LONGITUDINAL MONITORING OF PERSONAL TRANSPORT MODE FROM SMARTPHONE-DERIVED CONTINUOUS SENSING DATA

JACK EDWARD GEORGE HARRISON

UCL

UNIVERSITY COLLEGE LONDON

MRES ADVANCED SPATIAL ANALYSIS & VISUALISATION

I, JACK EDWARD GEORGE HARRISON CONFIRM THAT THE WORK PRESENTED IN THIS THESIS IS MY OWN. WHERE INFORMATION HAS BEEN DERIVED FROM OTHER SOURCES, I CONFIRM THAT THIS HAS BEEN INDICATED IN THE THESIS.

Abstract

Smartphones are becoming ever-more ubiquitous, and with most smartphones containing both GPS and accelerometer sensors they are well suited to providing insight into personal movement. Until recently, such research has been conducted using ad-hoc, wearable sensors or analogue methods such as diaries. These established methods typically have one or more significant limitations, including inadequate coverage or detail of movement, difficulties in scaling to larger userbases or lack of user interactivity or feedback.

Here, the idea of using smartphones to provide *continuous*, *automated* monitoring of personal movement is explored, with the intention of practically addressing many of the limitations seen in previous studies through the construction of a prototype workflow. Crucial to this workflow is the development of a classification algorithm, which is able to infer the transport mode of the user from a number of features extracted from the smartphone accelerometer data. This provides a new level of understanding around personal movement, allowing the *transport choices* of a user to be studied and, if desired, influenced through feedback. This feedback is demonstrated in a health context through the considered visualisation of a users movement data, with the process of creating such visualisations also detailed in full.

From experimental data, the classification of transport modes was highly successful, with the algorithm accuracy averaging at 93.2% across five possible modes. This figure was higher than almost all of the other work discussed in this study, a success that was attributed to the appositely small number of classes and a simple, robust algorithm. Using the classified data, two visualisation programs were then constructed to communicate the data back to the user, capable of taking the output from the classification algorithm and *automatically* producing a full visual interface. Direction was also given on how further development of the system could refine the workflow into a continuous, integrated process from collection to visualisation.

Table of Contents

INTRODUCTION

In a growth trend that does not show any signs of slowing, smartphones are becoming ever more ubiquitous in our society. While not all of these devices will contain the very latest technologies, most will at least contain a now-standard array of sensors, such as a microphone, GPS (Global Positioning System) receiver or accelerometer. This has resulted in the genesis of a huge network of sensing platforms which, through utilising the wealth of data such a network could produce, has the potential to provide unique insights for the behavioural sciences and contribute to the continued growth of innovative pervasive computing techniques (Zheng et al. 2008).

What is now becoming clear, however, is that entirely new techniques are often required to deal with such volumes of data (Laurila et al. 2012). The term 'Big Data' rises from this realisation, and the commonly accepted definition is that it describes datasets too large to be dealt with using 'traditional' analysis and visualisation tools. Location data, produced by almost every smartphone, is the archetypal big data problem - a single smartphone could produce several million entries over the course of a day, often from a number of discrete sensors. Such data has the potential to provide an unprecedented understanding of the movement patterns and actions of individuals, but until now there have been several procedural and technological barriers that have made longitudinal studies difficult to undertake on a large scale - particularly the size and cost of equipment and user burden (such as manual annotation).

Movement data is collected for a multitude of purposes, from individual health and fitness to countrywide travel demand modelling, but something that is rarely considered outside of specific research is the opportunity to communicate the data collected back to the participants. There are likely to be far more willing users in any given trial if they perceive the results to be beneficial to them as an *individual* - this is evidenced in the huge successes and millions of users that commercial

sensor-based products such as Nike+ (nikeplus.nike.com) and MyFitnessPal (myfitnesspal.com) have had.

Here, the combined use of location and peripheral sensors is proposed as part of an *activity monitoring* system. The primary function of the system will be as a health support tool for both the user and their physician, to provide insight and promote behavioural change around a users transport mode choices. Previous systems of this type have often required a large amount of ad-hoc equipment (Bao & Intille 2004), and systems that *have* run on existing equipment have either relied on user annotation (Doherty 2009) or not studied the system from a healthcare perspective (Manzoni et al. 2010). A considered review of the work of these (and other) authors will be undertaken which, in combination with the concurrent developments in visualisation and analysis software, will be used to produce a prototype workflow for automated and continuous collection, processing, analysis and visualisation of transport data.

The main requirements of the system are that it can run on preexisting equipment (a smartphone), allow continual use throughout a day and require limited user interaction once running. Studies have shown that humans are in general *poor* at data entry when the event is not recent or in situ (Sonderegger & Sauer 2010), so an automated system is likely to greatly improve data coverage over previous efforts.

The application will also provide constructive, impartial feedback to the user through an interactive visualisation interface and, although it is primarily intended as a tool for retrospective analysis by a physician, this feedback will need to be carefully considered as it can have a marked impact on the continued use of the tool (Arteaga et al. 2009). A feedback mechanism of particular interest is the highlighting of 'short' trips that are currently not being taken by an active form of transport (walking/running/cycling). One of the key questions to be addressed is whether such journeys can be identified from a continuous stream of sensor data reliably *without* manual annotation from the user (i.e. automatically).

The following sections detail the suite of techniques required to turn raw sensor output into an understanding of human transport choices, from the initial data collection through to the visualisation and communication of results. There will be a focus throughout on work that produces detailed accounts of *individual* movement patterns, although wider and more aggregate studies will also be considered. Through the surveying of these methods it will be assessed as to whether the smartphone system being proposed here is feasible - and if so, which techniques are the most appropriate for its implementation.

The first step of this process is to review the existing literature on the subject, both to inform the subsequent sections and to summarise the current state of the research field.

Related work

Mobile phones as location data sources

The use of cellular devices in capturing location data in a general sense is not novel, and there are many successful studies at a range of scales that are worth reviewing before embarking on further research.

Many studies have used cell towers (González et al. 2009) as a proxy for user location, with the location of the closest cell tower being logged each time the user makes a call. This allowed an inferred location to be provided independent of GPS technologies, potentially allowing access to a much larger populations (those with any mobile phone >> those with smartphones). It does, however, require negotiating heavily with the network providers, who are often very reluctant to share such sensitive information. Such agreements have been made, however, allowing human movement patterns across space and time to be studied in projects such as *Real-Time Rome* (Reades et al. 2009) where millions of people were (anonymously) tracked to indicate the flows and immobilities of an urban system. However, while appropriate for the scale, there are limitations to cell tower derived location, primarily in that it is too approximate for finer-scaled (meso/micro) analysis, and that it can be difficult to follow individual users over time (due to privacy concerns).

To study movement patterns at a more disaggregate scale, different tools and approaches are required. Portable GPS receivers, employed by the military for almost two decades, are a common solution, as they can provide significant locational accuracy - usually enough to locate and track an individual user. Until more recently, however, they were specialised tools that required a significant investment, and as such they were rarely used for large trials in academia. Fortunately, the price (and size) of the receivers have subsequently been reduced dramatically, and now GPS receivers are widely available both as standalone devices and as a core component in almost all smartphones.

This technological development has not gone unnoticed in the academic community, and several key studies have shown that GPS data can be used to begin monitoring and understanding the complexities of individual human movement in urban environments. Liao et al. (2006), for example used GPS data in combination with a series of models to create personal maps for each participant, using a discriminative filtering method to extract and label significant or commonly travelled locations. The system also suggested an average or 'normal' routine for the user, and it as such it was able to identify deviations from this routine (such as a user taking a new route). The study proposes a number of potential applications for such a system, such as personalised navigation solutions with 'just in time' service provision that can alert a user with information that is likely to be useful to them at that time and location (such as traffic conditions or bus schedules).

Two projects of particular note are those of Nadalutti & Chittaro (2007) and Neuhaus (2009). Nadalutti & Chittaro used a GPS receiver and heart rate monitor to produce speed and cardiovascular fitness performance data for users, where Neuhaus used GPS to study long-term personal 'rhythms' in an urban environment, but both studies were distinct in their *holistic* approach to the methodology, considering not only the collection and processing of the data but also the visualisation and communication of the findings back to the user. Neuhaus in particular focuses heavily on the analytical power of visualisation, studying the changing routines of individuals over time to better understand urban dynamics.

There are, however, limitations to GPS as well. Some are immutable, such as the inability of a GPS receiver to get a lock indoors or underground. In the Neuhaus study, for example, one of the biggest challenges was that no satellite signal could be recorded when the user was on the London Underground, and buses and high-rise buildings also caused data loss (Neuhaus 2009).

Others are closer to technological limitations, such as the large power drain caused by a GPS receiver over long periods. For almost all smartphones on the market, for example, it would be unfeasible to utilise GPS for a full working day before consuming the battery, with a recent study having to utilise both an external battery *and* an external GPS receiver to make the phone last the whole day (Doherty 2012). Finally, GPS can only tell us where a user is (directly) and their velocity (indirectly) so making a distinction between cycling and driving, for example, can be challenging (especially in urban environments).

Classifying human movement

To better understand human movement, then, requires much more than just where we are and how fast we're going - it needs some measure of *how* we are travelling. One potential source of such information could be found in the accelerometer, another common component of smartphones. A standard tri-axial (3 axis) accelerometer is able to measure acceleration in the X, Y and Z directions (*Fig. 1*), as well as providing a total acceleration value through integration of the three.

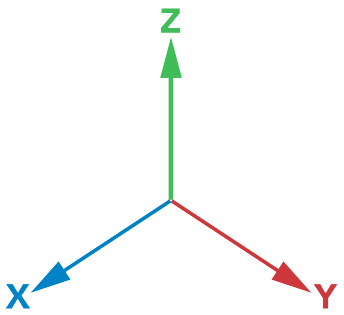

Fig. 1: The X, Y and Z axes of a tri-axial accelerometer.

Its role in recognising physical activity has been explored for well over a decade (Bouten et al. 1997), but it was not until recently that the sensor became so widely used in consumer technology, changing the scope of any potential application dramatically. In order to fully understand the implications of this shift it is necessary to evaluate the range of studies that

have been undertaken around activity recognition, from early attempts using purpose-built arrays of accelerometers to more recent work using integrated sensing devices.

Several early studies required large numbers of sensors to accurately detect activities (Van Laerhoven et al. 2002), with the general opinion being that more sensors gave more accuracy to any recognition method used. Most of this work, however, was conducted under controlled laboratory conditions - due in part to the awkward and bulky equipment required in the multi-sensor arrays. Bao & Intille (2004) recognised this shortfall and set out to validate activity recognition under real-world circumstances. They developed an activity recognition system with 5 bi-axial (two-directional) accelerometers placed around the body of a participant, intending to create a system that could reliably detect several everyday tasks (walking, ascending stairs, brushing teeth, falling) without supervision or user interaction. The classification was achieved through a decision tree process, where the values from each sensor were considered in an automated progression of binary (yes/no, greater/lesser) stages in order to reach a likely interpretation of the data (*Fig. 2*).

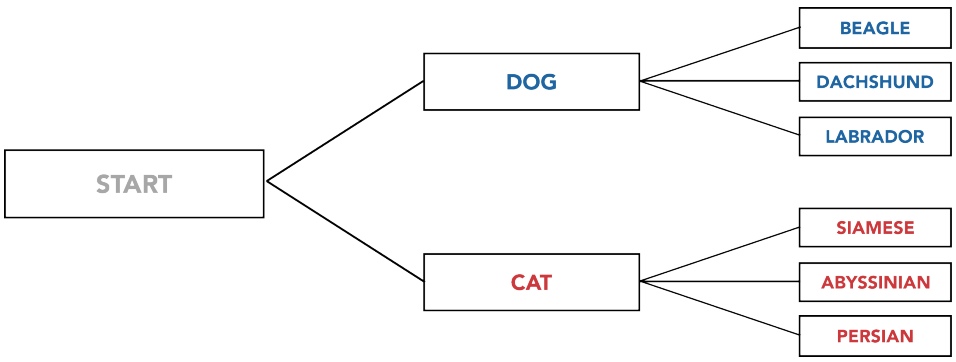

Fig. 2: A typical decision tree, with a series of choices leading to an increasingly specific determination.

In most activities they found the classification methods to be universally applicable, suggesting that the movements of these activities (walking and navigating stairs, in particular) were likely to be consistent among participants. The aggregate recognition rates across the whole study were good, with an overall accuracy rate of 84%. The deviation of

accuracy amongst activities was high, however, with the most difficult activities to recognise being those with limited movement, such as stretching or riding elevators (both were around 40% accurate). While the extent of activities sampled in this study was commendable, not all applications would require such a wide range of activities - especially those related to health and fitness. Being able to recognise walking, running, inactivity and falling, for example, is undoubtedly more use in such a setting than detecting toothbrushing or watching TV.

Ravi et al. (2005) looked to reproduce the results of the Bao & Intille study with a single tri-axial accelerometer, looking in particular at the suitability of different classifiers that could be extracted from the signal. They used the Matlab software package (mathworks.co.uk/products/matlab/) to process the accelerometer data and extract key features, focusing on the mean, standard deviation, correlation and energy (or power) of the signal from each axis (X, Y and Z). The features were calculated for a moving window of 256 samples, and with a sampling rate of 50Hz this gave a representation of 5.12 seconds of data in each window. The window was also overlapped by 50%, as this had demonstrated success in previous work (Bao & Intille 2004). This produced a stream of data for 12 attributes (four for each sensor), and all but the mean values were explained in detail by the authors in regard to activity recognition. The standard deviation value was included to show the changing range of acceleration values that would be expected from different activities (i.e. running and driving). While the mean may show a strong overall acceleration, the deviation shows whether it is consistent or varying over time - potentially allowing a different activity type to be detected. The energy feature was calculated, to give some indication of the periodicity of the signal, through the summing of the squared discrete Fourier transform component magnitudes of the signal. Finally, the overall correlation of the signal was calculated, to differentiate single-dimension activities such as running (X & Y) from multi-dimensional activities like climbing stairs (X, Y & Z).

The authors go on to improve on the decision tree method proposed in the Bao & Intille study, surveying a range of classifying techniques and concluding that a form of plurality voting was preferable in producing the greatest accuracy in activity recognition. This technique uses the value of each of the features to predict a likely class, with the final output being the class with the majority of predictions. The selected workflow resulted in >99% accuracy of the classifications, a great improvement over previous work and still with a relatively large number of varied activities. Another important outcome from their study was that this accuracy was achieved with the use of just one tri-axial accelerometer (in place of the multi-sensor arrays used in previous studies). This meant that it was possible to classify physical activity using a component was likely to be found in most smartphones, thus providing an opportunity to work with a much larger network of users in future work.

In 2010, a group of researchers at the MIT SENSEable City Lab (senseable.mit.edu) succeeded in creating a system that could utilise standard smartphone sensors to monitor and classify human movement (Manzoni et al. 2010). The aim of the project was to create an account of the personal CO2 contributions of a users transport choices. While these kind of 'calculators' have been done before, the authors noted that previous work has been web-based (requiring manual input from the user), GPS-derived (high energy cost and coverage issues in urban environments) or required ad-hoc sensors - which they suggested had limited adoption and deployment. In contrast, by making use of an existing platform that was already widely adopted their method had the potential for 'unprecedented data collection of mobility patterns'. Through automated inference of transportation mode, the system was able to provide a CO2 emissions estimate that was personal to the user, providing a real-time understanding of their contribution without requiring any manual input. Similarly to the work by Ravi et al. (2005), the first stage of the classification algorithm involved feature extraction from the accelerometer signals. However, one of the limitations of using a stock component is that a

regular sampling rate (required for the FFT transformation) cannot be relied upon. As a result a pre-processing stage was required before the FFT was performed, where the signal is interpolated and resampled. They tested both cubic and linear interpolation methods, finding that the results were comparable and selecting the linear method due to its speed and ease of implementation on a mobile device. Features were then extracted from this resampled signal and a decision tree was used to provide a classification.

The system considered a wide range of possible transportation modes, covering bus, metro, train, car and motorcycle as well as walking and cycling. A large and relatively complex decision tree was required to distinguish between these (some of which were relatively similar - car and bus, for example), but this was essential in providing a realistic CO2 estimate. They used experimental data to calibrate the algorithm, with several hours of data being collected for each transport mode along different routes. They also used multiple participants to calibrate certain transport modes, as some (such as walking or cycling) differed slightly with gait or sensor position. This calibration resulted in an average of 82% classification accuracy over the 7 modes, with the authors noting that this method did not rely on a certain position or orientation of the phone and that it did not require data from GPS (as previous studies have). However, some relatively low-resolution location data *was* required to derive speed and distance metrics for CO2 estimation, and they commented that classification accuracy could be improved further by incorporating some of this supplementary data.

Identifying longitudinal patterns in travel behaviour

Through the use of previously discussed techniques that describe the micro-scale patterns in sensor traces, it is possible to perform a *discrete* query on a users data to return their likely transportation mode or activity.

To provide a more detailed understanding of travel behaviour, however, requires the consideration of patterns that occur over larger scales - both spatially and temporally. The techniques involved in recognising patterns in longitudinal travel data will be reviewed and evaluated in this section.

Initially, sensing technologies such as GPS were used primarily to provide supplementary or validating data to more established methods of assessing travel behaviour. Travel surveys, for example, provide useful data for transport demand modellers, but the typically used trip diaries often have data quality issues where trips are missing detail or simply under-reported (Richardson 2003). Several studies attempted to improve the accuracy and completeness of this data through the use of GPS receivers, fitted to the vehicles of the survey participants. One early study found it problematic to match the observed GPS data with the trip diaries provided by respondents (61% matched), citing the tendency towards rounding inaccuracies concerning trip start times, travel times and distances as the main difficulty in comparing the two datasets (Wagner 1997). As an alternative to this supplementary method, Wolf (2000) trialled the *sole* use of GPS kits to carry out the travel survey. She found several advantages in using this method, particularly in the identification and recording of short or multi-part trips that were often omitted from paper diaries. However, she also described how the GPS units could only record personal vehicle trips, leaving other modes of transport (walking, cycling, transit etc) to be recorded manually.

As GPS equipment shrank, though, it became possible to provide participants with battery-powered portable receivers - rather than the 12V receivers installed in cars in previous studies. This allowed more continuous, longitudinal data to be collected, giving travel behaviour researchers insight into a previously abstruse area of their field. Schönfelder et al (2002) describe how previously, with the limitations imposed by the survey techniques available (mainly travel diaries and surveys), the high-quality data available to researchers was primarily short-

term. Thus, one or two days of data was often assumed to be an 'optimal decision' of a traveller, describing their normal behaviour. The authors note that this is adequate for studies of *inter-personal* comparisons, looking at the behaviour of different groups or individuals, but it is not sufficient to gain insight into *intra*-personal patterns of behaviour - the variability of an individual or households mobility over time (*Fig. 3*).

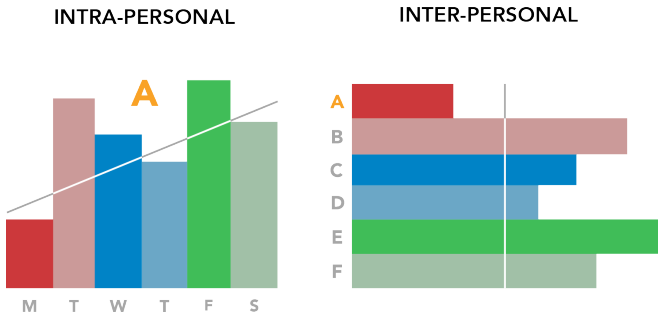

Fig. 3: Intra-personal and inter-personal level of travel behaviour study. Adapted from Schönfelder et al (2002).

With the use of portable GPS receivers they were able to collect appropriate data for this intra-personal analysis, detailing their attempts with several different datasets. One of the first stages in both this study and Wolf (2000) was the defining of the interval between trips that would represent a break in journeys. Some visual analysis methods, such as heatmaps or other density-based visualisations, can be performed on continuous datasets to produce an aggregate, summary representation. To study the changes in individual routes, though, requires the dataset to be split into discrete journeys - which in turn requires an appropriate cut-off measure to describe where one journey ends and another begins. Both studies found that an interval of 120 seconds where speed was close to zero was an appropriate definition of a journey end (there were exceptions, of course, such as non-moving traffic, but these were relatively rare). By performing cluster analysis on these start and end points, they could identify key activity locations of users (which could be a home or work area), again allowing similar routes to be compared. Street parking outside an office, for example, would result in a slightly different trip start/end location for each journey, but using a tolerance allows them all to be

treated in the same way. A similar method could be applied to analyse GPS data collected en-route, where individual traces may not match precisely even when the driver is taking the same route (due to road position and locational accuracy). It is important to note, however, that these studies were conducted at a large-scale, using a survey group in order to get an aggregate picture of the travel behaviour of a population. These methods are not designed to provide any great insight into *individual* users, because as discussed there have existed several large obstacles to fine detailed & long-term data collection (specifically equipment impracticalities and the requirement of manual input). However, the removal of these obstacles has now produced a movement of self-reporting, data-driven individuals, producing a wide range of opportunities which will be discussed in the next section.

Ubiquitous computing and the quantified self

As discussed, advances in mobile computing and networking technologies have created an unprecedentedly pervasive environment for technology, with near-ubiquitous access to networked computing in much of the developed world. In particular relevance to this project, one of he products of this shift has been an increase in the awareness and access to quantitative self-reporting tools, known colloquially as the 'quantified self'. The core ideas behind the quantified self are best described by the authors who coined the term, the technology bloggers at quantifiedself.com, who describe it succinctly as the progress of 'self knowledge through self-tracking' - or gaining personal insight into our own lives through keeping measures and logs about our activities and attributes. Several scientific studies have been undertaken to formally assess the benefits of self-reporting, with all reporting at least some degree of positive impact. Stuckey et al. (2011) provided self-reporting tools to those with type-2 diabetes that facilitated regular monitoring of blood sugar levels and automated physical activity logging. They found

that participants had 'improved body composition, increased daily exercise and interest in [positive lifestyle changes]', and found that compliance and confidence in the system was high and grew throughout the trial. Another medical study, this time looking at weight loss, found that participants who used a digital method to track their diet and activity were more successful than those using paper, with a significant causal effect on weight loss over 12 months as well as an indirect effect resulting from better adherence to the monitoring (Wang et al. 2012). A third example from medicine describes how smartphones can be used as part of an intervention to treat people suffering from major depressive disorders. Behavioural activation involves ranking in difficulty a series of actions that will facilitate recovery (reinforcing actions), and then systematically attempting them from easiest to hardest. In this study, participants used smartphones to progress through a behavioural activation-based guided self-help treatment programme. As well as more general conclusions regarding the potential for more widespread adoption of smartphone applications in health, the study concluded that the pervasiveness of smartphones was a great help in reinforcing positive behaviours (immediately after the event) and stipulated that this may lead to 'improved compliance' to the treatment (Ly et al. 2012).

Most of the projects discussed so far have involved groups of between 10 and 100 participants - large enough to draw conclusive results but far from what would be considered widespread adoption. The self-reporting techniques trialed in these studies are not limited to the scientific or medical communities, however, and demand within the technology sector for products and services that relate to the quantified self is growing rapidly.

Although these systems cover a wide range of fields such as health, sport, environment, location and education, it is often applications relating to lifestyle and health that have the most active users. Applications such as the Nike+ suite of running and sport monitoring tools (nikeplus.nike.com)

or the diet reporting platform of MyFitnessPal (myfitnesspal.com) are good examples, and they are also part of a meaningful trend - the positive effects of tracking health behaviours having been well-documented in medical literature. Hollis et al. (2008), for example, describe how 'behavioural measures (e.g., diet records and physical activity) accounted for most of the weight-loss variation' in a recently conducted diet and exercise study.

Nike+ began with the release of the Nike+iPod device, which used a piezoelectric sensor to measure footfall in runners and produced an approximation of distance, pace and calories for each of the users workouts. It did not gain wider adoption, however, until the free Nike+GPS Running application was released for the iPhone. Rather than requiring specialised footwear and additional sensors, this application relied (and relies) only on the capable range of sensors already present in the phone (namely the clock, GPS receiver and accelerometer) to provide an even greater range of tracking options than was possible with the iPod device. With the GPS, the user could now view maps of their routes, and the increased cellular capacities of the phone also allowed the user to share their workouts with friends. This sort of functionality was previously only possible with prohibitively expensive GPS devices, and as such it tended to be in the realm of professional athletes rather than fitness enthusiasts. While the smartphones themselves are often expensive, the functionality provided by applications such as Nike+ generally make up only a very small proportion of the perceived value proposition of the whole device.

MyFitnessPal is a more recent example, and it has achieved its popularity by successfully integrating a diet reporting interface with a comprehensive database of nutritional information for almost every conceivable food or drink item. Users can input a few descriptors about their meal (such as a brand and variety of a ready meal) and the application will retrieve the calories (as well as other nutrients) and add it to the daily total. While analogue food diaries are valuable tools (and a clear

predecessor to this process), they usually require a large amount of user input (manually sourcing calorie counts for every meal) or the aid of a nutrition expert to properly quantify and analyse the content. Again, this application makes high-level quantitative reporting significantly more attainable for an individual, being cheap, quick and effective.

There are three key insights to take from these examples. Firstly, they reiterate the willingness of these markets to adopt and engage in self-monitoring - especially when it relates to fitness and health. Second, they highlight some common attributes of successful products; that they are easy to use and place very little burden on the user (e.g. new equipment, maintenance, manual input). Lastly, they offer several different views of the users data - usually a combination of short-term reports and long-term visualisations. These communication choices are detailed and evaluated in the following section.

Communicating movement data

Most of the studies discussed so far have focused heavily on the data collection and processing aspects of studying movement data, with very few considering in detail the visualisation and feedback mechanisms that are employed when communicating research back to participants. This is fine in a large-scale transport modelling context, but in a more personal health-driven application these concluding stages can have a notable impact on user behaviour. Unfortunately, visual output from the scientific community is often perceived as an 'end product' of research rather than a tool to use throughout for verification and exploratory analysis (Fox & Hendler 2011), which is especially improvident when trialling of methods and systems with sample users is involved. Effective visualisations require a holistic appreciation of the visualisation process - including the consideration of aesthetics, which are often dismissed as superfluous but in fact have been shown to affect performance and usability (Sonderegger &

Sauer 2010). In contrast, this section highlights a number of studies where *visualising* travel behaviour (or activity monitoring) results *is* the subject - or at least a large part - of the research hypothesis.

There are a number of stages involved in preparing analysed data for communication back to a user or analyst, starting with the considered selection of processed data and summary statistics and moving through platform, layout, design, and aesthetics. With the breadth and depth of data that is produced by sensing platforms, it is likely to be highly favourable to employ visualisation techniques, graphic representations of quantitative data that facilitate *easily navigable* and *understandable* communication. Goodman & Foucault (2006) conducted a 3 month study to assess the impact of technology on fitness, specifically asking for feedback on the use of data visualisations. They found that from a psychological perspective, participants often interpreted visualisations as 'proof of progress' - such as the positive trend over time of a fastest mile. From this, they concluded that *long-term* trend visualisations of a *small range* of metrics are particularly suited to encouraging and motivating fitness improvement over time. However, the authors highlighted the dangers of these visualisations, where poor or lapsing performance is highlighted just as immutably.

This is cautioned because of its potential to cause *cognitive dissonance* (Festinger, 1957). This theory describes the discomfort and stress (dissonance) caused by an inconsistency between an individuals knowledge and behaviour. In this case, knowing that exercise is necessary to become healthier, while not exercising. There are two resolutions to this dissonance: either changing the behaviour (e.g. exercising more) or changing the knowledge (e.g. cease participation in any monitoring). To mitigate this, they suggest a combination of daily/per session reports, which remain in the context of that session, and more longitudinal trends, reducing the chance that the platform will stop being used.

These recommendations were implemented in the MOPET (MObile PErsonal Trainer) Analyser tool (Nadalutti & Chittaro 2007). In evaluating the commercial products that offered fitness visualisation, the authors found that most offered only 'superficial analysis', with limited options for querying and comparing data. The data available when mobile was even leaner, usually offering only in situ values. The MOPET Analyser was developed to improve on these products by offering considered and informative visual communication on two platforms; desktop and mobile. Based on the findings of Goodman & Foucault, they designed the desktop application as a retrospective, detailed, long-term analysis tool and the mobile application as a glanceable summary report. In the single-session version of the desktop application (*Fig. 4*), a number of design considerations were highlighted, such as the effective use of colour (e.g. connotations of green/red scales being good/bad) and multi-panel views (map, summary statistics, graph etc).

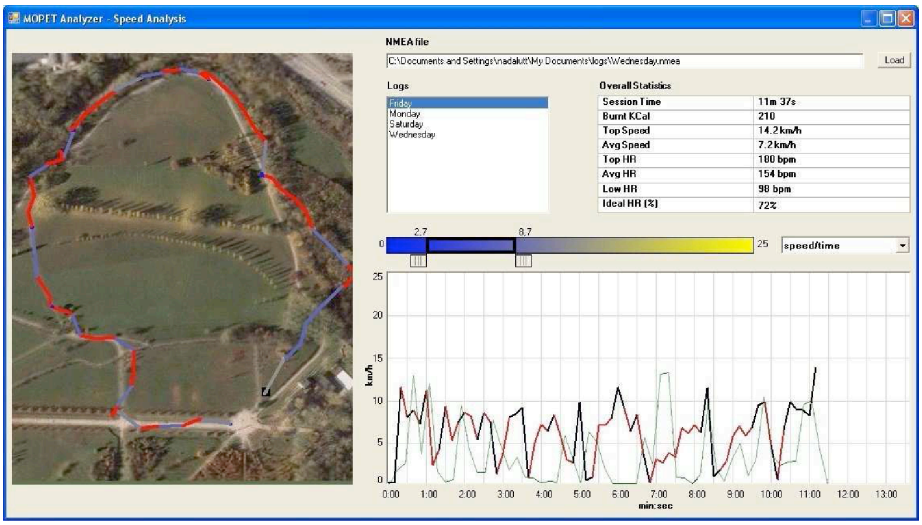

Fig. 4: Single-session visualisation interface, showing a map, summary statistics and a graph of speed over time. From Nadalutti & Chittaro (2007).

One novel element of their visualisation was the use of compressed, colour-coded bars to display activity data (running) from multiple sessions (*Fig. 5*). As the data was all in the same timeframe, the bars could be directly compared to see how the performance varied over and between different runs, and the compression allowed a week or more of data to be compared. A graph was displayed above the bars showing an aggregate

metric, such as pace or heart rate, highlighting trends across multiple sessions.

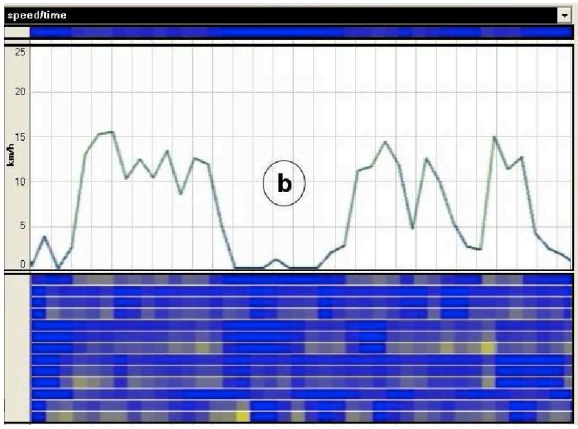

Fig. 5: Multi-session visualisation interface, with coloured bars for each session and a graph showing the aggregate trend. From Nadalutti & Chittaro (2007).

In describing the design choices around the tool, they explained how their decisions were always made with awareness of their audience - non-professional individuals intending to increase their cardiovascular fitness and performance. They note, for example, that if designing a tool for a medical professional they would not have to consider motivation, and may emphasise health indicators such as heart rate to a greater extent.

In more recent examples it is now commonplace to see examples of interactive and/or web-based visualisations. Doherty & Oh (2012) detail the successful construction of a multi-sensor monitoring system for physical activity and travel behaviour, with their web-based visual interface (*Fig. 6*) favourably described as the systems 'culmination'. It is positioned as an analysis tool, to provide assistance to physicians in 'identifying patient-specific spatial-temporal situations that contribute to risky health situations' from large, richly sampled datasets. By creating a web application, *processed* data can be served to the user as required - allowing large datasets to be explored quickly without the need to store and manage data on the users machine. The technologies that power the web also allow researchers to build interactivity into their visualisations, in this case by providing scaling and querying tools that facilitate the study of both long- and short-term patterns at several levels of detail. This is

implemented through 'drill-down' design, where the user can request additional information from an overview by mousing over or clicking the point of interest. A mouseover provides a quick summary of the data at that point, whereas a click will display all available data in a dedicated *'linked view'* window in the top right of the interface.

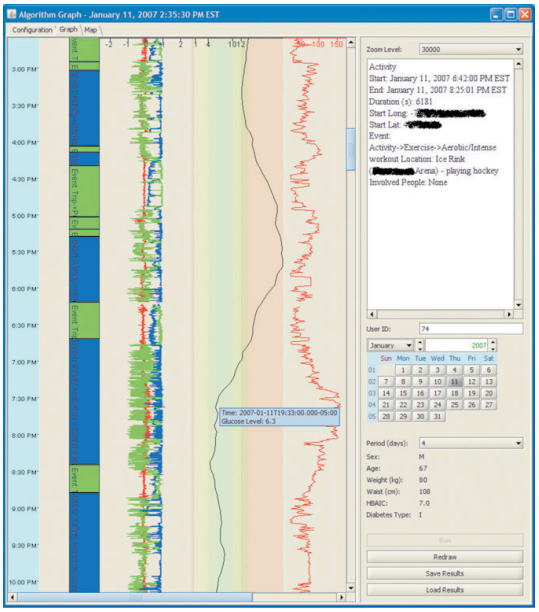

Fig. 6: Web-based analysis interface, displaying data from multiple sensors. From Doherty & Oh (2012).

In literature that deals directly with the construction of such interfaces, these interfaces are known as 'information dashboards' (Few 2006). Few begins by stating that although the technologies that produce, compute and display data are in flux there are many principles and practices that remain relatively constant, allowing widely applicable guidelines to be constructed. In displaying proportional data, for example, he describes how the use of pie charts is almost always inappropriate, even if they are used correctly in displaying parts of a whole. This is because the human eye is not well-suited to accurately comparing two-dimensional areas or angles, the two means that pie charts use to encode quantitative data. A better option would be a horizontal bar graph that clearly shows the high and low values, as well as the comparative differences between them (*Fig. 7*).

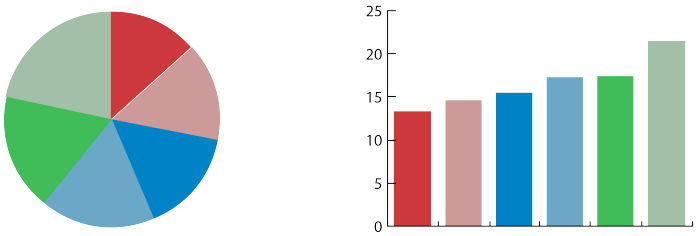

Fig. 7: The same proportional dataset displayed as a pie chart (left) and a bar graph (right). In the pie chart, the relative sizes of the sections are hard to differentiate. Adapted from Few (2006).

The books is full of such suggestions, but several are particularly relevant to the visualisations that will be produced in this project. One, for instance, describes the consideration of colour based on the data type. If the data is sequential, for example, it is appropriate to step from one hue to another ( ). If this colouring was used in a qualitative visualisation, however, it would make the classes hard to differentiate, so it is more appropriate to use a range of different hues ( ). Care is recommended when dealing with colour of any kind, however, with neutral and low saturation colours recommended through the majority of any system. As well as reducing the reading stress caused by an excess of bright colour, the restraint also allows the more vivid colours to stand out as an emphasis for key values. Colours are an important, though not exclusive, example of *preattentive* visual attributes - those which are subconsciously processed - and Few describes their application in differentiating visual elements (*Fig. 8*) as well as detailing the full list of preattentive attributes that can be implemented (*Fig. 9*).

Fig. 8: Finding the number of 5s in the list on the top is much harder than that on the bottom, as it requires ***attentive*** processing to distinguish between and recognise the complex shapes. The list on the right uses colour intensity, an attribute which can be recognised immediately using ***preattentive*** processing, making the number very clear.

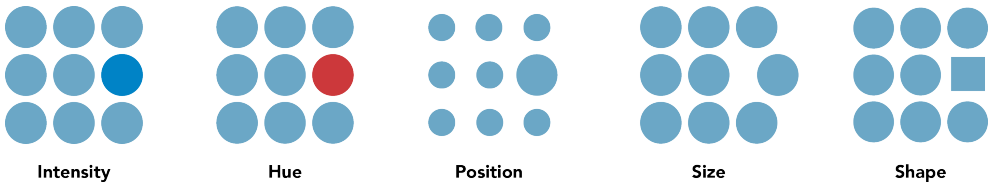

Fig. 9: A selection of preattentive attributes, including colour, form and position. Adapted from Ware (2004).

The utilisation of the techniques and recommendations detailed in this section will be critical in creating an effective visual interface for the application, and they will be considered throughout the construction of the system.

METHODOLOGY

The following sections describe the methods and rationale for each of the stages in the system, from the initial collection of sensor data through processing, analysis and visualisation to produce a finished output.

Data collection

As discussed in the introduction, the system is intended to be based on equipment that is readily available or already owned by the majority of any user group (a smartphone). As we have seen, Ravi et al. (2005) were able to recognise the nature of many activities with good accuracy solely with the accelerometer. Manzoni et al. (2010), however, described how this accuracy was increased further by including GPS data (and speed, derived), and how this also allowed them to better estimate metrics based on distance and speed travelled. Most smartphones produced in the last 3-4 years have included these two sensors, and while the manufacturers and operating systems are still fluctuating in terms of market share, the two most popular systems (Android android.com and iOS apple.com/ios/) are very capable and are designed to run custom software applications that can access these sensors. The data collection process is conducted directly on the unit, requiring an application that can query the sensors and record the results at a regular sampling frequency. This would ordinarily have to be coded from scratch in one of the supported languages of the platforms, which would likely involve a sizeable time investment. However, by utilising the power of open source software it is possible to create a comparable system using only a fraction of the development time. The AntiMap (theantimap.com) is described as an 'open source creative toolset for recording and visualising your own data'. It has been demonstrated in a number of applications, including recording movement data as a snowboarder descends a mountain. It has been released under the open source MIT License (opensource.org/licenses/mit-license.php), which permits the rights to 'use, copy, modify, merge and publish' the software freely. The system is comprised of two parts; a data collection application

for Android and iOS smartphones (AntiMap Log) and a visualisation application for desktop use (AntiMap Simple), and it is the logging application that will be repurposed for this study. The source code is provided in full for both operating systems (github.com/trentbrooks/AntiMap), and in its current release version it is able query and log data from the GPS receiver, clock and compass at a frequency of 30Hz. From this original code, several changes need to be made to adapt the code for this project. These changes are detailed in full in Appendix I.

Firstly, to provide activity recognition it is necessary to query the accelerometer, recording the values from each of the three axes. Secondly, the application does not need to query the compass, as the direction of travel is not an essential feature for activity recognition and disabling it will reduce power drain. Finally, the 30Hz sampling frequency is not essential. While a better frequency does correlate with improved accuracy, it is still possible to retain good accuracy with a lower rate such as 20 or 25Hz (Manzoni et al., 2010) - again reducing power drain and making it more feasible to run the application continuously. As such, a sampling frequency of 20Hz will be used in this study. The data is stored on the device as a comma-separated value (CSV) file, a common and popular open source format because of its non-proprietary and interoperable approach to data storage. This could either be transferred via USB cable or a Wi-Fi network connection to a desktop machine for processing and analysis.

Data processing

The established framework (*Fig. 10*) for taking raw sensor data and translating it into an inferred activity type was set down by Bao & Intille (2004). For each axis a 'window' of samples from the accelerometer trace is taken, usually a base 2 number (such as 128, 256, 512, for an efficient transform computation), with the intention of representing between 6 and 8 seconds of activity (512 samples with 76.25Hz sampling frequency, for

example, gives ~6.7s of data). From this window, the mean (DC feature) of the signal is calculated and a fast Fourier transform (FFT) is applied. Two further features are extracted from the FFT, energy, and entropy (information entropy). The use of mean and energy features have been shown to result in accurate recognition of certain postures and activities, but several activities (such as running and cycling) may produce similar values for these features. To distinguish between such activities, the entropy value is used. An FFT with a single dominant frequency (e.g. 1Hz) - such as cycling - will have a much lower entropy than a less regular signal (e.g. 0.5-2Hz) produced by activities like running. The standard deviation feature, discussed in an earlier section, was used by Ravi et al. (2005) but subsequent studies (Manzoni et al., 2010; Doherty & Oh, 2012) have suggested that it is not necessary and the *energy* of a signal is sufficient to distinguish between activities like walking and running.

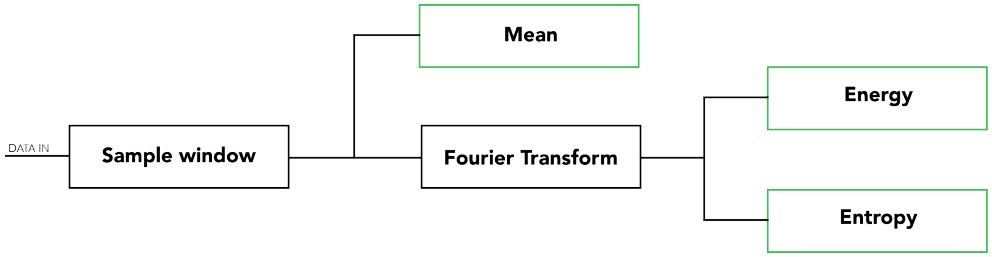

Fig. 10: A simple flow chart demonstrating the data processing actions involved in feature extraction from acceleration data. Outputs shown in green. Adapted from Bao & Intille (2004).

In the Bao & Intille study, however, every sensor used was tested and calibrated before the study to ensure a constant sampling rate - useful for any feature computation and essential for the performance of an FFT. With this study, a wide range of smartphones are to be considered, each with differing components and operating systems. The data coming out of such devices is rarely evenly sampled, so a *pre-processing* stage (*Fig. 11*) is required before the feature extraction (Manzoni et al. 2010). The raw data from each accelerometer axis undergoes a linear interpolation, where the signal is reduced to a series of straight lines. These lines are then resampled at a regular interval to provide an evenly sampled dataset. The

final stage, a low-pass filter, is included to dampen the high-frequency artefacts that are often introduced by this method.

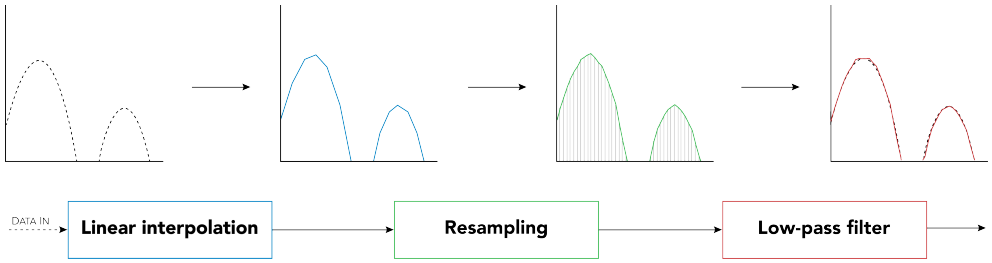

Fig. 11: The pre-processing method for producing an evenly sampled dataset. Adapted from Manzoni et al., 2010.

In order to carry out these methods efficiently the whole process needs to be formalised in a programming language, producing a program that can take raw data as an input and produce the required features as output. A simple overview of the steps involved has been given by figures 10 & 11, but the actual development of the program involves a number of considerations including software, platform and language. The data processing work in this project will be done in R (r-project.org), an open source domain-specific programming language that is particularly well-suited to statistics applications. The standard R distribution provides a solid range of statistical and visualisation techniques, but the real power of the language comes from its expandability through community-created content. Users can submit 'packages' of functions to the Comprehensive R Archive Network (CRAN), a network of file servers that host R and its extensions, and once submitted other users can install the packages using a single line of code within the R command line. In this project, for example, 6 packages are specified; ggplot2, reshape2, splines, signal, entropy and RSEIS, to provide graphical options, reshaping of data arrays, interpolation tools, Fourier transforms, information entropy calculations and filtering tools. The functions contained in these packages are fully explorable and usually have sufficient customisation through parameters that they are a good approximation of the alternative - an ad-hoc function laboriously constructed from scratch.

The lower sampling rate of this study (20Hz) requires a shorter sampling window than the 512 sample window of Bao & Intille (2004) to represent the same recommended 6-8s of data. 128 samples will give a 6.4s window, and is still a base 2 number, so this value is chosen for the first stage of the program. The mean of the sample window is taken first, and then the Fourier transform is applied. The energy is calculated as the sum of the modulus of the transformed sample window, and is normalised by the window length. The entropy is then calculated by estimating the the Shannon entropy (Shannon 1948) of the discrete FFT component magnitudes and the values are written to a row in the results table - along with latitude and longitude values from the GPS signal. Finally, so as not to rely on a particular orientation of the smartphone, the accelerometer-derived features from each of the three axes are then resolved into values that represent the 'total' acceleration with the following method, producing a complete set of features for analysis:

$$a_{tot} = \sqrt{(a_x^2 + a_y^2 + a_z^2)}$$

(From Manzoni et al., 2010)

Data analysis

The data analysis involved in this project consists of two main stages. First, the set of features extracted from the raw data in the processing stage has to be utilised to predict an activity type for each of the sample windows. Then, the longitudinal data needs to be split into individual journeys, bearing in mind that several different trips and transport modes and likely to be seen throughout a typical day. The methods for completing each of these stages are discussed in this section.

The approaches available for activity or transport recognition from sensor data are numerous, with many from the literature already having been discussed in earlier sections. In general, all of the techniques are

based on the idea that an activity produces a certain, predictable value in one or more of the features discussed in the previous section. For example, driving or being driven in a car may result in a low *energy* value (as the accelerometer is not likely to be moving much past the initial acceleration) and a low *entropy* value (since vibrations in cars tend to have a limited number of dominant frequencies), whereas running would give relatively high *energy* values and a greater value for *entropy*. The way these differences are formalised into an automated system is largely where the methods diverge, with some (Ravi et al., 2005) opting for more complex methods such as plurality voting and some (Bao & Intille 2004; Manzoni et al. 2010; Doherty 2012) opting for a simpler decision tree.

In this study, the activities that are intended to be recognised are walking, running, driving, cycling and metro. It will also need to be able to detect inactivity, primarily to distinguish between periods of rest and actual gaps in data coverage. With this relatively low number of fairly distinct activities it is likely to be sufficient to use the decision tree method, an advantage due to its ease of generation and efficiency. This is because once a likely answer is found, the process stops - as opposed to methods such as plurality voting which must evaluate every attribute before making a decision. The tree can be constructed in a number of programming languages, here it will be coded in the Processing language (www.processing.org), another open source effort that builds on the Java language by providing a simpler and more visually driven syntax and development environment. Processing is chosen to keep the number of different software packages to a minimum, as it will also be used in the visualisation part of the project where its unique advantages will be discussed in detail. A series of if/else statements make up the decision tree, an example of which is given here:

```
if (x > 0 && y < 2)          {Activity 1}

else if (x > 0 && y > 2)     {Activity 2}

else if (x > 0 && z < 3)     {Activity 3}
```

An example implementation of a decision tree in Processing, using the features **x**, **y** and **z** to identify four activities.

Since the scaling and sampling values of the processing method used in this study (and every other) are likely to be project-specific, the decision method will need to be *trained* before it is able to perform an automatic recognition. To achieve this, any sample data collected needs to be partitioned into a *training* set and a *test* set before the study, with all calibration and machine learning carried out on the training set leaving the test set as an independent verifier. As a result, the key features will need to be explored and discovered once the actual sample data have been processed, so the construction of *this* decision tree will be detailed in a later section.

With the sensors intended to be recording continuously, there exists another problem in separating individual trips out from the daily record. As discussed in the introductory section, one of the aims of this project is to be able to recognise and highlight discrete, short trips that are currently being taken by car or bus. Schönfelder et al. (2002) and Wolf (2000) concluded, as seen in the review of existing work, that an interval of 120 seconds with little or no activity was an appropriate threshold for defining the split between trips. Both studies considered the splitting of trips to be a processing stage, coming before the analysis discussed in this section. However, this project includes an additional complexity that their studies did not - varying and specific modes of activity and transport. As a result, it is logical to study the journeys not only based on a period of inactivity, but also with a change in transport mode. The first pass of this algorithm would split based on the 120 second interval, and the trip could then be augmented with information about the transport mode, allowing multi-

modal trips to be properly recognised. This would, for example, allow a trip involving two short walks and a 15 minute car journey *(Fig. 12)* to be quantified and possibly highlighted as a walkable trip while also emphasising the fact that almost half of the trip length is already on foot.

Fig. 12: The two-step trip splitting algorithm to be implemented in this study. The trip itself is isolated at the Data Collection stage, and then this trip is processed into multi-modal sections if they are present.

The interval-based split was simple to implement in Processing *(Appendix III)* using a buffer around the user location. If the user did not move out of the buffer within 120 seconds, the trip was considered ended and a new one began. As a new location is read into the algorithm (as signed decimal degrees - e.g. 50.45701), the old location coordinates are compared to the new values. If they are **equivalent** to *three decimal places*, the user location is considered to be near-unchanged and the interval value is increased. The West-East buffer imposed by this process is approximately 11m, and the North-South buffer is dependent on the latitude (~5m at ±60°, ~11m at 0°). If the old and new values are **not equivalent** to three decimal places, the location is considered new and the interval value is reset to zero. The *transport mode* 'split' did not actually separate the data any further, since each trip was already separated into 6-7 second sample windows by the classification algorithm, but the inclusion of such data at this stage facilitates analysis and visualisation later on in the project (as discussed).

Like the decision tree method, the process of recording this trip data is also able to be implemented in Processing, and in this project it will be achieved through utilising the Java class HashMap (docs.oracle.com/javase/1.4.2/docs/api/java/util/HashMap.html). HashMaps (*Fig. 13*) are data structures that store sets of variables as *key* and *value* pairs. The key is the identifier, and is queried to return its associated value. HashMaps are useful for three main reasons. Firstly, they do not require a predetermined

length, so they can be added to as required without producing Out of Bounds error (where a function is trying to place data in a row that doesn't exist as it is beyond the length of the structure). Secondly, they can easily be set up to accept almost any data type for the key/value pair, including custom classes and other structures such as arraylists. This also permits their third major advantage, which is the use of *nested* HashMaps. This technique involves using the value attribute from one HashMap as a key for another, so a HashMap with journey ID (key) and start/end location (value), for example, could be used with a HashMap containing location (key) and place name (value) to return start/end place names for any given journey ID.

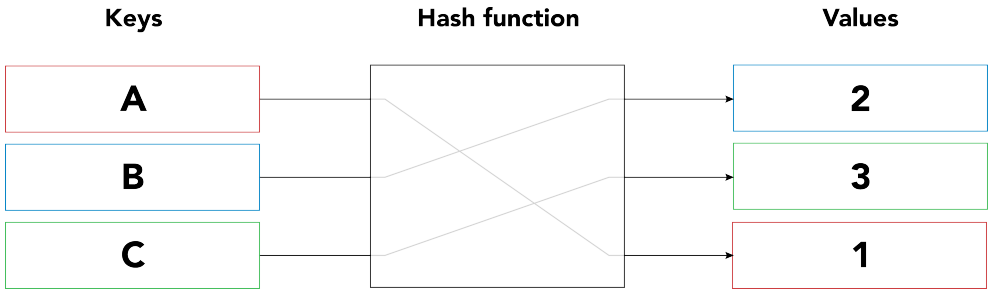

Fig. 13: A schematic representation of a simple HashMap. Keys **A**, **B** and **C** are paired with values **1**, **2** and **3**.

In this project, a HashMap is created for each unique trip. This map pairs a time (key) with an array of data for that time (value). The array will contain latitude, longitude, distance (between current and last locations), speed (over last distance) and inferred activity type (driving, walking, running, cycling, stationary). From this structure, it is possible to look at any trip and query, for example, whether there are any sections of the trip that contain driving, or whether a trip is a particular length or duration. This will be crucial later on in identifying short trips that are currently being driven. The structure of the data also allows it to be visualised easily by iterating through time, which will explained in further detail in the visualisation section of the methodology.

The discrete trips are then collected up into a *master* HashMap, which provides a trip ID number (key) for each complete trip HashMap (value). This is an example of the nested HashMaps discussed earlier, and it allows all of the trips for a particular time period to be collected. It could, for example, contain all the discrete trips from a single day (*Fig. 14*), and using the same process all of these *day* HashMaps could be stored in a *week* or *month* HashMap. This nested structure makes the analysis and display of different time periods of data much easier to explore, encouraging the use of multiple timescales in any visualisation output.

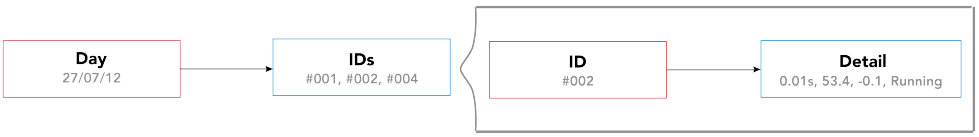

Fig. 14: A schematic representation of the data structure to be used in this project. The second and third HashMaps are 'nested' to the first, allowing their contents to be accessed as required with a single trip ID.

Data visualisation

With the data structure in place, the visualisations of the project output can start to be built. Before going too far, however, the question of *what* to visualise needs to be considered. What is going to be of most benefit to the user, and what is going to keep them interested long enough to understand what they are being shown?

The processes detailed in the previous section produce a number of outputs with varying degrees of applicability and aggregation. One of the most useful statistics is likely to be the overall proportions of each transport type that a user has taken. If a user had decided to cycle into work that day, for example, they would be rewarded with a graphical representation of their progress (e.g. 100% cycling, 15km, 420 cal). As we have discussed, Goodman & Foucault (2006) described these graphical representations as 'proof of progress' - but also cautioned that this can cut

both ways, highlighting failure as clearly as success. Several studies have sought to address this problem in practice, through the use of several different time scales for viewing aggregate data. This ensures that short- and long-term breaks from the program do not discourage the user from continuing to use the system, as a short lapse will not noticeably affect the monthly averages and a daily report will reward renewed progress when the monthly report might be more demoralising (Consolvo et al. 2008).

The multiple timescale approach also provides some secondary benefits. The daily report can help by serving as a persistent reminder of goals and progress. In the form of a 'glanceable display' on a smartphone - a quick and easy to comprehend visual prompt - it too has been shown (*Fig. 15*) to produce a noticeable increase on the long-term maintenance of exercise routines (Consolvo et al. 2008). Also, the weekly or monthly views help greatly in highlighting *long-term* patterns of change. Behavioural economics, which has similarities in trying to describe behavioural change, suggests that awareness of long-term patterns can also have a marked impact on the decision-making process (Rachlin, 2000). In the context of exercise, this would involve a shift in mindset from 'Should I exercise today?' to 'How much have I exercised this week?', which is shown to encourage users to persevere with fitness programs (Chittaro 2011).

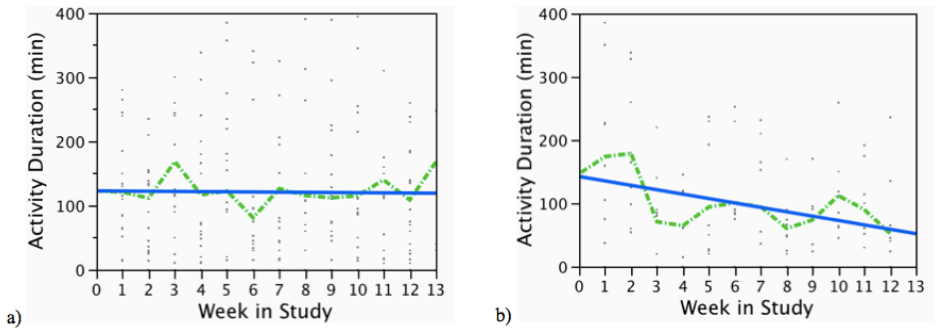

Fig. 15: Average activity duration for participants **a)** with the glanceable display and **b)** without. From Consolvo et al., 2008.

While the daily reports are often described as 'simple' and 'quick' (Chittaro 2011), the weekly and monthly views tend to be deeper and more encouraging of exploration. This, and the inherently spatial

nature of the data, makes them ideally represented as *maps*. Krygier & Wood (2011) suggest that the map is a *'graphic statement that locates facts*, in that it is the visual representation of a statement or proposition that is being made through the display of facts or assertions. In this case, the map is a statement of a users transport choices, with routes that they *could* walk/run/cycle being displayed as a reference tool to help them (and their physicians) identify where they could increase their physical activity. This will form the main part of the long-term visualisation, with a secondary linked view displaying trend data for the same time period.

With the nature and content of the visualisation decided, the implementation of these ideas in code can begin. As discussed the environment and language that will be used for this is Processing, which its creators accurately describe as being designed for *'people who want to create images, animations, and interactions'*. The language is very similar to Java, but with a simple syntax and default functions that make visualisations much easier to create than using Java alone. Having said that, it is also possible to integrate any Java functions, classes or other elements (such as the HashMap class described earlier) directly into a Processing script without translation - making it highly extensible. Additionally, with the new 2.0 version of Processing it is possible to export and convert any programs (or 'sketches') directly to Android and Web platforms with a single click. This means that an application developed for desktop use can quickly be demonstrated on a mobile device or website, covering the most likely development platforms for a given user group. While the final 'release' candidate of a program may in fact be created in a more efficient or capable language, the advantages of Processing facilitate rapid prototyping and iterative design in a way that is covetable in the development stages of an application.

Processing contains a number of classes that are suitable for displaying spatial data, as well as functions for making shapes and interactive display elements with relative ease. The PVector class

(processing.org/reference/PVector.html), for example, can be used to store pairs of values in two- or three-dimensional space. This makes them well-suited to store locations, which are typically made up of a pair of latitude and longitude coordinates. Not limited to storing the attributes, the class also contains a number of methods to make vector calculations simpler to code - such as returning the magnitude (length) of a vector or the distance between two PVectors. In a simple, flat two-dimensional space this would make the calculation of an attribute like speed very straightforward (distance/time), but calculating the distances between two Lat/Lon locations is typically a little more complicated. Because the points are on a sphere (or *almost* a sphere - the Earth), we would ordinarily need to calculate the *great circle* distance to properly estimate how far the user has travelled (*Fig. 16*).

Fig. 16: This schematic describes the problem with Euclidean geometry on a sphere - on the left a straight-line calculation is performed, resulting in a shorter (and incorrect) distance than that of the right - which uses a great-circle calculation.

The established method of calculating the great-circle distance between two points is the Haversine formula (Sinnot, 1984) given by:

$$\mathbf{a} = \sin^2(\Delta lat_{uv} \div 2) + \cos(lat_u) \times \cos(lat_v) \times \sin^2(\Delta lon_{uv} \div 2)$$

$$\mathbf{c} = 2 \times \operatorname{atan}^2(\sqrt{\mathbf{a}}, \sqrt{(1-\mathbf{a})})$$

$$\mathbf{d} = \mathbf{R} \times \mathbf{c}$$

The great-circle distance **d** between between lat_u/lon_u and lat_v/lon_v. **R** is the mean radius of the Earth (6371km).

However, since the distances between GPS points in this study are likely to be relatively small (i.e. less than a few hundred metres), it is a reasonable simplification (and more computationally efficient) to treat the Earth as *locally flat*, resulting in the following formula being used in place of the Haversine:

$$\mathbf{c} = \Delta lat_{uv}^2 + (\cos(0.5 \times (lat_u + lat_v)) \times \Delta lon_{uv})^2$$

$$\mathbf{d} = \sqrt{(\mathbf{R}^2 \times \mathbf{c})}$$

These distances allows the speed of a journey to be derived from lat/lon coordinates, which in turn provides the information necessary to estimate calorie expenditure. Speed is necessary because energy use from physical activity does not scale linearly with effort, so walking at 8km/h, for example, burns more than three times as many calories as a 3.5km/h walk (Ainsworth et al. 2000). Calories are a good measurement to communicate to a user because they are easily relatable in the context of diet, which also provides calorie estimates and daily guidelines.

The final designs of the both the long- and short-term visualisations are likely to change over the course of the study as experimental data will always illuminate a number of potential refinements alongside successes. Krygier & Wood (2011) promote this fluidity as part of their workflow when creating maps, encouraging regular, formative evaluations of the work before adjusting the map (or visualisation) to address any issues that may arise. Fortunately, as discussed, Processing is very flexible in supporting this sort of iterative process.

Although flexible, this section has described the likely format of the visual output as well as some methods and approaches which are appropriate for the data. The final output will be presented and detailed in full in the following Results sections.

RESULTS

Experimental data were collected from *four* participants who collected data over a *minimum continuous* period of *7 days* with no prior instructions given about any particular location or orientation of the phone. Each participant used a different model of smartphone, the devices used being the Apple iPhone 4 (apple.com/iphone/iphone-4/specs.html), the iPhone 3GS (apple.com/iphone/iphone-3gs/specs.html), a Samsung Galaxy II (samsung.com/global/microsite/galaxys2/html/specification.html) and an HTC One X (htc.com/www/smartphones/htc-one-x/#specs).

Most of the data were collected in an urban environment, resulting in mainly *short trip durations* (less than 30 minute). Several longer trips were also included to ensure the system was widely applicable. The following sections detail the undertaking and findings of the methods described above, covering both the classification and visualisation of the sensor data.

Classification

As detailed in the methodology, the decision tree method used in this study for classification of activity types requires a preliminary assessment of training data before it can be fully constructed. This involved applying the classification algorithm (in R) to a discrete half of the total data available, and studying the parameters where the activity type is already known. Through this, it was possible to identify the key features that are consistently distinct between activity types. All of the code for the processes discussed in this section can be found in Appendix II.

The first task, however, was to assess the resampling process (required to generate data with a fixed sampling rate). The method was fairly simple, and so it was reasonably unsurprising that the output of the resampling process was very similar to the raw input signal. This was not verified through any quantitative measures, but sections of the data were

compared throughout the processing stages to verify that the process was not distorting the data beyond reasonable limits (*Fig. 17*). Once this had been completed, the feature extraction algorithm was applied.

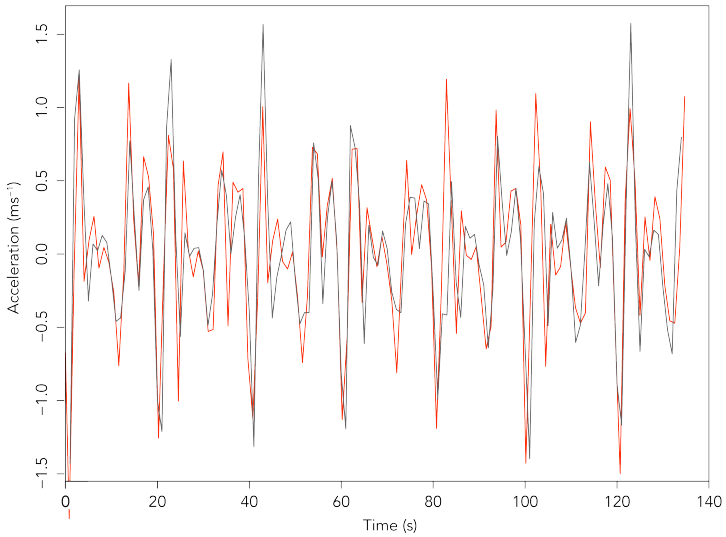

Fig. 17: The **original** and **resampled** signals over a section of accelerometer output.

Energy, or at least its approximation in this study, was seen to be very good at distinguishing between running, walking and driving (*Fig. 18*). When a user is driving (or being driven) the sensor experiences very little movement beyond the initial acceleration and stopping of the vehicle. As a result, the energy value is seen to be low throughout the training data, below 2 at all times. Predictably, walking produced higher energy values, usually between 5 and 10 depending on gait, phone location (i.e. pocket, armband) and pace. While again dependent on pace and technique, running was generally found to give an energy value ~1.5 times larger that walking, although data for some participants showed the multiplier to be closer to 2. Cycling, however, was very unpredictable with regards to energy values, with the sensor placement have a marked effect on the results. If the sensor was place in a back pocket or otherwise close to the the riders centre of gravity, the values were similar to that of driving. However, if the sensor was in a deeper side pocket, or in an armband, the motion was quite large - resulting in an energy value closer to that of running. This resulted in a very low accuracy for cycling recognition in early testing of the algorithm. Clearly, another factor was needed to prevent the

need for prescriptive sensor positions (i.e. 'place the smartphone in your left pocket').

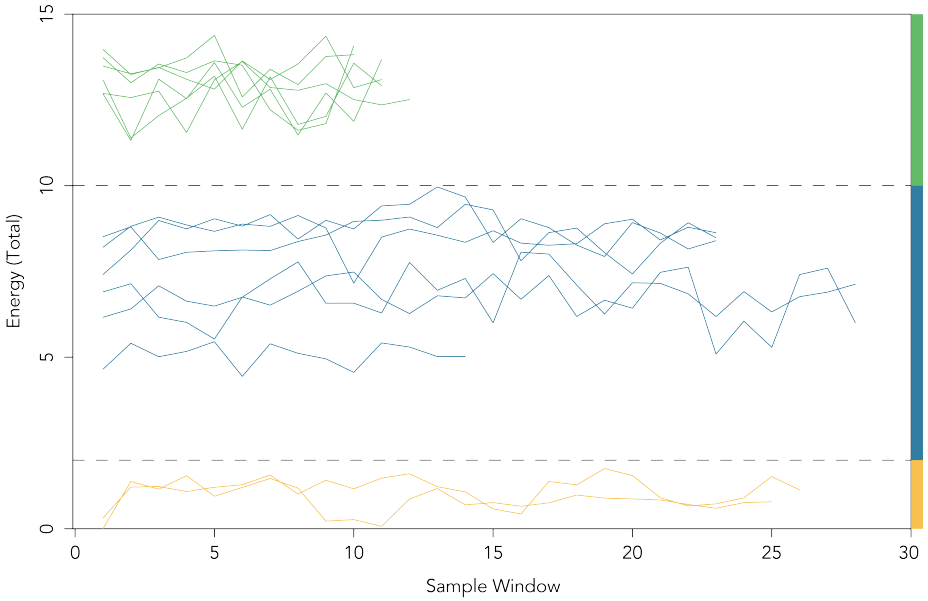

Fig. 18: Trip analysis output from R, showing the total (resolved) values for energy for each sample window along the trip. The distinction between driving, walking and running is highly visible, allowing the fixed values at 2 and 10 (dashed line) to be used in a decision tree.

The feature that worked best in differentiating cycling from the other modes was *entropy* (*Fig. 19*). As discussed in the methodology, this feature is an indication of whether a signal contains just one or two dominant frequencies or a wider range of smaller frequencies (or a 'noisier' signal). When cycling at any speed, the motion of the user is (unsurprisingly) very cyclical. This produces a very strong signal at the frequency of that motion, which in turn produces a relatively *low* entropy value. This is distinct from the other modes of transport being study, which fall into the 'noisy' signal range of higher entropy values. Some cars also displayed a dominant frequency, possibly because of the suspension or a particular engine vibration, but it was weaker and more changeable than that of cycling due to it being a product of both the speed of the vehicle and the changing road surface. As a result, a clear line could be drawn between high and low entropy values to construct a step in the decision tree.

The schematic of the decision tree is given in *Fig. 20*, with each stage from left to right decreasing the number of possible 'candidates' for the transport mode. This was then implemented as a Processing application to make up the final stage of the classification algorithm, where it could be applied to the *test* data (which had been removed from the training set) to validate.

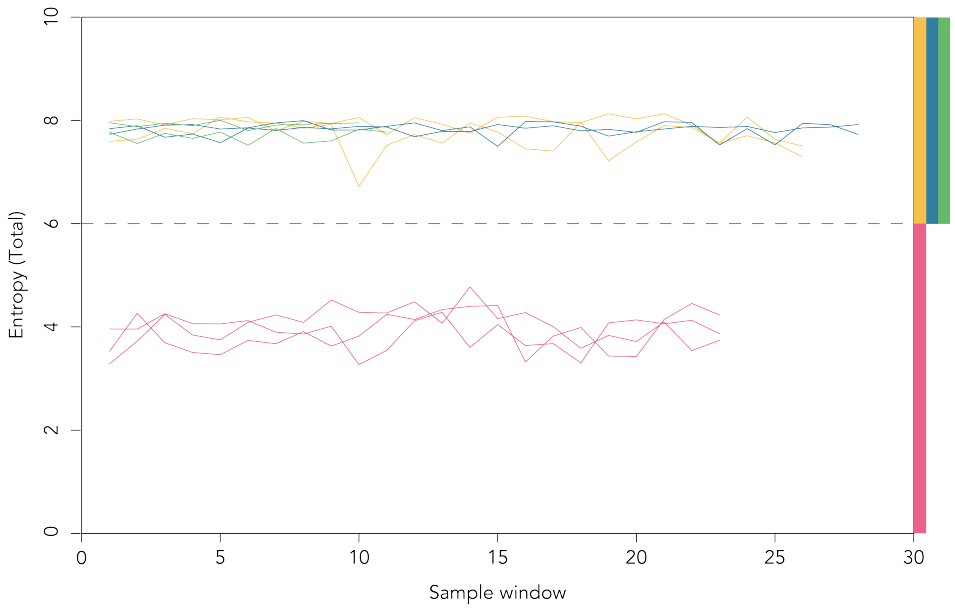

Fig. 19: Trip analysis output from R, showing the total (resolved) values for entropy for each sample window along the trip. The distinction between cycling and driving, walking and running is clear, so a fixed point can be placed at 6 (dashed line).

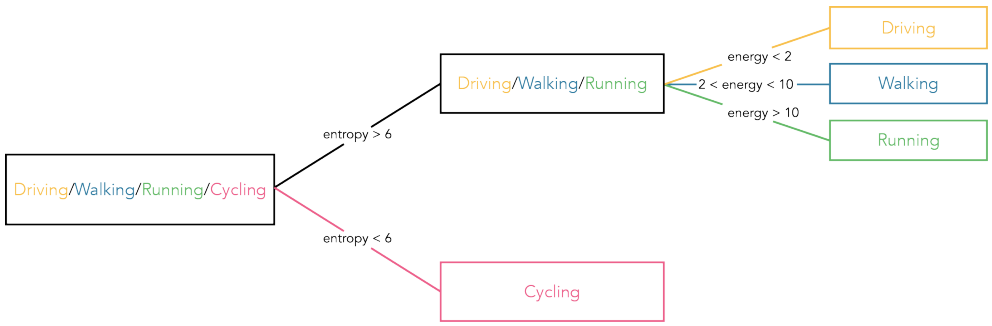

Fig. 20: A schematic diagram of the decision tree used in the classification algorithm. Two factors, **energy** and **entropy**, are used to classify activity into one of four transport modes.

Validation was conducted through this test set using a confusion matrix - a common statistical technique for checking the accuracy of a recognition method. For each of the sample windows, the algorithm predicts a transport mode. This prediction is then checked against the *actual* transport mode, which is known, and the confusion matrix describes

how accurate the algorithms predictions are. This matrix (*Fig. 21*) was created in R by merging two data frames (containing predicted and actual results), and it was then reconstructed in a vector editing program to make it more legible (the default output from R is often very basic).

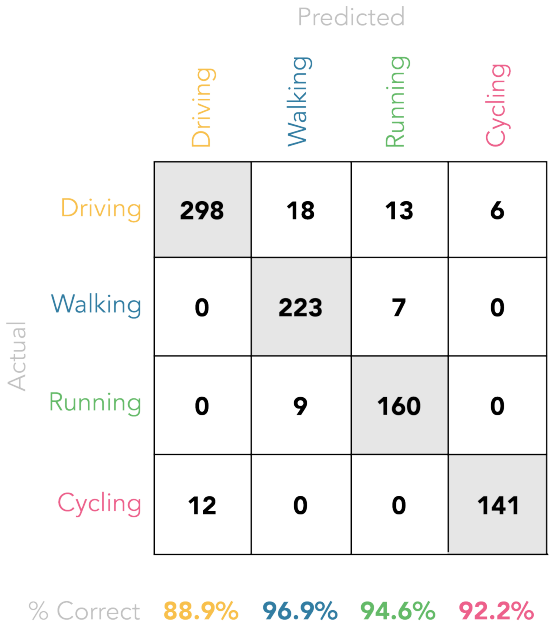

Fig. 21: Confusion matrix from the classification of the test data. The top-left diagonal (shown in **grey**) contains the correctly identified samples, with the total counts in each row being the *actual* number of samples of that type.

The overall classification accuracy of the algorithm was **93.2%**. Walking and running both had high accuracies, with 96.9% of walking and 94.6% of running successfully identified. The false classifications here occurred *between* these two modes, which may say more about the difficulty of retrospectively annotating a trip with activity types than the algorithm accuracy ('Were you fast walking or slow running?'). Cycling accuracy was also respectable at 92.2%, the main problem being that it struggles to recognise 'freewheeling' - where the user does not pedal (when going downhill, for example) thus dropping the strong periodic signal and causing it to be identified as driving. Driving posted the lowest accuracy at 88.9%, which further analysis revealed to be most evident at the stop/starting points of the journey where the strong ac/deceleration caused the energy value to spike (causing it to be wrongly identified as walking or running). This was especially true of heavy traffic, which to the

algorithm looked like periods of being stationary interspersed with short bursts of running or walking.

Before visualisation, the continuous data needed to be split into individual trips. As discussed, this is done by observing the continuous stream of data and looking for periods of non-movement that are greater than 120 seconds. This is relatively easy to implement in the code, placing a small buffer around each sample location and increasing a count when the next location was within that buffer. If the count exceeded the equivalent of 120 seconds (around 19 sample windows), the existing trip was ended and a new trip began. This method worked well for almost all of the data, with the main problems again occurring when driving in heavy traffic and moving only a short distance in the 120 second period. This caused some limited errors in the identification method, as an hour long drive could instead show as four or five 'short' drives.

The selection of the short trips for visualisation was also relatively simple. Rather than trips that were short in duration, the *distance* of the full, *interval-split* (120s gap) trip was used to define 'short' - since a 30 minute drive cannot feasibly be attempted via the other modes of transport in this study. The cutoff distance was designated at **2.41km**, which is the average distance that can be covered at a moderate walking pace (Ainsworth et al. 2000). Once processed in this way, these trips could then be marked and passed through to the visualisation stage of the system.

Visualisation

As discussed in the methodology, the visual interfaces used in this project were all constructed in the Processing language and environment. This section details the iterative processes involved in creating these interfaces, including the design choices and revisions made and the reasoning behind them. While the visualisations are 'finished' in terms of

the project scopes and aims, it should be restated that Processing is most effective as a *prototyping* tool, and the limitations that this imposes on its output are also evaluated in this section. All of the code for the applications discussed in this section can be found in Appendix III.

Early attempts at visualisation focused on assessing the functionality of the data structure (discussed in the methodology) for reading the processed data from R into Processing. This was first achieved through displaying a single trip, simply by drawing a line between GPS locations that was coloured based on the speed and inferred mode of the signal (*Fig. 22*).

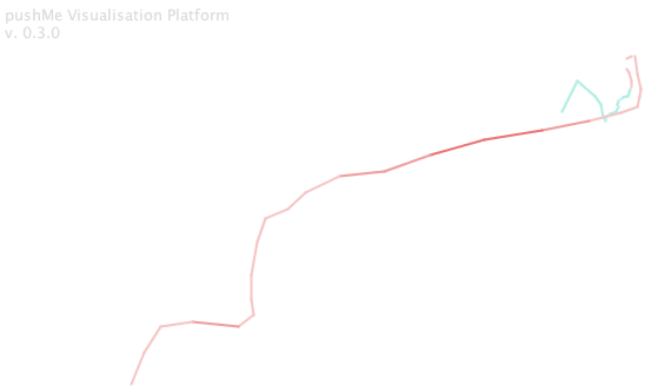

Fig. 22: Early version of the interface, testing the algorithm on a multimodal trip including both walking and driving sections.

With the basic data entry and display processes now confirmed, the first of the visualisations - the daily report - could be constructed. As the data was read in from the R output, values for distance, time and speed were calculated and stored cumulatively for each transport mode. These values could then be used in the daily report, giving a proportional breakdown of distance covered by different transport modes, as well as an estimation of calories burned. The calorie estimations were calculated using a medical compendium of different activities, which provided approximate calorie expenditures for different speeds and body weights (Ainsworth et al. 2000). Speed can be inferred automatically from the GPS readings, but the weight of the user needed to be inputted manually - and it was important that this figure is accurate and current as the difference in

calorie expenditure between a 60kg and 90kg individual was more than 60% for some activities. The daily report needed to be easily understood and require only a short amount of attention, so the bar graph (possessing preattentive attributes, as discussed in Few, 2006) was chosen as the main visual tool to base the interface around. One of the main choices here was whether to make the main attribute of the visualisation a proportional breakdown of *distance* or *time* by each transport mode. Distance was initially dismissed, largely because of concerns that the driving distance could very easily dwarf that of other modes. However, while time looked to be a good alternative, it had the disadvantage of making a 30 minute walk and a 30 minute run look equivalent at a glance - greatly reducing the perceived 'reward' for choosing a more active mode. Calorie estimations were a third potential attribute, but they excluded one of the transport modes (driving) so were not a good choice for the focal point. After trialing the system with the experimental data, it became evident that most users did not make a regular trip of substantial distance by vehicle - and if they were, it was unlikely that it could be taken by an alternative transport mode anyway. As a result, distance was used as the main attribute in the final version of the daily report (*Fig. 23*), with calorie estimations as a secondary attribute to provide context to and support the understanding of the distance values.

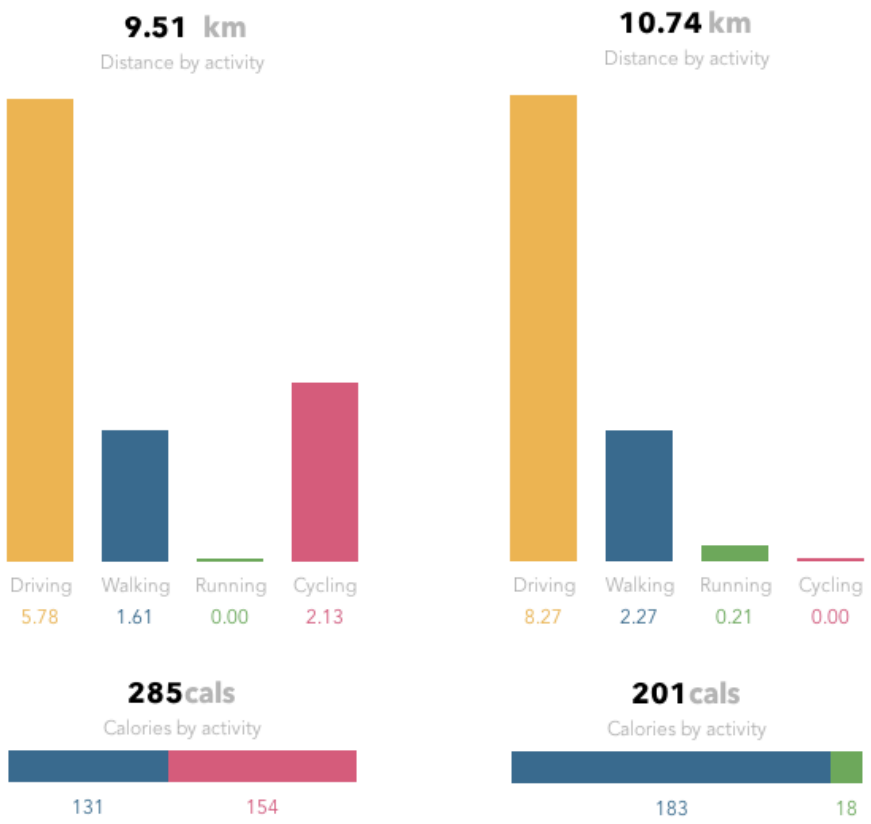

Fig. 23: The daily report visualisation for two different days. Cumulative figures across all transport modes over the day are shown in **black**, with proportional bars and values given in the respective colours of each transport mode.

The construction of the long-term interface was more challenging, as it involved the manipulation of data for displaying on a map. In order to create an automated process, the code needed to be able to pull in a range of different trips and display them all on map at the appropriate scale while maintaining the original dimensions of each trip. This required a responsive design, where the size of the lat/lon canvas would change to fit any given selection of trips, which was achieved through repeated utilisation of the '*map*' command in Processing (processing.org/reference/map_.html). Unrelated to geographic maps, this function is able to take a value, an input range and an output range and *remap* the value from one range to another (*Fig. 24*).

Fig. 24: In this example of the *map* function, the value **4** in the range 0:10 is *re-mapped* to the value **8** in the range 0:20.

This function allows the lat/lon values from the imported trips to be mapped to the same horizontal and vertical ranges in every instance of the program, with the minimum and maximum lat/lon values used for the initial ranges of the map function. The map formed the focal point of the final interface (*Fig. 25*), with short trips (as defined during the data processing stage) over the given period displayed using their relative transport mode colour.

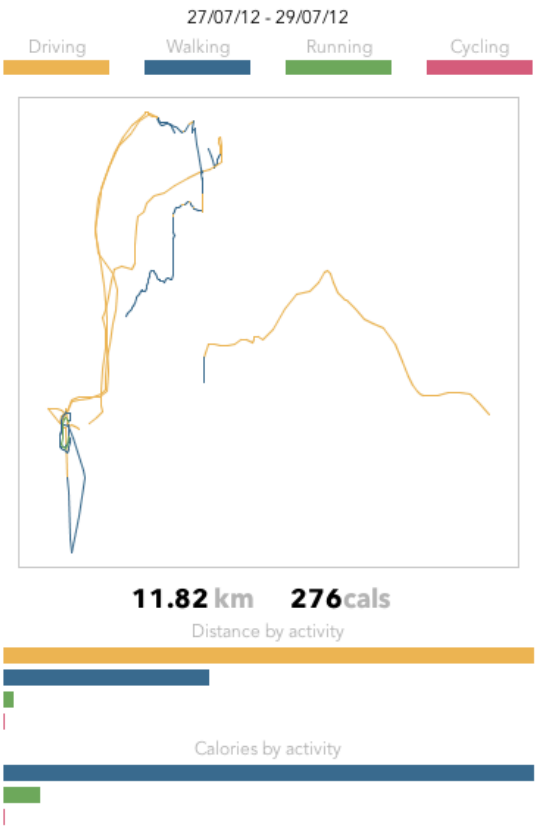

Fig. 25: The long-term viewer, showing sample data from a single user over a week period. The overlaying of trips here serves to illuminate a *regular*, *short* route (going from bottom-to top-left) that is currently being driven, representing a potential opportunity for a physician to try and increase the users weekly physical activity.

As in the daily report, summary statistics across all of the data (including longer trips) are included underneath the map to provide context. Further context would be provided through the use of a background map containing a road layout and building/landmark outlines, but this proved difficult within the confines of the Processing environment. This, and several other potential future enhancements, are detailed in the concluding sections.

Discussion

The prototype workflow developed through this report is fully functional as a procedure for monitoring transport choices on a continuous, individual level. The data collection and processing operations were able to deliver a cross-validated transport mode classification accuracy of 93.2% over five unique states (driving, walking, running, cycling and stationary). This is noticeably higher than the early, multi-sensor approaches to activity recognition, which tended to have accuracies of between 84% (Bao & Intille 2004) and 89% (Ravi et al. 2005). However, although these studies were using between two and five accelerometers (as opposed to the single unit in this study), they were often attempting to recognise between *10 and 20* different activities (some very specific, such as toothbrushing), which was likely to have taxed the classification algorithm significantly more than the four modes in this study. A closer parallel can be found in the work of Manzoni et al. (2010), who achieved an 82% classification accuracy with a single unit approach and observing only 7 different activities (all of them modes of transport). In the literature reviewed during this study only one study was identified with a greater accuracy, having achieved 95% in classifying between two modes - walking and driving (Doherty 2012). In addition, the classification method is relatively simple to implement, and has been shown to be fully deployable using open source technologies such as the R programming language and interface.

The system also performed well in splitting the continuous record into discrete trips. Quantitative assessment of the algorithm proved difficult, as producing a set of 'correct' trips suffered the same data quality problems as the earlier retrospective diary methods (Richardson 2003; Wagner 1997). Small-scale validation and user feedback suggested that the method performed well, with only *specific* instances of heavy traffic causing known failures. Heavy traffic, which was found to have an acceleration signature quite distinct from driving at normal speed, was a problem throughout the

processing stages of the workflow. A simple solution would be to implement a minimum time limit before a new trip could be started. This could create problems in itself, however, so perhaps a better approach would be to augment the algorithm with an additional higher function that was checking for an irregularly rapid generation of new trips and was able to implement some countermeasure.

For the visualisation sections of the project, no experimental assessments of the system, such as focus groups or surveys, were able to be conducted. However, the work produced *is* able to be assessed in the framework of the wider literature on the subject. This literature was considered throughout the design and execution of the visualisation and, as suggested in the methodology, these considerations continued to materialise until the completion of the system.

One of the most evident products of this iterative process is the final colour palette (seen in *Fig. 23)*, which changed a number of times over the course of the project. Harrower & Brewer (2003) describe the difficulty of using colour effectively in visualisations (specifically maps), stating that a 'good' colour scheme '*needs to be attractive, [...] support the message of the map and be appropriately matched to the nature of the data*'. In this visualisation the data is categorical, so the colour scheme should ideally be based on a colours that are easy to differentiate. Thus, the initial colour scheme consisted of red, green, blue and yellow, highly distinctive hues that were likely to aid recognition of different transport mode on a map or graph (they are also known as the 'unique' hues - Hering, 1964). Red was used for the driving mode, green for running, blue for walking and yellow for cycling. This worked relatively well for the daily report, where the largely self-contained bars had equal impact and the negative connotations of the red colour were well employed in the least active transport mode. When seen in the mapped output from the long-term visualisations, however, it became clear that the driving traces created a dominance of the red spectrum, diminishing the other traces and

potentially resulting in an negative response from the user. As a result, the hues were rearranged to make driving yellow, and the red hue was shifted slightly into a pink to reduce the emotional impact. Further refinements saw the lightness of all of the colours reduced, according to the Few (2006) principle that recommends the use of subtler, muted colour throughout the majority of a visualisation to reduce the strain on the viewer. The result was a well-rounded palette that worked equally well in a chart or map graphic.

One of the biggest challenges in producing the visualisation was the creation or sourcing of background maps over which the sensor data would be displayed. The process is relatively simple in theory - using the pairs of minimum and maximum lat/lon values, a geographically accurate quadrilateral can be extracted from a larger map and placed behind the trip traces in the visualisation. One method of implementing this is through referencing a *local* resource that contains map tiles (a common format for storing digital maps) in a way that the appropriate area is returned as an image. This is possible, and was implemented successfully in this study using the TileMill software from MapBox (mapbox.com/tilemill/), which can prepare and export a map for almost any given scale and size. TileMill is able to make use of OpenStreetMap data, and one of its major advantages is the ease in which this resource can be remodelled into an appropriated styled map. In this application, for example, the map needed to be neutral (so as not to distract or otherwise interfere with the actual data) and the design and deployment of such a map was made simple by the software (*Fig. 26)*. Unfortunately, even if limited to the UK and a single zoom level, the combined size of these tiles is in the range of several gigabytes. This increases the install size by several orders of magnitude and as such is likely to be beyond reasonable in what is otherwise a very lightweight piece of software.

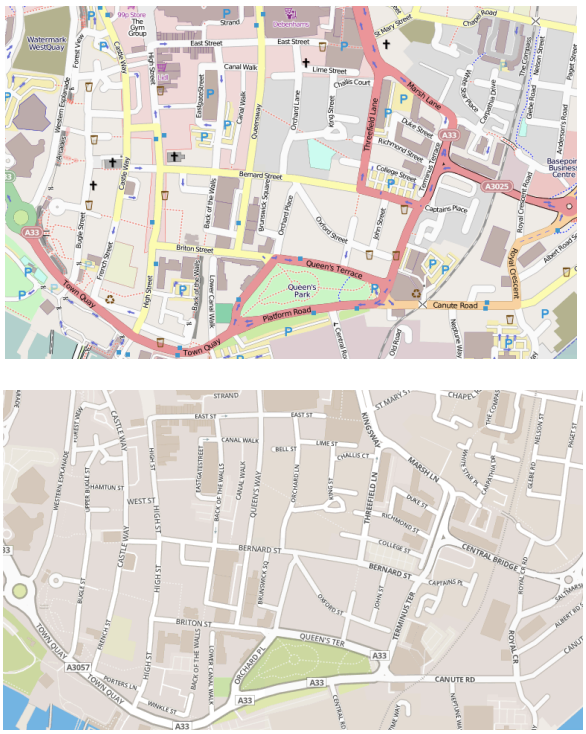

Fig. 26: Comparison between original OpenStreetMap data with default styling (top) and TileMill output (bottom).

The alternative to local storage is to request the map area from a *web mapping* service, such as Google, Yahoo or Bing, via an application programming interface (API) request. Given some input parameters these services are able to provide the appropriate tiles as a download which can then be place into the visualisation, removing the need for the large database of tiles to be stored locally. However, rather than quadrilateral of max/min lat/lon values, all of the APIs that were tested used a *centre point* location and a fixed *zoom level* to make up the request, resulting in a map that was very unlikely to fit neatly behind the data without translation or additional code. More important than this, though, it that the maps supplied by these services are all designed to aid navigation, so they tend to be more similar to OpenStreetMap in *Fig. 25* than the TileMill styling. Once combined with the experimental data this can easily result in a visually crowded and unreadable map, making this method suboptimal as well.

There is a solution in between these two methods, which involves hosting custom map tiles on a web server. TileMill is again a good choice

for this approach, as it allows the user to upload their content to MapBox, the parent company of the software package who specialise in map hosting. This approach is able to combine the advantages of the previous two methods, permitting a *custom* TileMill map to be served to a user in a similar way to the other web mapping services. With MapBox, it is also free for low-usage applications, and it is accessible using a JavaScript (another programming language) API. It is not possible to implement in the current (Java-based) Processing version of the code but, with the advent of Processing 2.0 and its JavaScript export option, it could be possible in a future, web-based version of the visualisation.

The project, then, excels in providing a functional workflow, and crucially the initial data collection *is* able to be run continuously - one of the ambitions set out in the introduction. The reduced sampling rates of the accelerometer (20Hz, down from 80-25Hz in previous studies) and GPS receiver meant that it was possible to run the application for 8 hours continuously on all of the Android phones (Samsung and HTC) trialed over the course of the study, which had previously not be possible or had required the use of external battery packs (Doherty 2009). Unfortunately, the iPhone does not currently allow applications to collect sensor data when the display is locked, and the additional power drain caused by the backlight having to be on meant that it fell short of a full day of use (4-5 hours was the average) and battery packs needed to be used for the two devices used in the study. There is currently no way around this problem, making the Android platform more favourable for a larger-scale deployment, although the iPhone does perform just as well in every other regard when augmented with a battery pack. Despite the successes of the system, power demand in both devices remains a consideration for future versions of the system, as the demand was still substantial and so could foreseeably impact long-term continuous use of the system.

The other desired property of the workflow was that it would be *fully automated*. While the system does not require any user annotation or

equipment maintenance (as has been shown in previous studies), it currently functions through three separate pieces of software - requiring the user to manually pass the files from program to program. Each of the three software packages; AntiMap, R and Processing have been chosen for their unique strengths, and in their current iterations the algorithms they contain could not be easily converted into a single, integrated package. The manual process is not especially complex, involving some file movement and running of scripts, but it could sufficiently hinder a less computer-literate user into avoiding the system.

Two pathways exist for the solution of this problem, both providing an integrated, end-to-end application that requires no user input other than some initial personal details and pressing 'start'. The first would be to produce a web application. Data collection on the phone would work in a very similar manner using a simple application, but rather than storing the whole database locally, it would upload it using a data connection (where possible) and refresh the local storage. This data can then be accessed through a web interface, which could be made to work very similarly to the current Processing application using the aforementioned JavaScript export option. One of the best features of a web application is that it can be made universal, so that any device with a web browser (regardless of platform, operating system or browser) could access the same visual output. There would be an initial tradeoff in terms of battery life, as uploading the data to a remote server would increase the power demand of the application, but this could be offset to some extent by moving some or all of the data processing operations onto the server.

The second approach would be to create an dedicated software application containing the entire workflow for both the smartphone and desktop environments. The main advantage of a bespoke approach over a web application is that it can take full advantage of the system it is being run on, such as taking up the whole of the screen or accessing more system resources. Web applications are often limited by the browser in this

regard. The conversion and integration of the existing code would again be reasonably straightforward, as most operating systems will support several different languages, but it would typically require a much greater time investment to get the application working satisfactorily on most of the popular platforms (Windows, Mac, Linux, Android, iOS, Blackberry, Windows Mobile, etc) than a web application would require. In both approaches, the R script would need to be rewritten in a more transposable format, as its domain-specific syntax does not integrate well with more widely used languages such as Java. This would likely require sourcing equivalent libraries to replace the packages used in the R script (such as the fast Fourier transform), even rewriting the functions out from scratch if necessary.

Both approaches, then, have advantages and disadvantages, and these would need to be evaluated fully in any future work that considered developing the application. It should also be noted that they are not necessarily mutually exclusive, with many companies taking a combined approach (*Fig. 27*).

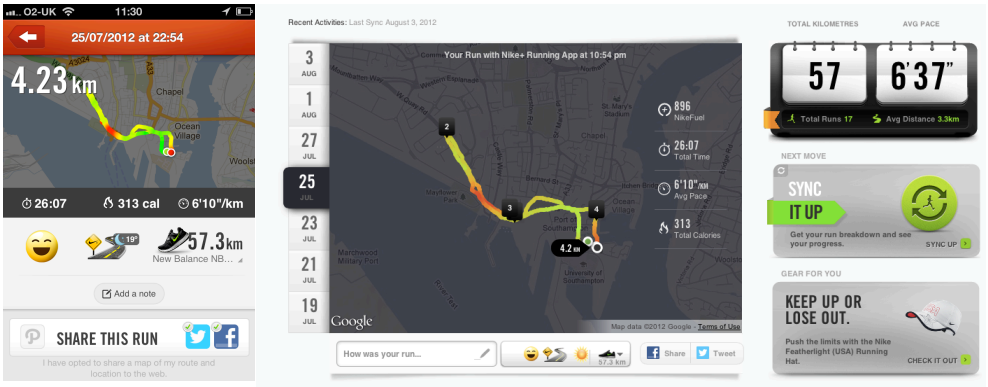

Fig. 27: The bespoke (left) and web (right) versions of the Nike+ interface.

One final development that could make the application more relatable would be the inclusion of additional supplementary statistics calculated from the transport data. This could include, for example, an estimation of the CO_2 emissions caused by the users transport choices, which could be influential if a user is concerned about the environment. For the more thrifty users, an estimation of fuel cost savings could be displayed when

the user decides not to drive. This additional information could help to encourage the user to make positive transport choices, by increasing the positive reinforcement mechanisms established in this study with distance and calorie information.

Conclusion

The primary goal of this study was to assess the feasibility of producing an automated, continuous monitoring system that could log, process, classify and visualise personal transport data. This essay has argued that such a system is entirely viable, detailing the considerations and technologies required to implement the constituent processes and proposing, then demonstrating, a working prototype workflow for the system. Despite opting for a lower sampling rate than previous work, cross-validated results from the experimental data demonstrated the successes of the system in classifying movement by transport type, with an accuracy value that was higher than almost all of the existing work reviewed during the study. In contrast to most of the research of this type (which has typically stopped at the classification stage), the processed data was then able to be communicated back to the user through one of two visual interfaces, with their design informed by existing work in the field and established wisdom from general information visualisation theory. The data structure outlined in the processing stages allowed structured queries to be made on the data, allowing trips of specified length and transport mode to be highlighted to the user. This was utilised to display short distances that the user was currently driving, one of several decisions intended to positively influence a users transport choices.

Several limitations to this small-scale study were discussed, including some known classification errors, producing and serving cartographically appropriate maps for the visualisation, and the manual requirements of the system workflow. These limitations were addressed in kind with suggestions for future development, along with a feature list of additional elements that provide direction on future development within the field.

In closing, the increasing ubiquity and capabilities of smartphone technology are bringing about an exciting opportunity to provide greater access to previously specialised healthcare tools. The use of existing equipment and the functional, non-invasive approach detailed in this study

are complemental to this opportunity, addressing several of the current barriers to the widespread adoption of such systems. Further studies with a refined workflow and larger sample groups need to be conducted, but this study presents a clear step forward in the provision of a viable system.

REFERENCES

Ainsworth, B.E. et al., 2000. Compendium of physical activities: an update of activity codes and MET intensities. *Medicine & Science in Sports & Exercise*, 32(9), pp.498.

Arteaga, S.M., Kudeki, M. & Woodworth, A., 2009. Combating obesity trends in teenagers through persuasive mobile technology. *ACM SIGACCESS Accessibility and Computing*, (94), pp.17–25.

Bao, L. & Intille, S., 2004. Activity recognition from user-annotated acceleration data. *Pervasive Computing*, pp.1–17.

Bouten, C.V.C. et al., 1997. A triaxial accelerometer and portable data processing unit for the assessment of daily physical activity. *Biomedical Engineering, IEEE Transactions on*, 44(3), pp.136–147.

Chittaro, L., 2011. Designing visual user interfaces for mobile applications. *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*, pp.331–332.

Consolvo, S. et al., 2008. Flowers or a robot army?: encouraging awareness & activity with personal, mobile displays. *Proceedings of* the 10th international conference on Ubiquitous computing, pp.54–63.

Doherty, S.T., 2012. A Multi-Sensor Monitoring System of Human Physiology and Daily Activities. *Telemedicine and e-Health*.

Doherty, S.T., 2009. Emerging methods and technologies for tracking physical activity in the built environment. *Transport survey methods: keeping up with a changing world*, Emerald Group Publishing Limited, Bingley, UK, pp.153–190.

Festinger, L., 1957. *Theory of Cognitive Dissonance*. Stanford, CA: Stanford University Press.

Few, S., 2006. *Information Dashboard Design*, Sebastopol, CA: O'Reilly Media, Inc.

Fox, P. & Hendler, J., 2011. Changing the Equation on Scientific Data Visualization. *Science*, 331(6018), pp.705–708.

González, M.C., Hidalgo, C.A. & Barabási, A.L., 2009. Understanding individual human mobility patterns. *Nature*, 458(7235), pp.238– 238.

Goodman, E. & Foucault, B.E., 2006. Seeing fit: visualizing physical activity in context. *CHI'06 extended abstracts on Human factors in computing systems*, pp.797–802.

Harrower, M. & Brewer, C.A., 2003. ColorBrewer.org: An Online Tool for Selecting Colour Schemes for Maps. *The Cartographic Journal*, 40(1), pp.27–37.

Hering, E., 1964. *Outlines of a Theory of the Light Sense*. Harvard University Press, Cambridge, MA.

Hollis, J.F. et al., 2008. Weight loss during the intensive intervention phase of the weight-loss maintenance trial. *American journal of preventive medicine*, 35(2), pp.118–126.

Krygier, J., Wood, D., 2011. *Making maps: a visual guide to map design for GIS*. 2nd ed. New York, NY: Guildford Press.

Laurila, J.K., Gatica-Perez, D. & Aad, I., 2012. The mobile data challenge: Big data for mobile computing research. *Proc Mobile Data ....*

Liao, L. et al., 2006. Building personal maps from GPS data. *Annals of the New York Academy of Sciences*, 1093(1), pp.249–265.

Ly, K., Carlbring, P. & Andersson, G., 2012. Behavioral activation-based guided self-help treatment administered through a smartphone application: study protocol for a randomized controlled trial. *Trials*, 13(1), p.62.

Manzoni, V. et al., 2010. Transportation mode identification and real-time CO2 emission estimation using smartphones.

Nadalutti, D. & Chittaro, L., 2007. Visual analysis of users' performance data in fitness activities. *Computers & Graphics*, 31(3), pp.429– 439.

Rachlin, H., 2000. *The science of self-control*. Cambridge, MA: Harvard University Press.

Neuhaus, F., 2009. UrbanDiary - a Tracking Project. *CASA Working Paper Series*, 151.

Ravi, N. et al., 2005. Activity recognition from accelerometer data. *Proceedings of the National Conference on Artificial Intelligence*, 20(3), p.1541.

Reades, J., Calabrese, F. & Ratti, C., 2009. Eigenplaces: analysing cities using the space-time structure of the mobile phone network. *Environment and Planning B: Planning and Design*, 36(5), pp.824–836.

Richardson, A.J., 2003. Behavioral mechanisms of nonresponse in mail-back travel surveys. *Transportation Research Record: Journal of the Transportation Research Board*, 1855(-1), pp.191–199.

Schönfelder, S. et al., 2002. Exploring the Potentials of Automatically Collected GPS Data for Travel Behaviour Analysis: A Swedish Data Source.

Shannon, C.E., 1948. A Mathematical Theory of Communication. *AT&T Bell Labs*.

Sinnott, R., 1984. Virtues of the Haversine. *Sky and Telescope,* 68 (2), pp.159

Sonderegger, A. & Sauer, J., 2010. The influence of design aesthetics in usability testing: Effects on user performance and perceived usability. *Applied Ergonomics*, 41(3), pp.403–410.

Stuckey, M. et al., 2011. Remote Monitoring Technologies for the Prevention of Metabolic Syndrome: The Diabetes and Technology for Increased Activity (DaTA) Study. *Journal of diabetes science and technology*, 5(4), p.936.

Van Laerhoven, K., Schmidt, A. & Gellersen, H.W., 2002. Multi-sensor context aware clothing. *Wearable Computers, 2002.(ISWC 2002). Proceedings. Sixth International Symposium on*, pp.49–56.

Wagner, D.P., 1997. Lexington Area Travel Data Collection Test. *GPS for Personal Travel Surveys, Final Report for OHIM, OTA, and FHWA, Battelle, Columbus*.

Wang, J. et al., 2012. Effect of adherence to self-monitoring of diet and physical activity on weight loss in a technology-supported behavioral intervention. *Patient Preference and Adherence*, p.221.

Ware, C., 2004. Information visualization: perception for design. *Morgan Kaufmann Publishers*, 22.

Wolf, J., 2000. Using GPS data loggers to replace travel diaries in the collection of travel data.

Zheng, Y. et al., 2008. Understanding mobility based on GPS data. *Proceedings of the 10th international conference on Ubiquitous computing*, pp.312–321.

APPENDIX I: ANTIMAP CODE REVISIONS

DATA COLLECTION

The following is presented as a series of ***before*** and ***after*** code snippets from the version 1.0 AntiMap Log application (github.com/trentbrooks/AntiMap/). The changes made to adapt the software to this study (either **edits**, **additions** or **removals**) were very small in what is is large amount of code, so a succinct approach is appropriate.

Querying Accelerometer

In method setup().

Original

```
// DEFAULT properties
    // gps + compass
    currentLatitude = 0;
    currentLongitude = 0;
    currentLatitudeStr = "0.000000";
    currentLongitudeStr = "0.000000";
    direction = 0; // compass, not bearing.
    directionStr = "0";
    currentSpeed  = 0;
    currentSpeedStr = "0.00";
    cumulativePhoneKms = 0.0f; // value from phone, kilometers
    cumulativePhoneKmsStr = "0.00";

// setup sensors
    ofRegisterTouchEvents(this);
    ofxiPhoneSetOrientation(OFXIPHONE_ORIENTATION_PORTRAIT);
    coreLocation = new ofxiPhoneCoreLocationExtra();
            hasCompass = coreLocation->startHeading();
            hasGPS = coreLocation->startLocation();
    //[UIApplication sharedApplication].idleTimerDisabled = YES;
    ofxiPhoneDisableIdleTimer(); // disable phone sleep
    printf("\nPhone: %s\n", iPhoneGetDeviceRevision().c_str());
```

Edit

```
// DEFAULT properties
    // gps + compass
    currentLatitude = 0;
    currentLongitude = 0;
    currentLatitudeStr = "0.000000";
    currentLongitudeStr = "0.000000";
    direction = 0; // compass, not bearing.
    directionStr = "0";
    currentSpeed  = 0;
    currentSpeedStr = "0.00";
```

```
    cumulativePhoneKms = 0.0f; // value from phone, kilometers
    cumulativePhoneKmsStr = "0.00";
    currentX = 0;
    currentY = 0;
    currentZ = 0;
    interval = 0;

// setup sensors
    ofRegisterTouchEvents(this);
    ofxiPhoneSetOrientation(OFXIPHONE_ORIENTATION_PORTRAIT);
    coreLocation = new ofxiPhoneCoreLocationExtra();
            hasCompass = coreLocation->startHeading();
            hasGPS = coreLocation->startLocation();
    coreMotion = new CMMotionManager(); // Create a new object to access coreMotion
framework.
            hasAccelerometer = startAccelerometerUpdates(); // This readies the
accelerometer to be queried.
    //[UIApplication sharedApplication].idleTimerDisabled = YES;
    ofxiPhoneDisableIdleTimer(); // disable phone sleep
    printf("\nPhone: %s\n", iPhoneGetDeviceRevision().c_str());
```

In method updateLocation().

Addition

```
if (hasAccelerometer)
{
            currentX = coreMotion->accelerometerData.acceleration.x);
            currentY = coreMotion->accelerometerData.acceleration.y);
            currentZ = coreMotion->accelerometerData.acceleration.z);
}
```

In method logData().

Original

```
// write the latitude, longitude, compass direction, speed, distance, duration, and location
tag to the text file.
output << currentLatitudeStr + "," + currentLongitudeStr +      "," + directionStr + "," +
currentSpeedStr + "," +     cumulativePhoneKmsStr + "," + ofToString(time) + "," + ...
```

Edit

```
// removed compass reference, made first value to be written time
output << ofToString(time) + "," + currentX + "," + currentY + "," + currentZ + "," +
currentLatitudeStr + "," + currentLongitudeStr + ...
```

Removing compass

Removed

In method updateLocation().

```
if(hasCompass) {
      direction = int(360 - coreLocation->getTrueHeading());
      directionStr = ofToString(direction);
   }
```

Reducing sampling rate

In method setup().

Original

```
// SETUP
ofBackground(255,255,255);
ofSetFrameRate(30);
ofSetCircleResolution(32);
sw = ofGetWidth();
sh = ofGetHeight();
halfScreenX = (sw * .5);
halfScreenY = (sh * .5);
```

Edit

```
// SETUP
ofBackground(255,255,255);
ofSetFrameRate(20);
ofSetCircleResolution(32);
sw = ofGetWidth();
sh = ofGetHeight();
halfScreenX = (sw * .5);
halfScreenY = (sh * .5);
```

APPENDIX II: FEATURE EXTRACTION R SCRIPT

DATA PROCESSING

R Script

```
require(ggplot2)
require(reshape2)
require(splines)
require(signal)
require(entropy)
require(RSEIS)
results = data.frame("time"=0, "meanX"=0, "meanY"=0, "meanZ"=0, "sdX"=0, "sdY"=0,
"sdZ"=0, "sdT"=0, "energyX"=0, "energyY"=0, "energyZ"=0, "energyT"=0,
"lat"=mean(sample$Latitude[1:140]), "lon"=mean(sample$Longitude[1:140]),
"entropyX"=0, "entropyY"=0, "entropyZ"=0, "entropyT"=0)

# Read in .csv file.
sample=read.csv('sample.csv')

# State sample-specific variables
time = sample$Elapsed.Time

# Split sensor output into 3 dataframes
dX=data.frame(x=time, y=sample$X)
```

```
dY=data.frame(x=time, y=sample$Y)
dZ=data.frame(x=time, y=sample$Z)
ntot = floor(nrow(sample)/140)

for(u in seq(140, 140*(ntot-1), by = 140)) {

  # Define range of samples
  rangeS = u:(u+140)

  # Location
  mlat = mean(sample$Latitude[rangeS])
  mlon = mean(sample$Longitude[rangeS])

  # Create interpolated spline along raw values for each sensor
  interpX = interpSpline(dX$x[rangeS], dX$y[rangeS])
  interpY = interpSpline(dY$x[rangeS], dY$y[rangeS])
  interpZ = interpSpline(dZ$x[rangeS], dZ$y[rangeS])

  # Resample along spline, with a number of segments proportional to the range that the
data covers
  resampleX = predict(interpX, nseg = 128)
  resampleX = data.frame(x=resampleX$x, y=resampleX$y)
  resampleY = predict(interpY, nseg = 128)
  resampleY = data.frame(x=resampleY$x, y=resampleY$y)
  resampleZ = predict(interpZ, nseg = 128)
  resampleZ = data.frame(x=resampleZ$x, y=resampleZ$y)

  # Calculate means of signal components
  meanX = mean(resampleX$y)
  meanY = mean(resampleY$y)
  meanZ = mean(resampleZ$y)

  # Calculate standard deviation of signal components
  sdX = sd(resampleX$y)
  sdY = sd(resampleY$y)
  sdZ = sd(resampleZ$y)

•  # Calculate energy proxy and information entropy from Fourier transform
  window = 128
  freq = 20
  start = 0

  fftX = Mod(fft(resampleX$y-mean(resampleX$y)))
  fftY = Mod(fft(resampleY$y-mean(resampleY$y)))
  fftZ = Mod(fft(resampleZ$y-mean(resampleZ$y)))

  energyX = sum(fftX)/window
  energyY = sum(fftY)/window
  energyZ = sum(fftZ)/window

  entropyX = entropy(fftX)
  entropyY = entropy(fftY)
  entropyZ = entropy(fftZ)

  row = nrow(results)+1
  results[row,1] = time[u]
  results[row,2] = meanX
  results[row,3] = meanY
  results[row,4] = meanZ
  results[row,5] = sdX
```

```
  results[row,6] = sdY
  results[row,7] = sdZ
  results[row,8] = sqrt((sdX^2)+(sdY^2)+(sdZ^2))
  results[row,9] = energyX
  results[row,10] = energyY
  results[row,11] = energyZ
  results[row,12] = sqrt((energyX^2)+(energyY^2)+(energyZ^2))
  results[row,13] = mlat
  results[row,14] = mlon
  results[row,15] = entropyX
  results[row,16] = entropyY
  results[row,17] = entropyZ
  results[row,18] = sqrt((entropyX^2)+(entropyY^2)+(entropyZ^2))

  cat("Pass",nrow(results)-1,"of",ntot-1,"Complete ","[",u,"to",u+140,"]","\n")
}
print("Run Complete")
write.csv(results, "results.csv")
```

Package References

ggplot2: H. Wickham. *ggplot2: elegant graphics for data analysis.* Springer New York, 2009.
reshape2: http://cran.r-project.org/web/packages/reshape2/reshape2.pdf
splines: http://stat.ethz.ch/R-manual/R-devel/library/splines/html/interpSpline.html
signal: http://r-forge.r-project.org/projects/signal/
entropy: http://cran.r-project.org/web/packages/entropy/entropy.pdf
RSEIS: http://cran.r-project.org/web/packages/RSEIS/RSEIS.pdf

APPENDIX III: PROCESSING SKETCHES

DATA ANALYSIS & VISUALISATION

Trip Splitting

```
String [] entries = loadStrings("results.csv");
PrintWriter output;
int tNo = 1;
int interval = 0;
int i = 1;

void setup()
{
  // Create a new trip file in the sketch directory
  String s = "trip" + tNo + ".csv";
  output = createWriter(s);
}

void draw()
{
  if (i < entries.length);
  {
    String[] tparse = split(entries[i], ",");
    Float lat = parseFloat(tparse[13]);
    Float lon = parseFloat(tparse[14]);
    int latRound = Math.round(lat*10000);
    int lonRound = Math.round(lon*10000);
```

```
    if (latRound == lonRound) interval++;   // If equivalent to 3dp, these two number will be
                                            // equal and the boolean will be true
    else interval = 0;

    if (interval < 20) // An interval of 19 trip windows is equivalent to 121.6 seconds (19*6.4)
    {
      output.println(entries[i]);  // Write the entire data line to the new trip file
    }
    else newTrip();

    i++;
  }
}

void newTrip() {
  output.flush();
  output.close();
  tNo++;
  setup();
}
```

Daily Report

```
Float s, e, lon, lat, time, cals, lastS, smax, latmax, latmin, lonmax, lonmin;
Float dDist, wDist, rDist, cDist, dTime, wTime, rTime, cTime;
Float wCals, rCals, cCals;
Float walkSlow, walkFast, runSlow, runFast, cycleSlow, cycleFast;
Float [] values;
PFont fontSmall, fontBig;
HashMap<Float, HashMap<Float, Float []>> master = new HashMap<Float,
HashMap<Float, Float []>>();
PVector posn, l, lastR, lastP;
int wmeas;
//--------USER INPUT-------//
int trips = 7;            // <-- INPUT NUMBER OF SEPARATE TRIPS [trip1.csv, trip2.csv,
trip3.csv...]
int kg = 70;                                //
//-------------------------//

//-----------------------------------------------------------------------------------------//
//-----------------------------------------------------------------------------------------//
//-----------------------------------------------------------------------------------------//
void setup()
{
  size(400, 600);
  background(255);
  fill(0);
  strokeWeight(2);
  fill(220);
  noLoop();

  textAlign(CENTER);
  fontSmall = loadFont("Avenir-Book-13.vlw");
  fontBig = loadFont("AvenirNext-DemiBold-18.vlw");

  dDist = 0.0;
  wDist = 0.0;
  rDist = 0.0;
  cDist = 0.0;
```

```
  dTime = 0.0;
  wTime = 0.0;
  rTime = 0.0;
  cTime = 0.0;
  time = 0.0;
  wCals = 0.0;
  rCals = 0.0;
  cCals = 0.0;
  cals = 0.0;
  lastP = null;
  wmeas = height/12;

  for (int i = 1; i <= trips; i++)
  {
    String input = "trip" + i + ".csv";
    Float j = new Float (i);
    loadEntries(input, j);
  }

  calSetup();

  int keywidth = (width-40)/5;
  int keyspace = keywidth/3;

  noStroke();
  textFont(fontSmall, 12);
  textSize(12);
}

//-------------------------------------------------------------------------------------------//
//-------------------------------------------------------------------------------------------//
//-------------------------------------------------------------------------------------------//
void draw()
{
  smooth();

  for (int i = 1; i <= trips; i++)
  {
    Float j = new Float (i);
    HashMap<Float, Float []> data = master.get(j);
    while (time/10 <= smax)
    {
      drawTrip(data);
      time++;
    }
    time = 0.0;
    lastP = null;
  }

  // Activity bars

  int barWidth = (width-160)/4;
  textFont(fontBig, 18);
  textSize(18);
  fill(0);
  float tKM = Math.round(dDist + wDist + rDist + cDist)/10;
  text(String.format("%.2f", tKM/100), width/2-17, 60);
  fill(180);
  text("km", width/2+27, 60);
  textFont(fontSmall, 12);
```

```
textSize(12);
text("Distance by activity", width/2, 80);

text("Driving", 50+barWidth, height-190);
text("Walking", 50+2*barWidth, height-190);
text("Running", 50+3*barWidth, height-190);
text("Cycling", 50+4*barWidth, height-190);

float mDist = dDist;
if (wDist > mDist) mDist = wDist;
else if (rDist > mDist) mDist = rDist;
else if (cDist > mDist) mDist = cDist;

float mTime = dTime;
if (wTime > mTime) mTime = wTime;
else if (rTime > mTime) mTime = rTime;
else if (cTime > mTime) mTime = cTime;

noSmooth();
fill(245, 180, 60);
stroke(245, 180, 60);
rect(30+barWidth, height-210, 40, 0-map(dDist, 0, mDist, 0, height-310));
float dKM = Math.round(dDist/10);
text(String.format("%.2f", dKM/100), 50+barWidth, height-170);

fill(43, 107, 145);
stroke(43, 107, 145);
rect(30+2*barWidth, height-210, 40, 0-map(wDist, 0, mDist, 0, height-310));
float wKM = Math.round(wDist/10);
text(String.format("%.2f", wKM/100), 50+2*barWidth, height-170);

fill(85, 177, 85);
stroke(85, 177, 85);
rect(30+3*barWidth, height-210, 40, 0-map(rDist, 0, mDist, 0, height-310));
float rKM = Math.round(rDist/10);
text(String.format("%.2f", rKM/100), 50+3*barWidth, height-170);

fill(233, 71, 123);
stroke(233, 71, 123);
rect(30+4*barWidth, height-210, 40, 0-map(cDist, 0, mDist, 0, height-300));
float cKM = Math.round(cDist/10);
text(String.format("%.2f", cKM/100), 50+4*barWidth, height-170);

// Calorie estimate
textFont(fontBig, 18);
textSize(18);
fill(0);
text(Math.round(cals), width/2-17, height-120);
fill(180);
text("cals", width/2+17, height-120);

textFont(fontSmall, 12);
textSize(12);
fill(180);
text("Calories by activity", width/2, height-100);
noStroke();
fill(43, 107, 145);
rect(90, height-90, map(wCals, 0, cals, 0, width-180), 20);
if (wCals > 0) text(Math.round(wCals), 90+(map(wCals, 0, cals, 0, width-180))/2, height-50);
fill(85, 177, 85);
```

```
  rect(90+map(wCals, 0, cals, 0, width-180), height-90, map(rCals, 0, cals, 0, width-180), 20);
  if (rCals > 0) text(Math.round(rCals), 90+(map(wCals, 0, cals, 0, width-180))+(map(rCals, 0,
cals, 0, width-180))/2, height-50);
  fill(233, 71, 123);
  rect(90+map(wCals, 0, cals, 0, width-180)+map(rCals, 0, cals, 0, width-180), height-90,
map(cCals, 0, cals, 0, width-180), 20);
  if (cCals > 0) text(Math.round(cCals), 90+(map(wCals, 0, cals, 0, width-180))+(map(rCals, 0,
cals, 0, width-180))+(map(cCals, 0, cals, 0, width-180))/2, height-50);
}

//-----------------------------------------------------------------------------------------//
//-----------------------------------------------------------------------------------------//
//-----------------------------------------------------------------------------------------//
void loadEntries(String input, Float j)
{
  HashMap<Float, Float []> data = new HashMap<Float, Float []>();
  float speed;
  String [] entries = loadStrings(input);
  for (int i = 2; i < entries.length; i++)
  {
    String [] tparse = split(entries[i], ","); // Split row i of the imported .csv into columns, by
the separator ","
    Float s = parseFloat(tparse[1]);
    s = new Float ((Math.round(s*10)));
    s = s/10;
    Float nrg = parseFloat(tparse[12]);
    Float ent = parseFloat(tparse[18]);
    Float lat = parseFloat(tparse[13]);
    Float lon = parseFloat(tparse[14]);
    PVector loc = new PVector(lat, lon);

    //Create float array with segment parameters
    Float [] all = new Float[5];
    all[0] = lat;
    all[1] = lon;
    if (latmax == null || lat >= latmax) latmax = lat;
    if (latmin == null || lat < latmin) latmin = lat;
    if (lonmax == null || lon >= lonmax) lonmax = lon;
    if (lonmin == null || lon < lonmin) lonmin = lon;

    // Calculate speed
    if (lastR != null)
    {
      // Calculate change in time
      float t = s - lastS;

      // Declare variables
      float lat1 = lastR.x;
      float lat2 = lat;
      float lon1 = lastR.y;
      float lon2 = lon;
      double R = Math.toRadians(6371);

      // Calculate great circle distance (Haversine formula)
      // Haversine formula - R. W. Sinnott, "Virtues of the Haversine", Sky and Telescope, vol
68, no 2, 1984
  double dLat = Math.toRadians(lat2 - lat1);
  double dLon = Math.toRadians(lon2 - lon1);
  double l1 = Math.toRadians(lat1);
  double l2 = Math.toRadians(lat2);
```

```
   //   double a = ((Math.sin(dLat/2) * Math.sin(dLat/2)) + (Math.sin(dLon/2) * Math.sin(dLon/
2) * Math.cos(l1) * Math.cos(l2)));
   //   double c = (2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a)));
   //   double out = R * c;
   //   float d = (float)(Math.toDegrees(out)*1000);
   //   all[4] = d;

      // Calculate distance (simple, flat earth method)
      double c = dLat^2 + (cos(0.5*(l1+l2)) * dLon)^2
      double out = sqrt((R^2) * c)
      float d = (float)(Math.toDegrees(out)*1000);
      all[4] = d;

      speed = d / t;
      all[2] = speed;
      if (speed > 0 && nrg < 2 && ent > 6) all[3] = 1.0; // Driving
      else if (speed > 0 && nrg < 10 && ent > 6) all[3] = 2.0; // Walking
      else if (speed > 0 && nrg > 10 && ent > 6) all[3] = 3.0; // Running
      else if (speed > 0 && ent < 6) all[3] = 4.0; // Cycling
      else all[3] = 0.0; //Stationary
    }

    if (i == entries.length-1)
    {
      if (smax == null || s > smax) smax = s;
    }

    data.put(s, all);

    lastS = s;
    lastR = new PVector(lat, lon);
  }
  master.put(j, data);
}

//-----------------------------------------------------------------------------------------//
//-----------------------------------------------------------------------------------------//
//-----------------------------------------------------------------------------------------//
void drawTrip (HashMap<Float, Float []> drawing)
{
  if (drawing.containsKey(time/10))
  {
    values = drawing.get(time/10);
    Float x = map(values[0], latmin, latmax, 0+50, width-50);
    Float y = map(values[1], lonmin, lonmax, 0+80, height-220);
    posn = new PVector(x, y);
    noStroke();

    if (lastP != null)
    {
      Float spd = values[2];
      Float mode = values[3];
      if (mode != null && mode == 1.0) // Driving
      {
        dDist = dDist + values[4];
        dTime = dTime + (values[4]/spd);
        stroke(245, 180, 60);
      }
```

```
    else if (mode != null && mode == 2.0) // Walking
    {
      wDist = wDist + values[4];
      wTime = wTime + (values[4]/spd);
      float calHr = map(spd, 0.7, 2, walkSlow, walkFast);
      float calThis = (values[4]/spd) * (calHr/3600);
      wCals = wCals + calThis;
      stroke(43, 107, 145);
    }

    else if (mode != null && mode == 3.0) // Running
    {
      rDist = rDist + values[4];
      rTime = rTime + (values[4]/spd);
      float calHr = map(spd, 2, 4, runSlow, runFast);
      float calThis = (values[4]/spd) * (calHr/3600);
      rCals = rCals + calThis;
      stroke(85, 177, 85);
    }

    else if (mode != null && mode == 4.0) // Cycling
    {
      cDist = cDist + values[4];
      cTime = cTime + (values[4]/spd);
      float calHr = map(spd, 4, 8, cycleSlow, cycleFast);
      float calThis = (values[4]/spd) * (calHr/3600);
      cCals = cCals + calThis;
      stroke(233, 71, 123);
    }
    else noStroke();
    cals = wCals+rCals+cCals;
   }
   lastP = new PVector(x, y);
  }
}

void calSetup()
{
  // Walking
  walkSlow = map(kg, 58, 90, 140, 230);
  walkFast = map(kg, 58, 90, 470, 750);
  // Running
  runSlow = map(kg, 58, 90, 500, 745);
  runFast = map(kg, 58, 90, 1000, 1670);
  // Cycling
  cycleSlow = map(kg, 58, 90, 230, 370);
  cycleFast = map(kg, 58, 90, 700, 1100);
}
```

Long-term viewer

```
Float s, e, lon, lat, time, cals, lastS, smax, latmax, latmin, lonmax, lonmin;
Float dDist, wDist, rDist, cDist, dTime, wTime, rTime, cTime;
Float wCals, rCals, cCals;
Float walkSlow, walkFast, runSlow, runFast, cycleSlow, cycleFast;
Float [] values;
PFont fontSmall, fontBig;
HashMap<Float, HashMap<Float, Float []>> master = new HashMap<Float,
HashMap<Float, Float []>>();
```

```
PVector posn, l, lastR, lastP;
int wmeas;
//--------USER INPUT-------//
int trips = 8;            //        <-- INPUT NUMBER OF SEPARATE TRIPS [trip1.csv, trip2.csv,
trip3.csv...]
int kg = 70;                        //    <-- INPUT USER WEIGHT [for calorie calculations]
//------------------------//

//-----------------------------------------------------------------------------------------------//
//-----------------------------------------setup-------------------------------------------------//
//-----------------------------------------------------------------------------------------------//
void setup()
{
  size(400, 600);
  background(255);
  fill(0);
  strokeWeight(2);
  fill(220);
  noLoop();

  textAlign(CENTER);
  fontSmall = loadFont("Avenir-Book-13.vlw");
  fontBig = loadFont("AvenirNext-DemiBold-18.vlw");
  textFont(fontSmall, 12);
  textSize(12);
  fill(180);
  text("Distance by activity", width/2, height-162);
  text("Calories by activity", width/2, height-82);

  dDist = 0.0;
  wDist = 0.0;
  rDist = 0.0;
  cDist = 0.0;
  dTime = 0.0;
  wTime = 0.0;
  rTime = 0.0;
  cTime = 0.0;
  time = 0.0;
  wCals = 0.0;
  rCals = 0.0;
  cCals = 0.0;
  cals = 0.0;
  lastP = null;
  wmeas = height/12;

  for (int i = 1; i <= trips; i++)
  {
    String input = "trip" + i + ".csv";
    Float j = new Float (i);
    loadEntries(input, j);
  }

  calSetup();

  int keywidth = (width-40)/5;
  int keyspace = keywidth/3;

  noStroke();
  textSize(12);
```

```
  fill(0);
  text("27/07/12", width/2-30, 20);
  text("-", width/2, 20);
  text("29/07/12", width/2+30, 20);

  fill(180);
  text("Driving", 20+keywidth/2, 40);
  text("Walking", 20+keywidth+keyspace+keywidth/2, 40);
  text("Running", 20+2*keywidth+2*keyspace+keywidth/2, 40);
  text("Cycling", 20+3*keywidth+3*keyspace+keywidth/2, 40);

  fill(245, 180, 60);
  rect(20, 45, keywidth, 10);
  fill(43, 107, 145);
  rect(20+keywidth+keyspace, 45, keywidth, 10);
  fill(85, 177, 85);
  rect(20+2*keywidth+2*keyspace, 45, keywidth, 10);
  fill(233, 71, 123);
  rect(20+3*keywidth+3*keyspace, 45, keywidth, 10);

  stroke(200);
  strokeWeight(1);
  line(30, 70, width-30, 70);
  line(30, height-210, width-30, height-210);
  line(30, 70, 30, height-210);
  line(width-30, 70, width-30, height-210);
}

//-------------------------------------------------------------------------------------------//
//-----------------------------------------draw----------------------------------------------//
//-------------------------------------------------------------------------------------------//
void draw()
{
  smooth();

  for (int i = 1; i <= trips; i++)
  {
    Float j = new Float (i);
    HashMap<Float, Float []> data = master.get(j);
    while (time/10 <= smax)
    {
      drawTrip(data);
      time++;
    }
    time = 0.0;
    lastP = null;
  }

  textFont(fontBig, 18);
  textSize(18);
  fill(0);
  float tKM = Math.round(dDist + wDist + rDist + cDist)/10;
  if (tKM/100 < 10) text(String.format("%.2f", tKM/100), width/2-60, height-182);
  else text(String.format("%.2f", tKM/100), width/2-67, height-182);
  fill(180);
  text("km", width/2-23, height-182);

  float mDist = dDist;
  if (wDist > mDist) mDist = wDist;
  else if (rDist > mDist) mDist = rDist;
```

```
  else if (cDist > mDist) mDist = cDist;

  float mCals = wCals;
  if (rCals > mCals) mCals = rCals;
  else if (cCals > mCals) mCals = cCals;

  noSmooth();
  fill(245, 180, 60);
  stroke(245, 180, 60);
  rect(20, height-155, width-map(dDist, mDist, 0, 40, width), 10);

  fill(43, 107, 145);
  stroke(43, 107, 145);
  rect(20, height-140, width-map(wDist, mDist, 0, 40, width), 10);
  rect(20, height-75, width-map(wCals, mCals, 0, 40, width), 10);

  fill(85, 177, 85);
  stroke(85, 177, 85);
  rect(20, height-125, width-map(rDist, mDist, 0, 160, width), 10);
  rect(20, height-60, width-map(rCals, mCals, 0, 40, width), 10);

  fill(233, 71, 123);
  stroke(233, 71, 123);
  rect(20, height-110, width-map(cDist, mDist, 0, 40, width), 10);
  rect(20, height-45, width-map(cCals, mCals, 0, 40, width), 10);

  textFont(fontBig, 18);
  textSize(18);
  fill(0);
  text(Math.round(cals), width/2+33, height-182);
  fill(180);
  text("cals", width/2+67, height-182);
}

//-------------------------------------------------------------------------------------------//
//----------------------------------------loadEntries----------------------------------------//
//-------------------------------------------------------------------------------------------//
        void loadEntries(String input, Float j)
{
  HashMap<Float, Float []> data = new HashMap<Float, Float []>();
  float speed;
  String [] entries = loadStrings(input);
  for (int i = 2; i < entries.length; i++)
  {
    String [] tparse = split(entries[i], ","); // Split row i of the imported .csv into columns, by
the separator ","
    Float s = parseFloat(tparse[1]);
    s = new Float ((Math.round(s*10)));
    s = s/10;
    Float nrg = parseFloat(tparse[12]);
    Float ent = parseFloat(tparse[18]);
    Float lat = parseFloat(tparse[13]);
    Float lon = parseFloat(tparse[14]);
    PVector loc = new PVector(lat, lon);

    //Create float array with segment parameters
    Float [] all = new Float[5];
    all[0] = lat;
    all[1] = lon;
    if (latmax == null || lat >= latmax) latmax = lat;
```

```
    if (latmin == null || lat < latmin) latmin = lat;
    if (lonmax == null || lon >= lonmax) lonmax = lon;
    if (lonmin == null || lon < lonmin) lonmin = lon;

    // Colculate speed
    if (lastR != null)
    {
      // Calculate change in time
      float t = s - lastS;

      // Declare variables
      float lat1 = lastR.x;
      float lat2 = lat;
      float lon1 = lastR.y;
      float lon2 = lon;
      double R = Math.toRadians(6371);

      // Calculate great circle distance (Haversine formula)
      // Haversine formula - R. W. Sinnott, "Virtues of the Haversine", Sky and Telescope, vol
68, no 2, 1984
   double dLat = Math.toRadians(lat2 - lat1);
   double dLon = Math.toRadians(lon2 - lon1);
   double l1 = Math.toRadians(lat1);
   double l2 = Math.toRadians(lat2);

    //   double a = ((Math.sin(dLat/2) * Math.sin(dLat/2)) + (Math.sin(dLon/2) * Math.sin(dLon/
2) * Math.cos(l1) * Math.cos(l2)));
    //   double c = (2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a)));
    //   double out = R * c;
    //   float d = (float)(Math.toDegrees(out)*1000);
    //   all[4] = d;

      // Calculate distance (simple, flat earth method)
      double c = dLat^2 + (cos(0.5*(l1+l2)) * dLon)^2
      double out = sqrt((R^2) * c)
      float d = (float)(Math.toDegrees(out)*1000);
      all[4] = d;

      speed = d / t;
      all[2] = speed;
      if (speed > 0 && nrg < 2 && ent > 6) all[3] = 1.0; // Driving
      else if (speed > 0 && nrg < 10 && ent > 6) all[3] = 2.0; // Walking
      else if (speed > 0 && nrg > 10 && ent > 6) all[3] = 3.0; // Running
      else if (speed > 0 && ent < 6) all[3] = 4.0; // Cycling
      else all[3] = 0.0; //Stationary
    }

    if (i == entries.length-1)
    {
      if (smax == null || s > smax) smax = s;
    }

    data.put(s, all);

    lastS = s;
    lastR = new PVector(lat, lon);
  }
  master.put(j, data);
}
```

```
//-------------------------------------------------------------------------------------------//
//-----------------------------------------drawTrip------------------------------------------//
//-------------------------------------------------------------------------------------------//
void drawTrip (HashMap<Float, Float []> drawing)
{
  if (drawing.containsKey(time/10))
  {
    values = drawing.get(time/10);
    Float x = map(values[0], latmin, latmax, 0+50, width-50);
    Float y = map(values[1], lonmin, lonmax, 0+80, height-220);
    posn = new PVector(x, y);
    noStroke();

    if (lastP != null)
    {
      Float spd = values[2];
      Float mode = values[3];
      if (mode != null && mode == 1.0) // Driving
      {
        dDist = dDist + values[4];
        dTime = dTime + (values[4]/spd);
        stroke(245, 180, 60);
      }

      else if (mode != null && mode == 2.0) // Walking
      {
        wDist = wDist + values[4];
        wTime = wTime + (values[4]/spd);
        float calHr = map(spd, 0.7, 2, walkSlow, walkFast);
        float calThis = (values[4]/spd) * (calHr/3600);
        wCals = wCals + calThis;
        stroke(43, 107, 145);
      }

      else if (mode != null && mode == 3.0) // Running
      {
        rDist = rDist + values[4];
        rTime = rTime + (values[4]/spd);
        float calHr = map(spd, 2, 4, runSlow, runFast);
        float calThis = (values[4]/spd) * (calHr/3600);
        rCals = rCals + calThis;
        stroke(85, 177, 85);
      }

      else if (mode != null && mode == 4.0) // Cycling
      {
        cDist = cDist + values[4];
        cTime = cTime + (values[4]/spd);
        float calHr = map(spd, 4, 8, cycleSlow, cycleFast);
        float calThis = (values[4]/spd) * (calHr/3600);
        cCals = cCals + calThis;
        stroke(233, 71, 123);
      }
      else noStroke();
      cals = wCals+rCals+cCals;
    }
    if (lastP == null) point(x, y);
    else line(lastP.x, lastP.y, x, y);
    lastP = new PVector(x, y);
  }
```

```
}

void calSetup()
{
  // Walking
  walkSlow = map(kg, 58, 90, 140, 230);
  walkFast = map(kg, 58, 90, 470, 750);
  // Running
  runSlow = map(kg, 58, 90, 500, 745);
  runFast = map(kg, 58, 90, 1000, 1670);
  // Cycling
  cycleSlow = map(kg, 58, 90, 230, 370);
  cycleFast = map(kg, 58, 90, 700, 1100);
}
```