The use of motion in mobile device game mechanics

A study of accelerometers on the Nokia N95 using Python

Claire A Blackshaw
s0027525

May 26, 2008

BLANK

Contents

Code Listings

1 Module Details

1.1 Specific learning outcomes

1. *Analyse and use Python to develop a simple application.*
   This step is allow for the study of the platform and tools which will be used throughout the module. Learning Python to the degree required for this module. Setting up the Nokia N95 to run Python applications and deployment method. Learning the Python on the S60 3rd edition libraries.

2. *Analyse and use the data captured by a motion device.*
   Find an appropriate method to capture the data from the motion device. Format the data into a usable structure.

3. *Define and use the motion device to recognise gestures.*
   Examine the limitations of the device. Supply a proof for what motion recognition is possible.

4. *Develop a simple application using skills gained.*
   Using the previous knowledge of Python, n95 Platform and Motion study to develop a test application to demonstrate proof of concept.

1.2 Deliverables, including milestones

1. Learn python on S60 and produce simple Hello-World application

2. Learn the accelerometer API and produce Hello-Shake application

3. Analyse the functionality of the N95 accelerometer and describe range of possible motions / gestures that could be detected

4. Produce application that recognises a range of discrete gestures

5. Report describing progress on key learning outcomes

1.3 Support Materials

- Mobile Python: Rapid Prototyping of Application on the Mobile Platform by Jűrgen Scheible & Ville Tullos
- Nokia Support forums (Website)
  `http://discussion.forum.nokia.com/`
- Python 101 cheat sheet (Website)
  `http://www-128.ibm.com/developerworks/library/l-cheatsheet3.html`
- Python 2.4 Quick Reference (Website)
  `http://rgruet.free.fr/PQR24/PQR2.4.html`
- Py60 Documentation (Website)
  `http://sourceforge.net/projects/pys60`
- Nokia Activity Monitor (Website)
  `http://research.nokia.com/projects/activity\_monitor`
- Python in Latex (Website)
  `http://ubuntuforums.org/showthread.php?t=331602`

2 Choice of language and platform

2.1 Mobile Device

The choice of a mobile device was made as it is a highly limited device and has a natural facility for motion. The desktop machine or laptop is not inherently associated with motion. The device also has a very restrictive input method through traditional means due to the surface area.

The input method that we investigated first were motion dictation through image tracking. A method developed by Dr Joe Faith on the Symbian platform. The method uses the camera phone to detect motion. The method required the use of Symbian. upon further study Symbian was found to be an impractical choice of language for rapid prototyping and required a large degree of study to get started.

The decision was made to avoid Symbian due to these high overheads. There were also concerns about Symbian's basic design principles of avoiding phone lock would interfere with the real-time nature of motion input.

The next platform we examined was BREW (Binary Runtime Environment for Wireless). A platform which is written in C++ and designed to be optimised for games. The platform looked extremely appealing however limitations surfaced. BREW had s strict signing process which was expensive and was only supported a limited amount of US carriers. So even if we could have received an academic licence the platform was not supported by any carries in the UK.

Openmoko Linux was an appealing choice but again the very small subset of phones which support the platform eliminated it. The iPhone OS was not available at the start of this project.

We then came back to Symbian due to its popularity but we looked at higher-level applications. The Python interrupter then came to our attention. It is a widely supported platform. Python converts extremely easily and is acknowledge as a powerful rapid prototype language.

At this same time some Python apps on the Nokia N95 were discovered on some sites. Nokia had released some bindings for their accelerometer in the Nokia N95. Upon further examination the Nokia accelerometer appeared to be fully functional and thanks to the Wii-mote open source community several resources were available on the topic of accelerometers.

While Symbian is a widely distributed OS the use of accelerometers is still isolated to a handful of phones. The final choice while it appears limited is using a widespread OS and a very adaptable language. The use of accelerometers is also on the increase so we have selected this configuration on their assumption their use will become more widespread.

3 Running Python on the N95

The first attempt before receiving the physical device was an attempt to use the Symbian emulator. Symbian had only recently been converted from 2nd to 3rd edition. The emulator was openly admitted to be faulty. After several failed install the program constantly crashed on loading, most sources pointed to a dependency problem. After some discussion it was decided the emulator would add no value to the project and we awaited the physical device.

The first hurdle was installing a signed interrupter so we could access the low level functionality. With the help of the resources and Py60 [1]. Once Py60 was installed I examined the source of the supplied applications.

All my own applications have to run off a memory stick to comply with internal security.

The first step was a classic Hello World

Code Listing 1: Hello World on N95

```
import appuifw
name = appuifw.query(u"Type your Name:", "text")
appuifw.note(u"Hello World! Greetings from " + str(name), "info")
```

[1] see Support Materials

While that was deceptively easy I then went through the various sound, input and graphics libraries to see what relevant tools I would be needing. Here is an example of a simple program which better utilised program flow and the libraries.

Code Listing 2: Greet Bot

```
import appuifw, e32, audio, key_codes

def greet():
    appuifw.note(u"Hello World!")
    audio.say(u"Hi!")

def love():
    appuifw.note(u"I love you!")
    audio.say(u"I love you!")

def hate():
    appuifw.note(u"I hate you!")
    audio.say(u"I hate you!")

def bye():
    appuifw.note(u"Goodbye!")
    audio.say(u"Bye")

def quit():
    print "Greeting Robot Shutdown"
    app_lock.signal()

canvas = appuifw.Canvas()
appuifw.app.body = canvas

canvas.bind(key_codes.EKey4, greet)
canvas.bind(key_codes.EKey2, love)
canvas.bind(key_codes.EKey8, hate)
canvas.bind(key_codes.EKey6, bye)

appuifw.app.exit_key_handler = quit
appuifw.app.title = u"GreetBot"
appuifw.app.menu = [(u"Greet", greet),
                    (u"Converse",(
                      (u"Love",love),
                      (u"Hate",hate))),
                    (u"Bye",bye)]

print "Greeting Robot Ready"
app_lock = e32.Ao_lock()
app_lock.wait()
```

While this was functional what we really needed to expose was the accelerometer. The device was a secure device only a signed application could access. Fortunately a bunch of Nokia developers have released aXYZ and the Activity monitor [2]. This allowed us to read an up to date reading directly from the accelerometer.

Code Listing 3: Shake Bot

```
import appuifw, e32, axyz, graphics

WHITE = (255,255,255)
RED   = (255,  0,  0)
GREEN = (  0,255,  0)
BLUE  = (  0,  0,255)

acl = [1,1,1]
running = True

def read_xyz(x, y, z):
    global acl
    acl = [x, y, z]
    handle_redraw(None)

def handle_redraw(rect):
    if img:
        img.clear(BLUE)
        img.text((10,20), u"X: %d" % (acl[0]), fill = WHITE)
        img.text((10,40), u"Y: %d" % (acl[1]), fill = WHITE)
        img.text((10,60), u"Z: %d" % (acl[2]), fill = WHITE)
        canvas.blit(img)

def handle_event(event):
    if event["type"] == appuifw.EEvent.KeyDown:
        quit()

def quit():
    global running
    running = False

## Setup Globals
acl = [0,0,0]

## Setup Image
img = None
canvas = appuifw.Canvas(redraw_callback = handle_redraw,\
                        event_callback = handle_event)

w, h = canvas.size
img = graphics.Image.new((w, h))
img.clear(BLUE)

## Connect to Sensor
axyz.connect(read_xyz)
```

[2] Thanks to cyke64

```
## Setup App
appuifw.app.title = u"N95 accelerometer"
appuifw.app.screen = "large"
appuifw.body = canvas
appuifw.app.exit_key_handler = quit

running = True
## Game Loop
while running:
    handle_redraw(None)
    e32.ao_yield()

## Exit
axyz.disconnect()
print "Safely Exited"
```

Once this application ad a version which save the data to a file was available we could truly begin examining the device and its limitations.

4 Technical Range of Motion

4.1 Technical Device

> That 3D accelerometer inside N95 , N93i or N82 is from ST-Microelectronics (type LIS302DL). The European chip maker ST-Microelectronics (STM)has also supplied the motion-control chip for the Nintendo Wii console's controller. This chip is built around a technology known as Micro Electro-Mechanical Systems (MEMS). Essentially chips with tiny moving parts like gears. [3]

Upon further study of the LIS302DL from ST-Microelectronics we can immediately glean some useful information. The sensor can be set to ± 2 or ± 8gravities. The resolution however remains roughly the same. In the case of the Nokia N95 they have set the device to $\pm 2g$however the sample rate is rather low for a real-time application.

aXYZ is outputting a simple signed 8bit number per axis giving us the range of -128 to 127. Now assuming the fidelity is perfect and each number is a significant measurement that means the smallest motion we can measure is approximately 0.01575g. Now well this seems highly precise a high level of noise is introduced as shown by the recordings on page 12.

Now that the technical device has been examined we can examine the output produced.

[3] http://wiki.forum.nokia.com/index.php/S60_Sensor_API

4.2 Recorded Data

The data recorded below is in the following format, Blue is X, Red is Y, Green is Z. They have been adjusted so the aspect ratio is as close as possible but some variance is present.

The action chosen to record is a few variants of an over the should throw as done by a right handed person. The phone is held so when the throwers arm is in front of them the phone is held level pointing the screen towards the sky.

The to demonstrate orientation the device is placed on a flat service and rotated.

The second set of actions is describe in the motion analysis section.

4.2.1 Problems and Flaws Observed

Noise

A high degree of noise exists so smoothing of the data set is essential. This noise is present even when the phone is at rest. This degree of noise eliminates fine gesture recognition.

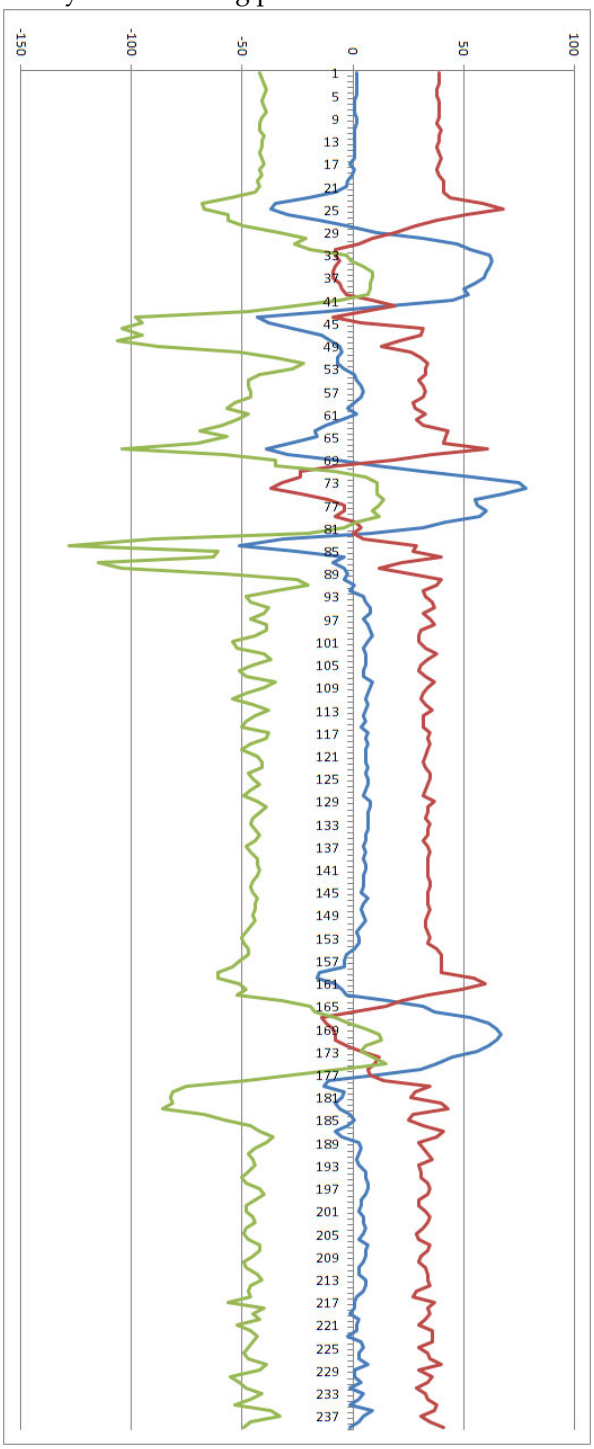
Limited Range

Two gravities is a very small range of for human motion. This speed cap limits the amount of large movements that can be detected.

As will become apparent in later discussion the use of such a low threshold for the device introduces some serious limitations. A realistic upper limit for a motion sensor device would be between 10-20 gravities. This would avoid the loss of data to ”peaks”, situations where device is experiencing forces above its 2g limit. In this process you lose information.

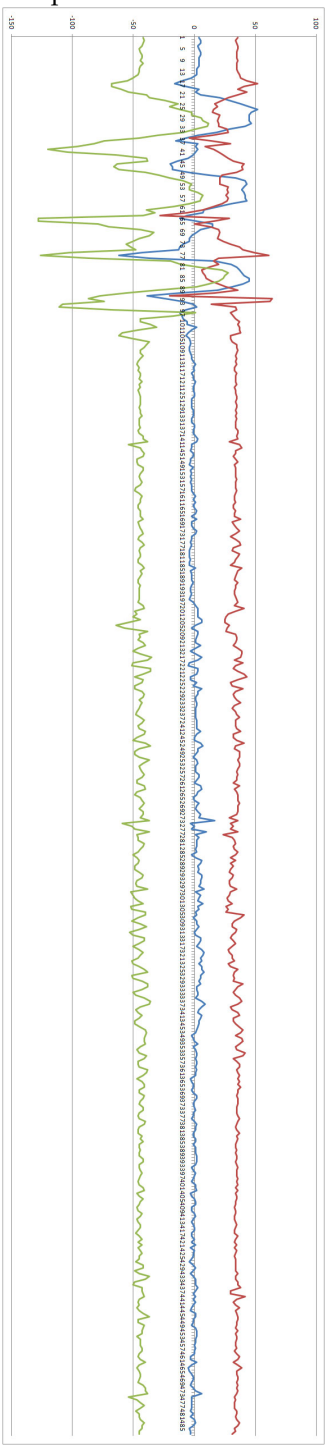
4.2.2 Small Shake back and forward

The phone is held in the default position screen facing up towards the sky. Several very small up down movements are followed by holding the phone steady in the starting position.



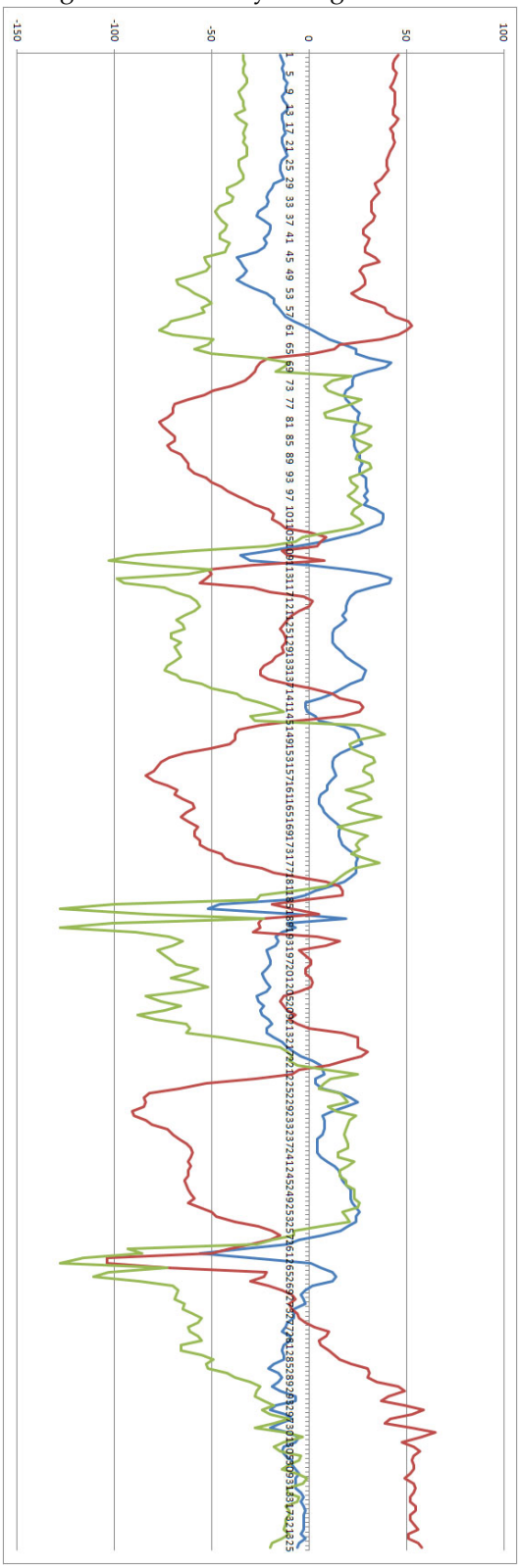
4.2.3 Medium Swing over shoulder

The phone is swung from over the shoulder to the default position the action is repeated



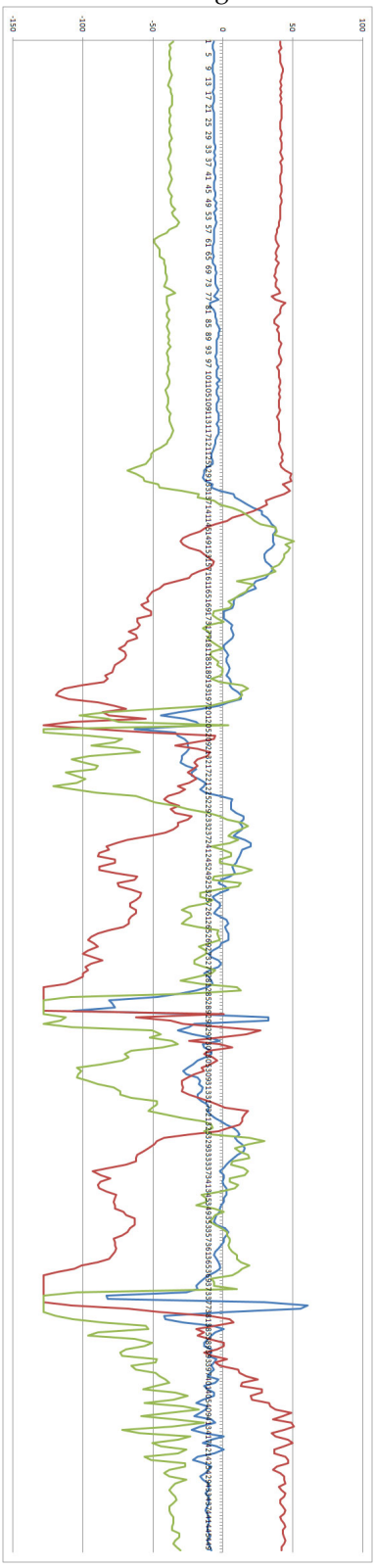
4.2.4 Large Swing

A large full arm steady swing over the shoulder. The motion is repeated.



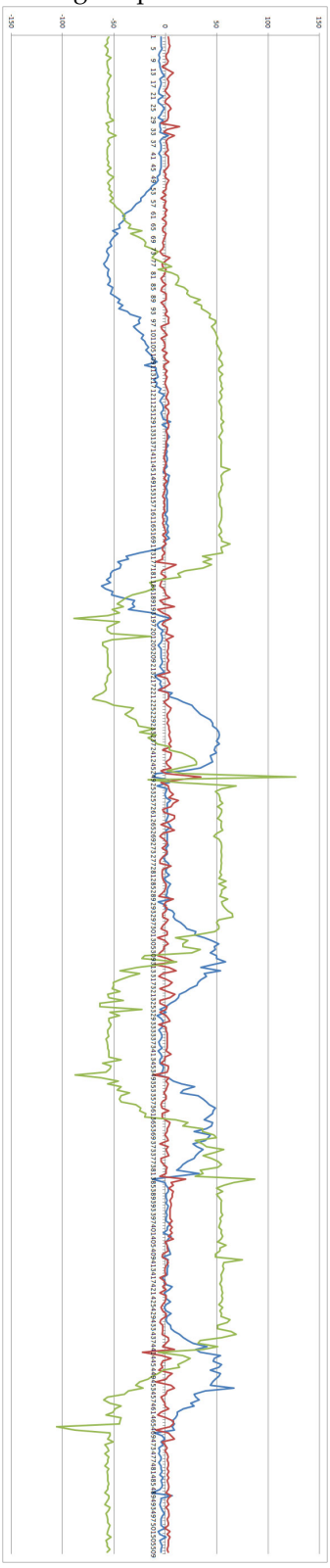
4.2.5 Violent Swing

Some violent swings over the should as fast as possible.

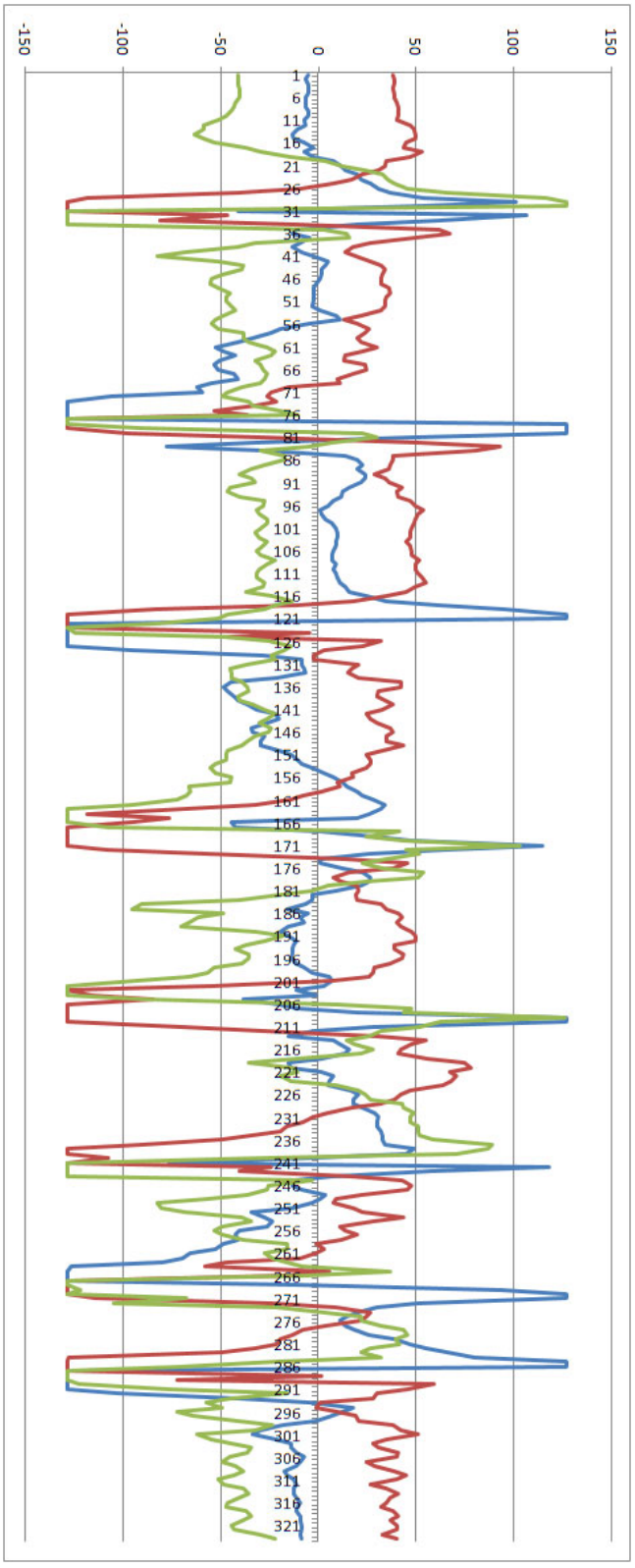


4.2.6 Flip over

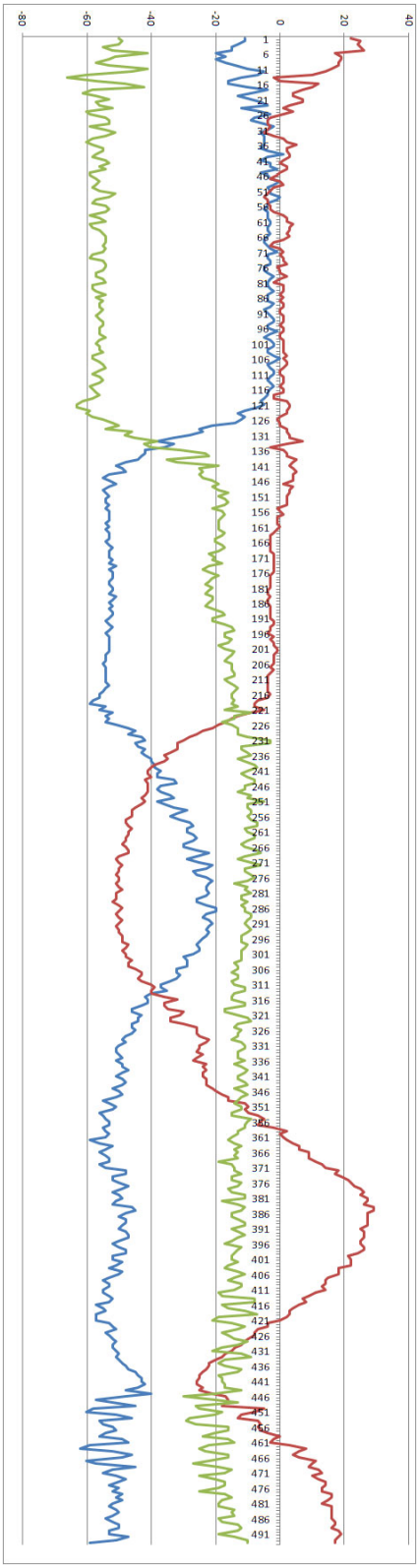
Placing the phone on a flat surface then gently flipping it several times.



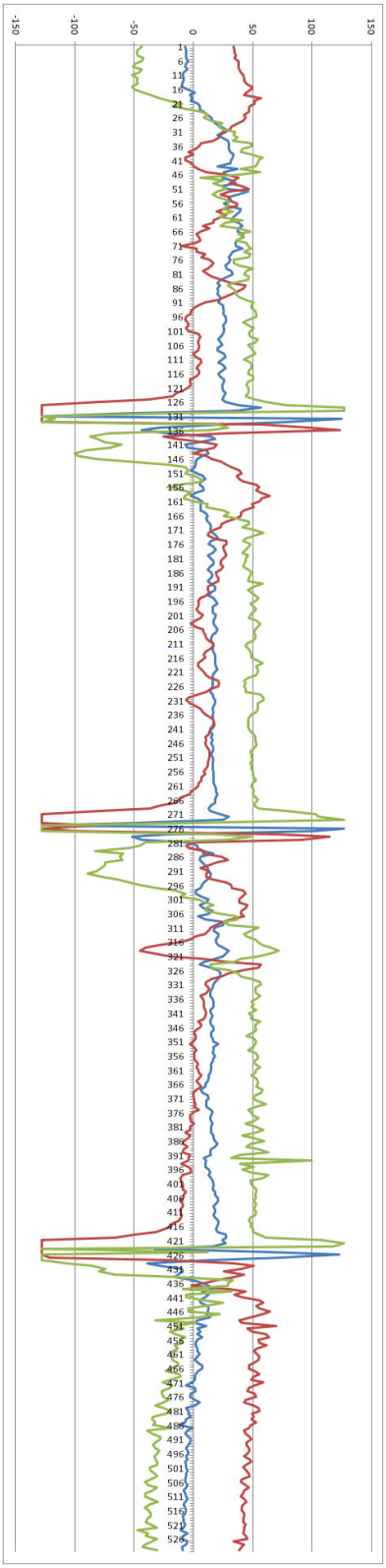
4.2.7 Jerk


-150
-100
-50
0
50
100
150
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
121
126
131
136
141
146
151
156
161
166
171
176
181
186
191
196
201
206
211
216
221
226
231
236
241
246
251
256
261
266
271
276
281
286
291
296
301
306
311
316
321

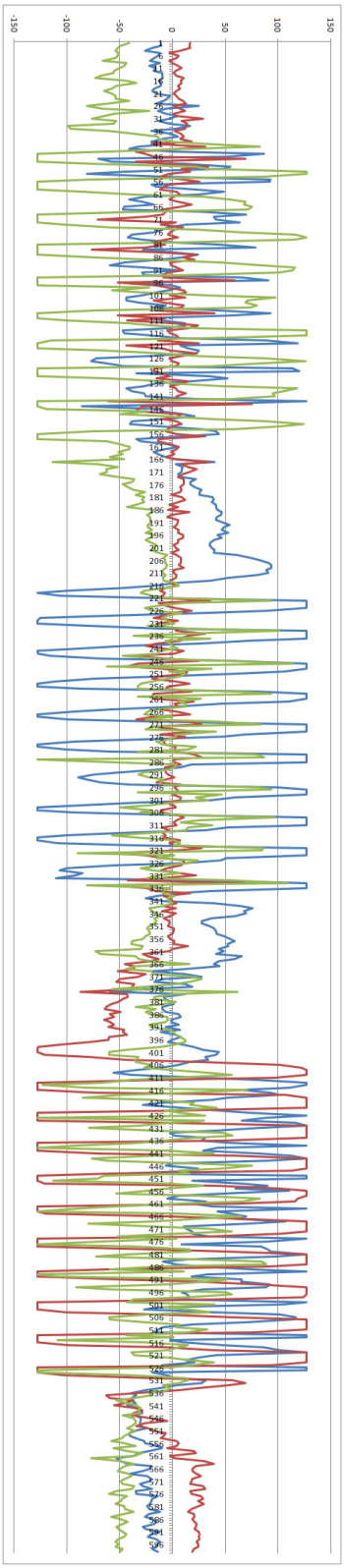
4.2.8 Balance



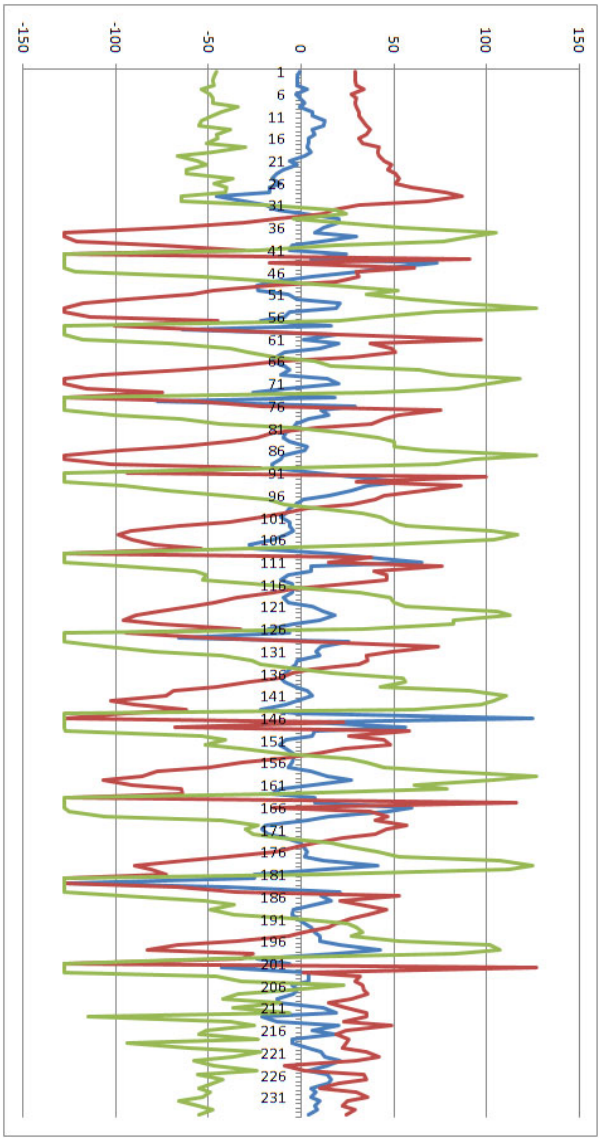
4.2.9 Flick



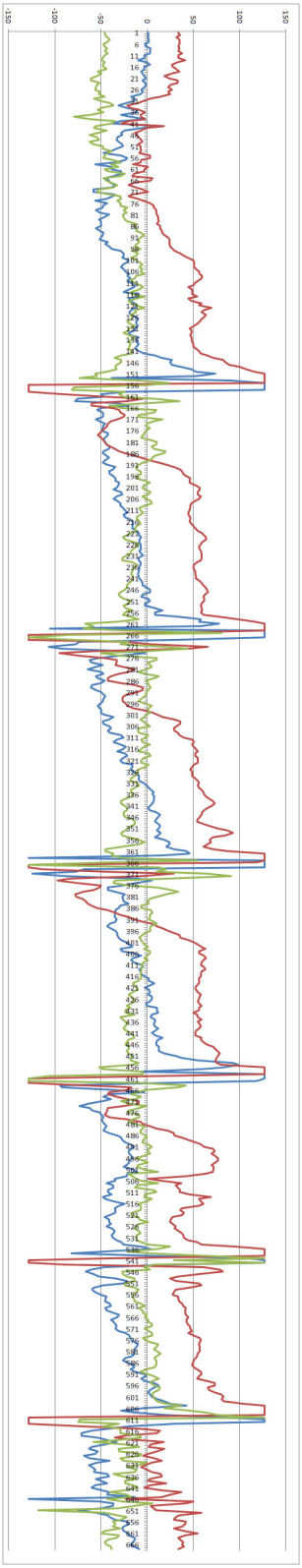
4.2.10 Pump



4.2.11 Hammer


-150
-100
-50
0
50
100
150
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
121
126
131
136
141
146
151
156
161
166
171
176
181
186
191
196
201
206
211
216
221
226
231

4.2.12 Javelin



5 Analysis of Motion

The main problem with the accelerometer when it comes to human gestures is we have a relative position. The best description I have found of this is the ball in the box.

If we take an empty box and place a ball in the box and close the box. We then rotate the box and gravity acts on the ball moving the ball along the edges of the box as predicted. The problem is if we introduce any form of motion to the box the ball cannot distinguish between box motion and gravity. No matter how many balls (sensors) we put in the box (phone) the relative forces will be relative and with peaks integration of those forces to meaningful coordinates becomes more guesswork than maths.

The only way to truly solve this problem is to have a fixed position which we can measure against. While looking into the problem I found several solutions using cameras, WiFi, and even standing waves.

The fact remains we must decide what is possible with this box and ball approach.

5.1 Orientation and Rest

The easiest and most reliable piece of information we can obtain is the orientation of the phone. Looking at the final motion capture we see a very predictable behaviour, if we smooth out the noise.

The orientation information is achieve by taking the normal of all the axis. The problem is this method only works at rest. The easiest and fairly reliable approach is to measure the length of the non-normalised vector. If the length falls within a defined range. I found between 50 and 62 worked fairly well.

This approach can be compromised if the acceleration vector and gravity vector are aligned in such a way the negate part of each other summing to the equivalent strength of resting.

5.2 Velocity and Position

The best approach we have for this is to integrate the acceleration to retrieve the velocity and through that the position. This approach has two major problems which immediately arise.

The first is whenever the device peaks information is lost so your calculations are immediately guesswork from that point on. You can use the speed at which the peak was reach but this is crude and integration expounds these errors.

The second problem is collision or gravity depending how you wish to look at it. You need to reliable track gravity through the orientation. You also cant tell the difference between a phone in free fall and a phone at rest on a table.

Any method that relies on positional tracking using this device is bound to be highly inaccurate and only workable at very slow speeds.

5.3 Peaks and Smooth

A commonly used motion approach for the wiimote which applies to this device is the peak and smooth game-play mechanic. As previously discussed it is exceptionally easy to peak the device but the speed at which the device peaks can be used to estimate the size of the peak.

A peak is also easy to detect the direction as the motion is violent enough to minimize the effect of gravity. For these reasons it has become possibly the most popular game mechanic used in Wii games. Many simple gestures can be defined by their peaks.

5.4 Example Gestures

- *Jerk:* We don't care about the direction, all that is required is the device peaks on a single axis. The most common tool in a motion sensor game. As a game mechanic we treat this as a very low resolution analogue button, or in some cases digital.

- *Balance* Using the orientation to steer. This has been used for a variety of balance games and driving games. Driving games simply use the orientation to direct the car or marble. As a game mechanic this is best implemented with jerk or another motion heavy gesture so the player must go from a violent action to steady balance.

- *Flick:* Similar to the peak this is normally a flicking motion of the wrist in a single direction. The final direction of the peak can be difficult to determine because the device normally peaks before reaching the flick point. Due to the small time interval it can be roughly estimated by taking the time it takes for the peak to end. Also if you can afford a second pause to read the orientation after the flick. As a game mechanic the flick or toss is an easy motion people understand and thanks to its imprecision in reality people are more forgiving of errors. The direction is not reliable enough to be a core game mechanic.

- *Pump:* A pumping motion along any axis can be measure by the time between peaks and the alignment of the peaks. A perfect pumping motion would involve a time between peaks approaching zero and the peaks would be exactly opposite of each other. As a game mechanic this is extremely effective. It can measure high-speed motion through the interval and yet maintains an element of control.

- *Hammer:* The Hammer is very similar to the Flick and Pump. The player repeatedly hammers a location in mid air. Like the pump we an use the symmetry of the two peaks to aid in measuring the action. The problem is introduced in the variance in arc size of the hammer strike. As the device is often peaking before the end of the motion much like the flick we can no longer line up the negatives such as with the pump. As a game mechanic you can assume a perfect 90 degree arc but ultimately this method is less robust than Flick or Pump.

- *Javelin:* By measuring the smoothness of the function leading up to the peak you can determine the accuracy of the throw. The ideal being a straight line (consistent forces). You can then compare the peak direction to the projection of the lines mean. Note this is only for a throw with no wrist motion. If wrist motion is introduced you either choose to be more lenient or will need a different approach.

5.5 Conclusions on Motion

The first thing you will notice besides the peaks and noise of the device is the similarity of actions. So these actions are not so much gesture recognition as motion detection. We need to know what we are going to get before we process it.

While this mostly sounds negative there is one alternative approach we haven't explored in this paper which is rather successful. Neural Networks are easily trained using several people demonstrating a motion. The Neural Network primary ability is pattern recognition so its ideally suited for this application.

It must be stressed a Neural Network cannot overcome the physical limitations of the device as we have discussed. However the range of motion it can detect is rather surprising, it still helps to have a context sensitive dictionary of gestures which you keep to a bare minimum. The reason we have not explored Neural Networks is the processing power available to us on a mobile device is not sufficient to the task.

The most deceptive part about first party demonstrations is they have specifically selected motions which are the least likely to cause confusion and give the illusion of much wider capability than actually exists. We can use this deception in our applications but it will not be long (in some circles it is already occurring) that the gaming public realise the limitation of this generation of motion controller and the repetitive nature of the motions.

6 Catapult Application

6.1 Design

We are presenting a simple game in which you load and fire a catapult at a target. The important part of the demo is the motion controller so we have laid out the motions clearly in this design.

The player has a catapult and a target which they must hit in the shortest possible time. Most feedback is auditory to allow free use of motions.

- *Load Catapult:* This consists of the player performing the pump motion several times to load the catapult stone.
- *Ready the Catapult:* The player performs several controlled flips of the phone. Any peaks will reset this step.
- *Aim the Catapult:* The player must tilt the phone gently to align the catapult to the target.
- *Fire Catapult:* This is performed the moment a jerk occurs during the aiming phase.

6.2 Challenges

The main challenge was finding simple solutions to the gesture recognition patterns which needed to be discovered. We decided to write a minimal method which would function but not measure the degree of success. This is due to the amount of balancing which is involved in fine tuning these functions.

The lack of a USB cable in the later stages of the project happened things and transfer times went from seconds to minutes (swapping in and out memory cards). The lack of a debugger or means to test on windows was also a problem. There is very recently a project which has started for Linux which involves writing a py60 emulator.

The actions you take for the catapult game mean the screen is barely visible. This was overcome using by using sound prompts.

In terms of a graphical interface we coded a simple graphical input but the lag introduced reduced the quality of the motion sensor.

6.3 Code

Code Listing 4: Catapult Test

```
import appuifw, e32, audio, axyz, graphics, math

WHITE = (255,255,255)
RED   = (255,  0,  0)
GREEN = (  0,255,  0)
BLUE  = (  0,  0,255)

## Global Varibles
running = 1
totalTime = 0
xyzValues = [0, 0, 0]

lastpeak = [0, 0, 0]
peakTime = 0
levelTime = 0
peak = False

catDir = 0
catAng = 0

def read_xyz(x, y, z):
    global xyzValues

    if (len(xyzValues) > 50):
        del xyzValues[0]
        del xyzValues[1]
        del xyzValues[2]

    xyzValues.append(x)
    xyzValues.append(y)
    xyzValues.append(z)

def handle_redraw(rect):
    if img:
        img.clear(BLUE)
        #img.text((10,60), u"Time: %d" % (levelTime), fill = RED)
        canvas.blit(img)

def handle_event(event):
    if event["type"] == appuifw.EEvent.KeyDown:
        quit()

def checkPeak(x, y, z):
    retValue = (abs(x) > 120) or (abs(y) > 120) or (abs(z) > 120)
    return retValue

##      Load Catapult
## This consists of the player performing the pump motion several
## times to load the catapult stone.
def load_tick():
```

```
    global xyzValues
    global levelTime
    global peakTime
    global lastpeak
    global peak
    global running

    xyzLen = len(xyzValues)
    if(checkPeak(xyzValues[xyzLen-3], \
                 xyzValues[xyzLen-2], \
                 xyzValues[xyzLen-1])):
        if(peak == False):
            if(peakTime > 200):
                peakTime = 0

                if( ((lastpeak[0] > 0) != (xyzValues[xyzLen-3] > 0)) or
                    ((lastpeak[1] > 0) != (xyzValues[xyzLen-2] > 0)) or
                    ((lastpeak[2] > 0) != (xyzValues[xyzLen-1] > 0)) ):
                    levelTime -= 1
                    print u"Pump: %d" % (levelTime)
                    lastpeak[0] = xyzValues[xyzLen-3]
                    lastpeak[1] = xyzValues[xyzLen-2]
                    lastpeak[2] = xyzValues[xyzLen-1]


        peak = True
    else:
        peak = False
        peakTime += 1

    if(levelTime < 1):
        print u"=== READY THE CATAPULT ==="
        audio.say(u"Ready, Ready the Catapult")
        running = 2
        levelTime = 8
        peak = False
    return True

##      Ready the Catapult
## The player performs slow flips moves to wind the catapult
def ready_tick():
    global xyzValues
    global levelTime
    global peakTime
    global lastpeak
    global peak
    global running

    xyzLen = len(xyzValues)
    normVector = [xyzValues[xyzLen-3], \
                  xyzValues[xyzLen-2], \
                  xyzValues[xyzLen-1]]

    if(checkPeak(normVector[0],normVector[1],normVector[2])):
```

```
        print u"FAIL!"
        audio.say(u"Fail")
        levelTime = 8
        print u"=== READY THE CATAPULT ==="
        return True

    flipDone = False

    if(peak):
        if((normVector[2] > 45) and (normVector[2] < 60)):
            peak = False
            flipDone = True
    else:
        if((normVector[2] > -60) and (normVector[2] < -45)):
            peak = True
            flipDone = True

    if(flipDone):
        levelTime -= 1
        print u"Flip: %d" % (levelTime)

    if(levelTime < 1):
        print u"=== AIM THE CATAPULT ==="
        audio.say(u"Aim, Aim the Catapult")
        running = 3
        levelTime = 200
    return True

##      Aim the Catapult
## At this point a bearing and distance is given.
## The player must tilt the phone gently to align the catapult to the target.
def aim_tick():
    global xyzValues
    global catDir
    global catAng
    global running
    global levelTime

    xyzLen = len(xyzValues)
    normVector = [xyzValues[xyzLen-3], \
                  xyzValues[xyzLen-2], \
                  xyzValues[xyzLen-1]]

    Magnitude = math.sqrt(normVector[0]*normVector[0] + \
                    normVector[1]*normVector[1] + \
                    normVector[2]*normVector[2])

    if(Magnitude > 0):
        # normalise vector
        normVector[0] = normVector[0] / Magnitude
        normVector[1] = normVector[1] / Magnitude
        normVector[2] = normVector[2] / Magnitude

        # if at rest
```

```
        if((Magnitude > 50) and (Magnitude < 62)):
            catDir += normVector[0] / 2
            catDir = (catDir + 360) % 360

            catAng = abs(normVector[1]) * 90
        else:
            if((levelTime < 1) and \
               (checkPeak(xyzValues[xyzLen-3], \
                          xyzValues[xyzLen-2], \
                          xyzValues[xyzLen-1]))):
                print u"=== FIRE!!! ==="
                audio.say(u"FIRE")
                running = 4
                levelTime = 1000

    if(levelTime > 0):
        levelTime -= 1

    print u"Angle: %d  Direction: %d" %(catAng, catDir)
    return True

##      Fire Catapult
## This is performed the moment a jerk occurs during the aiming phase.
def fire_tick():
    global running
    global levelTime
    global totalTime

    if(levelTime < 0):
        running = 0
        print u"Your shot took %d ticks" % (totalTime)

    levelTime -= 1
    return True


def quit():
    global running
    running = 0

## Setup Globals
acl = [0,0,0]

## Setup Image
img = None
#canvas = appuifw.Canvas(redraw_callback = handle_redraw,\
#                        event_callback = handle_event)

#w, h = canvas.size
#img = graphics.Image.new((w, h))
#img.clear(BLUE)

## Connect to Sensor
axyz.connect(read_xyz)
```

```
## Setup App
appuifw.app.title = u"N95 accelerometer"
# appuifw.app.screen = "large"
# appuifw.body = canvas
appuifw.app.exit_key_handler = quit

print u"=== LOAD THE CATAPULT ==="
running = 1
levelTime = 25


while (running > 0):
##      Load Catapult           1
    if(running == 1):
        load_tick()

##      Ready the Catapult      2
    if(running == 2):
        ready_tick()

##      Aim the Catapult        3
    if(running == 3):
        aim_tick()

##      Fire Catapult           4
    if(running == 4):
        fire_tick()

##    handle_redraw(None)
    totalTime += 1
    e32.ao_yield()

## Exit
axyz.disconnect()
print "Safely Exited"
```

7 Conclusion

The choice of platform and device was a well made one. The device is well suited for rapid prototyping and Python is still flexible. The libraries are well structured and light weight enough that minimal effort is required to start.

The amount of feedback a mobile platform can provide while doing these motions is limited. This coupled with the limiting processing power means as a motion gaming platform it is limited. The use of motion in other applications however is still a valid.

The motion sensor is a limited device which does not do what sales teams claim. The processing of motion is several degrees more complex than traditional controllers. This is compounded if a neural net is used. The range of motion detection is very limited. This is due to both the low range of the device and lack of absolute positioning.

If a game is made with the limits in mind a successful game mechanic can be achieved. The gaming audience is more likely to notice the limited patterns of this style of motion controller as time passes. This receptiveness could have a strong negative affect on games which could turn unique control systems into simple formulaic inputs.