Heads-Up Display Application for Automotive Testing

RDE VeriTest: A Progressive Web Application (PWA) for Validating the RDE Trip Requirements

Final Year Project
27th May 2021

Tilak.B.Bhavsar[1]
Department of Aeronautical and Automotive Engineering,
Loughborough University, Leicestershire LE11 3TU, UK

The Real Driving Emissions (RDE) test is an on-road emissions testing regime, recently legislated by the EU [1] in response to an emissions discrepancy of up to 45% [2] between laboratory-based type-approval and real driving. The regime introduces a complex set of 33 trip requirements (Boundary Conditions, BCs) which must be met for a test to be valid, to ensure tests are both representative of the range of real driving dynamics and well standardised. The financial costs of test failure can be high [3], as well as the associated time penalty. Considering the introduction of the Tactile Internet and 5G technology [3], the aim of this project is to develop a proof-of-concept smartphone application (RDE VeriTest) to assist a single driver in completing an RDE test. BCs should be verified autonomously and key information conveyed clearly to the user; helping them navigate through the requirements successfully without excessive distraction; risks of which include the potential for road traffic collisions. Other attempts to produce such an application have lacked a clear and concise User Interface (UI) and are native mobile apps, which this project hopes to improve upon. Thus far, a multi-platform Progressive Web Application (PWA) which is capable of tracking user position within an average accuracy of 4.6m has been coded and is available to access and download at https://rde-vt.netlify.app. The app can currently correctly verify 11 BCs and provides UI feedback including visual (text and colour based) and audio (sound alert) notification of BC statuses. The app is coded in the React library and has been successfully drive tested a total of 13 times on local routes. Overall, the project aims and objectives have been achieved in that a robust RDE test assistance application has been developed.

[1] Aeronautical Engineering MEng; B721275; Email: t.bhavsar-17@student.lboro.ac.uk
Project supervisor: Dr Byron Mason

1. Introduction

1.1. Emissions Testing

Vehicle testing is critical to ensuring the maintenance and consistency of high quality, safety, and performance standards in the automotive industry. Emissions testing is essential to monitor the pollutant levels and environmental 'efficiency' of new vehicles to guarantee compliance with legislation. Since the release of the 2018 IPCC Special Report [4] on global warming, the global focus on emissions reduction has heightened. The report highlighted the potentially catastrophic and irreversible impacts of further atmospheric CO_2 emission on the environment and ecology with a global temperature rise of 1.5°C, increasing the necessity of international efforts such as the Paris Climate Accords [5]. Similarly, considerable effort in lowering Nitrogen Oxide (NO_x), Carbon Monoxide (CO), Total Hydrocarbon (THC), and particulate emissions has been made, primarily due to health concerns relating to local air quality. Simultaneously, the capabilities of assistive human-vehicle interface technologies such as Advanced Driver Assistance Systems (ADAS) have advanced significantly [6], to the point where they can be manipulated to enable self-driving [7].

Naturally, automotive emissions testing regimes have come under increasing scrutiny, having been increasingly regulated since the 1970s. According to the EU, transport is estimated to produce 30% of European CO_2 emissions, 72% of which are caused by road vehicles [8]. Initially, CO_2 emission was regulated on a voluntary basis, which proved ineffective after failing to meet EU targets. This led to the successful implementation in 2009 of a limit of 130 g CO_2/km, and a mass-specific 'premium' cost levied on manufacturers for excess emissions [9], and the target was achieved fleet-wide by 2013 in most EU countries. The creation of the 'Euro 1' standard in 1992 introduced harsher regulation for NO_x, THC, CO, and particulate emissions [10], progression of which culminated in the 2014 'Euro 6' rules - mandating limits up to 20 times more stringent than those of 1992 [11]. The EU data [12] presented in Fig. 1 shows the tightening of NOx limits from Euro 3 to 6, however as can be seen the measured impact does not strongly reflect these changes, and there is a lag between implementation and results – indicating flaws in testing or measurement methodology. The 'Euro' standards primarily utilised the New European Driving Cycle (NEDC) laboratory testing model, where vehicles are statically 'driven' through a series of repetitive cycles while emissions are recorded. Following Euro 5, further evidence of a results disparity (see Section 2.1) between the NEDC and road-based "real" emissions led to the implementation of an on-road testing regime (Real Driving Emissions, or RDE). The scheme requires the use of a Portable Emissions Measurement System (PEMS) to be installed in the vehicle for emissions tracking during the test which takes place on public roads. In 2017, the NEDC was replaced with its successor, the World Harmonised Light vehicles Test Cycles (WLTC), which also better reflects real driving conditions. The RDE regime is complex, requiring the accurate validation of many trip characteristics, which act as 'boundary conditions' (BCs) to ensure test consistency. An RDE test is considered invalid if BCs are not achieved, regardless of whether the vehicle has passes or fails the emission limits which are the real test focus - thus the potential benefit of verification assistance is high.

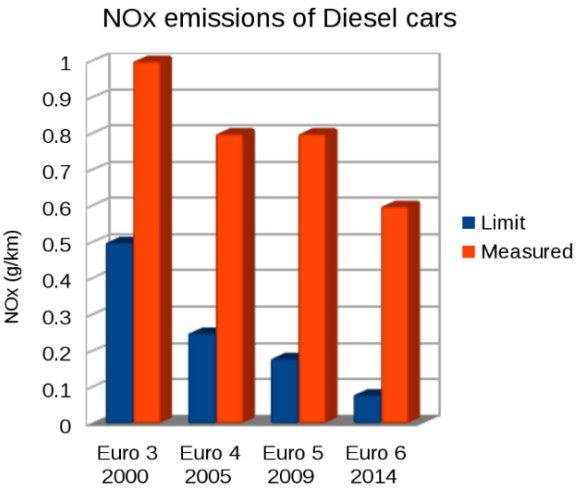

Fig. 1 Comparison of real EU NOx emissions and corresponding legislation [12]

1.2. Project Aims

The aim of this project is to develop a proof-of-concept smartphone application (app) to provide digital assistance in performing the RDE test, in order to reduce the number of invalid tests and their associated time and cost penalty. This assistance will be provided by validating specific BCs automatically, with minimal input from the driver. The output of test validation will then be communicated to the driver. The app should also be simple and non-distracting, such that it can be used safely and with ease by a single driver with little knowledge of the detailed RDE requirements. This report introduces the background to the study and focuses on the technical development of the app, including BC verification, testing, the Graphical User Interface (UI), and finally the project results and conclusions.

1.3. Project Objectives

Listed below are the objectives of the project.

I. Produce and consolidate a basic UI/UX structure into a simple smartphone app
II. Create a data flow structure to collate and process live measured data
III. Code a background verification system to indicate RDE BC validity
IV. Indicate test failure or success (in terms of measured BCs) to the driver

1.4. Project Scope

The project will validate those boundary conditions which can be measured and calculated without the use of a PEMS since emissions verification (see Fig. 8, Section 3.2.2) is considered unnecessary in proving the concept of test validation assistance via an app.

2. Background

2.1. Need for Real Driving Emissions Testing

Numerous studies within the last decade ([13][14][15][16][17][18][19]) have evidenced the existence of a growing emissions discrepancy between laboratory drive cycle testing (type-approval such as the NEDC and WTLC regimes) and on-road measurements. Several automotive sources are tracked in Fig. 2 which also evidence an increase in the divergence of CO_2 emission over time, the highest estimates reaching values of over 45%, and an average of 38%, in 2013. UK study supports this, suggesting a 45% gap between road-recorded and type-approval levels. [17] The issue identified is twofold. Firstly, a lack of parity of repetitive, controlled lab testing cycles to the physical conditions of real driving is seen; and secondly, the heavy standardisation of the test is found to allow exploitation through the use of so-called ‘defeat devices’. A 2016 Dutch study by Hejine et al. [18] showed that of 15 tested vehicles, recorded on-road NOx levels were up to 16 times higher than type approval emissions limits (Euro 6).

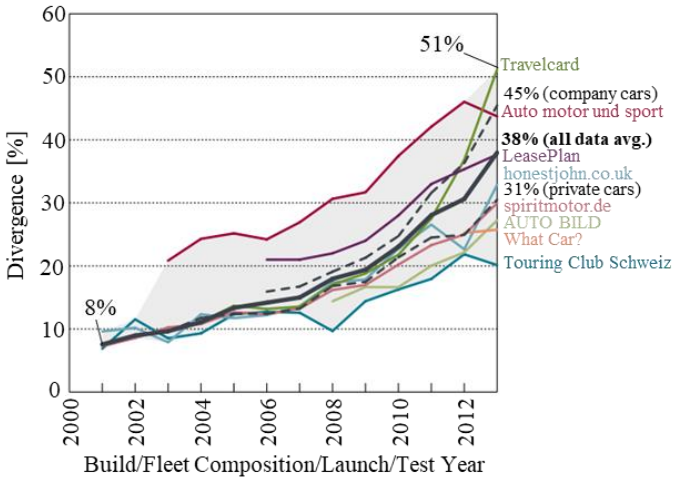

Fig. 2 Real Emissions-test CO_2 Divergence over time [16]

2.1.1. Lack of Realism

Rigid standardisation is required to maintain reproducibility and parity of test results over a wide range of vehicle models, as varying standards with vehicle models could render emissions results incomparable. The NEDC standard was used exclusively between the 1980s and 2017, having had its last significant update in 1997 [20]. An NEDC test comprises of five segments characterised primarily by velocity; the test vehicle is mounted onto a static chassis dynamometer and run through four repetitions of the Urban Driving Cycle (UDC) (left on Fig. 3, data plotted from [21]), and a single Extra-Urban Driving Cycle (EUDC) (centre). In comparison, the newer WLTC consists of 4 'phases', from 'Low' speed to 'Extra High'; a speed trace of the 'Moderate' phase is presented on the right of Fig. 3, with data from [22].

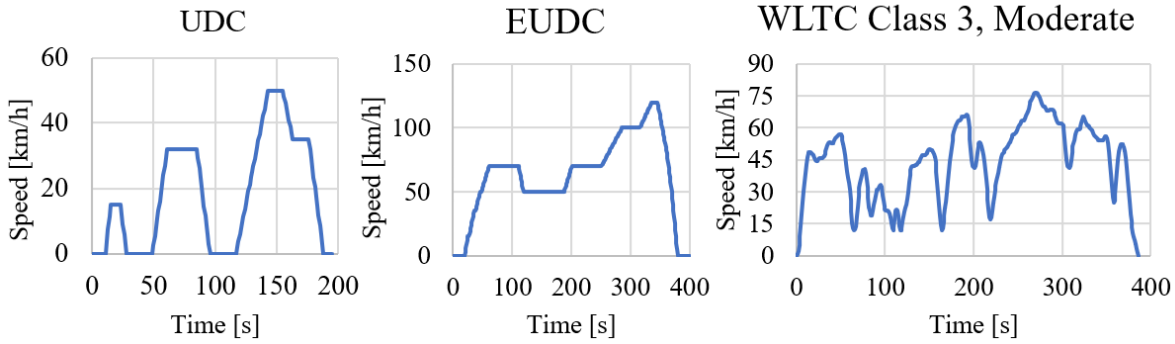

Fig. 3 Speed-Time Trace of UDC, EUDC and 'Moderate' Phase of WLTC Class 3 Drive Cycles [21] [22]

Real driving conditions are complex, involving aerodynamic drag, altitude change, varying driving styles and rolling resistance and constantly changing ambient conditions [23]. In contrast, NEDC conditions are strictly standardised and controlled, which could be seen to supply insufficient vehicle dynamics and operating points to represent real conditions [24][25]. As shown in Fig. 3, acceleration limits are fixed and non-smooth, and while a broad speed range is covered, there are sustained periods of constant speed. In addition, an ambient temperature range of just 10°C is imposed, which is also unrealistic when considering temperature variance between locations on a given drive can exceed this [26]. As Fig. 3 indicates, the WLTC contains higher speed, 'distance' and acceleration variation, and is more representative of the highly dynamic nature of real driving. Despite this, investigations such as that of Sileghem et al. [27] have shown inadequacies in the ability of the WLTC to cover certain speed and acceleration dynamics.

2.1.2. Exploitation of Standardisation

While the new WTLC regime is considered more realistic than the NEDC, it still carries the significant issue of being highly standardised [28], thus exposing itself to the potential for further emissions fraud in future. The Volkswagen Emissions Scandal of 2015 is an example of the 'cycle beating' phenomenon, wherein the manufacturer's "clean diesel" vehicles were found to violate U.S. emissions law. The diesel engines were programmed to activate mandatory emissions controls only during lab testing, releasing up to 35

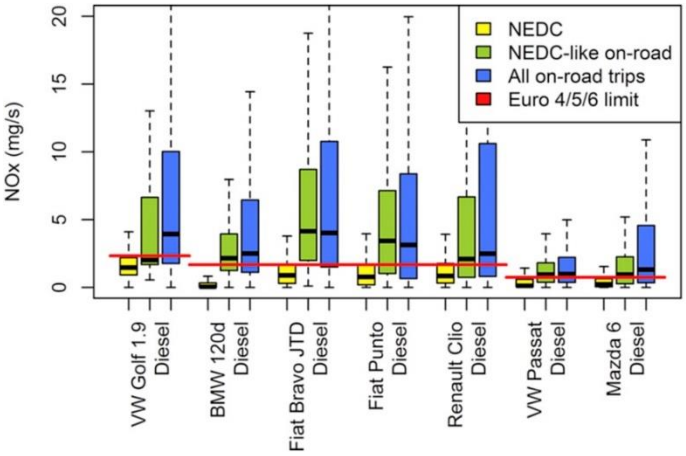

Fig. 4 Discrepancy between on-road 'NEDC-like' and NEDC laboratory NOx emissions for diesel vehicles [26]

times more NOx during real driving through an intentional software loophole (‘defeat device’) to falsify emissions [29]. This malpractice is not limited to Volkswagen and is considered by some to be widespread in the industry [14][26][30]. Weiss and Degraeuwe [26] argue that the use of defeat devices is the main driver of NOx emission discrepancy rather than the inadequacy of type-approval processes. The study tested 10 gasoline and diesel vehicles on the NEDC and on-road, finding that even when NEDC conditions were replicated on the road, resulting data (green box plots in Fig. 4) showed “strikingly different behaviour”, challenging the widely-held view that the NEDC itself is the main cause of emission difference [26]. However, other experiments such as the 2016 UK Department for Transport study found no evidence that vehicles other than those produced by the Volkswagen Group had cycle beating devices installed. The study instead found that track-based NEDC imitation (on-road driving) tests produced on average 4.5-5 times higher NOx emissions than lab-based NEDC tests [17], suggesting the inadequacies of the test are to blame rather than defeat devices. This conclusion is likely more accurate than the proposition of Weiss and Degraeuwe, due to their investigation not being specifically tailored to “identify legally relevant cycle beating” [26] and lacking evidence which solely identifies defeat devices as the cause of emissions discrepancy. Aside from direct industrial malpractice in emissions disclosure, manufacturers are also able to augment vehicle emission performance at the specific operating conditions which standard lab test regimes focus on (such as certain speeds), rather than for the full spectrum of driving conditions; providing a significant source of error [25][27].

2.2. Human-Vehicle Interfacing Technology and Smartphone Applications

The European Union seeks to rectify the stated issues through the introduction of the RDE test regime, while the WLTP is maintained as a baseline and to enable a reproducible comparison of fuel consumption [28]. While the need for RDE testing has been established, practical methods to help implement the procedure EU-wide are lacking. Studies such as Doshi et al. [31] have described successful attempts to produce Head-Up-Displays (HUDs) such as the laser-based Dynamic Active Display (DAD), the purpose being to provide useful information to the driver without distracting them excessively from the road. DAD achieves this using a laser-projected alert dashboard, which reduced time spent over a speed limit by 38% on average [31], which is a valuable concept for RDE BC verification, which relies on vehicle speed. While the readiness of this technology is still low, with the number of smartphone users in the world exceeding 1 Billion [32], the use of mobile devices and similar widespread hardware to solve the emissions and environmental challenges identified could create a large global impact far exceeding the amount of effort and cost involved. This is due to the simplicity and abundance of software tools and the innovation of new, lightweight and cross-platform applications such as Progressive Web Apps (PWAs) [33]. These enable the rapid development of apps capable of actively interfacing with the user in a similar way to HUDs when coupled with the Agile coding methodology [34][35][36]. When considering the effects of innovation, such as the currently accelerating introduction of the tactile internet through low-latency 5th Generation (5G) systems [3], higher interconnectivity will likely characterise the automotive sector in the near future. This is also evidenced by developments such as the world’s first dual-frequency smartphone (Xiaomi mi 8) released in 2018 [37], with ultra-high accuracy GPS being available at very low cost.

2.3. Real Driving Emissions Legislation

The inherent and unique advantage of driving on open, public roads is the exposure to the changing temperatures, pressures, roads and other conditions which characterise real driving. The RDE legislation is split into ‘packages’ which gradually introduced the new provisions,

giving automotive manufacturers time to accommodate the changes. The earliest RDE implementation was Euro 5b for Heavy-Duty Vehicles (HDV) released in 2011. Following this, Euro 6a,b,c and 6d-temp form the four main introductory RDE packages (2016-19) [23]. The 'Euro 7' legislation is expected to present harsher regulation as part of the EU Green Deal in late 2021, for implementation in 2025 [38]. As mandated by the latest package (Euro 6d, January 2021), the full trip requirements (BCs) for a valid test are elaborated in Table 2 (Section 3.2). Basic data types covered are time, speed, distance, and elevation (altitude). These comprehensive requirements are essential for two reasons: to maintain consistency between RDE tests, and to ensure that tests represent the dynamic conditions of 'real' driving as thoroughly as possible. Thus, BCs cannot be so restrictive that they constrain results to only certain aspects of driving, and yet must be defined enough to not over complicate emissions evaluation methods [39].

2.4. Test Validation Assistance Requirement

The complexity of the RDE test regime creates challenges in effective emissions verification, and it is proposed that this necessitates the use of an external device capable of assisting in this regard. Three major challenges are highlighted and explored alongside the proposed solution in the following sections.

2.4.1. Lack of Knowledge of Test Requirements

Table 2 lists 33 requirements for RDE validity. While these can be understood upon further inspection by those who are familiar with the RDE legislation, others may find the BCs difficult to decipher. This lack of knowledge of the testing regime due to its precision and complexity becomes a more significant issue when the RDE test (or its equivalents) become widespread globally; a rollout which is already planned to take place in countries such as India and Japan through UNECE Global Technical Regulation (GTR) [23]. It cannot therefore be expected that knowledge of the test requirements can be disseminated effectively to all parties who require testing (including OEMs) through legislation alone, despite this being essential to legal compliance. Furthermore, there is some evidence [23][24][40] that such regimes will increase in complexity, such as the 2019 study by Suarez-Bertoa et al which showed up to 8 times higher CO emissions with a more dynamic version of RDE [41], suggesting RDE-4 has a range of improvements to be made to fully represent the spectrum of real conditions. This, along with the high cost of failed tests (see Section 2.4.3) reinforces the current and future requirement for validation assistance to enable test engineers to conduct tests as per legal specification.

2.4.2. Difficulty of Data Monitoring and Interpretation

As shown in Table 2, less than 1% of recorded data points can breach BCs for the test to be considered successful. This criterion not being met results in immediate test invalidation, requiring another test to be taken [1]. The majority of BCs are difficult or impossible for any amount of test drivers to record and analyse simultaneously without the use of external devices (as described in Section 3.2). For example, the calculation of distance and speed shares or Real Positive Acceleration (RPA) as mandated by the legislation [1] requires the all of the recorded data points to be assessed, which can number in the tens of thousands (when calculated based on test time and data sampling requirements).

2.4.3. High Rate of Invalid Tests

Currently, the only official method to ascertain test success is to conduct one or more tests, and then run the resulting PEMS emission and trip data through a program such as EMROAD (the EU Joint Research Committee mandated tool for full test validation) post-test. This procedure is inefficient as it provides no indication of test validity (even of those trip

requirements which do not require PEMS emissions data to validate) during the test. Each test is required to take between 90-120 minutes [1]; thus a significant amount of time is taken in retaking tests upon failure. A 2020 study by the EU JRC found that while more tests are valid with RDE-4 than the RDE-3 package, a failure rate of up to 41% still exists [23]. Horiba estimates the failure rate as between 30-50%, and the yearly cost of failures as potentially exceeding €1 million [42]. A more effective system would alert the driver when boundary conditions are invalidated, allowing the unusable test to be cancelled and a repeat to be conducted immediately. The most effective system would prevent the driver from invalidating conditions altogether by alerting them whenever critical conditions are neared, allowing prompt action to be taken to preserve validation. For example, if vehicle speed nears the requirement limit, an audio or visual stimulus could alert the driver to reduce speed. For such a system to be fully accurate, real-time integration with the PEMS unit is necessary to validate the trip requirements which relate to vehicle emissions data calculations (see Section 2.3) Despite this, even a simple test assistance platform using basic trip data inputs (such as speed, time and altitude) could significantly reduce the time spent retaking tests, where tests with a high likelihood of success are considered ‘valid’. This also reduces the unnecessary emissions caused by taking invalid tests – which supports the initial aim of emissions legislation.

2.4.4. Previous and Proposed Solutions

Several ongoing commercial efforts such as HORIBA CoDriver [42] and AVL Smart Mobile Solutions™ [43][44] and academic [45] attempts at RDE validation app development have been made; all aiming to solve the challenges discussed. The shortfall of these attempts is a requirement for a secondary ‘navigator’ alongside the driver to interpret the complex trip verification data, which is often displayed in full on the main app screen during the test. The high fidelity of the data display (as demonstrated in Fig. 5) can be distracting to drivers and thus is unsafe to use for a single tester. In addition, the data is hard to interpret for users with minimal knowledge of the BCs. Therefore, this project proposes the creation of an app with a simplified user interface which reduces the ‘tester’ workload to the point where verification is possible with a single driver.

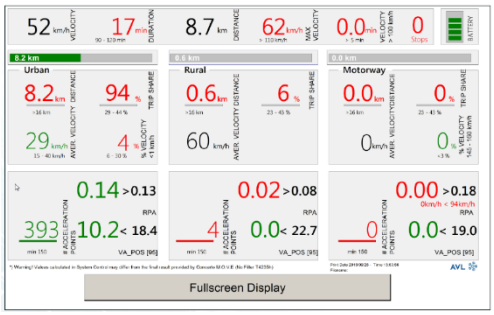

Fig. 5 AVL Move System Control App [43]

3. Technical Progress

3.1. User Requirements and Competitor Analysis

Requirements capture is vital in the context of app design, as the overall functionality, display interface and user experience of the app are governed by the needs of the end-user. Eliciting user requirements first is key to enable the measurement of the success of the final application [36]. Potential user profiles are elaborated below in Table 1 and consider a broad range of potential ‘customers’ should the app be produced in a commercial context. This approach ensures that all facets of required functionality are considered.

From these requirements, key app considerations are derived which add detail on how the goals of the project should be delivered through customising the User Interface (UI) and User Experience (UX) choices. Firstly, app simplicity is considered important to reduce the time taken to address BCs or trip status: this assists time-pressured OEMs and test engineers to efficiently complete tests. Simplicity can be implemented in the form of a visually clear (clutter-free) UI, as well as a reduced number of steps to reaching in-app goals [32]. Goals

are tasks which the user requests by interacting with the UI. An example of an inefficient UI approach in this context is the use of one which requires several buttons to be pressed to initiate the RDE test, and a lack of a menu bar requiring the 'back' button to be pressed several times to lead to key screens.

Table 1 Anticipated User Requirements Based on User Occupation and Knowledge Level

Potential App User	Key Anticipated Requirements
Test engineer employed at a private testing company	• App does not require extensive knowledge of RDE requirements • Guides user through the verification process • Multi-platform application for ease of access • Ability to learn from results or failure modes to reduce number of tests
Test engineer employed by EU public service firms	• Large amounts of complex test data displayed efficiently (post-test) • Highly reliable data processing
Automotive manufacturer (OEM) employee	• High validation reliability • Time reduction of RDE test initiation (quick and efficient process) • Indication of exact failure point (for vehicle development)
Amateur, hobbyist, non-commercial individual	• App requires no knowledge of BCs, verification occurs in background • Simple, easy to use and clear interface (minimalistic) • Safe and secure for use while driving (low driver input or interaction)

Conversely, a well-structured app would use a menu bar or similar strategy to allow the user to quickly switch view between a number of key screens. Equally, the display of key data in a fixed location is beneficial to reduce the user input required to a minimum. However, an overloading of data and the use of vibrant colours and small font sizes is highly detrimental (especially for numerical data which must be interpreted), and this is a common feature of previous commercial app efforts which could be said to be in the early stages of development and lacking improved interfaces. An example of this is the AVL Smart Mobile Solutions™ app RDE screen shown in Fig. 6.

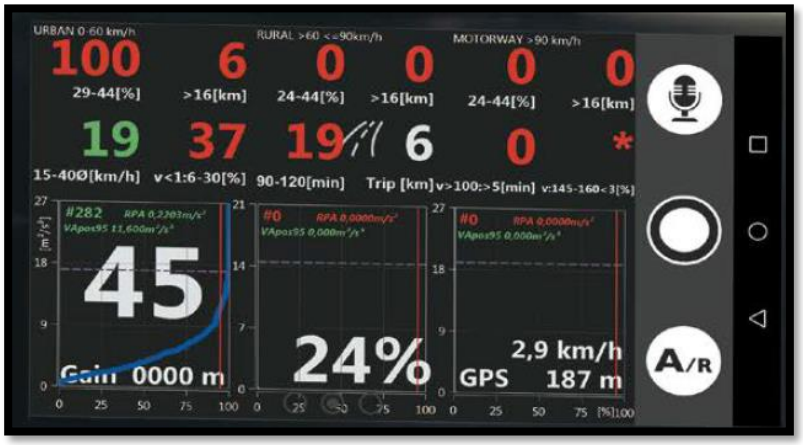

Fig. 6 AVL Smart Mobile Solutions™ RDE Test Data Screen View [44]

Although an effort has been made to increase the sizes of key data points (for example test share percentage), overall, the screen is crowded and graphical information is difficult to see. Reducing this level of data fidelity by only presenting graphical information after the test is completed, or not at all, is considered more appropriate to prevent driver distraction and increase safety. The driver should be able to obtain the key information by glancing at the screen rather than having to decipher useful information from raw data. The AVL app also does not provide audio alerts to the user, which would prove useful in avoiding any visual interruptions from the road and surroundings. The use of red coloured numerals which are constantly present on the screen until BCs are fulfilled could also be considered distracting

and unnecessary – red visual alerts and audio signals could be used in place of this so as to only attract attention when action is required from the driver to correct issues (such as when a BC is close to being invalidated). In contrast, the HORIBA RDE CoDriver app (presented in Fig. 7) is superior in visual clarity, using white boxes to represent unfulfilled BCs and green highlights to show achievement. An easy access menu leading to all app screens is used as an effective method to present the user with valuable test information, while not presenting excessive raw data. Nonetheless, the variety of data points on the main 'Trip Requirements' screen could be simplified to show only vital information (trip time, urban distance share, and total distance). Another advantage of CoDriver is the ease of access of the 'record' button (acting to start and stop the test), which enables testing to begin and end in a single step.

Fig. 7 RDE CoDriver Trip (Left), Moving Average Window (Centre) and Menu (Right) Screens [29]

Operational speed and reliability are also key factors for a positive user experience. A significant time lag between input and operation alongside any malfunction in data logging is undesirable due to the need for test repeats which undermines the purpose of the app. This requires a robust app back-end (data transfer system) and 'lightweight' code to reduce the computational intensity of processes, as well as a reliable wireless data connection for continuous streaming. Furthermore, it can be concluded from Table 1 that some post-test recording and analysis of data is required to meet the needs of OEMs and for further examination.

3.2. Technical Requirements

The RDE legislation outlines specific requirements (BCs) which form the basis of the core app functionality. These requirements are equally important to the UI an UX considerations outlined in Section 3.1, and relate directly to the RDE Test and subsequent data validation procedures.

3.2.1. RDE Test Procedure

According to EU Legislation 2018/1832 [1], following the installation of a Portable Emissions Measurement System (PEMS) in the vehicle to be tested, a WLTC test is first conducted in order to establish baseline emissions values and check all equipment and the vehicle is functioning as expected. The PEMS is a small unit which is usually mounted at the rear end of the vehicle along with necessary sensors and exhaust redirect pipes used to capture and record live emissions data. Upon the planning of an adequate test route via a topographical map or automated system, the vehicle is then driven on public roads through three segments spread over the test, which lasts between 90-120 Minutes. The segments (Urban, Rural and Motorway) are characterised by the vehicle speed (up to 45, 45-80 and 80-145 km/h respectively), in order to realistically replicate varying modes of normal modern driving. Resulting PEMS data is then analysed after one or more tests have been concluded.

3.2.2. Test Validation

Currently, beside commercial software packages such as the AVL Move System Control Suite [43], an add-in package for Microsoft Excel called 'EMROAD' is officially mandated to verify BCs (and test emissions) directly from the PEMS data. EMROAD is created and maintained by the EU Joint Research Committee (JRC) and follows the validation process set out in the flowchart in Fig. 8 (based on RDE legislation) in order to categorise a test as 'Pass' or 'Fail'. Trip validation is the process and calculations by which a test is legally categorised as meeting RDE requirements as presented in Table 2.

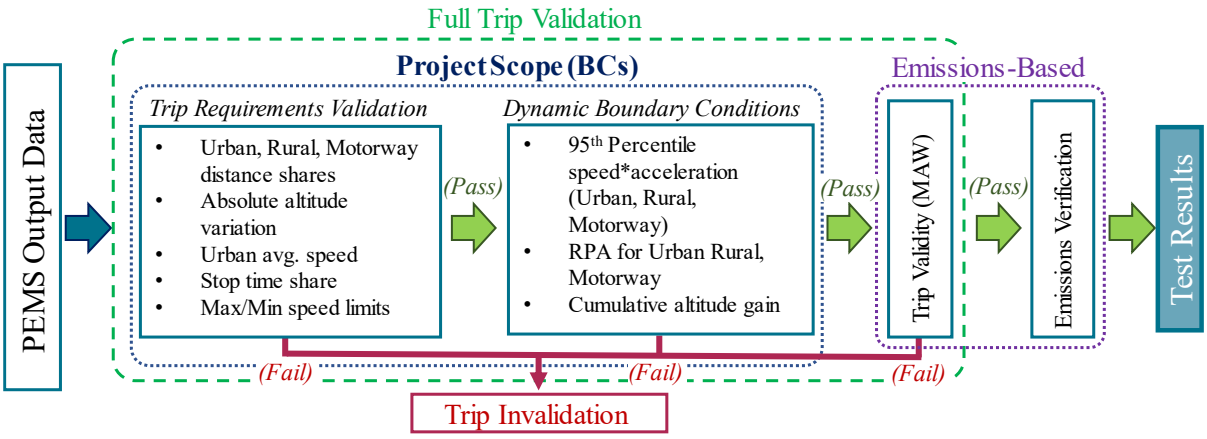

Fig. 8 Full RDE Test Verification and Emissions Calculation Based on EU 2018/1832 (6d) [1]

Table 2 RDE BCs (Trip Requirements) from EU Legislation 2018/1832 (6d) [1]

(U=Urban, R=Rural, M = Motorway)

Requirement	Value(s)	Unit(s)
	Time	
Total Trip Duration	90-120	min
Idling event(s)	None longer than 300 s Allowed	s
Initial Idling Duration	≤ 15	s
Urban Stop Time	6-30	%
	Cold Start	
Cold Start Duration	=5	min
Cold Start average speed	15-40	km/h
Cold Start max. speed	<60	km/h
Cold Start stop time	≤90	s
	Speed	
Average Urban speed	15-40	km/h
Motorway speed above 145 km/h	< 3% motorway time	%
Motorway speed above 100 km/h	≥5	min
	Dynamics	
U/R/M Relative Positive Acceleration	Pass/Fail according to limit curve	m/s^2
U/R/M 95th percentile Speed*Acc	Pass/Fail according to limit curve	m^2/s^3
	Distance	
U/R/M Distance (Share of test)	>16 (29-44, 23-43, 23-43)	km (%)
	Elevation	
Absolute Difference in Elevation from start-end	≤100	m
Total/U cumulative positive elevation gain	<1200	m/100km
	Emissions	
U/R/M Number of total counts	≥100	qty
U/R/M Share of normal windows	>50	%
	General	
Data in 'Extreme' Conditions	<1	%

Validation comprises of three stages: the checking of basic trip requirements including time, speed and distance shares of the three trip segments), checking of dynamic boundary conditions (including acceleration, elaborated in Appendix 7a of the legislation), and finally 'Emissions-Based' trip validity requirements (based on data from the PEMS), are calculated using the Moving Average Window (MAW) method [1]. The MAW method splits the trip data into 'windows', the length of which are based on the cumulative mass of CO_2 emitted as compared to a WLTP test of the same vehicle. Emissions are averaged in each window, and then later assessed against a CO_2 'Characteristic Curve' also based on equivalent WLTP results [39]. Subsequently, the vehicle emissions can be checked against limiting values, determining approval or failure. These final two stages are not included within the app scope as discussed in Section 1.4, therefore the app will only be able to provide a 'likelihood' of test validity and not a legal conclusion, with increasing confidence with the number of BCs verified; and no indication of test 'pass' or 'failure' is given.

3.2.3. Cold Start and Dynamic BCs

In Table 2, The 'Cold Start' is defined as the first 5 minutes after engine ignition and is included due to the high emissions contribution from this engine state [46]. 'Idling events' are characterised by a vehicle speed of 0 km/h, and Relative Positive Acceleration (RPA) is an indicator of a lack of sufficient vehicle dynamics ('soft' driving) [39], verified against set limit curves. Positive in this sense refers to 'forward' motion. RPA also uses the MAW method for verification, as per Eq. (1) from [25] (relevant units are given in square brackets).

$$RPA_k \left[\frac{m}{s^2}, \frac{kW \cdot s}{kg \cdot km}\right] = \frac{\sum_j \left[\Delta t \cdot (v \cdot a_{pos})_{j,k}\right]}{\sum_i d_{i,k}}, \;\; j = 1 \; to \; M_k, i = 1 \; to \; N_k, k = U, R, M \qquad (1)$$

Where:

- k = Urban (U), Rural (R) or Motorway (M) phases
- Δt [s] is the time difference (equal to 1 s as per recommended 1 Hz sampling rate)
- M_k is the selected sample number for U, R and M
- N_k is the total sample number for U, R and M
- a_{pos} $[ms^{-2}]$ is positive acceleration
- d_i [m] is distance travelled in each data point, i

As Eq. (1) shows, the product $(v \cdot a_{pos})$ is calculated for each data point up to the 'current' sample number, multiplied by the sampling time, summated, and divided by the total distanced travelled in the monitoring period, giving an acceleration value in ms^{-2}. This process is then repeated for data corresponding to each trip segment (U, R and M). Furthermore, the limiting of the 95th percentile of the product of actual vehicle speed (v) and any acceleration greater than 0.1 ms^{-1} (shown in Eq. (2)) ensures exclusion of extremely high (unnatural) dynamics from the test [25][39]. The percentile condition asserts that only 5% of data values can exceed the prescribed limits. Thus, $v \cdot a_{pos}$ forms an upper limit BC on the test, and RPA a lower limit.

$$(v \cdot a_{pos})_{k_{[95]}} \left[\frac{m^2}{s^3}, \frac{W}{kg}\right] \qquad (2)$$

3.3. Developmental Methodology

In terms of app development, the 'Agile' methodology as set out in [35] was followed. In this methodology, basic features defined by the user and technical requirements (Sections 3.1 and 3.2) are added in earlier build (coding) sessions called sprints, and feature complexity is increased rapidly in a flexible way with each sprint. This ensures that core functionality is

robust, while issues (which are likely to arise upon addition of complexity) can be debugged and solved more easily as they emerge, in contrast to more rigid, step-based approaches [34]. As shown in Fig. 9 the overall process involved five main stages. Firstly, in Phase 1 a review of the legislative literature was undertaken as preliminary work. Phase 2 formalised the user and technical requirements resulting from this work and investigation of relevant practical considerations (such as learning programming languages and locating data sources within hardware). Phase 3 introduced a developmental 'wireframe' UI app build to demonstrate core functions and initiated testing, and Phase 4 saw refinement of the UI, launching the final release candidate (final application). This report covers all four stages, presenting Phases 3 and 4 in detail.

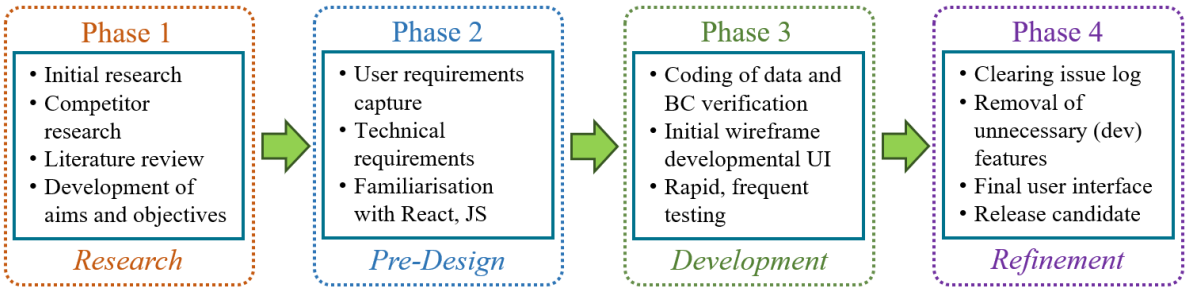

Fig. 9 Project Developmental Stages

3.4. Progressive Web Applications and GPS Location

3.4.1. Native Android and Progressive Web Applications

Initially, the app was planned to be coded as a native Android application in the Java or Kotlin languages, using the Android Studio Integrated Development Environment (IDE). This was based on the high offline stability of native apps and their access to all levels of smartphone functionality (including hardware sensors necessary for test validation). However, early in the project development process the concern of cross-platform usability arose, which was a result of the need to enable rapid testing, deployment, and simultaneous development within the project timeframe, as per the Agile methodology [34]. Native Android app development is strictly segmented between coding and testing, which requires a device emulator or physical connection to a smartphone to edit the app in real time. Furthermore, while the high complexity of the Android framework allows the creation of highly interactive UIs, it was found that these features were of less importance in meeting user requirements (Section 3.1). The use of a Progressive Web App (PWA) was considered a superior choice in satisfying the need for multi-platform usability.

A PWA is the application implementation of common web technologies coded in the JavaScript, HTML and CSS languages. The primary benefit of PWAs is near-universal cross-platform availability. A PWA can be accessed via an internet link by any device supporting a modern internet browser such as Google Chrome, including smartphones (on either iOS and Android platforms), laptops and tablets: and is totally indistinguishable from a native app when downloaded to the home screen. In contrast, a native Android app would not be compatible with Apple iOS phones. In addition, app download sizes can be reduced by factors of over 150 and have significantly faster load times [33] when compared with native apps. HTML is the language for reading (rendering) webpages in browsers, while CSS governs styles (formats), and JavaScript determines the behaviour (interactivity) of sites. 'React' is an open-source JavaScript library created by Facebook [2] for building User Interfaces, allowing for powerful component-based application design by changing the way these languages are implemented, allowing an entire application to be written mostly in JavaScript through an optional syntax extension known as JSX. Crucially for CPU-intensive

BC verification, React enables complex operations to be executed at speed and in a much more efficient (lightweight) way by using virtual processes which intelligently check for changes between renders before updating the view, unlike conventional rendering. In pursuit of these benefits, React was implemented for the project to code a PWA in JavaScript.

3.4.2. Comma.ai - Comma2 Device

Comma.ai is a start-up company producing low-cost self-driving technology. The company recently launched the 'Comma 2', capable of enabling compatible vehicles with Advanced Driver Assistance Systems (ADAS) to self-drive using the open-source 'openpilot' software and smartphone hardware [47][48]. The Comma 2 was acquired in the aim of providing live vehicle data for verifying BCs due to the ease-of-access of the openpilot code. The RDE legislation mandates that the speed sensor input used for verification of dynamic BCs must have a sampling frequency of 1 Hz [1], which the Comma 2 achieves by connecting to the vehicle Controller Area Network (CAN) Bus. In Phase 1 of the project, the Comma 2 was installed and drive tested to ensure it was compatible with the Prius PHEV (shown in Fig. 10 [49]). The device achieved sustained periods of self-driving. In Phase 2 the Secure Shell (SSH) Protocol [50] was used to investigate the live file structure of openpilot. SSH opens an encrypted channel through which files can be accessed and transferred and commands executed. Custom Python scripts were written to connect to the Comma 2, read CAN Bus data from the vehicle and export this to a laptop. This procedure was successfully performed several times, and several sets of raw CAN data were obtained.

Fig. 10 Comma 2 Installed and in Operation On-Road [49]

CAN Bus is an international standard [51] for the networking of Electronic Control Units (ECUs) on vehicles, allowing the sending and receiving of data messages [52]. The structure of a single CAN Bus message is categorised, as shown in Fig. 11, into nine fields. The main message content is contained within the 64-bit Data Field and the vehicle speed is identified with the standard OBD-II Parameter ID (PID) of Hexadecimal '0C'.

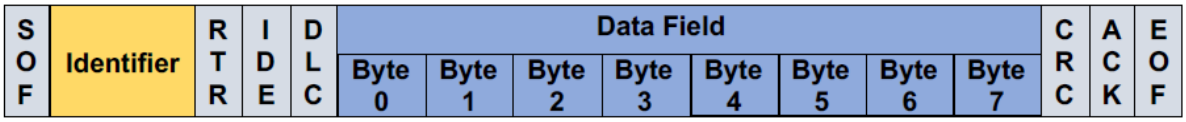

Fig. 11 CAN Bus Basic Message Structure [52]

However, a significant challenge was the decryption of the large volume of raw data obtained from the vehicle, as automotive manufacturers hold proprietary OBD-II PIDs in secrecy [52] and shift the values of CAN data. Knowledge of the encryption (such as a car 'decoder ring' [53]) is required to parse values through reverse engineering. While it was thus possible to extract the speed value from the data, the procedure was complex and difficult to replicate reliably in an app following initial testing. In addition, the dataflow structure requried to obtain, interpret and transmit the speed signal from the Comma 2 to the smartphone device for real-time display was considered inefficient when compared to sourcing the speed from GPS to validate BCs. While the key reasoning in selecting the Comma 2 initially was the high accuracy of the speed signal, the disadvantages of this choice are outweighed by the increased complexity of data transfer. Furthermore, while in previous years smartphone based GPS was considered to be of low accuracy [54], recent advances in receivers mean this is no longer the case [55]. Using in built smartphone hardware also enables the app to be fully cross-platform, allowing any user with an internet browser to use the app without having to purchase an OBD port logger. The full advantages and

disadvantages of both methods are outlined in Table 3; as shown, several factors were considered in comparing the two options. Based on this, a conclusion was made to use and refine the GPS signal from the smartphone device as the primary data source, rather than the Comma 2.

Table 3 Comma 2-GPS Speed Signal Cost Benefit Analysis

Green=High, Amber=Medium, Red=Low (Benefit)

Factor	Speed from Comma 2	Speed from GPS
Data processing requirement	Very high (requires decryption)	Minimal (instant access)
Signal latency	Med (requires external web server connection, but stable)	Low (direct access) but depends on Geolocation API frequency
Data accuracy	Very accurate (sensor recommended in legislation [1])	Med, but likely to improve in future [37]
UI Integration	More complex (external source)	Simpler (in-built)
Multi-Platform App Usability	Poor (requires Comma 2 unit)	Excellent (only requires GPS)
Signal reliability	Medium (multiple failure points)	High (GPS well established)
Hardware requirement	High (Comma 2 unit)	Low (smartphone, laptop, tablet etc)

3.5. Dataflow Structure

The greatest challenge facing the successful operation of the App is developing the speed and reliability of data acquisition, storage, and analysis. The Web Host in Fig. 12 is the platform on which the app is hosted; similar to a conventional internet page with an associated domain name. The free web host used for this project is Netlify [56], which has provided a reliable hosting service for the elapsed duration of the project.

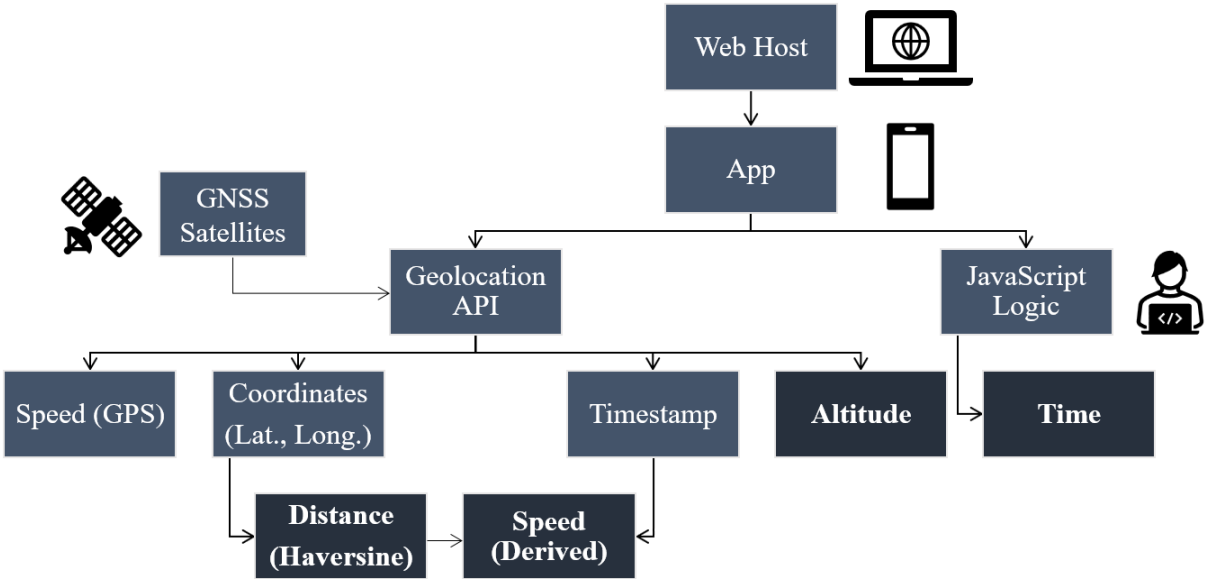

Fig. 12 App Dataflow Structure Overview (Key BC data values bold)

The functionality of the app takes inputs from two data sources; the Geolocation API [57] and simple or advanced JavaScript (or React) features. The API was used to locate the latitude and longitude of the user via Global Navigation Satellite Systems (GNSS) such as GPS, GLONASS and GALILEO (collectively referred to as 'GPS' in this report), many of which are accessible by most integrated smartphone receivers. The API also provided values of (GPS) Speed and Altitude, as well as a Timestamp in Milliseconds documenting when each data point was returned. Together, these data points are sufficient to verify all of the

modified testing BCs in Section 3.6.1. For example, the difference in sets of Coordinate values can allow a distance to be calculated using the Haversine formula (Section 3.6.2), and when this is combined with the change in Timestamp, a vehicle speed can be calculated. This was determined to be more accurate than the GPS speed estimate during testing as elaborated in Section 2.2.

3.6. **Boundary Condition Calculations**

3.6.1. Modified BCs

For developmental testing, it is impractical to set the BCs which are to be verified to those of the full RDE test. Testing according to the Agile methodology should be rapid and continuous, occurring at all levels of build complexity [35][34]. The actual BCs mandate trip times of between 90-120 Minutes, and trip segments (Urban, Rural and Motorway) have large speed ranges (Table 2, Section 3.2). Therefore, to allow frequent testing a set of 'temporary' BCs were developed as set out in Table 4, with actual values provided for comparison. This approach allows the functionality of BC verification (as per the legislation) to be built quickly to a point of reliability by using easily achievable limit values. Once operational functionality is well established, limiting values can then be changed to those of the RDE legislation, to provide fully functioning BC verification. In this aim, the trip segments were maintained, however the speed limits were lowered such that all three segments could be driven on local roads considered by the legislation to fall within the 'Urban' segment. This eliminates the need to perform long distance drive tests on motorways and rural roads, allowing many more tests to be performed. Equally, only BCs which required simple data manipulation were selected for initial testing to gradually build app complexity.

Table 4 Modified Testing BCs Compared to RDE

(U=Urban, R=Rural, M=Motorway, v=Speed)

ID	Requirement	Testing Value(s)	Actual RDE Value(s)	Unit	Data Required
A	Total trip duration	6-15	90-120	min	Time
B	Cold start duration	1	5	min	Time
C	U speed range	< 33	< 45	km/h	Speed
D	R speed range	$33 \leq v > 55$	$45 \leq v > 80$	km/h	Speed
E	M speed range	$55 \leq v > 80$	$80 \leq v > 145$	km/h	Speed
F	U test distance share	5-60	29-44	%	Distance, Speed
G	R test distance share	5-40	23-43	%	Distance, Speed
H	M test distance share	5-40	23-43	%	Distance, Speed
I	U, R, M Distance	> 0.2	> 16	km	Distance, Speed
J	Absolute Elevation change	≤ 100	≤ 100	m	Altitude
K	Idling event(s)	None > 200	None > 300	s	Time

3.6.2. Pre-Validation Data Acquisition

This section describes the process by which the app acquires the data required to process the BC verification calculations. The basic data values are Speed, Distance, Time, and Altitude, as depicted in Fig. 12. The only value provided by the Geolocation API is Altitude, whereas the remaining three must be calculated. Due to the curvature of the Earth, the actual distance between two Latitude and Longitude coordinates is larger than that approximated by a straight line, as shown by d and d' in Fig. 13. The distance can be calculated with reasonable accuracy using the Haversine formula shown in Eq. (3), which approximates the Earth to be a perfectly spherical body [58].

$$d_{hav} = 2R \cdot arcsin \cdot \sqrt{sin^2\left(\frac{\phi_B - \phi_A}{2}\right) + cos(\phi_A) \cdot cos(\phi_B) \cdot sin^2\left(\frac{\lambda_B - \lambda_A}{2}\right)} \qquad (3)$$

Where (as on Fig. 13):

- d_{hav} is the 'great circle' (surface) distance between two points on the Earth
- R is the spherical radius (in this case Earth's radius, R = 6371 km)
- ϕ_B, ϕ_A are the latitudes of points 'B' and 'A' respectively
- λ_B, λ_A are the longitudes of points 'B' and 'A' respectively

Therefore, the cumulative distance d_c is calculated by summation of the Haversine distances as in Eq. (4), where i is the 'current' iteration (position object retrieved from GPS), A is the value associated with the test starting location and n is the total number of elements in the dataset.

$$d_c = \sum_{i=A}^{n} \left(2R \cdot arcsin \cdot \sqrt{sin^2\left(\frac{\phi_i - \phi_{i-1}}{2}\right) + cos(\phi_{i-1}) \cdot cos(\phi_i) \cdot sin^2\left(\frac{\lambda_i - \lambda_{i-1}}{2}\right)} \right) \quad (4)$$

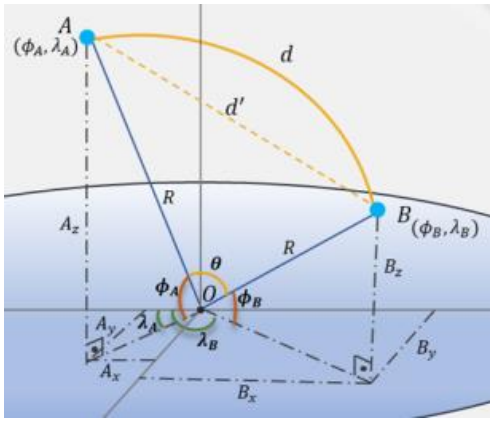

Fig. 13 Haversine Distance Calculation Parameters, Adapted from [58]

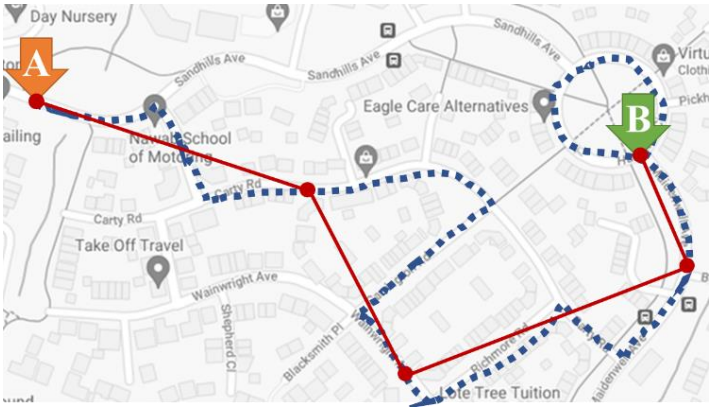

Fig. 14 Effects of Tracking Frequency on Calculated Distance Between Locations "A" and "B" (Extreme)

However, the Haversine method calculates the shortest curved (great circle) displacement and does not take account of the cumulative distance travelled, or path taken, between points 'A' and 'B'. This means that if the distance is calculated at an insufficient rate, the total value will be inaccurate as depicted in an extreme case illustrated in Google Maps in Fig. 14. The app therefore should measure user location at a frequency between 1 and 10 Hz, and the Haversine distance should be re-calculated with each detected position change. Thus, the accuracy of distance (and by extension, speed) measurements depends solely on the frequency at which a new GPS position is outputted by the Geolocation API – which is a disadvantage to using GPS derivation compared to a CAN Bus speed, although the quantified effect of this is low as elaborated in Section 3.6.4.

The Time data is calculated in two ways: using a React timer module [59], which outputs time values when the test is initiated and is stopped following test failure or user input; and by comparing the 'timestamps' of the previous and current GPS outputs. The React module allows several actions to be initiated at various 'checkpoints' throughout the test to change trip segments and check BCs, as detailed in Section 3.6.3, and is thus used for 'overall' timekeeping. On the other hand, the GPS timestamps can only be used comparatively with each other and do not indicate a continuous time (start to finish of the test). Consequently, timestamps are only utilised in calculating vehicle speed as in Eq. (5). As the trip segments form a key part of BC verification (Table 4), the accuracy of the speed value (which characterises each segment) is crucial. Speed (v_i) is therefore calculated every time a new position value is obtained from the Geolocation API. The calculation consists of the simple

distance-time relation in Eq. (5), where $d_{hav,i}$ is the Haversine distance of the current element (between the last and current recorded position) and t_i is the Timestamp.

$$v_i = \frac{d_{hav,i}}{t_i - t_{i-1}} \tag{5}$$

Based on this speed, the current trip segment (Urban, Rural or Motorway) can be ascertained depending on which speed range the value lies within. In the app this concept is extended further to all 'phases' of the test so that these can be presented to the user in the UI, including "Pre-Test", "Cold Start" and "Extreme" (speeds exceeding those allowed by the BCs). The 'isOn' variable is set to 'true' when the test is started by the user, changing the test phase to "Cold Start". Table 5 outlines the associated BCs with these phases. The cumulative distances of each trip segment (d_j) are then updated, in that when a segment (e.g. Urban) is reached, $d_{hav,i}$ is summed to a variable representing that segment. This finally allows the percentage distance share of each segment to be calculated as in Eq. (6), where j is equal to Urban, Rural or Motorway. In Fig. 15 If Statements coded to Update , the 'if' statements and ternary operators required to set these distance shares within the code are presented.

$$\%\, Share_j = \frac{d_j}{d_c} \tag{6}$$

Segment	Requirement (Testing BCs)
Pre-Test	Speed is 0 km/h, test not started
Cold Start	Within first 1 min of test start
Urban	Speed is > 0 and < 33 km/h
Rural	Speed is > 33 and < 55 km/h
Motorway	Speed is > 55 and < 80 km/h
Extreme	Speed is > 80 km/h

Table 5 Test Phases and Requirements

```
if (speed < urbanMax && speed > 0) {
  setUrbanDist(prev => prev + havDis);
}
if (speed < ruralMax && speed > urbanMax) {
  setRuralDist(prev => prev + havDis);
}
if (speed < motorwayMax && speed > ruralMax) {
  setMotorwayDist(prev => prev + havDis)
}

setUrbanShare(distance === 0 ? 0 : (urbanDist / distance) * 100);
setRuralShare(distance === 0 ? 0 : (ruralDist / distance) * 100);
setMotorwayShare(distance === 0 ? 0 : (motorwayDist / distance) * 100);
```

Fig. 15 If Statements coded to Update Cumulative Distances and Shares of Trip Segments

An overview of these nine key steps is presented in Fig. 16 in the order performed by the app in the pre-verification process. The cycle is then repeated from step 1 in each iteration. Note that the step numbers presented here do not refer to any other process within the code and are simply to illustrate the code life cycle.

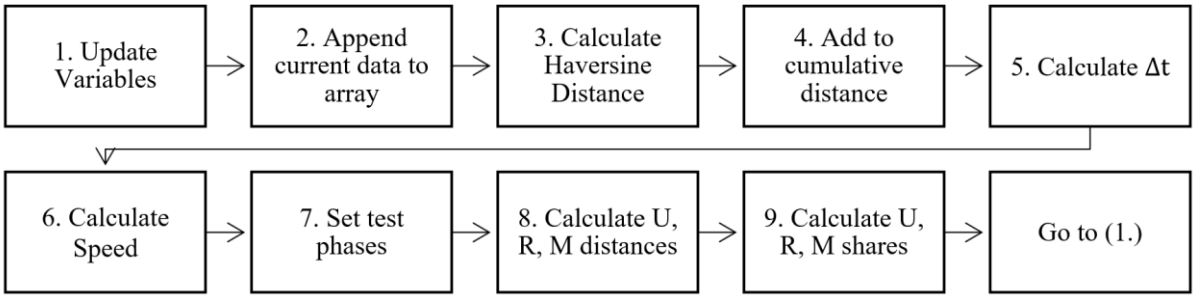

Fig. 16 Pre-BC Verification Data Procedure

3.6.3. BC Verification

React is a powerful framework for web application creation, providing tools such as the 'Hook'. An example of a Hook is the useState() function, which creates variables with a 'state' value – this is effective for data values which are constantly changing and are required to be updated virtually and displayed in the app UI rather than having to be re-assigned on

each iteration. All key data values, such as speed and cumulative distance travelled, were assigned to state variables owing to their changeability. A custom iterator function was coded using other React Hooks to repeat all data calculations and verification of BCs with each new position reading. During developmental testing (Section 3.6.4), it was discovered that due to minor time delays in the updates of state variables, the actual operational logging frequency (calculated from the change in timestamp of each position object output) was approximately half of the value selected in the iterator function. This issue was solved in Test 8a (Appendix A) by setting an optimal logging rate of 500 ms (0.5 Hz) to enable a 1 Hz 'actual' data frequency, satisfying this aspect of RDE legislation.

Upon the invalidation of any BC, a 'failure' is detected by the app (similar to the error detection inherent in programming languages). Two types are defined here to categorise failures, due to the need to perform differing actions within the verification logic based on the failure mode. The 'Hard' failure is one which causes the test to end immediately, as validity is irrecoverable. The 'Soft' failure allows the test to continue; but retains the 'failed' state such that if the test is manually ended by the user the failure status will be indicated. In either case, the exact reason (or reasons) for failure are recorded. Table 6 shows the failure modes and associated codes (similar to error codes) which the app detects upon BC invalidation. All statuses end with exclamation points and are capitalised to differentiate from nominal test phases. Failure codes start at 4 due to values 0-3 being reserved for the "Pre-Test", "In Progress", "Pass" and "Fail" statuses. The general "Fail" status is an indicator of overall test failure due to any specific failure.

Table 6 Failure Mode Definitions (Failure Codes Indicate Priority, Smallest to Largest)

(Associated BCs from Table 4)

Failure Code	Failure Mode Status	(Associated BC) Failure Mode	Failure Type
4	"MAX IDLING TIME!"	(K) Max. idling time exceeded	Hard
5	"MAX SPEED!"	(E) Max. speed exceeded	Hard
6	"HIGH TIME"	(A) Max. time exceeded	Hard
7	"LOW TIME!"	(A) Min. time not reached	Soft
8	"U MIN DISTANCE!"	(I) Urban min. distance not reached	Soft
9	"R MIN DISTANCE!"	(I) Rural min. distance not reached	Soft
10	"M MIN DISTANCE!"	(I) Motorway min. distance not reached	Soft
11	"U MIN SHARE!"	(F) Urban min. distance share not reached	Soft
12	"R MIN SHARE!"	(G) Rural min. distance share not reached	Soft
13	"M MIN SHARE!"	(H) Motorway min. distance share not reached	Soft
14	"U MAX SHARE!"	(F) Urban max. distance share exceeded	Soft
15	"R MAX SHARE!"	(G) Rural max. distance share exceeded	Soft
16	"M MAX SHARE!"	(H) Rural max. distance share exceeded	Soft
17	"MAX ALTITUDE!"	(J) Max. elevation change exceeded	Soft

It was not possible to test for failure of each individual BC (A-K in Table 4) simultaneously in the code, and this method is nonetheless inefficient from an operational perspective due to multiple updates occurring simultaneously. The more optimal solution used was to test BCs successively in the code, and this was achieved using individual 'if' statement blocks contained within the iterator function. Each BC was assigned a separate 'if' condition, allowing a particular failure to be detected and an error state variable to be updated. A consequence of this is that certain failures will be prioritised over others based on their position in the code, as JavaScript is mostly a Synchronous language (meaning only one operation can be in progress at any one time, running the script from top to bottom). This concept can be used as an efficiency advantage, as the 'hard' failures can be checked first by the code, while the 'soft' failures can be located later in the script and are only checked when failure is still recoverable – achieving the project objective of reducing wasted time on tests

which have already failed. Similarly, within each failure type (hard or soft) some failure modes are considered to have higher priority than others. This priority hierarchy is reflected in the numbering of the failure codes, in that BC verification occurs from the lowest code to the highest. For example, the maximum test time limit (Code 6 in Table 6) is less likely to occur at the beginning of the test, and thus is of a lower priority, than the maximum speed limit (5) or maximum idling time (4) being exceeded. A simplified version of this logic is summarised in the process flowchart in Fig. 17.

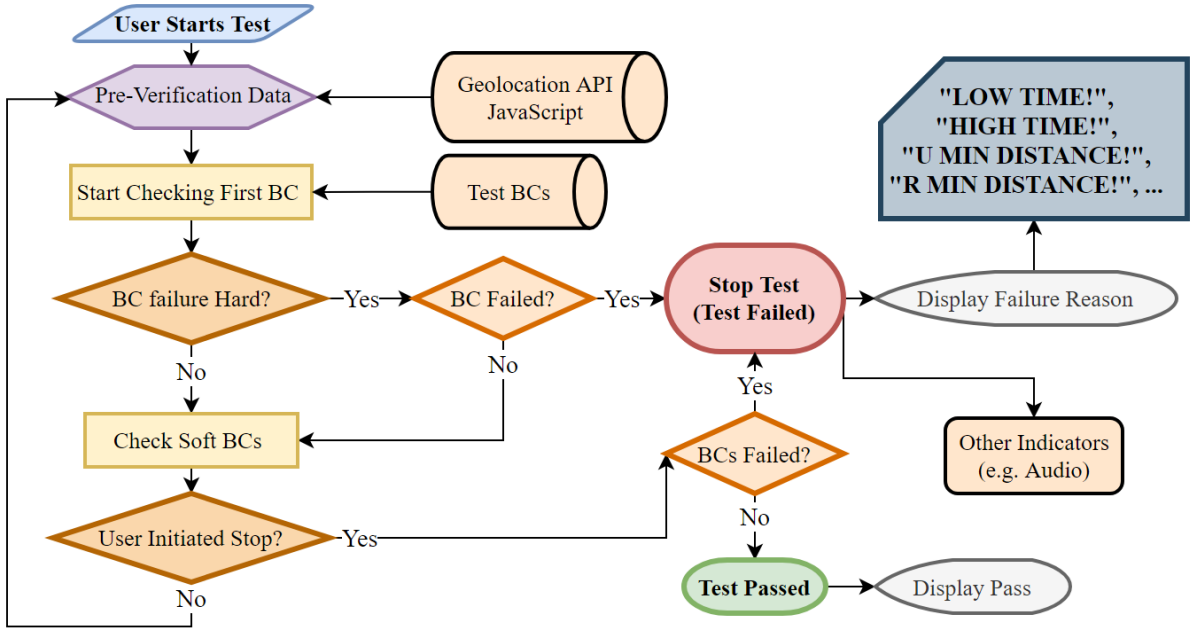

Fig. 17 BC Verification Process Flowchart

3.6.4. Developmental Testing and Results

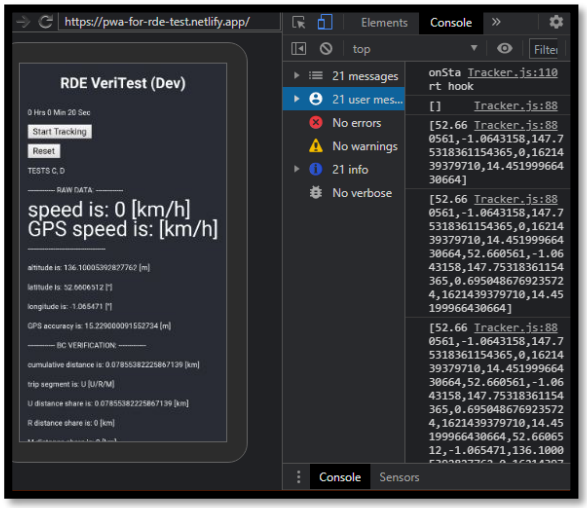

Fig. 18 Chrome DevTools Console and App UI

App testing during development was conducted in a local area, and a drive route was chosen based on the variety of driving speeds available to match the temporary testing BCs (Table 4, Section 3.6). A single test route was maintained throughout the development process for results consistency. Testing on more varied routes was scheduled to occur once early phases of app development were complete to cover variations in everyday driving. A developmental log of all completed tests is provided in Appendix A. The GPS tracking of the first drive test route is plotted in Fig. 19 and aimed to investigate the ability of the app to geolocate the smartphone position at each time interval and then output data to the console and append this to local storage. However, during initial testing, the acquired GPS position was of a low accuracy, frequency, and stability, in that results varied considerably between test to test and location updates were sporadic (as indicated by the large marker separations in Fig. 19 and Fig. 20). All map plots are made using the Mapbox API [60]. As the position input is critical to the effective operation of the app (in determining the speed, time change

and distance calculations), it was vital to establish exactly what factors influenced the unreliability of the signal. The key sources of unintentional GNSS error include multipath and atmospheric effects caused by signal reflection, as well as satellite errors [61].

Fig. 19 Test 1 Geolocation Tracking

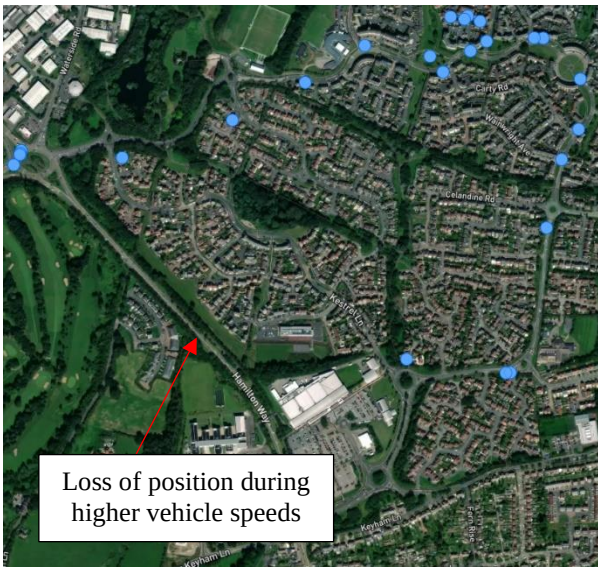

Fig. 20 Test 3 Geolocation Tracking

While it is difficult to mitigate these sources of error physically, improvements in accuracy through technical advancement are continuously being made as discussed in Section 2. Two functional methods are available for the API: 'getCurrentPosition()', which returns a static position fix and must be iterated artificially to track a moving target; and 'watchPosition()', which is invoked automatically whenever the device location changes. Both methods have an optional Boolean parameter 'high accuracy' which can be enabled or disabled. Enabling 'high accuracy' should be expected to improve the GPS location fidelity, however as the precise effects of the options appeared uncertain in initial tests; shown by the dissimilar results between Fig. 19 and Fig. 20 which used the same configuration. Thus, a testing regime was devised to conclusively investigate the options. The API contains an in-built 'accuracy' object which returns the current GPS accuracy to a confidence level of 95% [62]. Based on this object, a total of eight drive tests lasting 10 Minutes each (to meet trip duration BCs) were conducted to determine the most accurate GPS configuration (two per feature combination to reduce the effects of the 5% inaccuracy of the error signal). To further ensure test consistency, a single smartphone (Samsung Galaxy S9+) was used, as different devices used in initial tests evidenced notable variation in GPS quality. GPS receiver variation between models is common and depends on what frequency bands are accessible by the device, however in an in-depth 2013 experimental study, Bauer found that position accuracy varied between running tracking applications even on the same device [63], indicating that the app software must be optimised carefully. The data values were copied directly from the console in Google Chrome DevTools after connecting the mobile device to a PC (by enabling the 'USB debugging' feature), as shown in Fig. 18. The test device location could be virtually changed to test the effects of a changing GPS position signal. This allowed app operation to be logged and errors identified. For each test, an average accuracy was calculated from the recorded data points, and subsequently the two tests for each combination were also averaged. The results of this procedure are presented in Fig. 21.

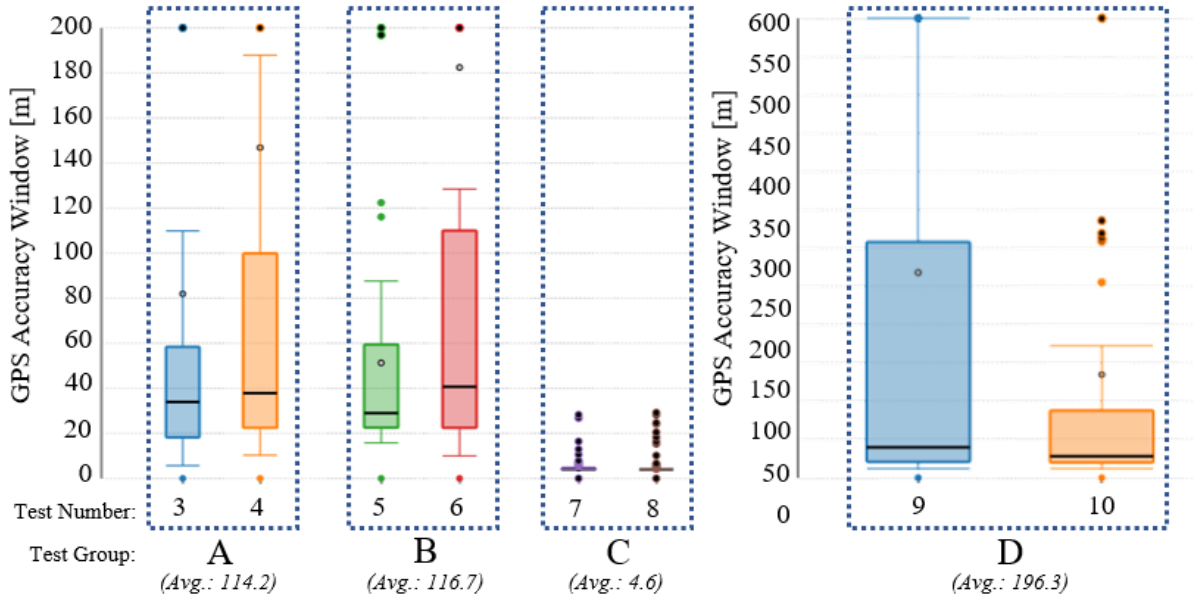

Fig. 21 Box Plots of GPS Accuracy in Tests 3-10 (2 Tests per Combination, Letters Refer to Table 7)

Table 7 Test Feature Combinations

Test Group	Tests	Geolocation API Method	High Accuracy
A	3, 4	getCurrentPosition()	Enabled
B	5, 6	getCurrentPosition()	Disabled
C	7, 8	watchPosition()	Enabled
D	9, 10	watchPosition()	Disabled

Initially, it was considered detrimental to use watchPosition due to the lack of position tracking during periods when vehicle coordinate change is not detected by the API. However, the results revealed this assumption to be misguided, as getCurrentPosition returned significantly fewer separate data points, despite being forcefully invoked on a frequent basis (at 1 Hz). As Fig. 21 Box Plots of GPS Accuracy in Tests 3-10 (2 Tests per Combination, Letters Refer to Table 7) clearly indicates, the lowest accuracy tests were those in Group D with the watchPosition and high accuracy disabled combination (tests 9 and 10), displaying an average error of 193.6 m. During Group D tests the distance calculated by the app became highly inaccurate, constantly triggering test failure. This was due to very few new position updates occurring, which meant that the Haversine distance was often very large, while the logging rate (and thus time difference between readings) remained the same, causing the calculated speed to ‘jump’ to up to two orders of magnitude higher than the actual speed. In contrast, the optimal combination was Group C, which was successfully implemented in the app to achieve an average accuracy of 4.6 m: 98% higher than that of Groups A and B. While these results could be expected, the findings of Group A and B tests are significant. The average GPS accuracy varied little between the tests, and irrespective of the high accuracy tag, both results were equally poor. An average accuracy window of 115.5 m between Groups A and B indicates that the getCurrentPosition method should not be used for any application requiring precision location tracking (using the location data for calculating other values) but could be acceptable for certain use cases. Even more so, the results imply that the watchPosition method should not be implemented in an application requiring results consistency without enabling the high accuracy tag, as the large range between tests 9 and 10 suggests. In this case, getCurrentPosition is the better alternative. However, Fig. 20 and Fig. 22 also indicate that the position logging is insufficient during high-speed (‘Motorway’

segment equivalent) portions of the test, which may be due to incapability of the API to output a position, or may reflect an overly low logging rate.

Fig. 22 Test 4 Geolocation Tracking

Fig. 23 Test 7 Geolocation Tracking

It should be noted that as this investigation was limited in its scope (having only run two tests per group) these Group A and B results are not entirely conclusive. However, the results in Fig. 21 and Fig. 23 indicate with some confidence that combination C should be used in tracking applications where possible. The geolocation plot of test 8 (not shown) is entirely consistent with that of test 7. The disadvantages of this combination are higher battery and CPU usage, however these are of lesser importance than the requirement for high accuracy for this app. During testing, another key observation was the accuracy of the GPS speed as compared with the speed calculated based on changing position coordinates. It was found that this speed was inaccurate by a factor of between 3-4 times when compared with observed vehicle speed. In contrast, the calculated speed was seen to be within 3 mph (4.8 km/h) of vehicle speed however this cannot be empirically verified due to the lack of recording of actual vehicle speed. The calculated cumulative distance travelled in the test was compared with a map-based estimate of the route using an online mapping API (MapBox [64]), and was found to be nearly exactly equal (less than 1% error), suggesting that the speed value derived from this is also similarly accurate.

Should experimentation with the final app be required, any user can initiate the app in a web browser and change the virtual device position using the ‘sensors’ tab within Chrome DevTools (inspect element) or other browser equivalents. However, changing the latitude and longitude positions too quickly will cause the test to fail due to excessive speed beyond the BC limits. If the app is used in a real test, the latitude, longitude, altitude, speed, GPS speed, timestamp and GPS accuracy are outputted to the development console and saved in local storage as a ‘coordinates’ array in this order, which can be copied and manipulated as required. Furthermore, the temporary testing BCs are left in place for ease of testing, although they can be easily altered from within the code.

3.7. User Interface

The final key component of App development is the User Interface (UI), which must be designed to provide a high-quality User Experience (UX) in line with the User Requirements outlined in Section 3.1. The UI is critical to conveying the information outputted by the verification code clearly and effectively to the user.

3.7.1. Developmental UI

During the developmental phase of the project (Phase 3, Fig. 9, Section 3.3), the requirements of the UI were very different to that of the final release candidate. While testing, it was important to be able to see all key values being calculated on screen as they vary throughout the test. This ensures that the functions of new features which are added can be checked in a real use-case, and issues such as those in Appendix A can be recognised and eliminated. Fig. 24 shows the change in UI as the background verification code became more complex and new features were added. Initially, the latitude, longitude, calculated speed and time were the only data variables available to be displayed, to test the GPS coordinate accuracy. As the app developed, version numbers were used to differentiate updated editions of the app. This was necessary as the UI became more consistent and changes between them were difficult to detect visually, so there was a high likelihood of old app versions being tested unknowingly. The GitHub platform was used for version control, however the offline access properties of the app meant that the browser cache and site data had to be cleared before new versions were loaded. By app version 1.2 (v1.2), trip segments were included, and v1.4 introduced an updated UI with smaller, organised data labels. Sound alerts were successfully trialled in the late development phase (v1.9.3).

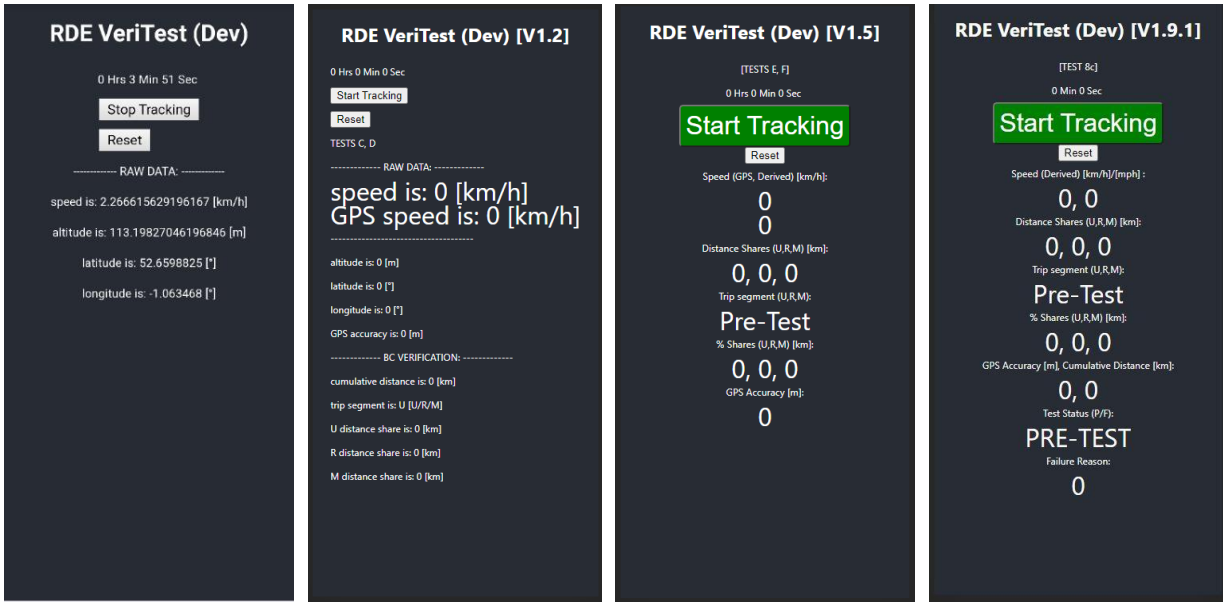

Fig. 24 Progression of the Temporary Developmental User Interface

3.7.2. Release UI and UX Overview

The aim of the final UI is to compactly display the majority of the information present on developmental UI in a clearer way. A drawback of the testing UI was the small font sizes of data labels, and during testing it was found that often significant attention was required to interpret the numbers on-screen. For the app to be safe for use during real driving this must not be the case. This problem was solved within the UI by avoiding the use of unnecessary labels for data, by instead making the identity of values clear by including their units directly adjacent and with a large font size. For example, the test timer (top left on Fig. 25) is universally recognisable due to the ‘min’ and ‘sec’ units, so it is unnecessary to label. The key data points to show to the user are those with which the user can understand the RDE BCs clearly and improve the success rate of the tests they conduct. The use of a single ‘status’ indicator is also much simpler as compared to displaying all failure modes as they occur. The key included BCs are given in Table 8.

Table 8 Key Presented Data Values and Associated Reasoning (ID from Fig. 25)

ID	Data Value	Reasoning for Inclusion
A	Time	To ensure users have an indication of the minimum time being fulfilled, and when maximum time is close to being exceeded
B	Total cumulative distance	Though there is no total distance requirement in the legislation, this parameter serves to indicate the app is functioning accurately as it can be compared with vehicle odometer values for a given trip
C/D	Speed	Presented in km/h and mph for European and UK roads, giving an indication of GPS and app accuracy and correct function
E	Idling time	This timer is initiated whenever the vehicle speed is approximately 0 km/h at any time during the test, and if the maximum idling time is nearly exceeded the colour of the numerals changes to reflect this
F	Cumulative distances for Trip Segments	Segment validation is a key part of the trip requirements, and data which can influence the progress of the test in real time, allowing the user to adjust their driving as needed
G	Distance shares (%) for Trip Segments	In the same way as distances, this data is key to passing the RDE test

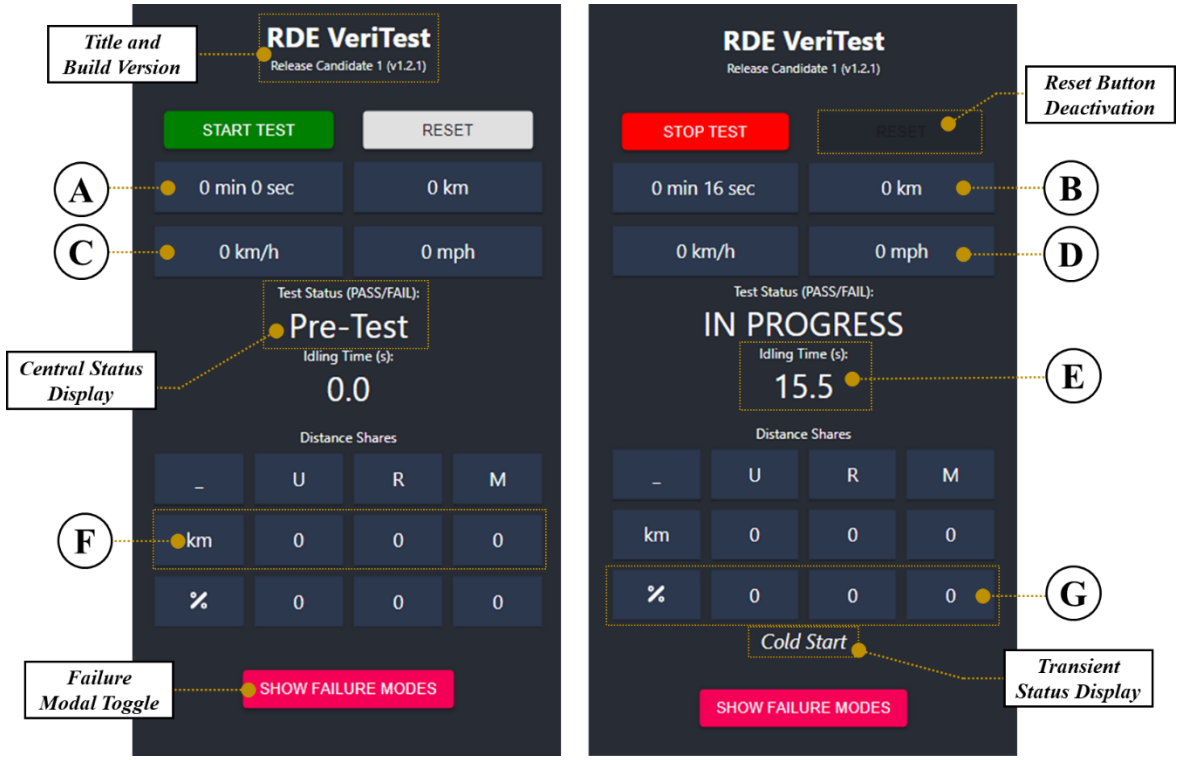

Fig. 25 Annotated Final App Graphical User Interface (Letters Refer to Table 8)

As per user requirements (Table 1, Section 3.1), a user with poor knowledge of the RDE requirements should also be able to rely on the application to govern their test without having to learn the necessary BC information. Equally, BCs are complex enough that it is easy to overload the user with information as in the case of almost all the previous RDE assistance apps created commercially (HORIBA and AVL, Section 3.1), which generally implement many small boxes containing fluctuating data values. The solution implemented here to this problem is the use of several forms of data representation to reduce visual complexity. The first method used is colour; whereby values which must be brought to the attention of the user (due to exceeding BCs during the test, for example) change colour to present this information without adding extra data. This allows the User Experience (UX) to be more intuitive and the information presented to be understood quickly, as per the findings of Doshi et al. [31]. The use of highlights (such as a red border) upon the changing of values achieves the same result.

As shown in Table 9, based on learnings from the RDE CoDriver app [42], the neutral (original) colour of each data point displayed is white, so as not to overwhelm the user. Red colour is used more sparingly (only when failures are detected), retaining the significance of the failure status for the user. In addition, the pass colour incentivises the user to drive within BC limits until all of the variables are Green, providing a visual goal and targets to aim for.

Table 9 Test Parameter Statuses and Associated Colours

Parameter Status	Status Definition	Colour
Neutral	Test in progress, no failure detected	White (Text)
Soft Failed	Soft failure detected	Red (Text)
Hard Failed	Hard Failure	Red (Border)
Passed	Parameter has achieved desired value	Green (Text)

Another method of conveying information succinctly to the user is bypassing the need for visual data altogether. The app does this using sound alerts (summarised in Table 10) which provide the user with an indication of test status without them having to look at the screen. Furthermore, the feedback the user receives improves UX by increasing interactivity [32]. The critical failure audio alert is also timed to cancel out within approximately 5 seconds of failure, ensuring that the user is not excessively distracted during driving and does not feel the urge to reach over and press the 'Stop Test' button until they can pull over in a safe place to do so.

Table 10 Sound Alerts (Audio Modified From Freesound [65] **and BigSoundBank** [66]**)**

Alert	Start Condition	Purpose	End Condition
1	Press 'Start Test'	Avoids accidental button press, provides feedback	Instantaneous
2	Press 'Stop Test'	Avoids accidental button press, provides feedback	Instantaneous
3	Test Failure	Instantly notifies user of failure and test stoppage	Timeout after 5 s
4	Test Success	Instantly notifies user of success	Instantaneous

The final key user requirement addressed in the UI is the need for learning from the test results to reduce the number of failed tests in future (Table 1). The app achieves this by providing the user with a "Show Failure Modes" button (Fig. 25), which opens a modal (pop-up) window containing logs of the final failure modes recorded during the test.

3.8. Discussion

3.8.1. Overall Achievement

The GPS and testing results indicate that the app is functioning to an excellent level of positional accuracy (as illustrated in Fig. 23, where the plotted points align exactly with the driven roads). The developmental process was thorough and provided a scheme of testing in moderate residential areas as well as higher speed settings (Appendix A). In total, 17 drive tests of the app (each lasting between 10 to 15 minutes) were undertaken throughout the project, the majority of which occurred during the developmental phase in the local residential road route (used during GPS accuracy testing). Where new (updated) app versions could not be directly tested on road, they were analysed for common failure conditions in Chrome DevTools by gradually altering the Latitude and Longitude positions. This testing flexibility was considered when preparing for the project and is one of the key realised benefits of coding a Progressive Web Application as compared to a Native Android or iOS application. It is unlikely that the quantity of tests undertaken, and the resulting developmental knowledge and solutions, would have been achieved and learnt in the case of a traditional app. The React library allows the creation of a LocalHost internal web server for

testing the full app functionality whenever required – and further to this, any changes made to the app can be viewed instantly as they occur.

The capability of the final app is measured against the initial aims and objectives of the project (Section 1) and the user and technical requirements (Section 3). The app fulfils the primary project aim of providing digital assistance for the RDE test by validating all 11 testing BCs in Table 4 (Section 3.6.1) correctly. A distinction is made here between the original aims and the achieved aims. The initial minimum intention was to verify the BCs after the test had been completed to indicate success or failure. The final app is capable of this (in verifying soft failures as in the code block in Fig. 26) but also verifies hard failures by iterating tracking on all BCs in real time and initiating failure immediately upon irrecoverable invalidation. This feature enables tests which have already failed to be ended, so that test time is not spent unnecessarily. The most successful and valuable feature of the app is the ability to see which BCs are in a (soft) state of failure during the test, and which have passed. This enables the driver to actively change their driving approach to satisfy BCs and is highly likely to reduce the number of failed RDE tests: during tests 12 and 13 which were run on different road types, the visual cues were noted as being effective in this regard. The final app functionality is summarised in Table 11.

```
// Max Distance Share [%]
if (urbanShare > uMaxShare) {
  setError14(errorStatus[14 - 4]);
} else setError14(false);
if (ruralShare > rMaxShare) {
  setError15(errorStatus[15 - 4]);
} else setError15(false);
if (motorwayShare > mMaxShare) {
  setError16(errorStatus[16 - 4]);
} else setError16(false);
```

Fig. 26 If-Else Statement Block for Verifying Maximum Distance Share BC

Table 11 Final App Functional Summary

App Function	Associated Process(es)
Accurately geolocate position of user	API watchPosition() method
Derive accurate speed (within 3 mph of actual value)	Haversine and time calculations
Indication of failure/success post-test	Validation of soft BCs
Summary of failure causes post-test	Modal ‘pop-up’ window (toggle)
Summary of neutral, failed or passed variables in test	6 visual cues (colours, boxes)
Provide sound alerts to user at key events	4 audio cues
Status of test presented to user	Central status display
Cold start and ‘extreme’ phases indicated	Transient status display
Indication of distance shares	Trip segment table
Prevent reset during active test	Temporarily deactivate reset button

3.8.2. App Limitations and Sources of Error

A limitation of the app in its current form is that users will still have to take their eyes away from the road to see the visual data cues, akin to GPS ‘Sat-Nav’ units. Tests 10 and 11 were passed without looking at the app, relying on audio cue 3 (Table 10) to signal failure, however this was done on a fixed route with moderated BCs, thus being more likely to pass the test. Increasing app autonomy for more realistic use cases could be achieved by increasing the quantity and frequency of sound alerts (a message could warn the user when nearing the limit of a segment distance share, for instance). However, achieving a balance between overwhelming the user with alerts and providing them with useful information is difficult. In addition, test results such as the plot in Fig. 23 indicates that the distance between position values increases with speed, which may affect the accuracy of calculated speed. A more optimal solution than a fixed data logging rate would be a variable rate which changes with some function of vehicle speed. Furthermore, the accuracy of the speed value would certainly be increased by using a direct OBD logger device rather than GPS – and this would also enable an empirical analysis of the speed accuracy, which was not completed in this project.

4. Conclusion

Overall, a fully functioning and operationally robust smartphone PWA exceeding all basic aims and objectives (I-IV) has been successfully coded and tested. The key purpose of assisting a single driver to successfully pass the trip verification requirements of the RDE test has been achieved in the case of the 11 BCs verified by the app. From the perspective of a proof of concept, this is sufficient to evidence the potential for the use of PWAs in an emissions verification context. The key result of initial Geolocation testing was the finding that the getCurrentPosition method within the API has a low level of accuracy and consistency when tracking the user's position, especially during periods of high vehicle speed. In addition, the watchPosition method, when called without the use of the High Accuracy tag, is especially unreliable for such purposes. The optimal solution is clearly the watchPosition, High Accuracy combination, as shown in the accurate tracking results in Fig. 23. If these options are used, reasonably accurate speed, distance and other associated parameters can be calculated effectively for BC verification. A shortcoming of the project analysis, however, is the lack of a valid empirical reference source for actual vehicle speed, due to the complexity of extracting and transporting this data from the CAN Bus. This results in a lack of conclusive confidence in assigning speed derived from the GPS to a similar status as that of the vehicle speed sensor. On the other hand, the simplicity and accessibility of the app; being installable on most devices and accessible in most browser windows (and not relying on any external hardware) enables a significant potential for creating an impact in increasing the feasibility and ease of the RDE legislative rollout. While React is complex to gain an understanding of initially, the tools and options offered by the library are powerful and well suited to this application. To note is the 'useState()' variable which can be used as an effective container for constantly updating variables such as latitude, longitude and speed for example, and enabling highly customisable decision matrices to be implemented as in Fig. 17. The full app source code can be found at: https://github.com/tbb77/rde-veritest.git, and the app can be accessed and downloaded at any time at **https://rde-vt.netlify.app**.

4.1. Further Work and Future Potential

There are many additional features which could be added to enhance the existing code. While for the purposes of a proof-of-concept (and based on the high accuracy of testing results) it was deemed unnecessary; the varying of logging rate with speed is a logical next step in improving results consistency on higher-speed routes. Moving forward, a key developmental area would be integration of the app with a direct OBD port logger, to either compare or directly source speed which would heighten test validation accuracy. In addition, verification of further trip requirements including the Dynamics and Cold Start BCs is now possible with the main structural foundation of app logic having been created. The failure alert system has been designed to simplify the process of adding new BCs, and to ensure no inherent limits are present on the number of conditions which can be added. This is made possible by simply appending new failure modes to a failure status array. Changing BCs for the purposes of app testing or to match updates in legislation may also be made simpler by providing the user with a 'settings' tab whereby BCs can be set within the app. Similarly, the ability to store and view recorded test datasets in-app could be valuable for post-analysis purposes, although this is considered an unnecessary feature in view of the main purpose of the app, which is live test validation. Following studies referenced in Section 2, a windscreen projected HUD version of the app logic could be designed to eliminate the need for the driver to break eye contact with the road – further increasing the safety of the app and richness of the UX. In future, the app scope could also be extended to integration with the PEMS to provide full emissions verification (not just the trip requirements covered in this project).

References

[1] European Commission, "COMMISSION REGULATION (EU) 2018/1832 of 5 November 2018," *Off. J. Eur. Union*, no. L 301, 2018, [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32018R1832&rid=1.

[2] Facebook Open Source, "React – A JavaScript library for building user interfaces." https://reactjs.org/ (accessed Mar. 04, 2021).

[3] J. A. Cabrera, R. S. Schmoll, G. T. Nguyen, S. Pandi, and F. H. P. Fitzek, "Softwarization and network coding in the mobile edge cloud for the tactile internet," *Proc. IEEE*, vol. 107, no. 2, pp. 350–363, 2019, doi: 10.1109/JPROC.2018.2869320.

[4] V. Masson-Delmotte, Z. P., P. H.-O., and D. Roberts, "Global Warming of 1.5°C. An IPCC Special Report on the impacts of global warming of 1.5°C above pre-industrial levels and related global greenhouse gas emission pathways, in the context of strengthening the global response to the threat of climate change," 2018.

[5] United Nations, "Adoption of the Paris Agreement," *Conf. Parties its twenty-first Sess.*, 2015.

[6] P. Koopman and M. Wagner, "Challenges in Autonomous Vehicle Testing and Validation," *SAE Int. J. Transp. Saf.*, 2016, doi: 10.4271/2016-01-0128.

[7] H. Schäfer, E. Santana, A. Haden, and R. Biasini, "A Commute in data: The comma2k19 dataset," *arXiv*. 2018.

[8] European Commission, "CO2 emissions from cars: facts and figures (infographics)," *European Parliament*, 2019. https://www.europarl.europa.eu/news/en/headlines/society/20190313STO31218/co2-emissions-from-cars-facts-and-figures-infographics (accessed Apr. 13, 2021).

[9] European Commission, "REGULATION (EC) No 443/2009 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 23 April 2009 setting emission performance standards for new passenger cars as part of the Community's integrated approach to reduce CO2 emissions from light-duty vehicles," *Off. J. Eur. Union*, 2009.

[10] European Commission, "Council Directive 91/441/EEC of 26 June 1991 amending Directive 70/220/EEC on the approximation of the laws of the Member States relating to measures to be taken against air pollution by emissions from motor vehicles," *Off. J. Eur. Union*, 1991.

[11] European Commission, "Commission Regulation (EU) No 459/2012 of 29 May 2012 amending Regulation (EC) No 715/2007 of the European Parliament and of the Council and Commission Regulation (EC) No 692/2008 as regards emissions from light passenger and commercial vehicles (Euro 6)," *Off. J. Eur. Union*, 2012.

[12] European Environment Agency (EEA), "Comparison of NOx emission standards for different Euro classes — European Environment Agency." https://www.eea.europa.eu/media/infographics/comparison-of-nox-emission-standards/view (accessed May 01, 2021).

[13] V. Franco, F. Posada Sánchez, J. German, and P. Mock, "Real-world exhaust emissions from modern diesel cars/ ICCT - The International Council On Clean Transportation," no. October 2014, 2014, [Online]. Available: http://www.theicct.org/sites/default/files/publications/ICCT_PEMS-study_diesel-cars_20141010.pdf.

[14] G. Kadijk, P. van Mensch, and J. Spreen, "Detailed investigations and real-world emission performance of Euro 6 diesel passenger cars," *TNO Rep. R10702. Delft, Netherlands.*, p. 75, 2015, [Online]. Available: https://repository.tudelft.nl/view/tno/uuid:40da980a-4c03-4b70-a9b1-240f9ea395f5.

[15] M. Weiss *et al.*, "Will Euro 6 reduce the NO x emissions of new diesel cars? - Insights from on-road tests with Portable Emissions Measurement Systems (PEMS)," *Atmos. Environ.*, vol. 62, no. 2, pp. 657–665, 2012, doi: 10.1016/j.atmosenv.2012.08.056.

[16] Peter Mock *et al.*, "From laboratory to road: A 2014 update | International Council on Clean Transportation," ICCT, 2014. [Online]. Available: http://www.theicct.org/laboratory-road-2014-update.

[17] The UK Department for Transport, "Vehicle Emissions Testing Programme - Moving Britain Ahead," no. April, 2016, [Online]. Available: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/548148/vehicle-emissions-testing-programme-web.pdf.

[18] V. Heijne *et al.*, "NOx emissions of fifteen Euro 6 diesel cars : Results of the Dutch LD road vehicle emission testing programme 2016," no. October, 2016, doi: 10.13140/RG.2.2.36653.90086.

[19] M. Ben Dror, L. Qin, and F. An, "The gap between certified and real-world passenger vehicle fuel consumption in China measured using a mobile phone application data," *Energy Policy*, 2019, doi: 10.1016/j.enpol.2018.12.039.

[20] United Nations Economic Commission for Europe, "AE/ECE/TRANS/505/Rev.2/Add.100/Rev.3 Uniform provisions concerning the approval of passenger cars powered by an internal combustion engine only, or powered by a hybrid electric power train with regard to the measurement of the emission of carbon dioxide a," p. 100, 2013, [Online]. Available: http://www.unece.org/fileadmin/DAM/trans/main/wp29/wp29regs/updates/R101r3e.pdf.

[21] Environmental Protection Agency, "Dynamometer Drive Schedules | Vehicle and Fuel Emissions Testing | US EPA." https://www.epa.gov/vehicle-and-fuel-emissions-testing/dynamometer-drive-schedules (accessed May 10, 2021).

[22] United Nations Economic Commission for Europe (UNECE), "WLTC Speed Profile," 2015. https://unece.org/DAM/trans/doc/2012/wp29grpe/WLTP-DHC-12-07e.xls (accessed Feb. 01, 2021).

[23] A. Zardini and P. Bonnel, "Real Driving Emissions Regulation: European Methodology to fine tune the EU Real Driving Emissions data evaluation method," Luxembourg, 2020. doi: 10.2760/176284.

[24] P. Mock, "Real-Driving Emissions Test Procedure for Exhaust Gas Pollutant Emissions of Cars and Light Commercial Vehicles in Europe," *Icct*, no. January, pp. 1–10, 2017.

[25] A. Clenci, V. Sălan, R. Niculescu, V. Iorga-Simăn, and C. Zaharia, "Assessment of real driving emissions via portable emission measurement system," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 252, no. 1, 2017, doi: 10.1088/1757-

899X/252/1/012084.
[26] B. Degraeuwe and M. Weiss, “Does the New European Driving Cycle (NEDC) really fail to capture the NOX emissions of diesel cars in Europe?,” *Environ. Pollut.*, vol. 222, 2017, doi: 10.1016/j.envpol.2016.12.050.
[27] L. Sileghem, D. Bosteels, J. May, C. Favre, and S. Verhelst, “Analysis of vehicle emission measurements on the new WLTC, the NEDC and the CADC,” *Transp. Res. Part D Transp. Environ.*, vol. 32, pp. 70–85, Oct. 2014, doi: 10.1016/j.trd.2014.07.008.
[28] M. A. Köhl, H. Hermanns, and S. Biewer, “Efficient monitoring of real driving emissions,” 2019, doi: 10.1007/978-3-030-03769-7_17.
[29] B. Blackwelder, K. Coleman, S. Colunga-Santoyo, J. Harrison, and D. Wozniak, “The Volkswagen Scandal,” *Univ. Richmond Scholarsh. Repos.*, vol. 1, 2016.
[30] K. Ekberg, L. Eriksson, and M. Sivertsson, “Cycle Beating - An Analysis of the Boundaries During Vehicle Testing,” *IFAC-PapersOnLine*, vol. 49, no. 11, pp. 657–664, 2016, doi: 10.1016/j.ifacol.2016.08.095.
[31] A. Doshi, S. Y. Cheng, and M. M. Trivedi, “A novel active heads-up display for driver assistance,” *IEEE Trans. Syst. Man, Cybern. Part B Cybern.*, 2009, doi: 10.1109/TSMCB.2008.923527.
[32] L. Punchoojit and N. Hongwarittorrn, “Usability Studies on Mobile User Interface Design Patterns: A Systematic Literature Review,” *Advances in Human-Computer Interaction*. 2017, doi: 10.1155/2017/6787504.
[33] A. Biørn-Hansen, T. A. Majchrzak, and T. M. Grønli, “Progressive web apps: The possibleweb-native unifier for mobile development,” *WEBIST 2017 - Proc. 13th Int. Conf. Web Inf. Syst. Technol.*, no. Webist, pp. 344–351, 2017, doi: 10.5220/0006353703440351.
[34] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, “Agile software development methods: Review and analysis,” *arXiv*, 2017.
[35] D. Martinez, X. Ferre, G. Guerrero, and N. Juristo, “An Agile-Based Integrated Framework for Mobile Application Development Considering Ilities,” *IEEE Access*, 2020, doi: 10.1109/ACCESS.2020.2987882.
[36] X. Villamil, T. Guarda, and G. N. Quina, “Agile software development for mobile applications and wireless interaction with hardware development board (Arduino),” 2018, doi: 10.23919/CISTI.2018.8399328.
[37] A. Elmezayen and A. El-Rabbany, “Precise point positioning using world’s first dual-frequency GPS/galileo smartphone,” *Sensors (Switzerland)*, vol. 19, no. 11, 2019, doi: 10.3390/s19112593.
[38] European Commission, “European vehicle emissions standards – Euro 7 for cars, vans, lorries and buses,” 2020. https://ec.europa.eu/info/law/better-regulation/have-your-say/initiatives/12313-European-vehicle-emissions-standards-Euro-7-for-cars-vans-lorries-and-buses (accessed Apr. 21, 2021).
[39] N. Ligterink, B. Rietberg, W. Hekman, van M. Pim, R. Cuelenaere, and S. van Goethem, “Review of RDE legislation: legislation text , evaluation methods and boundary conditions on the basis of RDE test data,” *TNO Rep.*, 2017.
[40] J. Merkisz, P. Bielaczyc, J. Pielecha, and J. Woodburn, “RDE testing of passenger cars: The effect of the cold start on the emissions results,” in *SAE Technical Papers*, 2019, vol. 2019-April, no. April, doi: 10.4271/2019-01-0747.
[41] R. Suarez-Bertoa *et al.*, “On-road emissions of passenger cars beyond the boundary conditions of the real-driving emissions test,” *Environ. Res.*, 2019, doi: 10.1016/j.envres.2019.108572.
[42] HORIBA, “RDE CoDriver Application Note.” 2021, [Online]. Available: https://www.avl.com/it/-/avl-smart-mobile-solutions.
[43] M. Schöggl, “REAL DRIVING EMISSIONS (RDE) Was steckt wirklich dahinter?,” *AVL Emiss. TechDay 2018*, 2018, [Online]. Available: https://www.avl.com/documents/1982862/8084402/10_RDE+-+Real+Driving+Emissions+-+was+steckt+wirklich+dahinter.pdf.
[44] AVL, “AVL Smart Mobile Solutions TM FOR RDE TESTING,” 2019.
[45] Saarbrücken Graduate School of Computer Science, “Lola Drives,” 2021. https://www.loladrives.app/ (accessed May 13, 2021).
[46] M. Weiss, E. Paffumi, M. Clairotte, Y. Drossinos, and T. Vlachos, “Including cold-start emissions in the Real-Driving Emissions (RDE) test procedure effects,” *Jt. Res. Cent.*, no. April, 2017, doi: 10.2760/70237.
[47] Comma.ai, “Openpilot GitHub.” https://github.com/commaai/openpilot (accessed Apr. 18, 2021).
[48] Y. Hou, Z. Ma, C. Liu, and C. C. Loy, “Learning to steer by mimicking features from heterogeneous auxiliary networks,” *arXiv*. 2018, doi: 10.1609/aaai.v33i01.33018433.
[49] T. Bhavsar, “Comma 2 Installed - On Road Operation.” Leicester, 2021.
[50] T. Ylonen and C. Lonvick, “RFC 4251: The Secure Shell (SSH) Protocol Architecture,” *Cisco Systerms Inc.*, 2006. https://www.hjp.at/doc/rfc/rfc4251.html (accessed May 18, 2021).
[51] ISO, “ISO - ISO 11898-1:2015 - Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling,” 2015. https://www.iso.org/standard/63648.html (accessed May 20, 2021).
[52] H. Wen, Q. Zhao, Q. A. Chen, and Z. Lin, “Automated Cross-Platform Reverse Engineering of CAN Bus Commands From Mobile Apps,” no. February, 2020, doi: 10.14722/ndss.2020.24231.
[53] Comma.ai, “GitHub - commaai/opendbc: democratize access to car decoder rings.” https://github.com/commaai/opendbc (accessed Apr. 20, 2021).
[54] A. Chowdhury, T. Chakravarty, and P. Balamuralidhar, “Estimating true speed of moving vehicle using smartphone-based GPS measurement,” *Conf. Proc. - IEEE Int. Conf. Syst. Man Cybern.*, vol. 2014-Janua, no. January, pp. 3348–3353, 2014, doi: 10.1109/smc.2014.6974444.
[55] A. Hesselbarth and L. Wanninger, “Towards centimeter accurate positioning with smartphones,” *2020 Eur. Navig. Conf. ENC 2020*, pp. 1–8, 2020, doi: 10.23919/ENC48637.2020.9317392.
[56] Netlify, “Netlify: Develop & deploy the best web experiences in record time.” https://www.netlify.com/?utm_source=google&utm_medium=paid_search&utm_campaign=12755510784&adgroup=118788138897&utm_term=netlify&utm_content=kwd-

371509120223&creative=514583565825&device=c&matchtype=e&location=1006867&gclid=Cj0KCQjwwLKFBhDPARIsAPzPi-J86Pf (accessed Apr. 20, 2021).
[57] w3.org, "Geolocation API Specification 2nd Edition." https://www.w3.org/TR/geolocation-API/ (accessed May 18, 2021).
[58] M. Basyir, M. Nasir, S. Suryati, and W. Mellyssa, "Determination of Nearest Emergency Service Office using Haversine Formula Based on Android Platform," *Emit. Int. J. Eng. Technol.*, vol. 5, no. 2, pp. 270–278, 2018, doi: 10.24003/emitter.v5i2.220.
[59] volkov97, "react-compound-timer - Github," 2021. https://github.com/volkov97/react-compound-timer (accessed May 21, 2021).
[60] Mapbox, "Maps, geocoding, and navigation APIs & SDKs | Mapbox." https://www.mapbox.com/ (accessed Apr. 01, 2021).
[61] M. Karaim, M. Elsheikh, and A. Noureldin, "GNSS Error Sources," in *Multifunctional Operation and Application of GPS*, InTech, 2018, pp. 69–83.
[62] Mozilla, "GeolocationCoordinates.accuracy - Web APIs | MDN," 2021. https://developer.mozilla.org/en-US/docs/Web/API/GeolocationCoordinates/accuracy (accessed May 18, 2021).
[63] C. Bauer, "On the (in-)accuracy of GPS measures of smartphones: A study of running tracking applications," *ACM Int. Conf. Proceeding Ser.*, pp. 335–340, 2013, doi: 10.1145/2536853.2536893.
[64] Mapbox, "Measure distances | Mapbox GL JS | Mapbox." https://docs.mapbox.com/mapbox-gl-js/example/measure/ (accessed Apr. 01, 2021).
[65] Freesound, "Freesound - sound search." https://freesound.org/search/?q=phone+disconnect&f=&s=score+desc&advanced=0&g=1 (accessed Apr. 16, 2021).
[66] Joseph Sardin, "Tone, busy 1 (Free Sound Effect) • BigSoundBank.com." https://bigsoundbank.com/detail-1610-tone-busy-1.html (accessed Apr. 16, 2021).

Appendix

Appendix A: App Developmental Testing Log

Test	v.	Test Route	Test Results (P)ass/(F)ail	Resolved Issues
1	1.0	Local	Not capable of determining P/F Speed inaccurate for tests 1-6 (low)	- Started Log
2	1.1	Local	Not capable of determining P/F	- Added GPS accuracy and other data
3	1.2	Local	Not capable of determining P/F	- Added U,R,M segments
4	1.3	Local	Not capable of determining P/F	- Fixed speed = 0
5	1.4	Local	Not capable of determining P/F	- Fixed all numbers frozen
6	1.4	Local	Not capable of determining P/F	- Fixed constantly stuck in "U" segment
7	1.5	Local	Not capable of determining P/F Large improvement in all parameters	- Gps logging rate, position error by using watchPos - Speed accuracy
8	1.6	Local	- Fail	- Fixed distance shares not adding
8a	1.7	Local	- Pass, nominal	-Fixed % share showing 1 instead of 100
8b	1.8	Local	- Pass, Number rendering issues -% distance shares incorrect	- Completed adding full BCs, apart from altitude -Removed GPS speed from UI
8c	1.9	Local	- Pass, nominal	- % distance share - Round numbers to 1/2dp for display - Periodic number rendering (caused by non-changing position values)
8d	1.9.1	Local	-Pass, nominal	- Disable reset button in test added.
9	1.9.3	Local	- Fail, Failure warning sound constantly playing after break failure, endless loop of stopwatch()	- 'N/A' failure reason appearing after failure - Fixed Viewport rendering
10	1.9.6	Local	- Pass - Distance and % shares accurate	- Changed failure status numbering to verification order - Fixed stopwatch() loop - Added 17 individual error switches
11	1.9.7	Local	-Pass, nominal	- Enabled HighAccuracy, re tested to check BC validation
12	1.9.9	City Centre	-Pass, nominal	- Reduced sound volumes
13	2.0	Urban/Dual Carriageway	-Pass, nominal	- Resolved all issues

Appendix B: Project Meeting Log

Meeting: Weekly Group **Date**: 12/01/21 Review of actions from meeting: • Discussed issues and progress on Project Preparation and Literature Review • Outlined structure and purpose of Semester 2 group meetings	Agreed actions for next meeting: • Start to prepare for final implementation of the project in the next semester Date/time of next meeting: 19/01/21, 2PM
Signed Student: Tilak Bhavsar Supervisor: Byron Mason	
Meeting: Weekly Group **Date**: 19/01/21 Review of actions from meeting: • Discussed next required steps and plans • Emphasized the necessity to be able to conduct drive testing regularly and before the Easter break to leave time for coding and debugging, as per plan in literature review	Agreed actions for next meeting: • Make initial progress as per literature review Gantt chart Date/time of next meeting: 02/02/21, 2PM
Signed Student: Tilak Bhavsar Supervisor: Byron Mason	
Meeting: Weekly Group **Date**: 02/02/21 Review of actions from meeting: • Discussed my strategy to connect to the Comma 2 unit via the Secure Shell (SSH) protocol via a computer for initial coding • Established importance of defining exact boundary conditions and values required for app validation	Agreed actions for next meeting: • Establish SSH connection and explore installed Openpilot software in real time • Complete Boundary Condition outline document Date/time of next meeting: 09/02/21, 2PM
Signed Student: Tilak Bhavsar Supervisor: Byron Mason	

Meeting: Weekly Group **Date**: 09/02/21 Review of actions from meeting: • Discussed successful SSH connection to the Comma 2 and exploration of Openpilot • Expressed my ideas on how to extract CANBUS data	Agreed actions for next meeting: • Locate critical required data within the software • Develop CANBUS data acquisition method Date/time of next meeting: 16/02/21, 2PM
Signed Student: Tilak Bhavsar Supervisor: Byron Mason	

Meeting: Weekly Group **Date**: 16/02/21 Review of actions from meeting: • Talked about necessity of a laptop in order to test the connection of the 'Comma Panda' to the CANBUS system in real time within the vehicle	Agreed actions for next meeting: • Get code ready for first vehicle test for connection and data test, fix time of meeting so laptop can be acquired for that time Date/time of next meeting: 23/02/21, 2PM
Signed Student: Tilak Bhavsar Supervisor: Byron Mason	
Meeting: Weekly Group **Date**: 23/02/21 Review of actions from meeting: • Talked about CAN Bus formatting and data extraction • CANAlyzer software license available, could use for data analysis/extraction	Agreed actions for next meeting: • Make progress on Python vehicle test code, ensure it takes account of contingencies/issues Date/time of next meeting: 08/03/21, 2PM
Signed Student: Tilak Bhavsar Supervisor: Byron Mason	
Meeting: Individual **Date**: 08/03/21 Review of actions from meeting: • Discussed plan of action for report, including components, direction • Explained code/SSH progress • Determined standard CAN Bus PID codes for data analysis • Agreed to give access to Prius sometime this week for initial connection testing	Agreed actions for next meeting: • Keep working on connection code • Start work on report Date/time of next meeting: 09/03/21, 2PM
Signed Student: Tilak Bhavsar Supervisor: Byron Mason	
Meeting: Weekly Group **Date**: 09/03/21 Review of actions from meeting: • Python script completed – problems debugged and CAN Bus logging possible • Tested script at home using blank .csv files. SFTP protocol used to then transfer files to desktop, proving data transfer	Agreed actions for next meeting: • Test script in Prius with physical connection to CAN Bus • Acquire one set of CAN Bus data to decipher data structure and extract key required information Date/time of next meeting: 12/03/21, 11:30PM
Signed Student: Tilak Bhavsar Supervisor: Byron Mason	
Meeting: In person test **Date**: 12/03/21 Review of actions from meeting: • Was able to set up connection through SSH to the Comma 2, using laptop and smartphone mobile hotspot • Python script failed to record CAN Bus data however, sending over blank .csv files	Agreed actions for next meeting: • Fix script problems • Re-test script sometime the week after next (next week is GDP week) • Acquire first CAN Bus data in this test

• Tried to solve problem but attempts were futile	Date/time of next meeting: 16/03/21, 2PM
Signed Student: Tilak Bhavsar Supervisor: Byron Mason	
Meeting: Weekly Group **Date**: 16/03/21 Review of actions from meeting: • Fixed Python script issue • Script was mostly working, required small rearrangement of code blocks, now ready for re-test	Agreed actions for next meeting: • Develop Kotlin code version of Python script to test alongside Date/time of next meeting: 26/03/21, 3PM
Signed Student: Tilak Bhavsar Supervisor: Byron Mason	
Meeting: 2nd Drive Test **Date**: 26/03/21 Review of actions from meeting: • Tested updated python scripts (can_logger_mod.py, CanAccessV2.py) • Scripts were successful, however an internal class error prevented successful reading of CAN Bus data	Agreed actions for next meeting: • Try and find the root cause of the problem ("LIBUSB" Library), and a solution • The Panda WiFi connect option may solve the issue as it doesn't use USB Date/time of next meeting: 1PM, 07/04/21
Signed Student: Tilak Bhavsar Supervisor: Byron Mason	
Meeting: 3rd Drive Test **Date**: 07/04/21 Review of actions from meeting: • Panda WiFi connection option failed • Successfully obtained first CAN Bus data using 'Tmux' command to end Comma processes	Agreed actions for next meeting: • Analyse CAN Bus data, obtain speed signal Date/time of next meeting: 20/04/21, 3PM
Signed Student: Tilak Bhavsar Supervisor: Byron Mason	
Meeting: Weekly Group **Date**: 20/04/21 Review of actions from meeting: • Discussed high complexity of CAN Bus signals and encryption by car manufacturers to discourage hacking • Softwares able to decrypt • Alternative is using pre-interpreted speed signal from OP	Agreed actions for next meeting: • Locate and channel pre-interpreted speed signal within Openpilot • Start report writing Date/time of next meeting: 27/04/21, 2PM
Signed Student: Tilak Bhavsar Supervisor: Byron Mason	
Meeting: Weekly Group **Date**: 27/04/21 Review of actions from meeting: • Discussed using internal GPS • Discussed app progress	Agreed actions for next meeting: • Continue with app development • Continue report writing – background section Date/time of next meeting: 04/05/21

<table>
<tr><td colspan="2">Signed Student: Tilak Bhavsar Supervisor: Byron Mason</td></tr>
<tr><td>Meeting: Weekly Group Date: 04/05/21
Review of actions from meeting:
Explained app progress, report writing progress</td><td>Agreed actions for next meeting:
Continue writing report – complete all key sections and then add more detail
Date/time of next meeting: 18/05/21</td></tr>
<tr><td colspan="2">Signed Student: Tilak Bhavsar Supervisor: Byron Mason</td></tr>
<tr><td>Meeting: Report 1 to 1 Date: 18/05/21
Review of actions from meeting:
Discussed report contents, background, technical development
Addressed formatting questions</td><td>Agreed actions for next meeting:
N/A

Date/time of next meeting: N/A</td></tr>
</table>