ISAT 280 – Mobile App for Beginners Using Python/Kivy
Spring 2015
Lab 05 – Plyer (Sensors), Pyjinus (Java Classes) and Network Interfaces

1. Learning Objectives:

- Sensors (Plyer)
- Pyjnius
- Network Interfaces

2. Hardware/software

- Android Nexus 10
- Desktop Host Linux Ubuntu 14.04 (x64) - password : hellocheckin
- VMware WorkStation 10 Ubuntu 14.04
- Optional iMac with Maverick 10.9.4 and Virtualbox Win7

3. Introduction

- Sensor Framework: You can access these sensors and acquire raw sensor data by using the Android sensor framework. The sensor framework is part of the android.hardware package and includes the following classes and interfaces:SensorManager, Sensor, SensorEvent and SensorEventListener
- Check out the android.hardware package interfaces and classes in http://developer.android.com/reference/android/hardware/package-summary.html
- You should read the following overview on sensors http://developer.android.com/guide/topics/sensors/sensors_overview.html

4. Exercises:

.1 Exercise 1: Plyer and Android Hardware

- Plyer is a Python library for accessing features of your hardware/platforms. https://plyer.readthedocs.org/en/latest/
- Plyer tries not to reinvent the wheel, and will call for external libraries to implement the api in the easiest way, depending on the current platform. On python-for-android, pyjnius is used, on kivy-ios, pyobjus is used and on windows/mac/linux, commonly found libraries and programs will be used. See Appendix A for plyer support.
- Plyer is a platform-independent api to use features commonly found on various platforms, especially mobile ones, in Python. The idea is to provide a stable API to the user for accessing features of their desktop or mobile device. Under the hood, plyer uses PyJNIus and PyOBJus.
  https://github.com/kivy/plyer
  https://github.com/kivy/pyjnius
  https://github.com/kivy/pyobjus
- Pyjnius (JAVA API) is a Python module that lets you access java classes directly from Python, automatically converting arguments to the right type, and letting you easily convert the java results to Python. Pyobjus is to ios what pyjnius is to android.
- Additional capabilities are also available through the android module (Python API). It consists of multiple parts which are mostly there to facilitate the use of the Java API.

At this point, there is no plan to exercise the python API, as plyer and (under the hood pyjinus) should suffice for our immediate needs.

- Let us build our first application where we can manipulate one of the hardware elements of the Android platform, that is, vibrator. We will create the application using the plyer framework first. In later exercises, we will attempt to replicate the app using pyjnius framework.

STEP 1 (VIBRATE)

- Let us download plyer-master https://github.com/kivy/plyer. Extract into a folder by the same name under /home/checkout. Copy the content of the folder called vibrator under plyer-master/examples to a folder for your lab5 exercises (just like we did in previous labs). Make sure to use "git" to keep track of your projects and codes. Next, upgrade plyer in your environment by typing: sudo pip install –upgrade plyer.
- Open main.py for the vibrator and explore its structure. Identify any new constructs or packages being imported. Once, this is done, let us run the code using python2.7 main.py.
  The application will crash and give us an interesting error.
  Q1 – Explain the error and use Appendix A to explain the reason behind the error.
- You need to check whether KivyLauncher is installed on your tablet. If it is not then download KivyLauncher-0-11-16.2014-1-debug.apk from Canvas/Lab5 and install on the tablet using adb install <KivyLauncher package>.
- If it is, then you need to check its permission list (settings >apps> kivylauncher). If permissions include control vibration, then you are set to go. If not, then you need to uninstall the KivyLauncher currently installed on your tablet and install KivyLauncher-0-11-16.2014-1-debug.apk
- Before we proceed we need to add android.txt file with three lines in the folder where the main.py resides.

  title=vibration
  author=<your name>
  orientation=landscape
- Next, let's create a folder on Nexus called kivy under sdcard directory. In the kivy folder we should create a folder called kivy-lab5-exe1-1. You can accomplish so using adb shell > cd sdcard > check if kivy folder is there if not > mkdir kivy > cd kivy > mkdir kivy-lab5-exe1-1 > exit
- On the VM, navigate to the directory where the main.py and android.txt reside. cd ../ to go one level and push the folder (all files) from the desktop to Nexus 10

  adb push <name of the folder> sdcard/kivy/kivy-lab6-exe1-1

  Q2 – Find the source code for plyer.vibrator and explain how the various methods work

STEP 2 (GPS): Implementation 1

- Let us turn our attention to another hardware device, that is, GPS (location). GPS is typically not considered a sensor. However, for all practical purposes, it is a hardware device that provides data (lat, long, etc.).
- Check that your install KivyLauncher permissions include approximate location (network based) and precise location (GPS and network based).
- Next, download GPS implementation I created at the early stage of learning Kivy. Download lab5-exe1-2.tar.bz2. Extract the folder lab5-exe1-2. Open both the main.py and floatlayout.kv. Study the code
  Q1 - Identify "poor" programming practices in both files.
- Try to run the application on the desktop, again by typing python2.7 main.py. It should fail again.
  Q2 – identify the error and propose a solution. Hint: see Appendix A
- Push lab5-exe1-2 to the Nexus 10 tablet and see if you can run it using KivyLauncher. If it does not work, you need to attempt fixing it.
- Once you get the application to run (see Figure 1-1), test it and see what it does.
  Q3 – identify all the methods available under plyer.gps. Provide detailed explanation of what they do with emphasis on the arguments accepted by the methods.

STEP 3 (GPS): Implementation 2 and Integration

- Access the GPS example provided under plyer-master. Run it on the desktop (see Figure 1-2) and on Nexus (See Figure 1-3). Compare the functionality of the first implementation with the second (from tito).
  Q1 – Provide your assessment of the differences. Do both implementations take advantage of the fine_location and coarse_location information?
  Q2 - In both implementations explain the following statement IN DETAILS:
  self.gps_location = '\n'.join(['{}={}'.format(k, v) for k, v in kwargs.items()])
  Q3 – Explain why and how the implementation 2 of GPS did not crash when launched under Linux Desktop.

  Extra Credit - Figure out a way to combine the vibration with the GPS functionality in one program. For example, use the altitude value to trigger a certain number of short vibration

4.2 Exercise 2: Networking & WiFi Mgr (Pyjinus)

Make sure to install mysql-connector-python in your environment using the following command: sudo pip install –allow-external mysql-connector-python mysql-connector-python (yes, you are to repeat mysql-connector-python twice).

- In this exercise, we will explore another side of Kivy where management of Android hardware is not as straight forward as calling classes and methods from a single module such as plyer (which we used in the previous exercise with ease).
- The main focus of this exercise is to integrate two applications where one of them is to do with checking Wi-Fi connectivity and the other is to do with remote account management.

- However, before we go into that we need to learn the basics of a new module called Pyjnius (typically imported as jnius). We will use the vibrator example to demonstrate the use of pyjinus.

STEP 1 (Learn Pyjnius using Vibration)

- Pyjinus enables us to have access to Android JAVA classes (including their methods and properties). The first step is to find the java code for a particular service such as Vibrator Service. Here is the code I found at http://programmerguru.com/android-tutorial/android-vibrate-example/
  Check it out. Most of the code is about building the GUI interface on android. Here is the code (with comments) to do with the vibrator services (in JAVA):

```
// Create Vibrator instance for current context
mVibrator = (Vibrator) getSystemService(Activity.VIBRATOR_SERVICE);
// Vibrates for 10000 Milliseconds
 mVibrator.vibrate(10000);
```

  Note that I have changed the duration from 300 to 10000 msec.
- The corresponding code in python will require one or more classes of the pyjnius module. Check that pynius is already available in your environment by typing pip freeze and look for the module jnius in the returned list. If the module is not already installed, you could install it using sudo pip install jnius. See Java API (pyjnius) documentation at http://python-for-android.readthedocs.org/en/latest/javaapi/

- To make use of pyjnius in your program, you need to import some of the pyjmius classes such as autoclass

```
from jnius import autoclass
```

  We will use it in the following manner:

```
PythonActivity = autoclass ('org.renpy.android.PythonActivity')
```

  If you want the instance of the current Activity, use:

```
activity = PythonActivity.mActivity
```

  "autoclass" is the function that takes a java class and produce a python class with the same methods and properties that let us call that class from python (it is a kind of proxy that reflects all the methods and properties available from the "autoclassed" Java class).

  PythonActivity is not a normal android api class but it's actually the android activity used by kivy app when it is running as an android apk. The reason, we need it, is that it stores a reference to the kivy code running activity. Also, we need the mActivity which is an instance of the PythonActivity that allows us to call python api.

- To recreate the vibrator app, we will need to re-write the equivalent of JAVA code in python. Here is the JAVA code again:

```
// Create Vibrator instance for current context
mVibrator = (Vibrator) getSystemService(Activity.VIBRATOR_SERVICE);
// Vibrates for 10000 Milliseconds
 mVibrator.vibrate(10000);
```

- Here is the corresponding code in python:

```
Activity= autoclass('android.content.Activity')
vibrator = activity.getSystemService(Activity.VIBRATOR_SERVICE)

class AndroidApp (App):
   def build(self):
      vibrate.vibrate(10000)
```

  The entire code is given in Appendix B.
- Create a folder under /sdcard/kivy on the Nexus 10. Now, push main.py and android.txt (3 lines) to that folder. Run the code by launching KivyLauncher on Nexus. Nexus 10 should vibrate for 10 seconds.
- Whenever you have code involving plyer and/or pyjinus that is designed to manipulate hardware features on android, you will need to test your code on an android and therefore you need to get into the habit of opening 4 terminals (shells) on your desktop:
  - one for editing your main.py (and push it along with other files to android platform),
  - one for *adb logcat*,
  - one for *adb shell* to create folders on android and verify the files pushed from the desktop to the android platform and the
  - one for editing *.kv file and run the code on your desktop for simple syntax errors debugging.

  Open 4 shells as described above. Change
  vibrator = activity.getSystemService(Activity.VIBRATOR_SERVICE)
  to
  vibrator = activity.getSystemService(Activity.VIBRATO_SERVICE)

- Launch KivyLauncher and run your vibration code. It should crash. Using logcat identify the error and confirm that it is due to the change we made!
  Extra credit – recreate the example vibrator that you downloaded and used in exercise from plyer using pyjinus instead of plyer.

STEP 2 (Learn more about Pyjnius using WiFi)

- Let us return back to building the application we really need which is WiFiCheckConnect app. The first thing we need, is to find the JAVA code that can give

us some guidance as to the java class methods and properties. Here is the android JAVA code for WiFi Check Connectivity (minus imports and onCreate) :

```
public boolean checkConnectivity(){

  ConnectivityManager conMgr = (ConnectivityManager)getSystemServic
e(Activity.CONNECTIVITY_SERVICE);

  boolean conn = conMgr.getNetworkInfo(ConnectivityManager.TYPE_WIF
I).isConnectedOrConnecting();
        if(conn){
            return true;
        } else {
            return false;
        }
    }
```

- In the above code, the author is using the ConnectivityManager Class (android.net.ConnectivityManager). Check android online API documentation: http://developer.android.com/reference/android/net/ConnectivityManager.html
- This is a class that answers queries about the state of network connectivity (wifi, cellular, etc.). It also notifies applications when network connectivity changes. One can get an instance of this class by calling (in JAVA)
  getSystemService(Activity.CONNECTIVITY_SERVICE)
- The corresponding code in python will require autoclass from the pyjnius module. So the first thing we need to do is again to import the following class autoclass in our main.py file
  from jnius import autoclass

- We will use the autoclass function in the following manner:

```
.........
.........
PythonActivity = autoclass('org.renpy.android.PythonActivity')
activity = PythonActivity.mActivity

Activity = autoclass('android.app.Activity')
conn_mgr = autoclass('android.net.ConnectivityManager')

......
......
class AndroidApp(App):
   def __init__(self, **kwargs):
      super(AndroidApp,self).__init__(**kwargs)

   def build(self, *args, **kwargs):
      print "I am here."
     return FloatLayoutWidget()
```

```
    def check_connectivity(self):
        self.con_mgr = activity.getSystemService(Activity.CONNECTIVITY_SERVICE)
        self.conn =
self.con_mgr.getNetworkInfo(conn_mgr.TYPE_WIFI).isConnectedOrConnecting()

        if self.conn:
            print "it's up"
            return "it's up"
        else:
            print 'it is down'
            return "it's down"
```

You will need to complete the above code with imports and Widgets to get the result shown in Figure 2-1

Check out the following web pages of the Android API Reference for Connectivity Manager Class (see above) and for Network Info method in *http://developer.android.com/reference/android/net/NetworkInfo.html*

Note that your program should display "It's up" if the WiFi is connected to a network, and "It's down" when Wifi is not connected to a network.

STEP 3 (Learn more Kivy/ScreenManager, PopUp with AMA)

- Download the Account Management & Authentication (AMA) application (lab5-exe2-3.tar.bz2) from Canvas (Lab 5 folder). Un-compress and save the files in, say, kivy-lab5/lab5-exe2-3 folder.
- Launch the VM with mysql db (mysql280-Ub-Server32bit).

  If you do not have access to mysql280-Ub-Server32bit, see the instructions in Appendix C of creating your own.

  Make sure that the network adapter of the VM is in Bridged connection mode. Make sure that mysql is running using sudo service mysql status.

  Log into mysql using the following command:

  mysql –u root –p

  When you're asked for a password, enter checkout

  Check the databases using

  show databases;

  Navigate to the database name my280 using the following command

  use my280;

  Don't forget the semi-colon at the end of the command

  Check the tables in my280 using

  show tables;

  Check the record in the userme table using

*select * from userme;*

If you need to delete a record, use

delete from userme where uid=<#>;

Leave the database command line user interface running so that you can check the results of the sign up (adding a new account) capabilities.

- Find out the ip address of the database server, that is, mysql280-Ub-Server32bit VM. Launch your Ubuntu Linux development VM. Make sure that network adaptor of this VM is also configured for NAT connection.
- On your development environment VM, navigate to lab5-exe2-3 (where AMA is) and open main.py with an editor of your choice. Change the ip variable to the ip address of your database server to ip='<your server ip address>'. Don't forget about the quotes.
- Create lab5-exe2-3 directory on Nexus 10 using adb shell. Push main.py, Accountmgr.kv, android.txt and background.jpg to Nexus 10. Connect Nexus 10 to the ESXI wifi and make sure that you can ping Nexus 10 from mysql280-Ub-Server32bit VM (that is, Nexus 10 and VM can reach each other).
- On Nexus 10, launch KivyLauncher and run AMA application. You should be able to navigate through the 3 screens of the application as shown in Figure 3-1. Test the application thoroughly
  Q1 – List all the bugs you find in AMA. Describe how you fixed them.

STEP 4 (Integrate AMA & WiFiCheckConnect)

- Your job is not done yet. You need to integrate the AMA and WiFiCheckConnect. You should be able to check if the tablet is connected to a WiFi network before you attempt to log in or sign up on the database server.
  Q2 – Provide a complete commented integrated solution.

4.3 Exercise 3: DieRoller and Compass (Accelerometer & Orientation)

5. Exploration (Extra Credit)

6. Deliverables:

- Lab Report (ORGANIZE YOUR REPORT IN A WAY THAT HELP THE READER AND GRADER TO FIND THE ANSWERS AND SUPPORTING MATERIALS (INCLUDING CODE FILES) READILY, for example, provide sections and sub sections, number the sections, reference the names of the code files, etc.)
  - Cover page
  - Answers to questions
  - For each exercise, commented Code in the form of a compressed git repository (preferred compresses as tar.gz
  - Assessment of the lab (Observations, suggestions and Best Practices)

Appendix A: Plyer Support

Platform	Android < 4.0	Android > 4.0	iOS	Windows	OSX	Linux
Accelerometer	X	X	X		X	X
Camera (taking picture)	X	X				
GPS	X	X				
Notifications	X	X		X	X	X
Text to speech	X	X	X	X	X	X
Email (open mail client)		X	X	X	X	X
Vibrator		X				
SMS (send messages)	X	X				
Compass	X	X	X			
Unique ID (IMEI or SN)	X	X	X	X	X	X
Gyroscope	X	X	X			
Battery	X	X	X	X	X	X

Table 1 – Plyer Hardware Support on different platforms

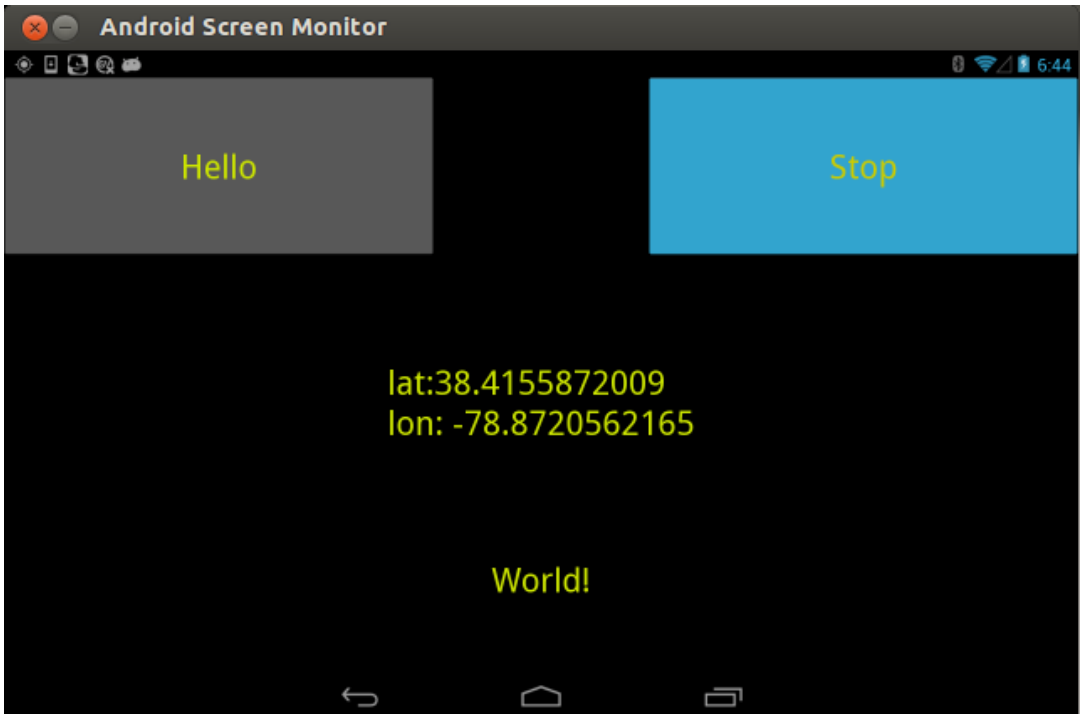

Figure 1-1 – Exercise 1 Step 2

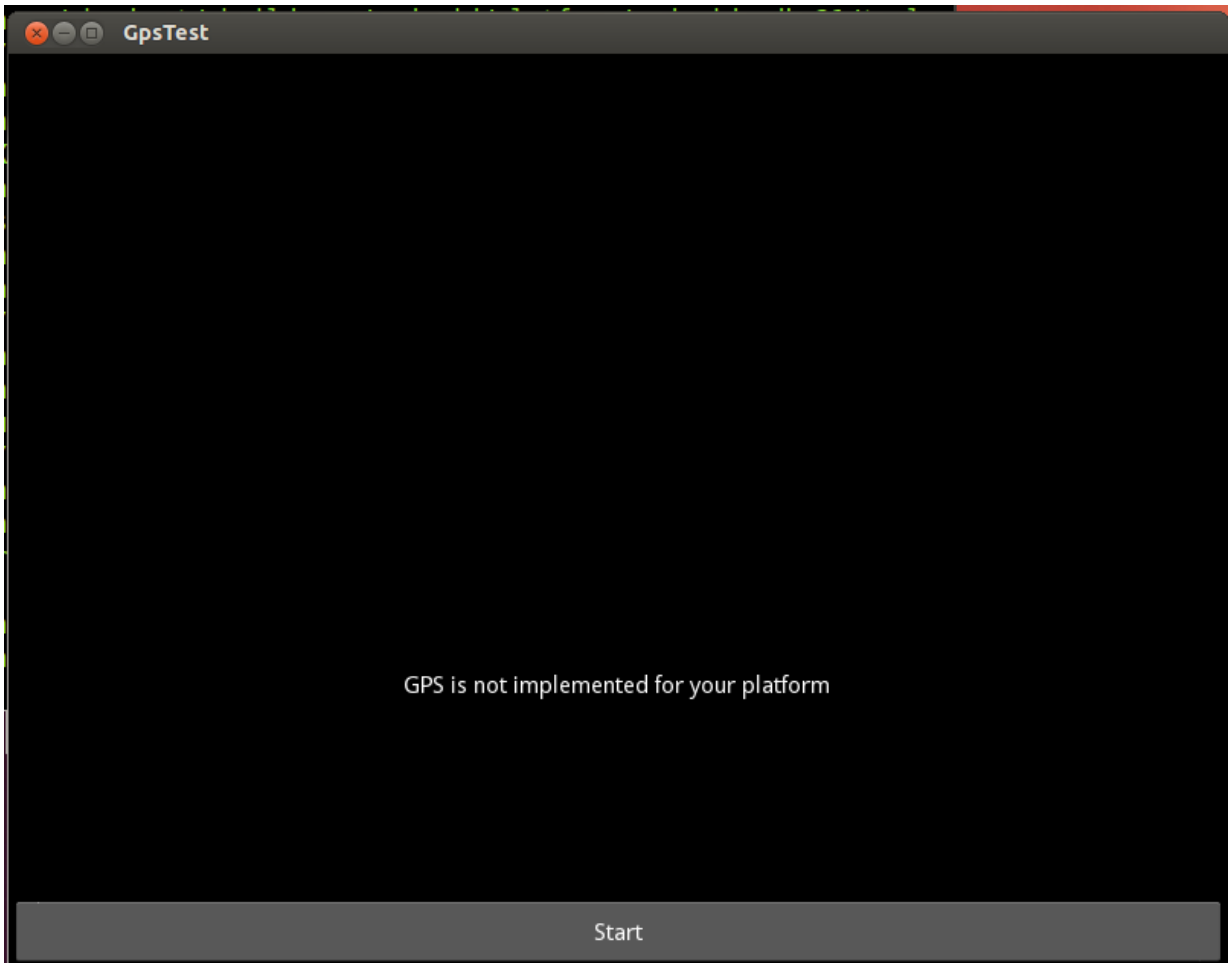

Figure 1-2 – Exercise 1 Step 3 GPS running on Linux

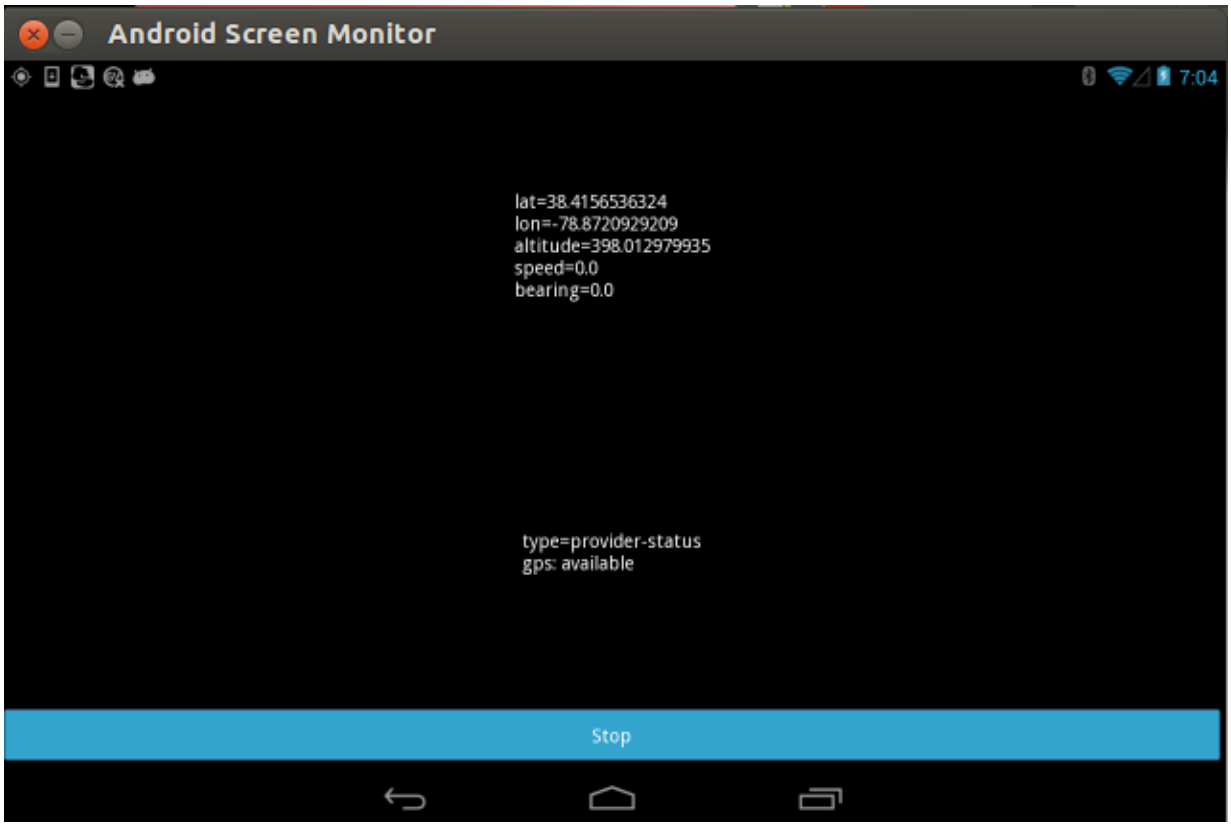

Figure 1- 3 – Exercise 1 Step 3 GPS running on Android

Appendix B

```
from jnius import autoclass
from kivy.app import App

PythonActivity=autoclass('org.renpy.android.PythonActivity')
activity =PythonActivity.mActivity

Activity=autoclass('android.app.Activity')
vibrator = activity.getSystemService(Activity.VIBRATOR_SERVICE)

class AndroidApp(App):
   def build(self):
      vibrator.vibrate(1000)

if __name__=="__main__":
  AndroidApp().run()
```

Figure 2 -1 WiFi Connectivity

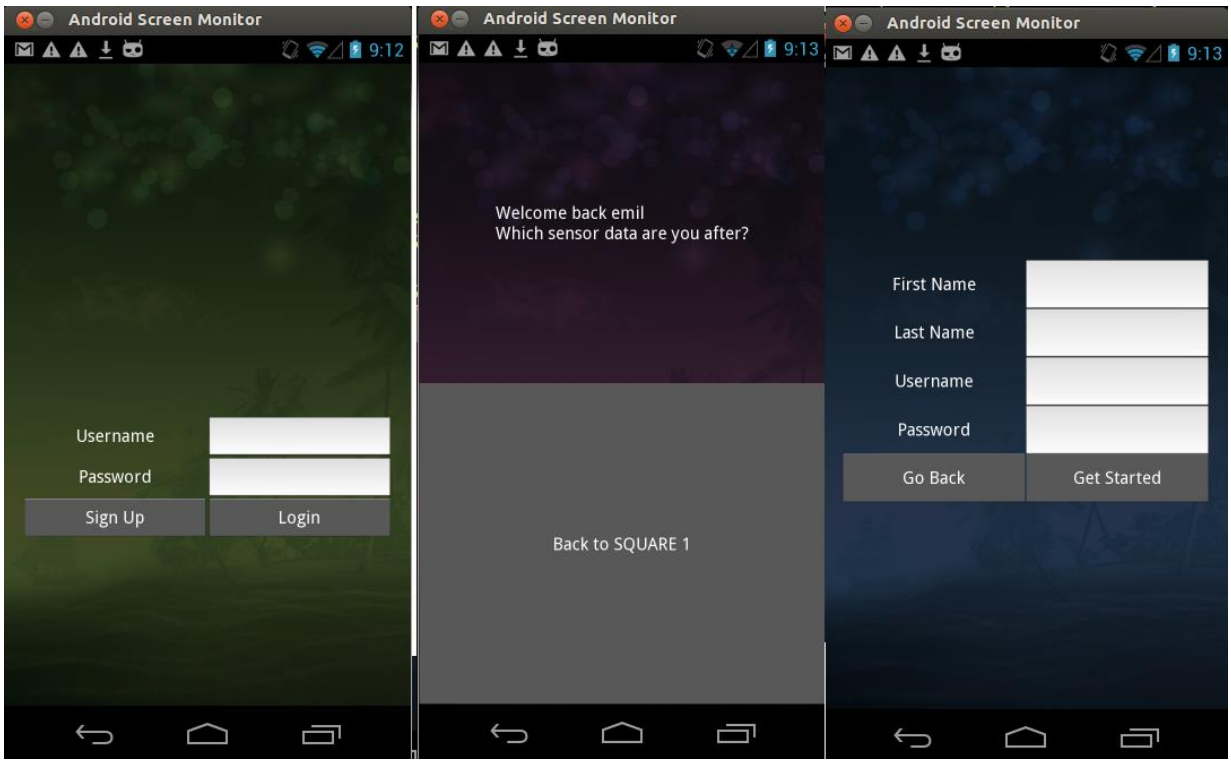

Figure 2-3 AMA Application

Appendix C

- Download Ubuntu 14.04 SEVER 32 bit iso. Create a virtual machine using your vmware WorkStation.
- Install mysql database server by typing the following in a command line terminal
  sudo apt-get install mysql-server
  When you're asked to enter a new password for the MySQL "root" user, enter the word checkout
- Once you install MySQL, you should activate it with this command:
  sudo mysql_install_db
- Finish up by running the MySQL set up script:
  sudo /usr/bin/mysql_secure_installation
  Enter the word checkout when asked for password and answer "no" to all the questions that follow.
- Create an empty database by logging into mysql (mysql –u root –p) and enter the following
  create database my280;
  exit
- Download my280.sql from Canvas under Lab 5 folder. Navigate to the Downloads directory (assuming that is where you downloaded my280.sql) by typing
  cd ~/Downloads
- To load the my280.sql into my280 database, open a terminal, enter the following
  mysql –u root –p my280 < my280.sql

Appendix C

Sensors:

- Motion Sensors – Accelerometer (gravity, linear acceleration), gyroscope, rotation vector.
- Position Sensors – Magnetic Field, Orientation, Proximity
- Environmental Sensors – Temperature, light, pressure, humidity
- Others – GPS, Camera, Microphone, Network, Vibrator

Sensor Coordinate System:

In general, the sensor framework uses a standard 3-axis coordinate system to express data values. For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation (see figure 1). When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face. In this system, coordinates behind the screen have negative Z values.

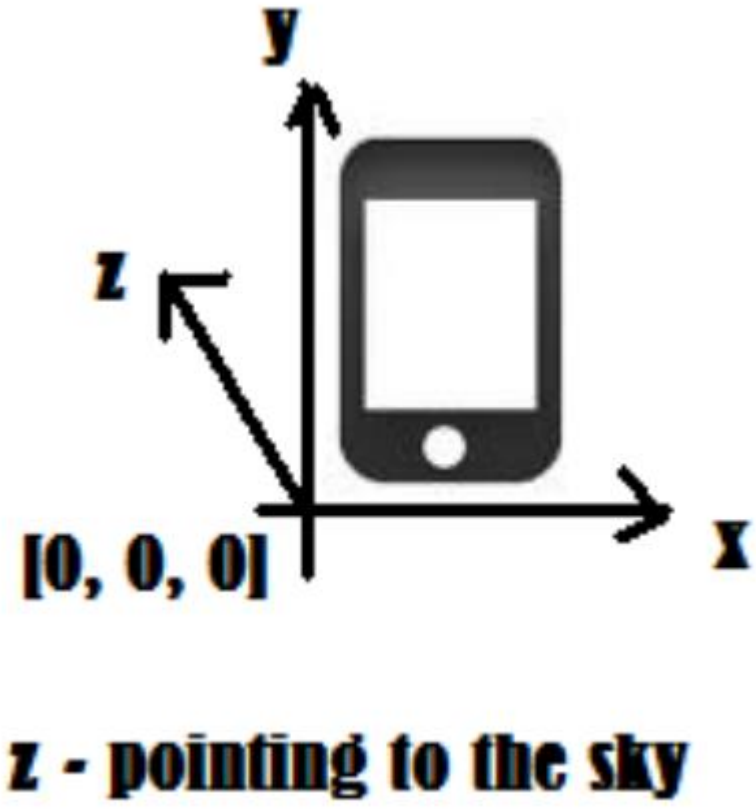

Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.

Accelerometer Sensor

- TYPE_ACCELEROMETER
- Values[3] = m/s^2, measure the acceleration applied to the phone minus the force of gravity (x, y, z)
- GRAVITY_EARTH, GRAVITY_JUPITER, GRAVITY_MARS, GRAVITY_MERCURY, GRAVITY_MOON, GRAVITY_NEPTUNE

Orientation sensor

- TYPE_ORIENTATION
- Values[3] – (Azimuth, Pitch, Roll) – angles 0-360
  – azimuth, rotation around the Z axis
  – pitch, rotation around the X axis
  – roll, rotation around the Y axis
- Different from plane yaw, pitch, roll (different axes and clockwise-ness)

Gyroscope sensor

- TYPE_GYROSCOPE
- Measure the orientation of a device
- Detect all rotations, but only few phones have it
- Values[] – iPhone gives radians/sec., and makes it possible to get the rotation matrix

Accelerometer vs Gyroscope

Accelerometer A
- senses linear movement, but worse rotations, good for tilt detection,
- Does not know difference between gravity and linear movement, shaking, jitter can be filtered out, but the delay is added

• Gyroscope G
- measure all types of rotation not movement does not amplify hand jitter

• A+G = both rotation and movement tracking possible

Appendix C Sensor types supported by the Android platform.

Sensor	Type	Description	Common Uses
TYPE_ACCELEROMETER	Hardware	Measures the acceleration force in m/s²that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (shake, tilt, etc.).
TYPE_AMBIENT_TEMPERATURE	Hardware	Measures the ambient room temperature in degrees Celsius (°C). See note below.	Monitoring air temperatures.
TYPE_GRAVITY	Software or Hardware	Measures the force of gravity in m/s² that is applied to a device on all three physical axes (x, y, z).	Motion detection (shake, tilt, etc.).
TYPE_GYROSCOPE	Hardware	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).	Rotation detection (spin, turn, etc.).
TYPE_LIGHT	Hardware	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.
TYPE_LINEAR_ACCELERATION	Software or Hardware	Measures the acceleration force in m/s²that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	Monitoring acceleration along a single axis.
TYPE_MAGNETIC_FIELD	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT.	Creating a compass.
TYPE_ORIENTATION	Software	Measures degrees of rotation that a device makes around all three physical axes (x, y, z). As of API level 3 you can obtain the inclination matrix and rotation matrix for a device by using the gravity sensor and the geomagnetic field sensor in conjunction with the getRotationMatrix() method.	Determining device position.
TYPE_PRESSURE	Hardware	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.

TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
TYPE_RELATIVE_HUMIDITY	Hardware	Measures the relative ambient humidity in percent (%).	Monitoring dewpoint, absolute, and relative humidity.
TYPE_ROTATION_VECTOR	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection.
TYPE_TEMPERATURE	Hardware	Measures the temperature of the device in degrees Celsius (°C). This sensor implementation varies across devices and this sensor was replaced with the TYPE_AMBIENT_TEMPERATURE sensor in API Level 14	Monitoring temperatures.

Appendix D. Sensor availability by platform.

Sensor	Android 4.0 (API Level 14)	Android 2.3 (API Level 9)	Android 2.2 (API Level 8)	Android 1.5 (API Level 3)
TYPE_ACCELEROMETER	**Yes**	**Yes**	**Yes**	**Yes**
TYPE_AMBIENT_TEMPERATURE	**Yes**	n/a	n/a	n/a
TYPE_GRAVITY	**Yes**	**Yes**	n/a	n/a
TYPE_GYROSCOPE	**Yes**	**Yes**	n/a[1]	n/a[1]
TYPE_LIGHT	**Yes**	**Yes**	**Yes**	**Yes**
TYPE_LINEAR_ACCELERATION	**Yes**	**Yes**	n/a	n/a
TYPE_MAGNETIC_FIELD	**Yes**	**Yes**	**Yes**	**Yes**
TYPE_ORIENTATION	**Yes**[2]	**Yes**[2]	**Yes**[2]	**Yes**
TYPE_PRESSURE	**Yes**	**Yes**	n/a[1]	n/a[1]
TYPE_PROXIMITY	**Yes**	**Yes**	**Yes**	**Yes**
TYPE_RELATIVE_HUMIDITY	**Yes**	n/a	n/a	n/a
TYPE_ROTATION_VECTOR	**Yes**	**Yes**	n/a	n/a
TYPE_TEMPERATURE	**Yes**[2]	**Yes**	**Yes**	**Yes**

[1] This sensor type was added in Android 1.5 (API Level 3), but it was not available for use until Android 2.3 (API Level 9).

[2] This sensor is available, but it has been deprecated.