Real-Time Velocity Optimization to Minimize Energy Use in Passenger Vehicles

Authors:

Thomas Levermore, M. Necip Sahinkaya, Yahya Zweiri, Ben Neaves

Date Submitted: 2019-03-26

Keywords: fuel, fuel consumption, Optimization, dynamic programming

Abstract:

Energy use in internal combustion engine passenger vehicles contributes directly to CO 2 emissions and fuel consumption, as well as producing a number of air pollutants. Optimizing the vehicle velocity by utilising upcoming road information is an opportunity to minimize vehicle energy use without requiring mechanical design changes. Dynamic programming is capable of such an optimization task and is shown in simulation to produce fuel savings, on average 12%, compared to real driving data; however, in this paper it is also applied in real time on a Raspberry Pi, a low cost miniature computer, in situ in a vehicle. A test drive was undertaken with driver feedback being provided by a dynamic programming algorithm, and the results are compared to a simulated intelligent cruise control system that can follow the algorithm results precisely. An 8% reduction in fuel with no loss in time is reported compared to the test driver.

Record Type: Published Article

Submitted To: LAPSE (Living Archive for Process Systems Engineering)

Citation (overall record, always the latest version): LAPSE:2019.0434
Citation (this specific file, latest version): LAPSE:2019.0434-1
Citation (this specific file, this version): LAPSE:2019.0434-1v1

DOI of Published Version: https://doi.org/10.3390/en10010030

energies

MDPI

Article

Real-Time Velocity Optimization to Minimize Energy Use in Passenger Vehicles

Thomas Levermore [1,*], M. Necip Sahinkaya [1], Yahya Zweiri [1] and Ben Neaves [2]

[1] Faculty of Science, Engineering and Computing, Kingston University London, London SW15 3DW, UK; M.Sahinkaya@kingston.ac.uk (M.N.S.); Y.Zweiri@kingston.ac.uk (Y.Z.)

[2] Jaguar Land Rover Limited, Gaydon CV35 0RR, UK; bneaves@jaguarlandrover.com

* Correspondence: t.levermore@gmail.com

Academic Editor: K.T. Chau

Received: 11 November 2016; Accepted: 15 December 2016; Published: 27 December 2016

Abstract: Energy use in internal combustion engine passenger vehicles contributes directly to CO_2 emissions and fuel consumption, as well as producing a number of air pollutants. Optimizing the vehicle velocity by utilising upcoming road information is an opportunity to minimize vehicle energy use without requiring mechanical design changes. Dynamic programming is capable of such an optimization task and is shown in simulation to produce fuel savings, on average 12%, compared to real driving data; however, in this paper it is also applied in real time on a Raspberry Pi, a low cost miniature computer, in situ in a vehicle. A test drive was undertaken with driver feedback being provided by a dynamic programming algorithm, and the results are compared to a simulated intelligent cruise control system that can follow the algorithm results precisely. An 8% reduction in fuel with no loss in time is reported compared to the test driver.

Keywords: dynamic programming; optimization; fuel; fuel consumption

1. Introduction

The increasing level of CO_2 in the Earth's atmosphere is known to be one of the leading causes of global warming [1]. The contribution of the transport sector globally is estimated to be 23% of which road transport was responsible for three quarters in 2013 [2]. Combustion of both gasoline and diesel fuel produces CO_2 [3], along with a number of undesirable emissions. In order to reduce the impact of passenger vehicle CO_2 emissions on global warming, regulations have been put in place by governments across the globe [4] and are increasingly being tightened. In addition to the regulation of emissions, there is an economic incentive to reduce energy use and fuel consumption in passenger vehicles.

The approaches to fuel consumption reduction can be grouped into two types: the first involves changes to the mechanical design of the vehicle, such as weight reduction, aerodynamic improvements, as well as more complex changes, such as the introduction of hybrid electric powertrains; the second type of approach involves modifying driver behaviour by such means as navigation systems that select the most economical route [5], car sharing [6] and training or guidance to improve the style of driving [7]. Such approaches can be applied to the existing vehicle population, thus having an impact in a fast and cost-effective manner.

It is the modification of driving style that will be the focus of this paper, specifically the longitudinal velocity and gear selection. Training drivers in economical or eco-driving has been shown to improve fuel consumption [7–9]; however, the increased mental workload in complex traffic environments means eco-driving techniques cannot always be applied. The following general rules were presented in an EcoWILLpublication [10] for providers of driver training:

1. Anticipate traffic flow
2. Maintain a steady speed at low engine speed
3. Shift up early

The ability to anticipate traffic flow depends on the road layout and visibility amongst other factors and, therefore, is unreasonable to be expected consistently of a driver; however, the use of map data is becoming more prevalent in modern vehicles allowing a real-time eco-driving guide to be implemented in a vehicle. From the rules noted above, the velocity and gear selection are clearly the most important factors for the eco-driving guide to focus on.

Optimization of vehicle velocity has been frequently investigated in the literature for Heavy Goods Vehicles (HGVs) due to the cost of fuel consumption to haulage companies [11]. Publication [12] proposes a Model Predictive Control (MPC) algorithm to minimize fuel consumption, as well as deviation from a desired velocity. This approach is simulated with an HGV with artificial road profiles. An MPC algorithm is also developed in [13] to produce a time and fuel optimal velocity profile, with an average computation time of 1.26 s in simulation, but real-time implementation is considered only for further work.

Dynamic Programming (DP) [14] is applied to the problem of HGV velocity optimization in [15] at free-way speeds with a reduction in fuel consumption of 3.5% shown compared to a standard cruise control system. As HGVs are limited to a maximum speed that is lower than a passenger vehicle and drive predominantly on free-ways, the optimization problem is less complex than that of a passenger vehicle. DP is applied to a hybrid electric passenger vehicle in [16] considering road sections with varying speed limits on both highway and free-way routes. Traffic conditions are included in the optimization system presented in [17] with different levels of traffic information provided to the optimization system, and the effects are simulated. Breaking a journey into 2.5 km sections and optimizing the velocity for consecutive sections results in a fuel savings of 4% over a fixed velocity profile with no loss of time. Hybrid electric buses are the application of DP for energy management in [18] and in [19], where gear shift scheduling is the focus with fuel savings shown over the standard shift schedule.

It is seen then that DP shows promise as a method both for vehicle velocity and gear optimization; however, due to the dynamic nature of the road environment, there is a requirement to regularly update the control policy based on the current situation the vehicle faces. The application of DP in optimizing both vehicle velocity and gear selection is detailed here with emphasis on the real-time implementation of such an algorithm. As DP reduces a complex optimization problem into a multi-stage discrete optimization problem, consideration must be given to the number of stages and the discretisation of the system. The exponential increase in algorithm complexity with an increased decision variable range is investigated to ensure a compromise between the quality of results and algorithm computation time and, thus, the feasibility of real-time implementation.

2. Methodology

In [20], a DP problem is presented as having two main features, a cost function that increases cumulatively and a system that can be described by a discrete time model. Such a model can be described as follows:

$$x_{k+1} = f_k(x_k, u_k), \quad k = 0, 1, ..., N-1. \tag{1}$$

where x_{k+1} represents the state of the system currently and f_k is the transition relationship, which is a function of the current state x_k and u_k the control decision. The state and control vectors are:

$$x_k = \begin{bmatrix} v & g \end{bmatrix}^T \tag{2}$$

$$u_k = \begin{bmatrix} u_v & u_g \end{bmatrix}^T \tag{3}$$

where v is the vehicle velocity (m·s^{-1}), g is the vehicle current gear, u_v is the demanded velocity (m·s^{-1}) and u_g is the demand gear selection. The other feature noted above, the cost function, is defined by the following:

$$J_k = \frac{\lambda}{\mu_t} J_t + \frac{1-\lambda}{\mu_f} J_f \tag{4}$$

where J_k is the total cost for a discrete step, k, J_t is the cost due to the time taken, J_f is the cost due to the fuel consumed, λ is a weighting factor to prioritize time or fuel and μ_t and μ_f are normalisation factors for time and fuel, respectively. As the units of fuel and time are not comparable, normalisation is required to ensure that the maximum values for each correspond to a cost value of one in the overall cost function (4). The cost due to the time taken is calculated according to:

$$J_t = \frac{\Delta s}{v_{avg}} \tag{5}$$

where Δs (m) is the distance covered during the step and v_{avg} (m·s^{-1}) is the average speed for the step.

$$J_f = \dot{m}_f(\tau_e, \omega_e) \left(\frac{\Delta s}{v_{avg}} \right) \tag{6}$$

where $\dot{m}_f$ is the fuel consumption rate (g·s^{-1}) multiplied by the duration, in seconds, as calculated in Equation (5). The fuel consumption rate assumes a steady state engine model based on torque, τ_e, and speed, ω_e. The total cost for a velocity profile is described by the summation of each step cost and a terminal cost, $g(x_N)$:

$$J = g(x_N) + \sum_{k=0}^{N-1} J_k(J_t, J_f) \tag{7}$$

The possible decisions at each step can be presented as a multi-dimensional grid, known as a search space. In this case, the distance along the road, the vehicle velocity and the gear selection make up a three-dimensional search space, as shown in Figure 1. Three transitions are shown illustrating velocity and gear changes, where transition (a) is to a higher velocity, while shifting to a higher gear (b) is to a lower velocity while remaining in the same gear and (c) is maintaining velocity while shifting to a lower gear.

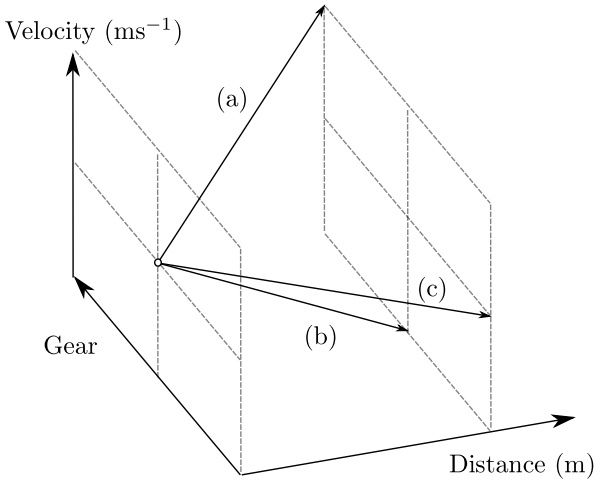

Figure 1. Dynamic programming search space with transitions (**a**) increasing velocity and selecting a higher gear; (**b**) decreasing velocity and (**c**) maintaining velocity and selecting a lower gear.

The complexity of the algorithm (O) can be calculated from the dimensions of the search space:

$$O(N_s \cdot N_v^2 \cdot N_g^2) \tag{8}$$

where N_s is the number of distance intervals between s_0 and s_{max}, N_v is the number of velocity intervals and N_g is the number of gears. As can be seen, the number of distance intervals has a linear relationship with complexity as opposed to the velocity and gear intervals, which both have an exponential relationship with complexity. The gear interval spacing is fixed by the discrete nature of the automatic transmission; however, the relationship described in Equation (8) assumes that all gears N_g are available to be reached from each of the gears, which in reality is not feasible. With the structure of the algorithm and search space specified, the calculation of each transition is detailed.

2.1. Vehicle Model

An essential part of the optimization algorithm is the vehicle model that is used in each transition calculation. It is important that the model is accurate, but can also produce results in a sufficient time so as to not impact the real-time deployment of the algorithm. For a given transition, the start and end velocity are known, and so, a backward or quasi-static vehicle model [21] is utilised. The vehicle velocity is dictated by the forces acting to accelerate or decelerate the vehicle. These forces are presented according to Newton's second law and developed from [22] in the time domain. As the road state changes with position and this can be measured directly rather than estimated based on velocity and time, there are advantages to presenting the model with position as an input variable. The drawback with this is that standard equations with time as a variable need to be converted to consider position instead. The relationship noted in [23] between acceleration in the time domain and the spatial domain equivalent is an application of the chain rule:

$$\frac{dv}{dt} = \frac{ds}{dt}\frac{dv}{ds} = v\frac{dv}{ds} \tag{9}$$

where s is the distance along the road (m), v is the vehicle velocity (m·s^{-1}) and t is the time (s). This results in an equivalent to the equation from [22] as follows:

$$m_v v(s)\frac{dv}{ds} = F_t - (F_a(v, v_w) + F_r(\alpha(s)) + F_g(\alpha(s))) \tag{10}$$

where m_v is the vehicle mass including a nominal equivalent inertial mass (kg) and $v(s)$ is the vehicle velocity (m·s^{-1}), as a function of position. F_t is the tractive force applied by the tyres of the driven wheels (N); F_a is the aerodynamic drag force of the vehicle (N), a function of vehicle and wind velocity, v_w. F_r is the rolling resistance (N); F_g is the gravitational force acting on the vehicle (N); and both are functions of the road gradient, $\alpha(s)$, which itself is a function of road distance, s.

The sensitivity of the vehicle model to changes in parameters is highlighted in Figure 2 with road gradient changes having by far the most impact on the total resistive forces under normal weather conditions, followed by wind speed, which becomes significant when above average wind speeds are considered. Both the gradient and wind speed can change during the course of a given journey. The road gradient is known from enhanced map data, and the wind speed can be estimated using historical local data or weather forecast data. The vehicle mass on the other hand has less effect, and while it can change between journeys with additional passengers or luggage, it is unlikely to change significantly during the course of a single journey being optimized.

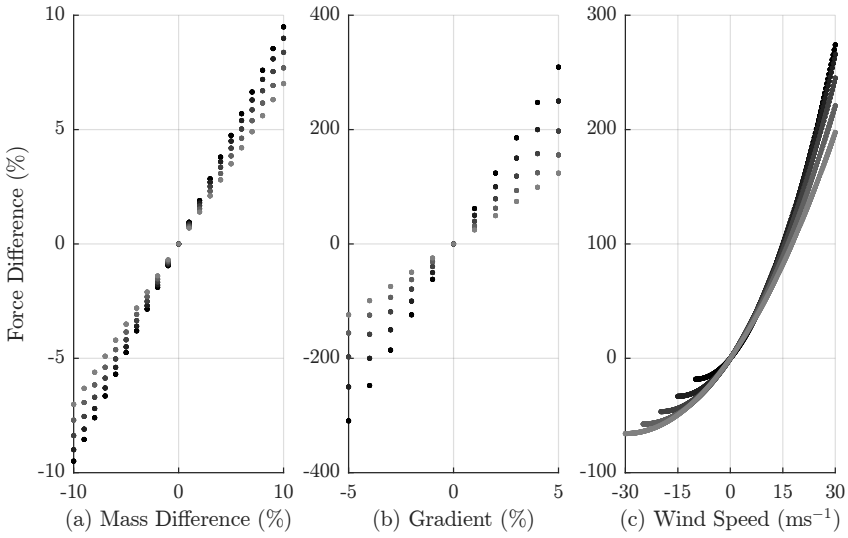

Figure 2. Vehicle model resistive force sensitivity to change in (**a**) mass; (**b**) gradient and (**c**) wind speed. The difference in total force resulting from a change from the default to the maximum and minimum of each variable is plotted for vehicle speeds from 10 m·s^{-1} to 30 m·s^{-1}. The range of values are +10–−10% for mass, +5%–−5% for gradient and maximum headwind to maximum tailwind of 30 m·s^{-1}.

In order to find the fuel consumption for a given transition, the engine speed and torque are required to be calculated. For a transition between v_0 and v_1, the acceleration, which is assumed constant, can be calculated and used in Equation (10) to find the required traction force at the wheels, F_t. Using this force and the wheel radius, r_w (m), the torque equivalent at the wheels (τ_w) in (N·m) is found. This traction torque is related to the torque available from the engine via the final drive and transmission ratios. Similarly, the engine speed, ω_e, can be calculated from the drivetrain ratios. The engine speed and torque are then used to find the fuel consumed from a Brake-Specific Fuel Consumption (BSFC) map produced by the vehicle manufacturer to identify the efficiency of the engine over a wide range of operating points. Where the engine operating point does not correspond exactly with a data point in the map, interpolation is required.

With the optimization problem and vehicle model defined, the algorithm was tested in the offline simulation on a number of real road profiles that were constructed based on measured data from real journeys undertaken unassisted by any eco-feedback system and with data logging equipment onboard. Using a commercial datalogger [24] to record GPS position, as well as all messages on the Controller Area Network (CAN) bus, journeys could be analysed retrospectively by reconstructing the road profile from the GPS longitude, latitude and elevation measurements. The position coordinate pairs were used to incorporate speed limit data into the road model by requesting such information from the Here routing service [25] at each position in the road. Pertinent data recorded from the CAN bus included vehicle speed, engine speed, calculated engine torque and a rolling fuel consumption measurement. The recorded fuel consumption is measured from the requested fuel injection quantities and updated at a rate of 5 Hz.

The velocity and gear profiles generated by the unassisted driver were used by the vehicle model to simulate fuel consumption and to compare to the algorithm results, and on average with $\lambda = 0.5$, the fuel savings were 12% and the time savings 3%. In order to assess if these improvements could be transferred from simulation to an in-vehicle eco-feedback system, such a system was developed.

2.2. Hardware

While the approach of DP aims to represent problems in a format that is suited to evaluation by a computer program, the process of implementing such a program to solve the DP problem still involves several obstacles. Two scenarios were considered where the DP algorithm would have to be implemented, the first being a simulation environment to allow testing of various scenarios and the second being implementation on a hardware platform capable of running the algorithm in real time in a test vehicle. MATLAB and Simulink are utilised across many academic and industrial fields, particularly the automotive industry [26] for modelling and simulation work, and were used to simulate the DP algorithm for testing.

(a)

(b)

Figure 3. In-vehicle deployment of the eco-feedback system (**a**) Test route and (**b**) on the extra urban roads, including multi-lane free-way and single-lane country roads.

To run the DP algorithm in a vehicle, a hardware platform was required that was compact, portable, had a flexible operating system and enough processing power to run the necessary software and hardware components that would form the complete system. The Raspberry Pi 2B single-board computer with a quad-core 900-MHz processor and 1 Gb of RAM was used for this application. The in-vehicle implementation is shown in Figure 3 along with the route used for testing. Details of the hardware are given in Table 1 and Figure 4. As the software implementation of the algorithm had to be flexible enough to perform adequately in both simulation and real-time implementation, the choice of programming language used to execute the algorithm is a crucial factor decision.

Table 1. Hardware components' details.

	Item	Model
1	Mini PC	Raspberry Pi 2 Model B
2	Display	Raspberry Pi Touch screen [27]
3	USB GPS Receiver	GlobalSat BU-353-S4 [28]
4	CAN Bus Interface	SK Pang PiCAN2 Board [29]
5	USB 4G Modem	ZTE MF823
6	CAN Bus OBD Connector	SK Pang OBDII to DB9F

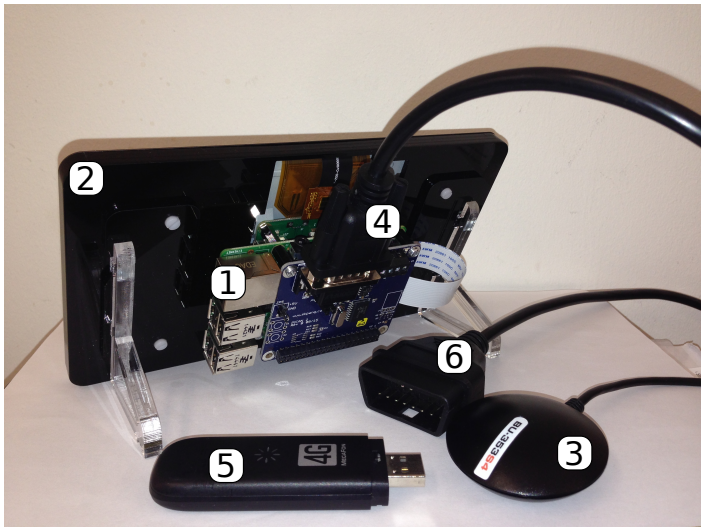

Figure 4. Hardware component setup.

2.3. Software

The C programming language has a long history of use in the automotive industry [30], and its speed of execution made it the most suitable candidate in which to implement the DP algorithm. The Raspberry Pi 2B has a Debian-based operating system pre-installed, capable of running a wide variety of programming languages; however, to ensure the consistency of results between the simulation and real-time implementation, the same code was to be used. Simulink is able to run code that has been produced in C as an S-function compiled using the MATLAB Executable (MEX) compiler, and so, the same DP algorithm code could be used in both simulation and real-time implementation, giving confidence that the simulated results would be transferable. While the core of the DP algorithm was to be written in C, an interface with external inputs, such as GPS position, was required. This interface was written as a Python script due to the ease of development and existing libraries for interacting with peripherals, such as GPS receivers and CAN bus interfaces. The DP algorithm implemented in C code can be run from within the Python script by compiling the

C code and linking it so that it can be accessed in the same way as a standard Python library function. The C code is compiled using the GNU Compiler Collection (GCC) [31].

The overall software package considers the acquisition of data, the integration of the DP algorithm, a road information database, data logging and the displaying of relevant data on a Graphical User Interface (GUI) along with audible feedback. In order to provide each of these tasks with sufficient computational power, the multiple cores of the Raspberry Pi 2 were utilised with the multiprocess library in Python [32].

Data Acquisition

The system relies on a number of data sources that need to be considered for data logging and extraction of the relevant data.

To ensure that the DP algorithm provides usable and relevant results, it is vital that the current status of the vehicle be known at the algorithm start. These data are available for transmission on the internal communication network of the vehicle, the CAN bus. In order to communicate on this network, a hardware interface is required between the Raspberry Pi and the vehicle On-Board Diagnostics (OBD) port, which has a connection to the CAN bus. Such an interface board is commercially available as the SK Pang PiCAN2 CAN Bus Interface Board [29], and an existing Python library [33] can be utilised to access the data on the network. Once the Raspberry Pi has been configured to communicate with the PiCAN board using the General Purpose IO connector (GPIO) and the device is set up as a network interface, the Python Library is able to access the CAN bus.

All control modules in the vehicle communicate on the CAN bus, resulting in a large amount of data passing through the bus network. The CAN specification [34] details a standard format for messages that includes an 11-bit identifier to allow filtering so that only messages required by the relevant system are read from the CAN bus. Using confidential manufacturer-specific message identifiers, provided by Jaguar Land Rover Limited, the required data can be extracted in this way.

Once retrieved, the required messages have to be converted so that the relevant data are in a readable format in the Python script and are subsequently logged, as well as used to update the current vehicle status. This process was developed using a Microsoft Windows-based CAN bus logging and replaying software package, BUSMASTER [35], which is able to replay previously-recorded CAN bus messages to simulate the network traffic of a particular vehicle from which CAN bus data have been recorded.

In order to locate the vehicle within the road section database, a Global Position System (GPS) receiver is used to measure the vehicle's longitude and latitude. A USB GPS receiver [28], a GlobalSat WorldCom Corporation Model BU-353-S4 is used with an existing Python library [36] that provides a conversion from a variety of GPS communication protocols to a standard, readable JavaScript Object Notation (JSON) format. With the vehicle longitude and latitude known, the road section data are interrogated to identify the closest road section to the current vehicle position using a brute-force search approach.

As the traffic conditions play an important role in the ability to follow an optimal velocity profile, a method for receiving real-time traffic data in the vehicle is required. A ZTE MF823 4G USB Modem is used to establish a mobile Internet connection, enabling traffic data to be acquired from the Here routing service [25]. A request is sent to the service for each pair of coordinates on the current road section, and the response is parsed to extract the "SpeedLimit" and "TrafficSpeed" data, which both contain the speed in $\mathrm{m \cdot s^{-1}}$.

2.4. Algorithm Implementation

When all of the current data required by the DP algorithm have been made available in the Python script, they are passed as a number of arguments to the DP C program for execution. The current vehicle speed, gear and engine torque are provided along with the weighting factors used in Equation (4). The current position in the road data is also provided along with a road identifier to

allow retrieval of the relevant road section from a database of road data. This database is developed to replicate the data fields from the Advanced Driver Assistance Interface Standard (ADASIS) that are relevant for the DP algorithm and would be provided by a commercial eHorizon system, but without the cost of such a system. Using a combination of GPS data recorded from previously-driven routes with speed limit data from the Here routing service mentioned above, the road database is constructed. Using the current vehicle longitude and latitude coordinate pair, the distance to each data point of the road can be calculated using the haversine formula [37] used in the navigation. The nearest point of road data to the current position is used as the starting point of the horizon data.

With the horizon data and current vehicle status provided to the DP algorithm, the optimisation can begin. A forward DP algorithm is implemented that calculates the transition cost for each step in the horizon from the current position to the end of the horizon. In order to reduce the computational load, an assessment is made at each step of the physically-realisable states possible in the next step, and only feasible transition costs are calculated. The physically-realisable states are assessed based on the maximum traction force available at the wheels that can be generated at the current engine speed and the maximum braking torque, each of which are used in Equation (10) to find a maximum and minimum velocity, respectively. During a gear shift, an idle fuel consumption is used during the disengagement, and an intermediate velocity is calculated with no propulsion force. This is followed by the engaged portion of the shift where the velocity of the next step is achieved and fuel consumption calculated as a transition from the intermediate velocity to the next step velocity. As the algorithm is to be used repeatedly with a receding horizon, the cost function includes a terminal cost to penalise velocity trajectories that benefit the current horizon at the cost of future horizons, for instance by reducing the velocity drastically at the end of the horizon and, thus, requiring a high acceleration at the beginning of the following horizon. An overview of the DP algorithm software implementation is shown in Figure 5.

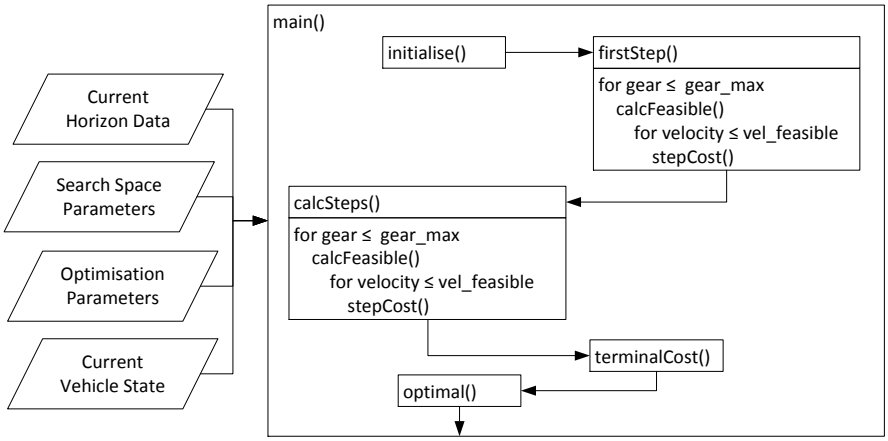

Figure 5. Dynamic programming software overview.

2.5. User Interface

To test the algorithm in a vehicle, it was required to provide a User Interface (UI) that would present information to both test personnel and the test driver. The operating status of the system components was required to be presented for the test personnel without interfering with the driver feedback system. Driving feedback had to be presented to the driver in the most straightforward way to ensure that the feedback could be followed without providing too much distraction. A combination of audible and visual feedback was implemented allowing driver preference to enable either or both of the elements. The design of UI is a discipline in its own right, and the design of automotive UI in particular presents a variety of challenges [38]; so the merits of the UI design in this case are out of the scope of this work. On completion of the DP algorithm, the initial portion of the optimal

velocity profile is compared to the current velocity, and the driver feedback is developed from this to advise either maintaining, increasing or decreasing speed to follow the optimal profile. The driver feedback is provided by means of a GUI deployed on a 7-inch touch screen display [27]. A tick icon or coloured arrow pointing up or down represents each piece of advice and is displayed on the GUI, as shown in Figure 6, along with a voice command generated from a text to speech library [39]. Current and aim velocity and gear selection are also displayed along with a number of system status indicators. The optimisation results are converted to whole miles per hour as the standard units on U.K. roads. A plot of the current optimal velocity profile and legal speed limit is also displayed for testing purposes.

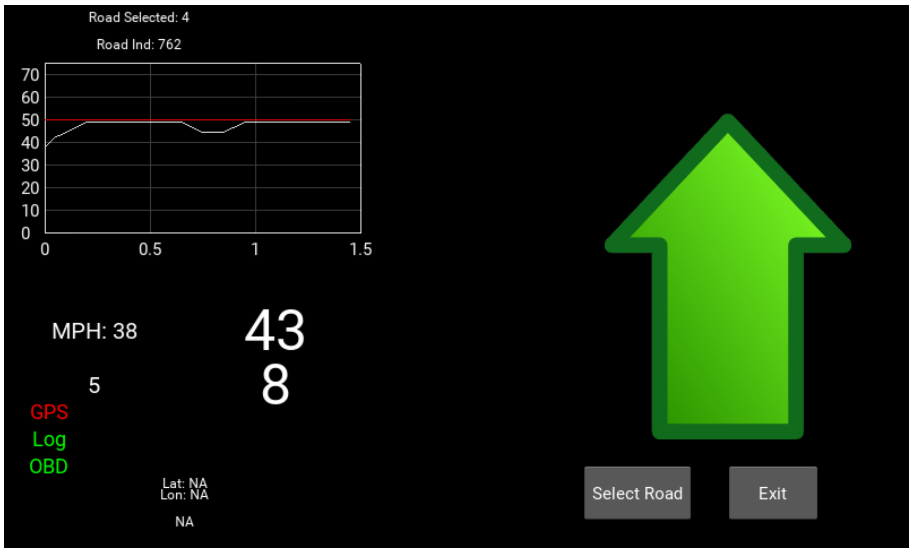

Figure 6. Driver feedback GUI with guidance to increase speed based on current speed (38) and aim speed (43) with current gear (5) and guidance gear (8). Status of GPS, datalogger (Log) and vehicle data connection (OBD) shown for testing purposes. Optimal Velocity profile shown on plot with speed limit for current road selection and position along that road.

3. Results and Discussion

Following deployment of the algorithm and associated interface on the Raspberry Pi hardware, the driver eco-guidance system was tested in a mid-sized Sports Utility Vehicle (SUV). Data were recorded from the vehicle during a journey on extra urban roads with driver guidance provided with a weighting of $\lambda = 0.5$, as a balance between fuel and time, and a receding horizon of 1.5 km. The results of this test drive were compared with simulated results to assess the ability of the driver to follow the eco-guidance, as well as the ability of the system to present guidance that is applicable to the current road conditions.

The velocity and gear profiles recorded during the test were fed into the vehicle model and used to verify that the fuel consumption produced by the vehicle model was accurate. As shown in Table 2, the model calculates a fuel consumption of 4.90 l/100 km, which is 2.9% higher than the recorded fuel consumption of 4.76 l/100 km. Similarly accurate fuel consumption results were produced across a range of recorded journeys, where historical wind speed data were available from the U.K. Meteorological Office [40]. To validate the vehicle model results, firstly, the recorded engine speed and torque were used with the BSFC map to test its accuracy, then the recorded vehicle and engine speeds and gear were used to test the transmission model. Finally, the entire model was tested with recorded vehicle speed and gear and the road profile as inputs and fuel consumption as the output. This simulated test drive fuel consumption was compared to simulated data generated by the vehicle model following the

DP profile precisely, as seen in Figure 7, and the fuel consumption and road section times are shown in Table 2. The DP algorithm was applied with fuel and time weightings of $\lambda = 0.3$, 0.5 and 0.7. The wind speed for the area of the test was on average 6.5 m·s^{-1} [40] in a north-westerly direction producing a tailwind for the duration of the journey. The use of simulated fuel consumption for the comparison as opposed to that recorded was to ensure that any difference in fuel consumption was due to the velocity and gear profile used rather than discrepancies between the model and the real vehicle.

Table 2. Fuel consumption and time from a 5 km section of the test drive. The recorded test drive velocity and gear profile are used to simulate fuel consumption and compared to results from the DP algorithm with an upper velocity constraint of (1) legal speed limit, (2) traffic speed +10% and (3) driver speed +10%.

Constraint		Fuel Consumption (l/100 km)	Difference (%)	Time (min)	Difference (%)
	Test Drive (Recorded)	4.76		3.88	
	Test Drive (Sim)	4.90	0	3.88	0
Legal Limit	DP, $\lambda = 0.3$	4.61	−5.95	3.36	−13.46
	DP, $\lambda = 0.5$	5.52	12.64	3.03	−21.98
	DP, $\lambda = 0.7$	5.91	20.46	2.95	−23.94
Traffic Speed	DP, $\lambda = 0.3$	4.44	−9.36	4.15	6.87
	DP, $\lambda = 0.5$	4.47	−8.80	4.13	6.50
	DP, $\lambda = 0.7$	4.85	−1.14	4.06	4.70
Driver Speed	DP, $\lambda = 0.3$	4.50	−8.28	3.88	0.01
	DP, $\lambda = 0.5$	4.99	1.71	3.74	−3.60
	DP, $\lambda = 0.7$	5.28	7.76	3.71	−4.49

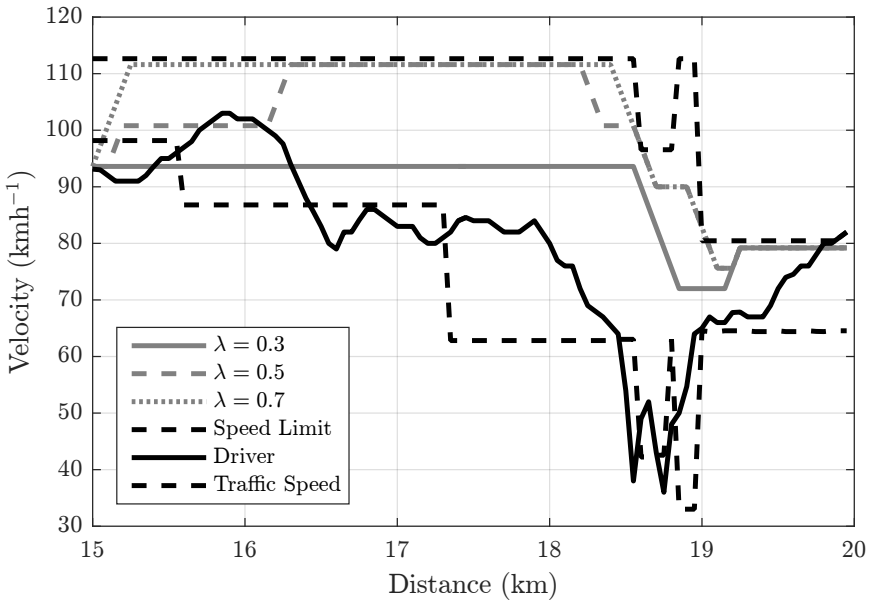

Figure 7. Speed limit-constrained velocity profiles from driver data (solid black) and DP optimal velocity (solid grey $\lambda = 0.3$, dashed grey, $\lambda = 0.5$ and dotted grey, $\lambda = 0.7$) with the speed limit upper velocity constraint. Legal speed limit and real-time traffic shown as the dashed black line, upper and lower, respectively.

The velocity profile produced by the driver is shown in Figure 7 with the legal speed limit and real-time traffic information, while the three DP velocity profiles are also shown. The driver velocity profile includes a reduction in speed prior to 19 km, which does not correspond with the speed limit data, but follows more closely the traffic velocity profile due to a roundabout. In this example, the DP algorithm only considers the legal speed limit and not the traffic speed, hence the deviation.

Using the real-time traffic information recorded during the journey, the effect of traffic on the DP algorithm can be investigated retrospectively. As seen in Figure 8, the driver velocity follows the traffic velocity rather than the speed limit; however, there is still deviation from the real-time traffic information around 16 km and 17.5 km. As such, the DP algorithm is unfairly disadvantaged by the implementation of the traffic speed as an upper limit; for this reason, a +10% upper bound is allowed above the traffic speed. The results are shown in the middle section of Table 2 with improvements in fuel consumption for $\lambda = 0.3$ and $\lambda = 0.5$ coupled with journey times that decrease, but do not meet the test drive time. Even for $\lambda = 0.7$, the results are still worse for time, which can be attributed to the upper speed limitation. For an entirely unbiased comparison, it would be necessary for the DP algorithm to exceed the traffic speed by more than the allocated 10% as the driver does. Increasing the reliability and timeliness of the traffic information would minimise this issue. In order to see the impact of a fairer comparison, a window of +10% was applied to the driver velocity and used as an upper limit for the DP algorithm, as shown in the lower section of Table 2 with the velocity profiles shown in Figure 9. This allows both lower journey times and fuel consumption results to be achieved with a 8.28% savings in fuel with no loss in time for $\lambda = 0.3$. A 3.6% reduction in time with a 1.71% increase in fuel consumption is achieved with $\lambda = 0.5$, with these results highlighting the potential for the DP algorithm when followed precisely by an intelligent cruise control system rather than driver feedback.

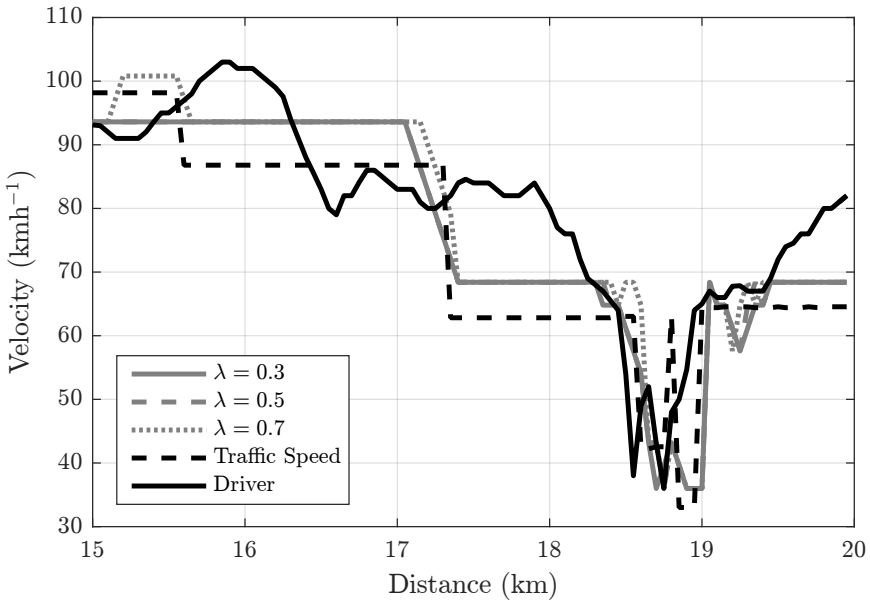

Figure 8. Traffic speed-constrained velocity profiles from DP and driver data with a 10% limit above the traffic speed-constraining DP algorithm. Driver velocity (solid black), traffic speed (dashed black), algorithm profiles with $\lambda = 0.7$ (dotted), $\lambda = 0.5$ (dashed) and $\lambda = 0.3$ (solid grey).

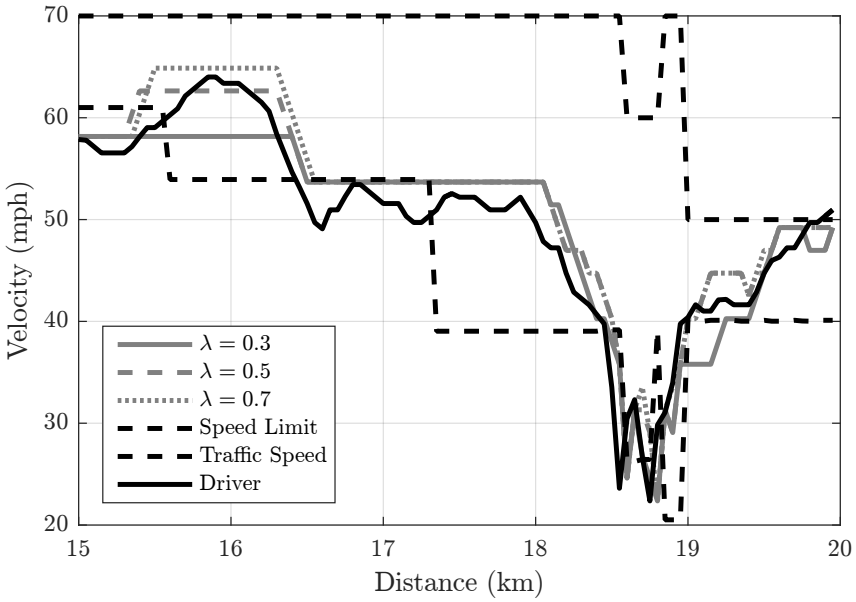

Figure 9. Driver speed-constrained velocity profiles from DP and driver data with a 10% limit above the driver velocity-constraining DP algorithm. Driver velocity (solid black), traffic speed (dashed black), algorithm profiles with $\lambda = 0.7$ (dotted), $\lambda = 0.5$ (dashed) and $\lambda = 0.3$ (solid grey).

The gear selection of the optimization algorithm and the driver are shown in Figure 10, highlighting the foresight of the algorithm; as the driver and traffic speed reduces at 18.5 km, the algorithm initiates coasting in neutral gear. It is also observed around 17 km that the driver uses Gear 8, while the algorithm remains in Gear 9; this behaviour was observed occasionally during the test run due to the automatic gearbox overriding the driver selection of Gear 9. The test road section elevation and gradient are shown in Figure 11 with a mostly downhill road slope, which along with the speed limit restrictions has the most influence on the DP algorithm velocity and gear profile.

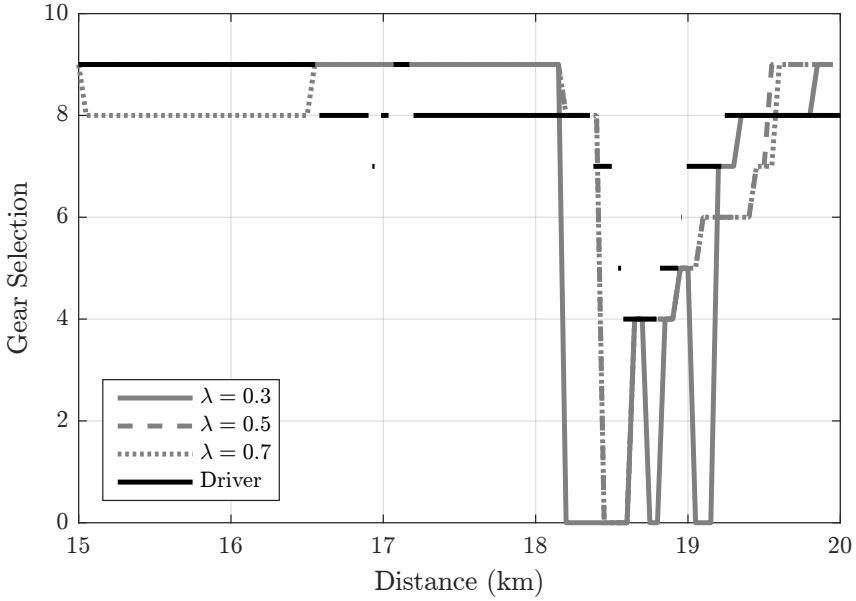

Figure 10. Gear profiles of the optimization algorithm and driver selection where zero corresponds to neutral, and the discontinuity in the driver gear selection is due to gear shift operations where no gear is selected.

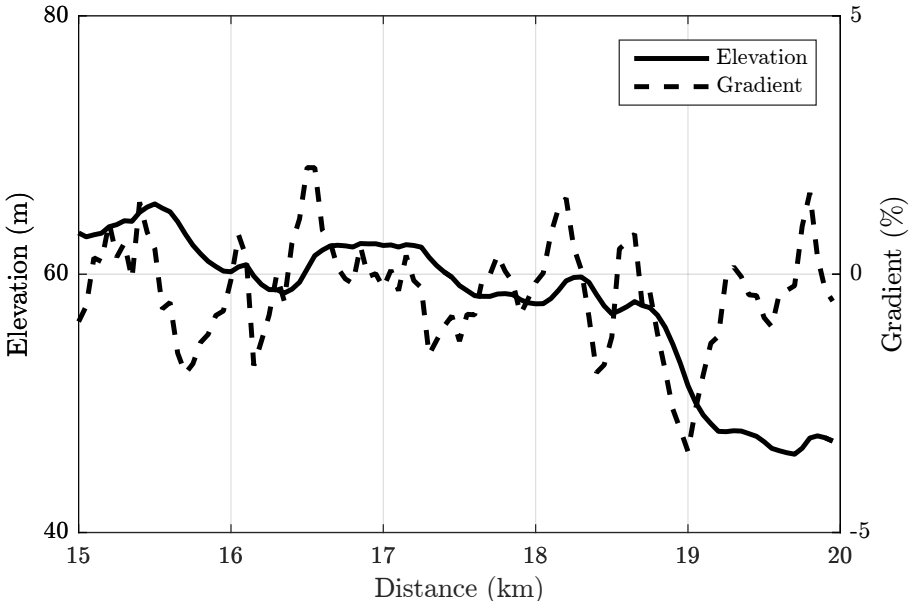

Figure 11. Test road section profile with elevation (left axis) and gradient (right axis).

Algorithm Sensitivity

The search space considered for the DP algorithm implemented in the test was sized to balance the quality of results with computation time, with a distance step interval of 50 m and 30 steps in the horizon, giving a total of 1.5 km. The velocity interval was $1\,\mathrm{m{\cdot}s^{-1}}$ as a compromise, as noted above, but also with consideration given to the ability of either a human driver or cruise control system to track precisely a target velocity. As a driver aid, the feedback system shows potential to save fuel; however, even greater benefits could be realised with automation of the vehicle velocity regulation in an intelligent cruise control system. In this scenario, the algorithm output is not limited by the driver, and a more detailed velocity profile can be accurately followed. Further investigation is presented here into the sensitivity of the algorithm to vary the search space parameters of the velocity discretisation interval and horizon length.

As noted in Equation (8), the complexity of the algorithm is exponentially related to the number of velocity intervals. This behaviour is confirmed by the simulation in Figure 12, which shows the number of calculations and computation time for a fixed horizon length of 1.5 km with a varying velocity discretisation interval. The total number of calculations of the vehicle model reduces from 3×10^8 down to 3×10^4 as the velocity interval increases from 0.1–2. Similarly, the calculation time reduces from 19 s down to 0.05 s. This relationship between complexity and velocity discretisation is consistent across a variety of roads with the caveat that the absolute values of calculation time and count decrease when the search space is restricted by speed limit, as shown by the two curves in Figure 12. The influence of this interval on the quality of the optimization results is shown in Figure 13, where the total cost (lowest plot) of the optimal profile is shown to increase as the discretisation interval increases. The total cost is a product of the journey time and fuel consumption, as in Equation (4); however, neither increase linearly with the interval size in Figure 13, which can be attributed to different velocity profiles being possible at a given interval, which presents opportunities to reduce the fuel consumption at the expense of journey time, and vice versa.

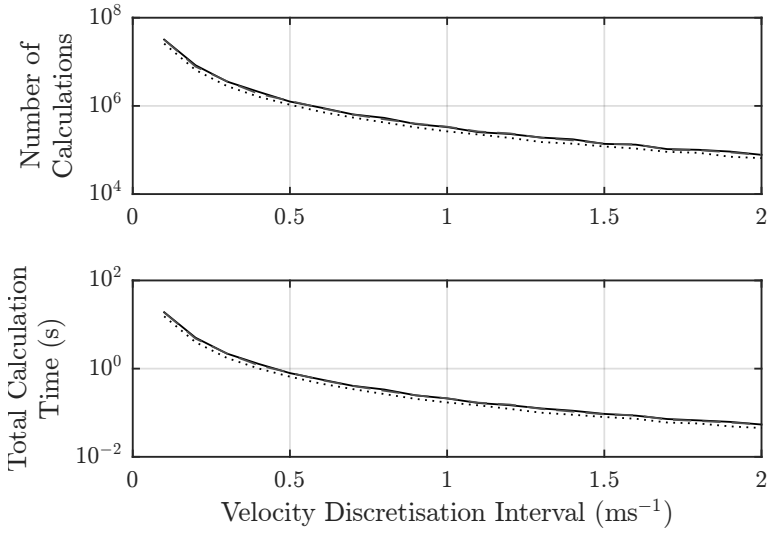

Figure 12. Optimization computational load and calculation time as a function of the velocity discretisation interval. Results shown for three different roads.

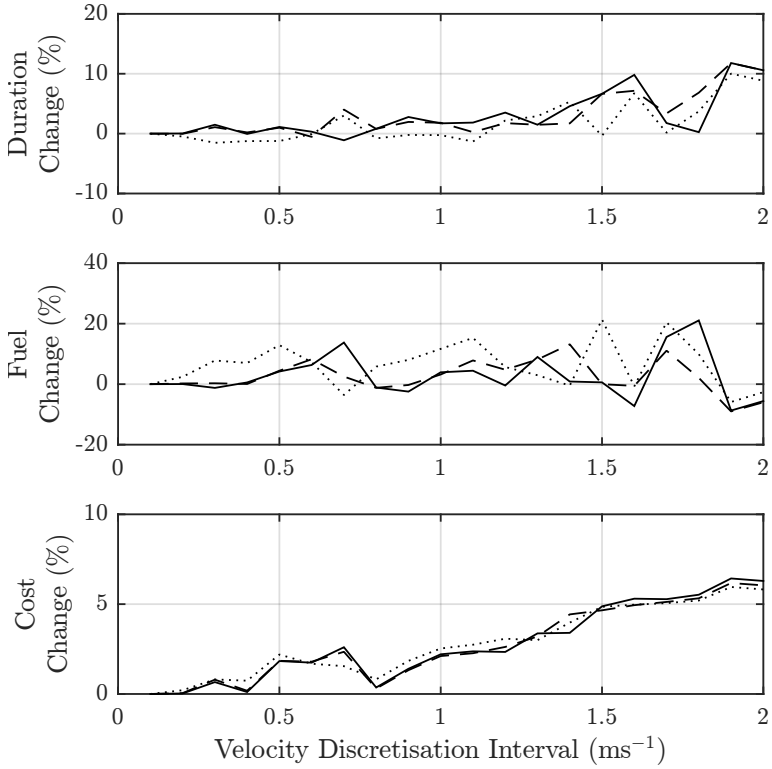

Figure 13. Optimization results as a function of the velocity discretisation interval, with journey time change (top), fuel consumption change (middle) and total cost change (bottom), using $\lambda = 0.5$ and three separate road sections.

4. Conclusions

This work has highlighted the potential of real-time vehicle speed optimization as a mechanism for energy use reduction. By balancing computation time with the accuracy of the results, it is shown that such optimization can be applied in real time and deployed in a vehicle, such that

changes to route or driving conditions can be updated and immediately impact the algorithm results. A driver feedback system is demonstrated in a test vehicle that uses the optimal velocity profile for the upcoming road to guide the driver to more economical driving. Even greater benefits are demonstrated in simulations of following the optimal velocity profile exactly, with an 8% reduction in fuel consumption produced. The development of an intelligent cruise control system to implement the optimal velocity profile could achieve similar results. The impact of traffic and wind speed on these results is highlighted as an area where improvements in live data sources for both would be beneficial.

Further tests are required to identify the energy saving potential of the system across a wide range of drivers and road profiles. The DP algorithm interface can be extended to communicate with a commercial map database as part of an eHorizon system using the Advanced Driver Assistance System Interface Specification (ADASIS) to allow testing on a large number of roads without additional road measurement required. Improvement in the feedback system user experience is not considered in this work; however, the integration of the algorithm with an eco-cruise control system would remove the need for driver feedback, thus improving the results and driver experience.

Acknowledgments: This work was part of a Ph.D. project funded by Jaguar Land Rover Limited. The provision of a test vehicle and driver is gratefully acknowledged.

Author Contributions: M. Necip Sahinkaya and Yahya Zweiri supervised the design of experiments and algorithm development; Thomas Levermore developed the algorithm, implemented the in-vehicle system and undertook the experiments; Ben Neaves organised the test vehicle and driver and provided vehicle data; Thomas Levermore wrote the paper.

Conflicts of Interest: Jaguar Land Rover Limited, the funding sponsor, collected vehicle data and made available a vehicle and driver for undertaking the testing described in this publication. The funding sponsor were involved in the decision to publish the results.

Abbreviations

BSFC	Brake-Specific Fuel Consumption
CAN	Controller Area Network
DP	Dynamic Programming
HGV	Heavy Goods Vehicle
MDPI	Multidisciplinary Digital Publishing Institute

References

1. Stocker, T.; Qin, D.; Plattner, G.; Tignor, M.; Allen, S.; Boschung, J.; Nauels, A.; Xia, Y.; Bex, V.; Midgley, P. *Climate Change 2013: The Physical Science Basis;* Fifth Assessment Report; Intergovernmental Panel on Climate Change: Geneva, Switzerland, 2013.
2. International Energy Agency. *CO_2 Emissions From Fuel Combustion Highlights 2015*; Technical Report; International Energy Agency: Paris, France, 2015.
3. Heywood, J.B. *Internal Combustion Engine Fundamentals*; McGraw-Hill: New York, NY, USA, 1988.
4. An, F.; Sauer, A. *Comparison of Passenger Vehicle Fuel Economy and Greenhouse Gas Emission Standards Around the World*; Center for Climate and Energy Solutions: Arlington, VA, USA, 2004.
5. Boriboonsomsin, K.; Barth, M.J.; Zhu, W.; Vu, A. Eco-routing navigation system based on multisource historical and real-time traffic information. *IEEE Trans. Intell. Transp. Syst.* **2012**, *13*, 1694–1704.
6. Minett, P.; Pearce, J. Estimating the energy consumption impact of casual carpooling. *Energies* **2011**, *4*, 126.
7. Barkenbus, J.N. Eco-driving: An overlooked climate change initiative. *Energy Policy* **2010**, *38*, 762–769.
8. Hiraoka, T.; Terakado, Y.; Matsumoto, S.; Yamabe, S. Quantitative evaluation of eco-driving on fuel consumption based on driving simulator experiments. In Proceedings of the 16th World Congress on Intelligent Transport Systems, Stockholm, Sweden, 21–25 September 2009; pp. 21–25.
9. Beusen, B.; Broekx, S.; Denys, T.; Beckx, C.; Degraeuwe, B.; Gijsbers, M.; Scheepers, K.; Govaerts, L.; Torfs, R.; Panis, L.I. Using on-board logging devices to study the longer-term impact of an eco-driving course. *Transp. Res. Part D Transp. Environ.* **2009**, *14*, 514–520.

10. ECOWILL Project. Available online: http://www.ecodrive.org/download/downloads/ecowill_brochure.pdf (accessed on 8 January 2016).
11. Kock, P.; Ordys, A.; Collier, G.; Weller, R. Intelligent predictive cruise control application analysis for commercial vehicles based on a commercial vehicles usage study. *SAE Int. J. Commer. Veh.* **2013**, *6*, 598–603.
12. Terwen, S.; Back, M.; Krebs, V. Predictive powertrain control for heavy duty trucks. In Proceedings of the International Federation for Automatic Control (IFAC) Symposium on Advances in Automotive Control, Salerno, Italy, 19–23 April 2004; pp. 451–457.
13. Passenberg, B.; Kock, P.; Stursberg, O. Combined time and fuel optimal driving of trucks based on a hybrid model. In Proceedings of the 2009 European Control Conference (ECC), Budapest, Hungary, 23–26 August 2009; pp. 4955–4960.
14. Bellman, R. *Dynamic Programming*; Princeton University Press: Princeton, NJ, USA, 1957.
15. Hellström, E.; Ivarsson, M.; Aslund, J.; Nielsen, L. Look-ahead control for heavy trucks to minimize trip time and fuel consumption. *Control Eng. Pract.* **2009**, *17*, 245–254.
16. Wahl, H.G.; Bauer, K.L.; Gauterin, F.; Holzäpfel, M. A real-time capable enhanced dynamic programming approach for predictive optimal cruise control in hybrid electric vehicles. In Proceedings of the 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013), Hague, The Netherlands, 6–9 October 2013; pp. 1662–1667.
17. Jiménez, F.; Cabrera-Montiel, W.; Tapia-Fernandez, S. System for road vehicle energy optimization using real time road and traffic information. *Energies* **2014**, *7*, 3576.
18. Wang, X.; He, H.; Sun, F.; Zhang, J. Application study on the dynamic programming algorithm for energy management of plug-in hybrid electric vehicles. *Energies* **2015**, *8*, 3225–3244.
19. Shen, W.; Yu, H.; Hu, Y.; Xi, J. Optimization of shift schedule for hybrid electric vehicle with automated manual transmission. *Energies* **2016**, *9*, 220.
20. Bertsekas, D. *Dynamic Programming and Optimal Control*; Athena Scientific: Belmont, MA, USA, 2005.
21. Fröberg, A.; Nielsen, L. *Dynamic Vehicle Simulation-Forward, Inverse and New Mixed Possibilities for Optimized Design and Control*; SAE Technical Paper; SAE International: Warrendale, PA, USA, 2004.
22. Rajamani, R. *Vehicle Dynamics and Control*; Springer: New York, NY, USA, 2011.
23. Monastyrsky, V.; Golownykh, I. Rapid computation of optimal control for vehicles. *Transp. Res. Part B Methodol.* **1993**, *27*, 219–227.
24. Influx Technology. Available online: http://www.influxtechnology.com/SharedFiles/Documentation/Rebel/Rebel%20XT%20Product%20Flyer.pdf (accessed on 8 January 2016).
25. HERE Routing API Developer's Guide. Available online: https://developer.here.com/rest-apis/documentation/routing (accessed on 8 July 2016).
26. Pretschner, A.; Broy, M.; Kruger, I.H.; Stauner, T. Software engineering for automotive systems: A roadmap. In Proceedings of the Future of Software Engineering, Washington, DC, USA, 23–25 May 2007; pp. 55–71.
27. Raspberry Pi Touch Display. Available online: https://www.raspberrypi.org/products/raspberry-pi-touch-display/ (accessed on 6 September 2016).
28. US GlobalSat Inc. Available online: http://usglobalsat.com/p-688-bu-353-s4.aspx (accessed on 8 July 2016).
29. SK Pang Electronics. *PiCAN 2 User Guide*, version 1.1; SK Pang Electronics: Essex, UK, 2016.
30. Broy, M.; Kruger, I.H.; Pretschner, A.; Salzmann, C. Engineering automotive software. *Proc. IEEE* **2007**, *95*, 356–373.
31. GNU Project. Availiable online: https://gcc.gnu.org/ (accessed on 8 July 2016).
32. Python Software Foundation. Available online: https://docs.python.org/2/library/multiprocessing.html (accessed on 28 June 2016).
33. Python-CAN SocketCAN. Available online: https://python-can.readthedocs.io/en/latest/interfaces/socketcan.html (accessed on 21 September 2016).
34. *CAN Specification, v2.0*; Robert Bosch GmbH: Gerlingen, Germany, 1991.
35. BUSMASTER. Availiable online: http://www.etas.com/en/products/applications_busmaster.php (accessed on 18 July 2016).
36. GPSD Service Daemon. Available online: http://www.catb.org/gpsd/ (accessed on 8 July 2016).
37. Robusto, C.C. The cosine-haversine formula. *Am. Math. Mon.* **1957**, *64*, 38–40.

38. Kern, D.; Schmidt, A. Design space for driver-based automotive user interfaces. In Proceedings of the 1st International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI 2009), Essen, Germany, 21–22 September 2009; pp. 3–10.
39. eSpeak Text to Speech. Available online: http://espeak.sourceforge.net/ (accessed on 8 July 2016).
40. Met Office. Available online: https://data.gov.uk/metoffice-data-archive (accessed on 8 July 2016).