

Carleton University
School of Computer Science
COMP4905 Honours Project

Remote Tracking of a GPS Tracking Device
with Google Maps via GSM Cellular Networks
By: Michael Nemat

Supervised By: Dr. Michel Barbeau
School of Computer Science
Carleton University

Submitted on July 30, 2010

Abstract

In the past few years, there has been a growing trend of incorporating geo-location technologies with mobile communication devices. Recently, this growth has been spurred by social networking and location-aware software applications. The demand for real-time geo-location tracking of assets and persons has been growing while the costs associated with these technologies have been dropping. As a result, many corporations, even individuals, are using these technologies in a cost-effective fashion to achieve goals which may have been too complex or prohibitively expensive to achieve in the past. This project will demonstrate how to build an end-to-end solution for near-real-time geo-location tracking by integrating a wide variety of readily available components and some custom hardware and software into a purpose-built geo-location platform. This solution, as opposed to purchasing an end-to-end product commercially, is much more desirable, both from a cost and security perspective.

Table Of Contents

1	Background and Objectives	5
2	Tracking Device Hardware Design	6
2.1	Hardware Architecture	6
2.2	Circuit Board Design and Fabrication	7
2.3	Breakout Boards and Component Selection	12
2.4	Bill of Materials	13
3	Tracking Device Hardware Assembly	15
3.1	Assembly Procedure	15
3.2	Assembly Pictures	17
3.3	Design Errors and Workarounds	29
4	Tracking Device Software Development	30
4.1	Software Architecture	30
4.2	Development Environment	30
4.3	Software Functionality	31
4.4	Power-Saving Considerations	34
4.5	Platform Limitations	34
4.6	Atmel Microcontroller Software	34
5	Tracking Device Software Testing	35
5.1	No GSM Signal at Initialization	35
5.2	No GPS Signal at Initialization	36
5.3	Normal Initialization and Successful Run Loops	36
5.4	Unexpected Loss of GPS Signal During Run Loop	37
5.5	Unexpected Loss of GSM Signal During Run Loop	38

6	Application Server Development	39
7	Google Maps Front-End Service	41
	7.1 Login	41
	7.2 View	41
	7.3 Database Schema	43
8	Usage Overview and Procedures	44
9	System Testing and Performance Results	51
	9.1 Battery Life and Time to Transmit	51
	9.2 Accuracy Testing	51
10	Lessons Learned	67
11	Potential Improvements	68
12	Conclusion	69
13	Relation and Contribution to Computer Science	70
14	References	71
	Appendix A - Submitted Files	72

1 Background and Objectives

An end-to-end commercial off-the-shelf tracking solution can be defined a product that consists of a hardware device and a software solution for viewing the location of that device in near-real-time using some sort of a service (web-based, etc...). Most often, these services are subscription based, with a monthly fee or licensing fee associated to their use. Additionally, these services are often closed and proprietary in nature, making them extremely undesirable for sensitive applications due to security and privacy concerns.

Typical applications of these technologies can include:

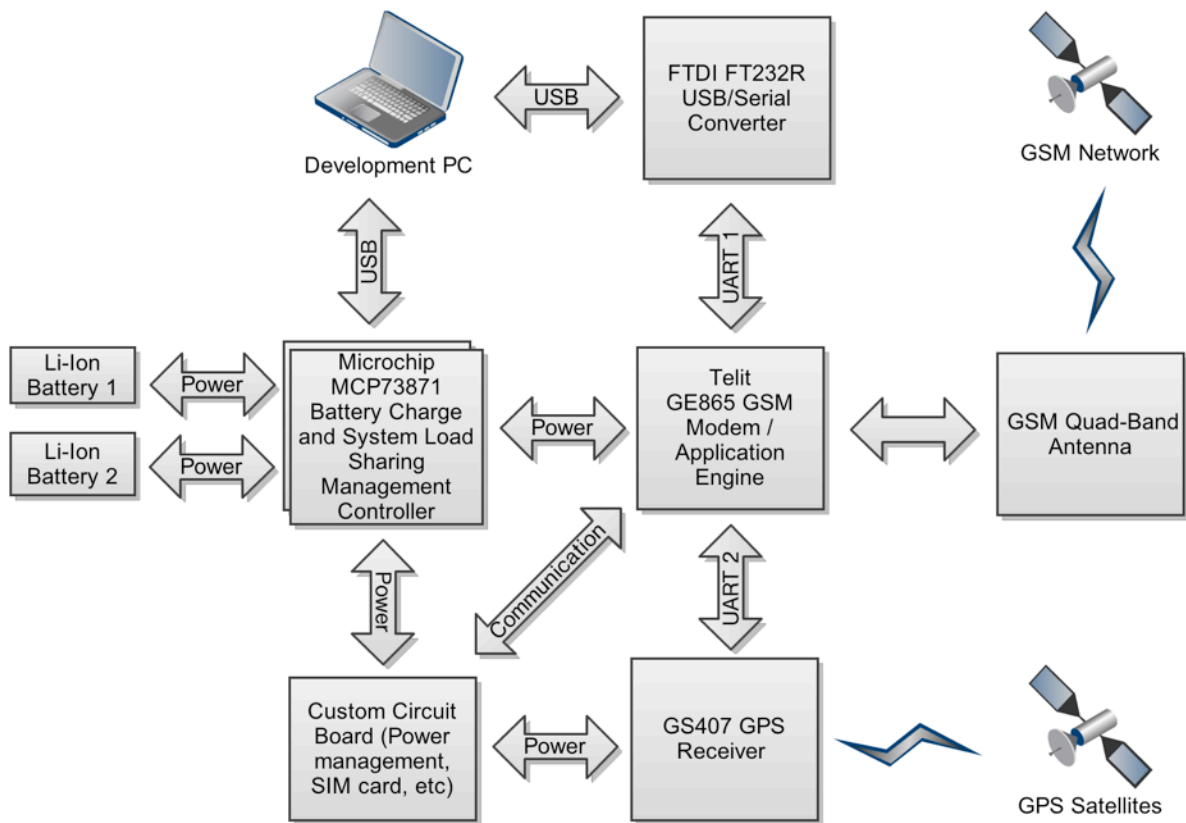
- Tracking a fleet of delivery vehicles to optimize re-routes, and optimal distribution of new pickup/delivery requests
- Tracking a fleet of public transportation vehicles to report accurate and dynamic stop schedules to passengers as opposed to static printed time-tables
- Tracking rental vehicles to ensure that policies regarding regional use and distance are followed
- Parents tracking the location of their children to/from school
- Parents tracking the location of their children as new drivers
- Covert tracking of business assets (company vehicles, domestic shipments)
- Covert tracking of human targets by law enforcement (either by planting a tracking device on or inside a target's vehicle or in an accessory carried by the target)
- Tracking for personal statistics (exercise related: bike rides, walking distance, etc...)
- Personal asset tracking (tracking personal vehicle in case of theft, etc...)

The objective of this project is to demonstrate how to develop both hardware and software solutions for near-real-time remote geo-location using GPS geo-location technology and GSM cellular mobile communication networks. In addition, this project will focus on approaching these issues with the primary objective of reducing the complexity of incorporating these technologies into a functional product by making use of existing technologies, many of which are open-source and/or free. As a result, persons or organizations interested in these solutions can develop their own products without any of the licensing or usage fees of similar commercial off-the-shelf technologies. Similarly, due to the often sensitive nature of the use of these technologies (privacy and/or covert monitoring), it is in one's own advantage from a security perspective to develop their own solution with as much open-source technology as possible to ensure that there is no unwanted, malicious, or otherwise undesirable functionality in the product which could compromise the successful application of the technology. This project will outline how to build a self-contained GPS tracking device capable of reporting its position to a remote server using GSM cellular networks and GPRS. The remote server will make use of the Google Maps API to generate a real-time view of this data. This data will be stored in an open format using an SQL database to simplify the task of developing an application which makes use of location information (whether it be for viewing a location on the map, generating statistics, etc..).

2 Tracking Device Hardware Design

This project will take an informal approach to hardware design due to skill and cost constraints. The only component that will be formally designed is the circuit board which facilitates integrating the various components. The enclosure (plastic case) will be an off-the-shelf product and modified in an ad-hoc fashion to fit any antennas, switches, or mounts as required.

2.1 Hardware Architecture



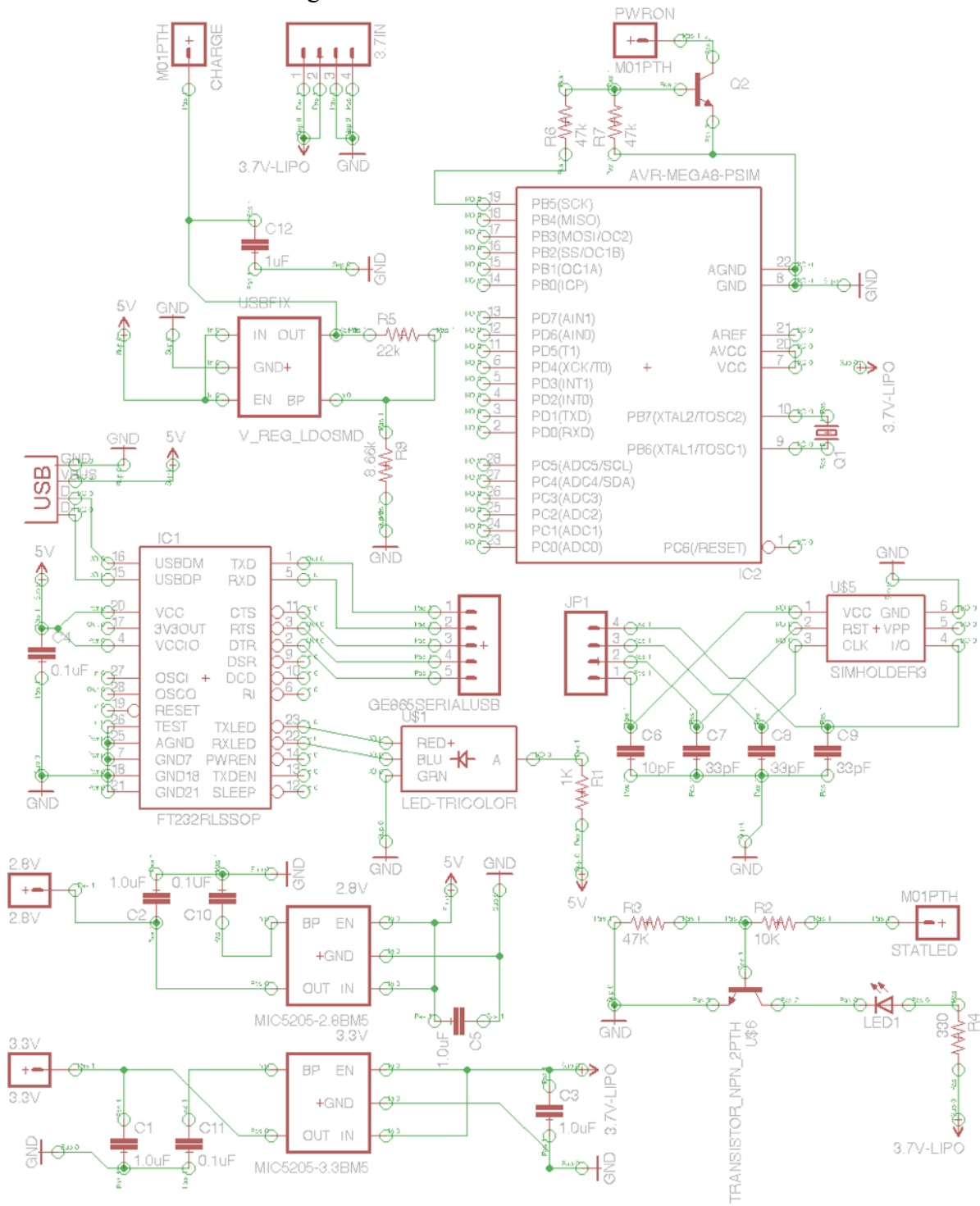
2.2 Circuit Board Design and Fabrication

In order for this project to fit in a small enclosure, it was necessary to build a custom circuit board to do a variety of things. This board offers the following facilities and all of the supporting circuitry for them:

- USB connector (data lines to USB/Serial chip, power line to breakout pin for battery charge boards)
- Voltage regulation for powering GPS receiver (3.3V), UART I/O reference voltage (2.8V), load sharing voltage (4.3V)
- Indicator LEDs (Status LED, USB I/O LEDs)
- SIM Card breakout
- Atmel Microcontroller for controlling GSM Module initialization
- Voltage divider for GPS UART (3.3V) to GSM Module UART (2.8V)

The circuit board was designed in EAGLE 5.6.0 Light [1]. The first generation board makes minimal use of surface mount components and utilizes a variety of secondary breakout boards for the GSM module, charge controllers, etc... This project will be built with the first generation board. A second generation surface-mount board with everything integrated was designed but never fabricated, but the schematic will be shown below as a system electrical schematic showing the entire system interconnected titled EAGLE System Schematic. This system schematic also makes use of some improvements like a load switch suggested throughout this report.

EAGLE Board Schematic Design:



The design “PCB Layout for Fabrication” was sent to BatchPCB for fabrication (BatchPCB.com). BatchPCB produces circuit boards in quantities as little as one piece for a very reasonable price [2]. Turnaround time is roughly one month, sometimes less.

2.3 Breakout Boards and Component Selection

Several of the components referred to in the above sections are not located on the custom circuit board but are rather located on secondary breakout boards (separate circuit boards with pin headers for wiring). In fact, due to either manufacturing error or assembly error, the FT232R USB to serial conversion chip located on the custom circuit board was non-functional and thus is also located on a breakout board in the assembled product.

The following breakout boards are used in this project:

- 2x Microchip MCP73871EV Li-Ion Battery Charging and System Load Sharing boards.
 - Used for charging the two lithium ion batteries and powering the system from USB while not charging
 - Two were required because the current peak requirement of the GSM modem can slightly exceed the maximum current output of a single board
- Sparkfun GE865 Breakout board
 - Contains a Telit GE865 GSM module and allows access to every pin via pin headers
 - Also has a surface mount U.FI antenna jack for the GSM antenna
- Sparkfun FTDI FT232RL USB to Serial breakout board
 - Has a USB mini-B connector and the FT232RL chip
- DIYDrones GS406/407 Breakout board
 - Converts the surface mount connector on the GPS unit to pin headers for easy access

In addition to the breakout boards, this project contains several large commercially available products as well as various smaller components (LEDs, resistors, etc described in the bill of materials):

- 2x Union Battery 2000mAh Lithium-Polymer batteries
- GS407 GPS Receiver
 - This is a generic GPS receiver which combines a Sarantel GeoHelix antenna with a U-Blox 5 GPS chipset. The Sarantel GeoHelix antenna is suited to projects where the antenna is not guaranteed to face the sky, as opposed to typical ceramic GPS antennas. This made the GS407 receiver very desirable.
 - The U-Blox 5 chipset also supports software configuration of various GPS functions and output formats which simplifies the work required to implement it into a project (especially from a software perspective). It also supports AGPS.
- Atmel ATmega168 microcontroller with arduino boot loader
 - Microcontroller that can be programmed with the Arduino development environment. Very straightforward to incorporate into a project.

2.4 Bill of Materials

Item	Store	Price
DIYDrones uBlox GS406/GS407 breakout adapter board	diydrone	19.50
Custom Circuit Board manufactured by BatchPCB	batchpcb	25.04
PRT-00587 USB miniB SMD Connector	sparkfun	1.50
2x PRT-08483 Polymer Lithium Ion Batteries - 2000mAh	sparkfun	33.90
COM-08532 Basic LED - Green	sparkfun	0.35
CEL-00675 Quad-band Cellular Duck Antenna SMA	sparkfun	7.95
COM-00521 Common BJT Transistors - NPN 2N3904	sparkfun	0.75
PRT-00548 SIM Socket	sparkfun	0.95
COM-07844 Triple Output LED RGB - SMD	sparkfun	2.25
WRL-09145 Interface Cable SMA to U.FL	sparkfun	4.95
GPS-09436 50 Channel GS407 Helical GPS Receiver	sparkfun	89.95
CEL-09297 GE865 Breakout	sparkfun	99.95
COM-00536 Crystal 16MHz	sparkfun	1.50
DEV-08846 ATmega168 with Arduino Bootloader	sparkfun	4.95
BOB-00718 Breakout Board for FT232RL USB to Serial	sparkfun	14.95
490-3810-ND 10x 0.1uF Radial Ceramic Capacitor	digikey	1.52
LP3990MF-2.8CT-ND Micrel 2.8VDC LDO Regulator	digikey	1.31
445-4804-ND 5x 1.0uF Radial Ceramic Capacitor	digikey	1.35
1.0KXBK-ND 5x 1.0KOhm 1/4W 1% Metal Film Resistor	digikey	0.60
490-3678-ND 2x 10pF Radial Ceramic Capacitor	digikey	0.80
490-3725-ND 4x 33pF Radial Ceramic Capacitor	digikey	1.56
445-2882-ND 5x 22uF Radial Ceramic Capacitor	digikey	2.98
478-1847-ND 100uF 20V 10% Radial Tantalum Capacitor	digikey	9.17
10.0KXBK-ND 5x 10.0KOhm 1.4W 1% Metal Film Resistor	digikey	0.60
330QBK-ND 5x 330 Ohm 1/4W 5% Carbon Film Resistor	digikey	0.36
47KQBK-ND 5x 47KOhm 1/4W 5% Carbon Film Resistor	digikey	0.36
LP3990MF-3.3CT-ND Micrel 3.3VDC LDO Regulator	digikey	1.31

Item	Store	Price
BC337-ND NPN Transistor	digikey	0.44
679-1886-ND Mini Slide Switch	digikey	4.24
377-1651-ND Plastic Box 3.62x2.60x1.10	digikey	3.29
1.1KQBK-ND 5x 1.1KOhm 1/4W 5% Carbon Film Resistor	digikey	0.36
2x MCP73871EV-ND Breakout board for MCP73871 charge chip	digikey	22.26
576-1281-1-ND Micrel Adjustable 500mA DC LDO Regulator	digikey	1.93
CMF8.66KHFCT-ND 2x 8.66KOhm 1% 1/2W Resistor	digikey	0.38
P22.0KCACT-ND 10x 22.0KOhm 1% 1/4W Metal Film Resistor	digikey	1.94
Total:		\$360.20

3 Tracking Device Hardware Assembly

3.1 Assembly Procedure

Building the hardware device assumed the availability of a number of tools: multimeter, soldering iron(s), illuminated loup, Arduino, kynar wire, 16/18 gauge stranded copper wire, electrical tape, hot glue and glue gun, double sided tape, tweezers, pliers, screwdrivers, wire cutters, and a USB mini-B cable.

The assembly procedure can be outlined generally as follows:

1. Tape batteries together
2. Tape charging boards on to the top of one of the batteries
3. Solder battery connections to respective inputs on the charging boards
4. Connect CHARGE_ENABLE pads on charging boards to Voltage Input pins (Kynar wire)
5. Connect Voltage Input and Voltage Output pads on charging boards together (wire the input and outputs in parallel)
6. Set DIP switches on charging boards to low-current USB charging (5V @ 100mA each)
7. Program microcontroller using Arduino
8. Solder components to circuit board
9. Insert SIM card in to SIM card slot
10. Use a dremel to remove protrusions from case (if any exist) so that it is flat
11. Cut holes for antennas, LEDs, and switches in the casing using a dremel
12. Mount batteries and charging boards with double-sided tape in to casing
13. Attach GS407 Breakout board to GS407 GPS using hot glue
14. Solder TX/RX/VIN/GND wires to GPS breakout board
15. Hot glue GPS receiver into case
16. Mount GSM Antenna jack (nut and bolt) to case (hot glue in place as well)
17. Mount power switch to case (hot glue).
18. Connect GPS receiver to arduino USB/Serial adapter pins to interface with development PC for software configuration and testing (optional step)
19. Connect bench power supply or some other 5V source to voltage input pins on charging boards to test and charge batteries
20. Solder wires from voltage output pins of charging boards to power switch
21. Solder wires from power switch to voltage input pin of the GE865 breakout board
22. Solder tantalum capacitor between voltage input and ground pin of the GE865 breakout board
23. Connect serial pins of GE865 board to arduino USB/serial adapter pins for testing
24. Connect SIM card pins, status LED pin, and power on pin from custom PCB to GE865 board

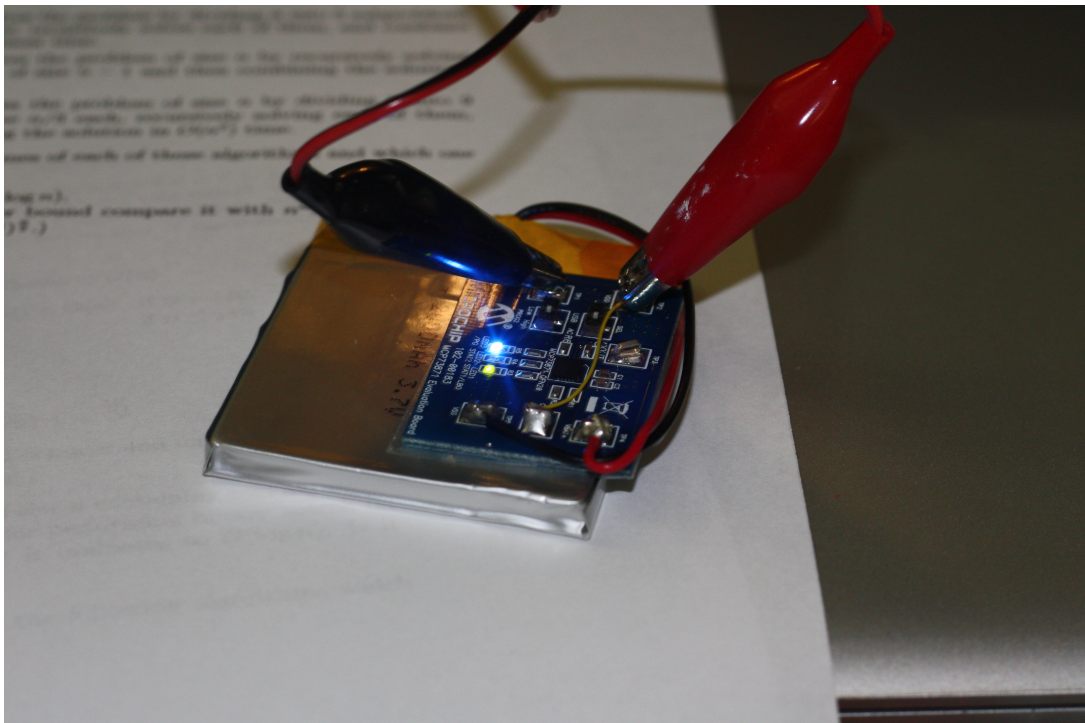
25. Connect power pins from custom PCB to power switch
26. Connect GPS power pins to custom PCB and TX/RX pins to GE865 board
27. Connect serial I/O pins from USB/Serial (FT232) breakout board to GE865 module
28. Connect GPS TX/RX pins to GE865 module
29. Connect USB 5V pin from FT232 Breakout board to voltage input pins on charging boards
30. Connect GSM antenna to antenna jack on GE865 breakout board
31. Test every subcomponent and communications to ensure functionality
32. Tape and glue all remaining components in place and close/seal the case

3.2 Assembly Pictures

Battery and charging boards taped together.



Testing the charging board with a bench power supply



Dremel rotary tool used for flattening mounting posts in the case.



Top half of the plastic case. Rough areas of plastic are where protrusions were removed.



GPS Receiver and connector breakout board glued in to position.



GSM SMA to U.FI jack and cable mounted to case.



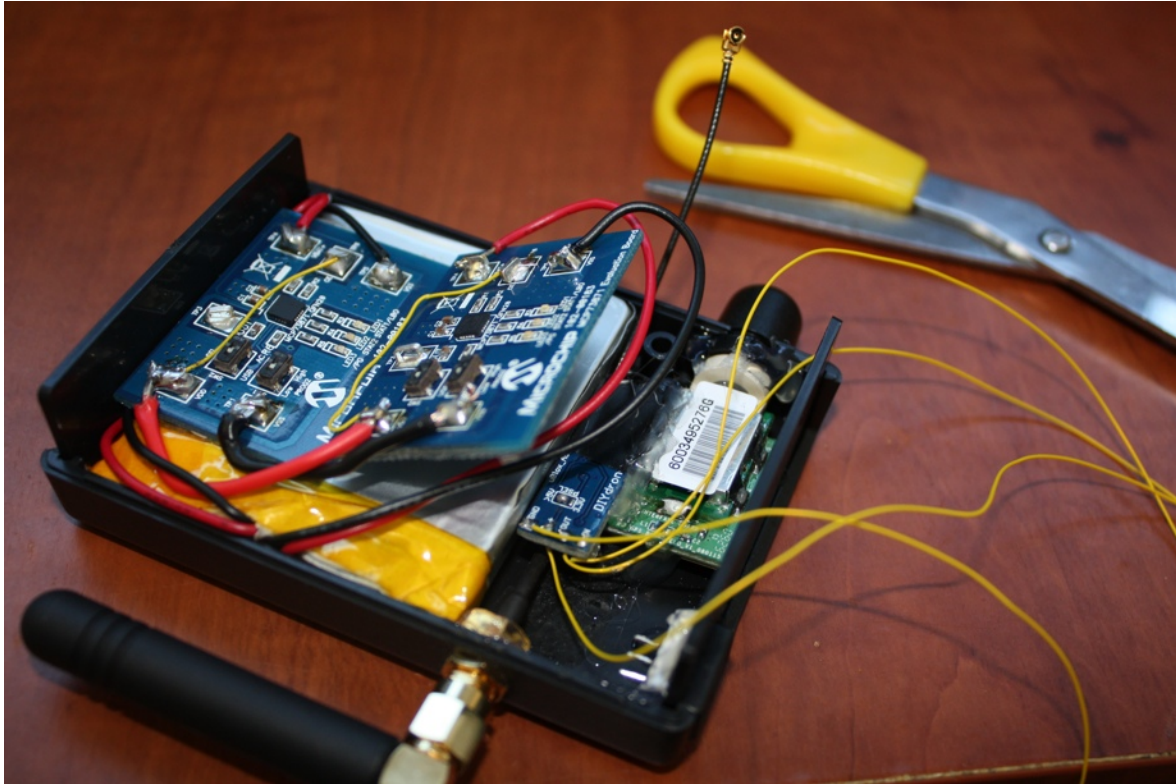
GSM antenna connected to jack. Power switch position at the back of the case (white object in the picture with the 3 pins).



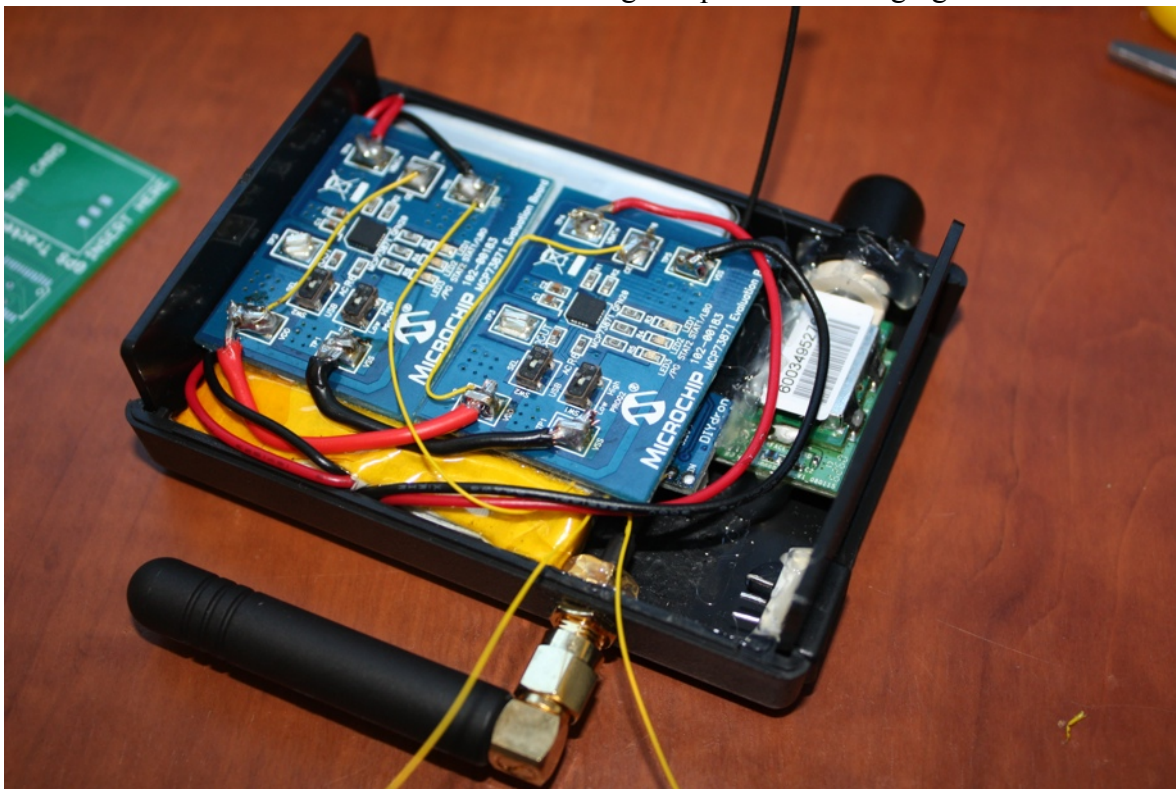
Power switch glued in to position.



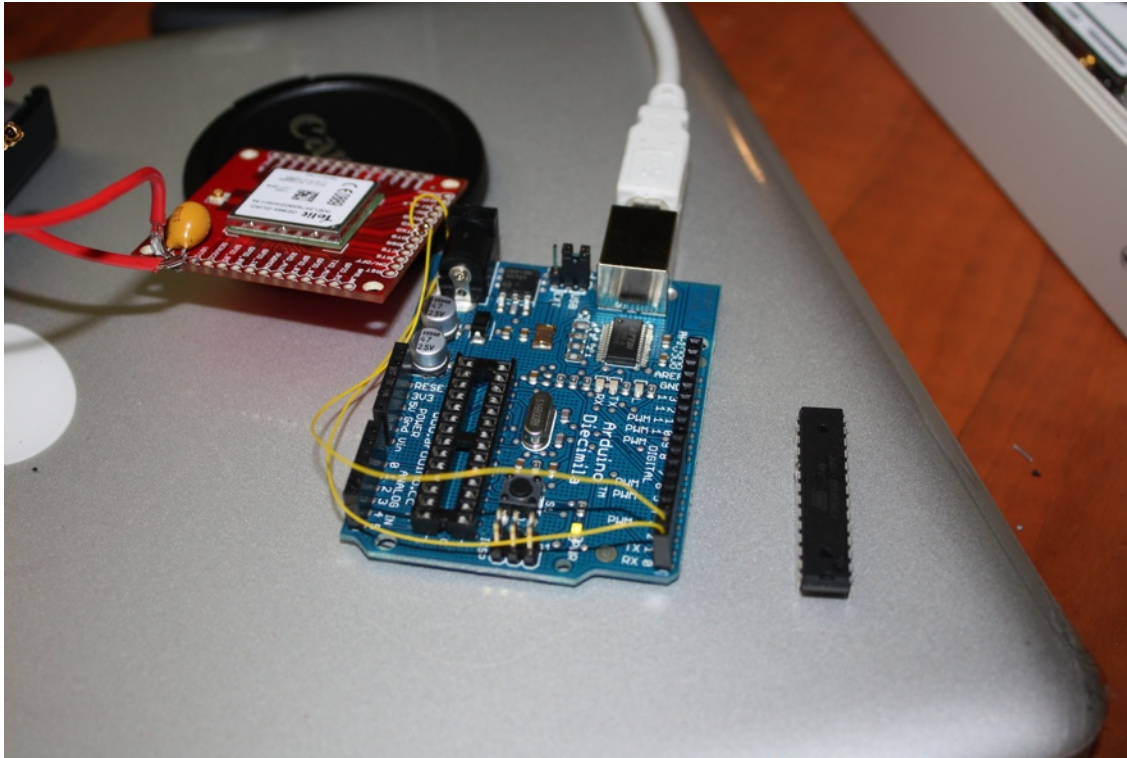
Battery and charging boards positioned inside the case.



Ground wire of GPS receiver connected to voltage output on the charging boards.



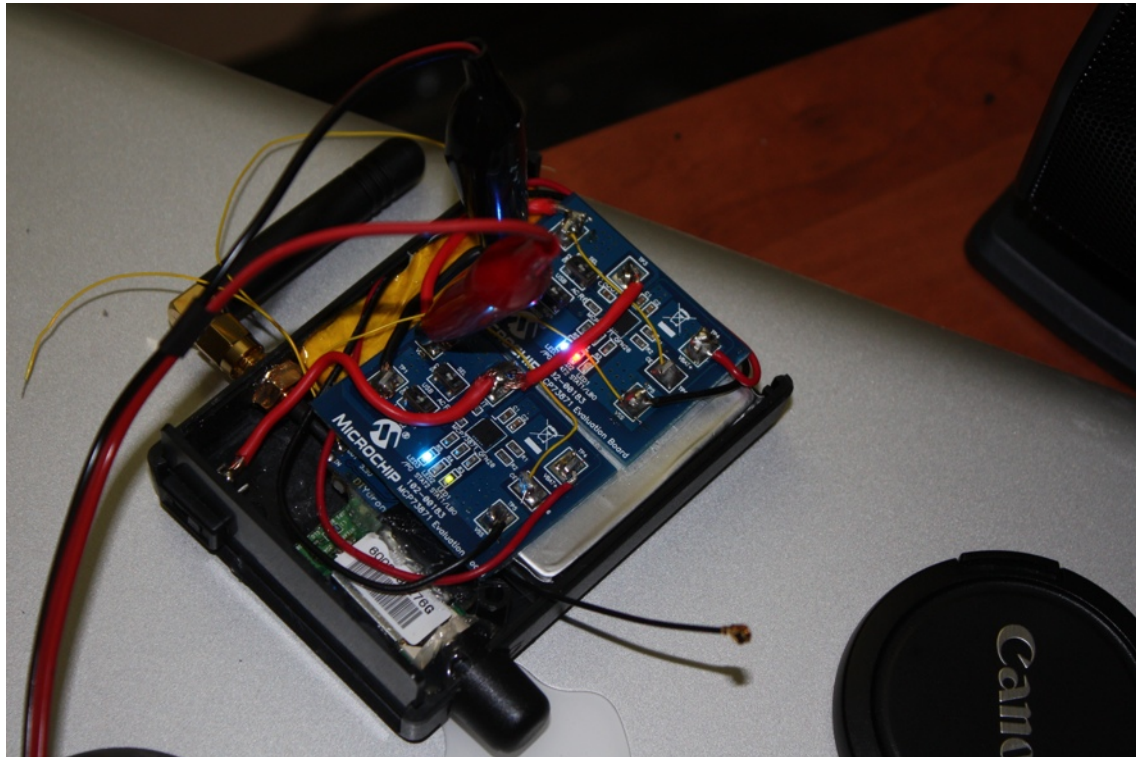
Telit GE865 breakout board with power wires and tantalum capacitor connected. Serial RX/TX pins connected to USB/Serial pins on Arduino development board (with USB cable connected).



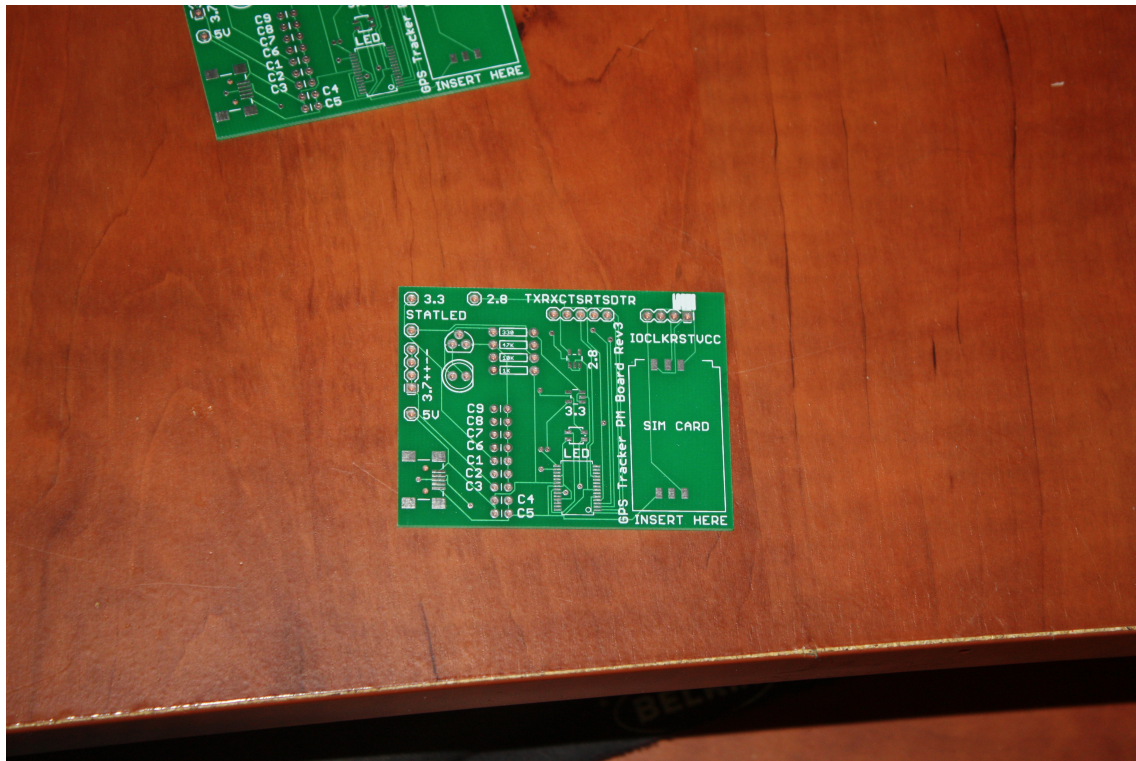
Multimeter, copper wire, and various tools used in the project.



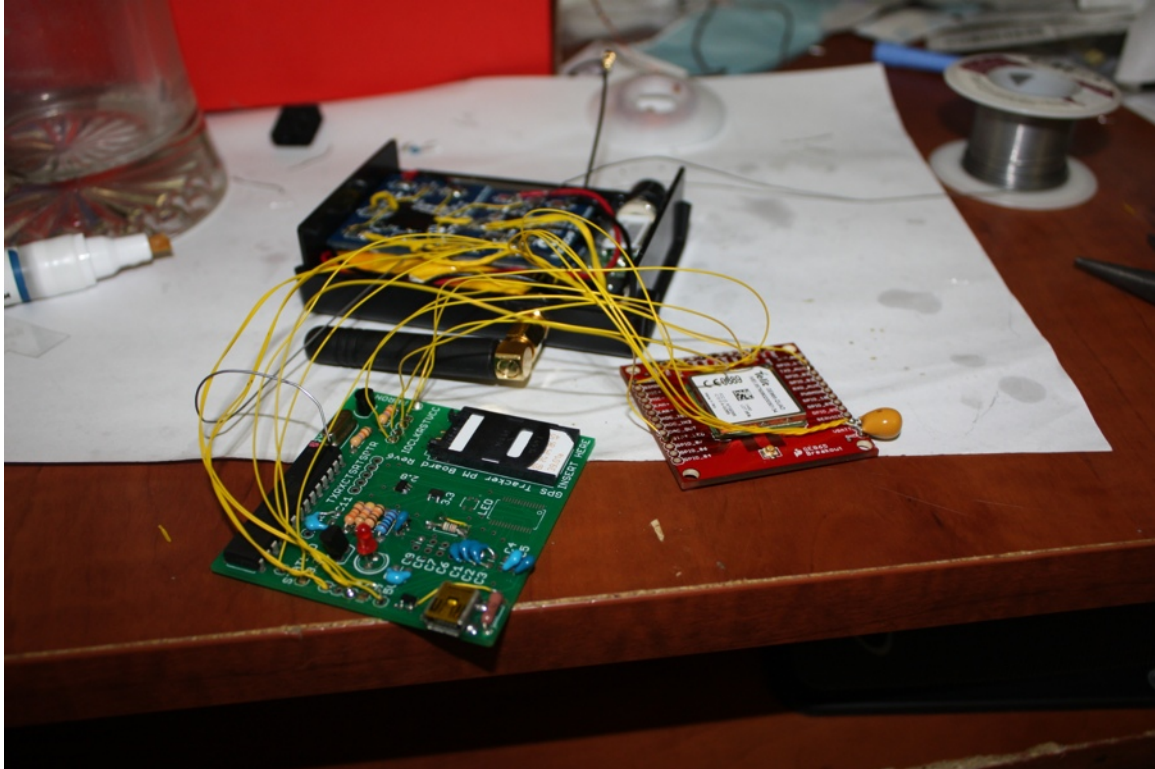
Testing both charging boards at once while mounted in the case.



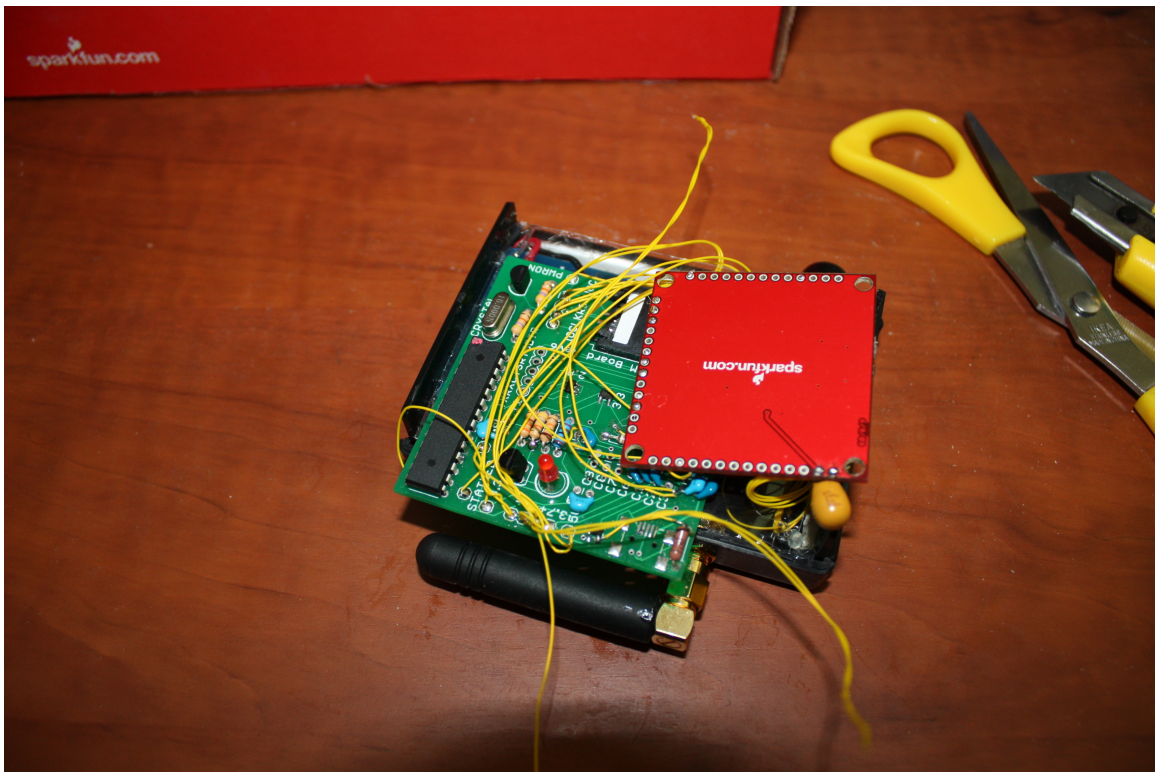
Circuit board fabricated by BatchPCB



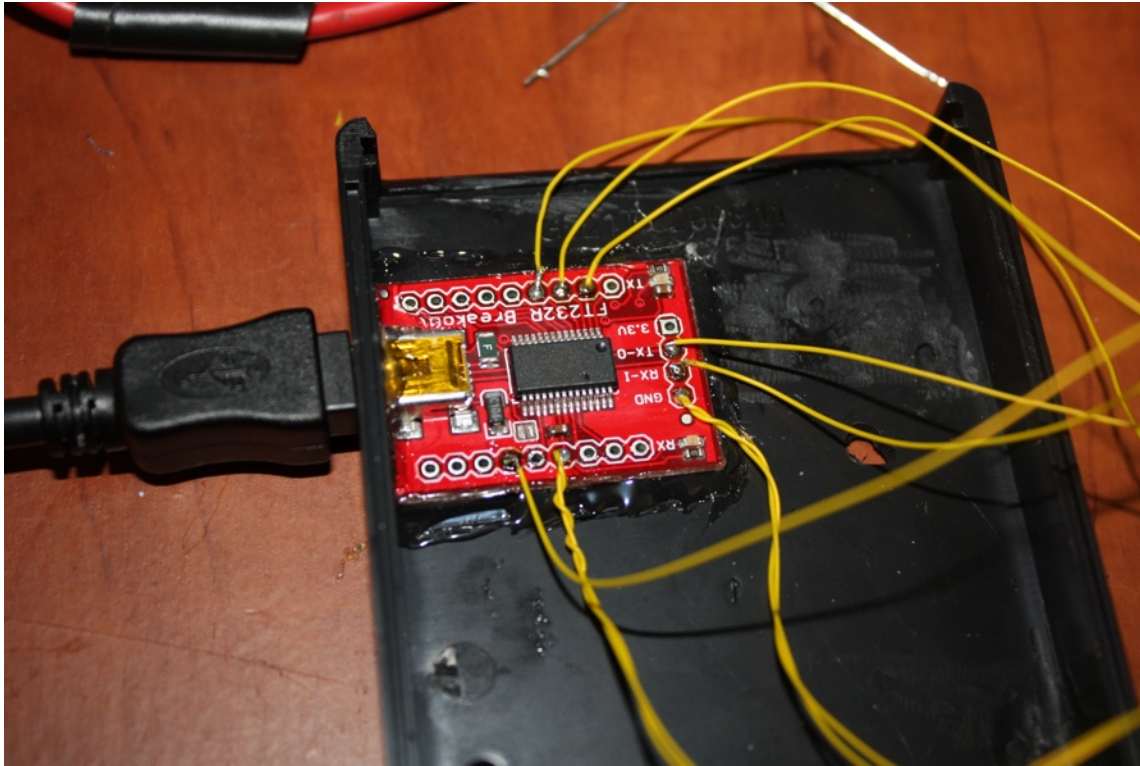
SIM card inserted into SIM card slot. Power wires connected. SIM card wires connected.



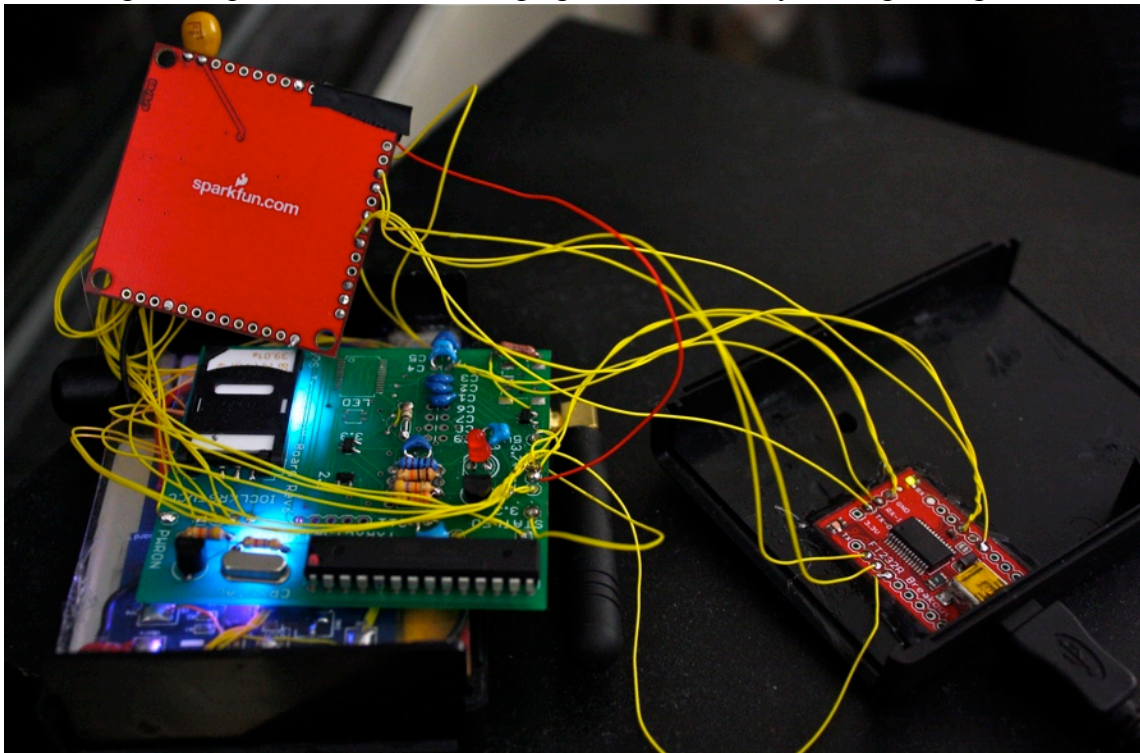
Beginning to align components together inside the case. (Telit GE865 turned upside down).



FT232R USB to Serial UART breakout board with pins soldered and mounted to top of the case. USB mini cable connected.



All wiring is complete. LEDs from charging board are visibly shining through the custom PCB.



Completed and fully assembled product.



Comparison in thickness between device and iPhone 3G.



Close-up shot completed device.



3.3 Design Errors and Workarounds

There were several design errors and workarounds.

- 1) GE865 maximum voltage is 4.3V, but system load sharing controller outputs 5V during charging. The solution was to refine the PCB design and add a third voltage regulator to regulate USB incoming voltage to 4.2V (which is the minimum voltage to charge the battery, but it still works)
- 2) GE865 does not start up if there is traffic on UART2 when as it is starting up with input voltages above 4V (rough estimate). This is documented in the data-sheet but is very unclear. The workaround to this is to disconnect the USB cable while starting the device. The GPS is constantly sending data on UART2. The solution of adding a load switch tied to a GPIO pin of the GE865 to be triggered after startup to power the GPS could be added into future PCB designs, but the workaround will suffice for the time being.
- 3) There was a short between ground and 2.8V out on the PCB. Clearly a design error. This was resolved in EAGLE and fixed in a future revision of the PCB (original rev was 3, final revision is 6).
- 4) The microchip battery charge controllers are powered in a way so that they are only active while the USB cable is connected, meaning that it does not show a low battery warning when running on battery. The solution could most likely be discovered by further research on the component, but is not needed for the scope of this report.
- 5) Measurements of the components were inaccurate (thickness) and the plastic case does not close completely. This is a human error but was not resolved due to cost constraints. Some electrical tape to seal the enclosure was an acceptable workaround.

4 Tracking Device Software Development

4.1 Software Architecture

The software on the tracking device is written in python and runs on the python interpreter on the GE865. Its purpose is to read and parse GPS data from the GPS receiver over the serial port, and to send that information to a remote server. It also performs some basic computation like calculating whether the tracking device is in motion, as to prevent unnecessary updates and to filter out errant GPS fixes. It is also responsible for any sort of power saving, and is capable of putting the tracking device to sleep. For the purpose of this project, the software is developed only to the point where it is capable of pseudo-realtime tracking with updates approximately every 5 seconds. Faster and slower update times will not be developed.

The programming paradigm used is procedural programming. There is minimal use of objects in the application. There is no end state, thus the application loops continuously until powered off. Aside from transmitting GPS co-ordinates, it is also responsible for initializing the GSM modem and configuring the GPRS context. The application is capable of basic recovery, and can re-initialize the connection if an error during data transmission is detected.

4.2 Development Environment

The development tools used for the tracking device software are:

- PythonWin for writing Python code for the module
- U-blox u-center for configuring GPS output format
- Round Solutions GSM terminal (serial terminal program with many predefined macros, used for uploading scripts to the GE865 module and testing)
- Netcat, used for testing network communication received from tracking device
- PuTTY, used as a serial console for long-term monitoring and logging

These tools are all freely available for Windows (Netcat was used in Linux, though it likely exists for Windows as well).

4.3 Software Functionality

The software begins by retrieving and storing the IMEI number with the command AT+CGSN. This number is used to identify the device in communications to the server. It then waits for a registration to a cellular network by waiting on valid output from AT+CREG. It then initializes the GPRS context using T-Mobile's APN. Once it has a valid IP address, it will establish either a UDP or TCP socket to the server, gpstrackeronline.com. The tracking device can either make use of the Java UDP server or the PHP TCP (HTTP) server. For the purpose of this report, we will be using the UDP server for all testing and evaluations.

Once there is a connection to the server, the software will go into a run loop where it sleeps for 4 seconds after every iteration. In each iteration, it will 1) Verify that it still has a connection to the server and re-initialize itself if it doesn't. 2) Parse the incoming data from the GPS into an array. 3) If the GPS string is valid (has a GPS fix), it will prepare an output string and send it to the server.

There are also pieces of logic that handle deciding whether the GPS tracker is moving or not, and various pieces of code dealing with changing at the rate which the LED flashes (as a status indicator) and a watchdog timer for rebooting the device in case of a fatal glitch.

A valid GPS string from the receiver is a proprietary u-blox message of type UBX,00. The message structure is \$PUBX,00,hhmmss.ss,Latitude,N,Longitude,E,AltRef,NavStat,Hacc,Vacc,SOG,COG,Vvel,ageC,HDOP,VDOP,TDOP,GU,RU,DR,*cs<CR><LF>. This is the data that we receive and parse from the receiver, paying close attention to 'NavStat' which tells us if the receiver has a fix, or not.

Field No.	Example	Format	Name	Unit	Description
0	\$PUBX	string	\$PUBX	-	Message ID, UBX protocol header, proprietary sentence
1	00	numeric	ID	-	Proprietary message identifier: 00
2	081350.00	hhmmss.sss	hhmmss.ss	-	UTC Time, Current time
3	4717.113210	ddmm.mmmm	Latitude	-	Latitude, Degrees + minutes, see Format description
4	N	character	N	-	N/S Indicator, N=north or S=south
5	00833.915187	dddmm.mmmm	Longitude	-	Longitude, Degrees + minutes, see Format description
6	E	character	E	-	E/W indicator, E=east or W=west
7	546.589	numeric	AltRef	m	Altitude above user datum ellipsoid.
8	G3	string	NavStat	-	Navigation Status, See Table below
9	2.1	numeric	Hacc	m	Horizontal accuracy estimate.
10	2.0	numeric	Vacc	m	Vertical accuracy estimate.
11	0.007	numeric	SOG	km/h	Speed over ground
12	77.52	numeric	COG	degrees	Course over ground
13	0.007	numeric	Vvel	m/s	Vertical velocity, positive=downwards
14	-	numeric	ageC	s	Age of most recent DGPS corrections, empty = none available

Field No.	Example	Format	Name	Unit	Description
15	0.92	numeric	HDOP	-	HDOP, Horizontal Dilution of Precision
16	1.19	numeric	VDOP	-	VDOP, Vertical Dilution of Precision
17	0.77	numeric	TDOP	-	TDOP, Time Dilution of Precision
18	9	numeric	GU	-	Number of GPS satellites used in the navigation solution
19	0	numeric	RU	-	Number of GLONASS satellites used in the navigation solution
20	0	numeric	DR	-	DR used
21	*5B	hexadecimal	cs	-	Checksum
22	-	character	<CR><LF>	-	Carriage Return and Line Feed

Table Navigation Status

Navigation Status	Description
NF	No Fix
DR	Dead Reckoning only solution
G2	Stand alone 2D solution
G3	Stand alone 3D solution
D2	Differential 2D solution
D3	Differential 3D solution
RK	Combined GPS + Dead Reckoning solution
TT	Time only solution

The string sent from the tracking device to the server is formatted as follows:

IMEI,LAT,E/W,LON,N/
S,SATS,HACC,VACC,ALT,GROUNDSPEED,BEARING,VSPEED,MODE

Specifically, these correspond to the following:

IMEI = IMEI Number of tracking device

LAT = Latitude

E/W = East/West (latitude)

LON = Longitude

N/S = North/South (longitude)

SATS = Number of satellites used in GPS fix

HACC = Horizontal accuracy

VACC = Vertical accuracy

ALT = Altitude

GROUNDSPEED = Ground speed (km/h)

BEARING = Direction (degrees)

VSPEED = Vertical speed

MODE = Tracking device movement status: Stationary (ST), Moving (MV), No Change from last update (NC), Update last status with a more accurate fix (UT)

4.4 Power-Saving Considerations

The primary power-saving consideration that has been made is to sleep the device every 4 seconds and to use UDP instead of TCP. A future enhancement could have been powering the GPS receiver via a load switch and being able to toggle power to it via a GPIO pin. That way the device would be able to efficiently support extended tracking operations, for example, by powering itself down (including the GPS) for 20 minutes at a time instead of merely a few seconds.

4.5 Platform Limitations

There are two primary limitations with this platform. Firstly, it does not support floating point math. This is a significant limitation because GPS co-ordinates by nature contain floating decimal points. As well, various values like speed, accuracy, etc contain floating-point decimals. Workarounds were found to avoid floating-point math.

The second limitation is memory. There is a limit to string variable sizes which make it prohibitively difficult to do significantly large I/O from the network. The APIs do not lend themselves well to reading data into a buffer or an array, further complicating any large data transfer. This made it prohibitively difficult to implement AGPS (augmented GPS, where the position/map of the GPS satellites and basic timing information is read from a network server to greatly decrease the amount of time it takes to get a GPS fix).

4.6 Atmel Microcontroller Software

The software for the Atmel microcontroller is very straight forward. Originally this device was going to support more complex features like interrupts to detect if the GE865 module had abnormally shut down to reboot it, etc. However, the purpose of this device is very simple in the final implementation. On startup, it pulls the GE865 PWR_ON pin HIGH for 3 seconds to turn on the device [3]. It then shuts itself off [7].

5 Tracking Device Software Testing

There are a variety of conditions which were tested. The following are various conditions which were tested, including brief descriptions and log excerpts.

5.1 No GSM Signal at Initialization

Script starts...

```
AT#EXECSCR  
OK  
TEST5
```

One or more loops of 'waiting for GSM CREG'

Waiting for GSM CREG

When successful:

```
+CREG: 0,5
```

```
OK
```

Usually followed by a few of these:

```
ERROR SETTING GPRS CONTEXT  
CREG:  
+CREG: 0,5
```

Finally:

```
OK  
UDP_CONNECTED
```

5.2 No GPS Signal at Initialization

Infinite repetitions of similar strings to the following, until the status 'NF' which stands for No Fix changes to something else:

```
[$PUBX,'00','000154.00','0000.00000','N','00000.00000','E','0.000','NF','6495192',
'4592794','0.000','0.00','0.000',' ','99.99','99.99','99.99','0','0','0*28\015']
```

2

21

```
[$PUBX,'00','000159.00','0000.00000','N','00000.00000','E','0.000','NF','6495192',
'4592794','0.000','0.00','0.000',' ','99.99','99.99','99.99','0','0','0*25\015']
```

2

21

```
[$PUBX,'00','000204.00','0000.00000','N','00000.00000','E','0.000','NF','6495192',
'4592794','0.000','0.00','0.000',' ','99.99','99.99','99.99','0','0','0*2E\015']
```

2

21

5.3 Normal Initialization and Successful Run Loops

Script starts..

AT#EXECSCR

OK

TEST5

GSM connection

Waiting for GSM CREG

+CREG: 0,5

OK

OK

UDP_CONNECTED

Valid GPS Signal..

```
$PUBX,00,234218.00,4521.37123,N,07539.09890,W,
112.802,G3,51,15,1.890,0.00,0.000,,13.33,7.16,13.43,5,0,0*70
```

Parsed GPS String into array...

```
['$PUBX', '00', '234218.00', '4521.37123', 'N', '07539.09890', 'W', '112.802', 'G3', '51', '15', '1.890', '0.00', '0.000', ',', '13.33', '7.16', '13.43', '5', '0', '0*70\015']
```

0

Output string to server...

```
357938020050134,4521.37123,N,07539.09890,W,5,51,15,112.802,1.890,0.00,0.000,ST
```

Going to sleep...

```
LOOP PASS
```

```
$PUBX,00,234235.00,4521.39570,N,07539.08893,W,121.230,G3,48,13,2.263,0.00,0.000,,13.54,7.27,13.65,5,0,0*76
```

```
['$PUBX', '00', '234235.00', '4521.39570', 'N', '07539.08893', 'W', '121.230', 'G3', '48', '13', '2.263', '0.00', '0.000', ',', '13.54', '7.27', '13.65', '5', '0', '0*76\015']
```

0

```
357938020050134,4521.39570,N,07539.08893,W,5,48,13,121.230,2.263,0.00,0.000,NC
```

```
LOOP PASS
```

5.4 Unexpected Loss of GPS Signal During Run Loop***Good signal.....***

```
357938020050134,4521.37926,N,07539.06044,W,5,47,12,159.601,2.370,0.00,0.000,NC
```

```
LOOP PASS
```

```
$PUBX,00,234304.00,4521.37619,N,07539.06744,W,169.462,NF,67,13,9.839,0.00,0.000,,72.92,19.82,73.15,5,0,0*32
```

```
['$PUBX', '00', '234304.00', '4521.37619', 'N', '07539.06744', 'W', '169.462', 'NF', '67', '13', '9.839', '0.00', '0.000', ',', '72.92', '19.82', '73.15', '5', '0', '0*32\015']
```

0

Signal lost....

```
['$PUBX', '00', '234310.00', '4521.37493', 'N', '07539.09644', 'W', '173.441', 'NF', '91', '13', '16.294', '0.00', '0.001', ',', '73.67', '20.05', '73.90', '5', '0', '0*0B\015']
```

2

21

Back to good signal....

```
$PUBX,00,234243.00,4521.38176,N,07539.07890,W,
```

```
133.087,G3,46,12,3.847,0.00,0.000,,13.64,7.32,13.76,5,0,0*72
```

```
['$PUBX', '00', '234243.00', '4521.38176', 'N', '07539.07890', 'W', '133.087', 'G3', '46', '12', '3.847', '0.00', '0.000', ',', '13.64', '7.32', '13.76', '5', '0', '0*72\015']
```

0

```
357938020050134,4521.38176,N,07539.07890,W,5,46,12,133.087,3.847,0.00,0.000,MV
```

```
LOOP PASS
```

5.5 Unexpected Loss of GSM Signal During Run Loop

21

```
['$PUBX', '00', '234213.00', '4521.37270', 'N', '07539.10840', 'W', '102.558', 'G3', '53', '15',
'5.737', '0.00', '0.000', ',', '13.26', '7.13', '13.36', '5', '0', '0*7F\015']
```

2

21

```
357938020050134,4521.37270,N,07539.10840,W,5,53,15,102.558,5.737,0.00,0.000,MV
```

LOOP PASS

ERROR ESTABLISHING SSEND CONTEXT

CREG:

+CREG: 0,5

OK

ERROR WITH GPRS INIT

CREG:

+CREG: 0,5

OK

```
$PUBX,00,234218.00,4521.37123,N,07539.09890,W,
```

```
112.802,G3,51,15,1.890,0.00,0.000,,13.33,7.16,13.43,5,0,0*70
```

```
['$PUBX', '00', '234218.00', '4521.37123', 'N', '07539.09890', 'W', '112.802', 'G3', '51', '15',
'1.890', '0.00', '0.000', ',', '13.33', '7.16', '13.43', '5', '0', '0*70\015']
```

0

```
357938020050134,4521.37123,N,07539.09890,W,5,51,15,112.802,1.890,0.00,0.000,ST
```

LOOP PASS

6 Application Server Development

There are two application servers. The primary server, used throughout this report and during testing, is the Java UDP server. This server receives strings from tracking device(s) over a raw UDP port and records them in a MySQL database with some manipulation. The PHP server receives strings as POST data from the tracking device in an HTTP request and performs the exact same manipulation and recording into a MySQL database as the Java version. The only difference is the transport layer. If the project was to be expanded to allow two-way communications, it would be easier to develop that in the PHP server. However, the UDP server been found to be much more reliable, due to the inherent complexities in maintaining a reliable TCP connection on 2G cellular networks. The application can be accessed via **REDACTED** at the server: **REDACTED**. The username is **REDACTED** and the password is **REDACTED**. Once connected, the command 'screen -r' will display the server console. The MySQL database can be accessed via **REDACTED** at **REDACTED**. The username is **REDACTED** and the password is **REDACTED**

The only logic per se in the servers is the math required to convert GPS co-ordinates into standard decimal latitudes and longitudes.

For example, here is a set of GPS co-ordinates sent from the tracking device for a location in Ottawa, Ontario. "4521.4014,N,07539.0934,W". If one was to paste this string in to google maps (or most other geolocation applications), it would be unrecognizable. These values are in Degree Minute Seconds format, and we must convert it into a purely decimal degrees format [4].

For the latitude, we do:

$$4521.4014 / 100 = 45.214014. \text{ This gives us our degrees (45)}$$

Now that we have our degree amount, we must convert the minutes and seconds to a fraction of a degree.

$$(.214014 * 100) / 60 = 0.35669 \text{ (Decimal value of minute-seconds)}$$

Add the decimal value of the minute-seconds to our degrees to get final co-ordinate value

$$0.35669 + 45 = 45.35669.$$

Finally, if the co-ordinate represented a South or West direction, multiply by -1.

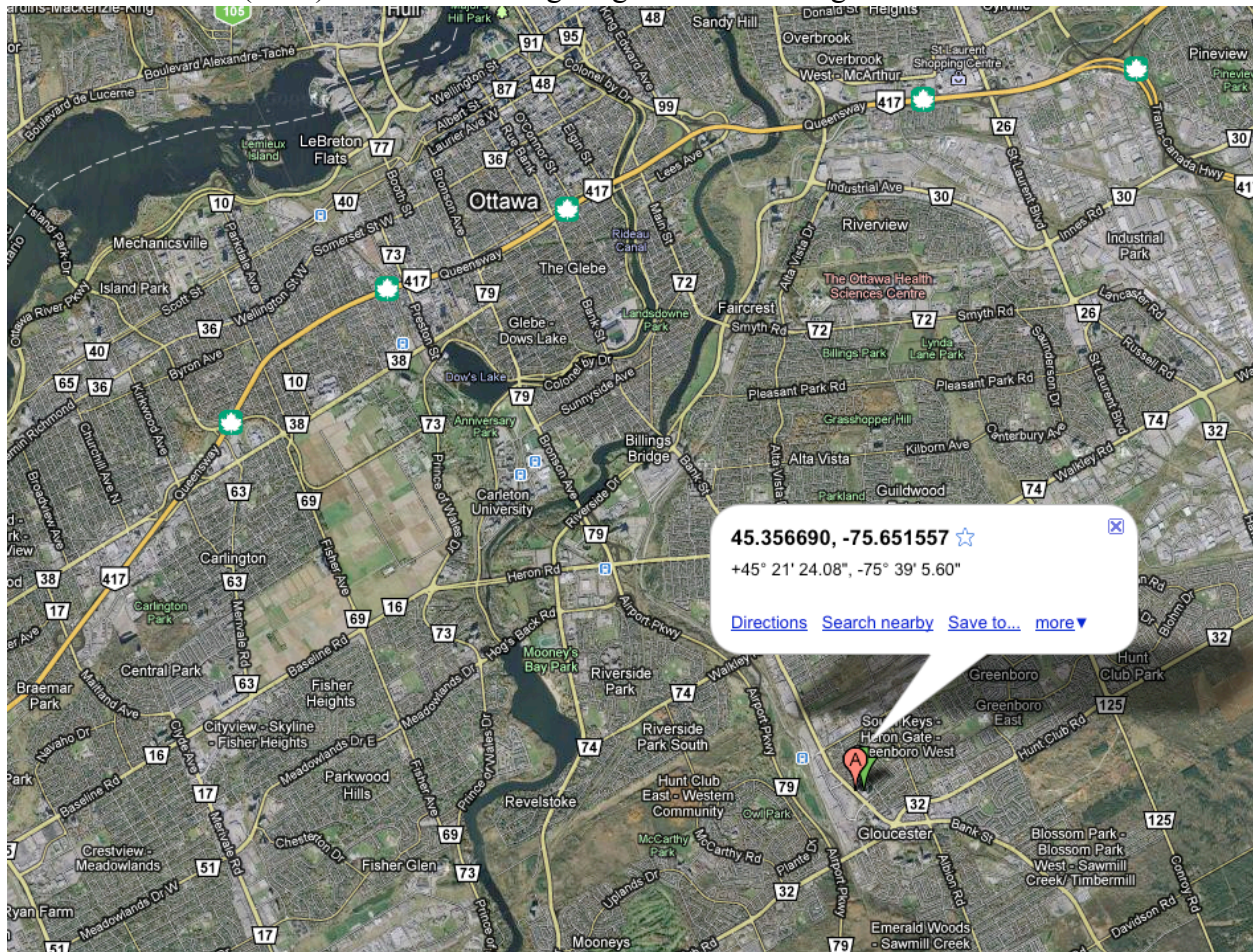
For longitude, we have:

$$07539.0934 / 100 = 75 \text{ and } .390934$$

$$(.390934 * 100) / 60 = 0.65155667$$

$$75 + 0.65155667 = 75.65155667$$

75.65155667 * -1 (West) = -75.65155667 giving us our final longitude



7 Google Maps Front-End Service

7.1 Login

The login page (index.php) is a very simple PHP script that generates a HTTP GET request to tracker.php for valid IMEI numbers. It takes an IMEI number of your device as input and sends you to tracker.php?imei=yourIMEInumber.

There is also a link to register (register.php) a new IMEI with the server. Once an IMEI is registered, the server will store any data from that IMEI into the database, otherwise it discards data from any unknown IMEI numbers.

From a security perspective, there is nothing inherently secure about this. However, this project is at a proof-of-concept stage only, and a final product would obviously require user management and enhanced security.

7.2 View

The view is generated by tracker.php. Tracker.php consists of some javascript: Google Maps V2 AJAX APIs in oldUtil.js, Google Maps V3 API from maps.google.com/maps/api/js, and the javascript code written for this project directly in tracker.php [5]. The rest of the file is PHP.

The code starts off by initializing the various icons (red, blue, orange, yellow) and a shadow. From there, it initializes a database connection and centers the map on the most recent co-ordinate in the database.

From there, the code is very simple. A GET Request string is generated every time you hit refresh (or every X seconds where X is the autorefresh interval). This string passes all of the client settings to the XML generator which generates XML data by querying the database for the desired co-ordinates. When this XML data is received, the map overlay is cleared and regenerated based on this data.

An info view is also configured for each co-ordinate so that details can be seen by clicking on a point. This is the HTML: "Latitude: " + lat + "
Longitude: " + lon + "
Altitude: " + alt + "m MSL" + "
Time Start: " + timestamp + "
Time End: " + timestamp2 + "
Horizontal Accuracy: " + hacc + "
Vertical Accuracy: " + vacc + "
 Sats: " + sats + "
Ground speed: " + speed + " km/h
Bearing: " + bearing + " degrees
Vertical speed: " + vspeed + " m/s
";

There is also a PHP class from Anthony Hand included (<http://www.mobileesp.com>) to detect whether the device is a mobile device or a full-featured PC [6].

The XML generator used from sending data to the view from the database via AJAX is called ajaxXmlGen.php.

It consists of a very simple distance algorithm and some SELECT queries which will not be explained in any further details because they are very straightforward.

7.3 Database Schema

Server: **REDACTED**

Username: **REDACTED**

Password: **REDACTED**

The database can be viewed from **REDACTED** with that username and password.

There is a table for each IMEI with the following schema:

```
CREATE TABLE IF NOT EXISTS `IMEINUMBER` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `lat` double NOT NULL,
  `lon` double NOT NULL,
  `timestamp` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `sats` tinyint(4) NOT NULL,
  `hacc` double NOT NULL DEFAULT '0',
  `vacc` double NOT NULL DEFAULT '0',
  `msl` double NOT NULL,
  `speed` double NOT NULL DEFAULT '0',
  `bearing` double NOT NULL DEFAULT '0',
  `vspeed` double NOT NULL DEFAULT '0',
  `mode` varchar(2) NOT NULL DEFAULT 'MV',
  `timestamp2` timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=32949 ;
```

Most of these columns are self-explanatory from earlier sections in the report, with the exception of timestamp and timestamp2. Timestamp is the original timestamp when the data was record, timestamp2 was the last time that the co-ordinates were updated (if a more accurate fix for a position without the device moving was detected).

8 Usage Overview and Procedures

There is some basic information which must be discussed first before showing screenshots and guides.

- 1) The website is www.gpstrackeronline.com, as mentioned in the previous section.
- 2) The IMEI number of the demo device is **REDACTED**. You can also use the short-name mntest2 which has been specially linked inside the code to map to the IMEI number. It is case sensitive, so Mntest2 or other variants will not work.
- 3) There are 4 different GPS fixes supported by the view. A red icon indicates the latest position IF the device is moving. An orange icon indicates the latest position IF the device is stationary. A yellow icon indicates a previous stationary fix (meaning, at this point, the device was non-moving). A blue icon indicates a previous moving fix (meaning, at this point, the device was moving).
- 3) GPS receivers are bad at tracking stationary objects. There will be lots of errant GPS fixes if the device is left stationary over a long period of time. These can be filtered out by increasing the minimum number of GPS satellites required for a fix, as to only allow the most accurate data. You can also enable stationary fixes only, and this will greatly reduce (but not completely get rid of) errant GPS fixes.
- 4) Clicking on a point on most mobile devices will not allow you to see the entire info window, some data will be truncated without scrolling being offered. This is a known bug with the Google Maps API. On mobile devices, the latitude and longitude are not at the top, instead the ground speed and the bearing are displayed.
- 5) This project, especially when displaying more than a few hundred points, makes use of intensive javascript. This project is best viewed in Google Chrome, or any other high-performance javascript browser. This does NOT include Mozilla Firefox or Internet Explorer.
- 6) The LED on the device is VERY important. It is the only status indicator available without being connected to the device via a USB cable.

Half-second ON, Half-second OFF (medium speed blinking): Device starting up, python code NOT running.

Two seconds ON, One second OFF (very slow blinking): Initializing GSM/GPRS connection (associating with cell tower, establishing a socket with server)

0.3 seconds ON, 1.2 seconds OFF (slow blinking): Device started up, GSM/GPRS connection active. No GPS fix. Will begin transmitting as soon as there is a GPS fix.

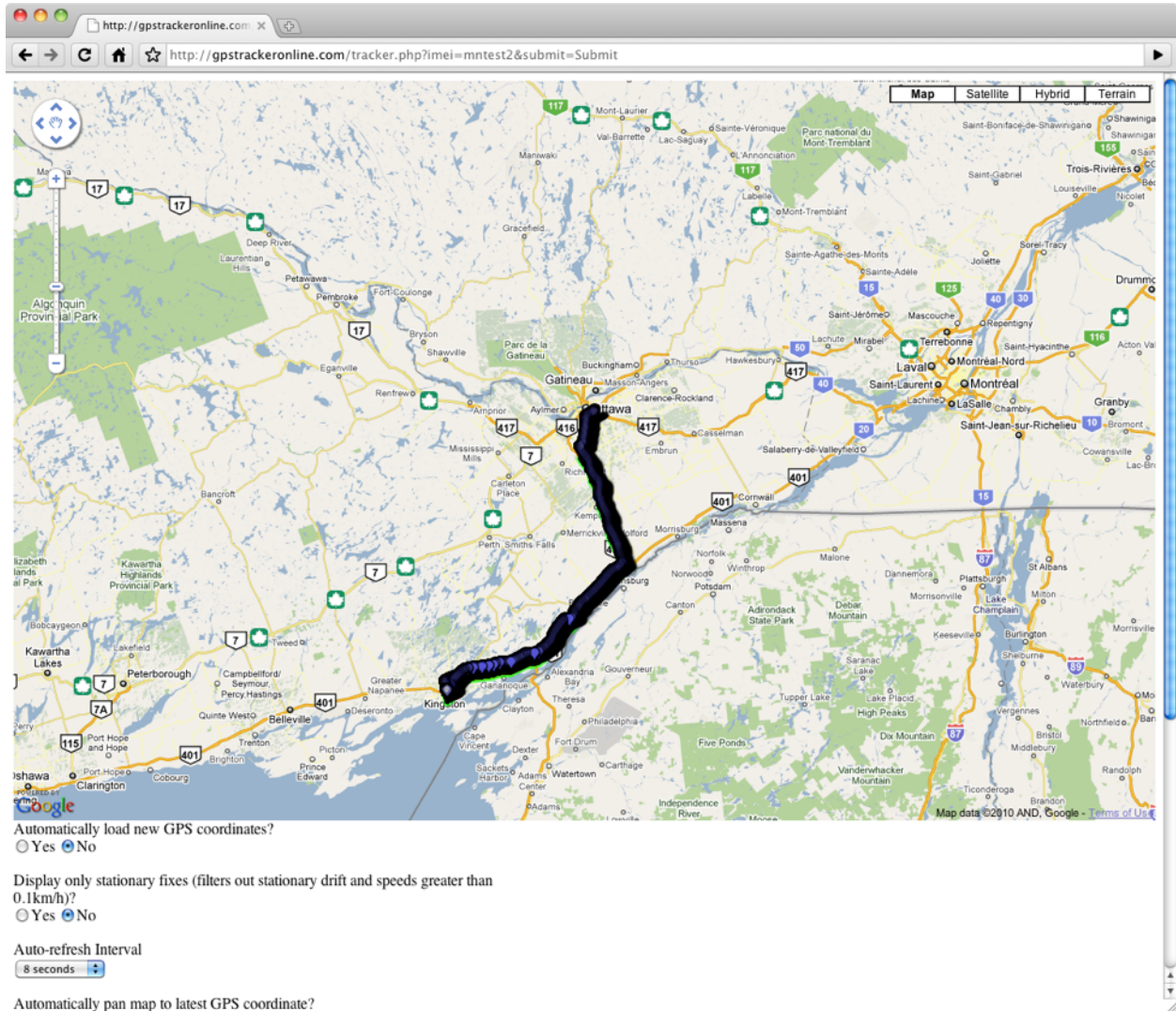
0.1 seconds ON, 0.1 seconds OFF (very fast blinking): Device transmitting valid GPS co-ordinates to server. This is the ideal state. The LED remains in this state even if the device sleeps momentarily between transmitting co-ordinates. It only leaves this state if an error is encountered or the GPS fix is lost.

The following guide will not discuss the use of the “Positions for the last...” feature, as this is time-sensitive and the test data collected is older than the available ranges. Please see the System Testing section for use of this feature.

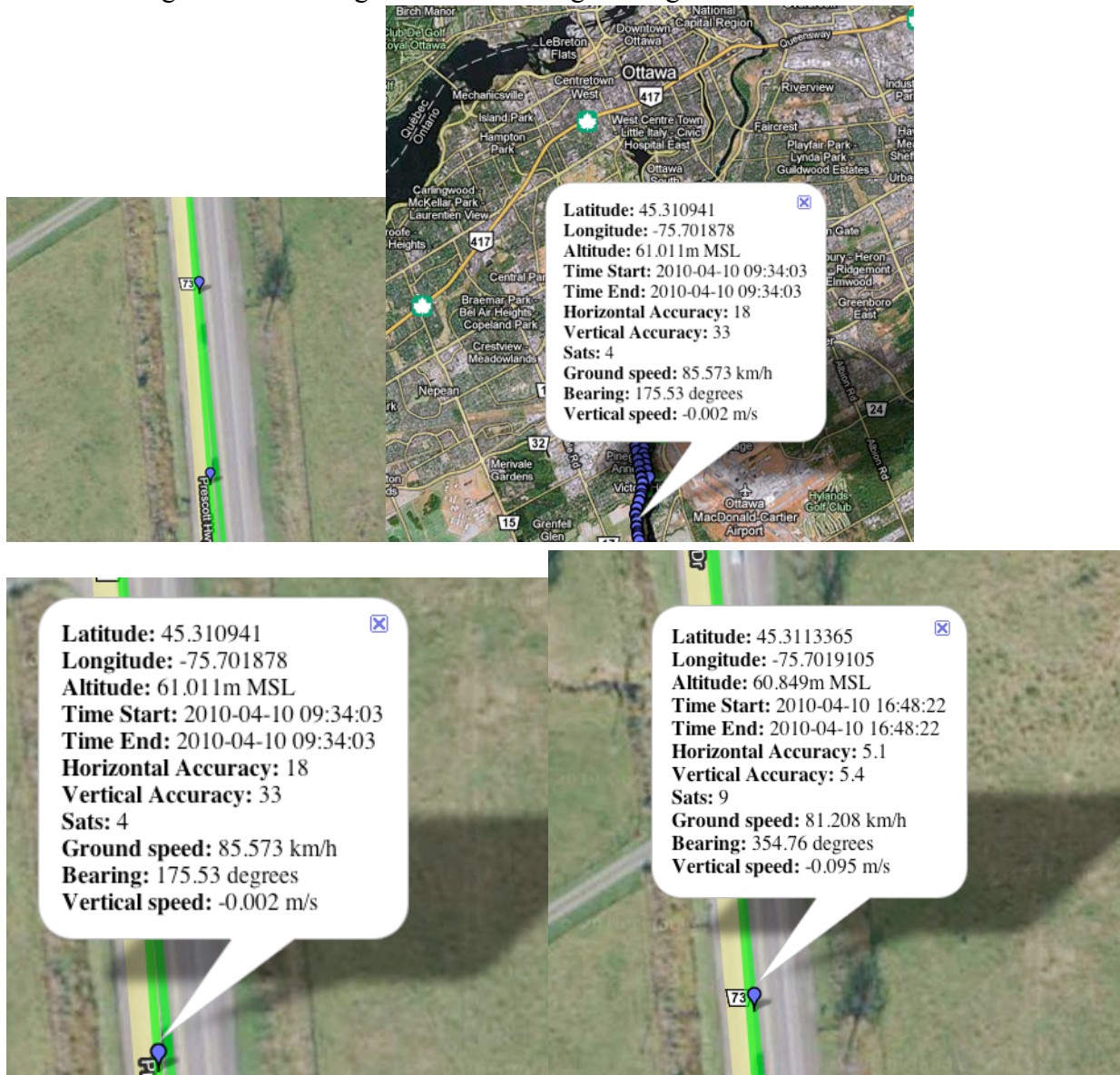
The following guide shows test data collected on a road trip to Kingston, Ontario, from Ottawa.

The interface has been set to display the last 3000 positions view the “Show last X data points” dropdown menu.

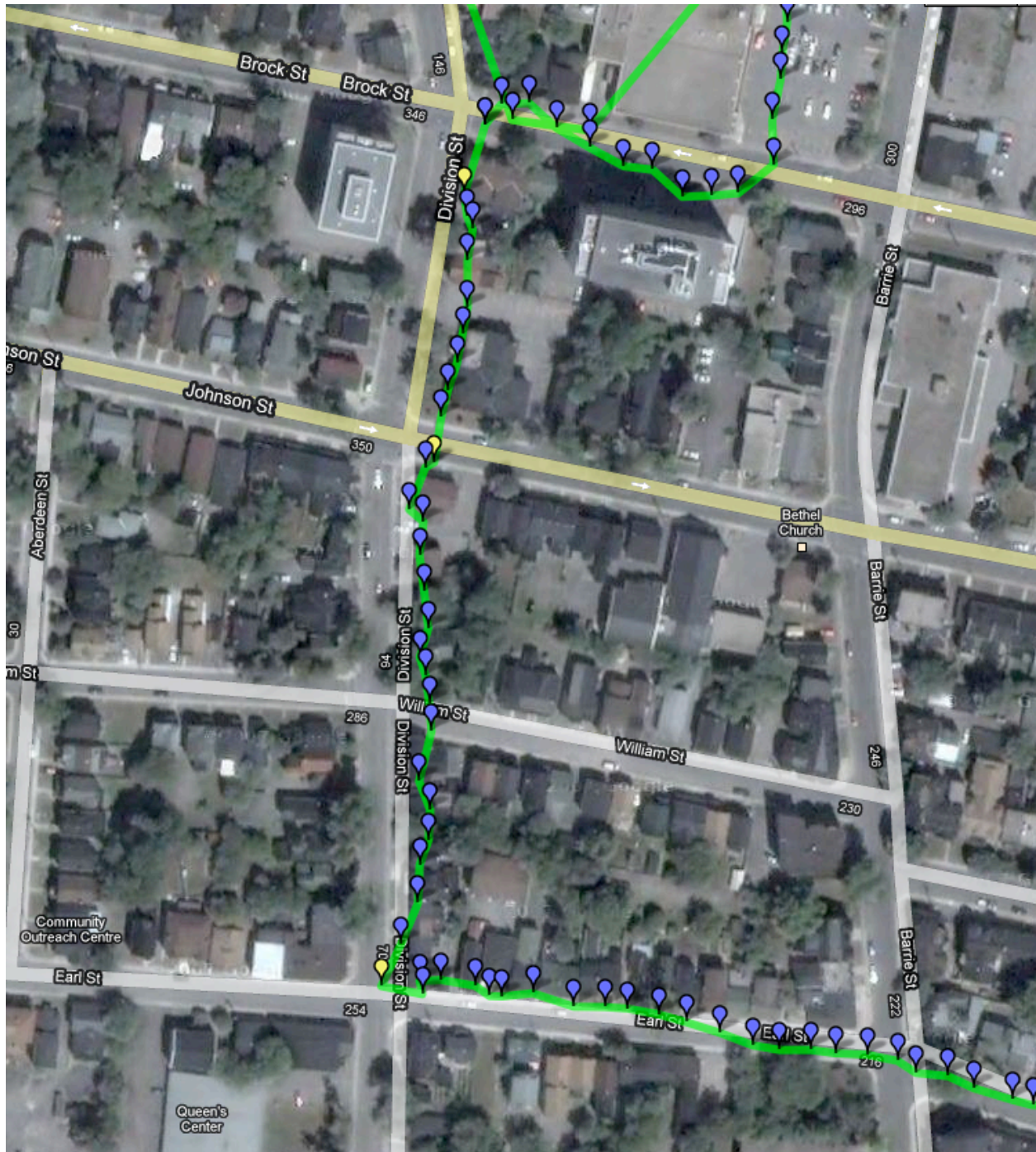
Here we can see a zoomed out view of the entire trip:



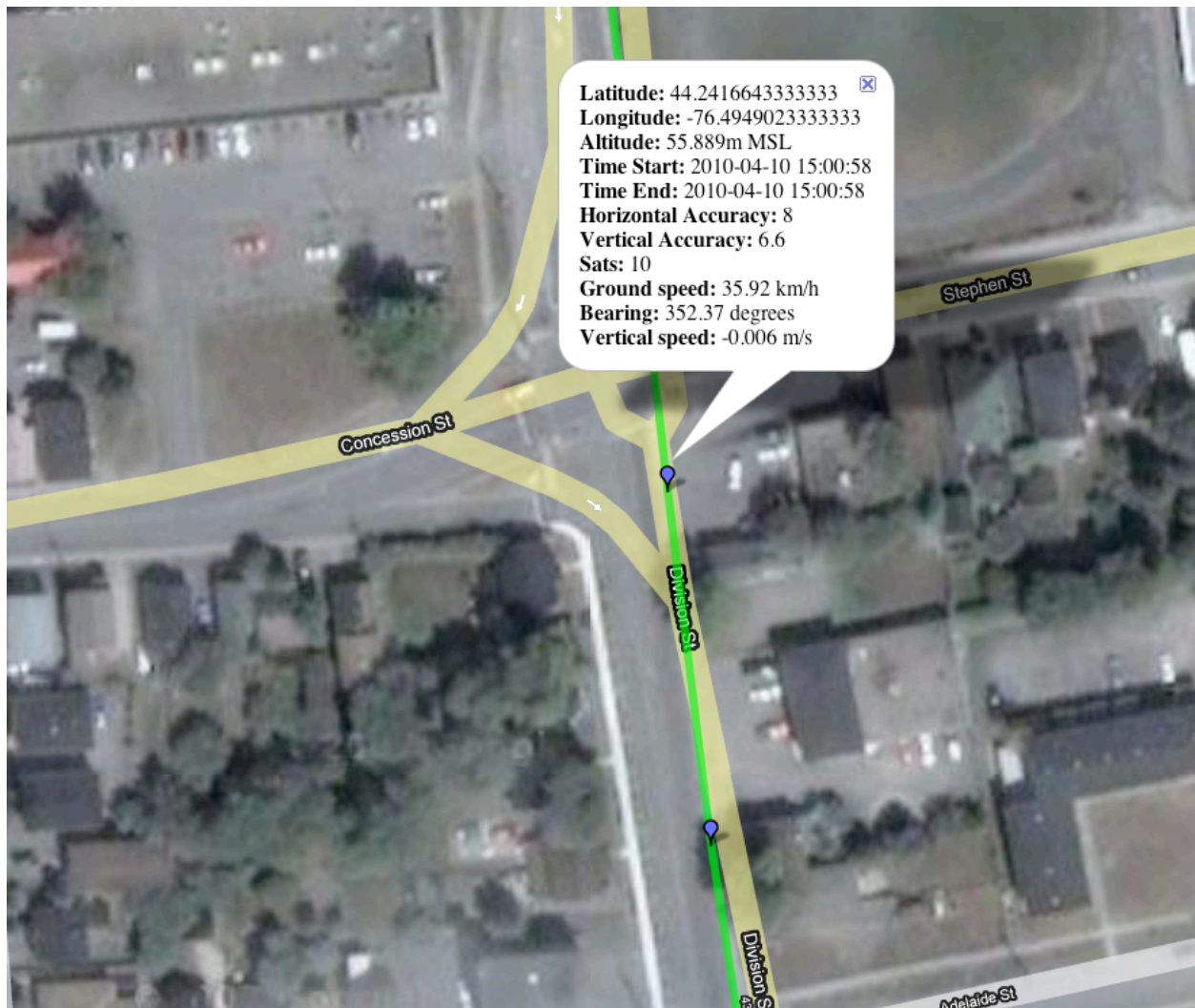
If we zoom in, we can see approximately when the tracking device was turned on. Notice the two sets of lines and how the points are on alternating sides of the road, we can see our data points from returning to Ottawa alongside when heading to Kingston.



If we zoom in on Kingston, we can inspect the various types of GPS fixes. Note that there are yellow points near some of the intersections. These are stationary fixes while walking in Kingston. It is plausible that a person would be stationary momentarily near an intersection.

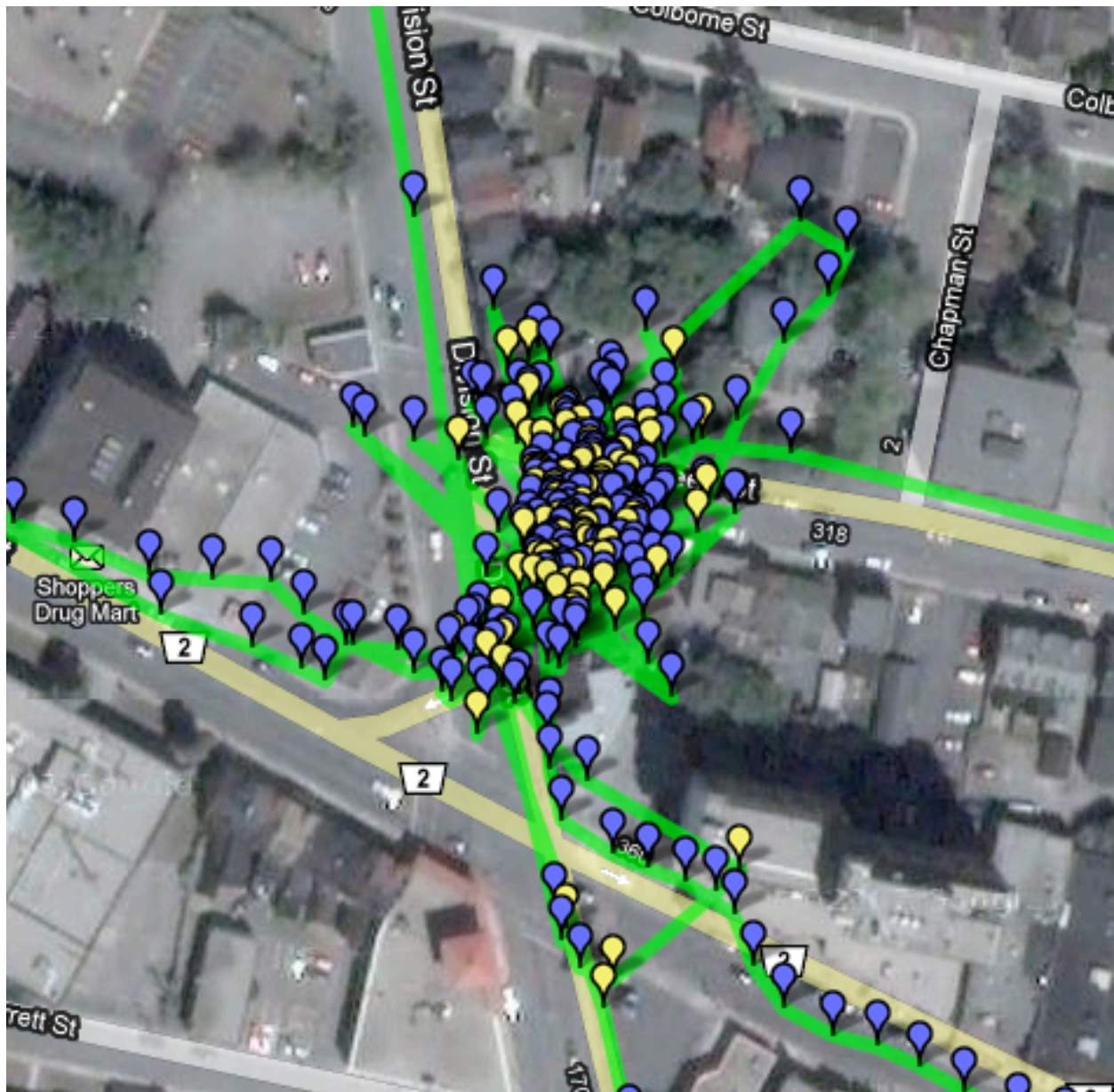


If we zoom in on a position while driving, we can see that the speed and direction are very accurate. Keep in mind that this is a residential street (40km/h speed limit) and that Google Maps is aligned north at the top of the screen.



Our bearing of 352.37 degrees (N=0/360, E = 90, S = 180, W = 270) is very accurate judging from the map. Our speed of 35.92km/h seems plausible.

A good example of errant GPS data and how to filter it is by zooming in on a period of time (lunch break) while the tracker was stationary:



In order to clean up our data to get a better impression of the event timeline, we will display only stationary fixes, and turn up the tolerance to 12, and set the minimum number of satellites for a fix to 5. We can see that the data is much cleaner, though still not flawless. However, it is much more readable now.



9 System Testing and Performance Results

9.1 Battery Life and Time to Transmit

Time to start up and begin transmitting co-ordinates while outdoors (cold start): **47 seconds**.

Time to start up and begin transmitting co-ordinates while outdoors (warm start, some GPS data still cached in RAM): **25 seconds**

Battery life while transmitting non-stop at 5 second intervals: **~30-36 hours (~1.2-1.5 days)**

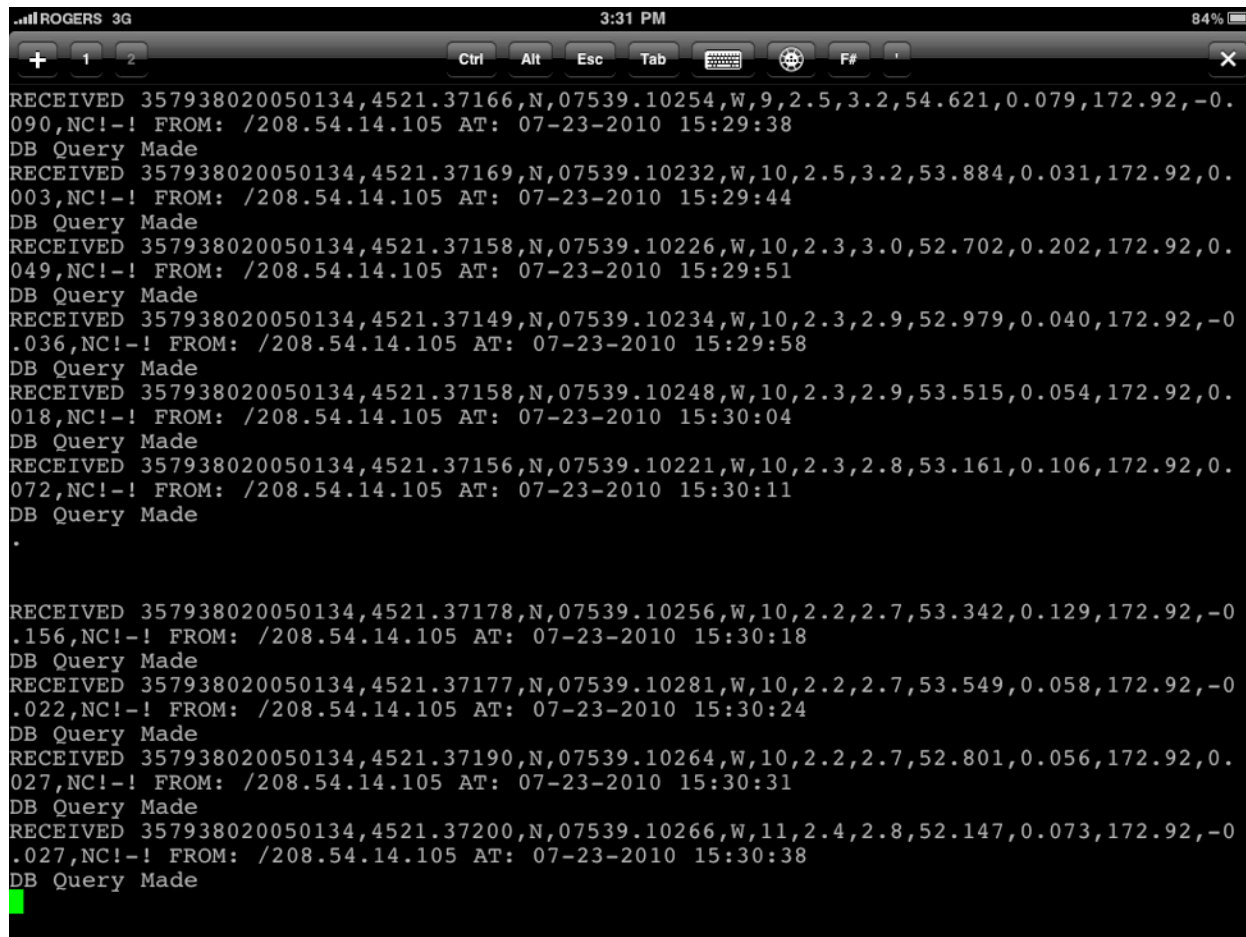
9.2 Accuracy Testing

An on-foot accuracy test of the tracker was performed with the assistance of a digital camera, an Apple iPad, and Google Maps. The iPad has a built-in GPS receiver that was used for real-time comparison.

Device sitting outdoors on a bus bench:

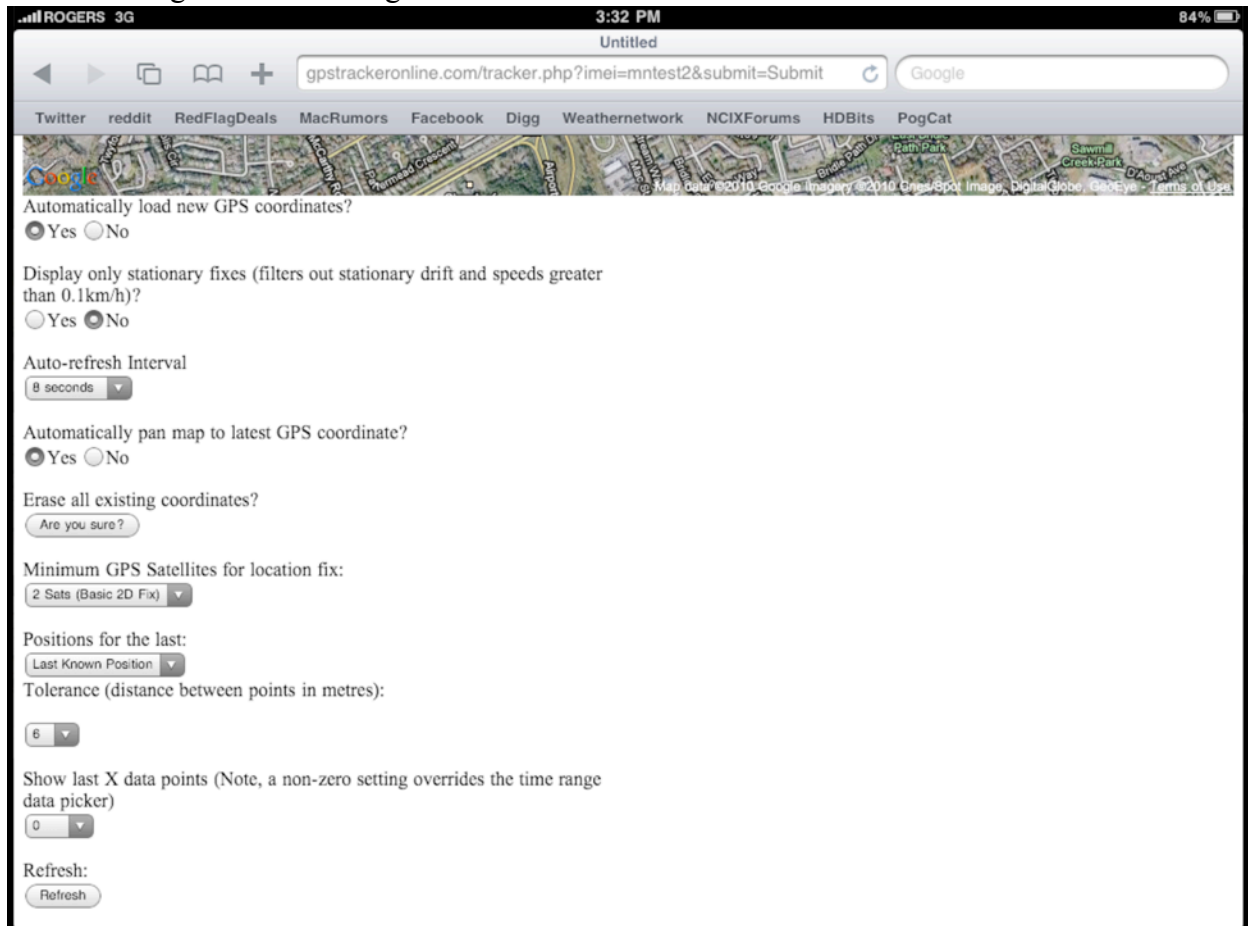


SSH Console on the iPad when the device was turned on (note the timestamps at the end of every line, July 23, 2010)



```
ROGERS 3G 3:31 PM 84%
+ 1 2 Ctrl Alt Esc Tab F#
RECEIVED 357938020050134,4521.37166,N,07539.10254,W,9,2.5,3.2,54.621,0.079,172.92,-0.090,NC!-! FROM: /208.54.14.105 AT: 07-23-2010 15:29:38
DB Query Made
RECEIVED 357938020050134,4521.37169,N,07539.10232,W,10,2.5,3.2,53.884,0.031,172.92,0.003,NC!-! FROM: /208.54.14.105 AT: 07-23-2010 15:29:44
DB Query Made
RECEIVED 357938020050134,4521.37158,N,07539.10226,W,10,2.3,3.0,52.702,0.202,172.92,0.049,NC!-! FROM: /208.54.14.105 AT: 07-23-2010 15:29:51
DB Query Made
RECEIVED 357938020050134,4521.37149,N,07539.10234,W,10,2.3,2.9,52.979,0.040,172.92,-0.036,NC!-! FROM: /208.54.14.105 AT: 07-23-2010 15:29:58
DB Query Made
RECEIVED 357938020050134,4521.37158,N,07539.10248,W,10,2.3,2.9,53.515,0.054,172.92,0.018,NC!-! FROM: /208.54.14.105 AT: 07-23-2010 15:30:04
DB Query Made
RECEIVED 357938020050134,4521.37156,N,07539.10221,W,10,2.3,2.8,53.161,0.106,172.92,0.072,NC!-! FROM: /208.54.14.105 AT: 07-23-2010 15:30:11
DB Query Made
.
RECEIVED 357938020050134,4521.37178,N,07539.10256,W,10,2.2,2.7,53.342,0.129,172.92,-0.156,NC!-! FROM: /208.54.14.105 AT: 07-23-2010 15:30:18
DB Query Made
RECEIVED 357938020050134,4521.37177,N,07539.10281,W,10,2.2,2.7,53.549,0.058,172.92,-0.022,NC!-! FROM: /208.54.14.105 AT: 07-23-2010 15:30:24
DB Query Made
RECEIVED 357938020050134,4521.37190,N,07539.10264,W,10,2.2,2.7,52.801,0.056,172.92,0.027,NC!-! FROM: /208.54.14.105 AT: 07-23-2010 15:30:31
DB Query Made
RECEIVED 357938020050134,4521.37200,N,07539.10266,W,11,2.4,2.8,52.147,0.073,172.92,-0.027,NC!-! FROM: /208.54.14.105 AT: 07-23-2010 15:30:38
DB Query Made
█
```

Web Tracking Interface settings:



ROGERS 3G 3:32 PM 84%
Untitled
gpstrackeronline.com/tracker.php?imei=mntest2&submit=Submit Google

Twitter reddit RedFlagDeals MacRumors Facebook Digg Weathernetwork NCIXForums HDBits PogCat

Automatically load new GPS coordinates?
 Yes No

Display only stationary fixes (filters out stationary drift and speeds greater than 0.1km/h)?
 Yes No

Auto-refresh Interval
8 seconds

Automatically pan map to latest GPS coordinate?
 Yes No

Erase all existing coordinates?
Are you sure?

Minimum GPS Satellites for location fix:
2 Sats (Basic 2D Fix)

Positions for the last:
Last Known Position

Tolerance (distance between points in metres):
6

Show last X data points (Note, a non-zero setting overrides the time range data picker)
0

Refresh:
Refresh

Google Maps view of stationary location fix at bus stop bus bench:

Map | Satellite | Hybrid | Terrain

Latitude: 45.356204
 Longitude: -75.6517465
 Altitude: 58.403m MSL
 Time Start: 2010-07-23 15:27:10
 Time End: 2010-07-23 15:32:19
 Horizontal Accuracy: 2.9
 Vertical Accuracy: 4.7
 Sats: 7
 Ground speed: 0.025 km/h
 Bearing: 0 degrees
 Vertical speed: -0.018 m/s

Map data ©2010 Google Imagery ©2010 GeoEye Terms of Use

Automatically load new GPS coordinates?

Yes No

Display only stationary fixes (filters out stationary drift and speeds greater than 0.1km/h)?

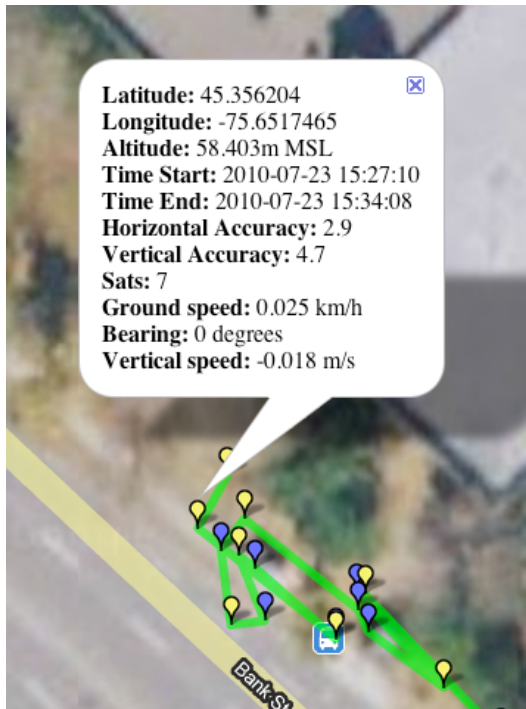
Yes No

Auto-refresh Interval

8 seconds

Outdoor pictures of location. It is evident that the GPS fix is EXTREMELY accurate, showing the position within at most 2 meters of error, if not much less.





In the screenshot on the left from the Google Maps tracking interface (at maximum zoom), we can see that the device is near an OC Transpo bus stop (blue icon resembling a Bus). From the outdoor image above, we can see that this is indeed the case, as the distance from the bus stop looks reasonably accurate.

Standing at the corner of a nearby intersection, we can again see that the GPS tracker is extremely precise:





Automatically load new GPS coordinates?

Yes No

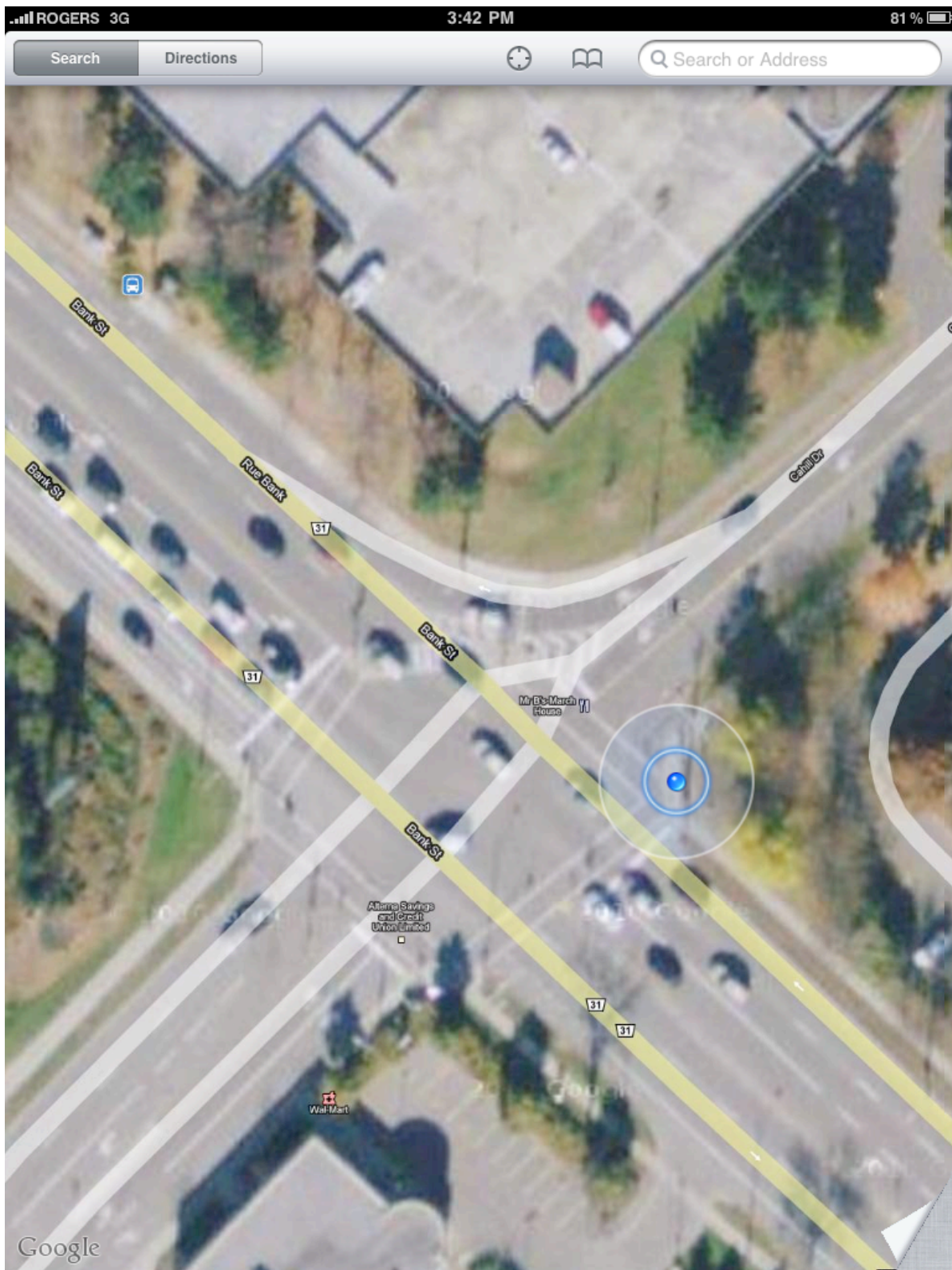
Display only stationary fixes (filters out stationary drift and speeds greater than 0.1km/h)?

Yes No

Auto-refresh Interval

8 seconds

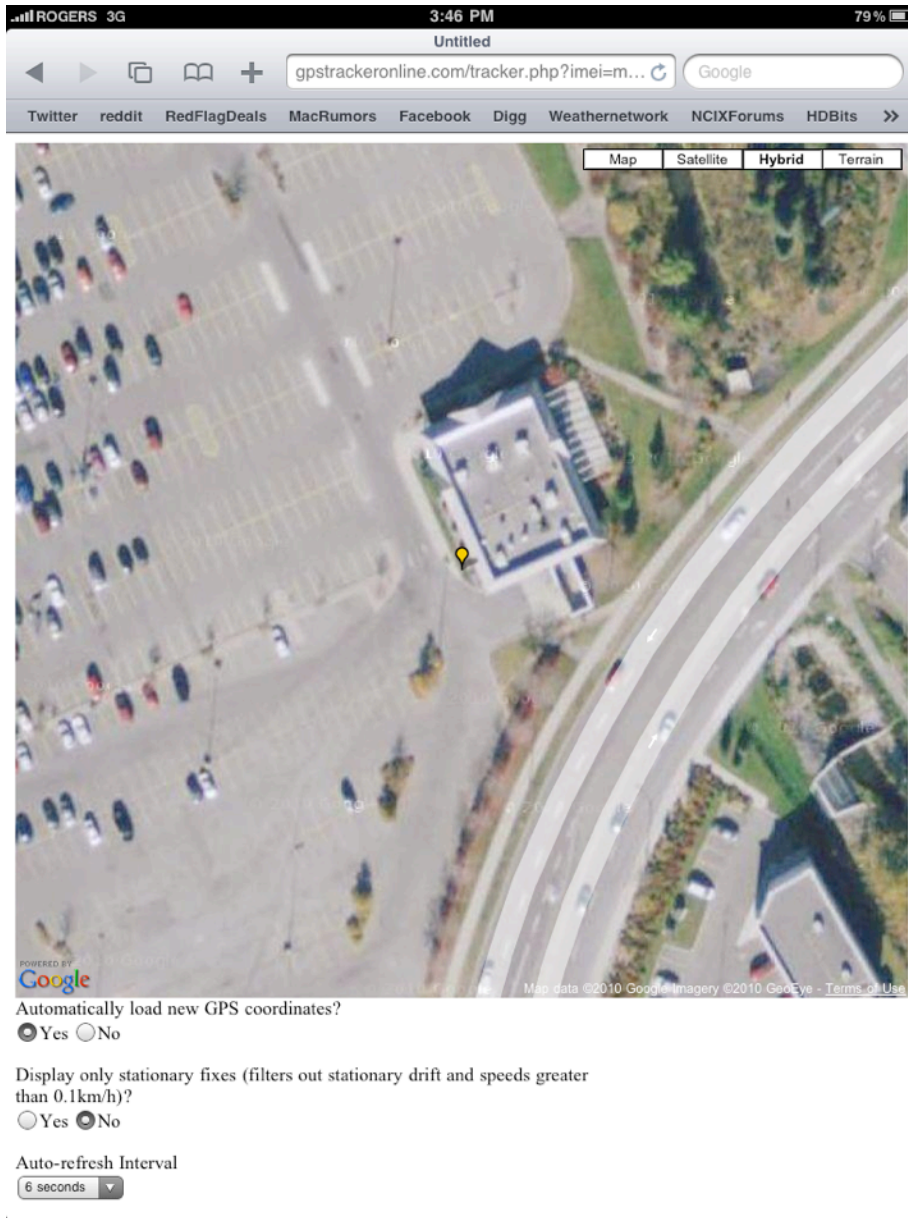
Here is a screenshot of the Maps application on the iPad, which makes use of the internal GPS in the iPad, and Google Maps. We can see that the position as reported by the iPad is similar if not identical to the position reported by the GPS tracker on the previous image.



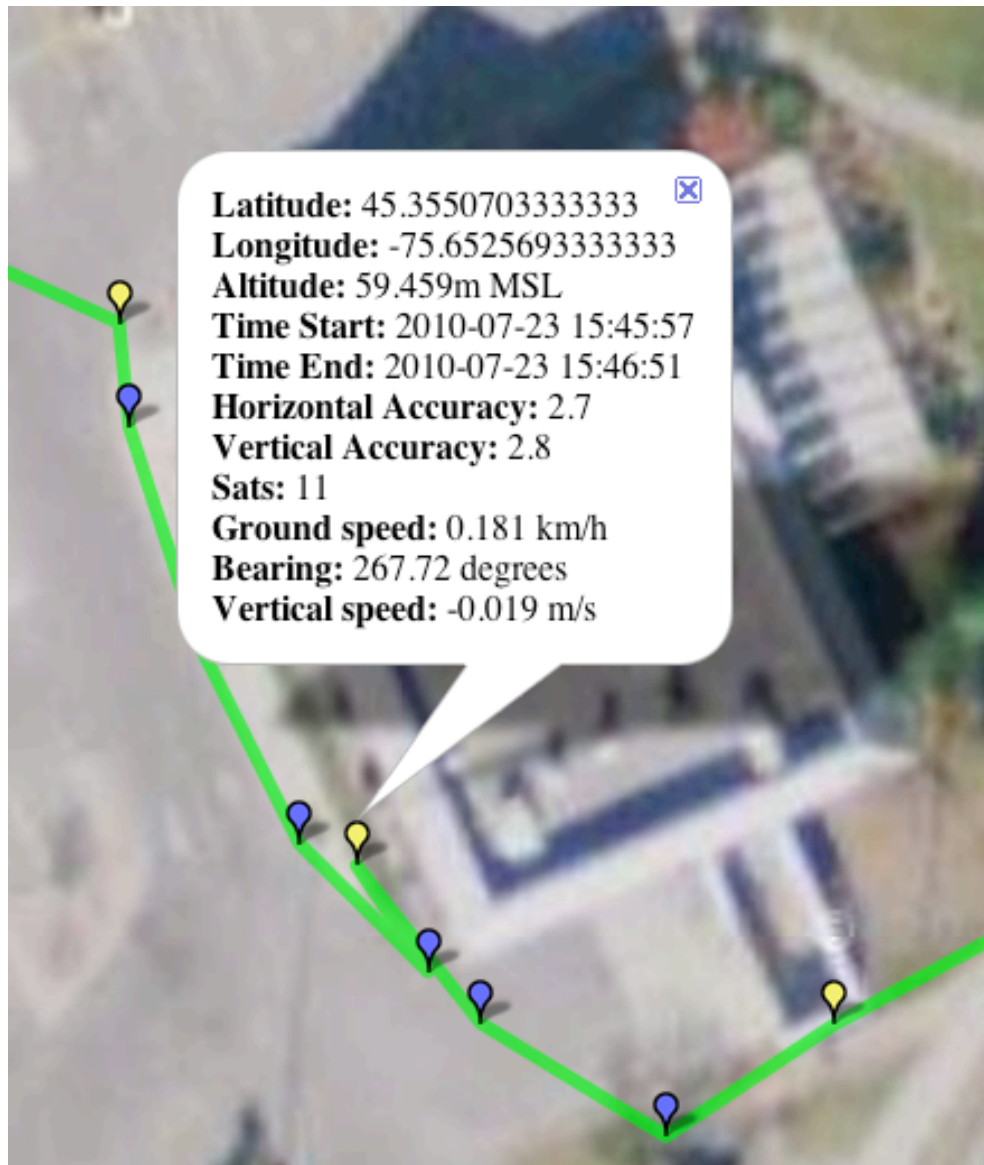
In the next image, a building as well as a grassy area with an adjacent concrete sidewalk is visible:



This location can be seen on the iPad in the previous picture, or more clearly, this screenshot taken at the same time:



More specifically, the exact information at this position is available by tapping the point itself:



The device can be seen here being tracked by the iPad right before entering a building:

Map Satellite Hybrid Terrain

Latitude: 45.3561093333333
 Longitude: -75.6552586666667
 Altitude: 62.957m MSL
 Time Start: 2010-07-23 15:50:47
 Time End: 2010-07-23 15:50:47
 Horizontal Accuracy: 7.1
 Vertical Accuracy: 2.7
 Sats: 10
 Ground speed: 4.99 km/h
 Bearing: 322.62 degrees
 Vertical speed: 0.007 m/s

POWERED BY Google

Map data ©2010 Google Imagery ©2010 DigitalGlobe, GeoEye - Terms of Use

Automatically load new GPS coordinates?

Yes No

Display only stationary fixes (filters out stationary drift and speeds greater than 0.1km/h)?

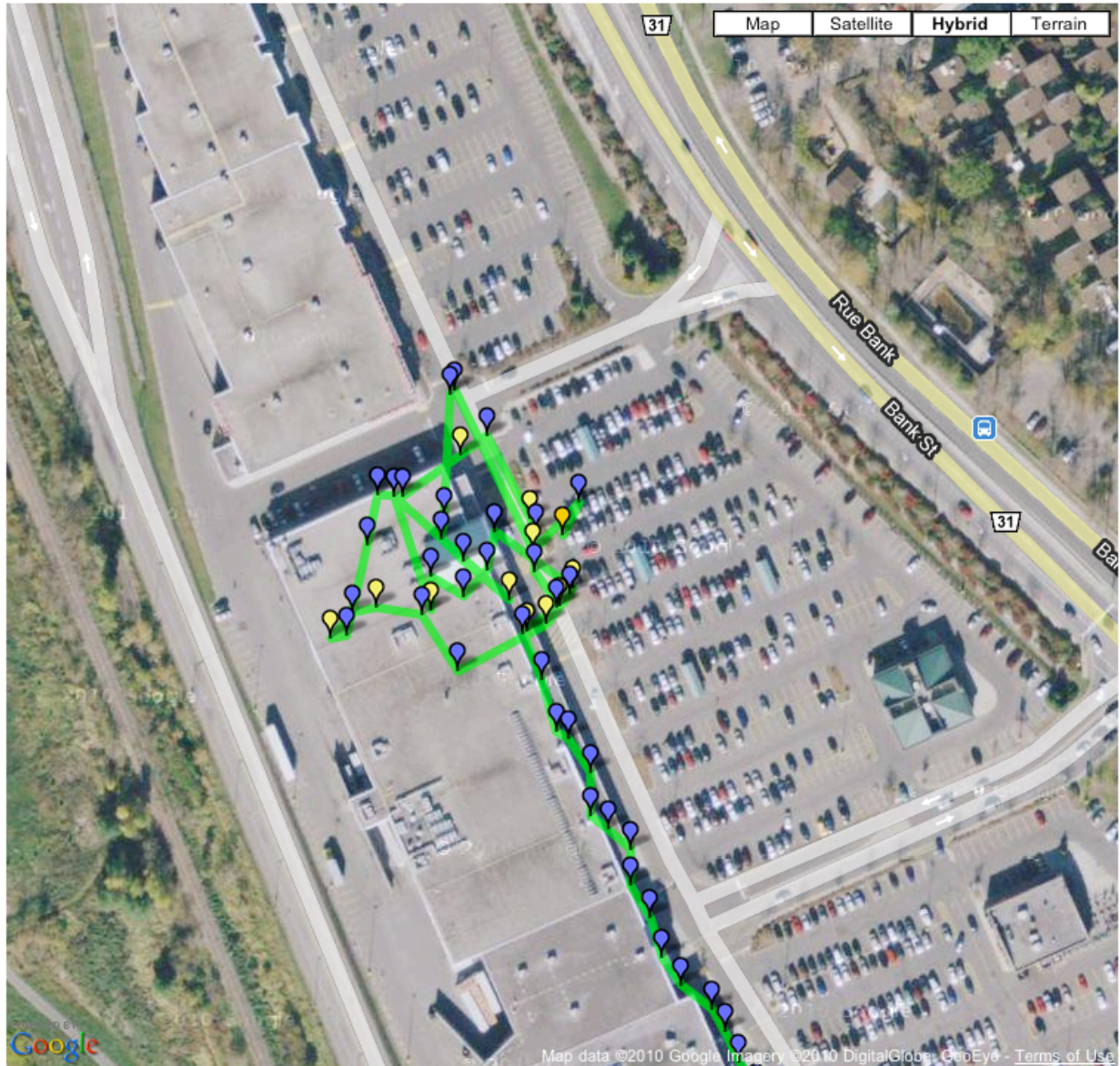
Yes No

Auto-refresh Interval

6 seconds

And after emerging from the building, with waypoint history turned on for the last 30 minutes:

...ROGERS 3G 4:02 PM 78%
Untitled
gpstrackeronline.com/tracker.php?imei=m... Google
Twitter reddit RedFlagDeals MacRumors Facebook Digg Weathernetwork NCIXForums HDBits >>



Automatically load new GPS coordinates?

Yes No

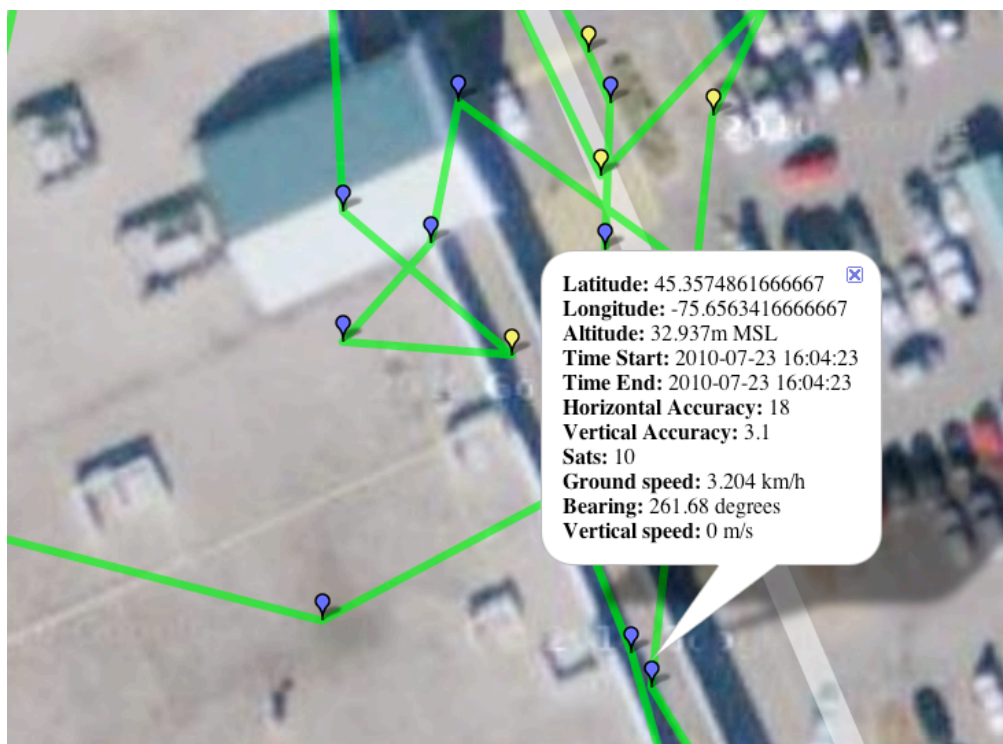
Display only stationary fixes (filters out stationary drift and speeds greater than 0.1km/h)?

Yes No

Auto-refresh Interval

6 seconds ▾

It is clear from the previous picture that the device did indeed enter the building. However, the poor GPS coverage from inside the building led to some errant GPS fixes. By clicking through the points and comparing the timestamps, an analysis of the target's whereabouts can be performed accurately. While it is unreasonable to expect the device to reveal the target's whereabouts while inside the building, our signal is restored immediately after the target emerges from the store, giving us no gaps in knowledge of the target's location.



In summary, it is clear that the tracking device performed exactly as expected during this phase of testing, demonstrating accuracy that rivals if not surpasses commercial grade geo-location hardware and software such as that found in the iPad. It is also evident that the software front-end tracking solution accurately renders the location of the device, and provides a reasonably friendly and powerful interface for analyzing the location of a target and developing a profile of its whereabouts over a period of time. The time to establish a GPS fix after startup and begin transmitting outdoors is very fast, coming in at well under a minute. The battery life is also quite impressive, and would most certainly be improved if the device was configured to transmit less often and sleep in-between. Minor hardware modifications would allow the python script to physically power down the GPS receiver and GSM radio during such a sleep period, further improving the battery life. These changes have been discussed in depth elsewhere in this report.

While this device was not intended to be used indoors, it is impressive that the device was capable of reporting a somewhat accurate location intermittently while inside a large commercial building (the Loblaws at the South Keys shopping centre).

10 Lessons Learned

- 1) Measure twice, cut once. The most obvious mistake made in this project was incorrectly reading the data-sheet for the plastic box housing the project and the height of all of the components stacked on top of each-other. This resulted in the case not being able to close entirely.
- 2) Lack of expertise cannot be remedied overnight. This project involved many areas which required knowledge of electrical engineering principles. Lacking this expertise and trying to learn it quickly resulted in a numerous amount of oversights ranging from the lack of ability to control power to the GPS receiver to various shorts and poor design decisions in the circuit board. Nonetheless, this was a very valuable learning experience in the principles of electrical engineering.
- 3) 2G cellular networks are unreliable for data communications. Throughout testing, there were numerous times when messages simply did not reach the server, whether using TCP or UDP. It was very important to learn how to deal with corrupted or missing communications and to make sure that everything was stateless. Having a stateful communication protocol in an unreliable environment would result in additional complexities and an unreliable product. While UDP provides no guarantee of transmission, the overhead and complexity in TCP communication resulted in failed communication quite often, so this was no better in practice than UDP. In fact, UDP was found to be much more reliable since the opportunity for error was much lower. TCP communication could fail during establishing the connection and transmitting data, where UDP could not fail per se, but the message simply would not arrive. While errors in both cases would be undesirable, UDP communication removed the necessity of handling unexpected states, like failed connection attempts, making UDP far less likely to put the device into an error state requiring reinitialization.
- 4) T-Mobile does not bill data going across UDP port 53 or HTTP requests without a Host header on port 80. This project made use of a T-Mobile SIM card since T-Mobile's international data roaming rates were cheaper than the local pay-as-you-go data rates of any Canadian provider. Throughout testing, it was noted that simple HTTP/1.0 requests without a host header or DNS requests (data on UDP port 53) were not being billed. Thus the decision was made to run the application servers on these ports to save costs.
- 5) This project can be used for covert surveillance. It is obvious that the low cost and complexity associated with this project vs. commercial solutions allow this technology to be developed in-house by individuals or organizations. With a minimal amount of work to enable power-saving features like sleeping for 20 minutes and shutting down the GPS receiver with a GPIO controlled load switch, and a case lined with neodymium magnets on one side, this project could be turned into a covert long-term (estimated 2 weeks battery life) vehicle or asset tracker with minimal effort.

11 Potential Improvements

- 1) Implement two-way communication to allow changes to update interval. A useful feature in the web interface would be one that allows you to change update speeds and thus enable power management features. For example, the device could be instructed to update in real-time, or update every 20 minutes and put itself and the GPS to sleep in the mean time.
- 2) Add a load switch to the GPS receiver. A load switch is a device that can control the flow of power to another device based on the voltage on one of its pins (LOW or HIGH). This pin could be tied to a GE865 GPIO pin allowing the python software to turn on and off the GPS receiver. This would significantly lengthen the battery life of the device in a long-term tracking situation (updates every 20-30 minutes as opposed to real-time).
- 3) Connect the PWRMON (power monitor) pin of the GE865 to the atmel microcontroller and set an interrupt on it being LOW. If the GE865 were to abnormally turn off (encountered nearing the end of the life of the battery), the microcontroller could reboot it. This would allow the device to make use of the remaining 10-20% of the battery, with some minor service interruptions.
- 4) Put everything on a single circuit board. This would allow the device to fit in a much smaller case. A design for this is included in this report and the submitted files. This would also require experience in surface-mount soldering and additional tools, though it is a feasible and significant improvement to the existing design.
- 5) AGPS. U-Blox (company which makes the GPS chipset) has a web-based AGPS API which allows using data networks to download GPS constellation (satellite position and timing) data as opposed to receiving the info from the GPS constellation. This allows a significantly quicker TTF (time-to-fix) meaning that the device could be ready to report its position within seconds instead of minutes (or not at all) after being turned on in an area with poor GPS coverage, such as indoors, surrounded by buildings, or in bad weather. An attempt was made to support this feature but transmitting data to the GPS UART was unsuccessful for an unknown reason. An attempt to resolve this issue was not made because it is not a requirement for this project, but it would certainly be a significant and welcome improvement.

12 Conclusion

This project was successful in terms of meeting all of the original requirements in the project proposal. The end result of this project is a functional GPS tracking device, capable of transmitting its location via GSM cellular data networks to a remote server where the data is stored and visualized using the Google Maps API. While there were some shortcomings, mainly due to budget, expertise, and time constraints, there were no fatal flaws in the project architecture or designs which would have resulted in unmet requirements.

This project was a beneficial learning experience in computer science, touching both hardware and software technologies and focusing on integration and delivering a functional product from the ground up. Undoubtedly, this project was a suitable, if not an ideal choice for the honours project course.

Experience in developing a functional end-to-end solution for meeting a set of requirements from start to finish is arguably one of the most important experiences that an individual can have, because there is no limit to the scope of failure. Unfortunately, this experience is rarely, if ever gained during a typical undergraduate class. Thus it has been immensely valuable to set such an ambitious goal for an honours project, and without a doubt, this project was an overwhelming success.

13 Relation and Contribution to Computer Science

This project relates to computer science because it consists of requirements from many different areas in computer science. This project involves hardware design and development, software design and development, client/server communication, software testing, software integration, geo-location, and cellular data networks. The successful outcome of this project has demonstrated the value in developing an understanding of these topics and represents a comprehensive education in computer science.

In terms of contributions to computer science, this project is an end-to-end guide or template to designing, developing, and operating a tracking device remotely using technologies like cellular data networks and the Google Maps API. Interested parties looking to develop similar geo-location technologies can use this project to complement existing research or as a framework for developing something new. Since these areas are undergoing continuous innovation, this project represents a valuable contribution to computer science.

14 References

- [1] CadSoft EAGLE <http://www.cadsoft.de/freeware.htm>
- [2] BatchPCB <http://batchpcb.com/index.php/AboutUs>
- [3] Interfacing Arduino with a Telit GM862 <http://tinkerlog.com/2009/05/15/interfacing-arduino-with-a-telit-gm862/>
- [4] GPS Latitude and Longitude Converter <http://www.csgnetwork.com/gpscoordconv.html>
- [5] Google Maps API v3 <http://code.google.com/apis/maps/documentation/javascript/>
- [6] Detecting Mobile Devices Using PHP <http://www.hand-interactive.com/resources/detect-mobile-php.htm>
- [7] Nightingale: Arduino Watchdog and Sleep functions
http://interface.khm.de/index.php/lab/experiments/sleep_watchdog_battery/

Appendix A - Submitted Files

Application Server Source Code and Binaries/Java Application Server Netbeans Project/GPSTrackerServer/src; This folder contains the source code for the application server.

Application Server Source Code and Binaries/Java Application Server Netbeans Project/GPSTrackerServer/dist; This folder contains the JAR for the application server which can be executed via `java -jar GPSTrackerServer.jar`

Application Server Source Code and Binaries/Experimental HTTP PHP Application Server/mysql.php; This file contains the authentication info for accessing mysql

Application Server Source Code and Binaries/Experimental HTTP PHP Application Server/server.php; This file is the HTTP/TCP Application server written in PHP. It is identical to the Java version, except the network layer (TCP instead of UDP). It is not used and has underwent limited testing.

Development Tools and Useful Utilities/Round Solutions RSTerm Telit Module Console; This folder contains a serial console application with many predefined macros and functions specific to Telit GSM/GPRS modules. It is also the preferred method of uploading python code to the tracking device. This is Windows-only.

Development Tools and Useful Utilities/FT232 USB to Serial Chip Drivers for Interfacing with the Tracking Device; This folder contains drivers to connect the tracking device to a PC for the purpose of accessing the serial console. Mac OS X and Windows versions are included.

Development Tools and Useful Utilities/Telit Python IDE PythonWin; This is the Python IDE from Telit to write code for the Telit GE865 module inside the tracking device. This is Windows only.

Development Tools and Useful Utilities/NetBeans Java and PHP IDE; The NetBeans IDE can be used to develop Java and PHP applications. Mac OS X and Windows versions are included.

Development Tools and Useful Utilities/EAGLE Lite Electrical Schematic CAD Program; This folder contains the installer for the EAGLE Electrical CAD program used to create the schematics discussed in this report. Mac OS X and Windows versions are included.

Development Tools and Useful Utilities/EAGLE Lite Electrical Schematic CAD Program/Sparkfun Electronics Library; This folder contains EAGLE device footprints for many items sold at www.sparkfun.com. It is used extensively in the schematics and must be imported into EAGLE before opening the schematics.

Development Tools and Useful Utilities/U-Blox u-center GPS Configuration Tool; U-Center is the tool from U-Blox (manufacturer of the GPS chipset in the tracking device) and can be used to customize the output strings (and many other settings) of the GPS receiver. This application is Windows-only.

Development Tools and Useful Utilities/Google Chrome (Web Browser with High-Performance JavaScript Engine); Google Chrome is a web browser with a high-performance JavaScript engine and is the ideal browser for accessing the tracking device web interface. Mac OS X and Windows versions are available.

Development Tools and Useful Utilities/PuTTY Serial and SSH Console; PuTTY is a console/terminal program used in this project for SSHing to the application server and accessing

the serial console of the device in Windows. This application is Windows-only. OpenSSH (commandline) can be used in Mac OS X and UNIX variants for SSH, and screen or minicom can be used for serial console access.

Google Maps Front-End Source Code; This folder contains the PHP and JavaScript source code for the front-end tracking interface.

Google Maps Front-End Source Code/oldUtil.js; This file contains the old AJAX libraries from the Google Maps API which fit better into this application than the new ones. This code is owned by Google.

Google Maps Front-End Source Code/index.php; This is the login page which generates the GET request for tracker.php.

Google Maps Front-End Source Code/tracker.php; This is the tracking page. It renders a Google Maps view of co-ordinates in the database given a correct IMEI number.

Google Maps Front-End Source Code/mdetect.php; This is the PHP application which inspects user agent headers to decide if a device is a mobile device or a full featured browser. It is used in tracker.php. This code is owned by Anthony Hand, www.hand-interactive.com.

Google Maps Front-End Source Code/ajaxXmlGen.php; This is the XML generator called by tracker.php which converts database co-ordinates into an XML file parsed by the tracking interface via an AJAX call.

Google Maps Front-End Source Code/mysql_info.php; This file contains the authentication info for MySQL.

Google Maps Front-End Source Code/register.php; This file creates the table structure for an IMEI number and must be called via index.php before a tracking device can make use of the application server.

Report; This folder contains both the PDF and the source Apple iWork .pages file for the report.

Source Images from Report/Hardware Development; This folder contains the images taken during development of the GPS tracking device.

Source Images from Report/Field Test Pictures; This folder contains pictures taken with a digital camera during a field test of the device.

Source Images from Report/Hi-Res TIFFs from Diagrams Used in Report; This folder contains the high resolution TIFF files exported from EAGLE and other applications to design the schematics used in this report.

Source Images from Report/iPad Testing Screenshots; This folder contains screenshots taken on the Apple iPad used during testing of the device.

Source Images from Report/Thickness Comparison with iPhone and Final Device Pictures; This folder contains some high-resolutions images of the completed tracking device as well as some size comparison shots comparing the device to an iPhone 3G.

Tracking Device Component Documentation; This folder contains documentation from manufacturers of components used in this project.

Tracking Device Component Documentation/GS407-090812.pdf; This file is the datasheet of the GPS receiver used in the project.

Tracking Device Component Documentation/

Telit_AT_Commands_Reference_Guide_r6-1.pdf; This file is the AT command reference guide from Telit for the Telit GE865 GSM/GPRS module used in this project.

Tracking Device Component Documentation/Telit_GE865-

QUAD_Hardware_User_Guide_r10.pdf; This file is the hardware datasheet from Telit for the Telit GE865 GSM/GPRS module used in this project.

Tracking Device Component Documentation/FT232RL-Breakout-Schematic.pdf; This file is the schematic of the FT232R USB to Serial breakout board from Sparkfun electronics.

Tracking Device Component Documentation/Telit_Easy_Script_Python_r10.pdf; This file is documentation for the python scripting engine from Telit for the Telit GE865 GSM/GPRS module used in this project.

Tracking Device Component Documentation/Telit_Easy_GPRS_User_Guide_r7.pdf; This file is the documentation for the GPRS functions from Telit for the Telit GE865 GSM/GPRS module used in this project.

Tracking Device Component Documentation/MCP73871EV-Microchip.pdf; This file is the datasheet for the MCP73871EV power management and charging boards from Microchip used in this project.

Tracking Device Schematics/Project Board (Used in Demo and Report); This folder contains the EAGLE schematics (electrical and board layout) for the custom circuit board used in this project.

Tracking Device Schematics/Revised All-In-One Board (Future Development, Untested); This folder contains the EAGLE schematics (electrical and board layout) for an all-in-one surface-mount revision of this project. This was never produced due to time and cost constraints, but is useful for seeing what a “commercial” version of this project would look like.

Tracking Device Source Code; This folder contains the python code which runs on the Telit GE865 module inside the tracking device.