Garrett Burton & Wendy Kwok
Bryce Himebaugh
CSCI-P442
20 April 2016

Trail Counter Time Group Final Report

Introduction

The task we were assigned was to find a time correction and error analysis algorithm that accurately corrects the time on our trail counter. Naturally the trail counter will be placed outside resulting in changes in temperatures over the course of it's year or so battery life. Due to these changing temperatures slowing down or speeding up the crystal oscillator that keeps the time in our trail counter, we needed to see how we could correct the time stamped to give us a more accurate view of traffic passing over the counter. In order to get to our final goal of the time correction and error analysis algorithm we needed to complete 3 milestones that acted as a guide to our goal. The milestones, in order, were to first collect time and position every minute from a GPS puck over 24 hours and compare accuracy to workstation time, second develop firmware to compare and store GPS time, RTC time, and temperature, and lastly deploy and collect system data and analyze results to develop error correction model.

Milestone I

The first milestone was very simple, what we needed to accomplish was collect GPS data from a puck provided to us and compare that data to the workstation we were connected to. In order to connect to the streams of data being pulled in by the GPS puck wired to a STM we used a python package called pyserial which allowed us to get streams of data via the serial port. For every 60 seconds we would flush the serial input buffer and collect the newest data. Next, we took the data block and tokenized it allowing us to split into more readable and comparable data.

The format in which we collected the data was “GPS time, PC time, latitude, direction of latitude, longitude, direction of longitude.” The below graph shows the difference in GPS time (blue) vs PC time (red).

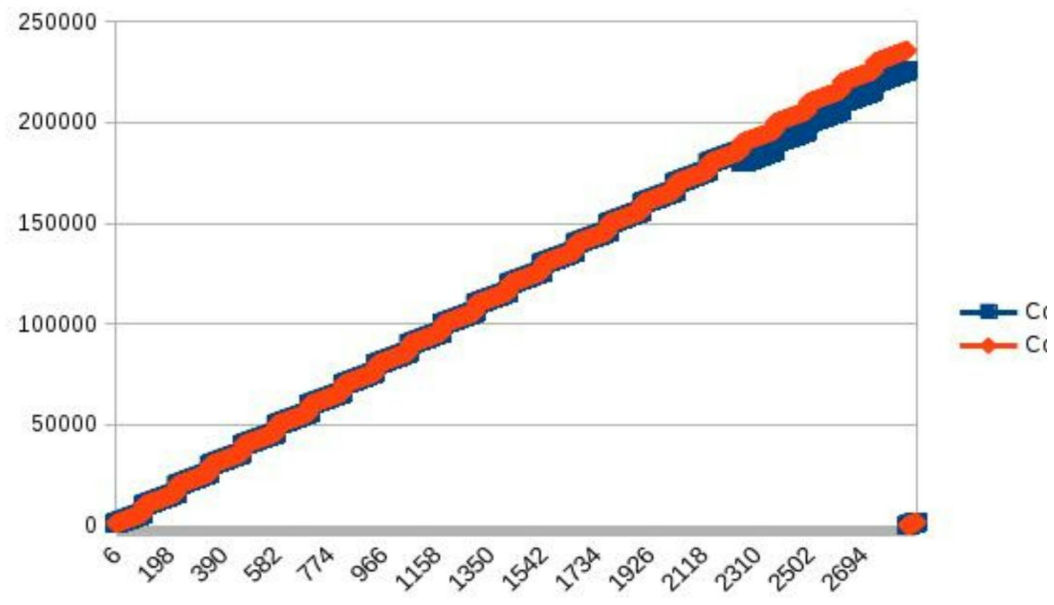


Figure 1: Blue = GPS, RED = PC

Here is a small section of our collected data that is represented in the graph.

```
1,001029.4, 001023.416414,1,3910.7136,1,N,1,08631.7284,1,W
2,001129.6,001123.620716,2,3910.7243,2,N,2,08631.7429,2,W
3,001229.8,001223.822795,3,3910.7163,3,N,3,08631.7360,3,W
4,001330.0,001324.033782,4,3910.7170,4,N,4,08631.7374,4,W
5,001430.2,001424.188830,5,3910.7163,5,N,5,08631.7374,5,W
6,001530.4,001524.421096,6,3910.7054,6,N,6,08631.7223,6,W
7,001630.6,7,001624.629745,7,3910.7047,7,N,7,08631.7215,7,W
8,001730.8,8,001724.764472,8,3910.7050,8,N,8,08631.7214,8,W
9,001831.0,9,001825.008417,9,3910.7058,9,N,9,08631.7225,9,W
```

As seen in the graph above, the RTC seemed to drift as the day neared an end.

Milestone II

Milestone 2 was by far the biggest hurdle in our process. In milestone 2 we needed to develop a firmware that could collect and store GPS time, RTC time, and temperature. This data is needed in order for us to find the RTC time drift compared to the GPS's accurate time. To store the data we decided it would be best to use an SD card on the board. We wanted to get a more accurate time off the STM and also a more accurate measurement between the times were sampled. In order to accomplish both of these we needed to first access the high resolution timer on the STM in order for our to get the most accurate time. Next, a pin was connected to the GPS's led light, which also was lighting on every second. A wire was soldered to the GPS to use a pin on the board as an interrupt for data collection. This signal to triggering an interrupt would increase the accuracy of data collection to minimize the varying time between collecting RTC time and GPS time. Here's a look at our call back function:

```
/* Callback for time compare */
static void extcb(EXTDriver *extp, expchannel_t channel) {
  (void)extp;
  (void)channel;

  chSysLockFromISR();
  chBSemSignalI(&bsem);
  chSysUnlockFromISR();
}
```

The real challenge was getting the temperature sensor up and running correctly; it is difficult to measure time if you're not managing time correctly. In order to get the temperature, we use the internal temperature sensor of the STM32 using the ADC. Next, we needed to read from channel 16, we worked on this issue for around 2 weeks to setup and make sure that it everything was working correctly and not disturbing time. Here is a look at the initialization that was the focus of our time:

```
Static const ADCConversionGroup adcgrpcfg = {
FALSE,
1,
NULL,
NULL,
0,
0,
ADC_CCR_VREFEN | ADC_CCR_TSEND, //Bryce's addition
{0,
ADC_SMPR@_AN16(ADC_SMPR_SMP_601P5)},
{ADC_SQR1_NUM_CH(1) | ADC_SQR1_SQ1_N(ADC_CHANNEL_IN16),
0,
0,
0},};
```

Once we successfully got the temperature sensor reading we took pre-data to make sure that temperature was working correctly. Unfortunately, there was no data about how to calibrate the temperature sensor, and there had to Upon reviewing this data, we noticed the temperature in the room was around 64 F we were receiving negative degrees in approximately -10 C. To make sure that the readings were just wrong by an offset, we put something hot directly onto the board(a.k.a Collin’s chicken marsala), and made sure that the temperature is climbing normally. Although we were not able to find the factory calibrated value for temperature calibration, we could still process data that we collected and find the offset later.

For the data collection, data was stored every second onto an SD card reader/writer. One of the difficulties with setting the SD card writer is that we ran into error 3 and error 9 constantly. Because the SD card writer requires big amounts of space in order to write, the solution was to increase the thread size of our data collection thread. Once we changed our thread size from 128 to 2048 we were able to successfully write and store onto an SD card.

Milestone III

Our third milestone that needed to be achieved to complete our goal was taking correct data so that we can utilize it to analyze and see how we can correct for time drift. Although we were able to successfully take 3 consecutive days of data, we realized after that the drift was too big and more inaccurate and rendered the data unusable. As a solution, we switched into a board that has a 100ppm oscillator; this should increase the accuracy of collecting drift data. Shortly after that, ~12 hours of data were taken to be analyzed.

Milestone IV

The last milestone was to analyze data collected from previous milestone and attempt to find a way to correct for the time drift caused by an inaccurate oscillator. With the information of time drift per second and the temperature during those drifts, we should be able to calculate the actual frequency of the oscillator at certain temperatures. Certain constraints are posed in this milestone and milestone III. Because we are trying to tie the correlation between temperature and drift, we must be able to take constant consecutive time of a certain temperature. If there were shifts between two temperatures over a time frame then drifts in time may be accidentally attributed to a certain temperature when unecessary. As a result, we were not able to collect enough data to with certain temperature ranges.

Analyzation Method

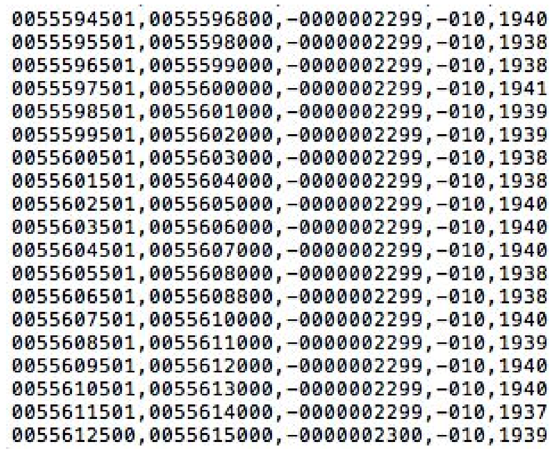
In order to analyze the data we require a temperature and consecutive data over the temperature. Compared to something with great accuracy of time(GPS), we were able to find the drifts in time based on that temperature. For the attached oscillator on the STM32, the expected oscillation was 32768 times per second, which is the frequency. We do not expect to have no oscillator error, but we do expect that if there exists any error it should be static. The drift will tell us how what frequency it actually is at each time, which enables us to find the number of milliseconds drifted in that temperature.

Expected Data Analyzation

Based on the oscillator ppm, the crystal should cause drift of at most 8 seconds a day. The data that we are to analyze should also create somewhat of a parabolic curve when comparing the actual frequency to the temperature. Assuming that even with a short temperature range, we can map out parts of the parabolic curve, and from there find an algorithm or equation that predicts the other temperatures' drift time.

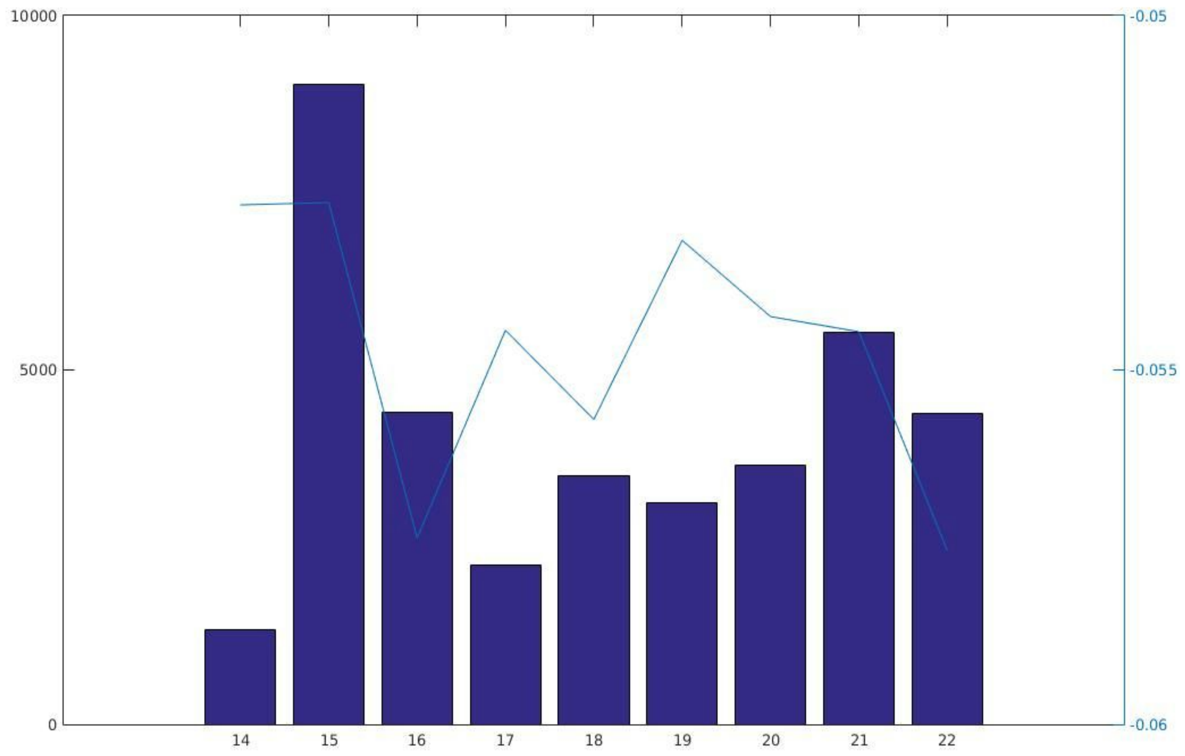
Analyzation Data

Here is a sample of the data taken, which shows for -10 degrees celsius, the drift is approximately 1ms per 19 seconds.



```
0055594501,0055596800,-0000002299,-010,1940
0055595501,0055598000,-0000002299,-010,1938
0055596501,0055599000,-0000002299,-010,1938
0055597501,0055600000,-0000002299,-010,1941
0055598501,0055601000,-0000002299,-010,1939
0055599501,0055602000,-0000002299,-010,1939
0055600501,0055603000,-0000002299,-010,1938
0055601501,0055604000,-0000002299,-010,1938
0055602501,0055605000,-0000002299,-010,1940
0055603501,0055606000,-0000002299,-010,1940
0055604501,0055607000,-0000002299,-010,1940
0055605501,0055608000,-0000002299,-010,1938
0055606501,0055608800,-0000002299,-010,1938
0055607501,0055610000,-0000002299,-010,1940
0055608501,0055611000,-0000002299,-010,1939
0055609501,0055612000,-0000002299,-010,1940
0055610501,0055613000,-0000002299,-010,1940
0055611501,0055614000,-0000002299,-010,1937
0055612500,0055615000,-0000002300,-010,1939
```

Taking all the consecutive data that we can process, and omitting the ones that do not have enough data samples compared to the other data samples, we compare the temperatures and how much they drift(based on collected data). The drift is measured into milliseconds per second, and the temperatures in celsius in increasing order.



The graph above compares the temperatures taken to the drifts per second. The bar graph symbolizes the temperatures while the line graph represents the drift per second. One difficulty was that we only had a short range of temperatures(approximately 55 to 75 degrees F). Another problem with the data was that there was some unknown factor that caused the temperature data to be taken with very much noise, causing it to not be as accurate.

Conclusions

Because we were not able to collect enough consecutive data to better measure the drift at individual temperatures, we think that more consistent data is necessary in order for us to see a more accurate visualization of the drifts. As a result of time constraint and unknown noise in the collected data, we were not able to conclude or find a function to correct the time drift accurately.