We reassigned the K[1] value from 250 to 33 on Fin 21BE, since it was only reading data at 4Hz and never upped the frequency to 33Hz, which is better for motion calibration data.

- **Konstants also remember value even after battery drains all the way**

Summary of Each Measurement

4.6 Three-Axis MEMS Gyroscope with 16-bit ADCs and Signal Conditioning

The MPU-9250 consists of three independent vibratory MEMS rate gyroscopes, which detect rotation about the X-, Y-, and Z- Axes. When the gyros are rotated about any of the sense axes, the Coriolis Effect causes a vibration that is detected by a capacitive pickoff. The resulting signal is amplified, demodulated, and filtered to produce a voltage that is proportional to the angular rate. This voltage is digitized using individual on-chip 16-bit Analog-to-Digital Converters (ADCs) to sample each axis. The full-scale range of the gyro sensors may be digitally programmed to ±250, ±500, ±1000, or ±2000 degrees per second (dps). The ADC sample rate is programmable from 8,000 samples per second, down to 3.9 samples per second, and user-selectable low-pass filters enable a wide range of cut-off frequencies.

4.7 Three-Axis MEMS Accelerometer with 16-bit ADCs and Signal Conditioning

The MPU-9250's 3-Axis accelerometer uses separate proof masses for each axis. Acceleration along a particular axis induces displacement on the corresponding proof mass, and capacitive sensors detect the displacement differentially. The MPU-9250's architecture reduces the accelerometers' susceptibility to fabrication variations as well as to thermal drift. When the device is placed on a flat surface, it will measure 0g on the X- and Y-axes and +1g on the Z-axis. The accelerometers' scale factor is calibrated at the factory and is nominally independent of supply voltage. Each sensor has a dedicated sigma-delta ADC for providing digital outputs. The full scale range of the digital output can be adjusted to ±2g, ±4g, ±8g, or ±16g.

4.8 Three-Axis MEMS Magnetometer with 16-bit ADCs and Signal Conditioning

The 3-axis magnetometer uses highly sensitive Hall sensor technology. The magnetometer portion of the IC incorporates magnetic sensors for detecting terrestrial magnetism in the X-, Y-, and Z- Axes, a sensor driving circuit, a signal amplifier chain, and an arithmetic circuit for processing the signal from each sensor. Each ADC has a 16-bit resolution and a full scale range of ±4800 μT.

Summary of the code documentation

Turning On the Fin

- These are given by definitions in the header file Smartphin_Firmware.h
    - The board has to be in the vertical position "nose pointing up" to actually be turned on
        - Attempting to turn on the board (following all instructions) while it is not vertical results in a pause and no green LED response
    - The board cannot be moving sporadically when turning on; keep it as still as possible
    - Tap the fin from the top of the board, or flat edge of the fin 3 times
        - There needs to be a minimum 600 ms delay between each tap
        - All 3 taps must be done within 3 seconds from first tap to last tap

Accelerometer:

- The data is stored as a 16-bit signed binary 2's complement (2's complement is not something we have to worry about unless we have to modify the RAW binary numbers themselves).
    - The first bit is used to indicate whether the number is positive or negative.
    - This gives us 15 bits, to work with, which in binary gives a max value of 32,768.
- There are four different sensitivity settings:
    - AFS_SEL==0 +/- 2G 16384 LSB/G
    - AFS_SEL==1 +/- 4G 8192 LSB/G
    - AFS_SEL==2 +/- 8G 4096 LSB/G
    - AFS_SEL==3 +/- 16G 2048 LSB/G
        - LSB/G = Least Significant Bits per G ($9.8 m/s^2$)
- The default setting is AFS_SEL==3, or 1/2048 G's per bit
    - Doing a simple calculation shows that 32,768 bits * 1/2048 G's/bit gives us 16 G's of acceleration.
- The RAW binary data, however, is bit-shifted to the right twice, making the max range only 8,192
    - This is not an issue either and can save on processing and recording times as well as taking up less space. I am not an expert computer scientist though and this is just a theory of why they did this..
        - Bit shifted digits are always set to 0 before shifting (acts like a floor function)
- To take the RAW readings that are present in the CSV files, we have to correct for bit shifting and the sensitivity to get accurate and understandable accelerations
    - The code gives a definition that we can use as a sample calculation:

```
#define Y_AXIS_GRAVITY_THRESHOLD 350  /* threshold for checking ~1G
                                         accel in Y axis, i.e. fin in 'surfing position' */
                                      // a value of 350 is about 0.7G or a 45 degree angle with fin upwards
```

- Their definition says 350 corresponds with about 0.7G. Let's see if our calculations can get that value.
- First we correct for bit shifting by multiplying by 4 (moving a bit left or right corresponds to multiplying the number by 2 or dividing by 2 respectively). Our number is now 1400.
- Next we take this number and use the sensitivity setting to convert bits into G's. 1400 bits * 1/2048 G's per bit = 0.683G's ~ 0.7G's. Our correction method should work.

- In short, dividing the CSV numbers by 512 will give us the correct acceleration in G's. Alternatively multiply by 0.019141 will give us a value in m/s^2.

Acceleration IMU Axes/Directions

- I performed a control test "surf" session with Fin 21BE on July 10th to find out the axis of directions relative to the board
  - In my findings (linked in this spreadsheet: https://docs.google.com/spreadsheets/d/1bQDcSCBISFPmUHQYo582VlvvW7jgZo0SENUfyWbZNXk/edit#gid=69018441), I found that:
  - The numbers in A1 represented forward and backward acceleration relative to the board.
    - Negative numbers → forward acceleration
    - Positive numbers → backwards acceleration
  - The numbers in A2 represented vertical acceleration relative to the board.
    - Negative numbers → upwards acceleration
    - Positive numbers → downwards acceleration (like gravity pulling downwards).
  - The numbers in A3 represented side-to-side acceleration relative to the board.
    - Negative numbers → acceleration to the left while surfing
    - Positive numbers → acceleration to the right while surfing.

Gyroscope:

- The data is stored as a 16-bit signed binary 2's complement, just like the accelerometer.
  - Again, the first bit is used to indicate whether the number is positive or negative.
  - This gives us 15 bits, to work with, which in binary gives a max value of 32,768.
- There are four different sensitivity settings:
  - FS_SEL==0 +/- 250 degrees/s 131.0 LSB/deg/s
  - FS_SEL==1 +/- 500 degrees/s 65.5 LSB/deg/s
  - FS_SEL==2 +/- 1000 degrees/s 32.8 LSB/deg/s
  - FS_SEL==3 +/- 2000 degrees/s 16.4 LSB/deg/s
    - LSB/deg/s = Least significant bits per degree per second (rotational velocity)
- The default setting is FS_SEL == 2, so each bit is 1/32.8 degrees per second.
  - The fastest rotation detection possible is 500 degrees per second, which is ~1.4 revolutions per second. This is most likely only achieved during a nasty wipeout and should not be seeing any motion of that magnitude.
- The RAW binary data, however, is bit-shifted to the right twice, making the max range only 8,192

    - The 3 rightmost bits are always set to 0 before bit shifting, making it act like a floor function
- To take the RAW readings that are present in the CSV files, we have to correct for bit shifting and the sensitivity to get accurate and understandable rotational velocities
    - Going off of what we've found, to correct the readings, we have to multiply by 4 to correct the bit shifting and multiply the new value by 1/32.8 degrees per second to get a new value.
- In short, dividing the CSV numbers by 8.2 should give us the correct rotational velocity in degrees per second.

Gyroscope IMU Axes/Directions

- I performed a control test "surf" session with Fin 21BE on July 10th to find out the axis of directions relative to the board
    - In my findings (linked in this spreadsheet: https://docs.google.com/spreadsheets/d/1bQDcSCBISFPmUHQYo582VlvvW7jgZo0SENUfyWbZNXk/edit#gid=69018441), I found that:
    - The numbers in G1 represented rotation in the roll axis relative to the board.
        - Negative → Roll left
        - Positive → Roll right
    - The numbers in G2 represented rotation in the yaw axis relative to the board.
        - Positive → Left yaw/turn
        - Negative → Right yaw/turn
    - The numbers in G3 represented rotation in the pitch axis relative to the board
        - Positive → Upwards pitch / flip backwards
        - Negative → Downwards pitch / flip forwards

Magnetometer:

- There is no scaling - it is always the same scale and range
- Scale is +/- 32760 with signed 16-bits (+/-8180 in 14-bit mode) with full scale being +/- 4192 micro-Teslas

  32760 4192(max) ← **Pretty sure this is in micro-Teslas**

  1 0.15 ← **Each bit is 0.15 micro-Teslas**

  -1 -0.15

  -32660 -4192(min)
- It is critical to understand that the Earth's magnetic field varies between 25 and 65 microteslas at earth's surface (+/- 166.67 - 433.33 bits)
    - This is far below the 4192 micro-Tesla range that is present on the device, and this range is most likely only useful for the magnetic swipe control that was implemented.
- Max reading from the "2018-05-01-Interactive-Smartfin-Surfrider-Team-Surf.ipynb" jupyter file was 363 → 363 bits * 0.15 uT/bit = 54.45 uT, which is within the range of Earth's magnetic field
- One issue that is present is figuring out which direction is north using only these magnetic readings which can sometimes be a little too high (i.e. greater than 65 microteslas)

Magnetometer IMU Axes/Directions

- I performed a control test "surf" session with Fin 21BE on July 10th to find out the axis of directions relative to the board. A better test focused on the magnetometer was done July 11th

- In my findings (linked in this spreadsheet): https://docs.google.com/spreadsheets/d/1n8sh-F3KD4iqb_XwLmbqX93C85kHDhSrqGeDH_QnqtE/edit?usp=sharing, I found that:
- **We know that IMU numbers are all oriented in the same direction.**
  - I.e.: A2, G2, and M2 all share the vertical axis of the board, rather than what the diagram in the datasheet says on Page 38
- Certain orientations of the sensor have axes pointing in the same direction. This helps with understanding a general number the sensor should read when facing a certain direction
- A good portion of orientations read the same value when facing the same direction
  - i.e.: IMU 1 in surfing position and IMU 2 in "nose" down position share the same northward orientation and magnetic reading
- It can be seen that, as a sensor pivots around, the field characteristics can be seen as a MAX, a MIN, and two MEAN values
  - MAX is NORTH and MIN is SOUTH in all cases shown in the control data
  - WEST and EAST are associated with the MEAN values
- **IMPORTANT: IMU 3 consistently reads much larger values than the other two IMUs**
  - The measurements are ~100 units or 15uTeslas stronger than the others in every direction
  - My guess of the cause is the inductive charging circuit in the fin being oriented on the same axis
  - Doing calculations on the leakage current causing this,
    We can use the integral of Biot-Savart Law to give us the following equation:

$$B = \frac{\mu_0 NI}{2R}$$

Where B is magnetic field, R is radius, I is current, and μ_0 is magnetic permittivity = 4π x 10^{-7} T•m / A. N is added to account for the # of coils in the wire. We know B=15μT, N = 15, and R ~2.7cm

  - Solving for I we get:

$$I = \frac{2BR}{N\mu_0}$$

  - Plugging in numbers gives us 430μA, which is the leakage current through the the inductive pad, assuming assumptions are fairly correct
- Even though Hubb's Hall, according to Google Maps, is oriented directly North and should have the stronger North reading in the tests, the compasses were pointing correctly towards the magnetic North Pole
  - Magnetic North is located on Ellesmere Island in Canada, which is ~500km away from Geographic North pole. This explains the inland direction that magnetic North was pointing towards in my tests.
  - Magnetic North pointed roughly 15° West of Geographic North, which lines up pretty well with Google Maps
- I expected negative values from the data when orienting an IMU in the opposite direction

- **IMPORTANT NOTE**: Magnetic North is Compass Heading **+11° 28'** (East) and Inclination **57° 52'** with field strength 46.174 uTeslas http://www.magnetic-declination.com/
    - This will present problems when the fin is used outside of San Diego for mapping, unless a dataframe of regularly surfed spots is created and used as reference to make the orientation calculations

Temperature:

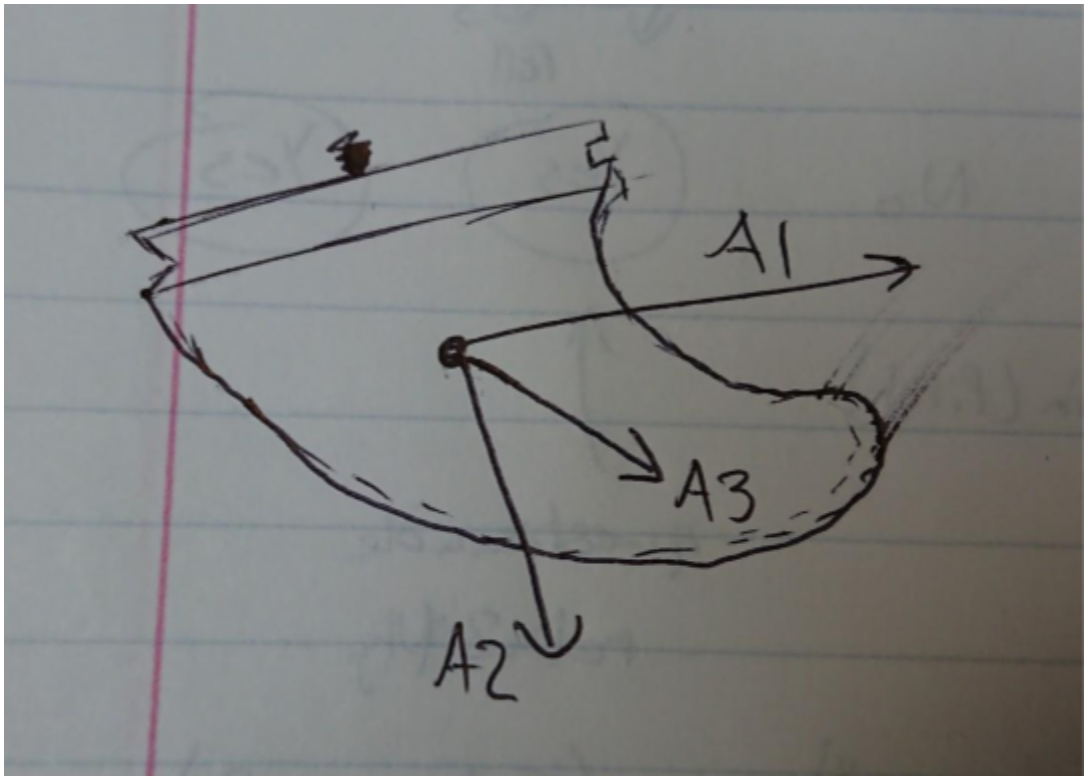
- The internal and external temperatures are read in different ways since one is integrated on the MPU-9250 and the other is an auxiliary.
    - The RAW internal temperature reading is 16 times larger than the actual temperature
    - The RAW external temperature reading is 256 times larger than the actual temperature
- The temperature conversion is automatically done on board and ready to go in the CSV files when exported

Other features to note:

- Polling times
    - When the fin is in the water and NOT detecting surfing motion (ie floating), the IMUs take 5 readings a second (200 ms intervals)
    - When the fin is in the water and detecting surfing motion, the IMUs take 30 readings a second (33ms intervals)
    - Temperature is always measured every 5 seconds, and it takes 750ms to re-calculate and record the temperature, since the fin calculates the temp from the RAW binary for us and is present in the CSV files
        - The RAW internal temperature reading is 16 times larger than the actual temperature
        - The RAW external temperature reading is 256 times larger than the actual temperature
    - The GPS is measured every 6 seconds (going off memory), it records this into the motion CSV on its own accord
        - This makes the interval between times not as accurate when GPS records data, since it'll go in-between rows of what was even intervals. Certain functions in scikit kinematics need an interval time, so when we get to that step we need to filter out the GPS rows from the CSV

Things about Orientation and Quaternions

AXES

- ~~The axes of the board are strange if their orientation of a right-handed triad is to be A1, A2, A3 (not right handed)~~ **RESOLVED: I had labeled A3 wrong to begin with (July 31st)**
- 

    - ~~This could be formatted in this A1, A2, A3 order through firmware or something~~
    - ~~Possible right-handed triad formations would be (A1, A2, -A3), (A1, A3, A2), (A3, A2, A1), (A2, A1, A3)~~
- Translating the axes to the Board's reference frame from the fin's reference frame (as shown in the picture) is very easy for gyroscope and accelerometer data (just multiply -1 when needed through simple analysis
- However, translating the magnetic data is difficult, since the field doesn't register linearly with the sensors
    - The minimum ranges from -20 to 40 bits and the max ranges between 330 and 360
    - If I have a reading of 300 in one direction, rotating the fin exactly 180 degrees does not give -300, but a small positive number
        - This is not easily translatable with just using negative numbers where needed like before.

Quaternions

- So far, we've been using the imus.kalman function in the scikit-kinematics toolkit to calculate orientation
    - It takes in arrays of the accelerometer, gyroscopic, and magnetic IMU readings and outputs a quaternion corresponding with the object's orientation relative to the local magnetic field (we'll get to that in a second)

- Currently, the function's first output quaternion always assumes the value [1, 0, 0, 0], which represent no rotation
    - This is confusing as to how to get it to orient relative to the earth on the first quaternion, as well as align an axis in the simulation with North
    - This is also troublesome as we don't know the position of the simulation relative to the fin, UNLESS we know exactly what the initial position is (such as in Ride 14816)
- This unknown orientation relative to the initial position (and not the local magnetic field) ~~combined with the confusing axes triad that the fin possess~~ makes it very difficult to know the true orientation of the fin (and therefore the board) despite having a working simulation
    - The only VALID way to know is to remember and know the exact initial position and orientation (like in Ride 14816) as well as all of the rotations the fin goes through throughout the course of the session (once again like 14816, and 14840 not being able to track well)

Links to sources and helpful PDFs

Earth's Magnetic Field Orientation - http://srjcstaff.santarosa.edu/~lwillia2/42/42Labmagprelab.pdf

Find Magnetic Field at Your Location - http://www.magnetic-declination.com/

Functions Regarding orientation view class/functions - https://github.com/thomas-haslwanter/scikit-kinematics/blob/master/skinematics/view.py

Robust, Easy to Implement IMU Calibration - https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6907297&tag=1

A Jupyter Notebook Guide on Kalman and Bayesian Filters https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python

Introduction to the Kalman Filter https://towardsdatascience.com/kalman-filter-interview-bdc39f3e6cf3

Introduction to the Extended Kalman Filter https://towardsdatascience.com/extended-kalman-filter-43e52b16757d

Introduction to Unscented Kalman Filter vs Extended Kalman Filter https://towardsdatascience.com/the-unscented-kalman-filter-anything-ekf-can-do-i-can-do-it-better-ce7c773cf88d

GPS and IMU Sensor Fusion using Kalman Filter https://www.stats.ox.ac.uk/~caron/Publications/J_Information_Fusion_2004.pdf

Initial Orientation and Reference Frame Issues (All quat measurements are in reference to the IMU frame, not earth frame) https://github.com/kriswiner/MPU9250/issues/154

Initial calibration and alignment of Low cost Inertial Navigation Units for land vehicle applications http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.457.164&rep=rep1&type=pdf

Determine 3D orientation based on a magnetometer read out https://w01f359.wordpress.com/2016/08/29/determine-3d-orientation-based-on-a-magnetometer-readout/
This is referenced in the paper as well and is a helpful reference for variance in inclination https://cdn-shop.adafruit.com/datasheets/AN203_Compass_Heading_Using_Magnetometers.pdf

Seven Ways to Find Heading (Some using magnetometer)
http://www.navlab.net/Publications/The_Seven_Ways_to_Find_Heading.pdf

Estimation of Heading using Magnetometer and GPS - Manne Henriksson
https://www.diva-portal.org/smash/get/diva2:648760/FULLTEXT01.pdf

Notes in code about the data format and settings of the sensors

Lines 1602 - 1673(MPU9250.cpp):

```
//--------------------------------------------------------
// NOTES FROM THE DOCUMENTATION - DATA FORMAT AND RANGING
//--------------------------------------------------------

// accel data - 3 high endian integers
// as for interpreting the accel data, see pg 34 in the register docs:
// "Each 16-bit accelerometer measurement has a full scale defined in ACCEL_FS (Register 28).
//  For each full scale setting, the accelerometers' sensitivity per LSB in ACCEL_xOUT is:"
// (from the table on page 34 in 9150 manual, pg 14 in 9250 manual, section 4.7)
//   AFS_SEL==0  +/-  2G  16384 LSB/mg      Least Significant Bits per milli-G (1/1000th of a G)
//   AFS_SEL==1  +/-  4G   8192 LSB/mg
//   AFS_SEL==2  +/-  8G   4096 LSB/mg
//   AFS_SEL==3  +/- 16G   2048 LSB/mg

// Currently these are stored in the IMU data record with a bit shift of 2.
// full range is +/- 32k which means that bit shift 2 --> +/- 8k for full range (bits are shifted to get rid of extremely precise, insignificant data) (guess)

// MPU6150::initialize sets AFS_SEL to MPU6050_ACCEL_FS_2 (AFS_SEL == 0)
// this program uses AFS_SEL == 3
```

- **RAW Accelerometer data is 16-bit signed 2's complement number**
- **1 bit for sign(+/-) and 15 bits for values (max binary number being 2^15 = 32,768**
  - **For 2g setting, each bit is 1/16384 of a G**
  - **For 4g setting, each bit is 1/8192 of a G**
  - **For 8g setting, each bit is 1/4096 of a G**
  - **For 16g setting, each bit is 1/2048 of a G**
    - **Each setting is G's divided by the total number of G's the accelerometers are set to measure (this is applied to all axes)**

```
// page 35, IMU temp (high endian integer)
// Temperature in degrees C = (TEMP_OUT Register Value as a signed quantity)/340 + 35

// gyro data - 3 high endian integers (page 36)
// "Each 16-bit gyroscope measurement has a full scale defined in FS_SEL (Register 27). For each full
//  scale setting, the gyroscopes' sensitivity per LSB in GYRO_xOUT is:"
// (from the table on page 36 in 9150 manual, pg 14 in 9250 manual, section 4.6)
//   FS_SEL==0   +/-  250 degrees/s   131.0 LSB/deg/s
```

// FS_SEL==1 +/- 500 degrees/s 65.5 LSB/deg/s
// FS_SEL==2 +/- 1000 degrees/s 32.8 LSB/deg/s
// FS_SEL==3 +/- 2000 degrees/s 16.4 LSB/deg/s
//

- **Same data format for gyro's as accelerometer: 16-bit signed 2's complement number**
- **1 bit for sign(+/-) and 15 bits for values (max binary number being 32,768**
  - **For 250 deg/s, each bit is 1/131 deg/s**
  - **For 500 deg/s, each bit is 1/65.5 deg/s**
  - **For 1000 deg/s, each bit is 1/32.8 deg/s**
  - **For 2000 deg/s, each bit is 1/16.4 deg/s**

// Currently these are stored in the IMU data record with a bit shift of 2.
// full range is +/- 32k which means that bit shift 2 --> +/- 8k for full range **Bits are shifted to get rid of extremely precise, insignificant data(guess)**
//
// MPU6150::initialize sets FS_SEL to MPU6050_GYRO_FS_250 (FS_SEL==0)
// this program currently uses **FS_SEL == 2 by default → Each bit is 1/32.8 deg/s**

// magnetometer data (stored in LITTLE ENDIAN FORMAT) pg 56
//
// magnetometer data is always **+/- 4095, always +/- 1229 for flux density (see section 6.5)**
// there is no scaling - it is always the same scale and range.
//
// (from 9150 manual, section 6.5)
//
// ~~"Measurement data is stored in **two's complement** and Little Endian format. Measurement range of each~~
~~// axis is from -4096 to +4095 in decimal."~~
~~//~~
~~// value hex magnetic flux density~~
~~// 4095 0fff 1229(max)~~
~~// 1 0001 0.3~~
~~// 0 0000 0~~
~~// -1 ffff -0.3~~
~~// -4096 f000 -1229(min)~~
//
// 9250 notes:
//
// On the 9250 this is different. Scale is +/- 32760 in 16-bit mode (+/-8180 in 14-bit mode)
// with full scale being +/- 4192 micro-T (table 4, section 5.6) **Makes sense as long as not in 2's complement**
//

```
//  value    hex   magnetic flux density
//  32760   7ff8      4192(max) ← Pretty sure this is in micro-Teslas
//      1   0001         0.15 ← Each bit is 0.15 micro-Teslas
//      0   0000         0
//     -1   ffff        -0.15
// -32660   8008     -4192(min)
```

- **IMPORTANT!!! → Earth's magnetic field varies between 25 and 65 microteslas at earth's surface (+/- 166.67 - 433.33 bits)**
- **Max reading from jupyter file was 363 → 363 * 0.15 = 54.45 uT, which is within the range of Earth's magnetic field**
- **Still have to figure out whether a positive values points North or South.**

```
// 'GetMotion6()' - like the legacy 3rd party provided class, 'GetMotion6()' gets accelerometer and gyro data only
//                  it can be called separately if you don't need magnetometer data, as with sleep/wake detect
```

Lines 801-803 in MPU9250.cpp:

```
short MPU9250::aYVal[TAP_ARRAY_SIZE]; // last n 'y' axis values
short MPU9250::aZVal[TAP_ARRAY_SIZE]; // last n 'z' axis values
short MPU9250::sLastXVal;
```

#ifdef VERTICAL_ONLY_TAP **(Lines 872 - 911)**

```
  nTemp = sLastXVal;
  nYAmp = aYVal[(nYTail - 1) & TAP_ARRAY_MASK];
  nZAmp = aZVal[(nZTail - 1) & TAP_ARRAY_MASK];

  // what we're looking for is a vertical alignment.  A right-handed fin should
  // have a negative X, and the ratio of X to Y and X to Z should be at least 0.466 (tangent of 25 deg)
  // for now we'll assume 2:1 (because it's a simple calculation) and compare that way

  if(nTemp < 0) // TODO: see if I'm a left fin or right fin, and only allow correct sign on X
  {
    nTemp = -nTemp; // for now
  }

  if(nYAmp < 0)
```

```
  {
    nYAmp = -nYAmp;
  }

  if(nZAmp < 0)
  {
    nZAmp = -nZAmp;
  }

  if(nTemp < (nYAmp << 1) || nTemp < (nZAmp << 1)) // orientation is wrong
  {
    nIndex = 0;

    if(bTapLEDFlashFlag) // this could happen, maybe?
    {
      cue_led_flash(0, TAP_DETECT_LED_BLINK_TIME); // CANCEL!  (wrong orientation, all
LEDs off, blink grace time follows)
//      led_reset(); // turn them all off

      nIndex = -1; // to indicate "tap canceled" (i.e. no 'rattling' allowed)
    }

    reset_tap_data(); // make sure this happens here

    return nIndex; // return 0 if "no tap", or -1 if "tap sequence canceled"
  }

Line 1085:
  // PHASE II:
  // calculate the average reading, by summing up all of the data points
  // and dividing by TAP_ARRAY_SIZE.  I will use the average to see what
  // the 'peaks' are, and if they are the right shape.


Line 1391:
// [9150/9250 notes]
  // it seems (from the docs) though it might not actually be the case...
  // the motion and zero motion thresholds are on the LSB.  with a scale of +/- 8G this
corresponds to about
  // 8/128 or 1/16 G.  If this triggers too often (like during travel or when floating) the HPF can be
  // configured (above) for a different value than 5hz.
```

- Set +/- 8G acceleration (max range)

Line 1515:
// see issue #319 for some data captures showing accel+gyro readings for floating around and surfing.
  // as well as walking/transporting. There's not a lot of difference!
  // More importantly, there seems to be more of a problem with the recording turning OFF, rather than
  // detecting motion. To accomodate that, the gyro motion detect is now 4 times as sensitive.

FROM HEADER FILE (Smartphin_Firmware.h)

#define MPU6050_GYRO_RANGE MPU6050_GYRO_FS_1000 /* +/- 1000 degrees per second */
#define MPU6050_ACCEL_RANGE MPU6050_ACCEL_FS_16 /* +/- 16G - set to MPU6050_ACCEL_FS_8 for +/- 8G */

#define SCALE_GYRO_DATA 2 /* number of bits to shift right for gyro data (less than that for accel) */

#define SCALE_ACCEL_DATA 2 /* number of bits to shift right for accel data */

#define Y_AXIS_GRAVITY_THRESHOLD 350 /* threshold for checking ~1G accel in Y axis, i.e. fin in 'surfing position' */

// a value of 350 is about 0.7G or a 45 degree angle with fin upwards

- **350 *4 (shift over by 2 bits) = 1400 → 1400 bits * (1/2048) G's = 0.6836 G's ~ 0.7G's**

#else // MPU6050_ACCEL_RANGE == MPU6050_ACCEL_FS_8 - normally not used, as 16G is needed for surf capture

POLLING TIMES WHEN IN WATER → Frequency of measurements

- Header line 486:

#define IMU_DEFAULT_POLL_DELAY 33 /* min time between IMU measurements when surfing. was 11, now 33 --> approximately 30 readings per second */ **(33 milliseconds)**

- Header line 487:

#define IMU_DEFAULT_POLL_DELAY2 200 /* min time between IMU measurements when floating. 200 --> approximately 5 readings per second */ (**200 milliseconds)**

#define IMU_MIN_MOTION_TIME 15 /* min time to capture high speed IMU data following surf motion (in seconds) */ **(15 seconds) (not sure if this is "fastest it can do it in" or "minimum time of surfing motion to start capturing high speed IMU data")**
#define IMU_MAX_NO_MOTION 600 /* max 'no motion' time for IMU data without surf motion (in seconds) (was 300) */ **(10 minutes)**
#define IMU_MAX_NO_CAPTURE 300000 /* up to 5 minutes of 'no capture' following 5 minutes no surf motion (milliseconds) */

#define TEMP_MEASUREMENT_POLL 5 /* Temperature 'poll' period in seconds - the total of POLL plus CONVERT = TEMPERATURE PERIOD */
#define TEMP_MEASUREMENT_CONVERT 1000 /* conversion time in milliseconds. See table 2 in DS18B20 docs for 'why'. 12 bit --> 750 msecs */

STRUCTURE OF DATA DEFINITIONS

```
// DELTA IMU data still uses 'IMU_DATA' struct to get info from the IMU,
// but stores that as the 'last collected data' instead.  if the delta
// differs too much, I store 2 records ('I' and 'i'), else one record ('P')
// 'P' being 'Partial' kinda like video data, where I got the idea.
// every 'n' records will store 2 records anyway, also like video data.
typedef struct _imu_data_
{
  int16_t a[3], g[3], m[3];       // high endian 9 axis data (18 bytes)
  uint32_t imu_sample_time;       // high endian sample time (4 bytes)
} IMU_DATA;

// this structure can be used to more efficiently do things
// with the IMU data samples by treating all 3 arrays as a single array
typedef struct _imu_data_v_
{
  int16_t v[9];                   // high endian 9 axis data (18 bytes)
} __PACKED__ IMU_DATA_V;


typedef struct __temperature__
{
  unsigned char bType;            // 'T'
  int16_t temp_1, temp_2;         // high-endian temperatures, deg C * 16
  uint32_t sample_time;           // high endian sample time (4 bytes)
  int16_t imu_temp;               // IMU temperature (from cached value) (RAW format is 16 times larger than actual Centigrade temp, verified with jupyter notebook)
                                  // deg C = imu_temp * 340 + 35
  //  add MAX31725 temperature (after this no more room)
  int16_t ext_temp;               // deg C = ext_temp / 256 (signed) (external temp RAW format is 256 times larger than actual Centigrade temp, verified with jupyter notebook)

} __PACKED__ TEMPERATURE;
```

- **Temperature changes of larger than 5 degrees between readings are also invalidated**

Line 296 of temperature.cpp

```
  // temperature readings must be within 5 degrees
  // and between the values of 0C and ~38C (32F and 100F (sanity test)
  // the reading for 38C is 608 (16 * deg C)
  // a delta of 5 degrees is 80
```

```
typedef struct __salinity__
{
  unsigned char bType;            // 'S'
  uint32_t dwSalinity;            // high-endian salinity reading (in time)
  uint32_t sample_time;           // high endian sample time (4 bytes)
  uint16_t wCalData1, wCalData2,
          wCalData3;              // salinity calibration data

  unsigned char bInTheWater;      // new element, 'in the water' indication

} __PACKED__ SALINITY;

typedef struct __phdata__
{
  unsigned char bType;            // 'H' (because 'P' is already in use)
  uint32_t dwPH;                  // high-endian scaled pH reading (1000000L * value)
  uint32_t sample_time;           // high endian sample time (4 bytes)
  uint16_t wCalData1, wCalData2,
          wCalData3;              // reserved - pH calibration data

} __PACKED__ PHDATA;
```

MOTION THRESHOLD DEFINITIONS (header line 259)

```
#define MOTION_THRESHOLD (192 << SCALE_ACCEL_DATA)  /* motion threshold value
(while recording) adjusted for MPU_ACCEL_RANGE */
                                    /* the 192 value is based on experimentation and should be
validated */
                                    /* NOTE:  this is for a DELTA in acceleration/gyro data, not the
actual value */
                                    /*       a value of 256 is approximately 1G on this scale */

#define TAP_MOTION_THRESHOLD 76 /* was 72, 80 */    /* the accel motion threshold for 'tap'
detect - also experimentally determined */
                                    // NOTE:  TAP_MOTION_THRESHOLD is not affected by bit
shift or scale
// other thresholds for tap and motion detect
#define WAKE_ON_MOTION_THRESHOLD 41    /* wake on motion threshold for interrupt, IMU
reg 31 - was a value of 63, then 31 */
```

```
#define SCALE_GYRO_MOTION_THRESHOLD 4  /* scale factor for gyro vs accelerometer motion */
#define SURF_MOTION_THRESHOLD 80/*5*/  /* motion threshold for surf motion on moving average gyro data, now 16 times the previous value */
```

Lines 1301-1309

```
// NOTE:  9250 uses 'best clock' (a feature the 9150 didn't have)

  // NOTE:  bits 2:0 of MPU6050_GYRO_RANGE need to be zero to use DLPF setting from config register 26 on the 9250
  //        this will be the case ALWAYS for the current forseeable future (see sections 4.5 and 4.6 in 9250 reg manual)

  // set desired gyro and accelerometer ranges.  Always assign 0 for the low 3 bits of GYRO/ACCEL range for 9250
  // the configuration range is assigned for the GYRO filter using reg 26, for the ACCEL using reg 29

  // simplification of config for 9250 - gyro range 'FS_SEL' and accel range 'AFS_SEL'
```

Lines 1556-1563

```
// analyze moving average'd data, to see if I'm surfing or floating

//  if(abs(mavg_ag[3]) >= system_vars.wSurfMotionThreshold ||
//     abs(mavg_ag[4]) >= system_vars.wSurfMotionThreshold ||
//     abs(mavg_ag[5]) >= system_vars.wSurfMotionThreshold)
  if(my_do_mavg_threshold_compare(mavg_ag[3]) ||
     my_do_mavg_threshold_compare(mavg_ag[4]) ||
     my_do_mavg_threshold_compare(mavg_ag[5]))
```