Technological Feasibility
October 24, 2016
Kine Jax

Sponsor/Mentor:

Dr. Kyle Winfree, NAU SICCS

Team Members:

Anthony Black
Christopher Whitney
Cherie Parsons
Grant Swenson
Jack Jenkins

Table of Contents

I. Introduction

As time progresses, technology will only become smaller, faster, cheaper, and more integrated into our everyday lives. We already wear devices that sense us and the world around us, giving us great insight into how we behave and operate. Using this insight, we can leverage technology to make better decisions and diagnoses, helping everyone live more fulfilling lives. Currently there is no wearable device that is adaptable in its data logging, meaning that there is currently no device that can be expanded to log a wide array of different extenteral sensors and monitors. The goal of the KineTrax is to create a mesh of wearable devices that can do just that. The KineJax team will continue the work of developing this device and the tools needed to configure and analyze the results. In this document we outline, describe and analyze the key technical challenges we predict for this project. However, it should be noted that the challenges laid out below are not mutually exclusive and that others may arise which could have not been predicted.

II. Technological Challenges

In this section we outline key technical challenges that we expect to encounter. However, the list is not mutually exclusive and we do expect/plan to encounter more challenges as the project develops.

1. Data Offload Software
    a. The software shall be written in a language and scheme that will allow for easy maintenance and modifications by future researchers.
    b. The software shall provide a means to retrieve the data from the devices and export it into a readable format for data analysis tools.
2. Self-Assembling Networks
    a. The software/hardware shall be able to link all seven devices together in one decentralized meshed network.
    b. The software/hardware shall be able to uniquely identify each device, calculate the distances between any given two devices, and provide a framework for future development.
3. Time Synchronization
    a. The software/hardware should be able to keep all seven devices time synchronized so that each device can agree on the time down to the millisecond.
    b. The hardware's efficiency will also have to considered including latency, drift, and clock speed.

4. Development Environment
    a. The software should be developed in an environment which is portable, easy to set-up, and easily handles dependencies.

III. Technological Analysis

In this section we present our analysis and research on the different technical challenges identified above. This includes alternative approaches, justification for the chosen approach, and a way to prove feasibility.

1. Data Offload Software

The challenge with developing the user interface and desktop program lies in the need to export data in a format usable by data analysis software, such as a CSV file, the ability to communication with and configure the low-level hardware across platforms (e.g desktop computers), and to have a code base easily expandable by a wide array of collaborators. The assumption here is that those collaborators potentially have little coding experience. For this issue, we considered several options: Python, C, and Processing.

a. Alternative Approaches

The alternatives considered were Python and C. Python offers an easy-to-learn language that is an interpreted language. C is a language that is a well-founded compiled language, which would provide a lower overhead, but is not as easy to learn. Processing language is a compiled language, with an easy-to-learn syntax like Python, and is the language of choice by the sponsor.

i. Python is a popular and widely-used programming language that is known for it's highly abstracted style and easy learning curve. The drawback to the language is the overhead. An abstracted and interpreted language such as Python may impact performance with serial communication.

ii. C is another widely used language and has the advantages of lower overhead when compared to Python and Processing, as well as being the language used on the KineTrax devices. This reduces the number of different languages used in the project, but it is more difficult to learn.

iii. Processing is the language desired by the sponsor, and is a language that allows for fast development of GUIs. Since it is not an interpreted language like Python, it will result in lower overhead in comparison. Additionally, while the initial Processing syntax is based on Java, it also includes support for other syntax styles, such as Python, giving the sponsor choices in how they wish to maintain the code.

b. Chosen Approach

i. Our chosen approach will be Processing. While all options possess the ability to read and export the data, Processing has the advantages of having lower overhead and is the language that will already be known by those tasked with maintaining the codebase after delivery.

c. Proving Feasibility

i. In order to prove feasibility, we shall develop a simple program in Processing that will display a simple user interface that will model the final product. The demo shall involve connecting to the KineTrax device and establishing some simple communications.

2. Self-Assembling Networks

The challenge of this self-assembling network is to have a system that can reliably establish connections that can leverages hardware constraints and compatible protocol stacks. We first have to have wireless network protocols that works with the CC2500 transceiver. Since the only compatible network protocol is SimpliciTi, we are forced to use it. However, this constraint limits our scope of research and analysis; research that might lead to insight. For this reason the alternatives below assume we do not have this constraint. We considered three approaches: Bluetooth, Zigbee, and SimpliciTi.

i. Alternative Approaches

The alternatives considered were Bluetooth,ZigBee, and SimpliciTi. Bluetooth is a widely used wireless standard that is commonly used for simple connection between two devices. For example, the Pebble Watch

uses a Bluetooth connection to an IOS/Android phone. ZigBee is a more complex wireless solution that offers additional resources such as synchronization and monitoring, i.e home automation. SimpliciTi is Texas Instruments' proprietary radio frequency protocol.

1. Bluetooth is a pure peer-to-peer protocol that use short-wave radio to communication between devices up to a range of 100m. Bluetooth is a widely accepted technology. However, the lack of internet mechanism to connect to multiple devices does not make it a viable alternative.

2. ZigBee is networking standard that allows for low-cost, low-power, mesh networks. The standard is targeted at devices with long battery consumption requirements and widely used in monitoring applications. It is a perfect solution for achieving multiple network connections asynchronously across multiple devices.
3. SimpliciTi is a low power proprietary protocol available for some of TI's chipsets including the CC2500. It supports star with extenders and peer to peer networks. It also supports sleeping devices which helps cut down on power usage. This is the only wireless protocol supported by KineTrax.

ii. Chosen Approach

1. Since we are limited by the hardware requirements, the only viable approach is the SimpliciTi protocol. However, we will be able to implement Zigbee like behavior using this protocol.

iii. Proving Feasibility

1. To prove feasibility, we shall implement a simple program that demonstrates mesh network behavior between multiple devices using the MSP430-EZ430-RF2500 development board and its internal temperature sensors.

3. Time synchronization

The challenge of time synchronization between devices will ultimately depend on the constraints of the hardware. These constraints are the configuration (i.e clock speed), transceivers latency, distance between devices, time drift (which is directly related to clock speed), and the choice of algorithms. These are just a few of the aspects that contribute to this challenge and the optimal solution will have to be holistic in its approach. This challenge is made even more complex by the need to maintain reliability, scalability and adaptability among devices.

a. Alternative Approaches

1. Cristian's algorithm: The server keeps track of the time while other devices synchronize to it. A node makes a request. The server preps a response and appends its own time. The node receives the response from server and sets the time to time from the server (T) + Round Trip Time(RTT)/2. The accuracy can be improved with multiple requests. It does not double check time accuracy because there is only one server. This algorithm assumes the RTT is short.

2. Berkeley algorithm: The main server fetches times from clients, average the results, and reports back to the clients with how much each node should change. Clocks with times outside a certain threshold are disregarded. Systems however, usually don't subtract time because it can break monotonic time (fundamental for certain algorithms).

3. Network Time Protocol (NTP): The network is split up into layers. The top layer of devices keeps track of the time. The next layer references the layer above for time synchronization and can check other devices in the same layer to verify. The next layer references the layer above them and so on and so forth.

4. Clock Sampling Mutual Network Synchronization: Nodes in the network recursively correct time based on the offset. This algorithm is scalable over mesh networks.

5. Precision Time Protocol: There are one or more master clocks that communicate their time to boundary clocks that then communicate their time to their nodes.

6. Reference broadcast synchronization: A transmitter sends out time to multiple nodes. The receiver nodes receive the broadcast and communicates with other receiver nodes. The difference between neighbors is calculated with average difference.

7. Reference Broadcast Infrastructure Synchronization: The clients/nodes receive data packets and look at time of arrival in the packet by observing the physical layer. The time is exchanged with its neighbors and compared. The average of the difference of times between nodes is calculated. This algorithm requires no modification of the access node.

ii. Chosen Approach

1. Because it matters most that all nodes are synced to a single clock the best approach to use is Cristian's algorithm. It is the most intuitive to implement and logical approach because the devices will be very close to each other cutting down on the round trip time. Furthermore, it is easy to increase accuracy by increasing the number of requests.

iii. Proving Feasibility

1. To prove this is a feasible approach we will use the MSP430 to send a timestamp between two devices. The node that sent the initial request will get a response from the other node that includes the time it was received. The node will use Cristian's algorithm to correct the time. To verify this algorithm is working we will use an oscilloscope to measure the pulse on the two nodes and verify the two pulses line up.

4. Development Environments

The challenge of choosing a development environment is difficult because there are so many options and compatibility issues. Since, the only options for programing the MSP is C and C++ (which is not recommended by most embedded system programmers) we are forced to use those languages.The key aspects of this challenge are the ability to handle dependency, load the program to the device, and debug the program once on the device.

a. Alternative Approaches

1.Code Composer Studio: This a development IDE (Interactive Development Environment) that is recommended by Texas Instruments and is built of Eclipse source code. A key

feature of this environment is that it allows one to see the device's registers live and can handle all the build dependencies. However, on Unix like environments it does not support the development board.

2.Energia: This is an open source IDE that was built to look and feel like the Arduino IDE and was made specifically for programing MSP devices. This is a great option for people who have experience programing Arduinos. However, its debug tools are more limited than Code Composer and its handling of dependencies it not the most intuitive.

3.Makefiles with terminal tools: This approach uses cmake files to handle the project build and then load the binaries onto the device using the command line tool *mspdebug*. One advantage of this approach is that it would allow the project to be platform independent and would require full understanding of all steps needed to build. However, a major drawback of this approach is that it would require the explicit declaration of all the dependencies and would not have as intuitive debugging.

b. Chosen Approach

The approach we are choosing is Code Composer using the Windows OS, because it is the fastest to get programing and allows for the most intuitive debugging. However, in the future it would be a great idea to develop a process to allow for development using makefile because it would allow for OS independence and would require researchers to fully understand the build process.

C. Proving Feasibility

If we are able to prove feasibility of the previously stated challenges it only following logically that we are able to build the system.

c. Technology Integration

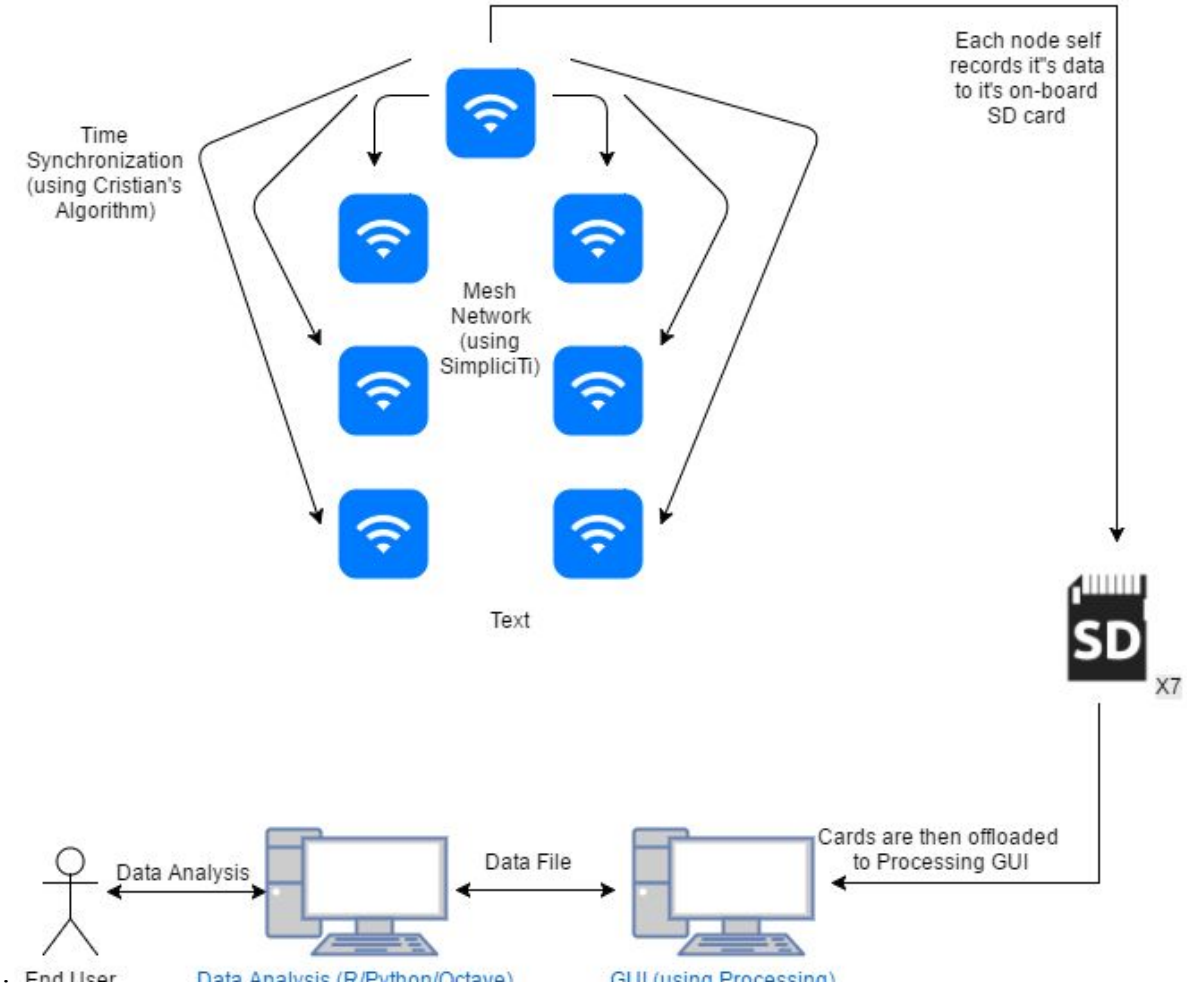
i. Problem and solution overview

1. Our software must effectively report data and offload it to some software accessible to the users. We have decided to do this with the Processing language, as it is offers the best balance of the criteria stated in Table 1 (pg. 5).
2. The devices must create a self-assembling network to effectively record this data. We have chosen to use SimpliciTi (using Zigbee ideas), as it is supported by our Ti processor already.
3. The devices must also all have their time synchronized down to the millisecond, which we plan to accomplish using Cristian's Algorithm.
4. The devices must not only remain synchronized, but must also maintain this synchronization over an extended period of time.

ii. All of these solutions should work together from the documentation our team has analyzed. The Processing language is a simple language to learn, and has straight forward documentation and tutorials on how to represent data to the end users. The data represented to the end users will be recorded through a mesh network, created using the SimpliciTi protocol. There is no question of if this protocol will work with our device since it was developed by Ti, the same manufacturer that our CPUs were created by. Finally, the clocks on each individual device will be synchronized over this mesh network through the implementation of Cristian's Algorithm. Since this is an algorithmic solution, it can be implemented using any programming language and should not conflict with any of our other technological choices.

iii. The following image shows the system diagram, depicting how every component works together.

Kine Trax System Diagram



IV. Conclusion

To summarize, the problem we hope to solve with this project is the lack of available technology to log data across multiple network devices seamlessly and precisely. The technological problems we predict to encounter with this device are communicating with the device using a GUI, timing syncing across devices, and creating a self-assembling network. We will show the feasibility of this project by creating a demo that will show the development device (e.g msp430 ez430-rf2500) self-assembling, exchanging information, time syncing and communicating through a terminal program. We are confident that this everything laid out in this document is feasible, however, there are still many unanswered questions that need to be addressed. For example, will the software developed on the development device also work on the actual Kine Trax device without modifications? Will the software be able to able overcome hardware limitations to efficiently accomplish the requirements? These are questions that will be answered once development begins.