Mobile Diagnostics for Personal Electrics Transportation Devices

Zachary A. Gilchrist
Electrical Engineering Design Laboratory
University of Kansas
Lawrence, KS 66044
zacharygil@ku.edu

Faculty Advisor:
Prof. Glenn Prescott

ABSTRACT

The ultimate purpose of this project was to improve on a new electric form of transportation. These methods of transportation have been gaining popularity for those who have relatively short commutes, and this project poises that group as the target audience. As a result of these trends and the practicality of small personal transportation, the design team made the purpose to attain usable real-world practicality and walking replacement ability. Other types of devices have been created before, however they usually have a short ride time and do not go much faster than the average walking speed. That being said, this design also uses telemetry to serve as a proof of concept that internal data can be sent to a receiver from a distant location. The end result was expected to be an enjoyable experience that gave a mobile diagnostics system.

INTRODUCTION

This paper is to discuss the research and development of the Telemetry involved in a mobile diagnostics system for a single wheeled electric transportation device. The device was referred to as Riding One-wheeled balanced electronic range transportation (ROBERT).

THE BODY

The telemetry system is made up of a few major components that come together to form one complete system. The complete system's function is to take in data, in this case temperature, and transmit that data to a computer wirelessly. The individual component/components are an Arduino Uno, Adafruit Temperature sensor, and two XBee transceivers. The components come together as seen in Figure 1. The temperature readings are taken into the Arduino that uses the XBee to transmit that data to an XBee receiver connected to a computer through USB. Whenever the Arduino based XBee transmits data through the extended antenna it flashes a red LED beacon. The XBees communicate through a 2.5-GHz frequency and have individual PAN IDs to allow for secure and uninterrupted communication.

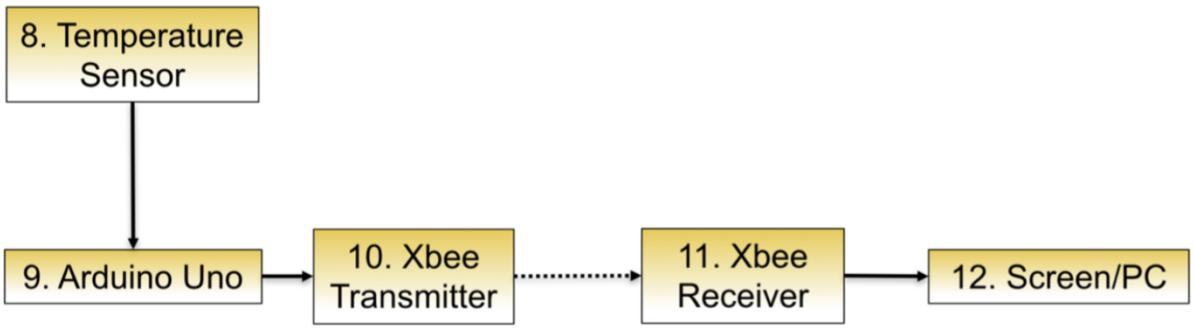

Figure 1 Telemetry Circuit functionality block diagram

Software aspects:

The software has two main areas, that are the Arduino IDE and then the XCTU program for the XBee configuration. First, the Arduino IDE is interesting in that it is written in Java and based on the Processing Project. Though the code used in programming the Arduino is a C and C++ mixture. Both languages will work, with the note that most of the standard libraries will NOT work. This is in part due to the fact that Arduinos have extremely low RAM compared to something like a PC. The code written for the Telemetry portion is C++ with imported hardware libraries for the sensor and XBee.

The Arduino code has a loop function that it continuously goes through. This is useful for microcontrollers because there isn't a required "wake-up" and "shut down" function, or a need to write code to create a live loop. Within this loop our code had two methods that were waiting on "YES" to come to their conditions. This can be seen in Figure 2. The first loop waited on data to be seen as being transmitted to it. More specifically, it waited for the XBee to say it had data being transmitted to it. In this code that data was irrelevant. For this code the data just showed that the user on the computer tapped some key and that data was sent to the Arduino based XBee. Once that condition was met, the data was read from the temperature sensor using methods within the downloaded library. That data was then algebraically converted to Fahrenheit and from hexadecimal form. Then transmitted back to the XBee that originally transmitted data to it. In essence, it showed that someone was asking for data, read the sensors, then sent data back to that someone.

The other of the two impeded loops was an LED switch. It waited for data to be transmitted. As soon as the data was transmitted from within the last loop, it set that digital PIN to high and turned on an LED. As soon as the data stopped transmitting, the LED is set to low. The LED is on for a very brief period of time relative to the data being transmitted. If the data was continuously transmitted (as tested) the LED simply stays on. This can be seen visually in Figure 4.

The other half of the software in the project is the program XCTU, which is a configuration and test utility software. In this case, we used a USB dongle to connect to the PC to be able write and read the XBee. Both XBees were first read and configured through the XCTU program, even though one was only used in the Arduino setup. XCTU allows for multiple parameters to be changed, but the most important and debatably easiest to change is the PAN ID. Ours is set to 1776 to separate it from any other XBees in the area. Then post calibration use, the XCTU program was used to monitor the XBee module connected to the Arduino. It sent a keystroke to the Arduino to signify that it wanted to receive data. That data was then received and displayed in the "Console Mode." It shows the data in two forms -- Hexadecimal and as received. As received in this case was the temperature in Celsius and Fahrenheit.

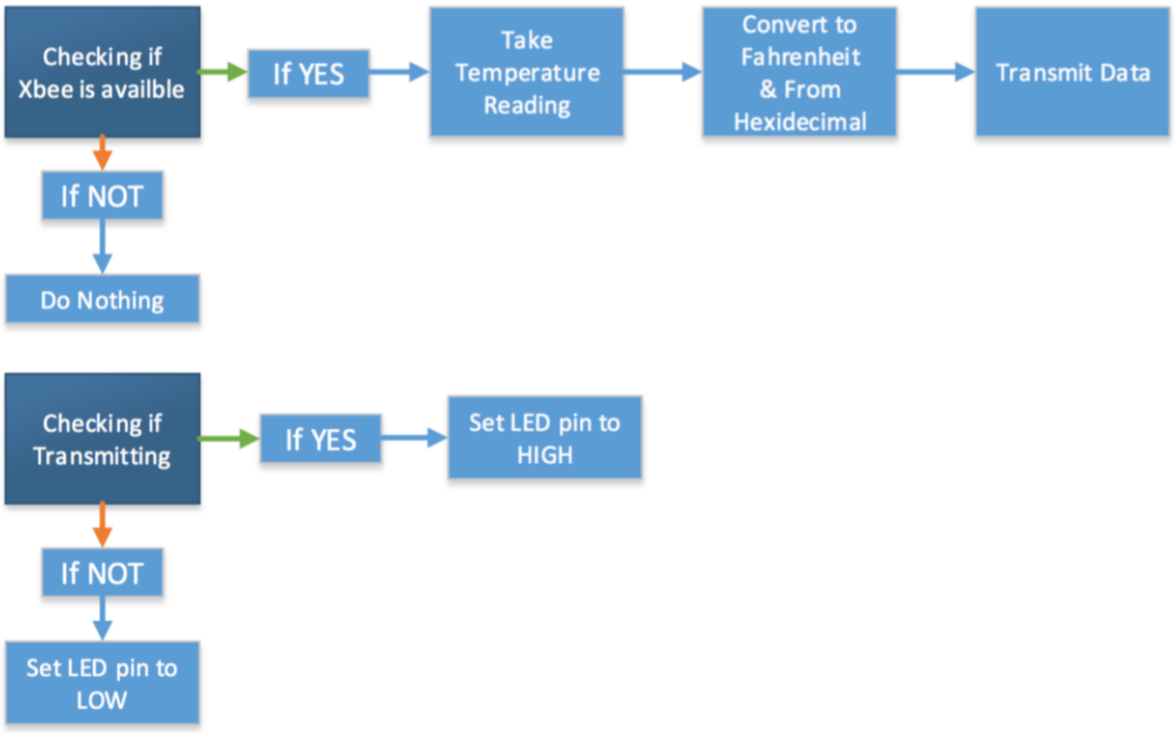

Figure 2: Arduino Software Flow Chart

How it was implemented:

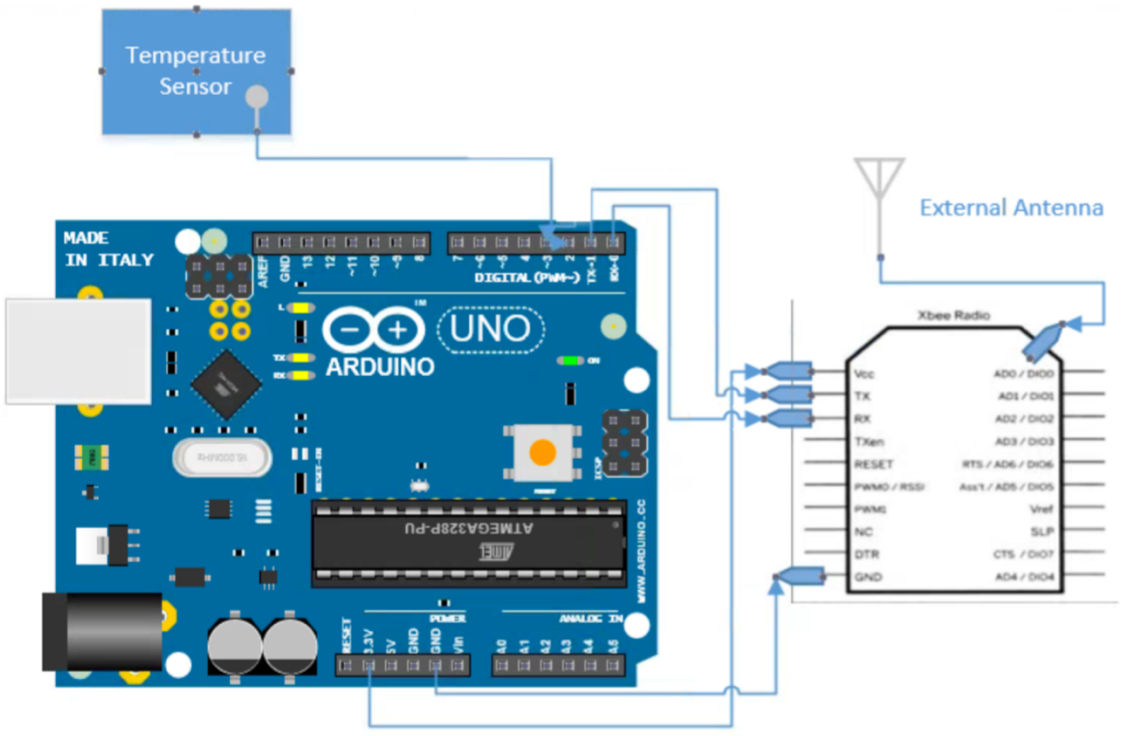

Figure 3: Component interconnections with Arduino for telemetry

The hardware connections are largely set up in a very direct and obvious way. The Xbee is connected to a custom shield for Arduino that lies directly on top of the Arduino Uno. The Xbee for the computer is connected directly to a USB dongle that was made for Xbees.

There are two Xbees; one with a SMA connector and one with a permanently attached wire antenna. While they are both interchangeable for placement with the USB dongle or Arduino Uno, the one with the SMA connector is to be used with the Arduino set up. This is because the 7.9-mm connector is to be used with an extended Wi-Fi antenna located outside of the board.

The temperature sensor is connected to two analog pins for data, and then 5-V power and ground. There are 4 unused pins on the actual temperature sensor for alerts and other unused options. The LED's anode is connected to a digital pin while the cathode is attached to a 220-W resistor that is attached to ground. This is all powered by a 9-V battery housed in a case. The case has a male 2.1-mm center positive barrel jack, that goes into the standard female 2.1- mm center positive barrel outlet of the Arduino Uno.

Lastly, all of these components are held together with industrial hook-and-loop fastener tape, Velcro, to insure a secure mount. It also allows for the Telemetry portion to be completely removed for sensor adjustment or added as an option.

Testing

The telemetry portion was evaluated through results and a distance test. The inflow of data can be seen in Figure 6. This was the data that was sent from the board to the XCTU program on the computer. The screen shot is of the console within the XCTU program after a "." keystroke was sent. It shows the output of the "." in blue, and the input of the temperature in red. Also, to be noted, the Arduino set the LED to be illuminated every time the data read that "." and transmitted back data. This can be seen in Figure 4. The returning data of the temperature shows not only that transmission was achieved, but also that the temperature readings were accurate. The temperature sensor was accurate when at room temperature within 1 degree. It was also able to rise accurately and quickly, but took over a minute to cool down if there was a quick influx of temperature. This is expected, as it is an *ambient* temperature sensor. We tested this aspect specifically with a heat gun.

After transmission was proved successful and the sensors were calibrated, the next aspect for evaluation was the maximum range of transmission. To do this, the telemetry Arduino module was attached to its 9-V power supply and taken away until the connection broke. Line of sight gave roughly 200-meters till no connection was available. The term roughly is used as the connection was in and out beginning at 178-meters, and completely out at 200-meters. Within a building, we were able to go from the fourth floor of Eaton Hall down to the second floor of Learned Hall's main hallway and still have connection for half of the length of the hallway. This direct line distance is 73.152-meters. It goes through 4 walls at minimum, and 2 stories.

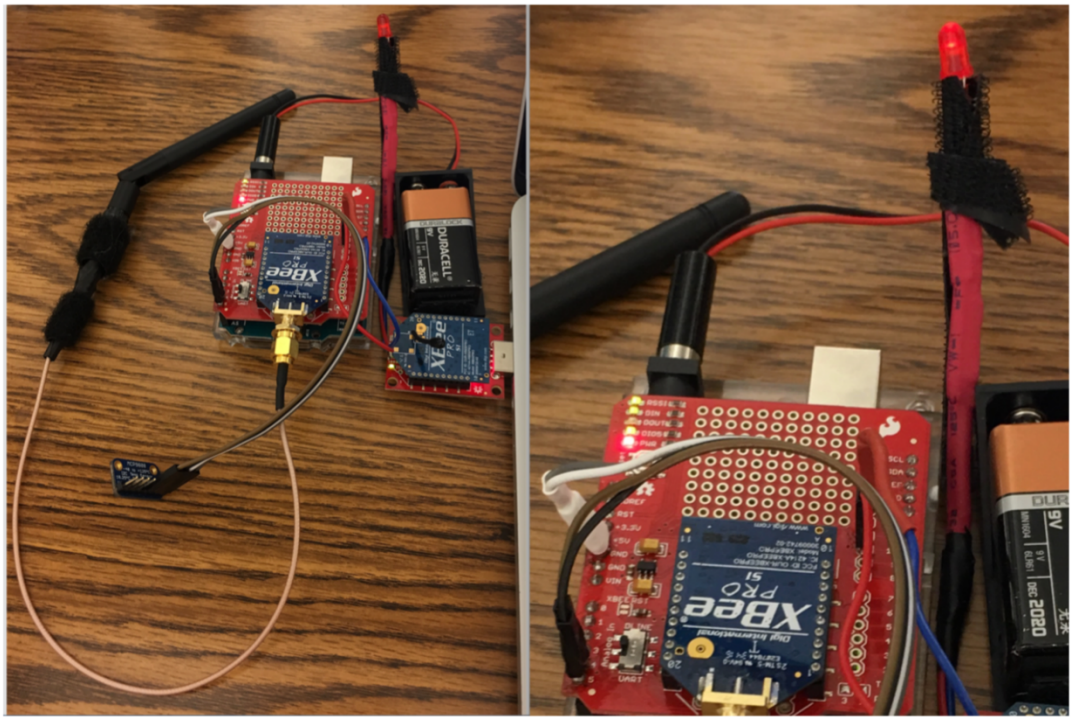

Figure 4: The complete circuit set up with illumination on right

Figure 5: The USB Dongle

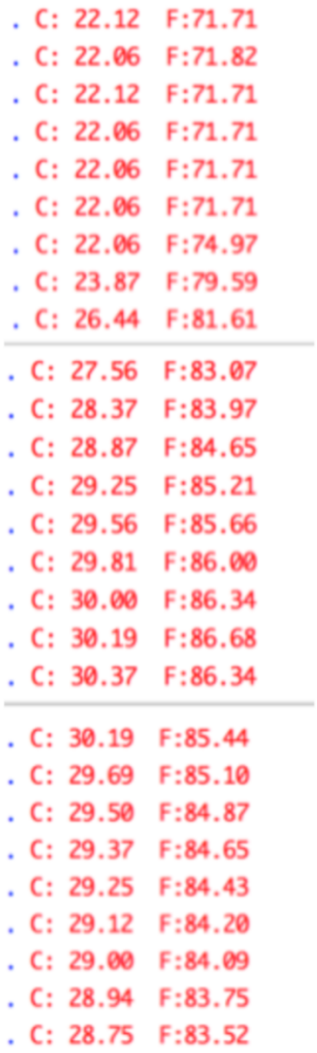

Figure 6: The Transmission Data as seen from XCTU console

Evaluation of Test Data

The complete system worked as the data shows. An expected range for transmitting with proof of concept values was achieved. The temperature sensor was ambient so it rose relatively quickly, but was left to its silicon and metal makeup that restricted it from cooling down just as quickly. The Arduino was never pushed as far as capabilities go, so it performed as necessary. The XBees worked well for transmitting, and allowed for antenna extension. The antenna in our case, compared to other noise factors, added a dismissible gain advantage. Though it did provide an advantage of getting the antenna outside and away from the wheel and elements. The antenna could be shot or soaked, and it would not affect the rest of the transceiver components. More so, in a case where the housing interrupts a 2.4-2.5 GHz signal, the antenna proves mandatory. The cut off distances were expected and relatively close to the ones listed on the data sheet.

CONCLUSIONS

The complete system worked well and met all expectations. With 9 digital pins unused there is room for the addition of multiple other digital sensors. The extended antenna and portable battery allowed for safe and independent placement of the Arduino based XBee within the housing. If any modifications were to be made for the next model it would be within the capabilities of the microcontroller and XBee. A raspberry Pi matched with a XBee series 3 PRO would allow for much greater range and processing capabilities. Though the proof-of-concept mobile diagnostics system was a success.

REFERENCES

[1] Iswandi, H. T. Nastiti, I. E. Praditya and I. W. Mustika, "Evaluation of XBee-Pro transmission range for Wireless Sensor Network's node under forested environments based on Received Signal Strength Indicator (RSSI)," *2016 2nd International Conference on Science and Technology-Computer (ICST)*, Yogyakarta, 2016, pp. 56-60.
doi: 10.1109/ICSTC.2016.7877347

[2] H. Kumbhar, "Wireless sensor network using Xbee on Arduino Platform: An experimental study," *2016 International Conference on Computing Communication Control and automation (ICCUBEA)*, Pune, 2016, pp. 1-5.
doi: 10.1109/ICCUBEA.2016.7860081

[3] J. Á. Ariza, "A proposal for teaching programming languages through open hardware tools," *2016 IEEE 8th International Conference on Engineering Education (ICEED)*, Kuala Lumpur, 2016, pp. 202-207.
doi: 10.1109/ICEED.2016.7856072