Online Road Vehicle Trajectory Specification in Presence of Traffic Lights with Stochastic Switching Times

Dynamic Systems and Simulation Laboratory

Department of Production Engineering and Management

Technical University of Crete

Diploma Thesis

Supervisor: Prof. Markos Papageorgiou

Author: VasileiosVolakakis

Chania, 2020

To my beloved grandfather, a true believer of education and moral values.

Abstract

The way someone is driving a road vehicle has an important impact on the fuel consumption, thus the term eco-driving was recently introduced to denote a driving style that reduces fuel consumption. This is correlated with many recent advances and developments that are taking place in vehicle communications and automated driving. One application of vehicle connectivity is to receive information about the next signal switching time, when a vehicle approaches a traffic light. Based on this information, appropriately developed systems, known as GLOSA (Green Light Optimal Speed Advisory), compute a fuel-efficient velocity profile for the vehicle to cross the traffic lights, e.g. without stopping, and provide drivers with speed advice.

The main goal of this work is to generate optimal trajectories for vehicles crossing a signalized junction, with traffic signals operating in real-time (adaptive) mode. Specifically, the switching time of the traffic signal is decided, in real time, based on the prevailing traffic conditions and is therefore uncertain in advance. This extended (stochastic) GLOSA problem is addressed by using probabilistic traffic lights information and calculates a velocity profile for the vehicle based on the vehicle's initial state (position and speed) and a fixed final destination state.

The problem is cast in the format of a stochastic optimal control problem, assuming availability of a time-window of possible signal switching times, along with the corresponding probability distribution, and is solved numerically using stochastic dynamic programming(SDP) techniques. As an ingredient of the stochastic solution, an appropriate deterministic optimal control problem is also formulated and solved analytically via Pontryagin’s Minimum Principle for the case of know switching times; the deterministic problem solution is used, as an initial trajectory for some extended SDP techniques that solve the problem in a significantly less amount of time compared to the standard SDP approach.

The extended SDP techniques used in this work are the Discrete Differential Dynamic Programming (DDDP) method and the Differential Dynamic Programming (DDP) method.With these approaches, the workload and computational time are both significantly reduced, making the proposedapproaches applicable in real time.

Acknowledgments

At this point, I would like to thank from the bottom of my heart everyone who has involved andcontributed with their own way and with any way,in the completion of my bachelor's thesis.

First, I would like to refer to my supervisor,in whom I am deeply indebted, Prof. Markos Papageorgiou, who gave me the opportunity, the stimulus and the means to be involved and be a part of something so unconventional and interesting, such Dynamic Programming along with the field of modern and future transportation and provided me with tools and knowledge that will be really useful in the upcoming future.

Mr. Panagiotis Typaldos (Prof. Papageorgiou's PhD student), who spent countless hours with me and tolerated my everyday intrusions into his office (even on Saturdays and Sundays), providing me with the necessary answers, knowledge and help, despite his enormous amount of work, and was always there, anytime and day needed.

My parents, my family, who gave me the opportunity, from every aspect considered, to be standing here today. I wouldn't have had any of those moments if it wasn't for them, for their character, their values, their love and patience and every little or big thing that I got from them.

My friends, my girlfriend, for their continuous support and patience, who were there for me, from the beginning until the end of this journey, playing a major role and having a great impact in me, by giving me things that I am honoured and very lucky to have, during the course of our lives.

Last but of course not least, I would like to thank the Technical University of Crete, everyone who is working here,all the staff and members, and of course all the professors and assistants that I was lucky and privileged to meet, for giving me the opportunity and the means to live this beautiful journey.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1: Prolegomena

As a result of cheap and productive energy resources shortage, lack of big scale energy storage ability and of course the excessive environmental pollution, it isbecoming more and morenecessary for transportation systems to operate with increased fuel efficiency. In the event of road vehicles, fuel efficiency relates to economic aspects, as fuel economy means fewer expenses for the driver, but also to the protection of the environment in an era of climate crisis, which is escalating day by day. Positively, a wide range of technologies have been developed in the past few years, in order to decrease fuel consumption of vehicles, including efficient engines, adjusted vehicle designs and lighter chassis. Additionally, considerable efforts in the field of road vehicle's transportation development and deployment of efficient intelligent transportation systems (including real-time traffic signals) lead to reduced congestion and fuel consumption.

First of all, traffic signals secure the safe crossing of vehicles at urban junctions in cities around the world. Therefore, enforcing safety via traffic lights implies that some vehicles will have to stop in front of a red light, and then accelerate after the traffic light switching to green, something that clearly increases significantly the fuel consumption of road vehicles. In order to reduce the resulting vehicle delays and number of stops, number of algorithms have been created and used over the last decades, pointing at optimizing the traffic signals operation. Fixed-time signal plans are derived off-line for respective times-of-day (e.g., rush hours like early morning congestion or in the afternoons, off-peak etc.) by use of appropriate optimization codes based on historical constant demands and are applied without deviations.Still, something like that implies that switching times of the traffic lights are always known in advance. On the other hand, real-time signal control methods make use of real-time measurements to calculate in real time suitable signal settings. Depending on the selected signal control strategy, the control update period may cover an area from one second to one signal cycle. Plainly, for real-time signals, the next switching time is unknown before the switching decision is actually made. It isalso important to be noted that real-time methods and implementations are deemed to be more advanced and possibly more efficient and productive than a fixed signal application. Fuel consumption is growingly considered as an optimization or evaluation criterion while developing and deploying signal control systems.

Suppose a vehicle is approaching a red traffic light at a given and known speed. If the vehicle continues its course with this speed, it may reach the traffic light after it has switched to green, in which case no fuel-intensive acceleration needs to be applied, although, the constant-speed vehicle might also reach the traffic light before it turns to green, so it will have to stop and accelerate to a higher speed after the light turns from red to green. On the contrary, if the vehicle decelerates smoothly in view of the red light, this may prove beneficial, or not, also depending on the time of the green switch. This quandary of vehicle movement in the direction of a red traffic signal may be addressed by appropriately designed systems. Some of the first attempts were displaying, on road-side dynamic advisory speed signs, the speed that would allow a vehicle to cross the downstream signalized junction at green. But if you take into consideration the emerging advances in vehicle communications, the current state and timing of a traffic signal can be transmitted to equipped vehicles, or applications that the driver possess, in order to enable sensible approaching speed decisions. Dependingon signal information, it is possible to navigate the driver or the vehicle itself (in the case of an automated vehicle) all the way from the current state to the traffic signal by giving speed advice, leading to the fuel consumption and gas emissions minimized, or at least decreased in a significant level. Systems or applications that optimize the vehicle approach to traffic lights are often referred to as Green Light Optimal Speed Advisory systems (GLOSA)[1].

In the presence of fixed signals and with prior knowledge of the next switching time, e.g. via broadcasting of corresponding messages by the signal controller, the problem of how to optimize the approach to traffic signals has been directed in different ways. In various researches, such us [2], speed profiles havebeen compared to their energy demand. Rule-based algorithms have been employed in different works (i.e. in [3]) in order to be speed advisors for vehicles approaching traffic signals, with the goal of reducing fuel consumption and emissions. However, rule-based algorithms are not always capable of finding the optimum, and may deliver sub-optimal results, in particular when the dynamic vehicle kinematics are not accounted for, such us a case with no vehicle acceleration, but the only thing involved is the vehicle’s speed. Optimal control approaches (i.e. [4]), considering only the vehicle kinematics, are, by their nature, more qualified in producing fuel-optimal speed profiles for vehicles approaching fixed traffic signals.

Working under even more complicated circumstances,referring to the existence of real-time signals with very short (e.g. second-by-second) control update periods, so it can be observed that exact prior knowledge of the next switching time is not available even with the signal controller. In this case, the most accurate information about the next signal switching can be presented as an estimate or as a probabilistic distribution. There are systems/applications (i.e. [5]) available that does not need communication to the infrastructure but relies only on an ensemble of the driver’s mobile phone data, in order to detect and foretell the traffic signal timing. A mobile phone is able to detect current traffic signals with its camera, cooperatively communicate and learn traffic signal timing patterns and eventually predict their future timing. An estimate of those future timings can be used as a proxy for any of the GLOSA systems that need a given switching time, along with the proper consideration about the consequences of estimation inaccuracy.

Integrated probabilistic information, unlike a single estimation for the next signal switching time can be produced in form of a probability distribution for the next switching time inside a short-term future time-window, by use of statistics gathered from previous signal operations over an enough rolling horizon. An approach to produce probability distribution for the next switching time, based on past signal switching, can be considered,although the available probability distribution is used heuristically instead of being used optimally, to accordingly time-weigh the objective function, within a deterministic optimal control problem that is solved via a Dynamic Programming algorithm. The problem can be solved via a discrete Stochastic Dynamic Programming (SDP) algorithm, giving us the ability to support the idea that an appropriate utilization of the available probabilistic distribution of the next signal switching time is taking place, within a stochastic optimal control problem. However, the formulated optimal control problem extends only up the point in time where the next switching time becomes known, thus neglecting the cost incurred after this time until the vehicle's final state is reached, something that may drives us away from the optimal, and subsequently the wanted trajectories.

In the current work, the problem of producing fuel-optimal vehicle trajectories for a vehicle approaching a traffic signal (GLOSA) for the scenario of probabilistic, namely stochastic switching times is investigated. Firstly, the problem is formulated with known probability distribution, and the problem is cast in the format of a stochastic optimal control problem, which is solved numerically using SDP algorithms. The difference with previous works is that, the analytic solution of the fixed switching time problem is being exploited by the stochastic approach, which is formulated as an optimal control problem with pertinent final state for the vehicle and is solved analytically via Pontryagin's Minimum Principle, something that provides that the obtained solution is consistent, separately of the actual switching time occurrence. This approach, as stated before, can be readily generalized to allow for switching time decisions to be taken at any time in advance of the actual switching time, not necessarily only between the last time period prior to the actual switching time.

Furthermore, twomodified SDP techniques are going to be used, known as theDiscrete Differential Dynamic Programming (DDDP) methodand the Differential Dynamic Programming (DDP) method. Regarding the DDDP method, each iteration of the corresponding algorithm solves a stochastic problem in a reduced state space, which is formed around the last solution trajectory. The initial trajectory to start the iterations is the analytic solution of the pessimistic case, which assumes that the traffic signal will switch at the latest possible time. With this approach, the work load and computation cost is significantly reduced, making the method applicable and realtime, i.e. capable of processing the given data to obtain the solution of the stochastic problem in few seconds.

The second method, DDP, solves analytically, in every iteration, quadratic approximation of the initial SDP problem, based on the trajectory of the previous iteration. A main advantage ofthe DDP method is that there is no need of use a discrete SDP approach, thanks to the quadratic approximation of the problem that allows for an analytical solution. Consequently, the computational time of the algorithm is reduced, making this approach feasible and accurate for real-time application.

`1.2: Related work

Vehicle trajectory specification systems, in presence of traffic lights, have gained interest from different research domains, such as computer science, civil engineering, transportation research and more. This increasing interest in this particular field led to the creation of a broad range of simulations,along with real-world implementations, but also various terminologies, applications and systems installed in a vehicle. In the following, there are summarized relevant publications in the context of technical and real-time evaluation of vehicle (optimal) trajectory specification systems when a traffic light is encountered (some, not all, with prejudice of course).

Proposing a sustainable predictive control in urban traffic networks based on general smoothening methods, Jamshidnejad et al. [6], along with gradient-based approaches, which can be applied to smooth optimization problems, state that they have proven efficient enough, both computationally and performance-wise, in finding optima of optimization problems. In their paper, an MPC system (model-predictive control) is proposed for an urban traffic network that covers a gradient-based optimization approach to solve the control optimization problem. The controller uses a new smooth integrated flow-emission model, with the purpose of getting a balanced tradeoff amid reduction of the congestion and of the total gas emissions. There are also introduced efficient smoothening methods for non-smooth mathematical models of physical systems.

While traffic signals are necessary to safely control competing flows of traffic, they inevitably enforce a stop-and-go movement pattern that increases fuel consumption, reduces traffic flow and causes traffic jams, as Koukoumidis et al. [5]states. Situations like these can be decreased in a significant portion, by providing drivers and their onboard computational devices, like a mobile phone, with information about the schedule of the traffic signals ahead, and after that the application installed in the device, based on the timing when the signal turn green can advice the driver or the vehicle to adjust his/its speed. Also, the computational device that the driver or the vehicle has can suggest an efficient detour that will eliminate stops and long waits at red lights ahead. This application is called SignalGuru and by using the mobile phone's camera, detects current traffic signals, communicate and learn traffic signal schedule patterns and finally predict a future pattern of theirs. The innovativeness in this work and also the horizons that opens for further implementations and research are really significant, providing the transportation community with a really helpful tool, but most importantly with new paths and knowledge, in terms of control competing flows and fuel consumption.

Lopez et al.[7] states that their preliminary results show that 25% savings are within range in urban circuits, by introducing and evaluating them, comparing with an established model called «Intelligent Driver Model» (IDM), a proper driver model (IDMP), in which a wireless sensor network has been proposed to deliver to the driver or the vehicle the needed information (data). The IDM model is used in order to simulate the longitudinal dynamics of the vehicles used in the various examinations. The difference among the two models is that, the IDMP model takes advantage of an extra knowledge that can be acquired from the upcoming traffic light, by using a wireless sensor network, that collects and distributes future time of red and green lights on the wanted street. In this work it is shown that, taking advantage of all the available knowledge of the system's environment, the chances of developing a more advanced model are highly increasing.

Looking back in the middle '80s, when Leersum[8], obtained and stated that dynamic advisory speed signs could be created and programmed in order to display speeds at which motorists can travel smoothly through downstream signalized intersections, appeared to be beneficial for both the national economy (and later the environmental crisis, that was even then operating, «underground», but still operating), along with the driver. Modifications to the traffic simulation program TRANSYT have enabled some of these benefits to be quantified on a typical urban road network. Taking as a fact that the greater proportion of motorists obey the signs, it is concluded that fuel consumption can be reduced by as much as 13% and the total number of stops by up to 38%.

Borkar et al. [9]discusesa proposed system for predicting the next intersection timing and generating the needed trajectories (speed) at the present intersection, in favor of crossing the next intersection without stopping. The system is speed module for next intersection prediction ingrained in intelligent traffic light control system at intersection. It can also be constructed for a GPS based navigation system. In order to efficiently predicting the needed time and speed for crossing next intersection (or a traffic light in case of a urban road, instead of a highway) without taking into consideration a centralized static approach, the distance between current intersection and next intersection and traffic signal timings of next intersection are considered as an input to the system.

Considering a multi-car system, a proper formulated algorithm by Asadi et al. for the purpose of receiving upcoming traffic light information to minimize idle at the lights and reduce fuel use, and further simulations shows exactly that in [10]. An optimal control algorithm is formulated for each equipped vehicle that usesa prediction ofshort range radar and traffic signal information to create an optimum velocity trajectory for the vehicle. The objectives are timely arrival at green light with the minimum usage of their breaking systems, maintaining safe distance among other vehicles and navigating at or near set speed. The Predictive Cruise Control (PCC) concept proposed in this paper reveals the capability of reducing the fuel consumption and consequently the CO_2 emissions, and also the driving times, by utilizing preview information of traffic signal timing and phase, and all that can be shown from the quite promising results this paper feeds back.

A paper by Hounsel et al.[11], focuses particularly on signalized junctions within computer-controlled urban traffic control (UTC) systems, which, as they state, are increasingly at the heart of traffic control in cities around the world, due toa feature of most urban networks, which is the high density of city streets with numerous road junctions, which require efficient control mechanisms in order to contain possible congestion. They also state that, traffic signals have become the most widely used form of control, with increasing sophistication in detection and real-time optimization providing new levels of efficiency, so it is needed to embrace and use them as a tool in order to achieve their goals (optimization of fuel consumption etc.).

A system studied by Wang et al. [12], refereed as Self-Adaptive Traffic Signal Control System, which is based on future traffic environment. The self-adaptive traffic signal control system actually is an effective measure for relieving urban traffic congestion. The system is able to adjust the signal timing parameters in real time according to the seasonal changes and short-term alternation of traffic request, leading in improvement of the efficiency of traffic operation on urban road networks. According to them, the evolution of information technologies has created a sufficient abundance of acquisition means for traffic data, which include the increase of available amount of holographic data, available data types, and accuracy. Also, the development of commonly used self-adaptive signal control systems in the world is explored, along with their technical features, the current research status of self-adaptive control methods, and the signal control methods for diversified traffic flow composed of connected vehicles and autonomous vehicles.

Typlados et al. state in [4] that themainpurposeoftheir workistogenerateoptimaltrajectoriesforvehiclescrossingasignalizedjunction,withtraffic signalsoperatingineitherfixed-timeorreal-time(adaptive)mode.Inthelattercase,thenext switchingtimeisdecidedinrealtimebasedontheprevailingtrafficconditionsandistherefore uncertaininadvance.TheGLOSAproblemisaddressed as an optimal control problem,byusingtrafficlightsinformationand calculatingatrajectoryandvelocityprofileforthevehiclebasedonthevehicle'sinitialstate (positionandspeed)andafixedfinaldestinationstate. Forthecaseofreal-timesignals,availabilityofatime-window of possible signal switching times, along with the corresponding probabilitydistribution, isassumed,andtheproblemiscastintheformatofastochasticoptimalcontrolproblemandis solved numerically using Stochastic Dynamic Programmingtechniques, with anappropriateoptimalcontrol problembeingformulatedandsolvedanalytically at first,viaPontryagin'sMinimumPrincipleforthecase ofknownswitchingtimes.

1.3: What is a GLOSA system

The highest fuel consumption on urban arterials is associated with driving in congested traffic, characterized by higher speed fluctuations and frequent stops at intersections. One way to reduce excessive stop-and-go driving on urban streets is to optimize signal timings, as stated before. New methods in traffic signal optimization have incorporated changes in driver's behaviour to achieve optimum performance at signalized intersections. Connected vehicles technology provides a two-way wireless communication environment enabling vehicle-to-vehicle and vehicle-to-infrastructure communications, which can be used for a variety of mobility and safety applications. All the above can be summarized, as a GLOSA system [13].

The goal of Green Light Optimal Speed Advisory (GLOSA) systems is to lower CO_2 emissions and to avoid unnecessary stopping in intersection approach scenarios by giving speed advices to drivers based on current and future traffic light signal phase timings. So, basically it is a system that provides the driver (or the vehicle in presence of automated systems and vehicles) with speed advises any time he/it needs them, in order to achieve all the above.

After extended researches, tests and evaluation of those systems, it can be stated that a potentially reduce of CO_2 emissions and fuel consumption can be done, in a percentage of 13% and potentially upper than that, and also up to 89% in average stop time[14],[15].The initial trajectories that are needed for a GLOSA system to calculate the optimal speed and feed it back to the driver/vehicle, are the distance to the next traffic light, along with the time to the next signal change.

GLOSA strategies can be categorized into two different sectors, as singe segment or as multi segment. In the single segment approach speed advisory is calculated for the segment preceding the nearest traffic signals, as soon as a vehicle enters the segment. It is presumed that traffic signals are pre-timed and traffic conditions allow vehicles to adapt their speed. In addition, vehicles have access to traffic signal schedules. In the case of the multi-segment GLOSA, a vehicle calculates a set of optimal speeds (one speed advice per each segment) before entering the first segment. The advisory speed for each segment is established as the average speed that a vehicle should travel on the segment[16].

Similar systems are going to play a major role in the field of future international transportations and should be available for every different type of adaptive traffic lights. Adaptive traffic lights range from semi-adaptive controllers that don't change the order of signals but only alter their length to fully adaptive controllers with the ability to change every aspect of its program. The most important inputs for these traffic lights (and what separate them) are detectors, who are capable of counting vehicles, detecting waiting pedestrians, or identifying approaching buses or emergency vehicles. Every one of them stimulates the controller and can alter its behavior. As you can see, it is critical for a prognosis algorithm to consider these detectors as they have a significant impact on the signal transitions of an adaptive traffic light.

Predominantly, GLOSA functionality is based on two message types: SPAT and MAP [4]. In favor of getting information about current and upcoming traffic light phases from the traffic light, a connection to the traffic light controller is established. A Signal Phase and Timing Message (SPAT) informs about the vehicle's current state, current phase and next phase for each lane of an intersection. Continuing Map Data Messages (MAP) provides information about the topology of an intersection such as number of lanes and turning constrains. In order to give the wanted speed advices to the driver, a vehicle must receive at least one message of every type and link them using the intersection's singular identification included in the messages that have been sent. When a message is received, the GLOSA application generates geometry from the MAP message to match the vehicle's position and determines the corresponding lane number. By the time the lane that the vehicle is on is finally known, signal phases and timing data related to this lane number can be matched. The way that SPAT and MAP messages are transmitted is by single-hop broadcast.

The application collects information about all vehicles that can currently communicate with the infrastructure. For every vehicle that is approaching a traffic signal, the system determines a range of feasible speeds, which in case of implementation, are able to assist the vehicle to pass through an intersection, or in our case from a traffic light without stopping, which means pass when the light is green. In case of having a speed price that belongs already within this range, the application examines the next incoming vehicle [17]. In that way, it is guaranteed that the GLOSA system is going to send a speed advice to a driver or to a vehicle only if it is necessary, and the word necessary aims to the situation when this action would prove beneficial, so pointless and no needed actions are avoided. So, the algorithm affects only those drivers/vehicles that if keep on travelling with the same speed, would arrive at a traffic light in its red phase.

1.4: Goals of this work

The main target of this work is to generate optimal trajectories for vehicles crossing a signalized junction, with traffic signals operated in real-time. The next switching time is decided in real time based on the prevailing traffic conditions and is therefore uncertain in advance. The GLOSA problem is addressed by using traffic lights information and calculating a trajectory and velocity profile for the vehicle based on the vehicle's initial state (position and speed) and a fixed final destination state.

In this particular work, focus is on real-time traffic signals, in order to reach and approachthat could be used and transformed in the futureto a viable application, that would be a very useful tool for every driver, and of course for the environment as well (directly or indirectly), by giving the appropriate speed advices to the driver or the vehicle, making driving an environmentally orientated process.

The first algorithm that was developed, by Typaldos et al. [4], had the same principals with the algorithm introduced in this thesis, which is no other than the provision of the needed optimal trajectories (position and speed), in order to assist the driver or the vehicle to cross the traffic light optimally, meaning, with the least possible cost (which is fuel, CO_2 emissions etc.). The main difference amidst the previous algorithm implemented by Typaldos et al. in [4] and the algorithm introduced in this work, is that the workload – computational time of the previous algorithm was way too big considering with the method used in the current algorithm, that is presented analytically in the following chapters, and so the application was not able to be developed as a real-time one. So, the new algorithmdo not provide better results (respectively), but they can provide the same results in a substantial less amount of time, due to the decrease of the workload the application has to deal with.

Chapter 2: Optimal Control Problem

Optimal Control Problems (OCPs) are obtained in the time state and their solution calls for initiating an operation index for the system. Due to the dynamic genre of the decision variables confront, optimal control problems are much more challenging to solve, resembled to normal optimization where the decision variables are scalars, in addition to the fact that OCPs become more demanding when uncertainty in any parameter or variable is involved[18].

2.1: Continuous Time Dynamic Systems formulation

The Optimal Control Problem refers to either continuous time dynamic systems, or to discrete time dynamic systems. In case of continuous time, the following problem is formulated:

Given the initial state $x(0) = \boldsymbol{x}_0, 0 \leq t \leq {t_e}^*$, the time functions of the control variables $\boldsymbol{u}^*(t), 0 \leq t \leq {t_e}^*$ are requested, along with the time functions of the state variables $\boldsymbol{x}^*(t), 0 \leq t \leq {t_e}^*$, and also the final time ${t_e}^*$, which minimize the cost criterion (also, with no violation of generality of the problem, the initial time t is being set as $t_0 = 0$)

$$J = \partial[\boldsymbol{x}(t_e), t_e] + \int_0^{t_e} \varphi[\boldsymbol{x}(t), \boldsymbol{u}(t), t]dt \tag{2.1}$$

and taking into consideration the following constrains $\forall t \in [0. t_e]$

$$\dot{\boldsymbol{x}} = \boldsymbol{f}[\boldsymbol{x}(t), \boldsymbol{u}(t), t] \tag{2.2}$$

$$\boldsymbol{h}[\boldsymbol{x}(t), \boldsymbol{u}(t), t] \leq 0 \tag{2.3}$$

here$\partial\boldsymbol{h}/\partial\boldsymbol{u}$ the (fully graded) Jacobi table, and the final target

$$\boldsymbol{g}[\boldsymbol{x}(t_e), t_e] = 0 \tag{2.4}$$

Following, a graphical illustration of the OCP (with no violation of generality):

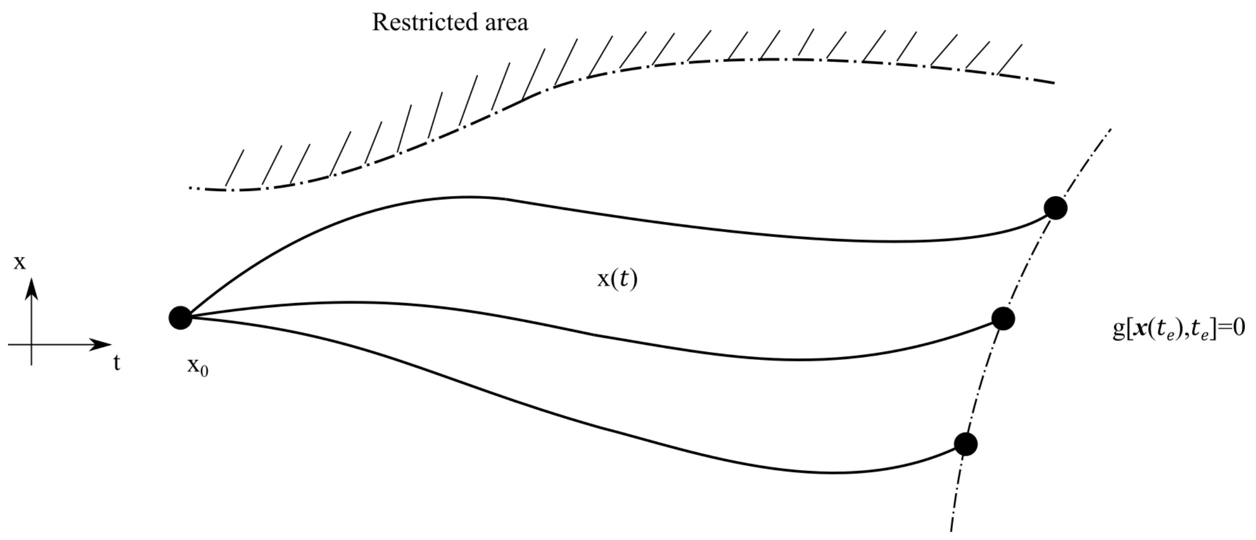


Figure 1: OCP example. Adapted with permission from [20].

Figure 1 explains graphically the OCP for a supposed, one dimensioned system. The initial state $\boldsymbol{x}_0$ of the system (ii) needs to be transferred in the final trajectory (iv), avoiding the restricted areas, which are being specified by the set of inequalities (iii). There are usually quite enough permissible control time functions $\boldsymbol{u}(t), 0 \leq t \leq t_e$, that render feasible this particular transfer. Among them, the solution of the OCP is achieved by the time functions $\boldsymbol{u}^*(t), \boldsymbol{x}^*(t), 0 \leq t \leq {t_e}^*$, which minimize the cost criterion ***Eq. (2.1)***.

The Optimal Control Problem has been formulated above in its general form. Depending on the data of every different application, there a series of specific instances:

a) Known final time t_e:
   - a1) Known final state $\boldsymbol{x}(t_e) = \boldsymbol{x}_e$
   - a2) Free final state
   - a3) Final trajectory $\boldsymbol{g}[\boldsymbol{x}(t_e)] = 0$.

b) Free final time t_e:
   - b1) Known final state $\boldsymbol{x}(t_e) = \boldsymbol{x}_e$
   - b2) Free final state
   - b3) Final trajectory.

Also, in the case of a continuous-time OCP, the following function, known as the Hamiltonian function is defined [19]:

$$H(x,u,t) = L(x,u,t) + \lambda^T f(x,u,t) \tag{2.5}$$

where the Lagrange multipliers are defined as $\lambda \in R^n$.

According to the above, the Pontryagin's Principle of Optimality deliver the below essential constraints (see [19]):

$$\dot{x} = \frac{\partial H(x,u,t)}{\partial \lambda} \quad (2.6)$$

$$\dot{\lambda} = \frac{\partial H(x,u,t)}{\partial x} \quad (2.7)$$

$$u = arg \min_{u} H(x,u,t) \quad (2.8)$$

2.1.1: Free final time

In the presence of free final time (t_e), the optimal control vector $u(t)$ for $t_0 \leq t \leq t_e$ needs to be calculated, along with the final time t_e, so the minimization of the cost criterion ***Eq. (2.1)*** could be achieved.

The following equations determines the value of the final time t_e:

$$H[x(t_e), u(t_e), \lambda(t_e), t_e] + \theta_{t_e} = 0 \quad (2.9)$$

Eq. (2.9) along with the initial and final conditions provide the following boundary constrains, which can be used in order to solve the two point value problem, and also in order to compute the value of t_e.

$$x(t_0) = x_0 \quad (2.10)$$

$$g[x(t_e), t_e] = 0 \quad (2.11)$$

$$\lambda(t_e) = \theta_{x(t_e)} + g_{x(t_e)}^{T} v \quad (2.12)$$

$$H[x(t), u(t), (t), t_e] + \theta_{t_e} = 0 \quad (2.13)$$

2.2: Discrete Time Dynamic Systems formulation

The Optimal Control Problem in the case of discrete time is being formulated in a similar way as the previous one considering continuous-time dynamic systems, as follows:

Given the initial state $x(0) = \boldsymbol{x}_0$ and the disorders $z(k)$, $0 \leq k \leq K^* - 1$, the time functions of the control $u^*(k)$, $0 \leq k \leq K^* - 1$ and state variables $x^*(k)$, $0 \leq k \leq K^* - 1$, are being wanted, as well as the final time K^*, which minimize the cost function:

$$J = \partial[x(K), K] + \sum_{k=0}^{K-1} \varphi[\boldsymbol{x}(k), \boldsymbol{u}(k), k] \quad (2.14)$$

by taking into consideration the following constraints $\forall k \in [0, K-1]$:

$$\boldsymbol{x}(k+1) = \boldsymbol{f}[\boldsymbol{x}(k), \boldsymbol{u}(k), k] \tag{2.15}$$

$$\boldsymbol{h}[\boldsymbol{x}(t), \boldsymbol{u}(t), t] \leq 0 \tag{2.16}$$

as well as the final target

$$\boldsymbol{g}[\boldsymbol{x}(K), K] = 0. \tag{2.17}$$

All the explanations, specifications and special cases that were given and presented in above, in section 2.1, considering the OCP for a continuous-time dynamic system, are valid in the case of a discrete-time dynamic system.

2.2.1: Free final time

When the final time is free in this occasion of the Optimal Control Problem (on a discrete time dynamic system), the final time t_e is considered as a variable for optimized, along with the control variable. The control vector $u(k)$, $k = 0, \ldots, K-1$and the final free time t_e needed for the optimization of the criterion:

$$J[x(k), u(k), k] = \partial[x(K)] + \sum_{k=0}^{K-1} \varphi[\boldsymbol{x}(k), \boldsymbol{u}(k), k] \tag{2.18}$$

where the number of time-steps K is finite, and also

$$t_e = KT \tag{2.19}$$

under the constraints

$$g[x(K), t_e] = 0 \tag{2.20}$$

$$x(k+1) = f[x(k), u(k), k, \Delta_t] \tag{2.21}$$

$$x(x_0) = x_0 \tag{2.22}$$

where T is considered as the time-step.

Chapter 3: Dynamic Programming

Dynamic Programming is a method developed during the 1950 decade by the American mathematician R. E. Bellman. Dynamic Programming is based on the Principal of Optimality, which is nothing more than a very simple and understandable capacity of the optimal control problems solution. The properties of this particular principal are quite significant, leading to the creation of a vast number of algorithms in order to cope with problems in the field of dynamics, and also with combinatorial optimization problems, deterministic and stochastic. The applications of Dynamic Programming (DP) are many and also expand in various areas, such as Operational Research, Economics etc, along with different types of problems (organization, automatic control, design and many more) [20].

3.1: Introduction in Discrete Time Optimal Control

Considering the minimization of the discrete time cost criterion

$$J = \partial[x(K)] + \sum_{k=0}^{K-1} \varphi[\boldsymbol{x}(k), \boldsymbol{u}(k), k] \tag{3.1}$$

with defined time-horizon K, taking into consideration the statutory constrains

$$\boldsymbol{x}(k+1) = \boldsymbol{f}[\boldsymbol{x}(k), \boldsymbol{u}(k), k] \tag{3.2}$$

and initial state $\boldsymbol{x}(0) = \boldsymbol{x}_0$ and final target which is define by

$$\boldsymbol{g}[\boldsymbol{x}(K)] = 0 \tag{3.3}$$

or with every other possible way of defining a set of points $\boldsymbol{x}(K)$, and only in those are permitted to end up the state **x** of the problem. It is also important to mention that, the results of this chapter can be applied in the particular case of the absence of a final target[20]. In this section, for convenience purposes, only the given final time K instance is being taken into consideration. Generalization of the results in case of free final time will be examined in subsection 3.4.

The allowed control area is defined as

$$\boldsymbol{u}(k) \in \boldsymbol{\mathcal{U}}[\boldsymbol{x}(k), k] = \{\boldsymbol{u}(k) | \boldsymbol{h}[\boldsymbol{x}(k), \boldsymbol{u}(k), k] \leq 0\} \tag{3.4}$$

and it is supposed that the Jacobi table $\partial \boldsymbol{h}/\partial \boldsymbol{u}$ is a full degree table. This admission is not fulfilled if there are state inequality constrains. If that happens, the following inequality constrains are assumed

$$\boldsymbol{x}(k) \in \mathfrak{X}(k) = \{\boldsymbol{x}(k) | \boldsymbol{h}^X[\boldsymbol{x}(k), k] \leq 0\} \tag{3.5}$$

with$\mathfrak{X}(k)$ being the allowed state area. Moreover, when it is considered necessary, the following equality constrains can be assumed

$$\boldsymbol{G}[\boldsymbol{x}(t), \boldsymbol{u}(t), t] = 0 \tag{3.6}$$

along with discrete control or state areas

$$u_i(t) \in \boldsymbol{\mathcal{U}}_i = \{u_{i,1}, u_{i,2}, \dots\}. \tag{3.7}$$

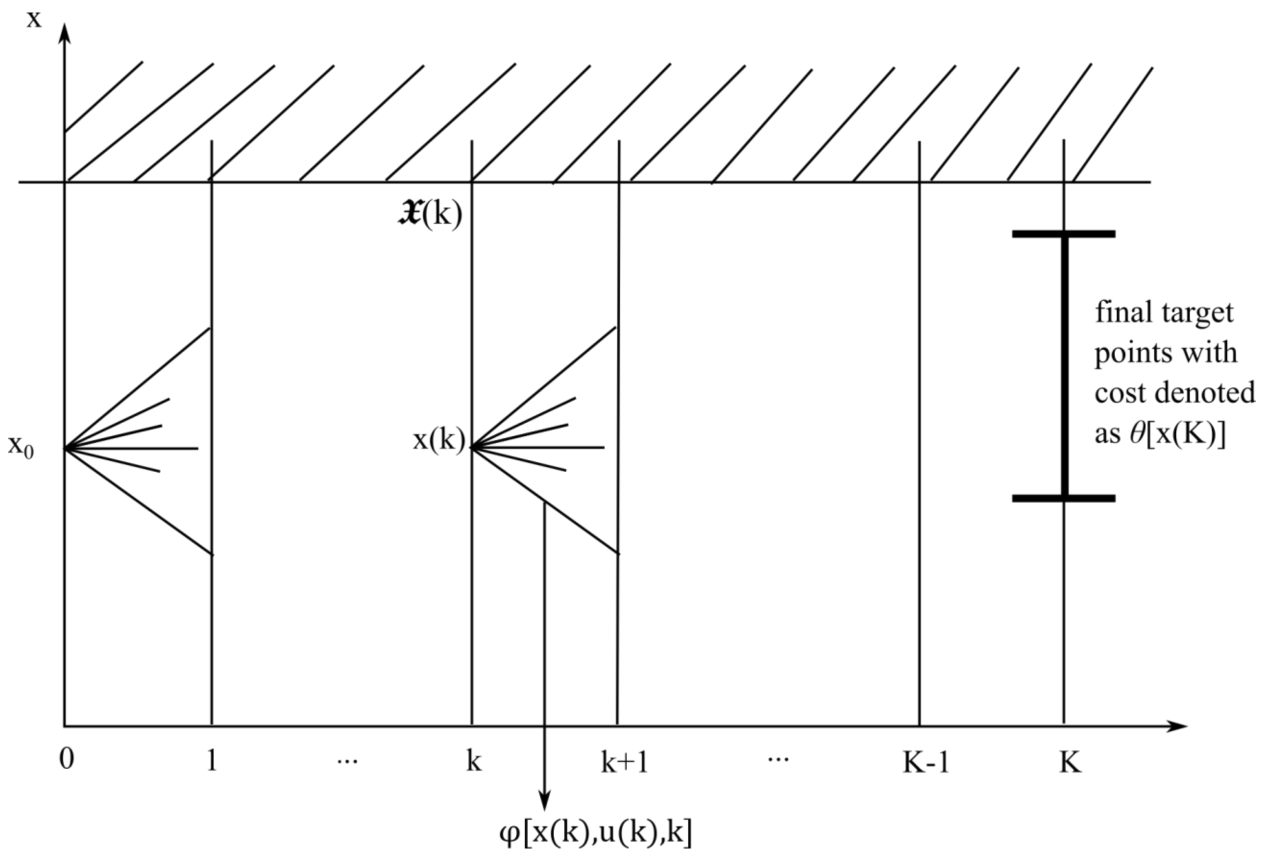


Figure 2: Graphical explanation of the discrete time OCP. Adapted with permission from [20].

In the event of discrete time $k = 0,1,2, \dots, K$ this particular problem can be considered as a multiple decision - multiple step problem, so an approach can be considered, based on the Bellman's Principal of Optimality[21], according to which: Every residual $\boldsymbol{u}^*(t), t \in [t_1, t_e], 0 \leq t_1 \leq t_e$, of the optimal control trajectory $\boldsymbol{u}^*(t), t \in [0, t_e]$, is an optimal trajectory for thetransfer of the corresponding intermediate state $\boldsymbol{x}^*(t_1)$ to the final trajectory $\boldsymbol{g}[\boldsymbol{x}(t_e), t_e] = 0$. For this purpose, an absolute minimum of the problem is presupposed.

3.2: Bellman's Recursive Equation

For the application of the Principal of Optimality in an Optimal Control Problem with discrete times, the residual cost, or cost-to-go J_k is defined, for the transportation of a state $x(k)$ to the final target ***Eq. (3.3)*** as following

$$J_k = \theta[\boldsymbol{x}(K)] + \sum_{\kappa=k}^{K-1} \varphi[\boldsymbol{x}(\kappa), \boldsymbol{u}(\kappa), \kappa] \tag{3.8}$$

Fora specific problem, the minimum cost-to-go $J_k^* = minJ_k$ (satisfying all the necessary constrains), depends exclusively on the transporting state $x(k)$ and the time k. This minimum cost is named $V[\boldsymbol{x}(k), k]$ with

$$V[\boldsymbol{x}(k), k] = minJ_k = min\{\varphi[\boldsymbol{x}(k), \boldsymbol{u}(k), k] + J_{k+1}\} \tag{3.9}$$

where the minimum is considered among all trajectories $\boldsymbol{u}(\kappa), \kappa = k, \dots, K-1$, that satisfies ***Eq. (3.2) - (3.5)***. By applying the Principal of Optimality on ***Eq. (3.9)***

$$V[\boldsymbol{x}(k), k] = min\{\varphi[\boldsymbol{x}(k), \boldsymbol{u}(k), k] + V[\boldsymbol{x}(k+1), k+1]\}. \tag{3.10}$$

Replacing $x(k+1)$from ***Eq. (3.2)*** to ***Eq. (3.10)*** the outcome is

$$V[\boldsymbol{x}(k), k] = min\{\varphi[\boldsymbol{x}(k), \boldsymbol{u}(k), k] + V[\boldsymbol{f}(\boldsymbol{x}(k), \boldsymbol{u}(k), k), k+1]\} \tag{3.11}$$

The right member of ***Eq. (3.11)***, bears the name Bellman's Recursive Equation [22], and it depends on $\boldsymbol{u}(k)$, not from the subsequent $\boldsymbol{u}(\kappa), \kappa = k+1, \dots, K-1$. Therefore, minimization on Bellman's Recursive Equation is understood only for the control variables of time k, meaning for $\boldsymbol{u}(k)$, of course always with respect to the constrains ***(3.4)*** and ***(3.5)***. This one-step minimization can be performed independently for every step, starting from the final time, for $k = K-1, K-2, \dots, 0$, so the initial multiple-step decision problem is distributed in Kone-step decision problems. Following, there is a detailed presentation of this step-process [20] (which is also known as Dynamic Programming):

Step K-1: For every $x(K-1) \in \mathfrak{X}(K-1)$ the corresponding $\boldsymbol{u}(K-1)$ is determined, which minimises

$$J_{K-1} = \theta[\boldsymbol{x}(K)] + \varphi[\boldsymbol{x}(K-1), \boldsymbol{u}(K-1), K-1]$$

under
$$\boldsymbol{x}(K) = \boldsymbol{f}[\boldsymbol{x}(K-1), \boldsymbol{u}(K-1), K-1]$$

$$\boldsymbol{g}[\boldsymbol{x}(K)] = 0$$

$$\boldsymbol{u}(K-1) \in \boldsymbol{\mathcal{U}}[\boldsymbol{x}(K-1), K-1]$$

$$\boldsymbol{x}(K) \in \boldsymbol{\mathfrak{X}}(K).$$

The outcome of this one-step minimization for every $\boldsymbol{x}(K-1) \in \boldsymbol{\mathfrak{X}}(K-1)$ is expressed via $\boldsymbol{u}(K-1) = \boldsymbol{R}[\boldsymbol{x}(K-1), K-1]$. The corresponding minimum values of J_{K-1} are being illustrated with the function $V[\boldsymbol{x}(K-1), K-1]$.

Step K-2: For every $x(K-2) \in \boldsymbol{\mathfrak{X}}(K-2)$the corresponding $\boldsymbol{u}(K-2)$ is determined, which minimises

$$\tilde{J}_{K-2} = V[\boldsymbol{x}(K-1), K-1] + \varphi[\boldsymbol{x}(K-2), \boldsymbol{u}(K-2), K-2]$$

under $$\boldsymbol{x}(K-1) = \boldsymbol{f}[\boldsymbol{x}(K-2), \boldsymbol{u}(K-2), K-2]$$

$$\boldsymbol{u}(K-2) \in \boldsymbol{\mathcal{U}}[\boldsymbol{x}(K-2), K-2]$$

$$\boldsymbol{x}(K-1) \in \boldsymbol{\mathfrak{X}}(K-1).$$

The final target, ***Eq. (2.7)*** doesn't need to be taken into consideration. The results of this one-step optimization are being illustrated with the functions $V[\boldsymbol{x}(K-2), K-2]$and $R[\boldsymbol{x}(K-2), K-2]$.

Step K-3

●
●
●

(every iteration follows the same path as before, with the appropriate changes of course each time)

Step 0: For $\boldsymbol{x}(0) = \boldsymbol{x}_0$ the corresponding $\boldsymbol{u}(0)$ is defined, which minimizes

$$\tilde{J}_0 = V[\boldsymbol{x}(1), 1] + \varphi[\boldsymbol{x}(0), \boldsymbol{u}(0), 0]$$

under $$\boldsymbol{x}(1) = \boldsymbol{f}[\boldsymbol{x}(0), \boldsymbol{u}(0), 0]$$

$$\boldsymbol{u}(0) \in \boldsymbol{\mathcal{U}}[\boldsymbol{x}(0), 0]$$

$$\boldsymbol{x}(1) \in \mathfrak{X}(1).$$

The outcome of this one-step minimization is expressed via $\boldsymbol{u}(0) = \boldsymbol{R}[\boldsymbol{x}(0), 0]$and $V[\boldsymbol{x}(0), 0]$.

In some steps there might be states $\boldsymbol{x}(k) \in \mathfrak{X}(k)$ in which the corresponding one-step minimization problem does not have a solution, because the feasible area that occurs from the constrains is empty. Such points $\boldsymbol{x}(k)$ cannot be transferred to the final target, satisfying all the constrains.

In the end of the presented K-step Dynamic Programming procedure, there have been calculated not just the optimal trajectories for the transfer of the initial state $\boldsymbol{x}(0) = \boldsymbol{x}_0$ to the final target***Eq.(3.3)***, but instead, an optimal closed loop rule, which is expressed from the results of the one-step minimization problems

$$\boldsymbol{u}(0) = \boldsymbol{R}[\boldsymbol{x}(k), k], k = 0,1, \dots, K-1. \tag{3.12}$$

In order to determine the field of applications of the optimal control rule, the aforementioned problem for the 0 step can be solved, for every $\boldsymbol{x}(0) \in \mathfrak{X}(0)$.***Eq. (3.12)*** includes sufficient elements for the optimal transfer not only regarding $\boldsymbol{x}_0$, but for any $\boldsymbol{x}(k) \in \mathfrak{X}(k), k = 0,1, \dots, K-1$for the final target (3.3), under the condition that the transfer is feasible.

The one-step minimization in every step can be attempted either analytically or numerically (with the help of a personal computer). Analytical solutions are generally applied for slightly simple problems. It is also worth mentioning that, Dynamic Programming leads to an absolute minimum of the discrete time OCP, if only the calculated one-step minimums are also absolute ones.

3.3: Discretization

The general numeric solution of a discrete time OCP is possible, if a discrete grid of points is entered in the feasible areas $\mathfrak{X}(k)$ and $\boldsymbol{\mathcal{U}}[\boldsymbol{x}(k), k]$ (see **Figure 3**below). The discrete intervals $\Delta\boldsymbol{x}(k)$ and $\Delta\boldsymbol{u}(k)$ can be chosen depending on the specific problem and the desirable solution's precision. In case of an unlimited feasible state or control area, it is necessary to attach appropriate bounds, in order to have a finite number of discrete points.

If someone applies in a discrete state $\boldsymbol{x}^i(k)$ all the discrete controls $\boldsymbol{u}^j(k)$, there are a finite number of transitions to the next step $k+1$, with the corresponding costs

$\varphi\left[\boldsymbol{x}^{i}(k), \boldsymbol{u}^{j}(k), k\right]$. Applying this procedure in all discrete state points of all the discrete times, the result is a discrete decision multiple-step system.

The application of a discrete control $\boldsymbol{u}^{j}(k)$ in a discrete state $\boldsymbol{x}^{i}(k)$ leads to the state

$$\boldsymbol{x}(k+1) = \boldsymbol{f}\left[\boldsymbol{x}^{i}(k), \boldsymbol{u}^{j}(k), k\right] \tag{3.13}$$

of the step $k+1$. There are two particular cases that need to be taken into consideration:

- State $\boldsymbol{x}(k+1)$ is out of the feasible area $\boldsymbol{\mathfrak{X}}(k+1)$. In this case, this particular transition does not be taken into consideration.
- State $\boldsymbol{x}(k+1)$ does not coincides with a discrete point of step $k+1$. In this case, it can approximately be considered that the state $\boldsymbol{x}(k+1)$ concurs with the closest discrete point. However, if a more precise solution is wanted, the answer is a linear interpolation approach. Also need to be stated that, if the problem is statutory, ***Eq. (3.2)*** is invertible as to $\boldsymbol{u}(k)$ (something that presuppose that $\dim(x) = \dim(u)$), meaning that if

$$\boldsymbol{u}(k) = \boldsymbol{F}[\boldsymbol{x}(k), \boldsymbol{x}(k+1), k] \tag{3.14}$$

can be educed analytically from ***Eq. (3.2)***, then there is no need of discretization of the feasible control area in advance. Instead of that, for every transition from a discrete point $\boldsymbol{x}^{i}(k)$ to a discrete point $\boldsymbol{x}^{j}(k+1)$ of the next step, the necessary control $\boldsymbol{u}^{ij}(k) = \boldsymbol{F}\left[\boldsymbol{x}^{i}(k), \boldsymbol{x}^{i}(k+1), k\right]$ can be computed through ***Eq. (3.14)***, and if $\boldsymbol{u}^{ij}(k) \in \boldsymbol{\mathcal{U}}[\boldsymbol{x}^{i}(k), k]$, then the transition is feasible, otherwise, the transition violates the control constrains and consequently, it does not taken into consideration.

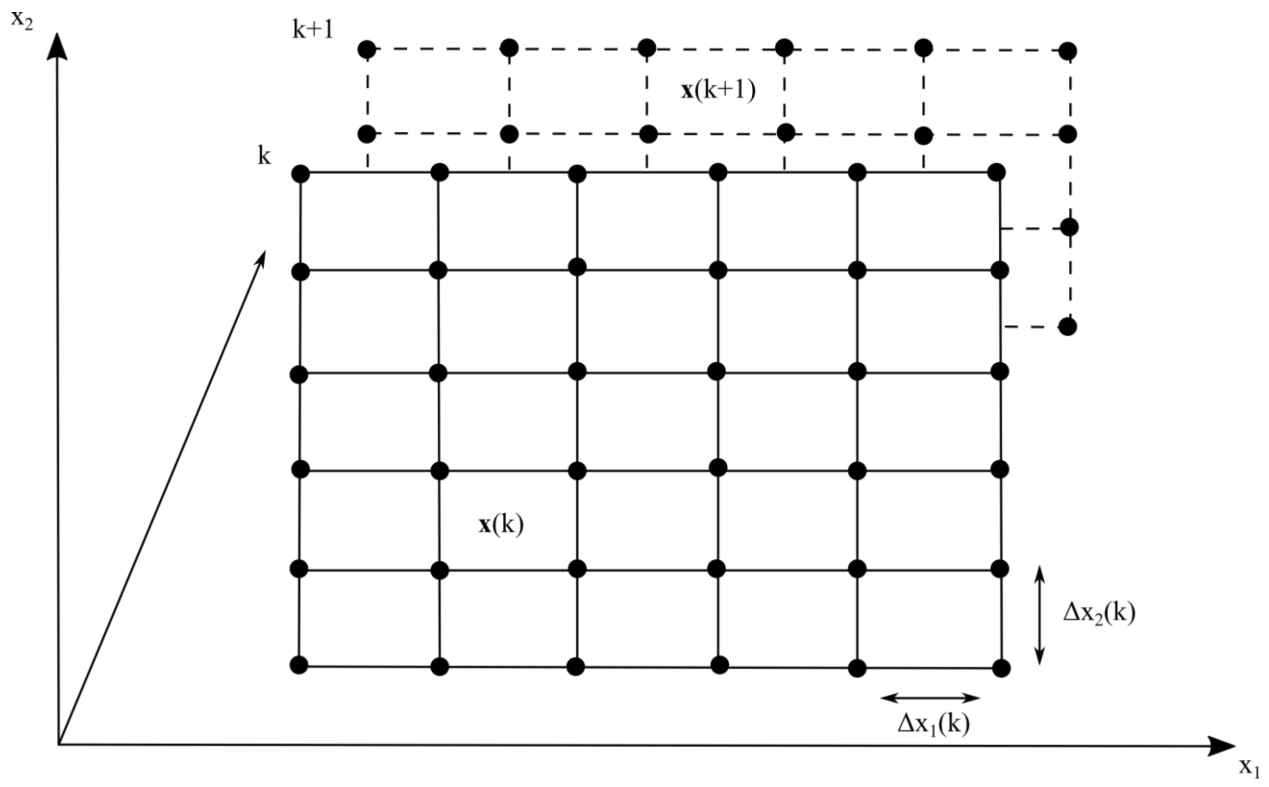


Figure 3: Discretization of the feasible state area for a two-dimension system. Adapted with permission from [20].

Final target $\boldsymbol{g}[\boldsymbol{x}(K) = 0]$ needs also to be adjusted into the problem's discrete environment. By entering a tolerance zone $\pm\delta$around the final target, the following discrete final target is formulated

$$Q = \{\boldsymbol{x}(K) | \exists \boldsymbol{\beta}: |\boldsymbol{x}(K) - \boldsymbol{\beta}| \leq \delta, \boldsymbol{g}(\boldsymbol{\beta}) = 0\} \tag{3.15}$$

It can now be assumed that the final target has been fulfilled (approximately), provided that $\boldsymbol{x}(K) \in Q$.

3.4: Computational time

Suppose $a_i(k), i = 1, \ldots, n$, the number of discrete points of the component i of the vector $\boldsymbol{x}(k)$ and $\beta_j(k), j = 1, \ldots, m$, the number of discrete points of the component j of the vector $\boldsymbol{u}(k)$. The total state grid includes

$$\sum_{k=0}^{K} \prod_{i=1}^{n} a_i(k) \tag{3.16}$$

discrete points and the number of transitions for each point is

$$\prod_{j=1}^{m} \beta_j(k). \tag{3.17}$$

If, for convenience purposes, it is assumed that $a_i(k) = a$ and $\beta_j(k) = \beta$ for each i, j, k, then the needed computational time τ for processing the multi-step procedure of a Discrete Dynamic Programming problem is equal to the total number of nods,

$$\tau \sim K \cdot a^n \cdot \beta^n. \tag{3.18}$$

The necessary storage space of the table that stores the control rule is m values for each discrete point $\boldsymbol{x}^i(k)$; for the m components of the control $\boldsymbol{u}^{l(i)}(k)$ vector, and in total

$$m \sum_{k=0}^{K} \prod_{i=1}^{n} a_i(k) = m \cdot K \cdot a^n \tag{3.19}$$

values to store. ***Eq. (3.18), (3.19)***point out that computational time during a Discrete Dynamic Programming application increases exponentially with the n, m dimensions of the problem, something that constitutes a major disadvantage of this, and similar to this, method. Following, an example is presented in order to understand better the term exponential increase [20].

Example 2

Assume $K = 10$ steps and one dimension control ($m = 1$). Also assume $\alpha = \beta = 100$ discrete points and $\tau_s = 100\ \mu s$the required computational time for the computation of the cost for one transition. Using ***Eq. (3.18)***, the necessary computational time of the total solution through a Discrete Dynamic Programming approach is estimated, as stated next:

for $n = 1\ \tau = 10\ s$

for $n = 2\ \tau = 17\ min$

for $n = 3\ \tau = 28\ h$

for $n = 4\ \tau = 3{,}9\ months$

for $n = 5\ \tau = 32\ years$

Due to the exponential increase of the computational time, narrow limits are being instated on a problem's dimensions, which can be solved with a Discrete Dynamic Programming method. An inquiry regarding a problem's dimensions limits can be performed as follows:

- For each particular problem, it is necessary to check if the feasible state and control area can be further confined, aiming to the reduction of the corresponding number of points of the grid, and also the number of transitions. Nevertheless, that extra constrain needs to be performing in a way that it does not excluding optimal solutions, something that it is frequently difficult to be assessed before the problem's solution.
- The length of the discrete intervals $\Delta\boldsymbol{x}(k)$ and $\Delta\boldsymbol{u}(k)$ needs to be chosen as big as possible, meaning, not smaller than necessary. Nevertheless, bigger intervals usually lead to less precise solutions. A reliable (in advance) assessment of the solution's precision is demanding, if not difficult in many applications.
- There have been some suggestions, regarding variations and simplifications of the Discrete Dynamic Programming method, aiming to the reduction of the workload and eventually the computational time. Such algorithms (DDDP and DDP) have been used in this work, which are being presented in the next chapters.
- The latest and ongoing evolution and changes in the field of computer science are being used, in order to investigate the dimensions of the Discrete Dynamic Programming applications. The simple structure of the Dynamic Programming multi-step procedure is offered for parallel processing in multiple computer systems.

3.5: Discrete Differential Dynamic Programming (DDDP)

The Discrete Differential Dynamic Programming algorithm solves in every iteration the discretized problem introduced in subsection 2.2.1, with the following, additional, constrains

$$\left|x_i(k) - x_i^{(l-1)}(k)\right| \leq \Delta_i^{(l)},\ i = 1, \dots, n \tag{3.20}$$

where$x_i^{(l-1)}$ is the determined optimal trajectory for the last iteration $l-1$. In other words, the discretized problem is solved in every iteration into a corridor with $\Delta^{(l)}(k)$width, about the previous approximation $x_i^{(l-1)}(k)$, which reduces significantly the corresponding work load and computational time. The selection of the $\Delta^{(l)}(k)$width, and also the discrete intervals $\Delta\boldsymbol{u}^{(l)}, \Delta\boldsymbol{x}^{(l)}$, can all be different (meaning, they can be changing from iteration to iteration) in every iteration, with declining tendency of course. The convergence criterion is satisfied if

$$\left\|\boldsymbol{u}^{(l)}(k) - \boldsymbol{u}^{(l-1)}(k)\right\|^{\infty} < \varepsilon \quad \forall k \in [0, K-1] \tag{3.21}$$

where$\varepsilon > 0$is a tolerance limit, and $\|\boldsymbol{u}(k)\|^{\infty} = \max_i(|u_i(k)|)$.

Below, an example is presented, aiming to the better understanding of the method [20].

Example 2

Minimization of

$$J = \frac{1}{2}\sum_{k=0}^{3}[x(k)^2 + u(k)^2]$$

considering

$$x(k+1) = x(k) + u(k), x(0) = 2, x(4) = 0$$

$$\mathfrak{X} = \{x(k) | 0 \leq x(k) \leq 2\}$$

$$\boldsymbol{\mathcal{U}} = \{u(k) | -2 \leq u(k) \leq 0\}$$

As an initial trajectory the feasible trajectory $u^{(0)}(0) = u^{(0)}(1) = 0, u^{(0)}(2) = u^{(0)}(3) = -1$ is designated, which corresponds to $x^{(0)}(0) = x^{(0)}(1) = x^{(0)}(2) = 2, x^{(0)}(3) = 1, x^{(0)}(4) = 0$, with cost equal to 7.5. In **Figure 4** below, a graphic approach of the problem's solution is being presented, using the DDDP method, displaying for every iteration $l = 1, \dots, 5$:

- The previous state approximation $x^{(l-1)}(k)$, presented by a solid line.
- The discrete points of the solution's corridor.
- The new state approximation $x^{(l)}(k)$, presented by a dashed line, and the corresponding cost (presented by a circled number on top of the initial state).

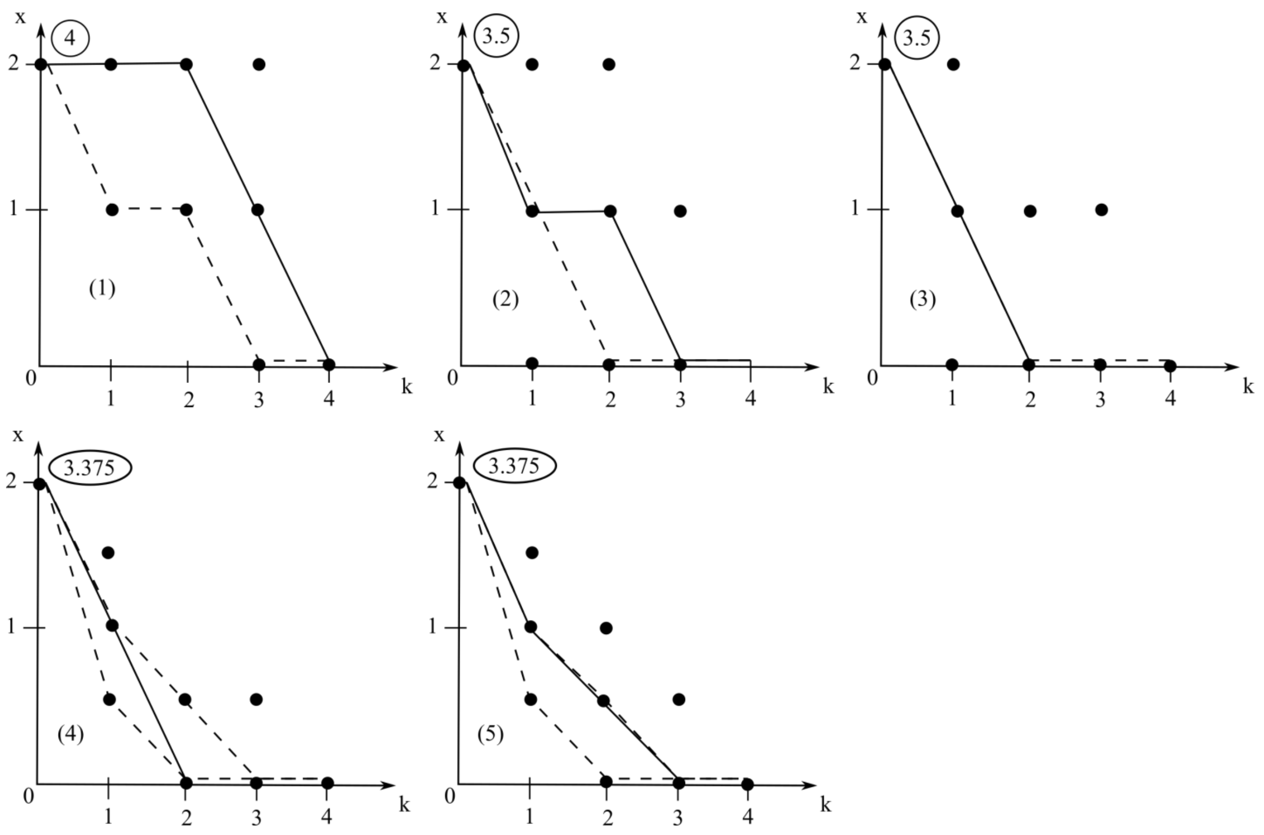


Figure 4: Consecutive approximations, regarding Example 2. Adapted with permission from [20].

In every iteration, the width of the corridor is considered to be equal to a discrete interval Δx. The width of the corridor is equal to zero for times 0 and 4, due to the specified initial and final state. It is also considered that the corridor doesn't expand to not feasible areas, meaning it is restrained from 0 to 2, $(0 \leq x(k) \leq 2)$ in this particular problem, and generally specified depending on the problem constrains.

The first three iterations taking place under the discrete intervals $\Delta x = \Delta u = 1$. During the first iteration, the cost criterion is being reduced from 7.5, to 4, and continuing, during the second iteration declines from 4 to 3.5. In the third iteration, it is obvious that no further improvement can be achieved. Reducing the discrete intervals from $\Delta x = \Delta u = 1$to $\Delta x = \Delta u = 0.5$, a cost criterion reduction is achieved, as it declines from 3.5 to 3.375 during the fourth iteration. In the fifth iteration no further improvement can be achieved, so the process can be stopped here, or it could be continued by further reduction of the discrete intervals, in order to improve further the problem's solution.

3.5.1: Advantages and Disadvantages of the DDDP method

This particular method is well-known and used in a variety of applications, due to the reduction in the workload and the faster results compared to other similar methods, as it can be observed by Feng et. al. in [23], that the “curse of dimensionality” (mentioned in Chapter 2) is posing a great challenge to the optimal operation of a hydropower system (OOHS) due to the exponential growth of the computational cost, with the increasing number of plants and the DDDP method is used, as well as Heidariet. al. state in [24], where an approach to water resources systems optimization is presented, based on Discrete Differential Dynamic Programming. According to [24], ,the major factors that led to a DDDP approach in their paper were the inherent drawbacks and disadvantages of a «traditional» dynamic programming approach, such as, memory, capacity and computer time requirements. By limiting optimization to a few points grid around the initial trajectory, the memory requirements appear to have been curbed considerably. So, a DDDP method can be quite applicable in many cases, if the workload of a DP method is considered, versus the one that a DDDP algorithm has to overcame, which is substantially less, as stated before.

The percentage of saved computational time, through a DDDP method depends on $\boldsymbol{\Delta}$, the corridor’s width, and also the number of iterations performed, which generally depends on the corridor’s $\boldsymbol{\Delta}$ width. In a problem with $n = 5$ and $a = 100$ for example, according to ***Eq. (3.20)***, the computational time of the problem is equal to $100^5 = 10^{10}\, sec$. Considering a corridor’s width equal to five discrete points, for every iteration of the DDDP algorithm there is computational time equal to $5^5 = 3125\, sec$. However, the exponential increase of computational time still exists in a DDDP algorithm, but with a much smaller impact compared to a Discrete Dynamic Programming algorithm.

The disadvantages of a DDDP algorithm compared to a Discrete Dynamic Programming algorithm are:

- Convergence of the algorithm can be proved only if relevant restrictive admissions take place.
- The possibility of convergence in a relative minimum cannot be excluded, except of specific problem categories.
- A control rule is computed only for the last iteration’s corridor, and not for the whole feasible state area $\mathfrak{X}(k)$.

Concluding, it is clear that someone can argue that there are available methods faster than a DDDP algorithm, and the convergence times are even lower. Nevertheless, it is important to take into consideration that a DDDP algorithm provides us with high precision results, in a quite small amount of time, mostly due to the use of the corridor Δ, considering the computational time that a DP algorithm is going to need to solve the exact same problem and also provide the same results. It is important to be mentioned that, the initial SDP algorithm [4], needs approximately from 10 to 20 minutes to solve and provide back the wanted results of a certain OCP. On the contrary, a same formulated problem, using a DDDP algorithm, and specifically the one that is described in chapter 5, is under the barrier of a second, meaning that a significant amount of computational time and workload is being avoided.

3.6: Differential Dynamic Programming (DDP)

Differential Dynamic Programming avoids the exponential increase of computational time, by leading in a quadratic convergence, inside an area around the wanted minimum, meaning that

$$\left\|\boldsymbol{u}^{(l)}(k)-\boldsymbol{u}^{*}(k)\right\|^{\infty} \leq A\left[\left\|\boldsymbol{u}^{(l-1)}(k)-\boldsymbol{u}^{*}(k)\right\|^{\infty}\right]^{2}, \text{ where } A<1. \tag{3.22}$$

In terms of better understanding, the DDP method is going to be presented for the case of graded state $x(k)$ and graded control $u(k)$. Generalization of the method in a vector instance does not present any methodology difficulties, although its presentation includes some quite complicated vector terms [20].

Let's consider the OCP from section 3.1, without taking into consideration the final target ***(3.3)*** and also without the inequality constrains ***(3.4)*** and ***(3.5)***. Except that, the admission that functions θ, φ and f are sufficiently differentiable for the following calculations is being made, something that narrows the implementation amplitude of the method. In order to properly present the algorithm, the following equations are being defined

$$\delta x(k)=x(k)-x^{(l-1)}(k)$$

$$\delta u(k)=u(k)-u^{(l-1)}(k) \tag{3.23}$$

The basic principle of the algorithm lies on the solution, for each iteration l, of a quadratic approximation of the initial OCP, based on the last solution $x^{(l-1)}(k)$, $u^{(l-1)}(k)$. The advantage of this procedure emanates from the fact that the solution of a quadratic approximation of the OCP; with no constrains involved, is possible analytically, without the use of a discretization variable, something that prevent the workload from increasing exponentially.

The procedure starts with the last step $K-1$, according to the algorithm presented in section 3.1, assuming the Bellman's retroactive ***Eq. (3.11)***. The under minimization term of ***Eq. (3.11)***, for $k=K-1$, is

$$\varphi[x(K-1), u(K-1), K-1]+\theta\big[f[x(K-1), u(K-1), K-1]\big] \tag{3.24}$$

The quadratic approach of this term, under the values $x^{(l-1)}(K-1)$, $u^{(l-1)}(K-1)$ returns

$$Q[x(K-1),u(K-1),K-1] = \frac{1}{2}D(k-1)\delta x(K-1)^2 + E(K-1)\delta x(K-1)\delta u(K-1) + \\ + \frac{1}{2}F(K-1)\delta u(K-1)^2 + G(K-1)\delta x(K-1) + H(K-1)\delta u(K-1) \quad (3.25)$$

where the zero grade constant term was omitted, because it does not affect the outcome of minimization. MatricesD, E, F, G and $H$of ***Eq. (3.25)*** can be computed as follows (with the same principles as for array D below):

$$D(K-1) = \frac{\partial^2\varphi[x(K-1),u(K-1),K-1]}{\partial x(K-1)^2} + \frac{\partial^2\theta[x(K)]}{\partial x(K)^2}\left[\frac{\partial f[x(K-1),u(K-1),K-1]}{\partial x(K-1)}\right]^2 \\ + \frac{\partial\theta[x(K)]}{\partial x(K)}\frac{\partial^2 f[x(K-1),u(K-1),K-1]}{\partial x(K-1)}$$

(3.26)

where every term is being computed for $x^{(l-1)}(k)$, $u^{(l-1)}(k)$.

Minimization of the quadratic approach Q, ***Eq. (3.25)***, with respect to $u(K-1)$ produces

$$u(K-1) = u^{(l-1)}(K-1) + \delta u(K-1) \quad (3.27)$$

with

$$\delta u(K-1) = a(K-1) + \beta(K-1)\delta x(K-1) \quad (3.28)$$

$$a(K-1) = -F(K-1)^{-1}H(K-1) \quad (3.29)$$

$$\beta(K-1) = -F(K-1)^{-1}E(K-1) \quad (3.30)$$

Using again ***Eq. (3.9)***, the approximate function $\hat{V}[x(K-1),K-1]$, without the constant term, from the minimum value of Q is

$$\hat{V}[x(K-1),K-1] = \frac{1}{2}A(K-1)\delta x(K-1)^2 + B(K-1)\delta x(K-1) \quad (3.31)$$

with

$$A(K-1) = D(K-1) - E(K-1)F(K-1)^{-1}E(K-1) \quad (3.32)$$

$$B(K-1) = G(K-1) - E(K-1)F(K-1)^{-1}H(K-1) \quad (3.33)$$

Computation of steps below $K-1$ is done in a similar way. Suppose that the by quadratic approximation function

$$\hat{V}[x(k+1),k+1] = \frac{1}{2}A(k+1)\delta x(k+1)^2 + B(k+1)\delta x(k+1) \tag{3.34}$$

is already been computed. Defining $Q[x(k),u(k),k]$ as a quadratic approximation of the term $\varphi[x(k),u(k),k] + \hat{V}[f[x(k),u(k),k],k+1]$, with respect to $x^{(l-1)}(k)$, $u^{(l-1)}(k)$ and without the constant term, the corresponding equation to ***Eq. (3.25)*** for the interval k is

$$Q[x(k),u(k),k] = \frac{1}{2}D(k)\delta x(k)^2 + E(k)\delta x(k)\delta u(k) + \frac{1}{2}F(k)\delta u(k)^2 + G(k)\delta x(k) + \\ +H(k)\delta u(k) \tag{3.35}$$

ArraysD, E, F, G and H of the above equation can be calculated as follows (using the same principles as for array D below):

$$D(k) = \frac{\partial^2\varphi[x(k),u(k),k]}{\partial x(k)^2} + A(k+1)\left[\frac{\partial f[x(k),u(k),k]}{\partial x(k)}\right]^2 + B(k+1)\frac{\partial^2 f[x(k),u(k),k]}{\partial x(k)} \tag{3.36}$$

where every term is being computed for $x^{(l-1)}(k)$, $u^{(l-1)}(k)$.

Function Q can now be minimized, with the same principles as in interval $K-1$, with respect to $u(k)$,

$$u^{(l)}(k) = u^{(l-1)}(k) + a(k) + \beta(k)\delta x(k) \tag{3.37}$$

where$a(k), \beta(k)$ are being calculated according to ***Eq. (3.29)*** and ***(3.30)***. From this minimization the approximation $\hat{V}[x(k),k] = \frac{1}{2}A(k)\delta x(k)^2 + B(k)\delta x(k)$ also transpires, where $A(k)$, $B(k)$ are being calculated using ***Eq. (3.32)*** and ***(3.33)***. Vectors $a(k)$, $\beta(k)$, need to be stored in every step of the method, counter to arrays $A(k+1)$ and $B(k+1)$, which do not have a further use and can be erased from the computer's memory.

After the calculation of all the steps of the problem, an improved version of the control variable, for every iteration l, can be formulated

$$u^{(l)}(k) = u^{(l-1)}(k) + a(k) + \beta(k)\delta x(k) \tag{3.38}$$

$$x^{(l)}(k+1) = f\left[x^{(l)}(k),u^{(l)}(k),k\right], x(0) = x_0 \tag{3.39}$$

In some instances, the use of the following ***Eq. (3.40)*** might be needed, instead of ***Eq. (3.38)***, in order to avoid a possible deviation of the algorithm.

$$u^{(l)}(k) = u^{(l-1)}(k) + \varepsilon[a(k) + \beta(k)\delta x(k)], 0 < \varepsilon \leq 1 \tag{3.40}$$

Nevertheless, it is possible even with this precaution (the use of constant ε), that, derivation of the algorithm might occur, so, other measures need to be taken. It is also important to mention that, the state and control variables do not have bounds, meaning that $\boldsymbol{x}$ and $\boldsymbol{u}$are not bordered amidst some ensuing regions.

3.6.1: Advantages and Disadvantages of the DDP method

The DDP method solves the initial OCP without the use of a discretization value, since the quadratic approximation problem is being solved analytically, so the computational time is being reduced even more than the DDDP method, something that makes this a very useful and vastly used method, like in an interesting work by Tassaet. al. [25], considering ways for generating goal-directed robot motion using a DDP algorithm, as well as in a similar work released by Levine and Koltun, [26], considering Variational Policy Search via trajectory optimization, using also DDP algorithmic implementations.

A Differential Dynamic Programming application can be also expanded in problems with linear inequality constrains. In this particular case, in every step of every iteration, a quadratic programming problem is being solved, something that increases the workload, however, retains the increase of the computational time as to the dimensions of the problem in a polynomial way.

The disadvantages of the DDP method, compared with a Discrete Dynamic Programming method can be summarized below

- The possibility that the algorithm does not converge, always exist.
- The possibility that the algorithm converges to a relative minimum.
- There is a difficulty taking into consideration generic inequality constrains, or allowed discrete range of values.

As stated by Pan et. al. [27], comparing a DDP algorithm with global optimal control approaches, the local optimal DDP shows superior computational efficiency and scalability to high-dimensional problems, due to the fact that is derived based on linear approximations of the nonlinear dynamics along state and control trajectories, therefore it relies on accurate and explicit dynamics models.

As specified by Levine et. al. [26], their results show that the developed algorithm (which is based on a variant of DDP called iterative LQR) outperforms prior methods because of two advantages: the use of a model-based trajectory optimization algorithm instead of random exploration, which allows the algorithm to outperform model-free methods, and also due to the decomposition of the policy search into two simple optimization problems that can each be solved efficiently by standard algorithms, which leaves this specific method less vulnerable to local optima than more complex methods like Guided Policy Search (GPS), introduced by Levine et. al. in [28].

Chapter 4: GLOSA problem formulation

This chapter contains the GLOSA problem implemented in this thesis, from its mathematic viewpoint, starting from the OCP with known signal switching times, which is a simplified version of the problem that is being researched. Continuing, the OCP with unknown signal switching times is being presented, which is a lot more complicated than the previous one, and along with that, the workload and the CPU-time are exponentially increasing, and that is the main reason that the two methods mentioned in this chapter (Discrete Differential Dynamic Programming and Differential Dynamic Programming) are being used in order to solve the Optimal Control Problem investigated in this thesis.

4.1: Optimal Control Problem with Known Signal Switching Time

In this subsection the theoretical background and the method followed of the algorithm introduced in [4] are presented, when the switching time is known in advance. Taking into consideration **Figure 5**, the studied vehicle starts from an initial state $\boldsymbol{x}_0 = [x_0, v_0]^T$, with $\boldsymbol{x}_0$ a given initial position and v_0 a given initial speed of the vehicle, which intends of reaching a final position x_e with a final speed v_e, with the limitation that the vehicle isn't able to pass through the traffic light (which current position is at let's say x_1) before the time t_1, which is the time that the traffic light turn from red into green. The purpose of the driver is to reach at his final destination, with the minimum possible fuel consumption, by regulating his acceleration accordingly with respect to the initial and final conditions $\boldsymbol{x}_0$and $\boldsymbol{x}_e$, along with not violating the traffic signal constraint. In order to avoid a possible myopic operation of the developed algorithm, it is necessary to take into consideration that the final position $\boldsymbol{x}_e$ needs to be adequate downstream of the traffic signal, e.g. 75 m., along with the final speed v_e, which needs to be sufficiently high (e.g. 11 m/s).

The purpose of the driver is to reach at his final destination before the time that the traffic light turn from red into green.

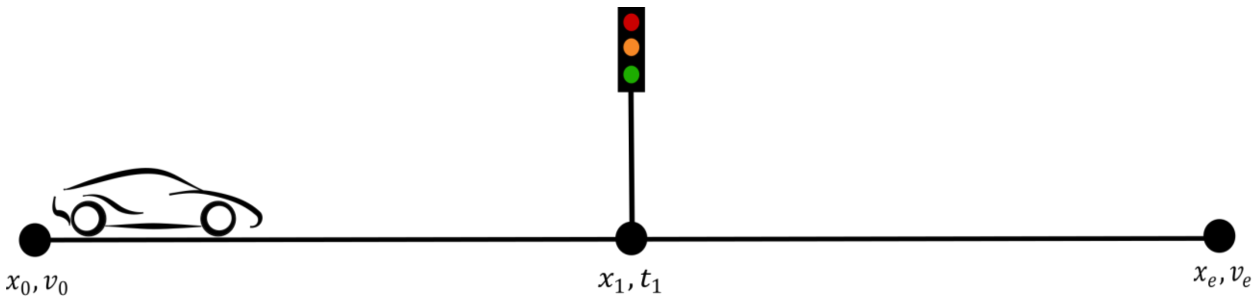


Figure 5: A vehicle starting from a given position x_0 and speed v_0, starting from a known initial state$[x_0, v_0]^T$, the traffic light with known position x_1 and also known switching to green time t_1.The main target is to enable the vehicle (by adjusting appropriately its speed and acceleration, which in this particular case acceleration is also our control variable) to reach at a wanted and also given state x_e with a speed v_e without stopping at the traffic light, which means passes it at the right time, meaning when the light is turned into green.

4.1.1: Problem formulation

The minimization problem presented above is formulated as an optimal control problem which accounts for the vehicle kinematics through the following state equations:

$$\dot{x} = v \tag{4.1}$$

$$\dot{v} = a \tag{4.2}$$

where$a$is the vehicle's acceleration which has the role of the control variable. The target is to navigate the driver or the vehicle from the initial state $\boldsymbol{x}_0 = [x_0, v_0]^T$to the determined final condition $\boldsymbol{x}_e = [x_e, v_e]^T$within the final time $t_{e,}$while minimizing the cost criterion

$$J = w \cdot \mathrm{t_e} + \frac{1}{2}\int_0^{\mathrm{t_e}} a^2 dt \tag{4.3}$$

Furthermore, the green light constraint $t_s \geq t_1$ must be respected, in which t_s is the time vehicle crosses from the signal position x_1. Also, the final time t_e is free (penalized by the parameter *w*), due to the desire to have a flexible problem formulation that applies to a multitude of certain instances of: initial positions (near and far beyond the traffic signal), initial speeds (low or high) and switching times. In addition, if the above not accounted for in the cost criterion, there is a chance of problem instances emerging and eventually leading to immoderate final time. After consideration, a minimum acceleration cost (a^2) is accomplished with*w* = 0, but with a high enough price for the final time t_e and therefore, an appropriate trade-off amidst the acceleration cost and time $t_e$was found with a value of w equal with 0.1, and the explanation of why w is equal to 0.1, is presented in the following paragraph, along with **Figure 7**. Finally, lower and upper limits can be placed to speed and acceleration a, bounding the speed in the middle of minimum (which in this case is zero) and maximum values, between minimum and maximum values for the acceleration.

Tracking and finally choosing the proper weighting factor (w) for every particular scenario investigated, plays a significant role in the shape and the outcome of the optimal trajectories. By creating the following diagram using the setup of Scenario 2, the behaviour of the weighting factor w in relation to the final free time t_e seems to be the exact opposite, since for different weighting factor values, but with identical boundary values, as w increases, the final free time decreases analogously.

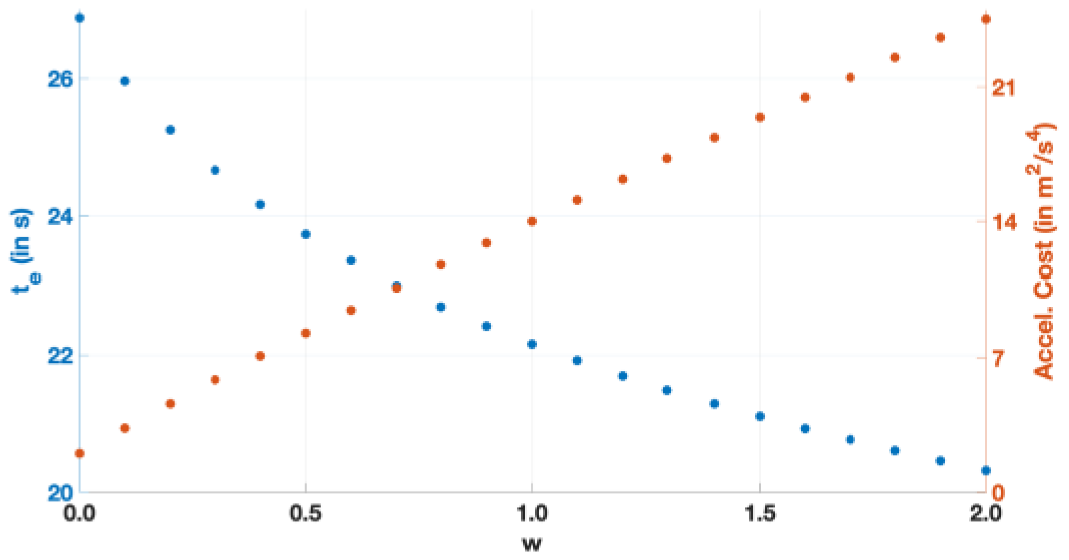


Figure 7: Optimal final time, acceleration cost, in comparison with weight w. Adapted with permission from [4].

4.1.2: Analytical solution

Firstly, the use of the Hamiltonian function [29], [19] is necessary in favour of the necessary conditions of the analytical solution of the problem

$$H(x, \lambda) = \varphi(x, a) + \lambda^T f(x, u) \tag{4.4}$$

in which λ represents the co-state vector for the analogous state equations and$\varphi(x,a) = \frac{1}{2}a^2$ represents the running cost. For this exact problem, the Hamiltonian function is:

$$H(v, a, \lambda_1, \lambda_2) = \frac{1}{2} + \lambda_1 v + \lambda_2 \alpha. \tag{4.5}$$

Following are presenting the necessary (for this particular problem's formulation) conditions of optimality

$$\dot{x} = \frac{\partial H}{\partial \lambda_1} = v \tag{4.6}$$

$$\dot{v} = \frac{\partial H}{\partial \lambda_2} = a \tag{4.7}$$

$$\dot{\lambda_1} = -\frac{\partial H}{\partial x} = 0 \tag{4.8}$$

$$\dot{\lambda_2} = -\frac{\partial H}{\partial v} = -\lambda_1 \tag{4.9}$$

$$\frac{\partial H}{\partial \alpha} = 0 \tag{4.10}$$

All the above equations must be fulfilled, along with the initial and final state conditions for certain problem formulation. Except that, as a result of the free final time t_e, an extra condition for optimality [29], [19] is added

$$H(t_e) + w = 0. \tag{4.11}$$

Initially, the unconstrained problem (UP) is taking into consideration, which means that the intermediate green light constraint is not considered. ***Eq. (4.8) – (4.10)***promptly wield an analytic linear-in-time optimal acceleration solution in this specific problem that is considering in this chapter [30], [31]

$$a(t) = c_1 t + c_2 \tag{4.12}$$

and after integration in agreement with ***Eq. (4.6), (4.7)***, also the speed and position solutions are

$$v(t) = \frac{1}{2}c_1 t^2 + c_2 t + c_3 \tag{4.13}$$

$$x(t) = \frac{1}{6}c_1 t^3 + \frac{1}{2}c_2 t^2 + c_3 t + c_4 \tag{4.14}$$

where$c_1, \dots, c_4$ are four integration constants, which, along with the optimal final time t_e, may be considered as the solution of a system of five algebraic equations, including the initial and final states and the final time condition ***Eq. (4.11)***.

Continuing with the constrained problem (CP), at which the vehicle's position is required at some given time t_c to be at the traffic signal, i.e. $x(t_c) = x_1$. In this instance, all necessary conditions (***Eq. (4.6)-(4.11)***) remain the same, but there is an additional necessary condition that must be fulfilled (see[19]), which, for this particular problem, suggests that the co-state $\lambda_1(t)$ may be dis-continuous at the intermediate time t_c. This and ***Eq. (4.8)-(4.10)***then compose a continuous two-branch piece-wise linear optimal acceleration solution

$$a(t) = \begin{cases} c_1 t + c_2 & 0 \leq t \leq t_c^- \\ c_5 t' + c_6 t_c^+ \leq t \leq t_e \end{cases} \tag{4.15}$$

and after integration with ***Eq. (4.6), (4.7)***, the corresponding speed and position solutions are

$$v(t) = \begin{cases} \frac{1}{2} c_1 t^2 + c_2 t + c_3 & 0 \leq t \leq t_c^- \\ \frac{1}{2} c_5 t'^2 + c_6 t' + c_7 t_c^+ \leq t \leq t_e \end{cases} \tag{4.16}$$

$$x(t) = \begin{cases} \frac{1}{6} c_1 t^3 + \frac{1}{2} c_2 t^2 + c_3 t + c_4 & 0 \leq t \leq t_c^- \\ \frac{1}{6} c_5 t'^3 + \frac{1}{2} c_6 t'^2 + c_{7t'} + c_8 t_c^+ \leq t \leq t_e \end{cases} \tag{4.17}$$

where$t' = t - t_c$ and $c_1, \ldots, c_8$ are eight integration constants, which, along with the optimal final time t_e, may be specified as the solution of a system of nine algebraic equations, including the initial and final states, the final time condition ***Eq. (4.11)***, the continuity conditions for control and states $a(t_c^-) = a(t_c^+), v(t_c^-) = v(t_c^+), x(t_c^-) = x(t_c^+)$ and the intermediate condition $x(t_c) = x_1$.

$J_{CP}(t_c)$is denoted as the optimal objective value of the constrained problem (CP) as a function of t_c and J_{CP}, the optimal objective value of the unconstrained problem (UP). Considering CP was derived by additional constraining of UP, $J_{CP}(t_c) \geq J_{UP}\ \forall t_c$, and, as a matter of fact, $J_{CP}(t_c)$obtains its minimum value for$t_c = t_{s,UP}$, where $t_{s,UP}$ isthe time when the UP trajectory attains the traffic signal position x_1. Additionally, it was verified that, as expected, the function $J_{CP}(t_c)$ increases monotonically for $t_c \geq t_{s,UP}$.

These observations conclude to the following approach for finding the solution of the original GLOSA problem in presence of a traffic signal at x_1 and switching time to greent_1:

a) Solve the Unconstrained Problem (UP). If $t_{s,UP} \geq t_1$, the GLOSA problem is solved, as the UP solution respects the green-light constraint. Otherwise:

b) Solve the Constrained Problem (CP). with $t_c = t_1$ to acquire the GLOSA solution.

Regarding the indications above, the solution of UP or CP for a specific problem instance requires the solution of a corresponding system of five or nine, respectively, algebraic equations. Those solutions were obtained analytically (yielding lengthy formulas) using symbolic differentiation tools (e.g. Mathematica [32]). The numerical solution of those analytical formulas to obtain the solution of (UP) or (CP), and subsequently of the GLOSA problem for a specific problem instance, needs only fractions of a second of computation time and as a result it can take place within the vehicle once the next switching time becomes known. Also, it may be reasonable to continuously update the vehicle trajectory, in a model predictive control (MPC) loop,to account for possible deviations from the first computed vehicle trajectory, something that may happen as a consequence of a variety of disturbances, for instance, a slower vehicle ahead. For a given junction, the final state is the same for any initial vehicle state $\boldsymbol{x}_0$ and any switching time t_1. Therefore, the optimal value of the cost criterion,***Eq. (3.7)***, of the deterministic GLOSA problem depends on these variables, and is denoted as$J_{CP}^*(x_0, t_1)$.

4.2: Optimal Control Problem with Uncertain Signal Switching Time

The fact that this particular problem cannot be approached analytically, leads to the use of various discrete techniques in order to reach a complete solution of the OCP. Following, the case of uncertain signal switching times is being presented.

In this section, the case of not known traffic light switching times is being presented, or when subjected in of short term decisions due to real-time signals. When that happens, based on proper statistics from past signal switching activity, vacancy of a time-window of viable signal switching times, along with the equivalent probability distribution can be presumed, in which case the problem can be transformed into a stochastic optimal control problem, which may be solved numerically using SDP (Stochastic Dynamic Programming) techniques. Except that, the analytical solution of the deterministic GLOSA optimal control problem obtained in section4.1and 4.1.2 is used within the stochastic approach, which is presented and explained following.

4.2.1: Problem variables and state equations

Stochastic Dynamic Programming algorithms have the need for a discrete-time system the discrete-time version of the vehicle kinematics with time-step T is formulated as shown following [33]:

$$x(k+1) = x(k) + v(k)T + \frac{1}{2}a(k)T^2 \tag{4.18}$$

$$v(k+1) = v(k) + a(k)T \tag{4.19}$$

where $x(k)$, $v(k)$relate to the vehicle's position and speed at discrete times $k = 0,1,2,\ldots$ while the acceleration (control variable) $a(k)$ remains constant during every time period k. The following feasible regions (***Eq. (4.20)*** and ***Eq. (4.21)***) bound the state and control variables

$$\boldsymbol{x}(k) \in \boldsymbol{X} = [x_{min}, x_{max}] \tag{4.20}$$

$$\boldsymbol{a}(k) \in U = [a_{min}, a_{max}] \tag{4.21}$$

where x_{min}, x_{max} and a_{min}, a_{max} are the upper and lower bounds of the state and control variables. The traffic light's switching time k_1 is not certain, but it is assumed that $k_1$exist in a known range $k_{min} \le k_1 \le k_{max}$, where k_{min} and k_{max} are the minimum and maximum possible switching times.

In order to achieve an appropriate problem formulation, a virtual state $\tilde{x}(k)$ is introduced, which accounts for the stochasticity of the traffic light's operation

$$\tilde{x}(k+1) = \tilde{x}(k) \cdot z(k) \tag{4.22}$$

$$\tilde{x}(0) = 1$$

where$z(k)$ is a discrete stochastic variable represented as

$$z(k) = \begin{cases} 0 & if\ traffic\ light\ switches\ at\ time\ k+1 \\ 1 & else \end{cases} \tag{4.23}$$

Based on ***Eq. (4.22), (4.23)***, the virtual state $\tilde{x}(k)$is either equal to 1, if the traffic light has not yet switched until time $k-1$, or equal to zero if switching takes place at time k or before. The virtual state $\tilde{x}(k)$is assumed measurable, meaning that the system is aware at each time kT of any switching that has taken place or not, within the last time period $[(k-1)T, kT]$.

The stochastic variable $z(k)$ is independent from its previous values $z(k-1), z(k-2), \ldots$ and takes values in according to a time-dependent probability distribution $p(z|k)$. Depending on the statistics of previous signal switching activity, availability of an a-priori discrete probability distribution $P(k), k_{min} \le k_1 \le k_{max}$ is assumed, for signal switching within the time-window, where $\sum_{\kappa=k_{min}}^{k_{max}} P(k) = 1$. In view of no switching takes place for $k \le k_{min} - 1$, then

$$p(0|k) = 0\ for\ k < k_{min} - 1. \tag{4.24}$$

When $k = k_{min}$, traffic light switching may take place, with a-priori probability $P(k_{min})$. Consequently

$$p(0|k_{min} - 1) = P(k_{min}). \tag{4.25}$$

In the case of the traffic light has not switched at time $k = k_{min}$, the probabilities of switching at some point later within the time-window are increased, contrasted with the respective a-priori distribution, and the updated probabilities can be computed through "crop and scale"[34], which means that the a-priori probability $P(k_{min})$ is distributed analogously to increase the probabilities of the remaining discrete times, inside of course the time-window. Using this same argument, the crop-and-scale procedure for updating the switching probabilities need to be carried out at each following time step, as long as the switching has not taken place. This update may be accomplished by use of the following crop-and-scale formula that applies for $k_{min} \leq k \leq k_{max} - 1$ and for any a-priori distribution $P(k)$

$$p(0|k) = P(k+1)\left[1 + \frac{\sum_{\kappa=k_{min}}^{k} P(k)}{\sum_{\kappa=k+1}^{k_{max}} P(k)}\right] \tag{4.26}$$

where the term inside the square brackets displays the crop and scale update.

4.2.2: Objective criterion

The cost criterion of the stochastic problem is the same as in the deterministic case ***Eq. (4.3)***. Nevertheless, in the stochastic case, the exact value of the criterion depends on the stochastic variable's realization, and consequently its expected value is being minimized

$$J = E\left\{w \cdot \mathrm{t_e} + \frac{1}{2}\int_0^{\mathrm{t_e}} \alpha^2 dt\right\} \tag{4.27}$$

where the expectation refers to the stochastic variable $z(k), k = 0, \dots, k_{max} - 1$. Also, when the switching time becomes known at time k_1, while the vehicle is at state $\boldsymbol{x}(k_1)$, proceeding to the traffic signal, the problem immediately turns into a deterministic GLOSA problem, and the corresponding optimal cost-to-go is $J_{DG}^*[\boldsymbol{x}(k_1), k_1]$, which will be denoted as the escape cost. According to the Principle of Optimality [35], obtained from ***Eq. (4.27)***, after introducing discrete-time notation

$$J = E\left\{\frac{1}{2}\sum_{k=0}^{k_{max}-1} a(k)^2 + J_{DG}^*[\boldsymbol{x}(k_1), k_1]\right\} \tag{4.28}$$

To obtain a formally proper cost criterion [33], the stochastic variable $z(k)$ and the virtual variable $\tilde{x}(k)$ introduced earlier are used, replacing the state $\boldsymbol{x}(k_1)$, from ***Eq. (4.18), (4.19)*** as a function of the state and acceleration of the previous time period. This yields the objective function in the necessary form, which is

$$J = E\left\{\tilde{x}(k)\sum_{k=0}^{k_{max}-1}\left[\frac{1}{2}a(k)^2 + [1 - z(k)]J_{DG}^*[\boldsymbol{x}(k), a(k), k+1]\right]\right\} \tag{4.29}$$

Eq. (4.18)-(4.26)and ***Eq.(4.29)***constitute an ordinary stochastic optimal control problem. Denoting the corresponding optimal cost-to-go function by $V[x(k), \tilde{x}(k), k]$,the Bellman's Recursive Equation [33], for $0 \leq k \leq k_{max} - 1$reads

$$V[x(k), \tilde{x}(k), k] = \min_{a(k)\in U} \left\{ E\left\{ \frac{1}{2}a(k)^2 + [1 - z(k)]J_{DG}^*[\boldsymbol{x}(k), a(k), k+1] + \right.\right.$$
$$\left.\left. + V[x(k+1), \tilde{x}(k)z(k), k+1] \right\}\right\} = \min_{a(k)\in U} \left\{ \frac{1}{2}a(k)^2 + p(0|k) \cdot J_{DG}^*[\boldsymbol{x}(k), a(k), k+1] + \right.$$
$$\left. + [1 - p(0|k)] \cdot V[x(k+1), 1, k+1] \right\} \qquad (4.30)$$

with boundary condition $V[x(k_{max}),1, k_{max}] = 0$. The minimum is required with respect to $a(k) \in U$ only, as typical in Dynamic Programming, which facilitates the numerical solution.

In this formulation, it is assumed that the decision on traffic light switching is taken between the last time period before the actual switching. The generalization to the case of taking a switching decision κ timeperiods ahead of the actual switching is simple. Accordingly, the time k_1 and the definition of the stochastic variable $z(k)$reflect the decision time (instead of the switching time), and the only change needed in the above equations is that the escape function J_{DG}^* needs to include as an argument for the switching time $k + \kappa$ instead of $k + 1$.

4.3: Numerical solution algorithm

For the sake of implementation of the discrete Stochastic Dynamic Programming algorithm for the numerical solution of the problem, the state and control variables need to be discretized. The level of discretization has an important impact on computational time, memory requirements and the amount of workload the application has to overcome, but also on the accuracy of the computed solution. Consequently, an appropriate trade-off needs to be specified concerning reasonable computation requirements against achievable solution quality.

For the discretization, firstly the discrete time-interval has been set as T = 1 s, which is a reasonable choice for the problem at hand. Continuing, is assumed a general discretization interval $\varDelta$ for the problem variables and the discretization interval of acceleration is set as$\varDelta a = \varDelta$. Taking into consideration ***Eq. (3.19)***, it is obvious that the speed and acceleration intervals are equivalent, and, hence, the discretization interval of speed can also assume the same value ($\varDelta v = \varDelta a = \varDelta$). By the same logic, in view of ***Eq. (3.18)***, the discretization interval for the position has being set according to the following

$$\varDelta x = \frac{1}{2}\varDelta \cdot T^2 \qquad (4.31)$$

Based on the above, it can be proved that, if $x(k), v(k), a(k)$ are discrete points, then $x(k + 1)$ and $v(k + 1)$ (resulting from ***Eq. (4.18)***and***(3.19)***) are also discrete points. It is then assumed that

$$x(k) = n\Delta x \tag{4.32}$$

$$v(k) = m\Delta v = m\Delta \tag{4.33}$$

$$a(k) = l\Delta a = l\Delta \tag{4.34}$$

wheren, m, l are positive integers. From ***Eq. (4.18)***

$$\begin{aligned} x(k+1) &= n\Delta x + m\Delta \cdot T + \frac{1}{2} l\Delta \cdot T^2 \\ &= \frac{1}{2} n\Delta \cdot T^2 + m \cdot \Delta + \frac{1}{2} l\Delta \cdot T^2 \\ &= \tfrac{1}{2}\Delta \cdot T^2 \left(n + \tfrac{2}{T} m + l\right) = \Delta(n + 2m + l) \end{aligned} \tag{4.35}$$

which proves that $x(k + 1)$ is indeed a discrete point, and the same holds trivially true for $v(k + 1)$ also in view of ***Eq. (4.19)***.

It is now easy to apply the discrete SDP algorithm to obtain an optimal closed-loop control law $a(k)^* = R[\boldsymbol{x}(k), k]$, which, for any specific vehicle state $\boldsymbol{x}(k)$ and time $k$carries out and delivers the optimal acceleration $a(k)$. A full vehicle trajectory can also be achieved by beginning at an initial state and time and following the optimal encountered acceleration values, and stop when the final state is reached.

The SDP algorithm is described as follows:

```
$V[\boldsymbol{x}(k_{max}), k_{max}] = 0 \quad \forall \boldsymbol{x}(k_{max}) \in \boldsymbol{X}$
for each $k = k_{max} - 1, \dots, 0$ do
    for each discrete state $\boldsymbol{x}(k) \in \boldsymbol{X}$ do
        for each discrete control $a(k) \in U$ do
            Calculate $x(k+1), v(k+1)$
            if $\boldsymbol{x}(k+1) \notin \boldsymbol{X}$
                $J[\boldsymbol{x}(k), a(k), k] \leftarrow \infty$
```

```
                    continue
                end if
                $J[\boldsymbol{x}(k), a(k), k] \leftarrow \frac{1}{2}a(k)^2 + p(0|k) \cdot J_{DG}^*[\boldsymbol{x}(k), a(k), k] + [1 - p(0|k)] \cdot V[\boldsymbol{x}(k+1), k+1]$
            end for
            $V[\boldsymbol{x}(k), k] = \min\{J[\boldsymbol{x}(k), a(k), k]\} \forall a(k) \in U$
            $R[\boldsymbol{x}(k), k] = a(k)^* = \underset{a(k) \in U}{\operatorname{argmin}}\{J[\boldsymbol{x}(k), a(k), k]\}$, with $a(\mathrm{k})^*$
             the optimal control of point $[\boldsymbol{x}(k), k]$
        end for
    end for
```

In the above formulation, there is an implicit assumption that the red light is active when the vehicle reaches for the first time on the link, at time $t = 0$, like in the deterministic case. The approach can of course be extended to an instance where the traffic light is green at the initial time, and its time duration is uncertain, but probabilistic information is available regarding the switching time from green to red. The generalized problem can be developed in a similar way as in this chapter, while using a longer time horizon, which involves two periods of switching uncertainty: one starting from red and then turning into green, and an earlier one, reflecting the uncertain switching from green to red.

This algorithm (SDP) needs several minutes (as will be reported in the next chapter) for a standard personal computer (AMD Ryzen 5 2400G 3.60 GHz processor, 8.00 GB RAM memory) to execute and achieve the optimal trajectories for the given vehicle. That is the main reason why this solution cannot be obtained in real time on the vehicle side. Nevertheless, the solution that the SDP provides is comprehensive, because it feeds back optimal acceleration for all feasible positions and speeds of a given vehicle. So, the SDP algorithm is being used as a basis, and continuing by developing other methods and algorithms, that, combined, can provide with those optimal results, in an affordable amount of time.

4.4: Discrete Differential Dynamic Programming

Continuing from the above, the DDDP method introduced in chapter 3 is used, in order to create an algorithm that can provide the results of the SDP, but in a substantial less amount of time. The course of action that was followed is briefly presented in the following paragraphs.Also, this section contains the way that the DDDP method was actually implemented.

As stated before, for application of the discrete SDP algorithm for numerical solution of the problem, the state and control variables must be discretized. The level of discretization has a significant impact on computational time and memory requirements, but also on the accuracy of the computed solution. The trajectories (position and speed) for each optimal control are being calculated according to ***Eq. (4.18)***and***(4.19)***, with the state and control variable being bounded amid the feasible regions, described by ***Eq. (4.20)***and***(4.21)***. Also, the algorithm needs initial trajectories in order to operate, due to the fact that, it creates around these trajectories a corridor. These initial trajectories are taken from the deterministic GLOSA problem, by solving the pessimistic case of the problem, which assumes that the traffic signal will switch at the latest possible time.

Knowing that discretization in every iteration can be changed,the first thought was that after every iteration of the algorithm, the discretization should be decreased. This approach worked properly, but after consideration and tests, it was concluded that, it does not necessarilyneed to downgrade the discretization variable of what it previously was in every iteration, because it might be working, but it adds extra workload and computation time. Therefore, a constrain was added, in which, if the cost of the previous iteration is equal to the cost of the current iteration, then thediscretization for the next iteration is being reduced.

Also, considering the corridor Δ, there two ways that Δ can be implemented, a fixed corridor, and a dynamically changing corridor, and the dynamic approach was selected, and the reason why will be justified in Chapter 5. The user chooses a length of Δ, which corresponds to the allowed points, so the user indirectly chooses the number of the allowed points Δ_c.There is a trade-off between computational time, number of iterations and this variable and the length of corridor Δ.So,the user has the opportunity of choosing amidst more iterations within a small amount of time, or less iterations but within a larger amount of time, by using a bigger corridor. Having fixed discrete points is also a way of reducing computational time, since the unnecessary and quite large increase of those points after every iteration is being avoided, which gives us nothing else but taken computer memory space, with no actual use.

Continuing, the algorithm’s termination criterion needs also to be analysed properly. The algorithm should be working and keep doing the tasks that has to do in every iteration, until it will converge to an optimal solution, and then stop. The termination criterion used, ***Eq. (3.23)*** from [19], and it is the norm of the subtraction, of the control variables for the current iteration and the previous one,which should be smaller or equal to a variableε. As ε gets smaller values, the accuracy that corresponds in the fuel consumption is increased, but after a certain point, the better accuracy given is quite insignificant, since the reduction on the fuel consumption is minor.Regarding the trade-offofabout the optimal selection of corridor Δ, further analysis and investigation, with the proper justification and explanation will be present in the followingChapter 5, with the form of figures and tables.

Chapter 5: Results

In this section are being presented the investigated scenarios, examinedwith the use of the DDDP algorithm described in the above Chapters 3 and 4, along with the results of the algorithm, for the different scenarios tested.

Firstly, the following variables used in each scenario are being clarified.

- x_0: initial position of vehicle in meters,
- x_e: final position of vehicle in meters,
- v_0: initial speed of vehicle in m/s,
- v_e:target (final) speed of vehicle in m/s
- x_1: traffic signal position in meters.
- $[k_{min}, k_{max}]$: switching time range for the traffic light in seconds.
- $[x_{min}, x_{max}]$:position bounds in meters
- $[v_{min}, v_{max}]$:speed bounds in m/s
- $[u_{min}, u_{max}]$: control bounds in m/s^2

Scenario 1: $v_0 = 5\frac{m}{s}$, $v_e = 11\frac{m}{s}$, $x_0 = 0\ m$, $x_e = 220\ m$, $x_1 = 150\ m$, $w = 0.1$

Scenario 2: $v_0 = 11\frac{m}{s}$, $v_e = 11\frac{m}{s}$, $x_0 = 0\ m$, $x_e = 220\ m$, $x_1 = 150\ m$, $w = 0.1$

Scenario 3: $v_0 = 11\frac{m}{s}$, $v_e = 11\frac{m}{s}$, $x_0 = 75\ m$, $x_e = 220\ m$, $x_1 = 150\ m$, $w = 0.1$

5.1: Experimental results on known switching times

This section contains results from Scenario 1 and 2, in the case of known signal switching times, with the switching time of the traffic light t_1, where $t_1 = 18s$.In these two scenarios, the initial and final position, the traffic light's position and switching time are the same, but the initial speed of the vehicle is different.

Consequently, in Scenario 1 the vehicle's optimal trajectory, resulting from UP (see subsection 4.1.2), does not interfere with the red light constraint and the vehicle passes from the traffic light through the green light phase, so there is no need of adjusting the vehicle trajectories through the CP. However,in Scenario 2 the solution of UP violates the green light constraint, so the vehicle’s trajectory,derived from CP(see subsection 4.1.2),guides it to pass through the traffic light the exact moment that it turns from red to green.

Figure 6, a and b, demonstrates the vehicle’s optimal acceleration, speed and position trajectories, for both Scenario 1 and 2.As far as Scenario 1 is concerned, the initial speed is less than the final speed, as a result the vehicle accelerates in order to reach the final states. In this case, as the UP solution leads the vehicle to reach the traffic signal during the green phase, no re-adjustment of the trajectories, via solution of CP, is needed.On the other hand, inScenario 2, since the initial and final speed of the vehicle is the same, the only reasons for a change are either a possible case of wanting to reach the final position in a shorter amount of time, so the vehicle would increase and the decrease its speed, or another possible case of crossing the traffic signal during the red light phase, something that would demanded a decrease and then an increase in the vehicle’s speed.

It can be observed in Figure 6 that the solution UP behaves according to the first reason, but fails to satisfy the traffic light constraint.On the contrary, the CP seems to identify the second reason (decrease and the increase of the vehicle’s speed) and guide the vehicle to, first,decelerate with the purpose of passing the traffic light in the exact moment that it turns from red to green, and then accelerate in order to satisfy the final conditions. In both scenarios the finalposition and speed are both fulfilled at the final time t_e as they should, and also without violation of the traffic signal constraint.

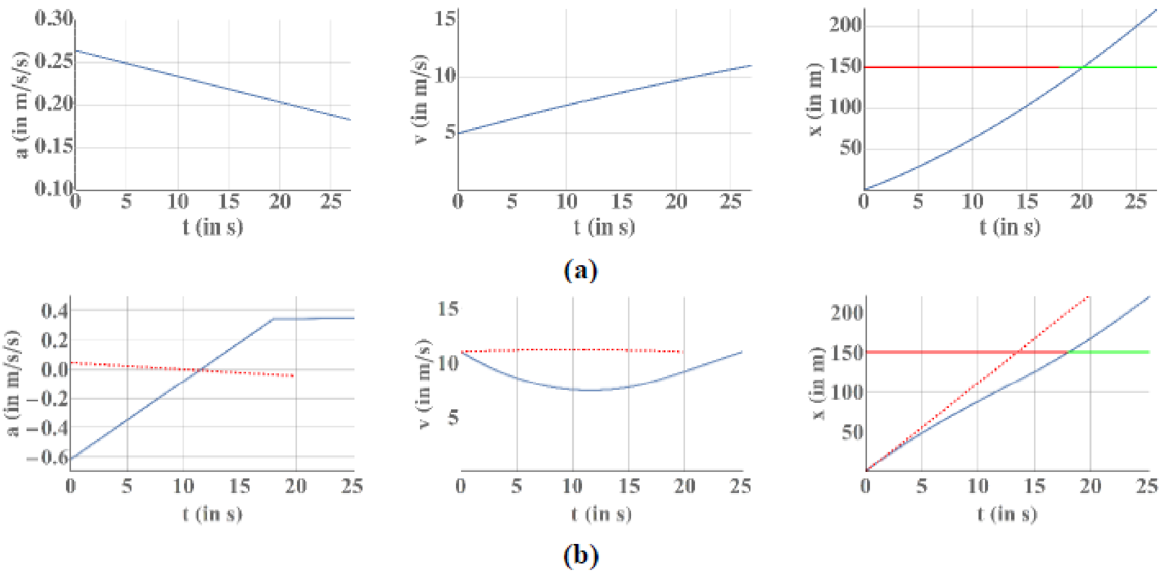


Figure 6: Optimal vehicle trajectories (blue solid lines) for Scenario 1 (a) and Scenario 2 (b), with the corresponding trajectories of the UP solution (red doted lines).Also, the straight red and green solid lines represent the red and green light phases[4].

5.2: Experimental results on unknown switching times

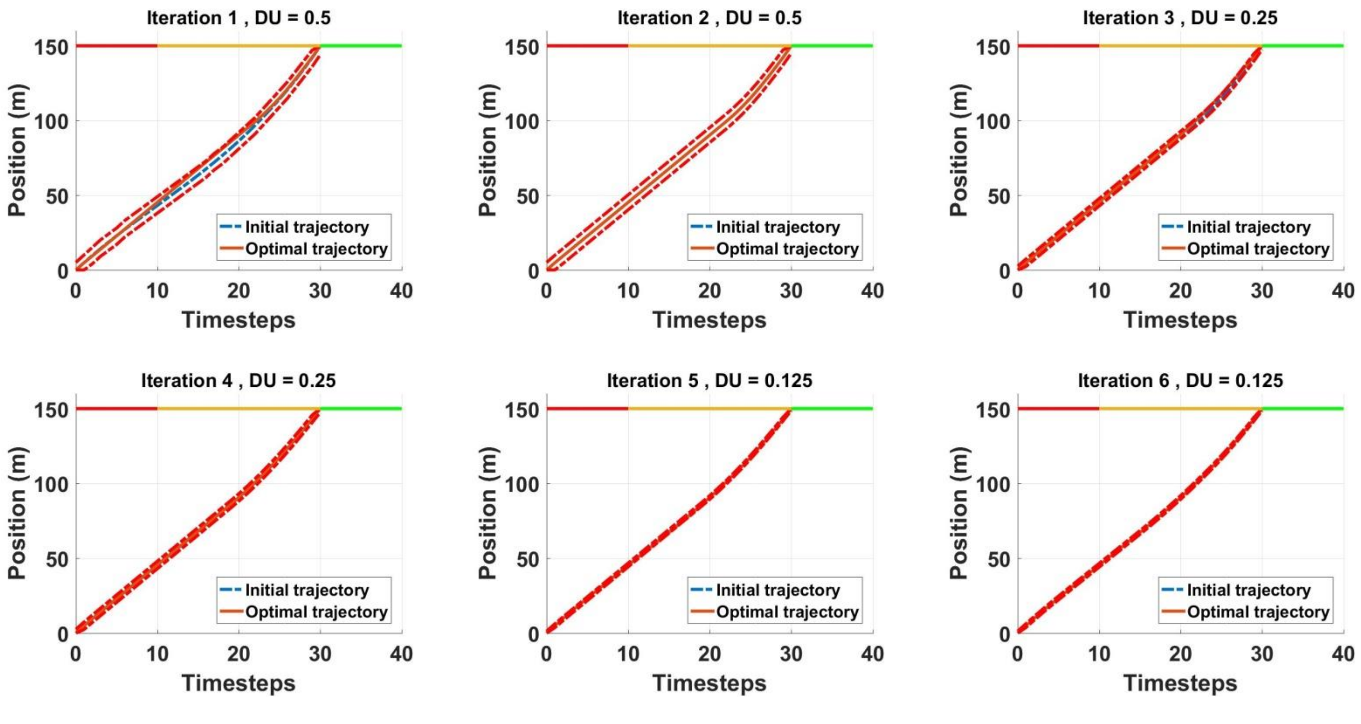
In this section, the results of the proposed DDDP approach, for all three Scenarios 1-3,are presented.In Scenario 1, the vehicle starts with a low initial velocity, which leads the vehicle to mostly accelerateuntil it reaches the final destination point, and, in the meantime, it has to pass through the traffic light at an optimal time and also minimize the cost criterion. In Scenario 2, the vehicle has the same objective, but the initial velocityis larger than the one of Scenario 1, which now leads the vehicle to, mostly, decelerate. Scenario 3has the same set up with Scenario 2 but now the vehicle starts from a position closer to the traffic light, in order to investigate the case where the vehicle is forcedto stop atthe red light, wait, and then accelerate after the traffic light turns green from red. The states and control bounds are set to $[x_{min}, x_{max}] = [0,150]$ m, $[v_{min}, v_{max}] = [0,16]$ m/s,$[a_{min}, a_{max}] = [-3, 3]$ m/s^2, respectively. The time step T is 1 s, which verifies the discretization properties mentioned in Chapter 4. The switching time range for the traffic light is $[k_{min}, k_{max}] = [10, 30]$ with uniform a-priori probability distribution.For the initial discretization, $\Delta a = \Delta v = 0.5$ is used, which leads to $\Delta x = 0.25$ and the initial corridor length is set to $\Delta = 4$.

Another thing worth mentioning is that, there are two dashed red lines in the figures representing the vehicle's velocity and position, among the lines that represent the initial and optimal trajectories, which constitute the corridor that was mentioned above. This is actually the corridor that the algorithm is forced to work under. So, the initial thought was that the algorithm should be working under a fixed corridor $[-\Delta, +\Delta]$. But soon it was determined that this approach increases the complexity and computational time in a not affordable scale, by making the algorithm searching for the optimal in a much bigger area. So, the searching radius was expanding vastly, but with nothing in return.

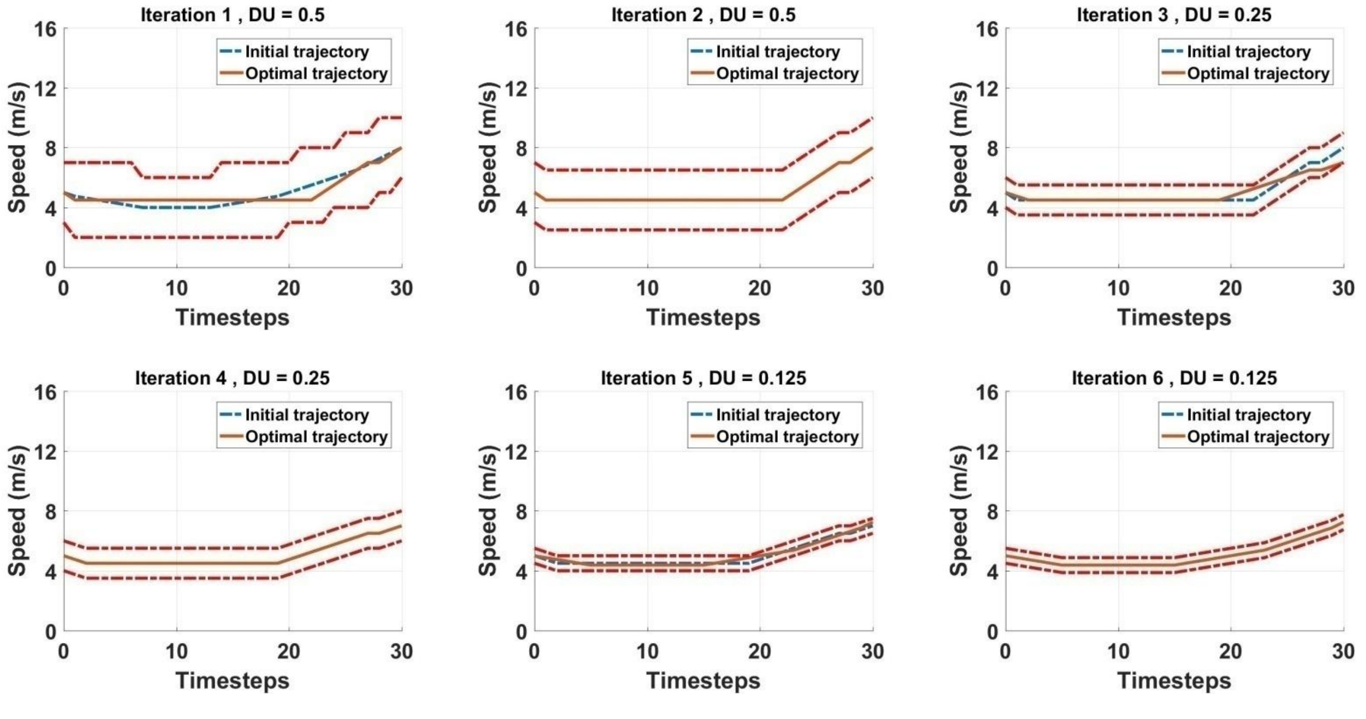
Consequently, the corridor is being implemented in a slightly different way, by multiplying the discretization variable ΔU with the allowed points, because with this approach, every time the discretization variable is being reduced, the corridor's length is being reduced as well. As far as speed is concerned, the corridor is $[-(\Delta * DU), +(\Delta * DU)]$, but for the position of the vehicle there is a difference, since the range of values for the position of the vehicle is $[0,150]$, and speed between $[0,16]$, so the interval $[-(\Delta * DU), +(\Delta * DU)]$ satisfies speed, but not the position, since a $(\Delta * DU)$ addition on the position's corridor is not equivalent to a similar addition to the speed's corridor for example. So, after some tests and consideration the corridor for the position of the vehicle was implemented as follows: $[-(\Delta * DU * e), +(\Delta * DU * e)]$, in order to have equivalent corridors for the vehicle's trajectories (the control variable does not have a corridor, since the bounds are fixed), where e is an integer, with values that the user sets, according to his/her preferences.

Figures 8-10 represent the evolution of the optimal state and control trajectories over each iteration of the DDDP for Scenarios 1-3. Specifically,in eachiteration we consider a corridor $\Delta^{(l)} = [-\Delta \cdot \Delta U, \Delta \cdot \Delta U]$ around the given initial state trajectories, which cannot of course extend out of the state bounds. That means that the problem's feasible region is reduced, which means that the DDDP solves, in each iteration, a reduced problem in terms of state space.Note that, the initial trajectory of the first iteration of the DDDP is the optimal solution of the deterministic GLOSA problem, assuming that the traffic light will switch from red to green at the latest possible time, that is, at $t_1 = k_{max}$, so as to be on the safe side.

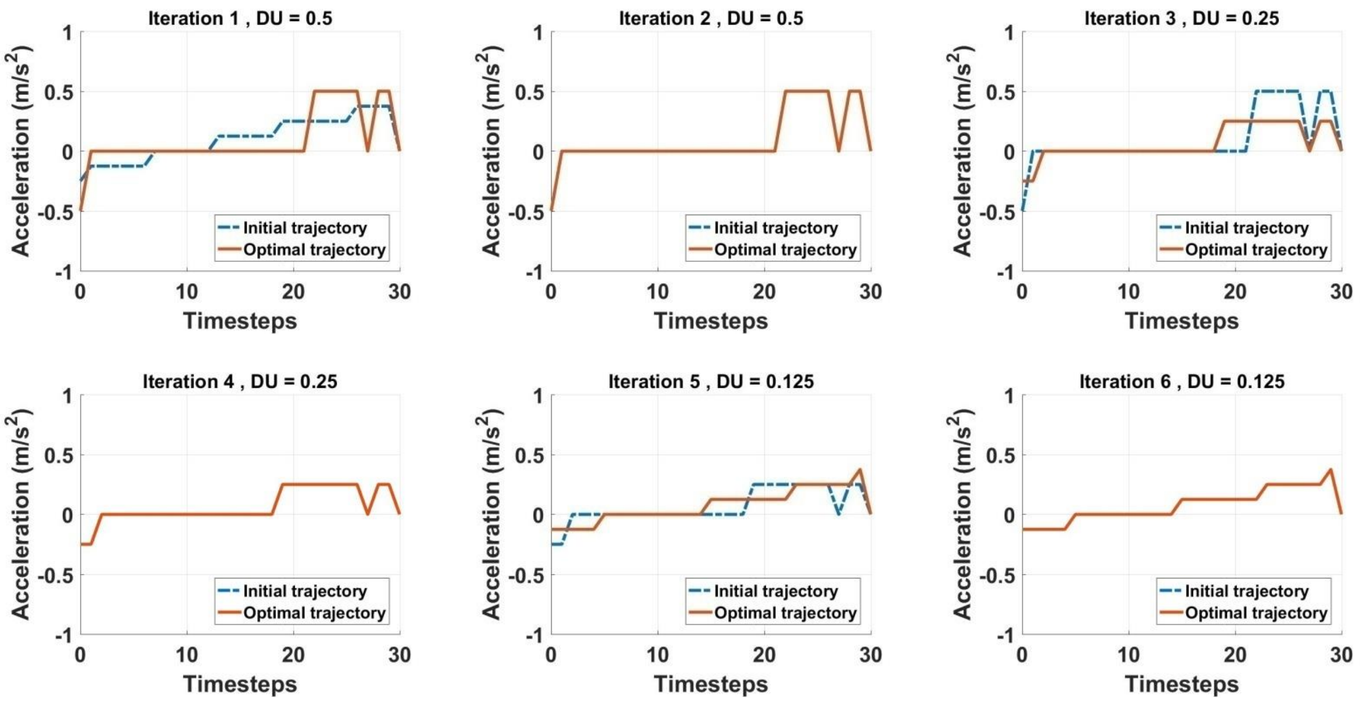
The dashed blue line represents the initial trajectory, the solid orange lines represents the optimal trajectories and the red dashed lines posing as the corridor.Also note that the traffic light phases are represented, in the position trajectories, as a straight line which is red for the red light phase, yellow for the stochastic switching period and green for the rest.It can be observed from **Figures 8 - 10**that, starting with the initial chosen discretization, in the first iterations the DDDP manages to improvethe initial trajectories, leading in better solutions, up to a point (i.e. Iteration 3 of Scenario 1) where no further improvement can be achieved. Reducing the discretization, a reduction in the corridors can be also observed, which is expected as its length depends on both $\Delta U$and the chosen length of corridor Δ. This reduction enables furtherimprovement of thetrajectories, again up to the pointwhereno better solution can be achieved and the procedure goes on until one ofthe terminal criterions is fulfilled.Following, the graphical representation of the vehicle's position, speed and acceleration for every iteration of the algorithm, considering Scenario 1:



(a)


Iteration 1 , DU = 0.5
Iteration 2 , DU = 0.5
Iteration 3 , DU = 0.25
Iteration 4 , DU = 0.25
Iteration 5 , DU = 0.125
Iteration 6 , DU = 0.125
Initial trajectory
Optimal trajectory
Speed (m/s)
Timesteps

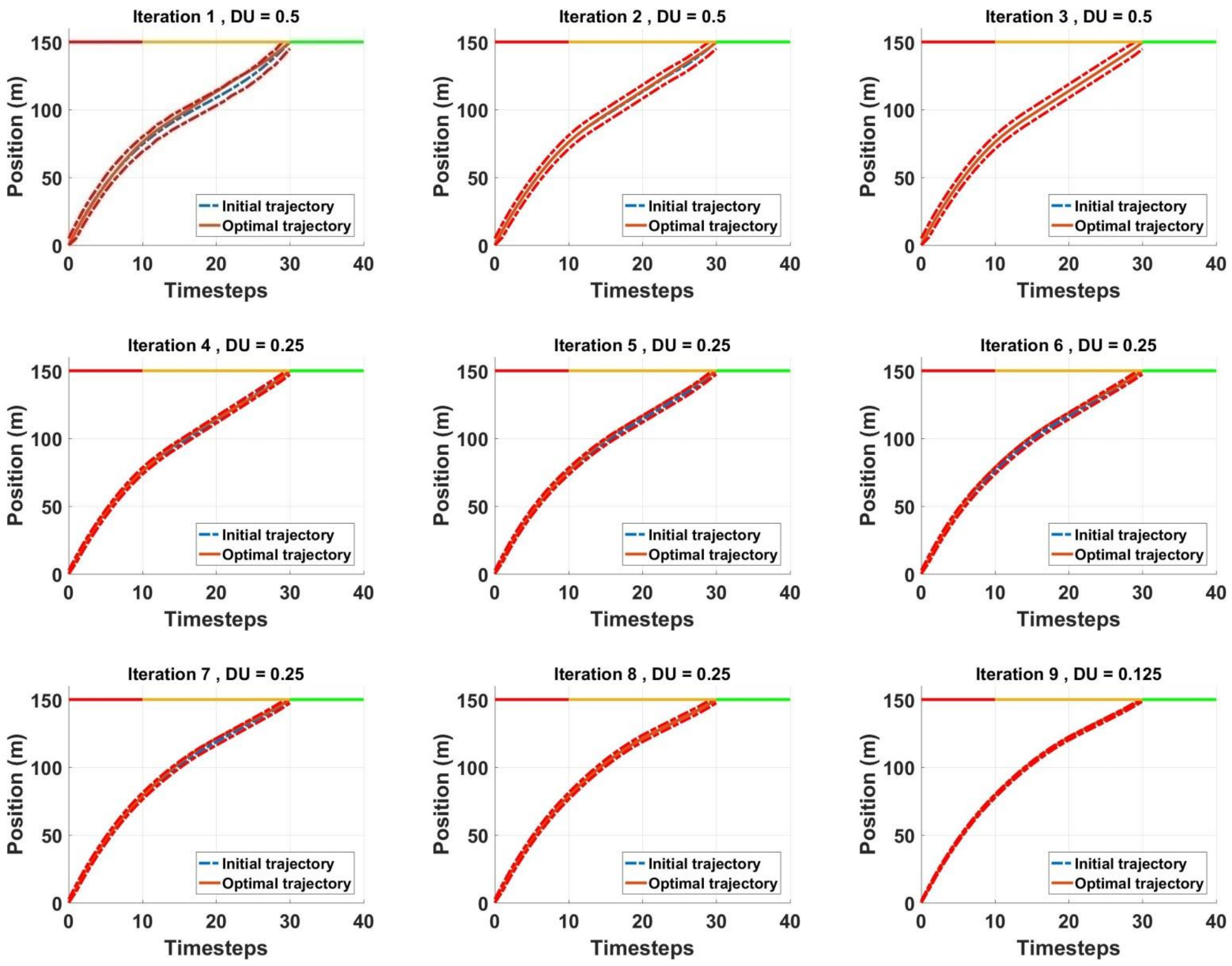
(b)

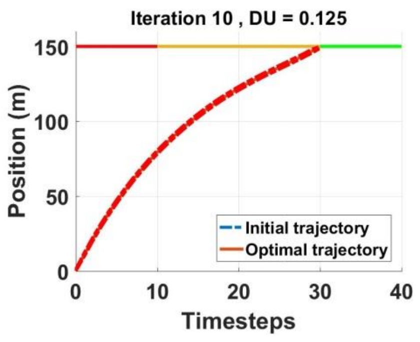

Iteration 1 , DU = 0.5
Iteration 2 , DU = 0.5
Iteration 3 , DU = 0.25
Iteration 4 , DU = 0.25
Iteration 5 , DU = 0.125
Iteration 6 , DU = 0.125
Initial trajectory
Optimal trajectory
Acceleration (m/s^2)
Timesteps

(c)

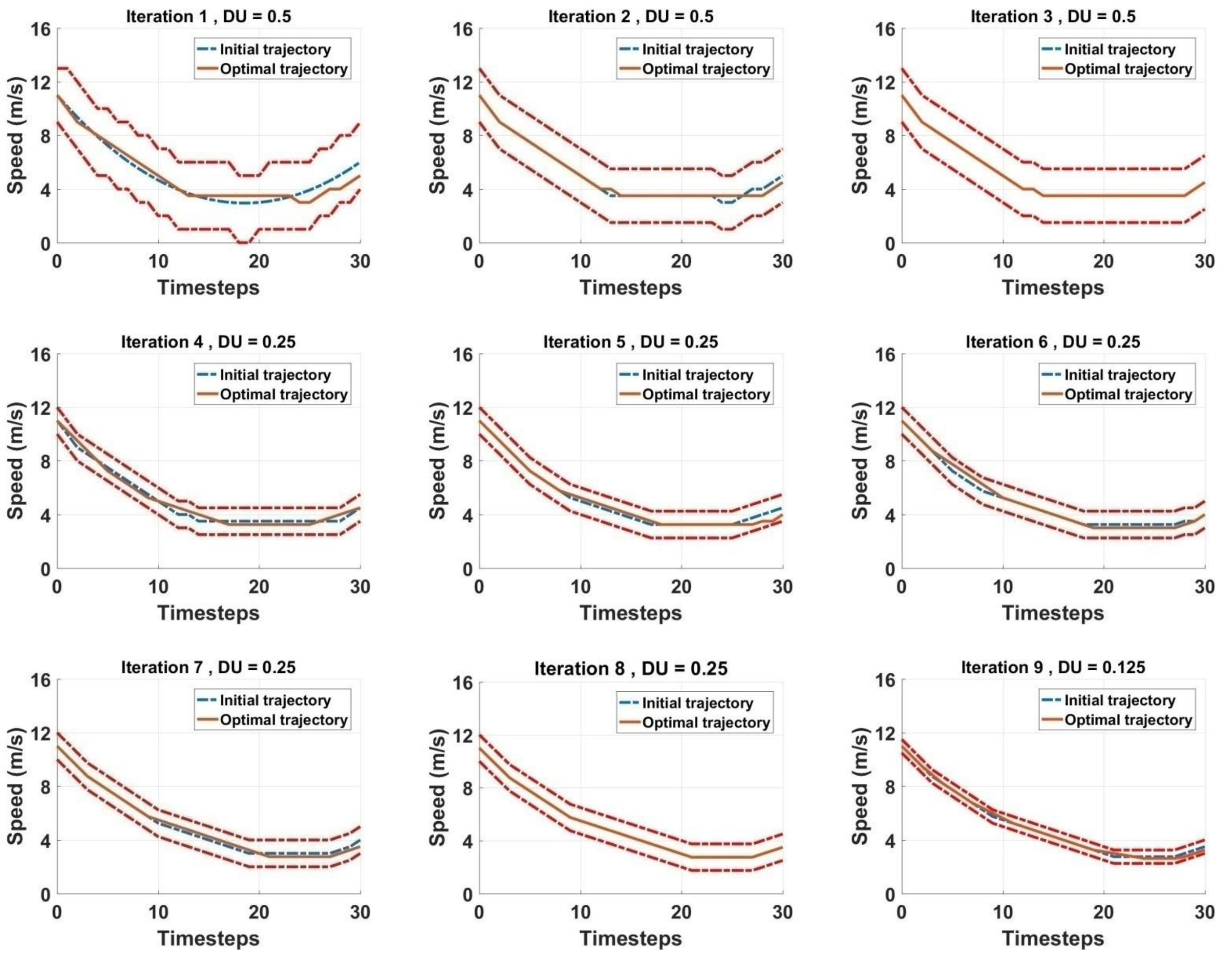
Figure 9: Initial and Optimal position (a), speed (b) and acceleration (c) of the vehicle, considering Scenario 1.

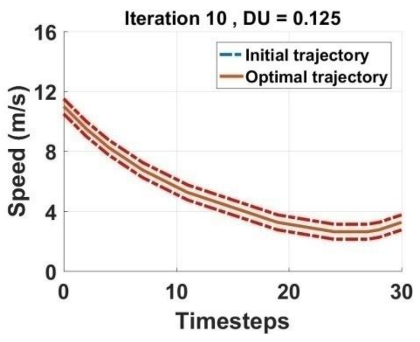
Continuing now to Scenario 2, in which the same as the above Scenario 1 apply for the shown figures, with a difference in the nature of the scenario itself, since the initial speed ($v_0 = 11\,m/s$) is the same as the targeted final speed ($v_e = 11\frac{m}{s}$). Following, the graphical representation of the vehicle's position, speed and acceleration for every iteration of the algorithm:



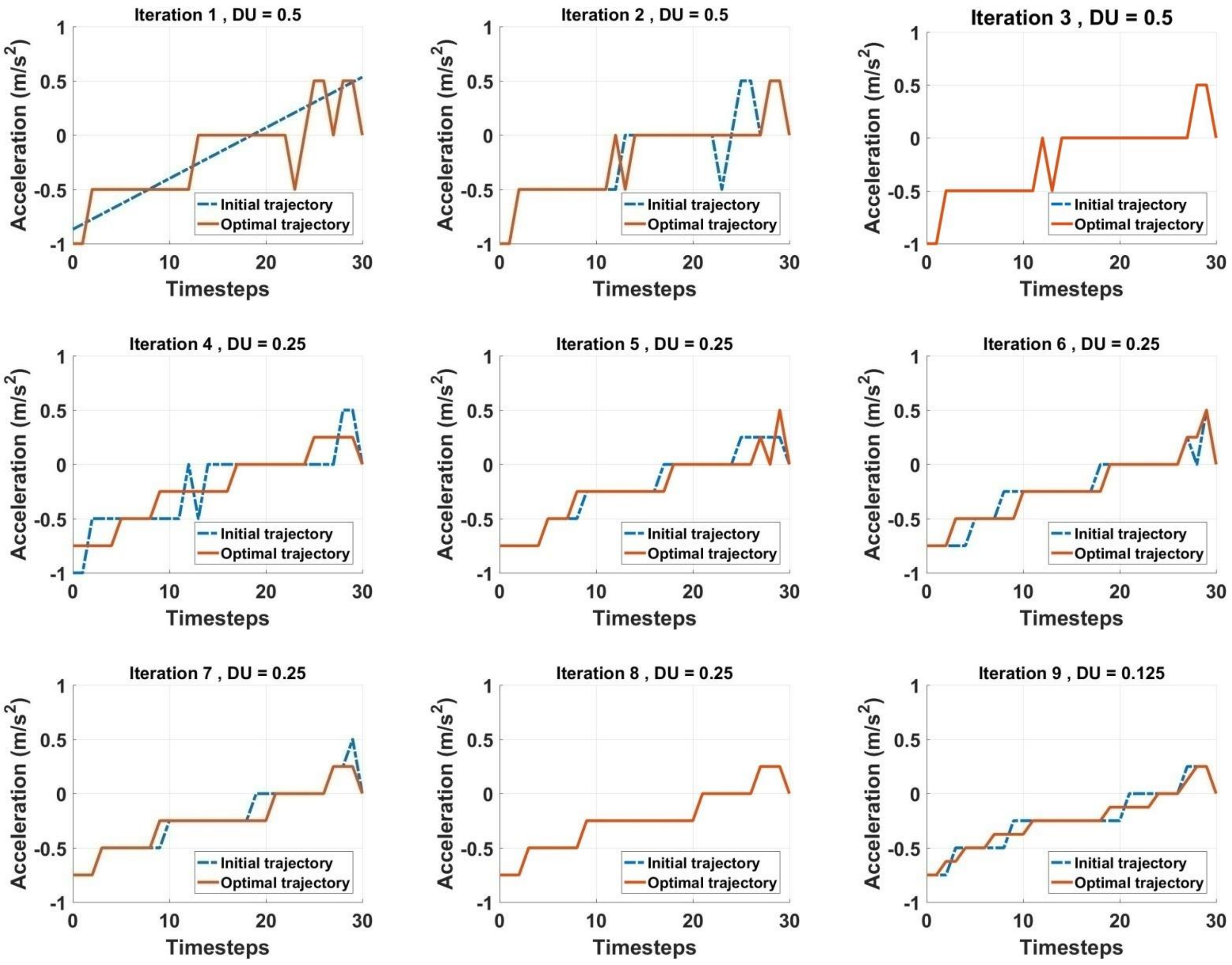

Iteration 10 , DU = 0.125
Position (m)
Timesteps
Initial trajectory
Optimal trajectory
0
50
100
150
0
10
20
30
40

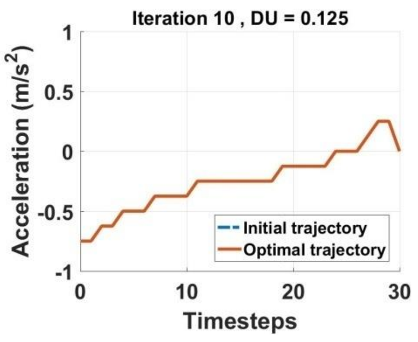
(a)


Iteration 1 , DU = 0.5
Iteration 2 , DU = 0.5
Iteration 3 , DU = 0.5
Iteration 4 , DU = 0.25
Iteration 5 , DU = 0.25
Iteration 6 , DU = 0.25
Iteration 7 , DU = 0.25
Iteration 8 , DU = 0.25
Iteration 9 , DU = 0.125
Speed (m/s)
Timesteps
Initial trajectory
Optimal trajectory
0
4
8
12
16
0
10
20
30


Iteration 10 , DU = 0.125
Initial trajectory
Optimal trajectory
Speed (m/s)
Timesteps

(b)

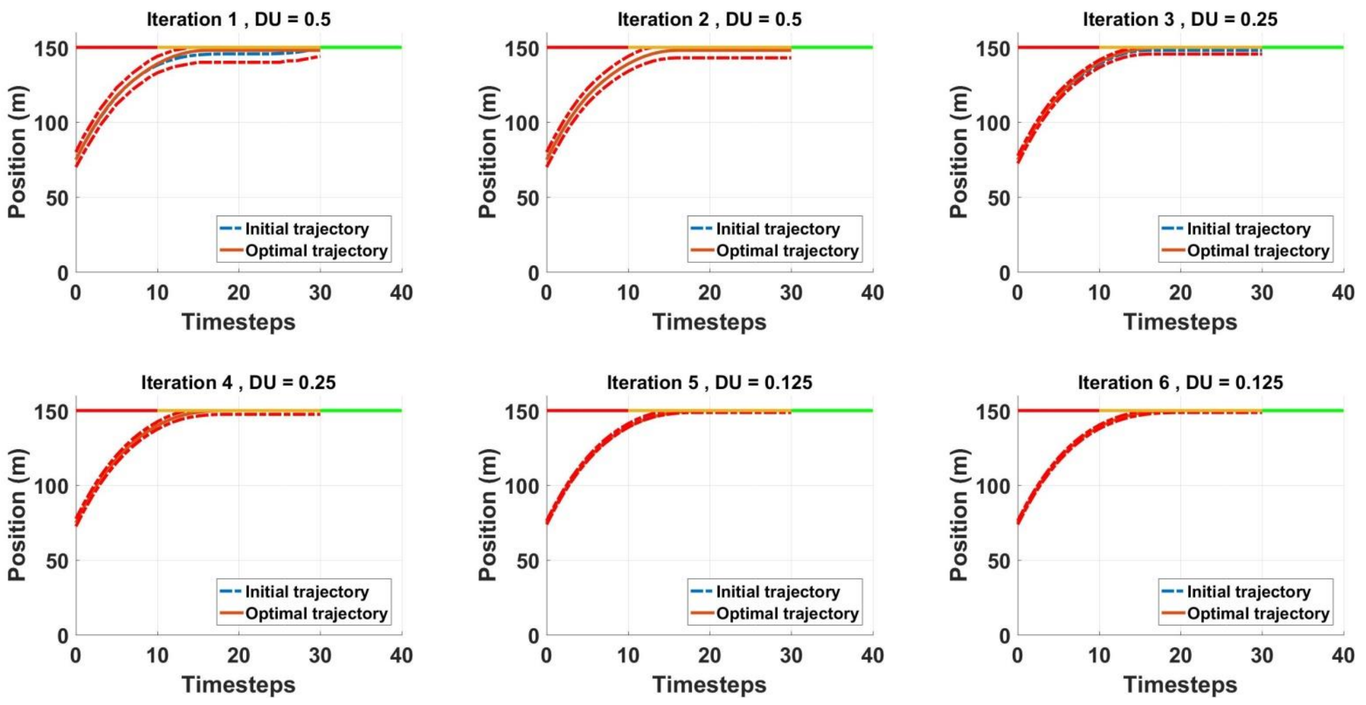

Iteration 1 , DU = 0.5
Iteration 2 , DU = 0.5
Iteration 3 , DU = 0.5
Iteration 4 , DU = 0.25
Iteration 5 , DU = 0.25
Iteration 6 , DU = 0.25
Iteration 7 , DU = 0.25
Iteration 8 , DU = 0.25
Iteration 9 , DU = 0.125
Initial trajectory
Optimal trajectory
Acceleration (m/s^2)
Timesteps



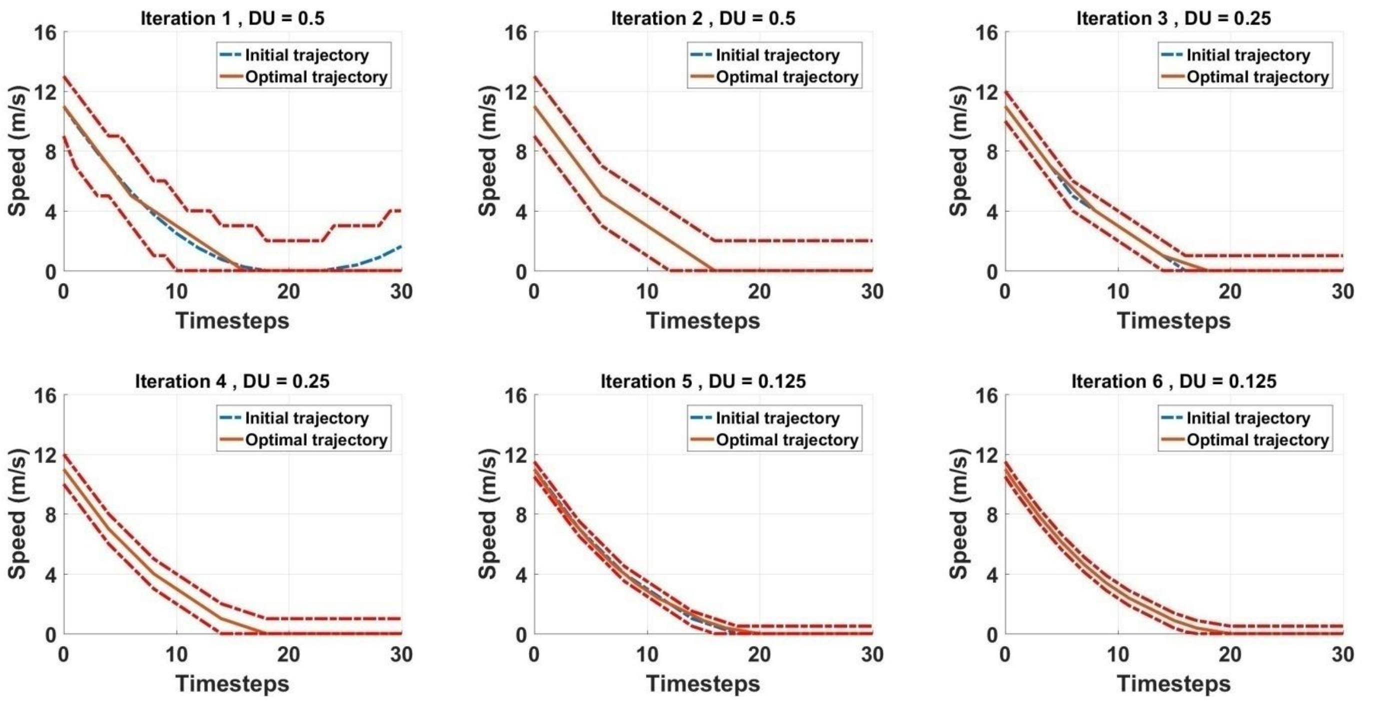
(c)

Figure 8: Initial and Optimal position (a), speed (b) and acceleration (c) of the vehicle, considering Scenario 2.

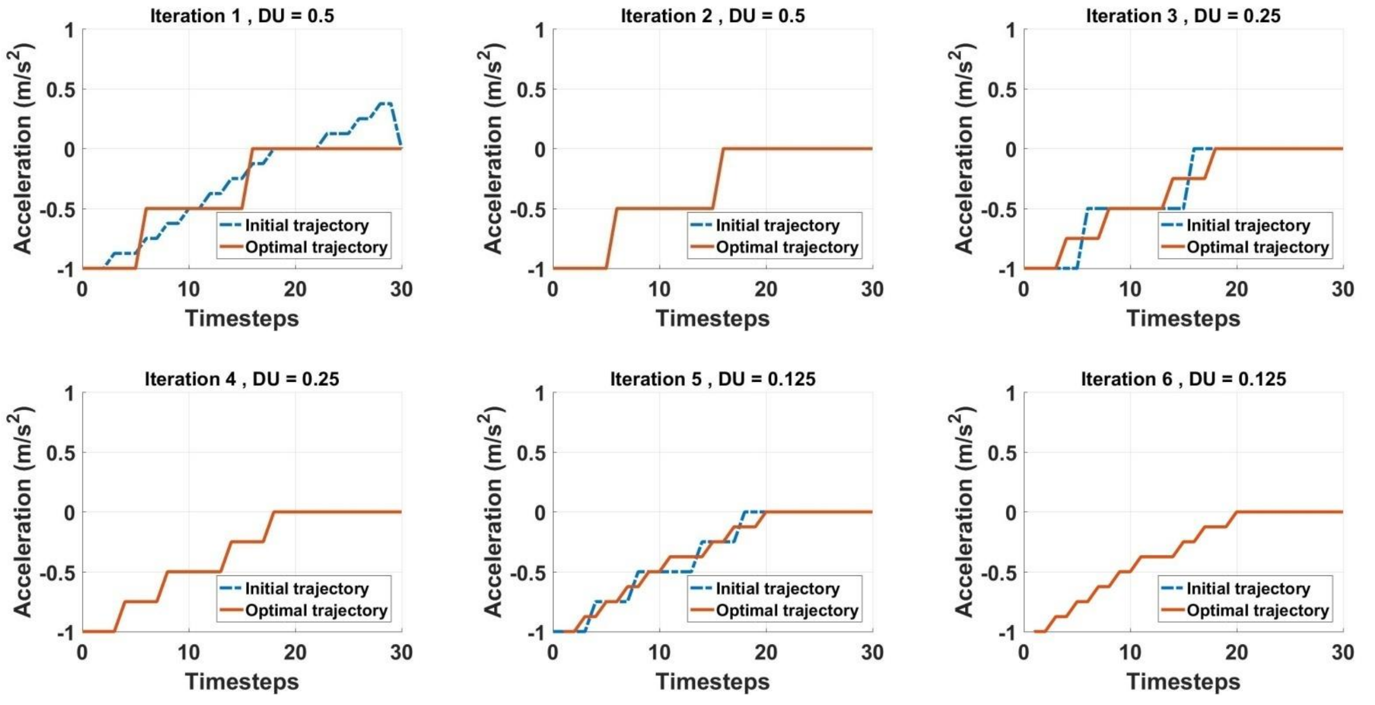
Coming now to Scenario 3, which is a different scenario in relation with the other two, because the vehicle initiates its course from a position quite closer to the traffic light, and so the vehicle one accelerates, there is no deceleration or stopping for the vehicle in this scenario. Also, in the figures considering the vehicle's speed, it can be observed that the lower bound of the corridor, as long as with some values of the trajectory itself are identical with the x-axis, which is actually value 0, since zero is the lower bound of the vehicle's speed.



(a)



(b)



(c)

Figure 10: Initial and Optimal position (a), speed (b) and acceleration (c) of the vehicle, considering Scenario 3.

5.3: Comparison of the discretization variable ΔU in relation to cost and computational time, considering the DDDP method

In this section, the resulting cost and the overall computing time of the DDDP algorithm are being presented, in relation to the discretization variableΔUused in every iteration of each algorithm,along with the length of the corridor for the vehicle's trajectories,which constitutes an important way of comparing the algorithm, in order to locate its strengths and weaknesses.

As far as the DDDP method is considered, different values of the corridor's lengthΔ, which eventually correspond in the allowed points (Δ_c) are being tested, in order to properly compare, present and evaluate the trade-off between cost and computational time, and the «golden section» that there is and should be followed as well.

Taking now into consideration the discretizationvariable ΔU, initially is being set equal to 0.5, and as the algorithm progresses, ΔU decreases, until it reaches the value 0.125, or beforethat, if***Eq. (3.23)*** is satisfied. The choice of the discretization variable can be explained by the results of the **Table 1**, **Table 3** and **Table 5**.It can be observed that, as expected, as the discretization interval is reduced, the obtained optimal cost is reducing as well.

5.3.1 Scenario 1 considering the DDDP method

The table below presents the differences between CPU-time, number of iterations and cost for different DU values:

		$\Delta U = 1.0$			$\Delta U = 0.5$	
Δ	**Iterations**	**CPU-time (in s.)**	**Cost**	**Iterations**	**CPU-time (in s.)**	**Cost**
2	19	1.6782	1.19993	20	1.9529	1.19331
3	19	3.6527	1.17698	8	1.6026	1.17517
4	15	4.4220	1.17517	6	1.8073	1.17517
5	12	5.3904	1.17517	6	2.7874	1.17517
6	10	6.1789	1.17517	6	3.8851	1.17517
7	10	8.1823	1.17517	6	5.2753	1.17517
8	8	7.7466	1.17517	6	6.2245	1.17517
9	8	9.2272	1.17517	8	10.2638	1.17517

		$\Delta U = 0.25$			$\Delta U = 0.125$	
Δ	**Iterations**	**CPU-time (in s.)**	**Cost**	**Iterations**	**CPU-time (in s.)**	**Cost**

2	10	1.0061	1.19331	5	0.8835	1.19331
3	6	1.2317	1.17907	5	1.13924	1.17907
4	7	2.2774	1.17517	6	2.1347	1.17517
5	6	2.9560	1.17517	5	2.6782	1.17517
6	5	3.2154	1.17517	4	2.8837	1.17517
7	4	3.8502	1.17517	3	2.9937	1.17517
8	4	4.3872	1.17517	3	3.7152	1.17517
9	4	5.6701	1.17517	3	5.2839	1.17517

Table 3: Number of iterations and CPU-time consumed, for every value of Δ, considering Scenario 1,for the DDDP algorithm,for different values of DU.

The following tables represent for different values of the corridor's length (Δ), the cost and CPU-time for every iteration of the algorithm, considering Scenario 1.

	Δ=2			**Δ=3**			**Δ=4**	
Iteration	**Cost**	**CPU-time (in s.)**	**Iteration**	**Cost**	**CPU-time (in s.)**	**Iteration**	**Cost**	**CPU-time (in s.)**
1	1.54512	0.1059	1	1.53603	0.1960	1	1.35775	0.3019
2	1.53625	0.0677	2	1.35775	0.1770	2	1.35775	0.3198
3	1.53446	0.0671	3	1.35775	0.1487	3	1.22327	0.2936
4	1.53446	0.1000	4	1.23237	0.1873	4	1.22327	0.2443
5	1.35933	0.1052	5	1.22327	0.1469	5	1.17517	0.2970
6	1.31703	0.1114	6	1.22327	0.1582	6	1.17517	0.2972
7	1.30435	0.1047	7	1.17517	0.1694			
8	1.28450	0.0950	8	1.17517	0.1652			
9	1.28143	0.0761						
10	1.27362	0.0812						
11	1.26617	0.0906						
12	1.26617	0.0871						
13	1.22864	0.0916						
14	1.22864	0.0962						
15	1.21013	0.0847						
16	1.20945	0.0820						
17	1.19898	0.0982						
18	1.19339	0.0983						
19	1.19331	0.0743						
20	1.19331	0.0844						

(a)

	Δ=5			**Δ=6**			**Δ=7**	
Iteration	**Cost**	**CPU-time (in**	**Iteration**	**Cost**	**CPU-time (in**	**Iteration**	**Cost**	**CPU-time (in**

		s.)			**s.)**			**s.)**
1	1.35775	0.5514	1	1.35775	0.6466	1	1.35775	0.9426
2	1.35775	0.4883	2	1.35775	0.5310	2	1.35775	0.8020
3	1.22327	0.3988	3	1.22327	0.5636	3	1.22327	0.7956
4	1.22327	0.4312	4	1.22327	0.5653	4	1.22327	0.7632
5	1.17517	0.4658	5	1.17517	0.5966	5	1.17517	0.9038
6	1.17517	0.4471	6	1.17517	0.5613	6	1.17517	0.8712

(b)

	Δ=8			**Δ=9**	
Iteration	**Cost**	**CPU-time (in s.)**	**Iteration**	**Cost**	**CPU-time (in s.)**
1	1.35775	1.1894	1	1.35775	1.3973
2	1.35775	0.9249	2	1.27815	1.2454
3	1.22327	1.0600	3	1.27815	1.1488
4	1.22327	0.9523	4	1.22716	1.1881
5	1.17517	1.0219	5	1.22327	1.2635
6	1.17517	1.0247	6	1.22327	1.2105
			7	1.17517	1.3151
			8	1.17517	1.3633

(c)

Table 4: Number of iterations, cost and CPU-time per iteration, considering Scenario 1, for the DDDP algorithm, for different values of $\varDelta$. Table 4 is constituted by (a), (b) and (c).

5.3.2 Scenario 2 considering the DDDP method

The table below presents the differences between CPU-time, number of iterations and cost for different DU values:

		$\boldsymbol{\varDelta U = 1.0}$			$\boldsymbol{\varDelta U = 0.5}$	
$\boldsymbol{\varDelta}$	**Iterations**	**CPU-time (in s.)**	**Cost**	**Iterations**	**CPU-time (in s.)**	**Cost**
2	30	2.8955	3.90985	18	1.6486	3.90985
3	22	3.9694	3.90683	14	2.5773	3.90683
4	14	4.0992	3.90683	10	3.0272	3.90683
5	11	4.6326	3.90683	7	3.1107	3.90683
6	14	7.8569	3.90683	10	5.8131	3.90683
7	13	9.5328	3.90683	9	6.8251	3.90683
8	11	9.8620	3.90683	8	7.8471	3.90683

9	11	11.9533	3.90683	8	10.0214	3.90683

	$\boldsymbol{\Delta U = 0.25}$			$\boldsymbol{\Delta U = 0.125}$		
Δ	**Iterations**	**CPU-time (in s.)**	**Cost**	**Iterations**	**CPU-time (in s.)**	**Cost**
2	16	1.6298	3.90985	21	2.3273	3.90985
3	12	2.3428	3.90683	18	3.7644	3.90683
4	8	3.4996	3.90683	13	4.4034	3.90683
5	8	3.7985	3.90683	11	5.3552	3.90683
6	7	4.3817	3.90683	9	5.8370	3.90683
7	6	4.9782	3.90683	7	7.5078	3.90683
8	6	6.7829	3.90683	7	8.4172	3.90683
9	6	8.2920	3.90683	6	9.2536	3.90683

Table 1: Number of iterations and CPU-time consumed, for every value of Δ, considering Scenario 2, for the DDDP algorithm, for different values of ΔU.

The following tables represent for a number of different number of allowed discrete points (Δ), the cost and CPU-time for every iteration of the algorithm, considering Scenario 2.

	Δ=2			**Δ=3**			**Δ=4**	
Iteration	**Cost**	**CPU-time(ins.)**	**Iteration**	**Cost**	**CPU-time(in s.)**	**Iteration**	**Cost**	**CPU-time(in s.)**
1	4.33065	0.1134	1	4.32185	0.2067	1	4.30026	0.3122
2	4.31389	0.0740	2	4.23127	0.1614	2	4.23127	0.3215
3	4.23179	0.0872	3	4.23127	0.1626	3	4.23127	0.2753
4	4.23127	0.0833	4	4.01207	0.1772	4	4.01207	0.2759
5	4.23127	0.0865	5	3.99559	0.1625	5	3.98839	0.2486
6	4.03823	0.0849	6	3.98839	0.1490	6	3.95883	0.2782
7	4.01207	0.0729	7	3.96973	0.2034	7	3.94089	0.2866
8	3.99559	0.0694	8	3.95883	0.1627	8	3.94089	0.2686
9	3.98839	0.1020	9	3.95795	0.1552	9	3.90683	0.2604
10	3.96973	0.0801	10	3.94089	0.1609	10	3.90683	0.2947
11	3.96973	0.0908	11	3.94089	0.1635			
12	3.93029	0.0851	12	3.90742	0.2159			
13	3.92264	0.0837	13	3.90683	0.1721			
14	3.91749	0.0853	14	3.90683	0.1773			
15	3.91420	0.0834						
16	3.91324	0.0733						
17	3.90985	0.0895						

18	3.90985	0.0960

(a)

	Δ=5			**Δ=6**			**Δ=7**	
Iteration	**Cost**	**CPU-time (in s.)**	**Iteration**	**Cost**	**CPU-time (in s.)**	**Iteration**	**Cost**	**CPU-time (in s.)**
1	4.23127	0.5551	1	4.23127	0.7581	1	4.19245	0.8663
2	4.18934	0.3856	2	4.18934	0.5241	2	4.17807	0.6983
3	4.18934	0.3940	3	4.17807	0.5402	3	4.17807	0.7084
4	3.94089	0.4086	4	4.17807	0.5193	4	3.96989	0.7158
5	3.94089	0.4035	5	3.98414	0.5387	5	3.95101	0.7663
6	3.90683	0.3934	6	3.95176	0.5552	6	3.94089	0.7286
7	3.90683	0.3917	7	3.94089	0.5743	7	3.94089	0.7319
			8	3.94089	0.6760	8	3.90683	0.8348
			9	3.90683	0.6171	9	3.90683	0.7827
			10	3.90683	0.5571			

(b)

	Δ=8			**Δ=9**	
Iteration	**Cost**	**CPU-time(in s.)**	**Iteration**	**Cost**	**CPU-time(in s.)**
1	4.18934	1.0430	1	4.18934	1.2102
2	4.17807	0.9024	2	4.17807	1.0632
3	4.17807	0.8494	3	4.17807	1.1143
4	3.95717	0.9979	4	3.95717	1.2115
5	3.94089	0.9719	5	3.94089	1.1760
6	3.94089	0.9531	6	3.94089	1.2845
7	3.90683	1.0161	7	3.90683	1.2803
8	3.90683	0.9751	8	3.90683	1.2844

(c)

Table 2: Number of iterations, cost and CPU-time per iteration, considering Scenario 2, for the DDDP algorithm, for different values of Δ.Table 2 is constituted by (a), (b) and (c).

5.3.3 Scenario 3 considering the DDDP method

The table below presents the differences between CPU-time, number of iterations and cost for different DU values:

	$\boldsymbol{\Delta U = 1.0}$			$\boldsymbol{\Delta U = 0.5}$		
$\boldsymbol{\Delta}$	**Iterations**	**CPU-time (in s.)**	**Cost**	**Iterations**	**CPU-time (in s.)**	**Cost**
2	13	1.0205	7.91072	9	0.5784	7.91072
3	13	1.6152	7.91072	8	0.8797	7.91072
4	9	1.5052	7.91072	6	1.0955	7.91072
5	9	2.0637	7.91072	6	1.4627	7.91072
6	9	2.6626	7.91072	6	1.9729	7.91072
7	8	3.1769	7.91072	6	2.5854	7.91072
8	8	3.6823	7.91072	6	3.2006	7.91072
9	8	4.5075	7.91072	6	3.6189	7.91072

	$\boldsymbol{\Delta U = 0.25}$			$\boldsymbol{\Delta U = 0.125}$		
$\boldsymbol{\Delta}$	**Iterations**	**CPU-time (in s.)**	**Cost**	**Iterations**	**CPU-time (in s.)**	**Cost**
2	8	0.6222	7.91072	9	1.5783	7.91072
3	7	0.9870	7.91072	6	1.5243	7.91072
4	5	1.0847	7.91072	5	1.3862	7.91072
5	5	1.4656	7.91072	4	1.5857	7.91072
6	4	1.5759	7.91072	4	1.9679	7.91072
7	4	1.9250	7.91072	4	2.5323	7.91072
8	4	2.3415	7.91072	3	2.3917	7.91072
9	4	3.1505	7.91072	3	3.2373	7.91072

Table 5: Number of iterations and CPU-time consumed, for every value of Δ, considering Scenario 3, for the DDDP algorithm, for different values of ΔU.

The following tables represent for a number of different values of Δ, the cost and CPU-time for every iteration of the algorithm, considering Scenario 3.

	Δ=2			**Δ=3**			**Δ=4**	
Iteration	**Cost**	**CPU-time (in s.)**	**Iteration**	**Cost**	**CPU-time (in s.)**	**Iteration**	**Cost**	**CPU-time (in s.)**
1	8.09149	0.0751	1	8.09149	0.1315	1	8.09149	0.1874
2	8.09149	0.0469	2	8.09149	0.0915	2	8.09149	0.1286
3	7.97288	0.0509	3	7.95554	0.1042	3	7.95198	0.1917
4	7.95198	0.0496	4	7.95198	0.0945	4	7.95198	0.1563
5	7.95198	0.0475	5	7.95198	0.0858	5	7.91072	0.1603
6	7.92050	0.0424	6	7.91526	0.0952	6	7.91072	0.1524
7	7.91526	0.0438	7	7.91072	0.1051			
8	7.91072	0.0473	8	7.91072	0.1057			
9	7.91072	0.0572						

(a)

Δ=5			Δ=6			Δ=7		
Iteration	**Cost**	**CPU-time (in s.)**	**Iteration**	**Cost**	**CPU-time (in s.)**	**Iteration**	**Cost**	**CPU-time (in s.)**
1	8.09149	0.2420	1	8.09149	0.3073	1	8.09149	0.3905
2	8.09149	0.2247	2	8.09149	0.2683	2	8.09149	0.3497
3	7.95198	0.2409	3	7.95198	0.3216	3	7.95198	0.4618
4	7.95198	0.2387	4	7.95198	0.2490	4	7.95198	0.3372
5	7.91072	0.2184	5	7.91072	0.3212	5	7.91072	0.4309
6	7.91072	0.2721	6	7.91072	0.3412	6	7.91072	0.4348

(b)

Δ=8			Δ=9		
Iteration	**Cost**	**CPU-time(in s.)**	**Iteration**	**Cost**	**CPU-time(in s.)**
1	8.09149	0.5114	1	8.09149	0.5665
2	8.09149	0.4788	2	8.09149	0.4773
3	7.95198	0.5085	3	7.95198	0.6212
4	7.95198	0.4377	4	7.95198	0.5722
5	7.91072	0.5191	5	7.91072	0.6041
6	7.91072	0.5118	6	7.91072	0.6346

(c)

Table 6: Number of iterations, cost and CPU-time per iteration, considering Scenario 3, for the DDDP algorithm, for different values of Δ. Table 6 is constituted by (a), (b) and (c).

Considering and studying the tables above, for every one of the scenarios investigated, it can be observed that for small values of Δ (e.g. $\Delta = 2$, or $\Delta = 3$), the cost does not converge fully. Therefore, the choice of values of $\Delta$is avoided, even if the CPU-time is significantly low. As the values of Δincreases though, the cost seems to converging properly, so the choice of a price for Δbetween 4 or 5 allowed discrete points is the reasonable thing to do. Taking now into consideration the fact that the computational time increases as the allowed points increase as well, so the choice of Δ_c between $5 - 9$ isn't preferred, due to the CPU-time increase. So, the choice of $\Delta = 4$ is considered, as the best one, considering the above mentioned trade-off.

It can also be concluded from **Table 2**, **Table 4** and **Table 6** that, the computational time does not seem to be increasing from iteration to iteration, for any given values of Δ, due to the fact that the discretization variable decreases, so the corresponding (to corridor Δ) allowed pointsΔ_c become more after every decrease of variableΔU, but in the same time the corridor is reduced as well, so there is a balance that does not allow the increase of the CPU-time after every iteration.

5.4: Investigating the use and the impact of thecorridor $\boldsymbol{\Delta^{(l)}(k)}$

As it was mentioned in section 4.4, that the corridor $\Delta^{(l)}(k)$(or Δ), is being implemented in a slightly different waythan the initial implementation (which was a fixed interval $[-\Delta, +\Delta]$), by multiplying the discretization variable ΔU with the length of the corridor. With this approach, every time the discretization variable is being reduced, the corridor is being reduced as well, making it particularly smaller, so the computational time is significantly diminished, since the algorithm has a limited searching area, in which the optimal is included, but the redundant points are not part of that area.

In this section, the usefulness and the necessity of the dynamically implemented corridor is going to be observed, through an example, in which two different cases are being investigated.Considering for Scenario 2, the corridor's length ($\Delta = 4$) and the same discretization variable ($\Delta U = 0.5$), the problem will be solved two times, one with a fixed corridor, and the other one with the corridor being able to be reduced accordingly and analogously with the discretization variable ΔU. The following table presents those two cases:

Iteration	$\boldsymbol{\Delta U}$	**Cost**	**CPU-time (in s.)**	**Total CPU-time(in s.)**
1	0.5	4.17807	1.8428	73.6321
2	0.5	4.17807	1.5150	
3	0.25	3.94089	6.0468	
4	0.25	3.94089	6.8988	
5	0.125	3.90683	26.1620	
6	0.125	3.90683	25.8061	

Table 7: Fixed corridor and $\Delta = 4$.

Iteration	$\boldsymbol{\Delta U}$	**Cost**	**CPU-time (in s.)**	**Total CPU-time(in s.)**
1	0.5	4.30026	0.3166	2.9176
2	0.5	4.23127	0.3267	
3	0.5	4.23127	0.2592	

4	0.25	4.01207	0.2787
5	0.25	3.98839	0.2632
6	0.25	3.95883	0.2502
7	0.25	3.94089	0.2809
8	0.25	3.94089	0.2715
9	0.125	3.90683	0.2593
10	0.125	3.90683	0.2582

Table 8:Dynamically changing corridor and $\Delta = 4$.

It can be observed from **Table 7** and **Table 8**that the existence of the dynamically changing corridor is a necessity, since the differences considering the computational time are quite remarkable, something that can be observed from the CPU-times per iteration, but also from the total times. In **Table 7**, the total CPU-time is equal to 73.6 seconds, almost one and a half minute, and on the contrary, in **Table 8**, the total CPU-time drops to 2.9 seconds, due to the fact that the corridor changes and becomes smaller every time the discretization variable is reduced.

5.5: Comparison of the DDDP algorithm with the SDP algorithm

In this section, the DDDP algorithm that was implemented as a part of this thesis, is being compared with the Stochastic Dynamic Programming algorithm developed in[4], considering the CPU-times of each one, and also the optimal trajectories that each algorithm found. Continuing, for every scenario investigated, **Figure 11**, **Figure 12** and **Figure 13**present the optimal trajectories of each algorithm, and also,**Table 9**, **Table 10** and **Table 11** present the CPU-time that each algorithm needs in order to reach the same cost value.

Below, the following figures present the optimal derived trajectories and the control variable, considering Scenario 1, for each one of the two algorithms.

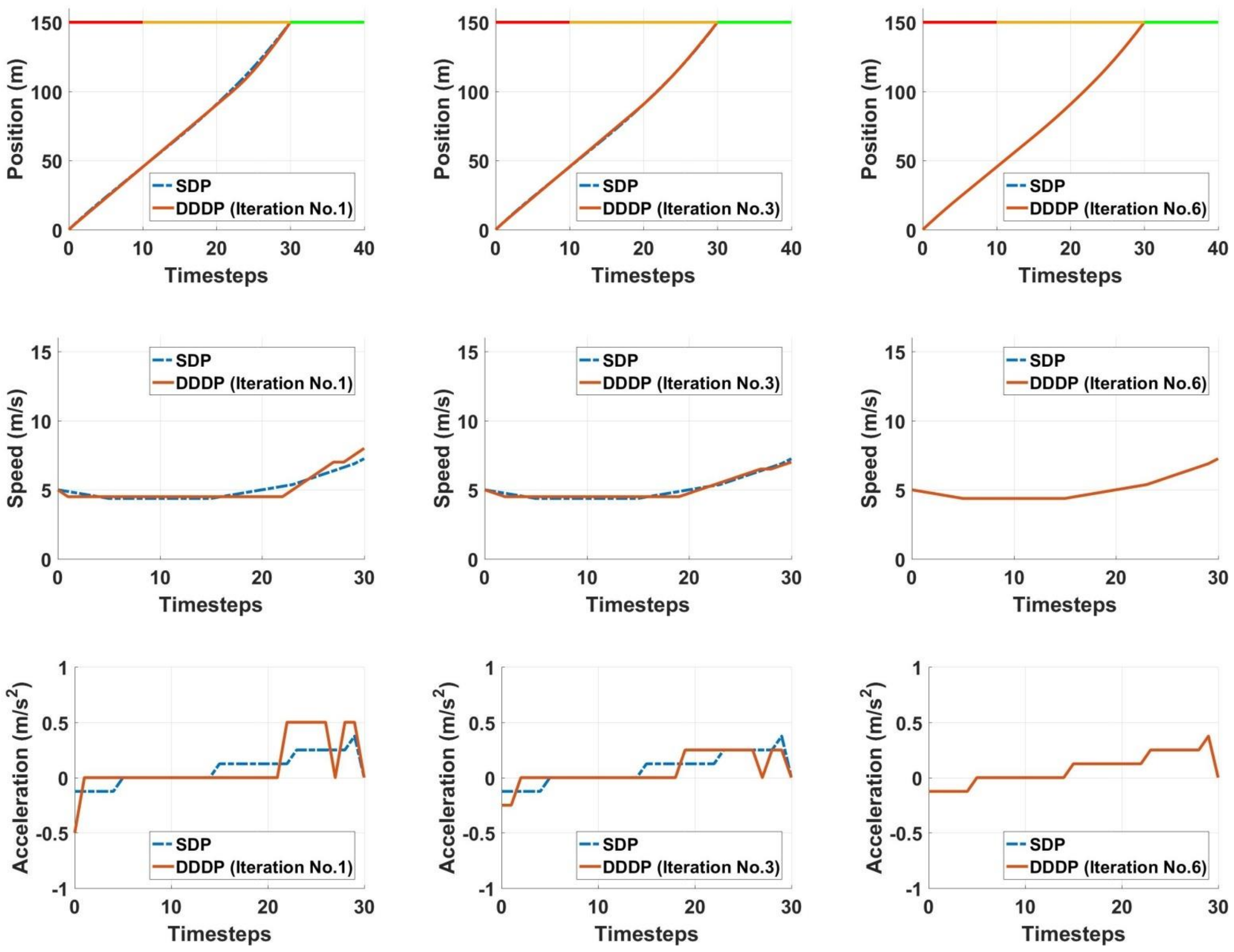


Figure 12: Control comparison, between the SDP and the DDDP algorithms.

Following, the tables mentioned above, comparing the computational time between the two methods:

	Cost	CPU-time (in seconds)
SDP algorithm	1.1752	613.8641
DDDP algorithm	1.1752	1.8073

Table 10: Comparing the SDP and the DDDP algorithms, considering CPU-time and cost reached, considering Scenario 2.

Below, the following figures present the optimal derived trajectories and the control variable, considering Scenario 2, for each one of the two algorithms. As it can be observed, the trajectories in the first iteration of the algorithm are quite different from the optimal ones (the SDP algorithm's), during the fifth iteration they seem to be closing up to one another and finally in the last iteration (the 10th) they are completely identical, which is the wanted outcome, due the fact that this indicates that the two algorithms have the exact same outcome, with the only differences in the operation and the CPU-time needed for each one.

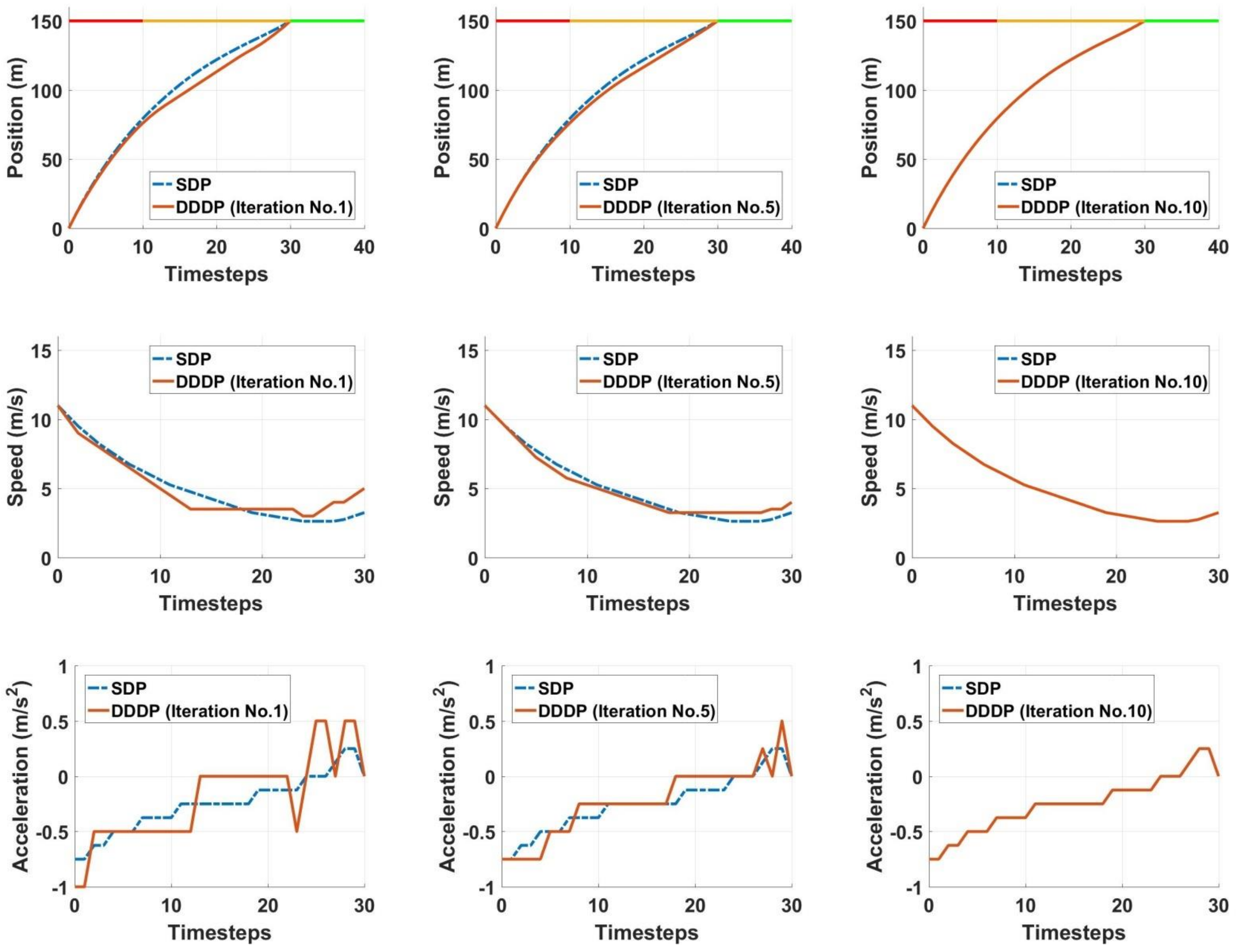


Figure 11: Trajectory and control comparison, between the SDP and the DDDP algorithms.

Following, the table mentioned above, comparing the computational time between the two algorithms:

	Cost	CPU-time (in seconds)
SDP algorithm	3.9068	617.5173
DDDP algorithm	3.9068	2.9176

Table 9: Comparing the SDP and the DDDP algorithms, considering CPU-time and cost reached, considering Scenario 1.

Below, the following figures present the optimal derived trajectories and the control variable, considering Scenario 3, for each one of the two algorithms.

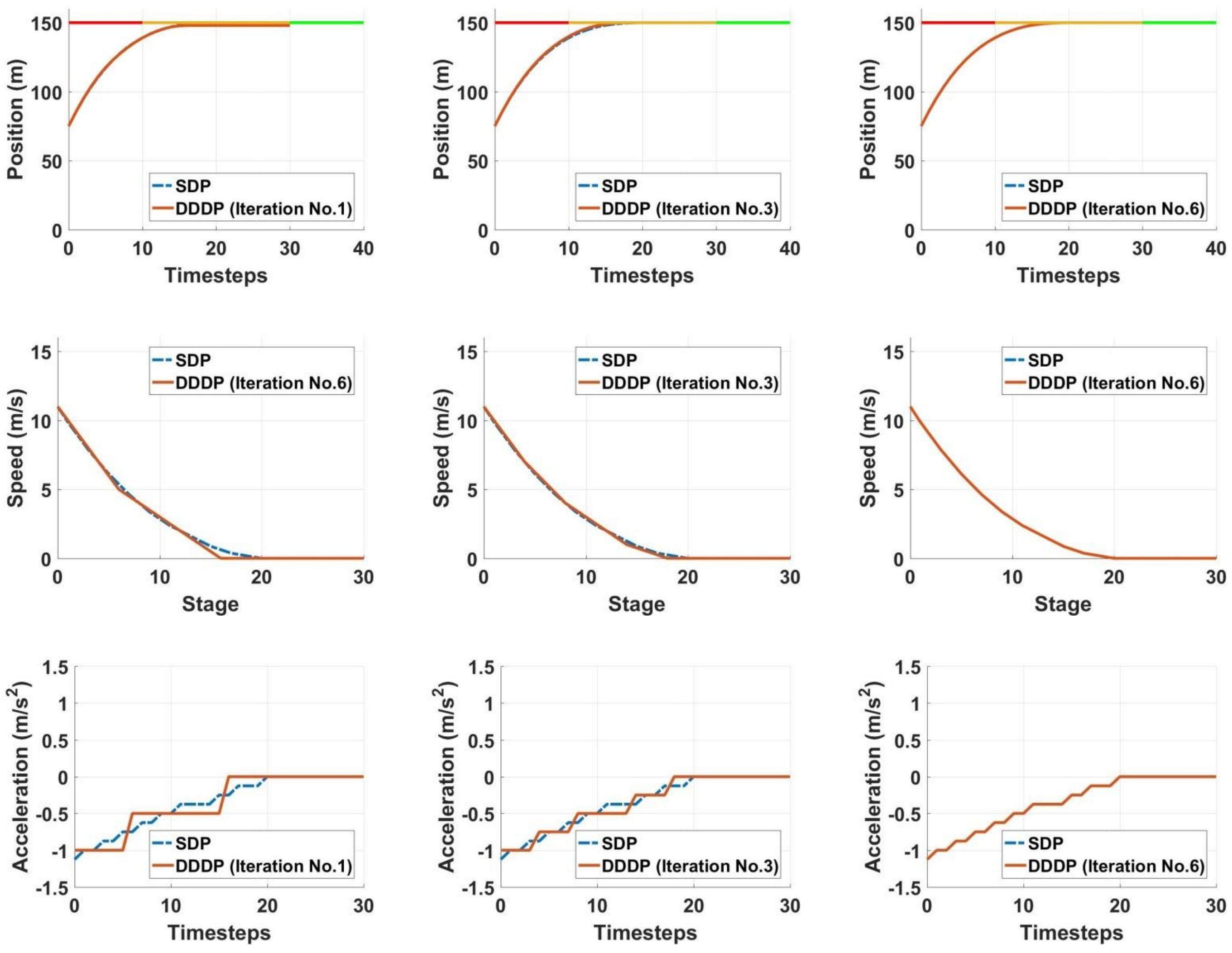


Figure 13: Control comparison, between the SDP and the DDDP algorithms.

Following, the tables mentioned above, comparing the computational time between the two methods:

	Cost	**CPU-time (in seconds)**
SDP algorithm	7.9107	610.5573
DDDP algorithm	7.9107	1.0955

Table 11: Comparing the SDP and the DDDP algorithms, considering CPU-time and cost reached, considering Scenario 3.

Summarizing, it can be noticed from the figures and tables above that, the two algorithms converge to the optimal, and the final trajectories are identical in both cases, meaning that the DDDP algorithm is doing the exact same assignment, and with the same accuracy, as the SDP algorithm, only within a substantial less amount of time, as it can be clarified from **Table 9**, **Table 10** and **Table 11**, making the DDDP algorithm capable of being used in an online vehicle trajectory specification algorithm.

Chapter 6: Conclusions and Future Steps

6.1: Conclusions

A stochastic GLOSA approach was developed, being basically the extension of the deterministic and stochastic GLOSA methodologies of Typaldos et. al. [4]. This particular method is the Discrete Differential Dynamic Programming Method. The GLOSA problem was formulated as an optimal control problem and solved firstly numerically through the DDDP method for the case of uncertain switching time of the traffic light. In both cases, the traffic light's switching time is assumed to be stochastic, with a given a-priori probability distribution between a known time interval$[k_{min}, k_{max}]$, which is properly updated as time advances. Also, the algorithm introduced in [4], is used to compare with the algorithms implemented in this work, in order to get a clearer view of what the suggested methods can accomplish.

Regarding the DDDP method, eachiteration of the corresponding algorithm solvesa stochastic problem in a reduced state space, which is formed around the last solution trajectory. The initial trajectory to start the iterations is the analytic solution of the pessimistic case,which assumes that the traffic signal will switch at the latest possible time.With this approach, the work load and computation cost is significantly reduced, making the method applicable and realtime orientated, i.e. capable of processing the given data to obtain the solution of the stochastic problem in few seconds.The DDDP algorithm has a considerable advantage over other methods, as it solves areduced problem in terms of state space. With this approach, it is consequent that the amount of computational time and the workload that the algorithm has to overcome, is reduced significantly, something that save a lot of time, examining trajectories that cannot be part of the problems solution.

6.2 Future Steps

As far as it concerns any future work and extensions of what is done in this work, here are some of the ideas that stand out, due to the fact that they are applicable in the imminent future, and also will probably elevate the quality of the work that has already been done.

To begin with, the adaptation of the DDDP algorithm implemented in this work into a usable application is considered, since these algorithms are developedin a way that makes them able to fulfil their tasks under the barrier of one second, meaning that the online trajectory specification is viable. For example, a cell-phone application that would be able to give online speed advices to a vehicle’s driver, or a pre-installed application in the vehicle’s electronics systems, or even a modified GPS (Global Position System) version with an installed speed advisor algorithm.This is one of the next steps, consisting another challenge (and the word "another" is being used, because of theobstacles and the roller-coaster troubleshooting process regarding the formulation of the algorithm, considering that building and deploying an application needs further actions and knowledge. With each algorithm’s code been written, a number of steps have already been completed, leaving operations like creating mock-ups for the application (a rough sketch of the application’s layout, also describing the flow and the interactions), making a graphic design (which contains graphic effects, image assets, even motion and animation design), building a landing page for the application (that explains what the application does, why it might provide useful for someone and generally giving a briefly explanation of its purpose, giving the opportunity to discover from early on potential users and people who are willing to test it and give back their opinion), yet to be done.

Furthermore, appliance of the method examined and used in this work in a multi-vehicle control system is contemplated. This endeavour is important due to the fact that, the optimization of a vehicle’s trajectory towards a traffic light would be viable for a number of vehicles, along with the aspect that this implementation can obtain further use, e.g. safer and less pollutive urban and highway transportation profile. As a result, an even higher level of fuel consumption and less gas (CO_2 mostly) emissions could be achieved, as well as a safer transportation environment prospect. A multi-vehicle control system presupposes the use of other methods and approaches, and it is a really demanding endeavour, so the implementation and the adaptation of the examined methods in such systems could be puzzling, but not impossible to happen. Also, the addition of scenarios with other vehicles or obstacles along the way is another important step, so the vehicle examined can either surpass the other vehicles from the second (speeding) lane, or stay behind them and adjust its velocity and acceleration accordingly.

Continuing, a complete overview of the DDP method is an also important step, since the investigation of the method has already started, but the completion of an overview and a thorough examination of this method is yet to be done.

Last but not least, a different and also interesting problem formulation is being examined, where a traffic signal exist along the course of the vehicle, and that particular traffic light when the vehicle approaches is under its green phase and gradually proceeding to the red phase.

References

[1] Stahlmann, R., Möller, M., Brauer, A., German, R., &Eckhoff, D. (2016, December). Technical evaluation of glosa systems and results from the field. In *2016 IEEE Vehicular Networking Conference (VNC)* (pp. 1-8). IEEE.

[2] Richter, A. (2005). Geschwindigkeitsvorgabe an lichtsignalanlagen. *DeutscherUniversitatsverlag*.

[3] Fogarty, T. C., & Bull, L. (1995). Optimising individual control rules and multiple communicating rule-based control systems with parallel distributed genetic algorithms. *IEE Proceedings-Control Theory and Applications*, *142*(3), 211-215.

[4] Typaldos, P., Kalogianni, I., Mountakis, K. S., Papamichail, I., &Papageorgiou, M. (2020). Vehicle trajectory specification in presence of traffic lights with known or uncertain switching times. *Transportation research record*, *2674*(8), 53-66.

[5] Koukoumidis, E., Peh, L. S., &Martonosi, M. R. (2011, June). Signalguru: leveraging mobile phones for collaborative traffic signal schedule advisory. In *Proceedings of the 9th international conference on Mobile systems, applications, and services* (pp. 127-140).

[6] Jamshidnejad, A., Papamichail, I., Papageorgiou, M., & De Schutter, B. (2017). Sustainable model-predictive control in urban traffic networks: Efficient solution based on general smoothening methods. *IEEE Transactions on Control Systems Technology*, *26*(3), 813-827.

[7] Sanchez, M., Cano, J. C., & Kim, D. (2006, June). Predicting traffic lights to improve urban traffic fuel consumption. In *2006 6th International Conference on ITS Telecommunications* (pp. 331-336). IEEE.

[8] Van Leersum, J. (1985). Implementation of an advisory speed algorithm in transyt. *Transportation Research Part A: General*, *19*(3), 207-217.

[9] Borkar, P., Welekar, A., Jenekar, S., &Karmore, S. (2012). Predictive traffic light control system: Existing systems and proposed plan for next intersection prediction. *International Technology Research Letters*, *1*(1), 107-111.

[10] Asadi, B., &Vahidi, A. (2009). Predictive use of traffic signal state for fuel saving. *IFAC Proceedings Volumes*, *42*(15), 484-489.

[11] Hounsell, N. B., & McDonald, M. (2001). Urban network traffic control. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, *215*(4), 325-334.

[12] Wang, Y., Yang, X., Liang, H., & Liu, Y. (2018). A review of the self-adaptive traffic signal control system based on future traffic environment. *Journal of Advanced Transportation*, *2018*.

[13] Stahlmann, R., Möller, M., Brauer, A., German, R., &Eckhoff, D. (2018). Exploring GLOSA systems in the field: Technical evaluation and results. *Computer Communications*, *120*, 112-124.

[14] Tielert, T., Killat, M., Hartenstein, H., Luz, R., Hausberger, S., & Benz, T. (2010, November). The impact of traffic-light-to-vehicle communication on fuel consumption and emissions. In *2010 Internet of Things (IOT)* (pp. 1-8). IEEE.

[15] Katsaros, K., Kernchen, R., Dianati, M., &Rieck, D. (2011, July). Performance study of a Green Light Optimized Speed Advisory (GLOSA) application using an integrated cooperative ITS simulation platform. In *2011 7th International Wireless Communications and Mobile Computing Conference* (pp. 918-923). IEEE.

[16] Seredynski, M., Dorronsoro, B., &Khadraoui, D. (2013, October). Comparison of green light optimal speed advisory approaches. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)* (pp. 2187-2192). IEEE.

[17] Stevanovic, A., Stevanovic, J., &Kergaye, C. (2013). Green light optimized speed advisory systems: Impact of signal phasing information accuracy. *Transportation research record*, *2390*(1), 53-59.

[18] Benavides, P. T., &Diwekar, U. M. (2012). Studying various optimal control problems in biodiesel production in a batch reactor under uncertainty. In *Computer Aided Chemical Engineering* (Vol. 31, pp. 385-389). Elsevier.

[19] Papageorgiou, M., Leibold, M., & Buss, M. (1991). Optimierung, Statische, Dynamische, StochastischeVerfahren f ur die Anwendung..

[20] Papageorgiou, M., "Dynamic Programming," *Technical University of Crete*, 2011.

[21] Bellman, R. (1954). *The theory of dynamic programming* (No. RAND-P-550). Rand Corp Santa Monica CA.

[22] Bellman, R. E. (1957). Dynamic programming, ser. *Cambridge Studies in Speech Science and Communication. Princeton University Press, Princeton*.

[23] Feng, Z. K., Niu, W. J., Cheng, C. T., & Liao, S. L. (2017). Hydropower system operation optimization by discrete differential dynamic programming based on orthogonal experiment design. *Energy*, *126*, 720-732.

[24] Heidari, M., Chow, V. T., Kokotović, P. V., & Meredith, D. D. (1971). Discrete differential dynamic programing approach to water resources systems optimization. *Water Resources Research*, *7*(2), 273-282.

[25] Tassa, Y., Mansard, N., & Todorov, E. (2014, May). Control-limited differential dynamic programming. In *2014 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 1168-1175). IEEE.

[26] Levine, S., &Koltun, V. (2013). Variational policy search via trajectory optimization. In *Advances in neural information processing systems* (pp. 207-215).

[27] Pan, Y., &Theodorou, E. (2014). Probabilistic differential dynamic programming. In *Advances in Neural Information Processing Systems*.

[28] Levine, S., &Koltun, V. (2013, February). Guided policy search. In *International Conference on Machine Learning* (pp. 1-9).

[29] Morin, D. (2008). The Hamiltonian Method. In *Cambridge University Press, Draft Version 2*.

[30] Ntousakis, I. A., Nikolos, I. K., &Papageorgiou, M. (2016). Optimal vehicle trajectory planning in the context of cooperative merging on highways. *Transportation research part C: emerging technologies*, *71*, 464-488.

[31] Rios-Torres, J., &Malikopoulos, A. A. (2016). Automated and cooperative vehicle merging at highway on-ramps. *IEEE Transactions on Intelligent Transportation Systems*, *18*(4), 780-789.

[32] Version, M. (2017). 11.2. Wolfram Research. *Inc. Champaign*.

[33] Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P., &Bertsekas, D. P. (1995). *Dynamic programming and optimal control* (Vol. 1, No. 2, p. 4). Belmont, MA: Athena scientific.

[34] Lawitzky, A., Wollherr, D., & Buss, M. (2013, November). Energy optimal control to approach traffic lights. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 4382-4387). IEEE.

[35] Bellman, R. (1966). Dynamic programming. In *Science,* vol. 153, no. 3731, pp. 34-37.

[36] Dasgupta, P. B. (2015). An Analytical Evaluation of Matricizing Least-Square-Errors Curve Fitting to Support High Performance Computation on Large Datasets. *arXiv preprint arXiv:1512.08017*.

[37] Bickel, P. J. K. A. (1977). K. A. Doksum, Mathematical Statistics. *Holden-Day Inc*, *181*, 611-617.

[38] Polynomial Regression Computations. Retrieved from http://www.public.asu.edu/~gwaissi/ASM-e-book/module403.html.

[39] Tospornsampan, J., Kita, I., Ishii, M., & Kitamura, Y. (2005). Optimization of a multiple reservoir system operation using a combination of genetic algorithm and discrete differential dynamic programming: a case study in Mae Klong system, Thailand. *Paddy and Water Environment*, *3*(1), 29-38.

[40] Liao, L. Z., & Shoemaker, C. A. (1992). *Advantages of differential dynamic programming over Newton's method for discrete-time optimal control problems*. Cornell University.

[41] Kalogianni, I. (2018). Fuel-minimizing vehicle trajectory specification in the presence of traffic lights with certain or stochastic switching times. In *Technical University of Crete.*

[42] Bodenheimer, R., Eckhoff, D., & German, R. (2015, October). GLOSA for adaptive traffic lights: Methods and evaluation. In *2015 7th International Workshop on Reliable Networks Design and Modeling (RNDM)* (pp. 320-328). IEEE.

[43] Larson, R. E. (1978). *Principles of Dynamic Programming: Basic Analytical and Computational Methods*. Marcel Dekker, Inc..

[44] Larson, R. E,Casti, J. L. (1982). Principles of Dynamic Programming - Part II: Advanced Theory and Applications. In *Dekker, New York.*

[45] Kobilarov, M., Ta, D. N., &Dellaert, F. (2015, May). Differential dynamic programming for optimal estimation. In *2015 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 863-869). IEEE.

[46] Ho, Y. C. (1975). *Applied optimal control: optimization, estimation, and control*. Hemisphere Publishing Corporation, distributed by Halsted Press.

[47] McReynolds, S. R. (1967). The successive sweep method and dynamic programming. *Journal of Mathematical Analysis and Applications*, *19*(3), 565-598.

[48] Lantoine, G., & Russell, R. P. (2012). A hybrid differential dynamic programming algorithm for constrained optimal control problems. part 1: Theory. *Journal of Optimization Theory and Applications*, *154*(2), 382-417.

[49] Lantoine, G., & Russell, R. P. (2012). A hybrid differential dynamic programming algorithm for constrained optimal control problems. part 2: Application. *Journal of Optimization Theory and Applications*, *154*(2), 418-442.

[50] Jacobson, D. H. (1967). Differential dynamic programming methods for determining optimal control of non-linear systems.

[51] Jacobson, D. H. (1968). New second-order and first-order algorithms for determining optimal control: A differential dynamic programming approach. *Journal of Optimization Theory and Applications*, *2*(6), 411-440.

[52] Rosenbrock, H. (1972). Differential Dynamic Programming. By D. H. Jacobson and D. Q. Mayne. Pp. viii, 208. 1970.(Elsevier.). *The Mathematical Gazette*, *56*(395), 78-78.

[53] Sato, N. (1969). Differential Dynamic Programming: An Optimization Technique for Nonlinear Systems.

[54] Plancher, B., &Kuindersma, S. (2018, December). A performance analysis of parallel differential dynamic programming on a GPU. In *International Workshop on the Algorithmic Foundations of Robotics* (pp. 656-672). Springer, Cham.

[55] DE O. PANTOJA, J. F. A. (1988). Differential dynamic programming and Newton's method. *International Journal of Control*, *47*(5), 1539-1553.

[56] Cheng, C., Wang, S., Chau, K. W., & Wu, X. (2014). Parallel discrete differential dynamic programming for multireservoir operation. *Environmental modelling & software*, *57*, 152-164.

[57] Press, W. H., Teukolsky, S. A., Flannery, B. P., &Vetterling, W. T. (1992). *Numerical recipes in Fortran 77: volume 1, volume 1 of Fortran numerical recipes: the art of scientific computing*. Cambridge University Press.

[58] Todorov, E. (2006). Optimal control theory. *Bayesian brain: probabilistic approaches to neural coding*, 269-298.

Chapter 7: Appendix

In this appendix are being presented the figures that represent the vehicle's position, in each time-step of the DDDP algorithm implemented, for every one of the three scenarios examined, from the initial position $x_0 = 0\,m$ until the final target $x_e = 220\,m$. The algorithm solves the problem with the DDDP method from $x_0 = 0\,m$ to $x_1 = 150\,m$ where the traffic light is located, and from x_1 until x_e the algorithm solves the analytical constrained problem (CP). So, in the following figures the trajectory that the analytical solution provides is being shown, along with the trajectory from x_0 to x_1 which the DDDP method provides. It can be observed that the corridor (red dashed lines) stops on x_1 and does not continue further, which is logical, since an analytical problem is being solved from there and on.

7.1 Appendix A

Considering Scenario 1:

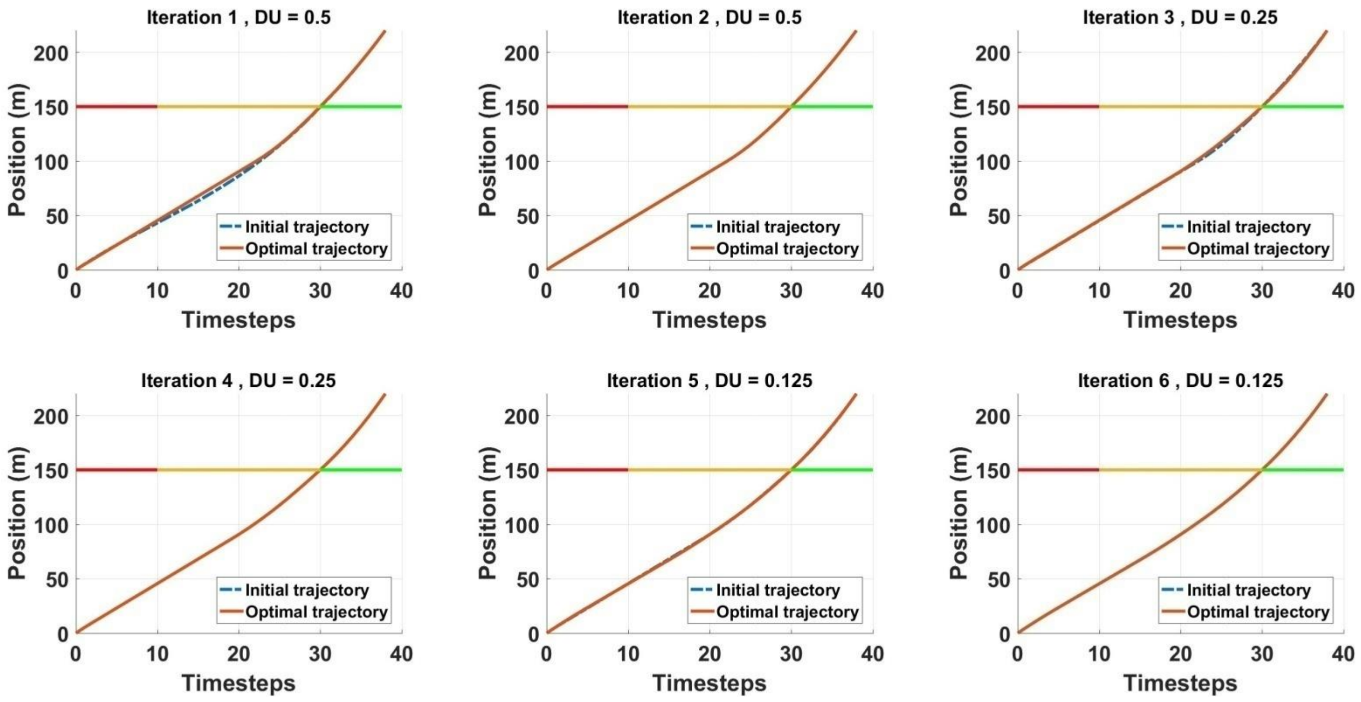


Figure 14: Initial and Optimal position of the vehicle, along with the analytical problem solution from x_1 to x_e, considering Scenario 1.

Continuing now with Scenario 2:

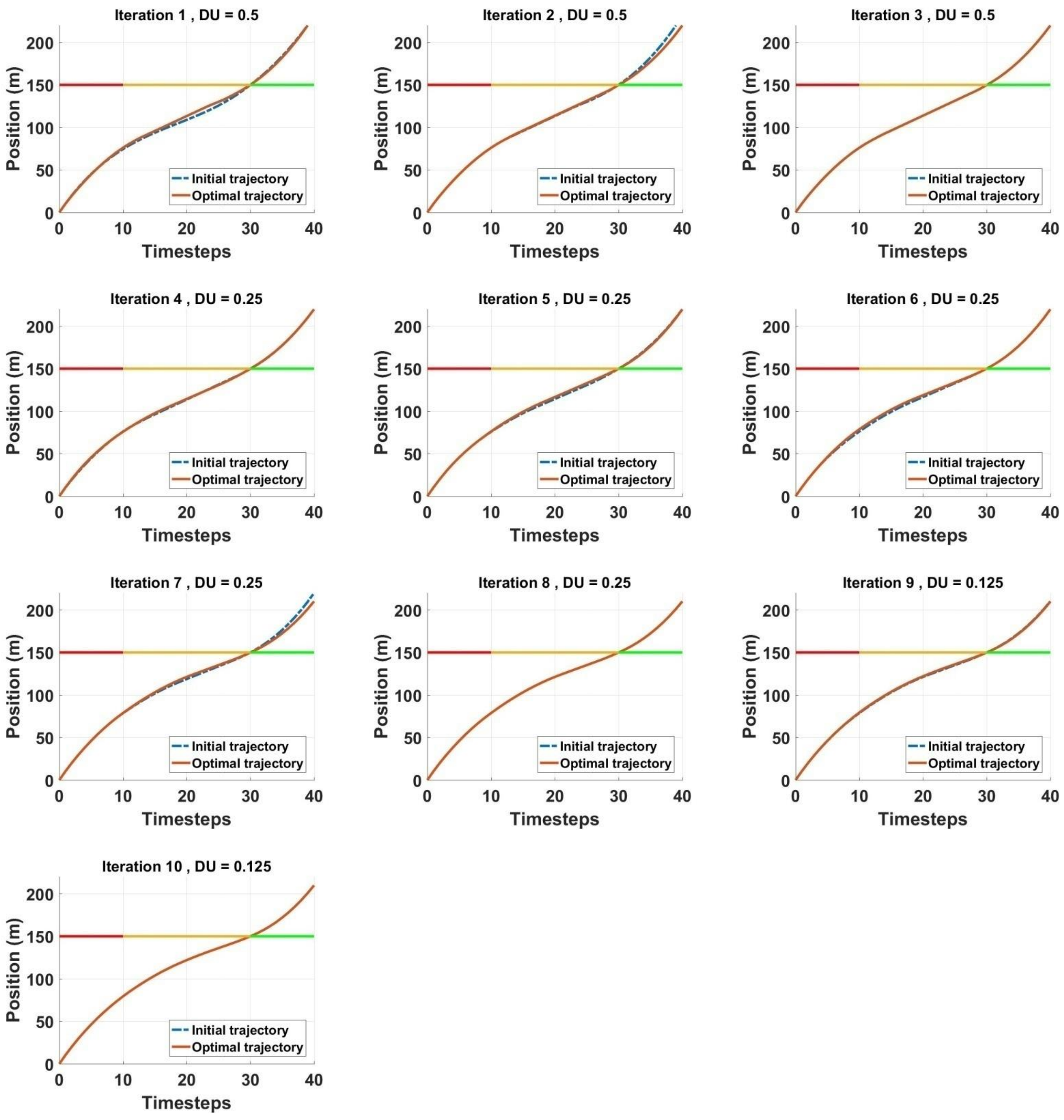


Figure 15: Initial and Optimal position of the vehicle, along with the analytical problem solution from x_1 to x_e, considering Scenario 2.

Closing with Scenario 3:

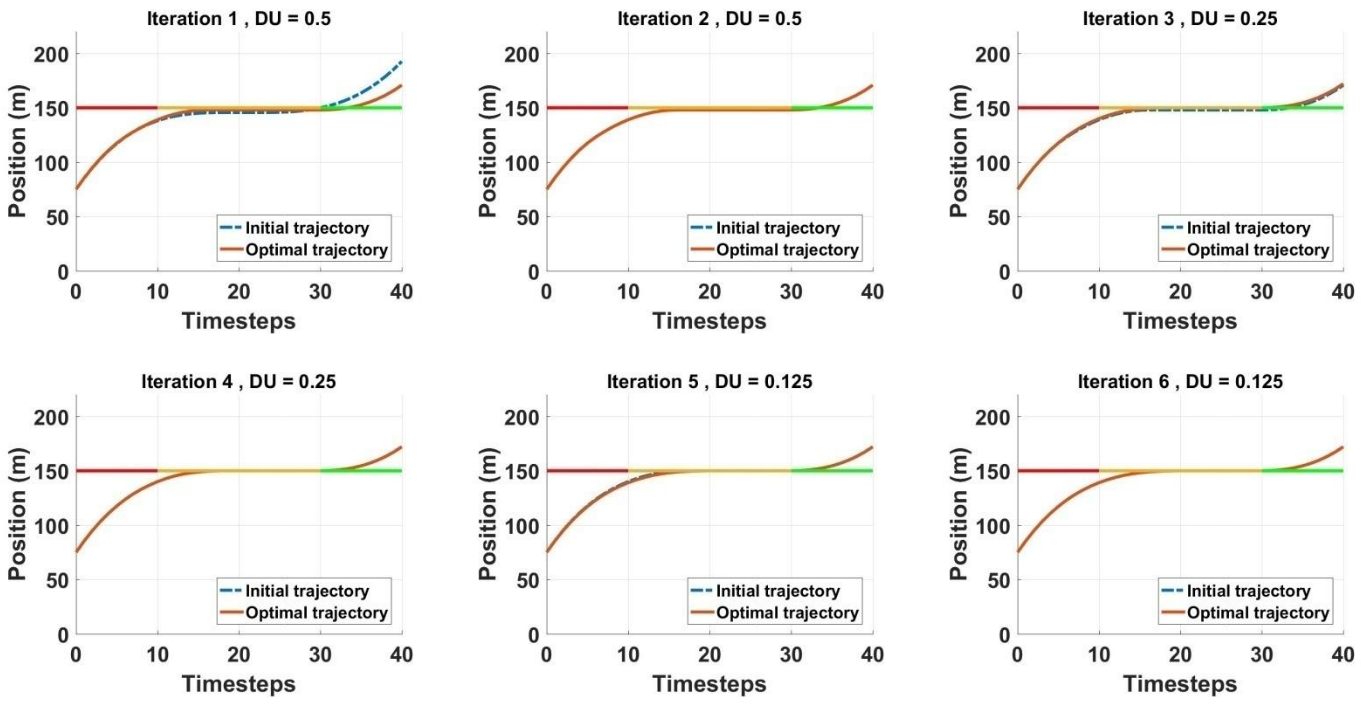


Figure 16: Initial and Optimal position of the vehicle, along with the analytical problem solution from x_1 to x_e, considering Scenario 3.