A Real-time Software GPS Receiver

A summary of GPS and optimizations needed for real-time receiver operation

Samuel R. Price

Advisor:
Dr. In Soo Ahn

Department of Electrical and Computer Engineering

Bradley University

Peoria, IL

Friday, May 29, 2009

Table of Contents

A List of Figures iv

Abstract 1

Acknowledgements 2

Introduction 3

- Theory of GPS 3
  - GPS Trilateration 3
  - GPS Communication Theory 3
- Software Receiver Overview 5
  - Sampler 6
  - Sampling 6
  - GPS Signal Acquisition 7
  - Attempted Coarse Acquisition 8
  - Tracking 9
  - Decoding 10
  - Positioning 11

Comparison of Real-time Open Source Projects 13

Future Recommendations 13

- Simulated Signal Data 13
- SIMD Optimizations 13
- Acquisition Improvements 13
- Locating the Carrier Frequency 13
- Tracking Improvement 1 14
- Tracking Improvement 2 14
- Tracking Improvement 3 14
- Position Updates 14

Conclusion 14

References 15

Appendix A - Coarse Acquisition 16

Appendix B – Cosine / Sine lookup 18

Appendix C - Optimized Tracking loop 19

A List of Figures

Figure 1 GPS Triangulation ... 3
Figure 2 Time Bias Error ... 3
Figure 3 Correlations strength, Offset vs Chips ... 4
Figure 4 Program Flowchart ... 6
Figure 5 Coarse Acquisition Structure ... 7
Figure 6 Correlation Strength Over Various Bins for Satellite 7 ... 7
Figure 7 Step 1 ... 7
Figure 8 Step 2 ... 7
Figure 9 Step 3 ... 7
Figure 10 Step 4 ... 7
Figure 11 Attempted structure ... 8
Figure 12 Signal Strength Vs # of Bins ... 8
Figure 13 Tracking loop ... 9
Figure 14 Basic tracking loop ... 10
Figure 15 Improved tracking loop ... 10
Figure 16 Google Earth Interface ... 12
Figure 17 Scaling error ... 12

Abstract

Software-defined receivers are used to implement and test various algorithms before hardware implementation. In this project, a real-time software-defined GPS receiver has been implemented on a dual core 3.0GHz x86 platform running Windows XP.

The object-oriented programming (OOP) methodology is used to separate the system into three modules: satellite acquisition, carrier tracking, and position estimates.

Single instruction multiple data commands and multi-threading programming are used to handle data sampling, acquisition loops, tracking loops, and position calculation.

The GPS receiver can track 7 satellites with less than 50% CPU usage. It successfully updates the position every 600ms.

Acknowledgements

I would like to thank Dr. In Soo Ahn, and Dr. Yufeng Lu for all of their help. The entire ECE department was always willing to answer any questions I had, and point me in a correct direction when faced with a problem. My years at Bradley have made me truly appreciate lifelong learning. A final thank you goes to Northrop Grumman for financially supporting the senior capstone project.

Introduction

Software defined radios are widely used in the scientific community for the study of different communication systems, prototyping new algorithms, and performing batch processing that would be impractical to do with hardware. Another benefit of software receivers is the fact that application specific hardware is not needed after a dataset is collected. This allows development to be completed by anyone with a collected dataset and a computer. In this paper a brief theory of GPS signal communication and the details of implementing a real-time software defined GPS receiver will be presented.

Theory of GPS

GPS Trilateration

GPS, like most positioning systems, works by using the distance from reference points to determine a user position. A more familiar example would be measuring the time it takes to send a message to a cell phone tower and receive an acknowledgement of successful reception. The distance from the tower would be half of this measurement multiplied by the speed of light. GPS is similar to this, however the reference points are constantly moving, and it is a transmission only system. To accommodate these last two points, the number of seconds since the start of the week is transmitted every six seconds, and ephemeris (satellite flight path) information is sent every 30 seconds. The time difference between a local clock and the time of reception of the current time is directly proportional to the distance from the satellite. The ephemeris information provides the current position, allowing the range from this point to be known. Using three satellites, an (X,Y,Z) position could be found by solving three equations simultaneously. If the reference clock is off by a millisecond, this would relate to an error of 186 miles. A fourth satellite is needed to resolve the time bias between the satellite clock and the receiver clock. An example of GPS triangulation and error caused by time bias is shown below in Figure 1.

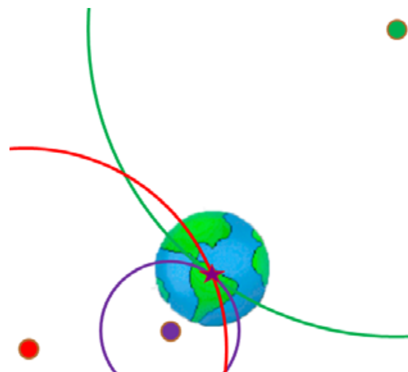


Figure 1 GPS Triangulation

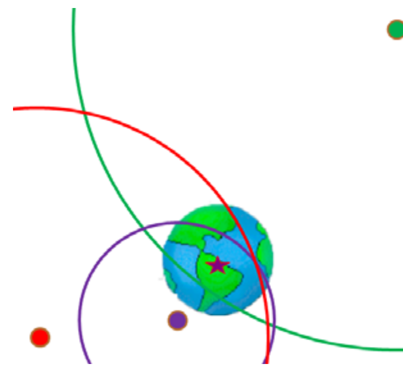


Figure 2 Time Bias Error

GPS Communication Theory

As stated in the above paragraph, GPS satellites are transmitting ephemeris, ionosphere, and time information. This data is transmitted on a carrier frequency using binary phase shift keying (BPSK). Code division multiple access (CDMA) is used to allow multiple satellites to transmit on the same frequency band. The satellite signal seen by a receiver can be summarized by the following expression.

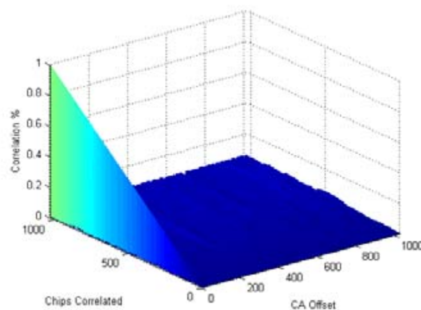
$$\text{Satellite signal at time t} = CA(t-\tau)N(t-\tau)\cos[2\pi(L_1 + D_f)t + \theta]$$

- C/A is the course acquisition code (C/A), it is a pseudo-random sequence transmitted using BPSK. The code length is 1,023 chips long, and transmitted at a rate of 1.023MHz. The terminology 'chips' is used rather than 'bits' to emphasize the fact that no information is contained in the C/A signal.
- N represents the navigation data; it is transmitted at a rate of 50Hz. Twenty complete C/A codes are sent during the transmission.
- θ is the phase offset seen at the receiver
- τ is the propagation delay from the satellite, this value generally ranges from 68ms to 87ms.
- D_f is the Doppler frequency offset caused by the movements of satellites, and receivers.
- L_1 is the satellite's 1575.42MHz carrier frequency of the BPSK signal.

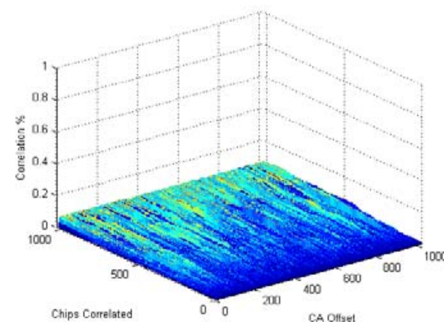
CDMA

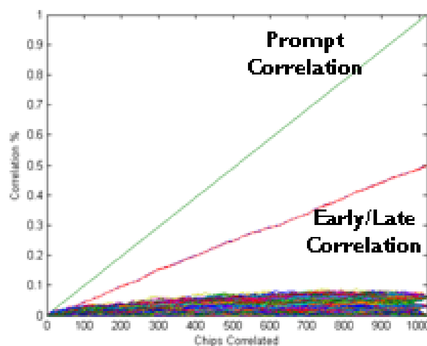
Code division multiple access (CDMA) is a channel access method that allows multiple transmitters to share the same frequency. The C/A code makes use of this method. Determination of a signal presence is done by correlating an incoming signal with a reference C/A code. Figure 3 show the correlation of different C/A codes with each other, different chip offsets, and amount of chips used in correlation.

C/A code 1 correlated with C/A code 1



C/A code 1 correlated with C/A code 2





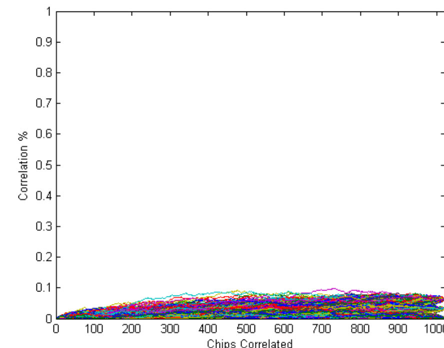


Figure 3 Correlations strength, Offset vs Chips

From Figure 3 several important statements can be made.

- Any correlation below 10% can be considered to be absent of a GPS signal.
- The correlation strength is 100% when 1,023 chips are correlated and the offset is 0. This is known as the prompt correlation and is shown in green in the lower right image.
- When the offset is ½ a chip early or late, the correlation strength is ½ of the prompt correlation. These offsets are known as the early/late correlations, and shown in red in the lower left Figure.
- When the offset is 1 chip early or late, the correlation strength drops below 10%.
- The early, prompt, and late correlations form a linear line as additional chips are correlated.
- The signal to noise ratio(SNR) is directly proportional to the slope of early, prompt, and late correlations.
- If the SNR ratio is low, additional chips may be needed to obtain a correlation above the noise floor.
- The code length of the C/A code will change due to the Doppler Effect, and the correlation offset will slowly increase as more chips are correlated.

Navigation Data

The navigation data is transmitted at a rate of 50Hz. Thus, twenty C/A correlations are completed for every navigation bit transmitted. The information contained inside of the navigation message includes the current time, parameters needed to calculate the satellite position, and ionosphere delay corrections. This information is transmitted at a rate of 6 seconds, 30 seconds, and 12.5 minutes. The parameters needed to calculate the satellite position make up the ephemeris file.

Software Receiver Overview

Software defined GPS receivers are composed of four main components; sampling, acquisition, tracking, and positioning. In order to maintain real-time operation, these components must operate simultaneously while accessing the same data. Posix threads are used to allow the parallel operation of these components. Mutexes are incorporated to protect data from being written and read by separate threads. Efficiency requirements and software structure change dramatically when compared with batch processing. Singleton classes are used for objects that can only exist once.

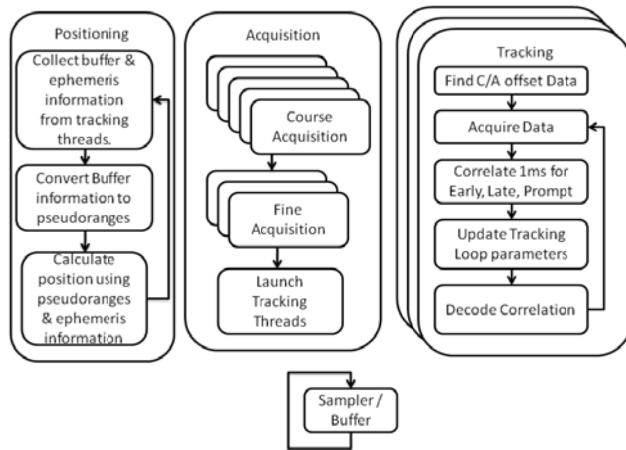


Figure 4 Program Flowchart

Sampler

For the reception of raw data, the SiGe GN3S sampler V1 was used. The sampler consists of two onboard chips. The first chip is the SiGe Semiconductor SE4120L. The SE4120L completes the following steps to the incoming signal. A low noise amplifier is applied to the signal. The GPS signal is down converted from 1.5 GHz to 4MHz. Finally a 2 bit quantization is applied to the signal. This 2bit quantized data is then passed to the Cypress Semiconductor EZ-USB FX2 and transmitted through a USB to a computer. The original firmware contained a 45 second recording limitation. This was overcome by porting fx2-programmer from Linux to Windows and reflashing the firmware with the GN3S*nix firmware.

In the receiver implementation pseudoranges were directly related to the sampling frequency. This results in a maximum error of 18.73 meters per satellite for a sampling frequency of 16MHz. Fine time approximations between samples allow finer results, and would be needed for slower sampling frequencies.

$$\frac{1}{16MHz} \bullet \text{Speed of light} = 18.737 \text{ m}$$

Sampling

The sampling component is responsible for collecting samples and storing them in a buffer. The overhead caused by dynamic memory creation/deletion can be avoided by using a circular buffer to store raw data. The tracking algorithm uses approximately 1ms for each loop. To simplify the tracking algorithm, approximately 1ms of data is copied from the end of the buffer to the beginning. The copied data allows tracking loops to avoid buffer overruns when processing 1ms data intervals. The minimum amount of data to be copied can be found from the following equation.

$$\frac{1.57542GHz + 10kHz}{1.57542GHz} \bullet 1\ ms \bullet 1 \frac{\frac{1}{2}\ chip}{1023\ chips} \approx 1.0068ms$$

Selection of the buffer size should be at least 20ms to allow acquisition of weak signals. This would result in 9% of the buffer being duplicated. In the implementation 2 seconds of data is used for the buffer resulting in .1% of the buffer being duplicated.

GPS Signal Acquisition

The acquisition thread's job is to locate the presence of a satellite signal, and launching tracking threads for each signal found. It performs this process in two steps. First, a course acquisition is performed over 41 different frequency bins. The implementation was based on a parallel code phase search described by Borre, et al[1]. Figure 5 shows the structure of each frequency bin search. Figure 6 shows the results of correlation strength over various frequency bins. Figures 7 through 10 show the results of steps 1 through 4 shown in Figure 5. The location of a single spike in step 4 is directly related to the C/A code offset position. This is shown in Figure 10. After a frequency bin is located, a satellite's carrier frequency needs to be located. This is done by removing the C/A code from the incoming signal and performing another FFT to determine the carrier frequency. The amount of time this takes makes it impractical to read directly from the sampling buffer. Instead, 20 ms of data is copied to a temporary buffer and is used for the acquisition loop. For a stationary receiver, the carrier signal will not change more than 2 Hz over a period of 2 minutes, so this is acceptable.

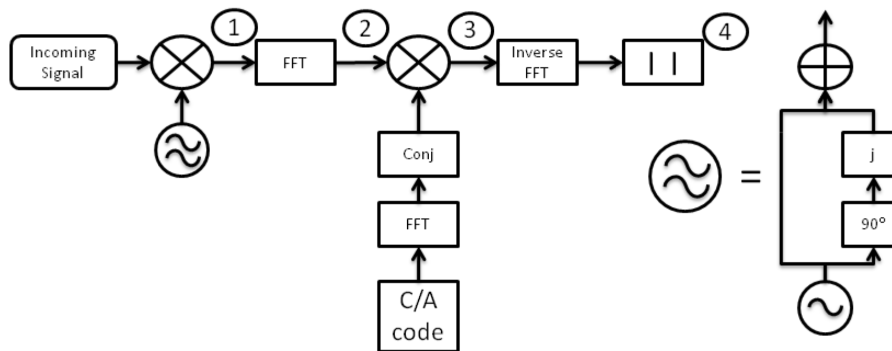


Figure 5 Coarse Acquisition Structure

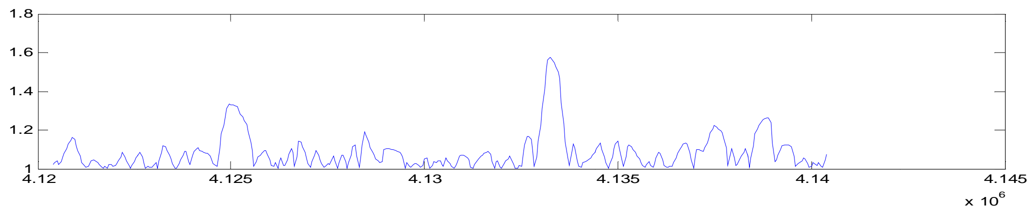


Figure 6 Correlation Strength Over Various Bins for Satellite 7

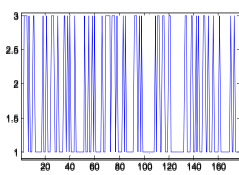


Figure 7 Step 1

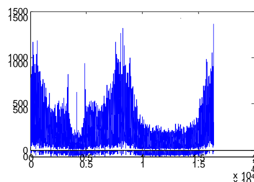


Figure 8 Step 2

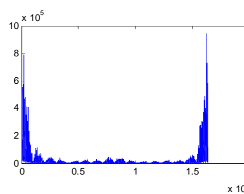


Figure 9 Step 3

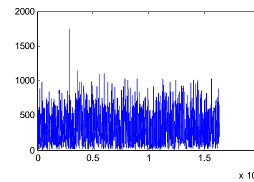


Figure 10 Step 4

Attempted Coarse Acquisition

Correlating multiple frequency bins at the same time was attempted as shown in Figure 11 by presenting a summation of all frequency bins to be searched; however the signal strength was degraded as more frequency bins were added. Had this worked, the frequency bins search would have decreased from 41 searches to 8 searches.

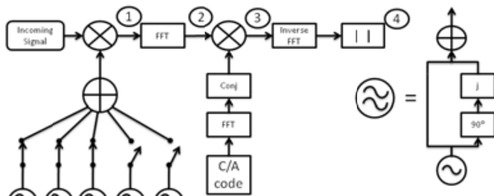


Figure 11 Attempted structure

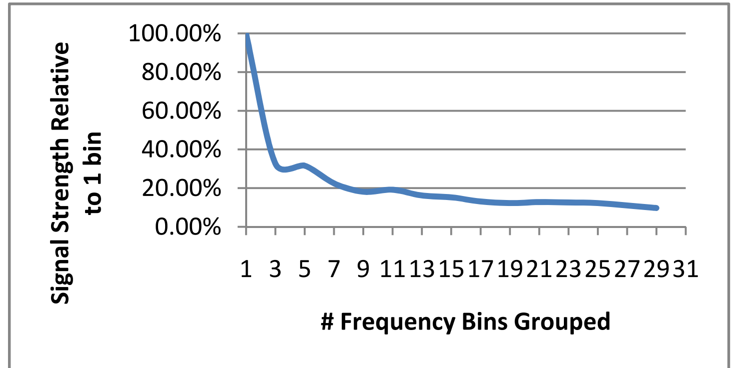


Figure 12 Signal Strength Vs a Number of Bins

Tracking

The tracking algorithm is responsible for keeping track of a satellites C/A code length and carrier frequency. A modified form of a phase lock loop is used for the tracking. It can be summarized by the following diagram.

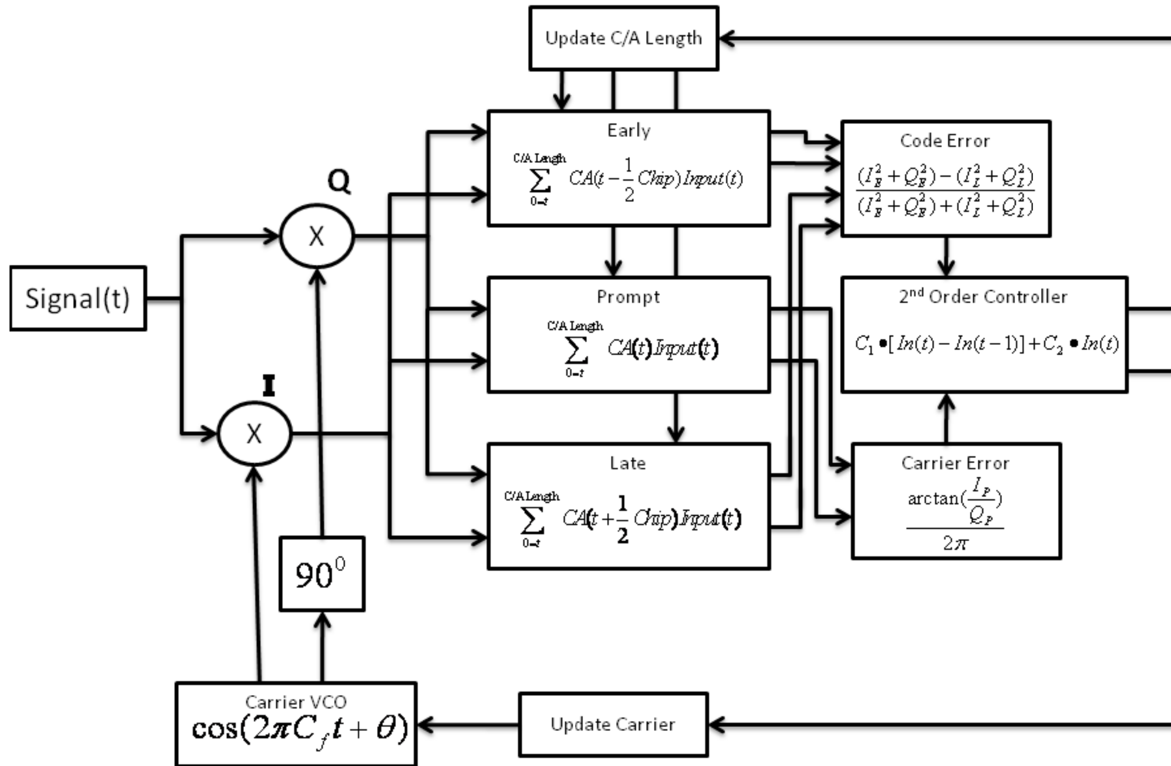


Figure 13 Tracking loop

Carrier VCO

One of the more expensive calculations for processors is the generation of sine and cosine values. For a speed optimization, a sine/cosine table look up is performed using a bitwise modulus operation. See Appendix B for the implementation.

Correlation

Early, prompt, and late correlation can be a computationally expensive task for computers. An initial approach would be to use three loops for early, prompt, and late correlations as shown in Figure 14. If 7 loops are used, two benefits happen. First, the overhead of two for loops is eliminated for chips 2 through 1,022. Secondly, the same statements are placed next to each other. Because of this, compilers can perform SIMD (single instruction multiple data) optimizations. Thus, many multiplications

can be completed in the same processor cycle. Also the values of chips 2-1,022 only need to be loaded into memory once instead of three times.

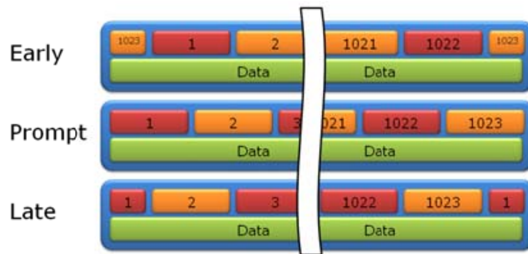


Figure 14 Basic tracking loop

Figure 15 Improved tracking loop

Decoding

The GPS navigation message is transmitted every 12.5 minutes. It is made up of 25 frames. Each frame is made up of 5 subframes and is received every 30 seconds. After the reception of a frame, the current time and satellite position will be available. There are 3 main steps in decoding this data.

1. Locating a bit change.
   Every bit contains 20 C/A correlations. To locate an exact bit change twenty correlations followed by another series of twenty correlations with an opposite sine is searched for. In the implementation, a shift register is used with a bit representing a single correlation. The hex code to determine a bit transition was 0x000FFFFF or 0xFFF00000.
2. Locating the start of a subframe.
   In order to successfully decode subframe data, the start of a subframe must be known. The following steps are taken to determine the start of a subframe. An inverted version of the incoming message should be tested for as the parity may be inverted from the BPSK transmission, Tsui and Borre emphasizes this also[1,2].
   I. Locate preamble.
      Every subframe begins with an eight bit marker known as the preamble. Once a preamble marker is found, further steps are needed to ensure that a false preamble was not located. [1,2]
   II. Preamble repeated in next subframe.
      The next subframe will contain a preamble.
   III. Check for a valid subframe ID, and time in current, and next subframe.
      Each subframe contains a subframe ID to identify itself, and the current time of week. Both of these values will have valid ranges.
   IV. Verify that the subframe's ID, and time increment in following subframe.
   V. Parity checks.
      The last six bits of every 30 bits transmitted are used to perform a parity check. An incorrect parity check can occur from incorrect correlations, or a misaligned preamble.[1,2]

3. Decoding subframes
   Decoding subframes is perhaps the most straitfowrad procedure. Three hundered bits are collected, and then passed to a subframe decoding routine provided by GPSTK. One technical note that should be mentioned is that the 30th bit of a word dictates if the next 24 bits need to be inverted or not. The last 6 bits of a word contain pairty checks that should be used to determine if a subframe's data is valid.

Positioning

Providing users with an updated position is the final step in a GPS receiver. This was implemented in the following steps.

I. Raw buffer position information is gathered from the tracking loops.
   Because the tracking loops are running in parallel they all must be paused while their pseudorange information is gathered. The tracking loops store the buffer position of each transmitted word. This results in a position update rate of 600ms.
II. Convert buffer positions into propagation delay times.
   This is done by subtracting the smallest buffer position from all buffer positions. This will initially result in one satellite having a pseudorange of zero. These values are then multiplied by the sampling frequency to obtain a transmitted time. The earliest arrival time a message can be received is approximately 68ms[2]. The earliest arrival time is then added to the transmitted times.
III. Convert propagation delays to distance.
   Multiplying by the speed of light converts the time traveled to distance traveled.

$$[.068s + (\text{data position- minimum data position})\ \text{samples} \bullet \frac{\text{seconds}}{sample}] \bullet \text{speed of light}$$

IV. Ephemeris information is collected from tracking satellites.
V. Pseudoranges and ephemeris information are then passed to the GPSTK and a RAIM algorithm is used to solve for position.
VI. A pushpin was placed on a Google earth interface using component object model as seen in Figure 16.

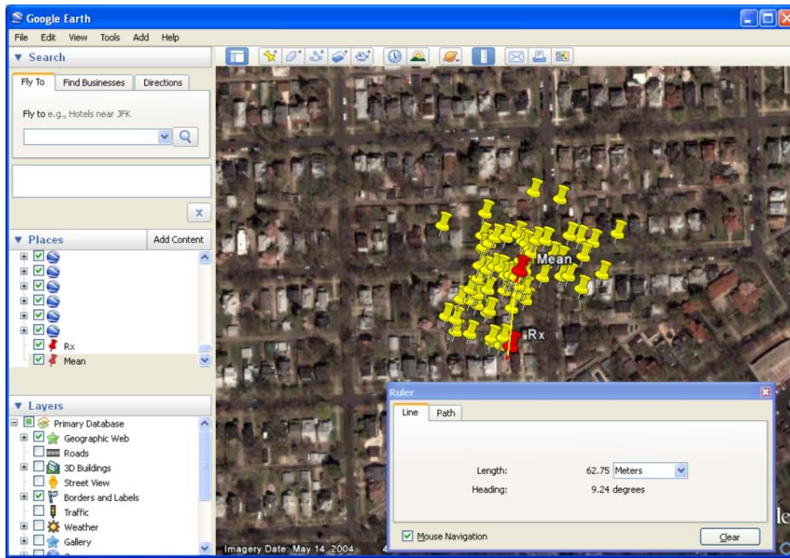


Figure 16 Google Earth Interface

Pseudorange Scaling Error

If the sampling frequency used in step III is not the same value as the hardware sampling frequency, a scaling error will occur. This error can be calculated as shown below.

$$\Delta t \bullet \text{speed of light} \bullet \left(1 - \frac{\text{actual Fs - offset}}{\text{actual Fs}}\right) = \text{Pseudorange error}$$

Δt is the reception time difference between different satellite signals. For GPS receivers Δt will range from 0 to 18ms. To help illustrate the effect of this Figure 17 shows the meters of error for a Δt of .009 ms and the offset is equal to 10Hz.

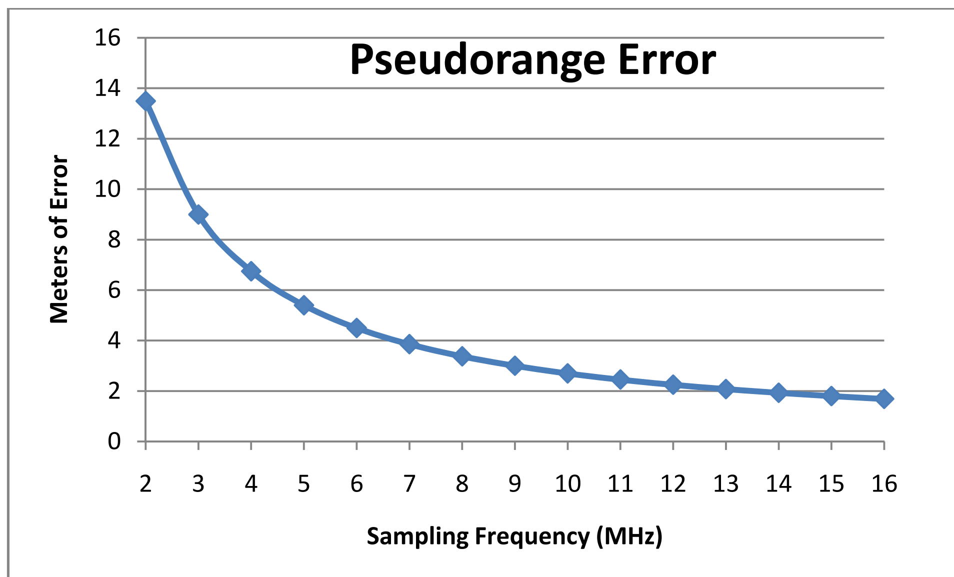


Figure 17 Scaling error

To account for this, a small application was written to calculate the sampling frequency from the number of samples collected in a period of 40 minutes. It was discovered that the sampling frequency was off by 200Hz, resulting in 43 meters of error when Δt is .009ms.

Comparison of Real-time Open Source Projects

	Bradley Project	GPS-SDR	OpenSource GPS
OS	Windows	Linux/Mac OSX	Linux
Code	C++	C++	C
Acquisition	1ms	1ms,10ms,15ms	1ms
Tracking update	1ms	Variable	20ms
Tracking order	2nd	3rd PLL, 1st DLL	2nd
GUI	Google Earth	In progress	Text
Sampling Frequency	16MHz	2.048MHz	16MHz

Future Recommendations

Simulated Signal Data

Having simulated signal data would help test tracking, and acquisition loops under different conditions. These would include different noise levels, multipath effects, and frequency shifts.

SIMD Optimizations

More optimizations using SIMD instructions exist. A majority of these optimizations will occur in the acquisition loop. The following library would provide a starting point for performing more of these optimizations.

http://sourceforge.net/projects/simd-cph/

The GPS-SDR makes use of many SIMD instructions throughout their code.

Acquisition Improvements

Currently acquisition is only performed upon 1ms of data at a time. A longer period for acquisition may be desired for the detection of weak signals. The GPS-SDR projects allows acquisition on 1ms, 10ms, and 15ms.

Locating the Carrier Frequency

Currently, a very large FFT is performed to obtain a carrier frequency before the PLL is used. This large FFT is impractical for embedded devices. The OSGPS project uses a pull in method to accomplish this. See section 4.4.4 of the following document.

http://home.earthlink.net/~cwkelley/OSGPS_chapter_4.pdf

Tracking Improvement 1

Multiple early, and late correlations could be used for the tracking loop. A curve fit could then be used for determining the proper location of the prompt code. See Fundamentals of Global Positioning System Receivers[2] pages 179-181.

Tracking Improvement 2

Early and late correlations are done ½ a chip away from the prompt correlation. If a variable width is used, more accurate calculations can be done, and nosier signals can be tracked. See Kaplan & Hegarty [3], page 175.

Tracking Improvement 3

Variable tracking update rates, and different order tracking loop may be desired.

Position Updates

Current position updates are calculated using only pseudoranges. Carrier phase information can be used to determine velocity changes. Also it may be desired to store receiver data in RINEX files. The GPSTK should provide an easy way of accomplishing this.

Conclusion

In this paper, a real-time software GPS receiver has been discussed and implemented as a senior capstone project. The current project is an extension upon a non-real-time software receiver design from last year's senior project. Several tracking optimizations were performed to allow real-time performance. Continuous position updates required restructuring of the previous project's code. SIMD instructions and the sampling frequency play a substantial role in obtaining real-time performance for software receivers. In the project, sampling, buffering, acquisition, tracking, and positioning stages of the software receiver were succesffully verified using both live and pre-recorded IF data.

References

[1] Borre, Kai, Dennis M. Akos, Nicolaj Bertelsen, Peter Rinder, and Soren Holdt Jensen. *A Software-Defined GPS and Galileo Receiver: A Single-Frequency Approach (Applied and Numerical Harmonic Analysis)* . Boston: Birkhäuser Boston, 2006.

[2] Tsui, James Bao. *Fundamentals of Global Positioning System Receivers A Software Approach (Wiley Series in Microwave and Optical Engineering).* New York: Wiley-Interscience, 2004.

[3] Kaplan, D., Elliott and Christopher J. Hegarty. Understanding Gps. City: Artech House Publishers, 2005.

Appendix A - Coarse Acquisition

```
clear all
clc
%Setup parameters
IF = 4.1304e6;
n = 16; %Number of samples per chip
Fs = 16367407;
bins = -10e3:500:10e3;
downsample = 8;
FileName = 'E:\public\dataBig.bin';
Milliseconds= 4;

Fs = Fs/downsample;
n = n/downsample;
len = n*Milliseconds*1023;
samplePerChip = round(len/1023);
%Setup time
t = (1:len)-1;
t=  t/Fs;

%Used to remove the spike
rmSpike = ones([1 len/Milliseconds]);
zs = samplePerChip*2+1;
rmSpike(1:zs) = zeros([1 zs]);
rmSpike = circshift(rmSpike,[0 -samplePerChip]);

counter = 1;
%Change ca1 to another frequency and see the results.
%IncomingSignal = ca1 .* cos((IF+000)*2*pi*t);
fid = fopen(FileName);
 %Read 3ms of data in.
for z= 1:5000 %Skip x milliseconds of data
    fread(fid,len*downsample,'int8')';
end
IncomingSignal = fread(fid,len,'int8',(downsample-1))';
fclose(fid);

satdata=zeros([32,length(bins)]);
for sat = 7%1:32%[4,5,12,29,30]%1:32
    tic
    ca = cacode(sat,n);
    ca = (ca-.5)*2;%Rescale to BPSK
    ca1 =[];
    for x=1:Milliseconds
     ca1 =[ca1 ca];
    end

    C/A1 = fft(ca1);
    C/A1 = conj(C/A1);

    for offset =  bins
         Step1I = IncomingSignal .* cos((IF+offset)*2*pi*t);
         Step1Q = IncomingSignal .* sin((IF+offset)*2*pi*t);
        %Generate the result in the frequency domain
        Step2 = Step1I + Step1Q*i;
        Step2 = fft(Step2);

        %Perform correlation
        Step3 = C/A1.*Step2;

        %Return the result to the time Domain
        Step4 = abs(ifft(Step3));

        %Store the results for later
        xval(counter)=offset+IF;
        Step4 = abs(Step4);

        [max1 pos] = max(Step4);
        pos = pos-1;
        %Remove the largest spike, and side lobes of it.
```

```
        Step4 = Step4(1:(len/Milliseconds));
        max2 = circshift(rmSpike,[0 pos]).*Step4;

        %Find the second largest spike
        max2 = max(max2);

        magnitude(counter)=max1/max2;
        counter = counter+1;
    end
    toc
    sat
    satdata(sat,:) = magnitude;
    counter = 1;
end
%Show the result
plot(xval,satdata(x,:));
```

Appendix B – Cosine / Sine lookup

```
//This needs to be a power of 2 and greater than 4
//Creation of the table is as follows
TableLength =64;
SineTable = new float[TableLength*5/4];

for ( int x=0;x<TableLength*5/4;x++){
      SineTable[x]= sin(x*2.0*M_PI/TableLength);
}
//cos(x) == sin(x+pi/2)
CosineTable = SineTable+(TableLength/4);



//Use of the table to remove a carrier frequency
double trig_coeff = 2*PI*Carrier;
double TimeStep= 1/ SamplingFrequency;
double trig_step = (TimeStep*trig_coeff)*TableLength/(2*PI);
//Put in range of 0 - 2pi
while(trig_step> TableLength) trig_step -= MySettings.CosinTableLength;
double trig_arg = CarrierRemainder*TableLength/(2*M_PI);
//Compute the in-phase and quadrature values
for(size_t c = 0; c < SampleLength_; c+=MySamplesToSkip)
{

      I[c] = CosineTable[((int)(trig_arg))&(TableLength-1)]*(Data[c]);
      Q[c] =   SineTable[((int)(trig_arg))&(TableLength-1)]*(Data[c]);
      //Increment the trigarg by the time step
      trig_arg +=trig_step*MySamplesToSkip ;
}
CarrierRemainder  = fmod((2*PI*Carrier*t+CarrierRemainder),(2*PI));
```

Appendix C - Optimized Tracking loop

```
//Correlate the first half chip for the early correlation
if(C/A_[1022]==1){
	for(size_t i = 0; i < HalfChipE1; i+=MySamplesToSkip)
	{
		I_E += I[sampleEarly];
		Q_E += Q[sampleEarly];
		sampleEarly+=MySamplesToSkip;
	}
}
else{
	for(size_t i = 0; i < HalfChipE1; i+=MySamplesToSkip)
	{
		I_E -= I[sampleEarly];
		Q_E -= Q[sampleEarly];
		sampleEarly+=MySamplesToSkip;
	}
}
//Perform the correlation values for early, late, prompt on chip 0
if(C/A_[0]==1){
	for(size_t k = 0; k < SamplesPerChip_[0]; k+=MySamplesToSkip)
	{	I_E += I[sampleEarly];
		Q_E += Q[sampleEarly];

		I_P += I[samplePrompt];
		Q_P += Q[samplePrompt];

		sampleEarly+=MySamplesToSkip;
		samplePrompt+=MySamplesToSkip;
	}
	//Correlate the first half chip of the late correlation
	for(size_t i = 0; i < HalfChipL1; i+=MySamplesToSkip)
	{
		I_L += I[sampleLate];
		Q_L += Q[sampleLate];
		sampleLate+=MySamplesToSkip;
	}
}
else{
	for(size_t k = 0; k < SamplesPerChip_[0]; k+=MySamplesToSkip)
	{
		I_P -= I[samplePrompt];
		Q_P -= Q[samplePrompt];

		I_E -= I[sampleEarly];
		Q_E -= Q[sampleEarly];

		sampleEarly+=MySamplesToSkip;
		samplePrompt+=MySamplesToSkip;
	}
	//Correlate the first half chip of the late correlation
	for(size_t i = 0; i < HalfChipL1; i+=MySamplesToSkip)
	{
		I_L -= I[sampleLate];
		Q_L -= Q[sampleLate];
		sampleLate+=MySamplesToSkip;
```

```
        }
}
//Correlate the middle chips
for(size_t j = 1; j < 1022; ++j)
{
      if(C/A_[j]==1){
            for(size_t k = 0; k < SamplesPerChip_[j]; k+=MySamplesToSkip)
            {
                  I_L += I[sampleLate];
                  Q_L += Q[sampleLate];

                  I_E += I[sampleEarly];
                  Q_E += Q[sampleEarly];

                  I_P += I[samplePrompt];
                  Q_P += Q[samplePrompt];

                  sampleEarly +=MySamplesToSkip;
                  sampleLate  +=MySamplesToSkip;
                  samplePrompt+=MySamplesToSkip;
            }
      }
      else{
            for(size_t k = 0; k < SamplesPerChip_[j]; k+=MySamplesToSkip)
            {
                  I_L -= I[sampleLate];
                  Q_L -= Q[sampleLate];

                  I_E -= I[sampleEarly];
                  Q_E -= Q[sampleEarly];

                  I_P -= I[samplePrompt];
                  Q_P -= Q[samplePrompt];

                  sampleEarly+=MySamplesToSkip;
                  samplePrompt+=MySamplesToSkip;
                  sampleLate+=MySamplesToSkip;
            }
      }
}
//Add the correlation values for the middle chips
if(C/A_[1022]==1){
      for(size_t k = 0; k < SamplesPerChip_[1022]; k+=MySamplesToSkip)
      {
            I_P += I[samplePrompt];
            Q_P += Q[samplePrompt];
            I_L += I[sampleLate];
            Q_L += Q[sampleLate];

            sampleLate+=MySamplesToSkip;
            samplePrompt+=MySamplesToSkip;
      }
      //Correlate the last half-chip for early
      for(size_t i = 0; i < HalfChipE2; i+=MySamplesToSkip)
      {
            I_E += I[sampleEarly];
            Q_E += Q[sampleEarly];
```

```
                sampleEarly+=MySamplesToSkip;
        }
}
else{
        for(size_t k = 0; k < SamplesPerChip_[1022]; k+=MySamplesToSkip)
        {
                I_P -= I[samplePrompt];
                Q_P -= Q[samplePrompt];

                I_L -= I[sampleLate];
                Q_L -= Q[sampleLate];
                sampleLate+=MySamplesToSkip;
                samplePrompt+=MySamplesToSkip;
        }
        //Correlate the last half-chip for early
        for(size_t i = 0; i < HalfChipE2; i+=MySamplesToSkip)
        {
                I_E -= I[sampleEarly];
                Q_E -= Q[sampleEarly];
                sampleEarly+=MySamplesToSkip;
        }
}
//Correlate the last half-chip for late
if(C/A_[0]==1){
        for(size_t i = 0; i < HalfChipL2; i+=MySamplesToSkip)
        {
                I_L += I[sampleLate];
                Q_L += Q[sampleLate];
                sampleLate+=MySamplesToSkip;
        }
}
else{
        for(size_t i = 0; i < HalfChipL2; i+=MySamplesToSkip)
        {
                I_L -= I[sampleLate];
                Q_L -= Q[sampleLate];
                sampleLate+=MySamplesToSkip;
        }
}
```