Project Report: Active Pipeline Encroachment Detector (Phase I)

Jim Ji, PhD, Electrical Engineering Department, Texas A&M University

Andrew K. Chan, PhD, Electrical Engineering Department, Texas A&M University

Leslie E. Olson, National Pipeline Safety and Operations Center, Texas Transportation Institute

1. Introduction

Of the many pipeline accident causes that occur to oil and gas pipelines, approximately 40% of are caused by third-party excavating activities into the buried pipeline right of way (ROW). According to DOT statistics, excavation damage is the second leading cause of accidents for hazardous liquid pipelines, after corrosion. In particular, the potential for mechanical damage to be inflicted to pipelines located in or near urban expansion areas has become a major concern for the pipeline owner/operators and government regulators charged with the safety of the community (1,2). Existing encroachment detecting systems such as the ThreatScan acoustic system, aerial surveillance using automated drones, video cameras and infrared detectors either are too expensive or cannot work continuously on battery for more than a few days/weeks.

The long-term goal of the project is to develop a low-cost, reliable, long-life, wireless sensor system to identify and report pipeline encroachment activities in a localized area. Our short term objective of this project is to study and prove the feasibility of such system, and identify the technical and economic hurdles/compromises to achieve the aforementioned long-term goal.

This document describes our preliminary work in the first phase of the project. Specifically, we will focus on the device, software, and test developed to detect and differentiate typical encroachment activities (such as digging by backhoe) from common activities on a construction sites (such as those from trucks and jack hammers). It will show that the wireless sensor network approach should be viable for detecting right-of-way (ROW) intrusion by digging equipment, though significant amount of work is needed to develop a fully functional prototype.

2. Milestones and Timeline

The project was performed in a relative short period, from the end of June to end of December. During this time, we were able to successfully identify the sensor network

platform and sensor type for the project, and perform initial tests to demonstrate its feasibility. A conference abstract based on the work was submitted to the International Pipeline Safety Conference. The table below lists the timeline of "significant" progress in this initial project phase.

Date	Development	Issue
6/10/08	Project started -- specified technical requirements and scope of device/equipment needed	Sensitivity, battery life, and cost
6/16/08	**Identified and purchased Sentilla Perk Wireless Sensor Kit**	Cost effectiveness and programmability
7/20/08	**Initial software development and testing for door closing and opening, i.e., strong vibrations, and reporting to the host computer by email**	Missing activities due to low sensitivity and restricted software access
7/17/08	Purchased and tested various Trossen and radioshack external sensors	Not compatible with Tmote
7/30/08	**Acquired Sentilla Development**	
9/15/2008	Submit conference abstract to pipeline 2009 based on the preliminary work	See Appendix A
10/15/08	**Developed software using the Sentilla Development Kit to achieve much improved sensitivity**	GUI; signal delay
11/11/08-12/11/08	**Tested and verified battery life**	Prediction based on regression
12/10/08-1/20/09	**Field tests on construction sites; data processing and analysis**	
	Report	

3. System Architecture

The general approach for this research is to take advantage of the recent advance in commercially available, low-cost, long-life wireless sensor network for our application. The remote sensors will be placed at the potentially "dangerous" areas such as construction sites. The sensors will detect the vibration signals due to heavy machinery, which are transmitted to a "host" laptop computer to recognize the signal signatures of

digging equipment and report to pipeline operator (see Figure 1). Due to limited time and resource, we focus on sensor nodes only in this project phase. In addition, our tests indicate that the on-board accelerometers are sufficient for detecting and differentiating typical construction equipments.

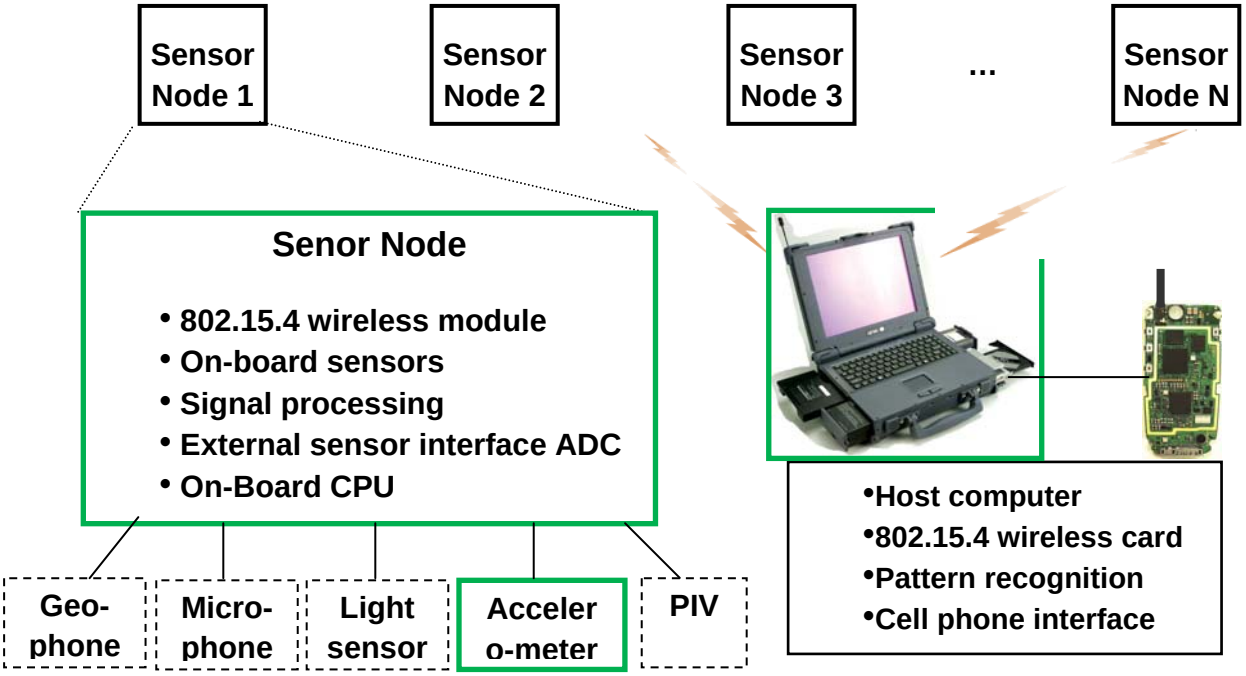


Figure 1. The architecture of the encroachment monitoring system.

3.1 Hardware

In our testing system, the host computer is a Dell Latitude Laptop. The wireless sensor nodes are Tmotes that consist of an 8 MHz TI MSP430 microcontroller, which provides them with sufficient processing power to analyze the data and send the results to the client computer. They are powered by two 1.5V AA batteries and contain a 250 Kbps 2.4 GHz IEEE 802.15 Chipcon wireless transceiver. The wireless receiver has an indoor optimal range of 125 meters and a data rate of 250 kbps. Each sensor unit currently equipped with a 3-axis accelerometer but has interfaces to connect to up to 8 other sensors. The accelerometer can detect vibration/motion as small as 0.1g acceleration.

Figure 2. The core of the Tmote wireless sensor node (after the hard plastic shell is removed). Courtesy: Sentilla

The accelerometer doubles as a vibration sensor for detecting the activities produced by heavy construction equipments. The unit is enclosed in a hard plastic shell to prevent damage from erosion. The Tmotes wirelessly transmit data back and forth between themselves and the host computer, which receive and display the accelerometer signals.

An important feature of the Tmote is that the sensor unit and wireless controller are normally kept in "sleep" state when there is no activation. This is done automatically after 30 seconds of loading the program onto the mote. A vibration threshold, defined by the user and completely adjustable, is preloaded so as to wake them up in 6 μs as soon as the threshold is reached. *With this feature, the unit can be installed and used without human intervention for a period of up to 6 months.* Section 4.1 will show the result of our battery testing and power consumption analysis.

3.2 Software

The Development Kit contains a Java-based software platform to control and program the entire mote network, e.g., the motes, the host computer, and their communication. The Development Kit provides access to the on-board Tmote software and allows programming the sensing unit through highly flexible interfaces.

- Wireless link or
- USB interface

JAVA is open source, object-oriented software which has a the desired tools for Graphical User Interface and also serves as a perfect union of modular and object oriented programming giving us the required flexibility. This is necessary because it is easily understandable, freely available and most importantly, flexible. In addition, change in one code or sensor does not affect the working of other sensors.

In our testing system, the necessary parameters, e.g., threshold, power levels, communication rates on Tmotes are setup through wireless link in the initialization stage. Then the mote goes to "sleep"/stand-by mode. Once an-overthreshold accelerometer signal is detected, the mote "wake-up" and transmit the signal to the host computer. As soon as the host computer receives the data, it is displayed on a graphics user interfact (GUI). The GUI shows the vibration/acceleration values on all the three axes in different colors. We will use this GUI plot to depict the kind and level of activity. We emphasis that in this project phase, only those software functions related sensor detection and communication are developed and explored. The intelligent signal processing and pattern recognition for reducing false alarms have not been developed.

Following is a list of major improvements we implemented on the system software. Appendix B shows the related software codes.

- Initial code written showing successful operation of motes waking up on detecting a certain level of activity and sending an email to the client
- Software scaling and threshold change to make accelerometer sensor more sensitive
- Using smaller bit sized buffer for faster data processing and reduced data transmission delay
- Implement animated GUI for displaying the dynamic data in a proper format on screen
- Steaming/storing data in CSV file from the GUI

4. Testing and Evaluation

4.1 Battery Life Testing

Battery life is a critical factor for the low-cost sensors. It is not practical to test the Tmote battery life for 6 months. *In our experiment, we observed and measured the battery use for about a month, and then predicted the total battery life using the data. Our test indicated that the sensor nodes should be able to run for at least 6 months on two AAA batteries.*

In order to test this, a software module was developed to subject the motes to the battery tests. The software continuously estimates the energy consumption of the system. In the software runs on each Tmote node and estimates the energy consumption, and send the data in each second to the host PC through the Java program interface. The Java program shows the nodes' power consumption for the last second.

When pushing the button on the nodes, they cycle through seven states as below. This is reflected by their power consumption, as shown in the Java program. The different states are:

- Red LED: sending one packet per second
- Green LED: radio listen 1% duty cycle
- Green, red LEDs: radio listen 10% duty cycle
- Blue LED: radio listen 100%
- Blue, red LEDs: radio listen 10%, CPU low-power mode disabled
- Blue, green LEDs: sending data 1.2 kilobytes/second
- Blue, green, red LEDs: sending data 12 kilobytes/second

When sending data, the radio is turned on for a while before the transmission to check if it is possible to send the packet. This is the reason why energy is spent on radio listening even when the nodes are only sending data. Figure 3 shows the typical power consumption of 6 nodes and the distribution of power use.

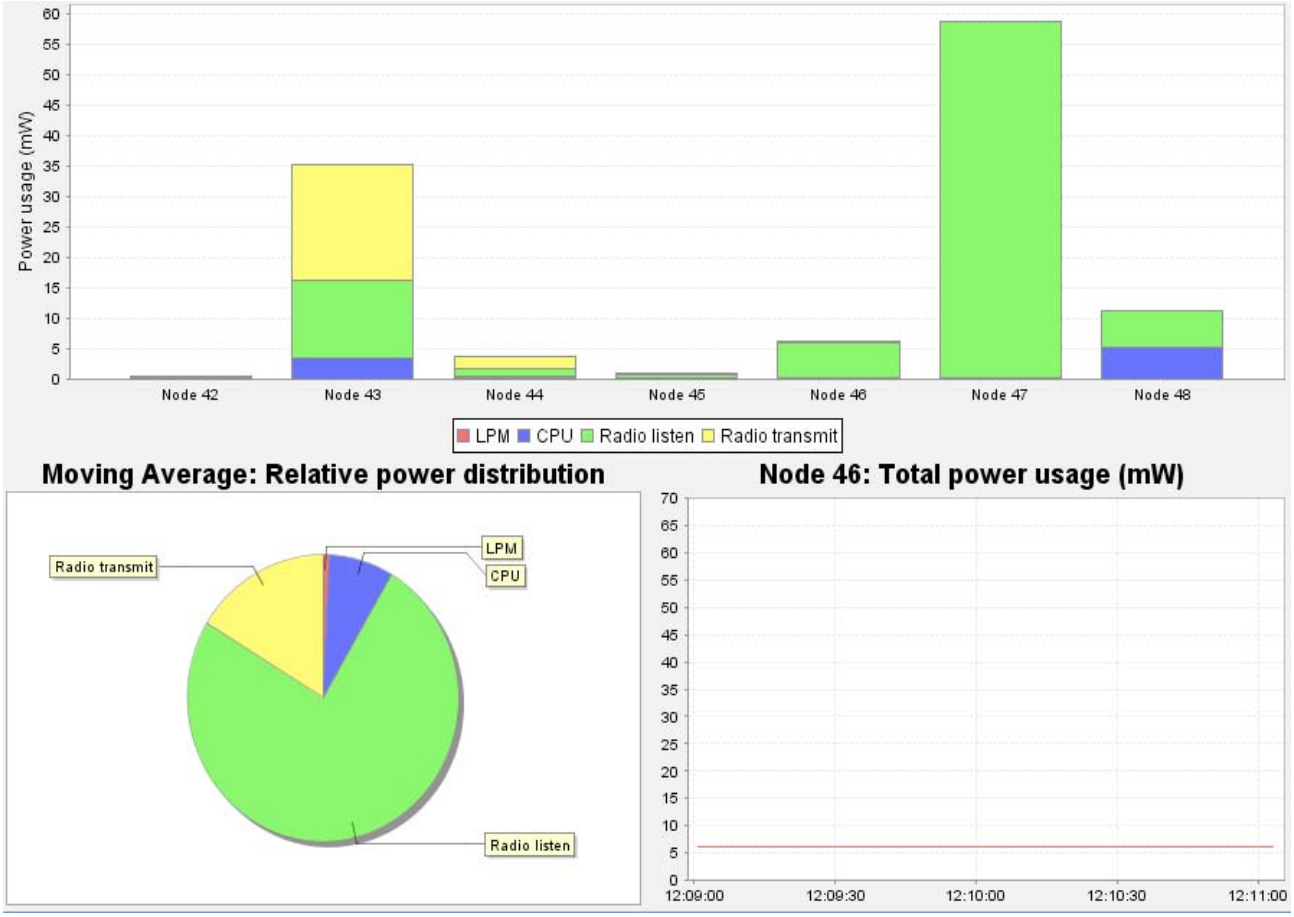


Figure 3. Typical power consumption of 6 nodes (top); The distribution of power use (bottom left); and average power use for one particular node.

The energy estimation mechanism uses a linear model for the sensor node energy consumption. The total energy consumption E is defined as

$$E / V = I_m t_m + I_l t_l + I_t t_t + I_r t_r + \Sigma I_{ci} t_{ci} \quad (1)$$

In the above equation, V is the supply voltage, I_m and t_m the current draw and the running time of the microprocessor. I_l and t_l the current draw and the time of the microprocessor in low power mode, I_t and t_t the current draw and the time when transmitting data, I_r and t_r the current draw and time of the communication device when receiving, and I_{ci} and t_{ci} the current draw and time of other components such as sensors and LED's. The energy model does not contain a term for the idle current draw of the board itself. This is embedded in the low power mode draw of the microprocessor.

Figure 4 shows the power consumption of motes running the data collection software. The left graph shows reading without optimization and the right graph shows them with optimization. As we can see from the above graphs, the power consumption is much less when the motes have been optimized. The states were performed repeatedly over a period of 4 weeks with optimization.

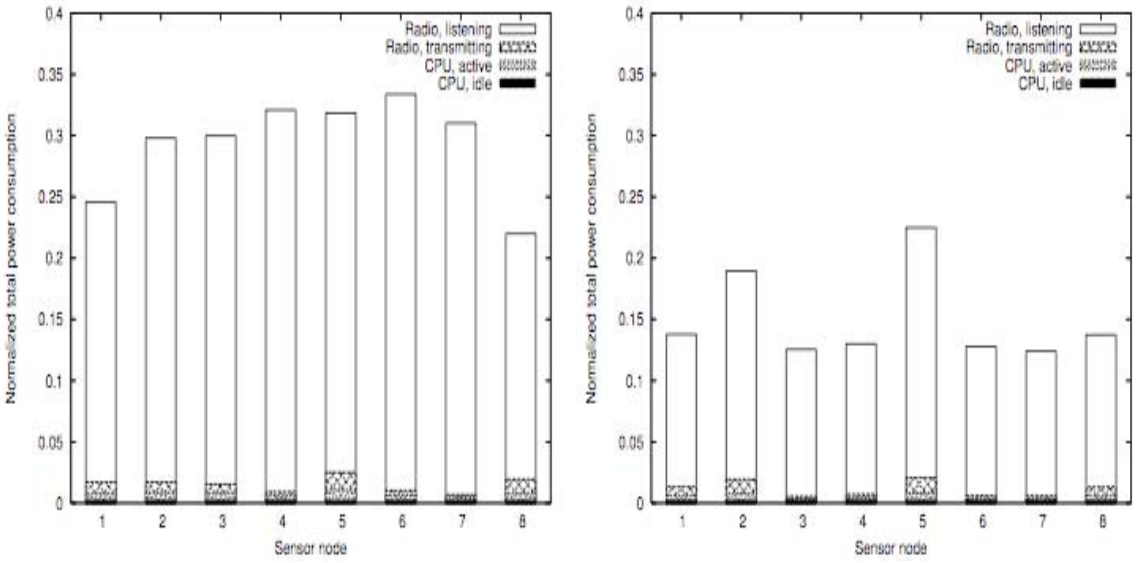


Figure 4. Power consumption of 8 motes. (Left) without optimization and (Right) with optimization.

The actual battery voltage over a period of one month was recorded and shown in Figure 5.

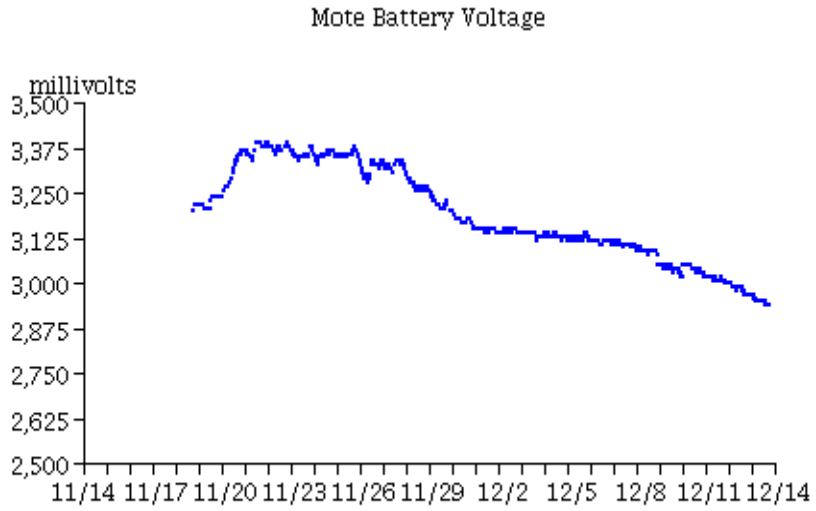


Figure 5. Battery voltage variation of a wireless sensor node during the month from November 14 and December 14, when the node was actively used for testing.

The low-voltage threshold for the Tmote to work is about 2.1V according to the specification. As we can see from the figure, the motes are at all intervals are above this threshold even after a month of heavy use as described above and in the next subsection. More importantly, the rate of drop is approximately 0.4V/month. Therefore the estimated battery life at this rate will be (3.5-2.1)/0.4 moth, or about 3.5 months under the heavy use condition. *Combining the information from Figs. 3-5, it is estimated that the battery life will be at least 6 months if the Tmote is mostly kept in "sleep" state, as in practical application condition, given that only a fraction of power is required under such condition.*

4.2 Field Testings

Three on-site tests were performed to acquire vibration signals from three typical construction equipments: Jack Hammer, Backhoe, and Heavy Truck. In each test, the signals were acquired at two distances to the equipment: 4.5 feet and 1.5 feet. Since the sites were in different phases of construction, no single site contained all the commonly used construction equipment. As a result, multiple sites on the campus of Texas A&M University (as shown in Sections 4.2.1, 4.2.2, and 4.2.3) were chosen for testing.

All tests were performed on a clear day with temperatures ranging from 25 ^{0}C to 32 ^{0}C. A Tmote was placed on soft porous surface by a hard – hat construction worker at the specified distance to the equipment to be tested. This was because we were not allowed to enter the construction area due to safety concerns and OHSA criteria.

The Laptop and Tmote were setup as described in Section 3.2. It involved starting the Sentilla Suite by clicking on its icon and plugging the USB Wireless Receiver into the host computer. The user is then loaded the mote software onto the various motes automatically connected on the network in the 'mote section' of Sentilla Suite. At the same time, the client software is loaded into the 'client section' of the above suite. This was required to setup proper communication and data transmission between the motes and client, but only has to be done once at the beginning of the application.

A single program code was used for process the data from all the on – site testing. The code was completely written in Java and provides a sensitivity of – 5.7g to 4.5g for detecting vibrations produced. The data is acquired at the rate of 5 Hz and maybe increased to up to 10 Hz if more samples are needed. However, it was noticed that larger the sampling rate, the more the latency. This latency however reaches close to

zero as time increases.

4.2.1 Jackhammer

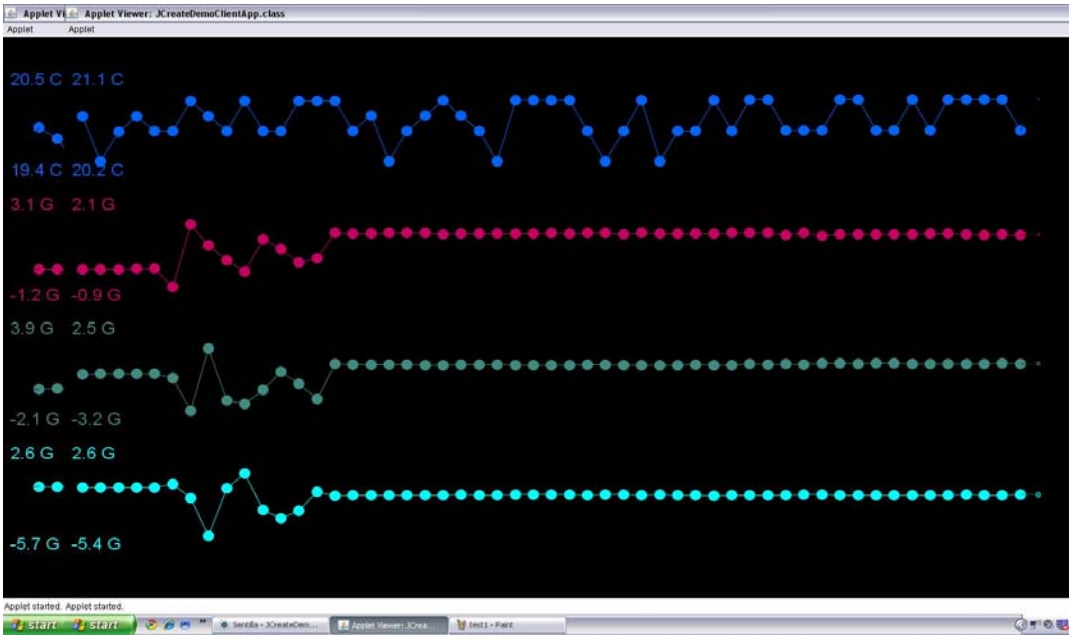
Site I: Ross Street

Date: Dec. 10, 2008

Jackhammer is one of the cheapest options available in the market for digging through concrete surfaces. It is under concrete surfaces in the city where most of the pipelines are dug and the workers, at times, are not aware of these pipelines.

Tmote distance from the equipment: 1.5 feet

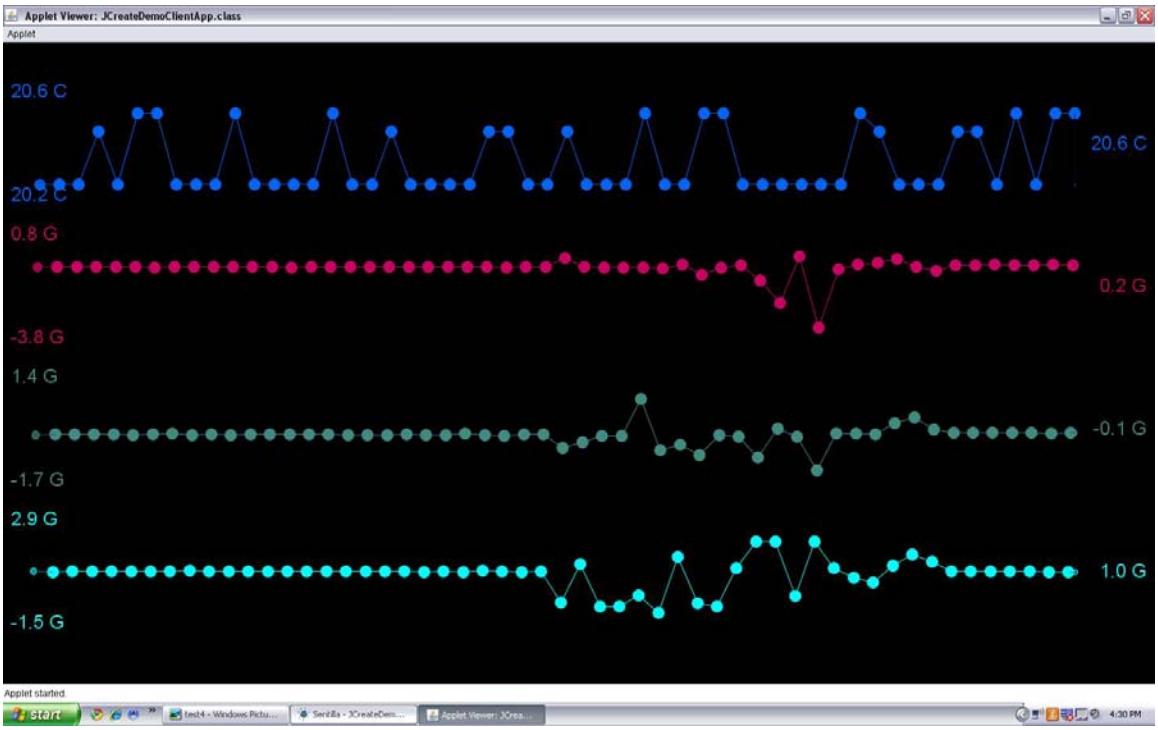
The follow graph shows a snapshot of 10-second capture sensors signals from the Tmote. In all signals graphs, the top (blue) is the temperature, and the other three correspond to the three axes of the accelerometer: X in red, Y in green, Z in cyan



In the test, the jackhammer was initially off, then turned on for about 2 seconds, then off again. It is clear from the graph that the vibration were detected and reflected in the sudden spikes in all the three (X is pink, Y is green, Z is cyan) axes. When the jackhammer was turned off, straight lines were observed indicating that there was no activity.

Tmote distance from the equipment: 4.5 feet

The following graph is from the test where the Tmote was kept at a distance of 4.5 feet away from the Jackhammer. It shows that the Tmote was able to pick up the vibrations but after a delay of 2-3 seconds. The delay may be due to the delayed Tmote “wakeup” or communication loss, which we expect to be able to eliminate in the future. It is seen that the vibrations readings picked up by the motes decreased by a only a slight amount. This can be attributed to the fact that some energy has been lost by the waves as they travel further.



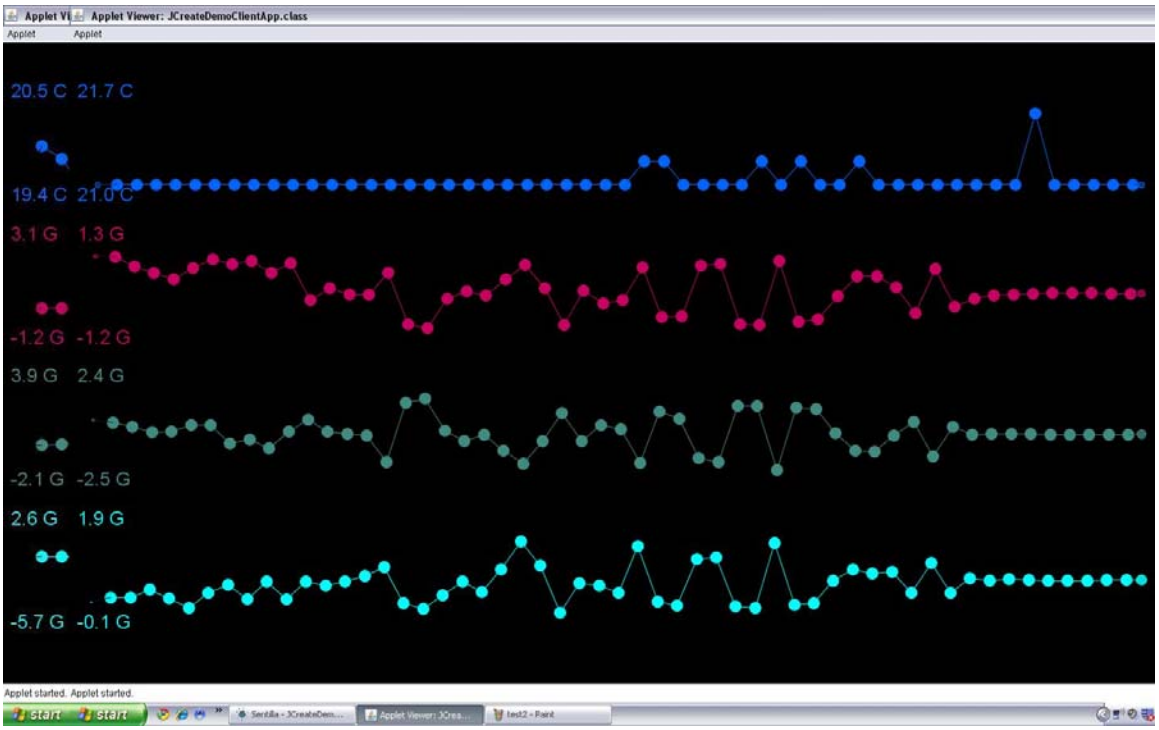
4.2.2 Backhoe

Site II: Zachary Parking Lot

Date: Dec. 12, 2008

A backhoe was chosen as equipment for testing since it poses danger to pipelines. They are powerful earth moving equipment that can cause significant damage.

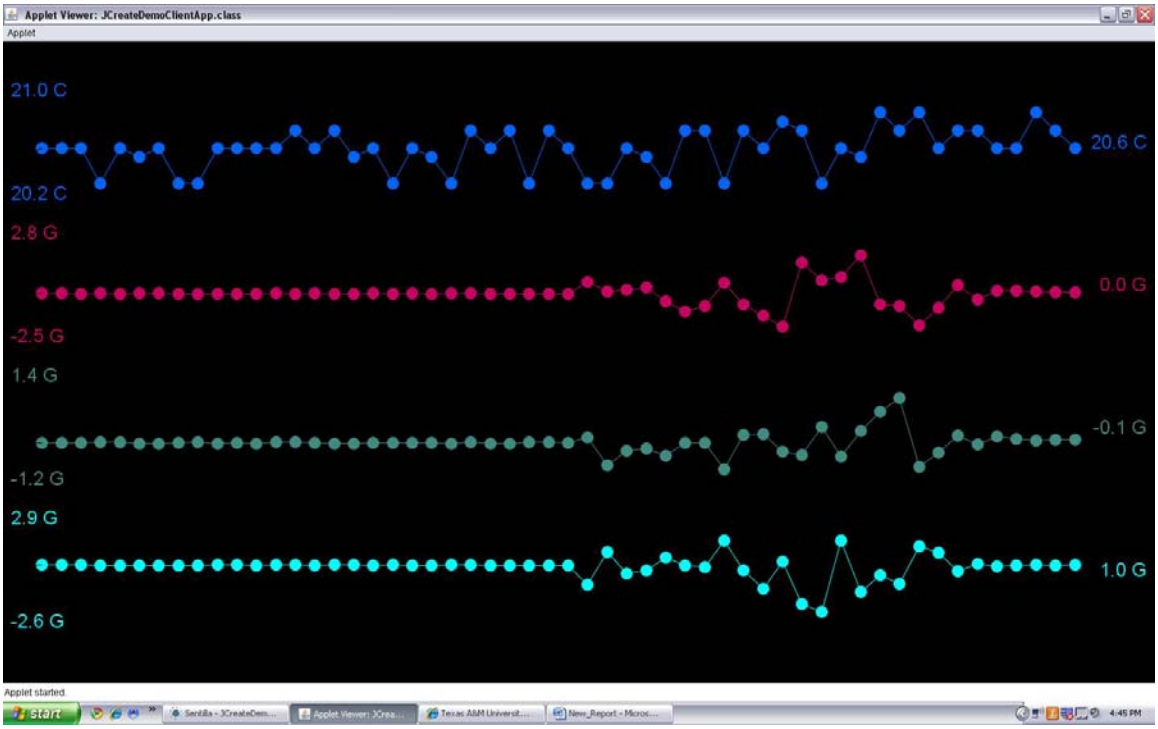
Tmote distance from the equipment: 1.5 feet



The graph shows a slow pattern of vibrations as the backhoe approached the ground with a sudden activity increase in the middle as soon as it started applying forces to dig

deeper. Towards the end, there was a sudden drop and stabilization of readings on all the axes. This indicates that the backhoe has finished one digging cycle.

Tmote distance from the equipment: 4.5 feet



At this distance, we again observe a delay of signal startup, however, the signal intensity is only slightly reduced as comparing to that of at 1.5 feet.

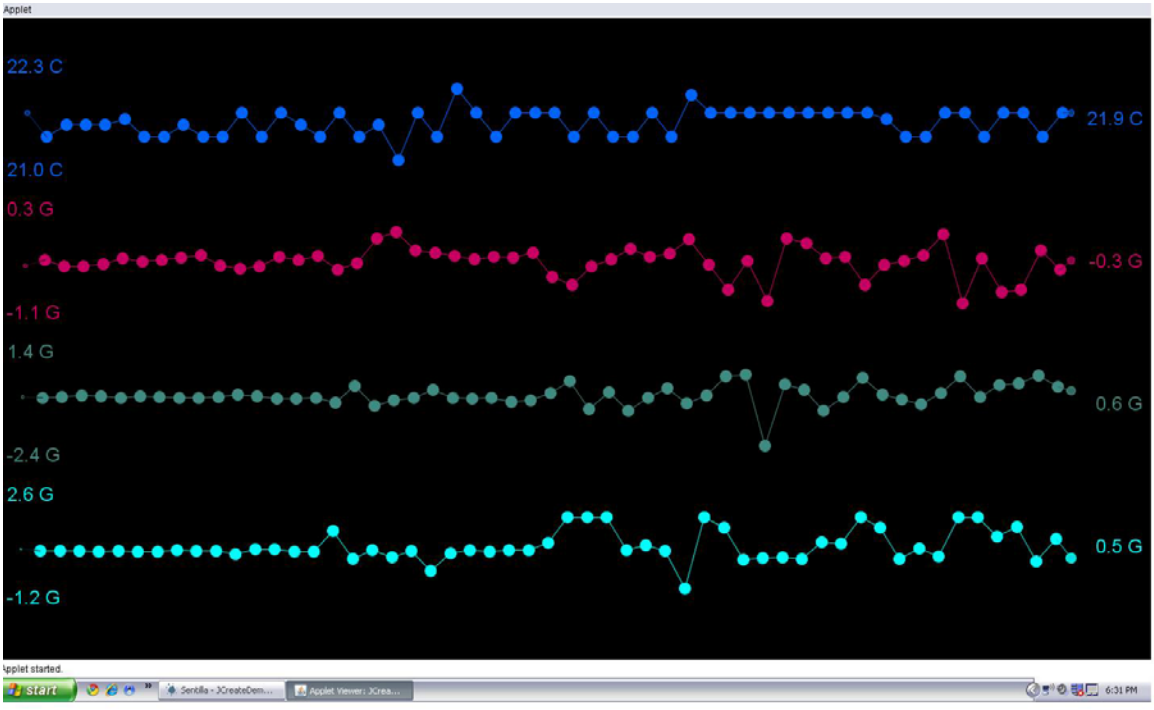
4.2.3 Construction Truck

Site III Physics Building on the University Drive

Date: Dec. 12, 2008

Trucks, as we all know are one of the most common construction site vehicles but they pose no harm to the pipelines. This experiment is designed to test whether the sensor signals from the truck can be differentiated from other construction equipments. It is important because we desire the motes not provide any false alarm.

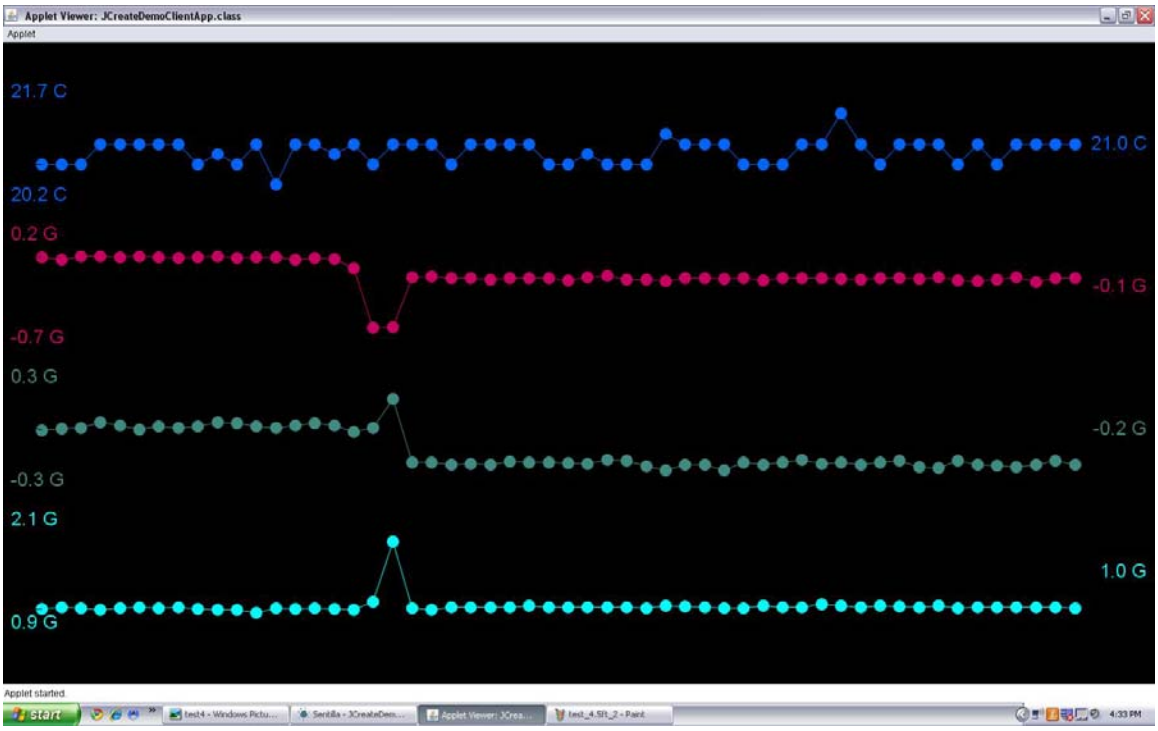
Tmote distance from the equipment: 1.5 feet



The graph shows a very steady increase in the activity pattern. This demonstrates the

engine starting in a truck. As the truck steadily accelerates from zero to some speed, we see a steady increase in the readings. As it goes away from the Tmote, the vibrations start reduce towards the end.

Tmote distance from the equipment: 4.5 feet



It is evident that the heavy diesel engine truck produces much less detectable vibrations at this distance. It seems that these vibrations decay quickly in distance. As a result, when motes are placed at a further distance from the equipment, the spikes were observed only when the engine was started.

In summary, these on-site tests show that: 1) there are sufficient signal signatures to differentiate the three equipment types; 2) intelligent signal pattern recognition is required to use these signatures; 3) the signals acquired at 1.5 feet and 4.5 feet have similar intensity, indicating that the vibration signals do not decay very fast over distance.

5. Conclusion and Future Work

We have completed the initial development and validation of the wireless sensor technology we proposed to use to detect ROW intrusion by construction type equipment. We developed and tested the technique on a single wireless sensor and a laptop computer, and successfully demonstrated its feasibility. *The outcomes of our testing show clearly that our approach should be viable for detecting ROW intrusion by digging equipment. Specifically, we conclude that: 1) the battery life of the low-cost sensor can be more than 6 months based on our test. 2) the on-board, sensitive accelerometer can capture sufficiently signals to detect and differentiate three types of construction equipments (Jackhammer, Backhoe, and Truck) at a distance of 1.5 feet and 4.5 feet.*

These initial results were extremely encouraging and exciting. However, significant challenges and work remain to further develop the technique to produce a fully-functional prototype that can be brought to pipeline operating companies for field testing. These include but not limited to: improving reliability and validating the timing accuracy of captured signals; testing at extended distance; developing automatic training algorithm to tune the event detection parameters; improving the package of the motes, the robustness of software; scaling up the sensor node numbers (50 to 100 nodes) to form a larger array capable of operating autonomously for an entire season of six months or more in the field; developing intelligent pattern recognition program on the host computer to reduce false alarms; and system integration.

References:

1. M. Buettner, G. V. Yee, E. Anderson, and R. Han. Xmac: a short preamble mac protocol for duty-cycled wireless sensor networks. In Proceedings of the 4th international conference on Embedded networked sensor systems, Boulder, Colorado, USA, 2006.
2. http://ncseonline.org/NLE/CRSreports/energy/eng66.cfm?&CFID=19199705&CFTOKEN=24439530
3. Altmann, J., “Acoustic and seismic signals of heavy military vehicles for co-operative verification,” Journal of Sound and Vibration, vol 273, no. 4, 713-740, 2004.
4. Arora, A. and Dutta, P. and Bapat, S. and Kulathumani, V. and Zhang, H. and Naik, V. and Mittal, V. and Cao, H. and Demirbas, M. and Gouda, M. and others, “A line in the sand: a wireless sensor network for target detection, classification, and tracking,” Computer Networks, vol. 26, no. 5, 605--634, 2004.
5. Crossbow Produce Reference http://www.xbow.com/Support/Support_pdf_files/Product_Feature_Reference_Chart.pdf

6. Wireless sensor networks, Nirupama Bulusu, Sanjay Jha, ed., Artech House, Boston, 2005

Appendix A: Abstract Submitted in September 2008 to Pipeline Safety Conference

Active Pipeline Encroachment Detector Using Wireless Sensor Networks

Leslie Olson, Harneet Singh, Andrew Chan, Jim Ji

Of the many pipeline accident causes to pipelines, approximately 60% of gas pipeline incidents are caused by outside forces (mechanical damage) from a digging or excavation activity. Pipeline industry statistics indicate that half of the accidents are caused due to incursion of excavating activities into the right of way. In addition, the potential for mechanical damage to be inflicted to pipelines located in or near urban expansion areas has become a major concern.

A recent technical advance is the ThreatScan system which relies on acoustic waves traveling through the product in the pipeline. However, installation and maintenance of such systems are expensive. Other security surveillance technologies have been developed or are being investigated. These technologies such as aerial surveillance using automated drones, video cameras and infrared detectors cannot work continuously on battery for more than a few days/weeks. Therefore, low cost, long-life, self monitoring sensors are desirable for localized installation and monitoring of pipeline right-of-way.

To address this problem, a pipeline encroachment detector system based on wireless sensor networks is developed. The system consists of a network of commercially available sensors, each connected to each other and a laptop client computer through IEEE 802.15.4 wireless link at up to 256 kbps rate. Each sensor unit is equipped with a 3-axis accelerometer. The accelerometer can detect motion as small as 0.1g acceleration. The sensor unit and wireless controller are set in a “sleep” mode. Using two standard AAA batteries, the unit can be installed and used without human intervention for a period of up to 6 months. The entire system is coupled with a 16-bit, 8 MHz TI MSP 430 microcontroller, which enables it to acquire and distinguish data from different sensors and make intelligent decisions before relaying the information to the client through a wireless link. The information relayed is displayed as waveforms on the screen via an Applet. This enables a visual representation of the data in real time at a remote location. An intelligent pattern recognition algorithm is applied to reduce the false alarms. The system can detect small irregular motion when attached to physical objects. Continued develop of the pattern recognition software and initial field testing is ongoing The results will be reported in the conference.

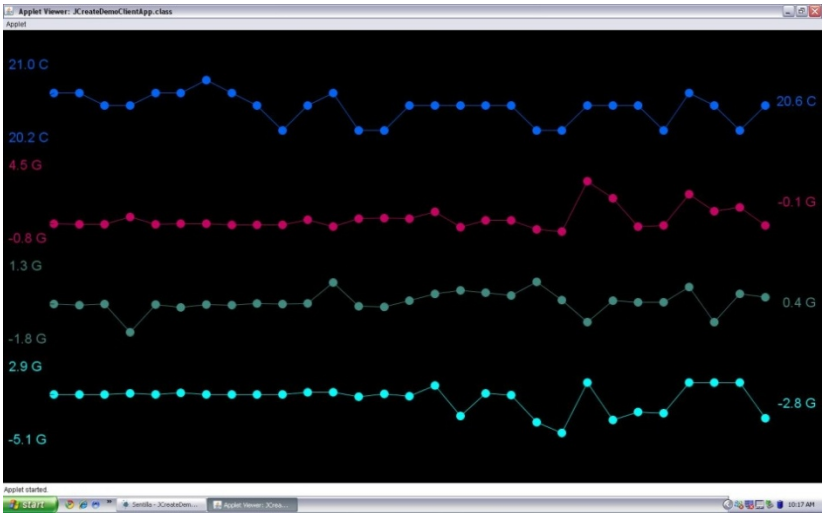


Fig. 1 Sensor data sampled during a static state period (left half) and a motion state period (right half). The sensor unit was put on a platform which undergoes a moderate motion. Data from the accelerometer are shown in red (x-axis), green (-axis) and cyan (z-axis).

Appendix B: Software Codes

Mote Code:

```
import com.sentilla.system.SensorDriver;
import com.sentilla.system.Sensor;

import com.sentilla.net.Sender;
import com.sentilla.net.SenderDriver;
import com.sentilla.platform.tmote.VoltageAdc;
import com.sentilla.platform.tmote.McuVoltage;
import com.sentilla.platform.tmote.McuTemperature;
import com.sentilla.net.*;

import java.io.Serializable;

import javax.measure.quantity.Acceleration;
import javax.measure.quantity.Temperature;
import javax.measure.quantity.ElectricPotential;
public class JCreateDemoMoteApp
{
        public static class JCreateMessage implements Serializable
        {
                long moteID;
                double temperature;
```

```
        double potential;
        double x;
        double y;
        double z;
}

public static void motemain()throws InterruptedException
{
        //Create Sender
        Sender sender = SenderDriver.create("local");

        //Create Sensors
        Sensor<Temperature> ts = SensorDriver.create("temp", Temperature.class);
        Sensor<ElectricPotential> ps = SensorDriver.create("volt",ElectricPotential.class);
        Accelerometer accel = new Accelerometer(true, true, true);
        accel.calibrate();


        //CreateMessage
        JCreateMessage jmsg = new JCreateMessage();

        //Get ID
        jmsg.moteID = Mac64Address.getLocalAddress().longValue();

        float[] aread = new float[3];
```

```
            while(true)
            {
                //Take the measurement;
                jmsg.temperature = ts.read().doubleValue(Temperature.UNIT);
                jmsg.potential = ps.read().doubleValue(ElectricPotential.UNIT);
                accel.getMultipleReadings(aread);
                jmsg.x = aread[0];
                jmsg.y = aread[1];
                jmsg.z = aread[2];



                //Send off
                sender.send(jmsg);

                //Sleep for 1 second;
                //Thread.sleep(500);
            }


        }



}
```

Client Code:

```
import com.sentilla.host.client.HostClient;
import com.sentilla.net.Receiver;
import com.sentilla.net.ReceiverDriver;
import processing.core.*;
import javax.measure.quantity.*;

public class JCreateDemoClientApp extends PApplet {

        //Instantiate basic components
```

```
Receiver receiver;

JCreateDemoMoteApp.JCreateMessage jmsg= new JCreateDemoMoteApp.JCreateMessage();

PFont font;
public void setup()
{

        // Set up the graph
        size(1000, 500);
        stroke(255);
        frameRate(30);
        background(0);
        font =  createFont("FFScala", 20);
        textFont(font, width/60);
        redraw();

        /* create the host server connection */
        HostClient host = new HostClient();
        try
        {
                host.connect();
        }
        catch (Exception e){}
```

```
                    //Start receiver
receiver = ReceiverDriver.create(JCreateDemoMoteApp.JCreateMessage.class);
                    receiver.cancel();
                    receiver.setReceive().submit();

          }

          //globals
          int arraysize = 30;
          int counter = 0;
          double[] xArr = new double[arraysize];
   double[] yArr = new double[arraysize];
   double[] zArr = new double[arraysize];
   double[] tempArr = new double [arraysize];
   double[] timeArr = new double[arraysize];

   //globals to help with animation
          float xposold;
          float yposold;
          float[] xpoints = new float[arraysize];
          float[] ypoints = new float[arraysize];

          public void draw()
```

```
{
        //if we've got a message get new data
        if (receiver.isDone())
        {
                jmsg = (JCreateDemoMoteApp.JCreateMessage) receiver.getData();
                receiver.setReceive().submit();

                addValue(xArr,jmsg.x);
                addValue(tempArr,jmsg.temperature - 273);
                addValue(yArr, jmsg.y);
                addValue(zArr, jmsg.z);
                addValue(timeArr,(double)System.currentTimeMillis());
                if ( counter < arraysize) counter++;
                animcounter = 0;


        }

        //draw plots, animate smoothly if we have new data
        background(0);
        smooth();
        plotlineanim(2*height/9,height/9,tempArr,0,100,255,"C");
        plotlineanim(4*height/9,height/9,xArr,200,0,100,"G");
        plotlineanim(6*height/9,height/9,yArr,65,140,130,"G");
        plotlineanim(8*height/9,height/9,zArr,0,255,255,"G");
```

```
        if (animcounter < animlimit) animcounter++;

}

public static void main(String args[]) {

        PApplet.main(new String[] { "--present", "EHVClientApp" });

}

//Effectively make a double a circular array
void addValue(double[] dubArr, double addVal)
{
        for (int i = dubArr.length-1; i > 0; i--)
        {
                dubArr[i] = dubArr[i - 1];
        }
        dubArr[0] = addVal;
}

//Simple plot a line, no animation
void plotline(float base, float maxheight, double[] dubArr, int r, int g, int b, String units)
{
```

```
float max  = (float)findMax(dubArr);
float min  = (float)findMinNotZero(dubArr);
float range = max - min;

float xpos;
float ypos;
float xposold;
float yposold;
float[] xpoints = new float[dubArr.length];
float[] ypoints = new float[dubArr.length];

//draw dots
for (float i = 0; i < dubArr.length; i ++)
{
        //Determine X Position
         xpos = (width-100) - ((arraysize - counter) + i)/((float)dubArr.length)*(width-
100);
         xpoints[(int)i] = xpos;

        //Determine Y Position, auto scale
         ypos = base-(float)((dubArr[(int)i]-min)/range * maxheight);
         ypoints[(int)i] = ypos;

         //Draw Dots
         ellipseMode(CENTER);
```

```
            fill(r,g,b);
            stroke(r,g,b);
            if (counter > (int)i) ellipse(xpos, ypos, width/100, width/100);

}

//connect dots
for (int i = 0; i < dubArr.length-1; i++)
{

            stroke(r,g,b);
            if (counter > (int)i+1) curve(
                        xpoints[i]*2/3+xpoints[i+1]/3,ypoints[i]*2/3+ypoints[i+1]/3,
                        xpoints[i],ypoints[i],
                        xpoints[i+1],ypoints[i+1],
                        xpoints[i]/3+xpoints[i+1]*2/3,ypoints[i]/3+ypoints[i+1]*2/3
                                    );
}

//Show current value
float now = (float)dubArr[0];
now = round(now*10);
now = now/10;
String out = String.valueOf(now)+" "+units;
fill(r,g,b);
```

```
        textFont(font, width/60);
        textAlign(RIGHT);
        text(out,width-10,base-maxheight/2);

        //Show min value
        max = round(max*10);
        max = max/10;
        out = String.valueOf(max)+" "+units;
        fill(r,g,b);
        textFont(font, width/60);
        textAlign(LEFT);
        text(out,10,base-maxheight-20);

        //Show max value
        min = round(min*10);
        min = min/10;
        out = String.valueOf(min)+" "+units;
        fill(r,g,b);
        textFont(font, width/60);
        textAlign(LEFT);
        text(out,10,base+20);

}
```

```
float animcounter = 0;
float animlimit = 500;

void plotlineanim(float base, float maxheight, double[] dubArr, int r, int g, int b, String units)
{

	//find max and min in range
	float max  = (float) dubArr[0];
	float min= (float) dubArr[0];
	float range;
	for (int i = 0; i < dubArr.length-1; i ++)
	{
		if (dubArr[i] > max) max = (float)dubArr[i];
		if (dubArr[i] < min && dubArr[i] != 0) min = (float)dubArr[i];
	}
	range = max - min;;

	//find old max and min
	float oldmax = (float) dubArr[1];
	float oldmin= (float)dubArr[1];
	float oldrange;
	for (int i = 1; i < dubArr.length; i ++)
	{
		if (dubArr[i] > oldmax) oldmax = (float)dubArr[i];
		if (dubArr[i] < oldmin && dubArr[i] != 0) oldmin = (float)dubArr[i];
```

```
		}
		oldrange = oldmax - oldmin;

		//Animate based on old values and new values, linearly interpolated
		float xpos;
		float xposnow;
		float xposold;
		float ypos;
		float yposnow;
		float yposold;


		//calculate points
		for (float i = 0; i < dubArr.length; i ++)
		{
			//Determine X Position
			xposnow = (width-100) - ((arraysize - counter) + i)/((float)dubArr.length)*(width-100);
			//Determine Y Position, auto scale
			yposnow = base-(float)((dubArr[(int)i]-min)/range * maxheight);

			//Old X-Point Determination
			if (counter == arraysize && i != 0)
			{
				xposold = (width-100) - ((arraysize - counter) + i-1)/((float)dubArr.length)*(width-100);
```

```
                    }
                    else
                    {
                            xposold = xposnow;
                    }

                    //Old Y-point Determination
                    if (i != 0)
                    {
                                    yposold = base-(float)((dubArr[(int)i]-oldmin)/oldrange *
maxheight);
                    }
                    else
                    {

                            yposold = yposnow;
                    }

                    //Weighted average slides as a function of animcounter
                    xpoints[(int)i] = ((animcounter*xposnow) + (animlimit-
animcounter)*xposold)/animlimit;
                    ypoints[(int)i] = ((animcounter*yposnow) + (animlimit-
animcounter)*yposold)/animlimit;
                    if (animcounter < animlimit)animcounter ++;
```

```
}

//Draw Dots
for (int i = 0; i < dubArr.length; i ++)
{



        ellipseMode(CENTER);
        stroke(r,g,b);
        //if point is new make the dot grow from nothing
        if (i == 0)
        {
                fill(
                                (float)r*(animcounter/animlimit),
                                (float)g*(animcounter/animlimit),
                                (float)b*(animcounter/animlimit)
                        );
                float rad = width/100 * animcounter/animlimit;
                if (counter > (int)i)        ellipse(xpoints[i], ypoints[i], rad, rad);
        }

        //if point is about to dying make it shrink to nothing
        else if (i == dubArr.length-1)
```

```
		{
			fill(
					(float)r*(1-(animcounter/animlimit)),
					(float)g*(1-(animcounter/animlimit)),
					(float)b*(1-(animcounter/animlimit))
				);
			float rad = width/100 * (1-(animcounter/animlimit));
			if (counter > (int)i)	ellipse(xpoints[i], ypoints[i], rad, rad);
		}

		//just draw the dot
		else
			{
			fill(r,g,b);
			if (counter > (int)i) ellipse(xpoints[i], ypoints[i], width/100, width/100);
			}


}

//connect dots
for (int i = 0; i < dubArr.length-1; i++)
{
		//if connection is new make line fade in
		if (i == 0)
```

```
{
        stroke(
                        (float)r*(animcounter/animlimit),
                        (float)g*(animcounter/animlimit),
                        (float)b*(animcounter/animlimit)
                );
}
//if connection is about to be lost make line fade out
else if (i == dubArr.length-2)
{
        stroke(
                        (float)r*(1-(animcounter/animlimit)),
                        (float)g*(1-(animcounter/animlimit)),
                        (float)b*(1-(animcounter/animlimit))
                );
}

//else just draw the line
else stroke(r,g,b);

if (counter > (int)i+1) curve(
                xpoints[i]*2/3+xpoints[i+1]/3,ypoints[i]*2/3+ypoints[i+1]/3,
                xpoints[i],ypoints[i],
                xpoints[i+1],ypoints[i+1],
                xpoints[i]/3+xpoints[i+1]*2/3,ypoints[i]/3+ypoints[i+1]*2/3
```

```
                                        );
}

//Show current value, smooth animation between (appears to scroll)
float now = (float)dubArr[0];
float then = (float)dubArr[1];
now = ((animcounter*now) + (animlimit-animcounter)*then)/animlimit;
now = round(now*10);
now = now/10;
String out = String.valueOf(now)+" "+units;
fill(r,g,b);
textFont(font, width/60);
textAlign(RIGHT);
text(out,width-10,base-maxheight/2);

//Show min value, same smooth animation
max = ((animcounter*max) + (animlimit-animcounter)*oldmax)/animlimit;
max = round(max*10);
max = max/10;
out = String.valueOf(max)+" "+units;
fill(r,g,b);
textFont(font, width/60);
textAlign(LEFT);
text(out,10,base-maxheight-20);
```

```
        //Show max value, smoof
        min = ((animcounter*min) + (animlimit-animcounter)*oldmin)/animlimit;
        min = round(min*10);
        min = min/10;
        out = String.valueOf(min)+" "+units;
        fill(r,g,b);
        textFont(font, width/60);
        textAlign(LEFT);
        text(out,10,base+20);

}

//rough attempt at 1D time-axis, imperfect
void plottime (float base, double[] timeArr)
{
        double timenow = timeArr[0]/1000;
        double timeold = timeArr[timeArr.length-1]/1000;
        String now = String.valueOf(timenow-timenow);
        double out = timeold - timenow;
        String old = String.valueOf(round((float)out))+"s";
        fill(255);
        textFont(font, width/60);
        text(old,width/(float)arraysize,base);
   text("0",width-width/(float)arraysize,base);
   stroke(255);
```

```
   line(width/arraysize,base-20,(arraysize-1)*width/arraysize,base-20);
}

double findMax(double[] dubArr)
{
        double dubMax = dubArr[0];
        for (int i = 0; i < dubArr.length; i++)
        {
                if (dubArr[i]>dubMax) dubMax = dubArr[i];
        }

        return dubMax;
}

double findMin(double[] dubArr)
{
        double dubMin = dubArr[0];
        for (int i = 0; i < dubArr.length; i++)
        {
                if (dubArr[i]<dubMin) dubMin = dubArr[i];
        }

        return dubMin;
}
```

```
	double findMinNotZero(double[] dubArr)
	{
		double dubMin = dubArr[0];
		for (int i = 0; i < dubArr.length; i++)
		{
			if (dubArr[i]<dubMin && dubArr[i] != 0) dubMin = dubArr[i];
		}

		return dubMin;
	}
}
```