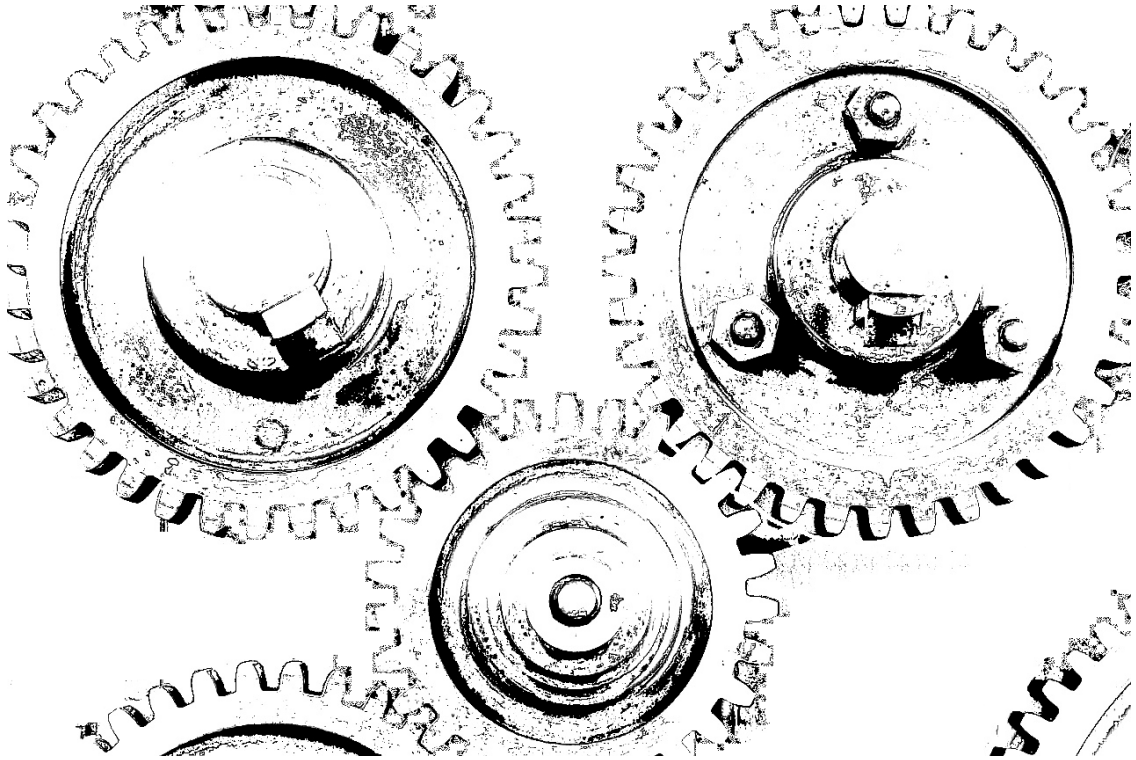
CHALMERS
UNIVERSITY OF TECHNOLOGY



# Versatile and Optimized Gear Selection Strategy

## Examining the Feasibility of Embedded Optimized Controllers for Gear Selection in Heavy-Duty Vehicles

Master's thesis in Systems, Control and Mechatronics

MARCUS LINDOHF
OSKAR NORDLANDER HURTIG

**Department of Electrical Engineering**
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

# Versatile and Optimized Gear Selection Strategy

Examining the Feasibility of Embedded Optimized Controllers for Gear Selection in Heavy-Duty Vehicles

MARCUS LINDOHF
OSKAR NORDLANDER HURTIG


CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2023

Versatile and Optimized Gear Selection Strategy
Examining the Feasibility of Embedded Optimized Controllers for Gear Selection in Heavy-Duty Vehicles
Marcus Lindohf, Oskar Nordlander Hurtig



Supervisors: Emma Svärling, Robin Karlsson, Volvo Groups Truck Technology
Examiner: Torsten Wik, Department of Electrical Engineering

Master's Thesis 2023
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A set of gears.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Versatile and Optimized Gear Selection Strategy
Examining the Feasibility of Embedded Optimized Controllers for Gear Selection in Heavy Duty Vehicles
Marcus Lindohf, Oskar Nordlander Hurtig
Department of Electrical Engineering
Chalmers University of Technology

# Abstract

We propose two versatile and optimized model predictive controllers suitable for embedded implementation in heavy-duty trucks, one based on the open-source mixed-integer nonlinear problem solver OpEn and one based on dynamic programming. The controllers are set to optimize the gear selections fed to an automatic gearbox during pedal driving. By utilizing road data in terms of the slope ahead, vehicle information such as the current engine speed, and the driver-demanded output shaft torque, the gear selections can be optimized with regard to the desired vehicle behavior depending on the tuning of the penalty coefficients within the controller. The hardware requirements are evaluated regarding the number of floating point operations per second needed per controller for embedded suitability. On an ARM Cortex A53 processor, the OpEn-based solver only requires approximately a tenth of the computational time compared to the DP-based controller. However, implementing the DP-based solver in C/C++ will probably improve the computational efficiency of the solver, decreasing the time difference between the two solvers. Also, the DP-based solver is more flexible in terms of altering the optimization algorithms and more robust in the sense of behaving according to the tuning parameters. It also returns a more optimized solution compared to the OpEn-based solver, given the problem formulation at hand. In conclusion, the DP-based MPC is the controller aligning most with the objectives of this thesis project.

Keywords: model predictive control, mixed-integer nonlinear program, OpEn, PANOC, dynamic programming, receding horizon controller, embedded, gear selection.

# Acknowledgements

To begin with, we would like to thank our partners for their emotional support throughout this project. We would also like to thank our examinator and supervisor Torsten Wik who supported us when we were in doubt as well as provided us with a lot of useful advice. The personnel at the predictive gear selection team at Volvo GTT has been very supportive as well, especially our supervisors Emma Svärling and Robin Karlsson. To conclude we would like to give a special thanks to Olof Lindgärde who helped us tackle some tough problems with great enthusiasm and knowledge!

Marcus Lindohf, Oskar Nordlander Hurtig, Gothenburg, June 2023

# List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

| | |
|---|---|
| B-OA | BONMIN - Outer Approximation |
| BONMIN | Basic Open-source Nonlinear Mixed Integer |
| CAN | Controller Area Network protocol |
| COIN-OR | Computational Infrastructure for Operations Research |
| CU | Control Unit |
| DP | Dynamic Programming |
| ECU | Engine Control Unit |
| FBE | Forward Backward Envelope |
| FLOPS | Floating Point Operations Per Second |
| fmincon | Find Minimum of Constrained Nonlinear multivariable function |
| GTT | Group Truck Technology |
| IPOPT | Interior Point Optimizer |
| L-BFGS | Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm |
| LTI | Linear Time-Invaraiant |
| LQR | Linear Quadratic Regulator |
| LQG | Linear Quadratic Gassian |
| MILP | Mixed-Integer Linear Programming |
| MINLP | Mixed-Integer Nonlinear Programming |
| MIPS | Millions of Instructions Per Second |
| MPC | Model Predictive Control |
| MPHC | Model Predictive Heuristic Control |
| NLP | Nonlinear Programming |
| NMPC | Nonlinear Model Predictive Control |
| OpEn | Optimization Engine |
| PANOC | Proximal Averaged Newton-type method for Optimal Control |
| PI | Proportional Integration |
| RHC | Receding Horizon Controller |
| RPM | Revolutions Per Minute |
| TCP | Transmission Control Protocol |
| TCU | Transmission Control Unit |

# Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

## Indices

| | |
|---|---|
| $k, j$ | Index for discrete time steps |
| $t$ | Index for continuous time steps |

## Sets

| | |
|---|---|
| $\mathcal{C}_g$ | Set of constraints |
| $\mathcal{U}$,$\mathcal{U}_g$ | Set of control inputs |

## Parameters

| | |
|---|---|
| $Q$,$Q_{factor}$ | Penalty coefficients related to states |
| $R$,$R_1$,$R_2$,$R_d$ | Penalty coefficients related to input |
| $\Delta z$ | Sample step in time or space domain |
| $m$ | Vehicle mass |
| $R_{wh}$ | Wheel radius |
| $g$ | Gravitational acceleration |
| $Cr$ | Rolling resistance coefficient |
| $i_d$ | Differential ratio |
| $i_g$ | Gear ratio |
| $A_f$ | Vehicle frontal area |
| $\rho$ | Air density |
| $C_d$ | Drag coefficient |

| | |
|---|---|
| $N$ | Length of prediction horizon |

# Variables

| | |
|---|---|
| $a$ | Acceleration |
| $v$ | Velocity |
| $F_{trac}$ | The tractive force delivered by the engine at the wheels |
| $F_{air}$ | Aerodynamic drag |
| $F_g$ | Perpendicular gravitational force |
| $F_{road}$ | The frictional force between the road and the tires of the vehicle |
| $E_v$ | Kinetic energy |
| $T_e$ | Engine torque |
| $\ell, \ell_N$ | Stage cost, terminal cost |
| $V_N$ | Prediction cost |

# Contents

# List of Figures

# List of Tables

# 1

# Introduction

With the rise of energy prices and the transition to a more sustainable society, optimizing and reducing the energy consumption of the sectors with the most consumption is of high interest for the related actors as well as for the society as a whole [1]. In the transport sector, heavy cargo trucks contributed to 21 % of the total emission of nitrogen oxide, $NO_x$, in Sweden 2021 [2]. In 2020, according to the Ministry of Energy, the transport sector used the equivalent of 138 TWh in diesel and bio-diesel fuel consumption for transports in Sweden [3].

Given the transport sector's energy consumption, developing new and more efficient control strategies with even a fractional improvement in energy efficiency can be considerably enticing for actors as the scale of the truck fleets renders large economic savings even with a minor improvement in energy efficiency.

Most conventional gear selection strategies for automatic transmissions are based on offline calculated shift maps, which are adjusted online depending on the driver's demand and the vehicle behavior [4]. Implementing a gear selection controller, independent of shift maps, based on optimization algorithms for online use could improve the energy utilization within the vehicle. Hence, a comprehensive analysis of optimal control strategies for gear selection strategies, considering both computational efficiency and optimality in terms of energy efficiency, has been conducted.

## 1.1 Objectives

The main objectives of this project were threefold and defined as:

- Investigate the potential adaptation of a model predictive control strategy for gear selection.
- Investigate different optimization routines and solvers in terms of optimality and solve time.
- Investigate the feasibility of an embedded implementation of an optimized gear selector for a heavy-duty truck.

## 1.2 Scope

Volvo GTT has provided simulation environments, based and equipped on their twelve-geared gearbox I-shift, to simulate and test the controller. However, the controller should be versatile, i.e. it should be able to handle different gearboxes, torque mappings, and driver profiles, especially with the electrification of the vehicle fleet. This requires that the model and the corresponding control structure are designed in such a manner that it can be configured to fit different powertrains. The provided simulation environment was only used for validation purposes towards a constructed `SIMULINK` simulation, as the use of Volvo GTT's proprietary equivalent would render most of the results classified.

The gear selection should always work independently whether the controller has available road data or not. The proposed solution should incorporate different control modes which activate accordingly to available input. As a starting point, the project proposes two distinct modes as follows:

- Map data unavailable:
  - Prediction based on current driver demanded torque together with current road incline.
- Map data available:
  - Prediction based on current driver demand torque together with road data.

# 2

# Optimal Control Theory

This chapter presents introductory material for the foundation of optimal control theory. First, an introduction to linear control is provided based on Rudolf E. Kálmán's work regarding the Linear Quadratic Regulator. Secondly, motivation and a brief explanation of the concept of Model Predictive Control are presented together with how the optimization problem is solved in practice using optimization engines or algorithms such as Dynamic Programming.

A more in-depth explanation is given of the inner workings of the optimization engine OpEn to provide an understanding of how it differs from more widely known optimization engines or routines such as IPOPT or SQP.

## 2.1 Early development and implementation

In the early 1960s Rudolf E. Kálmán worked on determining whether a linear control system can be said to be optimal given a quadratic objective function. The result is the well-known Linear Quadratic Regulator (LQR) [5].

Given a discrete-time Linear Time-Invariant model (LTI) defined by

$$\begin{aligned} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}(k) \\ \boldsymbol{y}(k) &= \boldsymbol{C}\boldsymbol{x}(k) \end{aligned} \tag{2.1}$$

where $\boldsymbol{x}_k$ defines the states, $\boldsymbol{u}_k$ the inputs, and $\boldsymbol{y}_k$ the output measurements. Evaluating the state and input sequence with a quadratic objective function defined as

$$V_N = \sum_{j=1}^{\infty} (||\boldsymbol{x}_{k+j}||_{\boldsymbol{Q}}^2 + ||\boldsymbol{u}_{k+j}||_{\boldsymbol{R}}^2), \tag{2.2}$$

where $\boldsymbol{Q}$ and $\boldsymbol{R}$ are penalty matrices for state deviation and the input, respectively. Kálmán showed that the solution to the LQR problem with an infinite horizon was a static feedback gain matrix $\boldsymbol{K}$ calculated by solving a Riccati equation, defined in Equation 2.3, resulting in the feedback $\boldsymbol{u}_k = -\boldsymbol{K}\boldsymbol{x}_k$.

$$\boldsymbol{K} = \left(\boldsymbol{R} + \boldsymbol{B}^T\boldsymbol{P}\boldsymbol{B}\right)^{-1}\left(\boldsymbol{B}^T\boldsymbol{P}\boldsymbol{A}\right) \tag{2.3a}$$

$$\boldsymbol{P} = \boldsymbol{A}^T\boldsymbol{P}\boldsymbol{A} - \left(\boldsymbol{A}^T\boldsymbol{P}\boldsymbol{B}\right)\left(\boldsymbol{R} + \boldsymbol{B}^T\boldsymbol{P}\boldsymbol{B}\right)^{-1}\left(\boldsymbol{B}^T\boldsymbol{P}\boldsymbol{A}\right) + \boldsymbol{Q} \tag{2.3b}$$

Kálmán would later expand the LQR theory to include state estimation and noisy measurements, which gave rise to the now well-known *Kalman Filter*. The method stemmed from probability theory, reflected in its name, Linear Quadratic Gaussian (LQG) [6].

Even though the LQG theory gave a solution to the optimal control of an LTI system given a quadratic objective function $V_N$, and was shown to stabilize the system as long as the penalty matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ were positive definite, it struggled to impact the control development in the process industry [7].

One of the main disadvantages of the LQG design that hampered the implementation in the industry was the lack of constraints on the input and states and the dependence on a linear plant, which made it impractical for some complex real-life industry processes. Another main disadvantage is the lack of robustness guarantees when the states need to be estimated.

## 2.2 The Rise of the Model Predictive Controller

The Model Predictive Control theory dates back to 1976 when Richalet et al. [7] presented a paper describing the control of a petrochemical plant using an approach they described as Model Predictive Heuristic Control (MPHC). It sought to optimize the control of a plant using a model and a prediction horizon to evaluate the input signal's influence on the plant over a horizon. The method enabled the implementation of constraints on the states and input, which reflects the nature of many real-world systems, where for example the actuators and different fluid flows have physical operating ranges and restrictions.

Richalet also showed that the most economical operating point of the plant often lies in the intersection of constraints, meaning that a more optimal industry controller can maintain the operating point close to constraints without violating them.

Driven by economic incentives and the slow nature of many chemical processes, the petrochemical industry started to adopt a more model-based approach when synthesizing controllers[8]. The slow response time and sampling interval were enough for the currently available computational power to find an optimized solution for the input.

The implementation and development of Richalet's model and control strategy in the industry sector gave rise to the concept we now refer to as Model Predictive Control (MPC), which has seen widespread use outside its original intention due to the advancement in computational hardware and development of more effective algorithms.

### 2.2.1 Theory behind Model Predictive Control

As the name implies, MPC heavily relies on the underlying dynamic model that describes the process or plant, where the quality of the predictions made by the MPC is a direct consequence of the quality of the model [9].

Given that the model is adequate for describing the dynamics of the process, the objective of the MPC is to make an optimized prediction along with a coupled optimized control sequence, $\boldsymbol{u}_k^\star$, that minimizes some objective function $V_N$ as defined in Equation 2.2.

An overview of the concept can be seen in Figure 2.1, where the controller makes an optimized prediction and control sequence given knowledge of the plant through measurements up to time $k$.

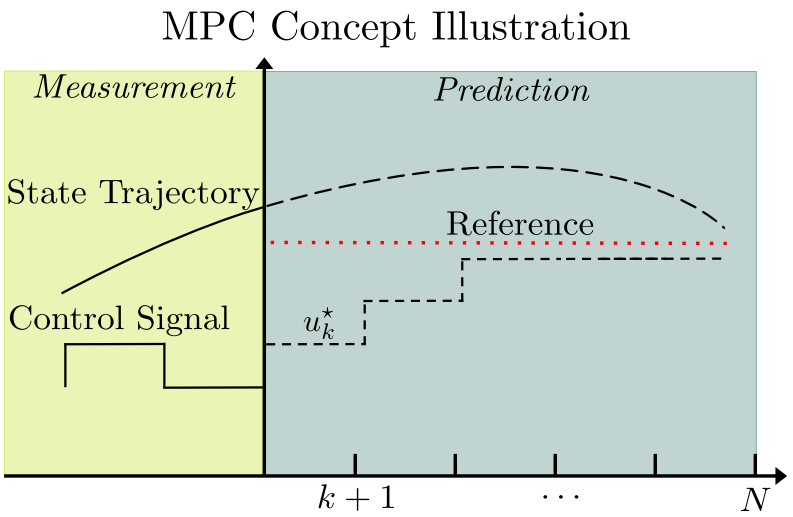



**Figure 2.1:** Illustration of the concept of a Model Predictive Controller. Given measurement from the plant coupled with the underlying model, it predicts an optimized state trajectory and control sequence according to the problem formulation.

Returning to the discrete linear system in Equation 2.1, the optimal static feedback gain $\boldsymbol{K}$ was calculated using the Ricatti equation given a linear quadratic cost function $V_N$. A coupling to the LQR solution and the MPC approach can be made by solving the optimal prediction and control sequence $\boldsymbol{u}^\star$ for the MPC by starting at the sample instance before the end of the horizon at $k = N - 1$. The MPC in this case uses the same dynamics as the LQR defined in Equation 2.1.

By dividing the optimization problem into smaller sub-problems, the optimization problem can be solved going from $k = N - 1$ to $k = N$. Repeating this process and optimizing backward from the end of the horizon at $k = N$ to the beginning at $k = 1$, a technique called Dynamic Programming (DP) can be utilized, which will be discussed more in detail in section 2.6.

The key insight to solving the MPC problem in this manner is that the backward optimization routine implicitly creates a closed-loop system due to the fact that the backward Dynamic Programming approach can be seen as state feedback, and the

resulting optimization problem at each sample instance is solved using the Riccati solution, as in the LQR problem.

The main difference is that the MPC approach uses a finite horizon and iteratively calculates the optimal solution for each sample step whereas the LQR solution in this approach assumes an infinite horizon, hence solving the Ricatti equations once.

As a consequence, the MPC approach does not guarantee stability whereas the LQR solution is asymptotically stable. Ending with the similarities of the unconstrained linear MPC and LQR solution, it can be noted that the two solutions will converge when increasing the horizon length for the MPC.

## 2.3 A Non-linear World With Constraints

Up until now, the previous sections have treated the history of optimal control resulting in the LQR/LQG control, and coupled it with the MPC approach in a scenario where the model is linear and variables are unconstrained.

As part of implementing an MPC control strategy on a real-world problem, the developer has to handle the often non-linear nature associated with the derived plant model. The trivial method, if the problem allows it, is to approximate the non-linear plant model with a corresponding linear model allowing for the utilization of classic control such as the LQR/LQG controllers discussed above.

However, real-world problems often come with physical limitations in actuators and hardware, which will result in constraints on the states and input values. This, coupled with a nonlinear model, renders the concepts of optimal control discussed above inadequate.

In practice, the optimization problems arising from the MPC approach are solved using an appropriate optimization engine or solver, where constraints are defined as a part of the problem formulation.

Using non-proprietary optimization engines, the more common ones would include Interior Point OPTimizer (IPOPT) [10] for nonlinear model predictive control (NMPC) and Basic Open-source Nonlinear Mixed INteger (BONMIN) for the mixed-integer case [11], both easily accessible using the software toolbox CasADi [12] for formulating the nonlinear optimization problems. For students using `MATLAB` the corresponding optimization library *fmincon* [13] is probably known.

With the focus on being suitable for embedded software applications, a relatively new optimization engine called Optimization Engine (OpEn) [14] is examined in the next section and given a brief explanation of how it differs from the more traditional solvers, such as IPOPT and *fmincon*.

## 2.4 Proximal Averaged Newton-type method for Optimal Control

The Proximal Averaged Newton-type method for Optimal Control (PANOC) is a combination of methods that are well suited for optimal control problems in an embedded context. It is the core optimization routine in the optimization engine OpEn.

It was first proposed in a paper called *A Simple and Efficient Algorithm for Nonlinear Model Predictive Control* [15] outlining the algorithm and performance comparing the performance to IPOPT and `MATLAB`'s *fmincon*.

### 2.4.1 Single-Shooting Formulation

For the sake of explaining PANOC, let an optimization problem take the following formulation:

$$\min_{\mathbf{u}} \quad \sum_{k=0}^{N-1} \ell_k\left(x_k, u_k\right) + \ell_N\left(x_N\right) \tag{2.4a}$$

$$\textbf{subject to} \quad x_0 = \bar{x} \tag{2.4b}$$

$$x_{k+1} = f_k\left(x_k, u_k\right), k = 0, \ldots, N-1 \tag{2.4c}$$

where $f_k : \mathbb{R}^{n_x \times n_u} \to \mathbb{R}^{n_x}$ are smooth mappings of the states corresponding to the system dynamics. $\ell_k$ and $\ell_N$ represent the stage cost and terminal cost, which are smooth mappings with $\ell_k : \mathbb{R}^{n_x \times n_u} \to \mathbb{R}^{n_x}$ and $\ell_N : \mathbb{R}^{n_x} \to \mathbb{R}$ as well. Here $n_x$ and $n_u$ represent the number of states and control inputs, respectively.

The problem statement can be reformulated to only contain one minimizing variable by introducing $F : \mathbb{R}^{Nn_u} \to \mathbb{R}^{(N+1)n_x}$ which is defined by

$$F\left(u_0, \ldots, u_{N-1}\right) = \left(F_0(\boldsymbol{u}), \ldots, F_N(\boldsymbol{u})\right) \tag{2.5}$$

where $F_0 \equiv \bar{x}$.

Utilizing Equation 2.4b we can express $F_{k+1}(u)$ as

$$F_{k+1}(\boldsymbol{u}) = f_k\left(F_k(\boldsymbol{u}), u_k\right), \quad k = 0, \ldots, N-1, \tag{2.6}$$

and denoting $\boldsymbol{u} = (u_0, \ldots, u_{N-1})$, a cost function $\ell$ can be expressed as functions of $\boldsymbol{u}$,

$$\ell(\boldsymbol{u}) = \sum_{k=0}^{N-1} \ell_k\left(F_k(\boldsymbol{u}), u_k\right) + \ell_N\left(F_N(\boldsymbol{u})\right) \tag{2.7}$$

The problem formulation in Equation 2.4 can then be formulated as

$$\underset{\boldsymbol{u} \in \mathbb{R}^{Nn_u}}{\textbf{minimize}} \quad \varphi(\boldsymbol{u}) \equiv \ell(\boldsymbol{u}), \tag{2.8}$$

which is known as the single-shooting formulation.

### 2.4.2 PANOC Algorithm

The PANOC algorithm breaks down into projected gradient steps and fast quasi-Newtonian directions. It is a line-search method designed to solve problems in the form of Equation 2.8.

In short, the projected gradient step is a method similar to a simple gradient-descent step. In contrast, it can be applied when the function is not differentiable which is the case in many practical applications [16].

The fast quasi-Newtonian direction is at its core a Newton step. However, the direction of the Newton-step $d^k$ can be calculated with a method called the limited-memory BFGS method [17]. The L-BFGS method is a quasi-Newtonian method that does not involve any matrix operations. It calculates the Newton step by a limited number of inner products which makes it suitable for embedded applications.

The line-search method can be expressed as:

$$u_{\nu+1} = u_\nu + \tau_\nu d^k + (1 - \tau_\nu)\, \gamma R_\gamma (u_\nu) \tag{2.9}$$

where $\tau_\nu$ holds the largest value such that

$$\varphi_\gamma (u_{\nu+1}) \leq \varphi_\gamma (u_\nu) - \sigma \left\| R_\gamma (u_\nu) \right\|^2, \tag{2.10}$$

and $\varphi_\gamma$ is the Forward Backward Envelope function which will be discussed in subsection 2.4.3.

At each iteration, the algorithm calculates an update direction $-\gamma R_\gamma$ and a candidate for the fast direction $d^k$, calculated by the Quasi Newtonian method. By appropriately averaging between $d^k$ and $-\gamma R_\gamma$ with $\tau_\nu$ such that Equation 2.10 is fulfilled, the algorithm enables global convergence and will transition to fast directions $d^k$ when close to a solution.

The method combines favorable properties in terms of convergence and swiftness which mainly stems from the Forward Backward Envelope function and the Quasi Newtonian method.

### 2.4.3 Forward Backward Envelope

Focusing on where PANOC differs from other optimization algorithms, one of the core novelties is the Forward Backward Envelope (FBE) function. The FBE function was first proposed in the paper *Proximal Newton Methods for Convex Composite Optimization* [18].

To understand how the FBE function contributes to the PANOC algorithm, a brief explanation of how Newtonian-type optimization algorithms converges is needed.

As a way to guarantee that the solution to the Newtonian-type methods converges, a method for evaluating if the next Newton step is closer to convergence than the current step has to be applied.

Often, this is conducted using a so-called merit function that approximates the function at the current step, the approximation is then used as a merit on where to take the next Newton step.

In sequential programming, this is usually done by a quadratic program such that the quadratic upper bound is minimized to evaluate the next step. The main disadvantage of this type of merit function is that constructing and finding the minimizing point to the quadratic program involves a number of inner iterative procedures which impact the computational time.

Going back to the cost function $\ell(\boldsymbol{u})$ in Equation 2.7, a point $u_k \in \mathcal{U}$ can be approximated by the quadratic upper bound as:

$$Q_\gamma^\ell(\nu; u_k) = \ell(u_k) + \nabla\ell(u_k)^\top(\nu - u_k) + \frac{1}{2}\gamma\|u_k - \nu\|^2, \tag{2.11}$$

where $\nu \in \mathcal{U}$, and $\gamma$ denotes the step size. Using the quadratic upper bound as a merit function involves finding the minima of $Q_\gamma^\ell$ which is where the high number of inner iterations materializes. This process is defined by the infimum $\inf_{\nu\in\mathcal{U}} Q_\gamma^\ell(\nu; u_k)$.

The FBE is constructed by

$$\varphi_\gamma(u_k) = \ell(u_k) - \frac{\gamma}{2}\|\nabla\ell(u_k)\|^2 + \text{dist}_{\mathcal{U}}^2(u_k - \gamma\nabla\ell(u_k)), \tag{2.12}$$

which, given that the distance to $\mathcal{U}$ is easily computed, only contains simple iterations.

For an arbitrary cost function $\ell(\boldsymbol{u})$, the quadratic upper bound $Q_\gamma^\ell(\nu; u_k)$ and its minima $\inf_{\nu\in\mathcal{U}} Q_\gamma^\ell(\nu; u_k)$, together with the FBE function $\varphi_\gamma(u_k)$, are illustrated in Figure 2.2.

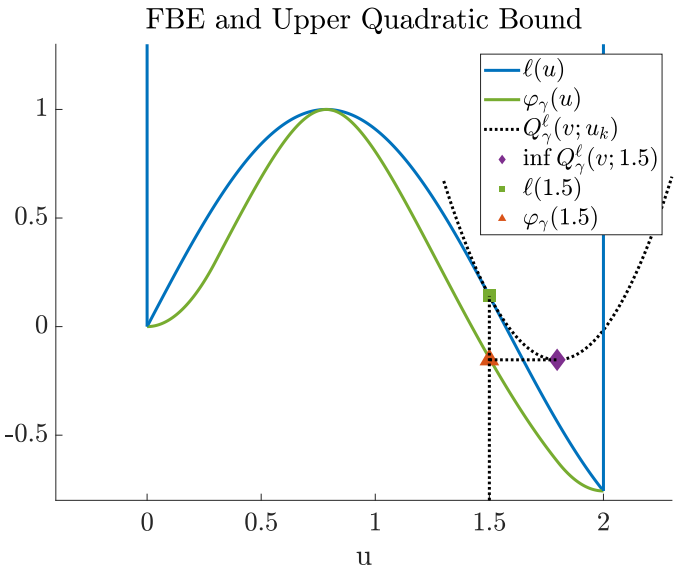



**Figure 2.2:** A comparison of the different merit functions. Given the point $u_k = 1.5$, both the FBE and upper quadratic bound method produce the same function value or minima, $y \approx -0.2$.

Observing the point that satisfies $\inf_{\nu\in\mathcal{U}} Q_\gamma^\ell(\nu; 1.5)$ in Figure 2.2, and the point $\varphi_\gamma(1.5)$, the minima of the upper quadratic bound, and the function value of FBE, is the same.

Hence, the two merit functions evaluate to the same value, with the difference that finding the minima of the quadratic upper bound is more computationally demanding than evaluating the FBE function, and is one of the key parts that contribute to PANOC's swiftness.

## 2.5 Mixed-integer Nonlinear Programming

On a steady course towards the explanation of how to apply the MPC approach to a more real-world-oriented problem, the sections above have dealt with the predictive nature of an MPC and how the optimization problem is often solved using optimization engines such as OpEn in the nonlinear case with constraints.

One last underpinning factor is the fact that real-world systems often involve a coupling between several kinds of continuous dynamics that coexist and interact based on discrete events. Such systems are called switched systems [19]. For example, the dynamics of the engine rpm are temporarily affected during a gear change, and after the gear change is completed, the system has switched dynamics with a new gear ratio affecting the dynamics.

Coupled with the addition that the continuous dynamics are nonlinear and that some variables can only take integer values, a lot of the legacy control methods are of no use and these problems often require heavy calculations for finding the solution.

Solvers that tackles these kinds of problem are referred to as Mixed-integer Nonlinear Programming (MINLP) solvers. The availability of such solver types is limited, with OpEn and BONMIN being two of the few open-source solvers handling such problems. However, with the development of faster, more efficient computer hardware, together with the nature of real-world dynamic systems being nonlinear and consisting of discrete events, scientists and engineers have during the last two decades started to give MINLP problems more attention [20].

## 2.6 Dynamic Programming

Stepping away from readily available solvers, another widely known method for solving mathematical optimization problems is Dynamic Programming (DP), which was introduced by Richard Bellman [21] in the 1950s. The process of solving an optimal control problem with dynamic programming is based on Bellman's *Principle of Optimality* which reads as:

**PRINCIPLE OF OPTIMALITY.** *An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.*

In control theory, given a discretized state space, this manifests as starting from a final state at the end of a horizon with length $N$, and optimizing each sample instance for a control sequence, one step at a time, until arriving at the initial state

at the beginning of the horizon. As the *Principle of Optimality* entails, the trailing sequence of optimized control actions at any sample instance, starting from the end and stepping backward, is still optimal regardless of any action taken, resulting in that state at that sample instance.

In the context of control synthesis, dynamic programming provides the designer with more control of the inner workings of the optimization process, mainly because it is up to the designer to construct the algorithm. This in turn facilitates the implementation of constraints and the process of working with mixed-integer problems.

However, without restricting the dimensions of the state space and the horizon length $N$, the dynamic programming algorithm often suffers from "the curse of dimensionality" [21], meaning that the available control actions or states to explore increase to a point where the computational time to solve the problem becomes infeasible.

### 2.6.1 Dynamic Programming Example

To envision Bellman's *Principle of Optimality*, a simple "find the shortest path"-problem is presented. Defining the problem as a deterministic finite state problem where the cheapest path to the final state is to be found, dynamic programming can be used [22].

Let the finite state space be defined by the different possible states, $x_i$ with $i \in [A, B, C, D, E, F, G, H, I]$, at the discrete sample instances $k \in \{1, 2, 3, 4, 5\}$, with the initial state $x_0 = x_A$ and the final state $x_f = x_I$.

At each sample instance $k$, a control action $u_k \in \{\pm 1\}$ can be taken defining which transition to take (-1 for southeast and +1 for northeast).

The problem is illustrated by the graph in Figure 2.3 below where the numbers are the transitions costs

$$
\begin{aligned}
&\boldsymbol{V}_{c2g}: \\
&V_{A\to I} = \min\left\{\begin{array}{c} 5 + V_{B\to I} \\ 7 + V_{C\to I} \end{array}\right\} \qquad V_{B\to I} = \min\left\{\begin{array}{c} 8 + V_{D\to I} \\ 3 + V_{E\to I} \end{array}\right\} \\
&V_{C\to I} = \min\left\{\begin{array}{c} 2 + V_{E\to I} \\ 6 + V_F \end{array}\right\} \qquad V_{D\to I} = 4 + V_{G\to I} \\
&V_{E\to I} = \min\left\{\begin{array}{c} 5 + V_{G\to I} \\ 6 + V_{H\to I} \end{array}\right\} \qquad V_{F\to I} = 1 + V_{H\to I} \\
&V_{G\to I} = 3 \qquad V_{H\to I} = 5
\end{aligned}
$$

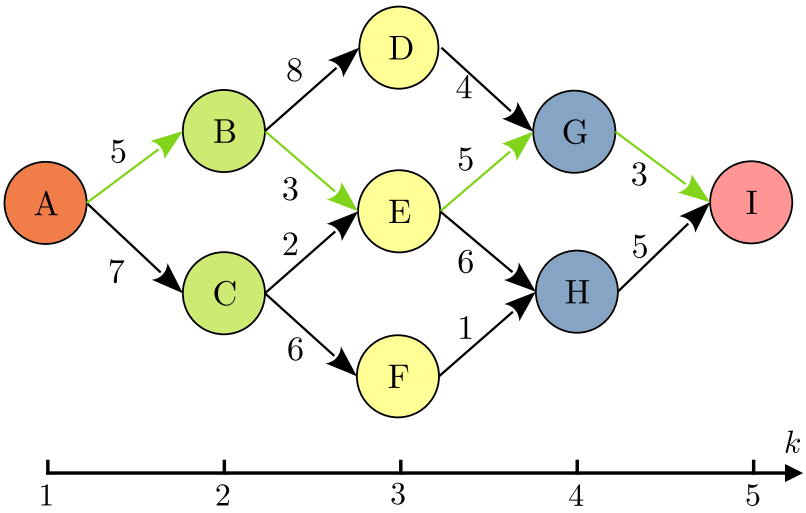



**Figure 2.3:** An example of a DP solution for a DAG (Directed Acyclic Graph). The green arrows highlight the optimal solution obtained by going backward from the final state.

Keeping the notation in line with the MPC formulation, the objective function to be minimized can be defined as:

$$V_N(x_A, \mathbf{u}) = \sum_{k=1}^{N-1} \ell_k(u_k) \tag{2.13}$$

where $\boldsymbol{u}$ is a control sequence and $\ell_k$ encodes the transition costs.

Applying the dynamic program approach, each state at every sample instance $k$ is evaluated starting from the final state $x_f$ and stepping backward to assemble a cost-to-go matrix, $\boldsymbol{V}_{c2g}$, defined as:

$$\boldsymbol{V}_{c2g,k} = \min_{u_k} \boldsymbol{V}_{c2g,k+1} + \ell_k(u_k). \tag{2.14}$$

Iterating the dynamic programming procedure and arriving at the initial state at $k = 1$, the cost-to-go matrix $\boldsymbol{V}_{c2g}$ contains the accumulated cost to $x_f$ from all states at different sample instances $k$.

Hence, the optimal path or control sequence minimizing the objective function $V_N(x_A, \mathbf{u})$, is the one encoded by taking the lowest cost in $\boldsymbol{V}_{c2g}$ at the index corresponding to $k = 1$, denoted as $\boldsymbol{V}^{\star}_{c2g,1}$ in:

$$\min_{\mathbf{u}} V_N(x_A, \mathbf{u}) = \boldsymbol{V}^{\star}_{c2g,1}. \tag{2.15}$$

The path or control signal resulting in the $\boldsymbol{V}^{\star}_{c2g,1}$ can either be found by encoding them in the backward pass or by making a forward pass and saving the control action corresponding to the lowest cost-to-go at each sample instance $k$.

# 3

# Synthesis of a Predictive Gear Controller

This chapter contains the methodology and tools used to translate the theory presented into the set of derived controllers used to produce the results illustrated later. The set of controllers follows the receding horizon controller principle and differs in the type of solver they utilize to solve the minimization problem. The solvers used are OpEn and a developed dynamic programming-based solver. The Bonmin solver is also presented in this section but is mainly used in comparison with OpEn since both are non-proprietary solvers used for MINLP problems.

Initially, the problem structure is presented with an overview of the general signal flow to and from the controller within the vehicle.

Secondly, a description of the vehicle dynamics in different domains is provided, together with the discretization method and the resulting state space used by the derived controllers. The principle behind the gear binary states is explained together with how they are affecting the engine torque and are utilized to constrain and penalize the controllers to a certain behavior.

Thirdly, the problem formulation is presented containing the formulation of the cost function, optimal control problem, and provided parameters. The chapter ends by explaining how the different receding horizon controllers are created, how the computational efficiency is measured for the solvers within the controllers, and how the simulations were conducted.

## 3.1 Problem Structure

To better understand the structure of the gear-selecting problem, some basic knowledge of the data flow and computing units of the vehicle is needed. The control of the various functions and parts of the truck are sectioned out to different Control Units (CUs), distributed throughout the vehicle. These are connected via a system interface called a Controller Area Network bus (CAN bus), enabling the various CUs to communicate and select the control signal or measurement needed to perform the controller task.

The CU controlling the gearbox is called the Transmission Control Unit (TCU) whose primary responsibility is to select an appropriate gear given the input it receives from the CAN bus. The gear-shifting operation is accomplished in conjunction

with other CUs controlling the clutch and ramping of the engine revolutions, among others.

The selection of the most appropriate gear is a complex and involved process that includes a vast amount of CAN bus data to produce a gear-selecting strategy that is, not only efficient but also matches the driver's expectations.

For example, a gear-selecting strategy increasing the performance of a vehicle is immensely different from the corresponding strategy to propel the vehicle in the most fuel-efficient way. The different behaviors are determined by the driving modes selected by the driver, and the demanded torque given by the accelerator pedal.

A simplified overview of the gear controller in the TCU can be seen in Figure 3.1, where the controller constantly evaluates different gears according to the driver's demand and the current state of the vehicle. This is illustrated by the subprocesses in the controller schematic.

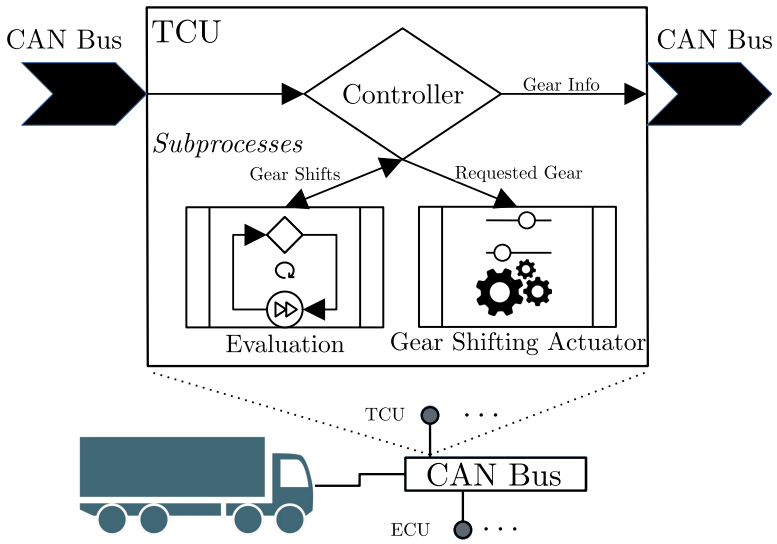



**Figure 3.1:** A simplified schematic of a vehicle CAN bus and TCU evaluating gear shifts. In this figure, "ECU" denotes the Engine Controlling Unit.

Given that the scope includes examining the implementation and feasibility of different gear-selecting strategies for a gearbox, a scaled-down version of the current simulation environment at Volvo Trucks is used for testing the different implementations. For time-saving purposes, the simulation environment is simplified such that the number of inputs and outputs, needed to run the gear selection software is minimized. The synthesized controller uses the demanded torque from the driver along with measured vehicle states and road data as seen in Figure 3.2. The proposed controller uses a vehicle model along with road data to predict future dynamics with the aim of selecting a gear sequence that minimizes an objective function $V_N$ at each iteration.

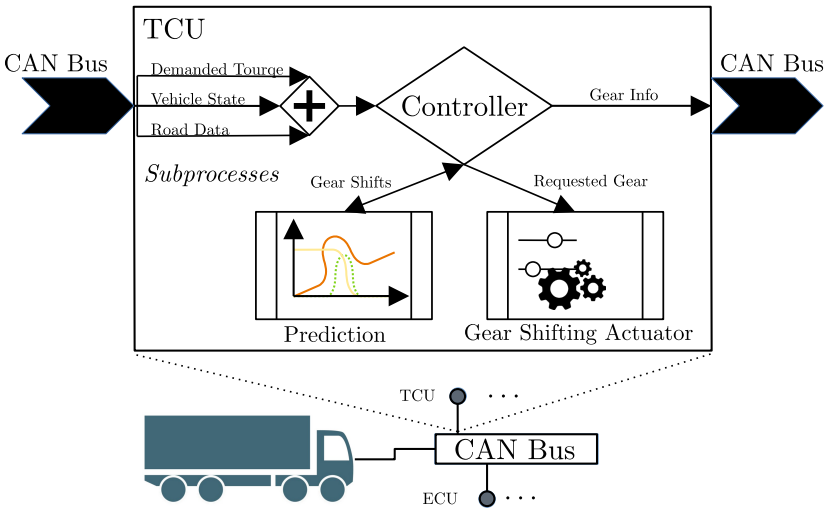



**Figure 3.2:** A simplified schematic of a vehicle CAN bus and a TCU executing a predictive gear-selecting strategy.

## 3.2 Vehicle Modeling

Making a valid prediction of the optimal gear selection requires thorough modeling of the vehicle dynamics corresponding to the real physics of the vehicle. Given that the road data is provided in terms of slope angles per distance, the vehicle model used by the predictive gear selecting strategy is expressed by its longitudinal dynamics.

The basis for the model stems from Newton's second law of motion $\sum F = ma$, where the forces affecting the vehicle are summed up, resulting in a longitudinal acceleration of the vehicle shown in Figure 3.3.

The forces affecting the vehicle are described by the tractive force $F_{trac}$, the air resistive force $F_{air}$, the perpendicular force vector $F_g$ resulting from the slope and gravitational acceleration, and the resistive friction force $F_{road}$ from the road contact with the wheels.

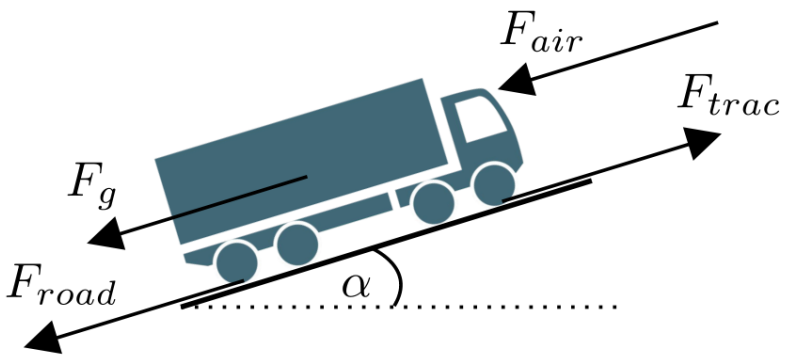



**Figure 3.3:** A free body diagram of the forces acting on the vehicle.

The dynamics of the vehicle can either be described in time domain or space domain by altering the derivation of the acting forces on the vehicle. Whilst the time domain is more commonly used and in some sense more intuitive, modeling the vehicle in the space domain has the advantage of linearizing the model and making the implementation of road data more streamlined as it is often given in space domain. Given the ambiguity of which domain to use, the synthesized controller is adapted for both, with the benefits and drawbacks further discussed in the section 4.3.

The sections below describe the methods to formulate the dynamics in the respective domain. The model-specific constants used for the dynamics are presented in Table 3.1.

| Acronym | Description | Value | Unit |
|---|---|---|---|
| $m$ | Vehicle mass | 33000 | [kg] |
| $R_{wh}$ | Wheel radius | 0.49 | [m] |
| $g$ | Gravitational acceleration | 9.82 | [m/s$^2$] |
| $C_r$ | Rolling resistance coefficient | 0.009 | [−] |
| $i_d$ | Differential ratio | 2.47 | [−] |
| $A_f$ | Vehicle frontal area | 3.5 | [m$^2$] |
| $\rho$ | Air density | 1 | [kg/m$^3$] |
| $C_d$ | Drag coefficient | 0.6 | [−] |

**Table 3.1:** Shared model constant acronyms, values, units, and descriptions.

### 3.2.1 Continuous-time Dynamics

For the unidirectional motion of a vehicle, using Newton's second law, the continuous time dynamics can be expressed by summing the forces acting on the truck, i.e.

$$ma(t) = F_{trac}(t) - F_{air}(t) - F_g(t) - F_{road}(t). \tag{3.1}$$

The tractive force $F_{trac}$ is given by

$$F_{trac}(t) = \frac{T_e(t) i_g(t) i_d}{R_{wh}}, \tag{3.2}$$

where $T_e(t)$ is the engine torque at time $t$, $i_g(t)$ is the gear ratio at time $t$, $i_d$ is the differential ratio, and $R_{wh}$ is the wheel radius. Furthermore, the air resistive force $F_{air}$, also known as drag, is

$$F_{air}(t) = \frac{1}{2} C_d \rho A_f v(t)^2, \tag{3.3}$$

where $C_d$ is the drag coefficient, $\rho$ is the air density, $A_f$ is the effective frontal area of the vehicle, and $v(t)$ is the vehicle velocity at time $t$. The gravitational force $F_g$ is

$$F_g(t) = mg \sin(\alpha(t)), \tag{3.4}$$

where $m$ is the vehicle mass including load, $g$ is the gravitational acceleration, and $\alpha(t)$ is the slope at time $t$. Lastly, the road resistive force $F_{road}$ is

$$F_{road}(t) = mgC_r \cos(\alpha(t)), \tag{3.5}$$

where $C_r$ is the rolling friction coefficient.

### 3.2.2 Continuous-space Dynamics

To describe the dynamics in space domain, instead of the acceleration as in time domain, the derivation stems from the kinetic energy of the vehicle $E_v$,

$$E_v = \frac{mv^2}{2}. \tag{3.6}$$

Differentiating the kinetic energy with respect to the distance $s$, the following reformulation is obtained

$$\dot{E}_v = \frac{dE_v}{ds} = \frac{mdv^2}{2ds} = mv\frac{dv}{ds} = mv\dot{v}(s). \tag{3.7}$$

Now, using an alteration of Newton's second law of motion $F = ma$, the following is obtained

$$F = ma(t) = m\frac{dv}{dt} = m\frac{dv}{ds}\frac{ds}{dt} = mv\frac{dv}{ds} = mv\dot{v}(s). \tag{3.8}$$

By observing that the derivative of the kinetic energy, Equation 3.7, equates the altered Newton's second law of motion, Equation 3.8, it is possible to express the dynamics in terms of the derivative of the vehicle's kinetic energy per distance, resulting in

$$\dot{E}_v(s) = F_{trac}(s) - F_{air}(s) - F_{fric}(s) - F_{slope}(s). \tag{3.9}$$

The tractive force $F_{trac}(s)$ is the same as in time domain except for the engine torque and gear ratio being space-dependent. The same applies to the gravitational force $F_g(s)$ and resistive force $F_{road}(s)$, where the slope angle $\alpha$ is space-dependent.

The air resistive force $F_{air}$ in space domain is formulated using an alteration of Equation 3.6

$$v = \frac{1}{\sqrt{\frac{m}{2E_v}}} \tag{3.10a}$$

$$F_{air} = \frac{1}{2}C_d A_f \rho v(s)^2 = \frac{C_d A_f \rho E_v(s)}{m}, \tag{3.10b}$$

where the resulting expression of $F_{air}(s)$ is linear in the space domain.

### 3.2.3 Discretization of Continous Dynamics

The continuous functions used to model the vehicle dynamics are discretized using the explicit fourth-order Runga Kutta method [23], with the resulting Runga Kutta step defined as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta z \left( \frac{1}{6}\boldsymbol{K}_1 + \frac{1}{3}\boldsymbol{K}_2 + \frac{1}{3}\boldsymbol{K}_3 + \frac{1}{6}\boldsymbol{K}_4 \right), \tag{3.11}$$

where $\Delta z$ is the domain-specific sample time or sample length. The coefficients are defined as

$$\boldsymbol{K}_1 = f\left(\boldsymbol{x}_k, u_k\right) \tag{3.12a}$$

$$\boldsymbol{K}_2 = f\left(\boldsymbol{x}_k + \frac{\Delta z}{2}\boldsymbol{K}_1, u_k + \frac{\Delta z}{2}u_k\right) \tag{3.12b}$$

$$\boldsymbol{K}_3 = f\left(\boldsymbol{x}_k + \frac{\Delta z}{2}\boldsymbol{K}_2, u_k + \frac{\Delta z}{2}u_k\right) \tag{3.12c}$$

$$\boldsymbol{K}_4 = f\left(\boldsymbol{x}_k + \Delta z\boldsymbol{K}_3, u_k + \Delta z u_k\right), \tag{3.12d}$$

where $f\left(\boldsymbol{x}_k, u_k\right)$ is the continuous dynamics expressed in Equation 3.1 with the gear ratios as input $u_k$.

For brevity the discretized dynamics in Equation 3.11 are described as

$$\boldsymbol{x}_{k+1} = f_{RK4}(\boldsymbol{x}_k, u_k, \alpha_k). \tag{3.13}$$

### 3.2.4 Gear Specific States

The equations up to this point are adequate to describe the longitudinal dynamics of the vehicle in either time or space domain given an applied torque by the engine, $T_e$, a selected gear $i_g$, and a slope angle $\alpha$ given by the road inclination.

However, there are no equations describing the dynamics in the event of a gear shift, rather than being instantaneous with no effect on the delivered torque by the engine.

To expand the vehicle dynamics in order to describe the event of a gear shift, two binary states and one timer state are added, defined as the gear request indicator, the gear binary state, and the gear delay counter.

The gear request indicator is a binary state, $x_{bin.state} \in \{0, 1\}$, that is active during one sample in the event of a changing control signal $u_k$, resetting the gear delay counter to the specified time or distance of a gear shift. The gear binary state is active for the duration of the gear delay counter holding a value greater than zero.

Using the gear binary state, the delivered engine torque $T_e$ can be set to zero during a gear shift using

$$T_e = T_e - x_{bin.state}T_e. \tag{3.14}$$

Using the same approach, two additional states are added, defined as the hold gear binary state and the hold gear delay counter. The hold gear binary state, $x_{hold.state} \in \{0, 1\}$, is a binary state that is activated after the event of a gear shift for the duration specified by the counter. It is used as a method to penalize frequent gear shifts in succession by penalizing the event of an active gear indicator during an active hold gear binary state. In other words, a method to penalize a gear shift taking place too close to the previous gear shift in time or space.

Figure 3.4 illustrates the behavior of the binary states, where a gear shift is conducted seven seconds into the simulation in the time domain. The gear shifting time and hold gear time in this example are one and five seconds, respectively.

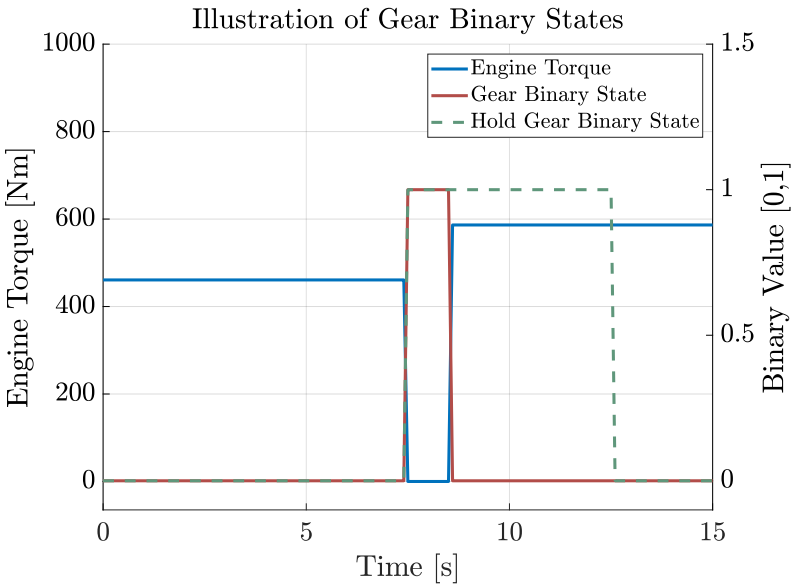



**Figure 3.4:** An illustration of the behavior of the gear binary states. The engine torque is momentarily canceled in the event of a gear shift. The gear binary state depicts the time for decoupling the engine torque, while the hold gear binary state is used for penalizing the event of another gear shift occurring within five seconds after the first gear shift.

### 3.2.5 State Space Representation

The full state space representation of the vehicle dynamics can be observed in Table 3.2, illustrating the structure of the state space for the respective domains.

| State | Description | Unit |
|---|---|---|
| | *Domain Shared States* | |
| $x_0$ | Velocity | [m/s] |
| $x_1$ | RPM | [rev/min] |
| $x_2$ | Engine torque | [Nm] |
| $x_3$ | Output shaft torque | [Nm] |
| $x_4$ | Current gear ratio | [11.73, 9.21, ...] |
| $x_5$ | Gear request indicator | [0,1] |
| $x_6$ | Gear delay counter | [ms] |
| $x_7$ | Gear binary state | [0,1] |
| $x_8$ | Hold gear binary state | [0,1] |
| $x_9$ | Hold gear delay counter | [ms] |
| $x_{10}$ | Kinetic energy | [J] |
| $\vdots$ | | |
| | *Time Domain Specific States* | |
| $x_{11}$ | Distance travelled | [m] |
| $x_{12}$ | Acceleration | [m/s$^2$] |
| | *Space Domain Specific States* | |
| $x_{11}$ | Time | [s] |

**Table 3.2:** Common states for the domains and domain-specific states.

The state $\boldsymbol{x}$ is either in the domain $\mathbb{R}^{1\times 11}$ or $\mathbb{R}^{1\times 12}$ depending on if the model is defined in space domain or time domain during the simulations.

### 3.2.6 Engine Map

The engine map is used to produce an RPM reference for the solver given a requested torque from the driver. The resulting reference is the RPM where the engine is most efficient given the demanded torque.

The engine map is comprised of a limiting max torque curve of the engine and contours indicating the efficiency of the engine at different operating points, see Figure 3.5. The event of a negative torque is described as an engine break, where the engine is deprived of the inflow of fuel, resulting in a retarding force due to the friction in the engine and the compressing of air in the cylinders, among others.

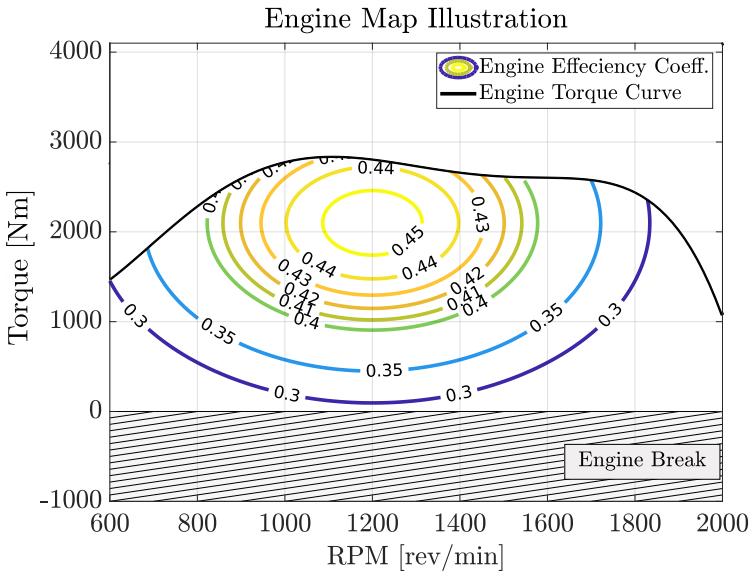



**Figure 3.5:** Illustration of the Engine map used.

The max torque curve of the engine is constructed using values to resemble a *D16K650* engine used in Volvo Truck's production vehicles. The values were taken from the data sheet of an engine found at Volvo Truck's website [24].

Using the `MATLAB` function *polyfit*, a polynomial of $5^{th}$ order

$$f_{MaxTq}(x_1) = c_1 x_1^5 + c_2 x_1^4 + c_3 x_1^3 + c_4 x_1^2 + c_5 x_1 + c_6, \tag{3.15}$$

was created to approximate how the max torque of the engine depends on the RPM $x_1$. The coefficients of the polynomial are

$$\begin{aligned} c_1 &= -1.15 \cdot 10^{-11} \\ c_2 &= 6.85 \cdot 10^{-8} \\ c_3 &= -1.15 \cdot 10^{-4} \\ c_4 &= 0.16 \\ c_5 &= -76.28 \\ c_6 &= 1.4 \cdot 10^{4}. \end{aligned} \tag{3.16}$$

The contours indicating the efficiency of the engine were created using ellipsoids defined by

$$f_{Eng.Eff}(x_1, x_2) = -\left( \frac{(x_1 - h_{RPM})^2}{a} + \frac{(x_2 - h_{tq})^2}{b} \right) \cdot c_{scale} + h_{offset}, \tag{3.17}$$

taking in the engine RPM and engine torque $(x_1, x_2)$ as argument to express the

efficiency at that operating point. The coefficients for the ellipsoid are

$$\begin{aligned} a &= 0.7 \\ b &= 7 \\ c_{scale} &= 2.7 \cdot 10^{-7} \\ h_{RPM} &= 1200 \\ h_{tq} &= 2100 \\ h_{offset} &= 0.455. \end{aligned} \tag{3.18}$$

Given the engine efficiency function $f_{Eng.Eff}$ and a demanded torque from the driver, the most efficient RPM reference together with the torque is passed as parameters to the solver.

Note that the engine map and its constituent parts are estimated or fabricated to resemble a production engine rather than to replicate its specific values. This is done to keep the paper unclassified and focus on the algorithms used.

## 3.3 Problem Formulation

The problem formulation reflects the overall aim of the controller and defines the problem for the solver to optimize. The formulation is based on a stage cost $\ell_k$, and a terminal cost $\ell_N$, which defines the cost at a sample instance $k$ and a cost at the end of the horizon $k = N$ respectively.

The input to the solver is organized into the current state vector $\boldsymbol{x}_k$ and a parameter vector $\boldsymbol{p}_k$ containing road data, the demanded torque, and the calculated RPM reference. The parameter vector $\boldsymbol{p}_k$ is assembled as:

$$\boldsymbol{p}_k = \begin{bmatrix} \boldsymbol{P}_{road} & P_{torque} & P_{ref} \end{bmatrix}, \tag{3.19}$$

where $P_{road} \in \mathbb{R}^{1\times N}$ contains the slope values from the road data, $P_{torque} \in \mathbb{R}^{1\times 1}$ is the demanded torque and $P_{ref} \in \mathbb{R}^{1\times 1}$ is the calculated RPM reference.

The stage cost at instance $k$ is defined by

$$\begin{aligned} \ell(\boldsymbol{x}_k, \boldsymbol{p}_k) =& Q_{factor} Q(x_{2,k} - p_{RPMRef,k})^2 \\ &+ R_1(x_{5,k}x_{8,k})^2 \\ &+ R_2 x_{5,k}^2, \end{aligned} \tag{3.20}$$

where $Q$, $R_1$, and $R_2$ are tuning parameters that affects which behavior the solution from the solver will have. A high $Q$-value will force the solver to find a solution closer to the calculated most efficient RPM. The $R_1$-value will penalize the current stage if the gear indicator state $x_5$, and hold gear state $x_8$, are activated at the same time, meaning the solver changes gear during the hold gear time frame. The $R_2$-value only affects the gear indicator state. A higher value will force the solver to take fewer gear shifts and vice versa for a lower value.

Finally, the $Q_{factor}$ is a dynamic penalization factor that scales with increasing velocity. This is implemented to slacken the penalization for RPM reference deviation at a slow velocity to get a higher acceleration from a stationary state.

The terminal cost $\ell_N$ is defined as

$$\ell_N(\boldsymbol{x}_N, \boldsymbol{p}_N) = R_d \left( \frac{1}{x_{11,k} - x_{11,0}} \right)^2, \tag{3.21}$$

which serves as a mean to penalize the distance traveled in time domain or the elapsed time in space domain. A high penalization on $\ell_N$ will force the solver to choose a solution resulting in a high average vehicle velocity for the prediction horizon.

The total cost function reads

$$V_N \left( \boldsymbol{x}_0, \boldsymbol{p}_0 \ldots, \boldsymbol{x}_{N-1}, \boldsymbol{p}_{N-1} \right) = \sum_{k=0}^{N-1} \ell \left( \boldsymbol{x}_k, \boldsymbol{p}_k \right) + \ell_N(\boldsymbol{x}_N, \boldsymbol{p}_N). \tag{3.22}$$

Expanding the problem formulation with constraints, the final problem for the solver to optimize becomes

$$\begin{aligned} &\min_{\boldsymbol{x},\boldsymbol{u},\boldsymbol{p}} \sum_{k=0}^{N-1} \ell(\boldsymbol{x}_k, \boldsymbol{p}_k) + \ell_N(\boldsymbol{x}_N, \boldsymbol{p}_N) \\ &\text{s.t.} \\ &\quad \boldsymbol{x}_{k+1} = f_{RK4}(\boldsymbol{x}_k, u_k, \boldsymbol{p}_k), \ \boldsymbol{x}(0) = \boldsymbol{x}_0 \\ C_{RPMMin} \leq\ & x_{k,1} \leq C_{RPMMax} \\ &\quad x_{2,k} \leq f_{MaxTq}(x_{1,k}) \\ &\quad u_k \in \mathcal{U}_g \\ &\quad \boldsymbol{x}_k \in \mathbb{R}^n \end{aligned} \tag{3.23}$$

where $C_{RPMmin} = 600$ and $C_{RPMmax} = 2000$ are the constraints for the engine RPM, $\mathcal{U}_g$ is the discrete set containing the gear ratios for the vehicle. The gear ratios used for creating the input set for the controller are the following: $\mathcal{U}_g = \{11.73, 9.21, 7.09, 5.57, 4.34, 3.41, 2.69, 2.12, 1.63, 1.28, 1, 0.785\}$.

## 3.4 Receding Horizon Control

The control structure follows the principle of receding horizon control, meaning that the controller at the current sampling instance $k$, will take into account what is known about the process up until the current sample and make a prediction for the process with a horizon length of $N$. Using the prediction for the given constraints, an optimized control sequence, $\boldsymbol{u}^\star$, will be calculated.

The receding horizon controller then only extracts the first control signal from $\boldsymbol{u}^\star$ and applies it to the process, before repeating the procedure at the next sample instance. An illustration of the procedure can be observed in Figure 3.6, where at each sample instance $k$, an optimized control signal $\boldsymbol{u}^\star$ is calculated.

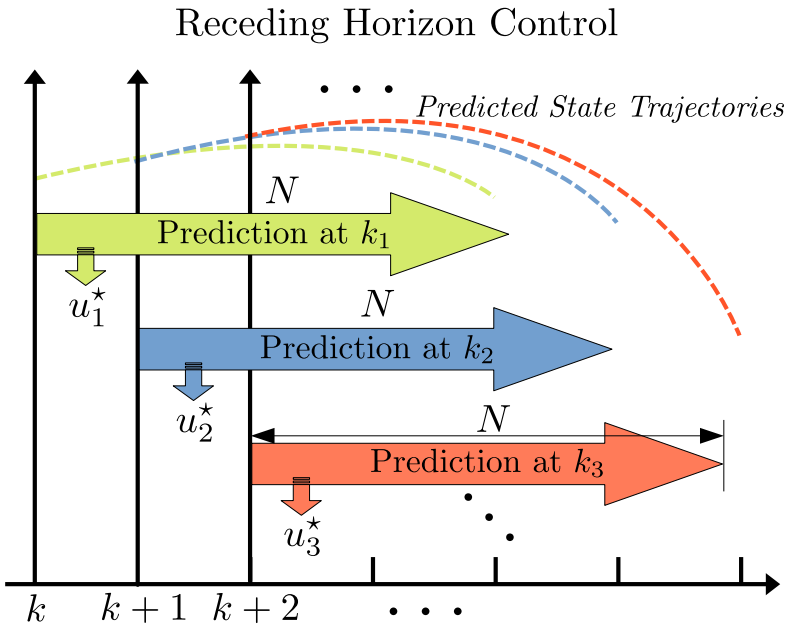



**Figure 3.6:** A visualization of the prediction horizons of a Receding Horizon Controller. The y-axes corresponds to any arbitrary state value while the x-axis denotes the time instances $k_i$. At each time instance, an optimal control sequence $\boldsymbol{u}^\star$ is obtained given a horizon length $N$ and an initial state $\boldsymbol{x}_{\boldsymbol{k_i}}$. The predicted state trajectories are also shown as the green, blue, and red lines, starting at $k+1$, $k+2$, and $k+3$, respectively.

The optimization problem to solve at each sample instance $k$ is defined by Equation 3.23, which requires a solver handling the constraints on the states and input, i.e. a mixed-integer problem, and possible nonlinearities in the model.

The requirement to handle a mixed-integer problem eliminates a large portion of the available solvers, especially coupled with the requirement of being non-proprietary or available in `MATLAB`.

Two optimization toolboxes that were deemed suitable for handling the requirements were BONMIN and OpEn which will be discussed more in detail. Apart from the difference in how the toolboxes implement the mixed-integer handling and how the problem formulation is constructed, they also differ in the optimization routine.

Given the lack of available solvers handling the requirements of the problem, an alternative optimization routine based on DP was constructed in `MATLAB`.

From here on, the receding horizon controllers are referenced to as MPCs.

### 3.4.1 Basic Open-source Nonlinear Mixed Integer Programming

**B**asic **O**pen-source **N**onlinear **M**ixed **IN**teger programming (BONMIN) is a solver developed for solving general MINLP problems [11] and is part of an ongoing project within an open-source environment, COIN-OR [25]. COIN-OR works as a database

for software components used to solve MINLP problems. Bonmin facilitates several algorithms for solving MINLP problems. The algorithm used for solving the Lotka-Volterra problem, presented in the results, is **B-OA**.

**B-OA: Outer approximation** decomposes the nonlinear part from the integer part [26]. It enables describing the solution of an MINLP problem as the intersection of an infinite collection of sets. The objective of the approximation is to provide a linear representation of an MINLP, i.e. a mixed-integer *linear* programming (MILP) problem.

Due to the unsatisfactory performance of the initial test conducted with the BONMIN solver and Lotka-Volterra problem (see section 4.1), no further endeavors were done to synthesize a predictive gear-selecting controller utilizing BONMIN as a solver.

### 3.4.2 Optimization Engine

Optimization Engine (OpEn) is an open-source code generation toolbox for optimization problems designed with real-time embedded applications in mind. It relies on a novel numerical method called PANOC, the inner workings' of which are discussed in the theory section 2.4. The compiled solver is written in the programming language Rust which is a high-performance, modern language with memory- and thread safety as its main focus points [27].

The solver can be interfaced directly using C/C++ bindings or over TCP, enabling it to be easily integrated into a C/C++ or a `SIMULINK` environment.

Figure 3.7 gives an overview of the design flow using OpEn.

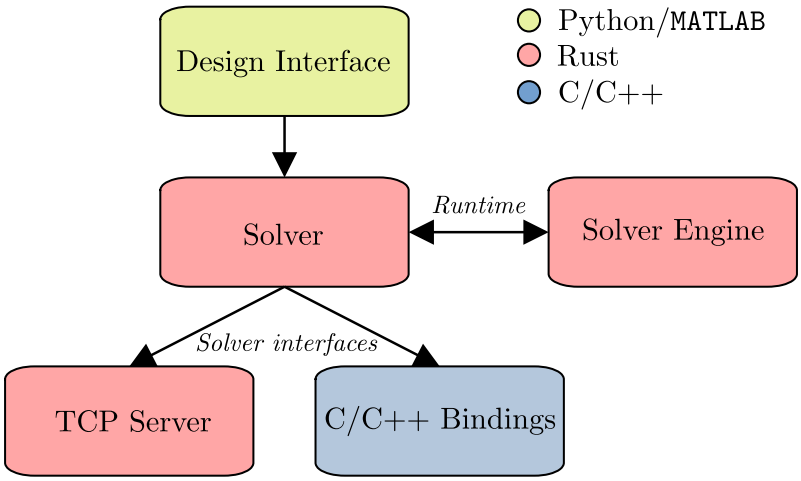



**Figure 3.7:** An overview of the OpEn design flow. The interface communicating with the solver is designed in either `MATLAB` or Python.

Using OpEn [14], it is possible to formulate a mixed-integer problem by defining a constraint $\mathcal{C}_g$ on the input signal. The constraint $\mathcal{C}_g$ is a set of finite sets defined as

$$\mathcal{C}_g = \begin{bmatrix} \mathcal{U}_g & \mathcal{U}_g & \dots & \mathcal{U}_g \end{bmatrix} \in \mathbb{R}^{\mathcal{U}_g \times N} \tag{3.24}$$

where $\mathcal{U}_g$ is the finite set consisting of the gear ratios, defined in Equation 3.23, and $N$ is the length of the horizon.

The constraint $\mathcal{C}_g$ forces the solver to choose a control input $u \in \mathcal{U}_g$ at every sample instance $k$ within the prediction horizon of length $N$, minimizing the cost function defined by the problem formulation.

Using OpEn, an MPC was developed with the structure illustrated in Figure 3.8. The solver is interfaced using the C/C++ bindings, returning an optimized control signal $u^\star$ together with the accumulated cost for the prediction.

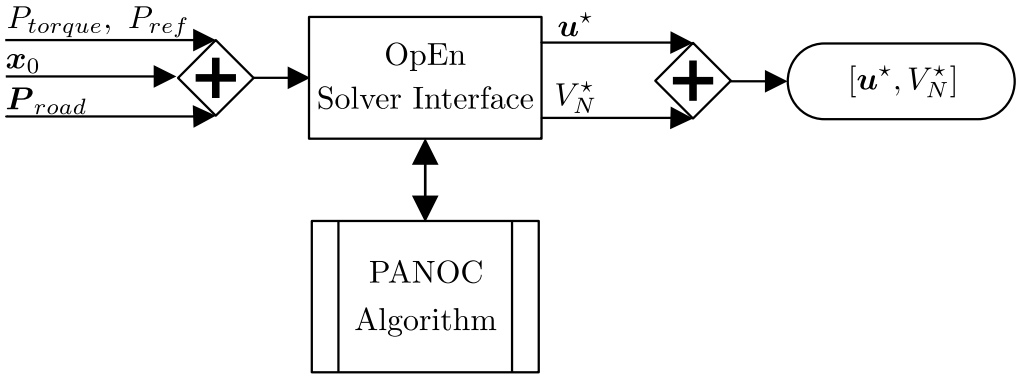



**Figure 3.8:** The control structure using OpEn as the solver for the problem formulation. It takes driver torque demand $P_{torque}$, an RPM reference $P_{ref}$, the initial state $\boldsymbol{x}_0$, and slope data $\boldsymbol{P}_{road}$ as input. The OpEn Solver Interface calls the PANOC Algorithm and returns a solution $\boldsymbol{u}^\star$ and a prediction cost $V_N^\star$.

### 3.4.3 Dynamic Programming Solver

Using the dynamic programming approach presented in section 2.6, a solver calculating the optimal control policy given the problem formulation in Equation 3.23 was developed. Based on this solver, an MPC was constructed.

Below, in Figure 3.9, is a flowchart describing the outer calculations of the proposed solver and controller. The inner calculations are included in every process block. Only the Cost-to-go calculations will be described in further detail since these are the foundation of the DP solver.

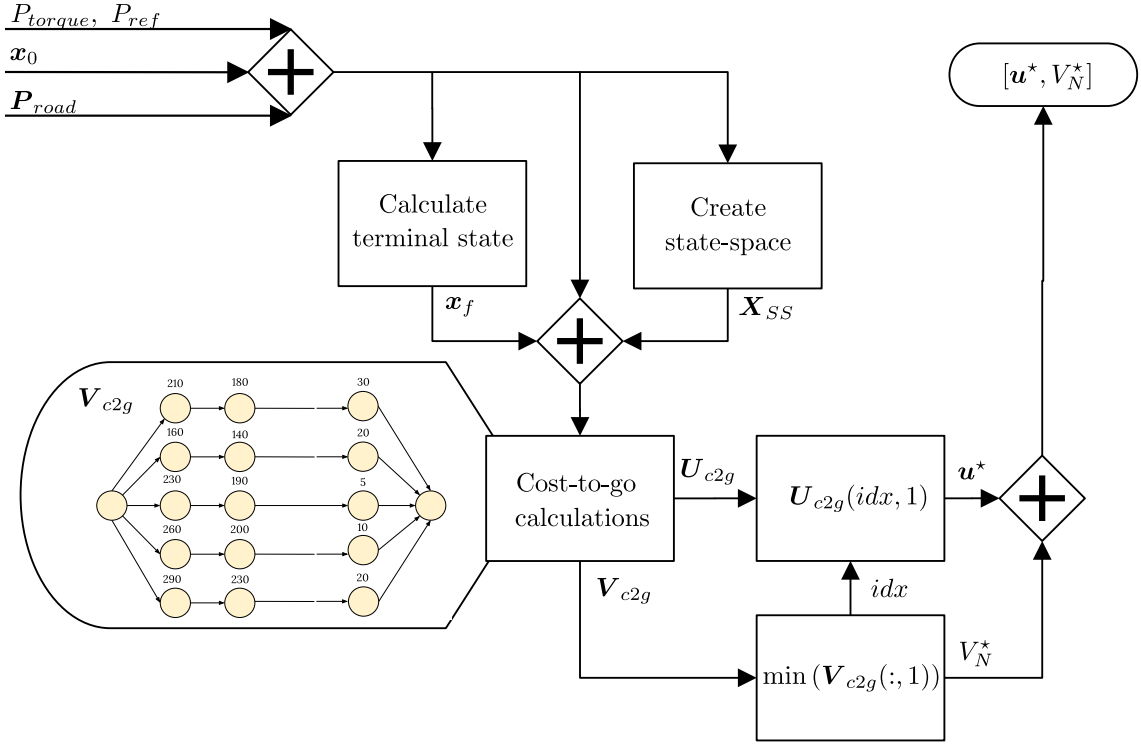



**Figure 3.9:** The control structure of the DP solver. The input to the controller is the driver torque demand $P_{torque}$, the RPM reference $P_{ref}$, the initial state $\boldsymbol{x}_0$, and the slope $N$ steps ahead, $\boldsymbol{P}_{road}$. The algorithm found in the cost-to-go calculations block is presented in algorithm 1, calculating the cost-to-go matrix $\boldsymbol{V}_{c2g}$ and the control options $\boldsymbol{U}_{c2g}$ corresponding to each entry in $\boldsymbol{V}_{c2g}$. The controller output is the optimal control sequence $\boldsymbol{u}^\star$ and the prediction cost $\boldsymbol{V}_N^\star$. The set of gears $\mathcal{U}_g$ is a global parameter set used within the cost-to-go calculations.

The input to the controller is the current vehicle state, the demanded torque from the driver, and, if available, road data describing the slope going forward. Then a terminal state $\boldsymbol{x}_f$ at the horizon $N$ is predicted using a simple if-statement-based algorithm with predetermined shift points based on the RPM of the engine. However, it still uses the same model dynamics as presented in section 3.2.

The domain, or state space $\boldsymbol{X}_{SS}$, with all reachable states going forward, is created based on the current vehicle state $\boldsymbol{x}_0$ and predicted terminal state $\boldsymbol{x}_f$, together with the horizon length $N$ and the admissible control action.

A visualization of the state space created at each iteration is shown in Figure 3.10, where each node in the graph represents the state of the vehicle, hence containing all the vehicle-specific states described in Table 3.2.

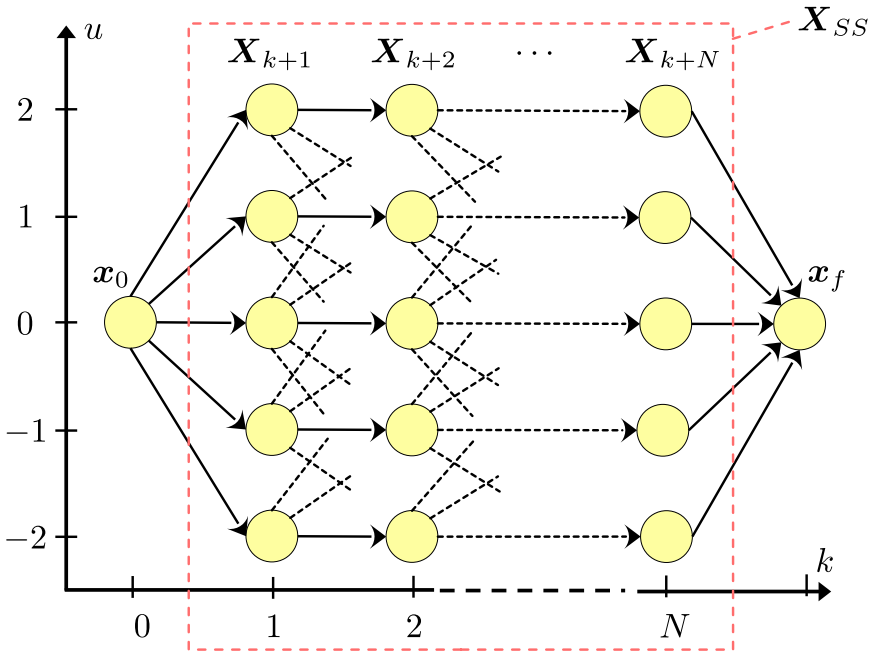



**Figure 3.10:** $5 \times N$ state space visualization. The algorithm is able to evaluate changing gears twice up or down over the horizon $N$ from the initial state $\boldsymbol{x}_0$.

With the state space defined, the cost-to-go matrix $\boldsymbol{V}_{c2g}$ and the gear matrix $U_{c2g}$, holding the gear selections for each time step in one horizon, are calculated as shown in algorithm 1.

**Algorithm 1:** Cost to go

**Data:** $\boldsymbol{x}_0, \boldsymbol{x}_f, \boldsymbol{X}_{SS}, \mathcal{U}_g, N, \boldsymbol{p}_k$
**Result:** $\boldsymbol{V}_{c2g}, \boldsymbol{U}_{c2g}$
$N_g \leftarrow \texttt{size}(\boldsymbol{X}_{SS}, 1)$; ▷ Size of gear-dimension in $\boldsymbol{X}_{SS}$.
$n_{g,tot} \leftarrow \texttt{size}(1, \mathcal{U}_g)$; ▷ Total number of gears in gearbox.
$\Delta_g \leftarrow \left\lfloor \frac{N_g}{2} \right\rfloor$; ▷ Gear changes up/down from initial gear.
$\boldsymbol{V}_{c2g} \leftarrow NaN_{N_g \times N}$;
$\boldsymbol{U}_{c2g} \leftarrow \boldsymbol{0}_{N_g \times N}$;
$idx_{g,\boldsymbol{x}_0} \leftarrow \texttt{find}(\mathcal{U}_g == x_{4,0})$; ▷ Find $\boldsymbol{x}_0$'s gear index in $\mathcal{U}_g$.
$k \leftarrow N$;
**while** $k > 0$ **do**
  **for** $i = 1 : N_g$ **do**
    $\boldsymbol{x}_{ik} \leftarrow \boldsymbol{X}_{SS}(i, k)$;
    **if** $k == N$ **then**
      $\boldsymbol{V}_{c2g}(i, k) \leftarrow \ell(\boldsymbol{x}_{ik}, \boldsymbol{p}_k, \boldsymbol{x}_0) + \ell_N(\boldsymbol{x}_{ik}, \boldsymbol{p}_k, \boldsymbol{x}_0, \boldsymbol{x}_f)$;
      $\boldsymbol{U}_{c2g}(i, k) \leftarrow x_{4,ik}$;
    **else**
      $\hat{V} \leftarrow 0$;
      $\boldsymbol{U}_{c2g}(i, k) \leftarrow x_{4,ik}$;
      $C_{SS,k+1} \leftarrow \boldsymbol{X}_{SS}(:, k+1)$; ▷ Extract the column of states one step ahead.
      **for** $g = \max(1, idx_{g,\boldsymbol{x}_0} - \Delta_g) : \min(idx_{g,\boldsymbol{x}_0} + \Delta_g, n_{g,tot})$ **do**
        $selGear \leftarrow \mathcal{U}_g(g)$;
        $idx_{V_{c2g,k+1}} \leftarrow 0$;
        **for** $s = 1 : N_g$ **do**
          $\boldsymbol{x}_{s,k+1} \leftarrow C_{SS,k+1}(s)$;
          **if** $x_{5,s,k+1} == selGear$ **then**
            $idx_{V_{c2g,k+1}} \leftarrow s$;
            `break`;
          **end**
        **end**
        $\hat{V} \leftarrow \ell(\boldsymbol{x}_{ik}, \boldsymbol{p}, \boldsymbol{x}_0) + \boldsymbol{V}_{c2g}(idx_{V_{c2g,k+1}}, k+1)$;
        **if** $\hat{V} < \boldsymbol{V}_{c2g}(i, k)$ **then**
          $\boldsymbol{V}_{c2g}(i, k) \leftarrow \hat{V}$;
        **end**
      **end**
    **end**
  **end**
  $k--$;
**end**

The input to the algorithm calculating the cost-to-go matrix $\boldsymbol{V}_{c2g}$ is the initial state $\boldsymbol{x}_0$, the terminal state $\boldsymbol{x}_f$, the state space $\boldsymbol{X}_{ss}$, the array of gear ratios $\mathcal{U}_g$ which values and size depends on which gearbox is used, and the horizon length $N$.

It returns the cost to go matrix $\boldsymbol{V}_{c2g}$ and the control matrix $\boldsymbol{U}_{c2g}$ corresponding to

the choice of gear for each entry in $\boldsymbol{V}_{c2g}$.

$\boldsymbol{V}_{c2g}$ holds the accumulated cost going from the terminal state to the state $\boldsymbol{x}_{ik}$ in $\boldsymbol{X}_{SS}$. This means that after having run the cost-to-go algorithm, it is enough to look at the first column of the cost-to-go matrix $\boldsymbol{V}_{c2g}(:, 1)$ and extract the index of that column having the lowest accumulated cost.

This index is then used with the cost-to-go control matrix to extract the optimal gear going forward. This is illustrated by the min-block in Figure 3.9, returning the optimal gear index as well as the accumulated cost corresponding to that control.

## 3.5 Performance metrics

To evaluate the performance of the different proposed algorithms across different hardware, a performance metric that decouples the algorithm and hardware has to be used.

Common ways of doing this include Millions of instructions per second (MIPS) and Floating Point Operations per second (FLOPS), which combined with the run-time gives an estimate of how many instructions or operations are executed by the algorithm.

Given the data types used by the algorithms, and that different hardware vendors implement arithmetic's using different instruction sets, FLOPS is used to evaluate the performance of the algorithms as it is a measurement of how many floating point operations is executed regardless of how many instructions may lay behind.

FLOPS is defined by

$$\text{FLOPS} = \text{Cores} \times \frac{\text{Cycles}}{\text{Seconds}} \times \frac{\text{FLOPs}}{\text{Cycle}}. \tag{3.25}$$

FLOPS for specific hardware can either be calculated using a purpose-built program or estimated using Equation 3.25 and the specification sheet for the hardware.

To estimate the performance of each algorithm, the FLOPS for an Intel i7-8750H and an ARM Cortex A53 is measured using a software library called LINPACK [28], with the results presented in Table 3.3.

| Hardware | Cores | FLOPS |
|---|---|---|
| i7-8750H | 6 | $22.6 \cdot 10^9$ |
| ARM Cortex A53 | 4 | $0.29 \cdot 10^9$ |

**Table 3.3:** Hardware used to compare the run-time of the algorithms on embedded hardware.

As the specific hardware used in the TCU used by Volvo Trucks is not available for publishing, the ARM Cortex A53 processor is used to estimate the run-time of the algorithms on a more embedded-oriented processor.

To calculate the estimated run-time on the ARM cortex A53 processor, the algorithms are first executed on the Intel i7-8750H processor and timed as $T_{\text{i7}}$. The

number of operations is then calculated and an estimate of the run-time, $T_{\text{A53}}$, on the ARM processor is calculated by

$$T_{\text{A53}} = \frac{\frac{\text{i7}_{\text{FLOPS}}}{\text{i7}_{\text{cores}}}}{\frac{\text{A53}_{\text{FLOPS}}}{\text{A53}_{\text{cores}}}} \cdot T_{\text{i7}} \tag{3.26}$$

## 3.6 Simulation

To simulate the different developed control structures in a cohesive manner, a simulation environment was developed in `SIMULINK` with interfaces for the DP-based and OpEn solver. The simulation environment is constructed to resemble the equivalent found on Volvo in a substantially scaled-down way, yet capturing the vehicle dynamics in an adequate way.

The layout, shown in Figure 3.11, is divided into three main sections. The left area of the top section has a switch that can change between the developed gear-selecting strategy or a conventional gear-selecting strategy. The top right area has four switches that can be adjusted to create different simulation scenarios.

The first lever changes between the driver mode and the constant torque mode. The driver mode is based on a simple PI-regulator that regulates the current vehicle velocity to the *setSpeed* variable, which is defined outside of the simulation environment and fed into the *TorqueSelector* block. The constant torque mode can be modified to a triangle-shaped input using the second lever.

The Gear Control section in the middle handles everything connected to the gear shifts. It also reads the predefined slope data from the workspace. The *GearShift-Controller* block contains the interfaces to the respective solver along with the shift map utilized when running in the conventional gear-selecting strategy mode.

The slope data is generated using different alterations of a sinus function together with flat road sections to form the defined test scenarios. This can easily be replaced with slope data gathered from real-world road data.

Lastly, the Vehicle Model section is used for simulating the vehicle dynamics as well as logging the data necessary for the analysis of the gear-selecting strategy.

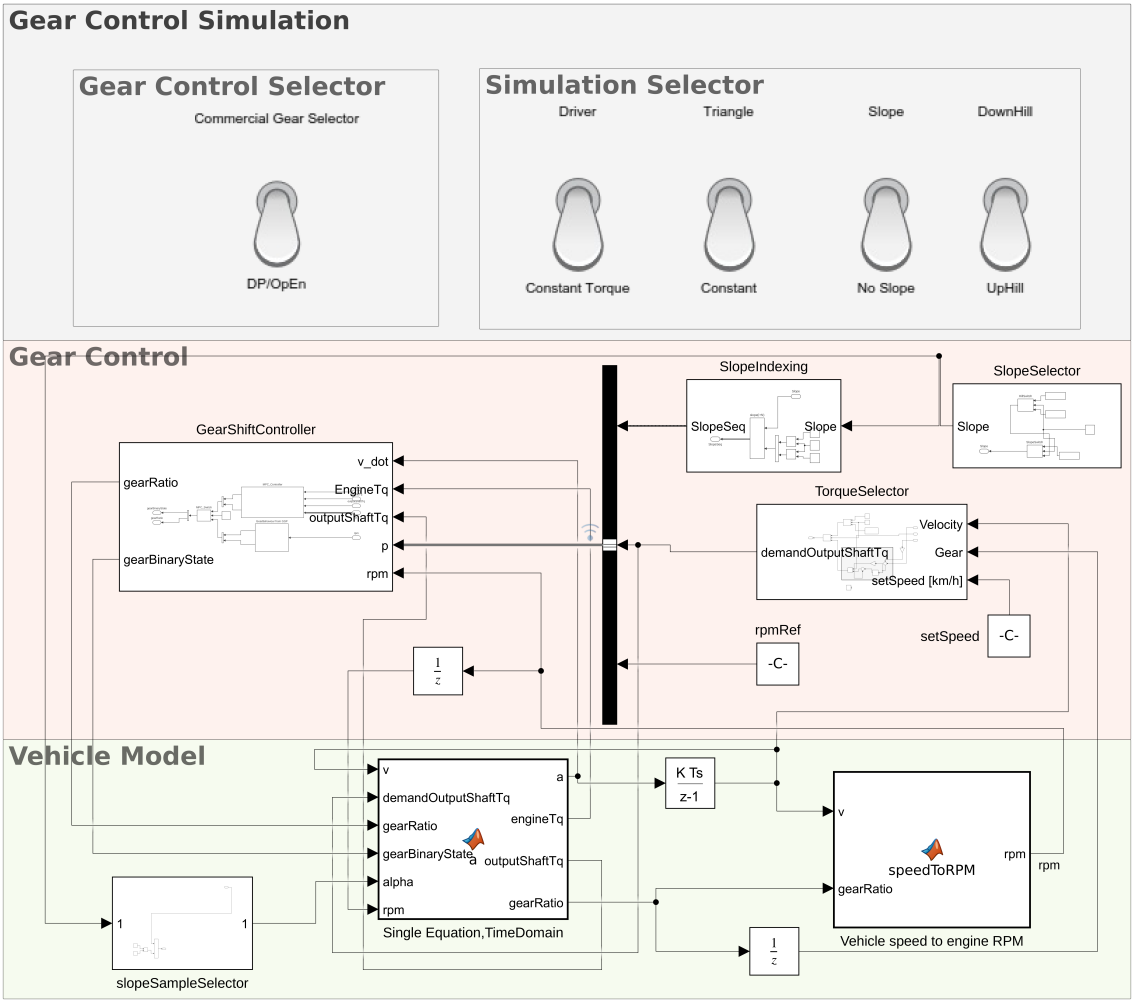



**Figure 3.11:** Simulation layout in `SIMULINK` used to evaluate the different gear control strategies and create the results.

# 4

# Results and Discussion

The results are presented in chronological order reflecting the workflow of the project. After each result there is a discussion about the obtained results and observations.

The first result to be presented is the solver comparison where the two identified non-proprietary solvers are benchmarked using a simple Lotka-Volterra problem to test the performance before continuing with the gear-selecting controllers.

With the choice of solver concluded, the next result to be presented is the validation results from the simulation environment described in section 3.6, where it is compared with a similar Volvo environment. The results to be shown thereafter are simulations highlighting the difference between modeling the dynamics in time and space domain and identifying the strengths and weaknesses of the two models.

Following the domain-specific results are the main bulk of results, which are coupled with the objectives described in section 1.1. Here are the results from the proposed MPCs utilizing the OpEn solver and the dynamic programming solver described in subsection 3.4.2 and subsection 3.4.3. The two controllers are tested in various scenarios, mainly to benchmark them in terms of optimality and secondly to test the robustness of each controller. Note that they are both referred to as MPCs with the distinction that they use different solvers to solve the optimization problem.

Concluding the result section, the times it takes for the OpEn and dynamic programming solver to calculate the optimized gear selection are examined. The solve time is then extrapolated to more embedded-oriented hardware using the FLOPS calculations described in section 3.5, to get an estimation of how the respective solver would cope in an embedded application.

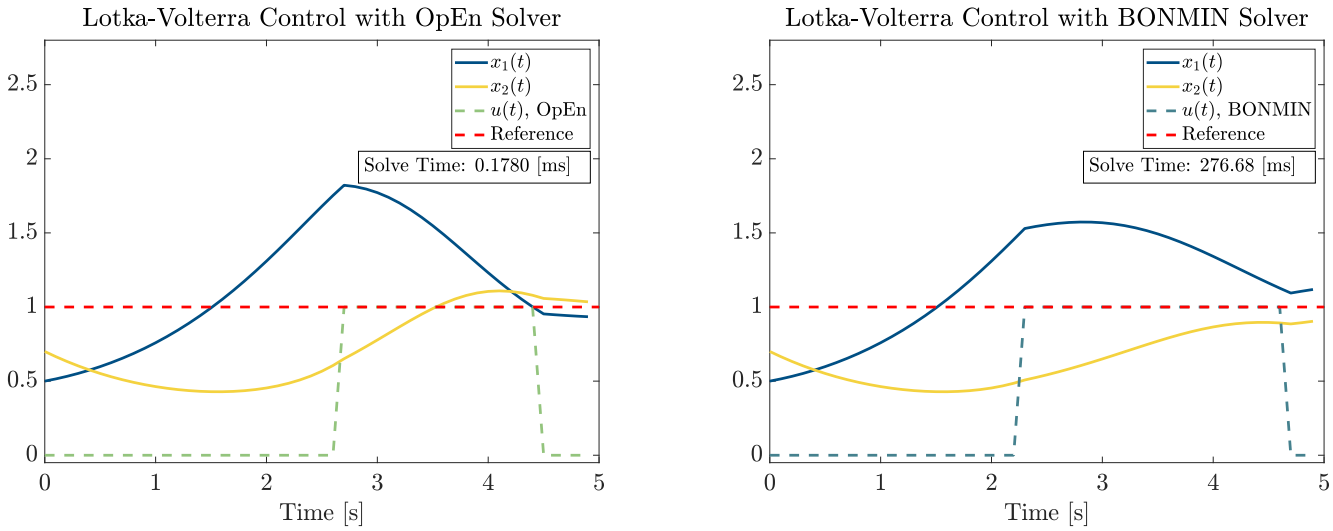
## 4.1 Solver Comparison

As part of identifying appropriate solvers for the specific optimization problem, Equation 3.23, the identified non-proprietary solvers were benchmarked using a Lotka-Volterra problem with the dynamics defined as:

$$\begin{aligned} \dot{x}_1 &= x_1 x_2 - c_1 x_1 u \\ \dot{x}_2 &= -x_2 x_1 - c_1 x_2 u. \end{aligned} \tag{4.1}$$

where $c_1 = 0.4$, $c_2 = 0.2$ and $u \in \{0, 1\}$. The resulting problem, therefore, constituted a mixed-integer and non-linear problem. As described in section 3.4, the available solvers for the problem at hand found were the OpEn and BONMIN solvers.

Applying the problem formulation with the dynamics as in Equation 4.1 and regulating towards a reference $x_1 = 1$ and $x_2 = 1$, the two solvers yielded the results according in Figure 4.1 below. The total solve time per solution was measured for each solver running a five-second simulation for the problem.




**(a)** **(b)**

**Figure 4.1:** State trajectories and control sequence of the Lotka-Volterra simulations using OpEn and Bonmin as solvers. $x_1$ and $x_2$ are the different model states, $u$ is the control signal obtained from the different solvers, and the red stretched line is the reference for both states.

According to Figure 4.1, the two solvers yielded quite similar results for the optimization problem in terms of solutions for the trajectory of $x_1$ and $x_2$, with similar control actions on $u$.

The main difference between the two solvers is in the solve time, where OpEn is roughly a factor $10^3$ times faster than BONMIN.

Given the obtained results in terms of total solve time for the OpEn and BONMIN solvers, the OpEn solver was selected for the MPC running a non-proprietary solver. Hence, OpEn was the solver used to obtain the results while comparing it to the MPC utilizing the dynamic programming solver.

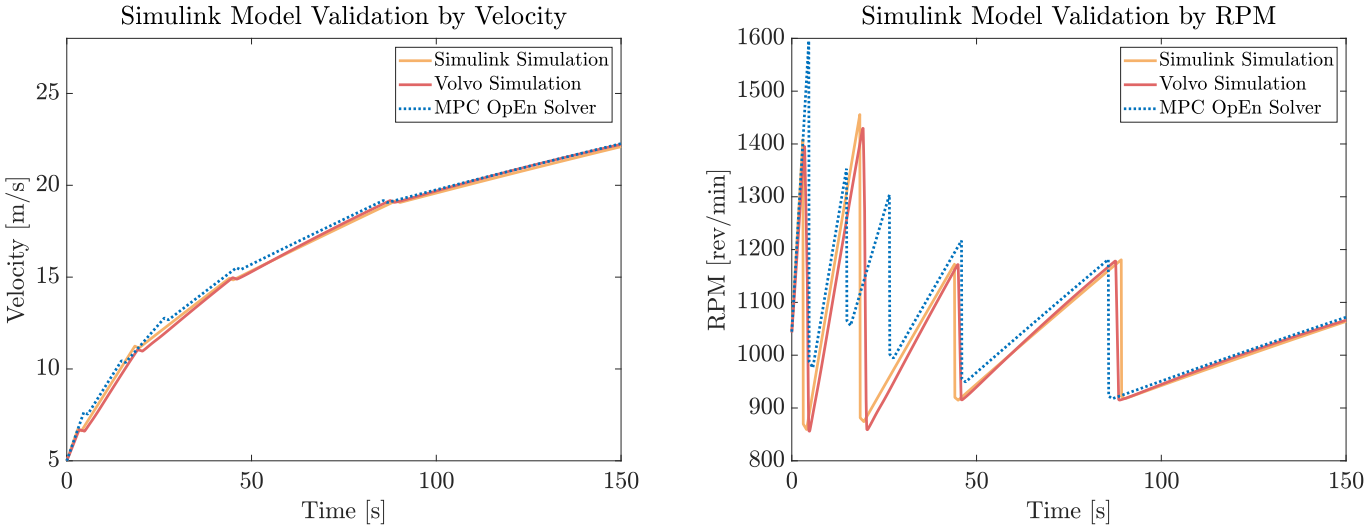
## 4.2 Simulation Environments

As mentioned in section 3.6, a simulation environment was constructed in `SIMULINK` to test the different proposed controllers in various scenarios, with the motivation to not conduct them in Volvo's dito described in section 3.1.

To validate the constructed simulation environment, a comparison between the `SIMULINK` environment and one of Volvo's environments was conducted, with the results in Figure 4.2.

The results were obtained by setting a constant driver-demanded torque and an initial velocity of 5 m/s for both environments. To replicate the gear-selecting strategy from the Volvo simulation in the `SIMULINK` equivalent, a look-up table holding the RPM values for the shift points for the different gears was constructed.

Along with the comparison between the two environments, a simulation in the `SIMULINK` environment running the MPC with OpEn as the solver, was conducted to showcase the similarities to the gear-selecting strategy inflicted by Volvo's simulation environment. The OpEn solver solution can be observed by the dotted blue line in Figure 4.2.




**(a)** **(b)**

**Figure 4.2:** Results from comparing the developed simulation environment to one of Volvo's internal simulation environments. The solution from the OpEn-based controller given the same conditions is also plotted as the blue dotted line to show the potential of using an MPC for gear selections instead. The scenario consists of an initial velocity of 5 m/s, a constant engine torque of 1000 N, and an initial gear ratio of 2.69.

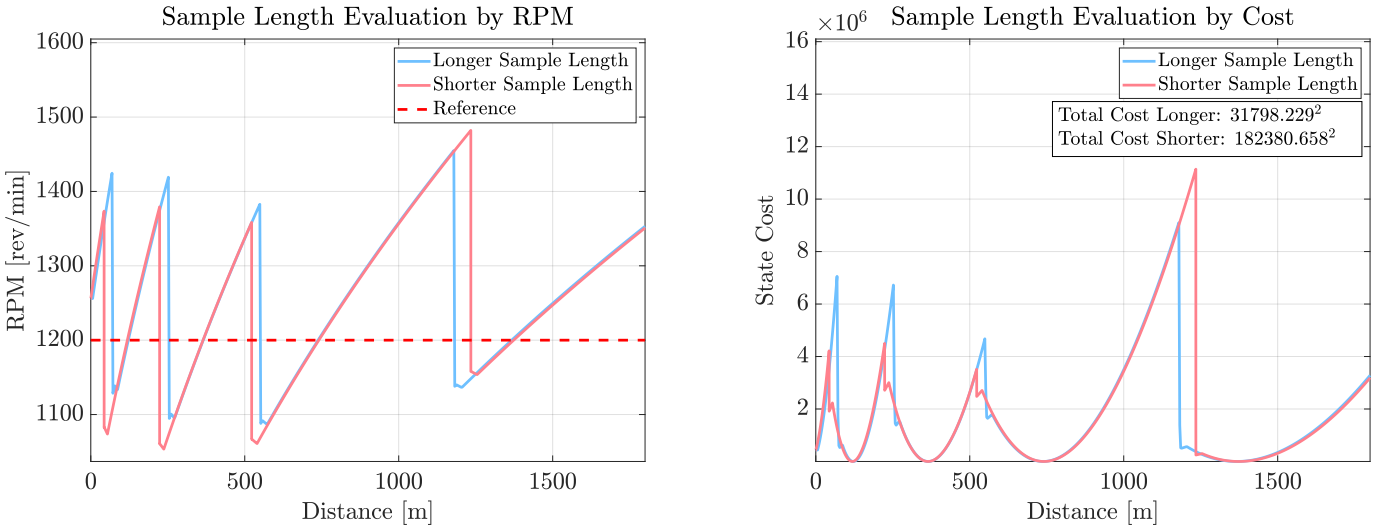
Observing the results in Figure 4.2, given the same demanded constant torque and initial values, the `SIMULINK` and Volvo simulation environments shows the same behavior. As a result, this validates the use of the `SIMULINK` environment as the main simulation environment for the proposed controllers.

Observing the behavior of the MPC with OpEn as the solver, it can be concluded that it mainly deviates from the controller actions in the Volvo simulation during low velocities and almost converges at velocities exceeding 15 m/s. The larger deviations at the lower velocity range can be derived from the dynamic $Q_{factor}$ implemented in the problem formulation for the OpEn solver, described in section 3.3, which slackens the penalization from the most efficient engine RPM at lower velocities to achieve a higher acceleration from a standstill.

## 4.3 Space Domain Evaluation

As described in section 3.2, the dynamics of the vehicle can be modeled in either time or space domain. To evaluate which domain was best suited for the MPC, a test scenario in space domain was conducted. The scenario consisted of a flat road with an initial velocity of 5 m/s and an initial gear ratio of 2.69, coupled with a constant demanded output shaft torque of 2000 N.

Controlling the gear sequence during the test scenario were two MPCs using OpEn as the solver. The only difference in the controllers was the sample length they utilized when discretizing the vehicle dynamics, with one being ten times longer than the other. The results for the two controllers can be observed in Figure 4.3.




**(a)** **(b)**

**Figure 4.3:** Space domain-based OpEn solvers running a short and a long sample length during an acceleration scenario. The vehicle simulation starts with an initial velocity of 5 m/s, an initial gear ratio of 2.69, with a constant driver demand output shaft torque of 2000 N. The RPM reference was set to 1200 rev/min.

From Figure 4.3a it can be seen that for low velocities, the solver with a short sample length manages to follow the RPM reference slightly better than the solver with a long sample length and vice versa for higher velocities after approximately 750 meters.

The same phenomenon is confirmed in Figure 4.3b, where at low velocities, the controller utilizing a short sample length has a lower cost compared to the controller with a long sample length. The opposite is true at high velocities where the controller with a long sample length has a lower cost.

This can be explained by taking the controller with a long sample length as an example. At low velocities and a relatively long sample length, the continuous dynamics are not accurately described by the discretized dynamics. The sample length is too long resulting in a suboptimal prediction and control actions with a

higher cost as a result.

On the other hand, at high velocities, the longer sample length is adequate to describe the continuous dynamics and due to the fixed prediction horizon $N$, results in a longer prediction horizon expressed in meters and a better prediction and control action with a lower cost compared to the controller with a short sample length.

Concluding the results in Figure 4.3, modeling the vehicle dynamics in space domain requires a sample length short enough to accurately describe the continuous dynamics at the lowest expected velocity, which at higher velocity will be excessive in terms of capturing the continuous dynamics, where it would be more beneficial to have a longer sample time yielding a longer prediction horizon. An alternative to this contradiction is to have two or more controllers with different sample lengths and switch between them according to the velocity of the vehicle.

Instead of having two controllers or more, acting at different velocities, the proposed MPC will be modeled in time domain since the sample length in terms of distance scales with velocity. The drawbacks of doing so are that the equations describing the dynamics are nonlinear, as discussed in section 3.2, and that the road data probably has to be transformed to time domain as it is often expressed in space domain. On the contrary, the advantage of modeling in time domain is that the equations dictating the behavior of the binary states are more easily expressed in time, the motivation is that the length of a gear shift is expressed by the duration in time and not in the distance in meters, hence, the equations of the binary states in time domain has no dependency on the velocity of the vehicle.

## 4.4 Road Data

To confirm that the proposed MPCs act upon the provided road data containing the slope of the road ahead, a scenario was created where the vehicle traveling at constant velocity was subdued to an uphill road. The initial velocity was set to 13.9 m/s, a constant driver demand output shaft torque was given as 1500 N, and the initial gear ratio was set to 1.63. The output shaft torque was set slightly lower compared to the previous tests such that the inflicted slope affected the velocity in such a manner that a gear change was evoked.

The prediction cost, namely the accumulated cost for each prediction at each time step, for the two controllers utilizing the OpEn and the dynamic programming solver, respectively, can be observed in Figure 4.4. Each controller was subdued by the same uphill over two scenarios, one with road data available and one where it was not available. During the latter scenario, the slope data sent to the solver consists of zeros.

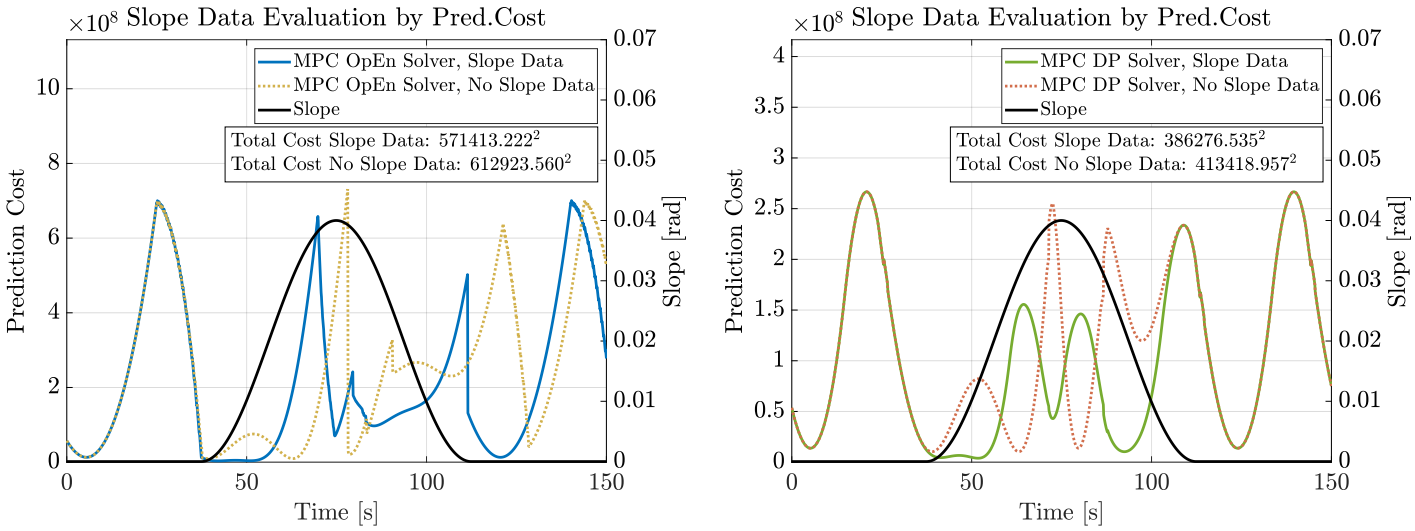



**Figure 4.4:** A comparison of prediction costs for the OpEn and DP solvers provided slope data and no slope data. The initial velocity was set to 13.9 m/s with a constant output shaft torque of 1500 N and an initial gear ratio of 1.63.

Observing the results in Figure 4.4, both controllers exhibit a lower prediction cost when exposed to the uphill with road data available. This confirmed that both controllers act upon the road data provided, utilizing it to produce a better prediction with a lower prediction cost.

In addition, observing the cost for the controller utilizing OpEn as the solver, it can be noted that the cost was higher than for the controller utilizing the dynamic programming solver.

## 4.5 Base Scenario

With the results concluding which non-proprietary solver to utilize, which domain was the most advantageous, and confirming that the solvers produce a lower prediction cost with available road data, it is time to present the results moving towards a more realistic scenario.

The first test scenario was a base scenario, consisting of a flat road with a constant demanded output shaft torque of 2000 N from the driver, and an initial velocity of 5 m/s.

The result from the two proposed controllers utilizing the OpEn and dynamic programming solver can be observed in Figure 4.5.

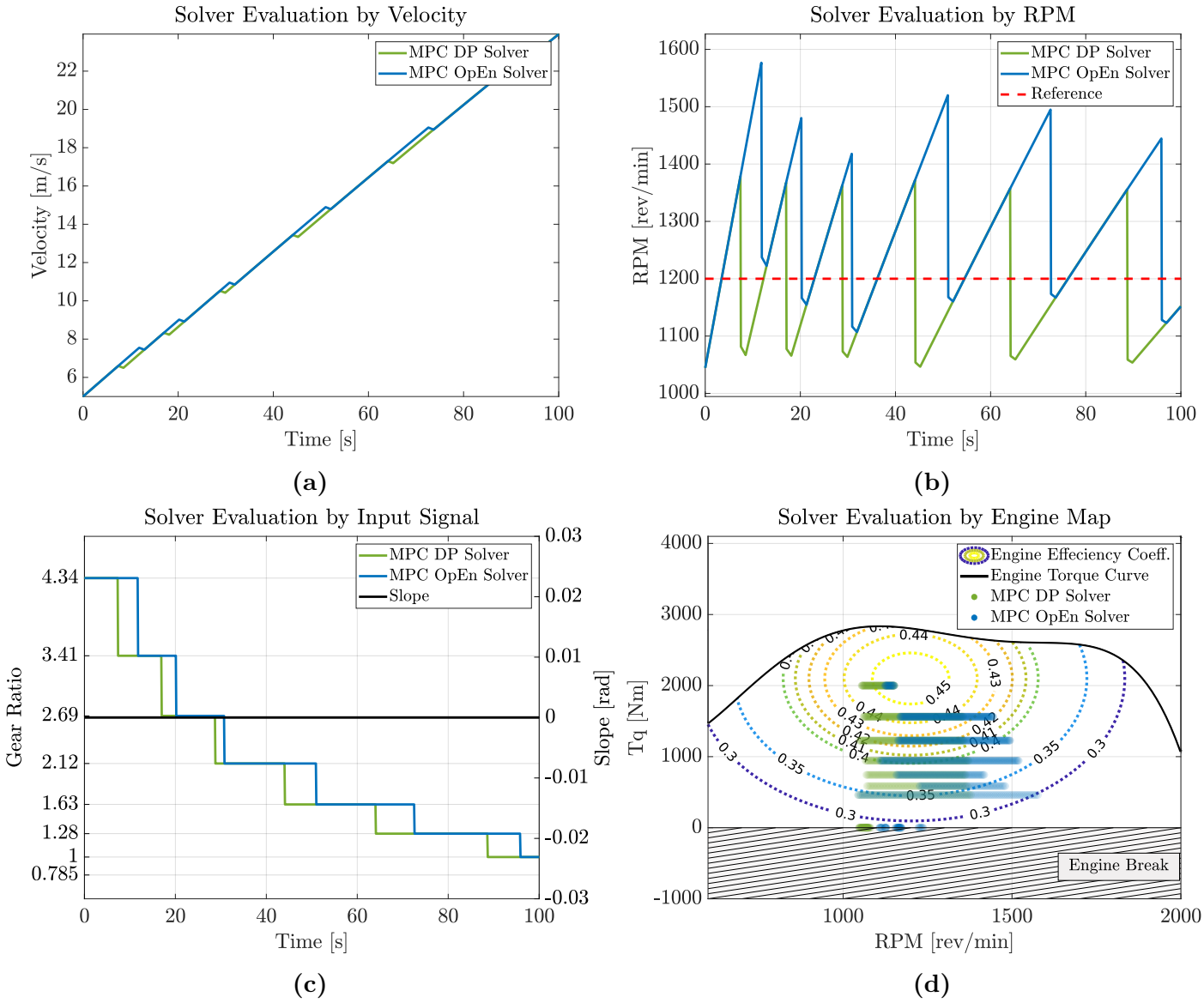



**Figure 4.5:** Results from the base scenario based on an acceleration phase with an initial velocity of 5 m/s, an initial gear ratio of 4.34, and a constant output shaft torque set to 2000 N.

From Figure 4.5c, it can be concluded that the dynamic programming solver was keen to change gear earlier than the OpEn solver, resulting from the behavior to more strictly follow the RPM reference, as can be observed in Figure 4.5b.

Although the different solvers change gears at different time instances, they reach the same velocity at the end of the simulation, as can be seen in Figure 4.5a. Looking at the Engine Map in Figure 4.5d it is apparent that the DP-based solver sustains a higher efficiency during the simulation.

The state cost for the different controllers running the base scenarios is illustrated in Figure 4.6.

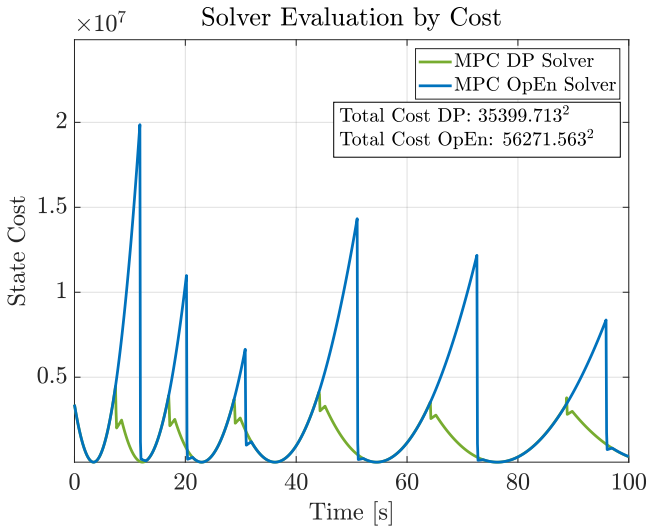



**Figure 4.6:** The intermediate state cost and accumulated total cost per solver for the base scenario.

Observing the state cost for the two controllers, the controller utilizing the dynamic programming solver exhibits a lower state cost across the whole duration of the scenario. In the worst case, during the first gear change, the intermediate cost of the OpEn solver was larger by a factor of four.

## 4.6 Robustness

In most cases, the driver would like to reach or maintain a certain velocity while driving. To evaluate how the controllers handle this scenario a PI-regulator was constructed simulating a driver aiming to hold a certain vehicle velocity. The PI regulation of the vehicle velocity was conducted using the demanded output shaft torque as a control input, translating to different pedal positions from the driver.

Together with a non-constant demanded torque from the driver, the robustness of the controllers was also evaluated when exposed to a road profile containing an uphill and a downhill.

### 4.6.1 Acceleration Scenario

The first scenario evaluating the robustness of the controllers was an acceleration scenario. It was conducted by simulating a driver aiming for a reference velocity of 11 m/s starting from an initial velocity of 5 m/s. The results from this scenario can be observed in Figure 4.7.

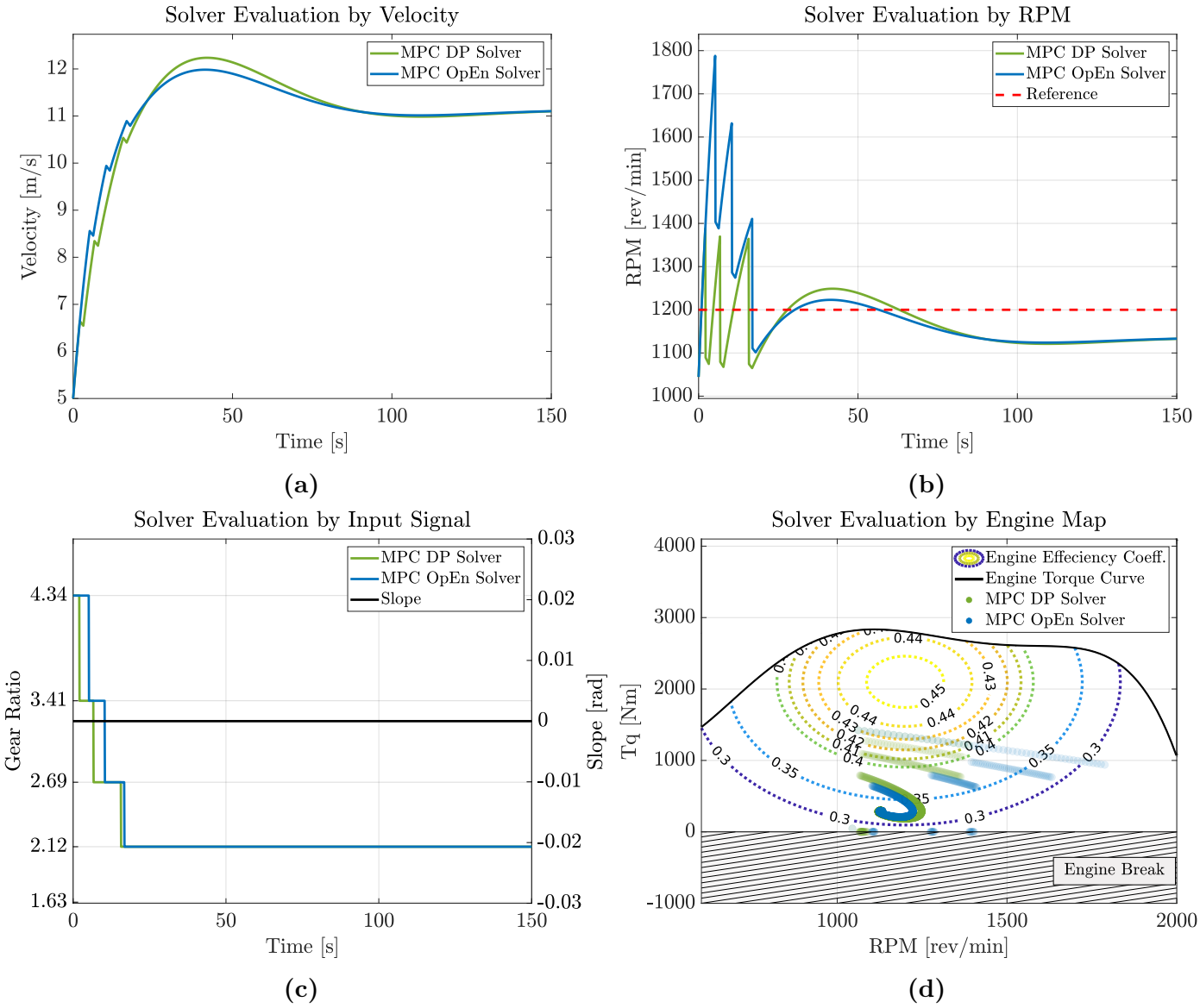



**Figure 4.7:** Results from a robustness test accelerating the vehicle from an initial velocity of 5 m/s to a reference velocity of 11 m/s. The driver demand output shaft torque was simulated using a PI regulator which converts the velocity difference from the reference to output shaft torque. The initial gear ratio was set to 4.34 and the RPM reference was set to 1200 rev/min.

Observing Figure 4.7a, it can be noted that both controllers, utilizing the different solvers, manage to handle the non-constant driver-demanded torque resulting from a driver aiming to accelerate up to 11 m/s. Hence, both controllers exhibit robustness in terms of handling a varying demanded torque from the driver.

Figure 4.7b shows that the controller utilizing OpEn as the solver was more prone to let the engine rev to higher RPMs whilst the controller using the dynamic programming solver follows the RPM reference more steadily. This manifests in gear shifts at an earlier stage as observed in Figure 4.7c and a trajectory closer to the most efficient RPM observed in Figure 4.7d.

The state cost for the two controllers can be observed in Figure 4.8. As for the base scenario, the controller utilizing OpEn as the solver exhibits a higher state cost compared to the dynamic programming dito, resulting from the controller letting the engine RPM be at a higher level.

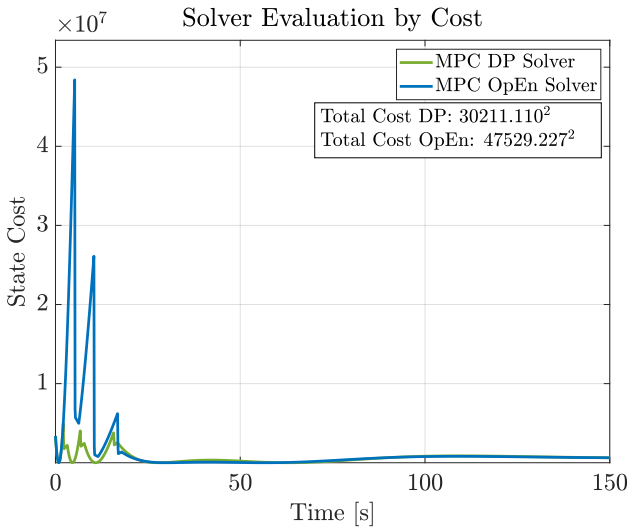



**Figure 4.8:** The intermediate state cost and accumulated total cost for the different controllers during the acceleration scenario.

### 4.6.2 Slope Scenario

To further investigate the robustness of the controllers, a slope scenario was constructed. The scenario consists of an uphill followed by a downhill. The combined slopes are succeeded or followed by a flat section.

The initial velocity and target velocity for the driver are both set to 13.8 m/s. The results from the scenario are shown in Figure 4.9.

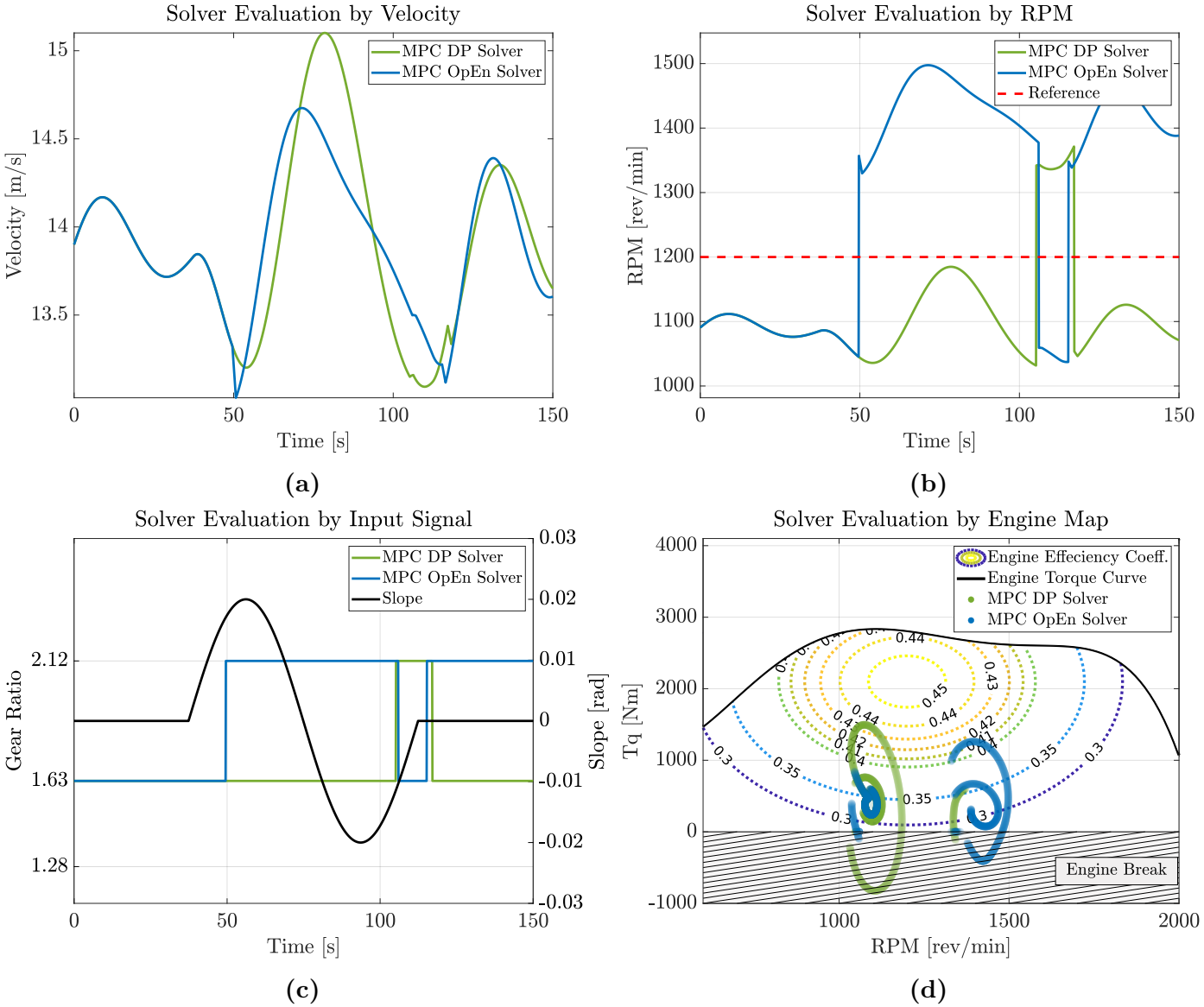



**Figure 4.9:** Results from the combined uphill and downhill scenario given an initial and reference velocity of 13.9 m/s, an initial gear ratio of 1.63, and a PI regulator controlling the output shaft torque based on the velocity deviation from the reference velocity.

Observing Figure 4.9b, it can be noted that the controller utilizing the dynamic programming solver once again manages to stay closer to the RPM reference than

the OpEn-based controller. This translates to the more optimized trajectory for the dynamic programming solver observed in the engine map Figure 4.9d.

Observing Figure 4.9a, the driver controller with the underlying gear controller utilizing OpEn as the solver manages to follow the set velocity slightly better.

To conclude, the results show that both controllers exhibit robustness in terms of the ability to handle a varying demanded torque and slope. Observing the control sequence from both controllers, they produced different optimized trajectories as can be observed in Figure 4.9b, where the OpEn-based controller shifts up while the dynamic programming-based controller keeps the gear. When the OpEn-based controller then shifts down, the dynamic programming-based solver shifts up.

The state cost from the scenario can be observed in Figure 4.10. The controller utilizing the dynamic programming solver manages to minimize the state cost in a more effective manner than the controller utilizing OpEn.

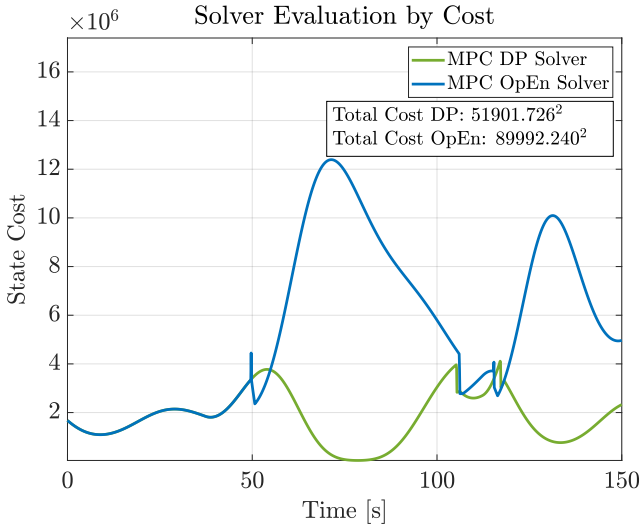



**Figure 4.10:** The intermediate state cost and accumulated total cost during the slope scenario.

In conclusion, the scenario-based tests conducted with the two MPCs, the controller utilizing a dynamic programming-based solver always obtained a control sequence that had a lower state cost compared to the controller using OpEn as the solver. This was despite the fact that the controllers had the same tunings in terms of $Q_{factor}$, $R_1$ and $R_2$ together with the same stage cost $\ell_k$ and terminal cost $\ell_N$.

While the dynamic programming solver exhibits a more optimized solution in all of the conducted scenarios, the solve time compared to the OpEn solver was higher as will be shown in the next section.

## 4.7 Embedded Environment

To estimate the performance of the different solvers in terms of solve time in an embedded environment, the number of operations conducted by the computer running the algorithms was estimated using the performance metric FLOPS as described in section 3.5.

The estimated solve times in an embedded environment were based on the solve time for the slope scenario in subsection 4.6.2, running on an Intel i7-8750H processor, presented in Figure 4.11.

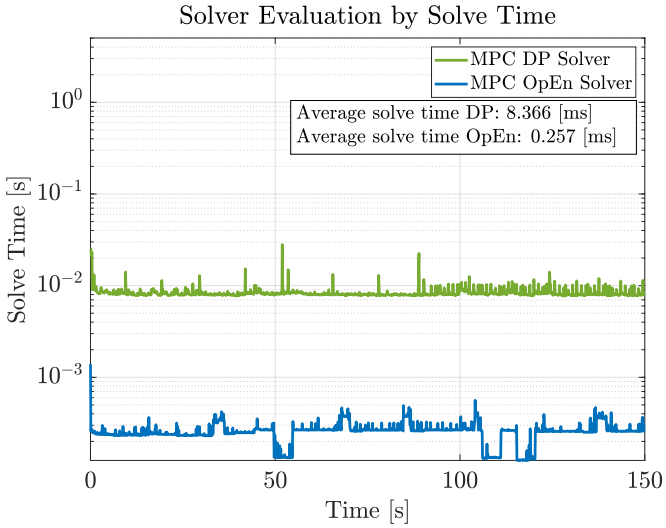



**Figure 4.11:** Solve time for the controllers utilizing the different solvers. The solve time reflects the time it takes for the solver to find a solution to the optimization problem. In line with the receding horizon principle, this was repeated at each sample instance.

Observing Figure 4.11, the OpEn solver was faster than the constructed dynamic programming solver, by roughly one order of magnitude. This was confirmed for all of the conducted scenarios; for brevity only the solve time for the slope scenario is shown.

Using the measured FLOPS on the computer running the scenario, the number of operations needed to run each algorithm was estimated and is presented in Table 4.1.

| Hardware | Solver | FLOPS | Solve Time [ms] |
|---|---|---|---|
| i7-8750H | OpEn | $9.704 \cdot 10^{5}$ | 0.257 |
| i7-8750H | DP | $3.159 \cdot 10^{7}$ | 8.366 |

**Table 4.1:** The solve time and the number of operations conducted by each solver, running the slope scenario on an Intel i7-8750H processor.

Using Equation 3.26 together with the results obtained in Table 4.1, the estimated solve time on the embedded oriented processor ARM Cortex A53 is presented in Table 4.2.

| Hardware | Solver | FLOPS | Est. Solve Time [ms] |
|---|---|---|---|
| ARM Cortex A53 | OpEn | $9.704 \cdot 10^5$ | 12.982 |
| ARM Cortex A53 | DP | $3.159 \cdot 10^7$ | 422.590 |

**Table 4.2:** The estimated solve time and number of operations conducted by each solver, running the slope scenario on an ARM Cortex A53 processor.

The estimated solve time on the ARM Cortex A53 processor was 12.982 ms and 422.590 ms for the OpEn and dynamic programming solver respectively. While the feasibility of implementing the solvers on embedded hardware in terms of solve time is up to the designer and the specific problem at hand, the results in Table 4.2 shows that especially the OpEn solver exhibits a solve time that could be deemed as feasible for a gear-selecting strategy.

The results for the dynamic programming solver should be observed in the light that no effort was conducted to optimize the routine and that `MATLAB` code does not result in compiled binaries in the same way as the Rust code for the OpEn solver.

It is therefore likely that the dynamic programming solver would have a higher performance in terms of solve time if it was written in a compiled language such as C or C++ together with more efficient code.

In conclusion, especially the OpEn solver is deemed feasible for an embedded application, with the DP-based solver deemed feasible but in need of a more optimized and compiled algorithm before being implemented in an embedded application.

## 4.8 Versatility

The versatility of the proposed controllers was evaluated in terms of the ease to change the behavior of the controller, the convenience to transfer it to another vehicle, and the ability to have control of the optimization routine.

Starting with the ability to alter the behavior of the controller output, the aspects dictating this are mainly the problem formulation with the constraints together with the penalization of the stage cost $\ell_k$ and terminal cost $\ell_N$. These are universal and do not depend on the solver used to solve the optimization problem.

An emphasis on the tuning parameters has been made to make them easily accessible and logically formulated such that it is intuitive for the user to change the behavior of the gearbox controller to replicate different drive modes. Among others, currently implemented is the ability to penalize the travel time of a route, hence reflecting if the solver should seek a solution that accentuates velocity and acceleration. In the same way, there is a tunable penalization of the deviation from the most optimal RPM reference, dictating if the solution should emphasize efficiency. Coupled with the drivability aspect is the penalization of the behavior of not changing gears often.

Using the available states in Table 3.2, it is straightforward to expand and tune the stage cost $\ell_k$ or terminal cost $\ell_N$ to get the desired behavior. In terms of ease for the designer to implement different behaviors from the controller, the proposed control structure is therefore deemed versatile.

Evaluating the ability to transfer the control structure to different vehicles involves whether or not the underlying model is able to sufficiently describe the dynamics of the vehicle it should be transferred to. The underlying model, as of now, is describing a single power plant or engine coupled with a gearbox. As the optimization is centered around the engine map, it is possible to replace it with another map to reflect a different engine or power plant, regardless of the propellant.

As for the gearbox, changing the control set $\mathcal{U}_g$ formulated in Equation 3.23, translates to changing the ratios in the gearbox. The number of gear ratios in the gearbox can be set by the designer.

Concluding the ability to transfer the control structure, the structure is able to be transferred to other vehicles if the driveline is of a similar structure. If needed, the underlying model can be expanded or altered to fit vehicles that differ substantially. The control structure is therefore versatile in this regard.

The last evaluation point is the ability to have control over the optimization routine, where the OpEn solver and dynamic programming solver exhibit the starkest differences. Here the OpEn solver can be viewed as a black box which is prompted a problem formulation and outputs a solution. There is no obvious way for the designer to debug why the solver chooses a specific solution. The solution is solely dependent on the problem formulation consisting of the modeling of the vehicle coupled with constraints on states and input.

In contrast, the dynamic programming solver is constructed with the specific gear shifting problem in mind and can be summarized as populating the cost-to-go matrix $\boldsymbol{V}_{c2g}$ using for-loops together with the stage cost $\ell_k$ and terminal cost $\ell_N$. Hence debugging the optimization routine and customizing it for different special events is straightforward compared with the OpEn solver. Here, special events can for example be starting in a steep uphill among others.

Concluding, the ability to have control of the optimization routine, the dynamic programming solver is deemed versatile while the OpEn solver is harder to debug and customize towards a specific problem, hence not being as versatile to the same extent.

# 5

# Conclusion

The objective of this project was to investigate the possibility to adapt a predictive control structure for the specified gear-selecting strategy in a vehicle. The control structure should be versatile and feasible to adapt to embedded hardware. To accomplish the objective, different available solvers were investigated together with the construction of a purpose-built dynamic programming solver.

The developed `SIMULINK` environment produced similar results compared to Volvo's simulation, indicating that the vehicle model used is advanced enough to give meaningful results.

When determining whether to use a space or time domain-based model, the results showed that a model based on time domain is more flexible when driving at different velocities since the horizon length varies in terms of distance.

It could also be concluded that the controller benefits from having available road data, producing more intelligent gear sequences when driving on a hilly road. When comparing the OpEn-based controller to the DP-based controller, the DP-based controller produced solutions that had an overall lower cost given the same stage and terminal cost functions. This was true for all scenarios tested where the DP-based controller changed gears such that the engine was kept in more efficient areas, consuming less fuel or energy. The fact that the OpEn-based controller produces inferior results compared to the DP-based controller could be explained by the OpEn-solver not being optimized for solving MINLP problems, yet it is possible to formulate constraints on the input signal such that it can only take certain values. Using another problem formulation for the MINLP problem might also improve the solution provided by OpEn. However, this is something that needs further research.

When benchmarking the controllers in terms of computational efficiency, the OpEn-based controller was able to find a solution in approximately one order of magnitude less solve time than the equivalent DP-based controller. Given that the DP-based controller was written in `MATLAB` with no particular effort to optimize the code, it would probably benefit in terms of solve-time by being written in a compiled language such as C or C++. However, at the current state, the OpEn-based controller might be a more viable option for embedded solely looking at the solve time.

In terms of versatility, a DP-based controller is more flexible when it comes to implementation. The algorithm is directly available for the developer to change, while OpEn is a more general solver for optimization problems whose inner workings can be seen as a black box while calculating the optimal gear sequence. Also, both

solvers are heavily affected by the formulation of the model, stressing the fact that the model must be able to capture the real-world dynamics of the vehicle and its inner workings to produce a usable prediction from the controller.

As a final conclusion, viewed through the lens of a developer for an optimized gear-selection strategy, the DP-based controller seems to be the most viable option, especially when considering the need to have control over the different stages of the optimization routine. Given the current implementation, the DP-based controller also produced more optimized solutions.

## 5.1 Future Work

Given the scope of the thesis coupled with a focus more oriented towards the optimization routines, there were numerous identified areas that could benefit from further development to achieve a more refined gear selection controller. Some are listed below:

- Jerk state for penalizing sudden acceleration variations to enable a more comfortable driving experience.
- Implement a model handling neutral gear.
- Implement a model including variable gear shifting times.
- Rewrite the DP solver in C/C++.
- Investigate further why the OpEn-based controller does not produce "optimal" gear sequences compared to the DP-based controller.

Implementing a jerk state into the model formulation can improve driving comfort. Vehicle jerks can be penalized within the cost function such that the controller aims to minimize the impact on acceleration while changing gears. To improve energy efficiency, it is important to be able to handle changing gears to neutral, letting the vehicle roll without a gear engaged.

Another essential dynamic of the gearbox is the time it takes for the gearbox to engage the next gear. Some gearshifts are conducted using only the split, some using the forks in the gearbox to couple a new set of cogwheels, some using the range selector, and some a combination of the mentioned. Therefore, to better capture the dynamics of the vehicle, a varying gearshift time, dependent on which gear is to be engaged should be implemented.

As mentioned in the conclusion, the dynamic programming algorithm would probably benefit from being rewritten in a compiled language such as C/C++. It would also be useful to analyze the algorithm in terms of computational complexity to better identify which operations can be optimized.

Lastly, the behavior of the OpEn-based controller differs from the DP-based controller although they are provided with equivalent problem formulations and tuning parameters. It might be possible to reformulate the problem such that OpEn produces results comparable to the DP solution.

# Bibliography


[1] Statistikmyndigheten. *Drivmedelspriserna på rekordnivåer*. accessed 2023-01-25. URL: https://www.scb.se/hitta-statistik/temaomraden/sveriges-ekonomi/fordjupningsartiklar_Sveriges_ekonomi/drivmedelspriserna-pa-rekordnivaer/.

[2] Naturvårdsverket. *Kväveoxider, utsläpp till luft från vägtransporter*. accessed 2023-01-22. URL: https://www.naturvardsverket.se/data-och-statistik/luft/utslapp/utslapp-av-kvaveoxider-till-luft-fran-vagtransporter/.

[3] Energimyndigheten. *Transportsektorns energianvändning*. accessed 2023-01-22. URL: https://pxexternal.energimyndigheten.se/pxweb/sv/Transportsektorns%20energianv%C3%A4ndning/.

[4] D. V. Ngo, Theo Hofman, Maarten Steinbuch, Alex Serrarens, and L. L. F. Merkx. "Improvement of fuel economy in Power-Shift Automated Manual Transmission through shift strategy optimization - an experimental study". In: *2010 IEEE Vehicle Power and Propulsion Conference* (2010), pp. 1–5.

[5] Tamer Basar. "Contributions to the Theory of Optimal Control". In: *Control Theory: Twenty-Five Seminal Papers (1932-1981)*. 2001, pp. 147–166. DOI: 10.1109/9780470544334.ch8.

[6] Torkel Glad and Lennart Ljung. *Control theory : multivariable and nonlinear methods*. London: Taylor & Francis, 2000. ISBN: 0748408770.

[7] J. Richalet, A. Rault, J.L. Testud, and J. Papon. "Model predictive heuristic control: Applications to industrial processes". In: *Automatica* 14.5 (1978), pp. 413–428. ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(78)90001-8.

[8] S.Joe Qin and Thomas A. Badgwell. "A survey of industrial model predictive control technology". In: *Control Engineering Practice* 11.7 (2003), pp. 733–764. ISSN: 0967-0661. DOI: https://doi.org/10.1016/S0967-0661(02)00186-7.

[9] J. B. Rawlings, D. Q. Mayne and M. M. Diehl. *Model Predictive Control: Theory, Computation, and Design, 2nd Edition*. Santa Barbara, CA 93101: Nob Hill Publishing, LLC, 2020.

[10] Frank E. Curtis, Olaf Schenk, and Andreas Wächter. "An Interior-Point Algorithm for Large-Scale Nonlinear Optimization with Inexact Step Computations". In: *SIAM Journal on Scientific Computing* 32.6 (2010), pp. 3447–3475. DOI: 10.1137/090747634.

[11] Pierre Bonami. *Basic Open-source Nonlinear Mixed Integer Programming*. accessed 2023-05-15. URL: https://www.coin-or.org/Bonmin/.

[12] Joris Gillis Joel Andersson and Greg Horn. *CasADi: Build efficient optimal control software, with minimal effort*. accessed 2023-05-25. URL: https://web.casadi.org/.

[13] Mathworks. *fmincon: Find minimum of constrained nonlinear multivariable function*. accessed 2023-05-19. URL: https://se.mathworks.com/help/optim/ug/fmincon.html.

[14] Pantelis Sopasakis and Emil Fresk. *Optimization Engine*. accessed 2023-03-15. URL: https://alphaville.github.io/optimization-engine/docs/open-intro.

[15] L. Stella, A. Themelis, P. Sopasakis, and P. Patrinos. "A simple and efficient algorithm for nonlinear model predictive control". In: *IEEE Conference on Decision and Control (CDC)*. Dec. 2017, pp. 1939–1944.

[16] Tom Goldstein, Christoph Studer, and Richard Baraniuk. "A Field Guide to Forward-Backward Splitting with a FASTA Implementation". In: *ArXiv* abs/1411.3406 (2014).

[17] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization: Springer Series in Operations Research and Financial Engineering*. English. Springer, 2006. ISBN: 9780387303031.

[18] Panagiotis Patrinos and Alberto Bemporad. "Proximal Newton methods for convex composite optimization". In: *52nd IEEE Conference on Decision and Control*. 2013, pp. 2358–2363. DOI: 10.1109/CDC.2013.6760233.

[19] Daniel Liberzon. *Switching in Systems and Control*. Birkhäuser, Boston, 1973. ISBN: 978-1-4612-6574-0. DOI: 10.1007/978-1-4612-0017-8.

[20] N.V Shadinis. "Mixed-integer nonlinear programming 2018. Optim Eng 20". In: (2019), pp. 301–306. DOI: 10.1007/s11081-019-09438-1.

[21] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957. ISBN: 0-691-07951-X.

[22] D. P. Bertsekas. *Dynamic Programming and optimal Control: Volume 1*. Athena Scientific, Belmont, MA, 1995.

[23] J. C. Butcher. "On Runge-Kutta processes of high order". In: *Journal of the Australian Mathematical Society* 4.2 (1964), pp. 179–194. DOI: 10.1017/S1446788700023387.

[24] Volvo Trucks Corporation. *Fact Sheet D16K650*. accessed 2023-04-11. URL: https://stpi.it.volvo.com/STPIFiles/Volvo/FactSheet/D16K650,%20EU6DS_Eng_05_310999633.pdf.

[25] Pierre Bonami et al. "An algorithmic framework for convex mixed integer nonlinear programs". In: *Discrete Optimization* 5.2 (2008). In Memory of George B. Dantzig, pp. 186–204. ISSN: 1572-5286. DOI: https://doi.org/10.1016/j.disopt.2006.10.011.

[26] Marco A. Duran and Ignacio E. Grossmann. "An outer-approximation algorithm for a class of mixed-integer nonlinear programs". In: *Mathematical Programming* 36 (1986), pp. 307–339.

[27] William Bugden and Ayman Alahmar. *Rust: The Programming Language for Safety and Performance*. 2022. arXiv: 2206.05503 [cs.PL].

[28] Jack J. Dongarra, Piotr Luszczek, and Antoine Petitet. "The LINPACK Benchmark: past, present and future". In: *Concurrency and Computation: Practice*

*and Experience* 15.9 (2003), pp. 803–820. DOI: https://doi.org/10.1002/cpe.728.

**DEPARTMENT OF ELECTRICAL ENGINEERING**
**CHALMERS UNIVERSITY OF TECHNOLOGY**
Gothenburg, Sweden
www.chalmers.se


CHALMERS
UNIVERSITY OF TECHNOLOGY