Melchisedek Dulcio

Dr. Jim Knisely

CPS 499

May 6, 2021

Abstract

The semester Spring 2021, I endeavored to build a perpetually powered tracking device coupled with a location editing app that would prototype what a regular distance tracking device like a Fitbit could do if it used the concept of energy harvesting from piezoelectric energy that is generated by human movement. Although the initial motivation for this project was to aid digital mapping of places like Haiti where reliable electric power is scarcer and that the big mappers like Google cannot fully map, working through the project made me realize the potential of the project to scope mapping of even underwater places without the need to continually resurface and of course remove the frequent need to charge one's electronic wearables. In the end, the various components of the device worked separately, even though the device did not work as a whole. This paper then expands the details of this journey of learning.

Background

"Computing under transient power conditions is hard" (Ransford, et al., 1). Transiently powered computing devices (TPCs) are devices that use energy harvesting to power circuits and electronic parts to perform computing operations.

Energy Harvesting

Energy harvesting is technology that makes use of ambient sources to produce electric current that is then used power a computing device. Ambient energy sources include, "solar…, thermal … and piezoelectric …" (Safaei et al., 2019). Much research has gone into developing ways to efficiently harvest ambient energy because, eventually, if it were possible to remove the need to include an energy storage unit, like a battery, it would transform how smaller low-power consumption devices are manufactured. Situations where the installation is ideally a one-time matter, such as placing small computing devices inside a person for medical reasons, would make use of this ability to generate electric energy to power the device without the inconvenience of surgical procedure to recharge or renew the battery. As Keli Li et al. puts it, "Energy renewal or battery recharge for the devices are too inconvenient to satisfy, 'plug and 'play'."

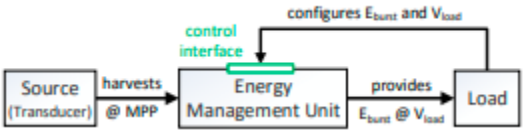

Figure 1: Diagram of Ideal System (Gomez, 2016)

When it comes to powering wearable technology, piezoelectric energy is one of the more promising energy replacement components for a regular battery. Keli et al. remarked that, "the basis of such a self-powered scheme [is]… the fact that the human body contains abundant kinetic energy sources during limb movements, such as joint ration." Kinetic energy released when moving can be captured by a piezoelectric transducer, the energy converter, to produce an electric current, which can in turn be captured by an

energy storage type unit that is then used to power the device. This hybrid approach which does not completely disregard the need for persistent power storage.

Transiently Powered Systems

Transiently powered computer systems are systems that use a form of energy harvesting technology while attempting to build a reliable device. There is also much research aimed at making these devices energy aware to mitigate the data loss from frequent power losses that are common with such systems. Andres Gomez et al. notes, "Cyber-physical systems have … been used in conjunction with energy harvesting and energy storing." This combination is the hybrid approach mentioned in the previous sections, where one does not directly use the power from generated by the transducer to perform computations, but rather uses the power to "charge" a storage unit that then powers the computation. This takes care of the data loss issue, since an operation executes atomically once the energy level required to perform the task is at the appropriate level as indicated by the Energy Management Unit, seen in Figure 1.

In summary, building such a system requires the components that will be listed below:

- Piezoelectric transducer – converts kinetic energy(vibrations) to electric energy.
- Optimal capacitor – stores the energy generated by the transducer
- Load – the computing device that utilizes the energy

Distance Tracking with 3-Axis Accelerometer

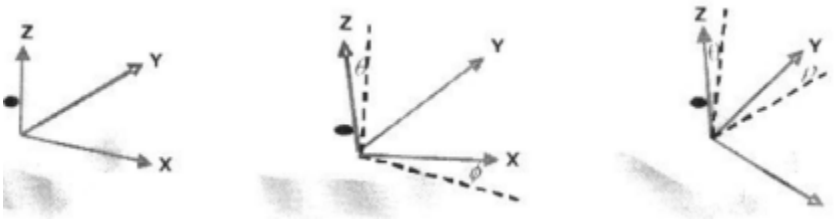

Figure 2: Accelerometer measuring tilt (Naghshineh, 2009)

In addition to the problems of the fields of energy harvesting and transiently powered computing, this research project also touched the problem of accurately tracking human motion using a 3-axial accelerometer without using a gyroscope. A triple-axis accelerometer is a sensor which measures acceleration and tilt relative to the ground, which it uses an altimeter to sense, using the tri-axial coordinate system to map data it senses as seen in Figure 2. As Sam Naghshineh et al. points out though, "accelerometers have limitations in detecting motion." To mitigate the limitations like the inability to truly measure static movement along a plane, a compass can be used to provide a more accurate reading.

Proposal

Initially, the plan was to build a transiently powered distance tracking device prototype that would emulate what a GPS-enabled Garmin or Fitbit fitness tracker could do on the level of distance tracking. Using a 3-axis accelerometer + compass (3AC), a real time clock (RTC), and SD card breakout, I attempted to build an Arduino circuit that would use the 3AC to detect the movement, the RTC to provide a timestamp for the data, and finally an SD card breakout piece to record the movement data. The energy harvesting unit would include 5 piezoelectric transducers, an optimal capacitor, and a battery pack that recharges from the capacitor connected to the transducers, and, in turn, the battery pack powers the device Arduino component which is the load. To couple the device, I planned to build an application that would process the data logged by the device and display a graph that was a "map" of the data. The app would then allow a person to edit locations and save it as exportable file to KML, the Google Maps data format.

Milestones

By the end of the Spring semester of 2021, I was to have completed 120 hours minimum with the Arduino circuit completely working by March 27th (end of last full week of March before April) and the application finished by April 26th (start of last week of April) to leave time for the project report due on May 1st. While I was not able to complete all the milestones, I did progress with the parts that were available to me at by the time is crucial to complete the project.

The first milestone set for March 27th was unable to be completed for a couple of reasons, the first of which was the timing of the ordering of parts and their delivery the last week of February. The device although put together, was not working as expected as by that time, even though from what I could tell, all the code and wiring was correct. After a week's research and stepping through the Arduino code, I discovered that the two pieces I was using were both supposed to use the same bus addresses, which interfered with the signals. Three days later, I discovered it the interference problem could be fixed with a special component, which I decided against speaking to the professor about and attempted to find another way to make it work. Unfortunately, that was unfruitful work.

Because the previous milestone was still incomplete and had spilled over into the timeframe for the next phase by 2 weeks, I went with a setback into the second phase, building the application. Without data to work with and the semester becoming increasingly busy, it was difficult to justify working on the application regularly, and, in the end, the basic layout window for the application was complete, but it was not functional.

Results

The Arduino Circuit

Each piece to the part when connected altogether does not work properly, even though they do once connected to Arduino separately. Since the accelerometer was the more important piece, I kept that along with everything else as I attempted to make it work together. Below is a picture of the Arduino unit put together.

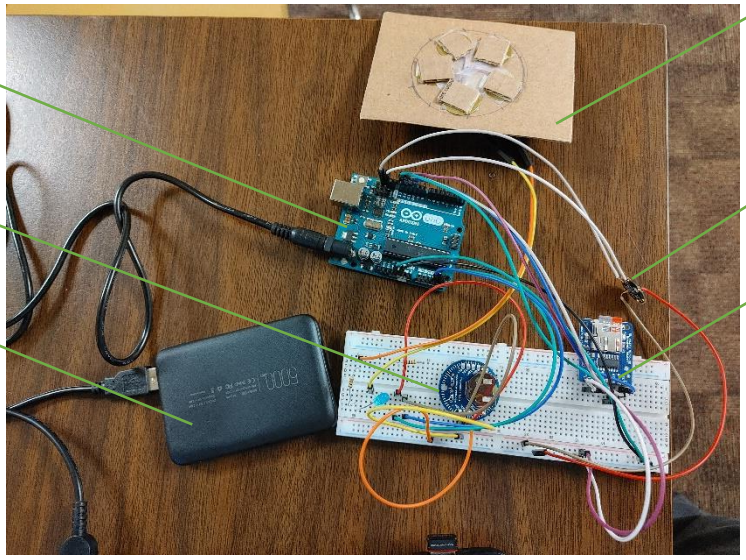

Figure 3: The Arduino Circuit Together

The code below, although now mostly commented since it was not working properly, is what I had initially uploaded to the microcontroller to run.

```
#include <Adafruit_LSM303_Accel.h>  //Accelerometer library.
#include <Adafruit_Sensor.h>
#include <Wire.h>  //Working with SDA and SCL
#include <SPI.h>   //also for SD card
#include <SD.h>    //SD card library
//#include <RTClib.h> //ChronoDot v2.1


/* Assign a unique ID to this sensor at the same time */
Adafruit_LSM303_Accel_Unified accel = Adafruit_LSM303_Accel_Unified(54321);

//Declare the chronodot sensor
//RTC_DS1307 RTC;

// set up variables using the SD utility library functions:

const int chipSelect = 10;

void setup(void) {
 //----------------------------Accelerometer Init-----------------------------------
 Serial.begin(9600);
 while (!Serial) {
  ; // wait for serial port to connect. Needed for native USB port only
 }
 /* Initialise the accelerometer sensor */
 while (!accel.begin());

 accel.setRange(LSM303_RANGE_4G);                //need to read up on this still
 lsm303_accel_range_t new_range = accel.getRange();
 accel.setMode(LSM303_MODE_NORMAL);

 //----------------------------Chronodot Setup---------------------------------
 /* Wire.begin();
 RTC.begin();

 while(!RTC.isrunning());

 //Not necessary, but it's nice to have accurate time although, I really just need to keep
track of timespan.
 DateTime now = RTC.now();
 DateTime compiled = DateTime(__DATE__, __TIME__);

 if(now.unixtime() < compiled.unixtime()){ //checking to make sure we don't need to
adjust the RTC time.
   RTC.adjust(DateTime(__DATE__, __TIME__));
  } */

 //--------------------------SD setup ---------------------------------------

 pinMode(chipSelect, OUTPUT);
 //Testing that all the wiring is correct and SD card is present;
 Serial.println("Done");
}
```

```
void loop(void) {
 if(!SD.begin(chipSelect)){
  Serial.println("Cannot find sd");
 }

 File datal = SD.open("track.txt", FILE_WRITE);
 //DateTime now = RTC.now();
 sensors_event_t event;
 accel.getEvent(&event);

 Serial.println("Writing");

 /*datal.print(now.year(), DEC);
 datal.print('/');
 datal.print(now.month(), DEC);
 datal.print('/');
 datal.print(now.day(), DEC);
 datal.print(' ');
 datal.print(now.hour(), DEC);
 datal.print(':');
 datal.print(now.minute(), DEC);
 datal.print(':');
 datal.print(now.second(), DEC);
 datal.print(' ');*/
 datal.print('(');
 datal.print(event.acceleration.x);
 datal.print(",");
 datal.print(event.acceleration.y);
 datal.print(",");
 datal.print(event.acceleration.z);
 datal.print(") m/s^2| ");
 datal.close();
 Serial.println("Done");
 delay(500);
}
```

The Application

The application window is designed, using PyQT as the Python GUI Framework, with Mat Plot Library to do the mapping with the data received from the accelerometer. However, there is no functionality in the application. The layout of the application is illustrated below in Figure 4, preceded by the code for it.

```
# -*- coding: utf-8 -*-
# Form implementation generated from reading ui file 'tracker.ui'
#
# Created by: PyQt5 UI code generator 5.15.2
#
# WARNING: Any manual changes made to this file will be lost when pyui
c5 is
# run again.  Do not edit this file unless you know what you are doing
.

from PyQt5 import QtCore, QtGui, QtWidgets
#from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg
#from matplotlib.figure import Figure

""" class MatCanvas(FigureCanvasQTAgg):
    def __init__(self, parent=None, width=5, height=5, dpi=100):
        fig = Figure(figsize=(width, height), dpi=dpi)
        self.axes = fig.add_subplot(111)
        super(MatCanvas, self).__init__(fig)
 """

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(865, 548)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")

        self.pushButton = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton.setGeometry(QtCore.QRect(20, 30, 111, 31))
        self.pushButton.setObjectName("pushButton")

        self.pushButton_2 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_2.setGeometry(QtCore.QRect(20, 80, 111, 31))
        self.pushButton_2.setObjectName("pushButton_2")

        self.pushButton_3 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_3.setGeometry(QtCore.QRect(20, 130, 111, 31))
        self.pushButton_3.setObjectName("pushButton_3")

        self.pushButton_4 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_4.setGeometry(QtCore.QRect(20, 180, 111, 31))
        self.pushButton_4.setObjectName("pushButton_4")

        self.openGLWidget = QtWidgets.QOpenGLWidget(self.centralwidget)
        self.openGLWidget.setGeometry(QtCore.QRect(170, 9, 671, 491))
        self.openGLWidget.setAutoFillBackground(True)
        self.openGLWidget.setObjectName("openGLWidget")

        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 865, 21))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "TrackMe"))
        self.pushButton.setText(_translate("MainWindow", "Open"))
        self.pushButton_2.setText(_translate("MainWindow", "Edit"))
        self.pushButton_3.setText(_translate("MainWindow", "Save"))
        self.pushButton_4.setText(_translate("MainWindow", "Save As"))

if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

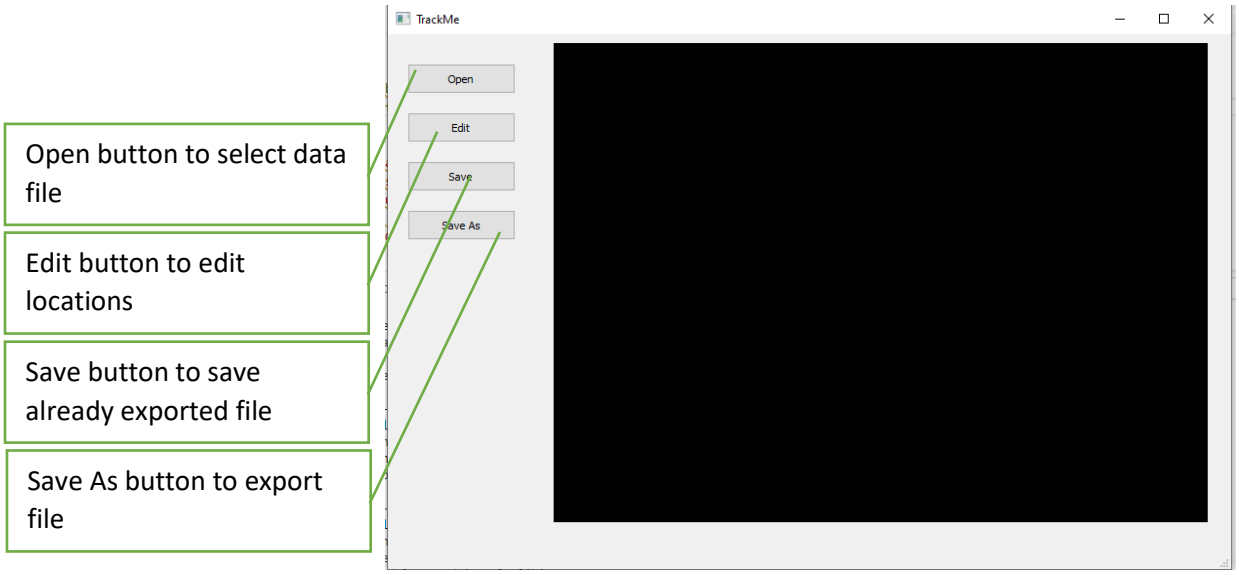

Figure 4: Application GUI Layout

Future Work

Though this project was semester project I did not complete during the semester, I plan to accomplish that during the summer. Among that things I thought I could build as a follow-up project is a transiently powered wireless keyboard where each key is a piezoelectric transducer unit that uses the energy from each key press to generate the electric energy transfer the correct signal to the computer. A note: this project, all the while I was working on it, felt much like a computer engineering project as much as it was computer science, so the next main project I plan to work on again this summer is a program that translates from English into another language using computational linguistics.

References

Ransford, Benjamin, "Transiently Powered Computers" (2013). *Open Access Dissertations*. 761. Pages 1 – 6

Benjamin Ransford, Jacob Sorber, and Kevin Fu, "Mementos: system support for long-running computation on RFID-scale devices" (2011). *SIGPLAN Not.* 46, 3 (March 2011), 159–170. DOI:https://doi.org/10.1145/1961296.1950386

A. Gomez, L. Sigrist, M. Magno, L. Benini and L. Thiele, "Dynamic energy burst scaling for transiently powered systems" (2016). *2016 Design, Automation & Test in Europe Conference & Exhibition*. pp. 349-354.

Naghshineh, S., Ameri, G., Zereshki, M., & Krishnan, D.S. (2009). Human Motion capture using Tri-Axial accelerometers.

Li, K., He, Q., Wang, J. *et al*. Wearable energy harvesters generating electricity from low-frequency human limb movement (2018). *Microsyst Nanoeng* **4,** 24 (2018). https://doi.org/10.1038/s41378-018-0024-3

Michael Buettner, Ben Greenstein, and David Wetherall. "Dewdrop: an energy-aware runtime for computational RFID" (2011). In *Proceedings of the 8th USENIX conference on Networked systems design and implementation* (*NSDI'11*). USENIX Association, USA, 197–210.

Safaei, M., Sodano, H., & Anton, S. (2019). A review of energy harvesting using piezoelectric materials: state-of-the-art a decade later (2008–2018). *Smart Materials and Structures, 28*, 113001.