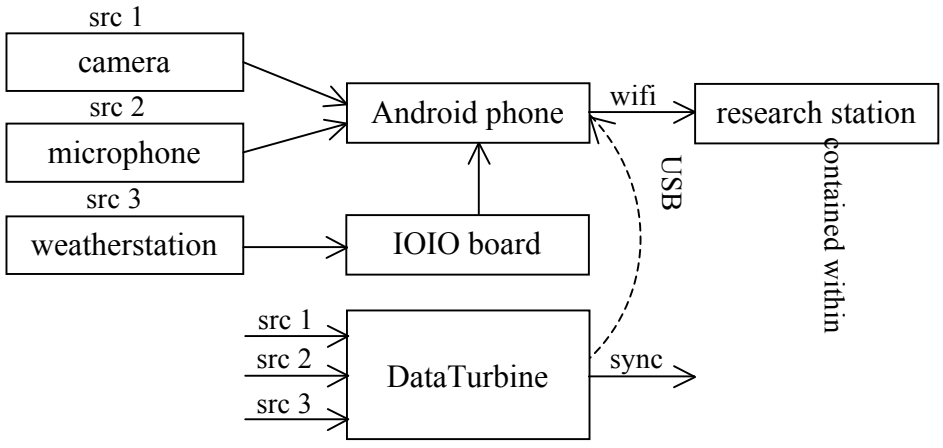
Ariana Tsai
PRIME 2012
Android Sensor Pod

Overview

The goal of the project is to set up an Android-based system in the Taiwanese rainforest that will collect and send back data gathered from various sensors to be analyzed. The analyzed data can then be used for protecting the rainforest from natural disasters like wildfires and for observing different animals and their behaviors. System parts include:

- an Android phone (Samsung Galaxy Nexus i9250) for taking pictures of the forest canopy and audio samples of various animals (for example, bird calls)
- a weatherstation (Vaisala WXT520) for gathering data on humidity, temperature, wind direction, pressure, and precipitation
- a IOIO board as a central hub for sensors
- solar panels to provide power to the system
- DataTurbine, which implements a ring buffer (where older data will be replaced first with newer data if the buffer is full – very convenient in case of a network or electricity outage), to sync (send out) sensor data.



The system is set up in the Lienhuahchih (LHC) research center. Originally, of the five towers in LHC it was supposed to be set up on the actual Tower 3, but for reasons mentioned later, it has been set up on a patch in front of the tower. This system is expected to stay online for the entire day, and thus must depend on solar power as a continuous energy source. Camera feed is currently captured every three seconds, whereas two minutes of audio (mono, sampling rate 44.1 kHz, also known as CD quality) are captured every fifteen minutes. The baud rate (similar to sampling rate) of the weatherstation is set to 9600, and sensor data is requested every one minute. When using a visualization tool to view the sensor data, one would receive a data point every minute.

The phone is charged by a USB connected to the IOIO board, which in return receives power from the solar panel set up that has been implemented. Currently, the phone captures media through use of its camera and headset, but in the future, the microphone can be exchanged with others when needed. (The microphone should optimally have amplification abilities as to be able to capture specific sounds like bird calls and minimize background noise.)

My part in the project focuses on maintaining code written for the phone to manage sensor data (collecting and sending). The code is managed using the Eclipse editor, and the Java Runtime Environment (JRE) 1.6 in particular is used, as it is the only JRE version that is compatible with IOIO coding. Furthermore, the code was run with the build targets Android 4.0.3 and Google APIs 2.3.3. There are three main applications installed on the phone:

- dtControl, which is in charge of capturing images and audio (upon starting the application, you can choose what parts of the application you want to start, including the phone's DataTurbine); the intervals (in milliseconds), sampling rate (in Hz), and

recording periods (in milliseconds) can be set by changing the values in DT.xml (found in the assets folder)

- DataGather_TFRI, which gathers from the on-board sensors and weatherstation and saves them to a text file ending in dataLog on the SD card (one file for each time the application is run); the baud rate and IOIO pins can be set by changing the code in DataGather_Service.java (found in the src folder)
- DataLineProcessor4RemoteDT, which focuses on parsing the data in the aforementioned data logs; in order for this to run properly, dtControl must have been started, as the application needs to access the phone's DataTurbine to sync the data.

If more sensors need to be added, this can be done in Configurator.java (found in the src folder of the library CleosAndroidLib). There are three arrays located in this file: chNames (channel names), dtypes (data types), units (data units), and MIMES (data formats). Each of these arrays must be updated accordingly when adding new sensors.

After the three aforementioned applications have been started, the phone will begin receiving and syncing the sensor data. This data is sent to the LHC server room through wifi and local network and can then be mirrored to other servers, in particular, the TFRI HQ servers.

Problems Faced

1. How do I even compile the code?

   I originally installed JRE 1.7 to run the code, which was a big no-no. Thanks to this, none of the libraries of code would compile correctly. My mentors and I then found out that IOIO coding requires JRE 1.6, and so I uninstalled 1.7 and installed 1.6 instead. However, many of the basic Java functions like printing "Hello World" would still not compile

properly. What was the problem? Eclipse was still attempting to run JRE 1.7 by default. It took a long time to figure that out, since I assumed that by installing a new version of JRE, Eclipse would update as well.

2. The battery keeps draining too quickly!

   The Samsung Galaxy Nexus has a 1,750 mAh Li-Ion battery, which means that by just leaving the phone on standby, it has enough energy to last around 264 hours. During my meetings with my mentors at UCSD, I was told that the battery of the Android phone would be a big problem. When they had tested the system, it would crash after around fourteen hours due to insufficient energy. I also met similar results when I tested. One instance showed that the battery drained by 8% within thirty minutes (charging with the computer, max brightness, screen never slept, dtControl with audio and image capture). Two similar data points:

   - 19% battery; 7:38:50 PM PST
   - 13% battery; 8:13:42 PM PST

   Upon doing research on how to conserve battery, I found that the top things that affect batteries the most are wifi, Bluetooth, GPS, and phone screens. The best ways to conserve battery turned out to be to…

   - change the brightness to the minimum – no problem, since after the system is set up, the phone screen does not need to be used anymore, and the screen actually takes up most of the power (upon checking the battery option in the phone settings, the screen usually takes up 83% or more of the battery)
   - change the sleep time to 15 seconds for the same reason above

- uninstall unneeded apps – for example, once the system is set up, the Chrome or Gmail applications won't need to be used anymore
- disable Bluetooth; the phone only uses wifi for transmitting data

Tests after these changes were made showed that the battery stopped draining more than it was being charged. It would also be preferable for the phone to charge only when it has 10% or less battery remaining, but this is too inefficient for the current system and only affects the overall battery life, rather than individual sessions of usage.

3. How should the weatherstation be configured?

   Throughout the project, two different models of weatherstations were used – one was Vaisala WXT510 (older model), and the other was Vaisala WXT520 (newer model that is now being used) The newer weatherstation not only had to be wired internally, but the settings and sampling rates also had to be configured through use of the Vaisala Configuration Tool, a program provided along with the weatherstation, or through use of a terminal emulator. At first, I used the Vaisala Configuration Tool to try and change the settings and sampling rates. This program attempts to connect to the weatherstation through a USB and default port COM1. Here lay the first problem – my COM1 was already in use by another device, so the program refused to let me connect to the weatherstation. After freeing the port and attempting to change the connection settings several times (the program is still rather buggy and does not always work), I finally succeeded in connecting to the device. Then, I changed the sampling rates to one minute – and here was the second problem.

   DataProcessorLine4RemoteDT requires the data log to be in a certain format and order (this can be seen in Configurator.java in the library CleosAndroidLib). All relevant

sensor data must be located on one line, every minute. The Vaisala Configuration Tool did not change all of the sampling rates, so the data log would show different sensor data on different lines all over the place. Because of the terrible experience I had with the program, I decided to try the alternative way of configuring the weatherstation – through use of a terminal emulator. Having never used one before, I thought that this would be a complicated way of configuring the weatherstation, but it turned out to be a more straightforward and relatively pain-free way of doing so. All that was required was sending ASCII commands through the terminal (I used HyperTerminal) and checking which sensors on the weatherstation were set to gather data.

4. How can the headset be replaced with an external microphone of better quality?

   The headset's microphone quality is not the best and definitely not optimal for recording sounds such as bird calls. As such, Dr. Wang requested that I experiment with a new setup. First, I connected a stereo Sony recorder, which has amplifying capabilities, to the phone and set it to a line out option in the settings. The phone would then detect the recorder as a microphone, allowing the attachment of another external microphone to the recorder for even better sound quality. However, it turns out that the phone jack had a different number of bands than the Sony recorder, so this setup was unable to be used. Nor is the current code capable of handling the stereo capability of the Sony recorder . The power of the Sony recorder proved to be another potential problem; it could be charged by being connected to the IOIO board, but the effect on the overall system power was not clear.

5. Where should the system be located?

The system setup incorporates solar panels, and therefore should be located near or at the top of Tower 3. However, lightning protection and earthquake protection for the solar panels had to be taken into consideration, so the system ended up being placed in a clear patch at the base of the tower. But the phone also must be able to take pictures of the rainforest canopy, leaving only two options: leave the phone in a separate area on top of the tower or leave it with the rest of the system below (and not take pictures). If the phone was left on top of the tower, a box would need to be constructed to protect it from lightning or other weather conditions. However, this would have to be a customized box because of the awkward positioning of the camera on the phone; the phone would have to be in an upright position, slightly slanted downwards. A wire long enough to connect the phone and system together would have to be found, too. To leave the phone on the top of the tower was deemed too risky (lightning may strike and damage the phone no matter what) and too inefficient (due to the need to design customized boxes). In the end, the phone was left with the rest of the system and unable to take pictures.

Next Steps

1. Instead of using an Android phone, a PandaBoard could be used for managing sensor data. The PandaBoard is a cheaper alternative to the Android phone that runs on a Linux system, although it does not come with a built-in camera.
2. Since there is no good way to deal with the camera capture function of the phone yet, it may be easier to first transmit IP camera images to the phone, then use the phone to send the transmitted images as originally intended. The IP cameras are already set up, and it

seems relatively simple to forward their images to an Android phone, as there are already some public applications that do so.

3. Perhaps a better data viewing tool would be better for testing and would certainly be better for analyzing. To test the audio capture, I use Real Time Data Viewer (RDV), but this program seems to playback all audio at 8 kHz. I had to separate the recording functionality in dtControl into another application temporarily in order to ascertain that the audio was being recorded at 44.1 kHz. When analyzing the data, it will definitely be important to have a better tool for listening to the audio.

Thoughts and Conclusions

Before this project, I had absolutely no experience with Android programming – I did not even own a smartphone before I started, but I did have much interest in mobile development. Thanks to this project, I was finally able to try it out and find that it was as fun as I imagined it would be. Moreover, I had never used many features of Eclipse before this project. I have gained much experience and knowledge from this experience, and would like to acknowledge the National Science Foundation, PRIME, and TFRI once again for giving me the opportunity to work on this project.

References

1. Nekrasov, Michael. Tutorial on DataTurbine. CaliT2. April 2012 through May 2012.
2. Shin, Peter. Tutorial on Android and Software. CaliT2. April 2012 through June 2012.
3. *Taiwan Forestry Resarch Institute*. 21 July 2010. <http://tfri.gov.tw>
4. "Documentation". *Open Source DataTurbine Initiative.* April 2012. <http://www.dataturbine.org/content/documentation>