The Global Positioning System (GPS) is rather large and complex. Thus any type of description I would write would be insufficient, thus I refer you to the following link

http://www.trimble.com/gps_tutorial/

Here is my code for reading in the NEMA strings from a GPS. The most important string being the GPGGA string the form of which is shown here. The time is shown in red, latitude is shown in green, longitude blue and altitude is cyan

```
$GPGGA,183738,3907.354,N,12102.480,W,1,05,1.6,646.4,M,-24.1,M,,*75
```

The following interpretations should help, time is Zulu or Universal Time Coordinated (UTC) and in the string above is 18:37:38. The latitude is written as the first digits being the degrees, then the two digits prior to the decimal point and those following are the minutes (60^{th} of a degree). Weird I know but such thing happen. The letter N indicates that the results are for the Northern hemisphere. The longitude comes next and it encoded the same as the longitude. Other data is available here but not that helpful in this discussion.

This string would be decoded as

Time = 18:37:38 (UTC)
Latitude = 39 + 7.354 / 60 = 39.122566
Longitude = 39 + 7.354 / 60 = 121.041344

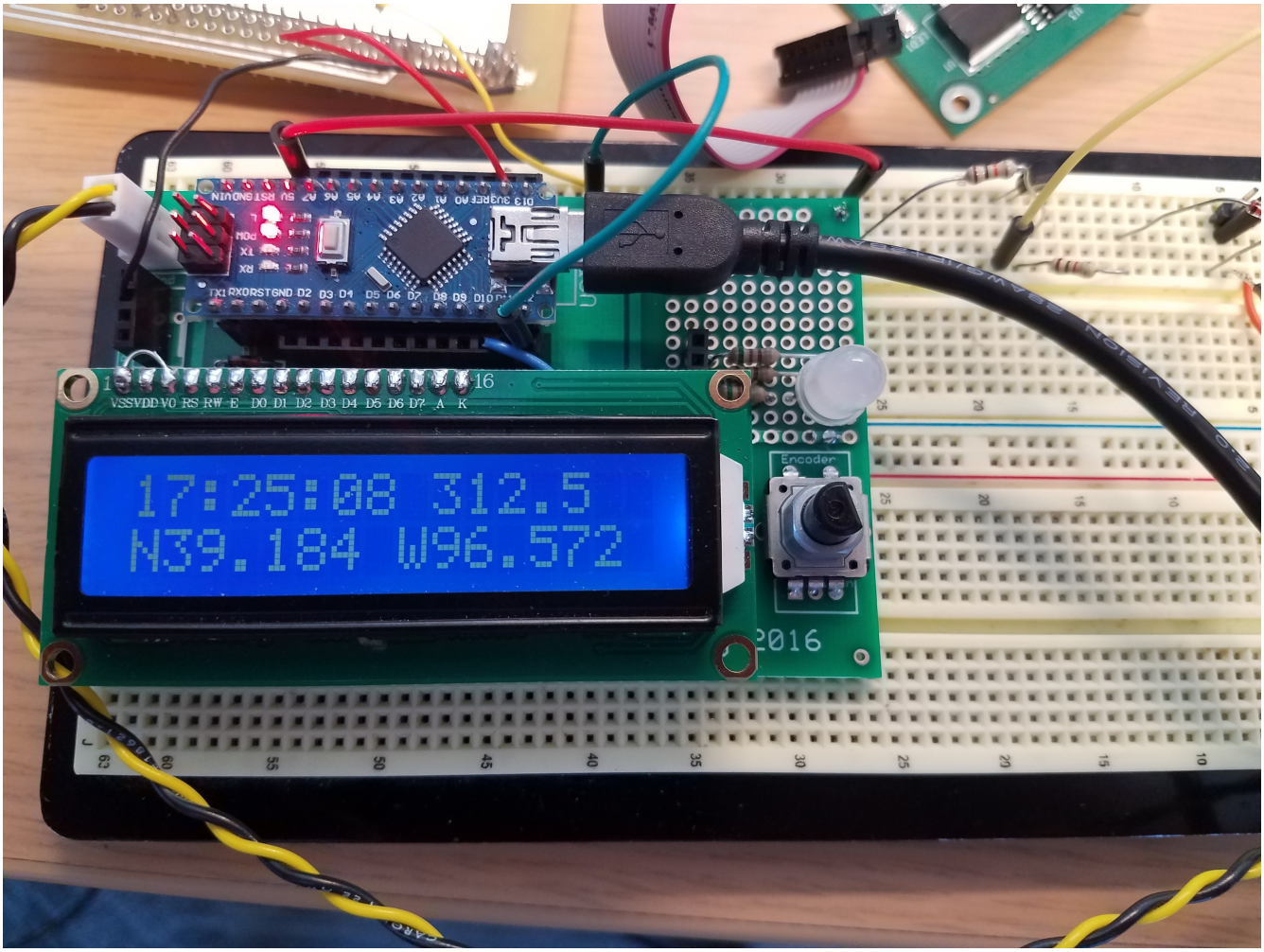
The Haversine formula for distance between two points defined by their lat/long would be.

$$a = \sin^2(\Delta\varphi/2) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2(\Delta\lambda/2)$$

$$c = 2 \cdot \operatorname{atan2}(\sqrt{a}, \sqrt{(1-a)})$$

$$d = R \cdot c$$

where φ *is latitude,* λ *is longitude,* R *is earth's radius (mean radius = 6,371km);*
note that angles need to be in radians to pass to trig functions!



Reference for location of Manhattan https://www.latlong.net/place/manhattan-ks-usa-13646.html

Manhattan DMS coordinates

39° 11' 0.9924" N and 96° 34' 18.0156" W or 39.183609, -96.571671.

https://en.wikipedia.org/wiki/Decimal_degrees

A value in decimal degrees to a precision of 4 decimal places is precise to 11.132 meters at the equator.
A value in decimal degrees to 5 decimal places is precise to 1.1132 meter at the equator.

Appendix A: Arduino Code.

```
#include <LiquidCrystal.h>
// Define LCD
LiquidCrystal LcdDriver(11, 9, 5, 6, 7, 8);

// Setup clock
#include "ClockBasics.h"
#define CLOCK_INTERVAL 1000
unsigned long ClockTimer;

// Setup GPS variables.
float Latitude = 0.0,
Longitude = 0.0,
Altitude = 0.0;
char  NorthSouth = 'N',
EastWest = 'W';
int   Fix = 0,
Sats = 0;

// Simple timer to Update Display.
#define INTERVAL 50
unsigned long Timer;

#include <SoftwareSerial.h> // Header for serial code.
SoftwareSerial SW_Serial(12, 13); // RX, TX
#include "DecodeGPGGA.h" // Header for GPS decoding

// Run once, To set up system
void setup()
{
       // Initialize update and clock time.
       Timer = millis();
       ClockTimer = millis();

       // Set up basic serial port.
       Serial.begin(9600);
       // Set up LCD.
       LcdDriver.begin(16, 2);

       // Software Based Serial.
       SW_Serial.begin(9600);

} // End of setup
```

```
// Code that is run continuously.
void loop()
{
	char ch;

	// Clock
	if (millis() - ClockTimer >= CLOCK_INTERVAL)
	{
		// update and display clock.
		UpdateClock();
		LcdDriver.clear();
		LcdDriver.setCursor(0, 0);
		SendClock();
		// Place GPS date on next
		LcdDriver.setCursor(9, 0);
		LcdDriver.print(Altitude, 1);
		LcdDriver.setCursor(0, 1);
		LcdDriver.print(NorthSouth);
		LcdDriver.print(Latitude, 3);
		LcdDriver.print(" ");
		LcdDriver.print(EastWest);
		LcdDriver.print(Longitude, 3);

		ClockTimer += CLOCK_INTERVAL;
	} // End of clock timer

	// Check for incoming serial data.
	ch = SW_Serial.read();
	if (ch != -1)
	{
		// send strings to serial monitor for DEMO.
		Serial.print(ch);

		// Scan incoming characters for GPGGA string
		if (GGADecoderProcessor(ch))
		{
			// Copy data from GPGGA string into
			// local variables.
			Hours = GPS_Time / 10000;
			Minutes = (GPS_Time - 10000 * (long)Hours) / 100;
			Seconds = (GPS_Time - 10000 * (long)Hours - 100 * (long)Minutes);

			Latitude = GPS_Latitude,
			NorthSouth = GPS_NorthSouth;

			Longitude = GPS_Longitude,
			EastWest = GPS_EastWest;

			Altitude = GPS_Altitude;

			Fix = GPS_Fix,
			Sats = GPS_Sats;

		} // End of GPGGA decode

	} // End of Serial available if

} // End of loop
```

Appendix B: Code for decoding GPS Strings.

```
#ifndef DecodeGPGGA_h
#define DecodeGPGGA_h 1

// Set up decoding state machine.
enum GPGGA_Decoder {
	GPS_WAIT, // waiting for starting $.
	GPS_G,    // Waiting for first G
	GPS_GP,    // Waiting for P
	GPS_GPG,   // Waiting for G
	GPS_GPGG,  // Waiting for G
	GPS_GPGGA, // Waiting for final A
	GPS_COMMA,
	GPS_TIME,  // Reading through time.
	GPS_TIME_MS,
	GPS_LATITUDE_INT,
	GPS_LATITUDE_FRAC,
	GPS_N_S,
	GPS_LONGITUDE_INT,
	GPS_LONGITUDE_FRAC,
	GPS_E_W,
	GPS_FIX,
	GPS_SATS,
	GPS_DILUTION,
	GPS_ALTITUDE_INT,
	GPS_ALTITUDE_FRAC,
	GPS_END
};

GPGGA_Decoder GGADecoder = GPS_WAIT;

long GPS_Time = 0;
int GPS_Time_ms = 0;
float GPS_Latitude = 0.0, GPS_Longitude = 0.0;
float GPS_Fraction = 0.0, GPS_Degrees;
char  GPS_NorthSouth = 'N', GPS_EastWest = 'W';
int   GPS_Fix = 0, GPS_Sats = 0;
float GPS_Altitude = 0.0;

int GGADecoderProcessor(char Incoming)
{
	switch (GGADecoder)
	{
	default:
	case GPS_WAIT:
		if (Incoming == '$')
			GGADecoder = GPS_G;
		break;
	case GPS_G:
		if (Incoming == 'G')
			GGADecoder = GPS_GP;
		else
			GGADecoder = GPS_WAIT;
		break;
```

```
case GPS_GP:
	if (Incoming == 'P')
		GGADecoder = GPS_GPG;
	else
		GGADecoder = GPS_WAIT;
	break;
case GPS_GPG:
	if (Incoming == 'G')
		GGADecoder = GPS_GPGG;
	else
		GGADecoder = GPS_WAIT;
	break;
case GPS_GPGG:
	if (Incoming == 'G')
		GGADecoder = GPS_GPGGA;
	else
		GGADecoder = GPS_WAIT;
	break;
case GPS_GPGGA:
	if (Incoming == 'A')
		GGADecoder = GPS_COMMA;
	else
		GGADecoder = GPS_WAIT;
	break;
case GPS_COMMA:
	if (Incoming == ',')
	{
		GPS_Time = 0;
		GPS_Latitude = -1.0;
		GPS_Longitude = -1.0;
		GPS_Altitude = -500.0;
		GGADecoder = GPS_TIME;
	}
	else
		GGADecoder = GPS_WAIT;
	break;
case GPS_TIME:
	if (Incoming == ',')
		GGADecoder = GPS_LATITUDE_INT;
	else if (Incoming == '.')
		GGADecoder = GPS_TIME_MS;
	else if (Incoming >= '0' && Incoming <= '9')
		GPS_Time = GPS_Time * 10l + (Incoming - '0');
	break;
case GPS_TIME_MS:
	if (Incoming == ',')
		GGADecoder = GPS_LATITUDE_INT;
	else if (Incoming >= '0' && Incoming <= '9')
		GPS_Time_ms = GPS_Time_ms * 10 + (Incoming - '0');
	break;
```

```
case GPS_LATITUDE_INT:
	if (Incoming == '.')
	{
		GPS_Fraction = 0.1F;
		GGADecoder = GPS_LATITUDE_FRAC;
	}
	else if (Incoming == ',')
		GGADecoder = GPS_N_S;
	else if (Incoming >= '0' && Incoming <= '9')
		GPS_Latitude = GPS_Latitude*10.0F
		+ (float)(Incoming - '0');
	break;
case GPS_LATITUDE_FRAC:
	if (Incoming == ',')
	{
		GPS_Degrees = (int)(GPS_Latitude / 100);
		GPS_Latitude = GPS_Degrees
			+ (GPS_Latitude - 100 * GPS_Degrees) / 60.0f;

		GGADecoder = GPS_N_S;
	}
	else if (Incoming >= '0' && Incoming <= '9')
	{
		GPS_Latitude = GPS_Latitude
			+ GPS_Fraction * (Incoming - '0');
		GPS_Fraction *= 0.1F;
	}
	break;
case GPS_N_S:
	if (Incoming == ',')
		GGADecoder = GPS_LONGITUDE_INT;
	else
		GPS_NorthSouth = Incoming;
	break;
case GPS_LONGITUDE_INT:
	if (Incoming == '.')
	{
		GPS_Fraction = 0.1F;
		GGADecoder = GPS_LONGITUDE_FRAC;
	}
	else if (Incoming == ',')
	{
		GPS_Degrees = (int)(GPS_Longitude / 100);
		GPS_Longitude = GPS_Degrees
			+ (GPS_Longitude - 100 * GPS_Degrees) / 60.0f;
		GGADecoder = GPS_E_W;
	}
	else if (Incoming >= '0' && Incoming <= '9')
		GPS_Longitude = GPS_Longitude*10.0F
		+ (float)(Incoming - '0');
	break;
```

```
	case GPS_LONGITUDE_FRAC:
		if (Incoming == ',')
		{
			GPS_Degrees = (int)(GPS_Longitude / 100);
			GPS_Longitude = GPS_Degrees
				+ (GPS_Longitude - 100 * GPS_Degrees) / 60.0f;
			GGADecoder = GPS_E_W;
		}
		else if (Incoming >= '0' && Incoming <= '9')
		{
			GPS_Longitude = GPS_Longitude
				+ GPS_Fraction * (Incoming - '0');
			GPS_Fraction *= 0.1F;
		}
		break;
	case GPS_E_W:
		if (Incoming == ',')
			GGADecoder = GPS_FIX;
		else
			GPS_EastWest = Incoming;
		break;
	case GPS_FIX:
		if (Incoming == ',')
			GGADecoder = GPS_SATS;
		else if (Incoming >= '0' && Incoming <= '9')
			GPS_Fix = (Incoming - '0');
		break;
	case GPS_SATS:
		if (Incoming == ',')
			GGADecoder = GPS_DILUTION;
		else if (Incoming >= '0' && Incoming <= '9')
			GPS_Sats = (Incoming - '0');
		break;
	case GPS_DILUTION:
		if (Incoming == ',')
		{
			GGADecoder = GPS_ALTITUDE_INT;
		}
		break;
	case GPS_ALTITUDE_INT:
		if (Incoming == '.')
		{
			GPS_Fraction = 0.1F;
			GGADecoder = GPS_ALTITUDE_FRAC;
		}
		else if (Incoming == ',')
			GGADecoder = GPS_END;
		else if (Incoming >= '0' && Incoming <= '9')
			GPS_Altitude = GPS_Altitude*10.0F
			+ (float)(Incoming - '0');
		break;
```

```
	case GPS_ALTITUDE_FRAC:
		if (Incoming == ',')
		{
			GGADecoder = GPS_END;
			return 1;
		}
		else if (Incoming >= '0' && Incoming <= '9')
		{
			GPS_Altitude = GPS_Altitude
				+ GPS_Fraction * (Incoming - '0');
			GPS_Fraction *= 0.1F;
		}
		break;
	case GPS_END:
		if (Incoming == '\n' || Incoming == '\r')
		{
			GGADecoder = GPS_WAIT;
			return 1; // End detected so return a true
		}
		break;
	}
	return 0; // by default return false.
}
#endif
```

Appendix C: Test Code for GPS decoder.

```
// GPS_Decoder.cpp : Defines the entry point for the console application.
//

#include "DecodeGPGGA.h"
int Hours, Minutes, Seconds;
int Integer;

int main(int argc, _TCHAR* argv[])
{
	FILE *fin;
	char ch;

	fopen_s(&fin, "GpsDataNULL.txt", "r");

	if (fin)
	{
		while (!feof(fin))
		{
			ch = getc(fin);

			if (GGADecoderProcessor(ch))
			{
				Hours = GPS_Time / 10000;
				Minutes = (GPS_Time - Hours * 10000) / 100;
				Seconds = (GPS_Time - Hours * 10000 - Minutes * 100);
				/*
				Integer = (GPS_Latitude / 100);
				GPS_Latitude = GPS_Latitude - 100.0f * Integer;
				GPS_Latitude = (float)Integer + GPS_Latitude / 60.0f;
				Integer = (GPS_Longitude / 100);
				GPS_Longitude = GPS_Longitude - 100.0f * Integer;
				GPS_Longitude = (float)Integer + GPS_Longitude / 60.0f;
				*/
				printf("%d, %f, %f \n",
					GPS_Time, GPS_Latitude, GPS_Longitude);
			}
		}
		fclose(fin);
	}
	return 0;
}
```

Sample GPS Data used to test decoder.

```
$GPRMC,183729,A,3907.356,N,12102.482,W,000.0,360.0,080301,015.5,E*6F
$GPRMB,A,,,,,,,,,,,V*71
$GPGGA,183730,3907.356,N,12102.482,W,1,05,1.6,646.4,M,-24.1,M,,*75
$GPGSA,A,3,02,,,07,,09,24,26,,,,,1.6,1.6,1.0*3D
$GPGSV,2,1,08,02,43,088,38,04,42,145,00,05,11,291,00,07,60,043,35*71
$GPGSV,2,2,08,08,02,145,00,09,46,303,47,24,16,178,32,26,18,231,43*77
$PGRME,22.0,M,52.9,M,51.0,M*14
$GPGLL,3907.360,N,12102.481,W,183730,A*33
$PGRMZ,2062,f,3*2D
$GPGGA,183734,3907.355,N,12102.481,W,1,05,1.6,646.4,M,-24.1,M,,*75
$PGRMM,WGS 84*06
$GPBOD,,T,,M,,*47
$GPRTE,1,1,c,0*07
$GPRMC,183731,A,3907.482,N,12102.436,W,000.0,360.0,080301,015.5,E*67
$GPRMB,A,,,,,,,,,,,V*71
$GPRMC,183729,A,3907.356,N,12102.482,W,000.0,360.0,080301,015.5,E*6F
$GPRMB,A,,,,,,,,,,,V*71
$GPGGA,183738,3907.354,N,12102.480,W,1,05,1.6,646.4,M,-24.1,M,,*75
$GPGSA,A,3,02,,,07,,09,24,26,,,,,1.6,1.6,1.0*3D
$GPGSV,2,1,08,02,43,088,38,04,42,145,00,05,11,291,00,07,60,043,35*71
$GPGSV,2,2,08,08,02,145,00,09,46,303,47,24,16,178,32,26,18,231,43*77
$PGRME,22.0,M,52.9,M,51.0,M*14
$GPGLL,3907.360,N,12102.481,W,183730,A*33
$PGRMZ,2062,f,3*2D
$PGRMM,WGS 84*06
$GPBOD,,T,,M,,*47
$GPRTE,1,1,c,0*07
$GPRMC,183731,A,3907.482,N,12102.436,W,000.0,360.0,080301,015.5,E*67
$GPRMB,A,,,,,,,,,,,V*71
```

Results from test code

```
183730, 39.122604, 121.041374
183730, 39.122604, 121.041374
183734, 39.122581, 121.041359
183734, 39.122581, 121.041359
183738, 39.122566, 121.041344
```