Hands-On Workshop: Sensor Data Collection and Mining: Intelligent Data Loggers

AMF-INS-T1702

Rod Borras | Freescale AMR FAE
Michael Steffen | Freescale AMR FAE

OCT.2015

External Use

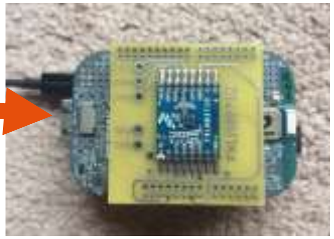
Hands-on: Collect data in the time domain and test and validate it on the custom Java GUI.

- Lab objective 1 – Review steps involved to collect data at different stations pertaining to the time domain.
- Lab objective 2 – Collect data at the stations – only at the time domain stations
- Lab objective 3 – Test you algorithms written on using the custom Java GUI!

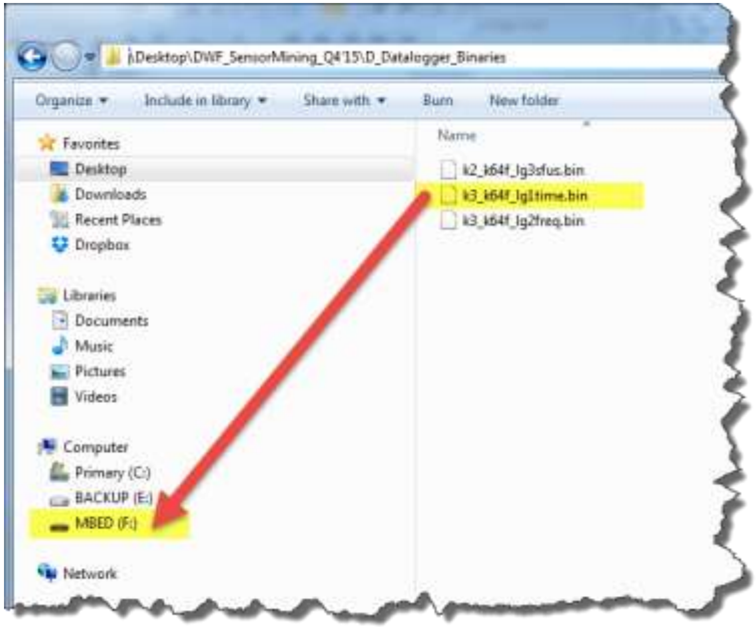
PLEASE REMEMBER THIS LAB GUIDE IS AN INSTRUCTION GUIDE not an exact step-by-step guide.

Programming your Hardware Board

1. Plug in board to micro USB port labeled “OpenSDA”…



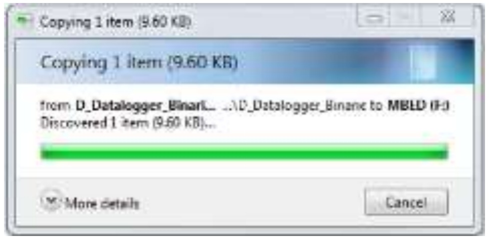
2. Open Windows Explorer and migrate to the desktop folder… *Desktop\DWF_SensorMining_Q4'15 \D_Datalogger_Binaries*



3. You should see a **MBED** (drive) letter.
   Please let instructor know if you don’t get a drive called “**MBED**”.

4. Drag and drop the file “*k3_k64f_lg1time.bin*” on top of the **MBED** drive.

5. You should see a progress screen…



Copying 1 item (9.60 KB)

Copying 1 item (9.60 KB)

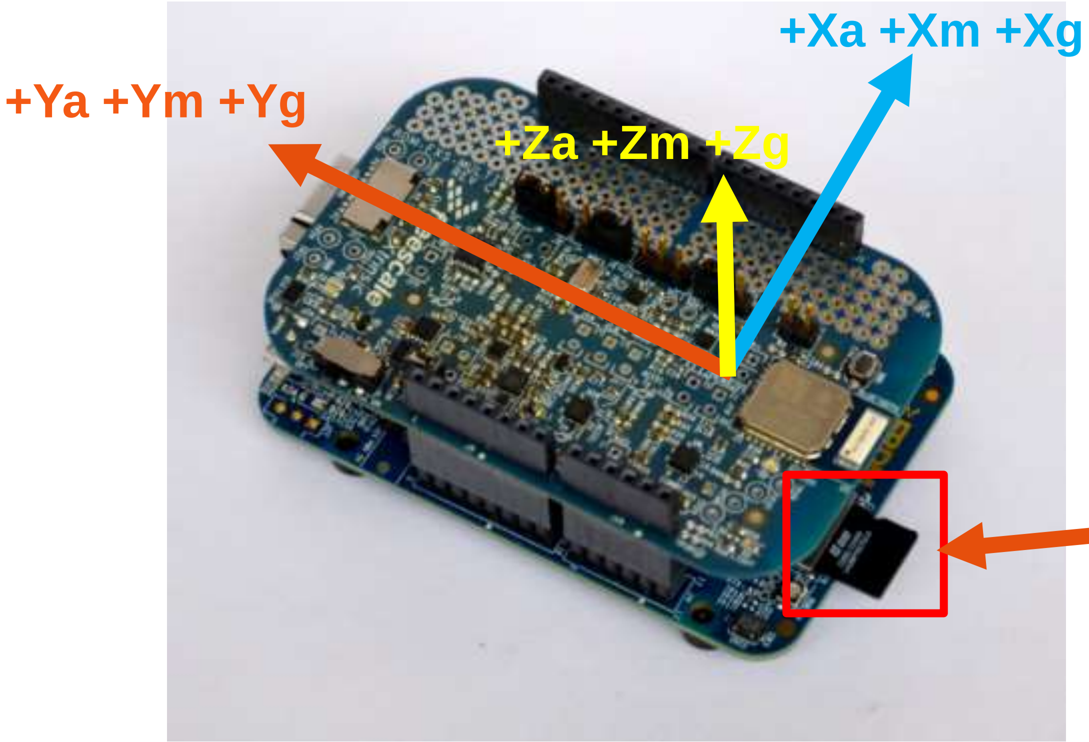
from **D_Datalogger_Binari...** ...\D_Datalogger_Binaries to **MBED (F:)**
Discovered 1 item (9.60 KB)...

6. Programming complete!

HARDWARE Board Description

Hardware Datalogger Board for the TIME DOMAIN LABS:

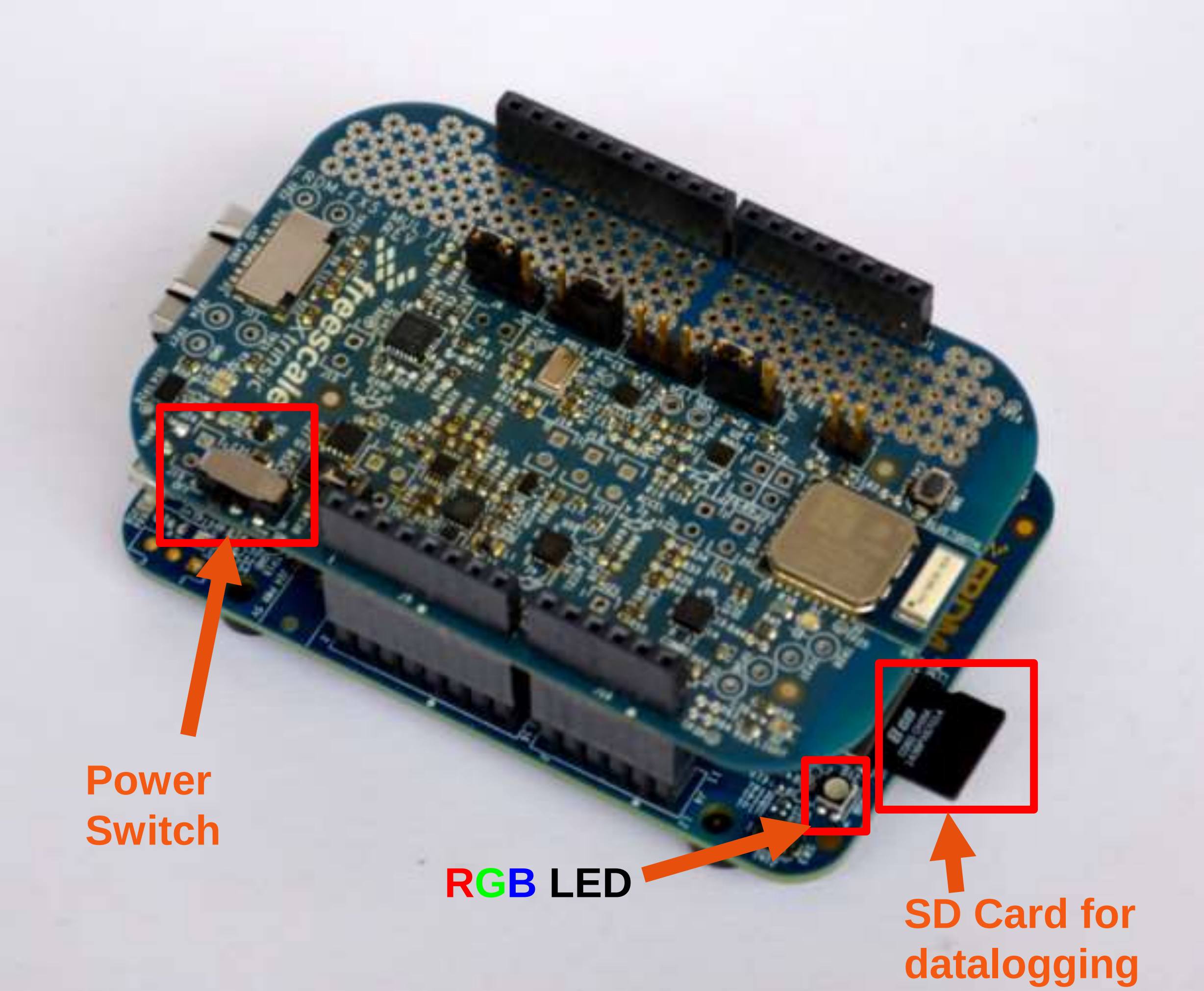
- FRDM-K64F board + FRDM-FXS-MULTI-B



All Sensors AXIS orientation
a = Accelerometer
m = Magnetometer
g = Gyroscope

SD Card Slot used for datalogging

INSTRUCTION - Starting the datalogger



To start the datalogger:

1. Insert SD Card into holder

2. Turn on Power switch (might have to cycle power if RED LED doesn't come on)

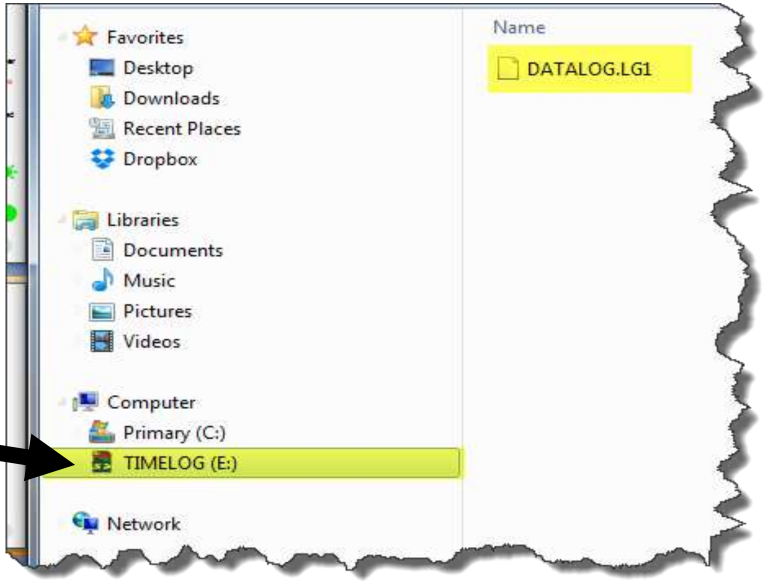
3. RGB LED will go solid red for 5 seconds 🔴

4. Begin Datalogging when RGB LED blinks green

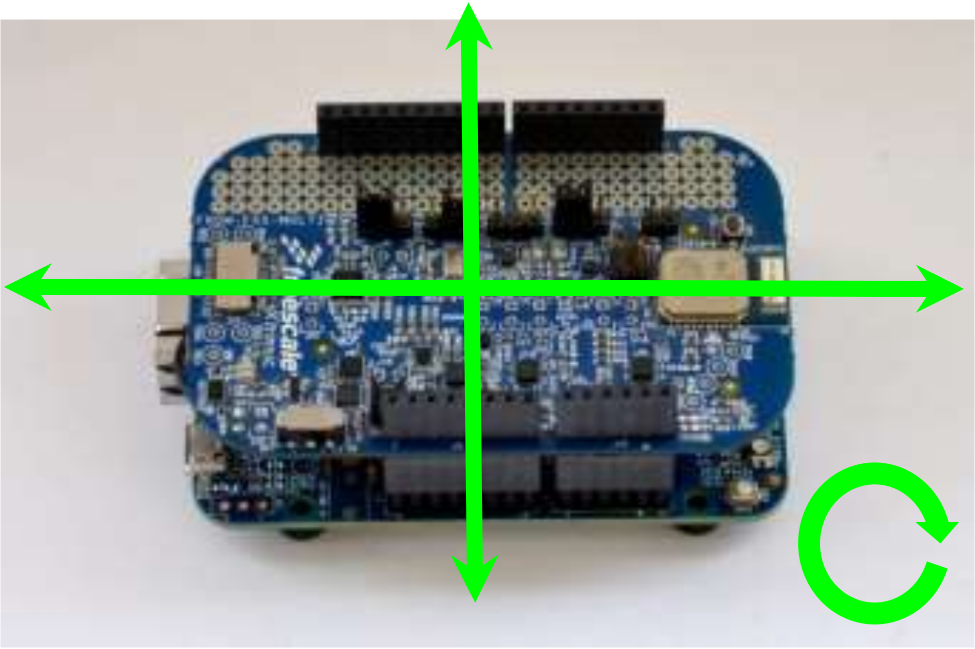
5. Datalogging stops after 30 seconds, when RGB LED goes solid green 🟢

INSTRUCTION - Copying datalogger files onto laptop

1. Turn off Datalogger.
2. Remove micro SDCard from hardware board.
3. Insert micro SDCard into adapter and insert into laptop SDCard slot.
4. The SDCard will show up as “TIMELOG (drive):
5. Copy “DATALOG.LG1” to location: *C:\Desktop\DWF_SensorMining_Q4'15\C_Sensor_Logs* (***Please do not add any additional folders***)
6. Rename “DATALOG.LG1” to any name with up to 9 characters, no spaces, no special characters. Do not change the extension.



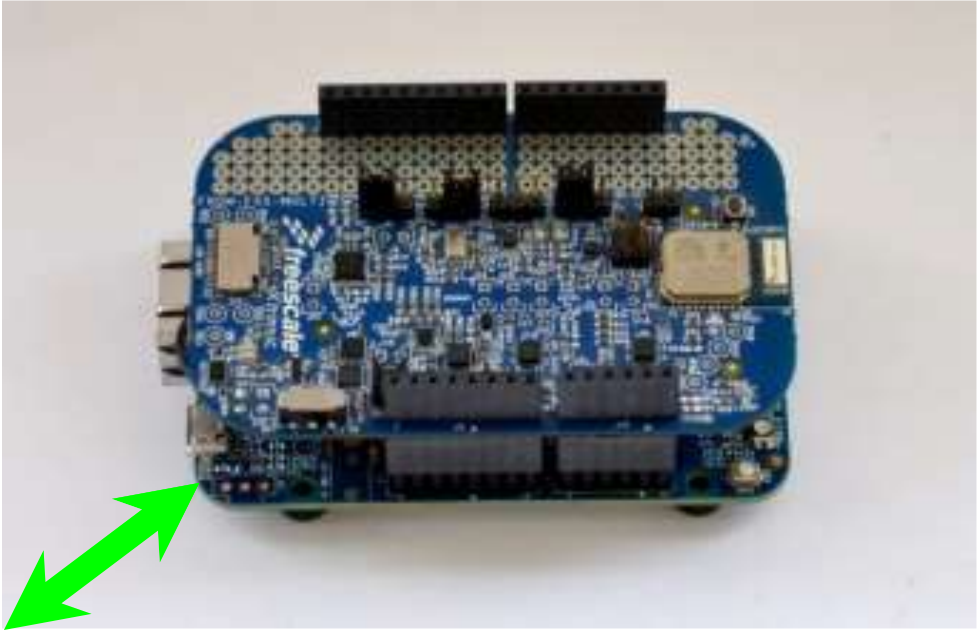
Lab Stations 1 - Motion Lab



1. Start Datalogger.
2. Gently move the board around, turn it upside down, move it back and forth, rotate it. Set it on table, pick it up. **Do not drop or throw board.**
3. When LED goes solid green, take back to desk and copy file onto laptop.

Lab Station #1 - This is done at your desk

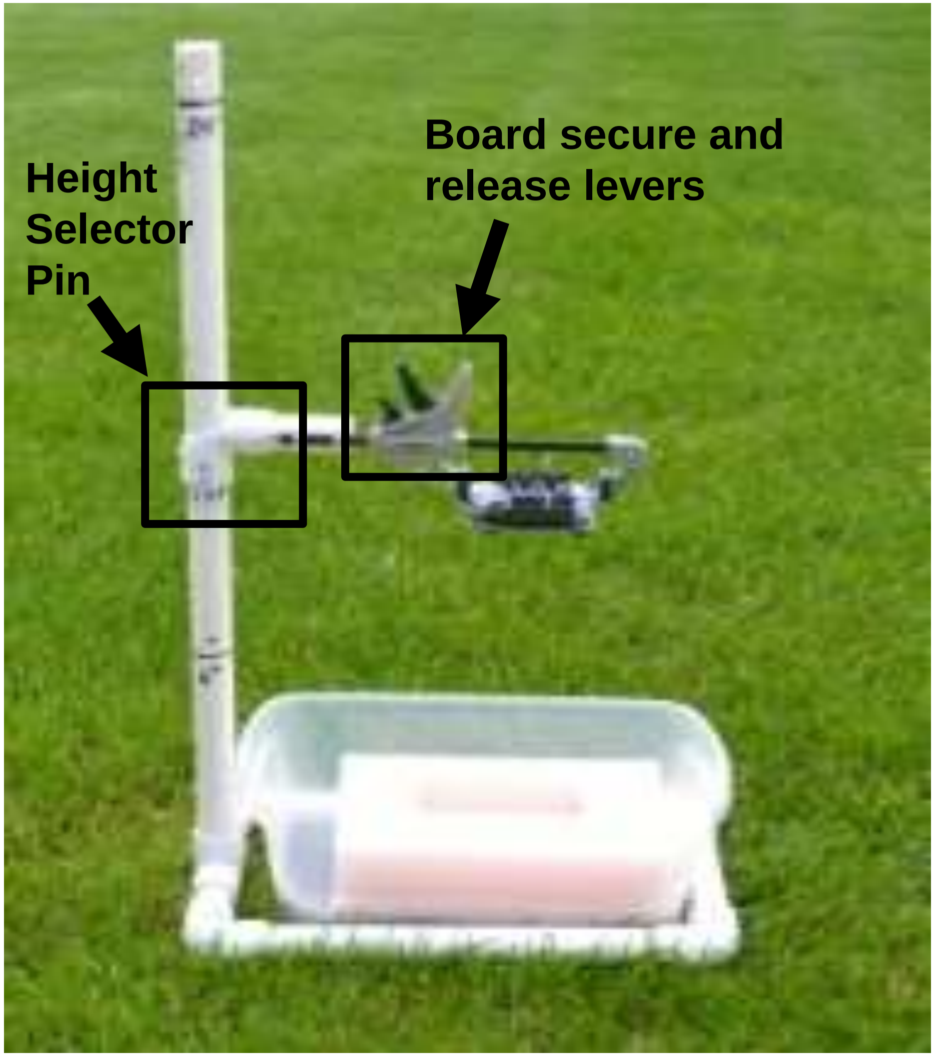
Lab Station 2 – Shock Lab



1. Start Datalogger.
2. Gently take the board and tap the corner on the desk several times. Set it on table, and repeat a few more times.
3. **Do not drop or throw board.**
4. When LED goes solid green, take back to desk and copy file onto laptop.

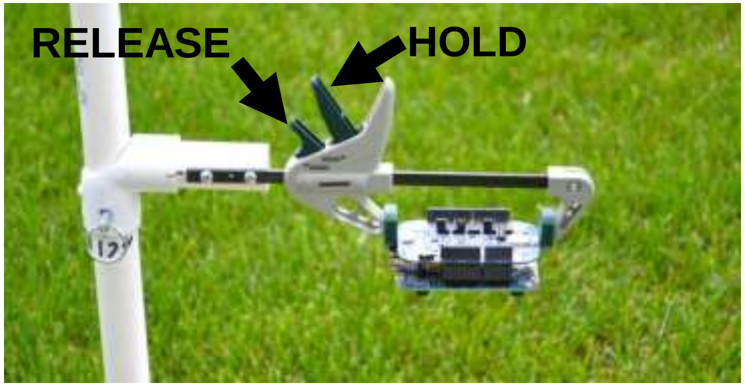
Lab Station #2 - This is done at your desk

Lab Station 3 – Freefall Lab



- **Freefall Apparatus**

1. Insert board into clamp, **do not squeeze clamp TIGHT!!!**

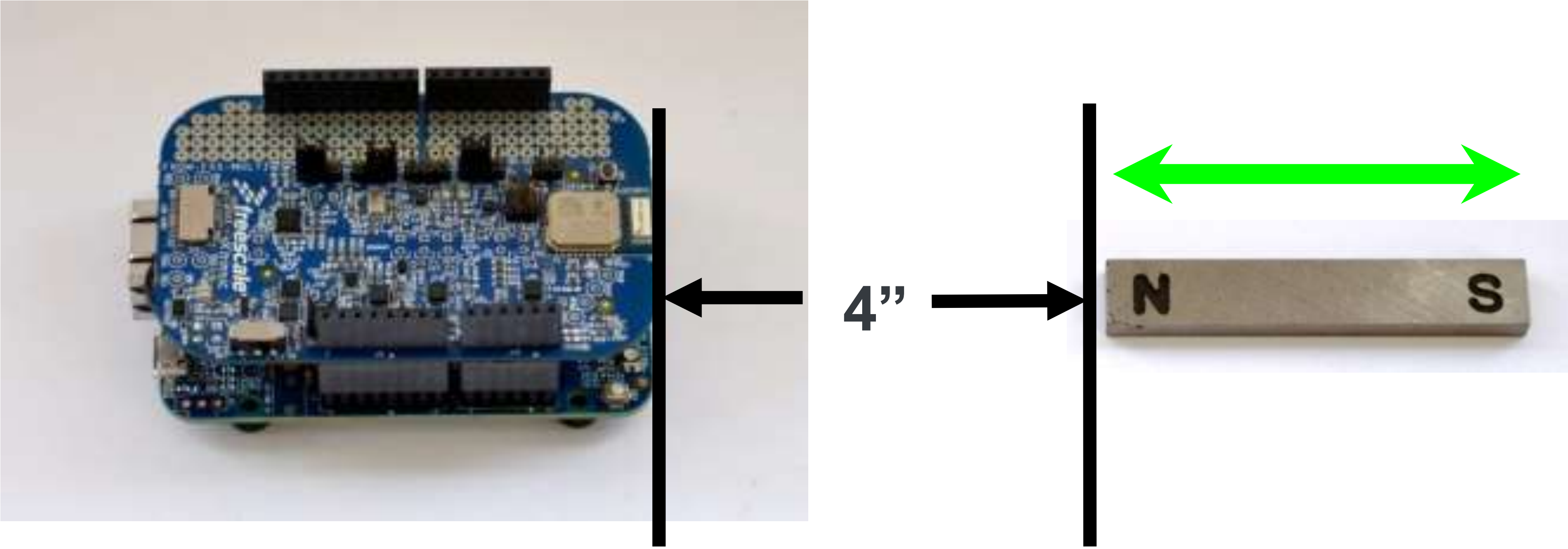


2. **Start Datalogger**
3. **Drop board the board by squeezing the release level.**
4. **Do not turn off board and repeat for another height.**
5. When LED goes solid green, take back to desk and copy file onto laptop.

Lab Station 4 – Magnets Lab

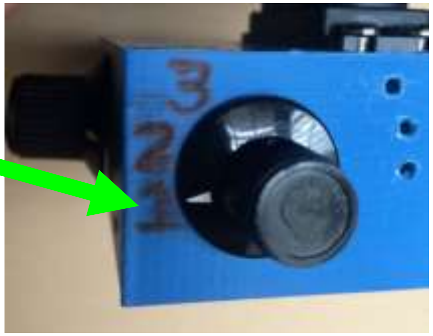
- Magnets – **DO NOT BRING MAGNETS BACK TO YOUR DESK**

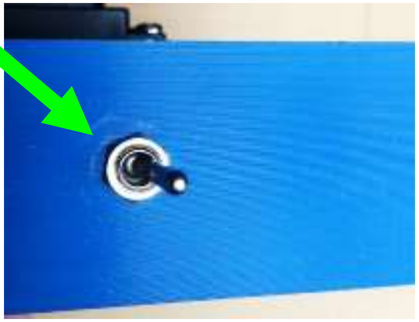
1. Orient the board and magnet like the picture below. Start Datalogger.
2. Bring Magnet in Proximity to board – about a hand size separation in distance.
3. Move the magnet back and forth slowly away and towards the board.
4. Repeat for North and South Pole sides of the magnet.
5. When LED goes solid green, take back to desk and copy file onto laptop.

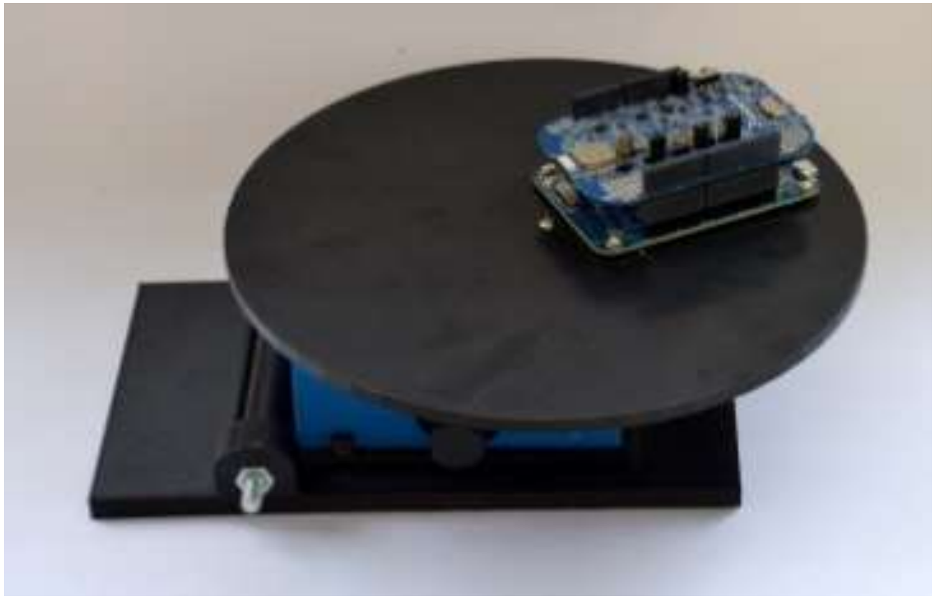


Lab Station 5 – Rotation Lab

1. Orient the board like the picture below. The feet of the board fit into the holes on the platter. Turn the speed switch to position #1.
2. Start the datalogger.
3. Turn on the power switch to and let it datalog for 10 seconds at each speed of 1,2,3.
4. When LED goes solid green, take back to desk and copy file onto laptop.



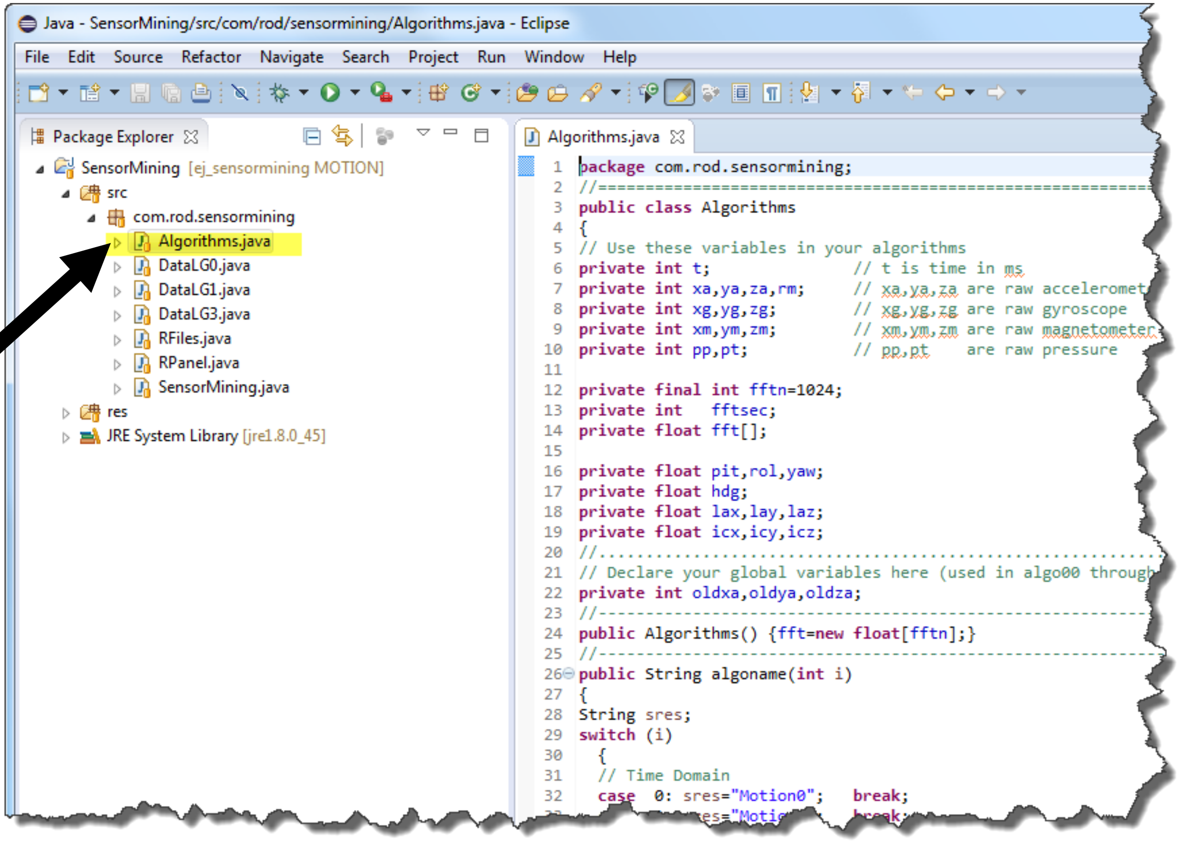




Open the Eclipse Java GUI

1. Open Eclipse Mars on the Desktop, or in the DWF_SensorMining_Q4'15\A_Installers

2. In the Java "Package Explorer" Window, drop down, SRC, and com.rod.sensormining. Double-Click in the file called "Algorithms.java"
   NOTE: This is the ONLY file you can modify any source code!



Java - SensorMining/src/com/rod/sensormining/Algorithms.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
SensorMining [ej_sensormining MOTION]
src
com.rod.sensormining
Algorithms.java
DataLG0.java
DataLG1.java
DataLG3.java
RFiles.java
RPanel.java
SensorMining.java
res
JRE System Library [jre1.8.0_45]
Algorithms.java

```
package com.rod.sensormining;
//==========================================================
public class Algorithms
{
// Use these variables in your algorithms
private int t;                 // t is time in ms
private int xa,ya,za,rm;       // xa,ya,za are raw acceleromet
private int xg,yg,zg;          // xg,yg,zg are raw gyroscope
private int xm,ym,zm;          // xm,ym,zm are raw magnetometer
private int pp,pt;             // pp,pt    are raw pressure

private final int fftn=1024;
private int   fftsec;
private float fft[];

private float pit,rol,yaw;
private float hdg;
private float lax,lay,laz;
private float icx,icy,icz;
//..........................................................
// Declare your global variables here (used in algo00 through
private int oldxa,oldya,oldza;
//----------------------------------------------------------
public Algorithms() {fft=new float[fftn];}
//----------------------------------------------------------
public String algoname(int i)
{
String sres;
switch (i)
  {
  // Time Domain
  case  0: sres="Motion0";   break;
```

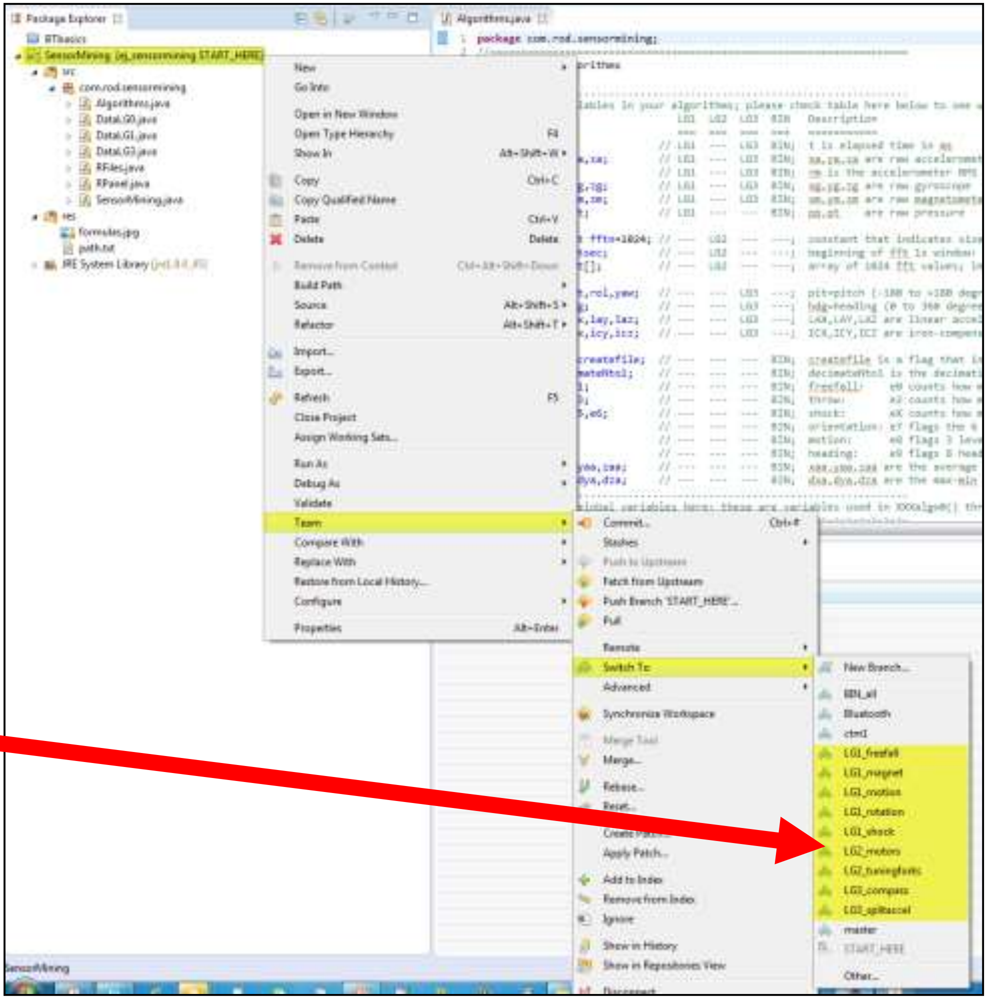
How to switch between the labs in this class

We will have 5 labs with several Algorithms in each lab to explore.

The Time Domain Labs are:

1. LG1_motion
2. LG1_shock
3. LG1_freefall
4. LG1_rotation
5. LG1_magnet

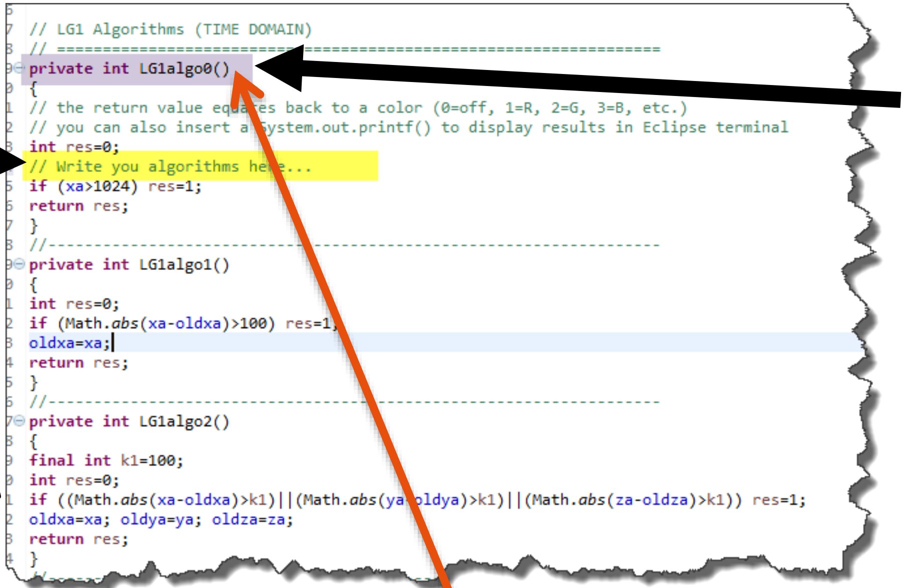
- To switch between them, right click on the *“SensorMining”* Project and select ***Team / Switch To / LG1,LG2,LG3*** (for example, anything in yellow), for the first lab of today we are going to start with LG1_motion.
- Next, Hit CTRL-F11 or Menu Run/Run (to run Algorithms).



Type you Algorithms here…and assign a button to it!

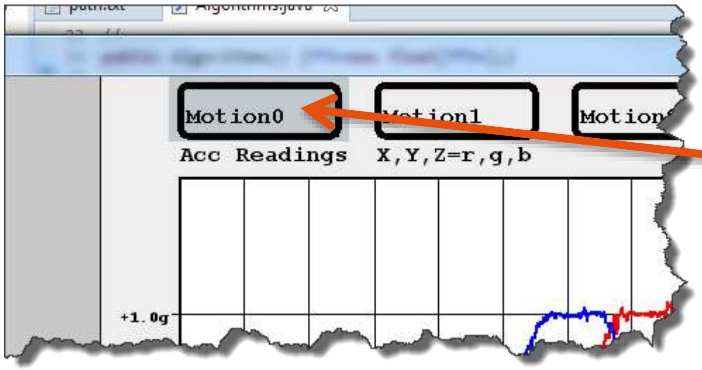
In the File, *Algorithms.java*, this is where you can type you algorithms **OR** observe and study the examples already done.

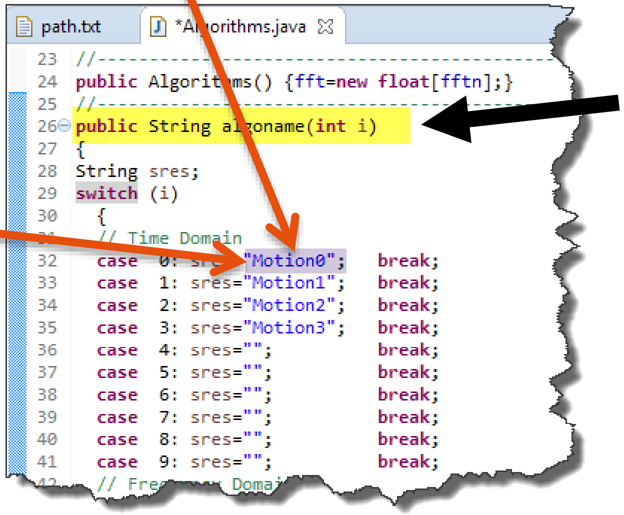
This slide is an example of how to re-assign buttons:
This applies to all labs:
LG1 – Time Domain
LG2 – Frequency Domain
LG3 – Sensor Fusion Domain



```java
// LG1 Algorithms (TIME DOMAIN)
// ===================================================================
private int LG1algo0()
{
// the return value equates back to a color (0=off, 1=R, 2=G, 3=B, etc.)
// you can also insert a System.out.printf() to display results in Eclipse terminal
int res=0;
// Write you algorithms here...
if (xa>1024) res=1;
return res;
}
//-------------------------------------------------------------------
private int LG1algo1()
{
int res=0;
if (Math.abs(xa-oldxa)>100) res=1;
oldxa=xa;
return res;
}
//-------------------------------------------------------------------
private int LG1algo2()
{
final int k1=100;
int res=0;
if ((Math.abs(xa-oldxa)>k1)||(Math.abs(ya-oldya)>k1)||(Math.abs(za-oldza)>k1)) res=1;
oldxa=xa; oldya=ya; oldza=za;
return res;
}
```

Notice the function:
private int LG1algo0():
This is the name of the algorithm that can be assigned to a button.
THIS IS AN EXAMPLE.





path.txt | *Algorithms.java

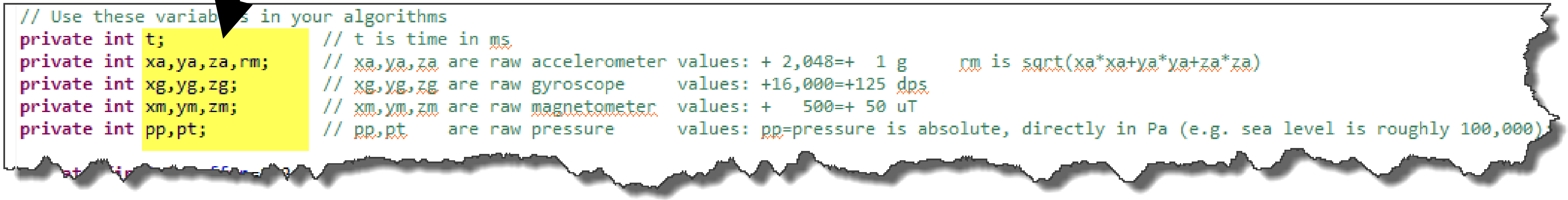
```java
//---------------------------------------------------------------
public Algorithms() {fft=new float[fftn];}
//---------------------------------------------------------------
public String algoname(int i)
{
String sres;
switch (i)
  {
  // Time Domain
  case  0: sres="Motion0";    break;
  case  1: sres="Motion1";    break;
  case  2: sres="Motion2";    break;
  case  3: sres="Motion3";    break;
  case  4: sres="";           break;
  case  5: sres="";           break;
  case  6: sres="";           break;
  case  7: sres="";           break;
  case  8: sres="";           break;
  case  9: sres="";           break;
  // Frequency Domain
```

Just change the name in the function called:
public String algoname (int i)

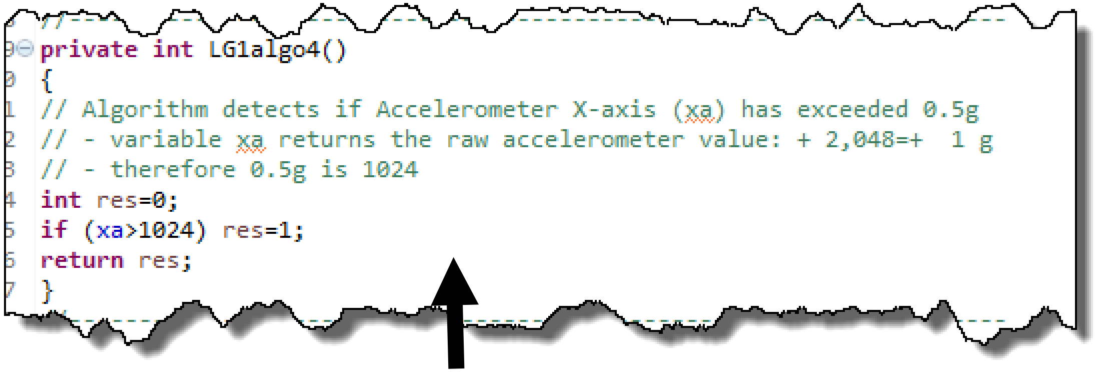
LG1algo0() is assigned to button name “Motion0”
LG1algo1() is assigned to button name “Motion1”
etc…etc…etc…

Here are the variable names you can use in you Algorithms…

LG1 Lab-
Time Domain variables



```
// Use these variables in your algorithms
private int t;                  // t is time in ms
private int xa,ya,za,rm;        // xa,ya,za are raw accelerometer values: + 2,048=+  1 g     rm is sqrt(xa*xa+ya*ya+za*za)
private int xg,yg,zg;           // xg,yg,zg are raw gyroscope     values: +16,000=+125 dps
private int xm,ym,zm;           // xm,ym,zm are raw magnetometer  values: +   500=+ 50 uT
private int pp,pt;              // pp,pt    are raw pressure      values: pp=pressure is absolute, directly in Pa (e.g. sea level is roughly 100,000);
```



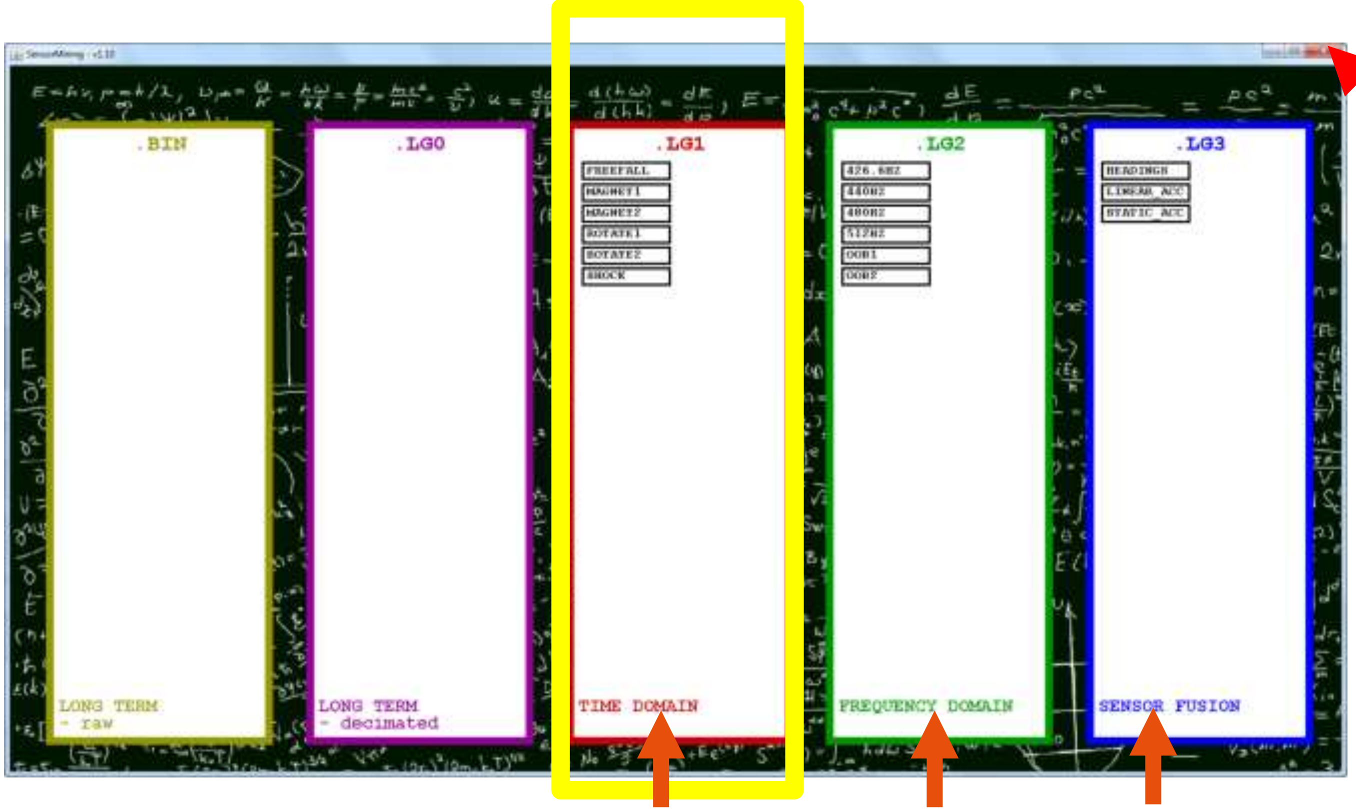
```
private int LG1algo4()
{
// Algorithm detects if Accelerometer X-axis (xa) has exceeded 0.5g
// - variable xa returns the raw accelerometer value: + 2,048=+  1 g
// - therefore 0.5g is 1024
int res=0;
if (xa>1024) res=1;
return res;
}
```

Here is a sample algorithm that looks for value greater than 1/2g on the accelerometer X-Axis and turns on the RED led whenever the condition is met.

SOFTWARE Description for the labs

- A description of the opening screen:

HIT ESC to return to this screen



To Close GUI

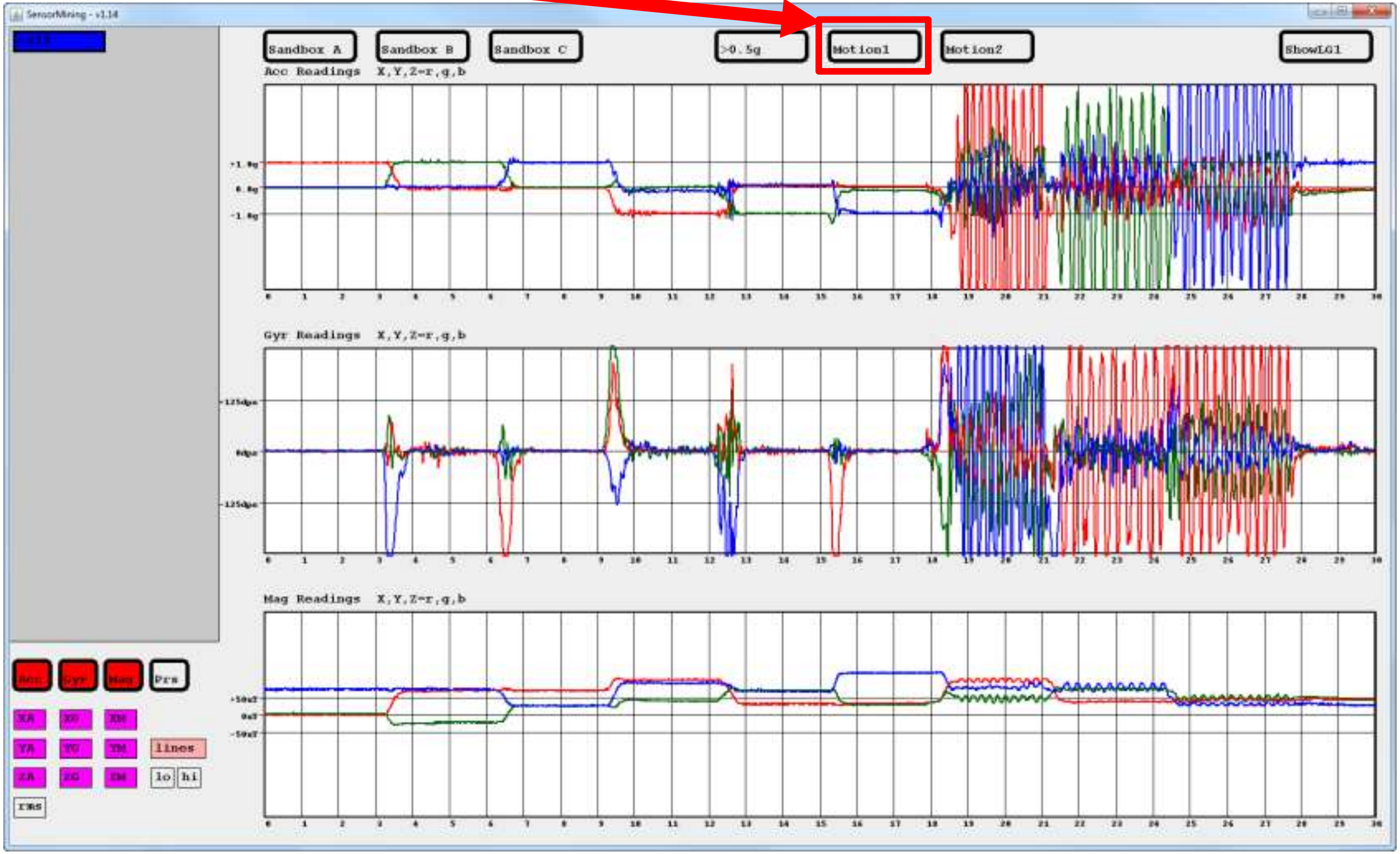
Time Domain Datalogger files Used in Part 1 of 3

Frequency Domain Datalogger files Used in Part 2 of 3

Sensor Fusion Datalogger files Used in Part 3 of 3

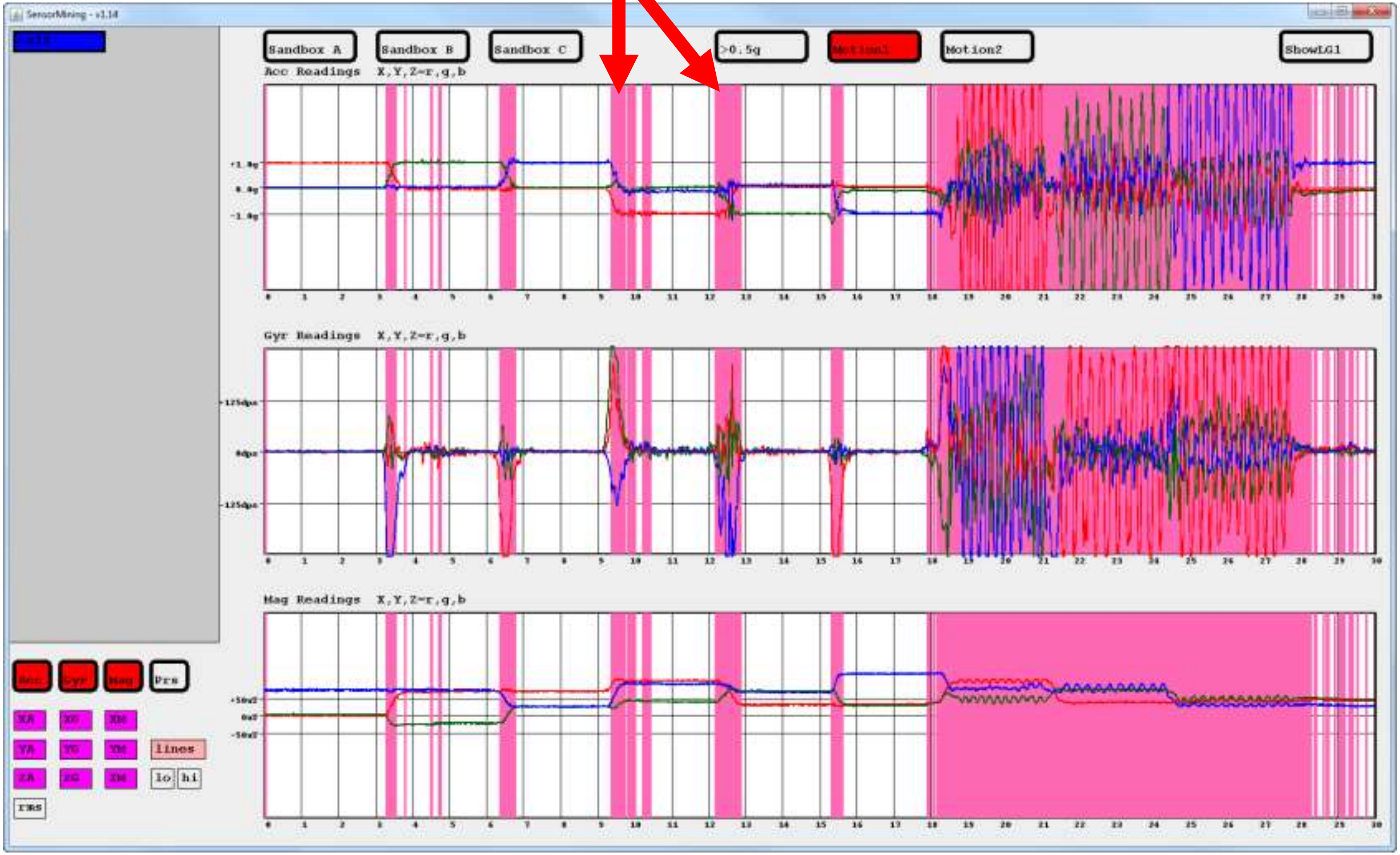
Explore the Algorithms in the Algorithms.java file

Toggle the button to apply / remove selected algorithm

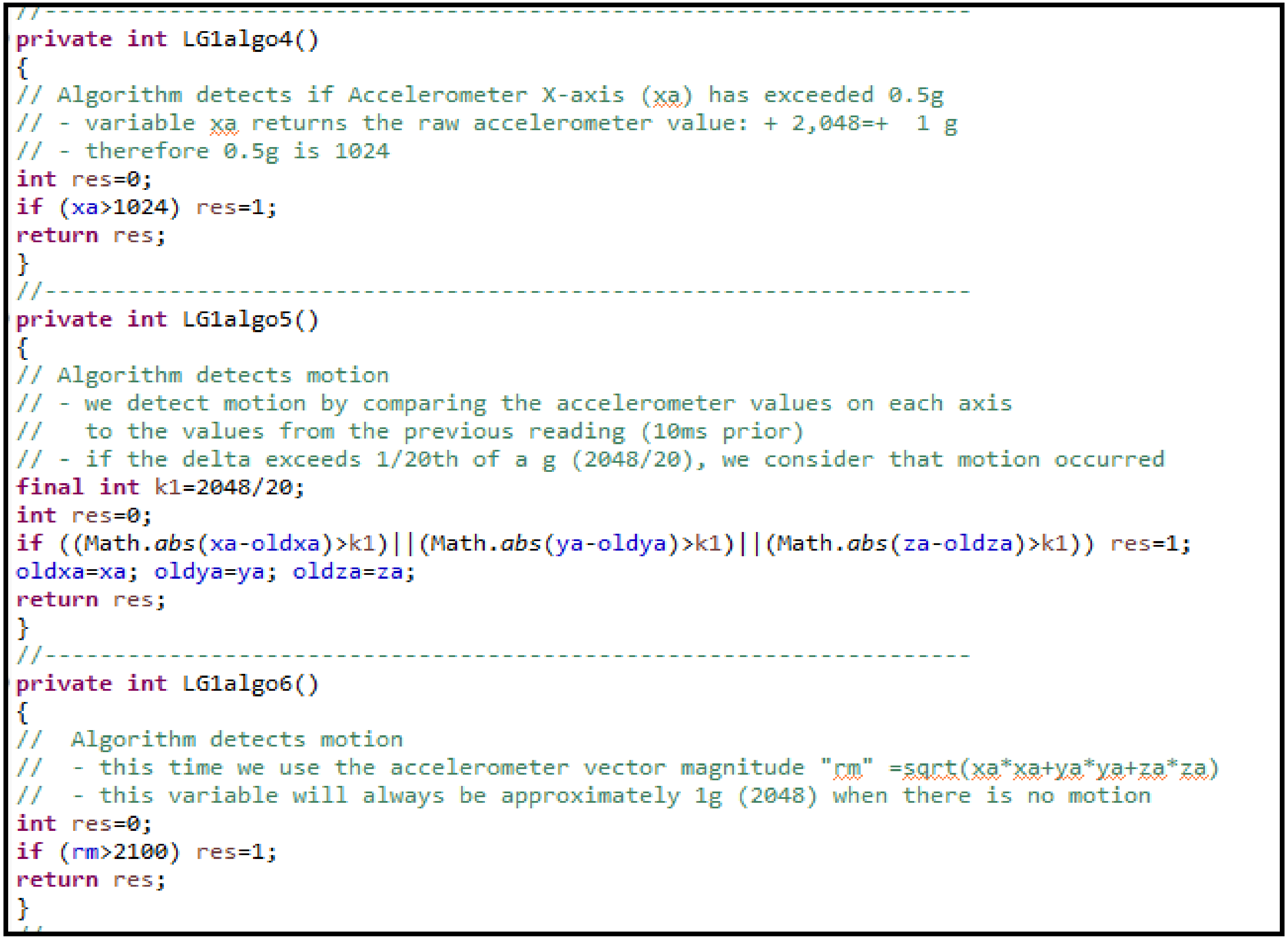


So what is happening when these different colors appear?

RED LED Simulation when condition is met in algorithm

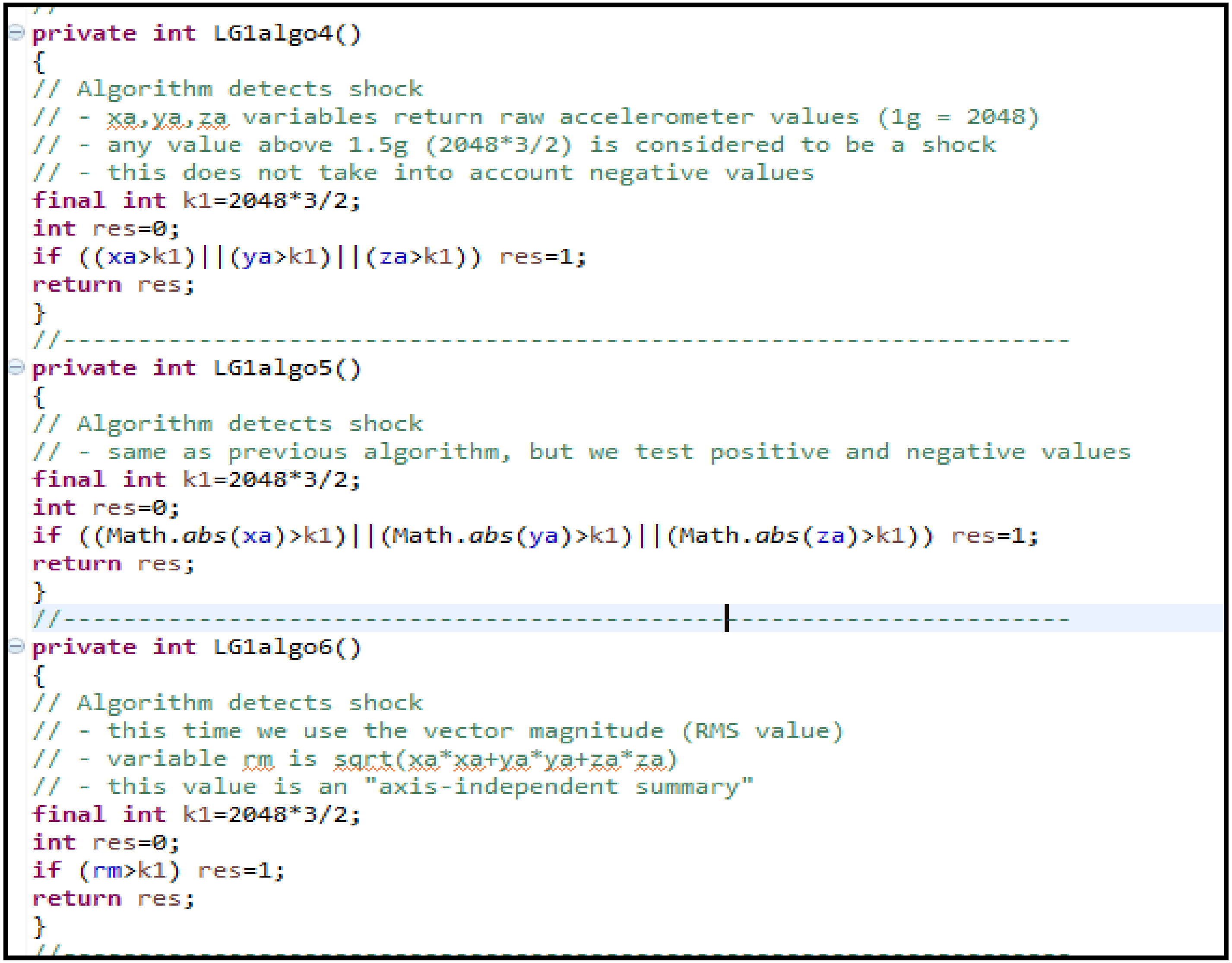


Motion Algorithms



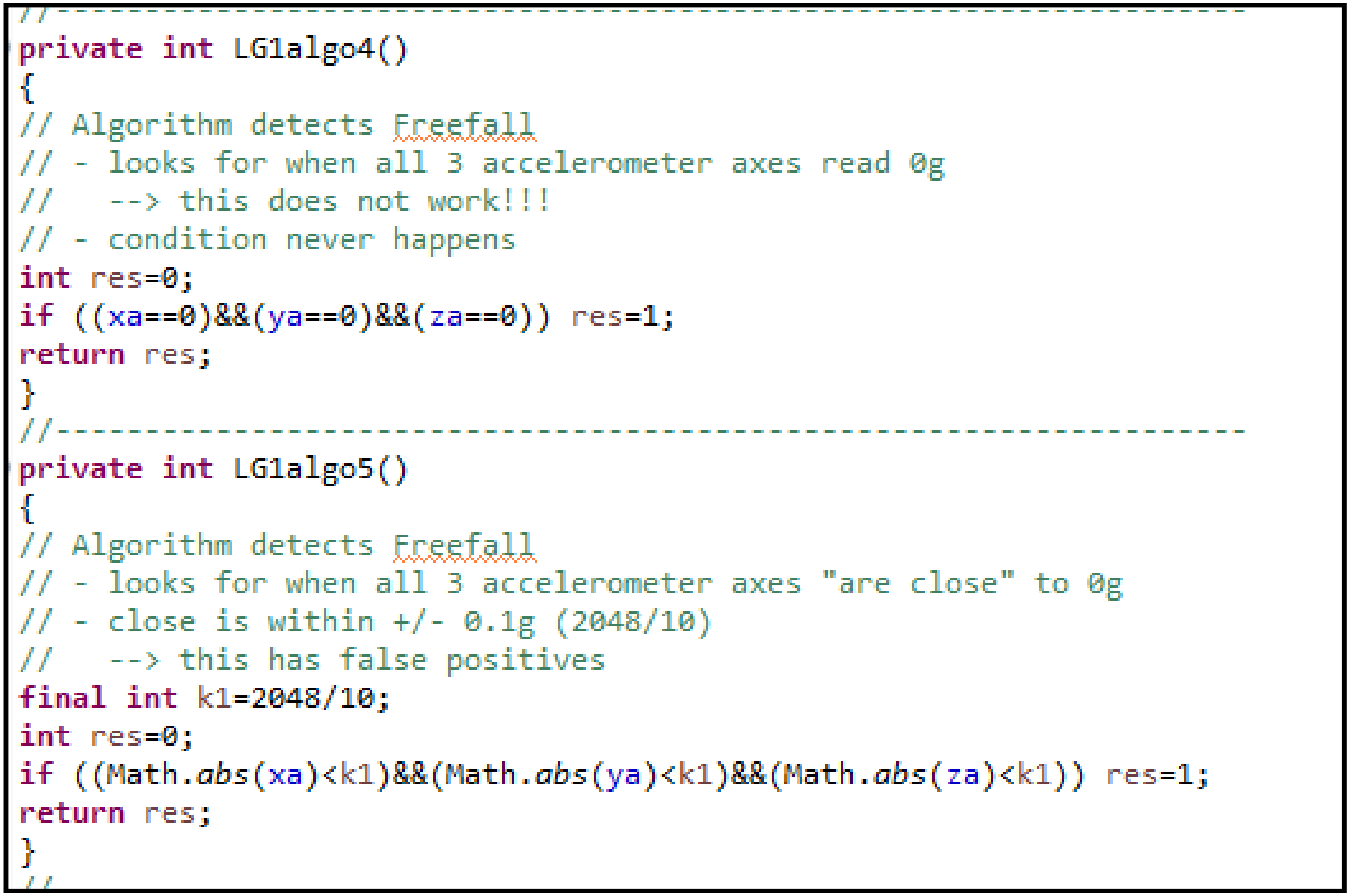
```
//-------------------------------------------------------------------
private int LG1algo4()
{
// Algorithm detects if Accelerometer X-axis (xa) has exceeded 0.5g
// - variable xa returns the raw accelerometer value: + 2,048=+  1 g
// - therefore 0.5g is 1024
int res=0;
if (xa>1024) res=1;
return res;
}
//-------------------------------------------------------------------
private int LG1algo5()
{
// Algorithm detects motion
// - we detect motion by comparing the accelerometer values on each axis
//   to the values from the previous reading (10ms prior)
// - if the delta exceeds 1/20th of a g (2048/20), we consider that motion occurred
final int k1=2048/20;
int res=0;
if ((Math.abs(xa-oldxa)>k1)||(Math.abs(ya-oldya)>k1)||(Math.abs(za-oldza)>k1)) res=1;
oldxa=xa; oldya=ya; oldza=za;
return res;
}
//-------------------------------------------------------------------
private int LG1algo6()
{
//  Algorithm detects motion
//  - this time we use the accelerometer vector magnitude "rm" =sqrt(xa*xa+ya*ya+za*za)
//  - this variable will always be approximately 1g (2048) when there is no motion
int res=0;
if (rm>2100) res=1;
return res;
}
//
```

Shock Algorithms



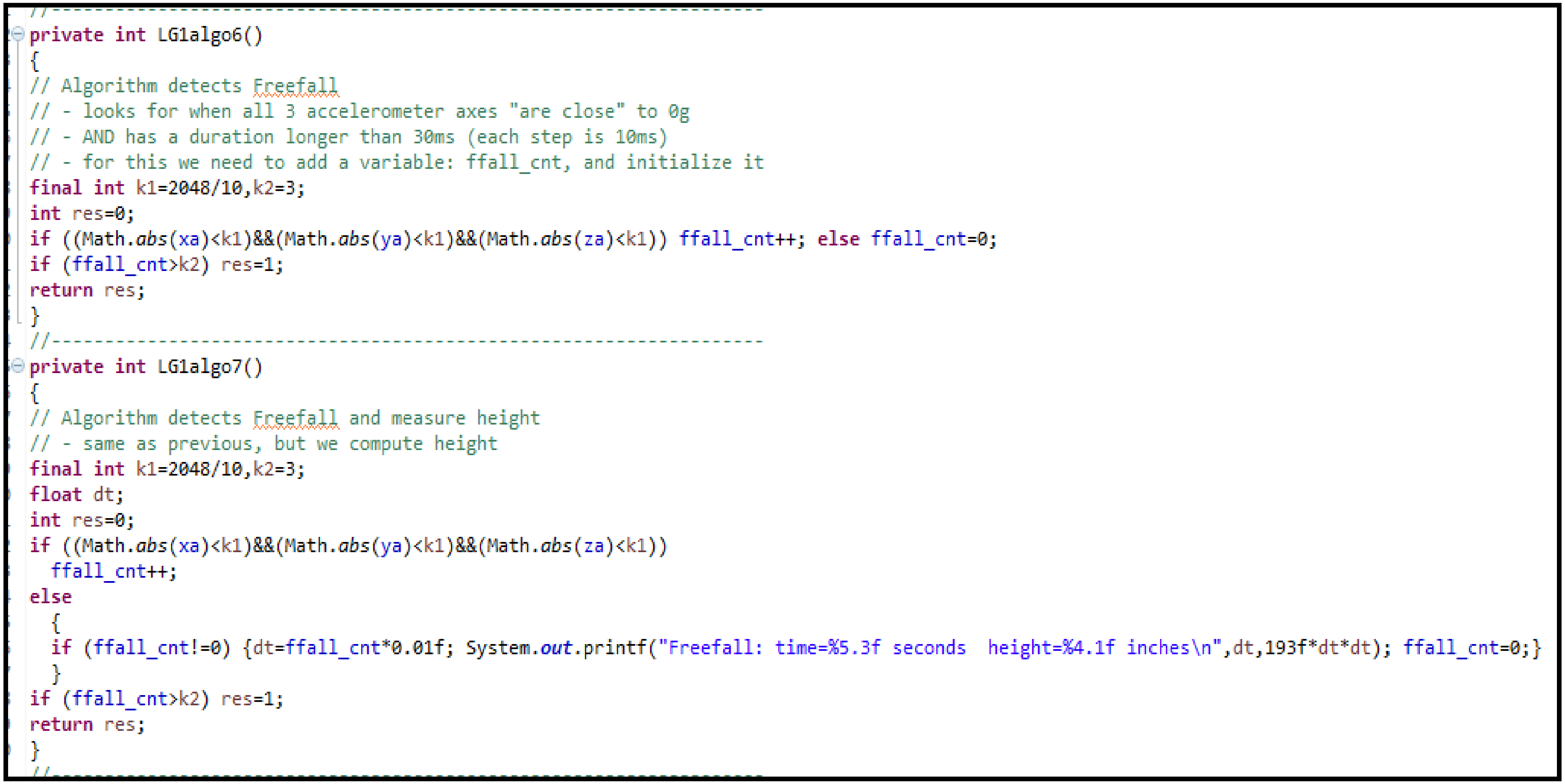
```
//
private int LG1algo4()
{
// Algorithm detects shock
// - xa,ya,za variables return raw accelerometer values (1g = 2048)
// - any value above 1.5g (2048*3/2) is considered to be a shock
// - this does not take into account negative values
final int k1=2048*3/2;
int res=0;
if ((xa>k1)||(ya>k1)||(za>k1)) res=1;
return res;
}
//-------------------------------------------------------------------
private int LG1algo5()
{
// Algorithm detects shock
// - same as previous algorithm, but we test positive and negative values
final int k1=2048*3/2;
int res=0;
if ((Math.abs(xa)>k1)||(Math.abs(ya)>k1)||(Math.abs(za)>k1)) res=1;
return res;
}
//-------------------------------------------------------------------
private int LG1algo6()
{
// Algorithm detects shock
// - this time we use the vector magnitude (RMS value)
// - variable rm is sqrt(xa*xa+ya*ya+za*za)
// - this value is an "axis-independent summary"
final int k1=2048*3/2;
int res=0;
if (rm>k1) res=1;
return res;
}
//-------------------------------------------------------------------
```

Freefall Algorithms



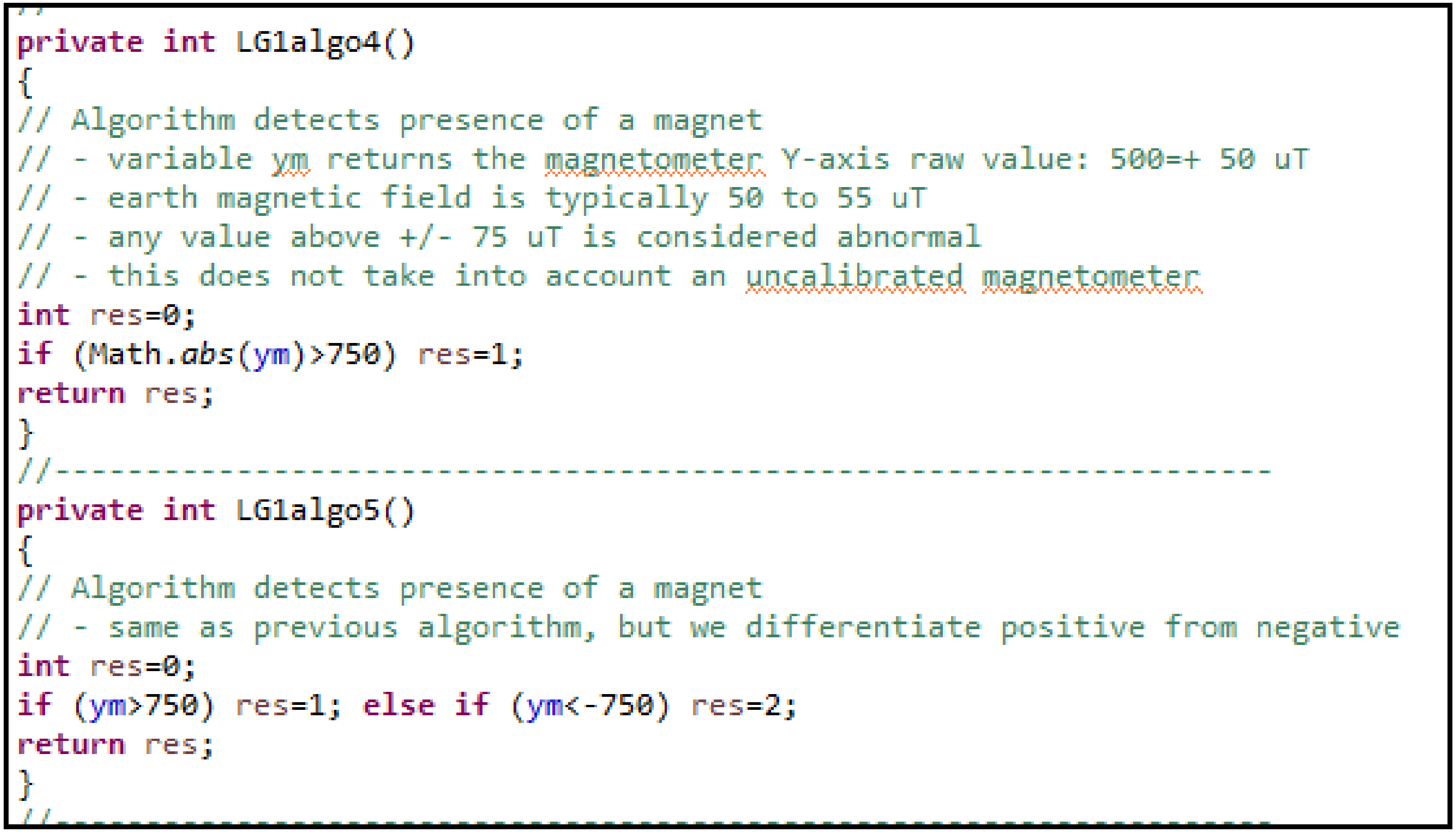
```
//-------------------------------------------------------------------
private int LG1algo4()
{
// Algorithm detects Freefall
// - looks for when all 3 accelerometer axes read 0g
//   --> this does not work!!!
// - condition never happens
int res=0;
if ((xa==0)&&(ya==0)&&(za==0)) res=1;
return res;
}
//-------------------------------------------------------------------
private int LG1algo5()
{
// Algorithm detects Freefall
// - looks for when all 3 accelerometer axes "are close" to 0g
// - close is within +/- 0.1g (2048/10)
//   --> this has false positives
final int k1=2048/10;
int res=0;
if ((Math.abs(xa)<k1)&&(Math.abs(ya)<k1)&&(Math.abs(za)<k1)) res=1;
return res;
}
//
```

Freefall Algorithms



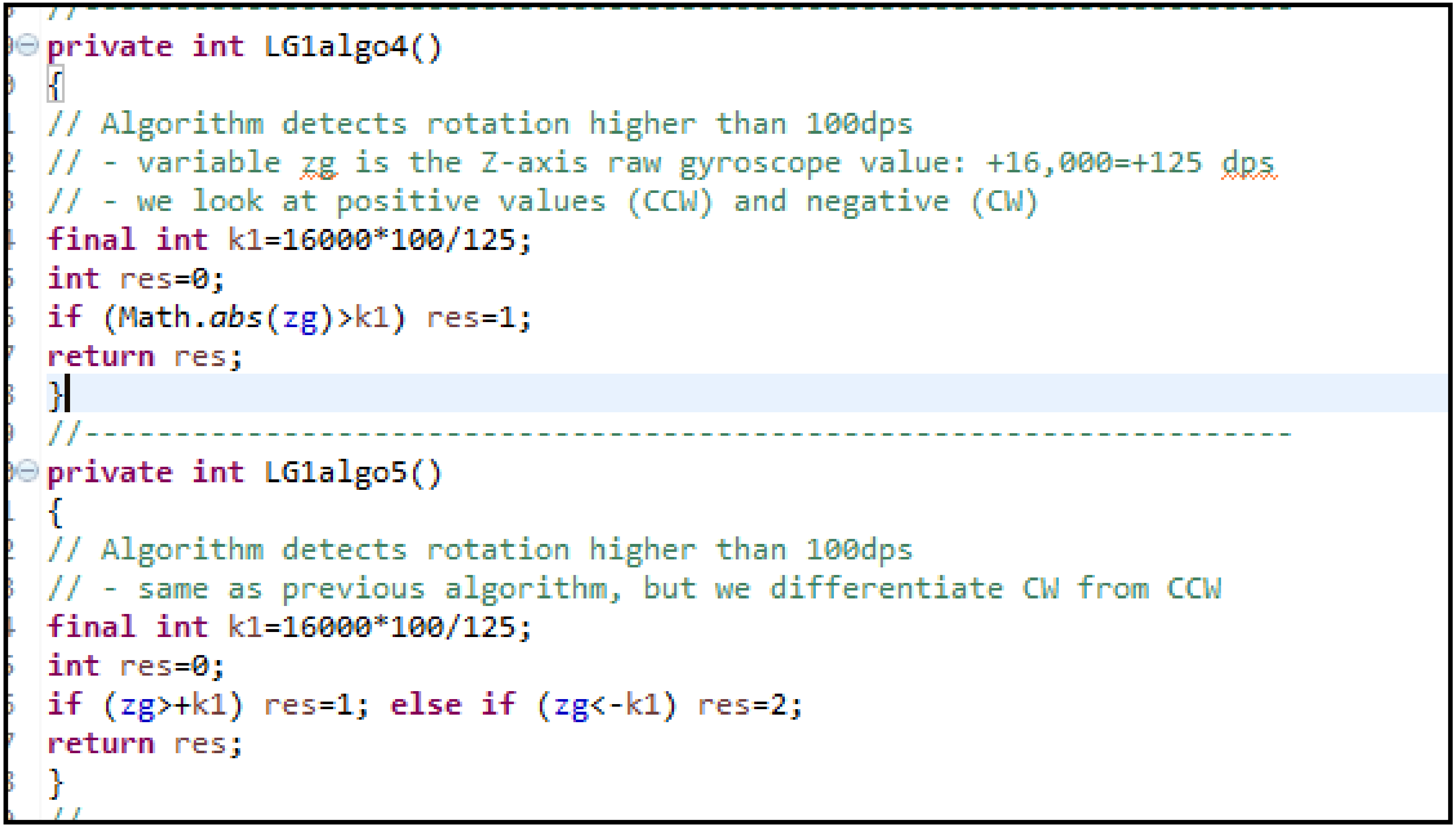
```
//-------------------------------------------------------------------
private int LG1algo6()
{
// Algorithm detects Freefall
// - looks for when all 3 accelerometer axes "are close" to 0g
// - AND has a duration longer than 30ms (each step is 10ms)
// - for this we need to add a variable: ffall_cnt, and initialize it
final int k1=2048/10,k2=3;
int res=0;
if ((Math.abs(xa)<k1)&&(Math.abs(ya)<k1)&&(Math.abs(za)<k1)) ffall_cnt++; else ffall_cnt=0;
if (ffall_cnt>k2) res=1;
return res;
}
//-------------------------------------------------------------------
private int LG1algo7()
{
// Algorithm detects Freefall and measure height
// - same as previous, but we compute height
final int k1=2048/10,k2=3;
float dt;
int res=0;
if ((Math.abs(xa)<k1)&&(Math.abs(ya)<k1)&&(Math.abs(za)<k1))
  ffall_cnt++;
else
  {
  if (ffall_cnt!=0) {dt=ffall_cnt*0.01f; System.out.printf("Freefall: time=%5.3f seconds  height=%4.1f inches\n",dt,193f*dt*dt); ffall_cnt=0;}
  }
if (ffall_cnt>k2) res=1;
return res;
}
//-------------------------------------------------------------------
```

Magnet Algorithms



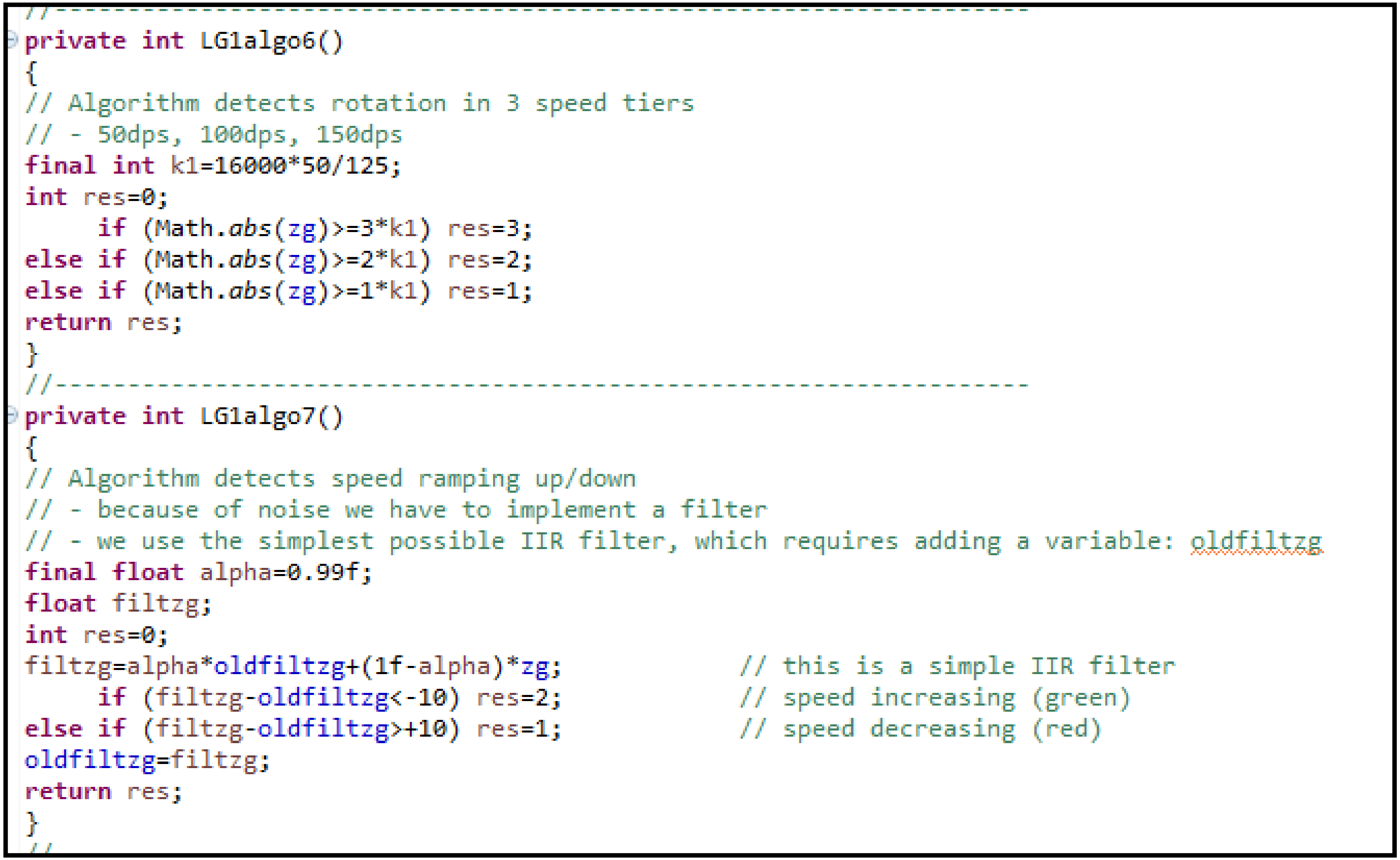
```
//
private int LG1algo4()
{
// Algorithm detects presence of a magnet
// - variable ym returns the magnetometer Y-axis raw value: 500=+ 50 uT
// - earth magnetic field is typically 50 to 55 uT
// - any value above +/- 75 uT is considered abnormal
// - this does not take into account an uncalibrated magnetometer
int res=0;
if (Math.abs(ym)>750) res=1;
return res;
}
//-------------------------------------------------------------------
private int LG1algo5()
{
// Algorithm detects presence of a magnet
// - same as previous algorithm, but we differentiate positive from negative
int res=0;
if (ym>750) res=1; else if (ym<-750) res=2;
return res;
}
//-------------------------------------------------------------------
```

Rotation Algorithms



```
//--------------------------------------------------------------------
private int LG1algo4()
{
// Algorithm detects rotation higher than 100dps
// - variable zg is the Z-axis raw gyroscope value: +16,000=+125 dps
// - we look at positive values (CCW) and negative (CW)
final int k1=16000*100/125;
int res=0;
if (Math.abs(zg)>k1) res=1;
return res;
}
//--------------------------------------------------------------------
private int LG1algo5()
{
// Algorithm detects rotation higher than 100dps
// - same as previous algorithm, but we differentiate CW from CCW
final int k1=16000*100/125;
int res=0;
if (zg>+k1) res=1; else if (zg<-k1) res=2;
return res;
}
//
```

Rotation Algorithms



```
//-------------------------------------------------------------------
private int LG1algo6()
{
// Algorithm detects rotation in 3 speed tiers
// - 50dps, 100dps, 150dps
final int k1=16000*50/125;
int res=0;
     if (Math.abs(zg)>=3*k1) res=3;
else if (Math.abs(zg)>=2*k1) res=2;
else if (Math.abs(zg)>=1*k1) res=1;
return res;
}
//-------------------------------------------------------------------
private int LG1algo7()
{
// Algorithm detects speed ramping up/down
// - because of noise we have to implement a filter
// - we use the simplest possible IIR filter, which requires adding a variable: oldfiltzg
final float alpha=0.99f;
float filtzg;
int res=0;
filtzg=alpha*oldfiltzg+(1f-alpha)*zg;             // this is a simple IIR filter
     if (filtzg-oldfiltzg<-10) res=2;             // speed increasing (green)
else if (filtzg-oldfiltzg>+10) res=1;             // speed decreasing (red)
oldfiltzg=filtzg;
return res;
}
//
```

Frequency-Domain

LAB GUIDE

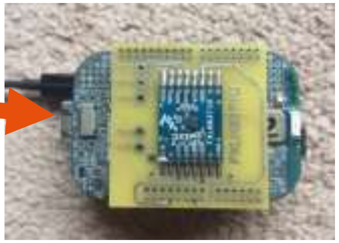
Hands-on: Compare data between dumb and intelligent data loggers, and understand the techniques used to compress the information.

- Lab objective 1 – Review steps involved to collect data at different stations pertaining to the frequency domain.
- Lab objective 2 – Collect data at the stations – only at the frequency domain stations
- Lab objective 3 – Test you algorithms written on real hardware for the frequency domain!

PLEASE REMEMBE THIS LAB GUIDE IS AN INSTRUCTION GUIDE not an exact step-by-step guide.

Programming you Hardware Board

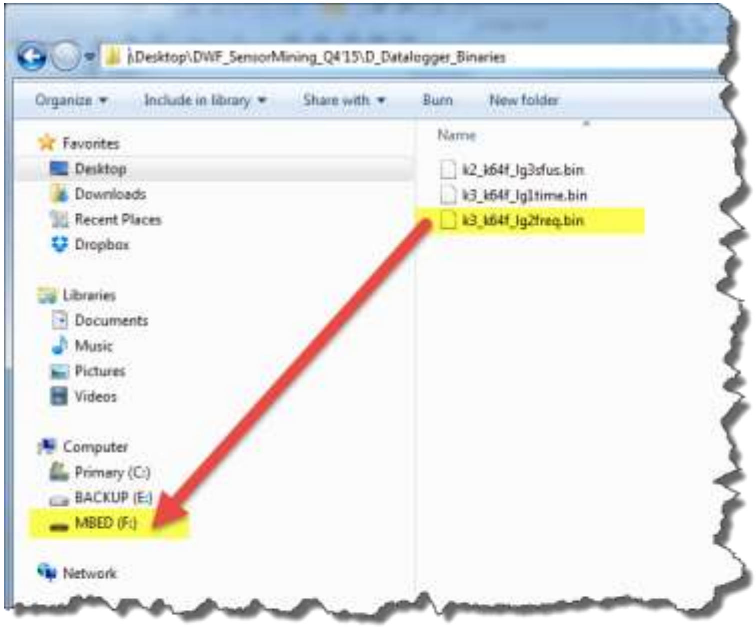
1. Plug in board to micro USB port labeled “OpenSDA”…



2. Open Windows Explorer and migrate to the desktop folder: *Desktop\DWF_SensorMining_Q4'15\D_Datalogger_Binaries*

3. You should see a **MBED** (drive) letter. Please let instructor know if you don’t get a drive called “**MBED**”.

4. Drag and drop the file “*k3_k64f_lg2freq.bin*” on top of the **MBED** drive.



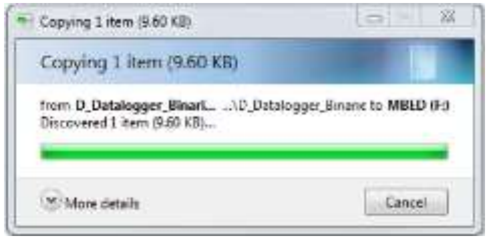
\Desktop\DWF_SensorMining_Q4'15\D_Datalogger_Binaries

k2_k64f_lg3sfus.bin

k3_k64f_lg1time.bin

k3_k64f_lg2freq.bin

5. You should see a progress screen…



Copying 1 item (9.60 KB)

Copying 1 item (9.60 KB)

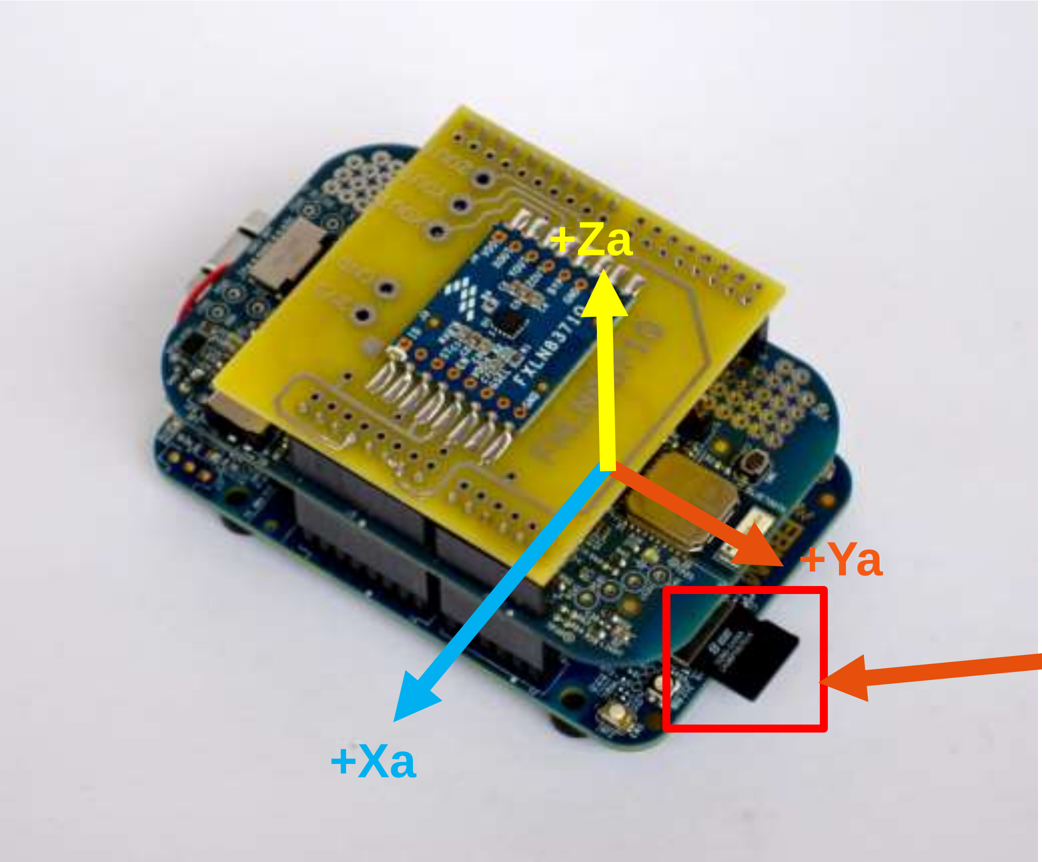
from **D_Datalogger_Binari...** ...\D_Datalogger_Binarie to **MBED (F:)**
Discovered 1 item (9.60 KB)...

6. Programming complete!

HARDWARE Board Description

Hardware Datalogger Board for the FREQUENCY DOMAIN LABS:

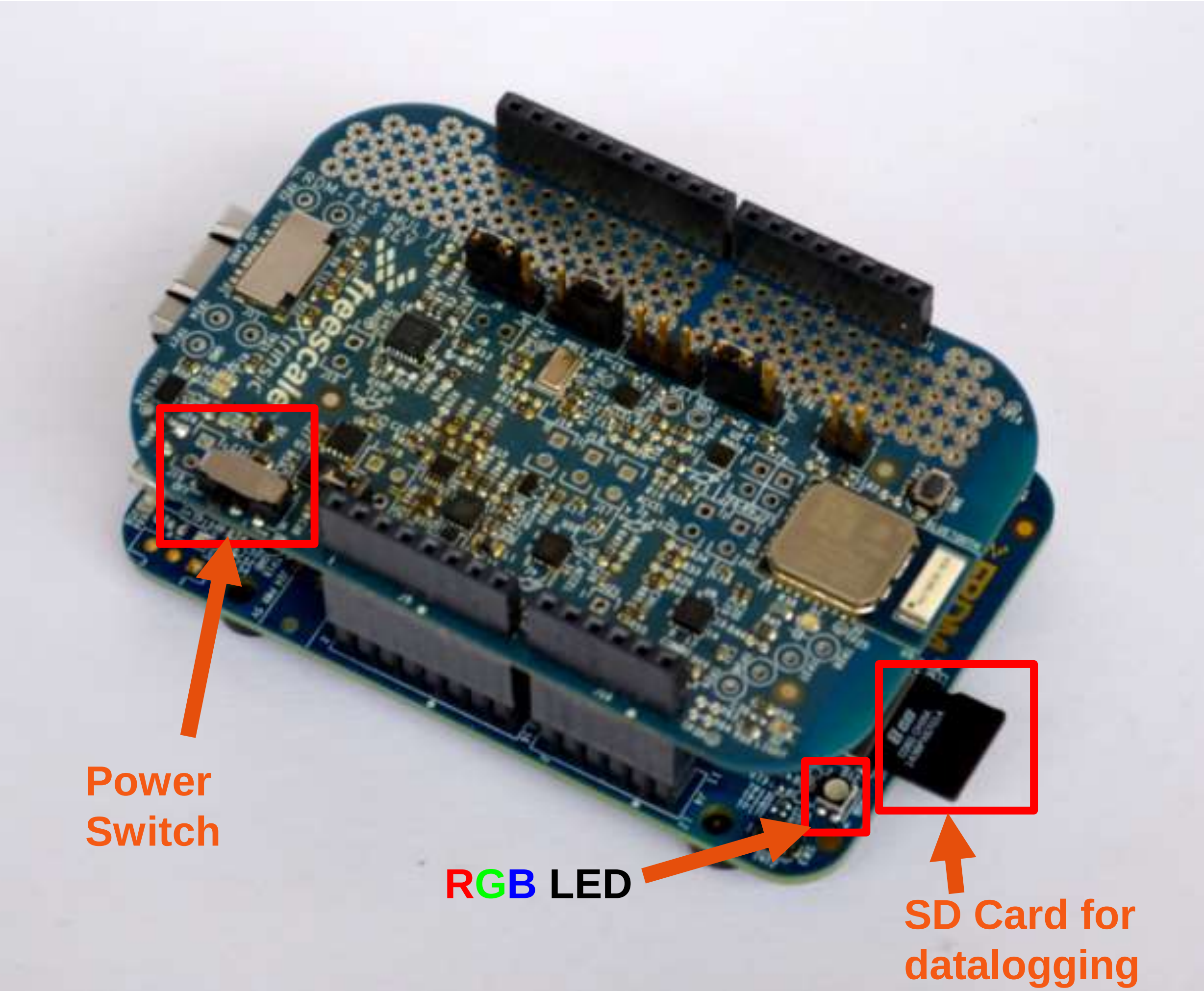
- FRDM-K64F board + FRDM-FXS-MULTI-B + BRKOUT-FXLN8371Q



Sensor AXIS orientation
a = Accelerometer

SD Card Slot
used for
datalogging

INSTRUCTION - Starting the datalogger

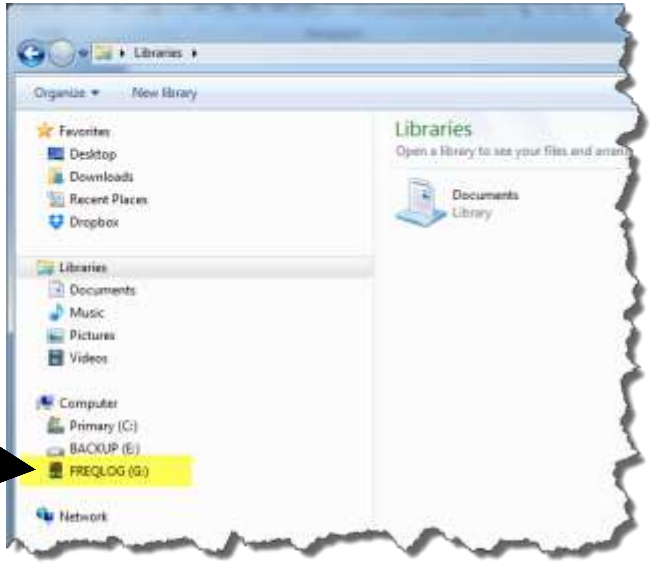


To start the datalogger:

1. Insert SD Card into holder
2. Turn on Power switch (might have to cycle power if RED LED doesn't come on)
3. RGB LED will go solid red for 5 seconds 🔴
4. Begin Datalogging when RGB LED blinks green 🟢
5. Datalogging stops after 30 seconds, when RGB LED goes solid green 🟢

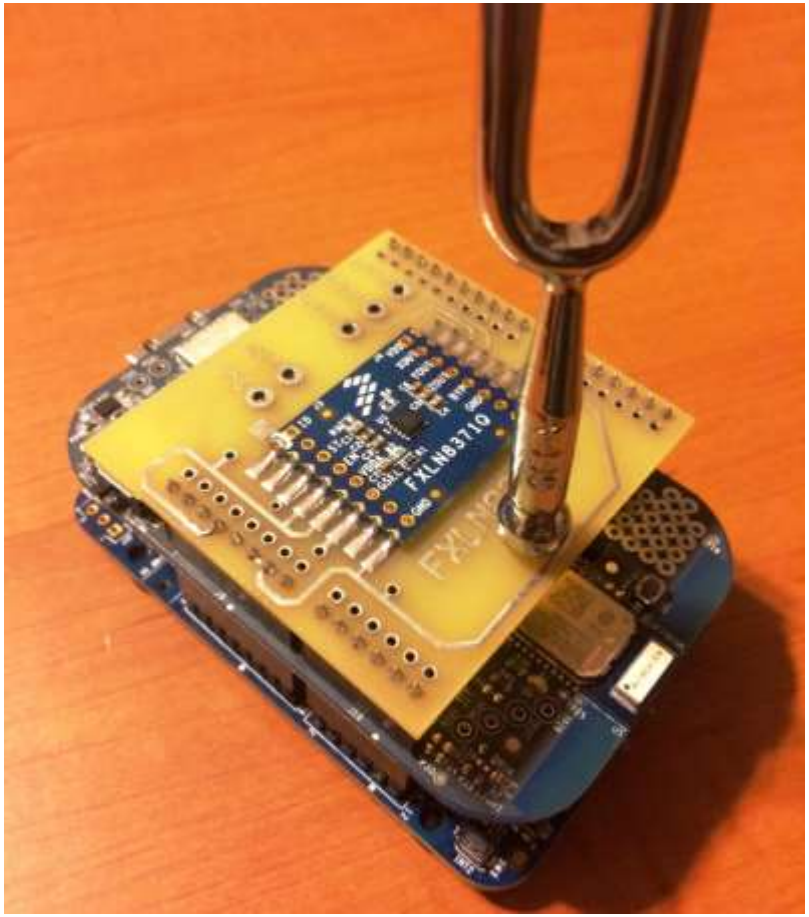
INSTRUCTION - Copying datalogger files onto laptop

1. Turn off Datalogger.
2. Remove micro SDCard from hardware board.
3. Insert micro SDCard into adapter and insert into laptop SDCard slot.
4. The SDCard will show up as “FREQLOG (drive):
5. Copy “DATALOG.LG1” to location: *C:\Desktop\DWF_SensorMining_Q4'15\C_Sensor_Logs* (***Please do not add any additional folders***)
6. Rename “DATALOG.LG1” to any name with up to 9 characters, no spaces, no special characters. Do not change the extension.



Open a library to see your files and arran

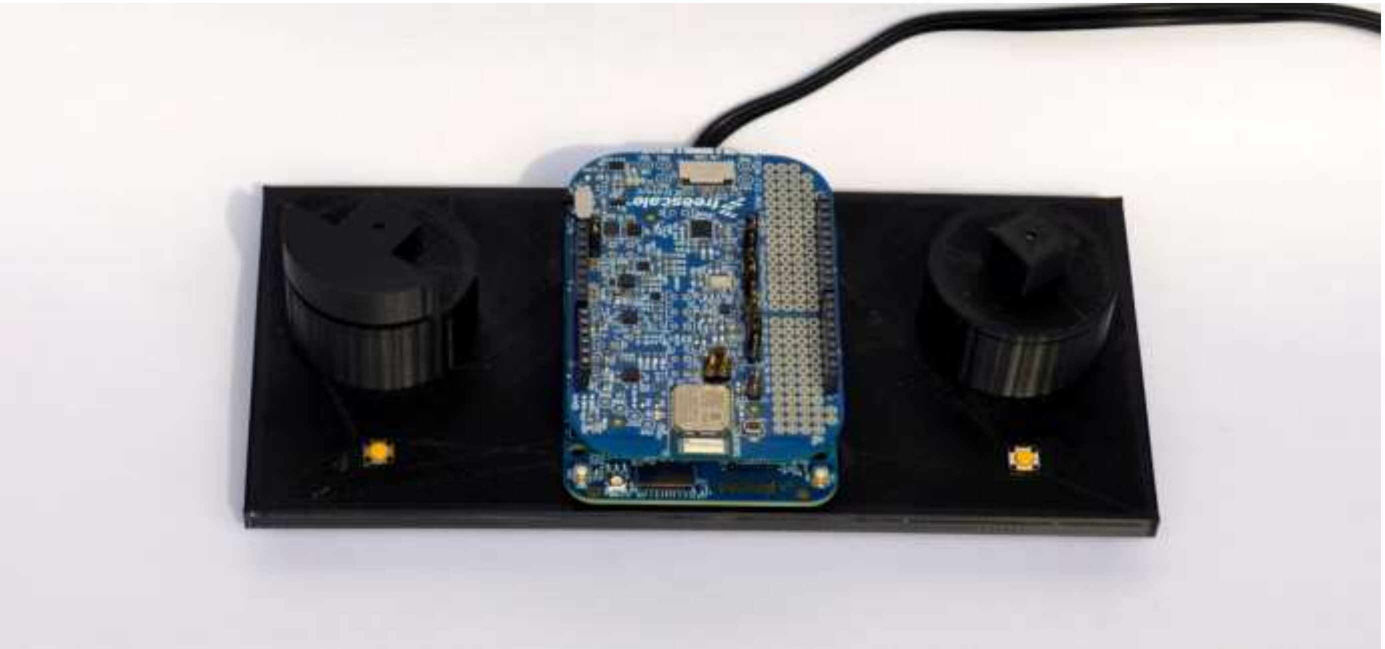
Lab Stations 1 – Tuning Forks Lab



1. Select a tuning fork.
2. Start Datalogger.
3. Strike Tuning Fork and place on the board like the picture. Wait about 5 seconds before removing and try another tuning fork within the 30 seconds.
4. When LED goes solid green, take back to desk and copy file onto laptop.

Lab Station 1 – Motor Imbalance Lab

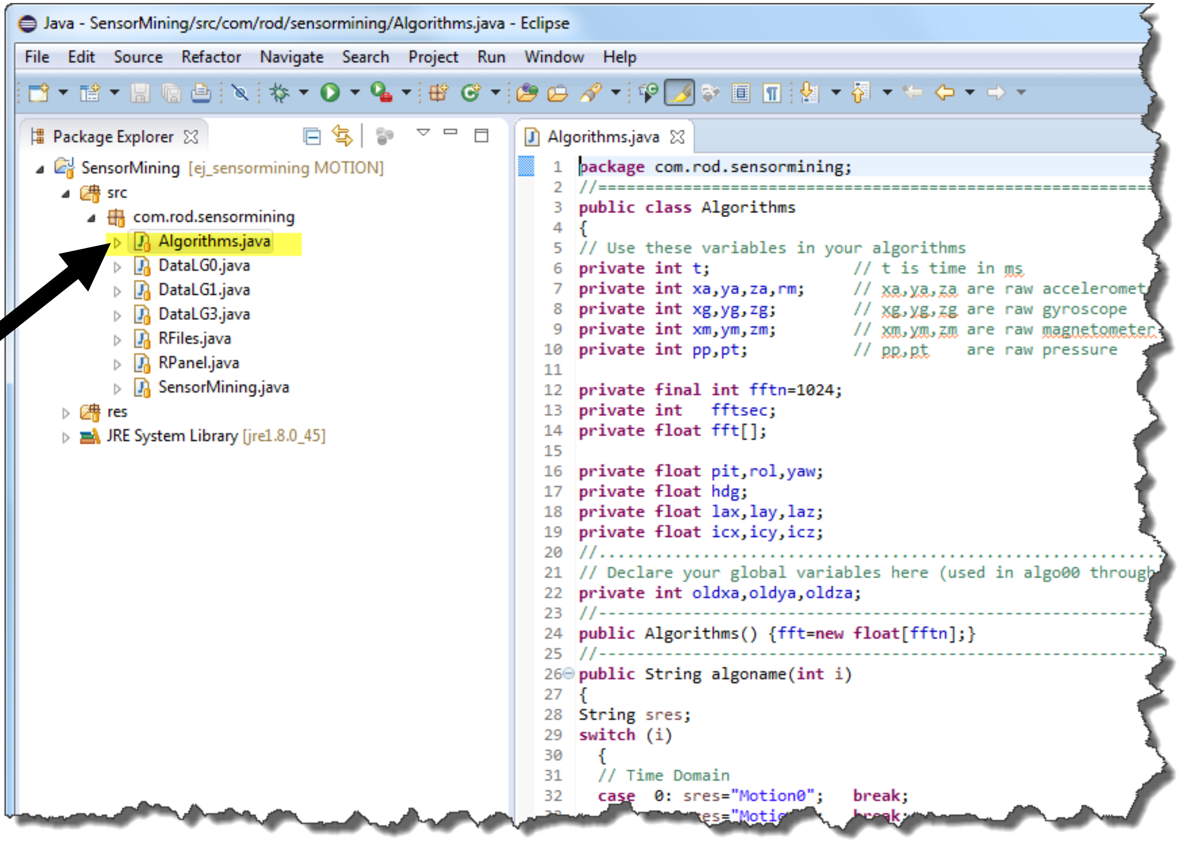
- Start Datalogger.
- Place the board on the motor apparatus like the picture below. The feet of the board fit into the holes on the apparatus.
- Start the datalogger.
- Press and hold the switch on the left for 10 seconds, then release. Next, press and hold the switch on the right, then release.
- When LED goes solid green, take back to desk and copy file onto laptop.



Open the Eclipse Java GUI

1. Open Eclipse Mars on the Desktop, or in the DWF_SensorMining_Q4'15\A_Installers

2. In the Java “Package Explorer” Window, drop down, SRC, and com.rod.sensormining. Double-Click in the file called “Algorithms.java” **NOTE: This is the ONLY file you can modify any source code!**



```
package com.rod.sensormining;
//==============================================================
public class Algorithms
{
// Use these variables in your algorithms
private int t;                  // t is time in ms
private int xa,ya,za,rm;        // xa,ya,za are raw acceleromet
private int xg,yg,zg;           // xg,yg,zg are raw gyroscope
private int xm,ym,zm;           // xm,ym,zm are raw magnetometer
private int pp,pt;              // pp,pt    are raw pressure

private final int fftn=1024;
private int   fftsec;
private float fft[];

private float pit,rol,yaw;
private float hdg;
private float lax,lay,laz;
private float icx,icy,icz;
//..............................................................
// Declare your global variables here (used in algo00 through
private int oldxa,oldya,oldza;
//--------------------------------------------------------------
public Algorithms() {fft=new float[fftn];}
//--------------------------------------------------------------
public String algoname(int i)
{
String sres;
switch (i)
  {
  // Time Domain
  case  0: sres="Motion0";   break;
```

How to switch between the labs in this class

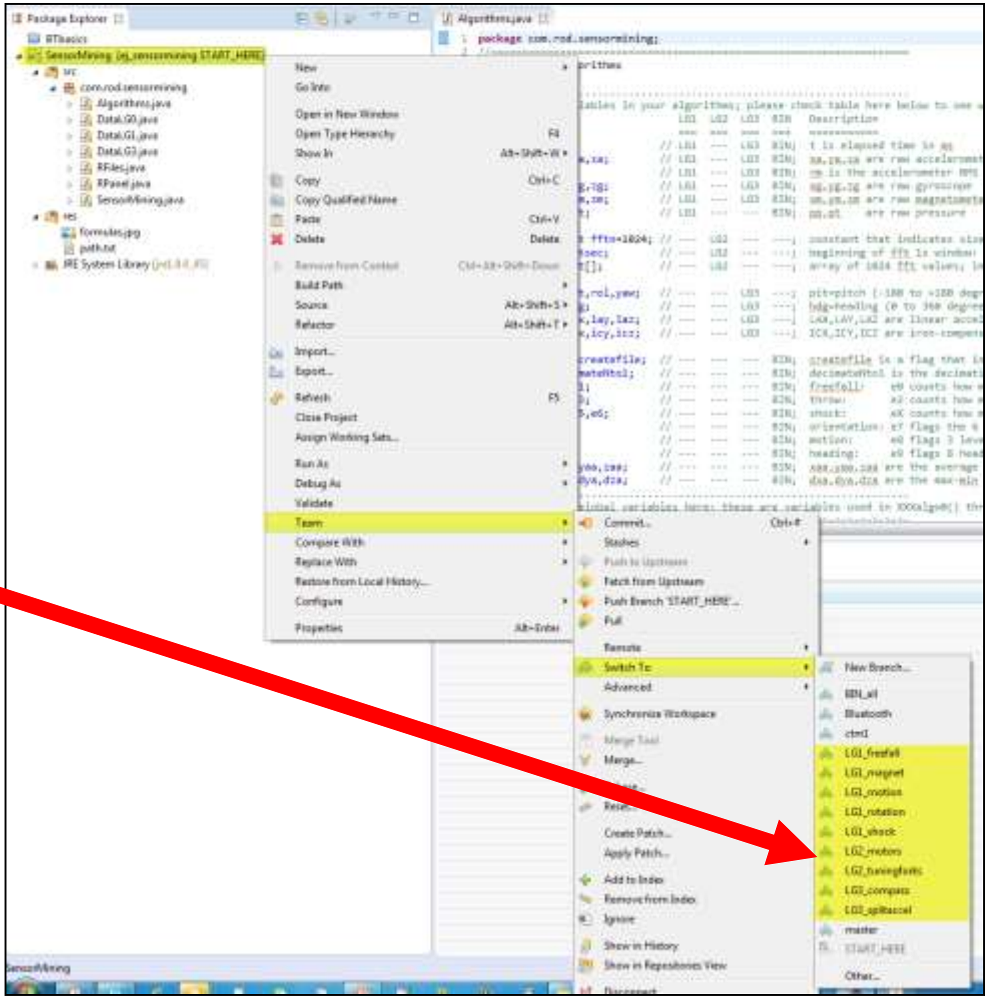
We will have 2 labs with several Algorithms in each lab to explore.

The Freq domain Labs are:

1. LG2_tuningforks
2. LG2_motor

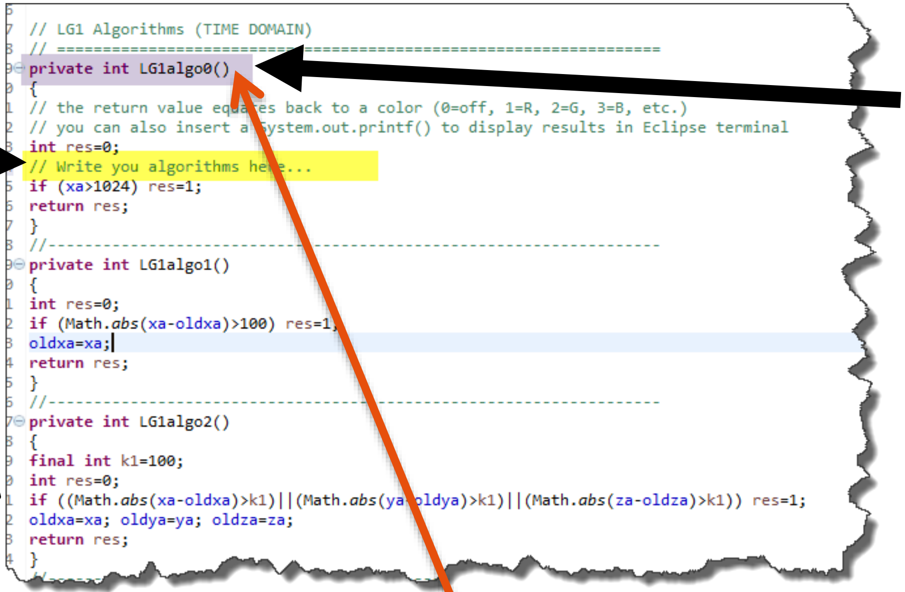
- To switch between them, right click on the “*SensorMining*” Project and select ***Team / Switch To / LG1,LG2,LG3*** (for example, anything in yellow), for this first lab of Freq domain, we are going to start with LG2_tuningforks.

- Next, Hit CTRL-F11 or Menu Run/Run (to run Algorithms).



Type you Algorithms here…and assign a button to it!

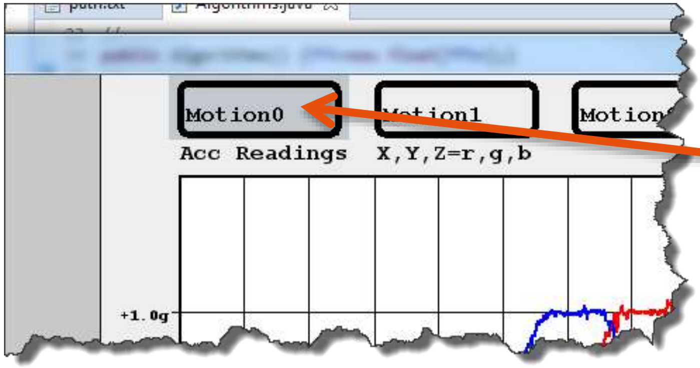
In the File, Algorithms.java, this is where you can type you algorithms **OR** observe and study the examples already done.

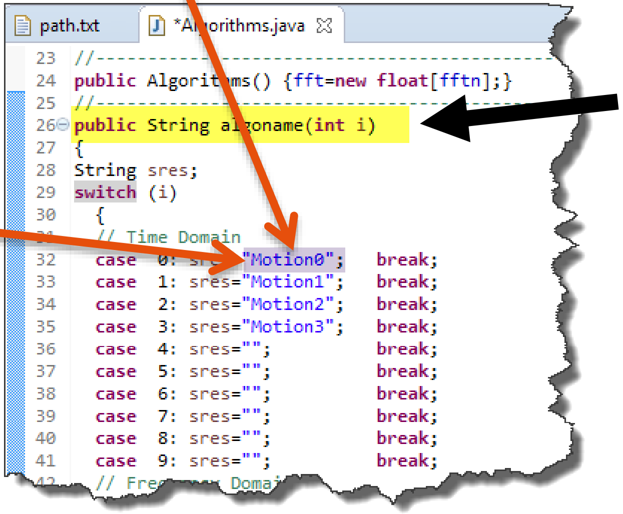


```
// LG1 Algorithms (TIME DOMAIN)
// ==================================================================
private int LG1algo0()
{
// the return value equates back to a color (0=off, 1=R, 2=G, 3=B, etc.)
// you can also insert a System.out.printf() to display results in Eclipse terminal
int res=0;
// Write you algorithms here...
if (xa>1024) res=1;
return res;
}
//------------------------------------------------------------------
private int LG1algo1()
{
int res=0;
if (Math.abs(xa-oldxa)>100) res=1;
oldxa=xa;
return res;
}
//------------------------------------------------------------------
private int LG1algo2()
{
final int k1=100;
int res=0;
if ((Math.abs(xa-oldxa)>k1)||(Math.abs(ya-oldya)>k1)||(Math.abs(za-oldza)>k1)) res=1;
oldxa=xa; oldya=ya; oldza=za;
return res;
}
```

Notice the function: private int LG1algo0(): This is the name of the algorithm that can be assigned to a button. THIS IS AN EXAMPLE.

This slide is an example of how to re-assign buttons: This applies to all labs:
LG1 – Time Domain
LG2 – Frequency Domain
LG3 – Sensor Fusion Domain





path.txt | *Algorithms.java

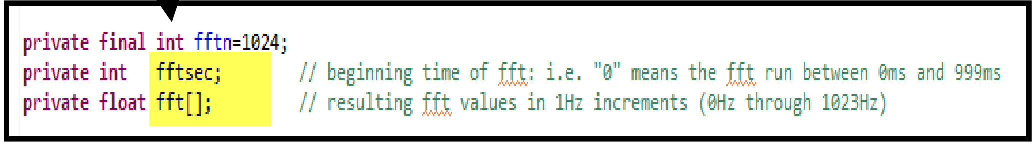
```
//--------------------------------------------------------------
public Algorithms() {fft=new float[fftn];}
//--------------------------------------------------------------
public String algoname(int i)
{
String sres;
switch (i)
  {
  // Time Domain
  case  0: sres="Motion0";   break;
  case  1: sres="Motion1";   break;
  case  2: sres="Motion2";   break;
  case  3: sres="Motion3";   break;
  case  4: sres="";          break;
  case  5: sres="";          break;
  case  6: sres="";          break;
  case  7: sres="";          break;
  case  8: sres="";          break;
  case  9: sres="";          break;
  // Frequency Domai
```

Just change the name in the function called: public String algoname (int i)

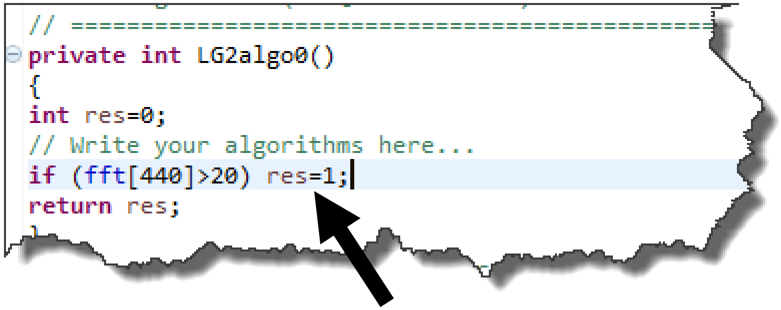
LG1algo0() is assigned to button name “Motion0”
LG1algo1() is assigned to button name “Motion1”
etc…etc…etc…

Here are the variable names you can use in you Algorithms…

LG2 Lab-
Frequency Domain variables



```
private final int fftn=1024;
private int   fftsec;       // beginning time of fft: i.e. "0" means the fft run between 0ms and 999ms
private float fft[];        // resulting fft values in 1Hz increments (0Hz through 1023Hz)
```



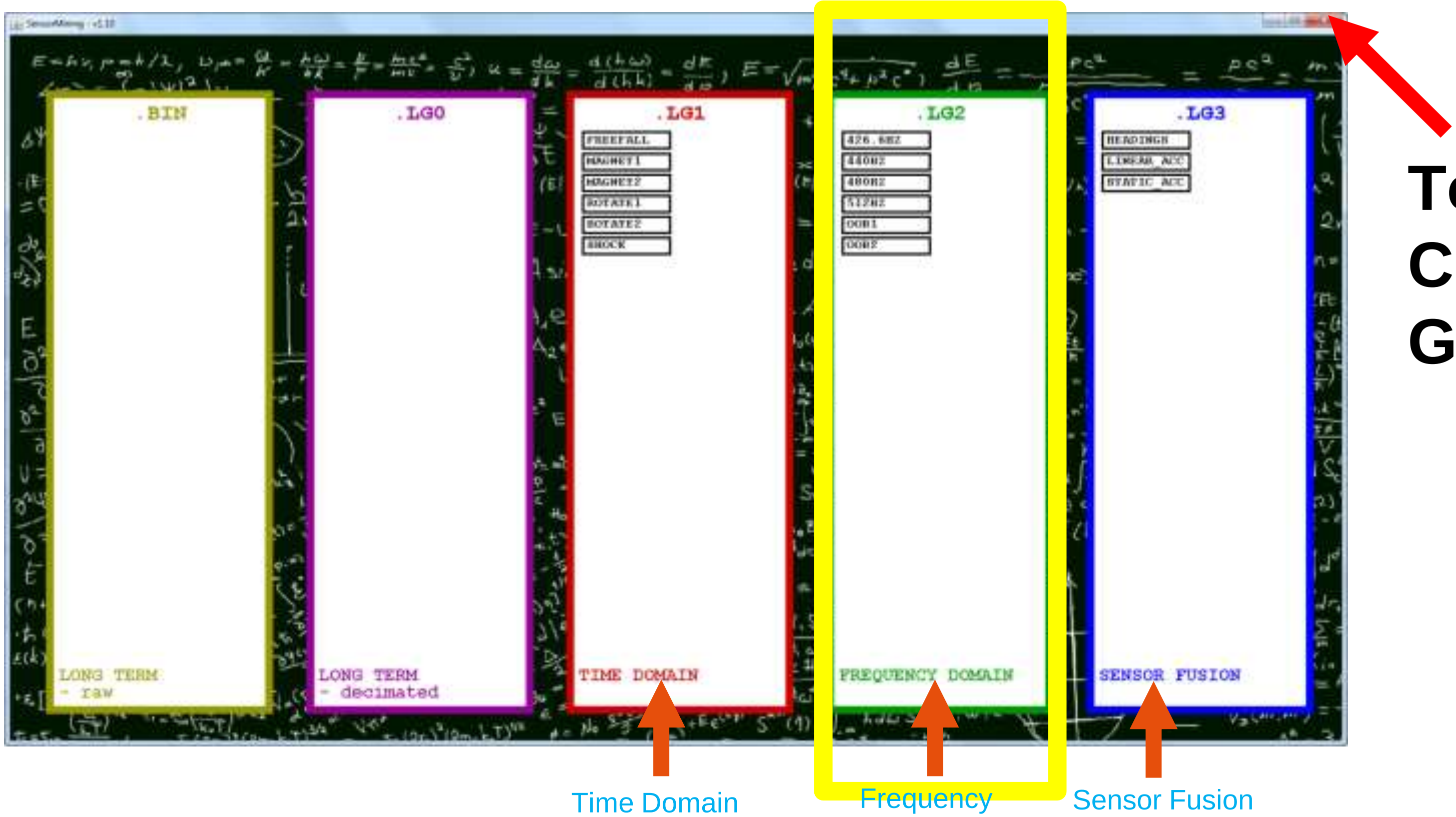
```
// ==============================================
private int LG2algo0()
{
int res=0;
// Write your algorithms here...
if (fft[440]>20) res=1;
return res;
}
```

Here is a LG2 Lab sample algorithm that detects a FFT frequency at 440Hz, and turns on the RED led if the condition is met.

SOFTWARE Description for the labs

- A description of the opening screen:

HIT ESC to return to this screen



To Close GUI

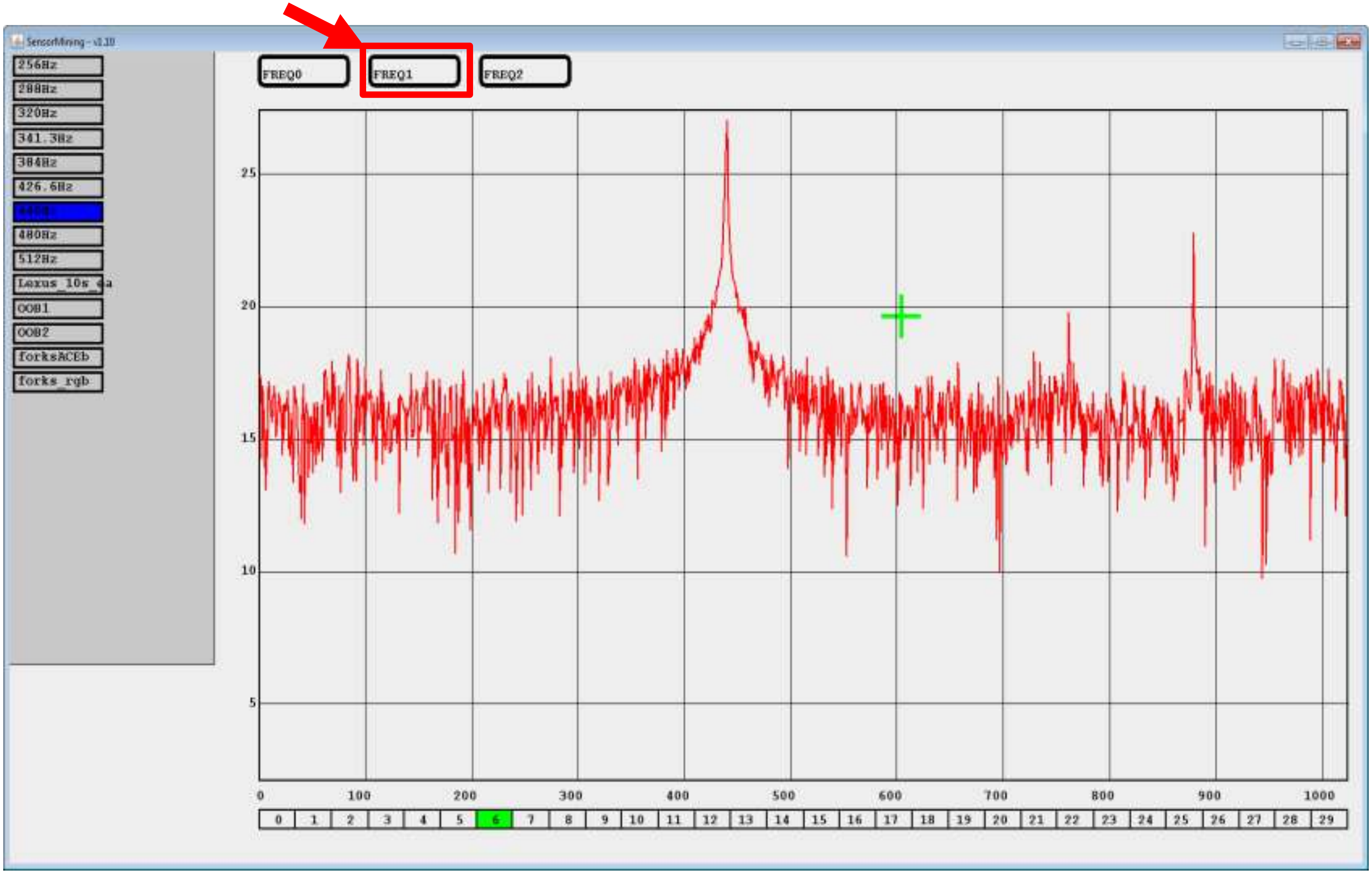
Time Domain Datalogger files Used in Part 1 of 3

Frequency Domain Datalogger files Used in Part 2 of 3

Sensor Fusion Datalogger files Used in Part 3 of 3

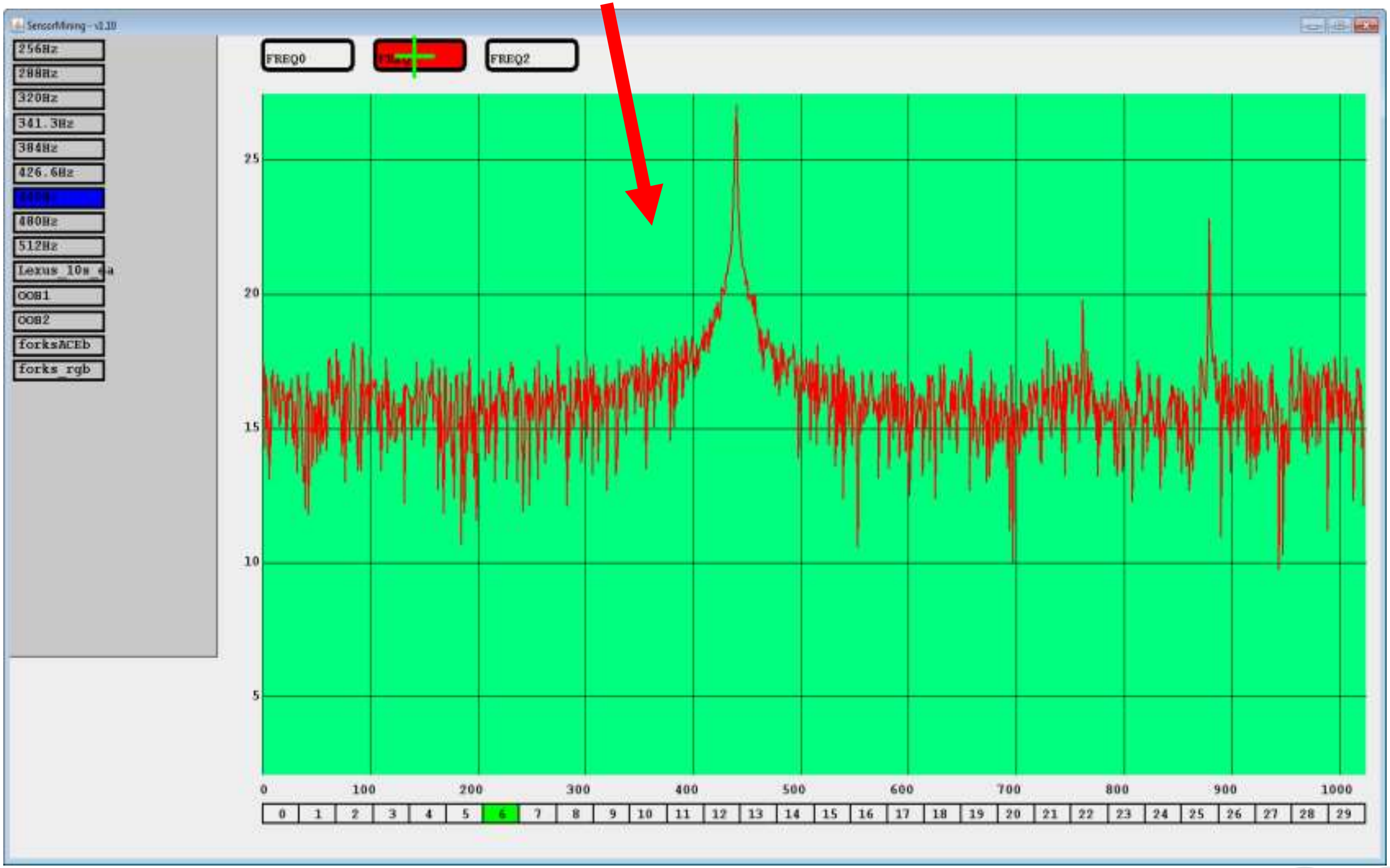
How do I apply an algorithm to my data in the Frequency Domain?

Toggle the button to apply / remove selected algorithm

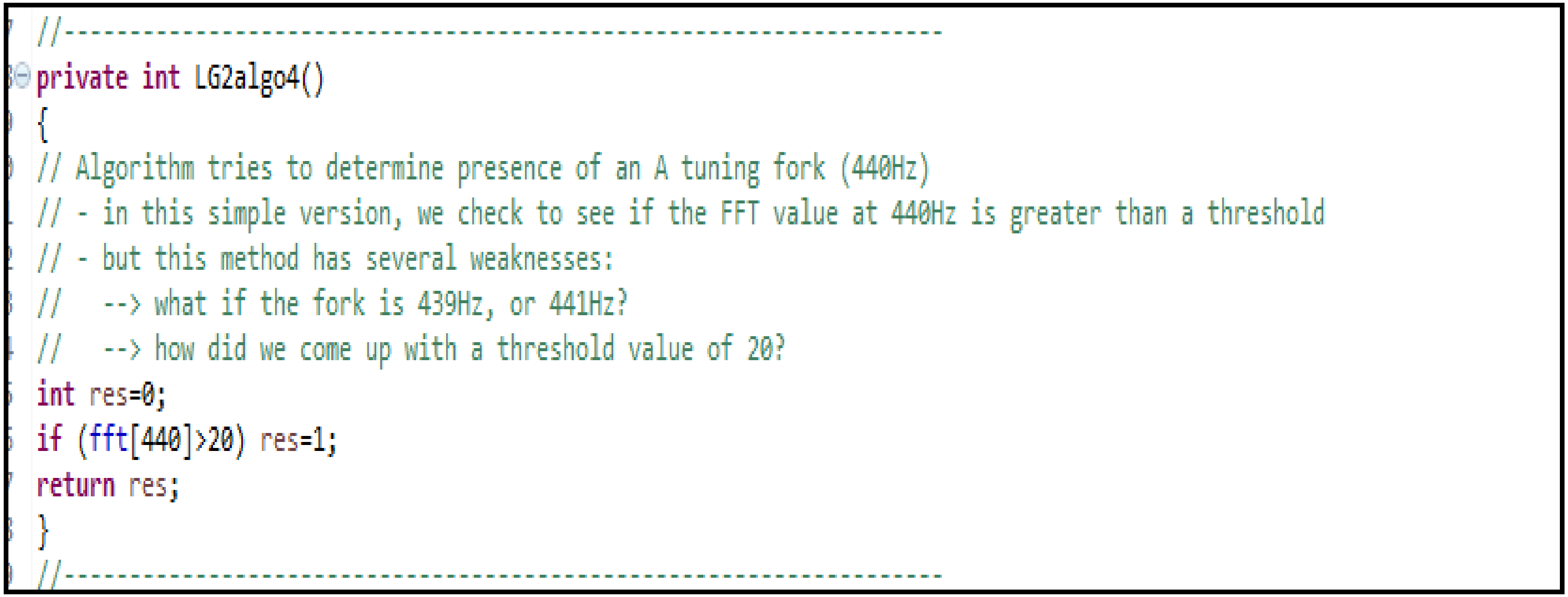


The FREQ1 algorithm is applied to my data in the Frequency Domain

GREEN LED Simulation when condition is met in algorithm

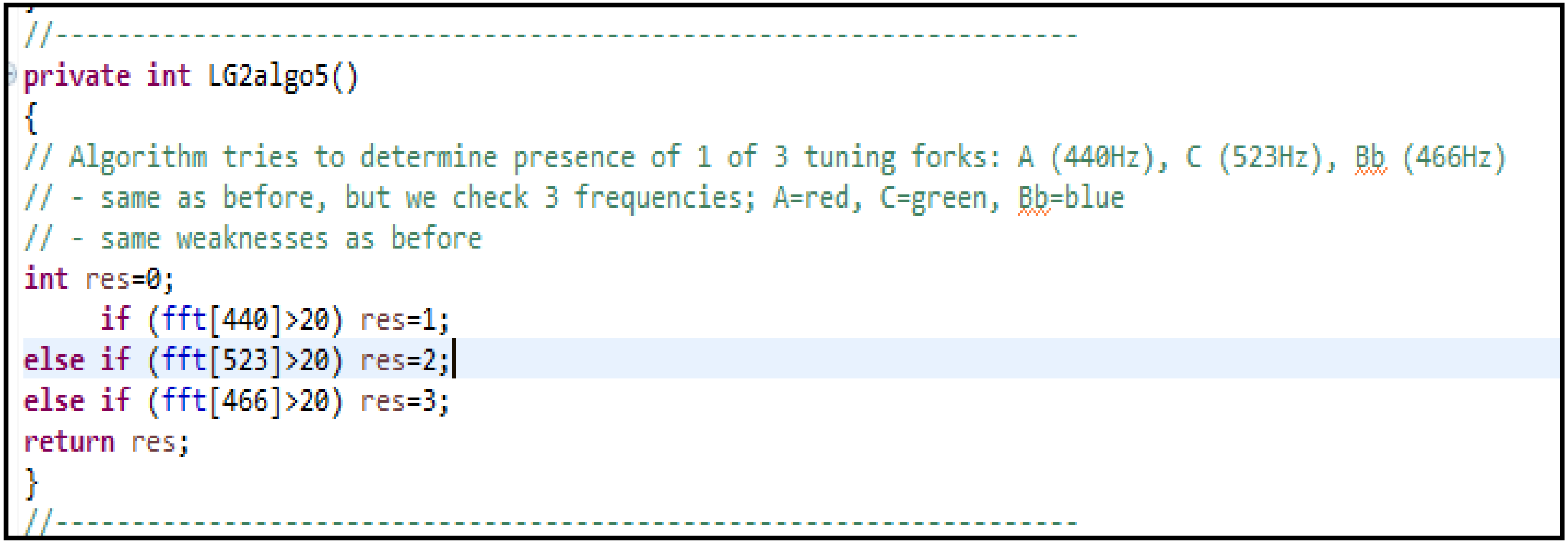


Tuning Forks Algorithms



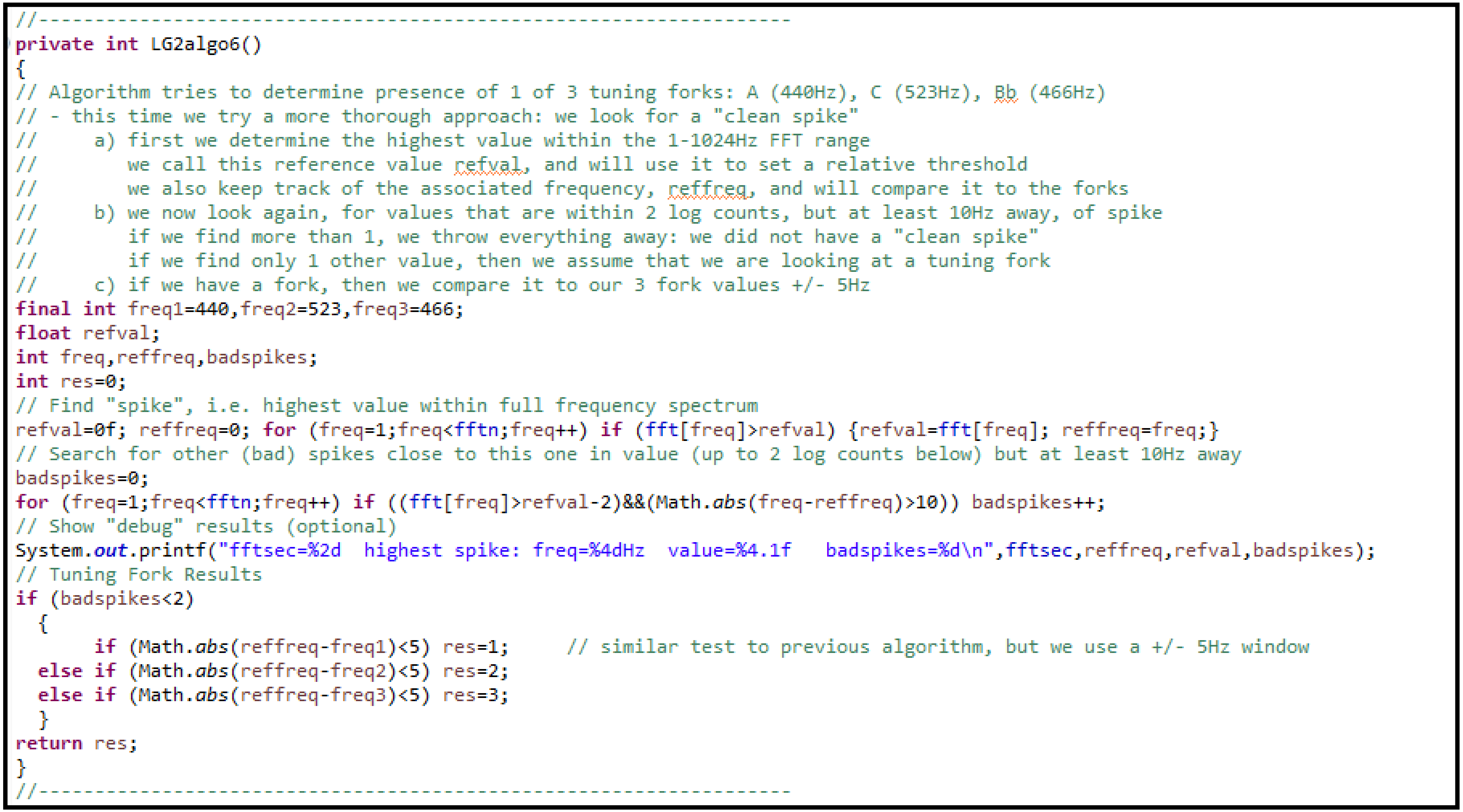
```
//-------------------------------------------------------------
private int LG2algo4()
{
// Algorithm tries to determine presence of an A tuning fork (440Hz)
// - in this simple version, we check to see if the FFT value at 440Hz is greater than a threshold
// - but this method has several weaknesses:
//    --> what if the fork is 439Hz, or 441Hz?
//    --> how did we come up with a threshold value of 20?
int res=0;
if (fft[440]>20) res=1;
return res;
}
//-------------------------------------------------------------
```

Tuning Forks Algorithms



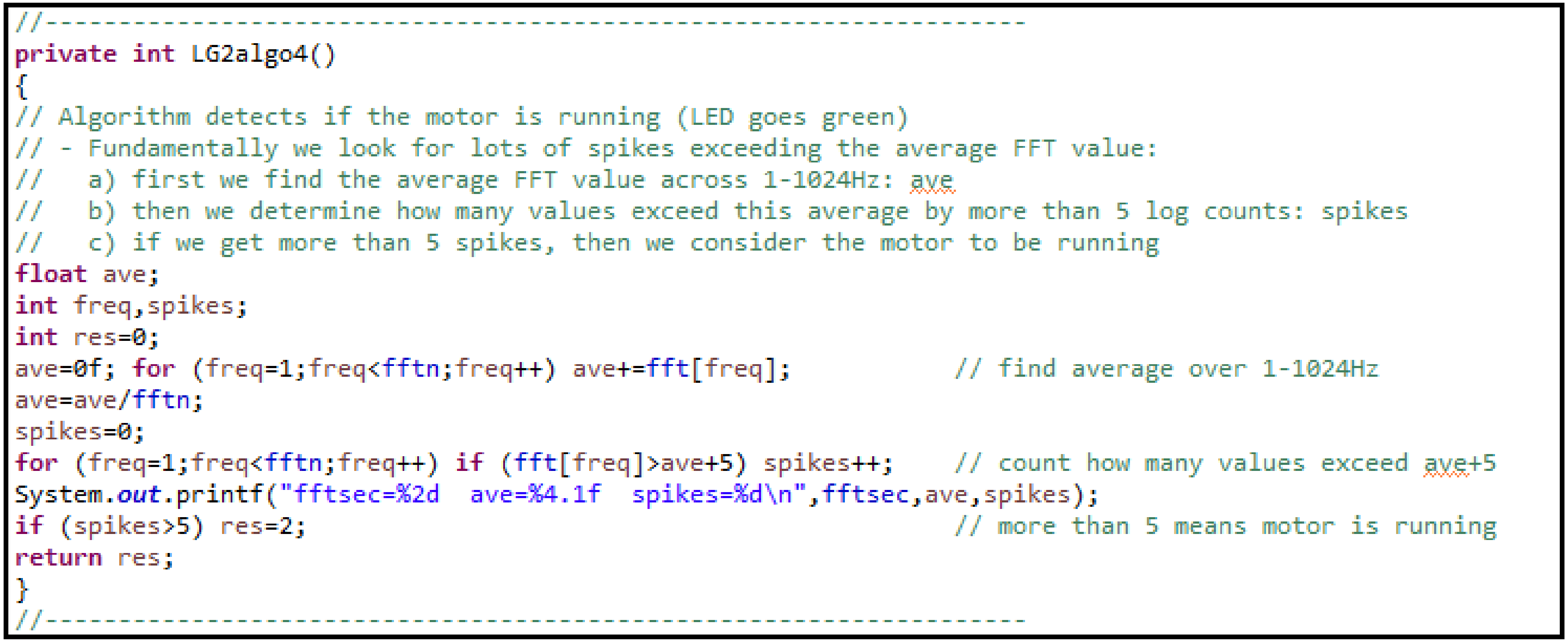
```
//----------------------------------------------------------------------
private int LG2algo5()
{
// Algorithm tries to determine presence of 1 of 3 tuning forks: A (440Hz), C (523Hz), Bb (466Hz)
// - same as before, but we check 3 frequencies; A=red, C=green, Bb=blue
// - same weaknesses as before
int res=0;
     if (fft[440]>20) res=1;
else if (fft[523]>20) res=2;
else if (fft[466]>20) res=3;
return res;
}
//----------------------------------------------------------------------
```

Tuning Forks Algorithms



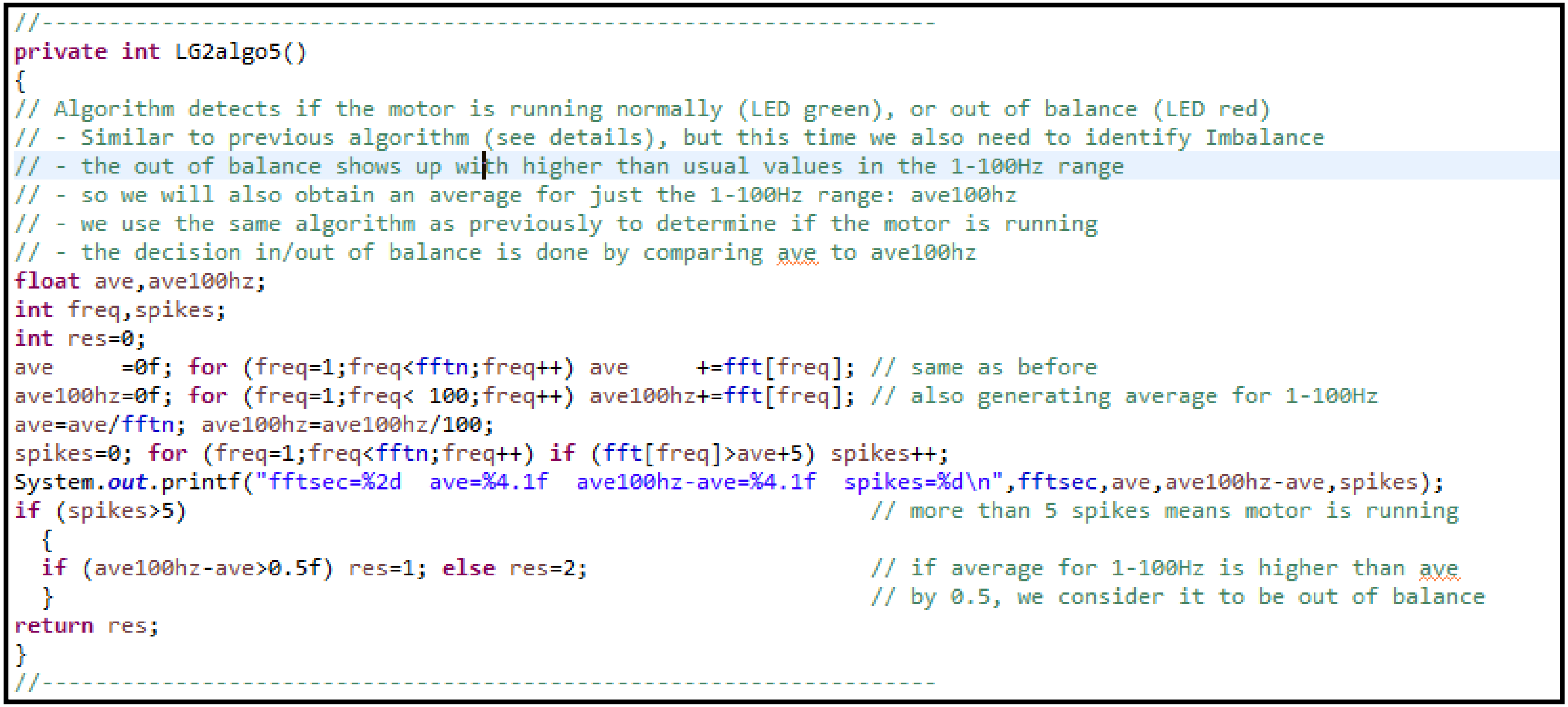
```
//-------------------------------------------------------------------
private int LG2algo6()
{
// Algorithm tries to determine presence of 1 of 3 tuning forks: A (440Hz), C (523Hz), Bb (466Hz)
// - this time we try a more thorough approach: we look for a "clean spike"
//     a) first we determine the highest value within the 1-1024Hz FFT range
//        we call this reference value refval, and will use it to set a relative threshold
//        we also keep track of the associated frequency, reffreq, and will compare it to the forks
//     b) we now look again, for values that are within 2 log counts, but at least 10Hz away, of spike
//        if we find more than 1, we throw everything away: we did not have a "clean spike"
//        if we find only 1 other value, then we assume that we are looking at a tuning fork
//     c) if we have a fork, then we compare it to our 3 fork values +/- 5Hz
final int freq1=440,freq2=523,freq3=466;
float refval;
int freq,reffreq,badspikes;
int res=0;
// Find "spike", i.e. highest value within full frequency spectrum
refval=0f; reffreq=0; for (freq=1;freq<fftn;freq++) if (fft[freq]>refval) {refval=fft[freq]; reffreq=freq;}
// Search for other (bad) spikes close to this one in value (up to 2 log counts below) but at least 10Hz away
badspikes=0;
for (freq=1;freq<fftn;freq++) if ((fft[freq]>refval-2)&&(Math.abs(freq-reffreq)>10)) badspikes++;
// Show "debug" results (optional)
System.out.printf("fftsec=%2d  highest spike: freq=%4dHz  value=%4.1f   badspikes=%d\n",fftsec,reffreq,refval,badspikes);
// Tuning Fork Results
if (badspikes<2)
  {
       if (Math.abs(reffreq-freq1)<5) res=1;     // similar test to previous algorithm, but we use a +/- 5Hz window
  else if (Math.abs(reffreq-freq2)<5) res=2;
  else if (Math.abs(reffreq-freq3)<5) res=3;
  }
return res;
}
//-------------------------------------------------------------------
```

Motors Algorithms



```
//-------------------------------------------------------------------
private int LG2algo4()
{
// Algorithm detects if the motor is running (LED goes green)
// - Fundamentally we look for lots of spikes exceeding the average FFT value:
//   a) first we find the average FFT value across 1-1024Hz: ave
//   b) then we determine how many values exceed this average by more than 5 log counts: spikes
//   c) if we get more than 5 spikes, then we consider the motor to be running
float ave;
int freq,spikes;
int res=0;
ave=0f; for (freq=1;freq<fftn;freq++) ave+=fft[freq];           // find average over 1-1024Hz
ave=ave/fftn;
spikes=0;
for (freq=1;freq<fftn;freq++) if (fft[freq]>ave+5) spikes++;    // count how many values exceed ave+5
System.out.printf("fftsec=%2d  ave=%4.1f  spikes=%d\n",fftsec,ave,spikes);
if (spikes>5) res=2;                                            // more than 5 means motor is running
return res;
}
//-------------------------------------------------------------------
```

Motors Algorithms



```
//------------------------------------------------------------------
private int LG2algo5()
{
// Algorithm detects if the motor is running normally (LED green), or out of balance (LED red)
// - Similar to previous algorithm (see details), but this time we also need to identify Imbalance
// - the out of balance shows up with higher than usual values in the 1-100Hz range
// - so we will also obtain an average for just the 1-100Hz range: ave100hz
// - we use the same algorithm as previously to determine if the motor is running
// - the decision in/out of balance is done by comparing ave to ave100hz
float ave,ave100hz;
int freq,spikes;
int res=0;
ave     =0f; for (freq=1;freq<fftn;freq++) ave     +=fft[freq]; // same as before
ave100hz=0f; for (freq=1;freq< 100;freq++) ave100hz+=fft[freq]; // also generating average for 1-100Hz
ave=ave/fftn; ave100hz=ave100hz/100;
spikes=0; for (freq=1;freq<fftn;freq++) if (fft[freq]>ave+5) spikes++;
System.out.printf("fftsec=%2d  ave=%4.1f  ave100hz-ave=%4.1f  spikes=%d\n",fftsec,ave,ave100hz-ave,spikes);
if (spikes>5)                                                   // more than 5 spikes means motor is running
  {
  if (ave100hz-ave>0.5f) res=1; else res=2;                     // if average for 1-100Hz is higher than ave
  }                                                             // by 0.5, we consider it to be out of balance
return res;
}
//------------------------------------------------------------------
```

Sensor Fusion-Domain

LAB GUIDE

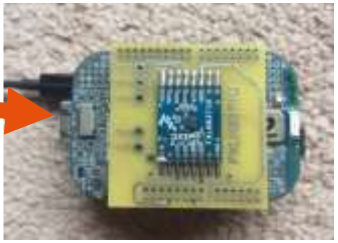
Hands-on: Compare data between dumb and intelligent data loggers, and understand the techniques used to compress the information

- Lab objective 1 – Review steps involved to collect data at different stations pertaining to the frequency domain.
- Lab objective 2 – Collect data at the stations – only at the sensor fusion domain stations
- Lab objective 3 – Test you algorithms written on real hardware for the sensor fusion domain!

PLEASE REMEMBE THIS LAB GUIDE IS AN INSTRUCTION GUIDE not an exact step-by-step guide.

Programming your Hardware Board

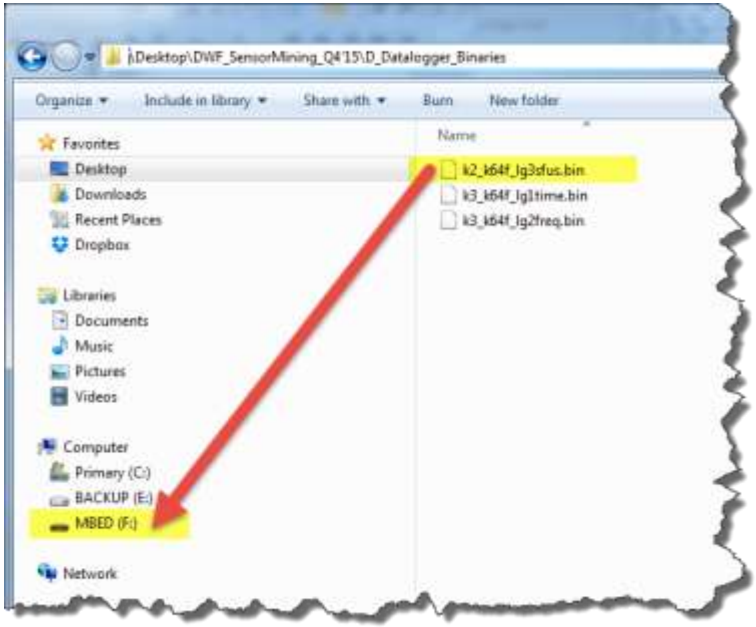
1. Plug in board to micro USB port labeled “OpenSDA”…



2. Open Windows Explorer and migrate to the desktop folder: *Desktop\DWF_SensorMining_Q4'15\D_Datalogger_Binaries*

3. You should see a **MBED** (drive) letter. Please let instructor know if you don’t get a drive called “**MBED**”.

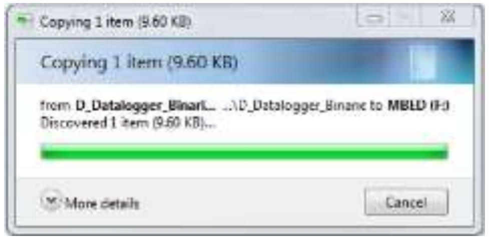
4. Drag and drop the file “*k2_k64f_lg3sfus.bin*” on top of the **MBED** drive.



\Desktop\DWF_SensorMining_Q4'15\D_Datalogger_Binaries

k2_k64f_lg3sfus.bin
k3_k64f_lg1time.bin
k3_k64f_lg2freq.bin

5. You should see a progress screen…



Copying 1 item (9.60 KB)

Copying 1 item (9.60 KB)

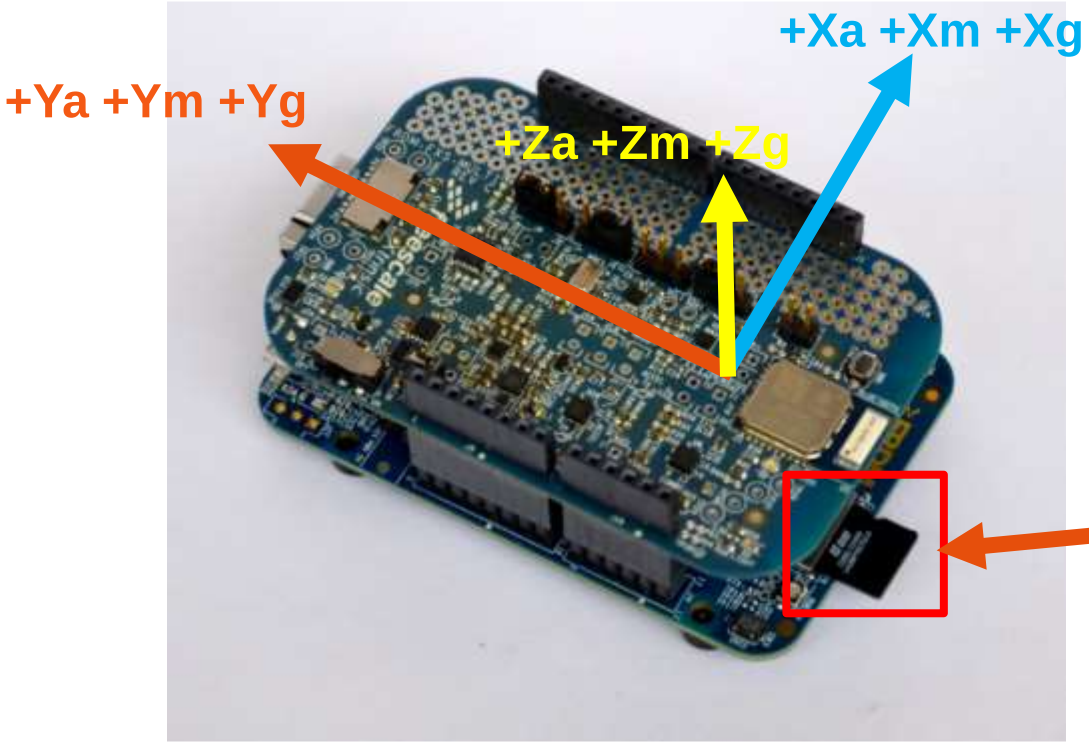
from **D_Datalogger_Binari...** ...\D_Datalogger_Binarie to **MBED (F:)**
Discovered 1 item (9.60 KB)...

6. Programming complete!

HARDWARE Board Description

Hardware Datalogger Board for the **SENSOR FUSION LABS:**

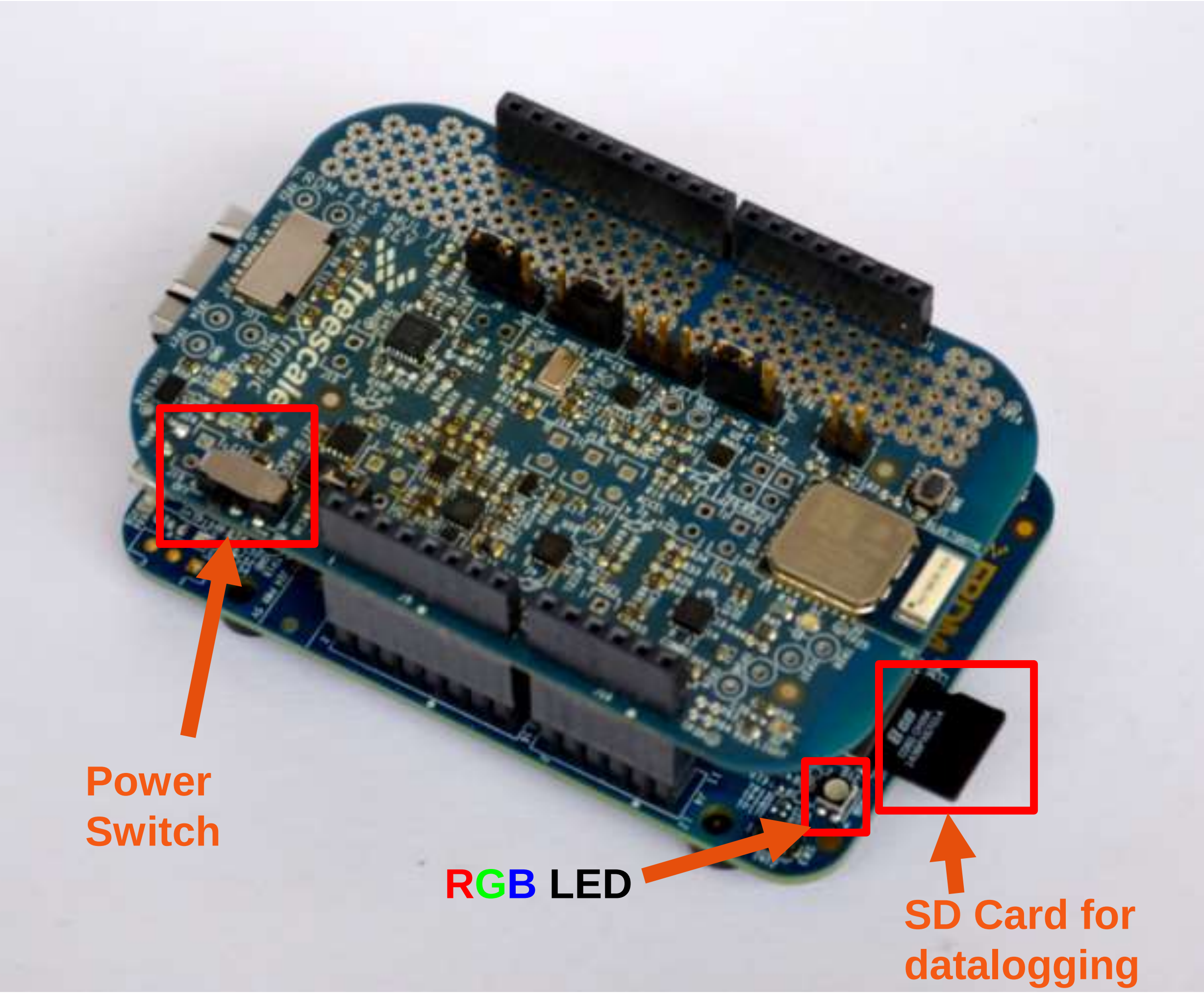
- FRDM-K64F board + FRDM-FXS-MULTI-B



All Sensors AXIS orientation
a = Accelerometer
m = Magnetometer
g = Gyroscope

SD Card Slot used for datalogging

INSTRUCTION - Starting the datalogger



To start the datalogger:

1. Insert SD Card into holder

2. Turn on Power switch (might have to cycle power if RED LED doesn't come on)

3. RGB LED will go blue. You need to calibrate the board by rotating it SLOWLY in all three axis. Calibration is complete when the LED turns RED. 🔵 🔴

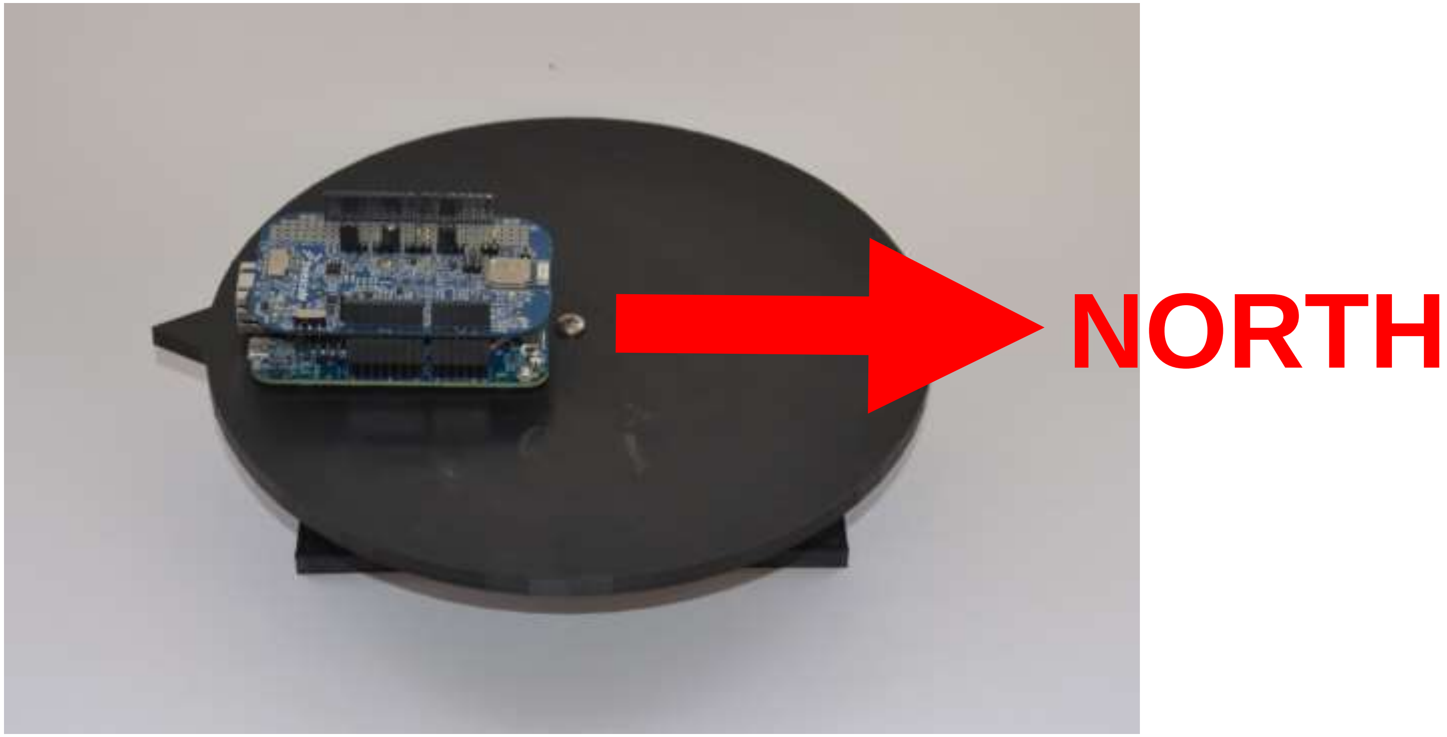
1. RGB LED will go solid red for 5 seconds

2. Begin Datalogging when RGB LED blinks green 🟢

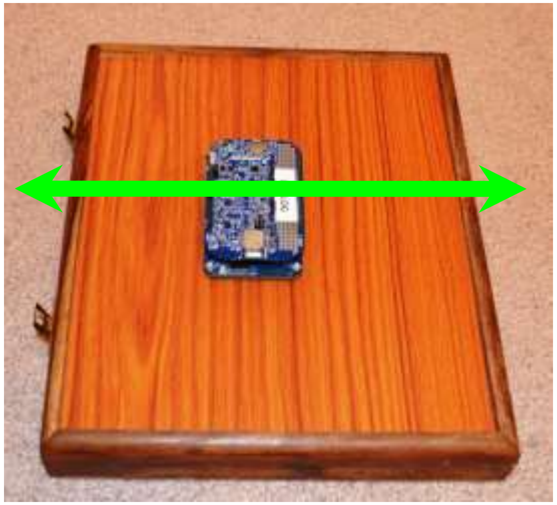
3. Datalogging stops after 30 seconds, when RGB LED goes solid green 🟢

Lab Station 3 – Compass Lab

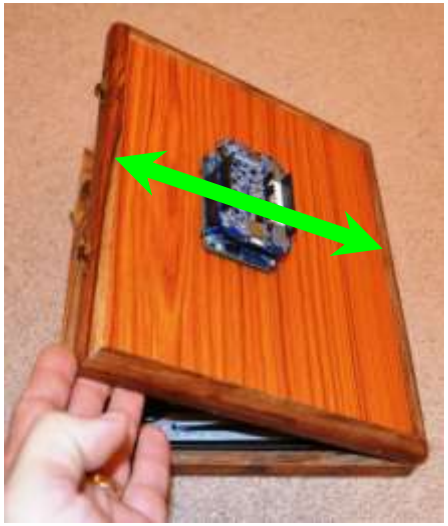
1. Orient the board on the platform like the picture below. Start Datalogger.
2. Manually rotate the platform to the eight positions from 0 - 360 deg, stopping at every 45 deg increments.
3. Stop at every 45 deg positions for about 2-3 seconds each.
4. When LED goes solid green, take back to desk and copy file onto laptop.



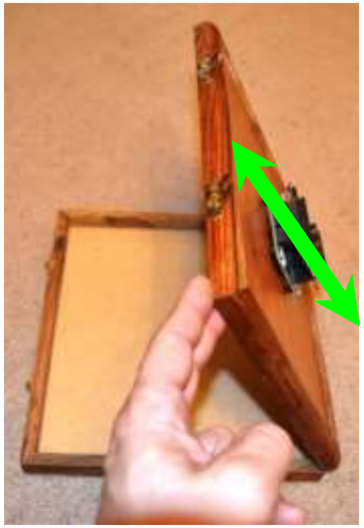
Lab Station 4 – Static and Linear Acceleration Lab



Move the board back and forth with one hand.



Continue to move the board back and forth, but now start tilting the platform up to 45 deg with you other hand.

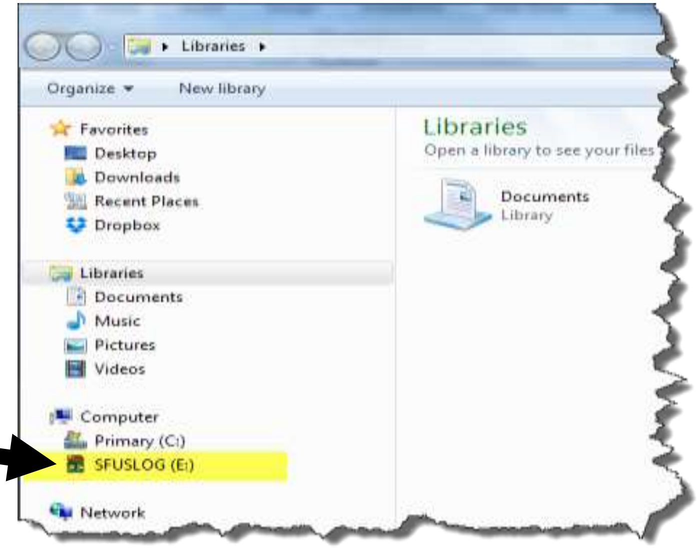


Continue to move the board back and forth, and finish tilting the platform up to 90 deg.

1. Start the datalogger.
2. Move the board back and forth your right hand, while slowly rotating the platform from 0 to 90deg, then 90 to 0 deg, with your left hand. Repeat now 0-90 deg, and 90-0 deg, without moving the board.
3. When LED goes solid green, take back to desk and copy file onto laptop.

INSTRUCTION - Copying datalogger files onto laptop

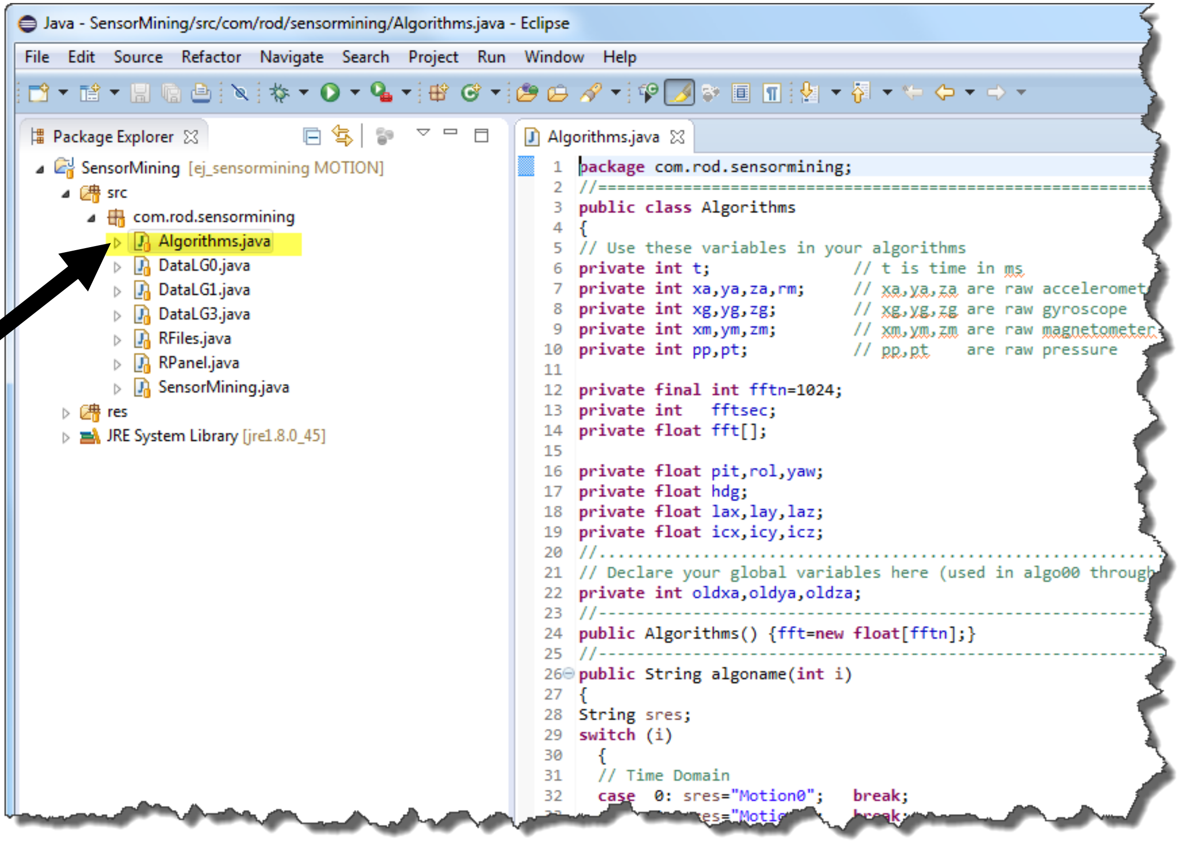
1. Turn off Datalogger.
2. Remove micro SDCard from hardware board.
3. Insert micro SDCard into adapter and insert into laptop SDCard slot.
4. The SDCard will show up as “SFUSLOG (drive):”
5. Copy “DATALOG.LG1” to location: *C:\Desktop\DWF_SensorMining_Q4'15\C_Sensor_Logs* (***Please do not add any additional folders***)
6. Rename “DATALOG.LG1” to any name with up to 9 characters, no spaces, no special characters. Do not change the extension.



Open the Eclipse Java GUI

1. Open Eclipse Mars on the Desktop, or in the DWF_SensorMining_Q4'15\A_Installers

2. In the Java "Package Explorer" Window, drop down, SRC, and com.rod.sensormining. Double-Click in the file called "Algorithms.java" **NOTE: This is the ONLY file you can modify any source code!**



Java - SensorMining/src/com/rod/sensormining/Algorithms.java - Eclipse

```
package com.rod.sensormining;
//=============================================================
public class Algorithms
{
// Use these variables in your algorithms
private int t;                  // t is time in ms
private int xa,ya,za,rm;        // xa,ya,za are raw acceleromet
private int xg,yg,zg;           // xg,yg,zg are raw gyroscope
private int xm,ym,zm;           // xm,ym,zm are raw magnetometer
private int pp,pt;              // pp,pt    are raw pressure

private final int fftn=1024;
private int   fftsec;
private float fft[];

private float pit,rol,yaw;
private float hdg;
private float lax,lay,laz;
private float icx,icy,icz;
//...........................................................
// Declare your global variables here (used in algo00 through
private int oldxa,oldya,oldza;
//-----------------------------------------------------------
public Algorithms() {fft=new float[fftn];}
//-----------------------------------------------------------
public String algoname(int i)
{
String sres;
switch (i)
  {
  // Time Domain
  case  0: sres="Motion0";   break;
```

How to switch between the labs in this class

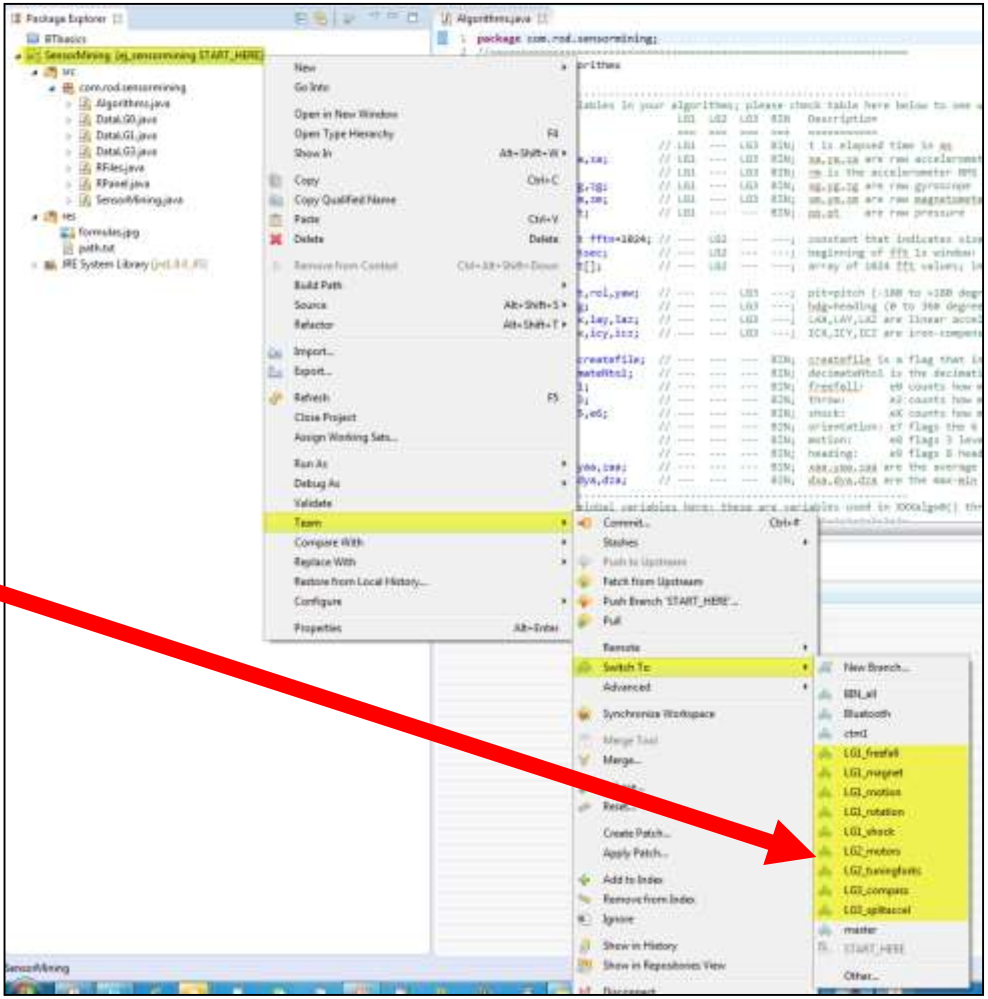
We will have 2 labs with several Algorithms in each lab to explore.

The Sensor Fusion Labs are:

1. LG3_compass
2. LG3_spitaccel

- To switch between them, right click on the "*SensorMining*" Project and select ***Team / Switch To / LG1,LG2,LG3*** (for example, anything in yellow), for this first lab of Freq domain, we are going to start with <u>LG3_compass</u>.

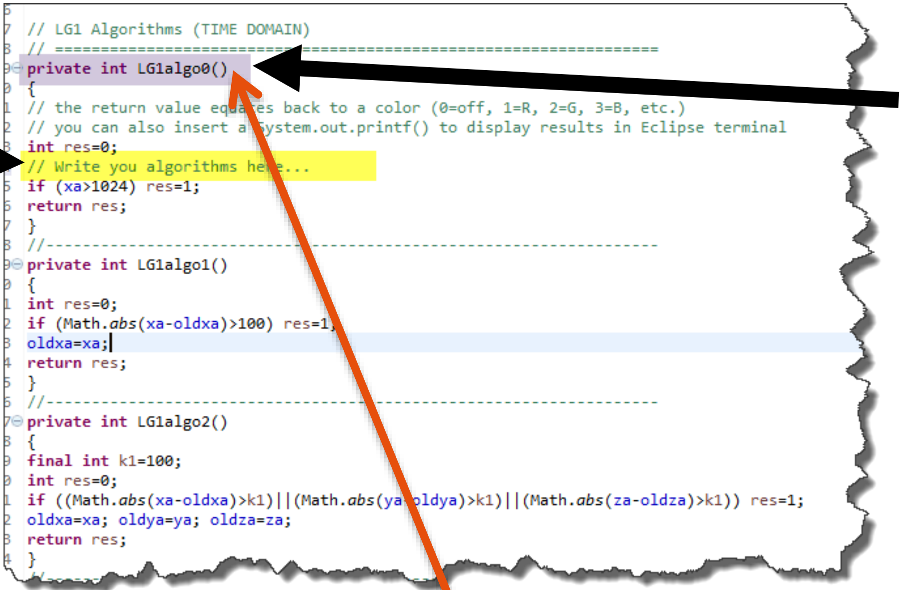
- Next, Hit <u>CTRL-F11</u> or Menu Run/Run (to run Algorithms).



Type you Algorithms here…and assign a button to it!

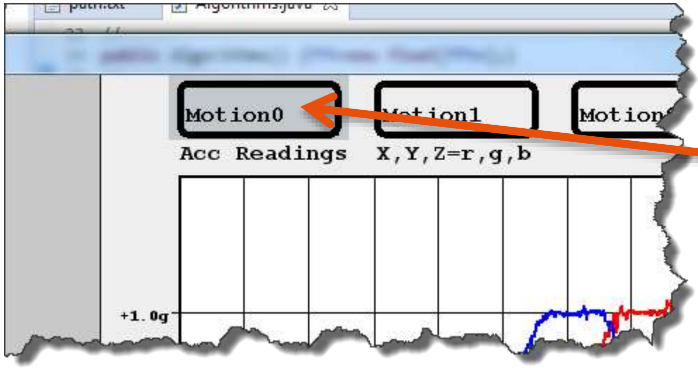
In the File, Algorithms.java, this is where you can type you algorithms **OR** observe and study the examples already done.

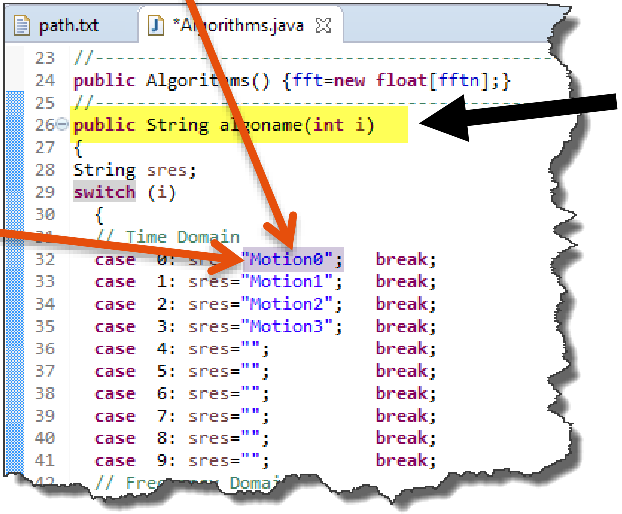
This slide is an example of how to re-assign buttons: This applies to all labs:
LG1 – Time Domain
LG2 – Frequency Domain
LG3 – Sensor Fusion Domain



```java
// LG1 Algorithms (TIME DOMAIN)
// ====================================================================
private int LG1algo0()
{
// the return value equates back to a color (0=off, 1=R, 2=G, 3=B, etc.)
// you can also insert a System.out.printf() to display results in Eclipse terminal
int res=0;
// Write you algorithms here...
if (xa>1024) res=1;
return res;
}
//--------------------------------------------------------------------
private int LG1algo1()
{
int res=0;
if (Math.abs(xa-oldxa)>100) res=1;
oldxa=xa;
return res;
}
//--------------------------------------------------------------------
private int LG1algo2()
{
final int k1=100;
int res=0;
if ((Math.abs(xa-oldxa)>k1)||(Math.abs(ya-oldya)>k1)||(Math.abs(za-oldza)>k1)) res=1;
oldxa=xa; oldya=ya; oldza=za;
return res;
}
```

Notice the function:
private int LG1algo0():
This is the name of the algorithm that can be assigned to a button.
THIS IS AN EXAMPLE.





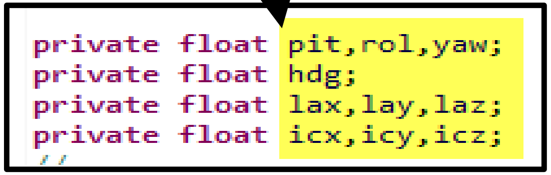
```java
//--------------------------------------------------------------------
public Algorithms() {fft=new float[fftn];}
//--------------------------------------------------------------------
public String algoname(int i)
{
String sres;
switch (i)
  {
  // Time Domain
  case  0: sres="Motion0";   break;
  case  1: sres="Motion1";   break;
  case  2: sres="Motion2";   break;
  case  3: sres="Motion3";   break;
  case  4: sres="";          break;
  case  5: sres="";          break;
  case  6: sres="";          break;
  case  7: sres="";          break;
  case  8: sres="";          break;
  case  9: sres="";          break;
  // Frequency Domain
```

Just change the name in the function called:
public String algoname (int i)

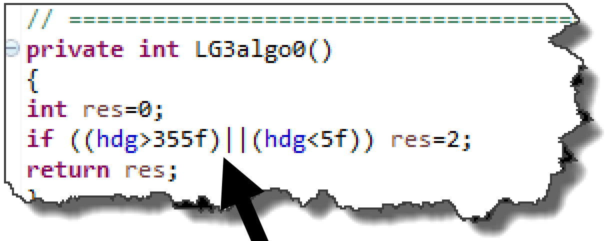
LG1algo0() is assigned to button name “Motion0”
LG1algo1() is assigned to button name “Motion1”
etc…etc…etc…

Here are the variable names you can use in you Algorithms…

LG3 Lab-
Sensor Fusion Domain variables



```
private float pit,rol,yaw;
private float hdg;
private float lax,lay,laz;
private float icx,icy,icz;
//
```



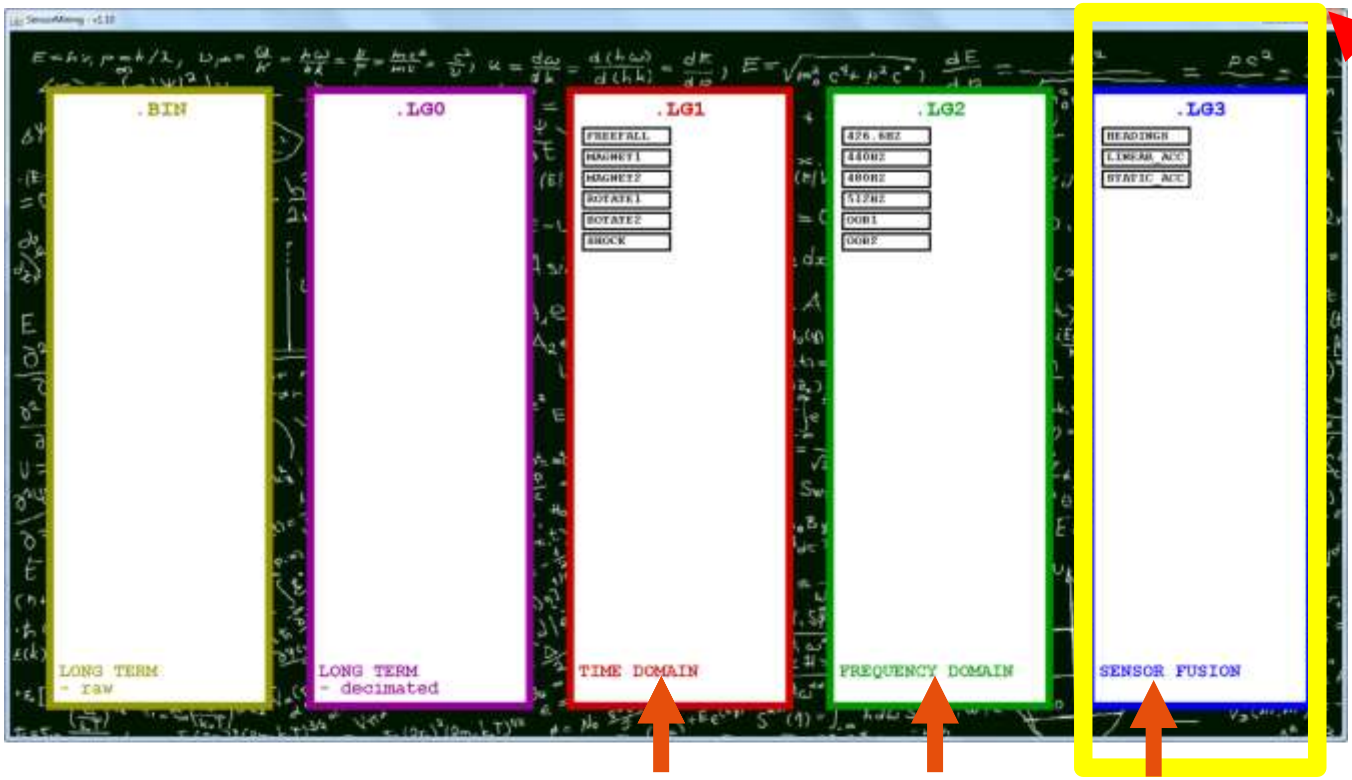
```
// ===================================
private int LG3algo0()
{
int res=0;
if ((hdg>355f)||(hdg<5f)) res=2;
return res;
}
```

Here is a LG3 Lab sample algorithm that detects a compass heading of North, and turns on the RED led if the condition is met.

SOFTWARE Description for the labs

- A description of the opening screen:

HIT ESC to return to this screen



To Close GUI

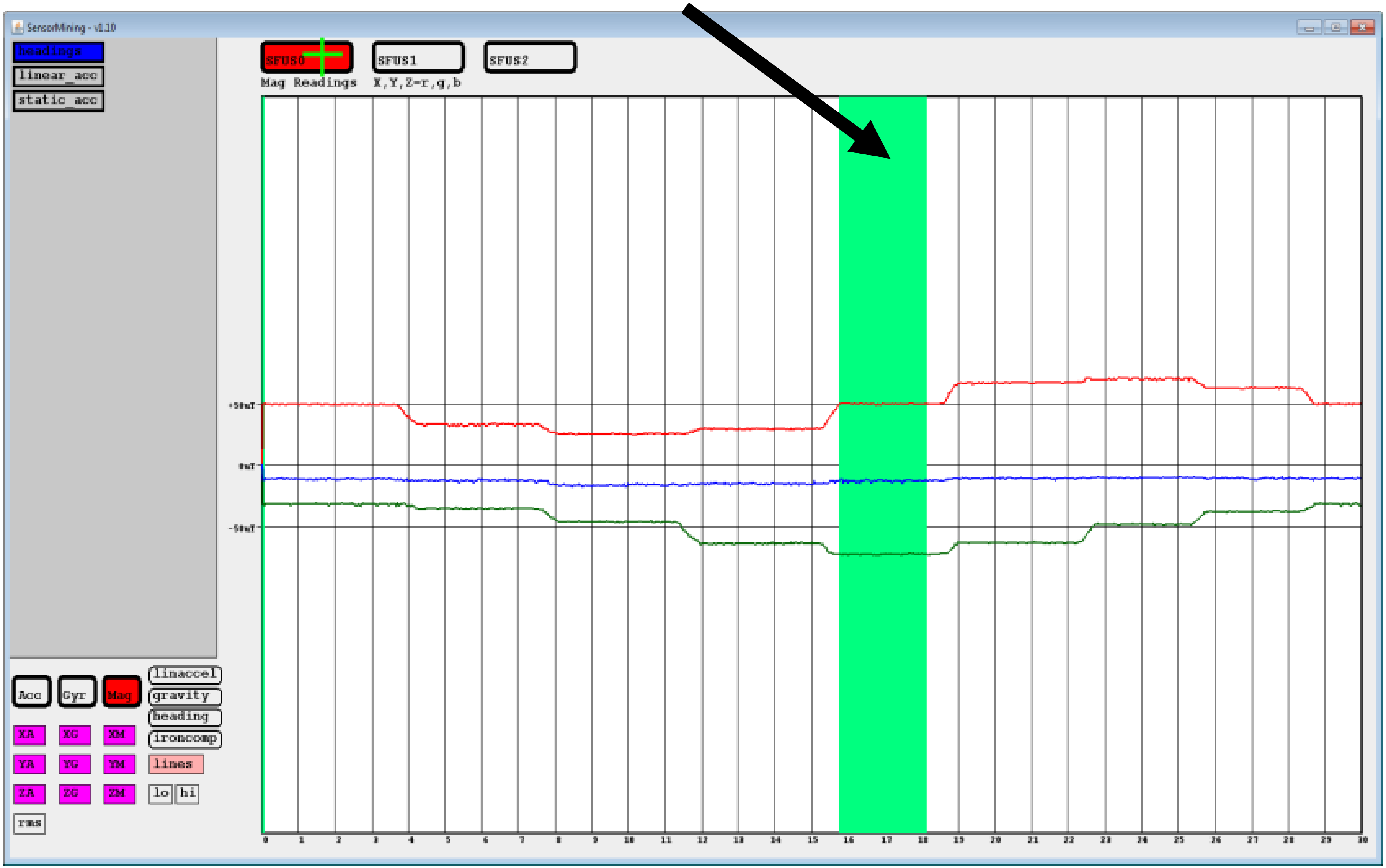
Time Domain Datalogger files Used in Part 1 of 3

Frequency Domain Datalogger files Used in Part 2 of 3

Sensor Fusion Datalogger files Used in Part 3 of 3

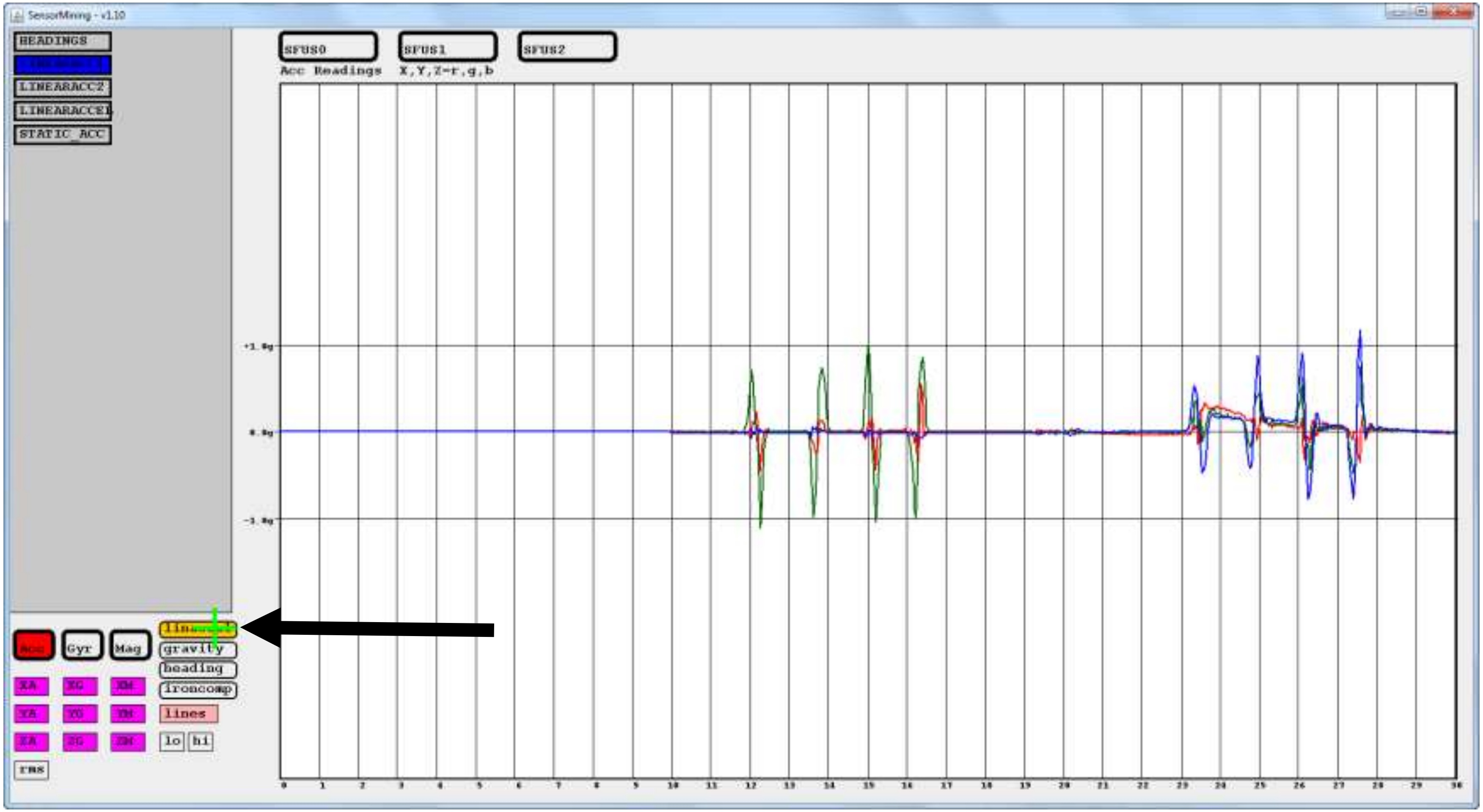
The SFUS0 algorithm is applied to my data in the Sensor Fusion Domain

GREEN LED Simulation when condition is met in algorithm



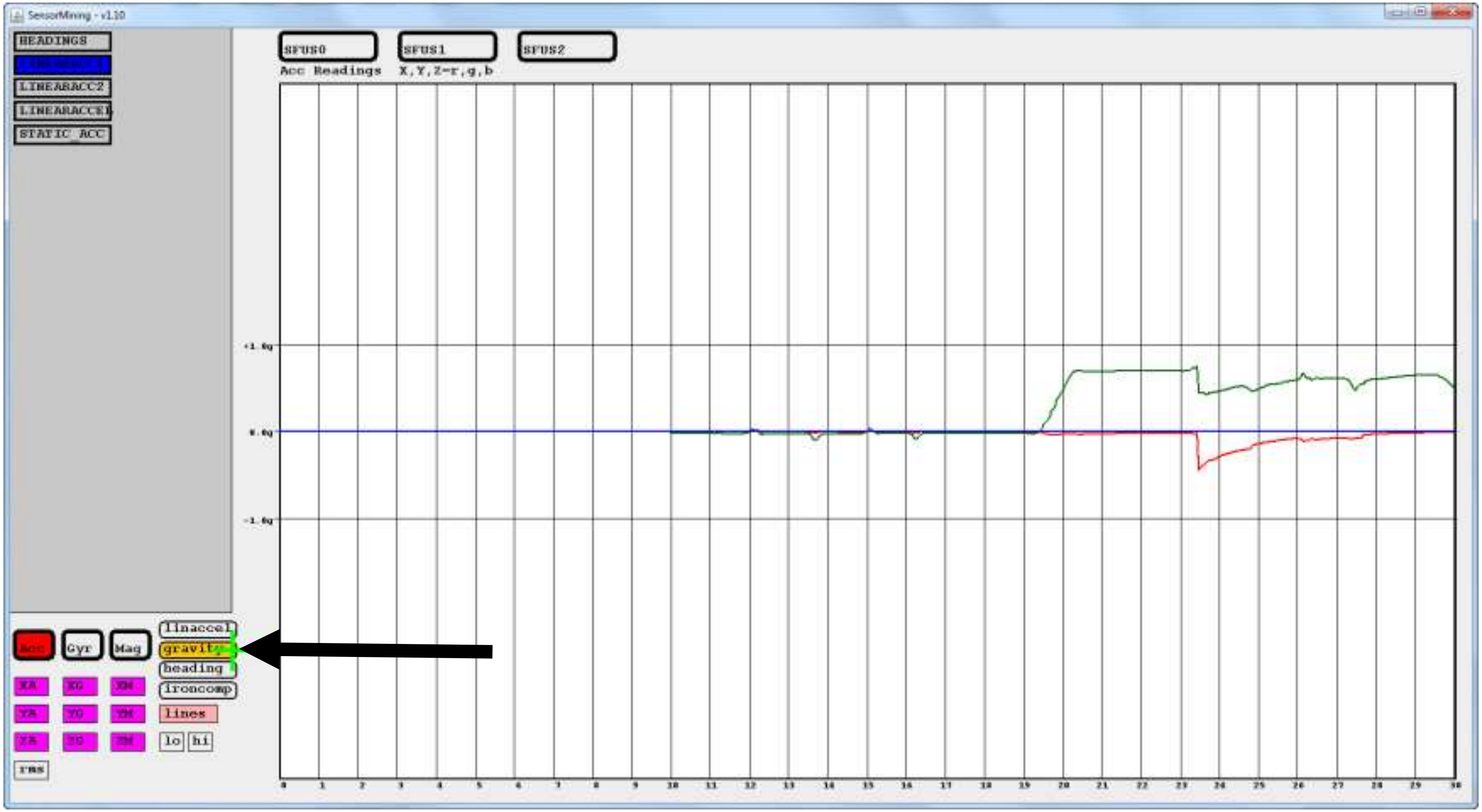
The linaccel button applied to my data in the Sensor Fusion Domain

Removes Static Acceleration

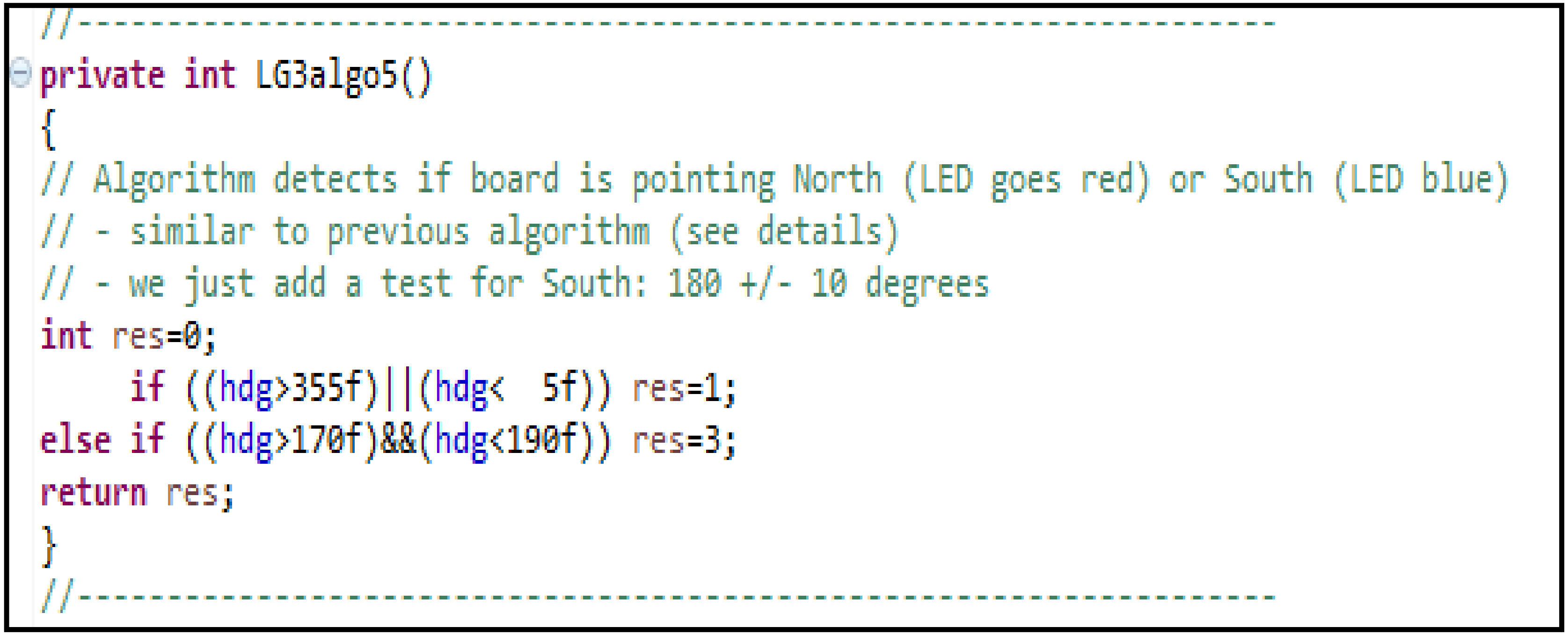


The gravity button applied to my data in the Sensor Fusion Domain

Removes Linear Acceleration

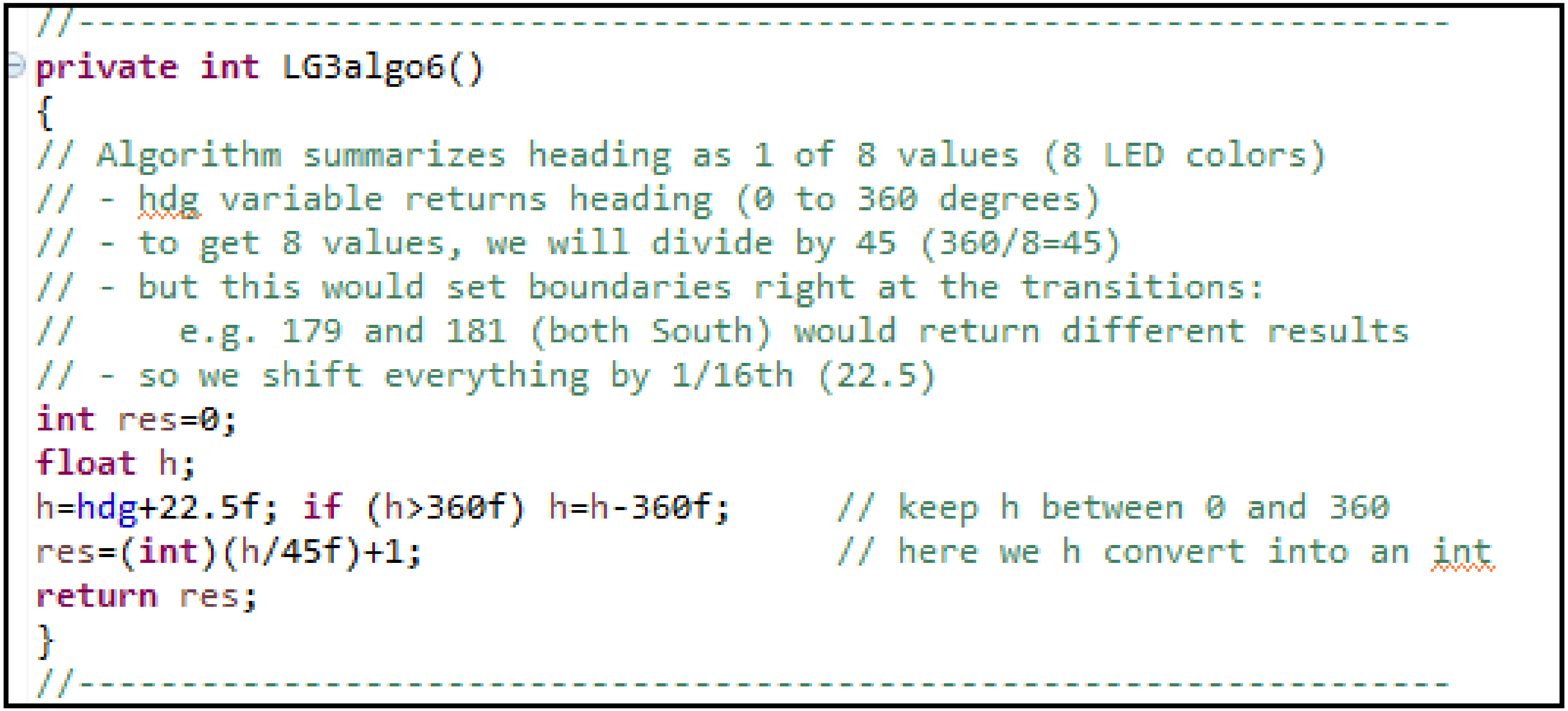


Compass Algorithms



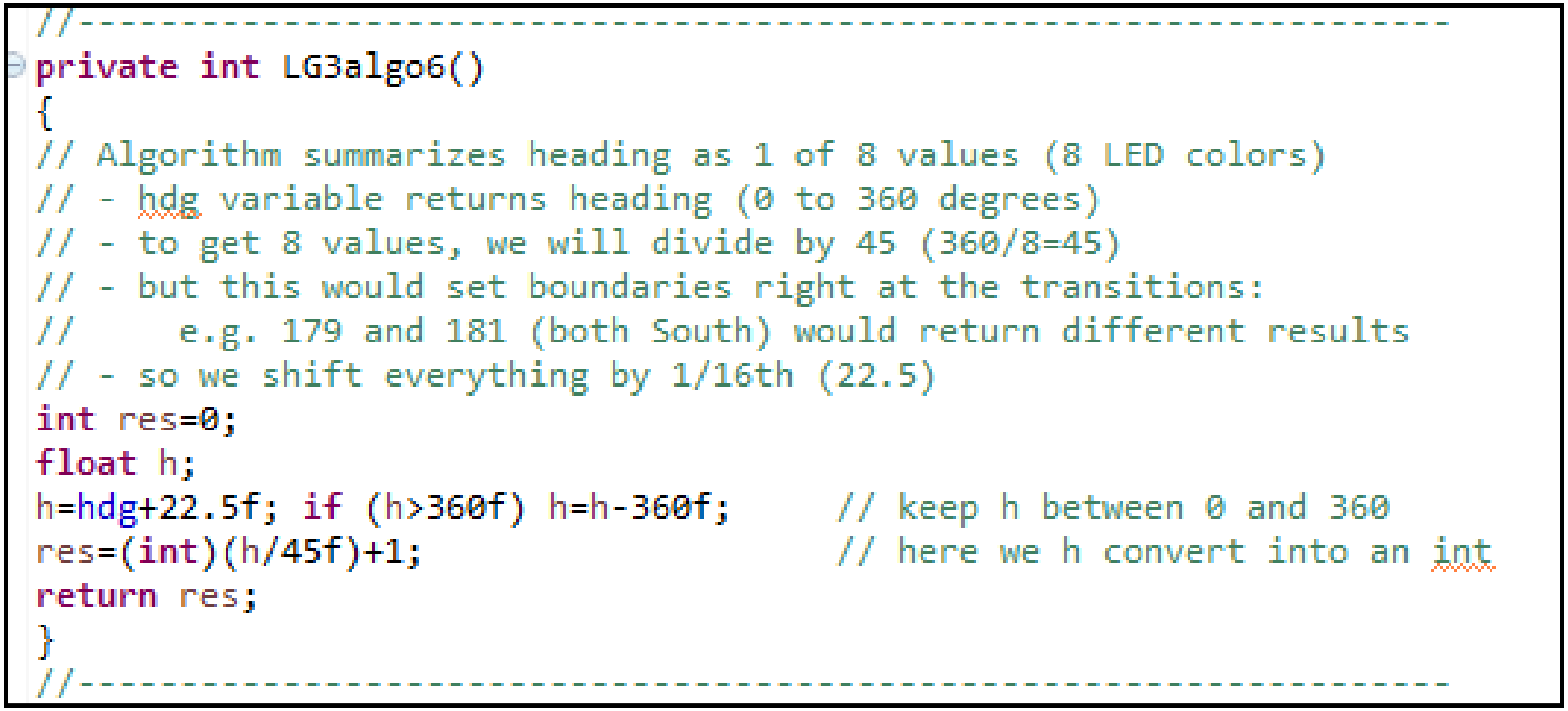
```
//-------------------------------------------------------------------
private int LG3algo5()
{
// Algorithm detects if board is pointing North (LED goes red) or South (LED blue)
// - similar to previous algorithm (see details)
// - we just add a test for South: 180 +/- 10 degrees
int res=0;
     if ((hdg>355f)||(hdg<  5f)) res=1;
else if ((hdg>170f)&&(hdg<190f)) res=3;
return res;
}
//-------------------------------------------------------------------
```

Compass Algorithms



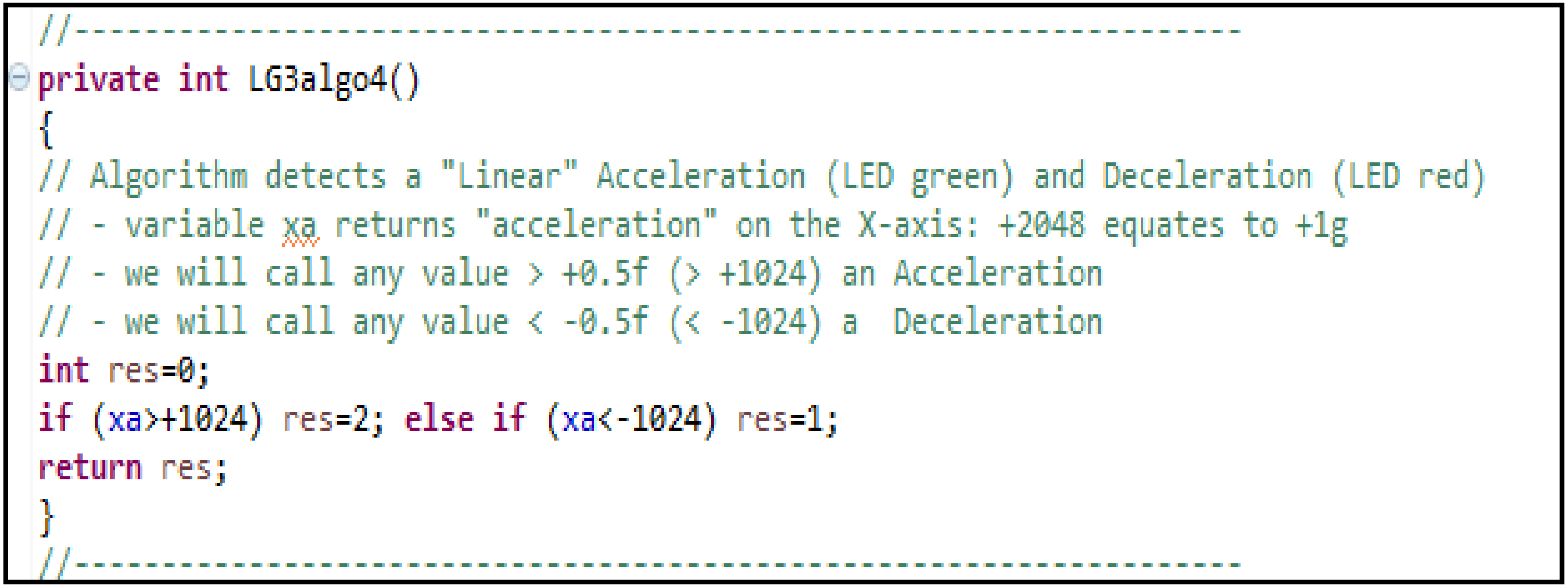
```
//-------------------------------------------------------------------
private int LG3algo6()
{
// Algorithm summarizes heading as 1 of 8 values (8 LED colors)
// - hdg variable returns heading (0 to 360 degrees)
// - to get 8 values, we will divide by 45 (360/8=45)
// - but this would set boundaries right at the transitions:
//     e.g. 179 and 181 (both South) would return different results
// - so we shift everything by 1/16th (22.5)
int res=0;
float h;
h=hdg+22.5f; if (h>360f) h=h-360f;      // keep h between 0 and 360
res=(int)(h/45f)+1;                      // here we h convert into an int
return res;
}
//-------------------------------------------------------------------
```

Compass Algorithms



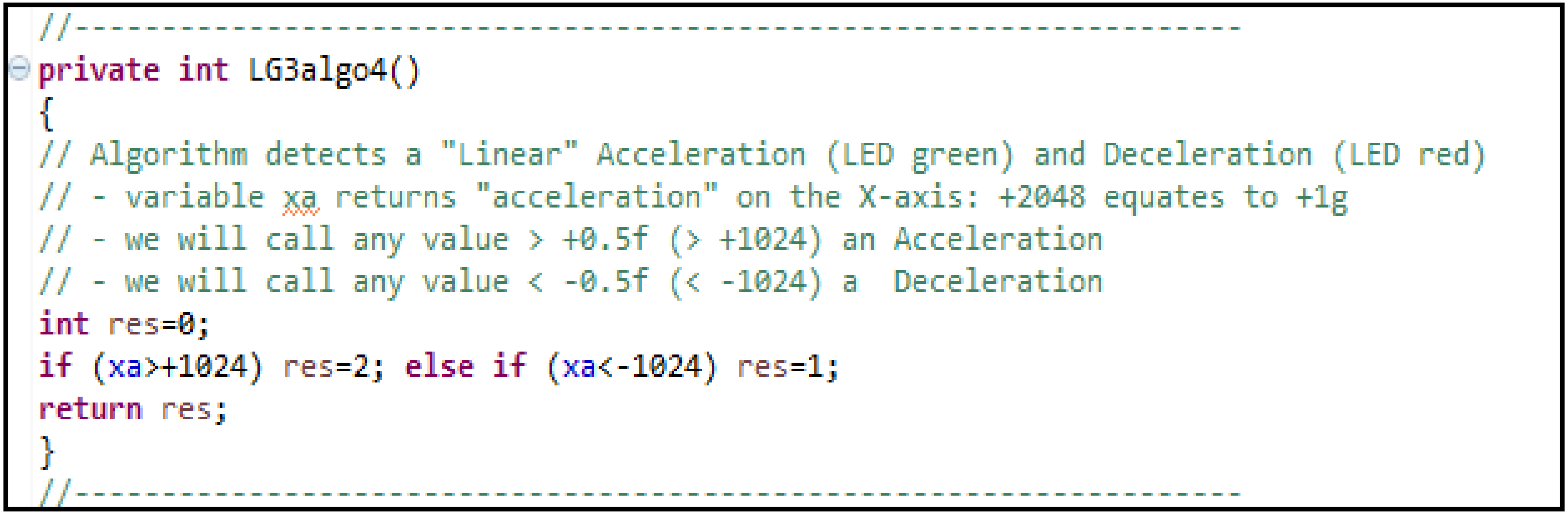
```
//-----------------------------------------------------------------
private int LG3algo6()
{
// Algorithm summarizes heading as 1 of 8 values (8 LED colors)
// - hdg variable returns heading (0 to 360 degrees)
// - to get 8 values, we will divide by 45 (360/8=45)
// - but this would set boundaries right at the transitions:
//     e.g. 179 and 181 (both South) would return different results
// - so we shift everything by 1/16th (22.5)
int res=0;
float h;
h=hdg+22.5f; if (h>360f) h=h-360f;      // keep h between 0 and 360
res=(int)(h/45f)+1;                     // here we h convert into an int
return res;
}
//-----------------------------------------------------------------
```

Static and Linear Acceleration Algorithms



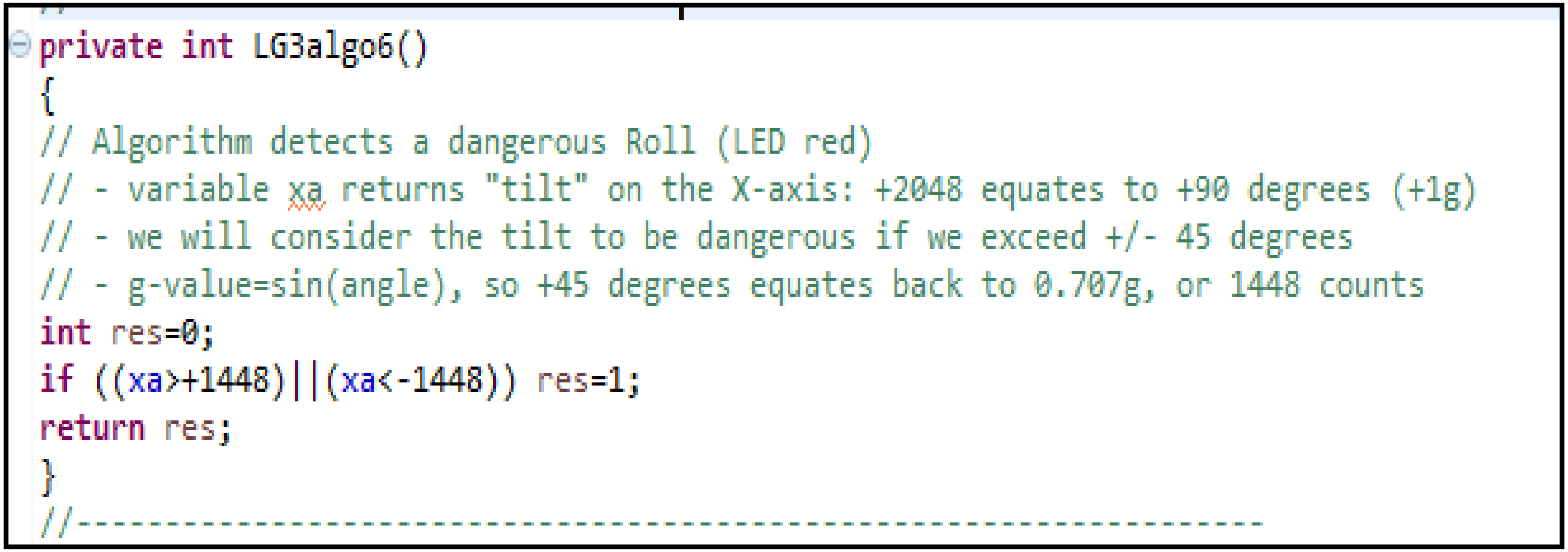
```
//-------------------------------------------------------------------
private int LG3algo4()
{
// Algorithm detects a "Linear" Acceleration (LED green) and Deceleration (LED red)
// - variable xa returns "acceleration" on the X-axis: +2048 equates to +1g
// - we will call any value > +0.5f (> +1024) an Acceleration
// - we will call any value < -0.5f (< -1024) a  Deceleration
int res=0;
if (xa>+1024) res=2; else if (xa<-1024) res=1;
return res;
}
//-------------------------------------------------------------------
```

Static and Linear Acceleration Algorithms



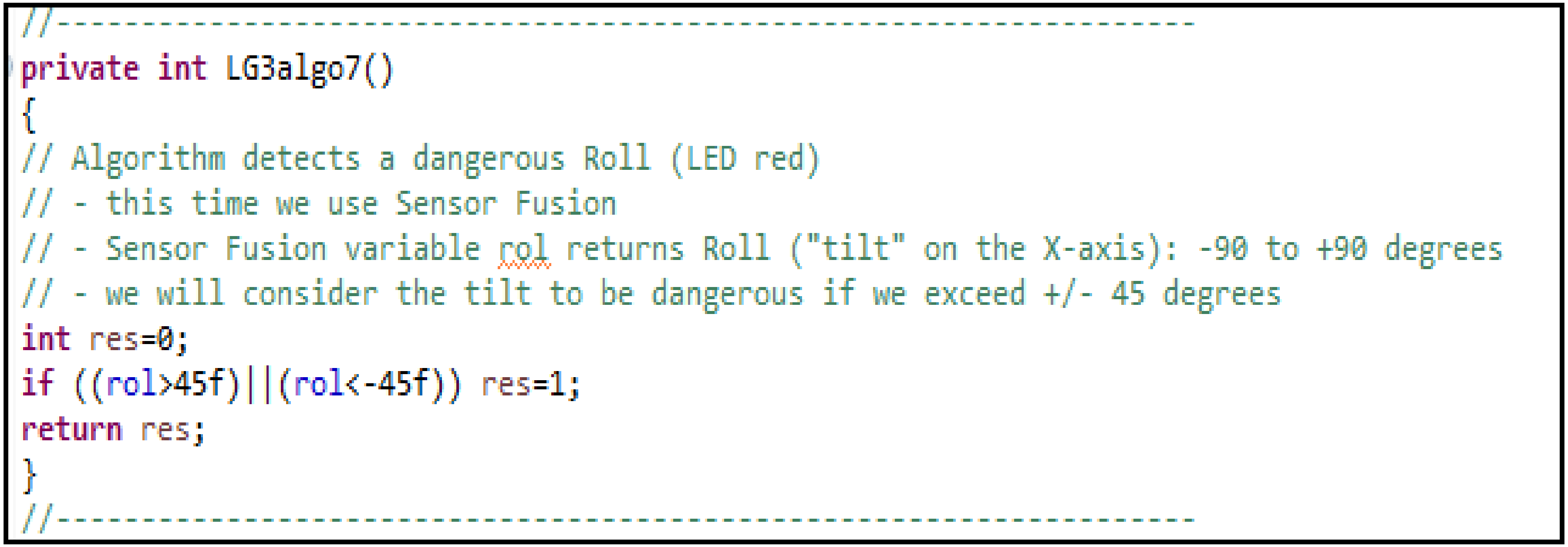
```
//------------------------------------------------------------------
private int LG3algo4()
{
// Algorithm detects a "Linear" Acceleration (LED green) and Deceleration (LED red)
// - variable xa returns "acceleration" on the X-axis: +2048 equates to +1g
// - we will call any value > +0.5f (> +1024) an Acceleration
// - we will call any value < -0.5f (< -1024) a  Deceleration
int res=0;
if (xa>+1024) res=2; else if (xa<-1024) res=1;
return res;
}
//------------------------------------------------------------------
```

Static and Linear Acceleration Algorithms



```
//
private int LG3algo6()
{
// Algorithm detects a dangerous Roll (LED red)
// - variable xa returns "tilt" on the X-axis: +2048 equates to +90 degrees (+1g)
// - we will consider the tilt to be dangerous if we exceed +/- 45 degrees
// - g-value=sin(angle), so +45 degrees equates back to 0.707g, or 1448 counts
int res=0;
if ((xa>+1448)||(xa<-1448)) res=1;
return res;
}
//------------------------------------------------------------
```

Static and Linear Acceleration Algorithms



```
//-------------------------------------------------------------------
private int LG3algo7()
{
// Algorithm detects a dangerous Roll (LED red)
// - this time we use Sensor Fusion
// - Sensor Fusion variable rol returns Roll ("tilt" on the X-axis): -90 to +90 degrees
// - we will consider the tilt to be dangerous if we exceed +/- 45 degrees
int res=0;
if ((rol>45f)||(rol<-45f)) res=1;
return res;
}
//-------------------------------------------------------------------
```

Free Tools we used

Tools work in Windows, Mac, and Linux!

- Eclipse IDE for Java Developers (we used Mars)
  - https://eclipse.org/downloads/

- Java JDK (we used Java JDK8u45)
  - http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

freescale™

www.Freescale.com