GONZAGA UNIVERSITY

School of Engineering

CPSC-1 Boeing B-Hive

April 20, 2010

Prepared By:

Robert Nertney

Scott Pfeifer

Jeffrey Hanneman

Boeing

Matthew Strohmeyer

Dr. Christopher Smith

Gonzaga University
Table of Contents

Objective.......................................................................................................................... 2
Vision, Mission.................................................................................................................. 2
Scope.................................................................................................................................. 2
Strategy............................................................................................................................. 2-3
Schedule............................................................................................................................ 3
Training............................................................................................................................... 4
Fuzzy Logic......................................................................................................................... 4
Neural Nets......................................................................................................................... 4
Data Source I..................................................................................................................... 4-5
Data Logger......................................................................................................................... 5
Data Logger Images........................................................................................................... 6
Data Logger Schematic....................................................................................................... 7
Data Logger Source Code.................................................................................................... 7-11
Data Source II................................................................................................................... 12
Programming Concept I................................................................................................. 12
Programming Concept II............................................................................................... 12
Programming Concept III............................................................................................... 13-16
Deliverables....................................................................................................................... 17
Conclusion........................................................................................................................ 18
Objective

The primary objective of this project was to learn how to use Boeing’s B-Hive API (Application Programming Interface), and then use it to show how quickly and completely we could go from knowing nothing to writing a full scale application. This included the recording any bugs that may have occurred, as well as suggesting improvements to the software from a client’s perspective.

Our secondary objective was to gather data from the City of Spokane’s Solid Waste Management (SWM), and then design our application to help improve the fuel mileage for their garbage collection vehicles. Due to time constraints and lack of data, we were forced to abandon our initial data source, and locate a new, pre-collected, dataset. We used movie ratings from Netflix in an attempt to use B-Hive to create a rating prediction application.

Vision, Mission

One member of our group was working with SWM before this project was assigned; he was on a contract between Gonzaga and SWM. He was tasked with equipping GPS units onto their fleet. SWM is currently using this data to track their employees’ driving patterns in order to increase fuel efficiency and design better pick-up routes. The group member’s advisor is also the Senior Design team’s advisor, and he suggested we use the GPS data from SWM as basis for our application.

Our vision for the project was to deliver an application that could easily be used by SWM to evaluate personality traits of various drivers and correlate them with fuel efficiency. This information could then be used to improve said fuel efficiency. When the data necessary for us to develop this application was delayed further and further, we had to locate a new data source. The new data source was a dataset of movie ratings. We then used B-Hive to create a movie rating prediction model. Our hopes for this model were to provide a system where users could rate movies and receive relevant suggestions for further viewing.

Scope

For this project, we initially met with SWM to attempt to understand what their opinion of the project was. Once we had an idea of what was desired, we met with a Boeing representative for training in the use of B-Hive. We had further meetings with both SWM and a representative of the Gonzaga University Sociology department to design a survey that could be used to obtain behavioral data. Time was concurrently spent developing hardware to measure acceleration of the vehicles, as well as further applications to obtain mileage, fuel usage, and other information. Once the project made the switch from SWM to the movie ratings system, the physical hardware solutions were no longer needed, and the scope of our project was limited to the movie data we already had, and the B-Hive API.

Strategy

Our strategy consisted of three major steps: first, learn the capabilities of B-Hive and how it could be used; second, obtain data to utilize B-hive; and third, do the actual programming. After the initial training was complete, we continued to meet with the SWM and Jeff Hanneman of Boeing to discuss the progress of the project. To gather the acceleration and fuel usage data,
one member of the group spent time developing a hardware solution to be placed in each vehicle, while also working with the SWM to implement GPS tracking. To obtain the behavioral data we also began meeting with Dr. Vikas Gumbhir, a faculty member in the Gonzaga University Sociology Department, to design a questionnaire that would provide us with the relevant data. Once the survey was initially designed, B-Hive was used to create an interface to administer the survey. Software was also written to read in the results. Those administering the survey (SWM) decided to administer the survey separately, and then return the results to us. The results never arrived.

It became obvious that the data would never get to us; this was combined with drivers unplugging the hardware solution. The SWM project was scrapped and the dataset for movie ratings, genres, and tags was obtained in its place. The team worked together to attempt to utilize B-Hive to make a prediction model that could make predictions as to what movies a user would enjoy based on their previous ratings, and the ratings of other users.

Schedule

We were given the B-Hive software midway through the fall semester and a professional from Digipen University came to Spokane to train us. The first part of the year was dedicated to finding out what SWM would like as a deliverable; from there we would start collecting data. Eventually, we came to the conclusion that the city did not know what they wanted, so the group decided to create a system to predict what a truck driver's daily fuel efficiency would be given their mood and actions before their shift.

To do this we needed to collect the behavioral traits of the drivers and the corresponding GPS data every day. The city claimed that they would have a surplus of quantitative data coming in from the GPS units measuring fuel efficiency; we set out to gather behavioral data. As SWM had a relationship in place with the Gonzaga University Sociology department, a representative was brought in to create the survey. We were told a survey would be ready by the start of the spring semester. However, SWM put our project on hold in order to administer an unrelated survey, also created by Dr. Gumbhir. Our survey wasn’t to be finished until mid-March. The group decided to construct their own survey to gather data about the drivers due to the dwindling time remaining.

We created a tool using B-Hive which was unique to the group’s survey to store and read in the results from the drivers. SWM insisted that they would administer the survey. The drivers had agreed to the group’s version of the survey under the agreement that the results were only to be seen by the group, and not SWM management; the overall results could be shared with the management, not the individual survey results. SWM disagreed with how the survey results would be handled, refusing anonymity for the drivers, and prevented the group’s survey from being administered. We cut ties with SWM, and began searching for a new dataset.

The movie rating dataset was obtained from a previous competition to improve on the Netflix “recommended for you” algorithm. Due to the limited amount of time left to work on the project, the group began meeting everyday and working with B-Hive and the datasets available.
Training

Due to the initial expectations of the project, during training we concentrated on the basic use of the software and its programming syntax. B-Hive was designed to incorporate neural networks, fuzzy logic, and genetic algorithms into the language itself; this feature makes it very easy to develop powerful artificial intelligence based applications. Initially, it was thought that fuzzy logic would be best for our project, so the majority of our training time was spent learning how to implement fuzzy logic within B-Hive.

Fuzzy Logic

Fuzzy logic tries to create an equation where the variables are inputs. It was initially chosen because it allows for a computer to give human readable outputs such as "very efficient driver", "efficient driver", "average driver", "inefficient driver", and "very inefficient driver". We could take data from the GPS and behavioral surveys as inputs, and show the aforementioned outputs. An example would be "IF a driver drinks (some amount) of coffee, THEN his efficiency will be affected (based on our equation, either positively or negatively)"

Neural Networks

Neural networks are networks of artificial neurons that are used to create artificial intelligence. Neural networks can be used to simulate real neurons. Like real neurons, neural networks rely on connections between nodes, or neurons, to form a basis for analysis, predictions, or behaviors.

Our SWM attempt would be to have all the behaviors as inputs, and predict a driver's daily fuel efficiency for the day. The network would read as "IF (behavior 1) AND (behavior 2) AND ... (behavior N), THEN the expected fuel efficiency will be (expected MPG)".

We modified our project to identify someone's likes and dislikes regarding movies. Our inputs were a unique user number (UID) and all the tags (i.e. genres and actors) for movies that the user rated above three stars (the median rating). It would train itself to recognize all the users and their preferred tags, and then recommend movies that other users who had a similar taste in movies (based on the tags) had seen and rated high.

The B-Hive neural networks were designed to have normalized inputs between [0, 1], where other neural network implementations normalize between [-1, 1]. With tens of thousands of ratings, the reduced range of normalization confused the neural network.

Data Source 1

Our initial source for data was going to come from the City of Spokane’s Solid Waste Management (SWM). SWM was already in the process (under the guidance of Rob Nertney) of equipping their fleet with GPS units to gather route data in order to produce a more efficient routing scheme. This dataset would include average miles-per-gallon, max speed, idle times, etc.

This data would be sampled every two seconds and stored into a buffer on the trucks themselves. Every 5 minutes, the buffer would fill and then be sent out over a 3G network to a
central data server hosted by the GPS companies. The data could be extracted in a comma-separated-value format (CSV), and easily parsed by B-Hive.

Our team agreed that the data provided by the GPS units was useful, but lacked a metric we thought imperative: accelerations. SWM outfitted their fleet with 6 different brands of GPS tracking solutions to determine which would best fit their needs, and none of them came with acceleration data. We wanted to measure whether or not the drivers were “putting the pedal to the metal” from stop to stop; that would incur a large fuel efficiency penalty.

This led us to design our own solution; our design of the accelerometer data logger can be seen in the figures below.

**Data Logger**

The data logging devices consisted of an Arduino microcontroller, an SD-card holder, and a tri-axis accelerometer. The accelerometer would read in changes in acceleration down to a granularity of .05 meters/second² in all 3 directions. These readings were each taken 20x/second and saved as text to the SD card. The data was calculated to consist of 720,000 data points for every 10-hour work day. The SD card retained the last saved memory position, and could begin writing where it had left off before; this was useful in the events that the devices lost power, either for lunch or for the end of day and for the fact that they did not need to be checked and dumped into B-Hive every day.

This idea was abandoned when we were forced to change datasets. The code for the data logger can be found in Appendix A. The libraries consist of approximately 21 pages, and can all be found at [http://www.rob-optics.com](http://www.rob-optics.com) if interested.
Data Logger Images
Data Logger Schematic

Data Logger Source Code

```c
#include <sd-reader_config.h>
#include <sd_raw.h>
#include <sd_raw_config.h>

boolean WRITE = false; //Interrupt flag
const int buttonPin = 2; //The Start Button
const int ledPin = 7; //The "running" led pin

int sample(); //prototype for the sample function
int readDisk(); //prototype for the read function
void printWelcome(); //prototype for the welcome function
void erase(); //prototype for the erase function

byte incomingByte; //the var that holds the serial byte
byte x1[2], y1[2], z1[2]; //arrays for the 3 dimensions of measurement
```

/* Function Name: Setup
 * Function Use: Initializes all necessary values/pins
 *
 * Variables Created: None
 * Variables Used: ledPin, buttonPin
 *
 * Explanation: This function will initialize the user interface (the start
 * button and the running LED. It also sets up the interrupt; it will watch
 * digital pin 2 for a falling edge. Finally, the serial interface is
 * initialized, and the card is as well.*/
void setup()
{
  pinMode(ledPin, OUTPUT);  // set the ledPin as an output pin
  pinMode(buttonPin, INPUT); // set the buttonPin as an input pin

  // interrupt 0(pin2) occurs on falling edge
  attachInterrupt(0, done, FALLING);

  Serial.begin(9600);       // begin serial interface
  delay(1000);

  printWelcome();          // print welcome message
  if(!sd_raw_init())       // init the card
  {
    Serial.println("MMC/SD initialization failed");
  }
}

/******************************************************************************
 * Function Name: done
 * Function Use: Interrupt routine for monitoring the start/stop button
 * *
 * Variables Created: None
 * Variables Used: ledPin, buttonState, WRITE
 * *
 * Explanation: This routine will monitor the button on the protoshield. Upon
 * initial depression, the unit will light the led signaling that sampling is
 * occurring. Upon the subsequent button press, the unit will stop sampling
 * acceleration data, and will loop infinitely until the user disconnects the power.
 * /

void done()
{
  digitalWrite(ledPin, LOW); // LED = 0 during init and after 2nd press
  if(WRITE == false)         // Check to see if it's the first press
    WRITE = !WRITE;         // if it is, then sample
  else
    while(1);              // else end.
}

/******************************************************************************
 * Function Name: loop
 * Function Use: Required Arduino function that loops. Waits for button
 * and displays saved data or clears it.
 * *
 * Variables Created: None
 * Variables Used: WRITE, ledPin, incomingByte
 * *
 * Explanation: This function is required in the arduino environment. In this
 * scenario, it checks to see if the WRITE flag has been triggered by a button
 * push. If not, it waits on the serial line for the command 'r' to dump it's data,
 * and 'x' to clear it.
 * /

void loop()
{
  if(WRITE)                 // if the flag is set to enable writing
  {
    digitalWrite(ledPin, HIGH); // turn on the LED
    sample();                  // and sample
  }
else  //else turn off the light (make sure)
{
digitalWrite(ledPin, LOW);

incomingByte=Serial.read(); //monitor the serial interface

switch(incomingByte)
{
    case 114:  //read the data 'r'
        readDisk();
        break;
    case 120:  //clear the sd card 'x'
        erase();
        break;
    default:
        break;
}
}

*****************************************************************************
* Function Name: sample
* Function Use: Samples the accelerometer in 3 directions and saves data.
* *
* Variables Created: i, x, y, z, lowx, lowy, lowz, highx, highy, highz,
* inByte
* Variables Used: WRITE, x1[], y1[], z1[]
* *
* Explanation: This function reads the analog pins and saves the data.
* The data is then separated into 2 bytes: its MSB and LSB. These bytes
* are then sent to the write function and then on to the SD card itself.
* *
* Issues: In order to keep up with the speed we needed for acceleration
* data, we had to forgo using arrays and loops. Although the code looks
* sloppy, there was no alternative.
* */

int sample()
{
    int i;  //index of the location on the SD card
    int x,y,z;  //raw, analog integers for the directions
    byte lowx,lowy,lowz;  //LSBs of the respective directions
    byte highx,highy,highz;  //MSBs of the respective directions
    byte inByte;

    Serial.println();
    Serial.println();
    Serial.println("Sampling...");

    while(WRITE)  //While the WRITE variable is true
    {
        if(Serial.available()>0) //manual Serial override
        {
            inByte=Serial.read();
            if(inByte==113)  //\'q\' will quit prematurely
                return 0;
        }

        x=analogRead(1);  //read x
        Serial.print(x,DEC);
// Convert int to 2 bytes
lowx=x&0xFF;    // LSB
highx=x>>8;     // MSB

x1[0]=lowx;    // store in an array for the write func.
x1[1]=highx;

if(!sd_raw_write(i,x1,2))//send to the write function
    Serial.print("Write error");
delay(50);       // delay as little as possible
i=i+2;          // increase index by 2 bytes

y=analogRead(2); // read y
Serial.print(y,DEC);
// Convert int to 2 bytes
lowy=y&0xFF;     // LSB
highy=y>>8;      // MSB

y1[0]=lowy;     // store in an array for write func.
y1[1]=highy;

if(!sd_raw_write(i,y1,2))//send to write function
    Serial.print("Write error");
delay(50);
i=i+2;

// see x and y above. It's the
z=analogRead(3); // same, I promise, it really is!
Serial.print(z,DEC);
lowz=z&0xFF;
highz=z>>8;
z1[0]=lowz;
z1[1]=highz;
if(!sd_raw_write(i,z1,2))
    Serial.print("Write error");
delay(50);
i=i+2;
}
return 0;
}

/***************************************************************************/

* Function Name: readDisk
* Function Use: Prints the contents to the serial bus
*
* Variables created: low, high, info[], i, result, lastresult
* Variables Used: None
*
* Explanation: This function traverses the memory card, and displays the
* data on the screen. The most data we will use is measured at
* 20Hz * 60s/min * 60min/hr * 10hrs/work_Day. This max value comes out to
* 720,000 cycles we need to check per day. We also want to stop the
* printing of data if the end of data is reached (if less than 720K).
* The card is initialized to 00000...00
*/
int readDisk()
{
    byte low;     // LSB
    byte high;    // MSB
byte info[2];         //Array to grab the data
unsigned long i;      //long number to index to our max value
int result, lastresult = 0;      //checking for last values
Serial.println();
for(i=0;i<720000;i=i+2)
{
    if(lastresult == result)
    {
        Serial.print("*");      //once the last value on the card found
        break;
    }
    sd_raw_read(i,info,2);    //call the read function
    low=info[0];              //store the LSB
    high=info[1];             //store the MSB
    result=high<<8;           //convert to DECIMAL
    Serial.print(result+low,DEC);//print it
    Serial.print(" ");
}

ERCHANT**************************************************************************************************

* Function Name: PrintWelcome
* Function Use: Show a display on Serial to start reading or erasing
* 
* Variables Created: None
* Variables Used: None
* 
* Explanation: None
*/

void printWelcome()
{
    Serial.println("------------------");
    Serial.println("Data sampling system");
    Serial.println("send r to read disk");
    Serial.println("send e to erase disk");
    Serial.println("Ready....");
    Serial.println("------------------");
}

void erase()
{
    x1[0]=0;
    x1[1]=0;
    for(unsigned long i=0; i < 720000; i=i+2)
    {
        Serial.println("x");
        if(!sd_raw_write(i,x1,2))
            Serial.print("Write error");
    }
}
Data Source 2

The movie ratings data set was obtained from MovieLens.com. The data is separated into three groups. One group has 100,000 ratings entries, then 1 million and finally 100 million entries. We elected to use the smallest of the three groups to train our algorithms and then used the larger groups to test. There are two additional files associated with the movies; the genres file which gave the genres of the movies and the tags file which includes tags given to the movies by users. This file includes movie id, user id, the tags given to the movie by the user, and the time at which it was given. The tags are completely random and highly disassociated. Any movie can be tagged with "good", "want to own", or any other tag. Frequently, the tags are not spelled correctly if they do contain information that could be pertinent such as actor names, directors, themes or some other logical descriptor. We identified patterns within the likes and dislike of different users and used those patterns to create predictions of which movies a user would like.

Programming Concept I

Initially we attempted to use the Neural Network capacity of B-Hive to do pattern recognition. Utilizing our dataset of 100,000 ratings, we attempted to show the neural network each iteration and rating so that it would recognize patterns of ratings attached to movies by users.

![Diagram]

Using this method, we were unable to generate good results. This design, although initially thought to be acceptable, was found to be inherently flawed. The diagram above tried to find a correlation between an arbitrary user id and an arbitrary movie id. Finding a correlation between two arbitrary numbers is impossible, and we spent valuable time training our neural network to predict, in essence, gibberish.

Programming Concept II

Due to the size of our dataset, and the normalization of the inputs, we found that we were unable to get proper results for known values even after training many times. Instead, the neural network tended to give us average values of all the movies, regardless of the user inputted. This, in combination with our lack of knowledge regarding the syntax of a neural network's weights, marked the end of our attempts to use neural networks.
Programming Concept III

We then used knowledge of C, and Objective-C to use B-Hive to implement a solution that would predict movies a particular user would enjoy. Flow charts can be found below.
How would a given user like a specific movie

How would user like Movie

User ID and Movie ID

create Correspondence Matrix

Predict Rating Using Correspondence Matrix

report predicted rating
Correspondence Matrix

Create correspondence matrix

For each User

For Each Movie

If both user and member have rated movie

Difference of user rating and member rating

Case statement

\[ \text{diff} = 0 \]
\[ \text{cor}(1, \text{UserID})++ \]

\[ \text{diff} = 1 \]
\[ \text{cor}(1, \text{UserID})++ \]

\[ \text{diff} = 3 \]
\[ \text{cor}(1, \text{UserID})-- \]

\[ \text{diff} = 4 \]
\[ \text{cor}(1, \text{UserID})=0 \]
Set bad member flag

True
Predict the rating function.

Predict Rating

For each member

If member has rated Movie

Predicted rating = numerator / denominator

multiply rating by $\text{cor}(i, ID)$ and add to numerator

add $\text{cor}(i, ID)$ to denominator

Function for getting top three movies a user should like.

User should like

create user likes list

create similar users list

Create user should like list

find three most listed items and report
Deliverables

Due to the availability of data, true programming with B-Hive was limited to the design of the survey, including reading in and writing out files, and then the past few weeks working with the software with the movie data. All told, we believe we spent 150 hours working with B-Hive. Given the appropriate availability of data, we feel we would have spent more time in the actual API of B-Hive, and as such, had a better result.

We felt that we had a skewed representation of training for B-Hive as we were trained on elements that were then not used (fuzzy logic). Instead, we ended up using Neural Networks, which we did not receive specific training on. We felt that if we had a better picture of what we wanted to accomplish with the API during training, we would have been better able to ask specific questions that would have led to a positive result.

This being said, we were able to develop a solution using B-Hive and some of its unique features to develop a model for predicting selections. The ability to create a Graphical User Interface rapidly with B-Hive made it much easier to develop our program. We also found some of the following B-Hive specific features to be useful:

- Lists
- File Inputs/Outputs
- Graphical User Interface (forms and windows)
- Debugging with the console (interpretive vs. compiled)

Boeing also requested that we identify issues, or ways to improve upon the software. We felt that the documentation often made assumptions about the understanding of syntax for the simple data types, which in practice created confusion. Many of the biggest frustrations were in working with matrices, arrays, and lists as what member functions were available were not easily found, and syntax for those we did find was not shown. The example code that was provided for neural networks did show how to use the software, but the lack of commenting meant that we blindly copied certain elements that we did not understand. This made it especially difficult to design and implement our own neural network for the model.

We did find a few bugs to report to Boeing. We found that accessing out of bounds matrix elements caused the program to hang, and not crash gracefully. This lead to no identification of the cause of the crash. It wasn't until we identified the issue with the out of bounds elements that we were able to fix it which increased time spent debugging. A graceful crash pointing to the relevant line would be an improvement. We also found that when a program was written poorly as to create an infinite loop, it was impossible to close the program without opening the task manager to kill the process. Too many open consoles were found to cause incorrect outputs in subsequent consoles; closing all consoles and restarting the software would remedy the situation.
Conclusion

In conclusion, we were able to develop a functioning model using the B-Hive software. We found that many of the elements of B-Hive that were the most helpful were not fully taken advantage of due to the issues of obtaining the data, limiting time to spend on. We feel that with a more adequate plan for the software, and with the proper availability of the data, we would have been much more successful. We also felt that the power of B-Hive is not necessary for many projects, and B-Hive should only be used when its features can be adequately appreciated by the client.

Even with our perverted experimentation with B-Hive due to complications with the City of Spokane and the drivers' union, we found that B-Hive was a very powerful API that would benefit from improved documentation, and a better understanding of how to take advantage of its power.