ON GENERATING DRIVING TRAJECTORIES IN URBAN TRAFFIC TO ACHIEVE HIGHER FUEL EFFICIENCY

BY

YÜN LONG HAN

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Aerospace Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2014

Urbana, Illinois

Adviser:

Professor Petros G. Voulgaris

ABSTRACT

In this thesis, a toolkit with the purpose of generating optimal policies for driving a vehicle with information about upcoming traffic signals has been developed. The toolkit can be used to investigate how to generate the optimal velocity profile with upcoming traffic signals based on a model of second-by-second fuel consumption. To this purpose, we employ an instantaneous fuel consumption model and formulate an optimization problem for fuel minimization.

Following the problem formulation, we explore different numerical ways to solve the minimization problem by discretization. The Runge-Kutta 4^{th} Order Method (RK4) is chosen to numerically deal with the differential constraints in the optimization problem since RK4 gives higher resolution with fewer partitions when discretizing along the time horizon. Then, we turn to Direct Transcription with RK4 Steps and Parallel Shooting (DTRPS) with which we translate the minimization problem to a nonlinear programming (NLP) problem.

We also include an extensive case study for a door-to-door trip with different traveling settings: travel during which there are no traffic lights; travel with one light and travel with two lights. The result shows the capability of the toolkit. For a specific setting of a trip, an optimal profile of instantaneous velocity, acceleration and fuel consumption is generated to achieve the lowest fuel consumption for the entire trip.

To Copyleft ↄ

ACKNOWLEDGMENTS

I am extremely grateful to my adviser, Professor Petros G. Voulgaris who has rescued me from crisis after crisis, for his unlimited patience, tolerance and wisdom.

I am also deeply indebted to Daniel J. Block, the Lab Manager of Control Systems Lab in ECE who has trained and supported me so that I can work as a Graduate Teaching Assistant in his lab for the past three years.

Discussions with Professor Bruce A. Conway are indispensable to the writing of this thesis; he enlightened me as to the crux of the problem and programming.

My gratitude also goes to Laura K. Haller, Braedon L. Salz, and Dr. Sean A. Hubbard for their being scrupulous in proofreading the thesis and providing me with precious advice.

Last but not the least, I am obliged to thank all those who have helped me and loved me; with their sincere kindness and glowing warmth I shall overcome and survive the harshest winter.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DTRPS	Direct Transcription with RK4 Steps and Parallel Shooting
H.O.T	Higher Order Terms
ICE	Internal Combustion Engine
ITS	Intelligent Transportation Systems
IVP	Initial Value Problem
MPG	Miles-Per-Gallon
NLP	Nonlinear Programming
ODE	Ordinary Differential Equation
PCCP	Piecewise Constant Control Parameterization
RK4	Runge-Kutta 4^{th} Order Method
s.t.	Subject To

CHAPTER 1

INTRODUCTION

1.1 Related Work

It is believed that transportation contributes almost thirty percent of all U.S. global warming emissions [1]. Sixty percent of U.S. transportation emissions come from cars and light trucks (as shown in Figure 1.1 [1]). Every gallon of gas emits 24 pounds of carbon dioxide and other pollutants into the air. To relieve pressure on the environment, scientists and engineers are working

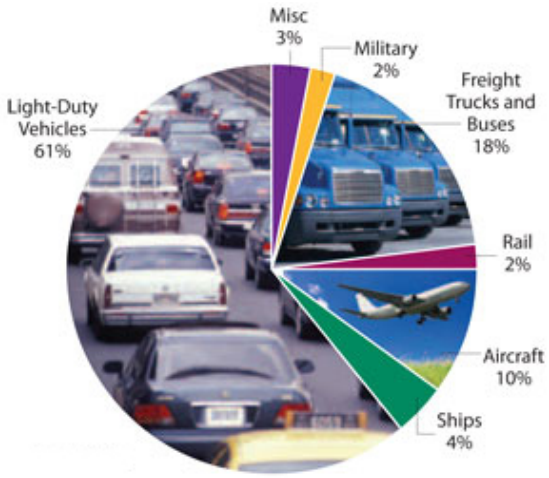


Figure 1.1: Transportation Emissions Pie Chart

hard to address the genuine public concern that fuel efficient vehicles are desperately needed. From a driver's perspective, however, to drive gasoline-powered cars with Internal Combustion Engines (ICEs) in an eco-friendly way will also lead to reducing emissions and burning less fuel. Meanwhile, less consumption of fossil fuels will help to slow down the pace of global warming and generate fewer emissions. Over the last few decades, researchers have investigated different ways to figure out possible optimal driving policies to

improve fuel efficiency. Nonetheless, modeling of fuel consumption of vehicles serves as the base to design such optimal driving trajectories.

1.1.1 Modeling of Fuel Consumption

In [2], a simple analytic relationship between fuel economy and vehicle parameters and driving cycle characteristics has been established. The model is based on engine map approximation and tractive energy. The authors further showed instead of second-by-second velocity pattern being needed, fuel consumption depends on a small number of speed characteristics that summarize a trip: average speed, average peak speed, braking time, stop time and number of stops per unit distance, of which the average speed is the main determinant of fuel use [3]. In [4], several regression models to predict fuel consumption and emission rates for light-duty vehicles and trucks have been proposed. The models utilize the vehicle's instantaneous speed and acceleration levels as independent variables so these models can be potentially used to evaluate the environment impacts of intelligent transportation systems (ITS). In [5], fuel economy has been measured using fixed urban driving schedules and fuel consumption is related to the average speed of urban traffic of a trip under a particular schedule. Consequences of nonlinearities [6] have been analyzed in specific fuel consumption (SFC) of a heavy truck combustion engine with focus on small road gradients. The results show that a constant speed is optimal if the engine torque has an affine relationship with respect to fueling. In [7], a holistic simulator has been demonstrated such that it constructs a map of fuel consumption versus speed, acceleration, and gear for the vehicle as a whole, based on statistical analysis of road test data. A data-based vehicular fuel economy modeling system has been described in [8] for the purpose of defining and documenting the optimal driving strategies for minimum fuel consumption. In [9], a model for estimating fuel consumption from instantaneous speed, acceleration and grade information has been proposed. The model is suitable for practices in the design of detailed traffic management schemes and for determining the incremental effects on fuel consumption resulting from changes in traffic management.

1.1.2 Improving Fuel Economy

An optimization problem of fuel economy subject to emission constraints without a mathematical model has been solved in [10], where the model is replaced by a sophisticated experimental test setup. Dynamic programming has been used to determine the optimal driver control of an automobile for fuel economy in [11]. The objective function is provided by a simulator that uses vehicle performance maps derived from statistical analysis of road data. Following [11], the authors did a simulation based on extensive on-road and dynamometer testing, where the results describe the optimal way to accelerate from rest to cruising speed, to drive a block between stop signs, and to cruise on hilly terrain while maintaining a given average speed [12]. In [13], a rapid computational method of optimal control of vehicles has been described. The results indicate the dependence of optimal fuel consumption on average speed for various vehicle masses. In [14], a motor vehicle velocity control has been presented to ensure a minimum fuel consumption for a vehicle with a stepped mechanical transmission and assuming the gear is unchanged during the movement. Longer distance travel has been studied in [15] and the authors directly re-derived the result that fuel consumption is approximately minimized by operation at constant speed of a land transport vehicle in traversing a path route. In [16] a scenario of driving mission for a heavy diesel truck has been studied, where look-ahead information is used in the optimization of the velocity trajectory with respect to a criterion formulation that weighs trip time and fuel consumption. In the authors' follow-up paper [17], they further developed a fuel optimal control algorithm for a heavy diesel truck that utilizes information about road topography ahead of the vehicle when the route is known. The results show that it is beneficial to formulate the look-ahead control problem in terms of kinetic energy in order to avoid oscillating solutions and reduce linear interpolation errors. In [18], a performance study of Green Light Optimized Speed Advisory (GLOSA) application in a suburban driving area has been discussed. The system has been used to monitor the impacts of GLOSA on fuel and traffic efficiency by introducing metrics of average fuel consumption and average stop time behind a traffic light. Reference [19] shows how to drive a heavy truck over various road topographies such that the fuel consumption is minimized. The results show that for level road and in small gradients the optimal solution is

to drive with constant speed. Upcoming traffic signal information has been used within the vehicle's adaptive cruise control system to reduce idle time at stop lights and fuel consumption. The optimum velocity trajectory achieves several control objectives including timely arrival at green light with minimal use of braking, maintaining safe distance between vehicles, and cruising at set speed [20]. More recently, a study in [21] shows how a driver could use traffic light information in order to adapt his/her speed profile to save fuel, where different average speed is assumed between different lights during a trip.

1.2 Outline of Chapters

It is this thesis' goal to investigate how to generate an optimal velocity profile with upcoming traffic signals based on a model of second-by-second fuel consumption. To this purpose, we will employ the Instantaneous Fuel consumption model proposed by [9] and a direct transcription method with nonlinear programming (NLP) technique to solve an optimization problem for fuel minimization.

In Chapter 2, we will have a detailed look at the Biggs-Akçelik family of models of fuel consumption and choose the Instantaneous Model to suit our aim at generating second-by-second speed velocity profile for driving within urban areas. Then based on the Instantaneous Model of fuel consumption, we formulate the optimization problem for fuel minimization of the travel.

Chapter 3 will give a brief introduction of numerical methods for solving ODEs and further dedicate the discussion of the transcription method to solving the optimization problem formulated in Chapter 2. Compared with the single step 1^{st} order Euler Method, the Runge-Kutta 4^{th} Order Method (RK4) will be used to numerically deal with the differential constraints in the optimization problem. It gives higher resolution with fewer partitions when discretizing the time horizon. Then we visit the Direct Transcription with RK4 Steps and Parallel Shooting (DTRPS) to reformulate the optimization problem defined in Chapter 2 and translate it to an NLP problem before we go to case studies.

We will have an extensive case study in Chapter 4. A door-to-door trip is considered with different schemes: travel with no lights; travel with one

light and travel with two lights. An optimal profile of instantaneous velocity, acceleration and fuel consumption is generated for each of the three schemes to achieve the lowest fuel consumption for the entire trip.

Chapter 5 can be used as a quick reference card or a short guide to the toolkit code. Particularly, this chapter gives concise explanations of parameters entering and exiting each MATLAB® `*.m` file.

Lastly, Chapter 6 concludes the thesis and proposes possible directions along which this MATLAB toolbox could further be polished, such as integrating a more automated routine to replace "sweeping" and a more user-friendly interface. We are most likely in the near future going to translate the current MATLAB code to SNOPT® Fortran code in the hope of speeding up the computation.

CHAPTER 2

MODEL OF FUEL CONSUMPTION

2.1 Overview

There are many models that have been developed over the years. Some are used to address pollutant emissions [2], [3], [22], [23]; some are based on road test data without mathematical models [4], [5], [7], [8]. There are also models that are used to address low level modeling of fuel consumption [9], [14]. Lower level models for individual passenger cars involving second-by-second velocity, acceleration etc., enable us to investigate the impacts of traffic signals on fuel consumption.

2.1.1 Four Different Fuel Consumption Models

Biggs *et al* [23] proposed a set of four different models of fuel consumption. This set of fuel consumption models covers a broad range of circumstances of a car traveling within traffic [22].

- an *Instantaneous Model* shows the instantaneous rate of fuel consumption of a specific car and how fuel consumption evolves continuously over time
- an *Elemental Model* shows the fuel usage with deceleration, acceleration, idling and cruising etc. taken into account for a shorter trip distance
- a *Running Speed Model* shows the fuel usage of a car traveling over longer distance
- an *Average Speed Model* shows the overall fuel consumption during the entire trip

The Instantaneous Model gives a detailed estimation of fuel consumption. It could be used for planning traffic lights, management of traffic at intersections, roads within small traffic networks where the instantaneous information is desired. The Elemental Model only takes cruise speed, initial speed or final speed in each driving cycle into account, where driving cycle refers to cruise, idle, acceleration or deceleration etc. The Running Speed Model requires travel time and traveled distance to assess the total fuel consumption of a trip. It ignores the traffic management factors and it is usually useful for a typical trip longer than 1 (km). The last model, the Average Speed Model, is used to estimate the total fuel consumption in a large urban traffic system where average speeds of transportation are used to assist management of traffic. The Average Speed Model is considered accurate when speed is less than 50 (km/h).

2.1.2 Biggs-Akçelik Family of Fuel Consumption Model and Urban Traffic

The latter three models of fuel consumption in Section 2.1.1 are all incrementally based on the Instantaneous Model. They are derivatives of the Instantaneous Model by dropping some required information in the Instantaneous Model. Therefore the latter three are less accurate when some aspects of the Instantaneous Model are discarded to suit one's particular purpose.

Our concern is to find the lowest fuel consumption based on second-by-second travel of a passenger car in small urban traffic area. So it is reasonable for us to use the Instantaneous Model to assess the fuel consumption in the management of different travel schemes.

2.2 Instantaneous Model of Fuel Consumption

The Instantaneous Model requires "instantaneous" speed $v(t)$, acceleration $a(t)$ and grade of roads G. Time unit is usually 1 second, hence the velocity in the model should be updated at least every 1 second to ensure the accuracy of this model. The fuel consumption rate per unit time (mL/s) for a default

car can be estimated by [9],

$$f_t(t) = \alpha + \beta_1 R_T(t) v(t) + \left[\beta_2 \frac{M}{1000} a^2(t) v(t)\right]_{a(t)>0} \quad \text{, if } R_T(t) > 0 \quad (2.1)$$
$$= \alpha \quad \text{, if } R_T(t) \leqslant 0$$

where $R_T(t)$ is the total *tractive* force required to move the car forward. It is given by,

$$R_T(t) = b_1 + b_2 v^2(t) + \frac{M}{1000} a(t) + g \frac{M}{1000} \frac{G}{100}. \quad (2.2)$$

Other required constants in calculating the fuel consumption rate in Equation (2.1) can be found in Table 2.1. These constants are specific to individual cars.

Table 2.1: Vehicle Parameters in Instantaneous Model of Fuel Consumption

Parameter	Default	Description
α	0.444	Idle fuel rate (ml/s)
β_1	0.09	Energy efficiency (ml/kJ)
β_2	0.04^a	Energy acceleration Efficiency (ml/kJ $\cdot$ m/s^2)
b_1	0.333	Drag force related to rolling resistance^b (kN)
b_2	0.00108^a	Drag due to aerodynamic resistance^b (kN/(m/s^2))
M	1200	Mass (kg)
g	9.81	Gravitational acceleration (m/s^2)

[a] A subtle disparity exists between Sanchez *et al* [24] ($\beta_2 = 0.04$ and $b_2 = 0.00108$) and Biggs *et al* [9] ($\beta_2 = 0.030$ and $b_2 = 0.0008$).
[b] Also b_1 and b_2 are related to drag associated with engine.

We can convert the fuel consumption rate per unit time $f_t(t)$ in Equation (2.1) to the fuel consumption rate per *distance* by dividing $f_t(t)$ by $v(t)$, i.e., $f_s(t) = \frac{f_t(t)}{v(t)}$. For a constant speed, level road trip (i.e., $a(t) \equiv 0$ (m/s^2) and $G \equiv 0$), Figure 2.1 shows the profile of fuel consumption rate versus speed. At around $v(t) = 13.2$ (m/s) or about 29.5 (mph), the fuel consumption rate reaches minimum in volume per unit distance (ml/m) or maximum in MPG.

For simplicity of our simulation and due to the lack of real data of topography, however, we only consider trips of a car on *level* roads. Thus, the G term in Equation (2.2) which represents percentage gradient of road will be

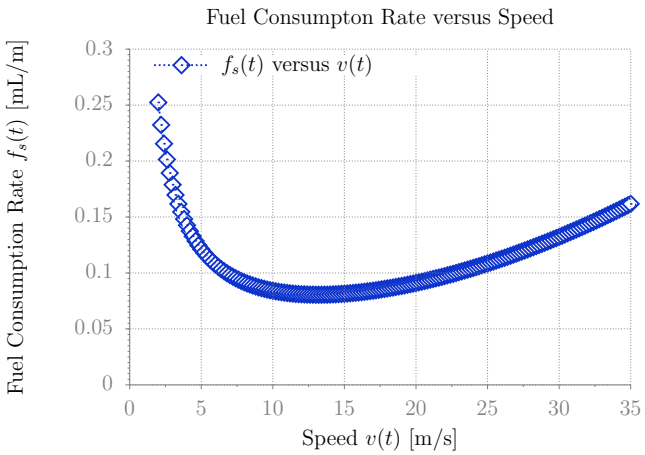


Figure 2.1: Fuel Consumption per Unit Distance versus Speed

dropped. If Real-Time topographic data should be incorporated later, it is not difficult to extend our code to update the fuel consumption model with the G term.

2.3 Problem Formulation

Now we are ready to define the fuel minimization problem for a trip of a car based on the Instantaneous Model in Section 2.2. First we consider a minimization problem in its general form,

$$\min_{\boldsymbol{p}} f(\boldsymbol{p}), \text{ s.t. } \boldsymbol{g}(\boldsymbol{p}) = \boldsymbol{0} \text{ and } \boldsymbol{h}(\boldsymbol{p}) \leqslant \boldsymbol{0}, \tag{2.3}$$

where a scalar function $f : \mathbb{R}^n \mapsto \mathbb{R}$ takes in a vector input argument $\boldsymbol{p} \in \mathbb{R}^n$. A vector function $\boldsymbol{g} : \mathbb{R}^n \mapsto \mathbb{R}^m$ (m equations) also takes $\boldsymbol{p}$ as input argument and so does another vector function $\boldsymbol{h} : \mathbb{R}^n \mapsto \mathbb{R}^l$ (l inequalities). In Problem (2.3), $f(\boldsymbol{p})$ is called *objective function*; $\boldsymbol{g}(\boldsymbol{p}) = \boldsymbol{0}$ is called *equality constraints* and $\boldsymbol{h}(\boldsymbol{p}) \leqslant \boldsymbol{0}$ *inequality constraints*.

We already defined the instantaneous fuel consumption rate in Section 2.2, so we can intuitively define a minimization problem for fuel consumption in the form of Equation (2.3) as follows,

$$\min \ J(t_f), \tag{2.4}$$

where,

$$
\begin{aligned}
J(t_f) &= \int_0^{t_f} \left(\alpha + \beta_1 \, v(t) \, R_T(t) \, q(t) + \beta_2 \, \frac{M}{1000} \, v(t) \, a^2(t) \, p(t) \, q(t) \right) \, \mathrm{d}t, \\
R_T(t) &= b_1 + b_2 \, v^2(t) + \frac{M}{1000} \, a(t), \\
p(t) &= \mathrm{sgn}^+\Big(a(t)\Big) \triangleq \begin{cases} 1, & \text{if } a(t) > 0 \\ 0, & \text{other} \end{cases} \\
q(t) &= \mathrm{sgn}^+\Big(R_T(t)\Big) \triangleq \begin{cases} 1, & \text{if } R_T(t) > 0 \\ 0, & \text{other} \end{cases}
\end{aligned}
\tag{2.5}
$$

Furthermore, displacement $s(t)$, velocity $v(t)$ and acceleration $a(t)$ are subject to a set of differential equations representing the system dynamics,

$$
\begin{cases} \dot{s}(t) = v(t), \text{ with } \; s(0) = 0 \text{ and } s(t_f) = D \\ \dot{v}(t) = a(t), \qquad\qquad v(0) = 0 \end{cases}
$$

where t_f is the *engine-on* time and D is the traveled distance between *home* and *destination*.

We can further reformulate the minimization Problem (2.4) by introducing a 3$^{\text{rd}}$ differential equation of $\dot{J}(t)$ to the system dynamics; the equivalent new problem becomes,

$$
\begin{aligned}
\underset{\substack{s(t),\, v(t),\, J(t),\, a(t) \\ 0 \leqslant t \leqslant t_f}}{\text{minimize}} \quad & J(t_f), \\
\text{subject to} \quad & \dot{s}(t) = v(t), \\
& \dot{v}(t) = a(t), \\
& \dot{J}(t) = \alpha + \beta_1 \, v(t) \, R_T(t) \, q(t) + \beta_2 \, \frac{M}{1000} \, v(t) \, a^2(t) \, p(t) \, q(t), \\
& s(0) = 0, \; s(t_f) = D, \; v(0) = 0, \; J(0) = 0,
\end{aligned}
\tag{2.6}
$$

where $R_T(t)$, $p(t)$ and $q(t)$ are identically defined in the Equation (2.5).

When discretizing the time horizon $[0, t_f]$ using a set of discrete time instances $t_i \in \{t_i\}_{i=0}^{N}$ ($t_0 = 0$ and $t_N = t_f$ at end points), we have a parameter

in the optimization problem in Problem (2.3) as,

$$\boldsymbol{p} = \left(\boldsymbol{s}_N^T, \boldsymbol{v}_N^T, \boldsymbol{J}_N^T, \boldsymbol{a}_N^T\right)^T,$$

where,

$$\begin{aligned}
\boldsymbol{s}_N &\triangleq \left(s(t_0), s(t_1), \cdots, s(t_i), \cdots, s(t_N)\right)^T, \\
\boldsymbol{v}_N &\triangleq \left(v(t_0), v(t_1), \cdots, v(t_i), \cdots, v(t_N)\right)^T, \\
\boldsymbol{J}_N &\triangleq \left(J(t_0), J(t_1), \cdots, J(t_i), \cdots, J(t_N)\right)^T, \\
\boldsymbol{a}_N &\triangleq \left(a(t_0), a(t_1), \cdots, a(t_i), \cdots, a(t_N)\right)^T.
\end{aligned}$$

Typically, in accordance to optimal control problems, we can call part of the parameter $\boldsymbol{p}$ as the "state", and the remaining part as "controls". For example, in Problem (2.6), the *state* can be considered as a tall vector $\boldsymbol{x}$,

$$\boldsymbol{x} = \left(\boldsymbol{s}_N^T, \boldsymbol{v}_N^T, \boldsymbol{J}_N^T\right)^T,$$

and *control* $\boldsymbol{u}$ as a vector $\boldsymbol{u} = \boldsymbol{a}_N$. The objective function is therefore the last component of the state vector (also the the last element of the 3$^{\text{rd}}$ group of components $\{J(t_i)\}_{i=0}^N$ in $\boldsymbol{x}$), i.e., $J(t_N) = J(t_f)$.

In Chapter 3, further details of the structure of equality constraints $\boldsymbol{g}(\boldsymbol{p}) = \boldsymbol{0}$ and inequality constraints $\boldsymbol{h}(\boldsymbol{p}) \leqslant \boldsymbol{0}$ will be discussed with incorporation of the Runge-Kutta method.

CHAPTER 3

OPTIMAL TRAJECTORY GENERATION

3.1 Euler Method in ODE

Consider an initial value problem (IVP) [25] as follows,

$$x' = \frac{\mathrm{d}x}{\mathrm{d}t} = f(t, x), \text{ with } x(t_0) = x_0. \tag{3.1}$$

For solving this problem, we can use the Euler Method or the tangent line method,

$$\begin{aligned} x_{n+1} &= x_n + h\, f(t_n, x_n) \\ &= x_n + h\, x'_n, \quad n = 0, 1, 2, \ldots \end{aligned} \tag{3.2}$$

where a uniform step size h is assumed between points t_0, t_1, $t_2, \ldots$, hence $t_{n+1} = t_n + h$. If the solution of Problem (3.1) is $x = \phi(t)$ and it is *smooth*, the first and second derivatives of $\phi(t)$ will be,

$$\begin{aligned} \phi'(t) &= f(t, \phi(t)), \\ \phi''(t) &= f_x(t, \phi(t)) + f_y(t, \phi(t))\, f(t, \phi(t)), \end{aligned}$$

A Taylor series of $\phi(t)$ around t_n up to 2^{nd} order gives an expansion with a remainder term,

$$\phi(t_n + h) = \phi(t_n) + \phi'(t_n)\, h + \frac{1}{2}\, \phi''(\bar{t}_n)\, h^2,$$

where $\bar{t}_n$ is a point in the interval $[t_n, t_n + h]$. Suppose at n-th step we have $x_n = \phi(t_n)$ then the error at point t_{n+1} is e_{n+1} and,

$$e_{n+1} = \phi(t_{n+1}) - x_{n+1} = \frac{1}{2}\phi''(\bar{t}_n)\, h^2. \tag{3.3}$$

We can see in the Equation (3.3) that for the Euler Method, the local computation error e_{n+1} has the order of h^2, if in the interval of interest we have a bounded second derivative of $\phi(t)$, i.e., $|\phi''(t)| \leqslant M, \forall\, t \in [a, b]$, where M is a constant and $a = t_0$, $b = t_{n+1}$ are end points of the interval. We can denote this by $e_i \propto h^2$ for any step, $i = 0, 1, \ldots, n+1$. A smaller step error requires a small step size h (or equivalently large number of points in $[a, b]$).

The Euler Method is simple to use when we implement the differential constraints such as the one in Problem (2.6). However it is also known that generally the Euler Method is not an *accurate* method as a result of the reduced number of steps in order to speed up computation. Table 3.1 shows the *unstable* behavior of Euler Method for Problem (2.6) when implemented for a scenario in Section 4.2.2.1. At this point, however, we don't have to delve into the details of Table 3.1. We only have to focus on the inconsistency in the first column of the table: for the same setting (especially the same number of steps N) except with different starting points required for calling nonlinear programming (NLP) solver `fmincon`, the minimum returned by `fmincon` is very different each time it runs. Therefore, it is necessary for us to turn to a more stable and more accurate method. The Runge-Kutta 4^{th} order method will be chosen to deal with the differential constraints in Problem (2.6).

3.2 Runge-Kutta 4^{th} Order Method in ODE

We again consider the IVP Problem (3.1). A Taylor series of $\phi(t)$ around t_n up to 4^{th} order gives an expansion,

$$x_{n+1} = x_n + x'_n\, h + \frac{x''_n}{2!}\, h^2 + \frac{x'''_n}{3!}\, h^3 + \frac{x^{(4)}_n}{4!}\, h^4 + \text{H.O.T.}(h^5).$$

Table 3.1: Fuel Consumption: Sweeping Results over Different Engine-on Time t_f's—Scenario in Section 4.2.2.1 with Euler Method

$\text{fuel}_{\min}$(ml)	N	N_s	t_f(s)	Time(s)
51.1972	216	1[a]	36	125.2862[b]
48.9710	216	1	36	107.6382
46.9427	216	1	36	113.5193
47.6133	228	1	38	176.5077
52.1434	228	1	38	60.1528
43.9654	228	1	38	228.8154
54.5306	240	1	40	61.4844
46.0729	240	1	40	135.4680
50.1064	240	1	40	53.6431
49.9283	252	1	42	83.1764
47.7905	252	1	42	80.9813
45.8448	252	1	42	86.7915
45.7251	264	1	44	105.4016
54.5566	264	1	44	210.8963
44.9184	264	1	44	174.7943

[a]We use $N_s = 1$ to ensure a fast sweep computation; if a finer trajectory is needed, one can increase N_s accordingly (typically $N_s = 3$).
[b]Matlab® R2013a Optimization Toolbox® `fmincon` was run on a Windows 7® installed HP® Z210 Workstation with a Quad core CPU Intel® Core® i5-2400; `matlabpool` was set to "4" to turn on parallel computing feature when approximating gradient used by `fmincon`.

The Runge-Kutta 4$^{\text{th}}$ order formula is given by,

$$x_{n+1} = x_n + h\left(\frac{y_{n_1} + 2\,y_{n_2} + 2\,y_{n_3} + y_{n_4}}{6}\right), \tag{3.4}$$

$$\begin{aligned}
\text{where,} \quad y_{n_1} &= f(t_n, x_n),\\
y_{n_2} &= f(t_n + \frac{1}{2}\,h, x_n + \frac{1}{2}\,h\,y_{n_1}),\\
y_{n_3} &= f(t_n + \frac{1}{2}\,h, x_n + \frac{1}{2}\,h\,y_{n_2}),\\
y_{n_4} &= f(t_n + h, x_n + h\,y_{n_3}).
\end{aligned}$$

The term $\left(\frac{y_{n_1} + 2\,y_{n_2} + 2\,y_{n_3} + y_{n_4}}{6}\right)$ is an "average" slope [25] in the interval $[t_n, t_{n+1}]$ when we compare Equation (3.4) to Equation (3.2). Similar to what has been done for the Euler Method in Section 3.1, we can show that the computation error of the Runge-Kutta 4$^{\text{th}}$ order method in the Equation (3.4) is proportional to h^5. We consider the Runge-Kutta method as a stable method and we will stick to Equation (3.4) when addressing the differential constraints in Problem (2.6).

3.3 Solving Optimization Problem with Parallel Shooting

Enright and Conway [26] proposed a direct transcription method to approximate optimal trajectories by discretization. It features a transcription of an optimal control problem with the explicit Runge-Kutta 4$^{\text{th}}$ order method to discretize the system dynamics.

Consider the dynamics equation of a general system,

$$x' = f(t, x, u),$$

with state vector $x(t)$ and control $u(t)$. We can discretize the time horizon with a uniform partition $[t_0, t_1, \ldots, t_N]$, where step size $h = t_i - t_{i-1}$, $i = 0, 1, \ldots, N$. At each mesh point t_i we have the state variables $x_i = x(t_i)$ and control variables $u_i = u(t_i)$, $i = 0, 1, \ldots, N$. There are also *intermediate* control variables $v_i = u(t_{i-1} + \frac{h}{2})$ at center points. By the Runge-Kutta method in Section 3.2, we integrate x_{i-1} one step forward from t_{i-1} to t_i

using control u_{i-1}, v_i and u_i,

$$\begin{aligned}
y_{i_1} &= x_{i-1} + \frac{h}{2} f(x_{i-1}, u_{i-1}), \\
y_{i_2} &= x_{i-1} + \frac{h}{2} f(y_{i_1}, v_i), \\
y_{i_3} &= x_{i-1} + h\, f(y_{i_2}, v_i), \\
y_{i_4} &= x_{i-1} + \frac{h}{6} \Big(f(x_{i-1}, u_{i-1}) + 2\, f(y_{i_1}, v_i) + 2\, f(y_{i_2}, v_i) + f(y_{i_3}, u_i) \Big).
\end{aligned} \tag{3.5}$$

After this one step of integration, y_{i_4} is a *propagated* state at mesh point t_i, by continuity and smoothness of the solution we require at t_i, the error between the propagated state y_{i_4} and the true state x_i be zero.

$$\Delta_i = y_{i_4} - x_i = 0, \;\; i = 1, 2, \ldots, N. \tag{3.6}$$

The derivation of Equation (3.5) can be found in [27].

3.3.1 Parallel Shooting Algorithm

Enright [27] further proposed parallel shooting for each interval $[t_{i-1}, t_i]$. Instead of propagating the state x_{i-1} from t_{i-1} to t_i by only one step, we can use the RK4 step N_s times in the interval. In that case, correspondingly, we need $(2\,N_s - 1)$ intermediate controls $v_{i_j} = u(t_{i-1} + j\,\frac{h}{2\,N_s})$, $j = 1, 2, \ldots, 2\,N_s - 1$. After N_s steps of integration, the resulting propagated state at t_i is denoted by x_i^*. Similar to the Equation (3.6), continuity and smoothness requires,

$$\Delta_i = x_i^* - x_i = 0, \;\; i = 1, 2, \ldots, N. \tag{3.7}$$

Figure 3.1 illustrates the parallel shooting scheme for $N_s = 1$. In Figure 3.1, the solid line is the trajectory of the state $x(t)$ while the dashed line represents the trajectory of the propagated state along which a start point x_{i-1} is taken via *one* RK4 step to the end of the interval $[t_{i-1}, t_i]$, with x_i^* as the resulting propagated state. The Parallel Shooting method allows us to use larger intervals (partitions) without losing accuracy and control resolution [27]. To use more RK4 steps to achieve accuracy and to use many intervals to control robustness, the Parallel Shooting guarantees both accuracy and robustness independently through the choice of number of subdivisions

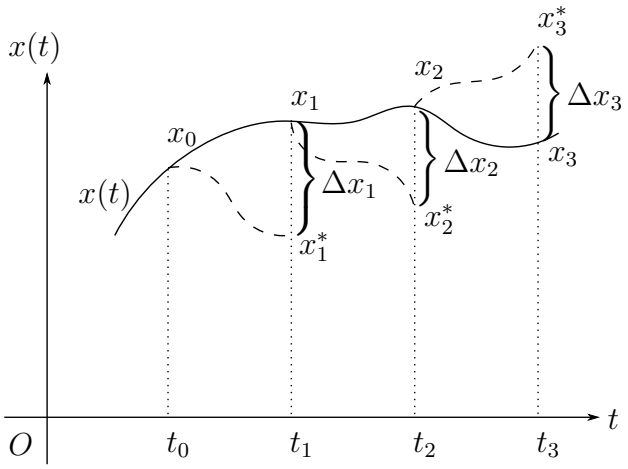


Figure 3.1: Parallel Shooting Method ($N_s = 1$)

(N_s) within $[t_{i-1}, t_i]$ and number of partitions of time horizon (N) for $[0, t_f]$ respectively.

3.3.2 A Benchmark Optimization Problem Using Parallel Shooting

Now we want to look at an example problem (Example 3.22 in [28]). We shall use this problem to sample test the transcription method in Section 3.3.1.

$$\begin{aligned} \underset{\substack{x(t),\,u(t)\\ 0\leqslant t\leqslant 1}}{\text{minimize}} \quad & \int_0^1 \frac{1}{2}\, u^2(t)\, \mathrm{d}t, \\ \text{subject to} \quad & \dot{x}(t) = -x(t) + u(t), \\ & x(0) = 1,\, x(1) = 0. \end{aligned} \tag{3.8}$$

To solve Problem (3.8), we follow a similar process to the problem reformulation process discussed in Section 2.3, Chapter 2. A reformulated problem

will be,

$$\begin{aligned} \underset{\substack{x(t),J(t),u(t)\\ 0\leqslant t\leqslant 1}}{\text{minimize}} \quad & J(1), \\ \text{subject to} \quad & \dot{x}(t) = -x(t) + u(t), \\ & \dot{J}(t) = \frac{1}{2}\,u^2(t), \\ & x(0) = 1,\ x(1) = 0,\ J(0) = 0. \end{aligned} \tag{3.9}$$

Analytically, the true solution to Problem (3.8) is,

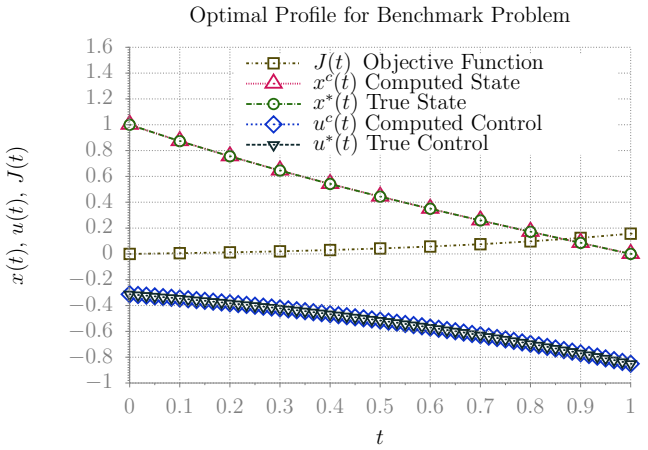


Figure 3.2: Comparison between Computed and Analytical Solutions for Problem (3.8)

$$\begin{aligned} u^*(t) &= -\frac{1}{\sinh(1)}\,\mathrm{e}^{t-1}, \\ x^*(t) &= \mathrm{e}^{-t} - \frac{1}{\mathrm{e}\sinh(1)}\sinh(t). \end{aligned}$$

Numerically, we can solve Problem (3.9) (Problem (3.8) equivalent) with *Piecewise Constant Control Parameterization* (PCCP) over N stages [28] or Direct Transcription with RK4 plus Parallel Shooting (DTRPS) as described in Section 3.3.1.

A comparison between the true solution to Problem (3.9) and the solution computed by NLP with the Parallel Shooting method is shown in Figure 3.2. For the state, control and objective function, the computed solution agrees very well with the analytical solution. We can get the minimum value of

Problem (3.8) by reading $J(1) = 0.1565$ in Figure 3.2. Therefore, we have the conclusion that the direct transcription method DTRPS is very effective in solving optimal control problems as shown in this example.

Table 3.2: Optimal Solutions by PCCP and DTRPS with $N = 2, 5, 10, 20$

N	$J(1)^a$	$J(1)^b$
2	0.1597	0.1565
5	0.1570	0.1565
10	0.1567	0.1565
20	0.1566	0.1565

[a]Solution by Piecewise Constant Control Parameterization.
[b]Solution by Direct Transcription with RK4 Steps and Parallel Shooting, $N_s = 3$.

A performance comparison between the PCCP and the DTRPS with different partition schemes is also shown in Table 3.2. It only confirms that with RK4 steps and parallel shooting, the direct transcription is fast and robust. Even with apparently significantly fewer number of partitions ($N = 2$), it still gives the correct local minimum with accuracy. This also implies the possibility that in order to speed up the computation without losing accuracy we can choose the DTRPS method with relatively fewer number of partitions.

3.4 Solving Optimization Problem of Fuel Consumption with Parallel Shooting

After we validated the parallel shooting code for the benchmark Problem (3.8), now we shall move on to extend the code for the fuel consumption problem of our interest—the Problem (2.6) in Section 2.3.

3.4.1 Programming with Parallel Shooting

We again assume a uniform partition of time horizon $[0, t_f]$ with N divisions. The nonlinear programming direct transcription of Problem (2.6) takes an

NLP parameter $\boldsymbol{p}$ defined as follows,

$$\boldsymbol{p} = \left[\underbrace{s_0, s_1, \ldots, s_N}_{\text{displacement } \boldsymbol{s}_N^T}, \overbrace{v_0, v_1 \ldots, v_N}^{\text{velocity } \boldsymbol{v}_N^T}, \underbrace{J_0, J_1, \ldots, J_N}_{\text{fuel use } \boldsymbol{J}_N^T}, \overbrace{a_0, a_1, \ldots, a_{2NN_s-1}, a_{2NN_s}}^{\text{[intermediate] controls } \boldsymbol{a}_{2NN_s}^T} \right]. \tag{3.10}$$

Furthermore, within each division $[t_{i-1}, t_i]$, $i = 1, 2, \ldots, N$, there are also N_s subdivisions. The number of *intermediate* controls in $[t_{i-1}, t_i]$ is $(2\,N_s - 1)$. Hence there are in total $2\,N\,N_s$ controls. We abandoned the notation v_{i_j}, $j = 1, 2, \ldots, 2\,N_s - 1$ for intermediate controls used in Section 3.3.1. However we can use subscripts (or indices) to differentiate the intermediate controls v_{i_j} within each division $[t_{i-1}, t_i]$ from controls at end points of the division $a(t_{i-1})$ and $a(t_i)$. The NLP parameter $\boldsymbol{p}$ is therefore decomposed as,

- $(N + 1)$ components for displacement s_i's, $i = 0, 1, \ldots, N$
- $(N + 1)$ components for velocity v_i's, $i = 0, 1, \ldots, N$
- $(N + 1)$ components for objective function J_i's , $i = 0, 1, \ldots, N$
- $(2\,N\,N_s + 1)$ components for controls and intermediate controls a_j's, $j = 0, 1, \ldots, 2\,N\,N_s$

We can group $\boldsymbol{p}$'s components into state and controls—$\boldsymbol{p}_{\text{state}}$ and $\boldsymbol{p}_{\text{control}}$, i.e., $\boldsymbol{p}_{\text{state}} = \boldsymbol{x} = \left(\boldsymbol{s}_N^T, \boldsymbol{v}_N^T, \boldsymbol{J}_N^T\right)^T$ and $\boldsymbol{p}_{\text{control}} = \boldsymbol{u} = \boldsymbol{a}_{2NN_s}$. In Problem (2.6), the equality constraints $\boldsymbol{g}(\boldsymbol{p}) = \boldsymbol{0}$ follows the Equation (3.7) where we already discussed the error of the propagated state x_i^* using the Runge-Kutta 4^{th} order method. The inequality constraints $\boldsymbol{h}(\boldsymbol{p}) \leqslant \boldsymbol{0}$ are formed by the upper and lower bounds on displacement, velocity, and acceleration. When travel schemes are considered where a car tries to avoid a stop due to traffic light signals, inequalities such as $s(t_{red}) \geqslant D_{light}$ or $s(t_{green}) \leqslant D_{light}$ also enter $\boldsymbol{h}(\boldsymbol{p}) \leqslant \boldsymbol{0}$. In Section 4.2, we will give a detailed discussion of situations with lights.

3.4.2 NLP Solver: Calling `fmincon`

We prepare parameter $\boldsymbol{p}$ before calling `fmincon` [29],

```
pmin = fmincon(@ObjFun,p0,A,b,Aeq,beq,lb,ub,@NonLinCon,options)
```

where,

- `ObjFun` (scalar), the objective function in an optimization problem; the minimum of which is of our interest
- `p0`, a start point[1] (vector) required to run `fmincon`
- `A` (matrix) and `b` (vector), where we can code in the linear inequality constraints `A*p < b`; use `[]` if none
- `Aeq` (matrix) and `beq` (vector), the equality constraints `Aeq*p = beq`; use `[]` if none
- `lb` and `ub`, the lower and upper bounds on `p`
- `NonLinCon` (vector), a function returning the nonlinear constraints. Two output arguments are: `c(p)` for the nonlinear inequality constraints[2] `c(p) < 0` and `ceq(p)` for the nonlinear equality constraints `ceq(p) = 0`; use `[]` if none
- `options`, the settings for calling `fmincon`

Chapter 4 details the setting for solving Problem (2.6) with nonlinear programming DTRPS and the results of simulation.

[1] We use the Direct Transcription with RK4 steps and Parallel Shooting (DTRPS) to implement our code. It is a *stable* method thus, the results are showing great consistency even if `p0` is a random point within bounds. But for comparison, we kept using one of the following three points: `p0 = [0,0,...,0]'`, `p0 = [1,1,...,1]'` and `p0 = [-1,1,...]'` consistently in Chapter 4.

[2] The `c(p)` output argument is very useful when we implement the inequality constraints such as $s(t_{red}) \geqslant D_{light}$. We simply translate the constraint, in this example, to `c = DL - p(floor(tR/tf*N));` in MATLAB® code. The parameter `p` follows the definition in Equation (3.10).

CHAPTER 4

OPTIMAL TRAJECTORY GENERATION—CASE STUDIES

In this chapter, we will look at specific cases where a car is supposed to travel between *home* and *destination*. We will illustrate how the car is traveling according to the trajectories generated by our toolkit.

Here we look at three different cases:

- *Travel without traffic lights*, where the car is traveling alone between home and destination without other cars on the road; also there will be no lights during the travel
- *Travel with one light*, where there is only *one* traffic light during the travel; and the location and scheduling of the light is known
- *Travel with two lights*, where there are *two* traffic lights during the travel; and the location and scheduling of the two lights is also known

A general setting for all cases is as follows:

- We consider a trip which covers a distance between home and destination of 400 (m)
- There is a lower bound as well as an upper bound on travel velocity, i.e., $0 \leqslant v(t) \leqslant v_{\max}$, where $v_{\max} = 25$ (m/s) or about 56 (mph)
- There is also a lower bound as well as an upper bound on the car's acceleration, i.e., $a_{\min} \leqslant a(t) \leqslant a_{\max}$, where $a_{\min} = -2$ (m/s^2) and $a_{\min} = 5$ (m/s^2)
- A door-to-door trip is considered, where $v(0) = v(t_f) = 0$ (m/s)

4.1 Travel without Traffic Lights

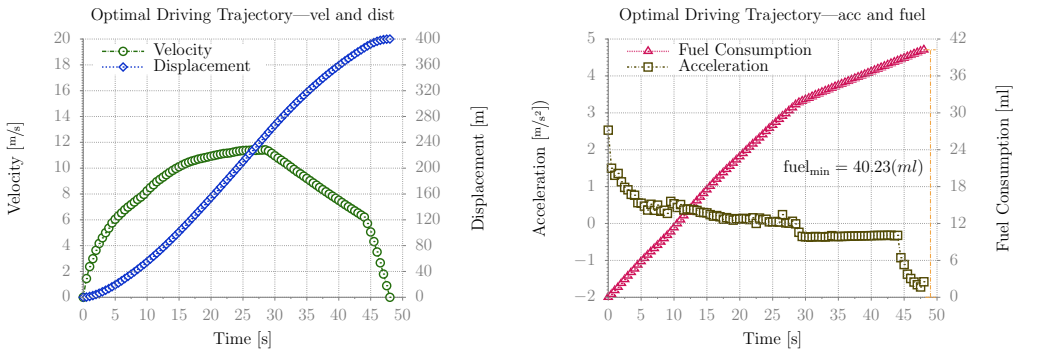
In the case of a trip where no traffic or any signals are taken into account, the trip only has to satisfy all the constraints stated at the beginning of Chapter 4. For a fixed engine-on time t_f, our toolkit gives an optimal profile of a trip where the automobile's engine is active for t_f seconds. In order to calculate the minimal fuel consumption, we need to do a "sweep" over *all* possible t_f's. Results for a sweep over $t_f = 30, 32, \ldots, 60$ (s) for the travel without lights are shown in Table 4.1 below. The first column lists the minimum fuel consumption for each t_f. We can pick the lowest 40.23 (ml) with corresponding engine-on time $t_f = 48$ (s) from the first and the second to last column. The $t_f = 48$ (s) case gives the optimal trajectory for the travel free of traffic lights as the minimum fuel consumption and corresponding engine-on time are highlighted in red in Table 4.1. Further visualization of this optimal profile of the travel with $t_f = 48$ (s) is shown in Figure 4.1.

Table 4.1: Fuel Consumption: Sweeping Results over Different Engine-on Time t_f's—Travel without Lights

$\text{fuel}_{\min}$(ml)	N	N_s	t_f(s)	Time(s)
60.7367	60	1[a]	30	122.6278[b]
53.4446	64	1	32	124.6964
48.9543	68	1	34	395.4807
45.6435	72	1	36	309.5363
48.7230	76	1	38	169.0759
43.7973	80	1	40	235.1002
41.3717	84	1	42	163.8752
40.5751	88	1	44	580.4537
40.9545	92	1	46	461.1145
40.2340	96	1	48	567.4311
40.2519	100	1	50	372.5695
40.4851	104	1	52	698.1033
40.9452	108	1	54	253.4016
41.1607	112	1	56	1150.3873

[a]We use $N_s = 1$ to ensure a fast sweep computation; if a finer trajectory is needed, one can increase N_s accordingly (typically $N_s = 3$).
[b]Matlab® R2013a Optimization Toolbox® `fmincon` was run on a Windows 7® installed HP® Z210 Workstation with a Quad core CPU Intel® Core® i5-2400; `matlabpool` was set to "4" to turn on parallel computing feature when approximating gradient used by `fmincon`.

Figure 4.1a shows that an optimal trajectory requires that the car acceler-



(a) Velocity and Displacement (b) Acceleration and Fuel Consumption

Figure 4.1: Optimal Driving Trajectory—Travel without Lights

ates to a speed (about $v = 12$ (m/s)) then at around the middle of the journey (around $t = 28$ (s)) it starts gliding. The gliding process interestingly contains two almost straight-line segments which means the car decelerates with two different constant braking force respectively as shown in Figure 4.1b. We also noticed that in this optimal profile, both bounds on velocity ($v_{\max}$) and acceleration ($a_{\min}$ and $a_{\max}$) are *not* active.

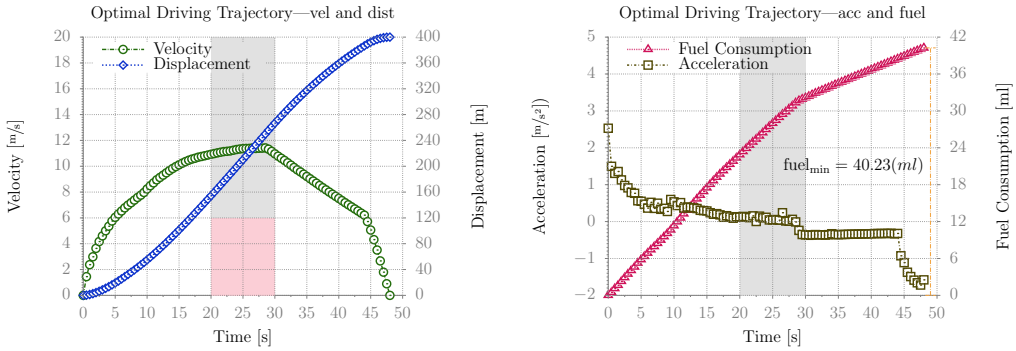
4.2 Travel with Traffic Lights

Now we will add lights information in addition to the setting in the previous Section 4.1. Here we first extend the case free of traffic lights to a case where one light is incorporated into the base setting; then we shall extend the case of one light into a new case with two lights.

4.2.1 A Case with One Light

Before we choose a case to study in this section, let's look at Figure 4.1a again. We add a light at a particular position D_{light} (Position of a Light) and we know it turns red at time instance t_{red}. Using displacement trajectory in Figure 4.1a as a base frame, if we study a case where a light which is located at a distance *within* the distance a car can cover during a free-of-traffic-light travel at time t_{red}, i.e., $s(t_{red}) \geqslant D_{light}$, then basically no matter what the time scheduling of the light is, this newly added light will not affect the old optimal trajectory in the new case of travel with one light, because when the

light turns red the car has already passed it. In that sense, the new optimal trajectory will be looking exactly the same as the old optimal trajectory.



(a) Velocity and Displacement (b) Acceleration and Fuel Consumption

Figure 4.2: Optimal Driving Trajectory—Travel with One Light; Light Ignored

In Figure 4.2 the region highlighted with red indicates that one light has been added accordingly at location $D_{light} = 120$ (m), and the scheduling of the light is that it turns red at $t_{red} = 20$ (s) and it turns green again at $t_{green} = 30$ (s), as shown in the highlighted gray area. From Figure 4.2a we know that at $t_{red} = 20$ (s) the car has passed the light at D_{light}. The light has been ignored. Therefore we shall study a case where the newly introduced light will force the shape of the old optimal trajectory to change. Hence we have the setting for the added one light as follows,

- There is a traffic light ahead at location $D_{light} = 200$ (m)
- The scheduling of this light is that it turns red at time instance $t_{red} = 20$ (s) and it turns green again at time instance $t_{green} = 30$ (s)

When we investigate optimal non-stop[3] trajectories, we have two options: we can either command the car to rush to the light to pass it before it turns red; or we can drive slowly when the light is red in front but when the car reaches location D_{light} it turns green again.

[3]If the car has to stop during the travel at the location of the light, i.e., it reaches D_{light} with $t \geqslant t_{red}$. This situation would be examined by dividing the entire trip into two door-to-door trips using travel without light in Section 4.1. The total fuel consumption will be the sum of consumption of two subtrips plus fuel consumption during idle time at the red light. It turns out generally non-stop trip is preferable in terms of either fuel consumption or travel time.

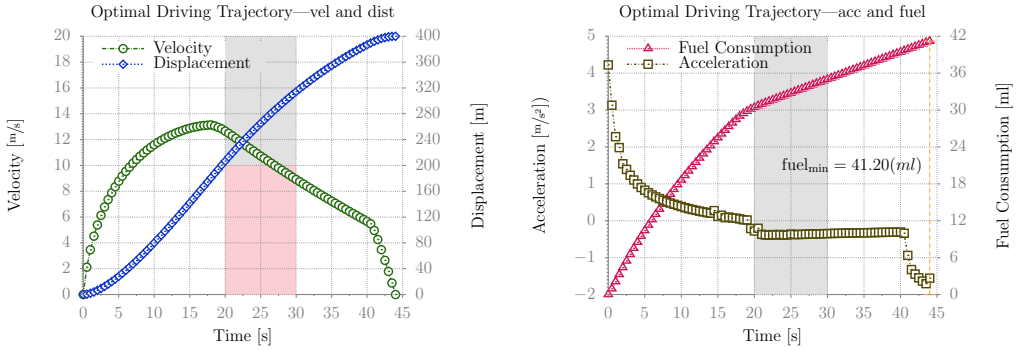
4.2.1.1 Rush to the Light to Avoid Red Signal

Based on the setting of the added light in Section 4.2.1, first we have a look at the optimal trajectory for the avoid-red-light case.

Table 4.2: Fuel Consumption: Sweeping Results over Different Engine-on Time t_f's—Travel with One Light; Avoid Red Light

$\text{fuel}_{\min}$(ml)	N	N_s	t_f(s)[c]	Time(s)
42.0536	80	1[a]	40	146.1700[b]
41.8259	82	1	41	171.2141
41.3837	84	1	42	183.7879
41.2489	86	1	43	176.4315
41.2051	88	1	44	235.4563
41.4759	90	1	45	208.7561
41.7669	92	1	46	191.7564
41.6982	94	1	47	206.2431
41.9370	96	1	48	346.9232
42.2005	98	1	49	409.0491
43.0551	100	1	50	367.0267

[a]Same as in Table 4.1.
[b]Same as in Table 4.1.
[c]$t_f = 40, 41, \ldots, 55$ (s) were tested. Here only a truncated table is shown.



(a) Velocity and Displacement (b) Acceleration and Fuel Consumption

Figure 4.3: Optimal Driving Trajectory—Travel with One Light; Avoid Red Light

Figure 4.3a shows that the car successfully avoids the red light by reaching the location of the light $D_{light} = 200$ (m) before the light turns red at $t_{red} = 20$ (s), i.e., $s(t_{red}) \geqslant D_{light}$. Compared with the case when there is no light, the car accelerates to a higher speed (about $v = 13$ (m/s)) and acceleration

completes earlier (around $t = 16$ (s)) than what is shown in Figure 4.1a. Also gliding process starts earlier as well but it is still composed of two piecewise almost linear segments.

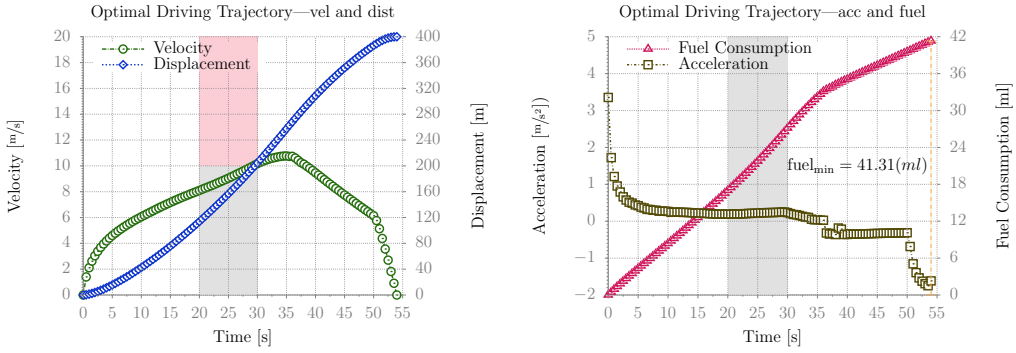
4.2.1.2 Drive Slowly towards the Light to Wait for It to Turn Green

Next we look at the optimal trajectory for the wait-for-green case.

Table 4.3: Fuel Consumption: Sweeping Results over Different Engine-on Time t_f's—Travel with One Light; Wait for Green Light

$\text{fuel}_{\min}$(ml)	N	N_s	t_f(s)[c]	Time(s)
43.0206	100	1[a]	50	315.5740[b]
42.3494	102	1	51	437.1227
41.7930	104	1	52	556.5442
41.6137	106	1	53	360.3817
41.3117	108	1	54	302.4756
59.0702	110	1	55	705.2406
41.4989	112	1	56	560.3283
42.2838	114	1	57	732.7435
41.7950	116	1	58	588.1486
42.0573	118	1	59	342.2440
42.1794	120	1	60	772.1389

[a]Same as in Table 4.1.
[b]Same as in Table 4.1.
[c]$t_f = 45, 46, \ldots, 60$ (s) were tested. Here only a truncated table is shown.



(a) Velocity and Displacement (b) Acceleration and Fuel Consumption

Figure 4.4: Optimal Driving Trajectory—Travel with One Light; Wait for Green Light

Figure 4.4 shows that the car slowly (relatively) reaches $D_{light} = 200$ (m) only after the time $t_{green} = 30$ (s), i.e., $s(t_{green}) \leqslant D_{light}$. Compared with the case when there is no light, the car accelerates to a lower speed (about $v = 11$ (m/s)) and acceleration completes only after the car passes the traffic light (around $t = 35$ (s)). Soon after the acceleration is completed, gliding process starts; it is still composed of two piecewise almost linear segments as shown in Figure 4.4a.

4.2.1.3 Best Driving Strategy for Travel with One Light

In summary, from what we have seen in Figure 4.3 and Figure 4.4, rushing to the red light is burning *slightly* less fuel than slowly approaching the light waiting for the light to turn green. Figure 4.3 shows the optimal trajectory for the travel with one light under the setting in Section 4.2.1. But we have to further comment on the process of choosing the best branch of the two. The computation result of the optimal trajectory is highly depending on the setting of the light. For instance, what Figure 4.3 shows is valid specifically for given light setting in Section 4.2.1. In other cases, slowly approaching the traffic light while waiting it to turn green *might* be a better choice.

4.2.2 A Case with Two Lights

Except for the general setting mentioned at the beginning of this chapter, and the setting used in the previous Section 4.2.1 where there is one light, we further impose the following setting for the second traffic light,

- There is a second traffic light ahead at location $D_{light_2} = 350$ (m)
- The scheduling of this light is that it turns red at time instance $t_{red_2} = 33$ (s) and it turns green again at time instance $t_{green_2} = 38$ (s)

Further we slightly modify the setting of the first light in Section 4.2.1 as follows,

- The first traffic light is still located at $D_{light_1} = 200$ (m)
- But scheduling of this light is that it turns red at time instance $t_{red_1} = 20$ (s) and it turns green again at time instance $t_{green_1} = 25$ (s)

We shrink the red-on time for the first light because we imposed an upper bound on travel velocity and there should be a time interval large enough between the scheduling of the two lights such that the car can possibly cover the distance between the two lights ($D_{light_2} - D_{light_1}$) within that time interval.

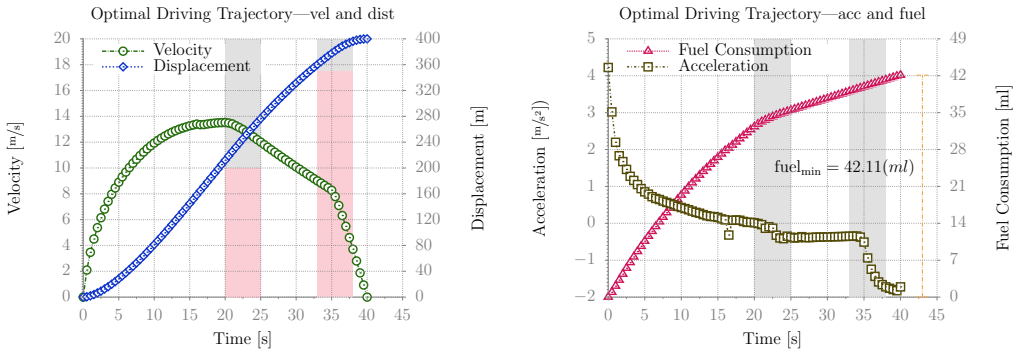
Now we increase the number of traffic lights from *one* to *two*. Like what we did already when we extended the case where there are no lights to the case where there is one light, we add one more constraint on time instances according to the scheduling of the second traffic light. Following a similar process and philosophy, there are *four* different combinations,

- To rush to the first light before it turns red, further rush to the second light before it turns red as well, i.e., $s(t_{red_1}) \geqslant D_{light_1}$ and $s(t_{red_2}) \geqslant D_{light_2}$
- To rush to the first light before it turns red, but wait for the second light to turn green, i.e., $s(t_{red_1}) \geqslant D_{light_1}$ and $s(t_{green_2}) \leqslant D_{light_2}$
- To wait for the first light to turn green, but rush to the second light before it turns red, i.e., $s(t_{green_1}) \leqslant D_{light_1}$ and $s(t_{red_2}) \geqslant D_{light_2}$
- To wait for the first light to turn green, further wait for the second light to turn green as well, i.e., $s(t_{green_1}) \leqslant D_{light_1}$ and $s(t_{green_2}) \leqslant D_{light_2}$

We shall look at individual cases one by one before we draw a conclusion which is the best for fuel economy under the setting at the beginning of Section 4.2.2.

4.2.2.1 Rush to the 1st Light to Avoid Red Signal and Rush to the 2nd Light to Avoid Red Signal

As shown in Figure 4.5, the car passes two traffic lights sequentially before the lights turn red. The displacement trajectory will not intersect the two highlighted red regions except the top left corners in Figure 4.5a. From Figure 4.5b we can see that the lowest fuel consumption is about 42.11 (ml) and the entire trip takes 40 seconds. Figure 4.5a also shows that after burning fuel in order to avoid the first red light, the car starts gliding soon after it reaches the first traffic light. Still, the gliding process consists of two almost linear segments.



(a) Velocity and Displacement (b) Acceleration and Fuel Consumption

Figure 4.5: Optimal Driving Trajectory—Travel with Two Lights; Avoid 1st Red Light and Avoid 2nd Red Light

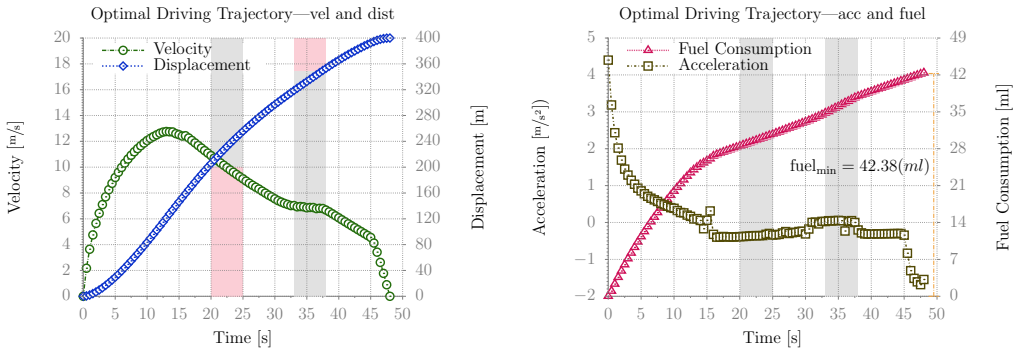
Table 4.4: Fuel Consumption: Sweeping Results over Different Engine-on Time t_f's—Travel with Two Lights; Avoid 1st Red Light and Avoid 2nd Red Light

$\text{fuel}_{\min}$(ml)	N	N_s	t_f(s)[c]	Time(s)
48.7395	68	1[a]	34	91.2869[b]
45.5802	72	1	36	106.0541
43.4489	76	1	38	126.0511
42.1135	80	1	40	166.0160
42.2807	84	1	42	132.0225
43.1279	88	1	44	177.4607
44.0242	92	1	46	199.0814
44.9233	96	1	48	201.9882
45.9295	100	1	50	209.1434
46.7109	104	1	52	271.6029
47.5838	108	1	54	265.8051
48.5430	112	1	56	292.6747
49.9948	116	1	58	343.3167
50.7831	120	1	60	248.6522
51.6356	124	1	62	288.6187
52.5740	128	1	64	372.7401

[a]Same as in Table 4.1.
[b]Same as in Table 4.1.
[c]$t_f = 16, 18, \ldots, 64$ (s) were tested. Here only a truncated table is shown.

4.2.2.2 Rush to the 1st Light to Avoid Red Signal but Wait for the 2nd Light to Turn Green

Figure 4.6 shows how the car avoids the first red signal and waits for the second light to turn green. In Figure 4.6a, the displacement trajectory snaps to the upper left corner of the first red block while to the lower right corner of the second red block, meaning the constraints imposed by the light signals are active. The lowest fuel consumption in this case is about 42.38 (ml) as shown in Table 4.5, only a slightly higher than that in Section 4.2.2.1. Gliding process in this driving strategy takes longer so the car accelerates to $v = 13$ (m/s) then starts gliding. When the car approaches the second light, it accelerates slightly again in order to keep the gliding to cover longer distance as shown in Figure 4.6b.



(a) Velocity and Displacement (b) Acceleration and Fuel Consumption

Figure 4.6: Optimal Driving Trajectory—Travel with Two Lights; Avoid 1st Red Light but Wait for 2nd Green Light

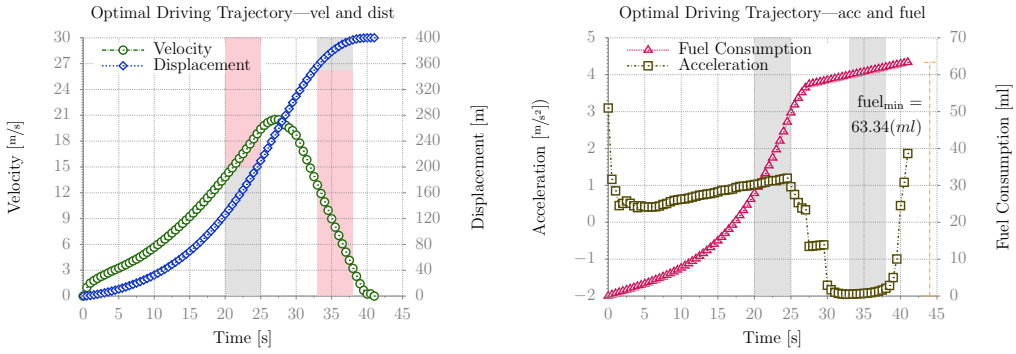
4.2.2.3 Wait for the 1st Light to Turn Green but Rush to the 2nd Light to Avoid Red Signal

Figure 4.7 shows a relatively aggressive driving strategy to complete a non-stop trip. For the first part of the trip between the start point and the first light, the car moves slowly towards the traffic light, but in order to cover the distance between the two traffic lights ($D_{light_2} - D_{light_1}$) within a short period of time ($t_{red_2} - t_{green_1}$), it accelerates to a high speed $v = 21$ (m/s) to accumulate enough energy to ensure the gliding can help the car to avoid the second red signal. The gliding process starts between the time instance when the first traffic light turns green (t_{green_1}) and time instance when the

Table 4.5: Fuel Consumption: Sweeping Results over Different Engine-on Time t_f's—Travel with Two Lights; Avoid 1st Red Light but Wait for 2nd Green Light

$\text{fuel}_{\min}$(ml)	N	N_s	t_f(s)	Time(s)
91.0159	80	1[a]	40	28.3909[b]
94.2492	84	1	42	35.6820
103.4679	88	1	44	42.1281
43.0630	92	1	46	223.5199
42.3812	96	1	48	252.7874
42.8454	100	1	50	273.5567
43.3891	104	1	52	342.5170
44.8263	108	1	54	194.7117
44.9493	112	1	56	417.3029
46.0761	116	1	58	378.8630
48.0413	120	1	60	539.4281
47.5687	124	1	62	611.8332
49.2236	128	1	64	444.9139

[a]Same as in Table 4.1.
[b]Same as in Table 4.1.



(a) Velocity and Displacement (b) Acceleration and Fuel Consumption

Figure 4.7: Optimal Driving Trajectory—Travel with Two Lights; Wait for 1st Green Light but Avoid 2nd Red Light

second traffic light turns red (t_{red_2}). Figure 4.7b also shows the car brakes hard between the lights and accelerates hard when it approaches destination. This is not a good policy in terms of drivability.

Table 4.6: Fuel Consumption: Sweeping Results over Different Engine-on Time t_f's—Travel with Two Lights; Wait for 1st Green Light but Avoid 2nd Red Light

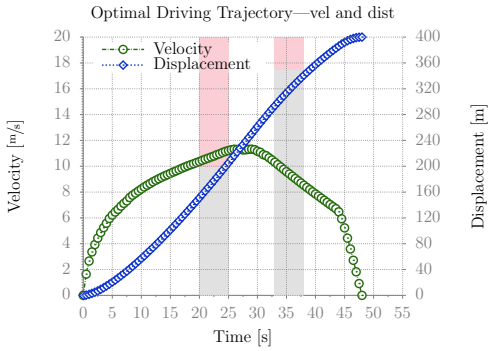
fuel_{min}(ml)	N	N_s	t_f(s)	Time(s)
110.6689	66	1[a]	33	123.1945[b]
87.2444	70	1	35	41.4568
147.9987	74	1	37	69.0607
146.3072	78	1	39	46.2471
63.3473	82	1	41	126.1665
63.9984	86	1	43	198.1851
65.0096	90	1	45	180.3143
65.5599	94	1	47	167.8898
66.6508	98	1	49	276.2946
67.5319	102	1	51	187.9627
68.3256	106	1	53	307.4104
69.0893	110	1	55	377.3062
71.2159	114	1	57	238.1883
70.8230	118	1	59	445.7694
73.3505	122	1	61	316.5715
72.1159	126	1	63	513.3052
75.3544	130	1	65	344.7921

[a]Same as in Table 4.1.
[b]Same as in Table 4.1.

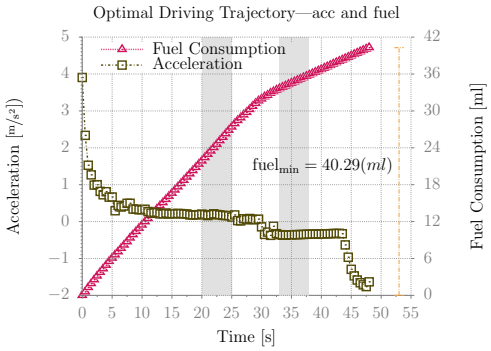
4.2.2.4 Wait for the 1st Light to Turn Green and Wait for the 2nd Light to Turn Green

Figure 4.8 shows an optimal driving policy with which the car can move slowly to both traffic lights to wait for both lights to turn green. In Figure 4.8a, displacement trajectory snaps to the lower corner of the first red highlighted region however it does not intersect (slightly off in Figure 4.8a) the second red highlighted region at all. After the car starts gliding around $t = 27$ (s), it ignores the scheduling of the second traffic light. Figure 4.8b shows that for most of the time during the trip the acceleration is flat except the starting part and when it is approaching the destination. This wait-

for-both-green light policy also gives the lowest fuel consumption 40.29 (ml) among other travel with two-light cases.



(a) Velocity and Displacement



(b) Acceleration and Fuel Consumption

Figure 4.8: Optimal Driving Trajectory—Travel with Two Lights; Wait for 1st Green Light and Wait for 2nd Green Light

4.2.2.5 Best Driving Strategy for Travel with Two Lights

In summary, we have to pick a driving policy with the lowest fuel consumption based on discussion in Section 4.2.2.1 through Section 4.2.2.4. For the setting at the beginning of Section 4.2.2, to slowly approach both traffic lights while waiting the lights to turn green will guarantee the minimum fuel consumption (Figure 4.8b), and the entire trip takes 48 (s) as seen in Table 4.7. The best driving policy for travel with two lights is therefore the one discussed in Section 4.2.2.4.

4.3 Summary of Different Cases

In this chapter, we have shown the results by our toolkit for travel cases with *no traffic lights*, with *one traffic light* and with *two traffic lights* for a door-to-door trip. The travel with no lights case in Section 4.1 serves as the base for the latter two cases in Section 4.2.1 and Section 4.2.2 respectively. In general, one can always follow the trajectory in the no-light case to get the lowest possible minimum fuel consumption. When taking traffic lights into account, if one wants to avoid the red signal by rushing to the traffic light however, when the light turns red, in the no-light case displacement trajectory covers

Table 4.7: Fuel Consumption: Sweeping Results over Different Engine-on Time t_f's—Travel with Two Lights; Wait for 1st Green Light and Wait for 2nd Green Light

$\text{fuel}_{\min}$(ml)	N	N_s	t_f(s)	Time(s)
105.9432	80	1[a]	40	233.5536[b]
126.8588	84	1	42	179.6956
113.1614	88	1	44	231.2357
40.8748	92	1	46	348.1562
40.2939	96	1	48	545.4418
40.3434	100	1	50	241.3485
40.4181	104	1	52	362.0106
41.0715	108	1	54	304.7185
41.2561	112	1	56	1122.1261
41.6386	116	1	58	384.7032
42.1714	120	1	60	881.4704
42.9613	124	1	62	783.8639
43.3636	128	1	64	932.3267

[a]Same as in Table 4.1.
[b]Same as in Table 4.1.

a longer range than the location of the traffic light (Section 4.2.1), one can simply ignore the light. The travel with lights case will be reduced to travel with no lights since lights are not active during the computation. Similarly, if one wants to wait for the traffic light to turn green, the scheduling of green light will only be a concern if at the time the light turns green the car could possibly go beyond the location of the light that is relatively close to the destination (Otherwise, as seen in Section 4.2.2.4, the second light is not active).

Figure 4.9 shows an overlaid graph of instantaneous fuel consumption rates (in Miles Per Gallon or MPG) for the best driving policies in each case discussed in this chapter. The dark gray region indicates there is one traffic light which will turn red between $t = 20$ (s) and $t = 30$ (s). Similarly, the two light gray blocks indicate that the two traffic lights will stay red during two time slots 20–25 (s) and 33–38 (s) respectively. We can see in Figure 4.9 that for travel with one light case, the instantaneous MPG jumps dramatically around the time when the light turns red. It implies that the car has accelerated enough then it will soon start gliding with a high MPG around 60 (MPG). For travel with two lights case, instantaneous fuel consump-

tion rate does not follow a similar trend in the one light case but wriggles through the time interval between two adjacent light signals. The *average* MPG's for travel free-of-light, with one light and with two lights in Figure 4.9 are 23.38[4] (MPG), 22.83 (MPG) and 23.34 (MPG) respectively.

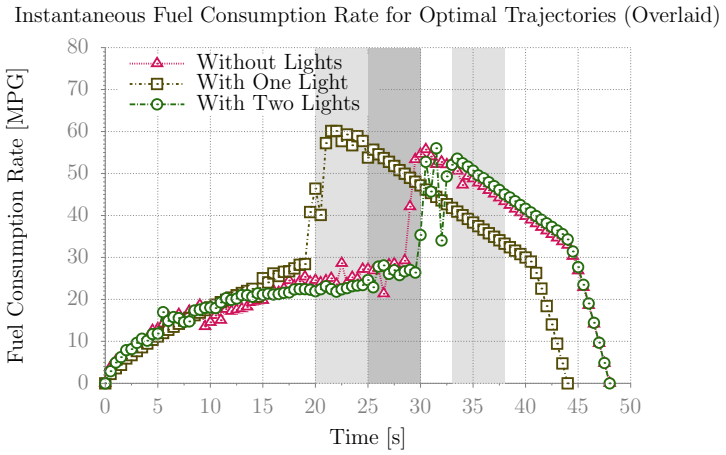


Figure 4.9: Instantaneous Fuel Consumption Rates in Case Studies

Lastly, though only door-to-door trips have been considered in the examples, for other cases with specified non-zero initial conditions and final conditions, the toolkit can also calculate the best driving policies of minimum fuel consumption. For example, $v(0) = v(t_f) = 0$ (m/s) has been implemented by forcing the upper bound and lower bound on $v(0)$ and $v(t_f)$ to be zero. If instead we use $v_{\min}$ and $v_{\max}$ on $v(0)$ or $v(t_f)$ as the lower bound and upper bound respectively in the computation, we can get the optimal trajectory for driving with *free* initial and final velocity. Of course, in practice, we are more interested in cases where the initial velocity of the trip is zero, i.e., $v(0) = 0$.

[4]The *best* MPG appears relatively low due to the only available *old* instantaneous model by [23] and the fact that the MPG was studied for urban driving in this chapter.

CHAPTER 5

A SHORT GUIDE TO MATLAB CODE USED FOR TRAJECTORY GENERATION

5.1 Overview

This chapter is dedicated to the description of code parameters that have been used in the case studies in Chapter 4. The case with two traffic lights is more general than the case with no lights and the case with one light, therefore, we only document the code for Section 4.2.2. The code for the two skipped cases can be treated as simplified versions of the two-light case.

The entire toolkit contains six different `*.m` files,

- `RunMe.m`, this file serves as an interface between the user and the toolkit; the travel distance, engine-on time, light location and scheduling etc., could be specified in this file
- `FuelInit.m`, this file is the main file which defines the NLP parameter `p` and dictates the settings for calling `fmincon` with a start point; it will call `fmincon` at the end
- `FuelObj.m`, this file defines the objective function in Equation (2.6)
- `FuelConstr.m`, this file defines the nonlinear equality constraints "Runge-Kutta defects" [26] described in the Equation (3.6) and the nonlinear inequality constraints specified at the beginning of Section 4.2.2
- `FuelOdeRHS.m`, this file defines the Right Hand Side (RHS) of the differential constraints in Problem (2.6)
- `RKProp.m`, this file implements the Runge-Kutta four stage (RK4) propagation in a single interval $[t_{i-1}, t_i]$ as described in the Equation (3.5)

5.2 `RunMe.m`

5.2.1 On Entering Parameters

- `D`, traveled distance of a trip between home and destination
- `Ti`[5], specified engine-on time t_f
- `Ni`, number of partitions of the time horizon $[0, t_f]$
- `Nsi`, number of subdivisions within the interval $[t_{i-1}, t_i]$
- `DL1` and `DL2`, location of the traffic lights; the first light is located at `DL1`, the second light at `DL2`
- `tR1` and `tR2`, the red light scheduling; the first light turns red at `tR1`, the second light turns red at `tR2`
- `tG1` and `tG2`, the green light scheduling; the first light turns green at `tG1`, the second light turns green at `tG2`
- `v_max`, the upper bound on velocity
- `vf_upp` and `vf_low`, the upper and lower bounds on final velocity
- `v0_upp` and `v0_low`, the upper and lower bounds on initial velocity
- `a_max` and `a_min`, the upper and lower bounds on acceleration

5.2.2 On Exiting Parameters

There are not any output arguments in `RunMe.m`. But after the code is run, it outputs four data files `data_file_vf0_XXX_XXX.txt`, where `XXX_XXX` is a combination of `{red1,green1}` and `{red2, green2}`. For instance, `XXX_XXX` could be `red1_green2`. One data file records similar data shown in any table of sweeping results in Section 4.2.2.

[5]Using `Ti` as t_f is unfortunate. `Ti` was originally designed as bound on t_f for t_f-free case however the code didn't work but the notation was kept.

5.3 `FuelInit.m`

5.3.1 On Entering Parameters

- All entering parameters in Section 5.2.1
- `GRSi`, a case selector to switch to a combination of `{red1,green1}` and `{red2, green2}`. `GRSi` ranges from 1 to 4 (four different combinations)
- `cnt`, a dummy counter variable

5.3.2 On Exiting Parameters

All exiting parameters will be stored in the data file as described in Section 5.2.2.

- `fuel_min`, the minimum value of fuel consumption in the optimization Problem (2.6)
- `N`, same as `Ni` in the current run
- `Ns`, same as `Nsi` in the current run
- `tf`, same as `Ti` in the current run
- `el_time`, elapsed time for current run in CPU time

5.4 `FuelObj.m`

5.4.1 On Entering Parameters

- `p`, the NLP parameter defined in the Equation (3.10)
- `N`, same as in Section 5.3.2

5.4.2 On Exiting Parameters

- `J`, a dummy variable for the objective function

5.5 `FuelConstr.m`

5.5.1 On Entering Parameters

- `p`, the NLP parameter defined in the Equation (3.10)
- `GRS`, same as `GRSi` in the current run

All other entering parameters are already defined in Section 5.2.1

5.5.2 On Exiting Parameters

- `ceq`, the nonlinear equality constraints for the Runge-Kutta defects defined in the Equation (3.6)
- `c`, the nonlinear inequality constraints for implementing the schemes of avoiding traffic light signals such as $s(t_{red}) \geqslant D_{light}$, as described in the footnote in Section 3.4.2.

5.6 `FuelOdeRHS.m`

This is a function which calculates the derivative of the state variable based on the current state $\Big(s(t_i), v(t_i), J(t_i)\Big)^T$ value and control value $a(t_i)$.

5.6.1 On Entering Parameters

- `x`, a 3×1 state vector representing $\Big(s(t_i), v(t_i), J(t_i)\Big)^T$ at time $t = t_i$
- `u`, a scalar representing control variable $a(t_i)$ at time $t = t_i$

5.6.2 On Exiting Parameters

- `f`, the derivative of state `x`

5.7 `RKProp.m`

This is an independent function for the implementation of the RK4 algorithm in the Equation (3.5).

5.7.1 On Entering Parameters

- `h`, the step size $h = t_i - t_{i-1}$
- `x0`, the state at the starting point $t = t_{i-1}$ in current interval $[t_{i-1}, t_i]$
- `u0`, the control at the starting point $t = t_{i-1}$ in current interval $[t_{i-1}, t_i]$
- `u1`, the control at the ending point $t = t_i$ in current interval $[t_{i-1}, t_i]$
- `v0`, the intermediate control at center point $t = t_{i-1} + \frac{h}{2}$

5.7.2 On Exiting Parameters

- `y`, the propagated state at $t = t_i$ as defined in the Equation (3.5)

5.8 Source Code

The complete MATLAB® code is documented in Appendix B.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

6.1.1 A Prototype of Toolbox for Generating Optimal Trajectory

We have demonstrated an Optimization Toolbox® based MATLAB® toolkit which generates the optimal driving trajectories for traveling cars. Based on the input information such as traveled distance, initial and final velocity, and travel time, the toolkit can generate an optimal trajectory achieving minimal fuel consumption with which the car can still reach the destination within the required time. If further information about the location of the traffic lights and scheduling of the lights are provided, the toolkit takes into account of further constraints on velocity, then it can generate a new optimal trajectory for travel with traffic signals. The toolkit works very well for

- Travel between starting and end point *without* traffic
- Travel between starting and end point with *one* light
- Travel between starting and end point with *two* lights

For a more general situation where there are n lights located between the start and end point of a journey, we can follow the way how we extended the one-light code to two-light code, i.e., adding more inequality constraints such as `c(i) = DLi - s(tRi)` (rush to the red light to avoid a stop) or `c(i) = s(tGi) - DLi` (slowly move towards a red light until it is green again) to nonlinear constraint function `@NonLinCon` when calling solver `fmincon(@ObjFun,...,@NonLinCon,options)`, where `c` is nonlinear constraint $\boldsymbol{c}(\boldsymbol{p}) \leqslant \mathbf{0}$, `DLi` is the location of the i-th light and `tRi`, `tGi` are

timing instances for the i-th light to turn red and green respectively. We can use the toolkit to calculate minimum fuel consumption for all combinations of non-stop schemes (to avoid a red signal by rushing to the light or wait for the light to turn green again). Though in theory a minimum could be found for the n-light case, it is also expected that the computation time increases combinatorially with increased number of lights.

6.1.2 Prediction Using Receding Horizon

A suboptimal approach to searching for minimum fuel consumption is to use Receding Horizon. When there are n lights, we only use, for example, the two lights ahead of *current position* in calculation. We use the trajectories returned by the toolkit as a reference trajectory to cover the distance between current position and the location two lights ahead. After the car reaches the location of the second light ahead, *current position* will be updated and calculation resets. Repeating the process we use the next two lights ahead of the *new* current position. This is also the reason why we want to make sure this toolkit works perfectly for at least the one and two-light cases.

6.2 Future Research

6.2.1 Improvement on Modeling

In this thesis, only a highly simplified traffic environment has been considered. When there are traffic signals, only the signal scheduling is considered so that further constraints are introduced. However other factors such as location of other vehicles and road slope are ignored. Likewise, a high fidelity model could also give more precise information about the vehicle position. For example, imagine a situation when a traffic light is red and the car has to stop in front of the red light but at the same time there are also two other vehicles in the front. In this case, rather than specifying car displacement at `DL` (the position of the traffic light), the actual location should be `s(tR) = DL - 2*CarLength - e`, where `CarLength` is the average length of a passenger car and `e` is a correction for adding small distances between cars. Car-to-Car (C2C) or Vehicle-to-Vehicle (V2V) communication as well

as distance and velocity requirements between cars could also be incorporated into the equation of motion of the car. In this way, the dynamics of the system in the minimization problem could be more realistic. Topographic data also could be taken into account in the model of fuel consumption if real data is available.

Another issue is although the Biggs-Akçelik Instantaneous Model [23] is highly detailed involving velocity, acceleration etc., it was developed three decades ago. So we are looking forward to a modern version of fuel consumption which could be used for modeling fuel usage of new cars. An updated model of instantaneous fuel consumption can move our toolkit closer to daily applications on mobile devices.

6.2.2 Improvement on Computation Speed

On software level, we have demonstrated the *possibility* of a toolbox for optimal trajectory generation during a car's travel within an environment including traffic signals, the ultimate goal would be to install a device in the car such that this device can generate the optimal trajectories in Real-Time (RT). The device would include a binary file which has all the features introduced here written in the MATLAB code. Furthermore it requires that such device can compute trajectories fast enough so that it provides a *prediction* for the driver during the travel. This prediction would tell the driver whether the car is on a "right" speed to achieve the lowest fuel consumption while the total travel time is within a specified upper bound.

In order to improve the program in terms of its computation time, we look at two possible approaches:

- To acquire a powerful embedded micro-computer on board to perform calculations of trajectory generation
- To translate current program in MATLAB code to programs written in other programming languages such as C or Fortran

The first option is less preferable since most embedded systems are designed for limited computing capabilities, i.e., less powerful in terms of their computation prowess. They are designed only for a specific type of task. If it takes much more time for a program to run on a normal Desktop PC or

a Workstation, it would take even longer for the same program to return the same results by running the code on an embedded device. Any discussion of improving the usability of our program based on the optimization of hardware is beyond our focus.

The second option is more preferable. It is well known that Real-Time applications require fast responses. Most Real-Time applications are running on binaries generated by code written in Compiled Languages, such as C and Fortran. Compiled Languages are well suited to this need compared to Interpreted Languages (such as MATLAB and Python). SNOPT is a general-purpose NLP solver known for its running speed due to its implementation in Fortran [30]. A more detailed comparison between NLP solvers `fmincon` and SNOPT has been documented in Appendix A.

Therefore, if the current project could move forward, we would migrate all MATLAB code to Fortran code and use SNOPT instead of `fmincon`. If the integration of our algorithm to SNOPT enables us to reach the last stage of implementation of our toolbox on hardware, we would consider translating Fortran code to C code such that we could obtain a portable binary that runs on many devices.

APPENDIX A

SOLVER COMPARISON

A.1 Two NLP Solvers: `fmincon` and SNOPT®

There are two nonlinear solvers available to our numerical simulations. The first one is the one that has been used in this thesis, i.e., `fmincon` from Optimization Toolbox® [29] distributed by MathWorks Inc.® . The other is SNOPT from SBS Inc.® (Stanford Business Software Inc.) [31].

`fmincon` is a wrapper for an Optimization Toolbox NLP solver with four different algorithms: `'interior-point'`, `'sqp'`,`'trust-region-reflective'` and `'active-set'`. What we are interested in is the *default* algorithm `interior-point` since it can handle "large" problem size and even gradient information about the objective function or the constraints is not provided, automatic differentiation by finite difference will be used by `fmincon`.

SNOPT is a general-purpose NLP solver written in Fortran featuring sequential quadratic programming (SQP) algorithm, especially designed for constrained optimization problems with linear or nonlinear objective functions subject to bounds on variables and sparse linear or nonlinear constraints. Search directions during problem solving using SNOPT also involves QP subproblems [30].

A.2 A Benchmark Problem: `springa.f` Included in SNOPT® Examples

The following Problem (A.1) [32] can be solved by the example code provided with SNOPT. As a comparison, we follow the same algorithm in `springa.f` and translate the Fortran code to MATLAB® code which will call NLP solver `fmincon` provided by Optimization Toolbox to solve the same problem. It is

interesting to see the results shown in Table (A.1) below.

$$\begin{aligned} \underset{x,y,u}{\text{minimize}} \quad & f(x,y,u) = \frac{1}{2}\sum_{t=0}^{T} x_t^2, \\ \text{subject to} \quad & \left.\begin{aligned} & x_{t+1} = x_t + 0.2\,y_t \\ & y_{t+1} = y_t - 0.01\,y_t^2 - 0.004\,x_t + 0.2\,u_t \\ & -0.2 \leqslant u_t \leqslant 0.2 \\ & y_t \geqslant -1.0 \end{aligned}\right\} \; t = 0, \ldots, T-1 \\ & x_0 = 10,\; y_0 = 0,\; y_T = 0. \end{aligned} \tag{A.1}$$

Table A.1: Minima Calculated by `fmincon` and SNOPT—A Comparison

T	Min(`fmincon`)	Time(s)	Minimum(SNOPT)	Time(s)
10	550	1.0614[a]	549.999772444230	0.00[a,b]
20	1050	1.2291	1049.99999776141	0.00
30	1550.0001	1.7761	1549.99937363066	0.00
40	2050.0001	2.8611	2049.93545567966	0.00
50	2550.01	4.0855	2549.97925514803	0.00
60	1155.34	5.4259	2153.87554883498	0.00
70	876.0949	5.4031	1190.86538582841	0.00
80	876.2412	6.7514	1186.41843673913	0.00
90	876.2418	11.854	1186.38203475892	0.00
100	876.2417	19.4412	1186.38207249671	0.01
110	876.2417	23.6467	1186.38209688117	0.01

[a]Both `fmincon` and SNOPT were run on a GNU/Linux Debian installed ThinkPad® X230 with a Duo core CPU Intel® Core® i5-3380m; `matlabpool` was set to "2" to turn on parallel computing feature when approximating gradient used by `fmincon`.
[b]Output summary of SNOPT item "Time for solving problem" only shows two digits after the decimal point; "0.00" means program runs very fast.

For $T = 10, 20, \ldots, 110$, minima returned by both solvers `fmincon` and SNOPT for each T are shown in the table above; CPU time required to run for each T is also shown in Table (A.1). From the table we can see that for $T = 10, 20, \ldots, 50$, both solvers give almost the same minimum except that SNOPT runs *significantly* faster. From $T = 60$ onwards, the two solvers go to different search directions however both results converge to its individual local minimum starting from $T = 80$. It takes longer for `fmincon` to converge to its minimum 876.24 however this minimum is lower than the minimum 1186.38 calculated by SNOPT.

In the MATLAB code for Problem (A.1), we did not provide gradient information for solver `fmincon` because the objective function of the problem in this thesis Section 3.4.1 is piecewise smooth; we do not want to code in gradient information if the finite difference approximation used by `fmincon` can give us reasonably good results though it takes more iterations during the calculation.

As for the discrepancy between the minima in Table (A.1), it is *possible* that there are multiple minima for Problem (A.1) however either solver correctly gives only one of them. Therefore, there are several reasons for us to stick to MATLAB and `fmincon`. Firstly, and more evidently, the minimum returned by `fmincon` for Problem (A.1) is smaller than that of SNOPT. Secondly, it is easier to code *array* or *vector* manipulation in MATLAB than in Fortran. And lastly, `fmincon` can also treat the gradient information as optional which is desired for our problem formulation in Section 3.4.1.

However, as already mentioned in Chapter 6, future improvement on the optimization problem in this thesis could focus on how to implement fast computation and applications in embedded systems. Then it is worth an effort at translation of current MATLAB code to Fortran to make full use of the speed of SNOPT.

APPENDIX B

DOCUMENTATION OF MATLAB CODE

For the same reason mentioned at the beginning of Chapter 5, we will only attach the MATLAB® code for the case of two-light travel.

B.1 `RunMe.m`

```
% ** preamble starts **
%
% fixed tf
% two lights are present
%
% scheduling is as follows,
% tR1 = 20, tR2 = 33, tG1 = 25, tG2 = 38
% location of lights
% DL1 = 200, DL2 = 350
%
% ** preamble ends **

clear all % clear all workspace variables
close all % close all existing matlab windows

% start parallel pool
NumOfCores = feature('numCores');
if (strcmp(version,'8.1.0.604 (R2013a)'))
    needNewWorkers = (matlabpool('size') == 0);
    if needNewWorkers
        % workstation # of cpu 4; laptop # of cpu 2.
        % old syntax R2013a
        matlabpool('open',NumOfCores)
    end
else
```

```
26      needNewWorkers = isempty(gcp('nocreate'));
27      if needNewWorkers
28          % Open a new MATLAB pool with NumOfCore workers.
29          parpool('local',NumOfCores)  % new syntax R2014a
30      end
31  end
32
33  D = 400;   % traveled distance is 400m
34  tR1 = 20;  % 1st light turns red
35  tR2 = 33;  % 2nd light turns red
36  tG1 = 25;  % 1st light turns green
37  tG2 = 38;  % 2nd light turns green
38  DL1 = 200; % location of the first light
39  DL2 = 350; % location of the second light
40  % bounds
41  v_max = 25; % upper bound on vel, 25m/s = 56mph
42  a_max = 5;  % upper bound on acc
43  a_min = -2; % lower bound on acc
44  % boundary condition
45  vf_low = 0; % lower bound on final vel
46  vf_upp = 0; % upper bound on final vel
47  v0_low = 0; % lower bound on init vel
48  v0_upp = 0; % upper bound on init vel
49  % For instance, vf_low = v_f = vf_upp means v_f is fixed
50
51  % number of running
52  N_iter = 2;
53  Nsi = 1; % subdivision, Ns RK4 steps
54  GRSi = [1, 2, 3, 4]; % scheme selector
55
56  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
57  %% rush to 1st red; rush to 2nd red
58  Ti = D/v_max:2:65; % range of 'sweep'
59  Ni = 2*Ti; % mesh grid size associated with Ti;
60
61  is_red_red = GRSi(1); % select red-red scheme
62  % assign file handle to datafile
63  datafile = fopen('data_file_vf0_red_red.txt','w');
64  % running log preamble
65  fprintf(datafile, ['# Start from: ', ...
66          datestr(now,'mmmm dd, yyyy HH:MM:SS.FFF AM'),'\n']);
67  fprintf(datafile, ...
68          '# ---- number of iterations %4d ----\n', N_iter);
```

```
fprintf(datafile, '# %10s %4s %4s %4s %10s %10s %10s\r\n', ...
        'fuel_min', 'N', 'Ns', 'T', 'tf0', 'tf', 'el_time');
% reset counter
cnt = 0;
if (is_red_red == 1)
    for i = 1:length(Ti)
        for j = 1:length(Nsi)
            while ¬(cnt ≥ N_iter)
                cnt = cnt + 1;
                [fuel_min, N, Ns, T, tf0, tf, el_time] = ...
                    FuelInit(D, v_max, a_max, a_min, ...
                    v0_low, v0_upp, vf_low, vf_upp, ...
                    Ti(i), Ni(i), Nsi(j), cnt, ...
                    tR1, tR2, tG1, tG2, DL1, DL2, GRSi(1));
                fprintf(datafile, ...
                '%10.4f %4d %4d %4d %10.4f %10.4f %10.4f\n',...
                [fuel_min, N, Ns, T, tf0, tf, el_time]);
            end
            cnt = 0;
        end
    end
end
% data saved
fclose(datafile);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% rush to 1st red; wait for 2nd green
Ti = (tG2 + (D - DL2)/v_max):2:65; % range of 'sweep'
Ni = 2*Ti; % mesh grid size associated with Ti

is_red_green = GRSi(2); % select red-green scheme
% assign file handle to datafile
datafile = fopen('data_file_vf0_red_green.txt','w');
fprintf(datafile,['# Start from: ', ...
        datestr(now,'mmmm dd, yyyy HH:MM:SS.FFF AM'),'\n']);
fprintf(datafile, ...
        '# ---- number of iterations %4d ----\n', N_iter);
fprintf(datafile, '# %10s %4s %4s %4s %10s %10s %10s\r\n', ...
        'fuel_min', 'N', 'Ns', 'T', 'tf0', 'tf', 'el_time');
% reset counter
cnt = 0;
if (is_red_green == 2)
    for i = 1:length(Ti)
```

```
        for j = 1:length(Nsi)
            while ¬(cnt ≥ N_iter)
                cnt = cnt + 1;
                [fuel_min, N, Ns, T, tf0, tf, el_time] = ...
                    FuelInit(D, v_max, a_max, a_min, ...
                    v0_low,v0_upp, vf_low, vf_upp, ...
                    Ti(i), Ni(i), Nsi(j), cnt, ...
                    tR1, tR2, tG1, tG2, DL1, DL2, GRSi(2));
                fprintf(datafile, ...
                '%10.4f %4d %4d %4d %10.4f %10.4f %10.4f\n',...
                [fuel_min, N, Ns, T, tf0, tf, el_time]);
            end
            cnt = 0;
        end
    end
end
% data saved
fclose(datafile);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% wait for 1st green; rush to 2nd red
Ti = (tG1 + (D - DL1)/v_max):2:65; % range of 'sweep'
Ni = 2*Ti; % mesh grid size associated with Ti

is_green_red= GRSi(3); % select green-red scheme
% assign file handle to datafile
datafile = fopen('data_file_vf0_green_red.txt','w');
fprintf(datafile,['# Start from: ', ...
        datestr(now,'mmmm dd, yyyy HH:MM:SS.FFF AM'),'\n']);
fprintf(datafile, ...
        '# ---- number of iterations %4d ----\n',N_iter);
fprintf(datafile, '# %10s %4s %4s %4s %10s %10s %10s\r\n', ...
        'fuel_min', 'N', 'Ns', 'T', 'tf0', 'tf', 'el_time');
% reset counter
cnt = 0;
if (is_green_red == 3)
    for i = 1:length(Ti)
        for j = 1:length(Nsi)
            while ¬(cnt ≥ N_iter)
                cnt = cnt + 1;
                [fuel_min, N, Ns, T, tf0, tf, el_time] = ...
                    FuelInit(D, v_max, a_max, a_min, ...
                    v0_low, v0_upp, vf_low, vf_upp, ...
```

```
                    Ti(i), Ni(i), Nsi(j), cnt, ...
                    tR1, tR2, tG1, tG2, DL1, DL2, GRSi(3));
                fprintf(datafile, ...
                '%10.4f %4d %4d %4d %10.4f %10.4f %10.4f\n',...
                [fuel_min, N, Ns, T, tf0, tf, el_time]);
            end
            cnt = 0;
        end
    end
end
% data saved
fclose(datafile);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% wait for 1st green; wait for 2nd green
Ti = (tG2 + (D - DL2)/v_max):2:65; % range of 'sweep'
Ni = 2*Ti; % mesh grid size associated with Ti

is_green_green= GRSi(4); % select green-green scheme
datafile = fopen('data_file_vf0_green_green.txt','w');
fprintf(datafile,['# Start from: ', ...
        datestr(now,'mmmm dd, yyyy HH:MM:SS.FFF AM'),'\n']);
fprintf(datafile, ...
        '# ---- number of iterations %4d ----\n',N_iter);
fprintf(datafile, '# %10s %4s %4s %4s %10s %10s %10s\r\n', ...
        'fuel_min', 'N', 'Ns', 'T', 'tf0', 'tf', 'el_time');
% reset counter
cnt = 0;
if (is_green_green == 4)
    for i = 1:length(Ti)
        for j = 1:length(Nsi)
            while ¬(cnt ≥ N_iter)
                cnt = cnt + 1;
                [fuel_min, N, Ns, T, tf0, tf, el_time] = ...
                    FuelInit(D, v_max, a_max, a_min, ...
                    v0_low,v0_upp, vf_low, vf_upp, ...
                    Ti(i), Ni(i), Nsi(j), cnt, ...
                    tR1, tR2, tG1, tG2, DL1, DL2, GRSi(4));
                fprintf(datafile, ...
                '%10.4f %4d %4d %4d %10.4f %10.4f %10.4f\n',...
                [fuel_min, N, Ns, T, tf0, tf, el_time]);
            end
            cnt = 0;
```

```
198         end
199     end
200 end
201 % data saved
202 fclose(datafile);
```

B.2 FuelInit.m

```
1  function [fuel_min, N, Ns, T, tf0, tf, el_time] = ...
2            FuelInit(D, v_max, a_max, a_min, v0_low, v0_upp, ...
3                     vf_low, vf_upp, Ti, Ni, Nsi, cnt, tR1, ...
4                     tR2, tG1, tG2, DL1, DL2, GRSi)
5  % start clocking
6  startTime = tic;
7
8  % Number of divisions over time horizon [0, tf]
9  N = Ni;
10 % In each division, further discretize it with Ns subdivisions
11 Ns = Nsi;
12
13 % x1 = s; x2 = v; x3 = J; u = a
14 N_var = 3*N+3+2*N*Ns+1; % length of the NLP parameter
15
16 % tf, T ≤ tf ≤ T, i.e., tf = T
17 T = Ti;
18
19 % Green-Red selector, choose a travel scheme,
20 % $GRS \in {1, 2, 3, 4}$
21 GRS = GRSi;
22
23 % init parameter p
24 p = zeros(3*N+3+2*N*Ns+1,1);
25
26 % record current tf for the 'sweep'
27 tf = T;
28 % update p, add tf as the last element of the NLP parameter
29 p = [p;tf]; % NOT so useful when tf is fixed
30
31 %% bound setting
```

```
% lower bound
lb = zeros(3*N+3+2*N*Ns+1,1);
% update lb for tf
lb = [lb;0];
lb(N+1) = D; % s(tf) = D
lb(N+1+1) = v0_low; % lower bound on init velocity
lb(2*N+2) = vf_low; % lower bound on final velocity
lb(3*N+4:end-1) = a_min; % lower bound on acceleration
lb(end) = T; % tf fixed, tf = T

% upper bound
ub = Inf(3*N+3+2*N*Ns+1,1);
% update ub for tf
ub = [ub;T];
ub(1) = 0; % s(t0) = 0
ub(2:N+1) = D; % s(tf) = D
ub(N+1+1) = v0_upp; % upper bound on init velocity
ub(2*N+2) = vf_upp; % upper bound on final velocity
ub(N+1+2:2*N+1) = v_max; % upper bound on velocity
ub(2*N+2+1) = 0; % J(t0) = 0
ub(3*N+4:end-1) = a_max; % upper bound on acceleration
ub(end) = T; % tf fixed, tf = T

% record tf0, useful for debugging
tf0 = T;

% 1st run goes with p_init = [0, 0, ..., 0]',
% 2nd run goes with p_init = [1, 1, ..., 1]'.
if (cnt == 1)
    p_init = zeros(N_var+1,1);
elseif (cnt == 2)
    p_init = ones(N_var+1,1);
end

%% fmincon func call
% set options for calling fmincon
opts = optimoptions(@fmincon,'Algorithm', ...
                    'interior-point','Display','iter', ...
                    'TolCon',1e-5,'TolFun',1e-5, ...
                    'TolX',1e-5,'MaxFunEvals',1e4*N_var, ...
                    'MaxIter',1e4,'UseParallel','always');
% call fmincon, collect minimizer and min objective func value
[p_min, fuel_min] = fmincon(@(p)FuelObj(p,N),p_init,[],[], ...
```

```
                    [],[],lb,ub,@(p)FuelConstr(p,N,Ns,tR1, ...
                    tR2,tG1,tG2,DL1,DL2,GRS),opts);
% record tf
tf = p_min(end);

% save variables in workspace
if (GRS == 1)
    % rush to 1st red; rush to 2nd red
    filename = ['tRR_N_', num2str(N), '_Ns_', num2str(Ns), ...
                '_v0low_', num2str(v0_low), ...
                '_vf_', num2str(p_min(2*N+2)), ...
                '_T_', num2str(T), ...
                '_red_red_run_', num2str(cnt), '.mat'];
elseif (GRS == 2)
    % rush to 1st red; wait for 2nd green
    filename = ['tRG_N_', num2str(N), '_Ns_', num2str(Ns), ...
                '_v0low_', num2str(v0_low), ...
                '_vf_', num2str(p_min(2*N+2)), ...
                '_T_', num2str(T), ...
                '_red_green_run_', num2str(cnt), '.mat'];
elseif (GRS == 3)
    % wait for 1st green; rush to 2nd red
    filename = ['tGR_N_', num2str(N), '_Ns_', num2str(Ns), ...
                '_v0low_', num2str(v0_low), ...
                '_vf_', num2str(p_min(2*N+2)), ...
                '_T_', num2str(T), ...
                '_green_red_run_', num2str(cnt), '.mat'];
elseif (GRS == 4)
    % wait for 1st green; wait for 2nd green
    filename = ['tGG_N_', num2str(N), '_Ns_', num2str(Ns), ...
                '_v0low_', num2str(v0_low), ...
                '_vf_', num2str(p_min(2*N+2)), ...
                '_T_', num2str(T), ...
                '_green_green_run_', num2str(cnt), '.mat'];
end
% data saved
save(filename)

% end clocking
el_time = toc(startTime);

%% debugging using plotting
% plot vel, acc and J profiles
```

```
fig1 = figure(1); % assign graphics handle to fig1
clf(fig1) % reset fig1
% optimal travel time
tf = p_min(end); % opt engine-on time (specified in a 'sweep')
h = tf/(N*Ns); % division size
t = 0:tf/N:tf;
% plot without displacement; velocity and J only
plot(t,p_min(N+1+1:2*N+2),'-r',t,p_min(2*N+2+1:3*N+3),'k*')
hold on
tt = 0:h/2:tf;
% overlay acceleration
plot(tt,p_min(3*N+3+1:end-1),'-m','Marker','o')
legend('velocity','J','acceleration','Location','NorthWest')
xlabel('time')
title(['Optimal traveling time ', num2str(tf), ...
       's, displacement ', num2str(D), ...
       'm, min fuel consumption ', num2str(fuel_min), ...
       ', T = ', num2str(T), 's, tf_0 = ', num2str(tf0), 's'])
% save fig1 as *.eps
print(fig1,'-dpsc', '-r400', ...
    ['fig1_v0low_',num2str(v0_low), ...
     '_vf_', num2str(p_min(2*N+2)), '_T_', num2str(T), '.eps'])

% plot displacement profile
fig2 = figure(2);
clf(fig2)
% plot displacement profile only
plot(t, p_min(1:N+1),'*-')
legend('distance', 'Location', 'SouthEast')
xlabel('time')
title(['Optimal traveling time ', num2str(tf), ...
       's and displacement ', num2str(D), ...
       'm, min fuel consumption ', num2str(fuel_min), ...
       ', T = ', num2str(T), 's, tf_0 = ', num2str(tf0), 's'])
% save fig2 as *.eps
print(fig2,'-dpsc', '-r400', ...
    ['fig2_v0low_',num2str(v0_low), ...
    '_vf_', num2str(p_min(2*N+2)), '_T_', num2str(T), '.eps'])
% clear screen
clc
```

B.3 FuelObj.m

```
function J = FuelObj(p, N)
% FuelObj calculates the fuel consumption using the NLP
% parameter p and the number of divisions N.

J = p(3*N+3);
```

B.4 FuelConstr.m

```
function [c, ceq] = FuelConstr(p, N, Ns, tR1, tR2, tG1, ...
                               tG2, DL1, DL2, GRS)
% Parallel shooting method
% Constraints in the optimization problem added

% Rush to red light to avoid a stop:
% s(k) ≥ DL, where k = floor(tR/p(end)*N)

% Wait for green light on again:
% s(k) ≤ DL, where k = ceil(tG/p(end)*N)

%% Inequality constraints c(p) ≤ 0
%
% Four combinations of two lights' signals
if (GRS == 1)
    % rush to 1st red; rush to 2nd red
    % s(tR1) ≥ DL1 and s(tR2) ≥ DL2
    c = [DL1 - p(floor(tR1/p(end)*N));
         DL2 - p(floor(tR2/p(end)*N))];
elseif (GRS == 2)
    % rush to 1st red; wait for 2nd green
    % s(tR1) ≥ DL1 and s(tG2) ≤ DL2
    c = [DL1 - p(floor(tR1/p(end)*N));
         p(ceil(tG2/p(end)*N)) - DL2];
elseif (GRS == 3)
    % wait for 1st green; rush to 2nd red
    % s(tG1) ≤ DL1 and s(tR2) ≥ DL2
    c = [DL2 - p(floor(tR2/p(end)*N));
```

```
          p(ceil(tG1/p(end)*N)) - DL1];
elseif (GRS == 4)
    % wait for 1st green; wait for 2nd green
    % s(tG1) ≤ DL1 and s(tG2) ≤ DL2
    c = [p(ceil(tG2/p(end)*N)) - DL2;
         p(ceil(tG1/p(end)*N)) - DL1];
end

%% Equality constraints c(p) = 0
% subdivision size
h = p(end)/(N*Ns);

% N divisions over time horizon [0, tf]
for i = 1:N
    % inner loop over Ns subdivision within each tf/N*[i-1, i]
    x_start = [p(i);p(N+1+i);p(2*N+2+i)];

    for j = 1:Ns
        % Ns RK4 steps
        x_start = RKProp(x_start, h, ...
            p(3*N+3+1+(i-1)*2*Ns+2*j-2), ...
            p(3*N+3+1+(i-1)*2*Ns+2*j-1), ...
            p(3*N+3+1+(i-1)*2*Ns+2*j));
    end % end of Ns steps of integration
    % x_star - a 3-by-1 column vector, the propagated state
    % at the end of each division
    % x_star = NEXT x_start;
  ceq(3*i-2:3*i) = x_start - [p(i+1);p(N+1+i+1);p(2*N+2+i+1)];
end % end of equality constraints
```

B.5 FuelOdeRHS.m

```
function f = FuelOdeRHS(x, u)
% FuelOdeRHS calculates the right hand side of the
% governing equation.

% Input arguments dimension,
% x := 3-by-1 vector, x1 - s; x2 - v; x3 - J
% u := scalar, u - a
```

```
8
9  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
10 % Constants table
11   alpha = 0.444;
12   beta1 = 0.09;
13   beta2 = 0.04;
14   b1 = 0.333;
15   b2 = 0.00108;
16   M = 1200;
17 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
18
19   f = zeros(3,1); % init f
20
21   f(1) = x(2);     % \dot s = v
22   f(2) = u;        % \dot v = a
23
24 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
25   % tractive force
26   Rt = b1+b2*x(2)^2+M/1000*u;
27
28   q = (Rt > 0);    % sgn+ func Rt(t)
29   p = (u > 0);     % sgn+ func a(t)
30
31   % fuel rate
32   f(3) = alpha + beta1*x(2)*Rt*q + beta2*M/1000*x(2)*u^2*p*q;
```

B.6 RKProp.m

```
1  function y = RKProp(x0, h, u0, v0, u1)
2  % RKProp calculates the propagated state at the end of
3  % an interval based on the interval start x0 and
4  % the interval length h. For a propagation over ONE RK4
5  % step in interval [x0, x1] with control u0 and u1,
6  % there is ONE intermediate control v0.
7  %   x0  ----------------------- x1
8  %   x0  ----------------------- x* (propagation, y)
9  %   u0  --------- v0 ---------- u1
10 %    t 0-------------------------t 1
11 %     |<---------- h ----------->|
```

```
12
13      y1 = x0 + 1/2*h*FuelOdeRHS(x0,u0);
14
15      y2 = x0 + 1/2*h*FuelOdeRHS(y1,v0);
16
17      y3 = x0 + h*FuelOdeRHS(y2,v0);
18
19      y4 = x0 + 1/6*h*(FuelOdeRHS(x0,u0) ...
20              + 2*FuelOdeRHS(y1,v0) ...
21              + 2*FuelOdeRHS(y2,v0) + FuelOdeRHS(y3,u1));
22
23  y = y4;
```

REFERENCES

[1] Union of Concerned Scientists, "Car emissions and global warming," www.ucsusa.org/clean_vehicles/why-clean-cars/global-warming/, 2014.

[2] F. An and M. Ross, "A model of fuel economy and driving patterns," SAE International, SAE Technical Paper 930328, 1993.

[3] F. An and M. Ross, "Model of fuel economy with applications to driving cycles and traffic management," *Transportation Research Record*, vol. 1416, pp. 105–114, 1993.

[4] K. Ahn, H. Rakha, A. Trani, and M. Van Aerde, "Estimating vehicle fuel consumption and emissions based on instantaneous speed and acceleration levels," *Journal of Transportation Engineering*, vol. 128, no. 2, pp. 182–190, 2002.

[5] L. Evans and R. Herman, "Automobile fuel economy on fixed urban driving schedules," *Transportation Science*, vol. 12, no. 2, pp. 137–152, 1978.

[6] M. Ivarsson, J. Åslund, and L. Nielsen, "Look-ahead control consequences of a non-linear fuel map on truck fuel consumption," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 223, no. 10, pp. 1223–1238, 2009.

[7] J. Hooker, A. Rose, and G. Roberts, "A holistic approach to vehicle simulation," *Mathematics and Computers in Simulation*, vol. 25, no. 3, pp. 259 – 267, 1983.

[8] A. Rose, J. Hooker, G. Roberts, and J. Hodgson, "A data-based simulator for predicting vehicle fuel consumption," SAE International, SAE Technical Paper 820302, 1982.

[9] D. C. Biggs and R. Akçelik, "An energy-related model of instantaneous fuel consumption," *Traffic Engineering & Control*, vol. 27, pp. 320–325, 1986.

[10] A. R. Dohner, "Optimal control solution of the automotive emission-constrained minimum fuel problem," *Automatica*, vol. 17, no. 3, pp. 441 – 458, 1981.

[11] J. Hooker, A. Rose, and G. Roberts, "Optimal control of automobiles for fuel economy," *Transportation Science*, vol. 17, no. 2, pp. pp. 146–167, 1983.

[12] J. Hooker, "Optimal driving for single-vehicle fuel economy," *Transportation Research Part A: General*, vol. 22, no. 3, pp. 183 – 201, 1988.

[13] V. Monastyrsky and I. Golownykh, "Rapid computation of optimal control for vehicles," *Transportation Research Part B: Methodological*, vol. 27, no. 3, pp. 219 – 227, 1993.

[14] A. Stoicescu, "On fuel-optimal velocity control of a motor vehicle," *International Journal of Vehicle Design*, vol. 16, pp. 229–56, 1995.

[15] D. Chang and E. Morlok, "Vehicle speed profiles to minimize work and fuel consumption," *Journal of Transportation Engineering*, vol. 131, no. 3, pp. 173–182, 2005.

[16] E. Hellström, M. Ivarsson, J. Åslund, and L. Nielsen, "Look-ahead control for heavy trucks to minimize trip time and fuel consumption," *Control Engineering Practice*, vol. 17, no. 2, pp. 245 – 254, 2009.

[17] E. Hellström, J. Åslund, and L. Nielsen, "Design of an efficient algorithm for fuel-optimal look-ahead control," *Control Engineering Practice*, vol. 18, no. 11, pp. 1318 – 1327, 2010, special Issue on Automotive Control Applications, 2008 {IFAC} World Congress.

[18] K. Katsaros, R. Kernchen, M. Dianati, and D. Rieck, "Performance study of a green light optimized speed advisory (glosa) application using an integrated cooperative its simulation platform," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, July 2011, pp. 918–923.

[19] A. Fröberg, E. Hellström, and L. Nielsen, "Explicit fuel optimal speed profiles for heavy trucks on a set of topographic road profiles," SAE International, SAE Technical Paper 2006-01-1071, 2006.

[20] B. Asadi and A. Vahidi, "Predictive cruise control: Utilizing upcoming traffic signal information for improving fuel economy and reducing trip time," *Control Systems Technology, IEEE Transactions on*, vol. 19, no. 3, pp. 707–714, May 2011.

[21] T. Galpin and P. Voulgaris, "Fuel minimization of a moving vehicle in suburban traffic," in *American Control Conference (ACC), 2013*, June 2013, pp. 4009–4014.

[22] M. A. P. Taylor and T. M. Young, "Fuel consumption and emissions models for traffic engineering and transport planning applications: Some new results," in *Proceedings Roads 96 Conference, 18(6)*, (ARRB Transport Research), 1996, pp. 189–204.

[23] D. C. Biggs and R. Akçelik, "Estimation of car fuel consumption in urban traffic," in *Proc 13th ARRB Conf 13(7)*, 1986, pp. 124–132.

[24] M. Sanchez, J.-C. Cano, and D. Kim, "Predicting traffic lights to improve urban traffic fuel consumption," in *ITS Telecommunications Proceedings, 2006 6th International Conference on*, June 2006, pp. 331–336.

[25] W. Boyce and R. DiPrima, *Elementary Differential Equations and Boundary Value Problems*. Wiley, 2008.

[26] P. J. Enright and B. A. Conway, "Discrete approximations to optimal trajectories using direct transcription and nonlinear programming," *Journal of Guidance, Control, and Dynamics*, vol. 15, pp. 994–1002, 1992.

[27] P. J. Enright, "Optimal finite-thrust spacecraft trajectories using direct transcription and nonlinear programming," Ph.D. dissertation, University of Illinois, Urbana-Champaign, 1991. [Online]. Available: http://hdl.handle.net/2142/22370

[28] B. Chachuat, *Nonlinear and Dynamic Optimization: From Theory to Practice*, ser. Polycopiés de l'EPFL. EPFL, 2009. [Online]. Available: http://infoscience.epfl.ch/record/111939/files/Chachuat_07(IC32).pdf

[29] Mathworks Inc., "Documentation Center `fmincon`," R2014a. [Online]. Available: http://www.mathworks.com/help/optim/ug/fmincon.html

[30] SBS Inc., "User's Guide for SNOPT Version 7: Software for Large-Scale Nonlinear Programming," 2008. [Online]. Available: http://www.sbsi-sol-optimize.com/manuals/SNOPT_Manual.pdf

[31] Stanford Business Software Inc. (SBS Inc.), "Distribution of SOL/UCSD Optimization software." [Online]. Available: http://www.sbsi-sol-optimize.com/asp/sol_product_snopt.htm

[32] B. Murtagh and M. Saunders, "A projected lagrangian algorithm and its implementation for sparse nonlinear constraints," in *Algorithms for Constrained Minimization of Smooth Nonlinear Functions*, ser. Mathematical Programming Studies, A. Buckley and J.-L. Goffin, Eds. Springer Berlin Heidelberg, 1982, vol. 16, pp. 84–117. [Online]. Available: http://dx.doi.org/10.1007/BFb0120949