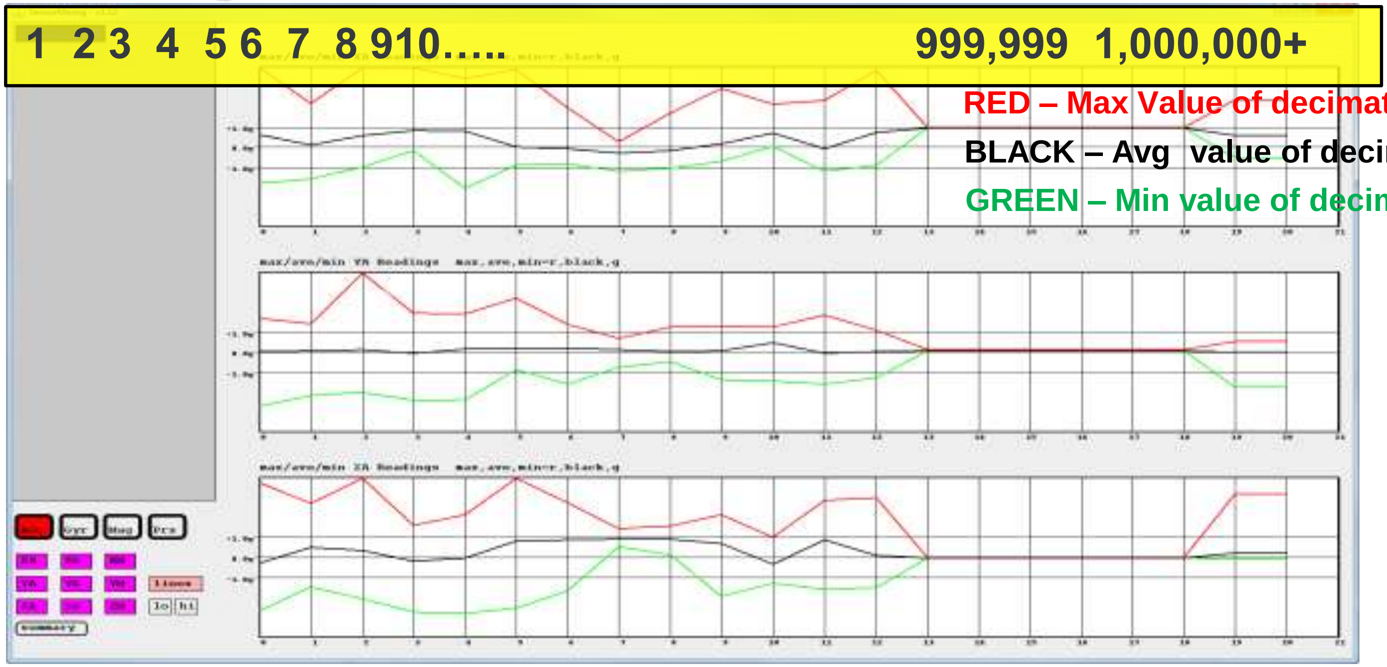
ALGORITHMS WILL EXECUTE Millions of times when the button is pressed!



```
// BIN Algorithms (LONG TERM)
//===============================================================
private int BINalgo0()
{
//Decimation runs by default; no code is needed here; just make su
int res=0;
if (xa>8190) System.out.printf("t=%d xa=%d\n",t,xa);
return res;
}
```



MILLIONS OF TIMES



1 2 3 4 5 6 7 8 910….. 999,999 1,000,000+

RED – Max Value of decimated data

BLACK – Avg value of decimated data

GREEN – Min value of decimated data

max/ave/min YA Readings max,ave,min=r,black,g
max/ave/min ZA Readings max,ave,min=r,black,g
Acc
Gyr
Mag
Prs
Summary
lo
hi

Long-Term Raw Datalog (BIN) File Structure

```
k_kl25_lg0LONG    v1.04  ACC=01 ODR=11 GYR=03 -----------------
F7A331E97FA48008881381558043846C8048800080C06005317B00000000-9D
F79F75987FB480048810815C804E84777F887EF880B06005317B00000001-9D
F79BB9357FB47FF48806815380568[illegible]517ED87EC080C86005317B00000002-9D
```

Raw values for each axis, in hex

```
Tim: F7A331E9
Acc: 7FA4 8008 8813
Mag: 8155 8043 846C
Gyr: 8048 8000 80C0
Prs: 6005317B
N:   00000000
Ite: -9D
```

Long-Term Decimated Datalog (LG0) File Structure

```
Long-Term Smart Datalogger v1.00 -----------------
000005DC76667B798090761D7E4F82057F3785A4871635F47F77B6FC15A07D7B99195ACF832DFB36829483B68429826382F983E27D837E007F6C08456084B A0851A000000000000002904E0
00000BB8752280148C6D740C7CDB81FC741784C28BFA000083BEFFFF00008C92FFFF00007A50D3D381AE82BC8467823B82C483C67D127E4281AF084B0084E 608550000000000000002F04FF
0000119470B97A6E90FC68457DAA932870C28007935200007DFBFFFF00007E9AF   0008021FFFF829D83BE844A813F82D683C67D297EF28055084DB0850 608577000000000501012B04FF
000017707 4F67E98847075A37B2D825F7FE685908AD100008230FFFF00008163E   2767EC0C2E5826E830A83C88326837083AC7D7E7E2A7F6C084F10855 4085A300000000000000002904E0
00001D4C7A447D427E5975E67C387DD07DB385B387455B868115A85B5A047E109   86A80BCB3C382D38346838583188355 83C17D657DE97F80085260857 F085F600000000000000290460
00002328746E7B0E8748743C81018B897DB284799D14114D8272FFFF0000810BE   00080F2FFFF82158392844C813E825A836A7D077DBD7EEF0851E0859 C086D3000000000402011904FF
00002904751A7E848B0D739D7EE28A4D7C2B86CA90C000008211FFFF00007E31F   50D7EDFD5778277833884298211 82EB83AE7D1D7D877F78085020858 5085EB000000000100002904F9
```

Min/Ave/Max values for each axis, in hex

Tim: 000005DC

Acc: 7666 7B79 8090 761D 7E4F 8205 7F37 85A4 8716

Gyr: 35F4 7F77 B6FC 15A0 7D7B 9919 5ACF 832D FB36

Mag: 8294 83B6 8429 8263 82F9 83E2 7D83 7E00 7F6C

Prs: 08456 084BA 0851A

Ext: 00 00 00 00 00 00 00 29 04 E0

Algorithm Timing on a Cortex-M0+ (KL27) @ 24MHz

Sensors are read 100x per second: **10ms intervals**

Decimation	66us	
Freefall/Throw	5us	
Shock	4us	
Orientation	5us	
Motion	3us	
Heading	2,807us	(probably should be decimated)
Algorithms-total	**<3ms**	

Q&A

- ✓ Did you understand the class objectives?
- ✓ Do you have a better understanding of how to create a smart long term datalogger?
- ✓ Did you learn a few things from our mistakes and resulting methodology?

freescale™

www.Freescale.com