Analyzing Loudspeaker Cabinet Vibrations with a Digital Accelerometer and an Arduino UNO

Thurman Gillespy

RETOLD BY WATTY PIPER ILLUSTRATED BY GEORGE AND DORIS HAUMAN

The Little Engine That Could®

Physics 221, 222, 223

‘Problem Based Learning’ Labs

Student initiated physics experiment

Develop a low cost apparatus and method of measuring loudspeaker cabinet vibrations


KRIX

Speaker acoustic output

- drivers - woofer, midrange, tweeter
- acoustic output from rear of drivers and driver mechanical motion puts energy into the cabinet
- re-radiated through the woofer cone
- from all sides of the enclosure
- all combine to adversely affect the audio quality

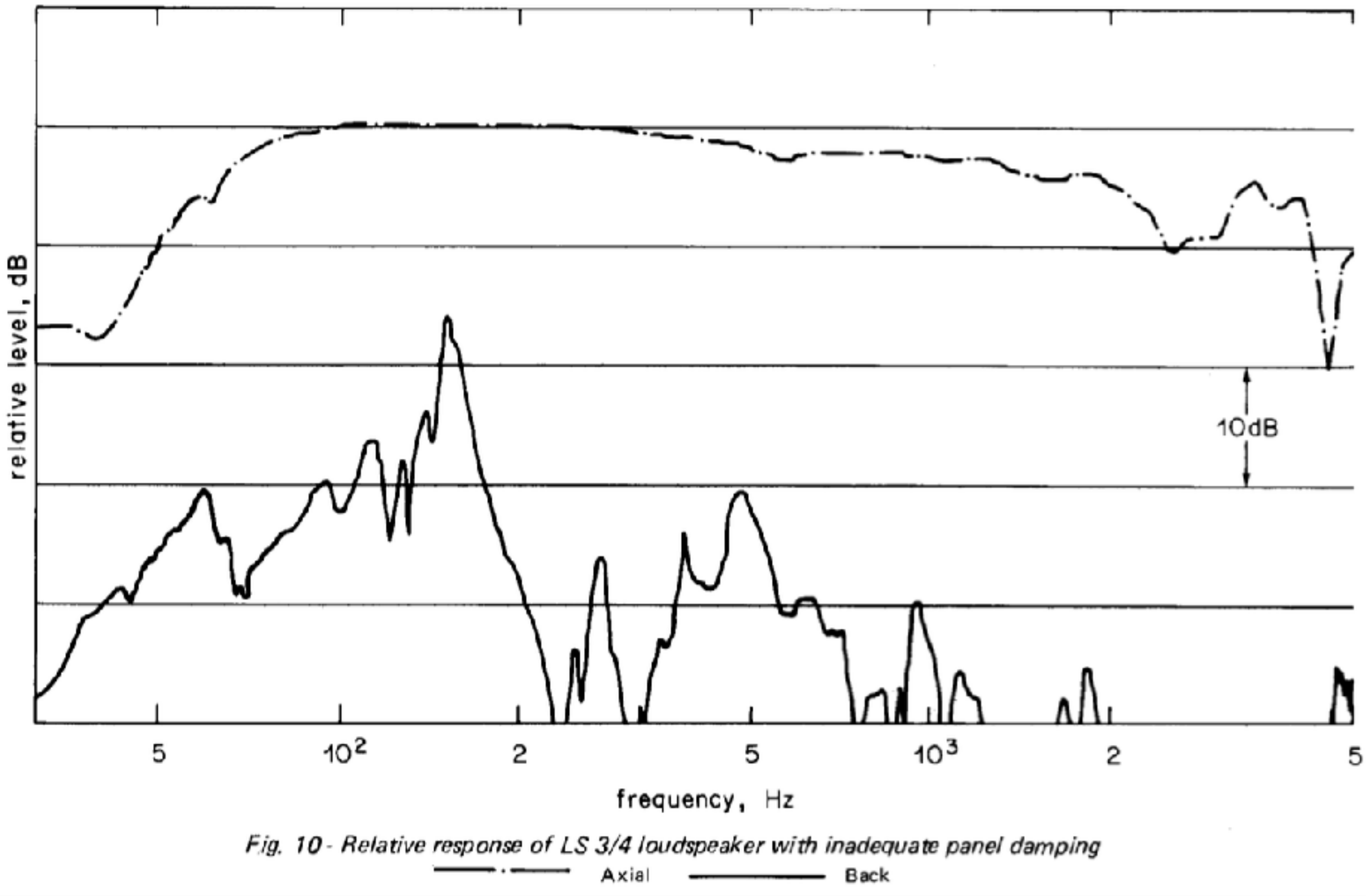


Fig. 10 - Relative response of LS 3/4 loudspeaker with inadequate panel damping

Axial — Back

Improve speaker audio quality by decreasing cabinet vibrations and resonances

Cabinet resonance reduction

- material selection
  - plywood, MDF (medium density fiberboard)
- internal bracing
- internal damping

Statement of the problem

- cabinet vibrations known to adversely affect audio quality of stereo loudspeakers
- many different techniques and commercial products recommended for reducing cabinet vibrations
- very little data to support claims
- no testing methods in wide use by audio amateurs for analyzing cabinet vibrations

System

- Arduino UNO
- digital accelerometer
- test loudspeaker
- power amplifier

Arduino UNO

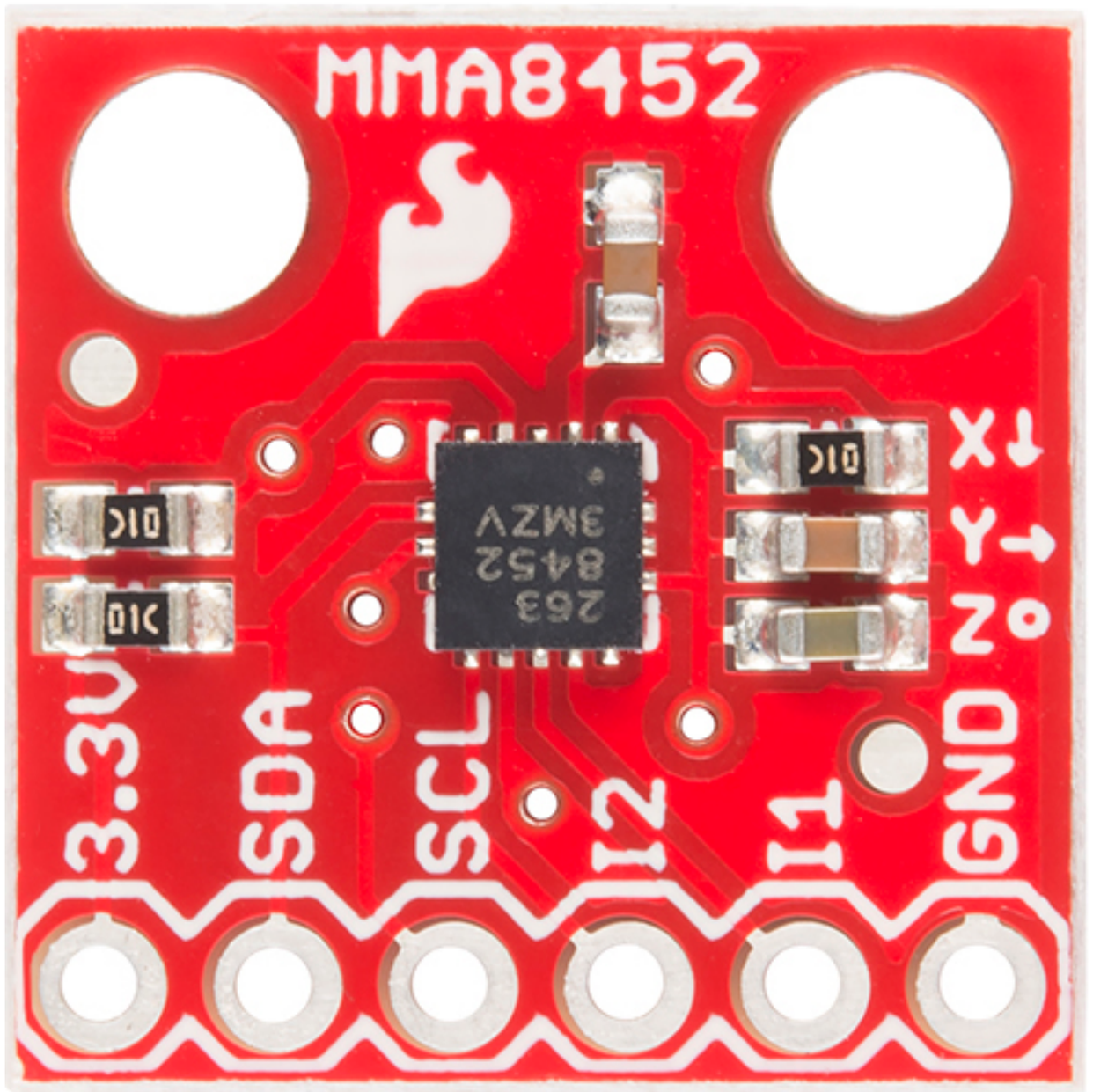
- Seattle Robotics kit
- 8-bit microprocessor, ATmega328P
- 5 volt
- 32k flash memory
  - program, libraries
- 2048 bytes SRAM
  - variables
- 1024 bytes EEPROM

MMA8254Q

- Freescale Semiconductor
- 3 x 3 mm, 16 pin, QFN package
- 1.95 V to 3.6 V supply voltage
- ±2g/±4g/±8g dynamically selectable full-scale
- Output Data Rates (ODR) from 1.56 Hz to 800 Hz
- 12-bit and 8-bit digital output
- I^2C digital interface

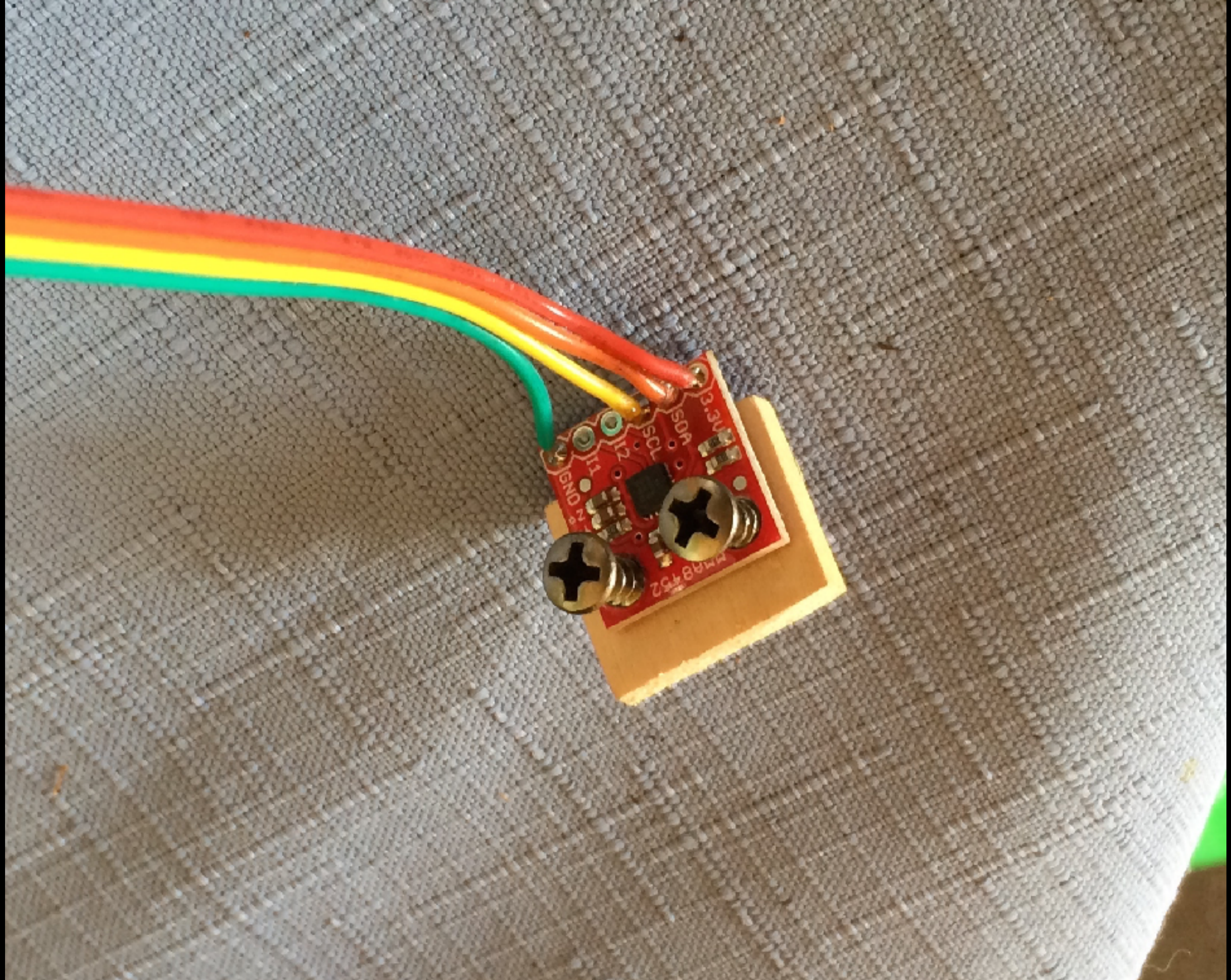
MMA8254Q

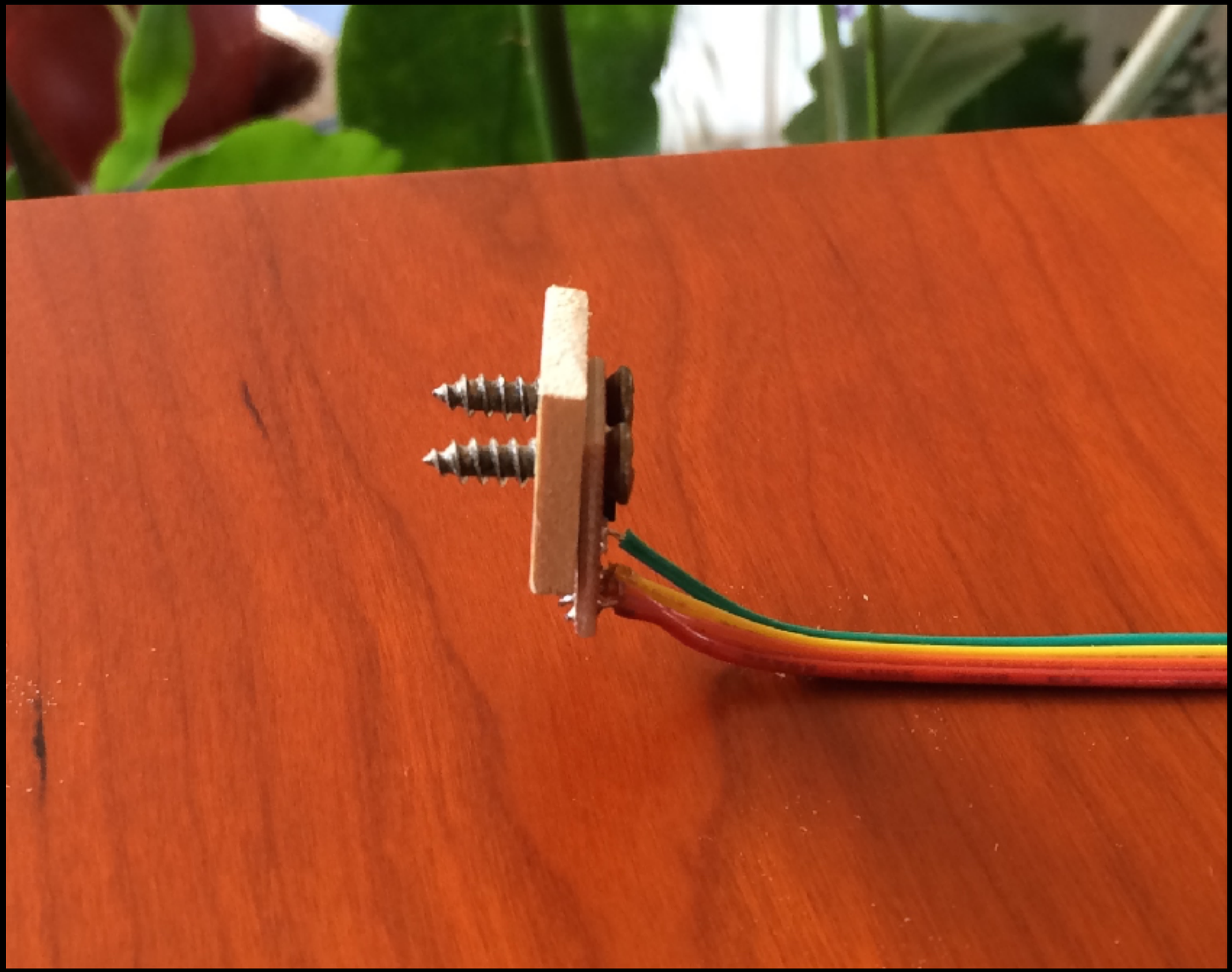
- 2 programmable interrupt pins
    - 6 interrupt sources
- 3 embedded channels of motion detection
- Portrait/Landscape orientation detection
- real time high pass filter data
- current consumption: 6 μA – 165 μA

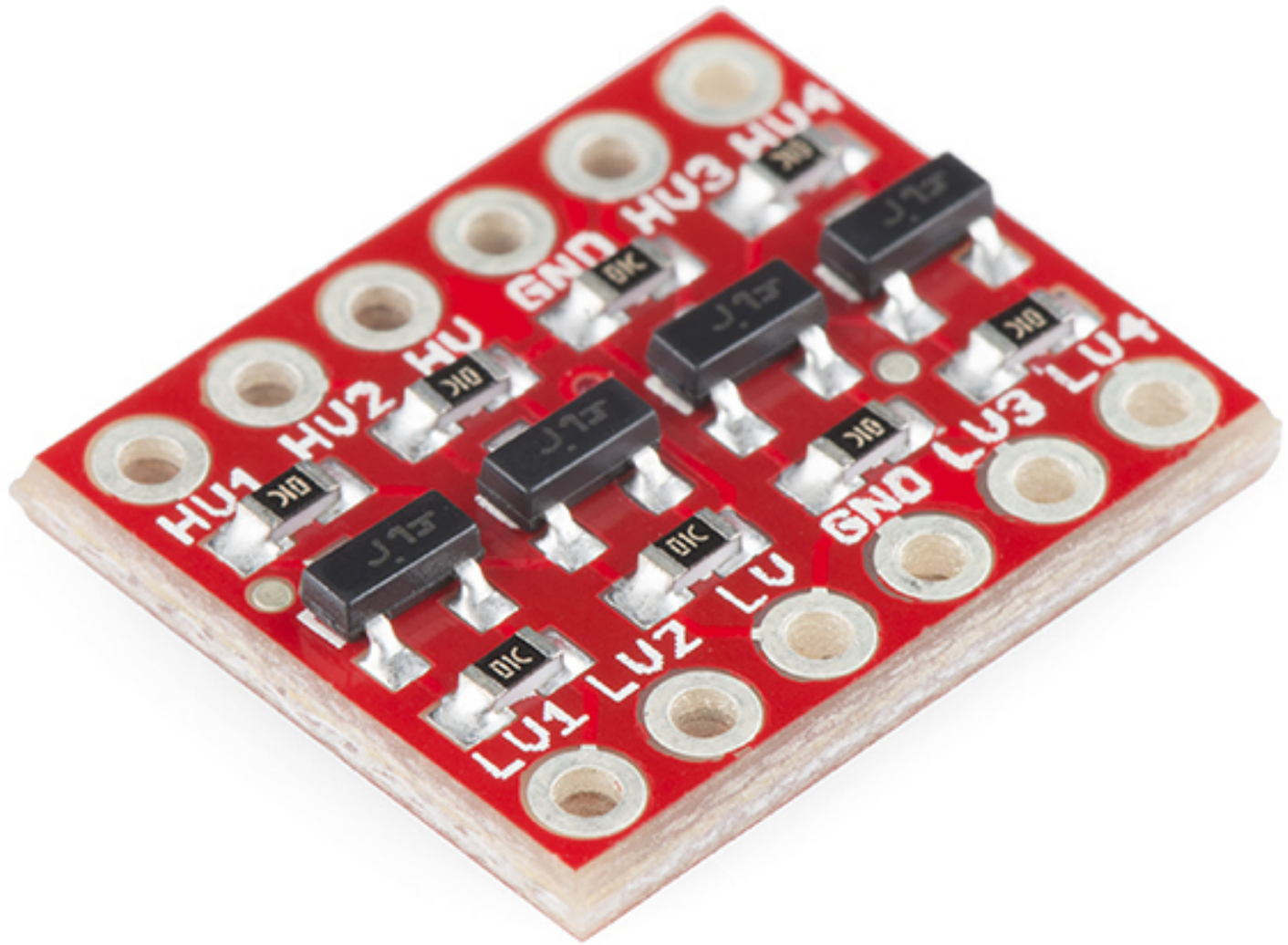

MMA8452
10K
10K
10K
263
8452
3MZV
X
Y
Z
3.3V
SDA
SCL
I2
I1
GND

UNO - MMA8254 communication

- I²C
- 4.7k resistors on I²C SCL and SDA lines
- SparkFun Bi-Directional Logic Level Converter
  - 5 volt - 3.3 volt
- 12 inch ribbon cable


3.3V
SDA
SCL
GND



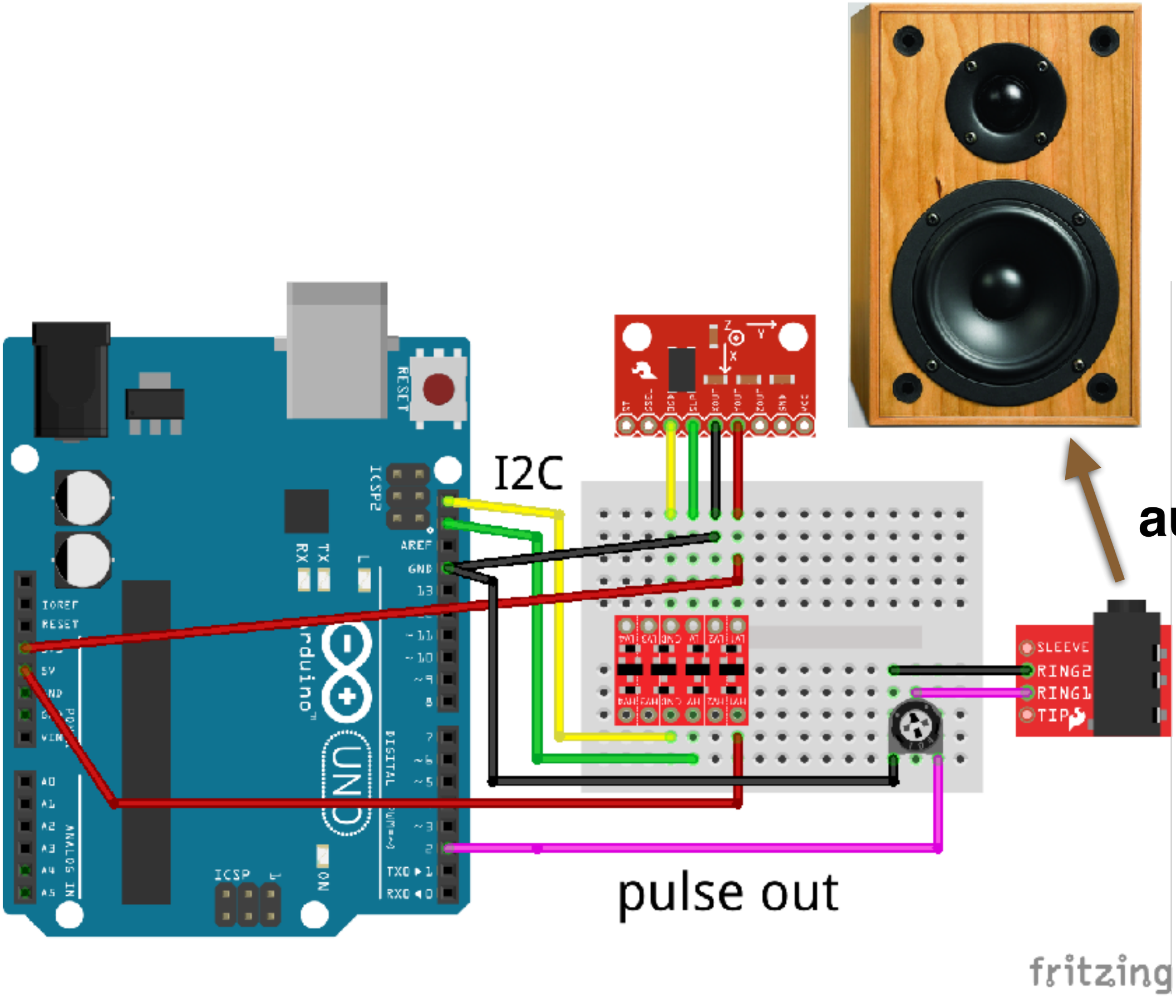

HU1
HU2
HU
GND
HU3
HU4
LU1
LU2
LU
GND
LU3
LU4
10K
6K
J13

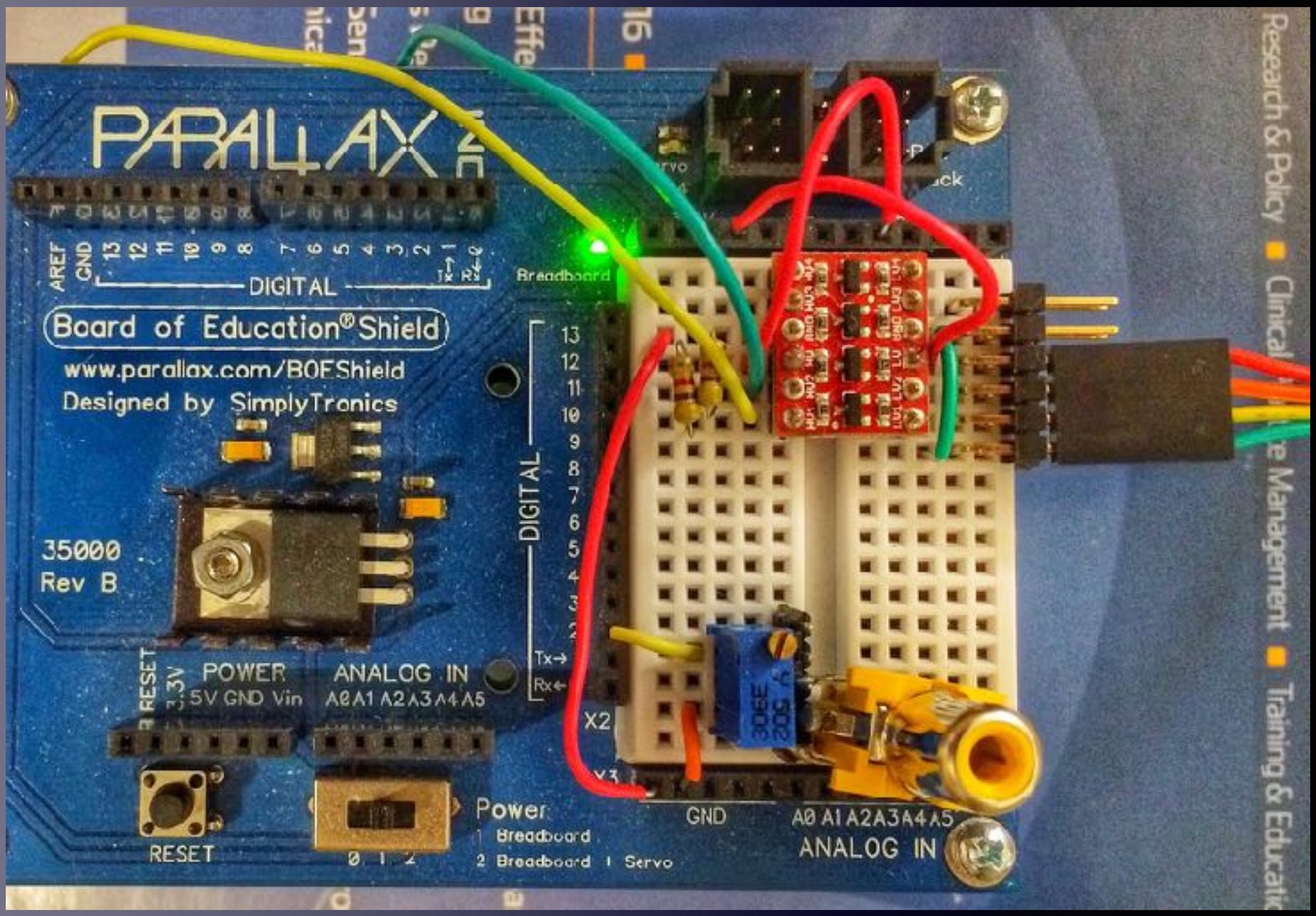
Test speakers

- (2) ELT524M audio loudspeakers
- 2-way base reflex with 5” woofer and 1” tweeter
- 6.12” W x11.25” H x 9.81” D
- all internal sound treatment removed

System operation

- Arduino UNO outputs square wave
- 20k potentiometer to power amp and speaker
- MMA8254Q attached to speaker cabinet
- I^2C connection to Arduino
- records and processes accelerometer output


I2C
audio
pulse out
RESET
ICSP2
AREF
GND
13
IOREF
RESET
3V3
5V
GND
Vin
A0
A1
A2
A3
A4
A5
ANALOG IN
DIGITAL (PWM=~)
TX
RX
L
ON
ICSP
TXD 1
RXD 0
SLEEVE
RING2
RING1
TIP
fritzing


PARALLAX
AREF GND 13 12 11 10 9 8 7 6 5 4 3 2 1 0
DIGITAL
Board of Education® Shield
www.parallax.com/BOEShield
Designed by SimplyTronics
35000
Rev B
RESET 3.3V 5V GND Vin
POWER
ANALOG IN A0 A1 A2 A3 A4 A5
RESET
Breadboard
13 12 11 10 9 8 7 6 5 4
DIGITAL
Tx→
Rx←
X2
GND
A0 A1 A2 A3 A4 A5
ANALOG IN
Power
1 Breadboard
2 Breadboard + Servo
0 1 2
Research & Policy
Clinical
Management
Training & Educatio

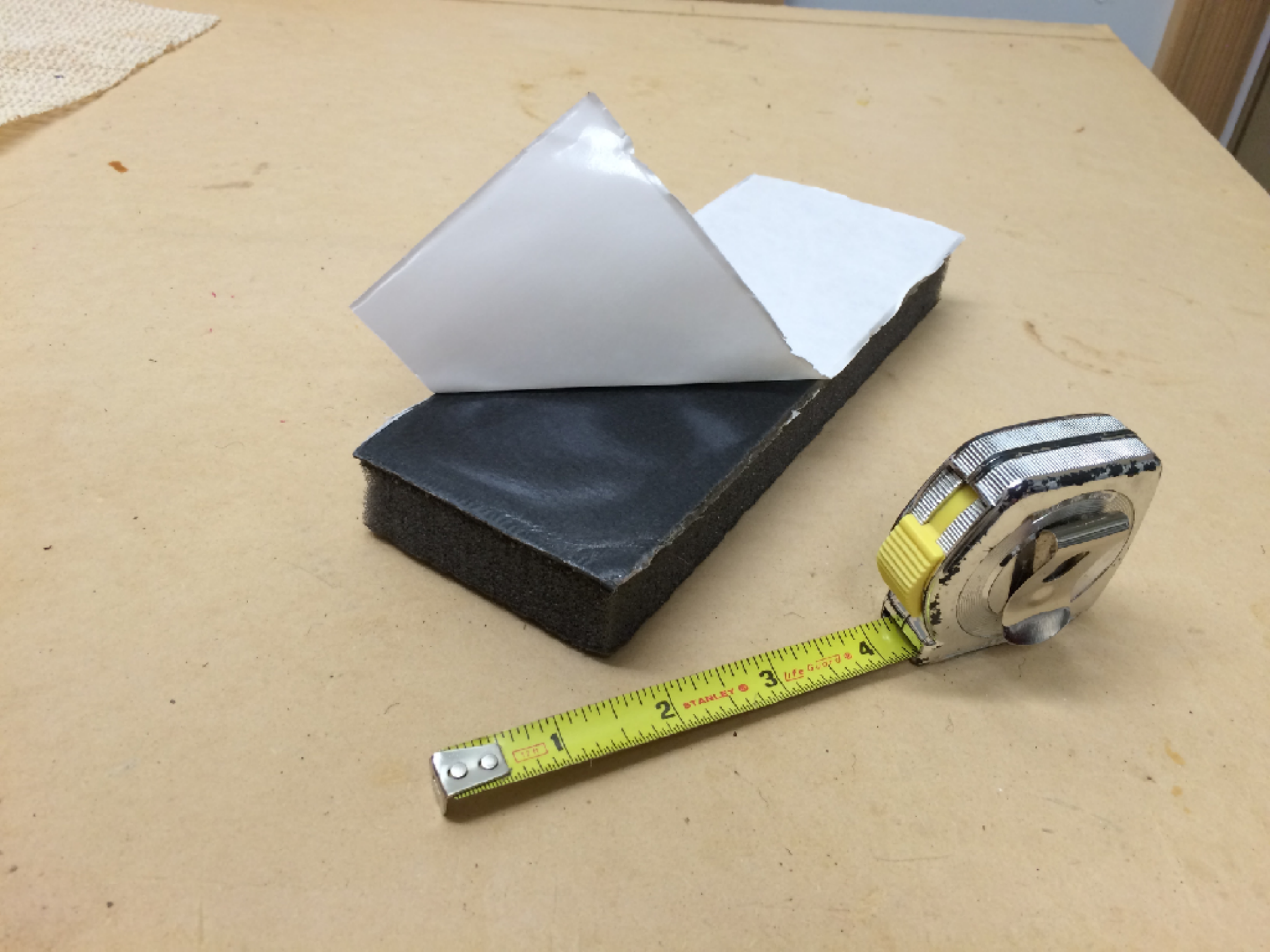

X
Y
Z
MMA8452
3.3V
SDA
SCL
I2
I1
GND

Damping material widely used to reduce speaker cabinet vibrations and resonances

No Rez

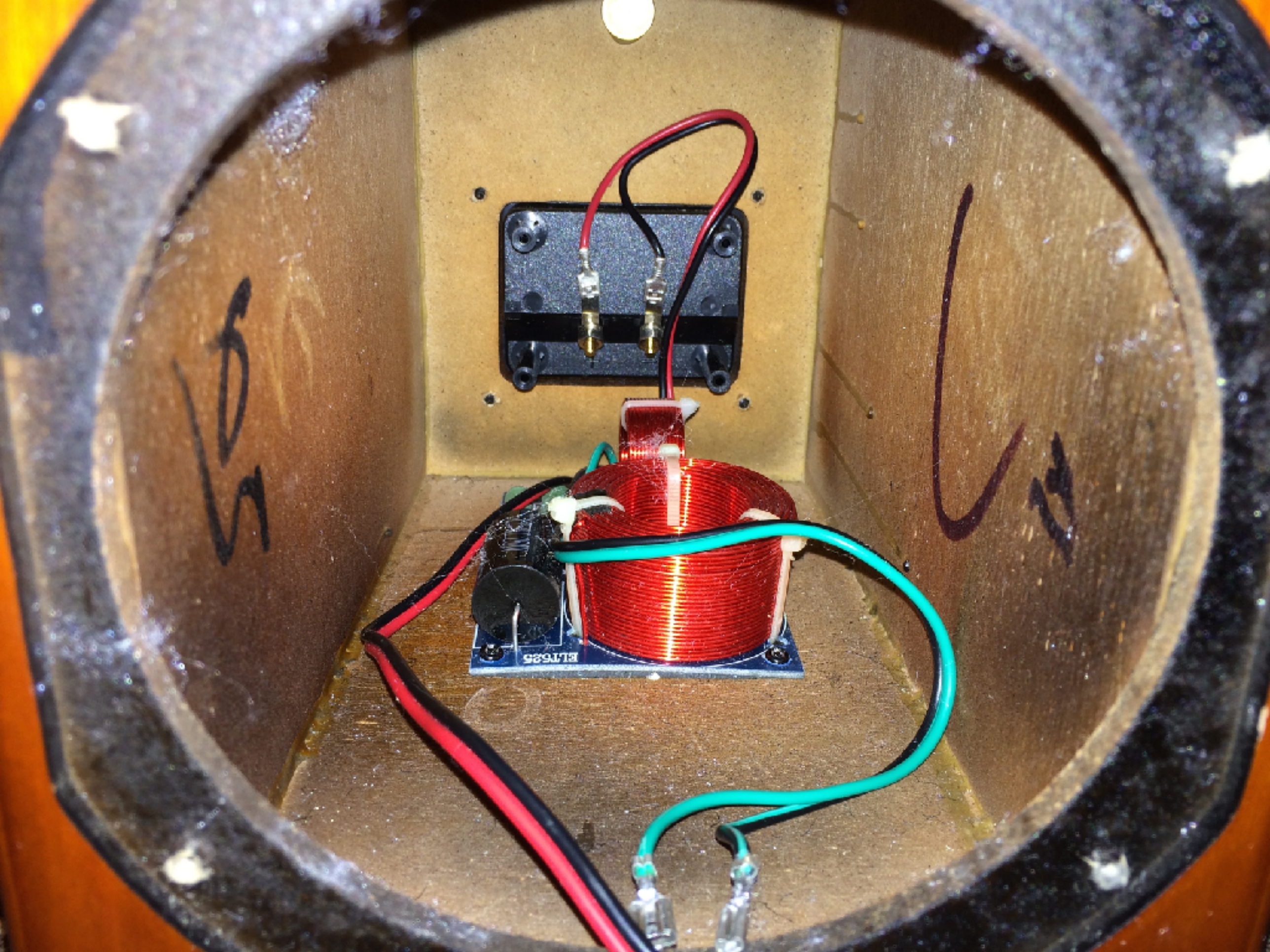
- commercial product for reducing speaker cabinet vibrations
- “damping material developed specifically for loudspeaker applications, [especially for] damping and eliminating enclosure resonances.”
- “heavy damping layer”
- “open cell foam” [to] “absorb standing waves and internal reflections"

No Rez


STANLEY
1
2
3
4

1st question: Does No Rez reduce cabinet vibrations?


RABK-O
RICUTE
66
Road Trip Game
AMERICAN TRIVIA
BOARD
Textbook of Anatomy
ARTHROGRAPHY


ELT625

Method

- 200 msec 200 Hz square wave (Arduino)
- measure at 3 different locations on the left side of both cabinets
  - with and without No Rez
- 15 samples per location, 4 sec delay between samples


3
2
1

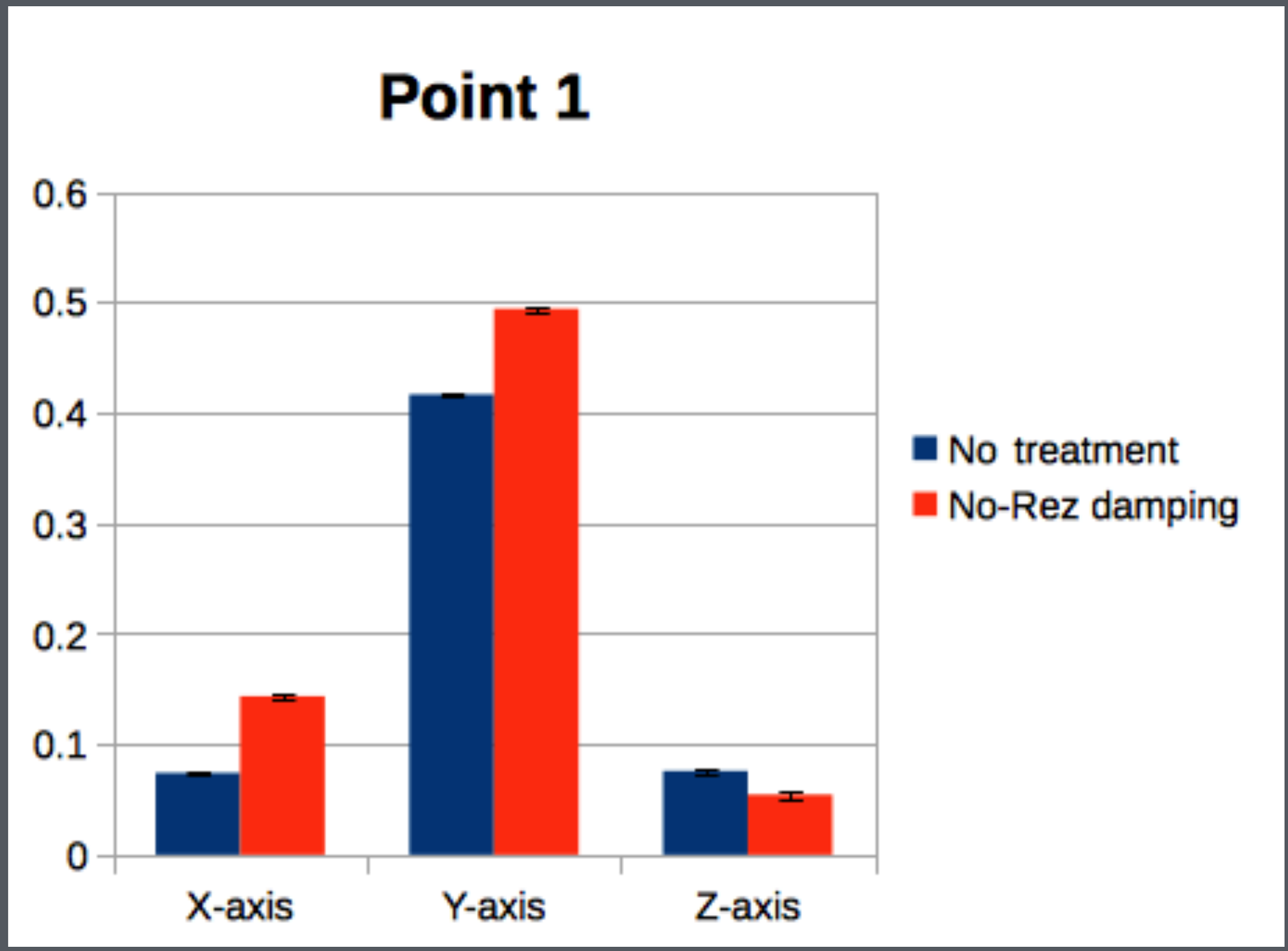
Data Analysis

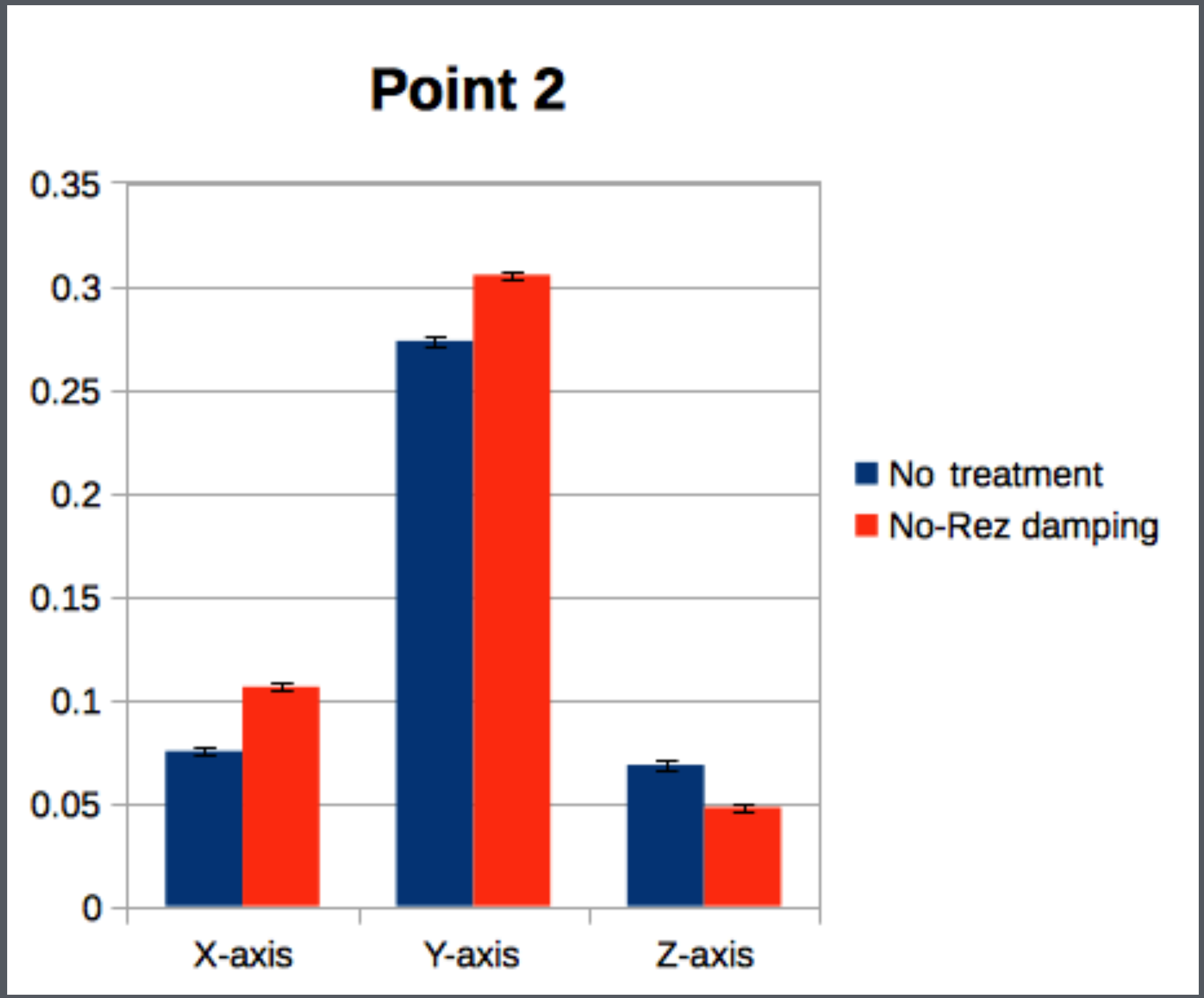
- RMS (root mean square) for x, y and z-axis calculated on Arduino (15 samples)
- data exported to iMac over USB serial port
- mean, standard deviation and variance calculated on iMac custom C++ program
- data plotted in Libre Office

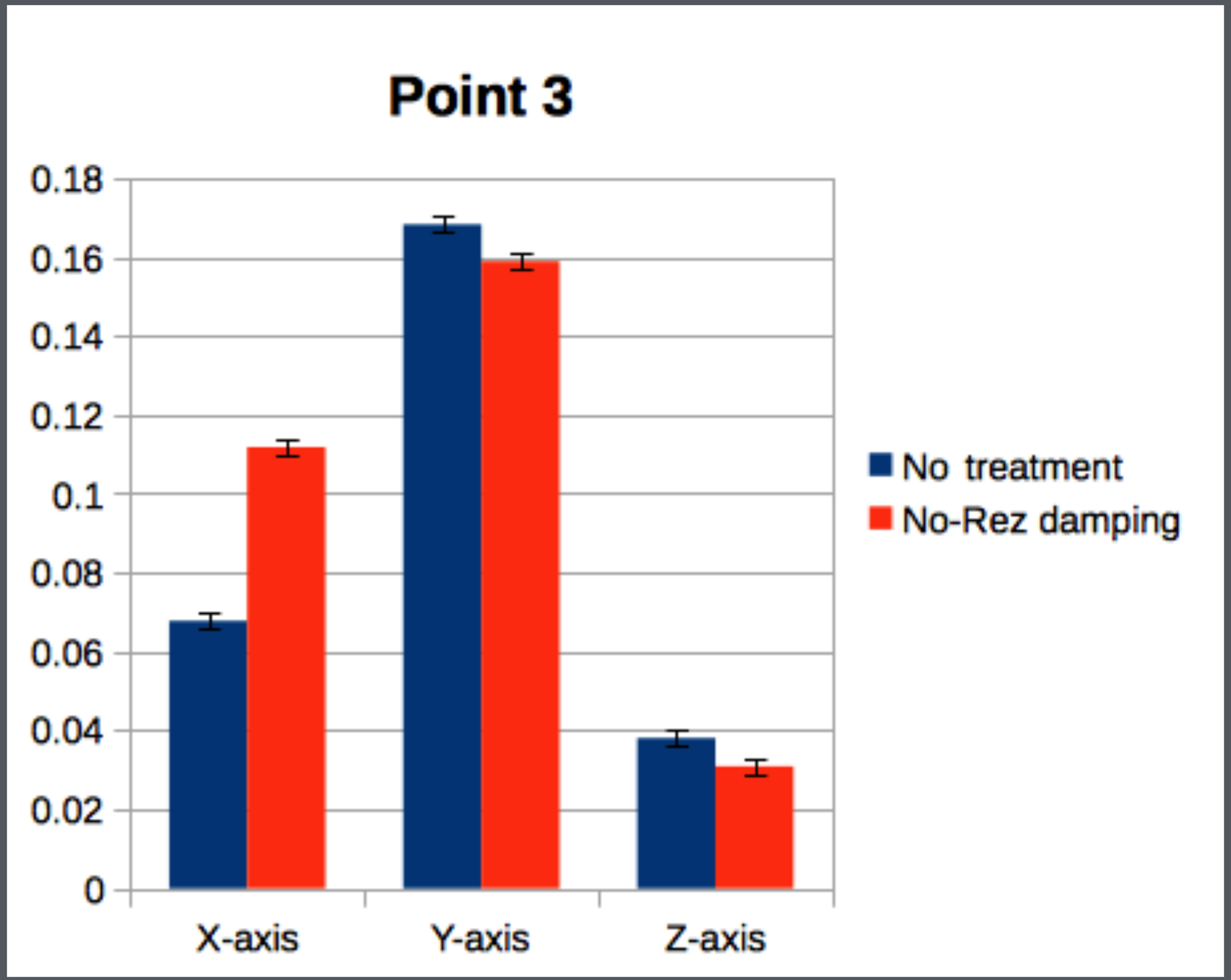
Box A Point 1	mean RMS	SD
X-axis	0.073833	0.001245
Y-axis	0.416484	0.002808
Z-axis	0.075614	0.001764

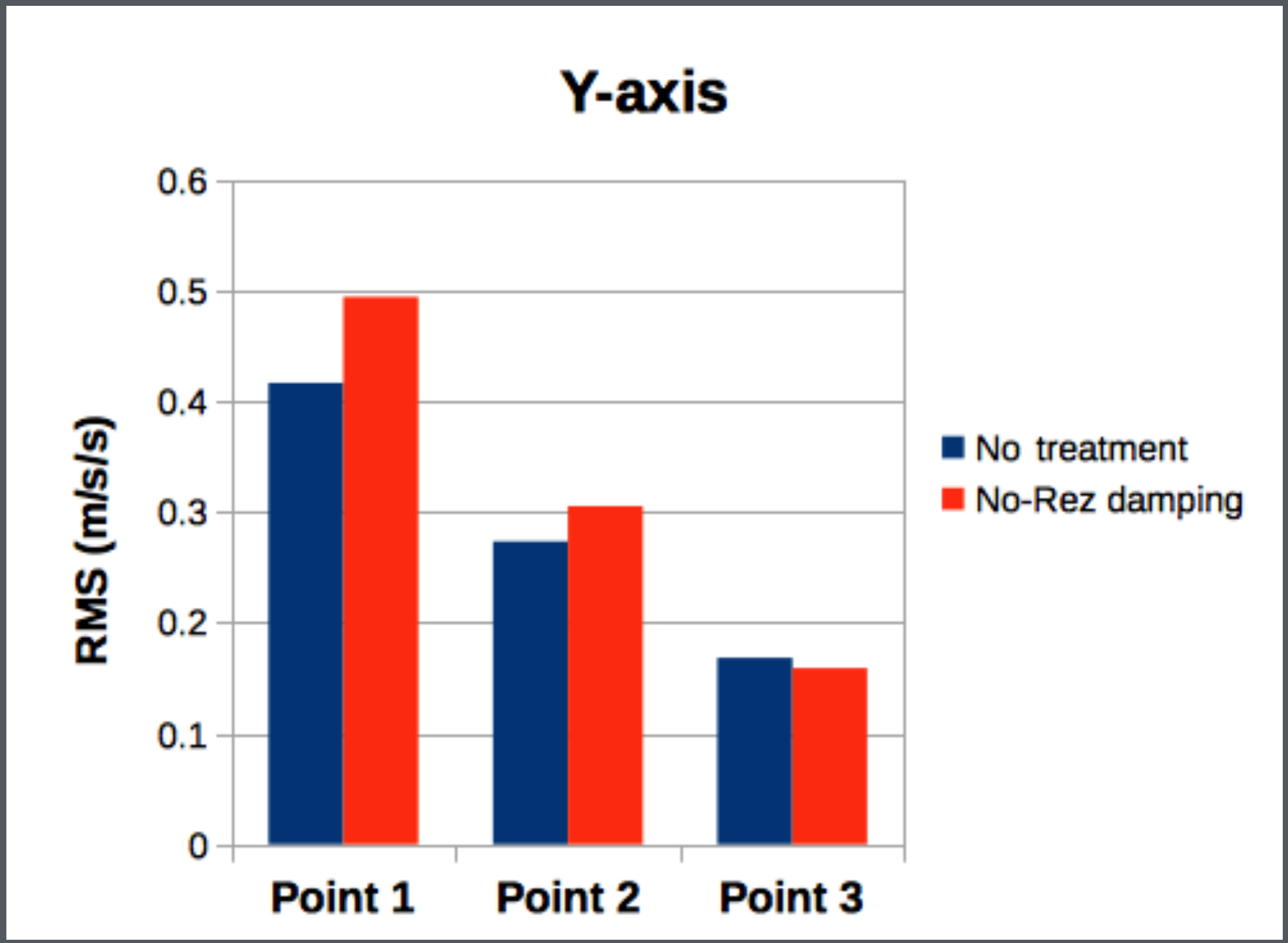
N = 15


X
Y
Z
MMA8452
3.3V
SDA
SCL
I2
I1
GND


Point 1
0.6
0.5
0.4
0.3
0.2
0.1
0
X-axis
Y-axis
Z-axis
No treatment
No-Rez damping


Point 2
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
X-axis
Y-axis
Z-axis
No treatment
No-Rez damping


Point 3
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
X-axis
Y-axis
Z-axis
No treatment
No-Rez damping


Y-axis
RMS (m/s/s)
0.6
0.5
0.4
0.3
0.2
0.1
0
Point 1
Point 2
Point 3
No treatment
No-Rez damping

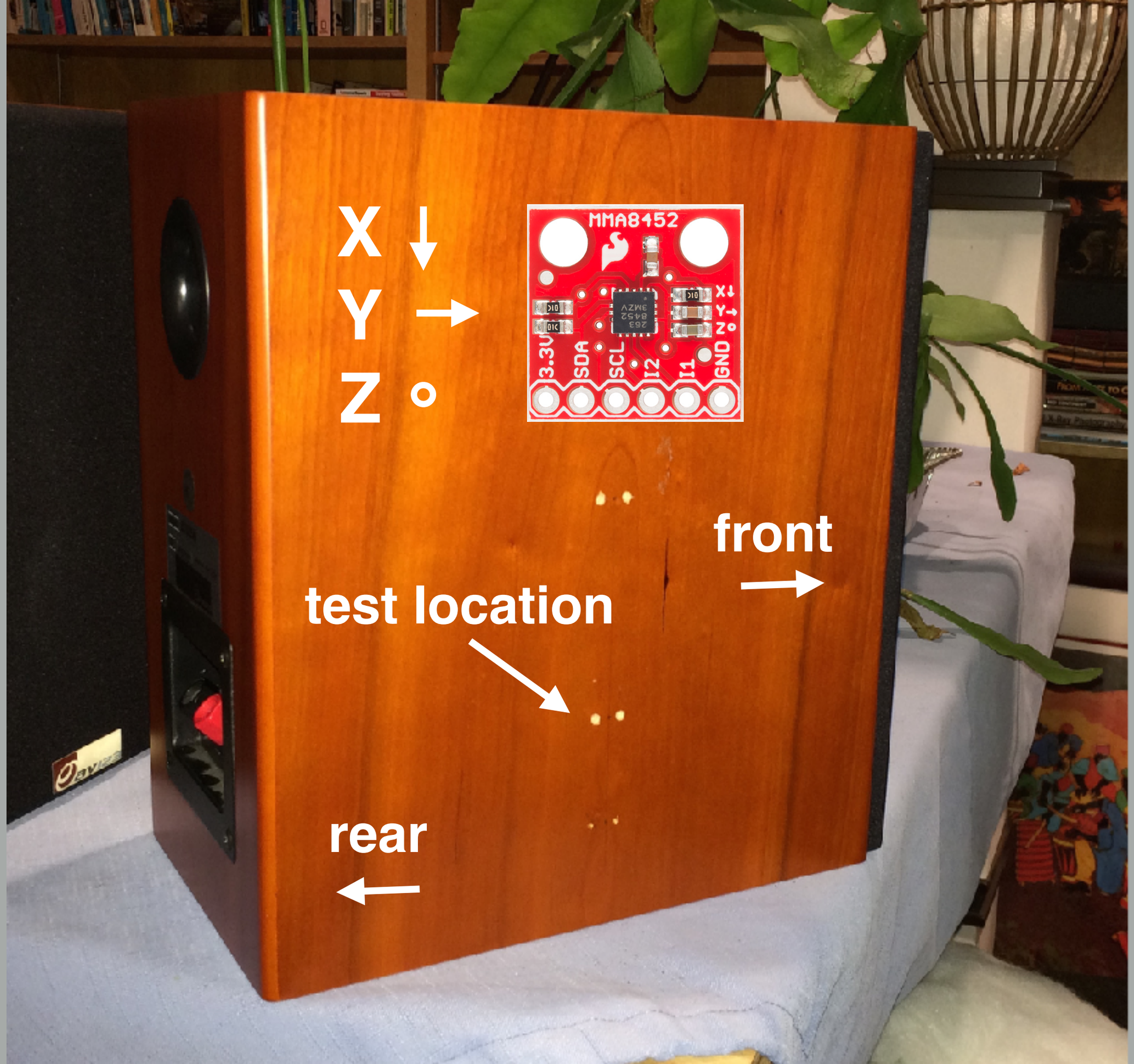
Conclusions

- successful use of 8-bit micro-controller to measure speaker cabinet vibrations with digital accelerometer
- tightly clustered data sets with low standard deviations
- Y-axis dominant vibration
  - direct effect of woofer vibration in Y-plane
- results opposite predicted
  - the treated cabinet showed larger vibrations

2nd question: How does cabinet base support affect speaker vibrations?

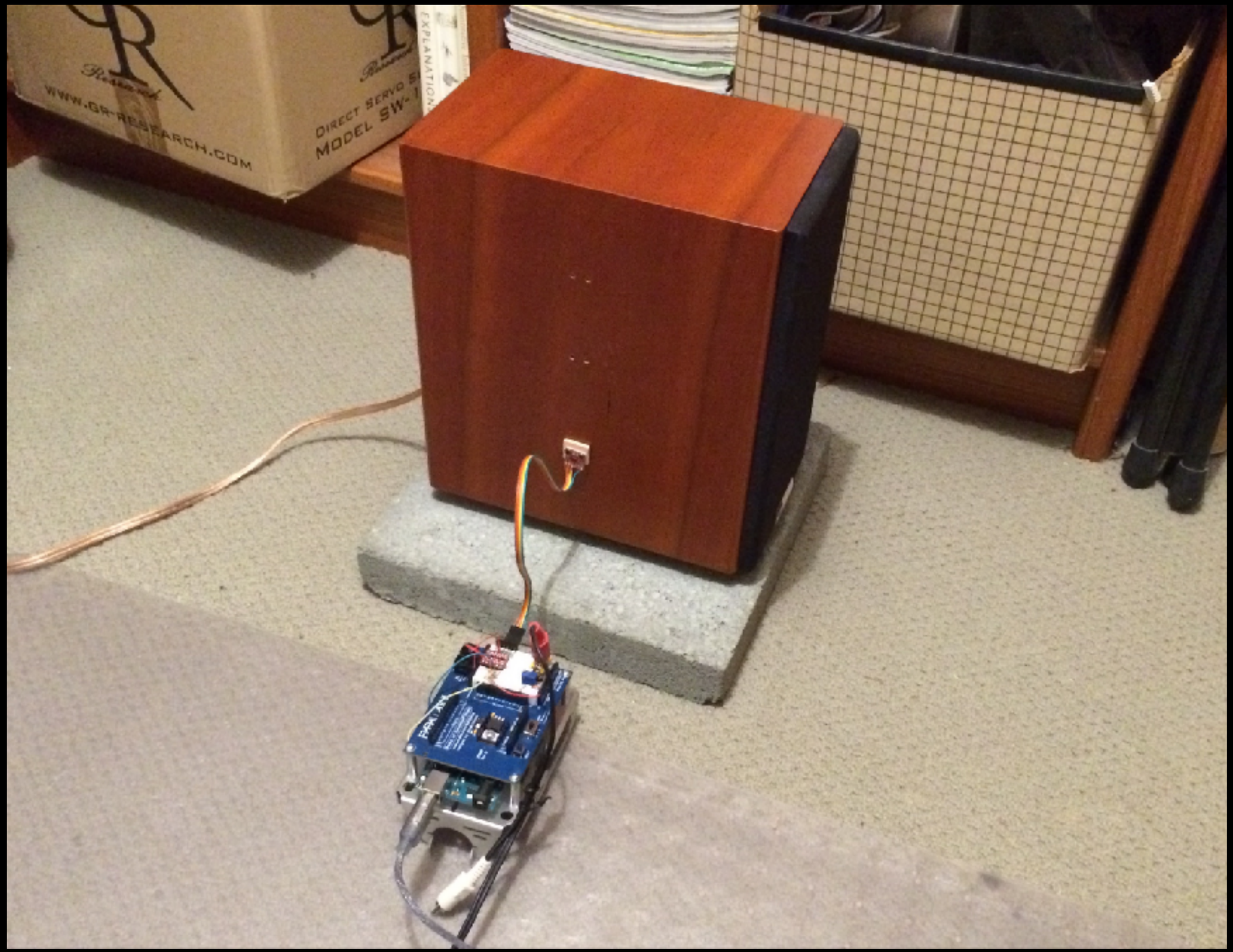
Testing location

- mid plane left wall (when viewed from front)
- 3” from bottom
- accelerometer orientation
  - y-axis: front-to-back (woofer movement)
  - x-axis: up and down
  - z-axis: in the plane of the panel


X
Y
Z
MMA8452
3.3V
SDA
SCL
I2
I1
GND
front
test location
rear

Speaker Base Test Surfaces

- 12” x 12” x 2” cement paver (Lowes Hardware)
- paver + shelf liner
- paver + bubble wrap
- paver + work grippers - Bench Cookies(tm)
- paver + speaker spikes
- floor carpet


WWW.GR-RESEARCH.COM
DIRECT SERVO
MODEL SW-


GOG TOOLS

Materials and Methods

- Arduino UNO + MMA8254Q
- power amp + speaker
- 100 Hz square wave (Arduino)
- 50 ms pulse
- data recorded for 75 ms
- 15 pulses for each surface

Data processing

- RMS of 50 ms pulse
  - mean, standard deviation (15 pulses)
- time series of 75 ms data with moving 3-point average of RMS
- all processing on Arduino
- results exported over USB serial port
- plots on Libre Office

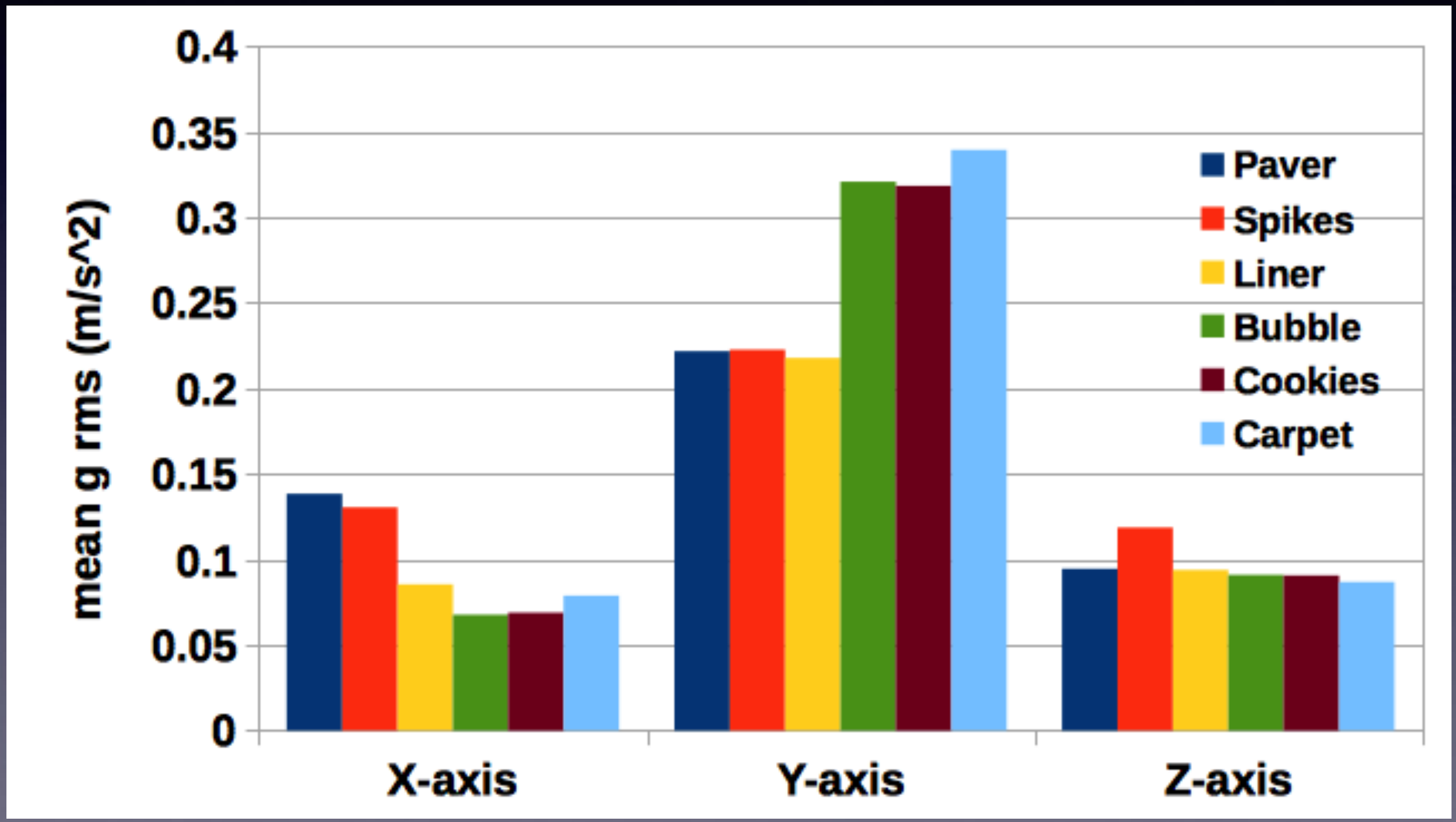
2 types of processed data

- RMS mean and sd of 50 ms pulse
- time series of entire 75 ms dataset

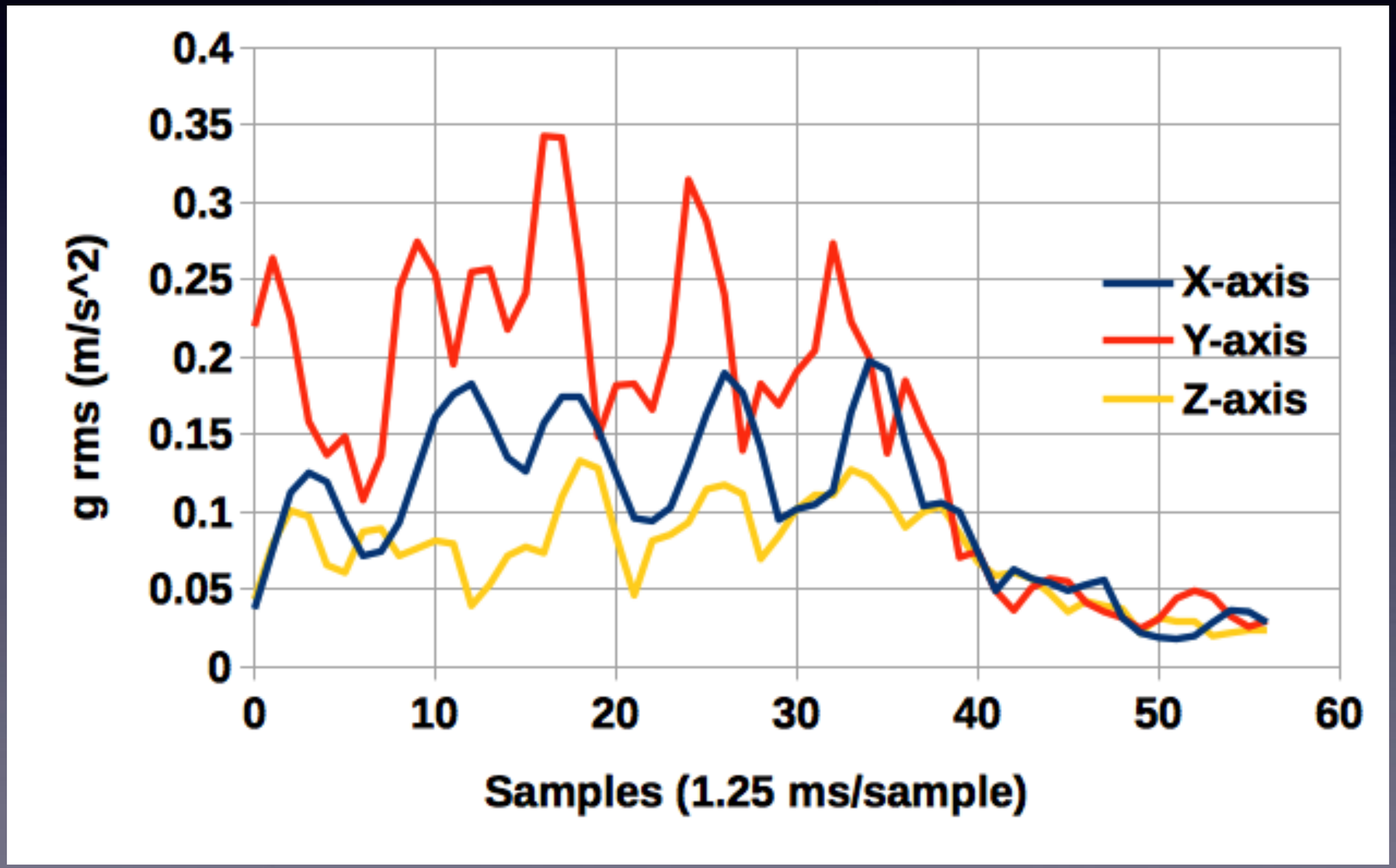
RMS 50 ms pulse

	mean	SD
X-axis	0.1383	0.0098
Y-axis	0.2217	0.0360
Z-axis	0.0945	0.0120

mean RMS of 50 ms pulse

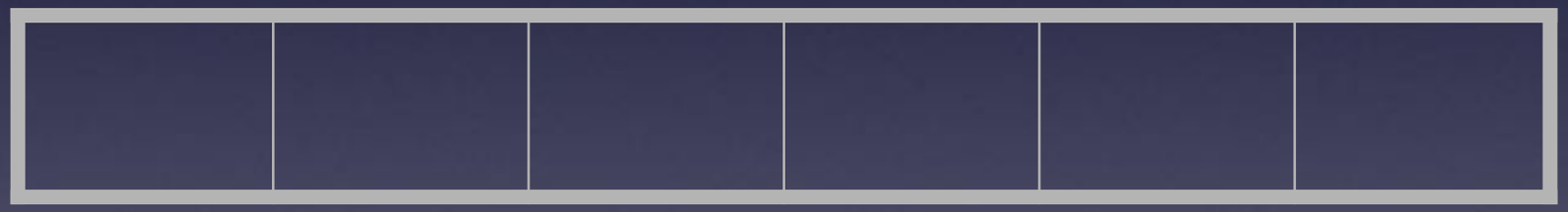


RMS of 3-point moving average entire 75 ms

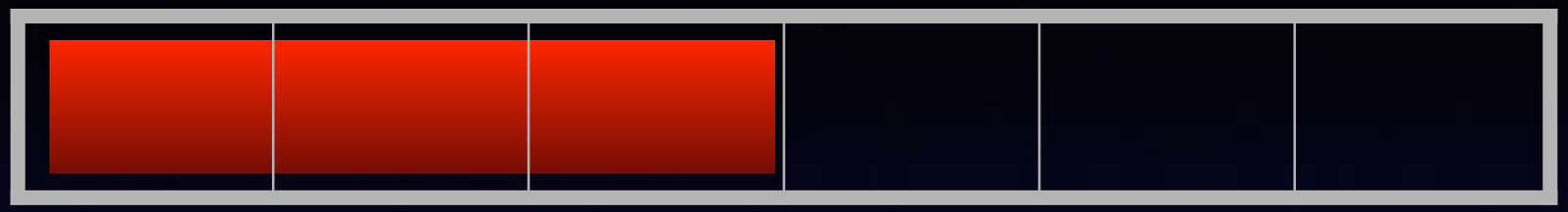


data

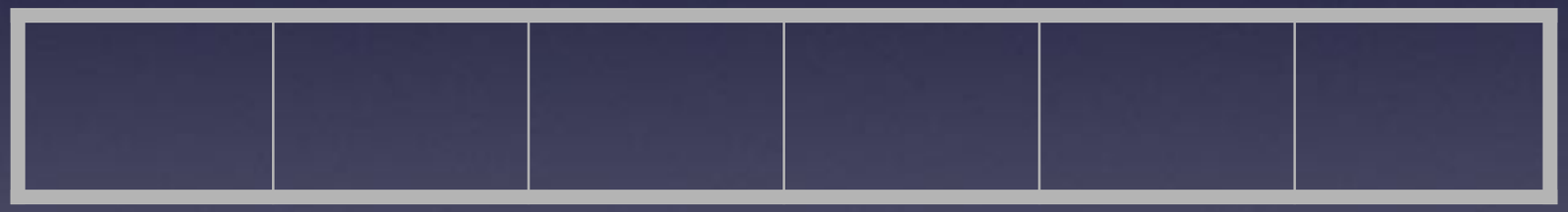
time series



data

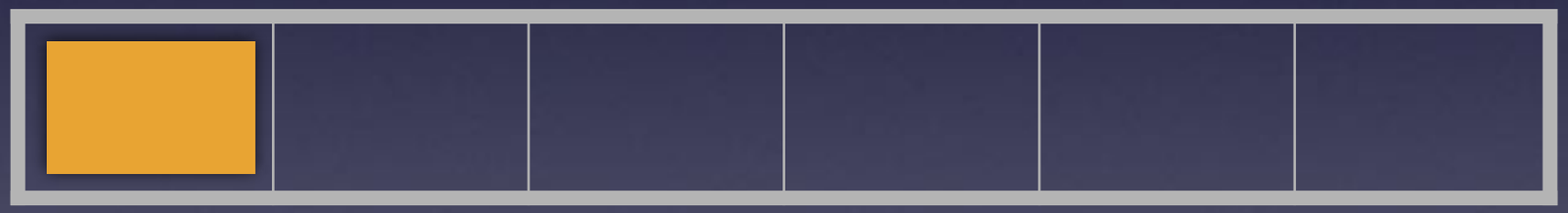


time series

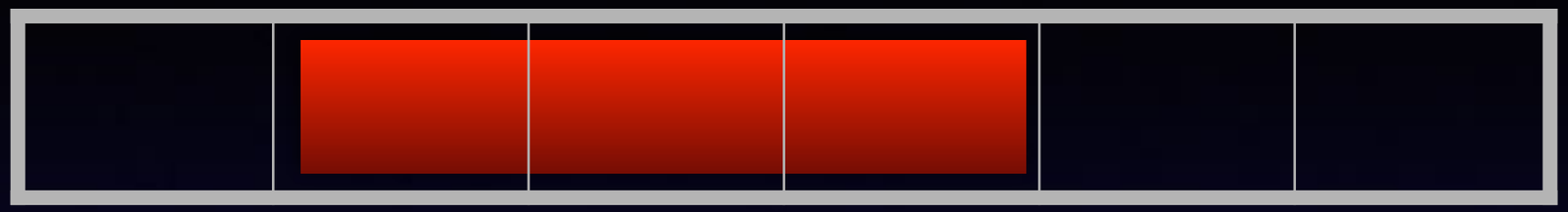


data

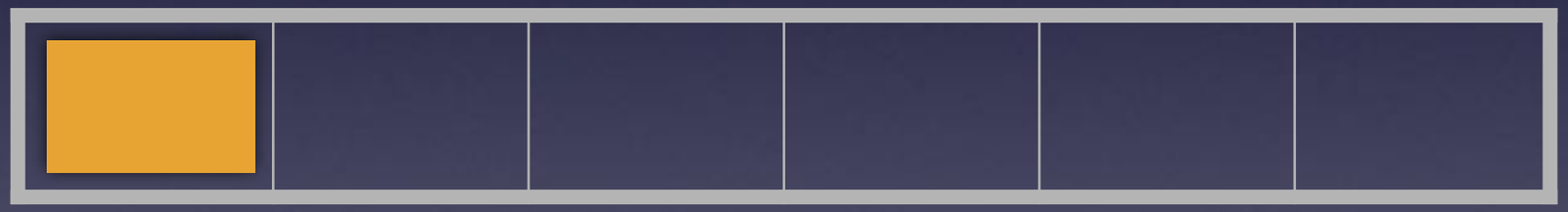
time series



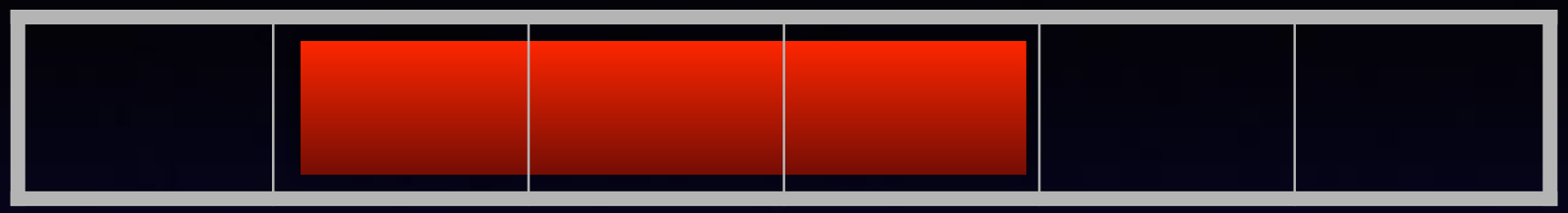
data



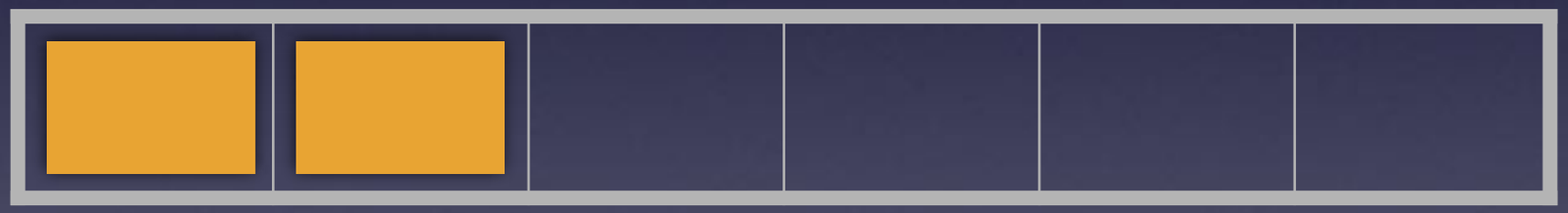
time series



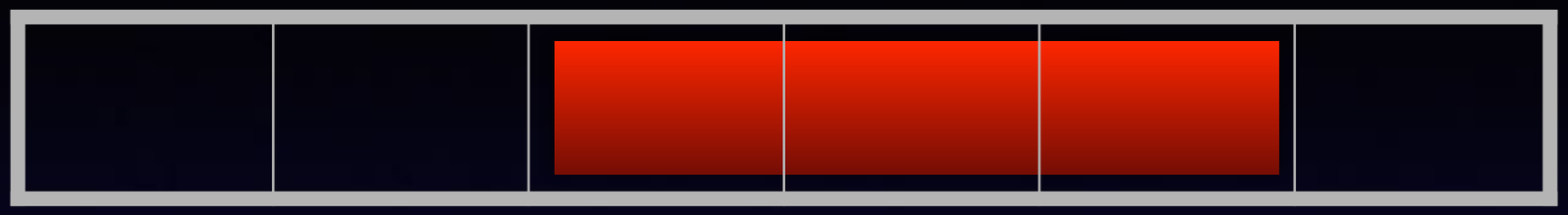
data



time series



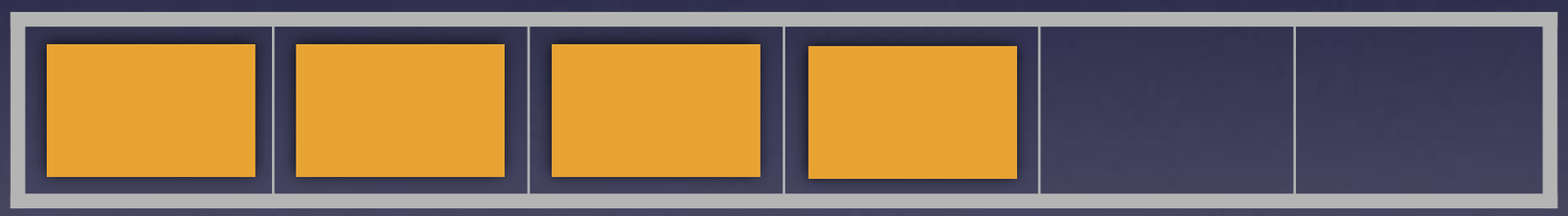
data

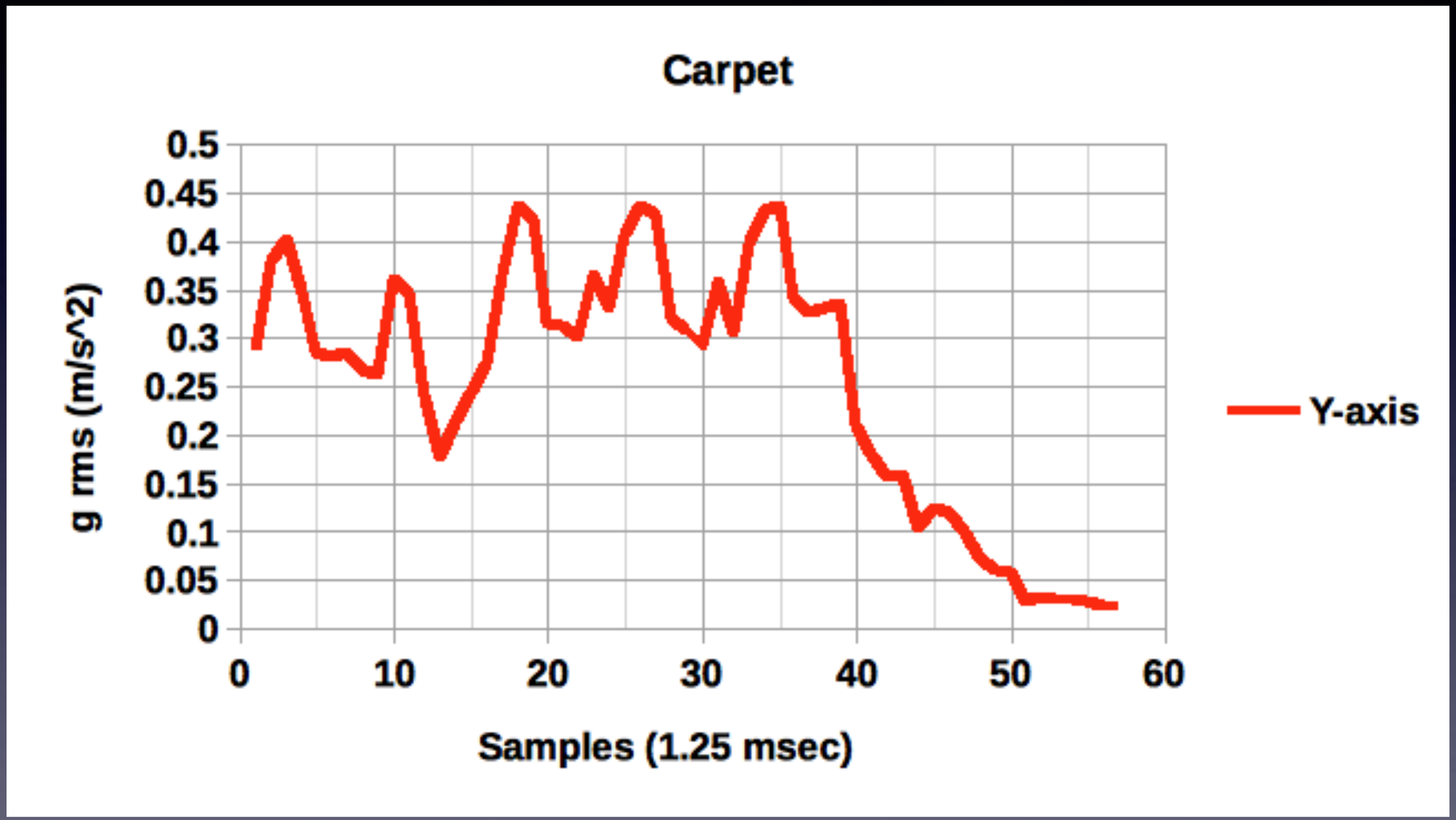


time series

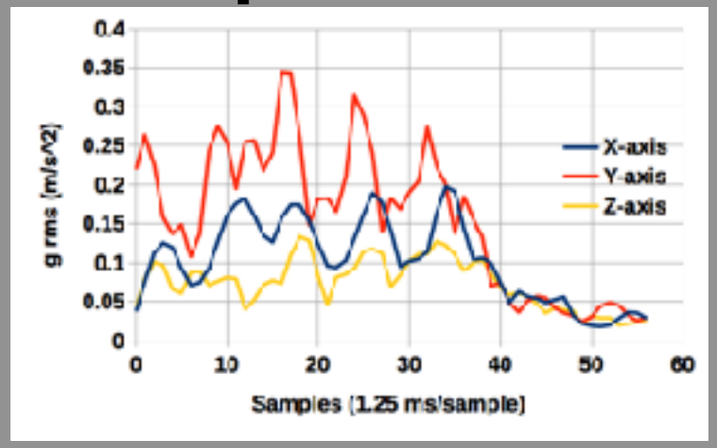
data

time series

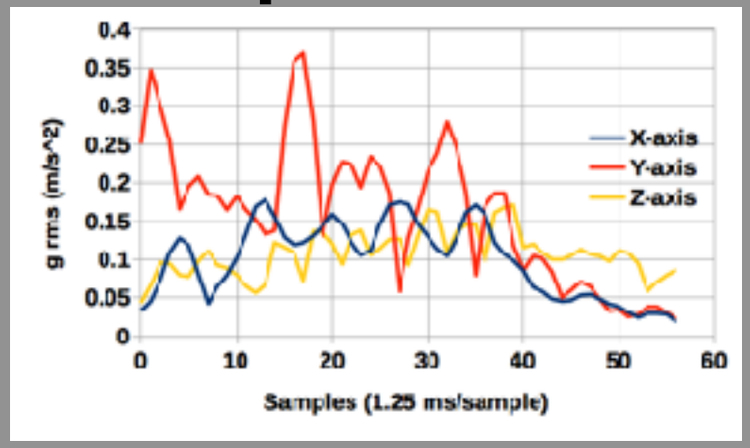



Carpet
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
g rms (m/s^2)
0
10
20
30
40
50
60
Samples (1.25 msec)
Y-axis

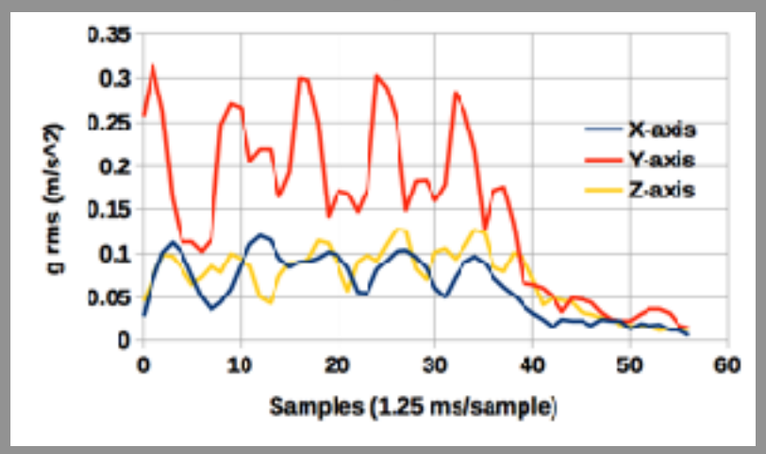
paver



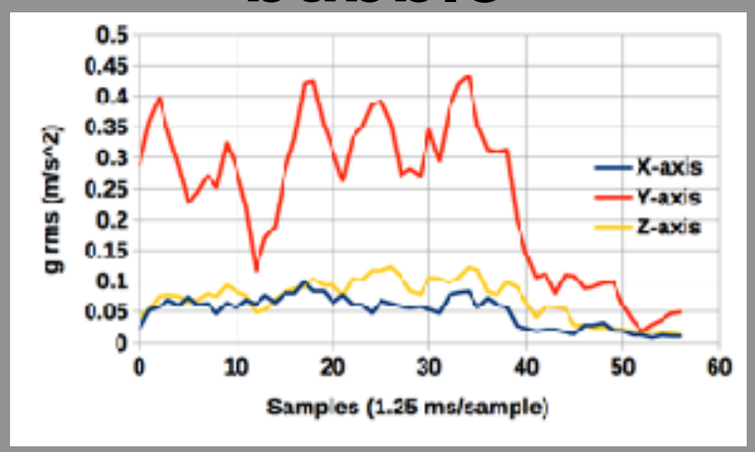
spikes



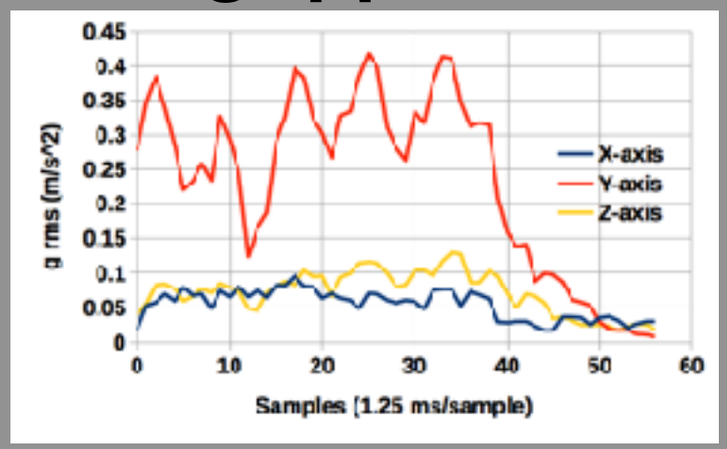
liner



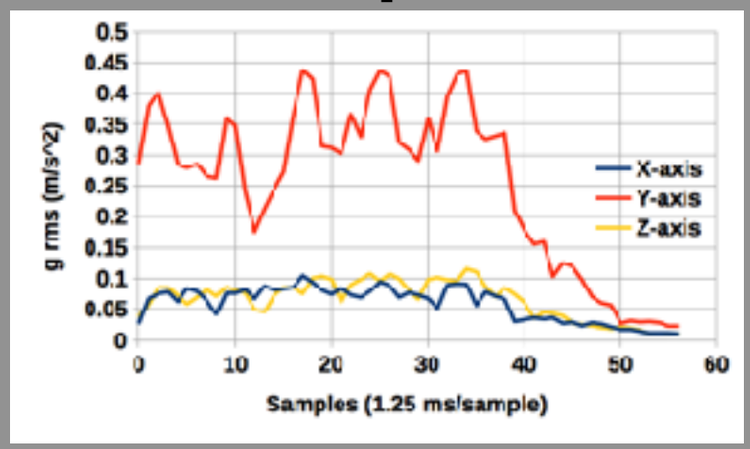
bubble



grippers



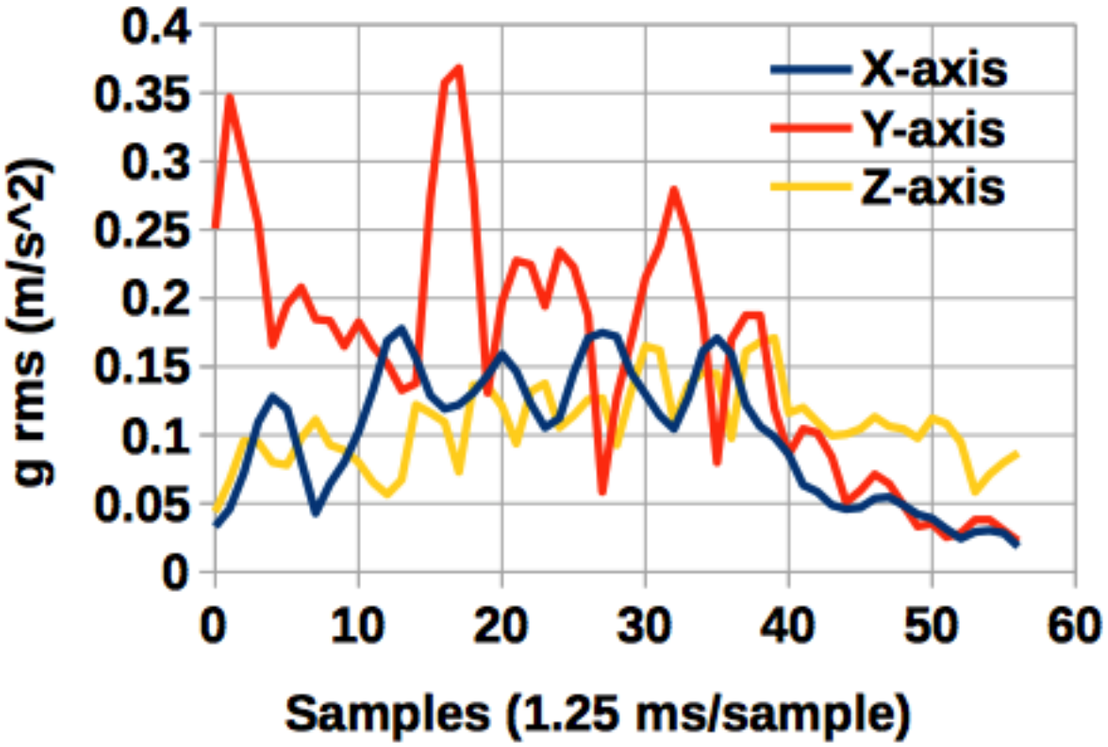
carpet



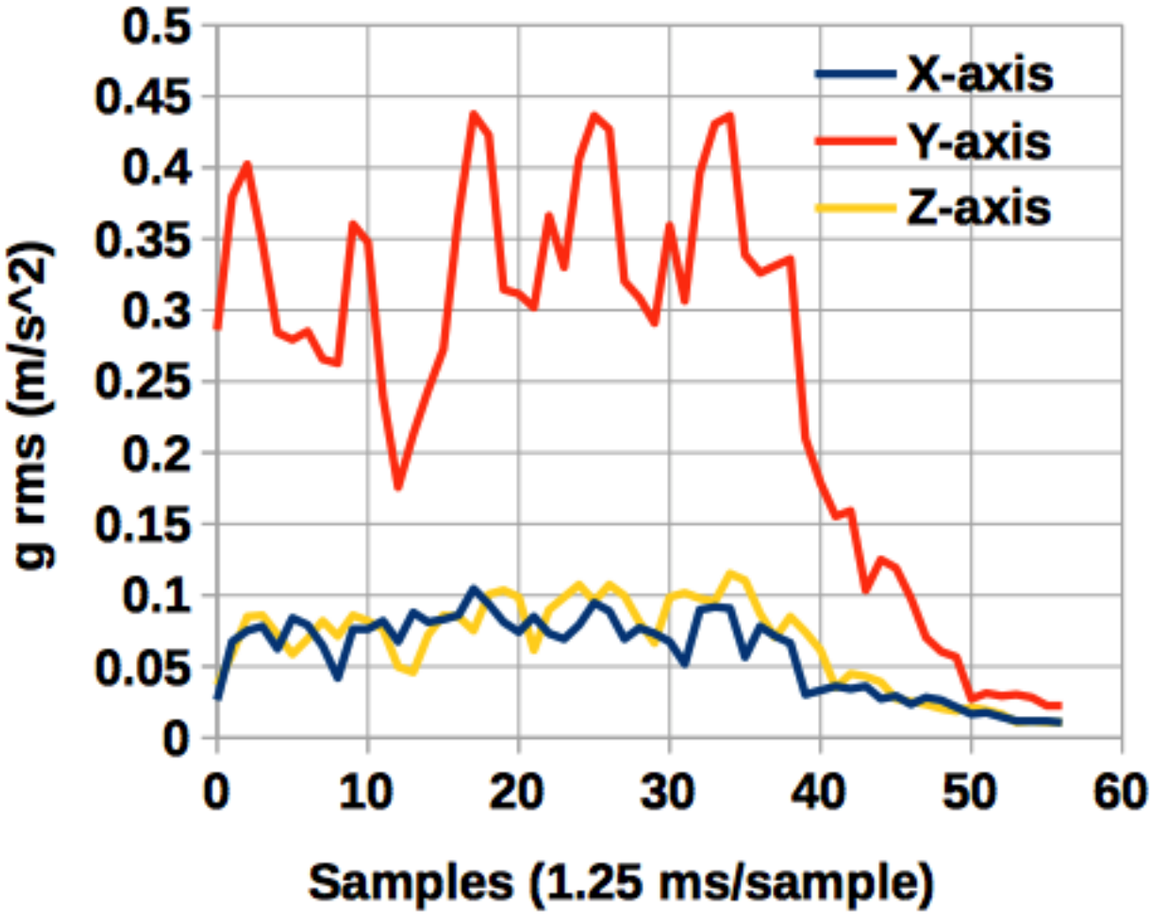
Two groups of data

- tightly coupled
  - speaker base more firmly attached to surface
- loosely coupled
  - speaker base more mechanical isolated from surface

tightly coupled



loosely coupled



spikes

X-axis: 0.1304 +/- 0.0120

Y-axis: 0.2226 +/- 0.0271

Z-axis: 0.1186 +/- 0.0098

carpet

X-axis: 0.0788 +/- 0.0061

Y-axis: 0.3394 +/- 0.0116

Z-axis: 0.0868 +/- 0.0059

Conclusions

- Analysis of speaker cabinet vibrations must account for how the cabinet is mounted on the support surface
- Comparison of different techniques to reduce cabinet vibrations may need to include both ‘tightly coupled’ and ‘loosely coupled’ conditions
- Both are commonly encountered in typical home stereo systems

Initializing the MMA5482Q accelerometer

```
#include <Wire.h>
#include <SFE_MMA8452Q.h>

MMA8452Q accel; // create MMA8452 object
accel.init(); // 800 Hz, +/- 2G

// additional parameters
accel.init(SCALE_8G, ODR_6);
accel.init(SCALE_2G, ODR_800);
```

Reading data - raw

```
short int x, y, z; // 2-byte ints

// 12-bit signed values
accel.read();
x = accel.x;
y = accel.y;
z = accel.z;
```

Reading data - calculated

```
float x_f, y_f, z_f; // calculated values in g;

accel.read();
x_f = accel.cx;
y_f = accel.cy;
z_f = accel.cz;

// equivalent result
x_f = x/1024.0;
```

How often should we read data?

- ODR 800 Hz
- T = 1/f = 1/800 = .00125 s = 1.25 ms
- how fast can we read the accel variables?

Streaming data out the serial port

```
void loop() {
  startTime = micros();
  for (i = 0; i < 100; i++)
    {
      accel.read();
      Serial.print(accel.x); Serial.print("\t");
      Serial.print(accel.y); Serial.print("\t");
      Serial.println(accel.z);
    }
  endTime = micros();
  Serial.print("Average time for cycle: ");
  Serial.print( ((endTime - startTime)/100.0)/1000);
  Serial.println(" ms");
}
```

Nowhere near fast enough!!

```
MMA8452 serial time test

-990 -42 264
-986 -40 262
…
-991 -37 262
-991 -38 264

Average per cycle: 6.95 ms (144 Hz)
```

Writing data to an array

```
short data[100][3];
void loop() {
  startTime = micros();
  for (i = 0; i < 100; i++)
  {
    accel.read();
    data[i][X_AXIS] = accel.x;
    data[i][Y_AXIS] = accel.y;
    data[i][Z_AXIS] = accel.z;
  }
  endTime = micros();
  Serial.print("Average time for cycle: ");
  Serial.print( ((endTime - curTime)/100.0)/1000);
  Serial.println(" msec");
```

MMA8452Q array write time test

Average time for cycle: 1.02 msec

How do we synchronize reads and ODR?

- match read interval with ODR

```
short data[100][3];
startTime = micros();
for (i = 0; i < 100; i++)
{
  curTime = micros();
  accel.read();
  data[i][X_AXIS] = accel.x;
  data[i][Y_AXIS] = accel.y;
  data[i][Z_AXIS] = accel.z;
  while (micros() - curTime < 1250)
    ;
}
endTime = micros();
Serial.print("Average time for cycle: ");
Serial.print( ((endTime - startTime)/100.0)/1000);
Serial.println(" msec");
```

MMA8452 time synch test

Average time for cycle: 1.28 msec

Calculating zero offsets

- stationary accelerometer measures the static gravitational force (g)
- 9.81 m/s
- what we want: change from resting position
- average of 100 values immediately prior to pulse

```
long offset_x, offset_y, offset_z;
offset_x = 0; offset_y = 0; offset_z = 0;
for (i = 0; i < 100; i++)
{
    curTime = micros();
    accel.read();
    offset_x += accel.x;
    offset_y += accel.y;
    offset_z += accel.z;
    while (micros() - curTime < 1250)
        ;
}
offset_x /= i;
offset_y /= i;
offset_z /= i;
```

```
for (i = 0; i < 100; i++)
{
  curTime = micros();
  accel.read();
  data[i][X_AXIS] = accel.x - offset_x;
  data[i][Y_AXIS] = accel.y - offset_y;
  data[i][Z_AXIS] = accel.z - offset_z;
  while (micros() - curTime < 1250)
    ;
}
```

Yes, but how many samples?

```
T = 1/f (sec) x 1000 = msec
pulse(msec)/T(msec) = # of samples

ODR = 800Hz
T = (1/800) x 1000 = 1.25 msec
50 msec pulse / 1.25 msec = 40 samples
```

Generating the audio pulse

```
#define TONE_PIN  2
#define NOTE 100 // 100 Hz square wave
#define NOTE_DURATION 50 //  millisec
```

```
tone(TONE_PIN, NOTE, NOTE_DURATION);
```

How long can I measure?

- for every sample
  - short data[1][3]; // 6-bytes
    - 2048/6 = 340 samples
    - ODR 800 Hz, 340 x 1.25 = 425 msec
- other program variables
- system variables and resources

RMS calculation

$$x_{\mathrm{rms}} = \sqrt{\frac{1}{n}\left(x_1^2 + x_2^2 + \cdots + x_n^2\right)}.$$

Calculating RMS from data

- for x, y and z-axis
- calculate sum of squares
- divide by n
- take square root
- when do we convert to floating point???
- when do we convert to g???

```
float sum_sq_x = 0, sum_sq_y = 0;
float sum_sq_z = 0;
float rms_x, rms_y, rms_z;
for (i = 0; i < SAMPLES; i++)
{
   sum_sq_x += (float)data[i][X_AXIS] *
(float)data[i][X_AXIS];
   sum_sq_y += (float)data[i][Y_AXIS] *
(float)data[i][Y_AXIS];
   sum_sq_z += (float)data[i][Z_AXIS] *
(float)data[i][Z_AXIS];
}
rms_x = sqrt(sum_sq_x / SAMPLES);
rms_y = sqrt(sum_sq_y / SAMPLES);
rms_z = sqrt(sum_sq_z / SAMPLES);

rms_x /= 1024.0, rms_y /= 1024.0, rms_z /= 1024.0;
```

```
float sum_sq_x = 0, sum_sq_y = 0;
float sum_sq_z = 0;
float rms_x, rms_y, rms_z;
for (i = 0; i < SAMPLES; i++)
{
   sum_sq_x += (float)data[i][X_AXIS] *
(float)data[i][X_AXIS];
   sum_sq_y += (float)data[i][Y_AXIS] *
(float)data[i][Y_AXIS];
   sum_sq_z += (float)data[i][Z_AXIS] *
(float)data[i][Z_AXIS];
}
rms_x = sqrt(sum_sq_x / SAMPLES);
rms_y = sqrt(sum_sq_y / SAMPLES);
rms_z = sqrt(sum_sq_z / SAMPLES);

rms_x /= 1024.0, rms_y /= 1024.0, rms_z /= 1024.0;
```

```
float sum_sq_x = 0, sum_sq_y = 0;
float sum_sq_z = 0;
float rms_x, rms_y, rms_z;
for (i = 0; i < SAMPLES; i++)
{
   sum_sq_x += (float)data[i][X_AXIS] *
(float)data[i][X_AXIS];
   sum_sq_y += (float)data[i][Y_AXIS] *
(float)data[i][Y_AXIS];
   sum_sq_z += (float)data[i][Z_AXIS] *
(float)data[i][Z_AXIS];
}
rms_x = sqrt(sum_sq_x / SAMPLES);
rms_y = sqrt(sum_sq_y / SAMPLES);
rms_z = sqrt(sum_sq_z / SAMPLES);

rms_x /= 1024.0, rms_y /= 1024.0, rms_z /= 1024.0;
```

```
float sum_sq_x = 0, sum_sq_y = 0;
float sum_sq_z = 0;
float rms_x, rms_y, rms_z;
for (i = 0; i < SAMPLES; i++)
{
   sum_sq_x += (float)data[i][X_AXIS] *
(float)data[i][X_AXIS];
   sum_sq_y += (float)data[i][Y_AXIS] *
(float)data[i][Y_AXIS];
   sum_sq_z += (float)data[i][Z_AXIS] *
(float)data[i][Z_AXIS];
}
rms_x = sqrt(sum_sq_x / SAMPLES);
rms_y = sqrt(sum_sq_y / SAMPLES);
rms_z = sqrt(sum_sq_z / SAMPLES);

rms_x /= 1024.0, rms_y /= 1024.0, rms_z /= 1024.0;
```

```
float sum_sq_x = 0, sum_sq_y = 0;
float sum_sq_z = 0;
float rms_x, rms_y, rms_z;
for (i = 0; i < SAMPLES; i++)
{
   sum_sq_x += (float)data[i][X_AXIS] *
(float)data[i][X_AXIS];
   sum_sq_y += (float)data[i][Y_AXIS] *
(float)data[i][Y_AXIS];
   sum_sq_z += (float)data[i][Z_AXIS] *
(float)data[i][Z_AXIS];
}
rms_x = sqrt(sum_sq_x / SAMPLES);
rms_y = sqrt(sum_sq_y / SAMPLES);
rms_z = sqrt(sum_sq_z / SAMPLES);

rms_x /= 1024.0, rms_y /= 1024.0, rms_z /= 1024.0;
```

```
float sum_sq_x = 0, sum_sq_y = 0;
float sum_sq_z = 0;
float rms_x, rms_y, rms_z;
for (i = 0; i < SAMPLES; i++)
{
   sum_sq_x += (float)data[i][X_AXIS] *
(float)data[i][X_AXIS];
   sum_sq_y += (float)data[i][Y_AXIS] *
(float)data[i][Y_AXIS];
   sum_sq_z += (float)data[i][Z_AXIS] *
(float)data[i][Z_AXIS];
}
rms_x = sqrt(sum_sq_x / SAMPLES);
rms_y = sqrt(sum_sq_y / SAMPLES);
rms_z = sqrt(sum_sq_z / SAMPLES);

rms_x /= 1024.0, rms_y /= 1024.0, rms_z /= 1024.0;
```

```
float sum_sq_x = 0, sum_sq_y = 0;
float sum_sq_z = 0;
float rms_x, rms_y, rms_z;
for (i = 0; i < SAMPLES; i++)
{
   sum_sq_x += (float)data[i][X_AXIS] *
(float)data[i][X_AXIS];
   sum_sq_y += (float)data[i][Y_AXIS] *
(float)data[i][Y_AXIS];
   sum_sq_z += (float)data[i][Z_AXIS] *
(float)data[i][Z_AXIS];
}
rms_x = sqrt(sum_sq_x / SAMPLES);
rms_y = sqrt(sum_sq_y / SAMPLES);
rms_z = sqrt(sum_sq_z / SAMPLES);

rms_x /= 1024.0, rms_y /= 1024.0, rms_z /= 1024.0;
```

```
float sum_sq_x = 0, sum_sq_y = 0;
float sum_sq_z = 0;
float rms_x, rms_y, rms_z;
for (i = 0; i < SAMPLES; i++)
{
   sum_sq_x += (float)data[i][X_AXIS] *
(float)data[i][X_AXIS];
   sum_sq_y += (float)data[i][Y_AXIS] *
(float)data[i][Y_AXIS];
   sum_sq_z += (float)data[i][Z_AXIS] *
(float)data[i][Z_AXIS];
}
rms_x = sqrt(sum_sq_x / SAMPLES);
rms_y = sqrt(sum_sq_y / SAMPLES);
rms_z = sqrt(sum_sq_z / SAMPLES);

rms_x /= 1024.0, rms_y /= 1024.0, rms_z /= 1024.0;
```

```
float sum_sq_x = 0, sum_sq_y = 0;
float sum_sq_z = 0;
float rms_x, rms_y, rms_z;
for (i = 0; i < SAMPLES; i++)
{
   sum_sq_x += (float)data[i][X_AXIS] *
(float)data[i][X_AXIS];
   sum_sq_y += (float)data[i][Y_AXIS] *
(float)data[i][Y_AXIS];
   sum_sq_z += (float)data[i][Z_AXIS] *
(float)data[i][Z_AXIS];
}
rms_x = sqrt(sum_sq_x / SAMPLES);
rms_y = sqrt(sum_sq_y / SAMPLES);
rms_z = sqrt(sum_sq_z / SAMPLES);

rms_x /= 1024.0, rms_y /= 1024.0, rms_z /= 1024.0;
```

```
float sum_sq_x = 0, sum_sq_y = 0;
float sum_sq_z = 0;
float rms_x, rms_y, rms_z;
for (i = 0; i < SAMPLES; i++)
{
   sum_sq_x += (float)data[i][X_AXIS] *
(float)data[i][X_AXIS];
   sum_sq_y += (float)data[i][Y_AXIS] *
(float)data[i][Y_AXIS];
   sum_sq_z += (float)data[i][Z_AXIS] *
(float)data[i][Z_AXIS];
}
rms_x = sqrt(sum_sq_x / SAMPLES);
rms_y = sqrt(sum_sq_y / SAMPLES);
rms_z = sqrt(sum_sq_z / SAMPLES);

rms_x /= 1024.0, rms_y /= 1024.0, rms_z /= 1024.0;
```

How do I know my code is valid???

- 2048 byes SRAM - memory is tight!
- decisions, decisions, decisions
- 2-byte vs 4-byte ints (avoid overflow)
- 4-byte floats (limited precision)
- converting to gravitational units (g)

Code validation

- C++ program on PC
- sin() function and a loop from 0 to 4pi
- array of values [-1, 1]
- calculated root mean square (RMS)
- value almost exactly as expected
  - RMS 1 volt signal = 1/sqrt(2) = 0.707107
  - RMS test program: 0.70616 (0.13% error)
  - code ported to Arduino gave identical result

Chassis vibrations

- motor => gear => wheels
- wheels - different composition, treads
- surface
  - smooth wood/linoleum
  - carpet
  - outdoors

Chassis vibrations

- develop characteristic profiles
- analysis of system behavior
- troubleshoot and optimize


QUOTATIONS
1945
1970
10/03/2011

Sketch uses 11,950 bytes (37%) of program storage space. Maximum is 32,256 bytes.
Global variables use 1,652 bytes (80%) of dynamic memory, leaving 396 bytes for local variables. Maximum is 2,048 bytes.
Low memory available, stability problems may occur.

Speaker energy transfer model