FINAL REPORT

18/10/12

FINAL YEAR PROJECT 2012
FINAL REPORT (REV B)

REALTIME ROUTE LEARNING AND VEHICLE TRACKING USING WEB-TECHNOLOGIES ON A MOBILE DEVICE

HADI MICHEL SALEM

TRC4000 - TRC4001
Department of Electrical and Computer Systems Engineering
Monash University, Australia

Academic Supervisor: Dr. Wai Ho Li

MONASH University

Department of Electrical and
Computer Systems Engineering

TRC4000/1 Final Year Project, Semester 2, 2012 Hadi Michael

Realtime Route Learning and Vehicle Tracking Using Web-technologies on a Mobile Device

I've seen the
FUTURE
It's in my
BROWSER

Project Aim

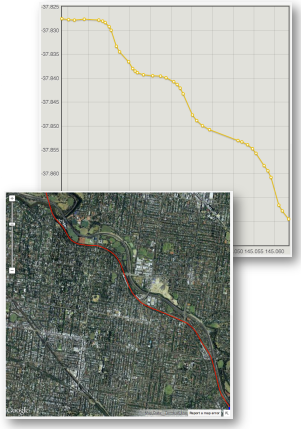
By exploring data-mining, mathematical modelling and machine learning algorithms, the V-Tracker project aims to achieve **realtime route learning and vehicle tracking using web-technologies on a mobile device**. Melbourne's public transport network serves as a testing ground for the project since it is a well-known and clearly constrained system.

APACHE CORDOVA™

HTML5 + CSS3 + JavaScript

Numeric Javascript

jQuery mobile framework.



Sensing on mobile devices

The application developed in this project **uses the motion and localisation sensors** available on typical smartphones and mobile devices.

Web technologies

The technologies used in this project run primarily in the device's web-browser and are platform agnostic, running on both iOS and Android. The application relies on front-end web-technologies to **access the device's sensors, collect spatial and temporal data and process the data within milliseconds of it being collected**.

Route learning and tracking

By applying mathematical modelling algorithms, **the application creates and visualises a model of the route in realtime**. This enables the application to **make intelligent decisions**, such as those required **for vehicle tracking**.

As one of the first projects in this space, this application is setting the foundations for future experiments that will investigate the use of web-technologies in areas of digital perception and robotics.

"We are pushing device web-browser engines to the edge"

… and it's all open source!

Visit the project wiki and explore the code online at:
http://github.com/hadimichael/V-Tracker

Executive Summary

As the World Wide Web develops, the popularity of content-rich web and e-commerce applications continues to grow. Modern software development frameworks enable the native deployment of applications written primarily using web-technologies onto a range of mobile operating systems. Furthermore, mobile devices have evolved as a computing platform and now incorporate a range of new sensors. As a result of this advancement, we are now able to investigate the use of web-technologies in areas of digital perception and robotics. This project focuses on exploring data-mining and mathematical modelling algorithms that aim to achieve **realtime route learning and vehicle tracking using web-technologies on a mobile device**.

In order to ensure that sensor data is reliably acquired using web-technologies and to facilitate the deployment of the application on multiple mobile devices, the Cordova (also known as PhoneGap) framework was selected. The application, its encompassed algorithms and documentation form the primary deliverable for this project. The user interface developed does not use native elements and is instead designed to be platform agnostic taking on the same form on all iOS and Android devices.

A modelling algorithm based on the root mean squared errors was derived to model routes. Melbourne's public transport network served as a testing ground for the project since it is a well-known and clearly constrained system. An experiment on a Melbourne railway line demonstrated the modelling algorithm's reliability in capturing route data measured on public transport. An experiment involving an unmanned aerial vehicle demonstrated the modelling algorithm's reliability in modelling unpredictable and quickly changing routes. Future work may explore the possibilities of transmitting realtime model data back to a central computer, which in turn can use the data to control and direct a swarm of drones. Driving on road was a good demonstration of the application's ability to model and track everyday vehicle activity, such as that required for tracking taxi or logistics fleets.

In an attempt to learn and track routes with obstructed GPS signals, a preliminary investigation of inertial navigation using mobile devices was conducted. The objective was to look into techniques for tracking trains while they went through the underground portion of Melbourne's City Loop. Overall, the assessment of the data in this experiment is inconclusive in asserting if a mobile device will be sufficient in tracking a train with obstructed GPS signals. Whilst this is an interesting area worth exploring, any further analysis was regarded as outside the scope of this project and has been set aside for future work.

Table of Contents

List of Abbreviations v

List of Figures vi

List of Tables vii

List of Code Snippets vii

List of Flow Charts vii

1 Introduction 1

2 Project Aim 1

3 Requirements Analysis 1

3.1 Types of statements 1

3.2 Project statements 1

3.2.1 Definitions 1

3.2.2 Assumptions 2

3.2.3 Requirements 2

3.2.4 Optional requirements 3

3.2.5 Exclusions 3

4 Web Technologies 3

4.1 HTML5 and emerging web standards 4

5 Design Decisions 5

5.1 Hardware 5

5.1.1 Mobile devices 5

5.1.2 Device sensors 5

5.2 Software 5

5.2.1 Selected subset of web-technologies 5

5.2.2 HTML5 storage 5

5.2.3 Route visualisation 6

5.2.4 Software tools, plugins and open-source libraries 6

6 Hardware 6

6.1 Sensing on mobile devices 6

6.2 Motion sensors 7

6.2.1 Accelerometer 7

6.2.2 Gyroscope 9

6.3 Localisation sensors 12

6.3.1 Compass 12

6.3.2 Global Position System (GPS) 13

7 Software and Algorithms (Method) 15

7.1 Overview 15

7.2 The core application (vtracker.js) 16

7.2.1 The “route” object 16

7.2.2 Route modelling (algorithm) 17

7.2.3 Route modelling (mathematics) 19

7.2.4 Route tracking (algorithm) 23

7.2.5 Route tracking (mathematics) 25
7.3 Known limitations 28
7.3.1 Sharp corners (algorithmic) 28
7.3.2 Live map visualisation (physical) 28
7.4 PhoneGap Plugins 28
7.4.1 Plugin 1: SensorsManager 29
7.4.2 Plugin 2: QuickStorage 29

8 Experimental Results and Discussion 29
8.1 On a train railway line 29
8.2 In flight (using an aerial drone) 30
8.3 Driving on road 31
8.4 A preliminary attempt at inertial navigation 32
8.4.1 Inertial Navigation System (INS) 32
8.4.2 Inertial navigation using an Apple iPhone 32
8.4.3 Attempting inertial navigation tracking on a railway line 32

9 Conclusion 38

10 Acknowledgements 39
10.1 Software libraries and plugins 39
10.1.1 PhoneGap - Apache 2.0 39
10.1.2 jQuery – MIT 39
10.1.3 jQuery mobile – MIT 39
10.1.4 flot – MIT 39
10.1.5 Numeric JavaScript – MIT 39
10.1.6 Modernizr – MIT & BSD 39
10.1.7 Google Maps – application must be free for users 40
10.2 Other tools 40
10.2.1 node.js 40
10.2.2 YUIDocs - Javascript Documentation Tool 40
10.2.3 AptanaStudio3 40
10.2.4 Mou App 40
10.2.5 GitHub 40

11 References 41

List of Abbreviations

ECSE – Electrical and Computer Systems Engineering

V-Tracker – Vehicle Tracker

WWW – World Wide Web

HTML – HyperText Markup Language

XML – Extensible Markup Language

CSS – Cascading Style Sheets

JS – JavaScript

PHP – PHP Hypertext Preprocessor

API – Application Programming Interface

GPS – Global Positioning System

MDS – Mobile Data Services

SMS – Short Messaging Service

VoIP – Voice Over Internet Protocol

SaaS – Software as a Service

WebGL – Web Graphics Library

W3C – World Wide Web Consortium

OWP – Open Web Platform

DOM – Document Object Model

CPU – Central Processing Unit

UI – User Interface

dof – degrees of freedom

MEMS – Micro Electro-Mechanical System

ASIC – Application Specific Integrated Circuit

AGPS – Assisted Global Positioning System

GSM – Global System for Mobile Communications

UTMS – Universal Mobile Telecommunications System

UAV – Unmanned Aerial Vehicle

INS – Inertial Navigation System

IMU – Inertial Measurement Unit

List of Figures

Figure 1: HTML5 tagline (left) and logo (right) 4
Figure 2: Some key areas introduced in HTML5 4
Figure 3: Logo for Apache Cordova, the project behind PhoneGap [6] 6
Figure 4: iPhone 4 main board with the STMicroelectronics LIS331DLH accelerometer and the L3G4200D gyroscope side-by-side [9] 7
Figure 6: LIS331DLH XY accelerometer sensor detail [9] 8
Figure 7: L3G4200D GK10A three-axis gyroscope die [11] 10
Figure 8: L3G4200D detail [11] 11
Figure 9: The 3 dof gyroscope on the iPhone 4 provides roll, pitch and yaw angular velocities [10] 12
Figure 10: The direction of an axis' positive angular change can be found using the Right Hand Rule [10] 12
Figure 11: AK8973 Hall sensor layout [12] 12
Figure 12: iPhone 4 main board with the AKM AK8975 electronic compass [12] 13
Figure 13: The Broadcom BCM4750IUB8 single-chip GPS receiver [15] 14
Figure 15: “Start” is the application’s primary page 15
Figure 16: "Debug" provides direct access to individual sensors, storage and notifications API 15
Figure 17: "Info" provides license and general project information 15
Figure 18: Plot demonstrating $\boldsymbol{vi}$ and $\boldsymbol{vj}$ 21
Figure 19: Plot demonstrating $\boldsymbol{vi}'$ and $\boldsymbol{vj}'$ 22
Figure 20: Route visualisation on a satellite map showing 72% travelled 23
Figure 21: Route visualisation on a hybrid map showing 69% travelled 23
Figure 22: A figure-eight route modelled using a 1m noise threshold and visualised on a satellite map 26
Figure 23: A figure-eight route modelled using a 2m noise threshold and visualised on a plot 26
Figure 25: Model visualised for the Glen Waverley railway line in Melbourne's southeast 29

Figure 26: Screenshot of the modelling process running in a computer web-browser (Google Chrome) 30
Figure 27: Raw geolocation measurements from a UAV route 31
Figure 28: The UAV route's model at noise threshold radius of 0.5m 31
Figure 29: Visualised model for a route travelled by car (the model uses a noise threshold radius of 3m) 31
Figure 30: Journey measurement intervals for the INS experiment 33
Figure 31: Acceleration data (over 17mins) with filtering 34
Figure 32: The sliding window offset superimposed on the acceleration data 35
Figure 33: Normalised acceleration after filtering 35
Figure 34: Normalised acceleration and speed estimates 36
Figure 35: Comparison of distance estimates 37
Figure 36: Residuals plot of the speed estimated using acceleration and that estimated using GPS 38

List of Tables

Table 1: Common update intervals for acceleration events in iOS [10] 9
Table 2: The route object's properties and functions 16

List of Code Snippets

Code snippet 1: Isolating the gravity component (low-pass filtering) [10] 9
Code snippet 2: Isolating instantaneous motion (high-pass filtering) [10] 9

List of Flow Charts

Flow chart 1: Logic flow for "onLearningGeoMeasurement" function 17
Flow chart 2: Logic flow for the modelling algorithm 18
Flow chart 3: Logic flow for the tracking algorithm 24

1 Introduction

Modern software development frameworks enable the native deployment of applications written primarily using web-technologies onto a range of mobile operating systems. As a result, it is now possible to use smartphone web-browsers to directly access device hardware, whilst providing realtime feedback through responsive user interfaces. Research in the Department of Electrical and Computer Systems Engineering (ECSE) at Monash University is exploring the boundaries of web-technology and pushing device web-browser engines to the edge. As one of the first projects in this space, the V-Tracker (or Vehicle Tracker) application is setting the foundations for future experiments that will investigate the use of web-technologies in areas of digital perception and robotics.

2 Project Aim

By exploring data-mining and mathematical modelling algorithms, the V-Tracker project aims to achieve **realtime route learning and vehicle tracking using web-technologies on a mobile device**. Melbourne's public transport network serves as a testing ground for the project since it is a well-known and clearly constrained system.

3 Requirements Analysis

The purpose of this section is to identify the requirements for the V-Tracker project.

3.1 Types of statements

There are five different types of statements that are considered for this project:

i. **Definitions**: are listed in the form of "DEF.xx". These seek to clarify key terms used throughout the project.
ii. **Assumptions**: are listed in the form of "AS.xx". These are statements made to facilitate the achievement of the requirements.
iii. **Requirements**: are listed in the form of "R.xx". These refer to the essential requirements necessary to fulfil the project expectations and therefore must be achieved.
iv. **Optional Requirements**: are listed in the form of "OR.xx". These refer to additional requirements that would be nice to have, but are not essential.
v. **Exclusions**: are listed in the form of "EX.xx". These refer to requirements that will not be developed or considered for this project.

3.2 Project statements

3.2.1 Definitions

- **DEF.01**: "Realtime" implies that data is processed within milliseconds of it being collected.
- **DEF.02**: A "Vehicle" is defined as any machine used for transporting people or goods on land or by air (this excludes maritime travel).

- **DEF.03**: “Route learning” implies creating and updating (in realtime) a mathematical model that represents a vehicle’s route as travelled from an initialised origin.
- **DEF.04**: “Tracking” implies spatial and temporal identification of a vehicle’s location on an existing modelled route.
- **DEF.05**: “Web-technologies” refers to any and all technologies associated with the World Wide Web (WWW). This includes, but is not restricted to, code written using markup and scripting languages such as HTML, XML, CSS, JS, PHP.
- **DEF.06**: “Web interfaces” refers to Application Programming Interfaces (API) that can be accessed using web-technologies.
- **DEF.07**: A “mobile device” is defined as an electronic computing device that can:
    - connect to a cellular network, and
    - be easily relocated or transported.
- **DEF.08**: “Melbourne’s Public Transport Network” includes all train services operated by Metro Trains Melbourne, all tram services operated by Yarra Trams and some metropolitan bus services.
- **DEF.09**: Melbourne’s “City Loop” is defined to include the following train stations: Flinders Street, Southern Cross, Melbourne Central, Flagstaff and Parliament.

3.2.2 Assumptions

- **AS.01**: There are no hard deadlines for processing data in realtime.
- **AS.02**: The project will use an Apple iPhone 4 as a mobile device.
- **AS.03**: The mobile device can be fixed to the vehicle such that any translation or rotation of the device is restricted once the vehicle is initialised at the route’s origin.
- **AS.04**: The device’s Global Positioning System (GPS) signal is never severely disrupted or degraded whilst on the route.

3.2.3 Requirements

- **R.01**: the project outcome must include a mobile software application (the “application”).
- **R.02**: the application must be developed in a form that allows it to be natively deployed on different mobile operating systems, namely: iOS 5.0+ and Android 4.0+.
- **R.03**: the application’s core libraries must be written using web-technologies.
- **R.04**: the application must use web interfaces to collect geolocation data using the Global Positioning System (GPS) receiver found on the mobile device.
- **R.05**: the application must use web interfaces to collect data that indicates magnetic north using the compass sensor found on the mobile device.
- **R.06**: the application must use web interfaces to collect acceleration data using the accelerometer sensor found on the mobile device.
- **R.07**: the application must use web interfaces to infer device orientation using the gyroscope sensor in combination with the accelerometer and compass sensors found on the mobile device.
- **R.08**: the application must use web-technologies for data storage.

- **R.09**: a user must be able to export collected data for further analysis in mathematical packages such as MATLAB™, Octave or Mathematica™.
- **R.10**: the application must use web interfaces to generate user notifications.
- **R.11**: the application must use web-technologies for realtime data analysis and interpretation.
- **R.12**: the application must learn a route by creating and updating (in realtime) a mathematical model that represents the vehicle's route as travelled from an initialised origin.
- **R.13**: The application must use web-technologies to generate a visual representation of a modelled route on the mobile device.
- **R.14**: the application must be capable of learning a railway route and tracking a service on that route - provided the rail segment is outside the City Loop.

3.2.4 Optional requirements

- **OR.01**: the application may learn tram routes.
- **OR.02**: the application may learn some metropolitan bus routes.
- **OR.03**: the application may track train services whilst within the City Loop.

3.2.5 Exclusions

- **EX.01**: the application will not consider user-experience.
- **EX.02**: the application will not seek user consent for any action.
- **EX.03**: in order to avoid premature optimisation during software development, the code will not be optimised unless absolutely necessary.

4 Web Technologies

As the World Wide Web develops, the popularity of content-rich web and e-commerce applications continues to grow. Morgan Stanley Research estimates 1.8 billion users connected via the Internet in 2009 [1] and growth in usage remains robust across the globe [2]. Furthermore, computing technologies continue to evolve with high-speed broadband and wider spread 3G networks now penetrating many developing nations [2]. The increased deployment of high-speed mobile networks is also encouraging Internet use on mobile devices in particular [1]. It is expected that "more than one third of European mobile subscribers will be using mobile Internet services by the end of 2013" [3]. Moreover, the report from Morgan Stanley's research division suggests that the number of mobile Internet users is actually expected to surpass that of desktop users in 2014 [1]. Researchers have defined mobile Internet use in the form of Mobile Data Services (MDS) as "all non-voice services afforded through mobile networks, except for interpersonal SMS exchanges, that the end users can employ whilst mobile" [3]. While improvements in mobile networks remain key to the success of mobile Internet, there are many other influencing factors that continue to drive the uptake of mobile use, primarily: the Global Position System (GPS), improved motion sensing, communication technologies such as Voice Over Internet Protocol (VoIP), growth in Software as a Service (SaaS) platforms and the global expansion of social media [2].

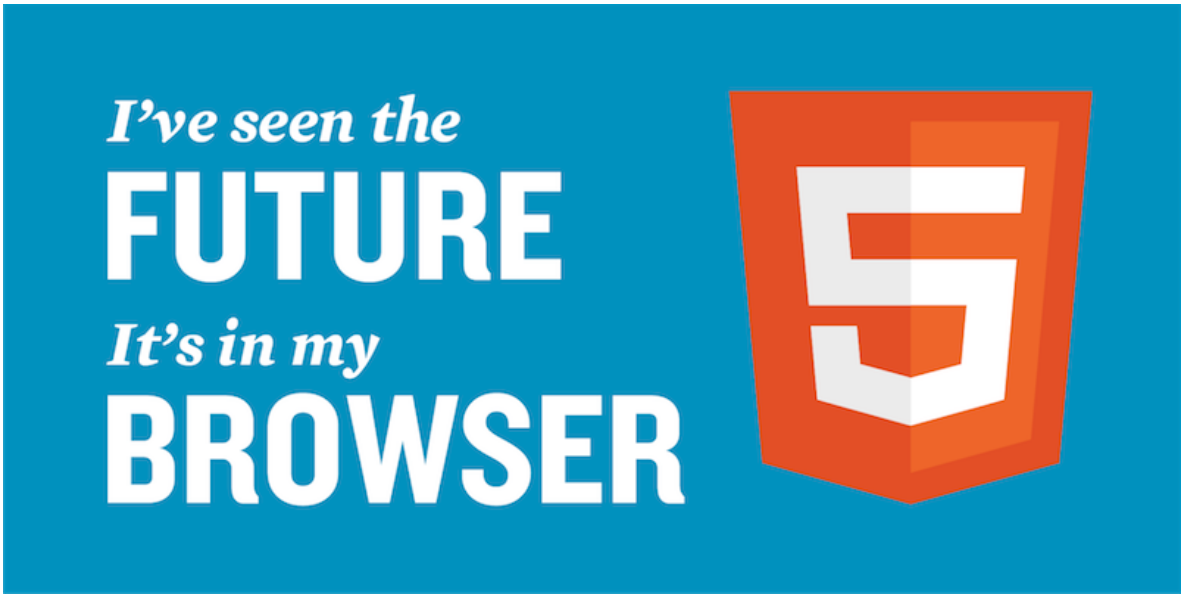


Figure 1: HTML5 tagline (left) and logo (right)

4.1 HTML5 and emerging web standards

Until recently, the development and effective deployment of complex, fully interactive web-applications has been hampered by a variety of obstacles [4]. Emerging standards, particularly those pertaining to HTML5 and WebGL are removing many limitations and providing developers with the platforms to transform the web [4]. Through the consensus and support of the broader international community, the World Wide Web Consortium (W3C) provides standards that define an Open Web Platform (OWP) for application development [5]. Of the many forthcoming standards, the HTML5 standard is commonly highlighted as a major step forward. The new standard complements the existing HTML standards by adding new features aimed at narrowing the distinguishing factors between web and desktop applications [4]. Some of these new features include: Offline Applications, Local Storage, Canvas API, Built-In Audio and Video Support, Asynchronous Script Loading, Drag-and-Drop Support, Context Menus and Cross-Document Messaging [4,5]. Further to the evolution in HTML and CSS standards, there have been significant developments to client-side JavaScript API, particularly those relating to geolocation, XMLHttpRequest and the Document Object Model (DOM) [5].

Figure 2: Some key areas introduced in HTML5

5 Design Decisions

The purpose of this section is to identify some of the major design decisions that have been made for the V-Tracker project. Design decisions are broken down into Software and Hardware decisions as outlined below.

5.1 Hardware

It is a project requirement that the software be developed in a form that allows it to be deployed on multiple mobile operating systems, namely: iOS 5.0+ and Android 4.0+. As a result, there can be no strict hardware requirements for this project. In future development, the project may incorporate additional hardware accessories that provide higher accuracy motion sensing and/or user-interfacing hardware that can enhance the user experience.

5.1.1 Mobile devices

An Apple iPhone 4 has been selected as a mobile device for use in prototype development. An Asus Nexus 7 and an HTC Desire have also been used to demonstrate the software's versatility across different devices.

5.1.2 Device sensors

It was decided that the application would attempt to collect and store information from the device's motion sensors (accelerometer and gyroscope) and localisation sensors (compass and GPS).

5.2 Software

5.2.1 Selected subset of web-technologies

It was decided that the application would be executed entirely inside the web-browser for this stage of the research. This implies that all necessary computation was to be performed locally using the web-browser's JavaScript engine. Furthermore, all data was to be stored locally using web-interfaces made available in HTML5. The primary benefit of this decision was the application's ability to run independent of a server and more importantly, without the need for persistent Internet or network connectivity. The obvious constraint presented here is the limited realtime computational power available to the web-browser's DOM. It is anticipated that future projects will explore the benefits of distributed computing and possibly offload the computation and storage to a dedicated server.

5.2.2 HTML5 storage

HTML5 standards propose two primary forms of data storage: local key/value storage (localstorage/sessionstorage) and local database storage (SQLite3) [5]. According to the W3C, the SQLite3 standards are no longer being maintained and as a result local key/value storage was chosen as the primary form of data storage for V-Tracker. Nonetheless, in order to satisfy the requirement relating to easy data export for further data analysis, the SQLite web-interfaces were used to enable users to create a SQLite3 database file that could be downloaded from the device. The database can provide comma-separated tables that can be imported into mathematical packages.

5.2.3 Route visualisation

Google serve an open API for accessing their maps platform. Given its power and well-established developer community, Google Maps was selected for satellite and map-based route visualisation. Access to and use of the Google Maps API is free providing the final application is also free for users.

5.2.4 Software tools, plugins and open-source libraries

The project outcome includes a mobile software application (V-Tracker). The application, its encompassed algorithms and documentation form the primary deliverable for this project. Many crucial decisions were made with regard to the tools, plugins and open-source libraries used in V-Tracker. Each of those was chosen on the merit of their unique functionality. More information, in addition to well-deserved credit, is given for all the tools, plugins and open-source libraries used in V-Tracker in following sections.

The most noteworthy framework used in this project is: PhoneGap. In order to ensure that sensor data was reliably acquired using web-technologies and to facilitate the deployment of the application on multiple mobile devices, the Cordova (also known as PhoneGap) framework was selected. According to the PhoneGap website:

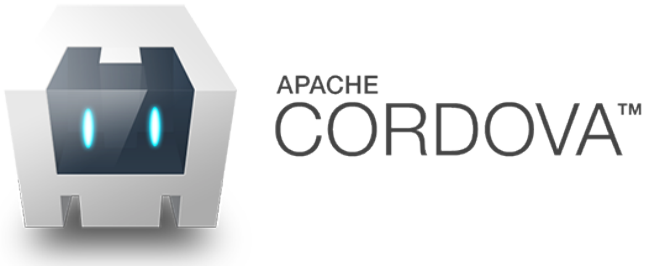


Figure 3: Logo for Apache Cordova, the project behind PhoneGap [6]

"PhoneGap is an HTML5 app platform that allows you to author native applications with web technologies and get access to APIs and app stores. PhoneGap leverages web technologies developers already know best... HTML and JavaScript." [6]

6 Hardware

The purpose of this section is to discuss and explain the hardware used in the V-Tracker project.

6.1 Sensing on mobile devices

When it comes to building embedded systems, mobile devices and smartphones offer a range of advantages that primarily include: ARM CPU, power supply, battery management, WiFi support, cellular and mobile data support, rich user-interfaces (UI) and much more [7]. Furthermore, "mobile phones have matured as a computing platform and [have] acquired richer functionality, these advancements often have been paired with the introduction of new sensors" [8]. For example, a standard Apple iPhone 4 has eight different sensors: accelerometer, GPS, ambient light, dual microphones, proximity sensor, dual cameras, compass and gyroscope [8].

6.2 Motion sensors

V-Tracker uses web-technologies to access the mobile device's motion sensors. Using the Apple iPhone 4 as an example, this section will explain the basics behind these sensors.

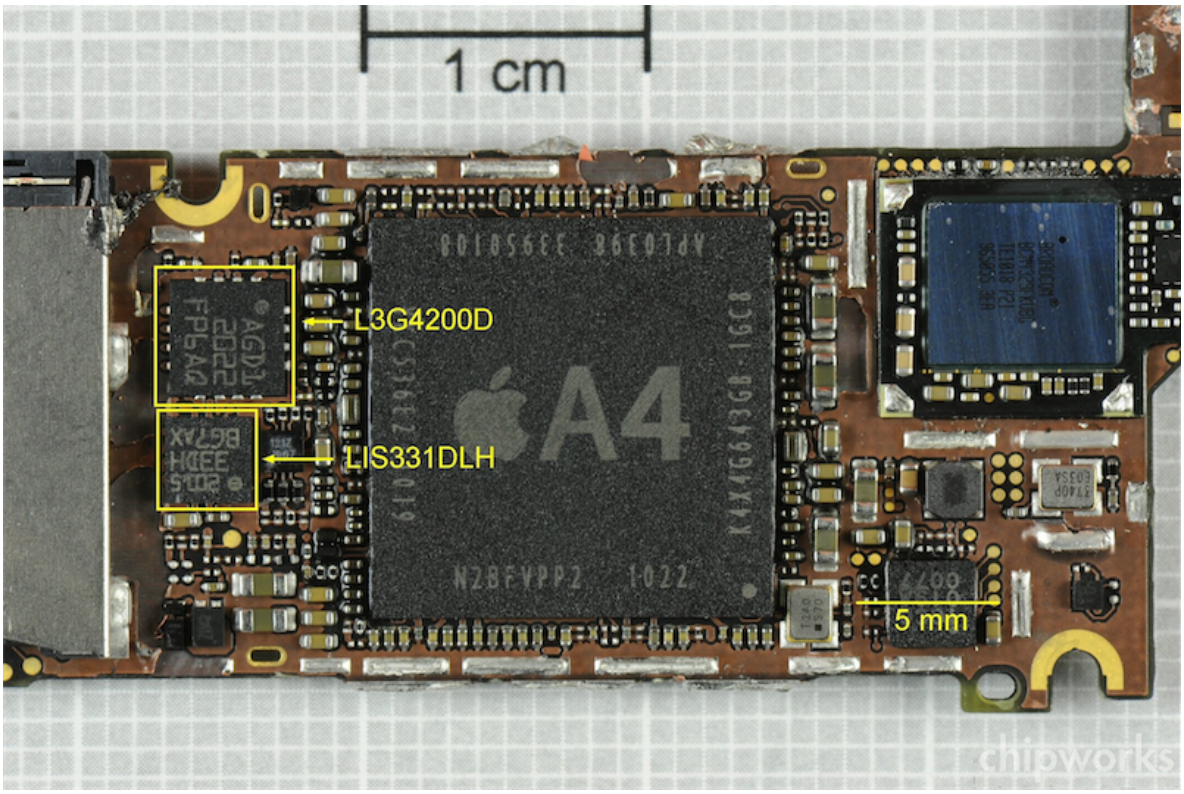


Figure 4: iPhone 4 main board with the STMicroelectronics LIS331DLH accelerometer and the L3G4200D gyroscope side-by-side [9]

6.2.1 Accelerometer

A piezoelectric accelerometer can be visualised as a mass suspended using springs. Variations in the physical properties of the springs are detected as the sensor's encompassing body moves. By monitoring the variations in the spring's physical properties, we are able to assert the body's proper-acceleration. In mobile devices, accelerometers are often used to sense the device's overall motion. The accelerometer in the iPhone 4 is a small three degrees-of-freedom (3 dof) STMicroelectronics LIS331DLH three-axis micro electro-mechanical system (MEMS) accelerometer, shown in Figure 4 above. The sensor's three degrees-of-freedom allow it to provide acceleration data on all three axes as shown in Figure 5. This sensor can be simply described as two cantilever beams that form capacitive plates, with a proof mass attached to the end of one of the beams [9]. Moving the device as a whole moves the cantilever plates closer or further apart. The change in capacitance between the plates creates a signal that is

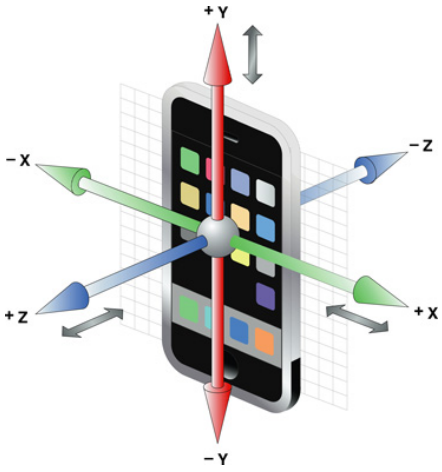


Figure 5: The 3 dof accelerometer on the iPhone 4 provides acceleration readings on the x, y and z axes [10]

measured using an on board Application Specific Integrated Circuit (ASIC) [9]. The ASIC is also used to maintain the spacing between the plates (i.e. the capacitance) by using capacitive feedback to maintain a constant DC bias across the plates [9].

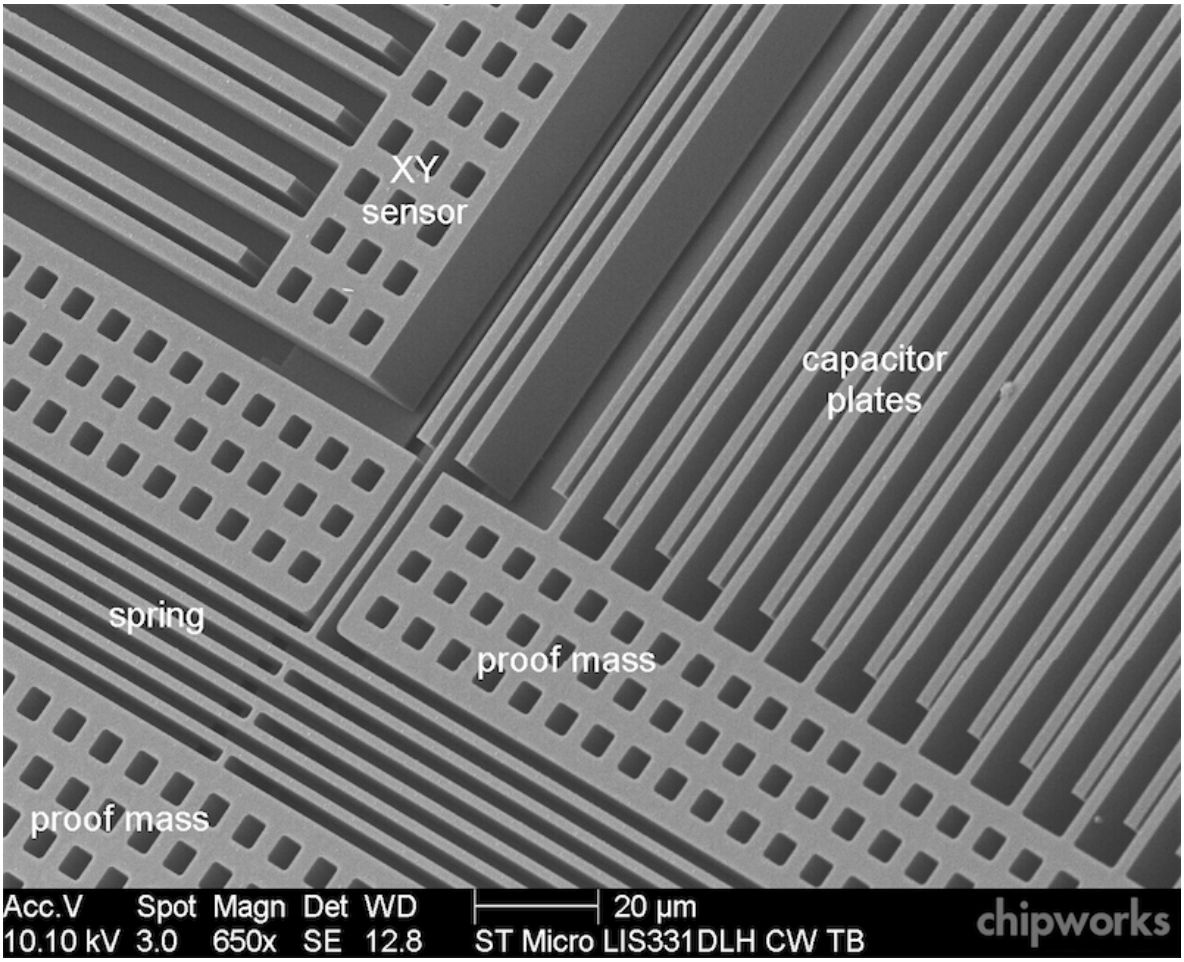


Figure 6: LIS331DLH XY accelerometer sensor detail [9]

6.2.1.1 Choosing an appropriate update interval

Selecting an interval that minimises the number of calls to the sensor will improve the device's battery life. It is therefore recommended that the update interval be appropriately set for the intended use case. Table 1 below is extracted directly from the Event Handling Guide for iOS and describes some of the more common update intervals chosen for acceleration events. Naturally, these intervals can be interpolated or extrapolated for unique uses. These intervals can also be specified for applications deployed on different mobile operating systems, provided of course that the device's sensor can respond at the frequencies specified.

Acceleration data is typically returned in the form of a three-dimensional vector with x-axis, y-axis and z-axis components. The acceleration data returned in iOS includes a gravity component, which can be isolated or filtered to suit specific applications.

Event frequency (Hz)	Usage
10-20	Suitable for use in determining the vector representing the current orientation of the device.
30-60	Suitable for games and other applications that use the accelerometers for real-time user input.
70-100	Suitable for applications that need to detect high-frequency motion. For example, you might use this interval to detect the user hitting the device or shaking it very quickly.

Table 1: Common update intervals for acceleration events in iOS [10]

6.2.1.2 Isolating the gravity component (low-pass filtering)

If you are using the accelerometer sensor to detect a device's orientation for example, it is suitable to filter out shakes and spikes from the acceleration data. In other words, in order to better detect a device's orientation, we only want the gravity component of each axis on the sensor. To do this we need to isolate the gravity component from the rest of the accelerometer data. The iOS Dev Center recommends a basic low-pass filter to reduce the effects of sudden jolts, shakes, or movements [10]. The filter can be implemented as follows:

```
// Use a basic low-pass filter to keep only the gravity component of each axis.
accelX = (acceleration.x * kFilteringFactor) + (accelX * (1.0 - kFilteringFactor));
accelY = (acceleration.y * kFilteringFactor) + (accelY * (1.0 - kFilteringFactor));
accelZ = (acceleration.z * kFilteringFactor) + (accelZ * (1.0 - kFilteringFactor));

// Use accelX, accelY, accelZ to do stuff…
```

Code snippet 1: Isolating the gravity component (low-pass filtering) [10]

The kFilteringFactor can be used to specify the sensitivity of the filter. A low-value filtering factor of 0.15 (or 15%) was found to be suitable for most applications in V-Tracker.

6.2.1.3 Isolating instantaneous motion (high-pass filtering)

If you are using the accelerometer to detect sudden movements such as shakes, it is recommended that a high-pass filter be used to reduce the effects of gravity [10]. The filter can be implemented using a technique similar to that in low-pass filtering.

```
// Subtract the low-pass value from the current value to get a simplified high-pass filter
accelX = acceleration.x - ( (acceleration.x * kFilteringFactor) + (accelX * (1.0 - kFilteringFactor)) );
accelY = acceleration.y - ( (acceleration.y * kFilteringFactor) + (accelY * (1.0 - kFilteringFactor)) );
accelZ = acceleration.z - ( (acceleration.z * kFilteringFactor) + (accelZ * (1.0 - kFilteringFactor)) );

// Use accelX, accelY, accelZ to do stuff…
```

Code snippet 2: Isolating instantaneous motion (high-pass filtering) [10]

6.2.2 Gyroscope

Unlike the accelerometer and compass, which rely on external forces, a gyroscope measures its own rotation. To do this, gyroscopes rely on the Coriolis effect or the

perceived deflection of a moving object when observed from a rotating frame of reference. The iPhone 4 is equipped with a MEMS three degrees-of-freedom STMicroelectronics L3G4200D gyroscope that uses the Coriolis effect [11].

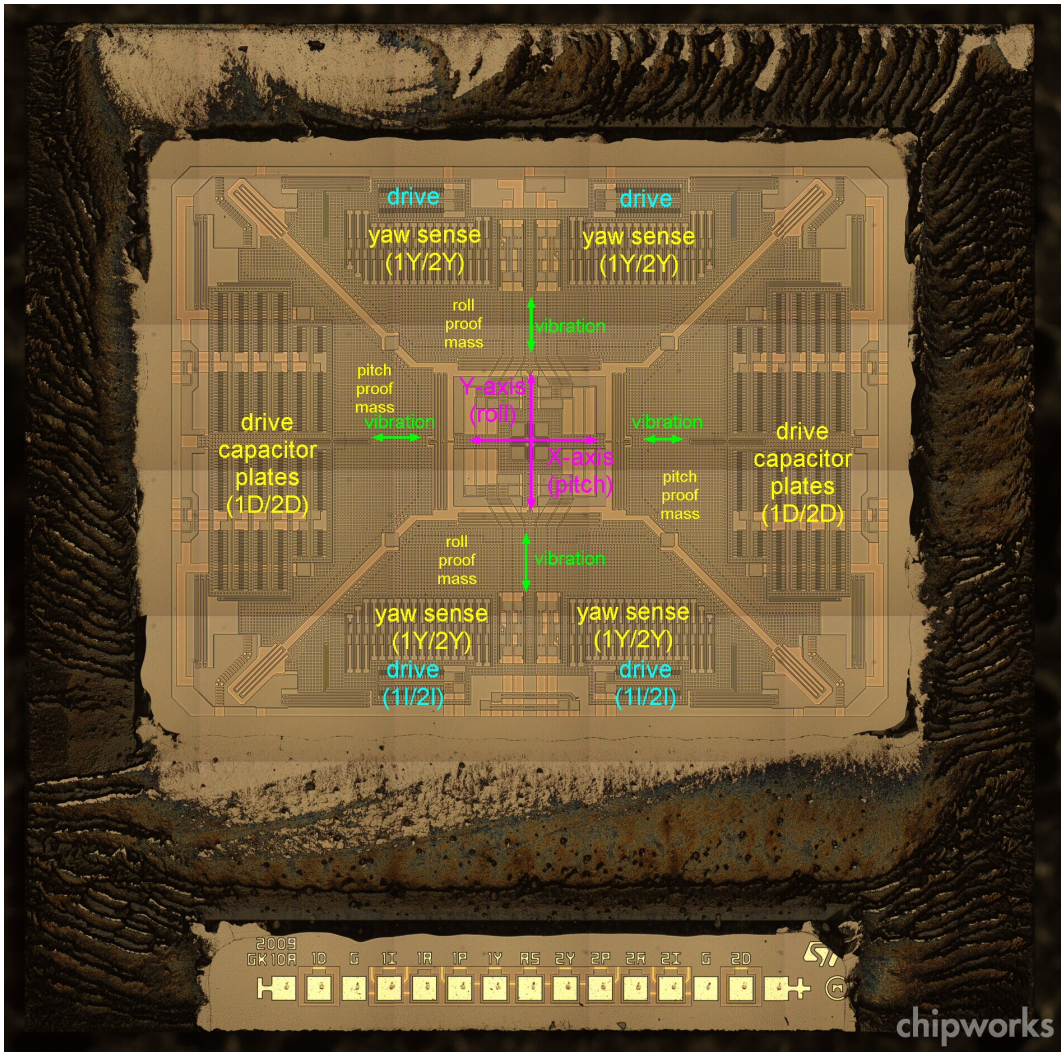


Figure 7: L3G4200D GK10A three-axis gyroscope die [11]

The figure above shows the die for the GK10A three-axis gyroscope. The drive capacitor plates seen on the figure are used to vibrate four proof mass elements. The vibration of the proof mass elements is managed via a spring that can be better seen in the figure below. As the device is rotated, differential out-of-plane deflection can be sensed by poly-silicon capacitor plates located beneath the proof mass elements [11].

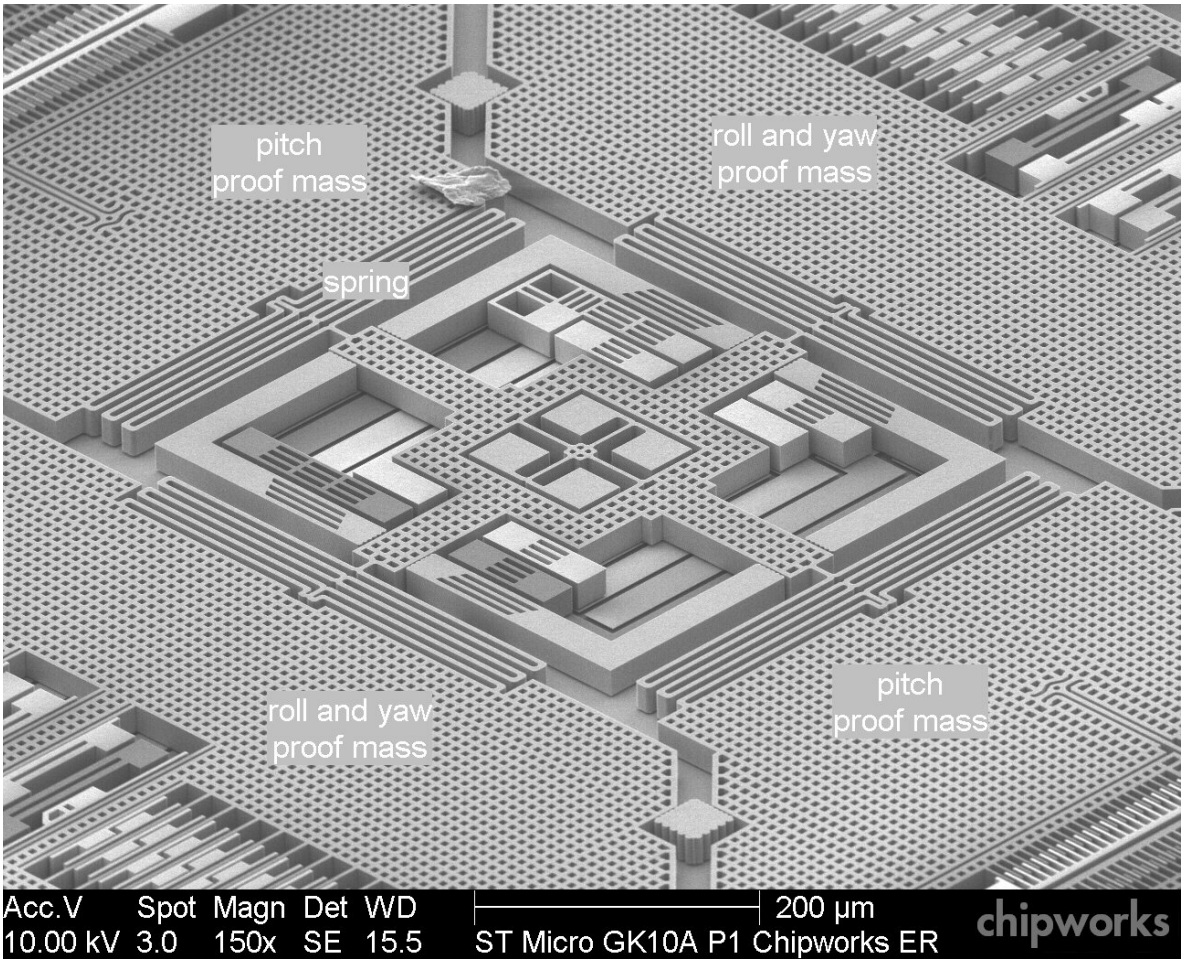


Figure 8: L3G4200D detail [11]

6.2.2.1 Asserting device orientation using the gyroscope

Gyroscopes in mobile devices only sense angular velocity or "the rate at which a device rotates around each of its spatial axes" [10]. In order to assert the device orientation using the gyroscope, it is necessary to rely on the accelerometer and compass (or magnetometer) sensors and utilise a series of filtering techniques to combine data from all three sensors. The accelerometer and compass (or magnetometer) are also used to regularly compensate for gyroscope drift. The three degrees-of-freedom gyroscope on the iPhone 4 provides angular velocity on all three axes. Like the accelerometer, the gyroscope uses the right-hand coordinate system shown in Figure 9 below. As a result, the positive angle of rotation can be determined using the right hand rule, which is visualised in Figure 10.

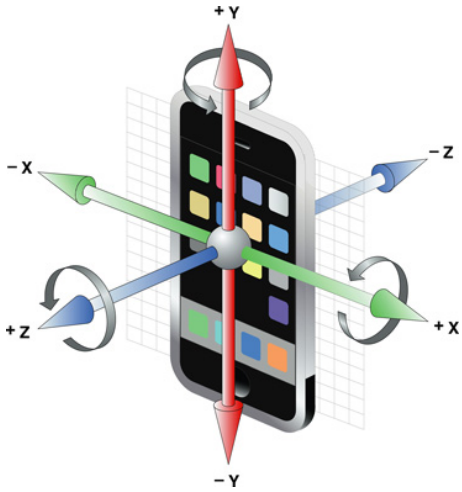


Figure 9: The 3 dof gyroscope on the iPhone 4 provides roll, pitch and yaw angular velocities [10]

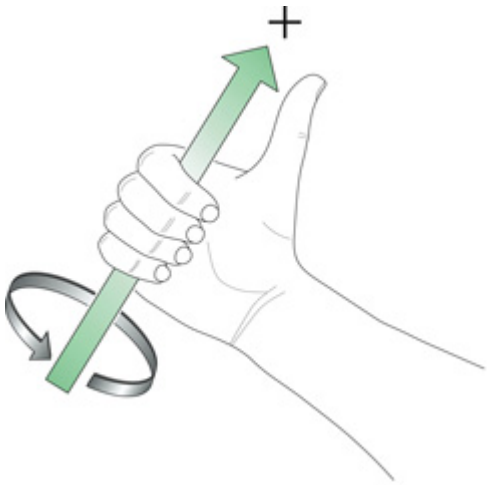


Figure 10: The direction of an axis' positive angular change can be found using the Right Hand Rule [10]

6.3 Localisation sensors

V-Tracker also uses web-technologies to access the mobile device's localisation sensors. Again using iOS and the Apple iPhone 4 as a guide, this section will explain the basics behind these sensors.

6.3.1 Compass

In mobile devices, the compass (or magnetometer) is used to assert Magnetic North. The iPhone 4 has an AKM AK8975/3 magnetic sensor that uses the Hall effect to sense its orientation relative to the earth's magnetic field [12]. A current is passed between the two contacts on the Hall sensors and the perpendicular effects of the Earth’s magnetic field are captured and processed [12]. As a result of the technique used for the magnetometer, it is common for the sensor to suffer from environmental interference and therefore requires regular calibration.

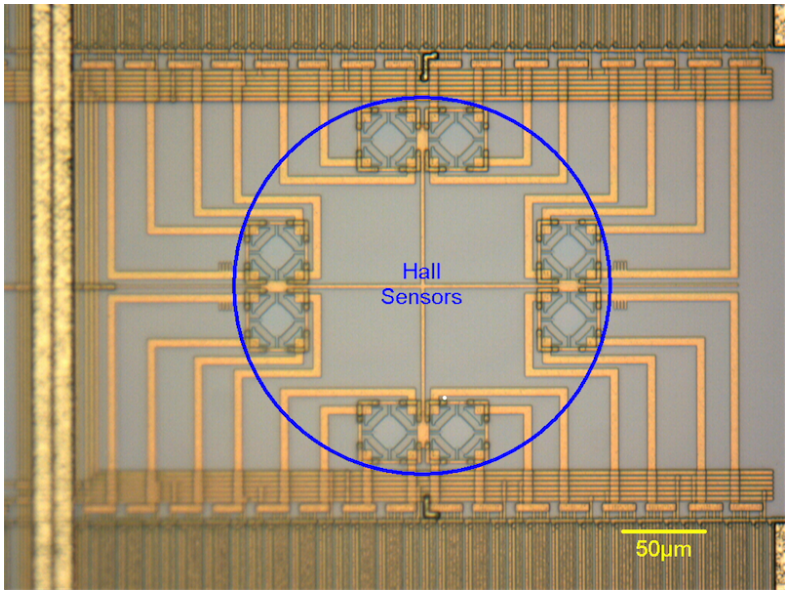


Figure 11: AK8973 Hall sensor layout [12]

The compass in the iPhone 4 is a three-axis sensor that can provide heading data regardless of the device's orientation in space. When combined with the accelerometer and gyroscope, these three sensors provide sensing information on nine degrees-of-freedom. The iPhone's magnetometer can be seen on the circuit board in the Figure below.

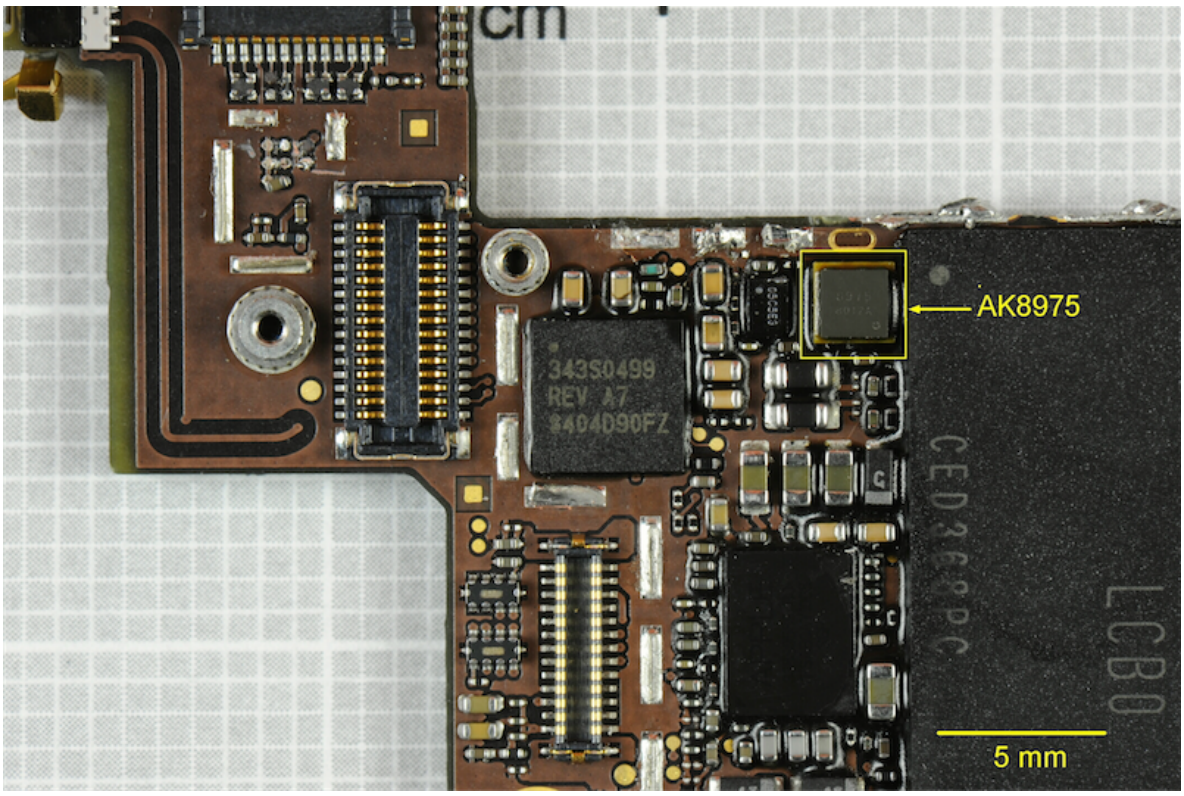


Figure 12: iPhone 4 main board with the AKM AK8975 electronic compass [12]

6.3.2 Global Position System (GPS)

Global Positioning System (GPS) receivers have become a common feature of modern mobile devices and form an integral part of V-Tracker. GPS and location services provide geographical context and allow developers to build location aware applications such as maps-supported navigation [13]. The Global Positioning System itself was setup by the U.S. Department of Defence in the 1970s and today consists of a constellation of 24 active satellites and 5 reserves [14]. Although originally intended for military use, the GPS is now available for civilian use with no subscription or setup fees [14]. Each GPS satellite travels on a precise orbit and its Ephemeris data (data that indicates where a satellite is throughout the day) is programmed into GPS receivers [14]. In order to find a user's location, a GPS receiver transmits a signal to its nearest satellites and measures the time taken for the transmitted signal to be returned from each satellite. The time differences can then be used to calculate the distance to each of the satellites pinged [14]. Fundamentally, GPS receivers are able to combine Ephemeris data and signal transmission time measurements with simple trilateration techniques to reliably assert a user's location to an accuracy radius of approximately 10m. A minimum of three satellites is required to determine a user's location with reasonable accuracy.

Several factors that contribute to errors in GPS signals do so by degrading the signal and affecting its accuracy [14]. Some of these factors include: ionosphere and troposphere delays, signal multipath, receiver clock errors, orbital errors, number of satellites visible, satellite geometry/shading and intentional degradation of the satellite signal [14].

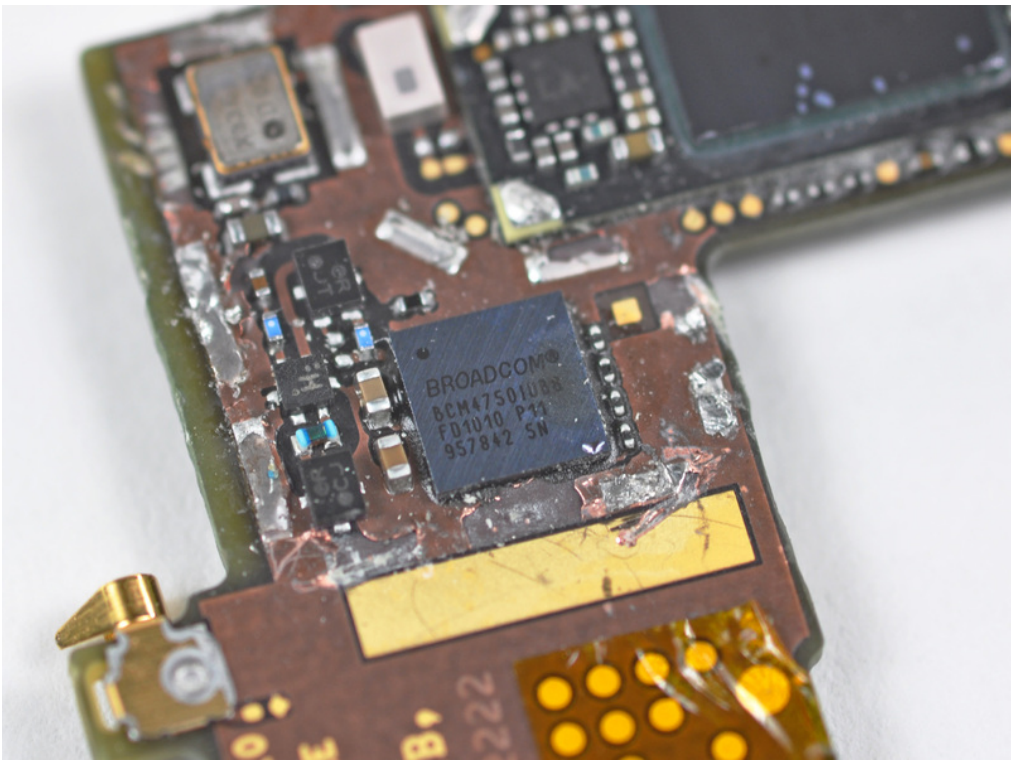


Figure 13: The Broadcom BCM4750IUB8 single-chip GPS receiver [15]

The iPhone 4 has a Broadcom BCM4750IUB8 single-chip GPS receiver [15], which can be seen in the figure above. The BCM4750 is a single-chip, single-die, low power Assisted-GPS (AGPS) solution that is optimised for mobile devices [16]. The receiver supports update rates of up to 2Hz [16]. Assisted-GPS chips are able to utilise alternative radios such as WiFi and GSM to improve the time-to-first-fix on GPS based positioning systems.

Figure 14: The iPhone 4's integrated UMTS, GSM, GPS, Wi-Fi and Bluetooth antennas on the stainless steel inner frame [15]

On the iPhone 4, Apple has integrated the UMTS, GSM, GPS, Wi-Fi and Bluetooth antennas into the stainless-steel inner frame [15]. This is shown the figure 14.

7 Software and Algorithms (Method)

The application, its encompassed algorithms and documentation form the primary deliverable for this project. This section will offer an overview of the computational logic behind some of the fundamental algorithms that drive V-Tracker. For a more detailed explanation of the software, please refer to the code documentation in the project wiki at http://www.github.com/hadimichael/V-Tracker.

7.1 Overview

The application uses a single HTML markup for all its pages. JS scripts and CSS classes provided in the jQuery and jQuery mobile frameworks provide the essentials for handling navigation and user interface (UI). The application's UI is broken down into three pages, screenshots of which are shown in the figures below. The UI does not use native elements and is instead designed to be platform agnostic – it takes on the same interface form on all iOS and Android devices. It is also designed to automatically scale with variations in screen size.

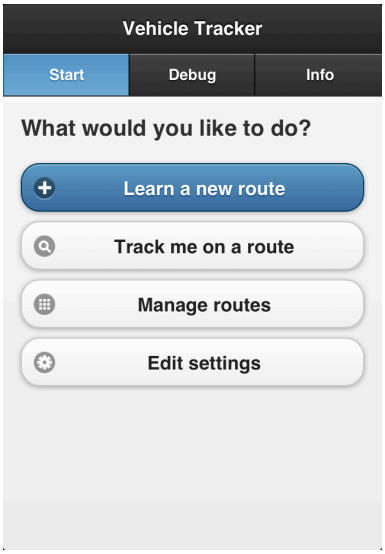


Figure 15: "Start" is the application's primary page

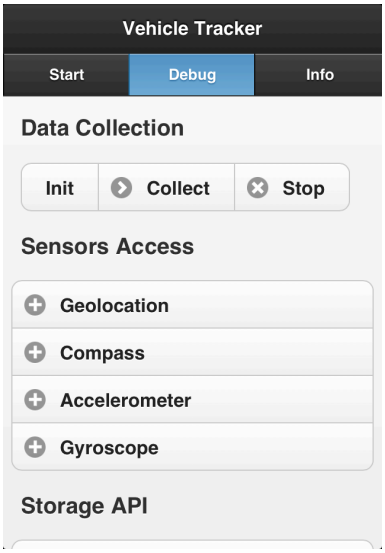


Figure 16: "Debug" provides direct access to individual sensors, storage and notifications API

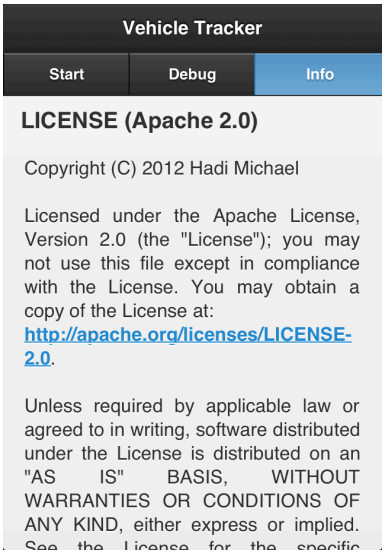


Figure 17: "Info" provides license and general project information

With the help of third party libraries, five principal JS scripts handle the application's key functionality:

1. *vtracker.js* – contains the application's core functionality. It controls the UI for the "Start" page and delegates all application tasks.
2. *debug.js* – provides testing and debugging access for the scripts used across the application. It also controls the UI for the "Debug" page.
3. *sensorsAPI.js* – is an extension wrapper for PhoneGap's device sensors API.
4. *storageAPI.js* – is an extension wrapper for PhoneGap's storage API.
5. *notificationsAPI.js* – is an extension wrapper for user notifications.

The code is well commented and explained in detail in the online documentation. Only key algorithms and the mathematics associated are explained herein.

7.2 The core application (vtracker.js)

The core application is handled in *vtracker.js*. This section will focus on the "route" object and the modelling and tracking algorithms that can be implemented for a route.

7.2.1 The "route" object

Every travel route in V-Tracker, which can be learnt, modelled and then used for tracking, is declared as a "route" object. By examining the route's properties and functions, we can easily establish a basic understanding of the characteristics that define a route.

Properties	Functions
•name •geoData •model •noiseThreshold •minAccuracy •learnCounter •timeoutLimit •trackingThreshold	•loadFromStored(storedRoute) •onLearningGeoMeasurement(measurement) •onTrackingGeoMeasurement(measurement) •onGeoMeasurementError(error) •recreateModel(callback) •getModelLength(from, to) •getRouteLength(from, to) •showModelOnPlot(divId) •showModelOnMap(divId, mapType) •replayModel(interval) •replayRoute(rate) •stopReplay() •learn() •stopLearning() •startTracking() •resetTracking() •stopTracking() •save() •exportRouteToDB() •exportModelToDB()

Table 2: The route object's properties and functions

The route object's properties are described as follows:

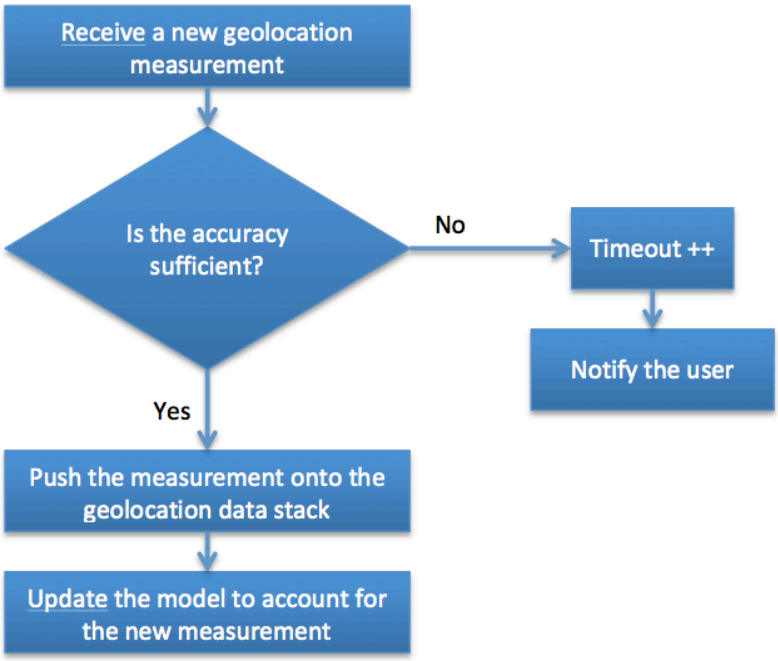
- *name* – is the route's name.
- *geoData* – is all the route's raw geolocation measurement data that has been captured during learning sessions. This data is ordered in chronological order and order matters. Each geolocation measurement consists of a: *timestamp, latitude, longitude, altitude, accuracy, altitudeAccuracy, heading, speed*.
- *model* – is the mathematical model generated for the route expressed using model control points. The model array consists of longitude and latitude control points, and an index indicating the extent of the raw geolocation data that is captured by the model. The *model* property contains: *lon, lat, index*.
  We use an index point, instead of matching the last model control point to the raw data, in order to avoid errors that can arise if the route has segments that overlap.
- *noiseThreshold* – is the threshold radius (in metres) that is used in the model to differentiate between what is considered to be noise fluctuation in measurements and what is considered to be an actual change in trajectory (default value = 2).

- *minAccuracy* – is the minimum GPS accuracy (in metres) that is considered to be sufficient for learning (default value = 50).
- *learnCounter* – is a counter that keeps track of how many times a route has been learnt.
- *timeoutLimit* – is the limit at which we decide that the accuracy of the data being measured is not going to improve and that we should take some action (default value = 10).
- *trackingThreshold* – is the threshold distance (in metres) used in tracking to decide if the user is still on the route or has deviated (default value = 50).

While the functions can be considered to be self-explanatory, some of the key algorithms are worth exploring in further detail. These are the functions particularly relating to the mathematical modelling and tracking of vehicles on a route.

7.2.2 Route modelling (algorithm)

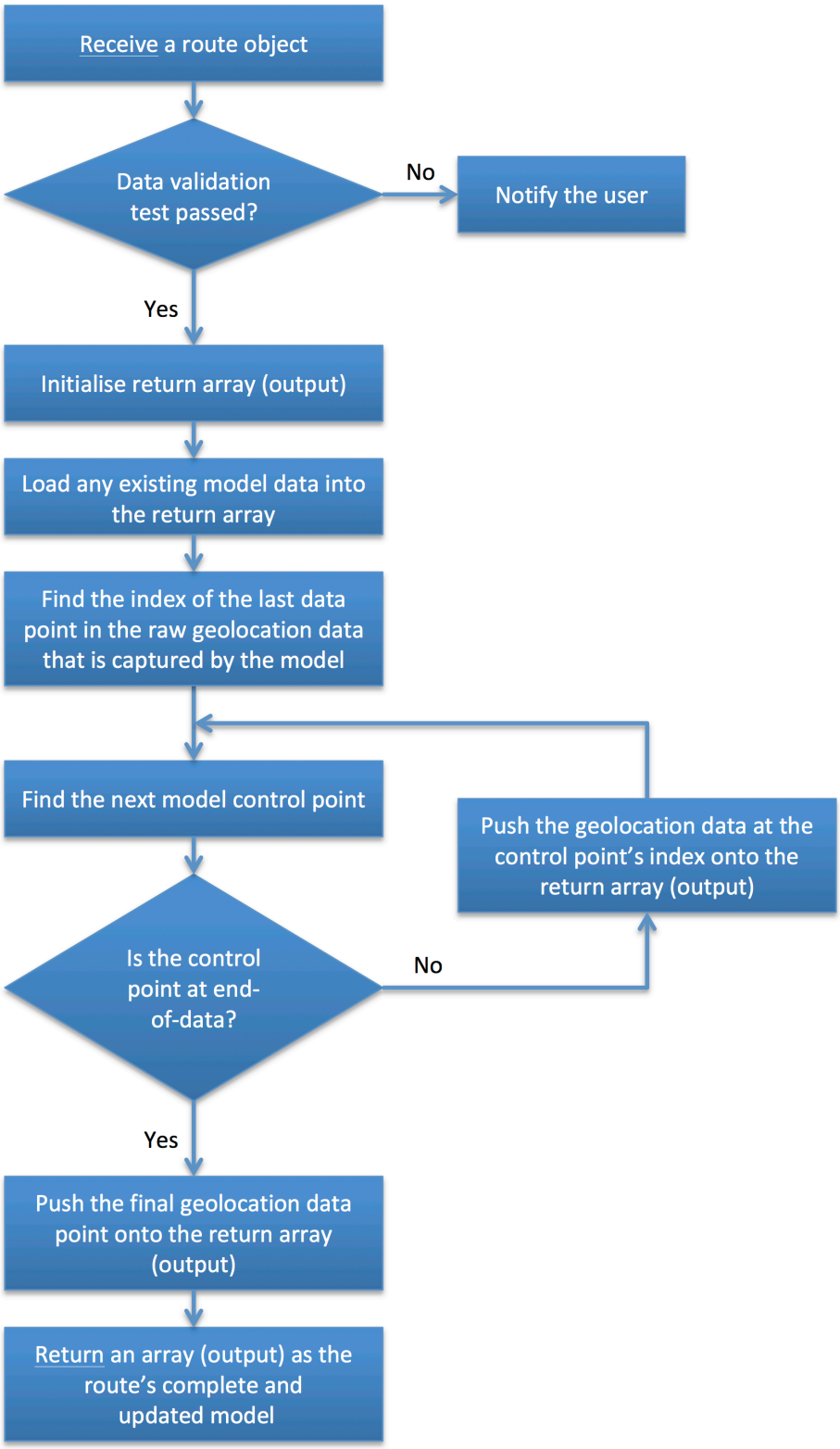
Route modelling takes place within milliseconds of data being collected or in other words, it takes place in realtime. When a user asks the application to learn a new route, the route object calls the geolocation API to start tracking GPS signals and registers an on-success callback function called "onLearningGeoMeasurement". The logic flow for the callback function is illustrated in the flow chart below:



Flow chart 1: Logic flow for "onLearningGeoMeasurement" function

Every time a new geolocation measurement is received, a quick check asserts that the measurement's accuracy is sufficient for learning. This precaution is taken to reduce the impact of drastically noisy data. If the data accuracy is considered to be insufficient, a timeout counter is incremented and the user is notified. Alternatively, if the data accuracy is sufficient, the measurement is pushed onto the geolocation data stack and the route is

passed to the modelling function for processing. The logic flow for the modelling function is presented in the flow chart below:



Flow chart 2: Logic flow for the modelling algorithm

Upon receiving a new geolocation measurement, a short test is conducted to ensure that the data structure of the geolocation stack is suitable for modelling. Once the test is passed, an output array is initialised and any existing model data is loaded into it to avoid processing data points that have already been modelled. At this stage a series of mathematical formulations calculate the model control points and the route's model object array is updated and returned to the route object.

7.2.3 Route modelling (mathematics)

We must undertake a series of mathematical transformations to find the model's control points. We start with a matrix of raw geolocation measurements that includes every measurement taken on the route thus far:

$$measurements = \begin{bmatrix} \emptyset_0 & \lambda_0 \\ \emptyset_1 & \lambda_1 \\ \emptyset_2 & \lambda_2 \\ \vdots & \vdots \\ \emptyset_n & \lambda_n \end{bmatrix}$$

Where:

$$\phi_{0\ldots n}\ is\ latitude\ in\ degrees$$
$$\lambda_{0\ldots n}\ is\ longitude\ in\ degrees$$

The Haversine formula dictates that the spherical distance (d) between two points $(\emptyset_a, \lambda_a)$ and $(\emptyset_b, \lambda_b)$ on a sphere is:

$$d = 2r \sin^{-1}\left(\sqrt{\sin^2\left(\frac{\phi_b - \phi_a}{2}\right) + \cos(\phi_a)\cos(\phi_b)\sin^2\left(\frac{\lambda_b - \lambda_a}{2}\right)}\right)$$

Where:

$$r\ is\ the\ radius\ of\ the\ sphere$$

Now we use the Haversine formula to transform the latitude and longitude measurements into two-dimensional Cartesian coordinates using $(\emptyset_0, \lambda_0)$ as the point of origin. We do this by finding:

$$\forall\, i \in Z_n : x_i = 2r \sin^{-1}\left(\sqrt{\sin^2\left(\frac{\phi_0 - \phi_o}{2}\right) + \cos(\phi_o)\cos(\phi_o)\sin^2\left(\frac{\lambda_i - \lambda_0}{2}\right)}\right)$$
$$= 2r \sin^{-1}\left(\sqrt{\cos^2(\phi_0)\sin^2\left(\frac{\lambda_i - \lambda_0}{2}\right)}\right)$$

and

$$\forall\, i \in Z_n : y_i = 2r \sin^{-1}\left(\sqrt{\sin^2\left(\frac{\phi_i - \phi_o}{2}\right) + \cos(\phi_i)\cos(\phi_o)\sin^2\left(\frac{\lambda_0 - \lambda_0}{2}\right)}\right)$$
$$= 2r \sin^{-1}\left(\sqrt{\sin^2\left(\frac{\phi_i - \phi_o}{2}\right)}\right)$$

Where the mean radius of the earth is specified as:

$$r = 6371000\ (metres)$$

We now have:

$$transformed\ data = \begin{bmatrix} x_0 & y_0 \\ x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix}$$

This is a matrix that we can use to start identifying the model's control points. To do this, we begin by including the first data point in $measurements$ (i.e. the point of origin) as the first model control point:

$$model = \begin{bmatrix} \emptyset_0 & \lambda_0 \end{bmatrix}$$

Subsequently we determine the next model control point. In particular, we are interested in finding the next point in the transformed matrix (point i in $transformed\ data$), where the root-mean-square-errors (RMSE) of the straight line fit between (x_0, y_0) and (x_i, y_i) equates to, or exceeds, the predefined "noise threshold". We do this by incrementally testing each point $(\forall\, i \in Z_n)$ in $transformed\ data$, starting at (x_1, y_1), until RMSE reaches the "noise threshold".

In physical terms, the "noise threshold" represents the radius (in metres) of the boundary line that is used to determine whether the fluctuation in the intermediary points (i.e. the residuals) can be ignored as noise or not. A typical "noise threshold" radius of 2 metres was found to be suitable for most routes.

The errors necessary for this modelling approach are calculated as the shortest perpendicular distance between the modelled line and the point of interest. To find this distance, we begin by first defining two vectors. The first $\overrightarrow{v_i}$ defines the straight line between the origin and the current point in the transformed data at which we want to test the RMSEs. The second $\overrightarrow{v_j}$ is the straight line between the origin and the current intermediary point of interest for which we want to find the residual error value.

$$\overrightarrow{v_i} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad \text{and} \quad \overrightarrow{v_j} = \begin{bmatrix} x_j \\ y_j \end{bmatrix}$$

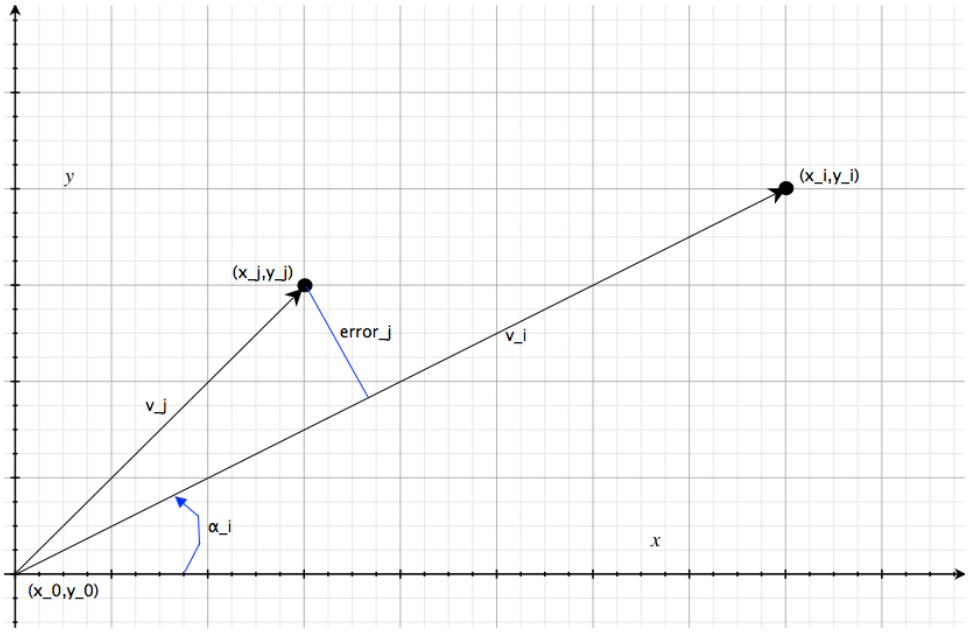


Figure 18: Plot demonstrating $\overrightarrow{v_i}$ and $\overrightarrow{v_j}$

The next step is to calculate the counter-clockwise rotation angle of $\overrightarrow{v_i}$ using the atan2 variation of arctangent:

$$\alpha_i = atan2(y_i, x_i)$$

α_i in this case describes the rotation angle of the current modelled line around the z-axis. We can now use this angle to rotate the second vector, $\overrightarrow{v_j}$, around the z-axis such that $\overrightarrow{v_i}$ aligns with the x-axis. To do this we apply a rotation around the z-axis of negative α using the rotation matrix $R_z(-\alpha_i)$.

$$\overrightarrow{v_j}' = R_z(-\alpha_i) \cdot \overrightarrow{v_j}$$

Where:

$$R_z(-\alpha_i) = \begin{bmatrix} \cos(-\alpha_i) & -\sin(-\alpha_i) \\ \sin(-\alpha_i) & \cos(-\alpha_i) \end{bmatrix}$$

Therefore:

$$\begin{bmatrix} x_j' \\ y_j' \end{bmatrix} = \begin{bmatrix} \cos(-\alpha_i) & -\sin(-\alpha_i) \\ \sin(-\alpha_i) & \cos(-\alpha_i) \end{bmatrix} \cdot \begin{bmatrix} x_j \\ y_j \end{bmatrix}$$

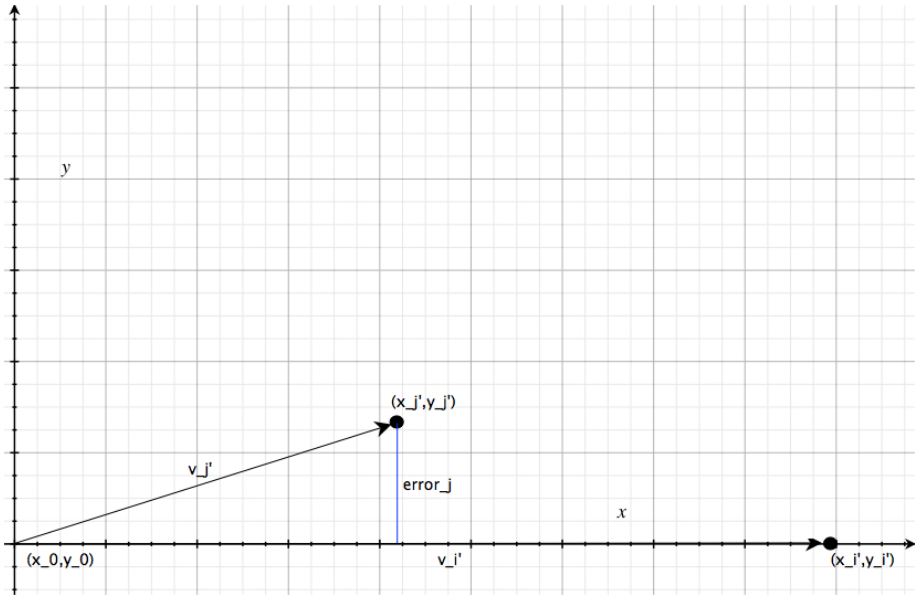


Figure 19: Plot demonstrating $\overrightarrow{v_i}'$ and $\overrightarrow{v_j}'$

We are only interested in the vertical component (the $y_j{}'$ component) of this rotation. Therefore we can find the error as:

$$y_j' = error_j = x_j \sin(-\alpha_i) + y_j \cos(-\alpha_i)$$

Finally, the next model control point (point i) in the transformed matrix, can be found using the following approach:

$$\forall\, i \in Z_n : RMSE_i = \sqrt{\frac{\sum_{j=1}^{i-1} error_j{}^2}{(i-1)}}$$

$$= \sqrt{\frac{\sum_{j=1}^{i-1} [x_j \sin(-atan2(y_i, x_i)) + y_j \cos(-atan2(y_i, x_i))]^2}{(i-1)}}$$

Where the conditional test at every increment of i is: $RMSE_i < noise\ threshold$.

As soon as the condition is violated, let us assume that this occurs at iteration $i = k$, we stop testing and add the k^{th} point of the $measurements$ matrix onto the $model$ matrix, such that it now looks like:

$$model = \begin{bmatrix} \emptyset_0 & \lambda_0 \\ \emptyset_k & \lambda_k \end{bmatrix}$$

In other words, we use k as an index pointer to indicate the specific point in $measurements$ that should be included as a model control point. Throughout the whole modelling process, we use an integer-incremented counter variable called the

$model\ index$ to indicate the extent of the raw geolocation data that has been captured by our model. At this stage:

$$model\ index = k$$

The general process is repeated until we reach the end of the $measurements$ matrix and have found all the model control points necessary such that: $i = n - 1$. At this stage, we can push on the n^{th} point of the $measurement$ matrix onto the $model$ matrix and are left with a complete model:

$$model = \begin{bmatrix} \emptyset_0 & \lambda_0 \\ \emptyset_k & \lambda_k \\ \vdots & \vdots \\ \emptyset_n & \lambda_n \end{bmatrix}$$

When a new geolocation measurement is captured, we pop off (or remove) the last control point in the model and repeat the process as above. However this time, to avoid processing data that has already been captured by the model, we start modelling from $(x_{model\ index}, y_{model\ index})$ as the point of origin.

7.2.4 Route tracking (algorithm)

Route tracking takes place in realtime whilst the user is on the route. The user's location information is also visualised in realtime, as shown in the figures below:

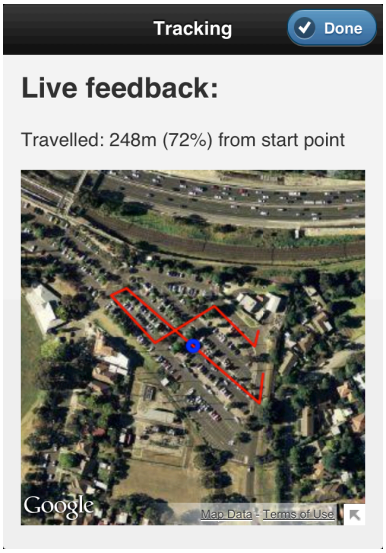


Figure 20: Route visualisation on a satellite map showing 72% travelled

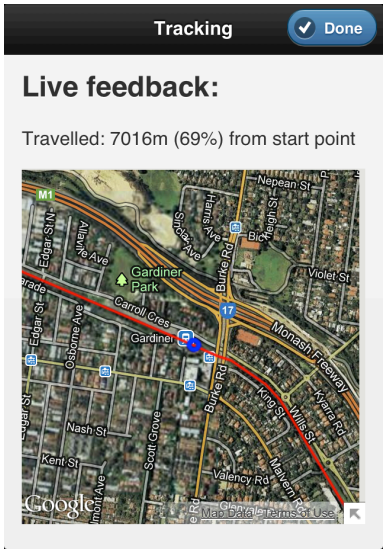
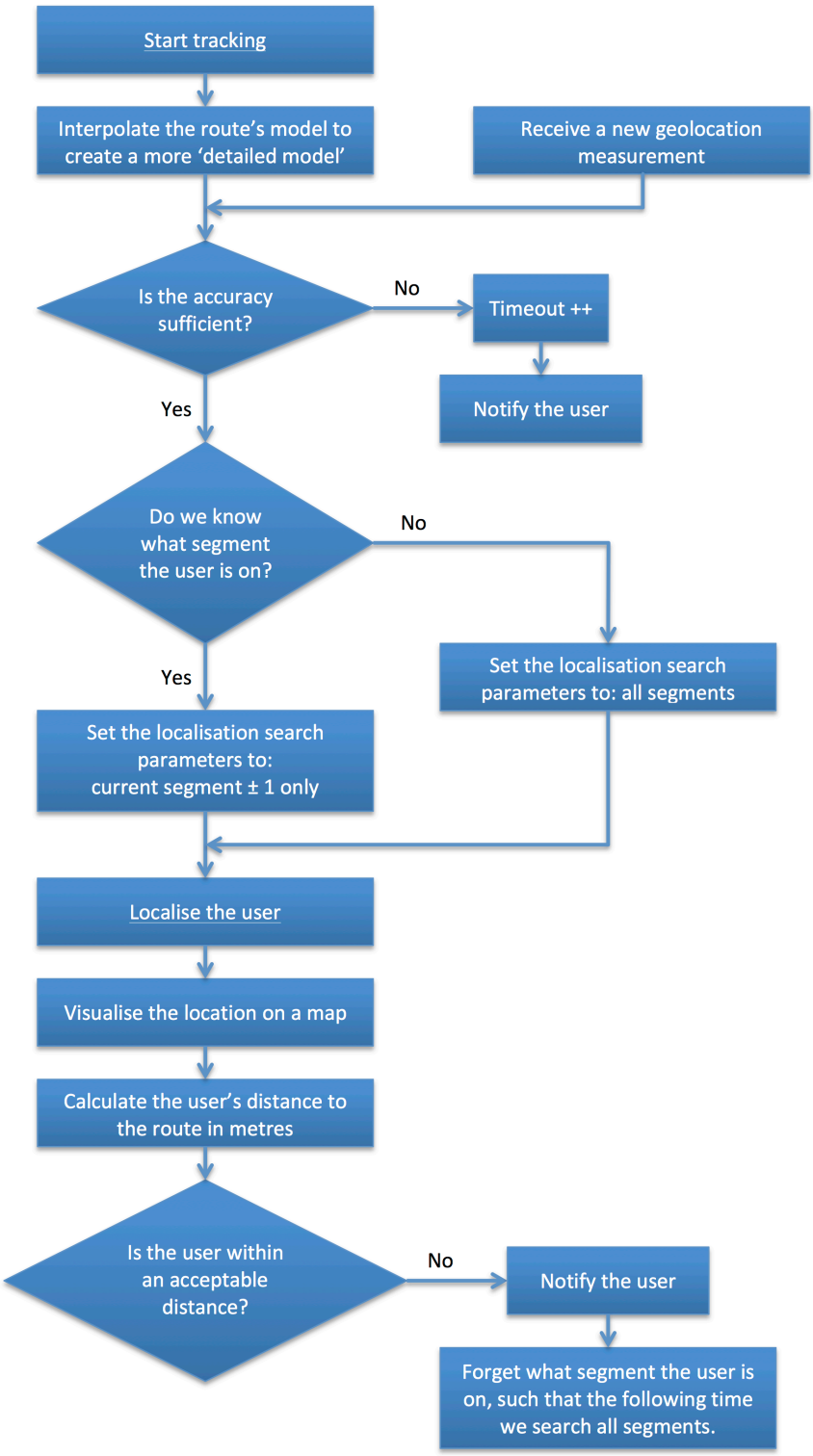


Figure 21: Route visualisation on a hybrid map showing 69% travelled

Vehicle tracking makes the fundamental assumption that the device running V-Tracker is on board the vehicle. When a user asks the application to start tracking them on an already learnt route, the route object begins by interpolating the model data to create a more 'detailed model'. After that, the route object calls the geolocation API to start tracking GPS signals and registers an on-success callback function called

"onTrackingGeoMeasurement". The logic flow for the tracking algorithm is presented in the flow chart below:



Flow chart 3: Logic flow for the tracking algorithm

7.2.5 Route tracking (mathematics)

We must conduct a series of mathematical calculations to localise a user on a route. We start with the model matrix containing all of the model's control points:

$$model = \begin{bmatrix} \emptyset_0 & \lambda_0 \\ \emptyset_1 & \lambda_1 \\ \emptyset_2 & \lambda_2 \\ \vdots & \vdots \\ \emptyset_n & \lambda_n \end{bmatrix}$$

Where:

$$\phi_{0\ldots n}\ is\ latitude\ in\ degrees$$
$$\lambda_{0\ldots n}\ is\ longitude\ in\ degrees$$

A vector that joins two consecutive model control points on a route is defined as a *segment*. An example of a segment would be the straight-line model that connects control points $(\emptyset_1, \lambda_1)$ to $(\emptyset_2, \lambda_2)$. Each *segment* consists of m number of *sub-segments*. In order to accurately track a user, we use the route's existing model to generate a $detailed\ model$, which consists of the same *segments* as well as additional *sub-segments*. We do this by creating a new matrix with linearly spaced vectors between the control points:

$$detailed\ model = \begin{bmatrix} \emptyset_{0,0} & \lambda_{0,0} \\ \emptyset_{0,1} & \lambda_{0,1} \\ \emptyset_{0,2} & \lambda_{0,2} \\ \vdots & \vdots \\ \emptyset_{0,m} & \lambda_{0,m} \\ \emptyset_{1,0} & \lambda_{1,0} \\ \emptyset_{1,1} & \lambda_{1,1} \\ \vdots & \vdots \\ \emptyset_{n,m} & \lambda_{n,m} \end{bmatrix}$$

Further calculations only take place upon receipt of a new geolocation measurement. When we receive a new geolocation measurement indicating the user's location at time t, provided that the accuracy is sufficient, we set up a $user's\ location_t$ matrix:

$$user's\ location_t = \begin{bmatrix} \emptyset_t & \lambda_t \end{bmatrix}$$

The next step is to specify the localisation search parameters using the $current\ segment\ index$. The localisation search parameters are the parameters that will tell the application where to search when localising the user. Throughout the tracking process we use a variable called the $current\ segment\ index$ as a buffer to remember the segment that we think the user is currently located on. Using this variable we can use a Markov chain to assert that a user can only move onto the next or previous segment.

Previous Segment (csi-1) <- Current Segment (csi) -> Next Segment (csi+1)

Using a Markov chain allows us to overcome localisation confusion that may occur on routes that have independent segments that are very close or intersecting.

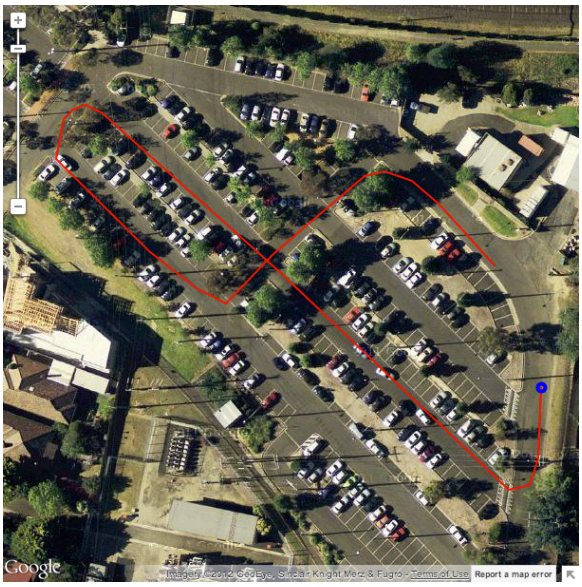


Figure 22: A figure-eight route modelled using a 1m noise threshold and visualised on a satellite map

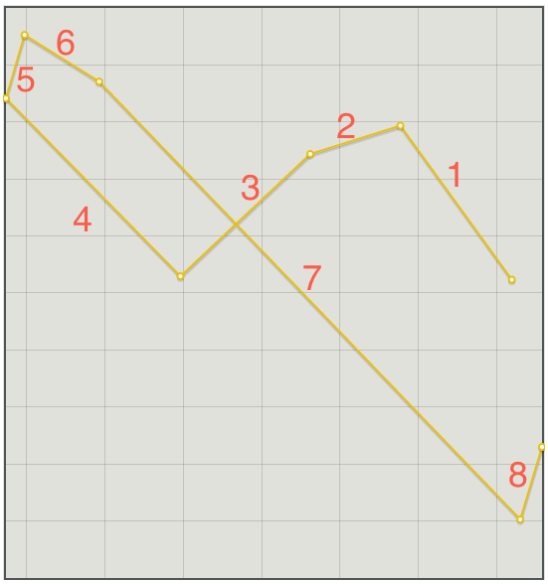


Figure 23: A figure-eight route modelled using a 2m noise threshold and visualised on a plot

By observing the figure-eight route shown above, it is noticeable that the 3rd and 7th segments intersect. By making the Markov chain assumption, we can assert that if a user is currently located on the 3rd segment ($current\ segment\ index = 3$) then it is expected that they would move onto the 4th ($current\ segment\ index + 1$) or 2nd ($current\ segment\ index - 1$) segments only, and not the 7th. A comparable scenario would exist if the user were located on the 7th segment; they would be expected to move onto the 6th or 8th segments only.

If we already know which segment a user is on and have a value for the $current\ segment\ index$, then we define the search parameters to include:

$$search \rightarrow current\ segment\ index \pm 1\ segment$$

If we do not know which segment a user is on and the $current\ segment\ index$ is $null$ or $undefined$, then we specify the search parameters to include all segments. This search case is common when we are initialising the segment index for the first time:

$$search \rightarrow all\ segments$$

The Euclidean distance between two points, p and q, is defined as the length of the straight-line segment that connects them ($\overline{pq}$). If we have two points with the Cartesian coordinates $p: (p_x, p_y)$ and $q: (q_x, q_y)$ in a two-dimensional Euclidean plane, the

Euclidean distance d between those points can be found using a formula equivalent to the Pythagorean theorem:

$$d(p,q) = d(q,p) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

Since we are only interested in the *relative* distance between the points, we assume that the geolocation measurements exist on a two-dimensional Euclidean plane. This assumption allows us to perform calculations using the $detailed\ model$ without having to transform it into Cartesian space.

We use the specified search parameters to extract a search matrix as a subset of the $detailed\ model$. In cases where the search parameter includes $all\ segments$, the search matrix is identical to the $detailed\ model$. In alternative cases, we end up with a $search$ matrix as follows:

$$search_t = \begin{bmatrix} \emptyset_{csi-1,0} & \lambda_{csi-1,0} \\ \emptyset_{csi-1,1} & \lambda_{csi-1,1} \\ \vdots & \vdots \\ \emptyset_{csi-1,m} & \lambda_{csi-1,m} \\ \emptyset_{csi,0} & \lambda_{csi,0} \\ \emptyset_{csi,1} & \lambda_{csi,1} \\ \vdots & \vdots \\ \emptyset_{csi,m} & \lambda_{csi,m} \\ \emptyset_{csi+1,0} & \lambda_{csi+1,0} \\ \emptyset_{csi+1,1} & \lambda_{csi+1,1} \\ \vdots & \vdots \\ \emptyset_{csi+1,m} & \lambda_{csi+1,m} \end{bmatrix}$$

Where:

$$csi\ is\ the\ current\ segment\ index$$

Once we have a matrix that defines where a user is expected to be, we find the point in that matrix that is nearest to the user using the Euclidean distance approach and then clip the user to that point on the route.

$$\forall\ csi \in search\ parameters: d_t = \sqrt{(\emptyset_{csi} - \emptyset_t)^2 + (\lambda_{csi} - \lambda_t)^2}$$

Once the shortest distance is determined, the Haversine formula is used to calculate the user's distance to the route in metres. This distance calculation is then compared to a threshold distance to test whether the user has significantly deviated from the route. If the user's distance exceeds the predefined threshold, a notification is issued to alert the user and the $current\ segment\ index$ is reset to $null$ signifying that we are no longer certain of the user's location on the route.

Finally, again using the Haversine formula the application estimates a distance value and percentage of travel on the route. This is then visualised on the device in realtime, along with a map visualisation as shown in the screenshots earlier.

7.3 Known limitations

As with most computational analysis there are limitations that must be accounted for when specifying parameters and using an application. Two key limitations of V-Tracker are discussed in this section, one algorithmic and one physical.

7.3.1 Sharp corners (algorithmic)

The first known limitation is the modelling algorithm's approach to handling sharp corners. Sharp corners are defined as corners with angles equal to or greater than 90°. The limitation can be observed in Figures 23 and 24. Another example of a route with sharp corners would be one that contains a U-turn (or a 180° corner). In order to adjust for this limitation, it is recommended that a smaller (less than 2 metres) noise threshold radius be specified for routes that contain several sharp corners.

7.3.2 Live map visualisation (physical)

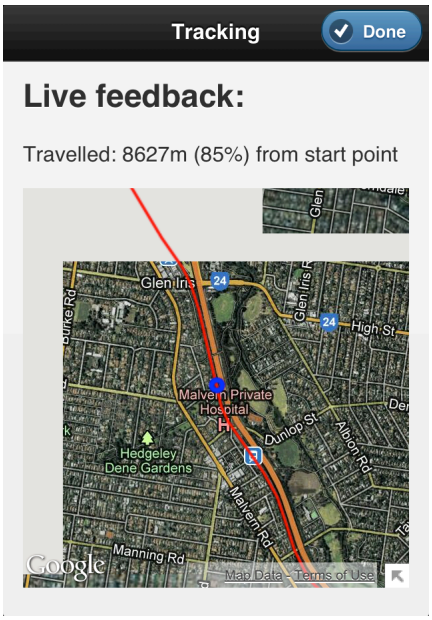


Figure 24: Screenshot showing the effects of disruptive MDS whilst tracking on a train route

Due to the nature of tracking, it is typical that the user be 'on the move' during the process. Despite continuous improvements in mobile networks and cellular infrastructure, users may still face intermittent network connectivity and disrupted data services as their devices switch between different cellular towers.

As a result of disrupted MDS, live map visualisation may be inconsistent and map tiles may not load in time. The effect of this limitation is shown in the screenshot in Figure 27.

Using offline maps or preloading the map content for a specific route can help mediate this limitation in the future.

7.4 PhoneGap Plugins

PhoneGap provides basic access to native functionality, however in order to achieve realtime route learning and vehicle tracking using web-technologies it was necessary to expand the available functionality. It is acknowledged that some of this additional functionality may be of use to other engineers and as a result, two free open-source PhoneGap plugins have been developed and released online for the developer community to use in alternative applications. The plugins were released for free under a completely open source MIT license.

7.4.1 Plugin 1: SensorsManager

A JS sensors manager based on *sensorsAPI.js* has been released as a plugin for PhoneGap. The sensors manager expands on existing PhoneGap API and provides access to some of the more advanced device-sensors functionality that was developed for this project.

7.4.2 Plugin 2: QuickStorage

A JS storage script based on *storageAPI.js* has been released as a plugin for PhoneGap. The QuickStorage plugin uses existing PhoneGap API to provide easier access to some of the more common SQLite3 and localstorage functions.

8 Experimental Results and Discussion

Several experiments were conducted to demonstrate V-Tracker's versatile use. The results and observations from some of the key experiments are shown and discussed in this section. The blue circle visible on the figures in this section is the indicator used in vehicle tracking to indicate a user's location on the route. In post-processing, the same blue indicator is used to replay the route or model.

8.1 On a train railway line

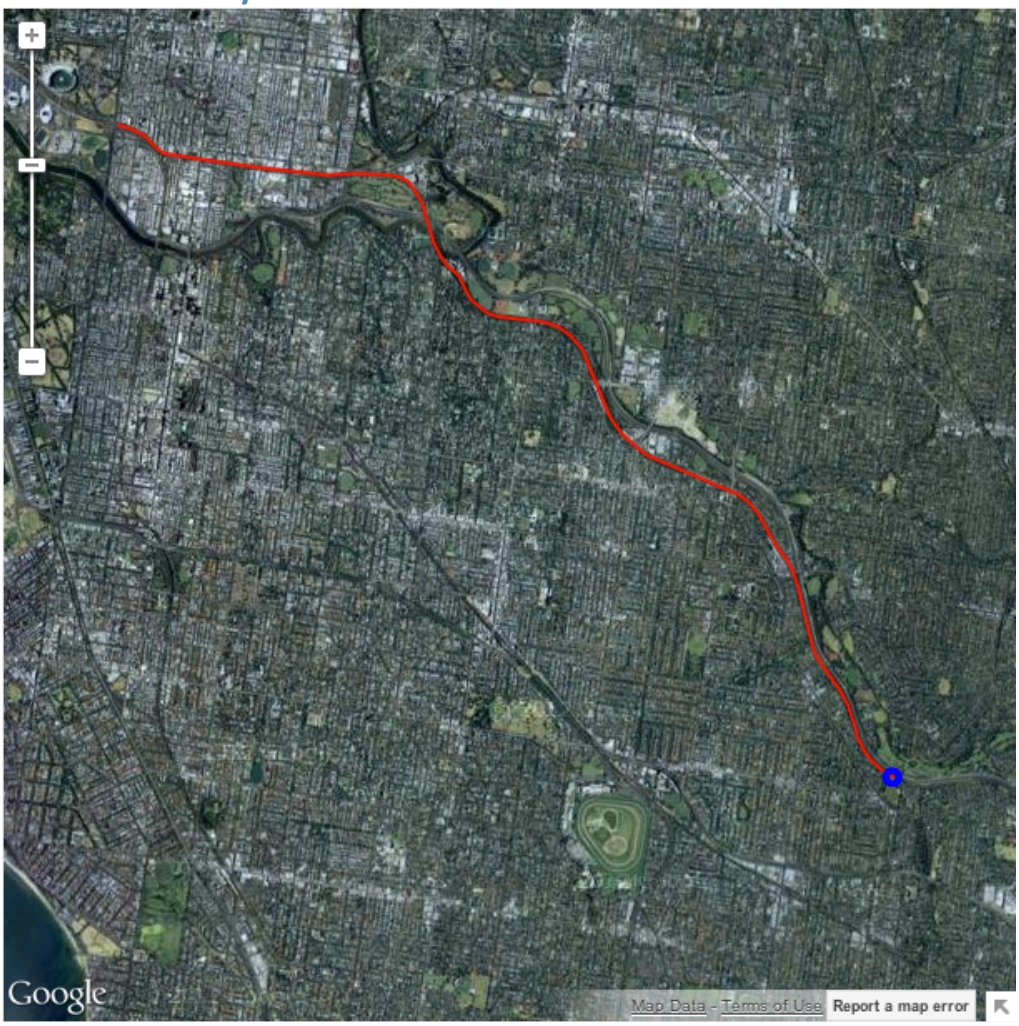


Figure 25: Model visualised for the Glen Waverley railway line in Melbourne's southeast

The figure above is the result of modelling a route captured whilst travelling on the Glen Waverley railway line in Melbourne's southeast. The Glen Waverley line was chosen for its unique wavy and irregular shape. This experiment demonstrates the modelling algorithm's reliability in capturing route data measured on a railway line.

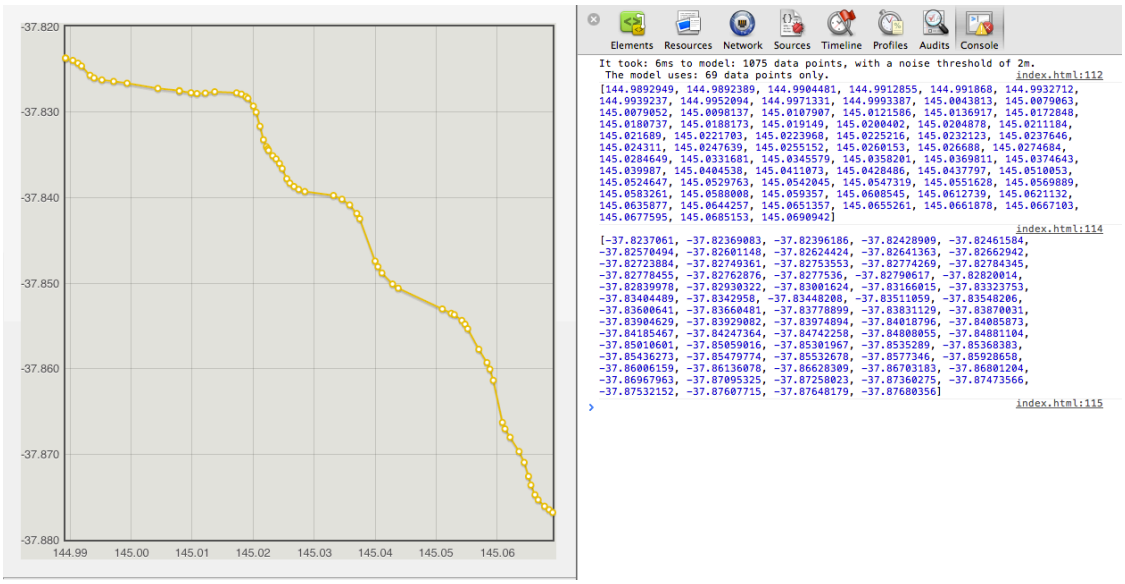


Figure 26: Screenshot of the modelling process running in a computer web-browser (Google Chrome)

The screenshot above shows a plot of the route's model as produced in post-processing. The console window provides feedback for the modelling process. As shown, when running in a computer web-browser (Google Chrome), the process took 6ms to model 1075 data points with a noise threshold radius of 2m. This was found to be sufficient for most railway applications. The final model consists of only 69 model control points. The console window shows the final control points selected for the model; these are shown in blue in the figure above.

8.2 In flight (using an aerial drone)

Some experiments were conducted with the mobile device attached to the frame of an unmanned aerial vehicle (UAV). The raw geolocation route measurements and the model generated from the experiment are both shown in the figures below. The modelling algorithm is designed to handle two-dimensional travel only (latitude and longitude). Consequently the model does not account for the UAV's altitude changes, though altitude data is available in GPS measurements and may be considered in future work.

The key observation from this experiment is the UAV's travel path. As is noticeable in the visualisation of the raw geolocation measurements, the drone takes on an irregular path that consists of several sharp corners. It is important to recall at this point that sharp corners can be a limitation to the model if not dealt with appropriately. As a result, it is necessary to tune the modelling algorithm's parameters to be sensitive to small and sudden movements. Furthermore, throughout flight the drone sways constantly as it stabilises for the effects of wind and its own motion. Hence it is necessary that the noise threshold radius be sufficiently wide to accurately distinguish between a change in

trajectory and a mere sway, whilst sensitive enough to still accurately capture sudden sharp movements. It was found that a noise threshold radius of 0.5m provided an appropriate balance for UAV drone routes.

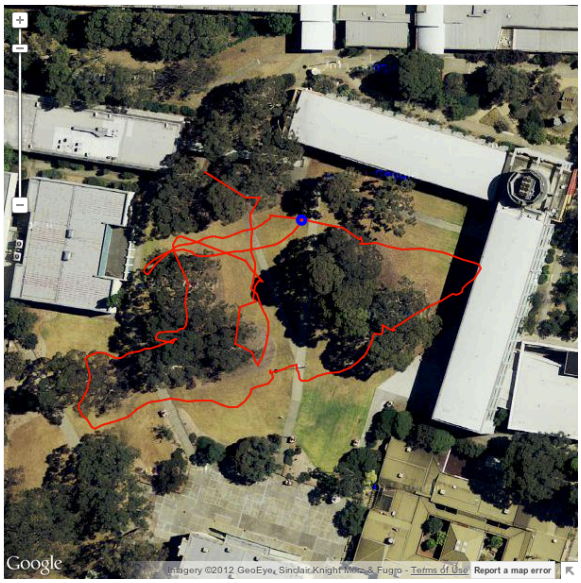


Figure 27: Raw geolocation measurements from a UAV route

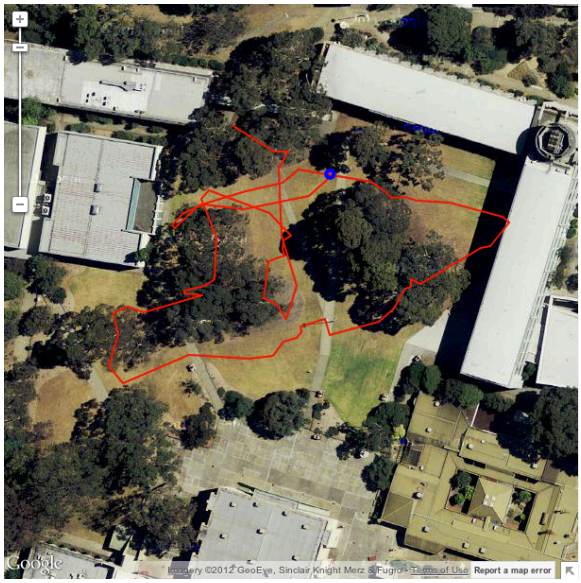


Figure 28: The UAV route's model at noise threshold radius of 0.5m

This experiment demonstrates the modelling algorithm's reliability in modelling unpredictable routes as commonly travelled by UAVs. Future work may explore the possibilities of transmitting realtime model data back to a central computer, which in turn can use the data to control and direct a swarm of drones.

8.3 Driving on road

Driving on road is a good demonstration of the application's ability to model and track everyday vehicle activity, such as that required for tracking taxi or logistics fleets. This experiment can also be used to demonstrate the application's reliability in modelling bus and tram routes. The figure below shows a visualisation of a model generated for a route travelled by car.

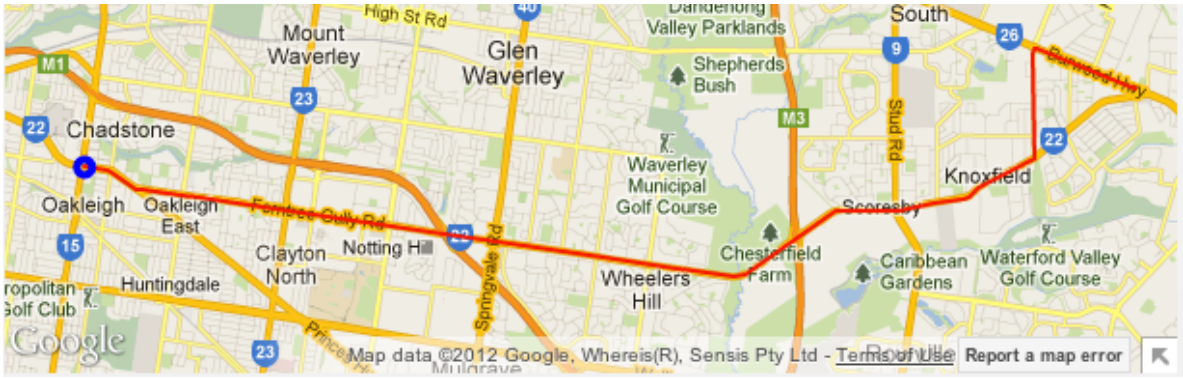


Figure 29: Visualised model for a route travelled by car (the model uses a noise threshold radius of 3m)

Despite the constant sharp corners, routes travelled by car tend to be regularly interrupted by road intersections and traffic lights. As a result, a noise threshold of 3m

proved sufficient for modelling typical inner city and suburban roads. A noise threshold radius of 3m also enables the model to filter out minor lateral changes in trajectory such as lane changes. The model shown in the figure above uses only 38 model control points to capture 1469 geolocation measurements taken over approximately 25 minutes.

8.4 A preliminary attempt at inertial navigation

In an attempt to learn and track routes with obstructed GPS signals, a preliminary investigation of inertial navigation using mobile devices was conducted during the early stages of the project. The objective was to look into techniques for tracking trains while they went through the underground portion of Melbourne's City Loop.

8.4.1 Inertial Navigation System (INS)

Inertial navigation systems (INS) provide a means of positioning a user with only minimal need for external infrastructure [17]. By recording a user's initial position and then monitoring the orientation and speed as interpreted from inertial sensors, it is possible to position a user within reasonable accuracy [17]. Normally, an INS will consist of:

- an entity outfitted with at least six inertial sensors: three acceleration sensors (x, y, z) and three rotation sensors (roll, pitch, yaw),
- an Inertial Measurement Unit (IMU) that measures the sensors' data, and
- an algorithm that can process the data and calculate the entity's position [17].

To calculate the entity's current position, the algorithm uses a technique called dead reckoning. A dead reckoning algorithm relies on knowledge of the initial position, as well as that of the individual movements made from the initialising point [17].

8.4.2 Inertial navigation using an Apple iPhone

Typical modern mobile devices, such as iPhone 4, ship equipped with the sensors required for INS. Researchers at the University of Munich have analysed the sensors on the iPhone 3GS and the iPhone 4 in an attempt to use the devices for indoor INS [17]. Despite the use of filters, it was concluded that the high inaccuracies and error associated with the mobile devices' sensors made it too challenging to build a precise INS [17]. They are currently exploring potential enhancements by using a multi-dimensional Kalman filter [17].

Engineers at the GNSS Research Center at Wuhan University have attempted to use an iPhone 4's inertial sensors for enhancing car navigation [18]. They concluded that it was possible to use the MEMS sensors on the iPhone 4 for car navigation purposes [18]. However, their research was focused on enhancing GPS positioning by using the inertial sensors to account for travel during short periods of GPS disruption [18]. A drift of 30m was observed after loosing the GPS signal for 30 seconds [18].

8.4.3 Attempting inertial navigation tracking on a railway line

An experiment was set up to test the possibility of using INS to track trains while they went through the underground portion of Melbourne's City Loop. To do this, an iPhone 4 was attached to a seat on a train journey and acceleration data was captured with the plan that it would be analysed after the journey.

The journey used herein was one that took place on the Glen Waverley line, starting at Richmond station and heading down-rail towards Glen Waverley. The journey lasted for approximately 17 minutes and acceleration measurements along the direction of travel were captured at regular intervals. The journey took place outside the City Loop in order to allow for the measurement of GPS data that could later be used to cross-reference the acceleration-based calculations. The journey's measurement intervals are shown in the map visualisation below.

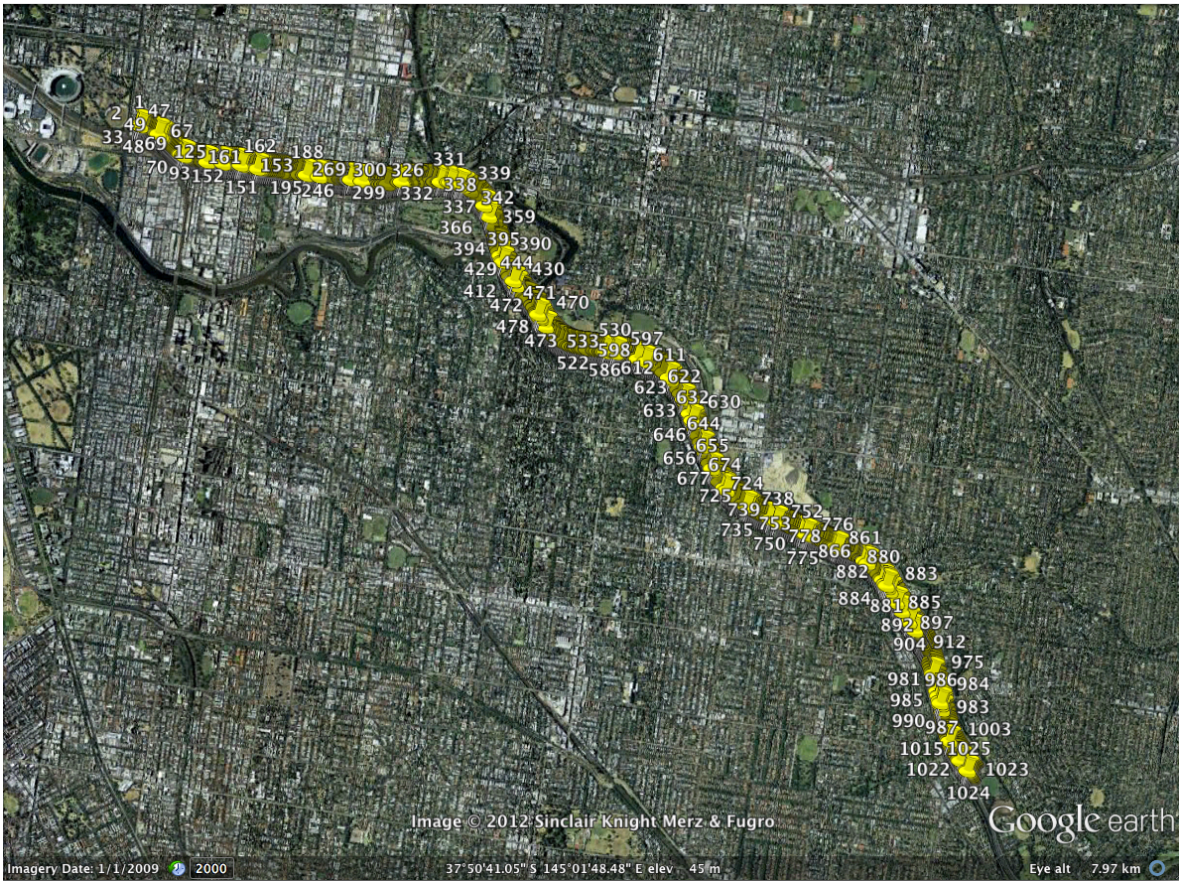


Figure 30: Journey measurement intervals for the INS experiment

The figure shown on the following page illustrates the raw acceleration measurements that were captured on the journey. Since the device's physical orientation was directly aligned with the direction of travel, no sensor fusion was required to translate the available acceleration measurements onto the direction of travel.

A low pass filter with a filtering factor of 0.15 (15%) was applied to the raw acceleration data to smooth out some high frequency noise. The technique used for low-pass filtering is the same one recommended earlier and is based on a moving-average low-pass filter with a depth of one. A wider Gaussian filter was trialled, however a simple low pass filter was found to provide a generally better smoothing function.

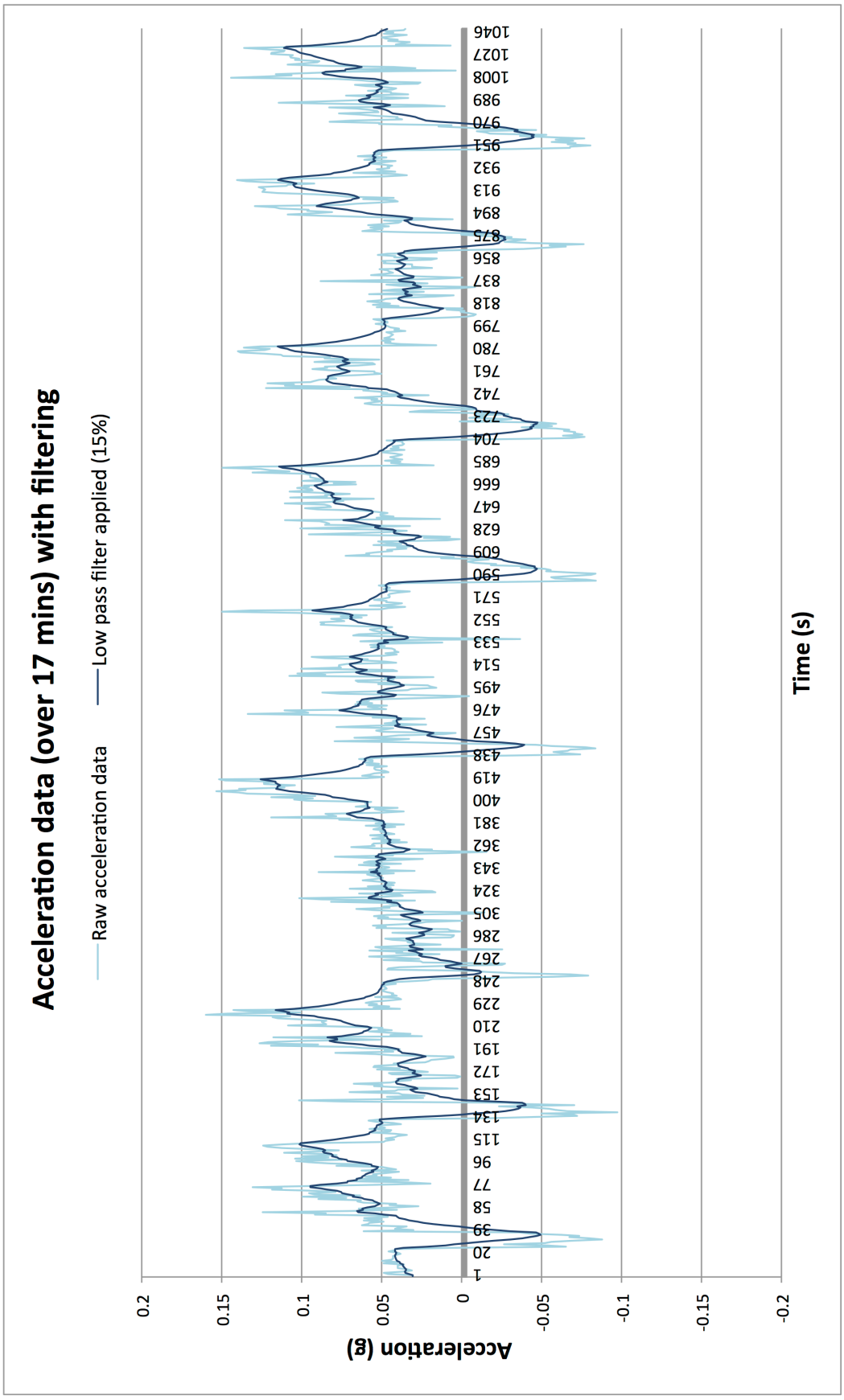


Figure 31: Acceleration data (over 17mins) with filtering

It is noticeable in the previous figure that the acceleration measurements suffer from low frequency offset. In order to adjust for this, a positive sliding window offset of size 80 was found to be most suitable. The low frequency offset is plotted in the figure below.

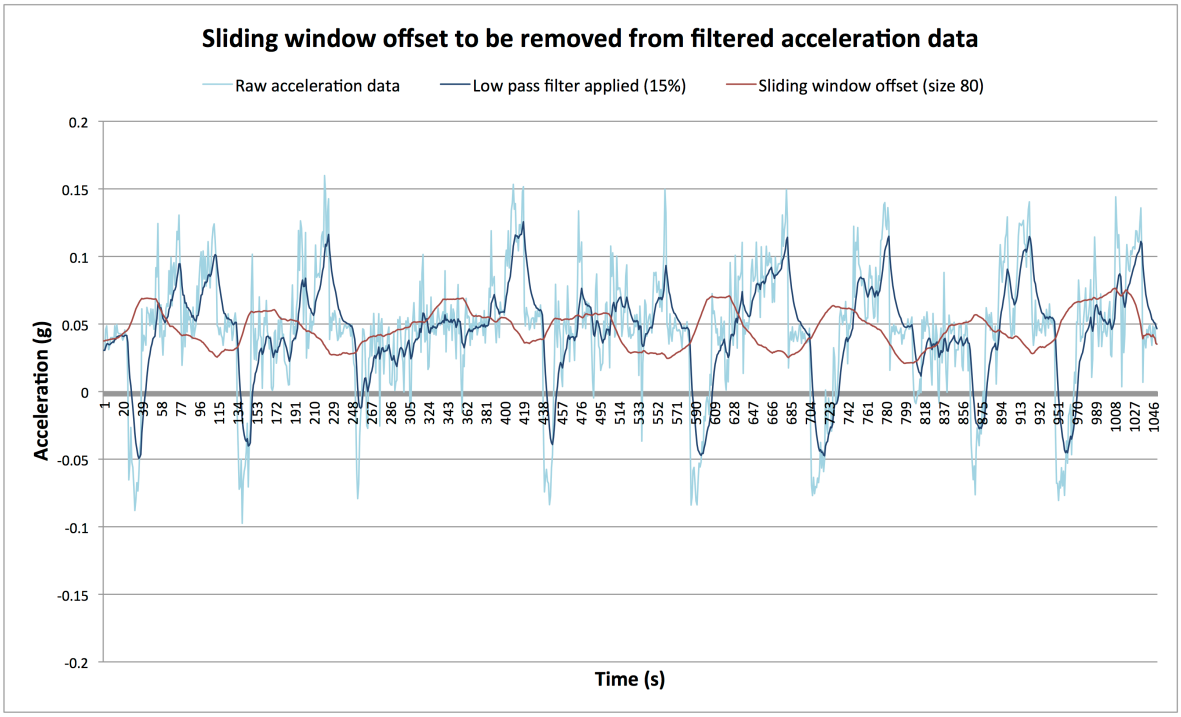


Figure 32: The sliding window offset superimposed on the acceleration data

After subtracting the sliding window offset and adjusting for the effects of gravity, the acceleration data can be normalised and integrated to produce the graphs below.

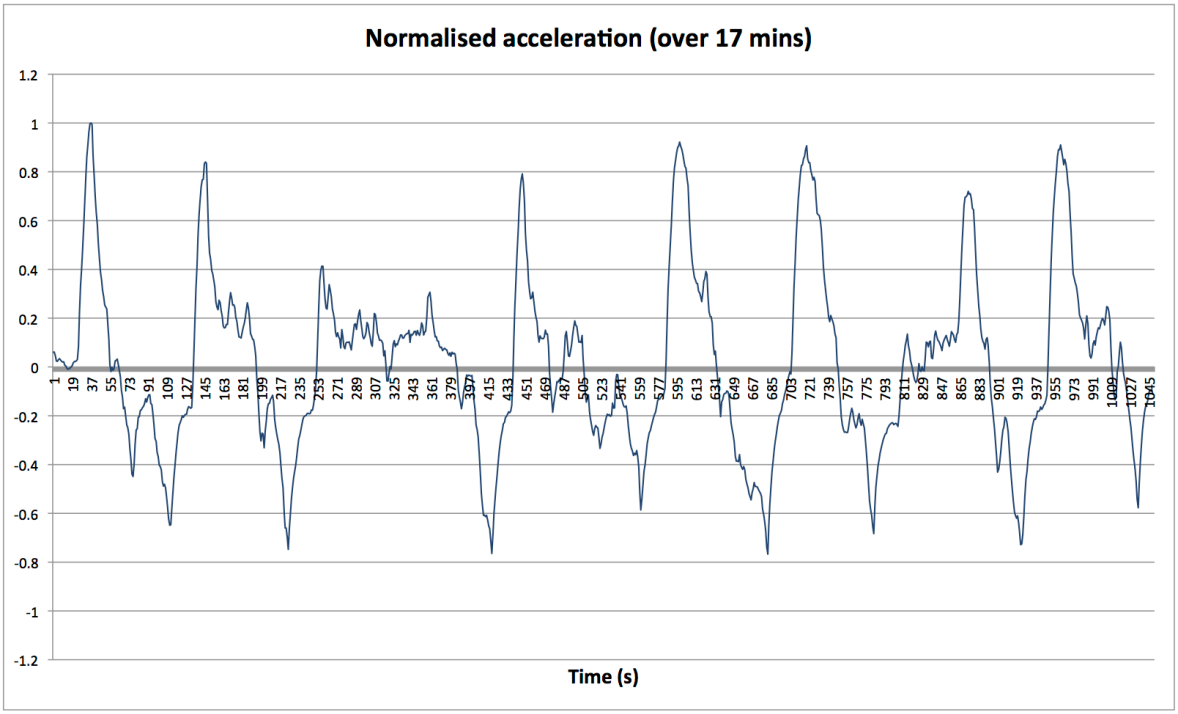


Figure 33: Normalised acceleration after filtering

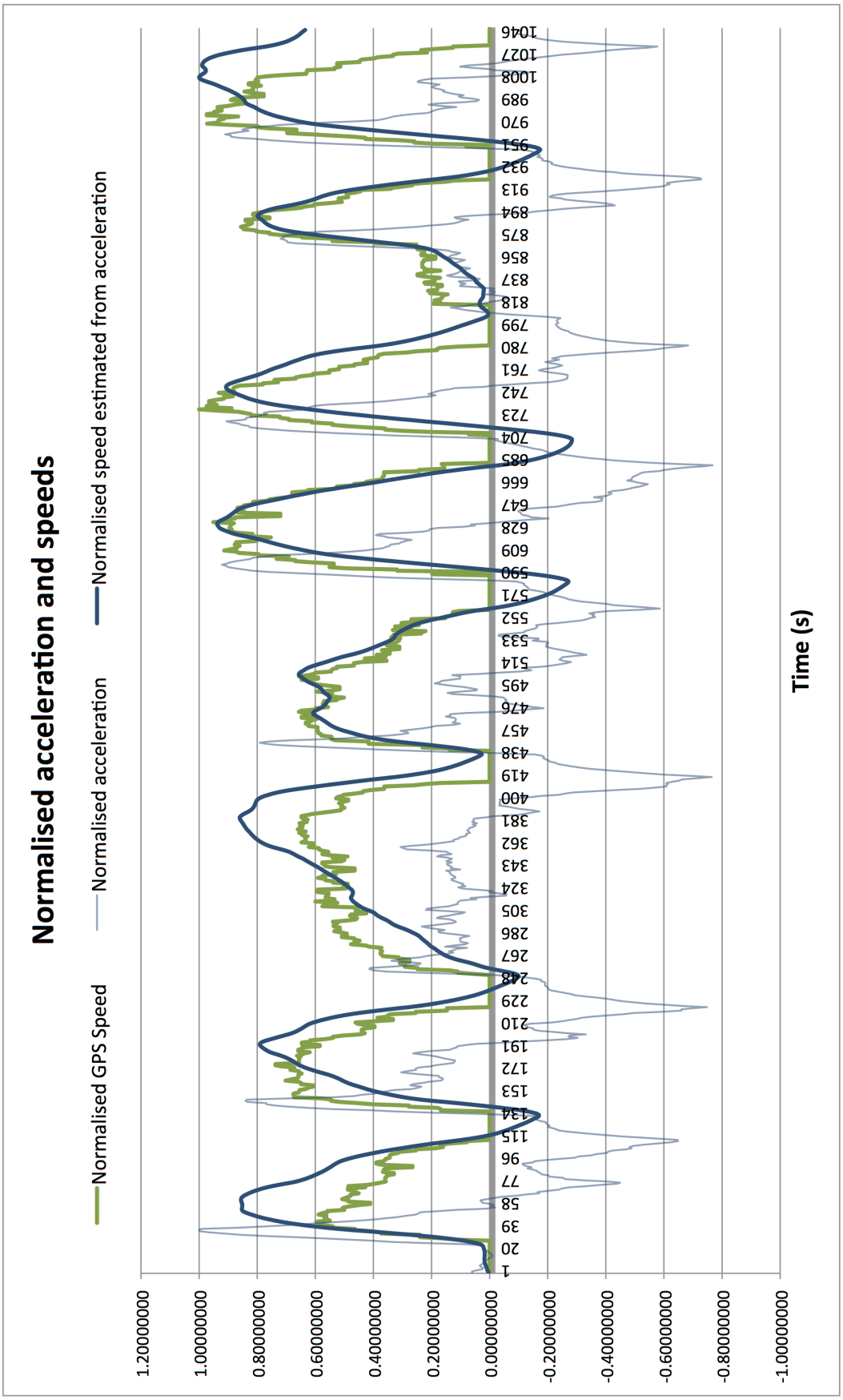


Figure 34: Normalised acceleration and speed estimates

Figure 34 (above) illustrates the outcome of applying a trapezoidal integration on the filtered acceleration data. It also shows the speed measured using the GPS receiver. The signals have been normalised to enable easier comparison. It can be observed that after applying two-stage filtering, the acceleration data produces a reasonable estimate of speed. Although far from ideal, this finding nevertheless indicates that this may be an area worth exploring further in the future.

Once speed was estimated from acceleration, it was again integrated using trapezoidal integration to generate an estimate for distance.

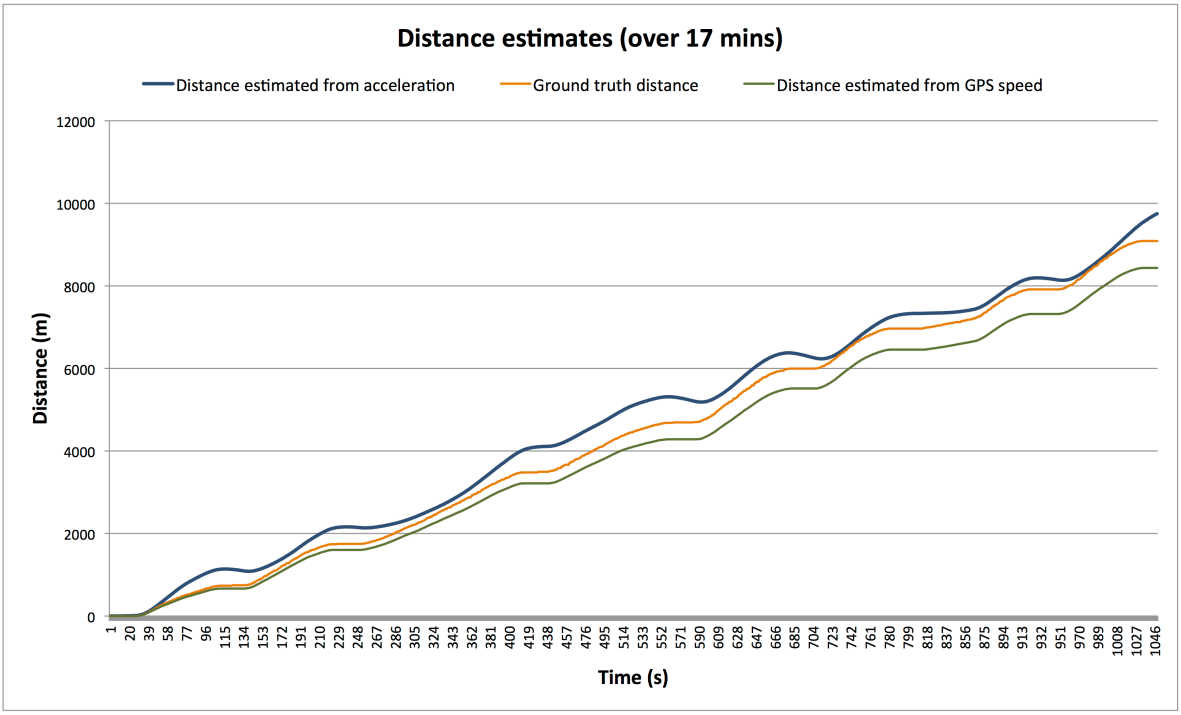


Figure 35: Comparison of distance estimates

The figure above shows the distance estimated from acceleration, as well as the ground truth distance, which is calculated using the Haversine formula. Although exhibiting minor drift, the distance estimated from GPS speed shows the adequacy of the integration technique used in estimating distance.

As shown in the figure, the acceleration's estimate of distance suffers from significant drift. Nonetheless, the data appears to show potential in being sufficiently accurate for estimating a train's location within the City Loop - especially if combined with train timetable data.

The residuals plotted in the figure below shows the extent of drift in the speed estimated from acceleration. Furthermore, a cyclic trend can be observed in the residuals indicating that the data may suffer from heteroskedasticity. An average error of approximately 313m was calculated.

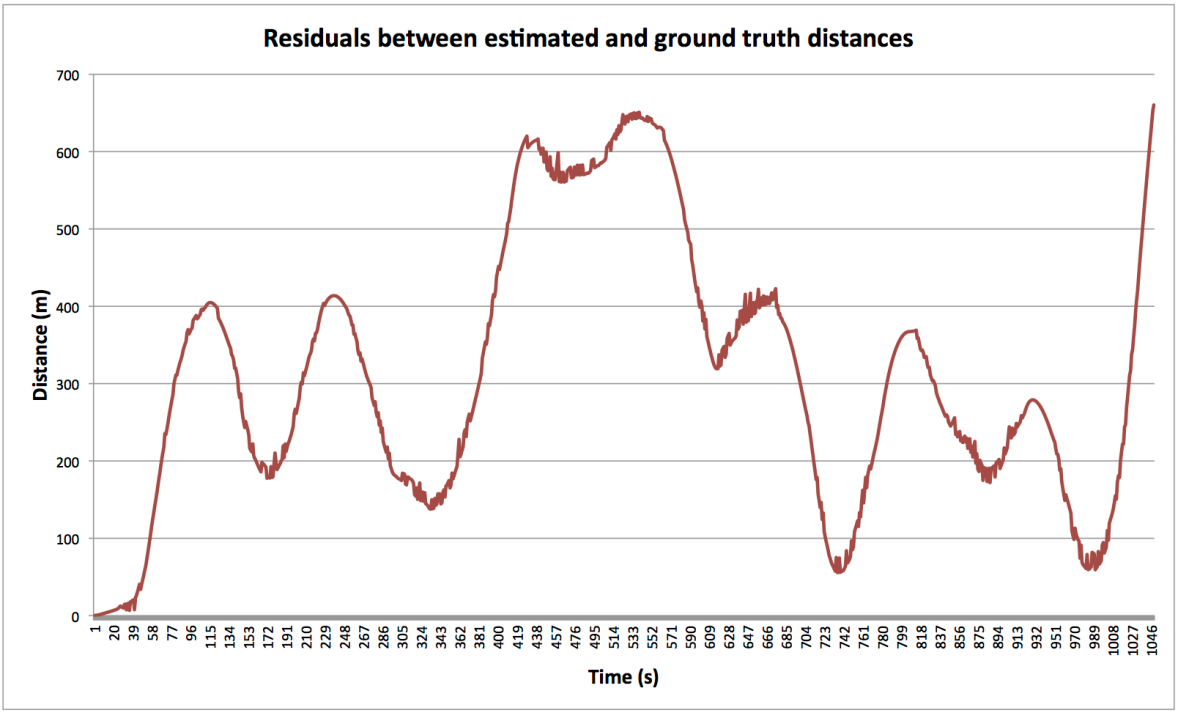


Figure 36: Residuals plot of the speed estimated using acceleration and that estimated using GPS

Overall, the assessment of the data in this experiment is inconclusive in asserting if a mobile device INS will be sufficient in tracking a train as it travels through Melbourne's City Loop. Whilst this is an interesting area worth exploring, any further analysis was regarded as outside the scope of this project and has been set aside for future work.

9 Conclusion

It is expected that "more than one third of European mobile subscribers will be using mobile Internet services by the end of 2013" [3]. Furthermore, emerging standards particularly those pertaining to HTML5 and WebGL are removing many limitations and providing developers with the platforms to transform the web [4]. By exploring data-mining and mathematical modelling algorithms, the V-Tracker (Vehicle Tracker) project has successfully achieved **realtime route learning and vehicle tracking using web-technologies on a mobile device**. As one of the first projects in this space, the application is setting the foundations for future experiments that will investigate the use of web-technologies in areas of digital perception and robotics.

The application, its encompassed algorithms and documentation form the primary deliverable for this project. It is acknowledged that some of the functionality developed may be of use to other engineers and as a result, two free open-source PhoneGap plugins have been developed and released online for the developer community to use in alternative applications. The plugins were released for free under a completely open source MIT license.

10 Acknowledgements

Several tools, plugins and open-source libraries were used to make V-Tracker possible. Gratitude is extended to each and every developer that contributed to these projects.

10.1 Software libraries and plugins

10.1.1 PhoneGap - Apache 2.0

"PhoneGap is an HTML5 app platform that allows you to author native applications with web technologies and get access to APIs and app stores. PhoneGap leverages web technologies developers already know best... HTML and JavaScript."

http://www.phonegap.com

10.1.1.1 PhoneGap LocalNotifications Plugin for iOS – MIT

The LocalNotifications plugin is used to generate and issue push-style notifications locally on iOS.

http://github.com/phonegap/phonegap-plugins/tree/master/iOS/LocalNotifications

10.1.2 jQuery – MIT

"jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is designed to change the way that you write JavaScript."

http:// jquery.com

10.1.3 jQuery mobile – MIT

"A unified, HTML5-based user interface system for all popular mobile device platforms, built on the rock-solid jQuery and jQuery UI foundation. Its lightweight code is built with progressive enhancement, and has a flexible, easily themeable design."

http:// jquerymobile.com

10.1.4 flot – MIT

"flot is a pure JavaScript plotting library for jQuery, with a focus on simple usage, attractive looks and interactive features."

http://code.google.com/p/flot/

10.1.5 Numeric JavaScript – MIT

"Numeric Javascript is a javascript library for doing numerical analysis in the browser. Because Numeric Javascript uses only the javascript programming language, it works in many browsers and does not require powerful servers."

http://www.numericjs.com

10.1.6 Modernizr – MIT & BSD

"Modernizr is a JavaScript library that detects HTML5 and CSS3 features in the user's browser."

http://modernizr.com

10.1.7 Google Maps – application must be free for users

“Google Maps has a wide array of APIs that let you embed the robust functionality and everyday usefulness of Google Maps into your own website and applications, and overlay your own data on top of them.”

http://maps.google.com

10.2 Other tools

10.2.1 node.js

“Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.”

http://nodejs.org

10.2.2 YUIDocs - Javascript Documentation Tool

“YUIDoc is a Node.js application that generates API documentation from comments in source, using a syntax similar to tools like Javadoc and Doxygen.”

http://yui.github.com/yuidoc/

10.2.3 AptanaStudio3

“Aptana Studio harnesses the flexibility of Eclipse and focuses it into a powerful web development engine.”

http://aptana.com/

10.2.4 Mou App

“Mou is the missing Markdown editor for web developers.”

http://mouapp.com/

10.2.5 GitHub

“GitHub is the best place to share code with friends, co-workers, classmates, and complete strangers.”

http://github.com/

11 References

[1] Mary Meeker, Scott Devitt, and Liang Wu, "Internet Trends," Morgan Stanley, Presentation 2010.

[2] Akamai, "The State of the Internet (1st Quarter 2012)," 2012.

[3] Dimitrios C Karaiskos, Panos Kourourthanassis, Panagiota Lantzouni, George M Giaglis, and Christos K Georgiadis, "Understanding the Adoption of Mobile Data Services: Differences among Mobile Portal and Mobile Internet Users," in *ICMB 2009. Eighth International Conference on Mobile Business*, 2009, pp. 12-17.

[4] Antero Taivalsaari and Tommi Mikkonen, "The Web as an Application Platform: The Saga Continues," in *37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, 2011, pp. 170-174.

[5] World Wide Web Consortium. W3C. [Online]. http://www.w3.org/standards/

[6] Apache. (2012, August) Apache Cordova. [Online]. http://incubator.apache.org/cordova/

[7] Daniel Brateris et al., "iOS hardware as a sensor platform: DMM case study," in *Sensors Applications Symposium (SAS), 2011 IEEE*, 2011, pp. 308-311.

[8] Nicholas D Lane et al., "A Survey of Mobile Phone Sensing," *Communications Magazine, IEEE*, vol. 48, no. 9, pp. 140-150, September 2010.

[9] Sinjin Dixon-Warren. (2012, September) MEMS Journal. [Online]. http://www.memsjournal.com/2010/12/motion-sensing-in-the-iphone-4-mems-accelerometer.html

[10] Apple Inc. (2012, April) iOS Developer Library. [Online]. http://developer.apple.com/library/ios/documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/MotionEvents/MotionEvents.html#//apple_ref/doc/uid/TP40009541-CH4-SW1

[11] Sinjin Dixon-Warren. (2012, September) MEMS Journal. [Online]. http://www.memsjournal.com/2011/01/motion-sensing-in-the-iphone-4-mems-gyroscope.html

[12] Sinjin Dixon-Warren. (2012, September) Mems Journal. [Online]. http://www.memsjournal.com/2011/02/motion-sensing-in-the-iphone-4-electronic-compass.html

[13] Apple Inc. (2012, August) iOS Developer Library. [Online]. http://developer.apple.com/library/ios/#DOCUMENTATION/UserExperience/Conceptual/LocationAwarenessPG/Introduction/Introduction.html

[14] Garmin Ltd. (2012, September) Garmin. [Online]. http://www8.garmin.com/aboutGPS/

[15] iFixit. (2012, April) iFixit. [Online]. http://www.ifixit.com/Teardown/iPhone-4-Teardown/3130/3

[16] Broadcom Corporation. (2012, September) Broadcom GPS Silicon Solutions. [Online]. http://www.broadcom.com/products/GPS/GPS-Silicon-Solutions/BCM4750

[17] Corina Kim Schindhelm, Florian Gschwandtner, and Michael Banholzer, "Usability of Apple iPhones for Inertial Navigation Systems," in *22nd International Symposium on Personal, Indoor and Mobile Radio Communications, 2011 IEEE*, 2011, pp. 1254-1258.

[18] Xiaoji Niu, Quan Zhang, You Li, Yahao Cheng, and Chuang Shi, "Using inertial sensors of iPhone 4 for car navigation," in *Position Location and Navigation Symposium (PLANS), 2012 IEEE/ION*, 2012, pp. 555-561.