Adding an Accelerometer Sensor Library to TI-Innovator using Python

Hans-Martin Hilbig

T³ DEUTSCHLAND

Teachers Teaching with Technology™

Overview

Accelerometers are widely spread across today's smart devices like Smartphones, Fitness Watches and are a key sensor element in modern automobiles for safety applications like airbag systems or dynamic drive assistant systems like ESP.
From a technical point of view, accelerometers are micro-electro-mechanical-systems (MEMS), sensing the momentary acceleration of the object they are part of, relative to a 3-dimensional x-y-z orthogonal coordinate system. The measurement result contains a dynamic component and a static component. Dynamic components may be the momentary acceleration due to a car accelerating, decelerating or driving along a curve, an object or a person falling, a person walking, a momentary vibration, etc. Static component is the gravity which acts on every object on earth.
This project is delivering a Python based ADXL335 driver library, to be used with the TI-Innovator System.

The ADXL335 Sensor

The ADXL335 is a 3-axis Accelerometer MEMS sensor manufactured by Analog Devices, Inc. All ADXL335 functional and parametric data mentioned in this document is based on datasheets and application reports issued by Analog Devices [1,2].
The ADXL335 is particularly suited for the TI-Innovator Hub, as its measurement results are offered as analog output voltages proportional to the acceleration of the sensor in the x-y-z axis. The analog.in() function of Nspire Python is being used to receive the analog output voltage and translate it into digital Integer values to be processed by the TI-Innovator System using Python.
ADXL335 is accepting a supply voltage of 3.3V and can be operated from a standard 3.3V power supply pin of the TI-Innovator Hub.
While the ADXL335 is offered in a micro surface mount package, there are modules with standard pin connectors available, such as the SeeedStudio ADXL335 board [3], which support easy connection to Microcontroller Boards such as the TI-Hub, without the need for an extra breadboard.
Although technically possible, the ADXL335 driver library is not supporting connection of the sensor through multiple grove connectors (IN1/2/3), but rather is based on the three analog BB inputs of the TI-Innovator Hub (BB5/BB6/BB7). Wiring of GND, 3.3V, x-out, y-out and z-out is done with 5 male/female breadboard cables, female end connected to Grove board, male end connected to the TI-Hub's BB port.
The ADXL335 driver library is not using the ST (self test) pin which is laid out on some of the sensor modules available on the market. This pin should be left open.

ADXL335 driver library from a STEM perspective

The driver library is structured in a strict hierarchical way, resulting in 5 hierarchy levels. The level 0 function simply reads the analog voltage from the x-, y-, or z-output and returns the digital integer value from the 14-bit Analog-to-Digital Converter (ADC) of the TI-Hub. With just this simple function, there are numerous experiments possible for even the younger grade students who are not yet experienced in Physics or Geometry.
Level 1 function calls the level 0 function for each of the three sensor planes (x,y,z) and returns all three sensor values altogether, as a 3-element Python list.
Level 2 function adds gain and offset and uses simple algebra to transform the digital integer values to gravitational (G) values. Possible static G values range from -1 (gravity vector opposite to axis vector), thru 0 (gravity vector orthogonal to axis vector) to +1 (gravity vector parallel to axis vector) for each of the three coordinate axes (x,y,z). Learning objectives from level 2 function are: normalizing integer measurements by using gain (the slope m of a linear function mx+b) and offset (x0 when gravity vector is orthogonal to coordinate axis), translating this into a parameter of the Physics domain. Experiments aimed at higher grade

students eventually lead to applying the parallelogram of forces, which is what the level 3 function does.
Level 3 function calculates the three Euler angles of the acceleration sensor exposed in the 3 dimensional space, by applying trigonometric rules. The angle of each of the x-, y- and z-vectors of the ADXL335 module is calculated by using the measurement results from all three x-, y-, z-sensor outputs, to augment the accuracy of the overall result. See [2] for details. This methodology is called sensor fusion and is a commonly used practice in today's high tech systems. Experiments with level 3 function are aimed at higher grade students, who are familiar with Physics, Geometry, vector calculus and trigonometry.
Finally, the level 4 function is created to support a simple Human-Machine-Interface (HMI), using human gestures with the ADXL335 sensor board on hand to control directional movements like done with the cursor keys (<up><down><left><right>) of a computer keyboard. The response time of the function returning one of these four directional commands is reverse proportional to the inclination angle of the accelerometer in the <forward><backward><left><right> direction (see adxl_demo2.tns, tab <1.8> for a demo code) or refer to the Hedgehog publication on T3 Europe [5].

Installation and use of the ADXL335 driver library

This project is supplied with two Nspire Python documents. ADXL335_1.tns contains the driver library, to be installed in the pylib folder of your PC Desktop and/or the Nspire Handheld. Don't forget to press <refresh libraries> to make the systems aware of the new driver library.

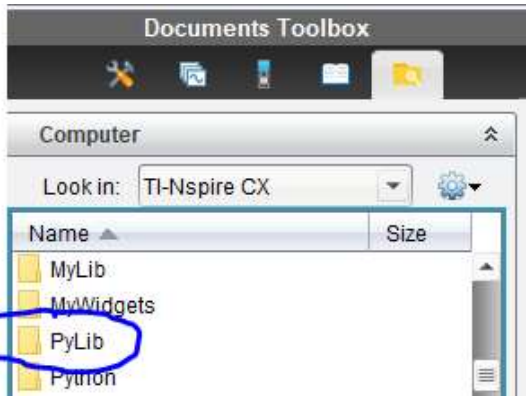


Figure 1 Installation of driver library

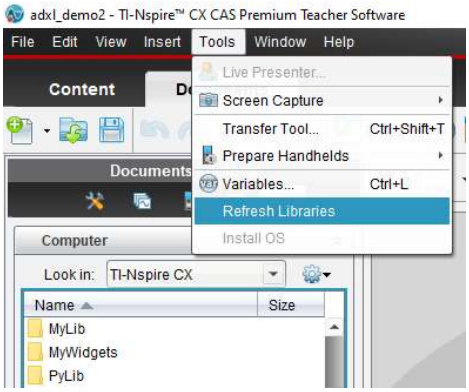


Figure 2 Refreshing the libraries

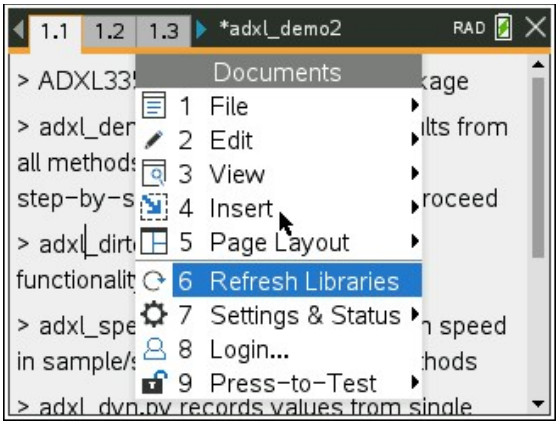


Figure 3 Refreshing the libraries on a Nspire CXII

When the user program is executed for the first time, an automatic calibration procedure is invoked at instantiation, asking the user to put the ADXL335 sensor board on a flat surface, followed by turning the sensor board upside down on the same flat surface. This procedure is needed to determine the min, max and offset values of the sensor-TI-Hub system.

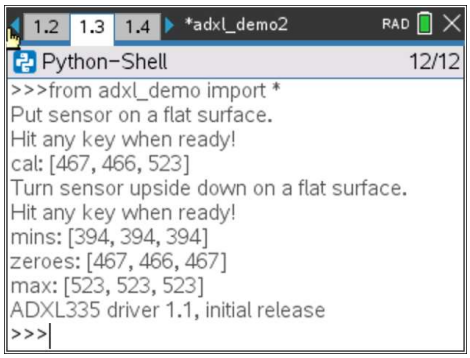


Figure 4 Calibration user interface

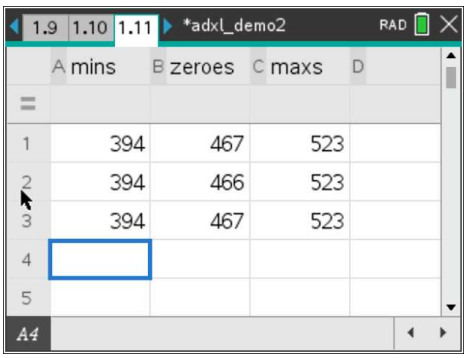
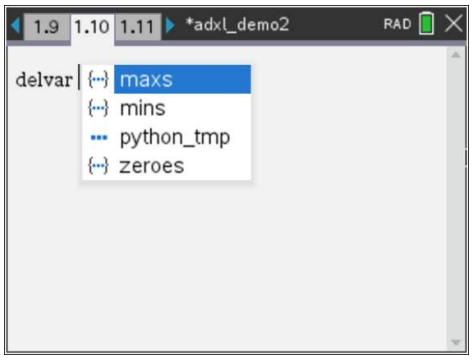


Figure 5 Calibration values in Lists & Spreadsheet

The calibration() method is communicating with the user in the Python Shell window (Figure 4). Calibration values are stored in Nspire lists in the user's document (Figure 5). On subsequent calls, the user may opt for either repeating the sensor calibration or to proceed with just restoring calibration values from the Nspire lists.



To simulate a fresh start of the adxl() class in a user program, add a <calculator> window to the .tns file and delete at least one of the calibration variables, using the delvar command (Figure 6), before you start your Python user code.

Figure 6 Calculator window

The second document, adxl_demo2.tns, contains a couple small Python demo programs to illustrate the use of the library functions and help you with testing your system setup.

Calibration - accuracy considerations and limitations

When using the ADXL335 library, users need to be aware of three limitations and simplifications done in the current implementation.
A 2-point calibration method is used in the current ADXL335 driver library, in order to simplify the calibration process.
When the sensor is placed on a flat surface, the gravity force is orthogonal to both, the x- and the y-axis of the sensor. The calibration method is storing these values as the measured offset values for the x- and y-axis, but also uses a copy of the x-axis value for the z-axis offset, assuming the pin-to-pin variance of the system can be neglected. The gravity force vector is in parallel to and in the opposite direction of the positive z-axis vector, resulting in a maximum gravity measurement value for the z-axis. This value is used as a copy for the x-max and y-max values, too.

When the sensor is faced upside down on the flat surface, the gravity force vector is in parallel, but same direction to the z-axis vector, gaining the minimum gravity measurement value. This value is being copied and used as the x- and y-axis minimum values, too.
The analog-to-digital converter (ADC) integrated on the MSP432 microcontroller used in the TI-Hub creates a 14-bit Integer based on the analog value measured at the BB input. The reference voltage used in the TI-Hub is nominal 3.3V, resulting in a 3.3V/16383=0.0002014V or 0.2mV per ADC step. Using the non-shielded breadboard cables to connect the accelerometer to the ADC will impose electrical noise to the input. To avoid this noise to affect the accelerometer measurement, the adxl() driver reduces the bit-width of the ADC to 10 bits, which is much more adequate in the system setup used with the TI-Innovator system. The bit width reduction is achieved by shifting the ADC Integer 4 bits to the right in the level 0 get_adcx/y/z() functions.

The third accuracy limitation is coming from the protective I/O circuitry built into the TI-Innovator Hub. While voltages below 1V are seeing a high impedance analog BB port input, the load seen at analog voltages beyond 1V is higher, due to parasitic leakage of the protective I/O circuitry gradually coming into effect. Unfortunately, the output impedance of the ADXL335 is specified at 32kOhms typically, making the sensor output vulnerable to those leakages imposed by the I/O protection circuitry of the TI-Hub.

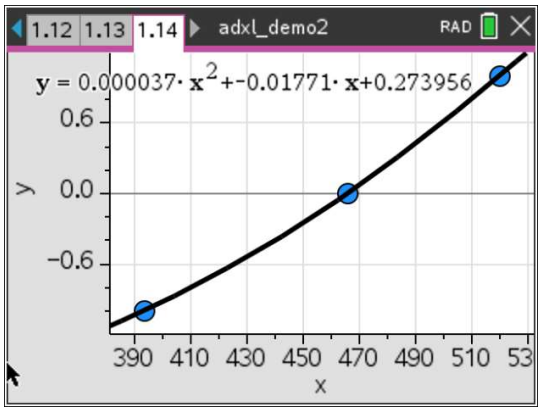


Figure 7 ADXL335 gain curve in a TI-Hub setup

Figure 7 illustrates the real gain curve of an ADXL335 sensor connected to a TI-Hub, which is not linear but rather is represented well by a 2nd order polynomial trendline.
The calibration routine is not taking this behavior into account and calculations made in the level 2 method <get_gforcexyz()> are assuming a linear behavior of the ADXL335 gains and offsets measured. This simplification leads to artifacts such as gravity values less than +/-1G or exceeding +/-1G at the inclination extremes of the sensor. Calculated Euler angles may show errors as high as +/-5 degrees at the 0/90/180 degree extremes.

Methods and attributes of ADXL335 driver library

As described before, all methods are coded in a strictly hierarchical scheme, with a higher level method building on a lower level method. All methods beginning with a <get_> syntax are functions returning sensor data at different abstraction levels. Methods starting with a <show_> syntax are intended to be used for displaying the sensor data for dashboard or monitoring applications.
All methods, their abstraction level, attributes, attribute defaults and method execution speed are listed here:

Method	Abstraction level	Attributes	Defaults	Description	Execution speed samples/s
get_adcx()	0	N/A	N/A	Returns the ADC value of the x-sensor (10 bit Integer)	22
get_adcy()	0	N/A	N/A	Returns the ADC value of the y-sensor (10 bit Integer)	22
get_adcz()	0	N/A	N/A	Returns the ADC value of the z-sensor (10 bit Integer)	22
get_adcxyz()	1	N/A	N/A	Calls all three level 0 functions and returns ADC values of the x-, y-, and z-sensor in a Python list (10 bit Integers)	8
get_gforcexyz()	2	N/A	N/A	Calls level 1 function and returns gravity values, calculated from offset & gain values generated by the calibration function (list of 2 decimals Float)	8
get_anglexyz()	3	N/A	N/A	Calls level 2 function and returns Euler angles, calculated from trigonometry & sensor fusion (list of 1 decimal angles Float)	8
show_adcx()	N/A	Text position	textpos=1	Displays the ADC value measured by a preceeding get_adcx() method	100
show_adcy()	N/A	Text position	textpos=1	Displays the ADC value measured by a preceeding get_adcy() method	100
show_adcz()	N/A	Text position	textpos=1	Displays the ADC value measured by a preceeding get_adcz() method	100
show_adcxyz()	N/A	Text position	textpos=1	Displays all three ADC values measured by a preceeding get_adcxyz() method	84
show_gforcexyz()	N/A	Text position	textpos=1	Displays all three G-force values calculated by a preceeding get_gforcexyz() method	84
show_anglexyz()	N/A	Text position	textpos=1	Displays all three Euler angle values calculated by a preceeding get_anglexyz() method	84
set_avg()	N/A	ADC averaging	avg=3	Sets the ADC averaging value (0-25) to filter sensor and ADC noise	168
get_avg()	N/A	N/A	N/A	Returns the ADC averaging value set by a preceeding set_avg() method	16k
ver()	N/A	N/A	N/A	returns ADXL335 library version info	N/A

Table 1 List of callable methods of ADXL335 library

Each of the <show_> methods needs to be preceeded by at least one corresponding <get_> method, in order to display meaningful results. Otherwise, an error (missing variable) might occur.

Adxl_demo2.tns – inspiring accelerometer projects in STEM lessons

The adxl_demo2.tns Python document contains a couple demo programs to inspire the user to create new experiments using the adxl335 driver library.

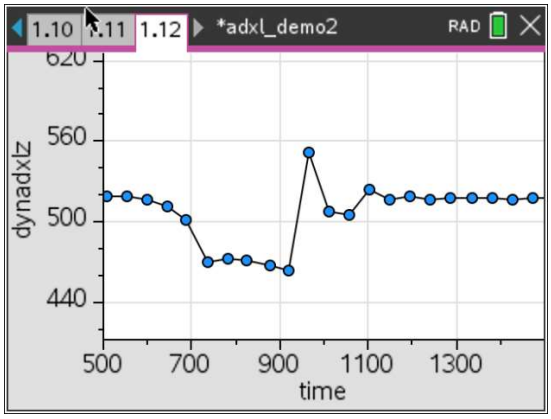
program name	tab #	brief desription
adxl_demo.py	1.2	Code steps through all <get_> methods of the adxl() class, using <show_> methods to build a dashboard on canvas. Proceed with pressing the <esc> key on the keyboard
adxl_speed.py	1.4	Code measures # of function calls per second. Place your own function into line 11. Code works for any TI-Hub function call
adxl_dyn.py	1.6	Code to sample accelerometer at maximum speed, stores results along with time tags into a spreadsheet window. Useful for dynamic accelerometer measurements
adxl_dirtest.py	1.8	Code to demonstrate use of the HMI interface function get_dirxy(). If inclination is <10 degrees, ADC sample rate is displayed. With inclination>10 degrees, key repetition rate is displayed
Calculator window	1.10	supports use of the delvar command to delete cal variable(s), to simulate a fresh start of the adxl() class.
list & spreadsheet window	1.11	shows the calibration variable lists
list & spreadsheet window	1.12	shows the data sampled by the adxl_dyn.py program
statistics window	1.13	shows the data sampled by the adxl_dyn.py program in a graphical form

Table 2 List of demo programs contained in adxl_demo2.tns

While the execution speed of all adxl() methods above level 0 is rather slow due to three sensors being sampled, the single-sensor level 0 functions can be used in dynamic accelerometer applications, like fall tests, acceleration tests, vibration tests, motion sensing, etc. adxl_dyn.py demonstrates how sampling the z-axis sensor at high speed could work. The sensor is sampled at maximum speed, with just the measurement results being stored in a Python list. After sampling has been completed (or aborted with the <esc> key), data is

written to an Nspire variable, to be analyzed in a list & spreadsheet window or displayed in a graphics window.
The following screenshots have been taken with using adxl_dyn.py and should inspire the user for thinking about new exciting projects in a STEM curriculum.

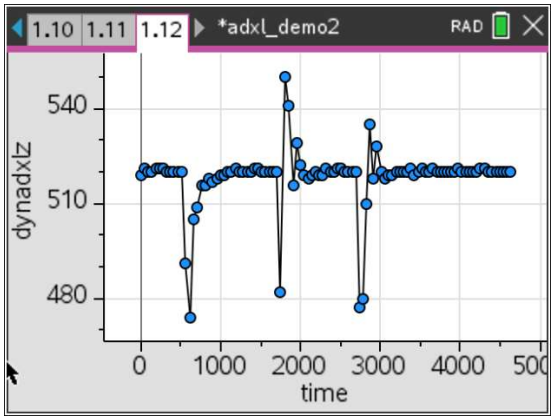
Fall test experiment



The accelerometer sensor had been fixed on top of the TI-Hub, sensor side facing up, using a duct tape. A cushion was placed on the table and the TI-Hub with accelerometer sensor was held at 60cm (24 in) above the cushion. 500 ms after the program was started, the TI-Hub was let go. Figure 8 shows the free-fall and landing-on-cushion part of the data sampled.

Figure 8 Fall test experiment

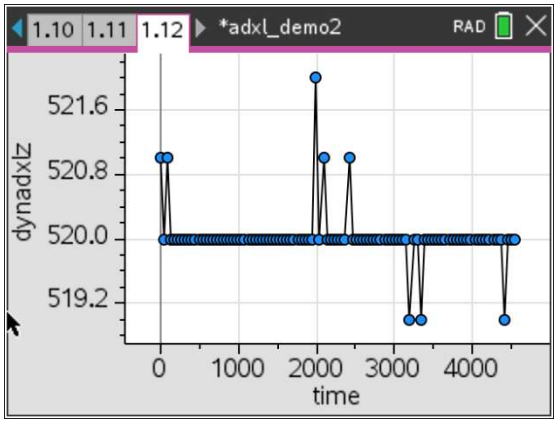
Vibration test experiment



The same sensor-TI-Hub setup as described above was placed on a wooden table. At 1 sec intervals, the table top was beaten with a fist.

Figure 9 Vibration test experiment

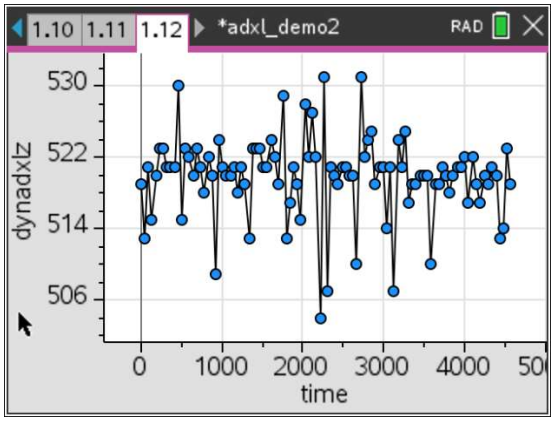
iPhone vibration alarm experiment



The sensor-TI-Hub assembly was placed on top of an iPhone and the Timer Alarm was set to 2 secs. After 2 seconds, the alarm went off, playing the 'Radar' tune, with the vibration motor of the iPhone playing the rhythm in parallel. As the acceleration caused by the vibrating iPhone is rather low, it is important to set analog averaging to 20 at the instantiation of the adxl() class, to filter out high frequency noise.

Figure 10 iPhone vibration alarm experiment

Loudspeaker experiment



An accelerometer can also be used as a (crude) microphone. Here, the sensor-TI-Hub assembly was placed on top of a Bluetooth Loudspeaker playing 'Locomotive Breath' from Jethro Tull. Again, it's important to use the average option at instantiation of the adxl() class, as the accelerations imposed to the sensor are very modest.

Figure 11 Accelerometer acting as a microphone

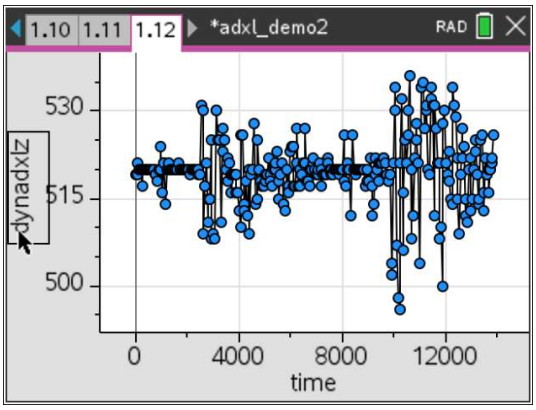


Figure 12 Time intro (0:42) by Pink Floyd

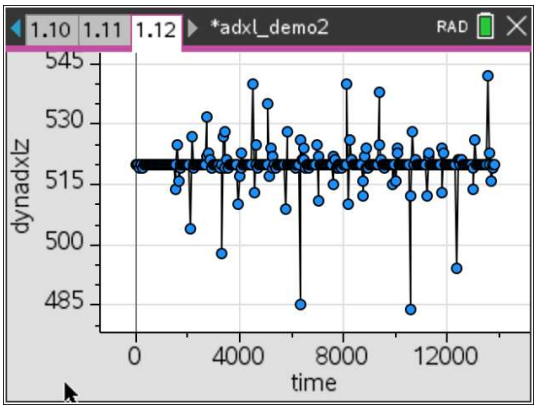


Figure 13 Online Drum Computer [4]

Mobile experiments, using the Nspire CXII Handheld

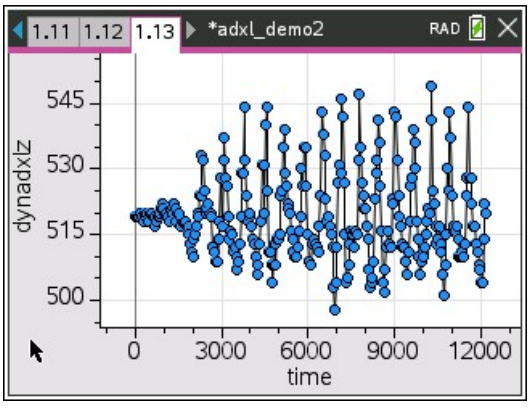


Figure 14 Walking down a stair (15 steps)

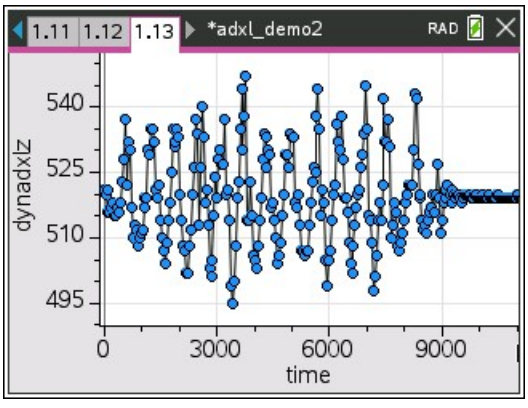


Figure 15 Walking 13 steps in the garden

A lot more exciting mobility experiments can be done, using the sensor-TI-Hub setup with the Nspire CXII. Fig 14 & 15 show two Fitness Watch experiments. Setting the number of samples to 1000 will gain about 40 seconds of recording time, if needed.

Summary:

An accelerometer probably is one of the most versatile sensors available for STEM classes today. Applications span all STEM domains (Science, Technology, Engineering, Mathematics) and numerous experiments can be created which are fun to perform and to learn from for both, teachers and students. The TI-Innovator system with its Nspire CXII Handheld extends the range of experiments to mobile applications. Take the system on the bus, the subway, the train, the car, the bicycle etc. and let students discuss their data recorded.

The ADXL335 Nspire Python driver library shall help T^3 and STEM teachers with providing a solid and versatile foundation for new experiments to come. Feel free to share your projects with the T^3 community!

Sources:

[1] ADXL335 datasheet, Rev B, © 2009,2010 Analog Devices, https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL335.pdf

[2] AN-1057 application note, Rev 0, © 2010 Analog Devices, https://www.analog.com/media/en/technical-documentation/application-notes/AN-1057.pdf

[3] ADXL335 accelerometer https://wiki.seeedstudio.com/Grove-3-Axis_Analog_Accelerometer/

[4] Online Drum Computer https://www.musicca.com/de/drumcomputer?data=%224.0.1-4.2.1-4.4.1-4.6.1-4.8.1-4.10.1-4.12.1-4.14.1-5.4.1-5.12.1-6.0.1-6.8.1-t.4-tmp.90-s.0%22

[5] Hedgehog – a 3D Graphics Library in Python; Berger, Hilbig; T3 Europe Materials Database, Nov 2020 https://resources.t3europe.eu/t3europe-home?resource_id=3129&cHash=129e21e4987c1a110e05c66b83bdd7a2