



Kionix, Inc. USB Demo Board Kit

User's Manual

Jun 7, 2006

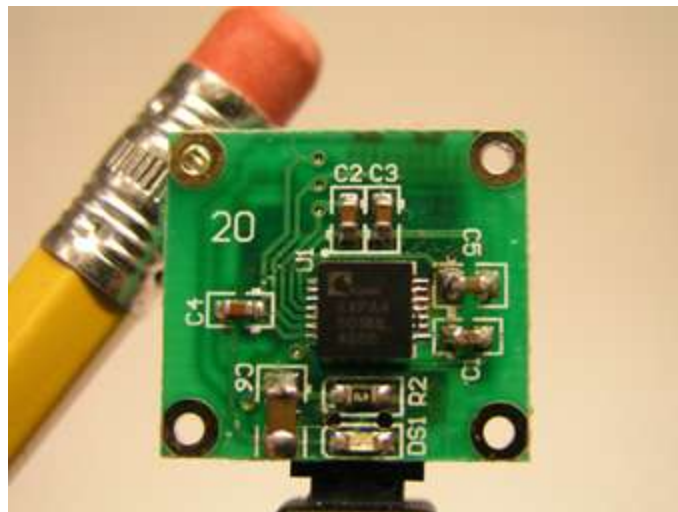


Table of Contents

Kit Contents.....	1
I. Kionix Demonstration Board Technical Overview.....	1
II. Software Installation.....	1
III. Configuration.....	2
IV. Demo Software.....	2
1. Acceleration_Data	3
2. Oscilloscope.....	3
3. Data_Logger.....	3
4. 3D Ball	4
5. Cursor.....	4
6. Freefall.....	4
7. Spaceship.....	5
8. Virtual Light Saber.....	5
9. Theft_Detection.....	6
10. Artificial Horizon.....	6
11. Screen Rotation.....	6
12. Bump Alert.....	6
PC Tilt Games.....	7
Freeware/Shareware/Demos.....	8
Appendix A: Programmer's Manual.....	9
1. Perl API.....	9
2. Communication.....	11
Appendix B: Basic Concepts of Motion.....	11
1. Calculating Velocity and Distance From Acceleration.....	12
2. Calculating Angle of Tilt From Acceleration.....	12
3. Free-fall Detection.....	13
4. Limitations of These Methods.....	13
Appendix C: Algorithm References.....	15
1. Converting From Acceleration to Tilt.....	15
2. Controlling a Rolling Ball by Tilting.....	16
3. Detecting Free-fall.....	18
4. Jolt Detection.....	19

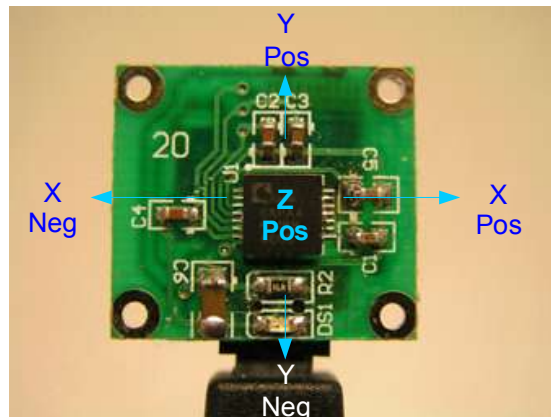
Kit Contents

1. Kionix USB demo board
2. USB A male – mini 4 cable
3. USB A male – mini 4 adapter
4. CD with program files and user's manual
5. Quick start guide



I. Kionix Demonstration Board Technical Overview

- Demo boards are available in 1.5g, 2g, 3g, 5g, 10g and 20g sensitivities.
- The demo board is powered by the USB port. It receives a 5V current, and converts it to the 3.3V Vdd used to power its components.
- The X, Y, and Z axis lines are connected to the analog to digital converters on the USB chip.
- The sensor runs at a bandwidth of 50Hz.
- The demonstration programs read samples at about 250 samples/second.



II. Software Installation

Connect the Demonstration Board to the mini USB connector on the USB cable.

Connect the USB A end of the cable to the USB port on your PC.

Run the “KionixDemos.exe” program from the CD or from the Kionix website
<http://www.kionix.com/Downloads/downloads.htm>

Run the Configuration program from Start Menu -> Kionix Demos to configure and calibrate the sensor. (See “Configuration” below.)

III. Configuration

The configuration program, shown in the figure below, will allow you to configure communication with the device.

- 1. Interface** – The method used to communicate with the device. Currently the method is Gamepad, the USB COM port of the computer.
- 2. Device** – The driver specific to the Kionix part being used. Because the tri-axis accelerometer is used through the gamepad driver, the driver named MiniUSB is accessed.
- 3. Serial Port** – This selects the serial port to which the device is connected. Press “Scan” to limit the list to ports that can actually be accessed. A label to the right of the button will appear showing how many ports were found to be available. Then select the port to which the device is connected from the address pull-down above, and press “Test” to check that the device is connected properly. If it is, a label will appear to the right of the Test button confirming that the test was a success.

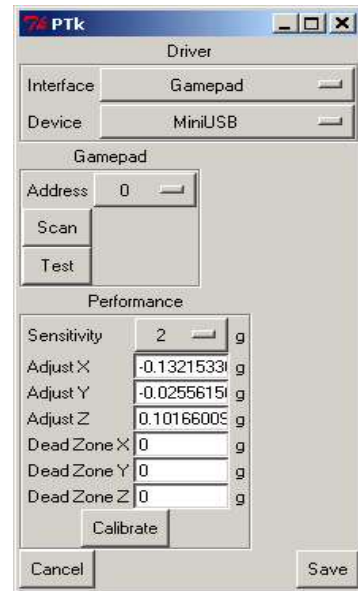


Figure 1 - configure.pl

- 4. Performance** – These boxes show the adjustments made to readings received. You may enter these manually if you wish, but it is easiest to lay the device flat, press “Calibrate” and accept the default settings.

Sensitivity – This is a pull down menu for selecting the sensitivity of the demo board. Sensitivity is set at the factory. You must select the correct g level for your board or the demos will not function properly. The default sensitivity is 2g.

Adjust X/Y/Z – These are the amounts by which each reading on the relevant axis is adjusted, in g's, before it is returned. This is to account for any slight variations in center the device might have.

Dead Zone X/Y/Z – This is the minimum absolute value a reading must reach before it is registered. If it is below this value, it will be read as zero. Having a dead zone can filter out noise, but at the cost of losing real readings if they are very small. The default of 0 is perfect for the demonstration programs.

When you are finished adjusting the settings press “Save” to save and exit the program.

IV. Demo Software

Various demos are included with this distribution to demonstrate the capabilities of the tri axis accelerometer and to get you started writing your own applications. The demo programs are written in the Perl scripting language, which is an easy-to-learn and extensible language for anything from simple tasks to complex programs and web applications. For more information on Perl, visit www.perl.org.

Several of these demos also use the OpenGL graphics system. For more information on OpenGL, visit www.opengl.org.

1. Acceleration_Data

The Acceleration_Data demo shows the current readings of the device in the most raw form as is possible. For those interested in pure data, a device could be attached to a piece of memory to store a running log of all readings. The data could be retrieved later for analysis. The actual collection and processing of accelerometer data does not require very much processor power, and the sample rate of the device is very high, so an accelerometer can be added to almost any application while creating minimal overhead.

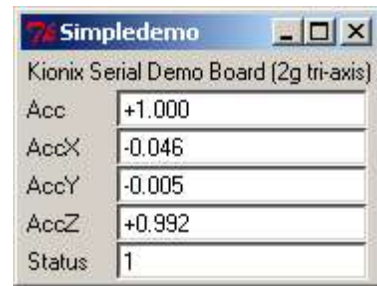


Figure 1 - Acceleration_Data

2. Oscilloscope

The Virtual Oscilloscope demo graphs data in a simple visual format to give the viewer a general idea of the patterns present in the motion of the device. By recognizing these patterns, such a device could become integral in several kinds of applications. A free-fall detector could be used to protect important data by spinning down a hard drive before it hits the ground. A jolt detector could create a record of package mishandling during shipping. A vibration detector could be placed on a piece of machinery to issue an alert if the pattern of movement changes significantly, indicating the possible need for maintenance.

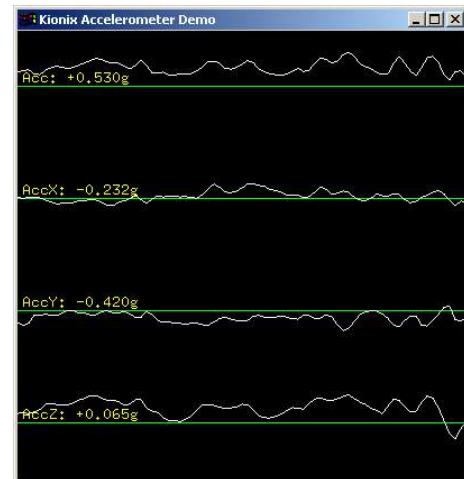


Figure 2 - Oscilloscope

3. Data_Logger

The data logger takes a constant stream of readings from the device and graphs them in real-time to the screen. Sampling is done by setting the time you wish to sample, the rate at which you wish to sample, and pressing "Go". Note that the sample rate you specify is the target sample rate. If you specify a very high number, the program may read less samples than you expect. If you wish to save the data you collected, press "Save" to write the data to a CSV (comma-separated values) file. You can then use Excel, MatLab, etc. to analyze and graph the data.

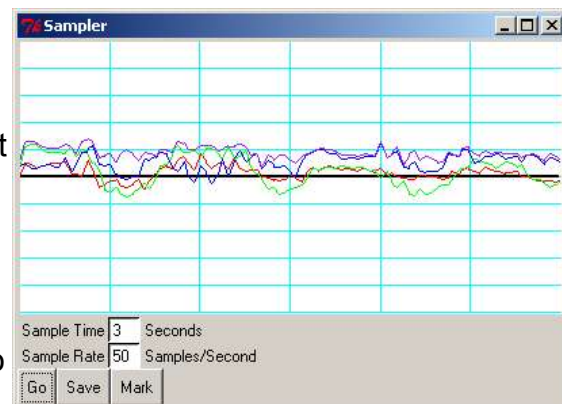


Figure 3 - Date_Logger

4. 3D Ball

The 3D Ball demo is a simple demonstration of accelerometer-based controls in video games. By tilting the device, the user is able to roll the ball around the board and roll over the red target. Additionally, a strong bump applied to the z axis of the device will cause the ball to bounce into the air. A game development team could use this unique control scheme to add a new level of playability to games like the classic Marble Madness by Atari Games, or to create an entirely new game of their own. Other games which this could be used with include motorcycle, racing, snowboarding, skateboarding, and jet fighter games.



Figure 4 - 3D Ball

5. Cursor

The mouse cursor is a simple demonstration of the idea of a tilt mouse. After opening the program, the mouse cursor can be moved by tipping the device. To end this program press CTRL C.

6. Freefall

The Free-fall Detector demo is an example of the free-fall detection algorithm described in Appendix B. It registers when it has been in free-fall for half a foot, then reports how far it fell when it reaches the bottom of its fall. Logging can be temporarily paused with the "Stop" button, or the log can be saved to a CSV (comma-separated values) file with the "Save" button. For more details on the algorithm, see Free-fall Detection in Appendix B.

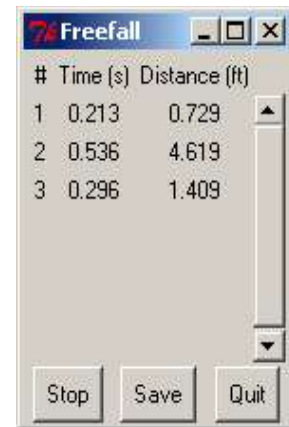


Figure 6 - Freefall

7. Spaceship

This Perl program demonstrates how an accelerometer can be used as a game controller. The tilt action is translated to control the movements of a virtual spaceship. Avoid the asteroids. If you crash into an asteroid the “game” will end and need to be closed and restarted to begin again.



Figure 7 - Spaceship

8. Virtual Light Saber

The virtual light saber shows the unique values available from a tri axis accelerometer. The light saber is activated by picking up (or moving) the demo board. As you move the demo board the light saber will mimick your moves. Ten seconds of inactivity will cause the saber to “turn off”. Moving the demo board will again activate the light saber.

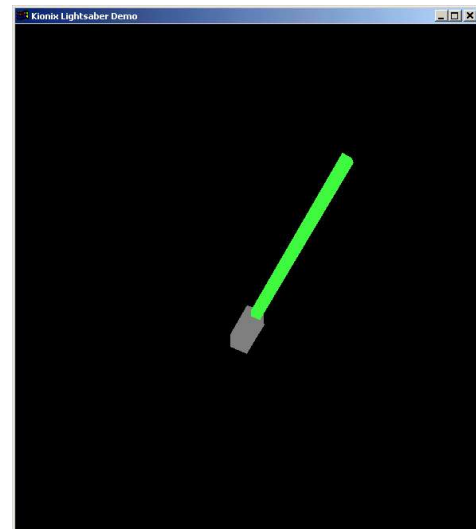


Figure 8 – Virtual Light Saber

Demo Software for the Dongle

The following four software demos work best when the demo board is plugged directly into the USB port of a notebook computer using the mini4 – male A USB adapter. (They still work with the board on the USB cable)

9. Theft_Detection

Movement detection for laptops. Plug the dongle into the USB port of the laptop. The accelerometer is set to detect tilt. This allows the user to type and move the laptop but even a small tilt will sound an alarm. The sensitivity can be changed and the password (for unlocking the alarm) can be set. The Lock button will arm the system so that any tilt will cause an alarm to sound through the computers speakers. Type your password to Un-arm the program and change the sensitivity. The default password is testpass. If you forget the password, close the program and start again.



Figure 9 – Theft Detection

10. Artificial Horizon

This demo is meant to show how an accelerometer can be used to sense tilt and pitch the way an artificial horizon shows roll and pitch in an airplane.

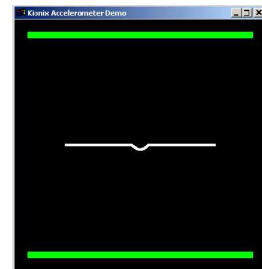


Figure 10 - Horizon

11. Screen Rotation

Similar to artificial horizon, this demo shows how an accelerometer sensing roll can keep an image or an object flat and level on the screen.

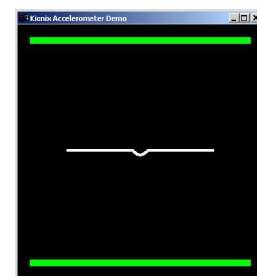


Figure 11 – Screen Rotation

12. Bump Alert

Detects even the slightest vibration. Plug the dongle into the USB port of the laptop and begin the program. The accelerometer is set to detect any motion. Even the slightest vibration will trigger the alarm.



Figure 12 – Bump Alert

PC Tilt Games

A collection of 2D games that have been modified to accept joypad inputs as well as key strokes. Windows identifies the USB demo board as a gamepad, with the X, Y and Z axis identified as joystick inputs. This makes the USB demo board a motion enabled controller. Tilt left to turn left, tilt right to turn right. Tipping it forward will move your player forward and tipping it back will move your player back. Several of these games were created with [GameMaker 6](http://www.gamemaker.nl/). (<http://www.gamemaker.nl/>)

1. 1945

2D scrolling game. X and Y axis move the plane. Use the space bar to shoot.

2. Asteroids

Asteroids game by Mark Overmars. Tipping the demo board forward (Y axis) will move the ship. There is no backward. Rotate the ship 180° and tip forward to slow down. Use the space bar to shoot.

3. Breakout

Standard 2D breakout game. Tipping the demo board left and right will move the bat.

4. Jump

Platform game. Use Y to jump.

5. Labyrinth

Maze game written by Keith Epstein. Hold the demo board with the Y axis up and the led facing toward the monitor. This game uses the X and Z axis to control the movement of the ball.

6. Pacman

GameMaker6 demo. This is a very basic version of PacMan.

7. Plop

Written by Keith Epstein. Hold the demo board with the Y axis up and the led facing toward the monitor. This game uses the X and Z axis to control the movement of the bear. Bounce the demo board to drop bombs.

8. roll the ball

Written by Keith Epstein. Hold the demo board with the Y axis up and the led facing toward the monitor. Hit the black ball 10 times, before your time runs out, to move to the next level.

9. Rotate

Written by Keith Epstein. This game uses all three axis of the accelerometer. Match the numbers by turning the board in all three axis (in the correct direction). This is unlike any game you have ever played.

10. screen_rotation

Demo by Keith Epstein.

11. Street race

Scrolling game by Mark Overmars. X and Y axis control the car.

Freeware/Shareware/Demos

12. Connman_150

A freeware pacman game from [ConnectiX](http://www.connectix.gr/). (<http://www.connectix.gr/>) Use X and Y axis to move the player. Freeware

13. FatHead

A [Groovy Lime](http://groovylime.com/) (<http://groovylime.com/>) platform game by Trevor Simpson. X axis moves the player left and right. Y axis jumps the player. Freeware.

14. Hamsterball

Demo game available from Raptisoft. (<http://www.raptisoft.com/hamsterball.shtml>)

Appendix A: Programmer's Manual

1. Perl API

The Perl API for the Kionix demo board provides an easy, object-oriented interface for a Perl programmer to access acceleration data. It is invoked in much the same way as any other Perl module, and returns an object which can be used to interact with the sensor.

Usage

Create a new Sensor object from scratch:

```
use Sensor;
my $Sensor = Sensor->new('Serial', 'TriAxis2g', 'Port' =>
'COM1');
$Sensor->open or warn "Failed to open sensor";
```

Create a Sensor object from an existing configuration file:

```
use Sensor;
my $Sensor = Sensor->newFromConfig('default.ini');
$Sensor->open or warn "Failed to open sensor";
```

Sensor Methods

The following methods are universal to all Sensor objects.

Sensor->new(\$Interface, \$Device, %Options)

Create a new object based on the interface driver *\$Interface* and the device driver *\$Driver*. The *%Options* hash will be used to override default options if it is included.

Sensor->newFromConfig(\$File, %Override)

Create a new object, reading configuration options from *\$File*. The *%Override* hash can be used to override options read from the configuration file.

\$Sensor->loadConfig(\$File)

Load configuration options from *\$File* and apply them to *\$Sensor*.

\$Sensor->configure(\$Key, \$Value)

Sets a configuration option for the object. Valid options are detailed in "Configuration Options" below.

\$Sensor->open()

Opens the sensor and prepares it for reading. Returns true or false, depending on if the open was successful.

\$Sensor->close()

Closes the sensor. Returns true or false, depending on if the close was successful.

\$Sensor->canMeasure()

\$Sensor->canMeasure(\$Reading)

\$Sensor->canMeasure(@Readings)

When called with no parameters, *canMeasure* returns a list of all the readings a

sensor can return. When called with one parameter, `canMeasure` returns a true or false indicating if that reading can be returned. When called with multiple parameters, `canMeasure` returns a true or false value indicating if all of the readings can be returned.

`$Sensor->status()`

Returns the status of the device, 1 indicating a ready status and 0 indicating a bad status. If there is no way to determine the status of the device, this command returns `undef`.

`$Sensor->$Reading()`

Return the desired reading, as indicated by `$Reading`. `$Reading` can be any of the measurements returned by `$Sensor->canMeasure()`. For an example, see “Measurements Available From a TriAxis2g Sensor” below.

Measurements Available From a TriAxis2g Sensor

`$Sensor->AccX()`

`$Sensor->AccY()`

`$Sensor->AccZ()`

Returns acceleration on the X, Y, or Z axis. This value is in g's, the acceleration due to the Earth's gravity. That is, $1g = 9.8m/s^2$. For tilt calculations, $1g = 90^\circ$. More information on using the values returned by an accelerometer can be found in Appendix B: Basic Concepts of Motion.

`$Sensor->Acc()`

Unlike the other readings, `Acc` is a calculated value. It is based on `AccX`, `AccY`, and `AccZ`, using the Pythagorean Theorem in three dimensions¹. It is a measurement of the magnitude of the acceleration currently being applied to the accelerometer, without the direction. It is useful for applications such as jolt and free fall detection.

`$Sensor->AccAll()`

Returns an array of the X, Y, and Z accelerations. Note that because all three of these are sampled anyway each time any one reading is taken, using this method is three times faster than calling `AccX`, `AccY`, and `AccZ` in succession.

Configuration Options

The following options can be passed to the `configure()` method.

`Port`

This option is accepted by any object using the Serial interface driver. It specifies which COM port the sensor is plugged into.

`AdjustX`, `AdjustY`, `AdjustZ`

These options are accepted by sensors which can return `AccX`, `AccY`, and `AccZ` respectively. They are the amount by which the reading is adjusted to account for slight offsets in the “zero” position of the sensor. These are usually set during a calibration process, like the one in the `configure.pl` example script.

¹ Specifically, the calculation used to get overall acceleration is:

$$a = \sqrt{x^2 + y^2 + z^2}$$

DeadZoneX, DeadZoneY, DeadZoneZ

These options are accepted by sensors which can return AccX, AccY, and AccZ respectively. They specify a minimum absolute value above which each reading must be. If the reading is within the dead zone for the axis, it is simply returned as zero. These are usually set to filter out noise when the device is level.

2. Communication

For those who wish to communicate with the device in their own program or programming language of choice, this section details how communication with the demo board takes place.

Commands

The following commands can be issued to the demo board once communication has been established.

Code	Description	Return
X	Get X-axis acceleration.	X (2-byte integer)
Y	Get Y-axis acceleration.	Y (2-byte integer)
Z	Get Z-axis acceleration.	Z (2-byte integer)
A	Return all three axes.	XYZ (three 2-byte integers)
T	Echo a 'T', useful for checking board status.	T (1-byte character 0x54)

Return Values

The value returned for an axis reading is stored as a 2-byte integer in little-endian (least significant byte first) order. That is, the value can be calculated with the following equation:

$$Reading = FirstByte + (SecondByte * 256)$$

The resulting value represents a number on an arbitrary scale set by the analog-to-digital converter taking the reading. To convert this value to millivolts, the following equation is used:

$$Millivolts = (Reading * 1000) / 1241.2121$$

Finally, to change the reading in millivolts into the acceleration in g's, subtract the 0g offset or center (usually Vdd/2, or 1650mv for a 3.3V part). Then, divide by the sensitivity rating of the part in mv/g. This demo board uses a device with a sensitivity of 660 mv/g, thus:

$$Acceleration = (Millivolts - Center) / 660$$

The resulting number is the acceleration value returned by the sensor in g's.

Appendix B: Basic Concepts of Motion

The concepts discussed in this section are widely available and are a part of any Physics course, but they have been reproduced here both as a refresher and as a quick

reference useful to anyone working with accelerometer data. There is also a discussion of the limitations of these methods when used to determine the position or tilt of a device using a tri-axis accelerometer.

1. Calculating Velocity and Distance From Acceleration

Given an acceleration (a) and a period of time (t), it is possible to calculate the change in velocity during the relevant time period. If the original velocity is also available, the velocity at the end of the time period and the change in position over the time period can be calculated. Lastly, if the original position is available, the position at the end of the time period can be calculated. This can be done according to the steps below.

Calculating Change in Velocity

Given a, the acceleration applied on the axis.

Given t, the time period for which the acceleration was applied.

$$\Delta v = at$$

Results in Δv , the change in velocity during the time period.

Calculating Final Velocity

Given Δv from the previous equation.

Given v_0 , the velocity at the start of the time period.

$$v = v_0 + \Delta v$$

Results in v, the velocity at the end of the time period.

Calculating Change in Distance

Given v_0 , v, and t from previous equations.

$$\Delta d = \frac{(v_0 + v)}{2} * t$$

Results in Δd , the change in distance during the time period.

Calculating Final Distance

Given Δd from the previous equation.

Given d_0 , the distance at the start of the time period.

$$d = d_0 + \Delta d$$

Results in d, the distance at the end of the time period.

Conclusion

At the end of these equations, we have both the final velocity (v) and the final distance (d). This information can be used to determine the same values in the next time period, resulting in a continuous flow of acceleration, velocity, and distance data.

2. Calculating Angle of Tilt From Acceleration

The acceleration data can also be used to find how far the device is tilted. This can be done because the Earth is always pulling on the device with 1g of acceleration. If the device is put on a flat surface and is completely still, all of that acceleration is on the Z

axis. The acceleration on the X and Y axes will be zero. If the object is put on its side, whichever axis is pointed toward the earth will read 1g. Getting the angle from the readings on an axis can be done two different ways. The simplest way, although it is not very exact, is to multiply the acceleration on the axis by 90:

$$\text{rotation} = \text{acceleration} * 90$$

For a much more exact value, take the inverse sine of the acceleration:

$$\text{rotation} = \text{asin}(\text{acceleration})$$

For more information see *Kionix Application Note AN0005 on Tilt Sensing*

(<http://www.kionix.com/App-Notes/AN005%20KXM52%20Tilt%20Sensing%20040916.pdf>)

3. Free-fall Detection

A tri-axis accelerometer can be used to detect when an object is in free-fall. The first step is to calculate the overall magnitude of the acceleration being applied to the object. This is done with the Pythagorean Theorem:

$$a = \sqrt{x^2 + y^2 + z^2}$$

If the object is in free-fall, the value will be very close to zero. Depending on the rotation of the object, however, it may be a somewhat higher number (see “Limitations of These Methods” below for details). The distance the object fell can be calculated by using gravity ($g = 9.8 \text{ m/s}^2$) as acceleration (a) in the equation for calculating distance (d) from acceleration (a) and time (t):

$$d = .5 * at^2$$

Note that this calculation will produce the wrong number if the object has been thrown upward or downward. Again, see “Limitations of These Methods” below for more details.

4. Limitations of These Methods

While these equations are effective for many applications, there are several limitations and “gotchas” that one has to watch for when using the data returned by the accelerometer.

Noise In Acceleration -> Velocity -> Distance Calculations

All measurements contain a small amount of background noise. Unfortunately, in an acceleration reading, noise can disrupt the apparent velocity of the device. This difference will become more apparent over time as the “phantom” velocity pushes the distance measurements farther and farther from reality. This change will be relatively slow, however, due to the low-noise nature of the device. The difference can also be made less visible by taking an average of several readings instead of acting on each reading as it arrives. A dead zone can also be implemented, risking the possible loss of the actual readings if they are very small.

Differentiating Between Tilt and Motion

While the ability to measure either motion or tilt is very useful, it comes with the unfortunate disadvantage of not being able to easily differentiate between the two. The

z-axis of this device may make this distinction possible on the x and y axes by watching the effect of the acceleration in question on the z axis, and the acquired data could be applied to future demos. Of course, associated equations will also be made available.

Rotation and Center of Mass in Free-fall Detection

Centripetal acceleration, present if the object is rotating, can throw off free-fall detection by causing the overall acceleration of the object to be significantly higher than zero even when the object is actually in free-fall. This effect can be reduced by putting the device in a heavy casing, positioning the accelerometer as close as possible to the device's center of mass, and allowing a range of values to represent free-fall. 0.0 to 0.5g is usually a safe range to use.

Free-fall Time When an Object is Thrown

An object is in free-fall as soon as no forces except for gravity are acting on it. That means that if you throw the device upward, it will be in free-fall even when it is "falling" upward. Similarly, being thrown downward will shorten the time the object is in free-fall before it hits the ground. Either of these events will throw off the equation presented in "Free-fall Detection" above, which assumes that the object was released into free-fall with a velocity of zero. This limitation is not an issue for applications such as hard drive protection, as they are only concerned with the fact that the object has been dropped, but can adversely affect applications in which the distance the object fell is important. In these cases, it may be better to use an accelerometer with a higher range and implement impact detection instead of using a low-range accelerometer for detecting free-fall.

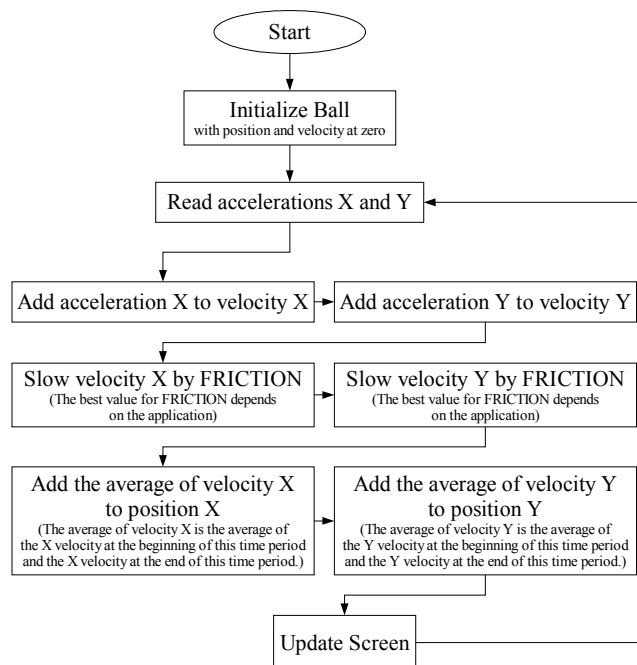
Appendix C: Algorithm References

1. Converting From Acceleration to Tilt

$$\phi = \arctan\left(\frac{X}{\sqrt{Y^2 + Z^2}}\right)$$

$$\varrho = \arctan\left(\frac{Y}{\sqrt{X^2 + Z^2}}\right)$$

2. Controlling a Rolling Ball by Tilting



```
ball.positionX = 0
ball.positionY = 0
ball.velocityX = 0
ball.velocityY = 0
```

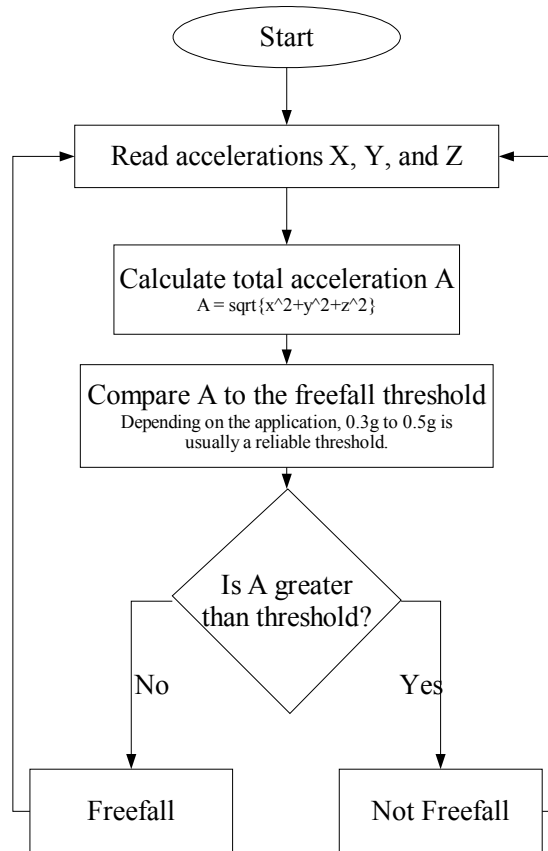
```
while running
    (ACCX, ACCY, ACCZ) = sensor.allreadings
    OLDVELX = ball.velocityX
    OLDVELY = ball.velocityY
    // Add acceleration to velocity
    ball.velocityX = ball.velocityX + ACCX
    ball.velocityY = ball.velocityY + ACCY
    // For a better feel, the following implements
    // friction. The best value for FRICTION depends on
```

```

// the application.
if absolute_value(ball.velocityX) < FRICTION then
    ball.velocityX = 0
else
    if ball.velocityX > 0 then
        ball.velocityX = ball.velocityX - FRICTION
    else
        ball.velocityX = ball.velocityX + FRICTION
    end if
end if
if absolute_value(ball.velocityY) < FRICTION then
    ball.velocityY = 0
else
    if ball.velocityY > 0 then
        ball.velocityY = ball.velocityY - FRICTION
    else
        ball.velocityY = ball.velocityY + FRICTION
    end if
end if
// Add average velocity to position
ball.positionX = ball.positionX + average(
    ball.velocityX, OLDVELX)
ball.positionY = ball.positionY + average(
    ball.velocityY, OLDVELY)
end while

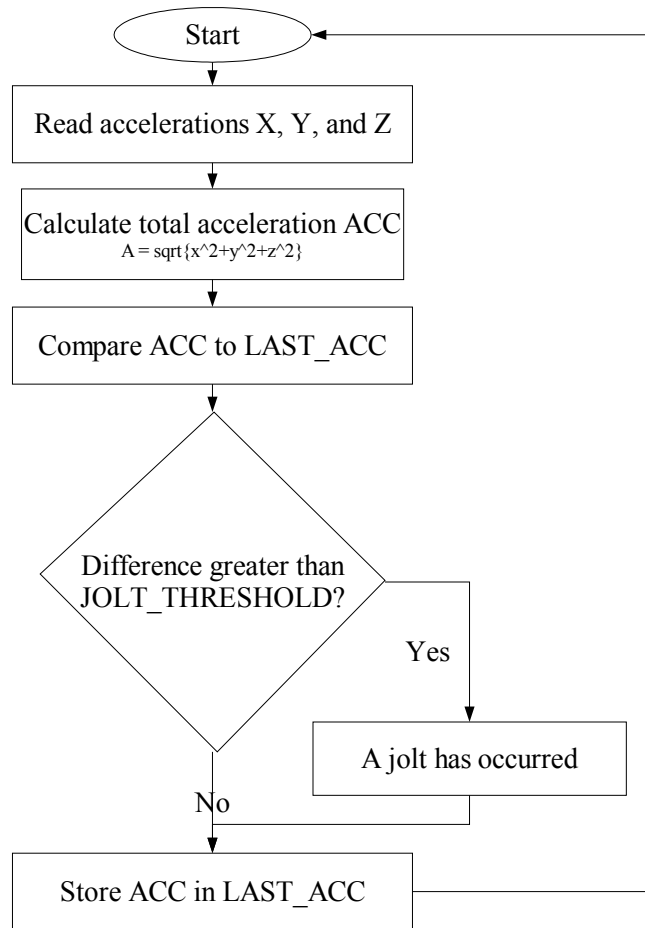
```

3. Detecting Free-fall



```
FREEFALL = 0.3 g // Threshold under which the object is
                // considered to be in free-fall
IN_FREEFALL = false // Stores whether or not the device is
                   // falling
while running
    (ACCX, ACCY, ACCZ) = sensor.allreadings
    TOTAL_ACC = square_root(ACCX ^ 2 + ACCY ^ 2 + ACCZ ^ 2)
    if TOTAL_ACC < FREEFALL then
        IN_FREEFALL = true
    else
        IN_FREEFALL = false
    end if
end while
```

4. Jolt Detection



```
JOLT_THRESHOLD = 1 g // Amount by which the overall  
                    // acceleration reading must change  
                    // between readings to be considered a  
                    // jolt
```

```
(ACCX, ACCY, ACCZ) = sensor.allreadings  
LAST_ACC = square_root(ACCX ^ 2 + ACCY ^ 2 + ACCZ ^ 2)
```

```
while running  
    (ACCX, ACCY, ACCZ) = sensor.allreadings  
    ACC = square_root(ACCX ^ 2 + ACCY ^ 2 + ACCZ ^ 2)  
    JOLT = absolute_value(ACC - LAST_ACC)  
    if JOLT >= JOLT_THRESHOLD then  
        // A jolt has occurred  
    end if  
    LAST_ACC = ACC  
end while
```