Software Design Document
February 15, 2017
Kine Jax

Sponsor/Mentor:

Dr. Kyle Winfree, NAU SICCS

Team Members:

Anthony Black
Christopher Whitney
Cherie Parsons
Grant Swenson
Jack Jenkins

Table of Contents

Introduction

The KineTrax is a custom-built, wearable device that aims to create a system to record full body kinematics. The data gathered will allow for gait analysis of patients with neurological impairments, physical impairments, prosthetic limbs, and healthy subjects of interest to sports medicine. Although there already exists wearable devices on the market that can perform clinic based gait analysis, they do not allow for the addition of external peripherals and sensors, nor the ability to interface with other devices. Furthermore, this system differentiates itself from other systems by the explicit design that data is collected in the ecologically valid community setting, not in a lab. This means that a user wearing the device daily is able to provide better analysis to doctors because it gives data on a user's gait from day to day and not just in a single lab session. To collect accurate data, seven KineTrax devices will be located on: each ankle, each knee, right and left side of the waist, and a single device on the chest. The KineTrax platform is designed to record and synchronize across a wirelessly distributed network of sensors. The devices will communicate between these seven devices to accurately log information.

The project will enable doctors to make more informed decisions when treating patients with neurological diseases and possibly could be used to predict illness. Additionally, it will merge the domain of wearable technologies with motion capture to give accurate gait information over time. We will also be laying the framework for a device that is expandable in it's scope, with the ability to customize and add sensors based on the user's needs. In this document we overview the implementation, describe the architecture, modules, and methods, and finally outline our implementation plan.

Implementation Overview

The KineTrax solution involves an array of custom-designed embedded circuit boards that will be able to gather day-to-day gait information over an extended period of time. The KineTrax platform will have capabilities of synchronizing across a wireless network of sensors that will accurately record and communicate between each of the seven nodes. SD card's will store the recorded data onboard, allowing for later offloading via our desktop GUI.

Our implementation approach involves creating modules that fulfill the following key requirements: Real-time clock, device configuration, peripherals, wireless, SD card, and serial communication. All of these modules will be achieved by using the following tools: Code Composer Studio (CCS) IDE, the SimpliciTi protocol stack, and the Processing Language.

1. Code Composer Studio is a development environment for the MSP430. The KineTrax will be programmed using the Standard TI libraries that is already packaged within the default download of Code Composer Studio.

2. SimpliciTi is the wireless protocol stack that is supported by our MSP430. This protocol will be used to implement a star network between seven separate KineTrax devices, allowing for distance calculations and time synchronization between them.

3. Processing will be used to create a GUI in order to offload the data collected by the KineTrax device. The Processing GUI will also be used to upload a configuration file to the KineTrax, allowing each the configuration of different external peripheral.

The key modules shall be implemented by separate team members in parallel. Clear communication methods such as Slack, email, and weekly meetings will be vital. In order to keep all of our code synced across development environments, our codebase will be stored and updated through Dr. Winfree's subversion (SVN) server. Additionally, documentation and notes will be maintained on the KineTrax wiki page.

Architectural Overview

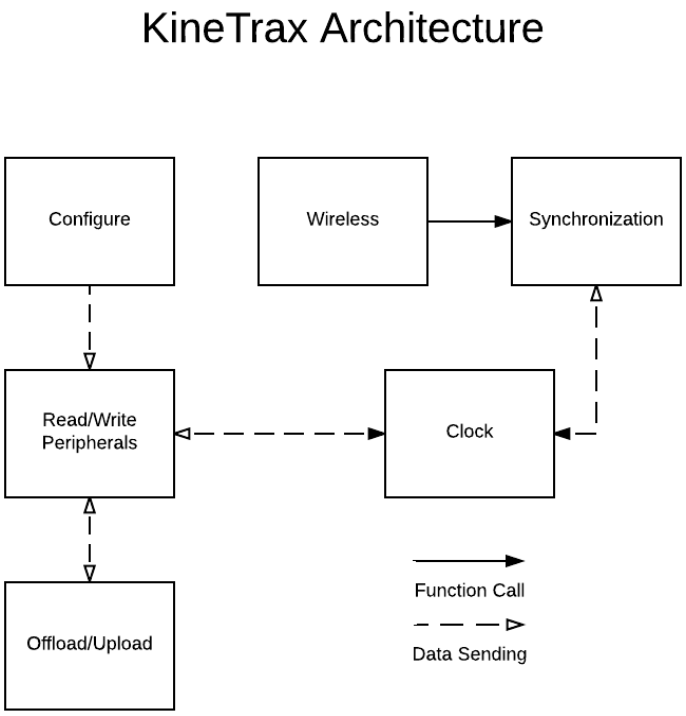

Figure 1: *The architecture of the KineTrax project. Each box represents a module with the connecting arrows showing the relations between the modules.*

Real-Time Clock

a) Key responsibilities
   i) Initialize the time of the real-time clock.
   ii) Get the current time from the real-time clock.
   iii) Set the time on the real-time clock.
b) Main communication mechanisms
   i) I^2C (Inter-Integrated Circuit)

Peripherals (accelerometers, SD card, etc.)

a) Key responsibilities
   i) Get accelerometer data from the onboard gyros.
   ii) Read the recorded data from SD card to the GUI.
   iii) Transfer accelerometer from the inertial measurement unit (IMU) to SD card.
   iv) Write clock information to SD card.
b) Main communication mechanisms

i) I^2C (Inter-Integrated Circuit)
ii) RS-232

Wireless Communication

a) Key responsibilities
   i) Connect to and communicate with other KineTrax devices.
   ii) Use radio signal strength to calculate distance moved and position.
b) Main communication mechanisms
   i) Functional calls

Serial Communication

a) Key responsibilities
   i) Set configuration data to the SD card.
   ii) Offload data from SD card to processing software.
b) Main communication mechanisms
   i) RS-232

Configuration

c) Key responsibilities
   i) Sets up peripherals
   ii) Sets up peripherals sample rate
d) Main communication mechanisms
   i) Function calls
   ii) Global variables

Module and Interface Description

In the following section we outline and describe the modules that we shall implement for this project. We use the term modules lightly because we are not working with an object oriented language, however, this style of code organization is extremely helpful for dividing the functionality. When using the term module we are referring to the individual C source code and header files.

Configure

Configure is the module that is responsible for configuring the hardware based on information set by the user as well as default values. Individual peripherals and sensors will need to have information such as sample rate and address locations set. Clock speed will also be set by a subroutine of this module. The subroutines in this module will

usually only be called once at initial boot-up and will pull information from the SD card. The subroutines in this module will be set and specified using the Processing GUI.

The first and most important of the subroutines is *setClockRate(char rate)* which takes a char of an enumerated type which will specify the clock speed of the MSP. It will return a boolean value which if true, flags that the operation was successful. The second routine in this module will be *addPeripheral(char name, int addr)*. This subroutine will add a sensor or peripheral to a list of peripherals that shall be sampled. It will take in the name and location of the peripheral. It will also return a boolean flag. Another routine in this module will be *setPeripheralRate(char rate, int addr)* this routine will take in enumerated char type and the address of the peripheral. It will also return a boolean flag value. There will also be a *removePeripheral(int addr)* function. This will be exactly the same as the add subroutine, however, it will remove the peripheral from the list to be sampled.

Read and write peripheral

The read and write peripherals module is responsible for the direct interactions with these peripherals. This includes reading sample information from a peripheral or sensor, as well as writing information to control registers. There are two main subroutines in this module, *readPeripheral(int addr)* which takes in an integer of the peripheral address and returns the information found, if any, at its registers. The other subroutine in this module will be *writePeripheral(int addr, int port, char data, char interface)* which is similar to the read function, but takes in a data argument and writes the register instead of reading.

Clock

Clock is the module responsible for anything related to time. This includes getting the current time as well as setting the time. This module is extremely important for the synchronization module because both functions are required in those operations. The first subroutines in this module are *setTime(byte hour, byte min, byte second)* this subroutine sets the time of the real time clock and takes in the hour, minute and second. The second routine in this module is *getTime()* which returns the current time.

Wireless

The wireless module is responsible for all the wireless communication between devices. It is the module which handles the sending and receiving of messages, which is important for the time-synchronization aspect of this project. The first and arguably the most important function in this module is *connect()* which creates a connection between two devices. This connection allows for the devices to communicate. Two other important subroutines in this module are *sendMessage (char data)* and

receiveMessage(). These subroutines do exactly as their names suggest and are used to send and receive messages respectively. The send message routine takes in data which it sends to the connected device and receive message waits to receive a message.

Synchronization

Synchronization is the module that is responsible for making sure that whenever a peripheral is sampled that its timestamp is the same as the other connected devices. This module depends heavily on the clock and wireless modules because it requires those modules subroutines to accomplish this task. It has one key function *sync()* which will be called when necessary to sync the clocks of the connected devices. This function will not take in any arguments and will return a signed int, telling the device the offset from the main node.

Offload/Upload

The offloading and upload module is responsible for all the tasks related to communicating with a personal computer. The first subroutine in this module is *checkForConnection()* which checks to see whether a computer is connected to the device through USB and will return a boolean value. The second routine is *sendInfo(char data)* which will send passed information to the computer through the serial connection. The third function is *getInfo()* which will pull the serial connection and return, if any, data currently on the serial buffer.

Implementation Plan

The implementation plan outline is shown in the two figures below, with tasks divided amongst team members, as well as estimated start and finish times. Our first goal is to implement the real-time clock, which is necessary for many of the subsequent tasks. Next, we will focus on recording and offloading the data, where we shall ensure the peripherals can be read properly, and their data transferred to a CSV file via the offload software. After completion of this, we shall focus on the time synchronization algorithm to ensure all KineTrax devices have the same time stamps when sampling. Finally we shall focus on the ability for the devices to self-configure in a network of devices, as well as the ability to calculate distances between each device in the network. Testing of each goal shall occur following completion of said goal, prior to integration.

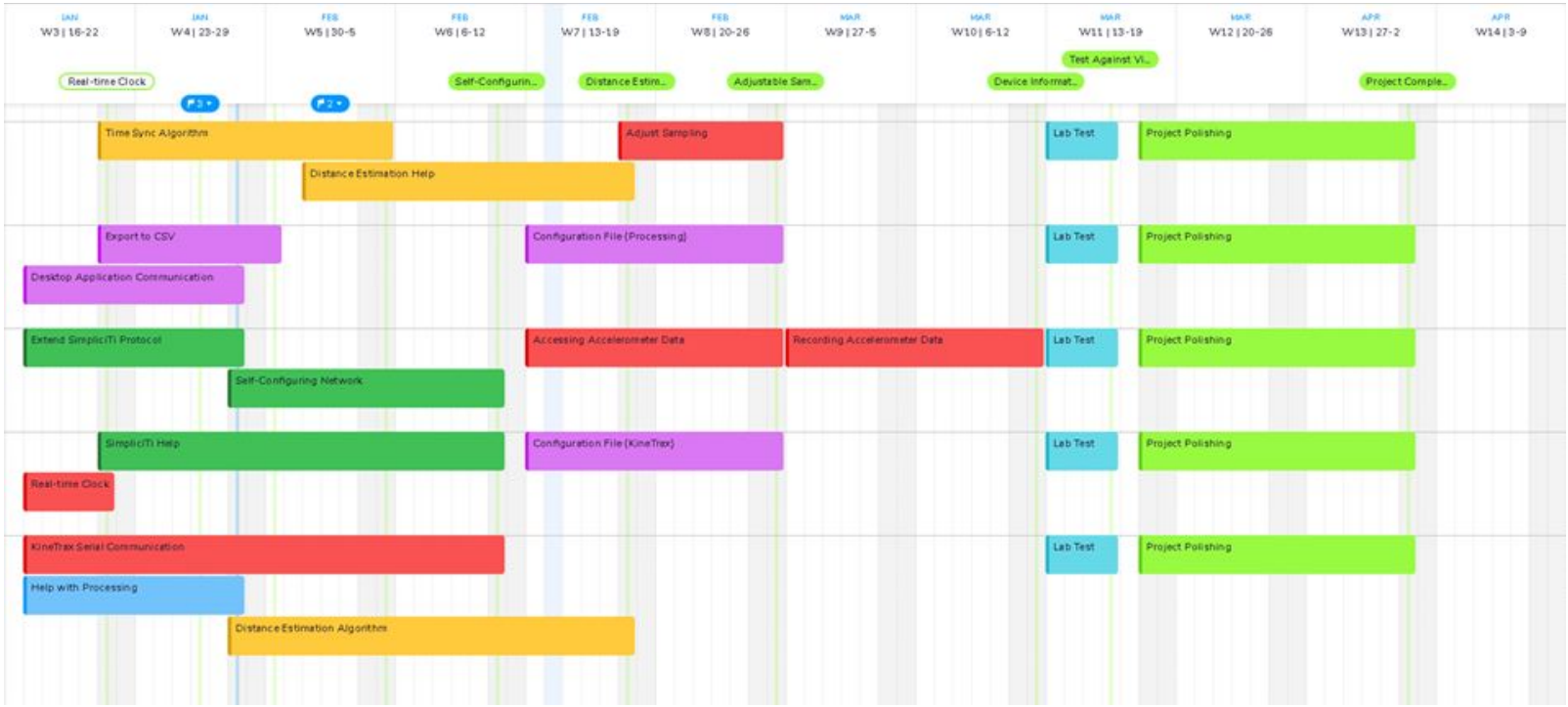
JAN
W3 | 16-22
JAN
W4 | 23-29
FEB
W5 | 30-5
FEB
W6 | 6-12
FEB
W7 | 13-19
FEB
W8 | 20-26
MAR
W9 | 27-5
MAR
W10 | 6-12
MAR
W11 | 13-19
MAR
W12 | 20-26
APR
W13 | 27-2
APR
W14 | 3-9
Test Against Vi...
Real-time Clock
Self-Configurin...
Distance Estim...
Adjustable Sam...
Device Informat...
Project Comple...
Time Sync Algorithm
Adjust Sampling
Lab Test
Project Polishing
Distance Estimation Help
Export to CSV
Configuration File (Processing)
Lab Test
Project Polishing
Desktop Application Communication
Extend SimpliciTi Protocol
Accessing Accelerometer Data
Recording Accelerometer Data
Lab Test
Project Polishing
Self-Configuring Network
SimpliciTi Help
Configuration File (KineTrax)
Lab Test
Project Polishing
Real-time Clock
KineTrax Serial Communication
Lab Test
Project Polishing
Help with Processing
Distance Estimation Algorithm

Figure 2: *Implementation Plan*

Conclusion

The KineTrax device has the potential to change gait analysis of patients with neurological impairments, physical impairments, prosthetic limbs, and healthy subjects of interest to sports medicine. As mentioned in the section, architectural overview, the project can be broken up into the following parts for each device: clock, read/write peripherals, wireless communication, device configuration and offloading of collected data via serial communication. In order to progress with the project, we have further broken down these major milestones into more manageable tasks and assigned deadlines for when we shall complete them. By breaking down our end goal for a finished product into milestones and tasks, we will be able to move forward in development. In turn, this means that the KineJax team will be able to deliver a complete project that has the potential to benefit a large number of people.