2016

As submission to

Mr. Kelly Shepherd, Instructor

Mr. Marc Anderson

English and Communications Department

Rudimentary GPS

Angelo M Sanches, student

CMPE2960

Computer Engineering Technology

December 2, 2016

1840 28 ave SW

Edmonton, AB T5X 1A5

December 2, 2016

Mr. Kelly Shepherd

Instructor, English and Communications Department

NAIT

11762 -106 St NW

Edmonton, AB T5G 2R1

Dear Mr. Shepherd,

I am submitting the rudimentary GPS report, at department request, for you to review. This reports purpose is to fill the requirement of the project for the CMPE2960: Capstone.

This report outlines the building of a rudimentary GPS done by our group. It goes over the required concepts, materials, building, and programming of the device. It also outlines the devices usefulness in business and other applications.

Please accept my thanks, as I extend it to you and the other NAIT instructors who have helped to shape my future. You and your colleagues dedication has helped to shape me and push me towards success, so again thank you.

Sincerely,

Angelo Sanches

CNT Student

Table of Contents

[1 Introduction 1](#_Toc468946119)

[2 GPS Technology Overview 1](#_Toc468946120)

[3 An overview of the project 2](#_Toc468946121)

[3.1 The Requirements of the Device 2](#_Toc468946122)

[3.2 Core Concepts 3](#_Toc468946123)

[3.2.1 NMEA Sentences 2.0 3](#_Toc468946124)

[3.2.2 Mathematical Geometry and the Haversine Formula 5](#_Toc468946125)

[3.3 Hardware 6](#_Toc468946126)

[3.3.1 CNT9S12XDP512 Micro Kit 6](#_Toc468946127)

[3.3.2 Adafruit Ultimate GPS Breakout 8](#_Toc468946128)

[3.3.3 Adafruit Fona GSM Breakout 9](#_Toc468946129)

[4 Creating the device 10](#_Toc468946130)

[4.1 Assembling the Device 10](#_Toc468946131)

[4.2 Configuring and programing the device 11](#_Toc468946132)

[4.2.1 Programming the Micros’ Tx and Rx Lines 12](#_Toc468946133)

[4.2.2 Sending GPS Commands and GPS Configuring 13](#_Toc468946134)

[4.2.3 Sending Cell Commands and Cell Configuring 14](#_Toc468946135)

[4.2.4 Programming the Data Parsers 16](#_Toc468946136)

[4.2.5 Programming Functionality 19](#_Toc468946137)

[5 Conclusion 20](#_Toc468946138)

**Table of Figures**

[Figure 1 : GPS NMEA Sentences 2.0 3](#_Toc468940723)

[Figure 2 : GPGGA Data Example and Info 4](#_Toc468940724)

[Figure 3: Bearing Calculation 5](#_Toc468940725)

[Figure 4: Haversine Formula 5](#_Toc468940726)

[Figure 5: Hardware Overview Schematic 7](#_Toc468940727)

[Figure 6: Adafruit Ultimate GPS Schematic 8](#_Toc468940728)

[Figure 7: Adafruit Fona GMS Schematic 9](#_Toc468940729)

[Figure 8: SCIDRH Bit Set Equations 11](#_Toc468940730)

[Figure 9: Setting the Baud in Code for SIC0 11](#_Toc468940731)

[Figure 10: Send and Receive Data 12](#_Toc468940732)

[Figure 11: Send Message Code 12](#_Toc468940733)

[Figure 12: Cell Get Http Data (Getter) 14](#_Toc468940734)

[Figure 13: Base Read Section for Parser 15](#_Toc468940735)

[Figure 14: GPS Data Parser Code 16](#_Toc468940736)

[Figure 15: GPS Data Parser Code 16](#_Toc468940737)

[Figure 16: HTTP Google Parse Code 17](#_Toc468940738)

[Figure 17: Distance Calculation Code 18](#_Toc468940739)

**Abstract**

Global Positioning System or GPS provide us with location data. They listen to data from satellites and use the time stamped locations to triangulate themselves. This data is generally encoded to the National Marine Electronics Association or NMEA standard strings. GPGGA sentences or Global Positioning System Fix Data. The Data is provides a satellite time stamp, altitude, latitude and longitude coordinates.

GPS consuming devices can then take this data and compare them with other known locations to calculate distance, bearing, speed, location, and even direction. This can be done by simply applying simple trigonometry and geometry math such as the Haversine formula.

A rudimentary GPS for demonstration could be made from CNT9S12XDP512, Adafruit Fona GSM Breakout, and Adafruit Ultimate GPS Breakout. The Device could easily be able to calculate location, direction, and distance.

GPS devices have been proven useful in the military, business, individuals, construction, governments, and even recreation. This report outlines how one could make their own GPS and even adapt it for more personal use.

# Introduction

A common problem faced by many people is the problem of finding direction when traveling to a new area. This often leads to being late or never even finding the location in time. This is common a problem for every day regular society; however, there is a solution that would solve this.

The National Marine Electrical Association or NMEAs’ Global Positioning System or GPS can help to find both fixed and moving positions. These positions can be accurate up to 2.5m or better. Positions can also be updated on a real time basis. These positions can be used to calculate with great accuracy a desired direction.

This documentation is to outline the development of a rudimentary version of a GPS device. Enclosed is the procedure of the development process, theory, challenges, and overview of such said project.

Data presented here will follow this outline: A quick overview of the following on GPS and data; an overview of the project itself; the required materials and concepts; the devices operation; the limitations of the device and a conclusion.

# GPS Technology Overview

It is important to know how and why this technology works before beginning. So we will quickly outline the process here.

A GPS is a device that listens to satellites to obtain a position. There are many different types of GPSs. When extrapolating where direction positional data is required. Global Positioning System Fix Data designated GPGGA is the GPS type that gives us world position. Using this data it is possible to know the exact location of the device. (Eric Gakstatter, February)

This data is obtained from satellites that are in geo synchronous orbit. These satellites are always orbiting the earth and pass over the same location every 12 hours. It can calculate its current location with this information and an internal atomic clock on board. This data is then broadcasted to the air.

Listening to this data and making use of it is the actual task of the GPS. The GPS uses the location and the time the signal takes to get to the board, to create a radius around each satellite. We can then intersect these to triangulate the location.

Taking this further we can get a direction and distance from two points obtained this way. The Haversine formula and geometry will give us the distance and barring.

# An overview of the project

The Goal of the project was to make a rudimentary GPS that would be able to give a very basic direction. The calculated direction will be direct to the target, with little account for obstacles. The devices output would show the devices location, the targets location, the distance to target, and the bearing to the target.

## The Requirements of the Device

The project uses some key concepts such as NMEA sentence notation and Haversine formula with mathematical geometry. In addition to this, it uses the CNT9S12XDP512 Micro Kit, Adafruit Ultimate GPS Breakout, and Adafruit Fona GSM Breakout (SMA) for hardware. Components were programed in C.

## Core Concepts

Outlined below are some core concepts.

### NMEA Sentences 2.0

The National Marine Electrical Association or NMEA has implemented the standard transition strings or sentences for serial communication. The NMEA sentences were needed for accuracy when transmitting data across machines using a serial line in an accurate and standardized fashion. This has been widely adopted as the standard for GPS communications.

All NMEA sentences have a basic pattern. They all start with a dollar sign and are followed by the data type. All the data fields are comma separated and ends on an asterisk separated check sum. (Eric Gakstatter, February) Figure 1 shows a quick overview of sentence types.

**Table 1 : GPS NMEA Sentences 2.0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Garmin | Magellan | Lowrance | SiRF | Notes: |
| GPAPB | N | Y | Y | N | Auto Pilot B |
| GPBOD | Y | N | N | N | bearing, origin to destination - earlier G-12's do not transmit this |
| GPGGA | Y | Y | Y | Y | fix data |
| GPGLL | Y | Y | Y | Y | Lat/Lon data - earlier G-12's do not transmit this |
| GPGSA | Y | Y | Y | Y | overall satellite reception data, missing on some Garmin models |
| GPGSV | Y | Y | Y | Y | detailed satellite data, missing on some Garmin models |
| GPRMB | Y | Y | Y | N | minimum recommended data when following a route |
| GPRMC | Y | Y | Y | Y | minimum recommended data |
| GPRTE | Y | U | U | N | Route data, only when there is an active route. (this is sometimes bidirectional) |
| GPWPL | Y | Y | U | N | waypoint data, only when there is an active route (this is sometimes bidirectional) |

**Figure 1 : GPS NMEA Sentences 2.0**

(Eric Gakstatter, February)

The sentence structure used by the project is GPGGA. GPGGA is current fixed data or current location on the planet. In GPGGA, the data fields are as follows in chronological order: time stamp, latitude, North or South, longitude, East or West, fix quality, number of satellites, horizontal dilution, altitude, altitudes units, height of geoid, heights units, and the check sum. Figure 2 expresses a GPGGA sentence in more a detailed example. (Eric Gakstatter, February)

**GPGGA Data Example**

|  |
| --- |
| $GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,\*47  Where:  GGA Global Positioning System Fix Data  123519 Fix taken at 12:35:19 UTC  4807.038,N Latitude 48 deg 07.038' N  01131.000,E Longitude 11 deg 31.000' E  1 Fix quality: 0 = invalid  1 = GPS fix (SPS)  2 = DGPS fix  3 = PPS fix  4 = Real Time Kinematic  5 = Float RTK  6 = estimated (dead reckoning) (2.3 feature)  7 = Manual input mode  8 = Simulation mode  08 Number of satellites being tracked  0.9 Horizontal dilution of position  545.4,M Altitude, Meters, above mean sea level  46.9,M Height of geoid (mean sea level) above WGS84  ellipsoid  (empty field) time in seconds since last DGPS update  (empty field) DGPS station ID number  \*47 the checksum data, always begins with \* |

Figure 2 : GPGGA Data Example and Info

(Eric Gakstatter, February)

### Mathematical Geometry and the Haversine Formula

This is the math to take GPS coordinates and turn them into directional information. There are great mathematical tools for this such as basic trigonometry and the Haversine formula.

To calculate the bearing there is simple trigonometry. Basic calculations must derive the degrees clock wise from north. Here is the formula. (Chris Veness, 2002-2016)

Equation 1: Bearing Calculation

|  |
| --- |
| Wheree and |

Figure 3: Bearing Calculation

(Chris Veness, 2002-2016)

The Haversine formula is used to calculate the spherical distance between two points. In this case that is the shortest distance over the earth’s surface. This is demonstrated as the following formula. (Chris Veness, 2002-2016)

Equation 2: Haversine Formula

|  |
| --- |
| Where, Δλ =,  , and |

Figure 4: Haversine Formula

(Chris Veness, 2002-2016)

## Hardware

Outlined below are some hardware components. The devices are connected by the MC9S12XDP512val serial ports. The GPS Breakout uses serial ports 1 and the Cell Breakout uses serial ports 0.

### CNT9S12XDP512 Micro Kit

The CNT9S12XDP512 Micro Kit form the NAIT CNT program is the base platform for the Device. The Device has built on to it many functional pieces.

The built in LED screen allows for four lines at twenty characters per line. The GPS makes use of this display to show data to the user.

It main benefit; however, is the MC9S12XDP512val Micro Controller. This device has around 64k of ram and 8 kHz clock speed in its current configuration. It also has 6 SIC ports to use and is C programmable. This means it can be the main controller for the entire device. Figure 5 shows the wiring and ports.

**Hardware Overview Schematic**

|  |
| --- |
|  |

Figure 5: Hardware Overview Schematic

Composite image from:

(Adafruit Industries, 2014); (Adafruit Industries, 2014); (NAIT CNT, 2014)

### Adafruit Ultimate GPS Breakout

This device is responsible for listening to the satellites and triangulating location. Most of the heavy lifting for triangulating is actually done by this device. Its sensitivity is quite high at -165 dBm fall off and a 2.5 m accuracy and transmits in standard NMEA sentences at 9600 bauds/second. Figure 6 show the device.

**Adafruit Ultimate GPS Schematic**

|  |
| --- |
|  |

Figure 6: Adafruit Ultimate GPS Schematic

(Adafruit Industries, 2014)

### Adafruit Fona GSM Breakout

This device is responsible for getting data from the cell towers. This device uses a sim card to tap into the cell network. It enables the device to talk to the web where we can get the location of real world objects.

**Adafruit Fona GMS Schematic**

|  |
| --- |
|  |

Figure 7: Adafruit Fona GMS Schematic

(Adafruit Industries, 2014)

# Creating the device

Assuming you have the base board CNT9S12XDP512 preassembled the circuit is easy to assemble. The programming is a little more difficult to pull off. The following is an outline of the process.

## Assembling the Device

The device is simple to set up. The process involves hooking up the correct devices outputs and inputs. For a diagram please re-reference figure 5 Hardware Overview Schematic.

The GPS device has nine outputs and inputs in total.

* Pin 1 is PPS. This pin is for syncing with the device. It is not needed and will not be used.
* Pin 2 is the Vo. This pin powers and sets the logic on the device. If 5 volts is placed here, it is hooked to a 5 volt rail.
* Pin 3 is the ground and is connected to digital ground.
* Pin 4 is the Rx and should be connected to the Tx pin of the micro. It is hooked up to Tx 1 on pin 92 on the micro.
* Pin 5 is the Tx and should be connected to the Rx pin of the micro. It is hooked up to Rx 1 on pin 91 on the micro.
* Pin 6 is the Fix. It is not needed and is therefore not hooked up.
* Pin 7 is the V backup and is for a backup battery. The device has one on its underside and is not needed. It is not hooked up.
* Pin 8 is the Enable and is hooked to Pin 59 on the Micro.
* Pin 9 is a Fixed 3.3 volts and is not used.

(Lady Ada, 2014)

The Cell device has twelve pins.

* Pin 1 is the VCC. It is not needed as the battery powers the device on the side.
* Pin 2 is the Network Status. It is not needed; however, is hooked up to pin 61 on the Micro.
* Pin 3 is the Rx and should be connected to a Tx pin of the micro. It is hooked up to Tx 0 on pin 90 on the micro.
* Pin 4 is the Tx and should be connected to a Rx pin of the micro. It is hooked up to Rx 0 on pin 89 on the micro.
* Pin 5 is the USB power and is not used.
* Pin 6 is the Key and turns on the device with a 2 second pulse. It is hooked onto pin 60 of the micro.
* Pin 7 is the Power status pin and is connected to pin 61 on the micro.
* Pin 8 is the reset pin and is connected to pin 62 on the micro.
* Pin 9 is the speakers’ negative poll and is not used.
* Pin 10 is the speakers’ positive poll and is not used.
* Pin 11 is the ground and is connected to the digital ground.
* Pin 12 is the Vo and the logic level set pin. It is hooked to a 5 volt rail.

(Adafruit Industries, 2015)

## Configuring and programing the device

The devices must be configured and programmed prior to use. This is done by programming the micro board to configure its ports and then send commands to the devices.

### Programming the Micros’ Tx and Rx Lines

The micro must talk to the both devices at a baud rate of 9600 bauds/second. The way the micro is configured for this is by setting the respective SICDRL register and SCIDRH register, to the correct bytes. The SICDRL register can receive no bytes as the defaults are good. The SCIDRH register needs some configuration. With a quick calculation we can find out the bits we need to set. (Freescale Semiconductor, 2008) Figure 8 shows the equation and figure 9 shows a code block setting the baud rate.

Equation 3: SCIDRH BIT SET EQUATIONS

|  |
| --- |
| Where C = the micros clock and T = the target baud rate |

Figure 8: SCIDRH Bit Set Equations

(Freescale Semiconductor, 2008)

**Setting the Baud in Code for SIC0**

|  |
| --- |
| void SIC0InIt9600(void)  {  // SCI0DRH B7 = 0 = no IF; SCI0DRH B6,5 = 00 = 3/16 NPW  // SCI0DRH, SCI0DRL {B4-0},{B7-1} = 0034 = 52 = 8Mhz /(9800hz \* 16)  SCI0DRL = 0x00;  SCI0DRH = 0x34;  SCI0CR1 = 0;  SCI0CR2 = 0x0C;  } |

Figure 9: Setting the Baud in Code for SIC0

In sending and receiving messages there needs to be a way to send and receive bytes. In the case of the devices a message is represented by characters in a string. The style used here is blocking calls or synchronous calls that poll the Rx and Tx that must be completed before moving on. To send while polling wait for the SCISR1 busy bit (0x80) to be cleared and then write to SCIDRL. To receive while polling wait for the SCISR1 waiting bit (0x20) to be cleared and then read from SCIDRL. Figure 10 shows demonstration code.

**Send and Receive Data**

|  |  |
| --- | --- |
| Void send(char Data)  {  while((SCI1SR1 & 0x80) == 0);  SCI1DRL = Data;  } | Void Receive(char\* Data)  {  while((SCI1SR1 & 0x20) == 0);  \*Data = SCI1DRL;  } |

Figure 10: Send and Receive Data

### Sending GPS Commands and GPS Configuring

The GPS still needs commands and some small configuration. Commands are sent as NMEA sentences. This means $ to start and check sums behind an \* at the end. The very end is signed with a <CR><LR> to signify the end. Figure 11 is code on sending the data in it we will both package in NMEA string and send it.

**Send Message Code**

|  |  |
| --- | --- |
| void StrCheckSum(char\* Data, char\* CheckSum)  {  int i = 0;  char Check = (char)0;    while(Data[i] != '\0')  {  Check ^= Data[i];  i++;  }    CheckSum[0] = (char)((byte)Check / 16 + 48);  CheckSum[1] = (char)((byte)Check % 16 + 48);    }  void GPSSendCommand(char\* String)  {  byte i =0;  char\* message = NULL;  message = MakeNMEAMessage(String);  while(message[i] != 0x00 && message[i] != '\0')  {  while((SCI1SR1 & 0x80) == 0);  SCI1DRL = message[i];  i++;    }  free(message);  } | char\* MakeNMEAMessage(char\* String)  {  int i = 0; // \r\n  char Check[2];  char\* StringOut = NULL; //(char\*)malloc(i + 1);    StrCheckSum(String, Check);    while(String[i] != '\0')  {  i++;  }      StringOut = (char\*)malloc(i + 8);  StringOut[0] = '$';  i = 0;  do {  StringOut[i + 1] = String[i];  i++;  } while(String[i] != '\0');  i++;  StringOut[i] = '\*';  i++;  StringOut[i] = Check[0];  i++;  StringOut[i] = Check[1];  i++;  StringOut[i] = '\r';  i++;  StringOut[i] = '\n';  i++;  StringOut[i] = '\0';  return StringOut;  } |

Figure 11: Send Message Code

For configuring the GPS we can leverage the code above and simply place a string in it during calls. The GPS will return a reply for some commands, but most do not return anything.

The GPS options that need configuring are as follows.

* Turn of the antenna status update to prevent messages space waste - PGCMD,33,0
* Set the data output frequency to 10 Hz - PMTK220,100
* Set the GPS to only put out GPGGA data - PMTK314,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

(GlobalTop Tech Inc., 2012)

For a complete list of commands visit <https://cdn-shop.adafruit.com/datasheets/PMTK_A11.pdf>.

### Sending Cell Commands and Cell Configuring

The cell model doesn’t need as much in the way of startup configuration it instead needs more of a sessional configuration. For this reason we will only configure it only where it gets used. This only happens when we use it to call to the internet. However, here there needs to be a bit of string building to make http strings.

A list of the commands we will use.

* Configure bearer profile - AT+SAPBR=3,1,”Contype”,”GPRS”\r
* To open a GPRS context. - AT+SAPBR=1,1\r
* Start http session - AT+HTTPINIT\r
* Configure cell for http - AT+HTTPPARA="CID",1\r
* Set http getter variables - AT+HTTPPARA="URL", “somewhere”
* Send data - AT+HTTPACTION=0\r
* Get response - AT+HTTPREAD\r
* Close http session - AT+HTTPTERM\r

(Shanghai SIMCom Wireless Solutions Ltd., 2013)

For a complete list of commands visit <https://cdn-shop.adafruit.com/datasheets/sim800_series_ip_application_note_v1.00.pdf>.

Figure 12 shows these commands in action. Note send command is simply just a blocking SIC Send command.

**Cell Get Http Data (Getter)**

|  |
| --- |
| void CELLGetHTTP(char\* URL, int URLsize, char\*\* Params, int\* ParamSize, char\*\* Value, int\* ValueSize, char\* Response, int ResponseSize, byte ParamsNum)  {  //Huge string building exert  char\* Command = "AT+HTTPPARA=\"URL\",\"";  char\* FullURL = NULL;  int i;  //capture the sizes to alot mem  int ParamsSIZES = URLsize;  for (i = 0; i < ParamsNum; i ++)  {  ParamsSIZES += ParamSize[i];  ParamsSIZES += ValueSize[i];  ParamsSIZES++; //for &  }  //alot mem  FullURL = (char\*)malloc(ParamsSIZES + 3 + 20); // for (& + \0 + \r ) + 18 for Command  (void)strcat(FullURL, Command);// set command ith open dting for use  (void)strcat(FullURL, URL); //add url  (void)strcat(FullURL, "&"); //add param sep  //add all params & values as stuff  for (i = 0; i < ParamsNum; i ++) //\r  {  (void)strcat(FullURL, Params[i]);  (void)strcat(FullURL, "=");  (void)strcat(FullURL, Value[i]);  }  (void)strcat(FullURL, "\"" ); //close string  (void)strcat(FullURL, "\r" ); //carage return for our thing  (void)strcat(FullURL, "\0" ); //end of string for us  CELLSendCommand( "AT+HTTPINIT\r" ); //start sesh  CELLSendCommand( "AT+HTTPPARA=\"CID\",1\r" );//confing cell for http  CELLSendCommand(FullURL); // set url param  CELLSendCommand( "AT+HTTPACTION=0\r" ); //Send it  CELLSendCommand( "AT+HTTPREAD\r" ); //GetResponse  SCI1ReadString(Response, &ResponseSize); //listen to response  CELLSendCommand( "AT+HTTPTERM\r" ); // clean up cell  free(FullURL); // clean up func  } |

Figure 12: Cell Get Http Data (Getter)

### Programming the Data Parsers

The Complex Data coming over the wire just comes out over a string. This creates the challenge of parsing it to be meaningful. The MC9S12DXP512val will need to be programmed to do this. This Problem is simple yet complex and requires a break down to solve.

#### Section Reader.

To solve it a simple base case of reading a section of data was explored. In it the micro programmatically steps though from a start position it till it finds the character to separate on. The steps are counted and stored. After this it allots memory and copies over the characters in the position from the last point through the count. Then the count is added to the start position. Since the start position is kept as a pointer the changes are seen at the source and the string is returned. This method would be repeated elsewhere and adapted often. Figure 13 shows the base Parser.

**Base Read Section for Parser**

|  |
| --- |
| char\* ReadSection(char\* Data, byte\* OutStart, char Spliter)  {  int i = 0;  char\* StringOut = NULL;  while (Data[\*OutStart + i]!= 0x00 && Data[\*OutStart + i]!= '\0' && Data[\*OutStart + i] != Spliter)  {  i++;  }    StringOut = (char\*)malloc(i + 1); // thanks to herb for showing me aloting data  i = 0;  while (Data[\*OutStart + i]!= 0x00 && Data[\*OutStart + i]!= '\0' && Data[\*OutStart + i] != Spliter)  {  StringOut[i] = Data[\*OutStart + i];  i++;  }  StringOut[i] = '\0'; //cap it as always  \*OutStart += i + 1;  return StringOut;  } |

Figure 13: Base Read Section for Parser

#### GPS Parser

Once a stream has been broken into smaller parts using the Read section function it is quite manageable. After that they need to be taken and passed through data converters using a standard set of C libraries. Figure 14 shows the code.

Figure 14: GPS Data Parser Code

|  |  |
| --- | --- |
| int ParseGPGGA(char\* Data, GPSDataType\* DataOut)  {  // check for GPGGAPrior  byte i;  int bytedata =0;  char\* ParsedData;  char Check = (char)0;    i = 1;  while(Data[i] != '\*')  {  Check ^= Data[i];  i++;  }    i = 8;  // check hhmmss.ss  // check time stamp;  ParsedData = ReadSection(Data , &i, ',');  (\*DataOut).TimeStamp = atof(ParsedData);  free(ParsedData);  // thanks to herb for showing me free data  // check for latitude  ParsedData = ReadSection(Data, &i, ',');  (\*DataOut).latitude = atof(ParsedData);  free(ParsedData);  //check for N or S  ParsedData = ReadSection(Data, &i, ',');  if(\*ParsedData == 'S') {  (\*DataOut).latitude \*= -1;  }  free(ParsedData);  //check for longitude  ParsedData = ReadSection(Data, &i, ',');  (\*DataOut).longitude = atof(ParsedData);  free(ParsedData);  //check for W or E  ParsedData = ReadSection(Data, &i, ',');  if(\*ParsedData == 'W') {  (\*DataOut).longitude \*= -1;  }  free(ParsedData);  // check to see the quality of indicator  ParsedData = ReadSection(Data, &i, ',');  bytedata = atoi(ParsedData);  free(ParsedData);  if(bytedata == 1)  (\*DataOut).Quality = Uncorrelated;  else if(bytedata ==2)  (\*DataOut).Quality= DifferentiallyCorrected;  else if(bytedata ==3)  (\*DataOut).Quality = RTKFix;  else if(bytedata ==4)  (\*DataOut).Quality = RTKFloat; | else  (\*DataOut).Quality = Unknown;    //denotes number of satellites used in the coordinate  ParsedData = ReadSection(Data, &i, ',');  free(ParsedData);    // check for HDOP  ParsedData = ReadSection(Data, &i, ',');  (\*DataOut).HDP = atof(ParsedData);  free(ParsedData);    // check for altitude  ParsedData = ReadSection(Data, &i, ',');  (\*DataOut).Alt = atof(ParsedData);  free(ParsedData);    //M  ParsedData = ReadSection(Data, &i, ',');  if(\*ParsedData == 'F') // convert to Meter  {  (\*DataOut).Alt \*= 3.28084;  }  free(ParsedData);    //check for geoidal separation  ParsedData = ReadSection(Data, &i, ',');  (\*DataOut).GeoidalS = atof(ParsedData);  free(ParsedData);  //M  ParsedData = ReadSection(Data, &i, ',');  if(\*ParsedData == 'F') // convert to Meter  {  (\*DataOut).GeoidalS \*= 3.28084;  }  free(ParsedData);  // age of the correction  ParsedData = ReadSection(Data, &i, ',');  (\*DataOut).ADGD = atof(ParsedData);  free(ParsedData);  //station ID  ParsedData = ReadSection(Data, &i, '\*');  (\*DataOut).StationID = atoi(ParsedData);  free(ParsedData);    ParsedData = ReadSection(Data, &i, '\*');  i = ParseHex(ParsedData);  free(ParsedData);    if(i == (int)Check)  return -2; //bad data    return 0;  } |

Figure 15: GPS Data Parser Code

#### HTTP Parser for Google

The parser for the data coming out of the cell is highly subject to the sites; therefore, all data off of parsers would have to be per site. As such, a parser for googles json API was made.

Since it is in json notation we can just throw out the first few useless fields and extract the first set of latitude and longitude. (It would have taken too long to be more precise with this given the time frame.) After we convert the data and store it.

**HTTP Google Parse Code**

|  |
| --- |
| void ParseJson(char \*jsonData, GPSDataType\* DataOutr) {  int i =0;  char\* ParseData = NULL;  ParseData = ReadSectionLong(jsonData, &i, '{');  free(ParseData);  ParseData = ReadSectionLong(jsonData, &i,'{');  free(ParseData);  ParseData = ReadSectionLong(jsonData, &i, '{');  free(ParseData);  ParseData = ReadSectionLong(jsonData, &i, '{');  free(ParseData);  ParseData = ReadSectionLong(jsonData, &i, ':');  free(ParseData);  ParseData = ReadSectionLong(jsonData, &i, ',');  (\*DataOutr).latitude = atof(ParseData);  free(ParseData);  ParseData = ReadSectionLong(jsonData, &i, ':');  free(ParseData);  ParseData = ReadSectionLong(jsonData, &i, '}');  (\*DataOutr).longitude = atof(ParseData);  free(ParseData);    } |

Figure 16: HTTP Google Parse Code

### Programming Functionality

Tying these sub systems together is simple. First step is calculating the direction. After that a UI needs to be built with a main overview system.

#### Calculating Direction

By using the above outlined concept of trigonometry and Haversine Formula it is trivial to calculate. Putting the formulas to code is the only problem here. The formulas are very easily convertible when working with floats, as the data has already been stored. Figure 17 shows the code.

Distance Calculation Code

|  |
| --- |
| #define M\_PI acos(-1.0)  //Distance Barring  void FindDirection(GPSDirectionType \*Out, GPSDataType\* Start, GPSDataType\* End)  {  float Lat1;  float Lat2;  float DLat;  float DLon;  float a;  float b;  Lat1 = (\*Start).latitude /100 \* M\_PI /180;  Lat2 = (\*End).latitude /100 \* M\_PI /180;  DLat = ((\*End).latitude /100 - (\*Start).latitude /100) \* M\_PI /180;  DLon = ((\*End).longitude /100 - (\*Start).longitude /100) \* M\_PI /180;    a = sin(DLat / 2) \* sin(DLat) + cos(Lat1) \* cos(Lat2) \* sin(DLon /2) \* sin(DLon /2);  b = 2\* atan2(sqrt(a), sqrt(1-a));    (\*Out).Distance = 6371e3 \* b;    a = atan2(cos(Lat1) \* sin(Lat2) - sin(Lat1) \* cos(Lat2) \* cos(DLon), sin(DLon) \* cos(Lat2)) \* 180 / M\_PI;    (\*Out).Barring = fmod(((int)a + 180 ), 360);    } |

Figure 17: Distance Calculation Code

Adapted from

(Chris Veness, 2002-2016)

#### UI and main overview system

The only thing left is the UI and System overview. For this system a state machine was created. This state machine simply switched between an active and an input mode.

During the active mode it would just step through a cycle. First it would display data starting with initialization data. Then it would display the data following a period of mixed sleeps and button polls. It will do this for the following sets of data: device location, target location, and distance. After it is done it would re-poll its location and it would then repeat the entire cycle.

The input cycle merely allows you to input a target. The first result would be taken for a target in the active search and then state would be moved to an active search.

# Conclusion

In conclusion of this report, GPS technology is a great step forward. It helps a great many people and industries alike. Its usage is seen everywhere from within the military, to inside the government, and even for the average everyday use.

The project overall was a success. The insight the project provided into the inner mechanics of a GPS, such as system design, serial communication, and data parsing proved to be invaluable. It was a great path to explore and develop a deeper understanding of.

**References**

Adafruit Industries (2014, March). *GPS\_FPMMOPA6H\_v0.3*, Adafruit Industries, 150 Varick St New York NY 10013 USA, <https://learn.adafruit.com/adafruit-ultimate-gps/downloads>

Adafruit Industries (2014, March). *GPS\_FPMMOPA6H\_v0.3*, Adafruit Industries, 150 Varick St New York NY 10013 USA, <https://learn.adafruit.com/adafruit-fona-mini-gsm-gprs-cellular-phone-module/downloads>

Adafruit Industries (2015, March). *GPS\_FPMMOPA6H\_v0.3*, Adafruit Industries, 150 Varick St New York, NY 10013 USA, <https://learn.adafruit.com/adafruit-fona-mini-gsm-gprs-cellular-phone-module/pinouts>

Lady Ada (2014, March). *GPS\_FPMMOPA6H\_v0.3*, Adafruit Industries, 150 Varick St New York NY 10013 USA, <https://learn.adafruit.com/adafruit-ultimate-gps/pinouts>

Chris Veness (2002-2016). *Movable Type Scripts: Calculate distance, bearing and more between Latitude/Longitude points,* [ku.oc.epyt-elbavom@oeg-stpircs](mailto:ku.oc.epyt-elbavom@oeg-stpircs), <http://www.movable-type.co.uk/scripts/latlong.html>

GlobalTop Tech Inc. (2012). *PMTK command packet,* No.16 Nan-ke 9th Rd Science-based Ind. Park Tainan 741-47 Taiwan R.O.C., <https://cdn-shop.adafruit.com/datasheets/PMTK_A11.pdf>

Shanghai SIMCom Wireless Solutions Ltd. (2013). *SIM800 Series\_IP\_Application Note\_V1.00,* Building A SIM Technology Building No. 633 Jinzhong Road Shanghai P. R. China 200335, <https://cdn-shop.adafruit.com/datasheets/sim800_series_ip_application_note_v1.00.pdf>.

Eric Gakstatter (2015, Febuary). *What Exaxtly Is GPS NMEA Data*, GPS World,

<http://gpsworld.com/what-exactly-is-gps-nmea-data/>

NAIT CNT (2014). *Development Board Main Schematic*, The Northern Alberta Institute of Technology, CNT Program, 11762 106 St NW Edmonton AB T5G 2R1 <https://moodle.nait.ca/pluginfile.php/3370613/mod_resource/content/1/XDP512CCT_Landscape_C.pdf>

Freescale Semiconductor (2008). *MC9S12XDP512 Data sheet, Rev 2.21*, Freescale Semiconductor Technical Information Center, CH370 1300 N. Alma School Road Chandler Arizona 85224, <http://www.nxp.com/assets/documents/data/en/data-sheets/MC9S12XDP512RMV2.pdf>