Real-time Trajectory Planning to Enable Safe and Performant Automated Vehicles Operating in Unknown Dynamic Environments

by

Huckleberry Febbo

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Mechanical Engineering)
at The University of Michigan
2019

Doctoral Committee:

Associate Research Scientist Tulga Ersal, Co-Chair
Professor Jeffrey L. Stein, Co-Chair
Professor Brent Gillespie
Professor Ilya Kolmanovsky

Huckleberry Febbo

febbo@umich.edu

ORCID iD: 0000-0002-0268-3672

For my mother and father

ACKNOWLEDGEMENTS

I want to thank my advisors, Prof. Jeffrey L. Stein and Dr. Tulga Ersal for their strong guidance during my graduate studies. Each of you has played pivotal and complementary roles in my life that have greatly enhanced my ability to communicate.

I want to thank Prof. Peter Ifju for telling me to go to graduate school. Your confidence in me has pushed me farther than I knew I could go.

I want to thank Prof. Elizabeth Hildinger for improving my ability to write well and providing me with the potential to assert myself on paper.

I want to thank my friends, family, classmates, and colleges for their continual support. In particular, I would like to thank John Guittar and Srdjan Cvjeticanin for providing me feedback on my articulation of research questions and ideas. I will always look back upon the 2018 Summer that we spent together in the Rackham Reading Room with fondness. Rackham . . . oh Rackham . . .

I want to thank Virgil Febbo for helping me through my last month in Michigan.

Finally, I am grateful for the funding that the Automotive Research Center provided to support my studies at the University of Michigan.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

LIST OF APPENDICES

Appendix

ABSTRACT

Need for increased automated vehicle safety and performance will exist until control systems can fully exploit the vehicle's maneuvering capacity to avoid collisions with both static and moving obstacles in unknown environments. A safe and performance-based trajectory planning algorithm exists that can operate an automated vehicle in unknown static environments. However, this algorithm cannot be used safely in unknown dynamic environments; furthermore, it is not real-time. Accordingly, this thesis addresses two overarching research questions:

1. How should a trajectory planning algorithm be formulated to enable automated ground vehicle safety and performance in unknown dynamic environments?

2. How can such an algorithm be solved in real-time?

Safe trajectory planning for high-performance automated vehicles with both static and moving obstacles is a challenging problem. Part of the challenge is developing a formulation that can be solved in real-time while including the following set of specifications: minimum time to goal, a dynamic vehicle model, minimum control effort, both static and moving obstacle avoidance, simultaneous optimization of speed and steering, and a short execution horizon. This thesis presents a nonlinear model predictive control-based trajectory planning formulation, tailored for a high mobility multipurpose wheeled vehicle (HMMWV), that includes the above set of specifications. This formulation is tested then with various sets of these specifications in a known dynamic environment. In particular, a parametric study relating execution horizon and obstacle speed reveals that the moving obstacle avoidance specification

is not needed for safety when the planner has a short execution horizon (≤ 0.375 s), and the obstacles are slow ($\leq 2.11 \frac{\text{m}}{\text{s}}$). However, a moving obstacle avoidance specification is needed when the obstacles move faster, and this specification improves safety without, in most cases, increasing solve-times. Overall, results indicate that trajectory planners for high-performance automated vehicles should include the entire set of specifications mentioned above unless a static or low-speed environment permits a less comprehensive planner.

Then, this thesis combines this comprehensive planning algorithm with a suitable perception algorithm to enable safe and performant control of automated ground vehicles in unknown dynamic environments. A high-fidelity, ROS-based proving ground with a 2D LiDAR model, in Gazebo, and a 145 degree of freedom model of the HMMWV, in Chrono, is developed to combine these algorithms. Six-hundred tests, realized with various obstacle speeds and sizes, are performed in this proving ground in both known and unknown dynamic environments. Results from this comparison demonstrate that operating in an unknown environment, as opposed to a known environment, significantly increases collisions, steering effort, throttle effort, braking effort, orientation and tracking error, time to goal, and planner solve times. To avoid this deterioration of safety and performance factors in unknown environments, the use of more accurate perception systems should be explored. Ultimately, however, these results demonstrate that the comprehensive trajectory planning formulation developed in this thesis enables safe and performant control of automated vehicles in unknown dynamic environments among small (≤ 2 m) obstacles traveling at speeds up to high ($20 \frac{\text{m}}{\text{s}}$).

To solve this formulation in real-time, an open-source, direct-collocation-based optimal control problem modeling language, called `NLOptControl`, is established in this thesis. Results demonstrate that `NLOptControl` can solve the formulation in real-time in both known and unknown environments. `NLOptControl` holds great

potential for not only improving existing off-line and on-line control systems but also engendering a wide variety of new ones.

CHAPTER I

Introduction

Companies that are developing automated ground vehicles, such as Tesla, Cruise, and Waymo, have Level 2 to Level 3 Automation [54]. Auto manufacturers such as General Motors, Ford, BMW, and Audi are claiming that within the next five years they will be manufacturing Level 4 to Level 5 automated ground vehicles [17]. However, research predicts that due to implementation requirements and cost, Level 5 automated ground vehicles will not become universal and affordable until the 2040 to 2050 time-frame [87]. Either way, it is clear that the ground vehicle transportation system of the future is likely to be automated and must be both safe and performant.

A safe and performance-based trajectory planning algorithm was developed to operate a high mobility multipurpose wheeled vehicle (HMMWV) in unknown static environments [92]. However, this algorithm cannot be used safely in unknown dynamic environments; furthermore, it is not real-time. The primary focus of this thesis is to introduce a trajectory planning algorithm that does not have these two limitations.

Accordingly, this thesis addresses two overarching research questions:

1. How should a trajectory planning algorithm be formulated to enable automated ground vehicle safety and performance in unknown dynamic environments?

2. How can such an algorithm be solved in real-time?

Trajectory planning algorithms are typically solved through a numerical optimal control formulation [108]. These formulations have enabled moving obstacle avoidance using either time-varying soft constraints [139] or using time-varying hard constraints [71, 103, 138]. However, current hard constraints formulations [71, 103, 138] have a safety limitation — they use a kinematic vehicle model, as opposed to a dynamic vehicle model. This limitation can result in a collision because the planner can determine a trajectory that the vehicle cannot physically follow [44, 45]. Moreover, for a feasible solution, soft constraints do not ensure obstacle avoidance [139], but hard constraints do. Therefore, for safety, trajectory planning algorithms should use hard constraints in their formulation and a dynamic vehicle model [38].

Such comprehensive trajectory planning algorithms can be reliably solved using a direct-collocation-based numerical optimal control method [92, 111, 125, 112, 73, 74, 64, 133, 44], but these methods often struggle to solve such algorithms in real-time [38, 92, 5, 55]. A large part of this solve-time is spent evaluating the nonlinear functions, which numerically approximate the continuous-time optimal control problem and their corresponding derivatives. [38] and [92] evaluate these functions and their derivatives in *MATLAB* using symbolic differentiation and analytic differentiation, respectively. On the other hand, a numerical optimal control software tool, called *CasADi*, uses the star-coloring method [52] and reverse automatic differentiation implemented in *C++*, to exploit the sparse structure of the Hessian matrix that is born from a numerical optimal control problem [3]. Research shows that *CasADi* solves a robot trajectory planning problem in real-time [133]. However, *CasADi* is not easy to use, because it requires its users to write the code for their selected direct-collocation method, which can lead to errors in the code and large development time [83]. Thus, this lack of an easy to use and fast numerical optimal control software tool makes formulating comprehensive planning algorithms and solving them in real-time challenging. Fortunately, advances in computational languages [13], sparse automatic

Table 1.1: Planner Specifications

specification	Description
S_1	static obstacle avoidance
S_2	minimum time-to-goal
S_3	dynamic vehicle model
S_4	minimum control effort
S_5	simultaneously optimize speed and steering
S_6	moving obstacle avoidance
S_7	small execution horizon

differentiation methods [59, 121], and optimization modeling languages [30] provide a unique opportunity to improve both the ease of use and the speed of direct-collocation-based numerical optimal control methods. This thesis identifies these advances and leverages them to help create a new optimal control modeling language called `NLOptControl` to solve comprehensive trajectory planning algorithms in real-time. With `NLOptControl` ready for use, the focus of this thesis shifts to back to developing the automated vehicle trajectory planning algorithm.

In many high-performance automated vehicle applications, e.g., in unmanned air vehicles (UAVs), unmanned ground vehicles (UGVs), and spacecraft, it is both desirable and challenging to plan safe trajectories in a dynamic environment. Part of this challenge is incorporating the set of specifications listed in Table 1.1 into a real-time planner, where real-time planning demands that the planner's solve-times are all less than the execution horizon. While trajectory planning systems that include subsets of the specifications listed in Table 1.1 exist, a planner that consists of all of them has not yet been developed.

Fig. 1.1 shows a conceptual scheme for comparing and developing trajectory planners. This scheme illustrates the conceptual performance and safety of a vehicle controlled using trajectory planners with different sets of specifications, operating either in a static environment with a stationary obstacle (top four traces) or a dynamic environment with a moving obstacle (bottom two traces). In all cases, the planning

and execution horizons are the same.

Static obstacle avoidance (S_1, Table 1.1) is a baseline specification in many trajectory planning systems, but is not, by itself, sufficient for either performance or safety. Regarding safety, if the trajectory planner does not use a dynamic vehicle model (S_3, Table 1.1), a trajectory that the vehicle cannot follow may be determined [44]. Such a trajectory may result in either a collision [44, 45] (Case A, Fig. 1.1) or some other catastrophic event, such as rollover in the case of a ground vehicle [95]. Despite this, some planners designed to avoid static obstacles for UAV applications [58] utilize a kinematic vehicle model (Case A, Fig. 1.1). By utilizing a dynamic vehicle model in trajectory planning, the actual vehicle can follow the prescribed trajectory more accurately. Planners designed to avoid static obstacles with a dynamic vehicle model (Case B, Fig. 1.1) exist for UGV applications [139]. However, the planner in Case B does not have a minimum time-to-goal specification (S_2, Table 1.1), which may result in failure for certain applications. For instance, in racing applications [81, 136], planning without this specification will likely result in a lost race. In these applications, the planner should include at least S_1-S_3 (Case C, Fig. 1.1), such that it can arrive at the goal in less time than a planner with only the static obstacle avoidance and dynamic vehicle model specifications. If minimizing fuel consumption and mechanical wear are additional concerns, then the minimum control effort specification (S_4, Table 1.1) needs to be included in the planner as well. Planners with S_1-S_4 exist in applications for UGVs [91] and UAVs [93, 101, 94]. A limitation of these planners is that they do not optimize both speed and steering (S_5, Table 1.1). Optimizing both allows the vehicle to both slow down more quickly and turn more tightly (shown in Case D, Fig. 1.1), thereby improving both performance and safety [92].

In a dynamic environment, while the trajectory planning specifications S_1-S_5 are necessary for both performance and safety, they are not sufficient (see Case E, Fig. 1.1). To improve collision avoidance (i.e., safety) in a dynamic environment there

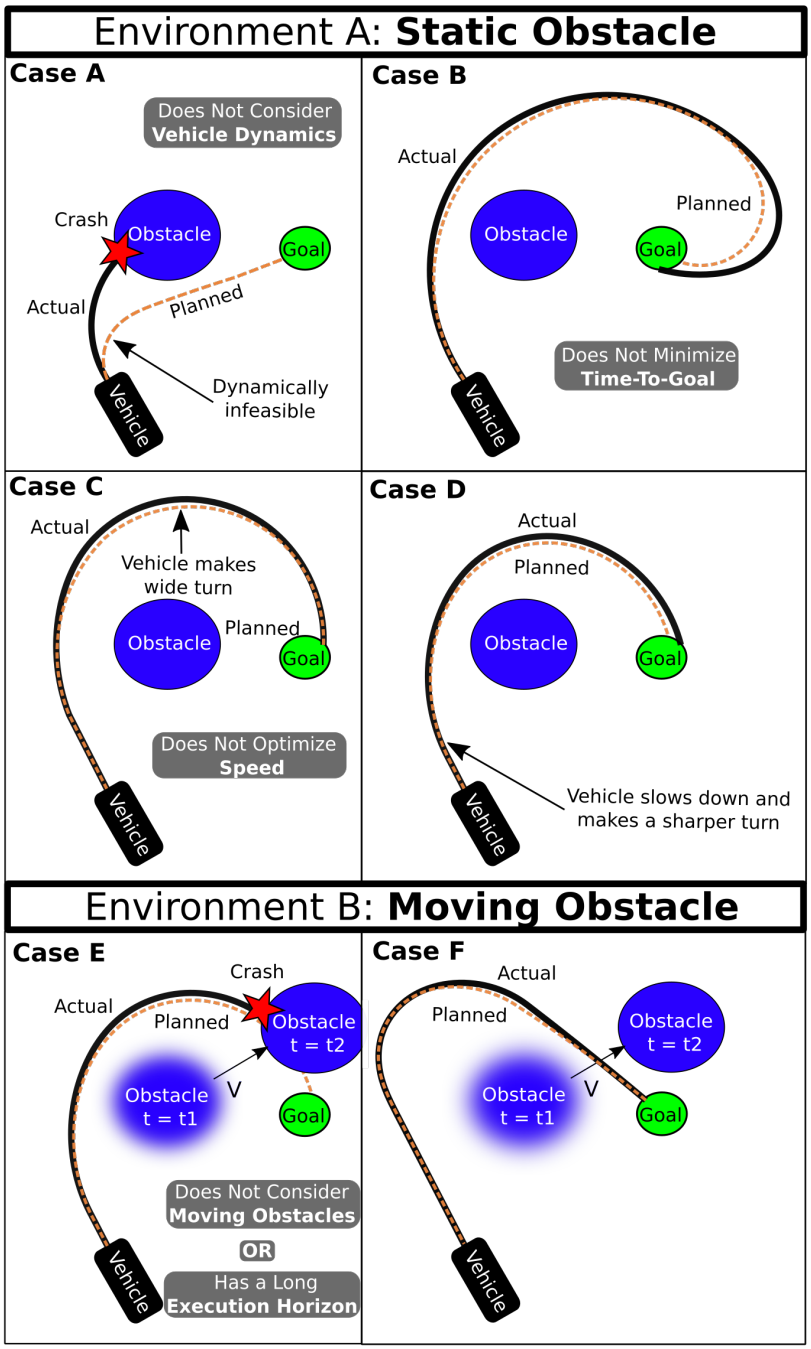


Figure 1.1: Comparison of trajectory planners illustrating the conceptual effect that planner specifications have on performance and safety within a given environment.

are three possibilities: including a moving obstacle avoidance specification (S_6, Table 1.1); including a small execution horizon specification (S_7, Table 1.1); or including both.

A moving obstacle avoidance specification accounts for the motion of the obstacle over the planning horizon, which increases safety (see Case F, Fig. 1.1). This specification has been implemented for applications in UGVs [103], UAVs [138], and spacecraft [71]. These developments, however, have a limitation: they use a kinematic vehicle model as opposed to a dynamic vehicle model; Case A, Fig. 1.1 depicts the potential outcome of using a kinematic vehicle model. Our preliminary work [38] developed a planner with S_1-S_6 for a UGV application. This work, however, has several limitations, three of which are: it does not investigate closed-loop performance and safety; it assumes that the goal is within the LiDAR's sensing range; and, finally, the planner's solve-times are, at best, nearly two orders of magnitude above real-time (assuming an execution horizon of 0.5 s). Among other things, this chapter addresses these three limitations.

A small execution horizon[1] specification engenders a more reactive planner with better obstacle avoidance capabilities. For instance, to avoid the collision in Case E (Fig. 1.1), a smaller execution horizon can be used. Previous research [92] includes a small execution horizon as well as S_1-S_5. While there is reason to expect that such a planner may operate safely around slowly moving obstacles, this hypothesis has not yet been tested. Therefore, this chapter also investigates, for the first time, whether a system with S_1-S_5 and a small execution horizon can operate safely in a dynamic environment for a range of obstacle speeds.

An major goal of this research is to develop a trajectory planning formulation that has all of the specifications listed in Table 1.1. The motivation for this investigation is the assumption that a planner with this set of specifications would represent an

[1] An execution horizon is described as "small" when reducing it does not improve safety within a given environment.

improvement in both safety and performance over planners with less comprehensive sets of specifications.

This work uses a nonlinear model predictive control (NMPC)-based trajectory planner; this approach is also used in [58, 139, 93, 101, 94, 53, 92, 103, 138, 71]. Unfortunately, it is challenging to solve the proposed planning formulation in real-time with a short execution horizon. For instance, the literature shows that *GPOCS*, *GPOPS-ii*, and our custom software, all written in the MATLAB computation language, are not fast enough for NMPC applications in aircraft [5], robot [55], and UGV [92, 38] systems, respectively. As part of this work, `NLOptControl`'s [37] ability to solve the proposed formulation in real-time with a short execution horizon is tested. As an example, the trajectory planning formulation developed in this work is tailored for an HMMWV but can be adapted to other vehicles as well. Together, this planner and vehicle are referred to as a UGV.

Finally, to safely relax the known environment assumption, the planner developed in this thesis must be combined with a perception algorithm that accurately predicts obstacle motion using sensor data. A Kalman-filter-based perception algorithm was recently developed that uses raw light detection and ranging (LiDAR) data to estimate obstacle size, position, and velocity [115]. However, this perception algorithm [115] has yet to be combined and tested with a planning algorithm. To combine and test these planning and perception algorithms, this thesis develops a ROS-based proving ground, called Michigan Autonomous Vehicles (MAVs), which uses a 2D LiDAR model in Gazebo and a 145 degree of freedom model of an HMMWV in Chrono. Results from this proving ground statistically quantify the accuracy of the perception system, which is the LiDAR sensor and perception algorithm combination. This quantification is realized by comparing collision avoidance, tracking and orientation error, planner solve-times, time to goal, steering effort, throttle effort, and braking effort in a known environment to those in an unknown environment.

The ultimate goal of this thesis is enabling real-time trajectory planning for automated vehicle safety and performance in unknown dynamic environments.

1.1 Dissertation organization

Chapter II formulates and compares trajectory planning algorithms using each hard and soft constraints to enable moving obstacle avoidance with a dynamic vehicle model. Chapter III describes `NLOptControl` in detail and compares its ease of use and speed to those of *PROPT*, a commercial optimal control software tool. `NLOptControl` is then used in Chapter IV to solve the proposed trajectory planning formulation in real-time in a known environment. Chapter V relaxes the assumption that the environment is known, which is made by Chapter IV, and tests the proposed planning algorithm in a high-fidelity, simulation-based proving ground. Finally, Chapter VI answers the overarching research questions and identifies the notable contributions of this thesis.

CHAPTER II

Trajectory Planning in Dynamic Environments

2.1 Introduction

Avoiding collisions with obstacles is an important problem for mobile robots, autonomous vehicles (AVs) and unmanned ground vehicles (UGVs). For computational efficiency, control of these vehicles is often carried out using a hierarchical scheme wherein a high-level path planner quickly generates a reference trajectory and then a vehicle-level controller is employed to track the reference trajectory. In the literature this hierarchical approach is refereed to as a two-level structure [43, 86]. However, when it becomes necessary to push the vehicle to its dynamical limits by either minimizing time-to-goal or maximizing progress-on-track, as in racing situations [135, 81] or with military applications, the high-level path planner may create dynamically infeasible trajectories because it often only considers simple vehicle dynamics. Additionally, collisions with obstacles may occur if the vehicle deviates from the reference trajectory, because vehicle-level controllers do not generally constrain the vehicle to avoid obstacles.

To mitigate these issues several researchers optimize the control commands for the vehicle using a single-level structure [43, 86]. In a single-level structure, there is no reference trajectory available to the vehicle-level controller. Instead, path planning and vehicle-level control are carried out simultaneously. Thus, for a properly con-

strained system with an appropriate vehicle model, the trajectories generated for all feasible solutions will be both dynamically feasible and collision free. Additionally, when the goal of the optimization is to minimize time-to-goal or progress-on-track and the problem is setup using a single-level structure, the entire state-space can be explored and the control signals that push the vehicle to its dynamic limits can be identified.

AVs and UGVs are often controlled using model predictive control (MPC) [7, 36, 46, 86, 89, 90, 139]. MPC is capable of controlling complex nonlinear systems bound by nonlinear constraints and it works by optimizing the control signals using a model of the system over a given prediction horizon and subsequently executing a portion of these optimized signals.

Using a two-level structure, where the higher level is a path planner and at the vehicle level MPC is used for path tracking, researchers controlled a sports vehicle to drive autonomously at high speeds along a mountain road [81]. Additionally, MPC has also been used to develop active steering algorithms to assist drivers in avoiding obstacles [77], to limit the driver input inside a safe handling envelope [7], and to stabilize a vehicle using an AV steering system [16]. In [77], authors compare the performance of an MPC controller where the vehicle model is linearized at the beginning of the prediction horizon with one where the vehicle model is linearized about a reference trajectory over the entire prediction horizon, an approach often referred to as linear time-varying MPC. In [7], authors develop an active steering algorithm that optimizes the front lateral tire forces in lieu of steering angle to formulate a convex optimization problem that can be solved quickly. In [16] authors introduced the notion of using nonlinear MPC (NLMPC) to control the steering angle of a vehicle along a reference trajectory. However, all of these optimizations utilize a two-level structure and time-to-goal minimization or progress-on-track maximization are not explicitly considered.

Compared to the two-level structure approach, there has been much less work that focuses on controlling AVs and UGVs using a single-level structure, but favorable results have been reported. In particular, researchers control $1:43$ scale race cars in highly nonlinear operating regimes while maximizing progress-on-track [86]. However, [86] focused on small radio-controlled race cars, whereas this work focuses on large vehicles such as a High Mobility Multipurpose Wheeled Vehicle (HMMWV) where rollover is a major concern and must be accounted for in the constraints. In [89], an NLMPC algorithm that uses a single-level structure is introduced that operates a large AV in unstructured environments (without lanes or traffic rules) to optimize the steering angle in order to minimize time-to-goal while avoiding static obstacles. This work is extended to include the optimization of reference longitudinal speed in addition to the steering angle in [90]. The obstacles, however, were still considered to be static.

Using a single-level structure, Yoon et al. utilize soft constraints for moving obstacle avoidance while considering the vehicle's dynamical limits using NLMPC, but this approach does not guarantee obstacle avoidance for a feasible solution and the authors focused on a low speed vehicle with a short prediction horizon [139]. Obstacle avoidance can only be guaranteed for a feasible solution if it is implemented using hard constraints.

In summary, a hard constraints approach to handling moving obstacles using a single-level structure for large vehicles with significant dynamics has not yet been investigated.

This work aims to fill this gap by developing an optimal control formulation that uses hard constraints to avoid moving obstacles using a single-level structure based on the prior effort in [88, 91, 89, 90]. It is assumed that an obstacle tracking and detection algorithm such as the one developed in [113] is utilized, so that both the shapes and time-varying positions of all obstacles are known. Additionally, the soft

constraints approach is also implemented into the algorithm and compared to the hard constraints approach. It is shown that the hard constraints method both yields a better obstacle avoidance performance and reduces optimization time when compared to the soft constraints approach.

Therefore, the novel and salient contributions of this work are:

1. Using hard constraints to avoid collisions with moving obstacles for a UGV using a single-level optimal control structure,

2. Comparing soft constraints to hard constraints in current context.

The remainder of this chapter is organized as follows. Section 2.2 presents the overall problem formulations for both hard constraints and soft constraints methods. In Section 2.3 two examples are investigated that compare the hard and soft constraints methods. Finally, in Section 2.4 the work is summarized and conclusions are given.

2.2 Problem Formulation

This work leverages the single-level optimal control problem (OCP) formulation in [90] and modifies it to accommodate moving obstacles using the hard constraints approach. As a benchmark, the soft constraints approach is also implemented. For simplicity, these two approaches are compared by comparing the OCP solutions without closing the loop.

The controller executes the solution from the following general OCP:

$$\underset{\boldsymbol{\xi},\,\boldsymbol{\zeta},\,T_p}{\text{minimize}} \qquad J = \mathcal{T}\left(T_p\right) + \left\{ \int_0^{T_p} \mathcal{I}\left[\boldsymbol{\xi}\left(t\right), \boldsymbol{\zeta}\left(t\right)\right] \mathrm{d}t \right\} \tag{2.1}$$

$$\text{subject to} \qquad \dot{\boldsymbol{\xi}}\left(t\right) = \boldsymbol{\mathcal{V}}\left[\boldsymbol{\xi}\left(t\right), \boldsymbol{\zeta}\left(t\right)\right] \tag{2.2}$$

$$\boldsymbol{\xi}_{\min}\left(t\right) \leq \boldsymbol{\xi}\left(t\right) \leq \boldsymbol{\xi}_{\max}\left(t\right) \tag{2.3}$$

$$\boldsymbol{\zeta}_{\min}\left(t\right) \leq \boldsymbol{\zeta}\left(t\right) \leq \boldsymbol{\zeta}_{\max}\left(t\right) \tag{2.4}$$

$$\boldsymbol{\mathcal{F}}\left[\boldsymbol{\xi}\left(T_p\right), \boldsymbol{\xi}\left(0\right)\right] \leq 0 \tag{2.5}$$

$$\boldsymbol{\mathcal{R}}\left[\boldsymbol{\xi}\left(t\right)\right] \leq 0 \tag{2.6}$$

$$\boldsymbol{\mathcal{S}}\left[\boldsymbol{\xi}\left(t\right)\right] \leq 0 \tag{2.7}$$

$$T_p \leq T_{p,\max} \tag{2.8}$$

When Eq. (2.1) is minimized subject to the constraints in Eq. (2.2) - Eq. (2.8), the optimal control vectors $\boldsymbol{\zeta}$, state vectors $\boldsymbol{\xi}$, and prediction time T_p can be calculated. In the following sections, these equations are expanded and described starting with the constraints and finishing with the cost function.

There are several sets of constraints that are identical in both the hard constraints and soft constraints approaches; the vehicle dynamics must be feasible (Eq. (2.2)), both the state and control trajectories must lie within their respective bounds (Eq. (2.3) and Eq. (2.4)), the vehicle must get to the goal (Eq. (2.5)), the vehicle must avoid collisions with moving obstacles (Eq. (2.6)), the maneuver must be dynamically safe (Eq. (2.7)), and the final prediction time must be less than $T_{p,\max}$ (Eq. (2.8)). The difference between the two approaches is the way that obstacle avoidance is implemented and is described in detail below.

Eq. (2.2)**: Vehicle Dynamics**

The level of model fidelity necessary in the model predictive controller for a large UGV represented by a 14 Degree of Freedom (DoF) vehicle model is well captured by a 3 DoF vehicle model with nonlinear bounds on acceleration/deceleration, a nonlinear tire model, and longitudinal load transfer [88]. The NLMPC vehicle model, governed by the state space equation shown in Eq. (2.9), is leveraged within the OCP to identify to control inputs over the prediction horizon.

$$\dot{\boldsymbol{\xi}} = \boldsymbol{\mathcal{A}}(\boldsymbol{\xi}) + \boldsymbol{\mathcal{B}}\boldsymbol{\zeta} \tag{2.9}$$

where

$$\boldsymbol{\mathcal{A}}(\boldsymbol{\xi}) = \begin{bmatrix} U\cos\psi - (V + L_f\omega_z)\sin\psi \\ U\sin\psi + (V + L_f\omega_z)\cos\psi \\ \omega_z \\ a_x \\ (F_{y,f} + F_{y,r})/M_t - U\omega_z \\ (F_{y,f}L_f - F_{y,r}L_r)/I_{zz} \\ 0 \\ 0 \end{bmatrix} \text{ and } \boldsymbol{\xi} = \begin{bmatrix} x \\ y \\ \psi \\ U \\ V \\ \omega_z \\ \delta_f \\ a_x \end{bmatrix}$$

$$\boldsymbol{\mathcal{B}^T} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } \boldsymbol{\zeta} = \begin{bmatrix} \gamma_f \\ J_x \end{bmatrix}$$

where the state and control vectors are $\boldsymbol{\xi}$ and $\boldsymbol{\zeta}$, respectively, a_x and J_x represents the longitudinal acceleration and jerk, respectively, γ_f is the steering rate, M_t is the total vehicle mass, and I_{zz} is the moment of inertia about the center of mass. As shown in Fig. 2.1, ψ represents the heading angle, L_f and L_r represent the distances between the front and rear axles to the center of mass, respectively, U and V represent the longitudinal and lateral speeds, respectively, x and y describe the global position of

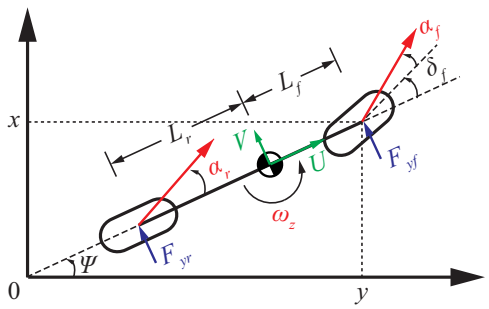


Figure 2.1: The 3 DoF vehicle model used in the optimal control problem formulation [90]

the center of the front axle, δ_f is the front steering angle, ω_z is the yaw rate, $F_{y,f}$ and $F_{y,r}$ represent the front and rear lateral tire forces, respectively, and finally, α_f and α_r represent the front and rear tire slip angles, respectively.

Eq. (2.3)**-Eq.** (2.4)**: State and Control Bounds**

γ_f and J_x are chosen as control variables to achieve smoother responses for δ_f and a_x, respectively. Additionally, γ_f and J_x are added to the optimization so that they can be bounded at each instant in time (t) based off of the physical limits of the vehicle:

$$\gamma_{f,\min} \leq \gamma_f(t) \leq \gamma_{f,\max} \tag{2.10}$$

$$J_{x,\min} \leq J_x(t) \leq J_{x,\max} \tag{2.11}$$

Nonlinear acceleration/deceleration bounds, determined by studying the acceleration/deceleration limits of the 14 DoF plant model in the previous work [90], are incorporated, which are a function of longitudinal vehicle speed:

$$a_{x,\min}[U(t)] \leq a_x(t) \leq a_{x,\max}[U(t)] \tag{2.12}$$

Additional bounds based off of both the vehicle limits and the desired vehicle

behavior are placed on x, y, U, ψ, and δ_f:

$$x_{\min} \leq x\left(t\right) \leq x_{\max} \tag{2.13}$$

$$y_{\min} \leq y\left(t\right) \leq y_{\max} \tag{2.14}$$

$$U_{\min} \leq U\left(t\right) \leq U_{\max} \tag{2.15}$$

$$\psi_{\min} \leq \psi\left(t\right) \leq \psi_{\max} \tag{2.16}$$

$$\delta_{f,\min} \leq \delta_f\left(t\right) \leq \delta_{f,\max} \tag{2.17}$$

There are no explicit restrictions on lateral speed or yaw rate.

Eq. (2.5)**: Final State Constraints**

Constraints are also placed on the vehicle's position (x, y) to be within a small distance σ from the goal position (x_g, y_g) at T_p. Mathematically, these constraints are expressed as:

$$x_g - \sigma \leq x\left(T_p\right) \leq x_g + \sigma \tag{2.18}$$

$$y_g - \sigma \leq y\left(T_p\right) \leq y_g + \sigma \tag{2.19}$$

Eq. (2.6)**: Moving Obstacle Avoidance using Hard Constraints**

In the hard constraints approach, obstacle avoidance is guaranteed for the NLMPC vehicle model for all feasible solutions, because hard constraints are added to insure that the trajectories of the vehicle and the obstacles do not intersect over T_p. Obstacles are represented as super-ellipses; thus, the following constraint is enforced on the trajectory for each obstacle:

$$\left|\frac{x(t) - x^i{}_{\text{obs}}(t)}{e+m}\right|^P + \left|\frac{y(t) - y^i{}_{\text{obs}}(t)}{f+m}\right|^P > 1$$

where ${x^i}_{\text{obs}}(t)$ and ${y^i}_{\text{obs}}(t)$ describe the global position of the center of the ith obstacle at t, e and f describe the semi-major and semi-minor axes, respectively, P defines the shape of the super-ellipse, and m is a safety margin that is added around each obstacle to account for the size of the vehicle.

Eq. (2.7): Dynamical Safety Constraints

In order to provide the plant with a dynamically feasible path, the vehicle is prevented from rolling over. Identical to [89, 90], this is done by constraining the vertical tire force on the rear tires to be above a particular threshold, $F_{z,\text{threshold}}$. Constraints are applied to the rear tires only due to the observation that, for the specific vehicle considered, rear tires experience lift-off before the front tires [90].

Eq. (2.1): Cost Function

The cost function consists of four terms that are linearly combined and multiplied by their respective weighing parameters w_t, w_{ψ_f}, w_{F_z}, and w_{cf}.

$$\begin{aligned} J = {} & w_t T_p + \\ & w_{\psi_f} \int_0^{T_p} \left[\sin(\psi_g)(x - x_g) - \cos(\psi_g)(y - y_g)\right]^2 \mathrm{d}t + \\ & w_{F_z} \int_0^{T_p} \left[\tanh\left(-\frac{F_{z,rl} - a}{b}\right) + \tanh\left(-\frac{F_{z,rr} - a}{b}\right)\right] \mathrm{d}t + \\ & w_{\text{cf}} \int_0^{T_p} \left[w_\delta \delta_f^2 + w_\gamma \gamma_f^2 + w_J J_x^2\right] \mathrm{d}t \end{aligned} \tag{2.20}$$

The first term term minimizes T_p. For a feasible solution, T_p is also the time that it takes for the vehicle to get to the goal. By minimizing this time, the vehicle is effectively pushed towards its dynamical limits. The next term helps insure that the vehicle passes the goal point (x_g,y_g) through a desired direction (ψ_g). The third term

in the cost function is a soft constraint on the vertical tire load that dissuades the vehicle from operating too close to the threshold on the vertical tire load; a and b are parameters. More information on the third term can be found in [90]. The final term in the cost function penalizes the control effort of the vehicle over the entire prediction horizon. w_δ, w_γ, and w_J are additional weighing terms on the steering angle, steering rate and longitudinal jerk, respectively.

Moving Obstacle Avoidance using Soft Constraints

In this approach, the hard constraints on obstacle avoidance (Eqn. (2.6)) are removed and an additional term SC_i is added to the cost function (Eq. (2.20)) for each obstacle to promote obstacle avoidance.

$$SC_i = w_{\text{obs}} \int_0^{T_p} \frac{\beta(t,i)}{\left(d(t,i)+\varepsilon\right)^2} dt$$

with

$$d(t,i) = \sqrt{\left(x(t) - {x^i}_{\text{obs}}(t)\right)^2 + \left(y(t) - {y^i}_{\text{obs}}(t)\right)^2}$$

where $\beta(t,i)$ is a boolean that is set to unity if the vehicle is within a certain distance of the obstacle. For circular obstacles ($e = f$ and $P = 2$) $\beta(t,i)$ is set to unity if $d(t,i) \leq e + m$. If the obstacle is not a circle, there is no analytical expression to determine if the vehicle is outside of the obstacle and m, so $d(t,i)$ is calculated numerically as described below. If the vehicle is at least a distance of m from the nearest point on the obstacle, then $\beta(t,i)$ is set to zero. Finally, w_{obs} is a weighing term that promotes obstacle avoidance, and ε is a small number used to avoid singularities.

To calculate $d(t,i)$ for an elliptical obstacle, the initial approach, referred to as SC_1, was to find the nearest point to the vehicle on the edge of a rectangular grid of query points enlarged around the obstacle by $1.2 \times m$ in both the x and y directions

similar to [139]. While this works for simple cases, this approach results in a lack of information in the gradient function when the optimization is evaluating solutions where the vehicle trajectory is within the grid of query points. This can result in convergence on a solution that crashes into an obstacle even when there is another feasible obstacle-free path. Therefore, in this work another approach, referred to as SC_2, was adopted wherein $d(t, i)$ is calculated to the center of the obstacle, so that there is gradient information available to the optimization; it is more costly to drive through the middle of the obstacles than through the sides. In the cases tested, SC_2 was found superior to SC_1 in terms of obstacle avoidance performance and convergence speed, so the Results and Discussion section focuses on comparing hard constraints to SC_2 and only a small section is included that demonstrates how SC_1 can fail to highlight that the designer must be careful when building SC_i.

Solving the OCP

The aforementioned continuous time OCP is transcribed into a nonlinear programming problem using Euler's Backward Difference method and solved using the Interior Point Method implemented in IPOPT [4].

2.3 Results and Discussion

Demonstration of the hard constraints formulation developed in this work is provided using two scenarios that involve multiple moving obstacles. Comparisons to the soft constraints approach are also included.

Case 1

In the first example, there are three circular obstacles; one is large and static, another is medium size and moving left in front of the vehicle between the start and goal points, and the last one is roughly the size of a HMMWV (called the small

obstacle for this case) and is moving in the direction from the start point to the goal point. Both hard and soft constraints formulations identify feasible solutions, albeit they are quite different as shown in Fig. 2.2. Using the soft constraints method the vehicle overtakes the small obstacle to the right and maneuvers just to the left of the large obstacle and to the right of the medium obstacle, whereas the hard constraints method identifies a solution that overtakes the small obstacle to the left and is able to maneuver to the left of both the medium and large obstacles. Thus the hard constraints method is able to reduce time-to-goal by 30% compared to the soft constraints approach. In Fig. 2.3, the vertical tire force, longitudinal speed and steering angle traces are shown for both methods. Towards the end of the trajectories, both vehicles are operated very close to their limit for minimum tire vertical load.

Another important consideration when comparing these optimizations is the optimization time. The optimization time was 0.74 and 1.83 min for the hard and soft constraints methods, respectively, on a 2.9 GHz CPU. While neither one of these times are considered to be fast enough for real-time experimentation, it is worth noting that the optimizations were implemented in MATLAB for fast development purposes, and an implementation in a compiled language can be expected to be significantly more efficient. More importantly, the hard constraints method was able to reduce the optimization time by 60% compared to the soft constraints approach.

Finally, it is noted that the soft constraints formulation converged on a local minimum, as the solution obtained by the hard constraints method yields a smaller objective function value also for the objective function used for soft constraints.

Case 2

In the second example, both algorithms are tested in a scenario with 17 moving obstacles of various shapes (to represent cars, HMMWVs, and tanks) and speeds that are moving horizontally between the vehicle starting and the goal points. Both the

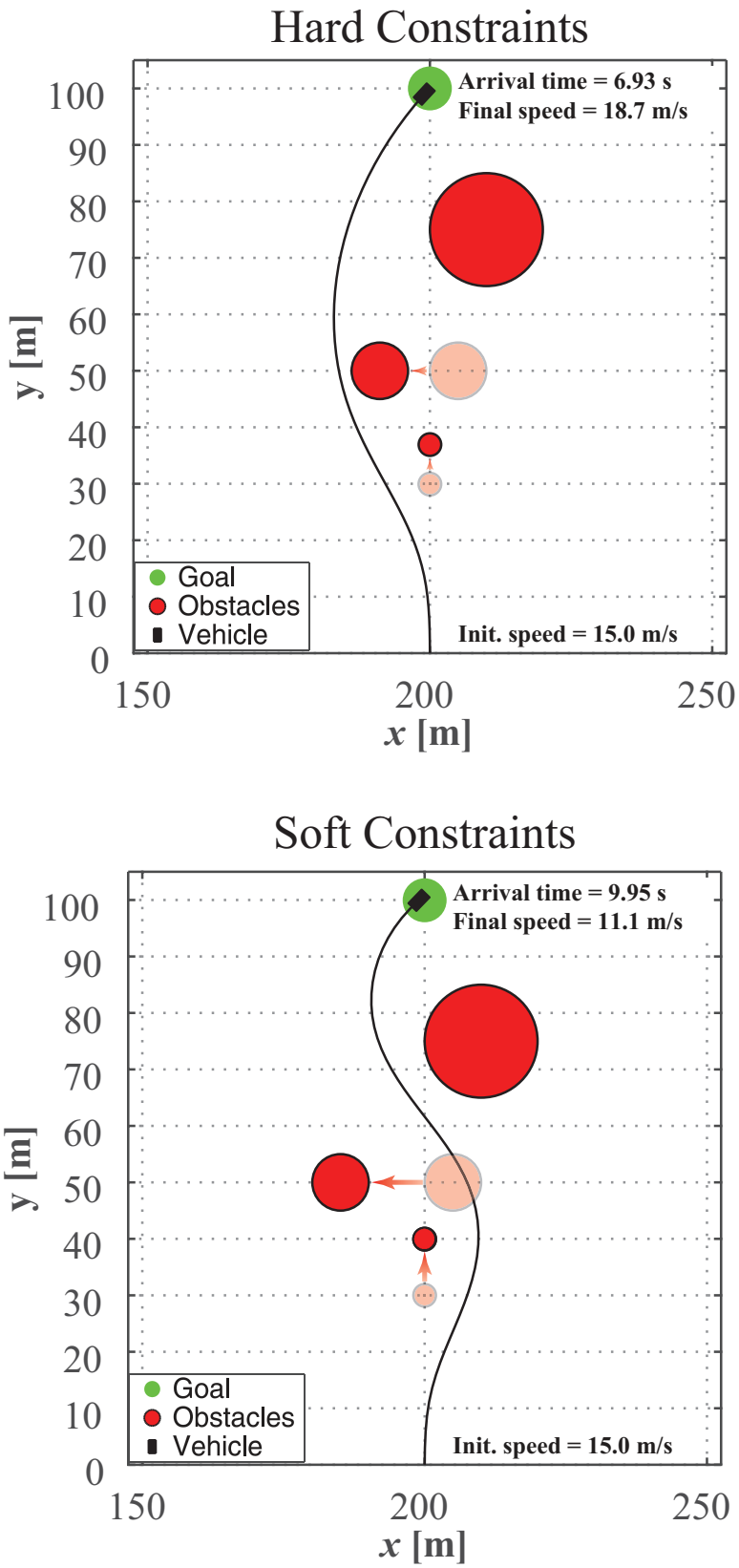


Figure 2.2: In Case 1, compared to the soft constraints approach, the hard constraints approach reduces the time-to-goal by 30%, number of iterations by 11% , and optimization time by 60%.

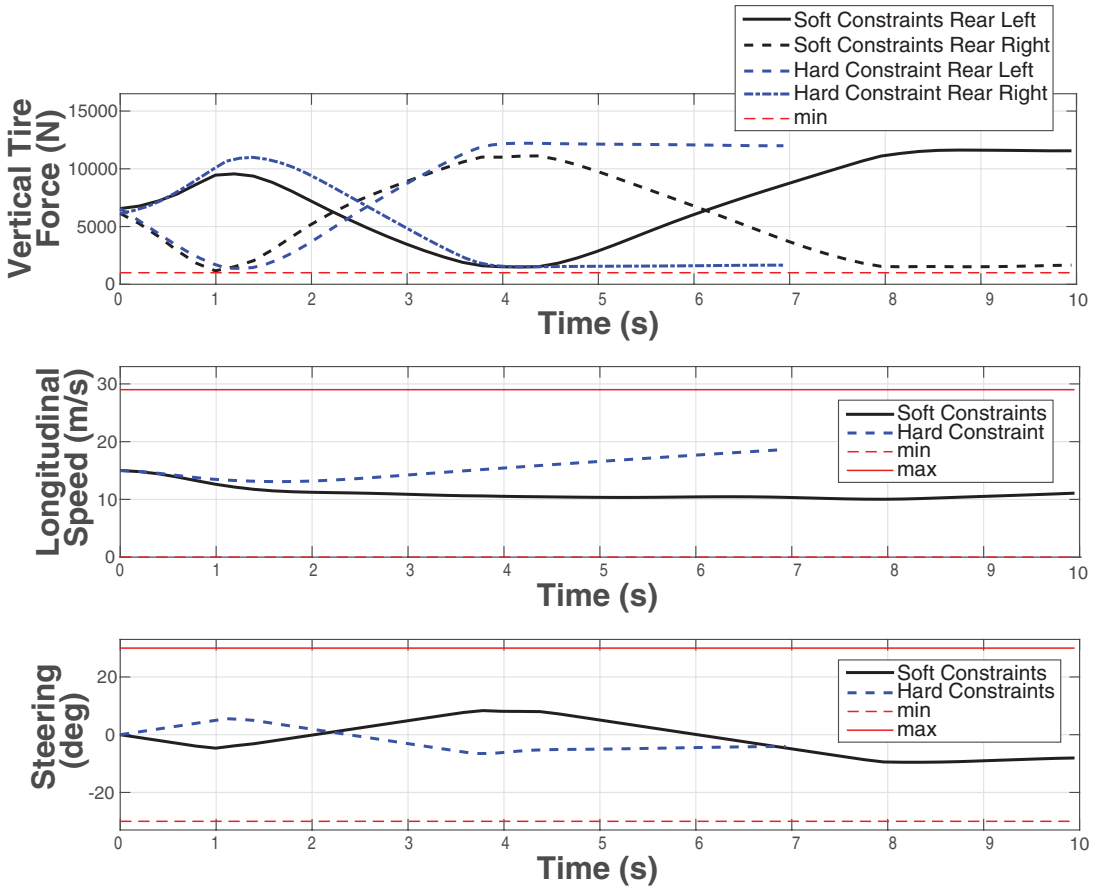


Figure 2.3: This figure shows three traces for Case 1; vertical tire force, optimized steering and optimized longitudinal velocity.

hard and soft constraints methods avoid all of the obstacles and successfully attain the goal position as shown in Fig. 2.4. To complete the mission, both vehicles follow very similar trajectories; this is likely due to the limited number of feasible paths for this particular scenario. As a result, time-to-goal is the same for both approaches and is 10 s. In Fig. 2.5, the vertical tire load, longitudinal speed and steering angle are shown for both methods. Between about 2.5 s and 4 s, both vehicles are operating at the minimum vertical load (set to 1000 N for these tests). This corresponds to the extreme maneuvers that the vehicles make between Frame 2 and Frame 3 in the top and bottom traces in Fig. 2.4 to avoid colliding with obstacles.

It is also noted that for both the hard and soft constraints cases the solution identified is not feasible at the last step, because at the last time step the vehicle accelerates above the maximum allowable acceleration in order to satisfy the constraint that the vehicle is within a particular distance (σ) in both the x and the y directions of the goal (Eq. (2.5)). This occurs only at the last time step and is an artifact of a discretization with a fixed time step as part of the solution strategy. Our preliminary investigations indicate that a variable time step implementation resolves this issue.

One of the major drawbacks of the soft constraints method is the amount of time that it takes to complete the optimization, which is 364.5 min and two orders of magnitude longer compared to the 3.21 min for the hard constraints formulation. This is likely due to the fact that when solving this problem using the soft constraints method for elliptical obstacles, there is no analytical expression for the distance from the vehicle to the nearest point on the ellipse and a numerical solution for this distance is needed. Without analytic expressions for the objective function, its gradient, the constraints and their Jacobian, IPOPT typically takes much longer to converge.

Additionally, collision avoidance is not guaranteed for a feasible solution using soft constraints. To demonstrate this, the formulation SC_1 is used to find a solution in Case 2. The optimization converges on a solution illustrated in Fig. 2.6 that drives

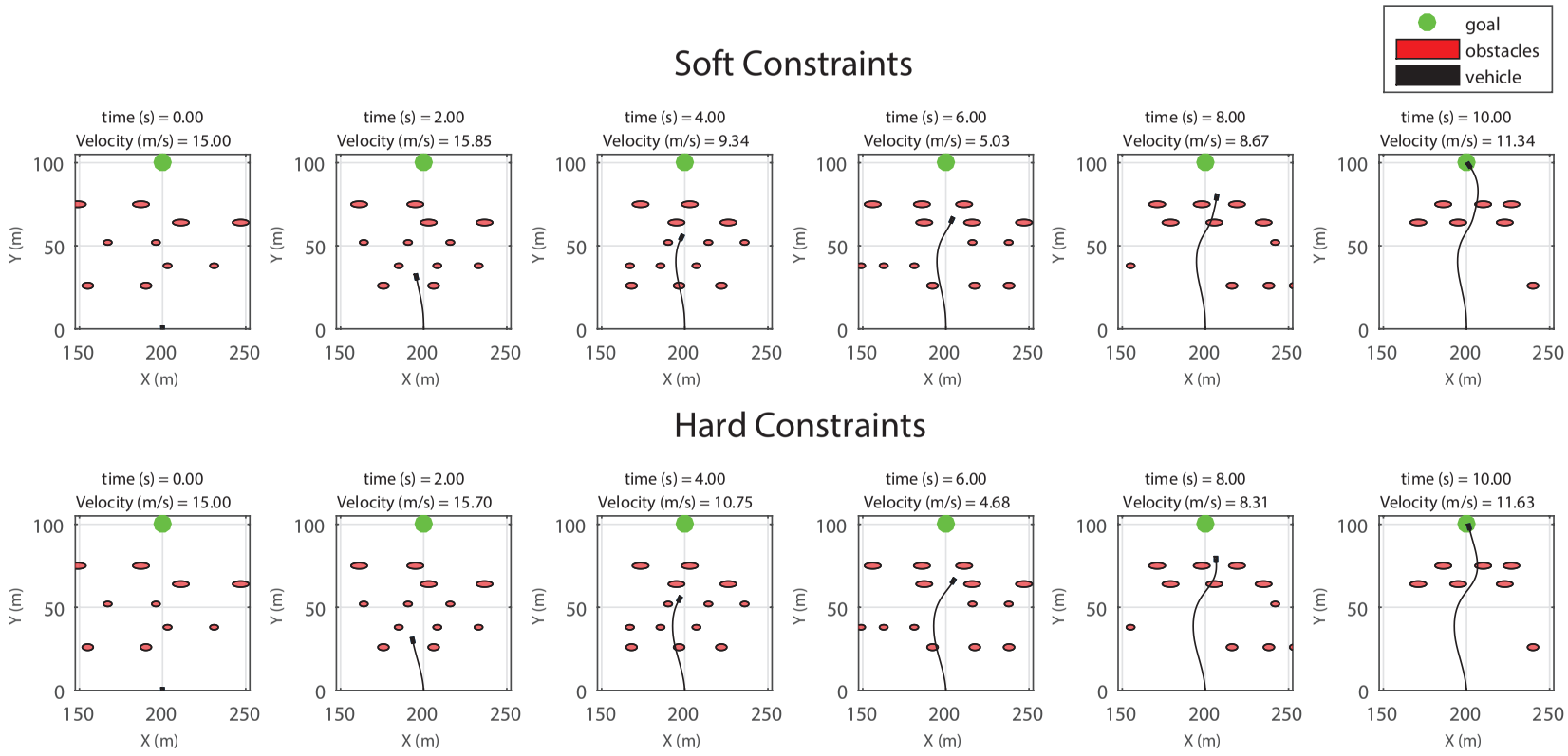


Figure 2.4: In Case 2, the hard and soft constraints approaches yield almost the same trajectory and time-to-goal. However, compared to the soft constraints approach, using hard constraints reduces optimization time by 99.1%.

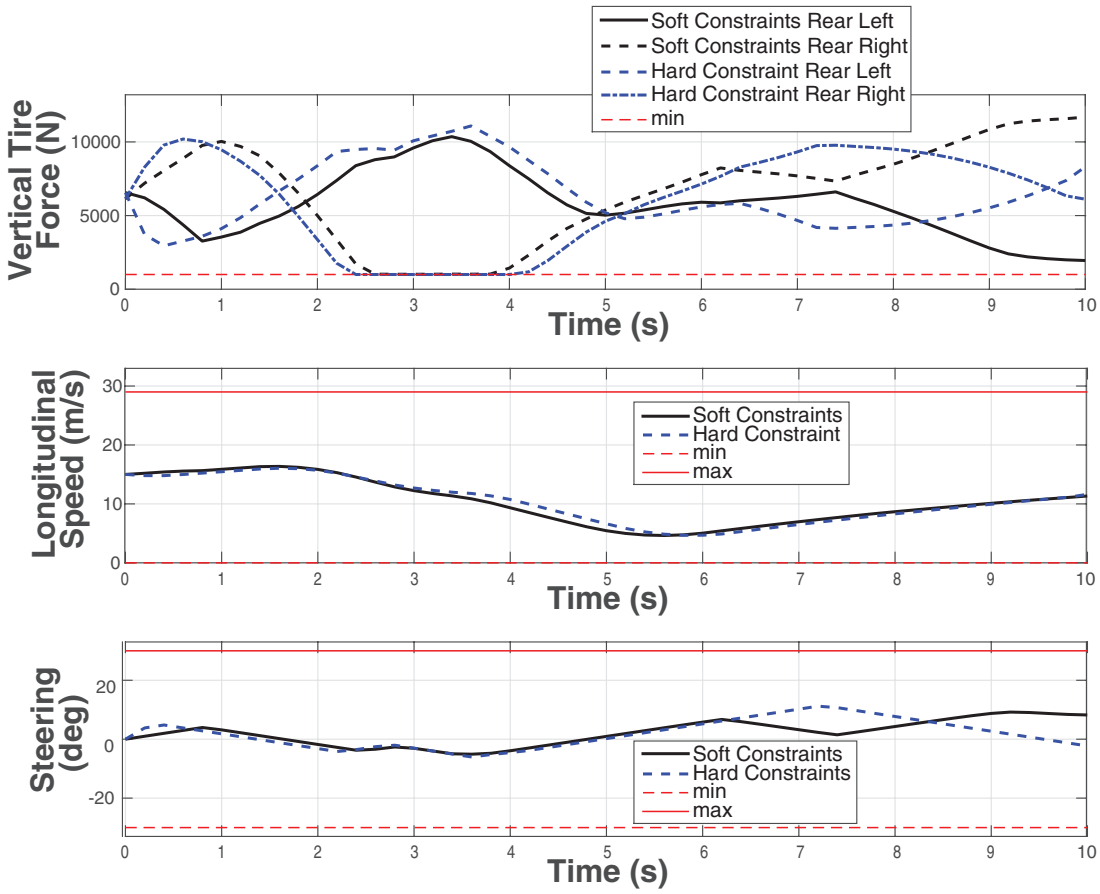


Figure 2.5: This figure shows three traces for Case 2; vertical tire force, optimized steering and optimized longitudinal velocity.

the vehicle into an obstacle at $t = 1.76$ s.

2.4 Conclusion

In this work, a hard constraints formulation is developed for moving obstacle avoidance in a large autonomous ground vehicle. To this end, a 3 DoF vehicle model is utilized in a single-level nonlinear optimization framework to find the optimal control signals (steering rate and longitudinal jerk) subject to constraints on both the dynamical limits of the vehicle and obstacle avoidance. This formulation is benchmarked against a soft constraints approach that also utilizes the same models, but relaxes the hard constraints on obstacle avoidance and augments the cost function with a term that promotes obstacle avoidance. Two comparative simulation case studies are given. It is found that both algorithms successfully avoid colliding with obstacles, however the proposed hard constraints formulation is deemed superior due to both a faster convergence time in optimization as well as obstacle avoidance performance. Future work includes: improvement of optimization times, identification of obstacles based off of LiDAR data, prediction of obstacles future path, using the 14 DoF model to close the loop, and experimental validation.

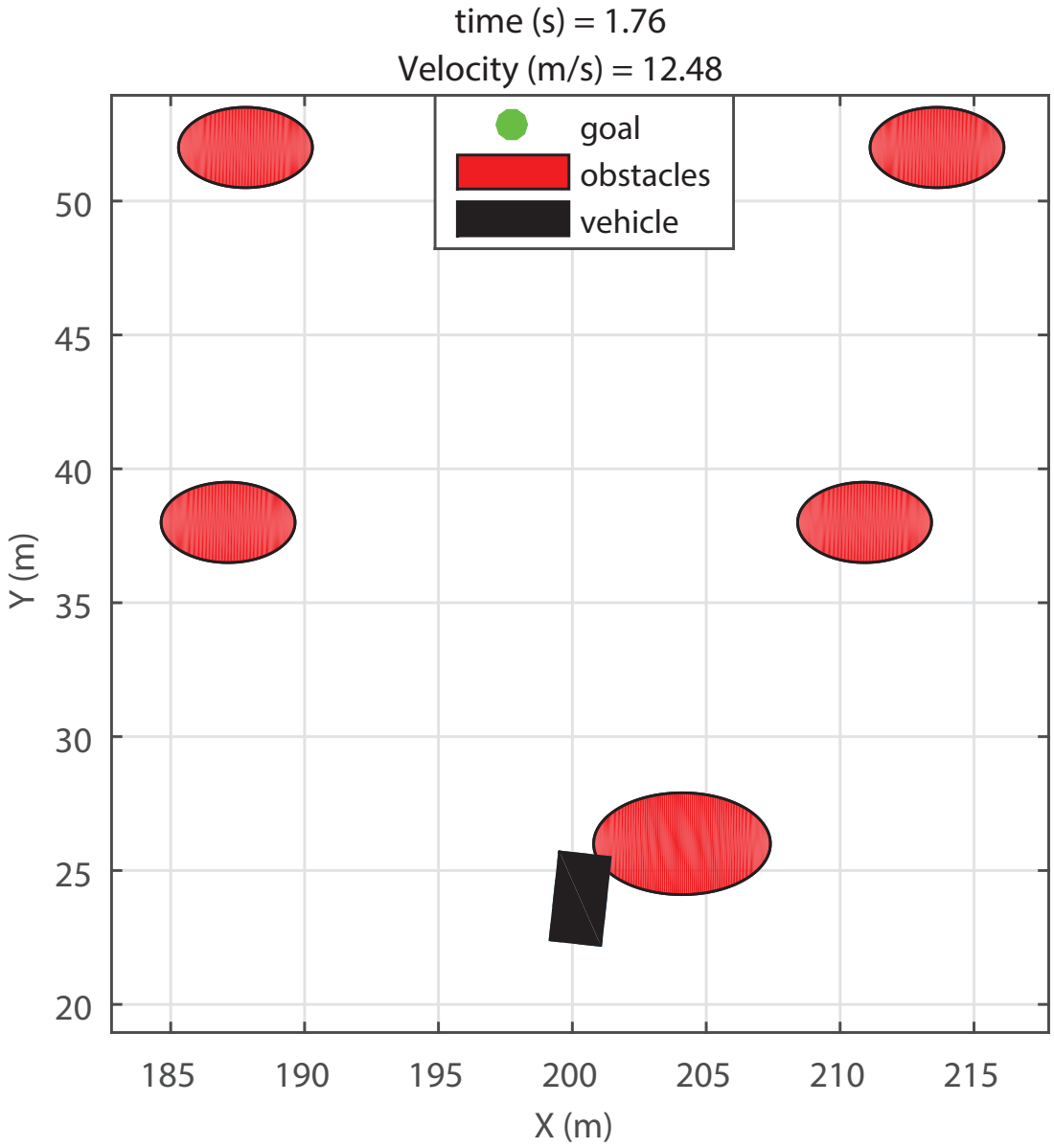


Figure 2.6: This figure demonstrates that if the soft constraints are not designed properly, the vehicle may crash into an obstacle . This particular example uses the SC_1 formulation and illustrates the solution at $t = 1.76$ s.

CHAPTER III

`NLOptControl`: A Modeling Language for Solving Optimal Control Problems

3.1 Introduction

Optimal control software packages that implement direct-collocation methods are used in a number of off-line [111, 125, 112, 73, 74, 64] and on-line [133, 44] applications as summarized in Table 3.1. The primary function of these packages is to directly transcribe a human modeler's formulation of an optimal control problem (OCP) into a nonlinear programming problem (NLP). A key challenge with this process is enabling human modelers (i.e., users) to easily formulate new and complex problems while producing an NLP that can be quickly solved by an external NLP solver. However, current direct-collocation-based optimal control software packages are generally either fast or easy to use, but not both. Thus, these package are not well suited for non-expert users trying to formulate complex problems for on-line applications, wherein speed is critical. Therefore, there is a need for a direct-collocation-based optimal control software package that is both fast and easy to use. In this chapter, an approach to bridging this gap is presented and incorporated into a new, open-source optimal control modeling language called `NLOptControl` [37].

As seen in Table 3.1, some of the most well-known optimal control software pack-

	Applications						Properties		
	Off-line			On-line					
Software	*Chemical*	*Space vehicle*	*Medical*	*Air vehicle*	*Ground vehicle*	*Robot*	*Open-source*	*Easy to use*	*Fast*
GPOPS-ii	[111]	[111, 125]				[55]†	✗	✓	✗
PROPT	[112]	[112]	[112]				✗	✓	✗
GPOCS		[73]		[5]†			✓	✓	✗
DIDO		[74]		[5]†			✗	✗	✗
ACADO					[44]		✓	✗	✓
CasADi	[64]					[133]	✓	✗	✓
Custom					[92, 38]†		✗	✗	✗
`NLOptControl`					[37]		✓	✓	✓

Table 3.1: Landscape of direct-collocation-based optimal control software focusing on their applications and properties. † indicates that the software is too slow for use the on-line application.

ages (*GPOPS-ii* , *PROPT* , *DIDO*) are closed-source and often require a licensing-fee. These drawbacks limit their research value, since they are not freely available to the entire research community, results may be difficult to reproduce, and if the details of the underlying algorithms cannot both be seen and modified, then open validation and development of the these algorithms is not possible [120, 8]. Fortunately, several noteworthy open-source optimal control software packages exist. For completeness, this chapter does not limit its discussions to these open-source packages.

Optimal control packages with an algebraic syntax that closely resembles the Bolza form of OCPs [78] are categorized as easy to use. It is noted that there are other design features that affect ease of use; for instance, not having a built-in initialization algorithm [11] reduces ease of use, but these aspects of ease of use are not addressed in this chapter. Table 3.1 shows that this work categorizes the direct-collocation-based optimal control software packages *GPOPS-ii* [111], *PROPT* [112] and *GPOCS* [119] as easy to use and *CasADi* [3] and *DIDO* [123] as not easy to use.

For ease of use, modeling languages should have a syntax that closely resembles the

class of problems for which they have been designed. Modeling languages like AMPL and GAMs are not embedded in a pre-existing computational language, which allows for syntactical flexibility, when developing them. However, this approach (1) makes development of the modeling language difficult and time-consuming, and (2) does not directly expose users to the breath of features available in a computational language such as *C++* or *MATLAB*. For these reasons, modeling languages are often embedded in a pre-existing computational language.

It can be difficult to establish a syntax for the modeling within the syntactical confines of a pre-existing computational language. To overcome this issue, operator overloading can be used. For instance, a multiple-shooting method based optimal control software package called *ACADO* [65] uses operator overloading to allow its user to define an OCP using symbolic expressions that closely resemble the actual mathematical expressions of the problem. However, a naive implementation of operator overloading can lead to performance issues [96]. Additionally, Moritz Dielhl, a researcher who developed *ACADO* and MUSCOD-II, later acknowledges that, *ACADO* Toolkit [65], DIRCOL [137], DyOS [124], and MUSCOD-II [29] restrict the problem formulations, particularly for users not involved with the development of these tools [3]. The above acknowledgment is included in a chapter [3] that introduces *CasADi*. *CasADi* allows users to formulate OCPs with fewer restrictions that *ACADO*. However, *CasADi* requires that users write the code for the transcription methods. Transcription methods are a general class of numerical methods used to approximate continuous-time OCPs; a direct-collocation method is a type of transcription method. *CasADi* lets users to code their own transcription methods to avoid creating a "black box" OCP solver that is only capable of solving restrictive formulations, as with *ACADO*. While this approach may be pedagogically valuable for users, it can lead to bugs and long development time [83] and it makes *CasADi*'s syntax not closely resemble OCPs. For these reasons, this chapter does not categorize *CasADi* as easy to use. On similar

grounds, *DIDO* is not categorized as easy to use.

For safety in on-line applications, the trajectory needs to be provided to the plant in real-time. An on-line optimal control example is a nonlinear model predictive control (NMPC) problem. Real-time is achieved when the NLP solve-times are all less than the chosen execution horizon. Otherwise, the low-level controllers will not have a trajectory to follow. Despite the need for small solve-times (i.e., speed), several implementations of direct-collocation methods within the *MATLAB* computational language are not able to achieve solve-times that are less than the execution horizon for a number of NMPC applications. As seen in Table 3.1, *GPOCS*, *GPOPS-ii*, and custom *MATLAB* software are not fast enough for NMPC applications in aircraft [5], robot [55], and UGV [92, 38] systems, respectively. On the other hand, *CasADi*, which is written in *C++*, is fast enough for an NMPC application in a robot system [133]. Given this practical limitation, this chapter will now discuss why some direct-collocation-based optimal control packages are fast while others are slow.

As seen in Table 3.1, this work categorizes *GPOPS-ii*, *PROPT*, *GPOCS*, and *DIDO* as slow and *CasADi* as fast. If a package uses sparse automatic differentiation methods implemented in a computation language that approaches the speeds of *C*, it is categorized as fast; the reasoning for this categorization is explained below.

The main algorithmic step in direct method based numerical optimal control is solving the NLP. The solve-time for this step consists of two major parts: (1) the time spent running optimization algorithms within the NLP solver, and (2) the time spent evaluating the nonlinear functions and their corresponding derivatives. Fortunately, low-level algorithms, which are available within several prominent NLP solvers, such as *KNITRO* [18], *IPOPT* [4], and *SNOPT* [56], can be used to reduce the time associated with running the optimization algorithms. The second component is discussed here in terms of current direct-collocation-based optimal control software packages.

The speed of direct-method-based optimal control software depends on the speed

of the differentiation method within the computational language in which it is implemented. *GPOPS-ii* uses a sparse finite difference method [110] to calculate the derivatives using the *MATLAB* computational language. However, finite difference methods, like the sparse finite difference method, are not only slow, but they are also inaccurate [130]. In addition to this, the dynamically-typed *MATLAB* computational language is typically slow in comparison to statically-compiled languages such as *C* and *Fortran*. Since *GPOPS-ii* uses a slow differentiation method within a relatively slow computational language, it is categorized as slow. *PROPT* uses either symbolic- or forward-automatic differentiation to calculate the derivatives using *MATLAB*. While *PROPT*'s methods are more accurate and generally faster than finite difference methods, they do not exploit the sparse structure of the Hessian matrices that is born from a direct-collocation method, like the sparse finite difference method in *GPOPS-ii*. Given this computational limitation and the slow speed of *MATLAB*, this chapter considers *PROPT* to be slow as well. On the other hand, *CasADi* uses the star-coloring method [52] to exploit the sparse structure of the Hessian matrix and reverse automatic differentiation implemented in *C++* [3]. Since *CasADi* employs a differentiation methods that is well suited for the sparse structure of the Hessian matrix and it is implemented in a fast computational language, *CasADi* is categorized as fast. On similar grounds, this chapter identifies *GPOCS* and *DIDO* as slow.

In sum, there is no direct-collocation-based optimal control software package is both fast and easy to use. *CasADi* is fast, but not easy to use; and *GPOPS-ii*, *PROPT*, and *GPOCS* are easy to use, but not fast. Thus, there is a need for a package that is both fast and easy to use.

This chapter investigates an approach for improving both speed and ease of use of optimal control software. As described in detail in Section 3.2, this approach uses recent advances in computational languages and differentiation methods in con-

trast to the computational languages and differentiation methods used by current direct-collocation-based optimal control software. Additionally, also unlike current direct-collocation-based optimal control software packages, this approach extends an optimization modeling language to include syntax for modeling OCPs. More specifically, this approach is as follows:

Approach

- For ease of use and speed, `NLOptControl` is embedded in the fast, dynamically-typed *Julia* programming language [13].
- For increased ease of use, `NLOptControl` extends the *JuMP* optimization modeling language [30], which is written in *Julia*, to include a natural syntax for modeling OCPs in Bolza form.
- For increased speed, `NLOptControl` uses the acrylic-coloring method [59] to exploit sparsity in the Hessian matrix and reverse-automatic differentiation through the *ReverseDiffSparse* package [121], which is also written in *Julia*.

Therefore, this work addresses the following research question: Can the above outlined approach improve speed and ease of use of direct-collocation-based optimal control software? This question is answered by comparing `NLOptControl`'s speed and ease of use to those of *PROPT*.

`NLOptControl` was released as a free, open-source software package in the summer of 2017 [37]. Since then, the literature has shown that `NLOptControl` is fast and easy to use. For speed, `NLOptControl` was leveraged to solve complex trajectory planning problems for an unmanned ground vehicle system in real-time — solving these types of problems in real-time using *MATLAB* was not feasible in prior work [92, 38]. For ease of use, `NLOptControl` was used to create a new optimal control based learning algorithm [85] without any help from the developers of `NLOptControl`.

The remainder of this chapter is organized as follows. Section 3.2 further describes `NLOptControl`'s approach to bridging the research gap. Section 3.3 describes the classes of off-line and on-line OCPs that can be solved using `NLOptControl`. Section 3.4 provides a brief background on numerical optimal control and a mathematical description of the direct-collocation methods implemented within `NLOptControl`. Section 3.6 provides an example that compares `NLOptControl`'s ease of use against *PROPT*'s and benchmarks `NLOptControl`'s speed against *PROPT*. Section 3.7 answers the research question and discusses further implications. Finally, Section 3.8 summarizes the work and draws conclusions.

3.2 Software ecosystem

Advances in computational languages, optimization modeling languages, and differentiation methods and tools made it possible to create `NLOptControl`. This section describes these software advances and shows how they can be leveraged to create a modeling language for a class of optimization problems.

3.2.1 Computational languages

Direct-collocation based optimal control software packages are embedded in either a statically- or a dynamically typed computational language. Dynamically typed languages enable users to quickly develop and explore new concepts, yet they are typically slow; statically typed languages sacrifice the user's productivity for speed. Recently, however, a dynamically typed computing language called *Julia* has become a popular alternative to the computing languages that the current optimal control software packages are embedded in. It has become popular, because it allows users to write high-level code that closely resembles their mathematical formulas, while producing low-level machine code that approaches the speed of *C* and is often faster than *Fortran* [13]. The claim that *Julia* is not only fast, but also easy to use, motivates the

investigation presented in this chapter. Specifically, this chapter investigates the ability of the *Julia* computational language to improve speed and ease of use for optimal control software.

3.2.2 Modeling optimization problems

In the late 1970's, researchers using optimization software were more concerned with the need to improve the software's ease of use than its speed [40]. Eventually, this concern led to the development a number of optimization modeling languages, such as *GAMS* [122] and *AMPL* [41]. The role of an optimization modeling language is to translate optimization problems from a human-friendly language to a solver-friendly language [132, 42]. In other words, optimization modeling languages do not solve optimization problems; they focus on modeling problems at a high-level and passing optimization problems to external low-level solvers, which are the NLP solvers and the differentiation tools in the context of this work. Similarly, in this work, the high-level problem is the NLP, given in Eqn. 3.1 - Eqn. 3.3 (i.e., the NLP model) as

$$\underset{z \in \mathbb{R}^n}{\text{minimize}} \quad f(z) \tag{3.1}$$

$$\text{subject to} \quad g(z) \leq 0 \tag{3.2}$$

$$h(z) = 0 \tag{3.3}$$

where the objective function $f : \mathbb{R}^n \to \mathbb{R}$, with n defined as the number of design variables; the inequality constraints $g : \mathbb{R}^n \to \mathbb{R}^e$; and the equality constraints $h : \mathbb{R}^n \to \mathbb{R}^q$, are all assumed to be twice-continuously differentiable functions [4, 10].

A number of standard optimization problem classes do not fit readily into the NLP model. In addition to this, translating these standard problem classes into the NLP model can require significant work. Thus, for users interested in simply modeling these standard problem classes, and not translating these problems into an NLP model, the

NLP model should be extended to include higher-level modeling languages for these standard problem classes. However, most optimization modeling languages are not designed to be extended in this fashion [42]. Because of this limitation, both the speed and ease of use of optimal control packages have suffered. *GPOCS*, *GPOPS-ii*, and *PROPT* are slow because the sparse-automatic differentiation methods — typically available through an optimization modeling language — are not available in *MATLAB*; so, these packages use less efficient differentiation methods. Additionally, since these packages are not built upon an existing NLP modeling language, the API tends to be overly flexible, which can lead to modeling errors [65].

JuMP [96], a recent optimization modeling language that is embedded in the fast, dynamically-typed *Julia* programming language [13], is designed to be extended to include new classes of optimization problems. *JuMP* extensions include: parallel multistage stochastic programming [66], robust optimization [31], chance constraints [97], and sum of squares [84]. Moreover, *JuMP* provides an interface for both the *KNITRO* and *IPOPT* NLP solvers as well the *ReverseDiffSparse* differentiation tool. *ReverseDiffSparse* [121] is also embedded in the *Julia* programming language and utilizes reverse automatic differentiation with the acrylic-coloring method [51] to exploit sparsity in the Hessian matrices. Research shows that the acrylic-coloring method is faster than the star-coloring method [51], which was used in the *CasADi* package [3]. These advances are leveraged to create an optimal control modeling language called `NLOptControl`.

3.2.3 Proposed software ecosystem

Fig. 5.1 presents `NLOptControl`'s software ecosystem and its function as an optimal control software package. In terms of this ecosystem, it is: embedded in *Julia*; extends *JuMP* to provide a natural syntax for modeling OCPs; leverages *ReverseDiffSparse*; and interfaces with *KNITRO*, *IPOPT*, and potentially other solvers to solve

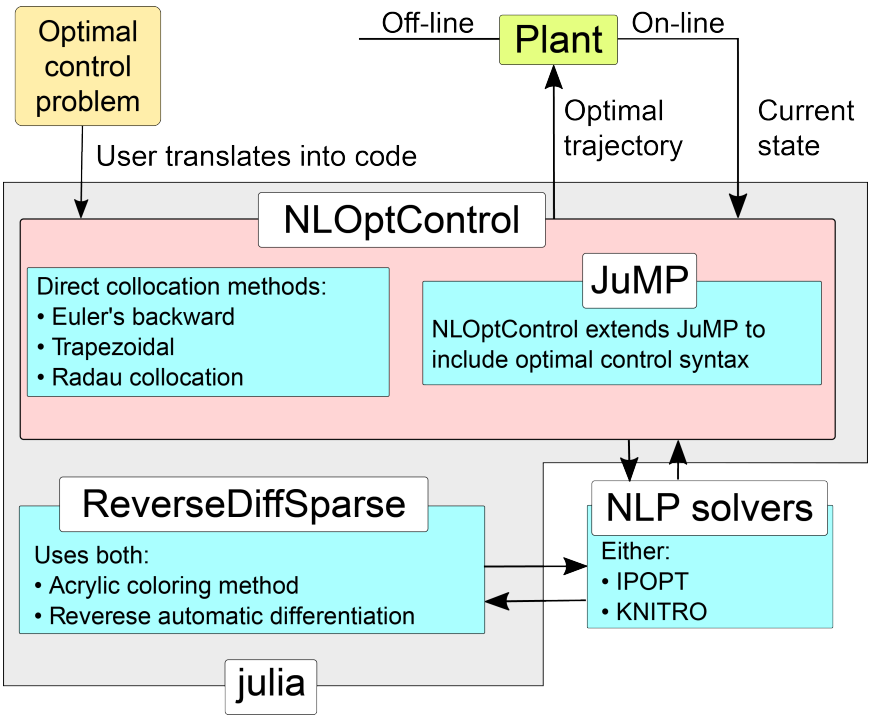


Figure 3.1: Proposed software framework for nonlinear OCPs.

the automatically formulated NLP problem. To use `NLOptControl`, users need only formulate their OCP into a syntax-based model of the OCP. This model is then approximated using one of the direct-collocation methods implemented in `NLOptControl`, which at the time of this writing include: the Euler's backwards, the trapezoidal, and the Radau collocation methods. After the model has been approximated, the software ecosystem solves this approximation to determine an optimal trajectory. This trajectory can then be followed using low-level controllers to control the plant for either an off-line or on-line tasks.

3.3 Scope of `NLOptControl`

`NLOptControl` is designed for modeling OCPs and solving them for either off-line or on-line applications. This section shows the types of problems that `NLOptControl` can model, and demonstrates `NLOptControl`'s visualization capabilities and salient design features for on-line applications (e.g., NMPC problems).

3.3.1 Modeling OCPs

An important class of optimization problems is the OCP. `NLOptControl` models single-phase, continuous-time, OCP in a Bolza form [78] that is tailored for NMPC problems and adds slack constraints on the initial and terminal states as

$$\underset{x(t),u(t),x_{0_s},x_{f_s}\,t_f}{minimize} \quad \mathcal{M}(x(t_0+t_{ex}),t_0+t_{ex},x(t_f),t_f) + \int_{t_0+t_{ex}}^{t_f} L(x(t),u(t),t)\,\mathrm{d}t + w_{s0}x_{0_s} + w_{sf}x_{f_s} \tag{3.4}$$

$$\text{subject to} \quad \frac{\mathrm{d}x}{\mathrm{d}t}(t) - F(x(t),u(t),t) = 0 \tag{3.5}$$

$$C(x(t),u(t),t) \leq 0 \tag{3.6}$$

$$x_0 - x_{0_{tol}} \leq x(t_0+t_{ex}) \leq x_0 + x_{0_{tol}} \tag{3.7}$$

$$x_f - x_{f_{tol}} \leq x(t_f) \leq x_f + x_{f_{tol}} \tag{3.8}$$

$$x_{min} \leq x(t) \leq x_{max} \tag{3.9}$$

$$u_{min} \leq u(t) \leq u_{max} \tag{3.10}$$

$$t_{f_{min}} \leq t_f \leq t_{f_{max}} \tag{3.11}$$

$$x_0 - x(t_0+t_{ex}) \leq x_{0_s} \tag{3.12}$$

$$x_0 + x(t_0+t_{ex}) \geq x_{0_s} \tag{3.13}$$

$$x_f - x(t_f) \leq x_{f_s} \tag{3.14}$$

$$x_f + x(t_f) \geq x_{f_s} \tag{3.15}$$

where t_0 is the fixed initial time, t_{ex} is the fixed execution horizon that is added to account for the non-negligible solve-times in NMPC applications, t_f is the free final time, t is the time, $x(t) \in \mathbb{R}^{n_{st}}$ is the state, with n_{st} defined as the number of states, and $u(t) \in \mathbb{R}^{n_{ctr}}$ is the control, with n_{ctr} as the number of controls. $x_{s0} \in \mathbb{R}^{n_{st}}$ and

$x_{sf} \in \mathbb{R}^{n_{st}}$ are optional slack variables for the initial and terminal states, respectively. The objective functional includes $\mathcal{M} : \mathbb{R}^{n_{st}} \times \mathbb{R} \times \mathbb{R}^{n_{st}} \times \mathbb{R} \to \mathbb{R}$ and $L : \mathbb{R}^{n_{st}} \times \mathbb{R}^{n_{ctr}} \times \mathbb{R} \to \mathbb{R}$, which are the Mayer and Lagrangian terms, respectively. Here $w_{s0}x_{0_s}+w_{sf}x_{f_s}$ is added to the Bolza form to accommodate slack variables on the initial and terminal conditions; this term is described in detail later in this section. $x_{0_s} \in \mathbb{R}^{n_{st}}$ and $x_{f_s} \in \mathbb{R}^{n_{st}}$ are vectors of weight terms on the slack variables for the initial and final state constraints. $F : \mathbb{R}^{n_{st}} \times \mathbb{R}^{n_{ctr}} \times \mathbb{R} \to \mathbb{R}^{n_{st}}$ and $C : \mathbb{R}^{n_{st}} \times \mathbb{R}^{n_{ctr}} \times \mathbb{R} \to \mathbb{R}^{p}$ denote the dynamic constraints and the path constraints, respectively; p is the number of path constraints. $x_0 \in \mathbb{R}^{n_{st}}$ and $x_f \in \mathbb{R}^{n_{st}}$ denote the desired initial and final states, respectively. $x_{0_{tol}} \in \mathbb{R}^{n_{st}}$ and $x_{f_{tol}} \in \mathbb{R}^{n_{st}}$ establish tolerances on the initial and final state, respectively. Constant upper and lower bounds on the state, control, and final time are included with Eqn. 3.9, Eqn. 3.10, and Eqn. 3.11, respectively. Finally, `NLOptControl` adds Eqn. 4.9 - Eqn. 4.12 to the Bolza form for optional slack constraints on the initial and terminal states.

`NLOptControl` is embedded in the *Julia* language and specializes *JuMP*'s syntax to better suit the domain of optimal control. *JuMP* leverages *Julia*'s syntactic macros [13] to enable a natural algebraic syntax for modeling optimization problems, without sacrificing performance or restricting problem formulations [96]. `NLOptControl` extends *JuMP* to include syntax for modeling OCPs in Boltza form in Eqn. 4.1 – Eqn. 3.11, with the option of including slack constraints on the initial and terminal states through Eqn. – Eqn. .

For a basic example of this syntax, `NLOptControl` is now used to model the

Bryson-Denham problem, which is given in mathematical form as

$$\begin{aligned}
&\underset{a(t)}{\textbf{minimize}} && \frac{1}{2}\int_0^1 a(t)^2 dt \\
&\textbf{subject to} && \dot{v}(t) = a(t), \qquad \dot{x}(t) = v(t), \qquad x(t) \leq \frac{1}{12} \\
& && v(0) = -v(1) = 1, \quad x(0) = x(1) = 0
\end{aligned}$$

The `define()` function is used to create a model object and define Eqn. 3.7 - Eqn. 3.10 as

```
n = define(numStates = 2, numControls = 1, X0 = [0.,1.], XF =
    ↪ [0.,-1.], XL = [0.,NaN], XU = [1/12,NaN], CL = [NaN,NaN], CU
    ↪  = [NaN,NaN])
```

where `n` is an object that holds the entire optimal control model, `numStates` and `numControls` are the number of states and controls, `X0` and `XF` are arrays of the initial and final state constraint, `XL` and `XU` are arrays of any lower and upper state bounds, `NaN` indicates that a particular constraint is not applied, and `CL` and `CU` are an arrays of any lower and upper control bounds.

The dynamic constraints in Eqn. 4.2 are then added to the model through the `dynamics` function as

```
dynamics!(n, [:(x2[j]), :(u1[j])])
```

where the ! character indicates that the model object `n` is being modified by the function. The elements of the array `:(x2[j])` and `:(u1[j])` represent $v(t)$ and $a(t)$; by default the state and control variables are `x1,x2,..` and `u1,u2,..`, but they can be changed. Differential equations must be passed within an array of *Julia* expressions (i.e., `[:(),:(),...,:()]`), and the index `[j]` must be appended to the state and control variables. `j` is used within `NLOptControl` to index particular time discretization points $\in [t_0 + t_{ex}, t_f]$.

The next step is to indicate whether or not the final time t_f is a design variable using the **configure** function as

```
configure!(n; (:finalTimeDV => true))
```

where `(:finalTimeDV=>true)` indicates that the final time is a design variable, which is the case for the Bryson-Denham problem. Additional options can be passed to the **configure** function. However, this chapter is not a tutorial; for a tutorial see NLOptControl's documentation [37].

At this point, any path constraints in Eqn. 4.3 can be added to the model using *JuMP*'s **@NLconstraint** macro. However, these constraints are not needed for this example.

Next, the objective function in Eqn. 4.1 is added to the model. To accommodate for a Lagrangian term, NLOptControl provides the **integrate** function—similar to the **dynamics** function, an expression must be passed and the `[j]` syntax must be appended to all state and control variables. For the Bryson-Denham, the objective functional is modeled as

```
obj = integrate!(n, :(0.5*u1[j]^2))
```

The *JuMP* macro **@NLobjective** is used to add the objective functional to the model as `@NLobjective(n.ocp.mdl,Min,obj)`. This problem is solved by passing the model `n` to the **optimize** function as

```
optimize!(n)
```

Visualization NLOptControl allows users quickly plot the solutions to their problems. For plotting —by default— NLOptControl leverages *GR* [63] as a backend, but it can be configured to utilize *matplotlib* [72] instead. The command `allPlots(n)` plots the solution trajectories for the states, controls, and costates[1].

[1] if `n.s.ocp.evalCostates` is set to **true**

Invoking this command to visualize the solution to the Bryson-Denham problem that is modeled above produces Fig. 3.2.

3.3.2 Nonlinear model predictive control

Fig. 4.1 depicts two ways that `NLOptControl` can be used to solve OCPs for NMPC applications; Fig. 3.3a neglects control delays and Fig. 3.3b accounts for them. This section describes these figures and discusses the design features that help `NLOptControl` users tackle NMPC problems.

Fig. 3.3a has three main components: the OCP, the plant, and the initialization block. Three inputs $\mathcal{G}$, $\mathcal{E}$, and x_0 are provided to the OCP to produce $u(t)$; $u(t)$ can be either a reference trajectory or control signals for the plant. In the case that $u(t)$ is a reference trajectory, then low-level controllers are added to the plant to allow it to track the trajectory.

Description: 3.3.1. *Goal information $\mathcal{G}$ includes the final desired state of the plant, which may not be equal to x_f. For instance, in an automated vehicle trajectory planning system, the goal range may be outside of the sensing range. In this case, the final desired state x_f may be near the boundary of the sensing range.*

Description: 3.3.2. *Environment information $\mathcal{E}$ includes any transient data. For example, this data may include the obstacle data that helps establish the constraints on obstacle avoidance for automated vehicle navigation problems.*

The plant can be either physical or virtual, but in either case is provided by the user. Because time can typically be allocated to initialize NMPC problems, the initialization block permits users to warm start their optimization problems so that the initial on-line solve time is much smaller. After initialization, at t_0, the first control signal $u(t)$ is sent to the plant and the first on-line OCP is solved. Each time an OCP-solve starts, t_0 is reset to the current time. An issue with this scheme is that

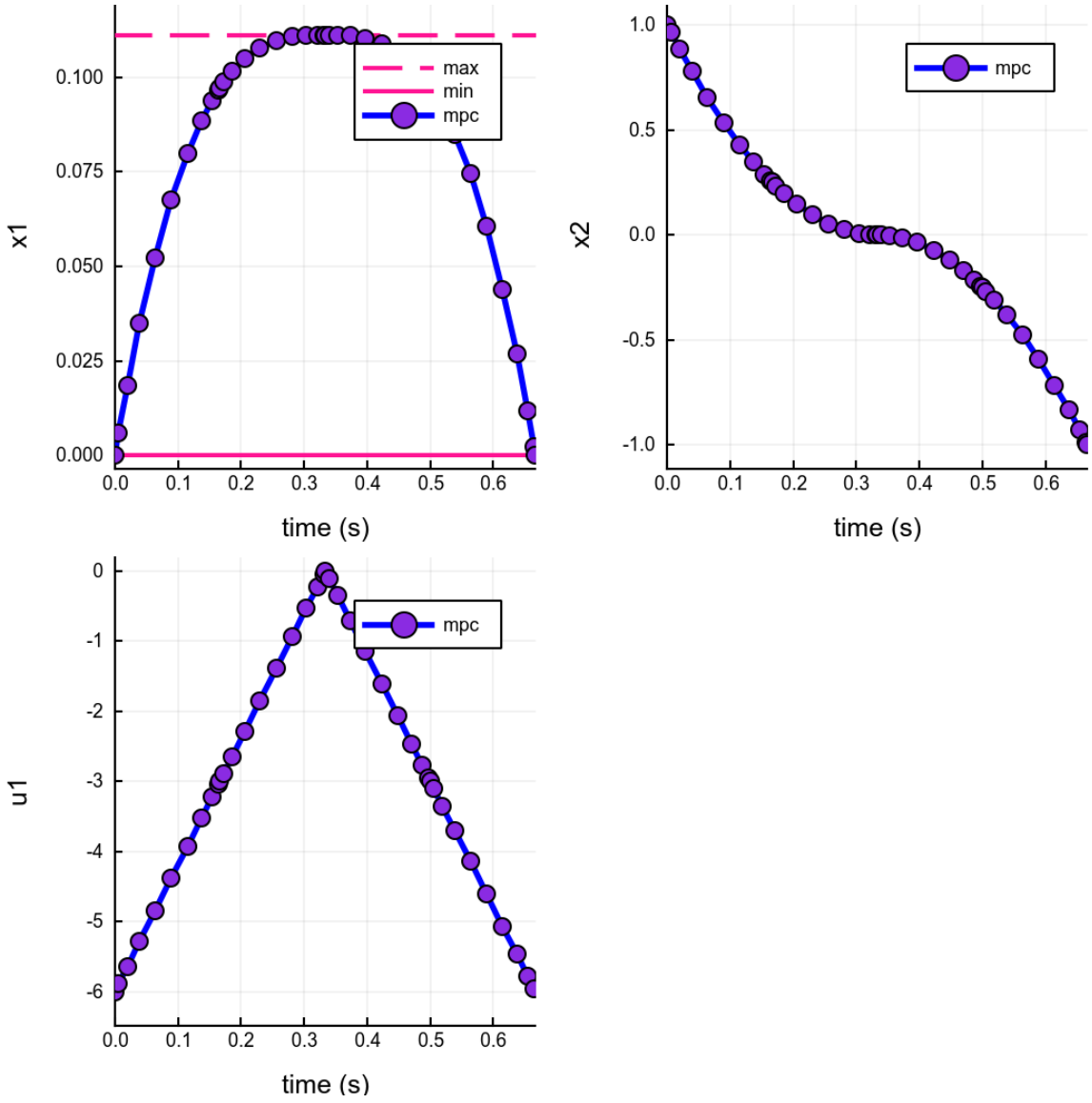
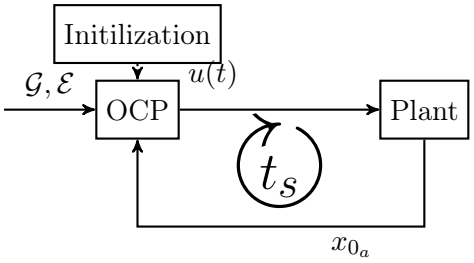


Figure 3.2: Output of `allPlots(n)` command after modeling and solving the Bryson-Denham problem using `NLOptControl`. Section `??` in the Appendices provides additional plots of the `NLOptControl`'s solution to the Bryson-Denham problem compared to the analytical solution, including the costates.

(a) Neglecting control delay t_s.



(b) Accounting for control delay t_s using a fixed execution horizon t_{ex} and a state prediction block.

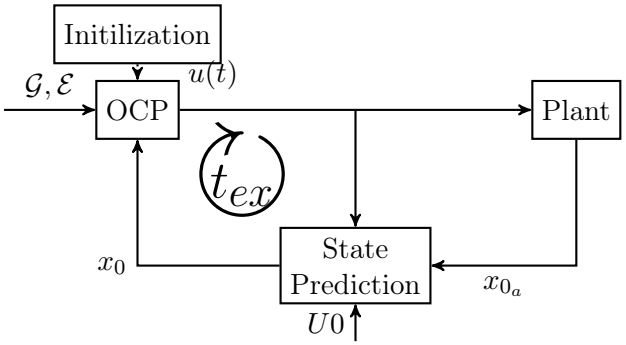


Figure 3.3: Nonlinear model predictive control framework available in `NLOptControl`.

it does not take into account the solve time (i.e., control delay t_s). That is, the initial state of the OCP is constrained to be the current state of the plant x_0 at the initial time t_0, so by the time the OCP has been solved t_s has elapsed, and the plant will have evolved to a new state. If this control delay is small relative to the time scale of the dynamics, then neglecting it will not compromise the robustness. However, if the control delay is relatively large, then it cannot be neglected.

Fig. 3.3b illustrates an approach that accounts for these control delays. This approach adds a block that predicts the plant state at the current time plus a fixed execution horizon $t_0 + t_{ex}$. The execution horizon t_{ex} can be chosen based on the a heuristic upper limit on the solve times; often solve times do not change drastically when solved in a receding-horizon with varying parameters for the initial conditions and path constraints. This approach avoids having to predict individual solve times t_s.

`NLOptControl` provides various functionality tailored for solving NMPC problems. The remainder of this section simultaneously describes these features and provides an example that uses `NLOptControl` to formulate an OCP and solve it in a receding horizon. To this end, consider the moon lander OCP [99], which is given in without slack constraints in Eqn. 3.16 as

$$
\begin{aligned}
\underset{a(t),\, t_f}{\text{minimize}} \quad & \int_0^{t_f} a(t)\mathrm{dt} \\
\text{subject to} \quad & \dot{x}(t) = v(t), \quad \dot{v}(t) = a(t) - g \\
& x(t_0) = 10, \quad x(t_f) = 0 \\
& v(t_0) = -2, \quad v(t_f) = 0 \\
& 0 \leq x(t) \leq 20, \quad -20 \leq v(t) \leq 20 \\
& 0 \leq a(t) \leq 3, \quad 0.001 \leq t_f \leq 400
\end{aligned}
\tag{3.16}
$$

where the $x(t)$ is the altitude, $v(t)$ is the speed, $a(t)$ is the thrust, $g = 1.5$ is the local gravitational acceleration, t_f is the final time. The objective is to minimize the thrust of the spaceship given the dynamic constraints, event constraints, control constraints, and final time constraints. Listing. III.1 shows the code needed to solve Eqn. 3.16 as an MPC problem. Line 1 creates a model `n` of the OCP with the initial and terminal state constraints, and the constant upper and lower bounds on the state and control variables. As is, the model `n` has low-tolerance hard constraints on the initial and terminal state conditions. However, these low-tolerance hard constraints can lead to infeasible problems and longer solve-times, especially when the loop is closed. That is, when the control drives the plant into an infeasible state space, an infeasible problem is engendered [127]; and typically, the solve-times increase as the problems become less-feasible. Therefore, an ability to easily adjust these low-tolerance constraints to high-tolerance hard constraints is desirable. As seen in Line 2, `NLOptControl`

Listing III.1: `NLOptControl` code needed to formulate and solve the moon lander as an MPC problem.

```
n = define(numStates = 2, numControls = 1, X0 = [10., -2], XF =
    ↪ [0., 0.], CL = [0.], CU = [3.])
defineTolerances!(n; X0_tol = [0.01, 0.005], XF_tol = [0.01,
    ↪ 0.005])
dynamics!(n,[:(x2[j]),:(u1[j]-1.5)])
configure!(n; (:finalTimeDV => true), (:xFslackVariables => true)
    ↪ , (:x0slackVariables => true))
obj = integrate!(n,:(u1[j]))
@NLobjective(n.ocp.mdl, Min, obj + 100*(n.ocp.x0s[1] + n.ocp.x0s
    ↪ [2] + n.ocp.xFs[1] + n.ocp.xFs[2]))
initOpt!(n)
defineMPC!(n; tex = 0.2, predictX0 = true)
function IPplant(n, x0, t, U, t0, tf)
  spU = linearSpline(t, U[:,1])
  f = (dx, x, p, t) -> begin
    dx[1] = x[2]
    dx[2] = spU[t] - 1.5
  end
  return DiffEqBase.solve(ODEProblem(f, x0, (t0, tf)), Tsit5()),
     ↪ [spU]
end
defineIP!(n, IPplant)
simMPC!(n)
```

enables this feature through the **defineTolerances** function. `X0_tol` and `XF_tol` are arrays that set the tolerances on the initial $x_{0_{tol}}$ and final states $x_{f_{tol}}$, respectively.

When going from low- to high-tolerance hard constraints on the initial and terminal states, slack constraints should also be added. Because, when using these high-tolerance constraints without slack constraints, there is nothing pushing the initial and terminal states away from the edge of the infeasible region. Thus, infeasible problems are just as likely to occur. Adding slack constraints on the initial and terminal state constraints helps to mitigate these infeasible problems. Before slack constraints are added to the model, slack variables must be added. The size of a slack variable corresponds to the size of the respective constraint violation [79]. As seen in

Line 4, `NLOptControl` allows such slack variable to be added using the **`configure`** ↪ function. `(:xFslackVariables=>true)` and `(:x0slackVariables=>true)` adds slack variables on the initial and final state constraint, respectively. Both the objective of the moon lander problem and the slack constraints are added to model as on Line 6. `n.ocp.x0s` and `n.ocp.xFs` are arrays holding the slack variables on the initial and terminal states, respectively, and all of the terms in w_{s0} and w_{sf} (in Eqn. 4.1) are set to 100—these weights are set large enough such that the respective constraint violations are nearly zero. On Line 7, `NLOptControl` warm starts the optimization using the **`initOpt`** function; the initialization block in Fig. 4.1 captures this step.

The **`defineMPC`** function adds several basic settings to the model `n`. `tex` is the value of the fixed execution horizon and `predictX0` is a bool, which, when set to **`true`**, indicates that the the framework in Fig. 3.3b is used. Thus, a prediction of the initial state needs to be made either by the user or using an internal model of the plant, which is added to `n`. In this simple example, the differential equations in Eqn. 3.16 govern the OCP, the plant, and the state prediction function. The plant and prediction model are defined by the `IPplant` function from Line 9 to Line 16 and passed to the model `n` using the **`defineIP`** function on Line 17. Here the `IPplant` function is showed for completeness, but its is not described in detail since it uses the well-documented *DifferentialEquations* package in *Julia* [116]. For safety and reduced time in experimental development, this initial step, i.e., making all of the models the same and running a simulation-based experiment, should be taken; especially when formulating more complex OCPs for practical NMPC applications.

Visualization The command **`mpcPlots`**`(n,idx)` plots the data for both plant and solution trajectories for the states and controls, the predicted initial state, and the optimization times, where `idx` is an integer representing the iteration number. For

instance, invoking this command to visualize data at the 4th and 15th iterations of the moon lander problem produces Fig. 3.4a and Fig. 3.4b, respectively; `NLOptControl` provides visualization functionality to combine the frames from all iterations into a single animation.

Section A.1.1 in the Appendices provides a plot `NLOptControl`'s closed-loop solution to the moon lander problem compared to the analytical solution.

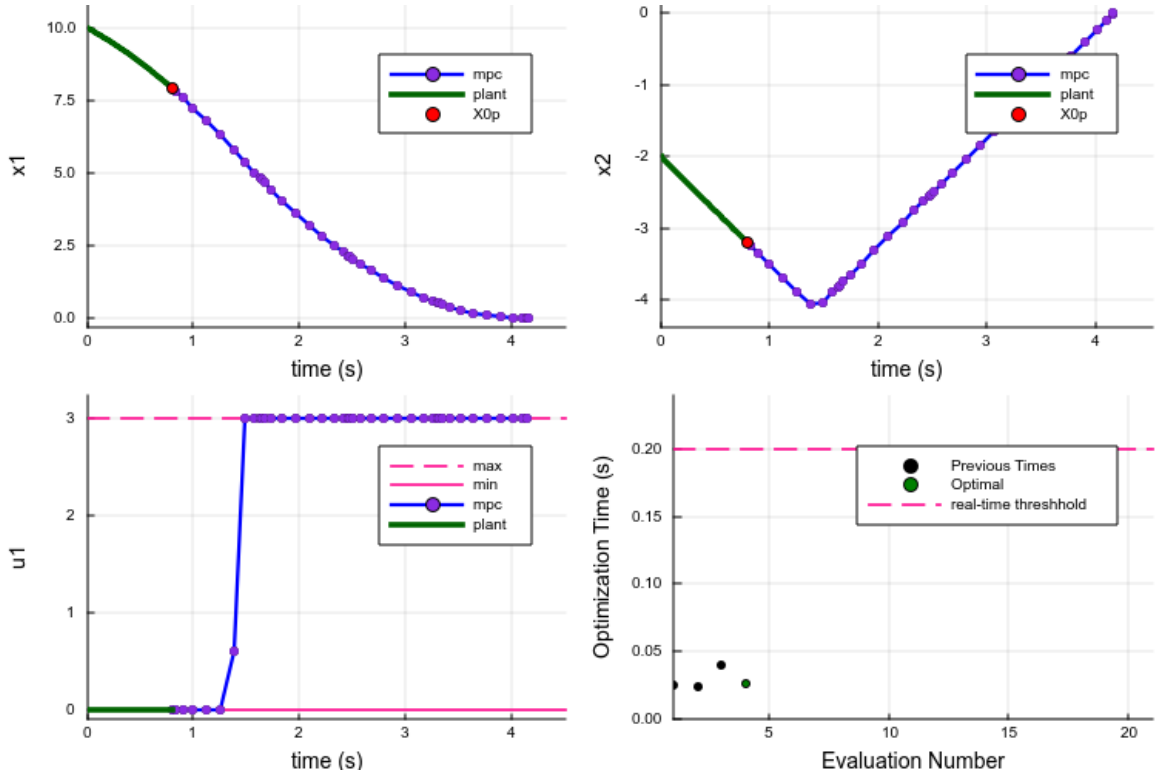
3.4 Numerical optimal control

This section provides an overview of numerical optimal control methods (i.e., transcription methods). The goal of this section is to motivate the choice of direct-collocation methods in `NLOptControl`, not to provide the reader with a complete description of numerical optimal control methods. Readers are referred to [9, 14, 10], for more comprehensive reviews on this subject. After these methods are discussed, the mathematics of the various direct-collocation methods as they are implemented in `NLOptControl` are provided.

3.4.1 Numerical optimal control overview

Tractable exact algorithms for solving OCPs suitable for practical applications do not exist; thus, numerical methods are used [108]. Numerical methods for solving OCPs (i.e., trajectory optimization problems) are generally broken into two categories: indirect and direct methods. Indirect methods seek the root of the necessary conditions for optimality [114] while direct methods seek the extrema of the cost functional [9]. Compared to direct methods, indirect methods produce better error estimates [78] and require less preliminary work to determine optimality [47]. However, indirect methods have several disadvantages: the necessary conditions must be derived [67]; the incorporation of path constraints requires an a priori estimation of the sequence of constrained/unconstrained singular arcs; and a guess needs to be

(a) Output of `mpcPlots(n,4)` command.



(b) Output of `mpcPlots(n,15)` command.

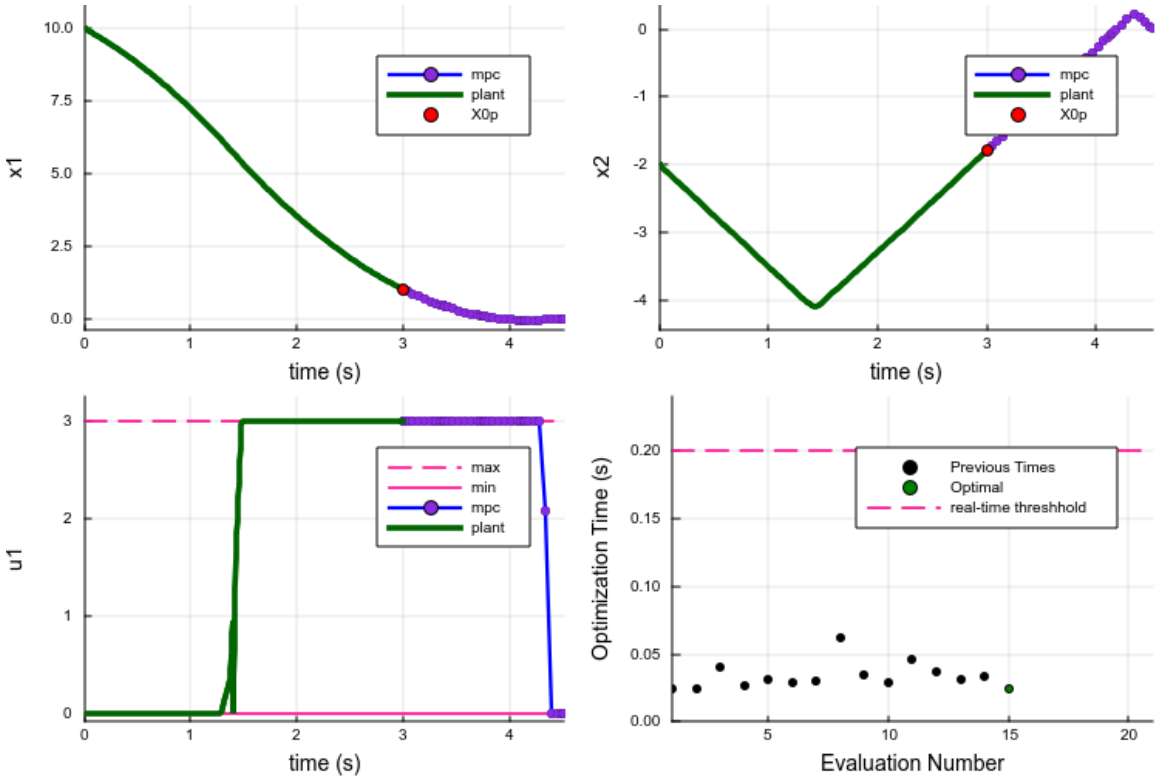


Figure 3.4: Closed-loop visualization of moon lander problem using `NLOptControl`.

made for the adjoint variables [10]. Due to these disadvantages, `NLOptControl` solves OCPs using direct methods.

Direct methods are broken into shooting methods [106], multiple shooting methods [15, 27], and direct-collocation methods. Shooting methods are not suitable for most practical applications because they do not work well when the number of variables is large [126]. The multiple shooting method is well suited to exploit parallel processing due to the structure of its formulation [10, 26]. However, there are several disadvantages to the multiple shooting method: an expensive numerical integration needs to be performed during each iteration of the NLP solve, it can be difficult to incorporate state inequalities [126], and using multiple integration steps reduces the sparsity of the Hessian and Jacobian matrices [12]. Direct-collocation methods overcome these issues by enforcing the dynamic state constraints within the NLP. While this results in a larger problem, there is no need to perform expensive integrations of the state dynamics between iterations, because constraints in the NLP enforce the state dynamics at the collocation points, the path constraints can be easily incorporated, and the sparsity in the derivative matrices is preserved.

3.4.2 Direct-collocation method overview

Direct-collocation methods are divided into three categories of polynomial approximation types: h-methods (or local methods) [62, 60, 9], p-methods (or global methods) [49, 35, 68], and hp-methods (a hybrid of the h- and p-methods) [109, 111, 24]. In an h-method, the dynamic state constraints are satisfied using local approximations; e.g., Euler's method or the trapezoidal method [62]. For h-methods, increasing the number and location of the collocation points [10, 111, 70] leads to convergence. However, a large number of points may be required for convergence, which can result in large solve-times [25]. p-methods can reduce the number of points needed for convergence, because they are more accurate than h-methods [33]. p-methods

approximate OCPs using global polynomials constructed by collocating the dynamics at Gaussian quadrature points [33]. p-methods were originally developed to solve problems in computational fluid dynamics [69] and since have been used in practice in optimal control. For instance, p-methods were used to rotate the International Space Station 180 degrees without using any propellant[2] [35]. A drawback with p-methods is that the Jacobian and Hessian matrices are much denser than with h-methods, which results in a larger NLP [23].

By construction, hp-methods help to mitigate the accuracy issues with h-methods and the NLP problem size with p-methods. Instead of using a single polynomial as with p-methods, hp-methods use multiple polynomials constrained to be connected to one another at the endpoints. This construction reduces the size of the NLP while maintaining accurate approximations [24].

3.4.3 Direct-collocation methods in `NLOptControl`

At the time of this writing, three direct-collocation methods are implemented in `NLOptControl`: two h-methods and one p/hp-method. The remainder of this section illustrates how these methods are implemented in `NLOptControl`.

3.4.3.1 h-Methods

Euler's backward method and the trapezoidal method are embedded in `NLOptControl`. However, before these h-methods are given, the h-discretization matrices used to approximate the continuous-time OCP are provided.

h-Discretization Matrices Consider that t is sampled at N evenly spaced discritization points $\in [t_0 + t_{ex}, t_f]$ and denote the result as the vector $\mathbf{T} = [T_1, \ldots, T_N]$. Then, for instance the $t_0 + t_{ex}$ and t_f are defined as T_1 and T_N, respectively. Denote

[2]The cost of the fuel saved was estimated at one million dollars and control of the space station orientation was accomplished using gyroscopes [74].

the state and control discretization matrices as

$$x(t)\Big|_{t=\mathbf{T}} = \mathbf{X}$$

and

$$u(t)\Big|_{t=\mathbf{T}} = \mathbf{U},$$

respectively. $\mathbf{X}[i]$ is the state at the i^{th} collocation point; thus, $\mathbf{X}[1]$ and $\mathbf{X}[N]$ index the values of the initial and final states, respectively. The control matrix is similarly defined; $\mathbf{U}[i]$ is the control at the i^{th} collocation point. Denote the minimum and maximum discretized state limit matrices as

$$x_{min}(t)\Big|_{t=\mathbf{T}} = \mathbf{X}_{min}$$

and

$$x_{max}(t)\Big|_{t=\mathbf{T}} = \mathbf{X}_{max},$$

respectively. Similarly, the minimum and maximum control limit matrices are denoted as

$$u_{min}(t)\Big|_{t=\mathbf{T}} = \mathbf{U}_{min}$$

and

$$u_{max}(t)\Big|_{t=\mathbf{T}} = \mathbf{U}_{max},$$

respectively.

Euler's Backward Method The dynamic constraints in Eqn. 4.2 are locally approximated at $(N-1)$ points defined by $\mathbf{T}[2:N]$. To accomplish this, $(N-1)\times n_{st}$

implicit constraints are added as shown in Eqn. 3.17

$$\begin{aligned} \mathbf{0} &= \mathbf{X}[i+1] - \mathbf{X}[i] - hF(\mathbf{X}[i+1], \mathbf{U}[i+1], \mathbf{T}[i+1]) \\ &= \boldsymbol{\eta}_i, \text{ for } i \in (1 : N-1) \end{aligned} \tag{3.17}$$

where h is the time-step size, which is determined by dividing the time span $(t_f - t_0 - t_{ex})$ by N.

The integral term in the cost functional in Eqn. 4.1 is approximated in Eqn. 3.18 as

$$I = h \sum_{i=1}^{N} L(\mathbf{X}[i], \mathbf{U}[i], T_i) \tag{3.18}$$

Trapezoidal Method Similar to Euler's backward method, the dynamic constraints in Eqn. 4.2 are locally approximated at $(N-1)$ points defined by $\mathbf{T}[2:N]$. To accomplish this, the $(N-1) \times n_{st}$ implicit constraints in Eqn. 3.19 are enforced with

$$\begin{aligned} \mathbf{0} &= \mathbf{X}[i+1] - \mathbf{X}[i] - \frac{h}{2}(F(\mathbf{X}[i], \mathbf{U}[i], \mathbf{T}[i]) + F(\mathbf{X}[i+1], \mathbf{U}[i+1], \mathbf{T}[i+1])) \\ &= \boldsymbol{\eta}_i, \text{ for } i \in (1 : N-1) \end{aligned} \tag{3.19}$$

Next, the integral term in the cost functional in Eqn. 4.1 is approximated in Eqn. 3.20 as

$$I = \frac{h}{2} \sum_{i=1}^{N} (L(\mathbf{X}[i], \mathbf{U}[i], T_i) + L(\mathbf{X}[i+1], \mathbf{U}[i+1], T_{i+1})) \tag{3.20}$$

Discrete OCP The h-method-based discrete OCP is given as

$$\underset{\mathbf{X},\,\mathbf{U},\,T_N}{\textbf{minimize}} \quad \mathcal{M}(\mathbf{X}[1], T_1, \mathbf{X}[N], T_N) + I \tag{3.21}$$

$$\textbf{subject to} \quad \boldsymbol{\eta} = \mathbf{0} \tag{3.22}$$

$$\mathbf{C}(\mathbf{X}, \mathbf{U}, \mathbf{T}) \leq \mathbf{0} \tag{3.23}$$

$$\boldsymbol{\phi}(\mathbf{X}[1], T_1, \mathbf{X}[N], T_N) = \mathbf{0} \tag{3.24}$$

$$\mathbf{X}_{min} \leq \mathbf{X} \leq \mathbf{X}_{max} \tag{3.25}$$

$$\mathbf{U}_{min} \leq \mathbf{U} \leq \mathbf{U}_{max} \tag{3.26}$$

$$t_{f_{min}} \leq T_N \leq t_{f_{max}} \tag{3.27}$$

where slack constraints can be included with the Mayer term in Eqn. 3.21 and Eqn. 3.23.

3.4.3.2 p-Methods

For generality, this chapter only describes hp-methods, since the single interval method (i.e., p-method) is merely the case where the number of intervals is equal to one.

3.4.3.3 hp-Methods

The form of Eqn. 4.1 – Eqn. 3.11 must be modified to directly transcribe the OCP into an NLP using hp-methods. To apply Gaussian quadrature the interval of integration must be transformed from $[t_0 + t_{ex}, t_f]$ to $[-1, +1]$. To accomplish this, $\tau \in [-1, +1]$ is introduced as a new independent variable and a change of variable, for t in terms of τ using the affine transformation, $t = \frac{t_f - t_0 - t_{ex}}{2}\tau + \frac{t_f + t_0 + t_{ex}}{2}$. Then, the interval $\tau \in [-1, +1]$ is divided into a mesh of K intervals to accommodate for multiple intervals. With this, as in [25], an array of mesh points $(M_0, \ldots, M_K)$ for

the boundaries of these intervals is defined, which satisfy

$$-1 = M_0 < M_1 < M_2 < \cdots < M_{K-1} < M_K = 1$$

Denote the continuous-time variables for the state and control are on each mesh interval, $k \in (1, \ldots, K)$, by the arrays $x^{(k)}(\tau)$ and $u^{(k)}(\tau)$, respectively. Next, denote arrays of continuous-time variables for both the minimum and maximum state and control limits on each mesh interval, $k \in (1, \ldots, K)$, as $x^{(k)}_{min}$, $x^{(k)}_{max}$, $u^{(k)}_{min}$, and $u^{(k)}_{max}$, respectively. The state continuity between the mesh intervals is ensured with the constraint $x^{(k)}(M_k) = x^{(k+1)}(M_k)$ for $k = (1, \ldots, K-1)$ [24]. Similar to [111], this constraint is enforced programatically by making $x^{(k)}(M_k)$ be the same variable as $x^{(k+1)}(M_k)$. To continue to describe the hp-method implemented in `NLOptControl`, the hp-discretization matrices are defined, which hold the discrete-time values of the approximation to continuous-time problem.

hp-Discretization Matrices First an array of time discretization vectors, $\boldsymbol{\tau}^{(k)} = [\tau^k_1, \ldots, \tau^k_{N^k}]$, is defined by evaluating the continuous functions at N^k specified τ's $\in [M_{k-1}, M_k)$ for $k \in [1, \ldots, K]$, where N^k notates the number of collocation points in mesh interval k; for instance, $\tau^1_1 = -1$. Let

$$\mathbf{N} = [N_1, N_2, \ldots, N_k, \ldots, N_{K-1}, N_K]$$

denote an array that holds the number of collocation points within each mesh interval, where N^k can be adjusted according to the desired level of fidelity for the k^{th} mesh interval. For $k \in [1, \ldots, K]$, denote the state and control discretization matrix arrays as

$$x^{(k)}(\tau)\Big|_{\tau=\boldsymbol{\tau}^{(k)}} = \mathbf{X}^{(k)}$$

and

$$u^{(k)}(\tau)\Big|_{\tau=\boldsymbol{\tau}^{(k)}} = \mathbf{U}^{(k)},$$

respectively. Next, denote the minimum and maximum discretized state limit matrix arrays as

$$x^{(k)}_{min}(\tau)\Big|_{\tau=\boldsymbol{\tau}^{(k)}} = \mathbf{X}^{(k)}_{min}$$

and

$$x^{(k)}_{max}(\tau)\Big|_{\tau=\boldsymbol{\tau}^{(k)}} = \mathbf{X}^{(k)}_{max},$$

respectively. Similarly, the minimum and maximum control limit matrices are defined as

$$u^{(k)}_{min}(\tau)\Big|_{\tau=\boldsymbol{\tau}^{(k)}} = \mathbf{U}^{(k)}_{min}$$

and

$$u^{(k)}_{max}(\tau)\Big|_{\tau=\boldsymbol{\tau}^{(k)}} = \mathbf{U}^{(k)}_{max},$$

respectively.

To approximate the modified OCP that is modified for hp-methods, `NLOptControl` builds on the work done in [49, 48, 110], which was implemented in *GPOPS-ii* [111]. Specifically, `NLOptControl` implements the Legendre-Gauss-Radau quadrature collocation method (Radau collocation method). For completeness, this section will briefly describe this method, but for a more thorough explanation, the reader is referred to the seminal work done in [111, 49, 48, 110].

Radau Collocation Method In hp-methods, the states are approximated within each mesh interval with a Lagrange polynomial as

$$x^{(k)}(\tau) \approx \sum_{j=1}^{N^k+1} \mathbf{X}[j]^{(k)} \mathcal{L}^{(k)}_j(\tau), \; k \in [1, .., K] \tag{3.28}$$

with

$$\mathcal{L}_j^k(\tau) = \prod_{\substack{l=1 \\ l \neq j}}^{N^k+1} \frac{\tau - \tau_l^k}{\tau_j^k - \tau_l^k},\ k \in [1, .., K] \tag{3.29}$$

$\mathcal{L}_j^k(\tau)$ is the (k^{th}, j^{th}) Lagrange polynomial within a basis of Lagrange polynomials defined by $j = (1, \ldots, N^k + 1)$ and $k = (1, \ldots, K)$, $\boldsymbol{\tau}^{(k)} = [\tau_1^k, \ldots, \tau_{N^k}^k]$ and is the k^{th} set of the LGR collocation points (also, called LGR nodes [1]), which are defined on the k^{th} mesh interval ($\tau \in [M_{k-1}, M_k)$). Then to approximate the entire state, M_k is added as a noncollocated point [49] for $k \in (1, \ldots, K)$.

The derivative of the state can then be approximated for each mesh interval as

$$\frac{\mathrm{d}x^{(k)}(\tau)}{\mathrm{d}\tau} \approx \sum_{j=1}^{N^k+1} \mathbf{X}[j]^{(k)} \frac{\mathrm{d}\mathcal{L}_j^{(k)}(\tau)}{\mathrm{d}\tau},\ k \in [1, .., K] \tag{3.30}$$

with

$$\frac{\mathrm{d}\mathcal{L}_j^{(k)}(\tau)}{\mathrm{d}\tau}\Big|_{\tau=\tau_j^k} = D_{ij}^k \tag{3.31}$$

where D_{ij}^k is an element of the $N^k \times N^{k+1}$ Legendre-Gauss-Radau differentiation matrix in the k^{th} mesh interval, as defined in [49].

Next, in order to approximate the integral of the Lagrange term in Eqn. 4.1, Gaussian-Legendre quadrature [50] is used as

$$\int_{t_0+t_{ex}}^{t_f} L(x(t), u(t), t)\, dt \approx \frac{t_f - t_0 - t_{ex}}{2} \sum_{k=1}^{K} \sum_{j=1}^{N^k} \frac{M_k - M_{k-1}}{2} w_j^k L(\mathbf{X}[j]^{(k)}, \mathbf{U}[j]^{(k)}, \tau_j^k; t_0+t_{ex}, t_f) \tag{3.32}$$

where $\boldsymbol{w}^{(k)} = [w_1^k, \ldots, w_{N_k}^k]$ is the k^{th} array of LGR weights[3].

Eqn. 3.32 is mathematically equivalent to the approximations made for the integral term in the cost functional in [24], but it is written in a slightly different form to reduce the computations needed within the NLP. Specifically, the $\frac{M_k - M_{k-1}}{2} w_j^k$ term

[3]To calculate both the LGR nodes and weights, `NLOptControl` leverages FastGaussQuadrature [105, 1], which uses methods developed in [61].

is calculated outside of the NLP, for $j \in (1, \ldots, N^k)$ and $k \in (1, \ldots, K)$. The result is stored in an array of vectors. Thus, the design variable t_f is removed from the summations in `NLOptControl`.

Discrete OCP The p-method-based discrete OCP is shown in Eqn. 3.33 - Eqn. 3.39 as

$$\underset{\mathbf{X}^{(k)},\, \boldsymbol{U}^{(k)},\, t_f}{\textbf{minimize}} \quad \mathcal{M}(\mathbf{X}[1]^{(1)}, t_0 + t_{ex}, \mathbf{X}[N_{K+1}]^{(K)}, t_K) + I \tag{3.33}$$

$$\textbf{subject to} \quad \sum_{j=1}^{N_k+1} \mathbf{X}_j^{(k)} D_{ij}^{(k)} - \frac{t_f - t_0 + t_{ex}}{2} \mathbf{f}(\mathbf{X}_i^{(k)}, \mathbf{U}_i^{(k)}, \tau_i^k; t_0 + t_{ex}, t_f) = \mathbf{0} \tag{3.34}$$

$$\mathbf{C}^{(k)}(\mathbf{X}[i]^{(k)}, \mathbf{U}[i]^{(k)}, \tau_i^k; t_0 + t_{ex}, t_f) \leq \mathbf{0} \tag{3.35}$$

$$\boldsymbol{\phi}(\mathbf{X}[1]^{(1)}, t_0 + t_{ex}, \mathbf{X}[N_{K+1}]^{(K)}, t_f) = \mathbf{0} \tag{3.36}$$

$$\mathbf{X}[i]_{min}^{(k)} \leq \mathbf{X}[i]^{(k)} \leq \mathbf{X}[i]_{max}^{(k)} \tag{3.37}$$

$$\mathbf{U}[i]_{min}^{(k)} \leq \mathbf{U}[i]^{(k)} \leq \mathbf{U}[i]_{max}^{(k)} \tag{3.38}$$

$$t_{f_{min}} \leq t_f \leq t_{f_{max}} \tag{3.39}$$

for $(i = 1, \ldots, N_k)$ and $(k = 1, \ldots, K)$

3.4.3.4 Transforming to an NLP

Depending on the method, either the discrete OCP in Eqn. 3.21 - Eqn. 3.27 or the discrete OCP in Eqn. 3.33 - Eqn. 3.39 is then transformed into a large and sparse NLP given by Eqn. 3.1 - Eqn. 3.3.

Now that design and methods of `NLOptControl` have been provided, the following two sections compares its ease of use and speed to existing commonly used optimal control software.

3.5 Evaluation description

The next section compares `NLOptControl` and *PROPT* in terms of ease of use and speed. This section describes the conditions under which these comparisons are made.

3.5.1 Ease of use

Claiming that a software package is easy to use is subjective; even with the definition provided for ease of use, i.e., syntax that closely resembles the underlying OCP. Therefore, the respective syntax in `NLOptControl` and *PROPT* needed to model the moon lander OCP, as given in Eqn. 3.16, is compared.

3.5.2 Benchmark

The conditions under which `NLOptControl`'s speed is benchmarked against *PROPT* include the benchmark problem, methodology, and setup.

3.5.2.1 Benchmark problem

An OCP suitable for an NMPC-based ground vehicle application is used to benchmark `NLOptControl` against *PROPT*. The purpose of this problem is to find the steering and acceleration commands that drive a kinematic bicycle model [118, 57] to a goal location ($x_g = 0$ m, $y_g = 100$ m) as fast as possible (i.e., in minimum time) while avoiding crashing into a static obstacle. The cost functional is shown in Eqn. 3.40 as

$$\underset{a_x(t),\, \alpha(t)}{\textbf{minimize}} \quad (x(t_f) - x_g)^2 + (y(t_f) - y_g)^2 + t_f \tag{3.40}$$

The dynamic constraints are shown in Eqn. 3.41 as

$$\begin{aligned} \dot{x}(t) &= u_x(t)\cos(\psi(t)+\beta(t)) \\ \dot{y}(t) &= u_x(t)\sin(\psi(t)+\beta(t)) \\ \dot{\psi}(t) &= \frac{u_x(t)\sin(\beta(t))}{l_b} \\ \dot{u}_x(t) &= a_x(t) \end{aligned} \tag{3.41}$$

where $x(t)$ and $y(t)$ are the position coordinates, $\psi(t)$ is the yaw angle, $u_x(t)$ is the longitudinal velocity, $\alpha(t)$ is the steering angle, $\beta(t) = \tan(\frac{l_a \tan(\alpha(t))}{l_a+l_b})^{-1}$, $l_a = 1.58$ m and $l_b = 1.72$ m are the distances from the center of gravity to the front and rear axles, respectively. The path constraints ensure that the vehicle avoids an obstacle, these constraints are shown in Eqn. 3.42 as

$$1 < (\frac{x(t)-x_{obs}}{a_{obs}+m})^2 + (\frac{y(t)-y_{obs}}{b_{obs}+m})^2 \tag{3.42}$$

where $x_{obs} = 0$ m and $y_{obs} = 50$ m denote the position of the center of the obstacle, $a_{obs} = 5$ m and $b_{obs} = 5$ m denote the semi-major and semi-minor axes, $m = 2.5$ m is the safety margin that accounts for the footprint of the vehicle. The event constraints ensure that the vehicle starts at a particular initial condition, these constraints are given in Eqn. 3.42 as

$$\begin{aligned} x(t_0) = 0 \text{ m}, \qquad y(t_0) = 0 \text{ m}, \quad \psi(t_0) = \frac{\pi}{2} \text{ rad} \\ u_x(t_0) = 15 \ \frac{\text{m}}{\text{s}}, \quad a_x(t_0) = 0 \ \frac{\text{m}}{\text{s}^2}, \quad \alpha(t_0) = 0 \text{ rad} \end{aligned} \tag{3.43}$$

That is the vehicle is traveling straight ahead at a constant velocity of 15 $\frac{\text{m}}{\text{s}}$. The

state and control bound constraints are given in Eqn. 3.44 as

$$\begin{aligned} -100 \text{ m} \leq x(t) \leq 100 \text{ m}, \qquad & -0.01 \text{ m} \leq y(t) \leq 120 \text{ m} \\ -2\pi \text{ rad} \leq \psi(t) \leq 2\pi \text{ rad}, \qquad & 5 \frac{\text{m}}{\text{s}} \leq u_x(t) \leq 29 \frac{\text{m}}{\text{s}} \\ -2 \frac{\text{m}}{\text{s}^2} \leq a_x(t) \leq 2 \frac{\text{m}}{\text{s}^2}, \qquad & \frac{-30\pi}{180} \text{ rad} \leq \alpha(t) \leq \frac{30\pi}{180} \text{ rad} \end{aligned} \tag{3.44}$$

The final time is constrained to be $0.001 \text{ s} \leq t_f \leq 50 \text{ s}$.

Solutions to Eqn. 3.40 - Eqn. 3.44 that are obtained in less than 0.5 s are deemed to be fast enough for real-time NMPC.

3.5.2.2 Benchmark methodology

Using the problem described above, a comprehensive benchmark is made between various solvers. A solver is defined by a particular combination of either `NLOptControl` or *PROPT* in conjunction with a particular direct-collocation method. The set of solvers S are listed in Table 3.2 as

Table 3.2: Set of solvers tested

Legend label	Description
$\texttt{NLOpt}_{\text{LGR}_1}$	`NLOptControl` with LGR nodes with a single interval
$\texttt{NLOpt}_{\text{LGR}_2}$	`NLOptControl` with LGR nodes with two intervals
$\texttt{NLOpt}_{\text{LGR}_4}$	`NLOptControl` with LGR nodes with four intervals
$\texttt{NLOpt}_{\text{E}}$	`NLOptControl` using Euler's method
$\texttt{NLOpt}_{\text{T}}$	`NLOptControl` using trapezoidal method
$PROPT_{\text{C}_1}$	*PROPT* with Chebyshev nodes with a single phase
$PROPT_{\text{C}_2}$	*PROPT* with Chebyshev nodes with two phases
$PROPT_{\text{C}_4}$	*PROPT* with Chebyshev nodes with four phases

Comparisons between the average solve-times of single interval/phase solvers (i.e., $\texttt{NLOpt}_{\text{E}}$, $\texttt{NLOpt}_{\text{T}}$, $\texttt{NLOpt}_{\text{LGR}_2}$, $PROPT_{\text{C}_1}$) and the multiple interval/phase solvers (i.e., $\texttt{NLOpt}_{\text{LGR}_2}$, $\texttt{NLOpt}_{\text{LGR}_4}$, $PROPT_{\text{C}_2}$, $PROPT_{\text{C}_4}$) must be considered in context. This is true because as the number of collocation points per interval/phase

is increased, the two interval/phase solvers (i.e., $\texttt{NLOpt}_{\text{LGR}_2}$ and $PROPT_{\text{C}_2}$) and the four interval/phase solvers (i.e., $\texttt{NLOpt}_{\text{LGR}_4}$ and $PROPT_{\text{C}_4}$) are solving problems that roughly two and four times larger than the single interval/phase solvers, respectively. However, there are advantages of these multi interval/phase solvers, as discussed previously, that may be more important than the decreases in solve times. Thus, these solvers are included in the comparison here for a more comprehensive comparison.

Comparisons between the average solve-times of the multiple interval solvers in `NLOptControl` and the multiple phase solvers in *PROPT* also require consideration. Ideally, the benchmark between *PROPT* and `NLOptControl` would include the same direct-collocation methods. Unfortunately, *PROPT* and `NLOptControl` do not have the same direct-collocation methods. As such, comparisons are made between single/multiple phase Chebyshev pseudospectral methods in *PROPT* and multiple single/interval LGR pseudospectral methods in `NLOptControl`. Unlike a multiple interval method, in a multiple phase method, between phases, the constraints can change and the optimal transition time can be determined. In this work, the constraints do not change and the final time is divided evenly by the number of phases to determine the transition time. By doing this, the OCPs formulated by the multiple phase and multiple interval methods have roughly the same size and level of complexity. Thus, comparisons between the two software packages can be made with this issue in mind.

Each solver s is used to solve a set of problems P. The benchmark problem is discretized over the range of collocation points $p = 2, 3, \ldots, 102$ per interval or phase to realize the set of problems P tested for each solver; a total of 101 different values of p (i.e., levels-of-fidelity or problems) are tested. Each test is performed three times to provide the data needed to calculate the average solve-time $t_{s,p}$ for the benchmark problem with a level-of-fidelity p using solver s. A polynomial is

interpolated through the (x, y) solution points and sampled at 200 points to determine if the solution drives the vehicle through the obstacle. If a collision is determined for a particular combination of solver s^* and level-of-fidelity p^*, then t_{s^*,p^*} is set to NaN; such solutions are not practically feasible.

Conducting many benchmark tests helps accurately rank the solvers. However, analyzing large sets of benchmark data can be overwhelming and the conclusions drawn from such analyses can be subjective. To help eliminate these issues, this work uses an optimization software benchmarking tool called performance profiles [32].

Performance profiles show the distribution function for a particular performance metric. Here, the performance metric is the ratio of the solver's average solve-time to the best average solver solve-time given as

$$r_{s,p} = \frac{t_{s,p}}{\min(t_{s,p} : s \in S)}$$

where this performance metric is calculated for each solver s at each level-of-fidelity $p = 2, 3, \ldots, 102$. If a solver does not solve a particular problem, then $r_{s,p}$ is set to r_M. r_M is chosen to be a large positive number; the choice of r_M does not effect the evaluation [32].

To assess a solver's overall performance on the set of problems, the cumulative distribution function for the performance ratio is defined as

$$\mathcal{P}_s(\Gamma) = \frac{1}{101}\text{size}(p \in P : r_{s,p} \leq \Gamma)$$

where $\mathcal{P}_s(\Gamma)$ is the probability that solver s can solve problem p within a factor Γ of the best ratio.

3.5.2.3 Setup

The setup is defined by the hardware platform and software stack. The results in this chapter are produced using a single machine running Ubuntu 16.04 with the following hardware characteristics; an Intel Core i7−4910MQ CPU @2.90GHz×8, and 16GB of RAM. For software, both `NLOptControl` 0.1.5 and *PROPT* use *KNITRO* 10.3 for the NLP solver with the default settings, except the maximum solve-time, which is set to 300 s.

3.6 Results

3.6.1 Ease of use

Listing. III.2 and Listing. III.3 show the respective syntax in `NLOptControl` and *PROPT* needed to model the moon lander OCP in Eqn. 3.16. Section A.1 in the Appendices shows `NLOptControl`'s and *PROPT*'s solutions compared to the analytical solution.

Listing III.2: `NLOptControl` code needed to formulate and solve the moon lander problem. The ! character indicates that the function is modifying the model.

```
n = define(numStates = 2, numControls = 1, X0 = [10, -2], XF =
    ↪ [0., 0.], XL = [0, -20], XU = [20, 20], CL = [0.], CU=[3.]);
dynamics!(n,[:(x2[j]), :(u1[j] - 1.5)]);
configure!(n;(:finalTimeDV => true));
obj = integrate!(n, :(u1[j]));
@NLobjective(n.ocp.mdl, Min, obj);
optimize!(n);
```

`NLOptControl` can model OCPs more succinctly than *PROPT*. `NLOptControl` models Eqn. 3.16 with 5 lines of code, while it takes *PROPT* 12 lines — there are two main reasons for this: (1) it takes *PROPT* 4 lines of code to include the initial

Listing III.3: *PROPT* code needed to formulate and solve the moon lander problem

```
toms t t_f
p = tomPhase('p', t, 0, t_f, 30);
setPhase(p);
tomStates x v
tomControls a
cbox = {0.001<=t_f<=400, 0<=icollocate(x)<=20, -20<=icollocate(v)
    ↪ <= 20, 0<=collocate(a)<=3};
ode = collocate({dot(x)==v, dot(v)==-1.5 + a});
cbnd = {initial(x == 10);initial(v == -2);final(x == 0);final(v
    ↪ == 0);};
x0 = {t_f == 1.5, icollocate({x == 0 v == 0}), collocate(a == 0)
    ↪ };
objective = integrate(a);
options = struct;
prob = sym2prob(objective, {cbox, ode, cbnd}, x0, options);
result = tomRun('knitro', prob, 1);
```

and final state conditions, and the upper and lower limits of the states and controls, while this is accomplished with a single line of code, with `NLOptControl`, and (2) several of *PROPT*'s features are required, while in `NLOptControl` they are optional; these features include an initial guess, an options structure, and the naming of the state and control variables. Additionally, *PROPT* has more verbose syntax than `NLOptControl` — *PROPT*'s `collocate()`, `initial()`, and `final()` functions require many characters per line of code.

3.6.2 Speed

The performances of the solvers in Table 3.2 are now examined on the set of problems realized by various discretizations of Eqn. 3.40 – Eqn. 3.44, as described in Sec. 3.5. The results for these examinations are in Fig. 3.5a and Fig. 3.5b. Fig. 3.5a shows the performance, or average solve-times $t_{s,p}$, for each solver s on each problem p. Fig. 3.5b shows the performance profiles for all of the solvers in four ranges of interest for Γ. Each range is on a separate plot. The purpose of this section is to (1) show the raw benchmark data in Fig. 3.5a and (2) provide an objective analysis

of this data in Fig. 3.5b. In the following section, this information will be used to draw conclusions regarding the speed of `NLOptControl` and the best solver for the benchmark problem.

Fig. 3.5a shows that `NLOptControl`'s solvers are faster than *PROPT*'s. At a high-level, `NLOptControl` solves 88% of the problems in real-time using h-methods (i.e., $\texttt{NLOpt}_{\mathrm{E}}$ and $\texttt{NLOpt}_{\mathrm{T}}$) and 46% of the time using p/hp-methods (i.e., $\texttt{NLOpt}_{\mathrm{LGR}_2}$, $\texttt{NLOpt}_{\mathrm{LGR}_2}$, and $\texttt{NLOpt}_{\mathrm{LGR}_4}$). *PROPT* only solves 0.05% of the problems in real-time using p/hp-methods (i.e., $PROPT_{\mathrm{C}_1}$, $PROPT_{\mathrm{C}_2}$, and $PROPT_{\mathrm{C}_4}$). At a lower-level, the zoomed-in subplot in the the bottom graph of Fig. 3.5a shows that `NLOptControl` solves the benchmark problem in real-time when the number of collocation points per interval is less than: 80 for the single-interval case; 45 for the two-interval case; and 25 for the four-interval interval case. *PROPT* obtains real-time solutions when the number of collocation points per phase is less than 27 for the single-phase case and less than 4 for the two-phase case. For the four-phase case, *PROPT* cannot solve any of the problems in real-time.

Fig. 3.5a also shows that as the number of intervals/phases increase from $\texttt{NLOpt}_{\mathrm{LGR}_1}$ to $\texttt{NLOpt}_{\mathrm{LGR}_4}$ and $PROPT_{\mathrm{C}_1}$ to $PROPT_{\mathrm{C}_4}$, the solve-times increase exponentially. Due to the large solve-times with *PROPT* 's solvers, these trends can only be seen in the top graph of Fig. 3.5a — the bottom graph shows the trends for `NLOptControl`'s solvers. As discussed in the previous section, this increase in solve time is largely due to the fact that with an increase in the intervals/phases larger problems are created and they take longer to solve. Even though the $\texttt{NLOpt}_{\mathrm{LGR}_4}$ solver is solving a problem that is roughly four times larger than the $PROPT_{\mathrm{C}_1}$ solver, $\texttt{NLOpt}_{\mathrm{LGR}_4}$ results in smaller solve-times.

Fig. 3.5a also shows that h-methods in `NLOptControl` are faster than the p-method for the benchmark problem. As the level-of-fidelity increases, the solve-times increase linearly with h-methods and exponentially with the p-method. Additionally,

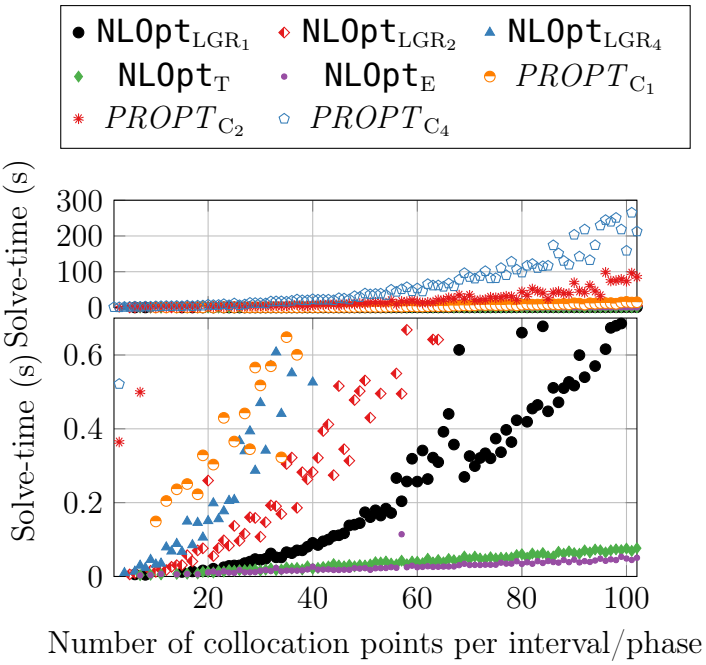
the number of collocation points needs to be greater than about 20 for the h-methods to ensure collision avoidance, while the p-methods need 23 and 27 for `NLOptControl` and *PROPT*, respectively.

The four plots in Fig 3.5b show the ranges of Γ wherein certain solvers dominate. Each profile in this figure shows the probability $\mathcal{P}$ that a given solver s will solve the set of problems P the fastest within a factor of Γ. At $\Gamma = 1$, the solver that has the highest probability of being the fastest is $\texttt{NLOpt}_{\mathrm{E}}$, with a probability of 0.881. $\texttt{NLOpt}_{\mathrm{E}}$ dominates until about $\Gamma = 1.8$, at which point $\texttt{NLOpt}_{\mathrm{T}}$ has the highest probability of being the fastest, with a probability of 0.891. $\texttt{NLOpt}_{\mathrm{T}}$ dominates until about $\Gamma = 80$. The remaining approximate ranges of domination are as follows: $\texttt{NLOpt}_{\mathrm{LGR}_2}$ from 80 to 160, $\texttt{NLOpt}_{\mathrm{LGR}_4}$ from 160 to $5,000$, $PROPT_{\mathrm{C}_4}$ from $5,000$ onwards. Given enough time, $PROPT_{\mathrm{C}_4}$ solves 100% of the problems. For this benchmark problem, while the $\texttt{NLOpt}_{\mathrm{T}}$ and $\texttt{NLOpt}_{\mathrm{E}}$ solvers are much faster than the $\texttt{NLOpt}_{\mathrm{LGR}_2}$, $\texttt{NLOpt}_{\mathrm{LGR}_4}$, and $PROPT_{\mathrm{C}_4}$ solvers they are not as reliable. However, the $\texttt{NLOpt}_{\mathrm{T}}$ and $\texttt{NLOpt}_{\mathrm{E}}$ solvers are both faster and more reliable than the $\texttt{NLOpt}_{\mathrm{LGR}_2}$, $PROPT_{\mathrm{C}_1}$, and $PROPT_{\mathrm{C}_2}$ solvers.

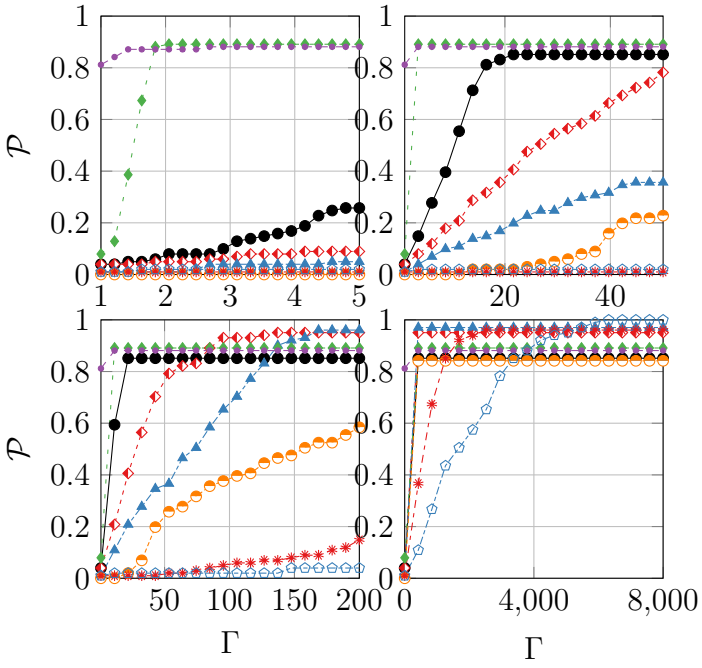
3.7 Discussion

The approach detailed in Section 3.2 yields a direct-collocation-based optimal control modeling language that is both faster and easier to use than *PROPT*. The results and the following discussion support this claim.

`NLOptControl` is easier to use than *PROPT*, because its syntax is more concise, and focused on building a model of the OCP in Bolza form. Differences between Listing. III.2 and Listing. III.3, in terms of number of lines of code and the number of characters per line of code, indicate that `NLOptControl` models OCPs more succinctly than *PROPT*. This work speculates that *PROPT* requires more lines of code to formulate other more practical problems as well.



(a) The top graph shows all average solve-times obtained. The bottom graph shows the solve-times obtained near or below the real-time threshold of 0.5 s.



(b) Performance profiles: each graph shows the performance profiles for a different range of Γ.

Figure 3.5: Benchmark results `NLOptControl` and *PROPT* for the kinematic bicycle problem, see Table 3.2 for legend explanation.

In addition to *PROPT*'s verbosity, its syntax is flexible to the extent that modeling errors are easier to be made. This claim is made because its users can more easily formulate problems that do not fit into the Bolza OCP form. As an example, consider using *PROPT* to model the dynamic constraints in Eqn. 4.2 for the moon lander problem — Line 7 in Listing. III.3. When using *PROPT*, if the user were to forget to include the second differential equation as

```
ode = collocate({dot(x)==v});
```

an error would not be displayed; such overly flexible syntax can lead to modeling errors. If that same mistake were attempted in `NLOptControl`, the user would be alerted as

```
julia> dynamics!(n,[:(x2[j])]);
ERROR: The number of differential equations must equal ocp.state.
    ↪ num.
```

Thus, `NLOptControl` helps avoid modeling errors better than *PROPT*, because `NLOptControl`'s syntax does not allow users to formulate problems that are not in the Bolza form, while *PROPT*'s syntax does.

Both `NLOptControl` and *PROPT* can be used formulate OCPs, but *PROPT* takes a functional approach to this task rather than a modeling approach, as with `NLOptControl`. Listing. III.2 is compared to Listing. III.3 to support this claim. Listing. III.3 shows that, with *PROPT*, the user creates all of the components of the OCP and finally assembles them on Line 12. With `NLOptControl`, in Listing. III.2, it is clear from the first line of code that a model named `n` is being built. Using this approach, `NLOptControl` can clearly model and solve multiple OCPs at once. Such an object-oriented approach can further reduce potential modeling errors.

The benchmark results in Fig. 3.5a and Fig. 3.5b show that `NLOptControl` is faster than *PROPT*. Differences between these packages that affect speed include: differentiation methods, underlying computational language, and available direct-

collocation methods.

PROPT uses symbolic automatic differentiation to calculate the derivatives. However, the structure of the Hessian matrices born from approximating an OCP using direct-collocation methods is sparse and symbolic automatic differentiation does not exploit this structure for speed. In contrast, `NLOptControl` uses the acrylic-coloring method to exploit the sparse structure of the Hessian matrix in conjunction with reverse automatic differentiation. Based on this difference, `NLOptControl` is expected to be faster than *PROPT*, especially when solving large problems that have a very sparse structure.

PROPT's differentiation methods are implemented in *MATLAB* and `NLOptControl`'s are implemented in *Julia*. Unfortunately, the literature does not contain benchmarks of each of these differentiation methods in both *MATLAB* and *Julia*. However, research has shown that *Julia* is much faster than *MATLAB* for a wide range of problem types [13]. Thus, *Julia* may be able to run the reverse automatic differentiation method combined with the acrylic-coloring method to identify sparsity in the Hessian matrix faster than *MATLAB* —if it were implemented in *MATLAB*.

Overall, this chapter speculates that `NLOptControl`'s unique combination of differentiation methods and computational language makes it faster than *PROPT*. This only a speculation since the direct-collocation methods are different between `NLOptControl` and *PROPT*. However, the following pairs of solvers can be considered roughly equivalent in terms of their direct-collocation methods: $\texttt{NLOpt}_{\text{LGR}_1}$ and $PROPT_{\text{C}_1}$, $\texttt{NLOpt}_{\text{LGR}_2}$ and $PROPT_{\text{C}_2}$, and $\texttt{NLOpt}_{\text{LGR}_4}$ and $PROPT_{\text{C}_4}$. Between these pairs, `NLOptControl` solves the problem roughly 14, 26, and 36 times faster than *PROPT*, respectively. It is unlikely that these large differences are due to either differences between collocating at Chebyshev nodes vs. LGR nodes, multiple interval vs. multiple phase methods, or some combination of the two. Thus `NLOptControl` is fast, which is especially important for MPC applications. A brief discussion of

`NLOptControl`'s salient MPC functionality follows.

`NLOptControl` has optional functionality that helps account for the non-negligible solve-times in MPC applications. In MPC, often the control delay (i.e., solve-time) is neglected [92]. When the control delay is neglected, the current state of the plant is used to initialize the problem as opposed to initializing the problem with a prediction of what the plant's state will be after the solve-time has elapsed. Typically neglecting the solve-time in linear model predictive control is not an issue; because in linear MPC, the quadratic program is solved so quickly that initial state of the trajectory can be set to the current state of the plant without compromising robustness [28]. However, neglecting the solve-time in NMPC is likely to deteriorate robustness [129], because the NLPs often take a non-negligible amount of time to solve, after which the state of the plant will have evolved significantly. To make matters even more challenging, ensuring that the NLP solve-times are smaller than a particular execution horizon remains an unsolved problem [102, 34, 104, 28, 2, 80]. Fortunately, for many problems, these NLP solve-times are similar and an upper limit determined based on experience. This upper limit can be used to determine a fixed execution horizon. In `NLOptControl`, after the user selects an execution horizon, as described in Section. 3.3, the framework in Fig. 3.3b can be used to account for non-negligible solve times. Frameworks such as this, can help establish conceptual schemes to improve safety and performance in NMPC applications.

3.8 Conclusions

This chapter introduces an open-source, direct-collocation method based OCP modeling language called `NLOptControl`. `NLOptControl` extends the *JuMP* optimization modeling language to include a natural algebraic syntax for modeling OCPs. `NLOptControl` is compared against *PROPT* in terms of ease of use and speed. *PROPT*'s syntax is shown to be more verbose and error-prone than `NLOpt-`

`Control`'s; thus `NLOptControl` is easier to use than *PROPT*. This ease of use is largely attributed to `NLOptControl`'s use of the *JuMP* optimization modeling language. In addition to being easier to use, results from the benchmark tests show that `NLOptControl` is much faster than *PROPT*. `NLOptControl`'s superior performance is likely due to the unique utility of the *Julia* programming language and the reverse automatic differentiation method in conjunction with the acrylic-coloring method to exploit the sparsity of the Hessian matrices. `NLOptControl` emerges as an easy to use, fast, and open-source [37] optimal control modeling language that holds great potential for not only improving existing off-line and on-line control systems but also engendering a wide variety of new ones.

Acknowledgements

This work was supported by the Automotive Research Center (ARC) in accordance with Cooperative Agreement W56HZV-14-2-0001 U.S. Army Tank Automotive Research, Development and Engineering Center (TARDEC) Warren, MI. Benoît Legat provided a thoughtful review of this chapter and discussions with the several *Julia* developers including Miles Lubin, Tony Kelman, and Chris Rackauckas were helpful.

CHAPTER IV

Real-time Trajectory Planning for Automated Vehicle Safety and Performance in Dynamic Environments

4.1 Introduction

Enabling performance and safety of automated vehicles in known dynamic environments necessitates a trajectory planning algorithm that is more comprehensive than existing formulations. Here, comprehensiveness is quantified by performance and safety specifications. This chapter develops a trajectory planning algorithm that is more comprehensive than existing algorithms. As motivated in Chapter I, this algorithm includes the following performance and safety specifications: minimum time to goal, a dynamic vehicle model, minimum control effort, both static and moving obstacle avoidance, simultaneous optimization of speed and steering, and a short execution horizon.

This chapter addresses the following research objectives:

- Introduce an NMPC-based trajectory planner with S_1-S_7, tailored for a UGV application.
- Investigate the effect that different sets of specifications have on safety, perfor-

mance, and solve-time.

- Investigate the need to include a moving obstacle avoidance specification for a range of execution horizons and obstacle speeds.
- Investigate `NLOptControl`'s ability to solve the proposed formulation in real-time with a short execution horizon.

This chapter assumes that

- both the goal and obstacle information are known,
- the vehicle state is known, and
- the terrain is flat.

The remainder of this chapter is organized as follows: Section 4.2 describes the NMPC framework developed to consider non-negligible trajectory planning problem solve-times and the underlying OCP formulation developed to include S_1-S_6. Section 4.3 describes the test conditions under which the proposed planner is evaluated. In Section 4.4, the effect that adding different specifications to trajectory planners has on safety, performance, and solve-time is tested in a variety of environments. The results of these tests are discussed in Section 4.5. Section 4.6 summarizes the chapter and draws conclusions.

4.2 Mathematical Formulation

4.2.1 NMPC Framework

At heart of an NMPC formulation lies an OCP. In NMPC simulation studies, OCP solve-times are often neglected [129, 92]. In such a case, first the plant simulation is paused, and the OCP is initialized at the current time t_0 with the current plant state $\mathbf{X0}$. Next, the OCP is solved to produce an optimal control signal $\boldsymbol{\zeta}^*$ (i.e.,

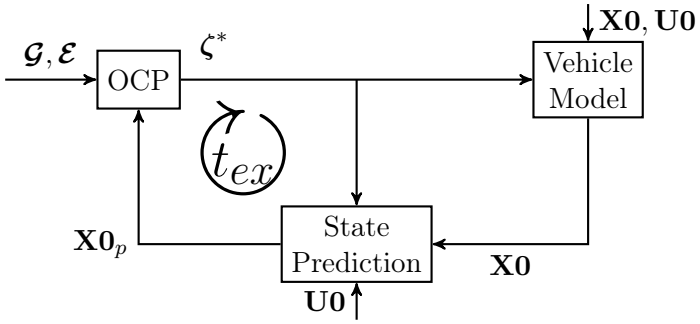


Figure 4.1: Nonlinear model predictive control framework used to account for non-negligible optimal control problem (OCP) solve-times.

trajectory). With this signal, the plant is then simulated starting at t_0 with $\mathbf{X0}$ until $t_0 + t_{ex}$; this yields a new initial state, which is then used to initialize the next OCP. However, most practical OCPs take a non-negligible amount of time to solve, after which, in a more realistic simulation, the plant will have evolved from its current state, where the OCP was initialized, to a new state [129]. This computational delay renders the control signal sub-optimal and potentially infeasible or unsafe. To achieve optimal safety and performance, non-negligible OCP solve-times must be taken into account. The NMPC framework used in this work, shown in 4.1, accounts for these non-negligible OCP solve-times.

This framework has three main components: the OCP, the vehicle model (or plant model), and the state prediction function. The OCP is provided with goal and environment information, defined as follows:

Definition IV.1. Goal information $\boldsymbol{\mathcal{G}}$ includes the goal position (x_g, y_g), the desired vehicle orientation at the goal ψ_g, and the radial tolerance for attaining the goal σ.

Definition IV.2. Environment information $\boldsymbol{\mathcal{E}}$ includes the sizes, initial positions, and velocities of the obstacles.

The obstacles are assumed to be ellipse-shaped, where $\mathbf{a_{obs}}$ and $\mathbf{b_{obs}}$ are arrays that describe obstacles' semi-major and semi-minor axes, respectively; $\mathbf{x0_{obs}}$ and

$\mathbf{y0_{obs}}$ are arrays of the obstacles' initial x and y positions, respectively; and $\mathbf{v_x}$ and $\mathbf{v_y}$ are arrays of the obstacles' speeds in the x and y directions, respectively.

During the first execution horizon, the OCP has not produced a control signal for the vehicle to follow. Therefore, the vehicle is sent a known control signal **U0** (see Fig. 4.1), set such that the vehicle will drive straight at a constant speed. To account for the evolution of the plant state during the execution horizon, a state prediction $\mathbf{X0}_p$, that is made for $t_0 + t_{ex}$, is used to initialize the OCP. The inputs of the state prediction function are the current state of the vehicle and the current control signal, which is **U0** during the first execution horizon and $\boldsymbol{\zeta}^*$ afterward. Then, the plant model is simulated from the initial time t_0 to $t_0 + t_{ex}$ and the first OCP is solved. Real-time feasibility of this framework requires that the OCP solve-times be all less than the execution horizon.

Model mismatch between the plant model, the vehicle dynamics model in the OCP, and the state prediction function can induce biases. These biases can affect the integrity of the research objectives, provided in the previous section. Addressing these biases is acknowledged as an important research problem, but is not one of the goals of this chapter. Therefore, to avoid these biases while focusing on our research tasks, in this chapter, the plant model, the vehicle model in the OCP, and the state prediction function all use the same set of differential equations, which is presented in detail later in this section.

This framework runs until the UGV either reaches the goal or fails the test. An algorithm is run after each execution horizon to determine if the vehicle has reached the goal within the radial goal tolerance σ. The test fails if

1. the vehicle crashes into an obstacle,

2. the vertical tire load in the plant model goes below 100 N for any of the four tires,

3. any of the solve-times exceeds 300 s, or if

4. the solution to the nonlinear programming problem (NLP) is not considered to be optimal based on the tolerances and the Karush-Kuhn-Tucker conditions.

4.2.2 Optimal Control Problem

This section describes how the set of planner specifications S_1-S_6 are incorporated into the OCP. At a high-level, these specifications are all incorporated into the single-phase, continuous-time OCP defined in Eqn. 4.1 - Eqn. 4.4 as

$$\underset{\xi(t),\,\zeta(t),\,t_f}{\textbf{minimize}} \quad \mathcal{M}(\xi(t_0+t_{ex}), t_0+t_{ex}, \xi(t_f), t_f) + \int_{t_0+t_{ex}}^{t_f} L(\xi(t), \zeta(t), t)\, dt \tag{4.1}$$

subject to

$$\frac{d\xi}{dt}(t) - f(\xi(t), \zeta(t), t) = 0 \tag{4.2}$$

$$C(\xi(t), \zeta(t), \mathcal{A}(t), t_f) \leq 0 \tag{4.3}$$

$$\phi(\xi(t_0+t_{ex}), t_0+t_{ex}, \xi(t_f), t_f) = 0 \tag{4.4}$$

where t_f is the free final time, $t \in [t_0 + t_{ex}, t_f]$ is the time, $\xi(t) \in \mathbb{R}^{n_{st}}$ is the state and $\zeta(t) \in \mathbb{R}^{n_{ctr}}$ is the control, with n_{st} defined as the number of states and n_{ctr} defined as the number of controls. The Mayer term is $\mathcal{M} : \mathbb{R}^{n_{st}} \times \mathbb{R} \times \mathbb{R}^{n_{st}} \times \mathbb{R} \to \mathbb{R}$ and the Lagrange term is $L : \mathbb{R}^{n_{st}} \times \mathbb{R}^{n_{ctr}} \times \mathbb{R} \to \mathbb{R}$. The dynamic constraints are given by $f : \mathbb{R}^{n_{st}} \times \mathbb{R}^{n_{ctr}} \to \mathbb{R}^{n_{st}}$. The path constraints are captured by $C : \mathbb{R}^{n_{st}} \times \mathbb{R}^{n_{ctr}} \times \mathbb{R}^{a} \times \mathbb{R} \to \mathbb{R}^{p}$, which bound: the state and control based on the vehicle's dynamic limits, and any additional information, denoted as $\mathcal{A}(t) \in \mathbb{R}^{a}$; and t_f based on a maximum final time $t_{f_{max}}$. Finally, the event constraints are expressed with $\phi : \mathbb{R}^{n_{st}} \times \mathbb{R} \times \mathbb{R}^{n_{st}} \times \mathbb{R} \to \mathbb{R}^{q}$.

The remainder of this section describes how S_1-S_6 are incorporated into Eqn. 4.1 - Eqn. 4.4.

Cost Functional First, the cost functional in Eqn. 4.1 is set to Eqn. 4.5 as

$$
\begin{aligned}
J &= w_t t_f \\
&+ w_g \frac{(x(t_f) - x_g)^2 + (y(t_f) - y_g)^2}{(x(t_0 + t_{ex}) - x_g)^2 + (y(t_0 + t_{ex}) - y_g)^2 + \epsilon} \\
&+ w_{cf} \int_{t_0 + t_{ex}}^{t_f} [w_{\delta_f} \delta_f(t)^2 + w_\gamma \gamma(t)^2 + w_J J_x(t)^2] \mathrm{d}t \\
&+ w_{F_z} \int_{t_0 + t_{ex}}^{t_f} [\tanh(-\frac{F_{z_{rl}} - a}{b}) + \tanh(-\frac{F_{z_{rr}} - a}{b})] \mathrm{d}t + \varrho \\
&+ w_{haf} \int_{t_0 + t_{ex}}^{t_f} [\sin(\psi_g)(x - x_g) - \cos(\psi_g)(y - y_g)]^2 \mathrm{d}t
\end{aligned} \tag{4.5}
$$

where $w_t, w_g, w_{cf}, w_{\delta_f}, w_\gamma, w_J, w_{F_z}, w_{haf}$ are weight terms, $x(t)$ and $y(t)$ represent the vehicle's global position coordinates on a flat plane, ϵ is a small number set to 0.01 to avoid singularities, $\delta_f(t)$ is the steering angle at the front of the vehicle, $\gamma(t)$ is the steering rate, $J_x(t)$ is the longitudinal jerk, a and b are parameters to prevent $F_{z_{rl}}$ and $F_{z_{rr}}$ from being close to the minimum vertical tire load limit, denoted as $F_{z_{min}}$, as described in [92], and ϱ is a term for penalizing the slack variables on the initial and terminal conditions.

There are six terms in Eqn. 4.5, the first of which minimizes the final time t_f, which helps establish a minimum-time-to-goal specification. The second term helps the vehicle reach the goal when the goal is not within the LiDAR range, denoted as L_{range}. If the goal is within a distance of L_{range}, then w_g is set to zero, and the vehicle is constrained to reach the goal. This constraint is described in greater detail later in this section. The third term minimizes the control effort, which encourages smooth control signals. The fourth term dissuades the controller from generating solutions near the minimum vertical tire load limit. This is done to prevent vehicle rollover and infeasible initializations in the next OCP. The fifth term establishes soft constraints

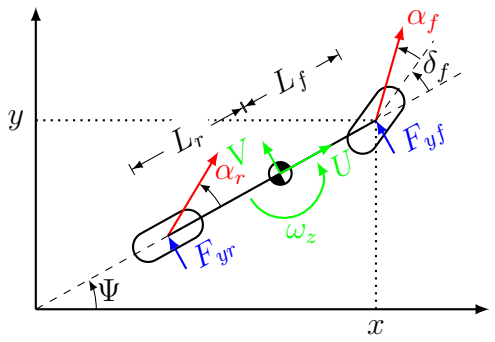


Figure 4.2: 3DoF dynamic vehicle model [91].

on the initial and terminal conditions. This term is also described in greater detail later in this section. Finally, to help the vehicle pass the goal location through the desired direction ψ_g the sixth term is added, which minimizes the area between a line in the (x, y) plane going through the goal in the desired direction ψ_g and the vehicle's trajectory in the (x, y) plane [92].

Dynamic Vehicle Model When including both the dynamic vehicle model and minimum time-to-goal specifications, it is important to consider that the vehicle may need to operate at its dynamic limits. Thus, this work leverages the 3DoF vehicle model developed in [91] (shown in Fig. 4.2). This model is designed to plan trajectories that operate a HMMWV safely at its dynamic limits [91]. To achieve this, it has eight states, two controls, uses a pure-slip Pacejka tire model [107], and considers the longitudinal load transfer effects when calculating the vertical tire forces. The differential equations that are used to model the plant, the vehicle dynamics in the OCP, and the state prediction function are shown in Eqn. 4.6 as

$$f(\xi(t), \zeta(t), t) = \mathcal{D}(\xi(t)) + \mathcal{B}\zeta(t) \tag{4.6}$$

where,

$$\mathcal{D}(\xi(t)) = \begin{bmatrix} U(t)\cos\Psi(t) - (V(t) + L_f\omega_z(t))\sin\Psi(t) \\ U(t)\sin\Psi(t) + (V(t) + L_f\omega_z(t))\cos\Psi(t) \\ (F_{yf}(t) + F_{yr}(t))/M_t - U(t)\omega_z(t) \\ (F_{yf}(t)L_f - F_{yr}(t)L_r)/I_{zz} \\ \omega_z(t) \\ 0 \\ a_x(t) \\ 0 \end{bmatrix},$$

$$\xi(t) = \begin{bmatrix} x(t) \\ y(t) \\ V(t) \\ \omega_z(t) \\ \Psi(t) \\ \delta_f(t) \\ U(t) \\ a_x(t) \end{bmatrix}$$

$$\mathcal{B}^T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } \zeta(t) = \begin{bmatrix} \gamma(t) \\ J_x(t) \end{bmatrix}$$

Eqn. 4.6 breaks the dynamics constraints in Eqn. 4.2 into two terms. The first of these terms, $\mathcal{D}(\xi(t))$, establishes the state dynamics for the global position of the vehicle, the lateral speed $V(t)$, the yaw rate $\omega_z(t)$, the heading angle $\Psi(t)$, the steering

angle $\delta_f(t)$, the longitudinal speed $U(t)$, and the longitudinal acceleration $a_x(t)$. The second term, $\mathcal{B}\zeta(t)$, relates state variable rates to their respective control variables, i.e., $\frac{d\delta_f}{dt}(t)$ to the steering rate $\gamma(t)$, and $\frac{da_x}{dt}(t)$ to the longitudinal jerk $J_x(t)$. Finally, L_f and L_r are the distances from the front and rear axles to center-of-mass (COM), I_{zz} is the moment of inertia about the COM, F_{yf} and F_{yr} are the front and rear lateral tire forces, and M_t is the total vehicle mass. Table C.3, which is in Appendix 6.2, contains all of the vehicle parameters used in this chapter.

The vertical tire load on each of the four tires is constrained to be above the minimum vertical tire load limit $F_{z_{min}}$. These constraints helps prevent vehicle roll-over and are incorporated into Eqn. 4.3. To calculate the vertical loads on the tires, this work uses a vertical load transfer model [92]. The vertical tire forces are approximated as

$$
\begin{aligned}
F_{z_{rl}} &= \frac{1}{2}(F_{z_{r0}} + K_{z_x}(a_x(t) - V(t)\omega_z(t)) - K_{z_{yr}}\frac{F_{yf}+F_{yr}}{M_t} \\
F_{z_{rr}} &= \frac{1}{2}(F_{z_{r0}} + K_{z_x}(a_x(t) - V(t)\omega_z(t)) + K_{z_{yr}}\frac{F_{yf}+F_{yr}}{M_t} \\
F_{z_{fl}} &= \frac{1}{2}(F_{z_{f0}} - K_{z_x}(a_x(t) - V(t)\omega_z(t)) - K_{z_{yf}}\frac{F_{yf}+F_{yr}}{M_t} \\
F_{z_{fr}} &= \frac{1}{2}(F_{z_{f0}} - K_{z_x}(a_x(t) - V(t)\omega_z(t)) + K_{z_{yf}}\frac{F_{yf}+F_{yr}}{M_t}
\end{aligned}
$$

where $F_{z_{rl}}$ and $F_{z_{rr}}$ are the rear left and rear right vertical tire loads, $F_{z_{fl}}$ and $F_{z_{fr}}$ are the front left and front right vertical tire loads, $F_{z_{r0}} = \frac{M_t L_f g}{L_f + L_r}$ is the static rear axle load, $F_{z_{f0}} = \frac{M_t L_r g}{L_f + L_r}$ is the static front axle load, and K_{z_x} is the longitudinal load transfer coefficient, $K_{z_{yf}}$ and $K_{z_{yr}}$ are the front and rear lateral load transfer coefficients [92].

State and Control Limits Actuator and other physical plant limits help establish the state and control bounds, which are added to Eqn. 4.3. Specifically, five of the

states and both controls are bounded with constant upper and lower bounds as

$$x_{\min} \leq x(t) \leq x_{\max}$$

$$y_{\min} \leq y(t) \leq y_{\max}$$

$$\psi_{\min} \leq \psi(t) \leq \psi_{\max}$$

$$\delta_{f,\min} \leq \delta_f(t) \leq \delta_{f,\max}$$

$$U_{\min} \leq U(t) \leq U_{\max}$$

$$\gamma_{f,\min} \leq \gamma_f\,(t) \leq \gamma_{f,\max}$$

$$J_{x,\min} \leq J_x\,(t) \leq J_{x,\max}$$

Finally, nonlinear functions of the vehicle's speed bound the vehicle's acceleration as

$$a_{x,\min}[U(t)] \leq a_x(t) \leq a_{x,\max}[U(t)]$$

Maximum deceleration/acceleration data collected from a 14DoF HMMWV model are used to establish these nonlinear functions for the maximum deceleration/acceleration [92].

No explicit lateral speed or yaw rate constraints exist.

Obstacle Avoidance Two possible approaches for incorporating the static and moving obstacle avoidance specifications into the OCP include soft constraints (or artificial potential-fields) and time-varying hard constraints [140]. There are two limitations to the soft constraints approach: (1) a trajectory may be generated that is deemed feasible according to the formulation, but actually goes through an obstacle, and (2), the NLP solve-times are known to be large, when compared with the time-varying hard constraints approach [38]. Therefore, in this formulation, time-

varying hard constraints for the avoidance of static and moving obstacles avoidance are incorporated into Eqn. 4.3.

Time-varying hard constraints enforce the vehicle's trajectory to avoid intersecting with the obstacles' trajectories, while accounting for the obstacles' shapes and sizes. Because this OCP will be transcribed into an NLP, the obstacles' shapes should be represented with twice continuously differentiable functions, e.g., a circle or an ellipse. As such, similar to planners tailored for spacecraft [71] and UGV [38] applications, this work establishes a moving obstacle avoidance specification using time-varying, elliptical hard constraints. Eqn. 4.7 defines these constraints as

$$\begin{aligned} &(\frac{x(t) - (\mathbf{x0_{obs}}[i] + \mathbf{v_x}t)}{\mathbf{a_{obs}}[i] + sm(t)})^2 + \\ &(\frac{y(t) - (\mathbf{y0_{obs}}[i] + \mathbf{v_y}t)}{\mathbf{b_{obs}}[i] + sm(t)})^2 > 1, \text{ for } i \in 1:Q \end{aligned} \tag{4.7}$$

where $sm(t) = sm_1 + \frac{sm_2 - sm_1}{t_f}t$ describes the time-varying safety margin, which enforces the vehicle to operate further from the obstacles as t increases, and Q is the total number of obstacles. The notation $\mathbf{x0_{obs}}[i]$ refers to the ith element of the $\mathbf{x0_{obs}}$ vector.

LiDAR Region Constraints To ensure that the vehicle's trajectory does not go beyond the LiDAR region, an additional path constraint is incorporated into Eqn. 4.3. This constraint is defined in Eqn. 4.8 as

$$(x(t) - x(t_0 + t_{ex}))^2 + (y(t) - y(t_0 + t_{ex}))^2 - (L_{range} + \kappa)^2 \leq 0 \tag{4.8}$$

where κ is the LiDAR relaxation range [92].

Initial and Terminal State Constraints In a low-tolerance hard constraints approach, if the plant is driven into an infeasible state space, a feasible control signal can-

not be computed [127]. To mitigate infeasible problems created using low-tolerance hard constraints on the initial and terminal conditions, soft constraints are introduced into this formulation. Soft constraints are introduced using slack variables, where the size of the slack variable corresponds to the respective constraint violation [79]. These slack constraints are shown in Eqn. 4.9-4.14 as

$$\mathbf{X0}_p - \xi(t_0 + t_{ex}) \leq \mathbf{x_{0}}_s \tag{4.9}$$

$$\mathbf{X0}_p + \xi(t_0 + t_{ex}) \geq \mathbf{x_{0}}_s \tag{4.10}$$

$$x_g - x(t_f) \leq \mathbf{x_f}_s[1] \tag{4.11}$$

$$x_g + x(t_f) \geq \mathbf{x_f}_s[1] \tag{4.12}$$

$$y_g - y(t_f) \leq \mathbf{x_f}_s[2] \tag{4.13}$$

$$y_g + y(t_f) \geq \mathbf{x_f}_s[2] \tag{4.14}$$

where $\mathbf{x_0}_s$ is the n_{st} dimensional vector of slack variables for the initial conditions, and $\mathbf{x_f}_s$ is the two dimensional vector of slack variables for the terminal conditions.

Adding slack variables to the cost functional reduces the size of the slack constraint violations. The weight for these slack variables is chosen to be large enough to keep the slack constraint close to zero. The ϱ term in Eqn. 4.1 is now defined in Eqn. 4.15 as

$$\varrho = \mathbf{w}_{s0}\mathbf{x_0}_s + \mathbf{w}_{sf}\mathbf{x_f}_s \tag{4.15}$$

where $\mathbf{w}_{s0}$ is a $1 \times n_{st}$ dimensional vector of individual weight terms on the slack variables for the initial state constraints, and $\mathbf{w}_{sf}$ is a 1×2 dimensional vector of individual weight terms on the slack variables for the final state constraints.

When using only soft constraints, "optimal" trajectories are found that have initial and terminal states which are too far from their desired values. Adding high-tolerance hard constraints on the initial and terminal state conditions mitigates this issue. Thus,

high-tolerance hard constraints are added to Eqn. 4.4, where the entire initial state is constrained to match $\mathbf{X0}_p$ within a specified tolerance $\mathbf{X0}_{tol}$. Eqn. 4.16 establishes these constraints as

$$\mathbf{X0}_p - \mathbf{X0}_{tol} \leq \boldsymbol{\xi}(t_0 + t_{ex}) \leq \mathbf{X0}_p + \mathbf{X0}_{tol} \tag{4.16}$$

Additionally, the vehicle's final x and y positions are constrained to be within the goal tolerance σ using Eqn. 4.17 - Eqn. 4.18 as

$$x_g - \sigma \leq x(t_f) \leq x_g + \sigma \tag{4.17}$$

$$y_g - \sigma \leq y(t_f) \leq y_g + \sigma \tag{4.18}$$

If the distance from the vehicle to the goal is greater than the vehicle's planning range L_{range}, then the soft (Eqn. 4.9 - Eqn. 4.14) and hard constraints (Eqn. 4.17 - Eqn. 4.18) on the final conditions are relaxed. Setting the elements in $\mathbf{w}_{sf}$ to zero relaxes the soft constraints, and setting σ to 10^6 m relaxes the hard constraints. The remaining parameter modifications and additional constraints needed to relax the assumption that the goal is within the vehicle's planning range [38] are now presented.

LiDAR Range Constraints If the goal is not within L_{range} of the vehicle, then the vehicle is constrained to arrive at the edge of the LiDAR region within a distance of κ at t_f. This is accomplished using Eqn. 4.19 and Eqn. 4.20 as

$$(x(t_f) - x(t_0 + t_{ex}))^2 + (y(t_f) - y(t_0 + t_{ex}))^2 - p1 \leq 0 \tag{4.19}$$

$$-(x(t_f) - x(t_0 + t_{ex}))^2 - (y(t_f) - y(t_0 + t_{ex}))^2 + p2 \leq 0 \tag{4.20}$$

where $p1$ and $p2$ are set to $(L_{\text{range}} + \kappa)^2$ and $(L_{\text{range}} - \kappa)^2$, respectively.

To avoid creating an infeasible problem while continuing to drive the UGV towards

the goal, the goal constraints described in the previous section (i.e., Eqn. 4.9 - Eqn. 4.18) are relaxed and a new soft constraint is used. This soft constraint minimizes the squared distance from the vehicle to the goal at t_f, normalized by squared distance from the vehicle to the goal at $t_0 + t_{ex}$ [92]. Setting the goal weight w_g in Eqn. 4.5 to a non-zero value enforces this constraint.

In the case that the goal is within a distance of L_{range} to the vehicle, w_g is set to zero and the vehicle is constrained to reach the goal using Eqn. 4.9 - Eqn. 4.18. This is done by setting σ to a much smaller goal tolerance, which enforces the hard constraints on reaching the goal, and setting the elements in the weight vector $\mathbf{w}_{sf}$ to large positive weights establishes soft constraint on reaching the goal through slack variables. Then, to avoid creating an infeasible problem, the hard constraints for reaching the edge of the LiDAR region at t_f are relaxed. To do this, $p1$ and $p2$ are set to 10^{-6} and -10^{-6}, respectively,

The above specifies the details of the NMPC-based trajectory planning formulation with specifications S_1-S_6. The evaluation of this formulation as a function of its specifications follows.

4.3 Evaluation Description

The next section presents comparisons among four planners within three different test environments, and evaluates the proposed planner's ability to improve both safety and performance without increasing solve-times. This section describes these planners and their test environments. Afterwards, the computer hardware platform used to produce the results presented in this chapter and the software configuration under which `NLOptControl` is evaluated are described.

Table 4.1: Planners compared in the work

Features	Planners			
	P_A	P_B	P_C	P_D
static obstacle avoidance	✓	✓	✓	✓
minimum time-to-goal		✓	✓	✓
dynamic vehicle model	✓	✓	✓	✓
minimum control effort			✓	✓
simultaneously optimize speed and steering	✓	✓	✓	✓
moving obstacle avoidance				✓
small execution horizon	✓	✓	✓	✓

4.3.1 Planners

Comparisons are made among four planners (denoted as P_A-P_D). The specifications of these planners are listed in Table 4.1, where P_A is used as the baseline planner. Note that P_A already includes the specifications of a dynamic vehicle model and simultaneous optimization of speed and steering, since previous work already illustrated the need to include them; see [93, 44, 91] for the first and [92] for the latter.

The set of parameters in the left-hand column in Table C.4, which is in Appendix 6.2, define the planners. The right-hand column in Table C.4 defines the values of P_A's parameters. All of the weight terms used in this work are obtained either from previous research [92] or manual tuning. In addition to this, setting the moving obstacle avoidance constraint in Eqn. 4.7 to $false$ means that Eqn. 4.7 is modified to Eqn. 4.21 as

$$(\frac{x(t) - \mathbf{x0_{obs}}\star[i]}{\mathbf{a_{obs}}[i] + sm(t)})^2 + (\frac{y(t) - \mathbf{y0_{obs}}\star[i]}{\mathbf{b_{obs}}[i] + sm(t)})^2 > 1, \text{ for } i \in 1 : Q \tag{4.21}$$

where $\mathbf{x0_{obs}}\star$ and $\mathbf{y0_{obs}}\star$ are arrays that describe the initial x and y positions of the obstacles, respectively. These arrays are updated to reflect the obstacles' current

position after each execution horizon.

Next, the values of P_A's parameters are modified to define P_B-P_D. The difference between P_A and P_B is that the weight on the final time w_t is set to 100 for P_B. To allow P_A to reach the goal or the edge of the LiDAR region at will, the final time is left as a design variable, but the weight on it (i.e., w_t) is set to zero. Next, the difference between P_B and P_C is that w_{ce} is set to 1 for the latter. Finally, the difference between P_C and P_D is that P_D has a moving obstacle avoidance specification, while P_C does not. P_D establishes this specification with Eqn. 4.7. Specifically, P_C assumes that the obstacles will be static over each prediction horizon, while P_D incorporates the movement of the obstacles into the position constraints over the prediction horizon.

Closed-loop comparisons are made among four different vehicles (denoted as $V_A - V_D$). P_A, P_B, P_C, and P_D control V_A, V_B, V_C, and V_D, respectively. Unless otherwise noted, the execution horizon is set to a value of 0.5 s for all comparisons.

4.3.2 Environment Categories

To evaluate the proposed planner, this work uses four distinct environmental categories: unknown vs. known; unstructured vs. structured; dynamic vs. static, and challenging vs. simple. A description of each follows.

Unknown vs. Known In an unknown environment [92], sensors collect data from the environment for algorithms that estimate factors including the obstacles' sizes, positions, and velocities. In addition to assuming that the obstacle information is known, this chapter assumes that the environment known.

Unstructured vs. Structured In an unstructured environment [131, 22, 82, 92], there are no roads to follow or traffic rules to obey. However, in a structured environment [76, 108], some combination of these factors needs to be considered. In this chapter the distinction between these environment categories is that in an unstruc-

tured environment there are no lanes to follow, while in a structured environment there is a lane to follow.

Dynamic vs. Static In a dynamic environment [20], at least one obstacle is moving. In a static environment, all of the obstacles are stationary. This chapter uses both dynamic and static environments.

Challenging vs. Simple In planning problems, the number of obstacles directly affects the computational load [20]. As such, the environment becomes more challenging as the number of obstacles increases. This chapter uses an environment with 38 obstacles as a challenging example and an environment with 3 obstacles as a simple one.

4.3.3 Environments

UGV safety, performance, and solve-times are evaluated in three different environments (denoted as E_A-E_C). Each of these environments consists of some combination of the above environment categories, which is now described in detail.

E_A: Simple, Static, Unstructured Environment Both the increase in performance and solve-times, consequent to including the minimum time-to-goal and minimum control effort specifications, can be evaluated in a simple, static, unstructured environment (denoted as E_A). E_A has three static obstacles (denoted as 0_1, 0_2, and 0_3) and Table C.5, which is in Appendix 6.2, lists E_A's parameters. The right trace of Fig. 5.13 shows the obstacle field and goal location of E_A.

E_B: Simple, Dynamic, Unstructured Environment The increases in both safety and solve-times, consequent to including a moving obstacle avoidance specification, can be evaluated in a simple, dynamic, unstructured environment (denoted

as E_B). E_B has three dynamic obstacles (also denoted as 0_1, 0_2, and 0_3). E_B is the same environment as E_A, except the obstacles are given non-zero velocities to test the planner's ability to avoid collisions with moving obstacles. The respective velocities of 0_1, 0_2, and 0_3 are as follows:

$$\mathbf{v_x} = [-2, -1, -0.5] \frac{\text{m}}{\text{s}} \text{ and } \mathbf{v_y} = [0, 1, 6] \frac{\text{m}}{\text{s}}$$

Fig. 4.4 shows the movement of these obstacles, which can be seen in the right trace by following the obstacles' position at the indicated times.

E_C: Challenging, Dynamic, Structured Environment Increases in safety and solve-times from including a moving obstacle avoidance specification can be further evaluated within a challenging, dynamic, structured environment (denoted as E_C). E_C is a double lane change scenario, which was originally developed to test a HMMWV within a static environment [98]. Fig. 4.5 shows E_C, which has two large obstacles that need to be avoided, labeled 0_1 and 0_2. In this test, the vehicle is started at the bottom of Fig. 4.5 traveling in the left lane at a speed of 17 $\frac{\text{m}}{\text{s}}$. From this point, it is restricted to perform a double lane change maneuver. To constrain the vehicle to perform this maneuver, first, minimum and maximum constraints on the vehicle's x position are imposed. This restricted region is colored in light blue, where the lower and upper limits on the vehicle's x position are $x_{min} = 0$ m and $x_{max} = 24$ m, respectively. Next, to ensure that the vehicle stays in the left lane during the first part of the maneuver, at the start of the track, until after $y = 175$ m, a series of 36 cones are placed at the edge of the lane boundary. If these cones are not present, or there are not enough cones, then the vehicle will change lanes earlier in order to minimize the sixth term in Eqn. 4.5. The first large obstacle, 0_1, is static and is located in the left lane. The second large obstacle, 0_2, starts at the back of the track in the right lane near the goal and moves towards the front of the track. Table

B.5 lists the parameters for E_C.

To improve safety and performance within E_C, several planner parameters, which are listed in Table C.4, are modified. Specifically, for all of the E_C simulations shown in this chapter, the L_{range}, N, and κ planner parameters in Table C.4 are modified. The L_{range} is modified because using an L_{range} of 50 m, the vehicle crashes into 0_2 over a large range of 0_2 obstacle speeds. To avoid limiting the UGV based on its sensing range, and not its dynamic limits, L_{range} is increased to 90 m for E_C. To accommodate for this extended planning range, the number of points in the discretization N is increased from 10 to 15 and the LiDAR relaxation range κ is increased from 5 m to 10 m.

4.3.4 Hardware Platform and Software Stack

The results in this chapter are produced using a single machine running Ubuntu 16.04 with an Intel Core i7 − 4910MQ CPU @2.90GHz × 8, and 31.3GB of RAM. This work evaluates `NLOptControl`'s ability to solve the complex OCP formulation presented in this work in real-time. As mentioned, `NLOptControl` is our open-source, direct-collocation based OCP solver. In this work `NLOptControl` 0.1.6 [37] is used with the *KNITRO* 10.3 NLP solver, where the default *KNITRO* settings are used, except the maximum CPU-time (i.e., solve-time), which is set to 300 s. Additionally, the trapezoidal method [10, 78] is used to approximate both the cost functional (Eqn. 4.1) and the dynamics (Eqn. 4.2). In order to more closely simulate practice, where time can typically be allocated to initialize trajectory planners, the optimizations are warm-started.

4.4 Results

4.4.1 Performance and Solve-Times within E_A

Planning with a minimum time-to-goal specification can reduce the time-to-goal without increasing the solve-times in a simple, static, unstructured environment. In particular, using either P_B or P_C in lieu of P_A within E_A, reduces the time-to-goal from 9.0 s to 7.0 s (see Fig. 5.13). This is because, until about $t = 5.5$ s, both V_B and V_C accelerate while V_A decelerates; this results in higher speeds for both V_B and V_C. Next, while both P_B and P_C run in real-time in E_A, P_A does not. This can be seen in the top left trace of Fig. 5.13, where the solve-times for both P_B and P_C are all less than t_{ex}, but several of the solve-times obtained using P_A go above t_{ex}. Again, this creates a safety issue because, in practice, if the solve-time is greater than t_{ex}, then the vehicle will not have a trajectory to follow.

In E_A, compared to P_B, P_C reduces the control effort without increasing either the time-to-goal or the solve-times. More specifically, even though V_B and V_C arrive at the goal in 7.0 s, V_C uses less control effort for all of the three control effort terms. The third term in Eqn. 4.5 calculates the control effort terms for the steering angle, steering rate, and longitudinal jerk. The overall values of each of these control effort terms, along with their percentage decrease, are in Table B.6, which is in Appendix 6.2. Next, in the top left trace in Fig. 5.13, it can be seen that the solve-times for P_B and P_C are below the real-time threshold of 0.5 s.

4.4.2 Safety and Solve-Times within E_B

Planning with a moving obstacle avoidance specification can increase safety without increasing the solve-times in a simple, dynamic, unstructured environment. This is shown in the comparison between P_C and P_D within E_B (see Fig. 4.4). At the start of this test, both vehicles accelerate and then turn in opposite directions: V_C to

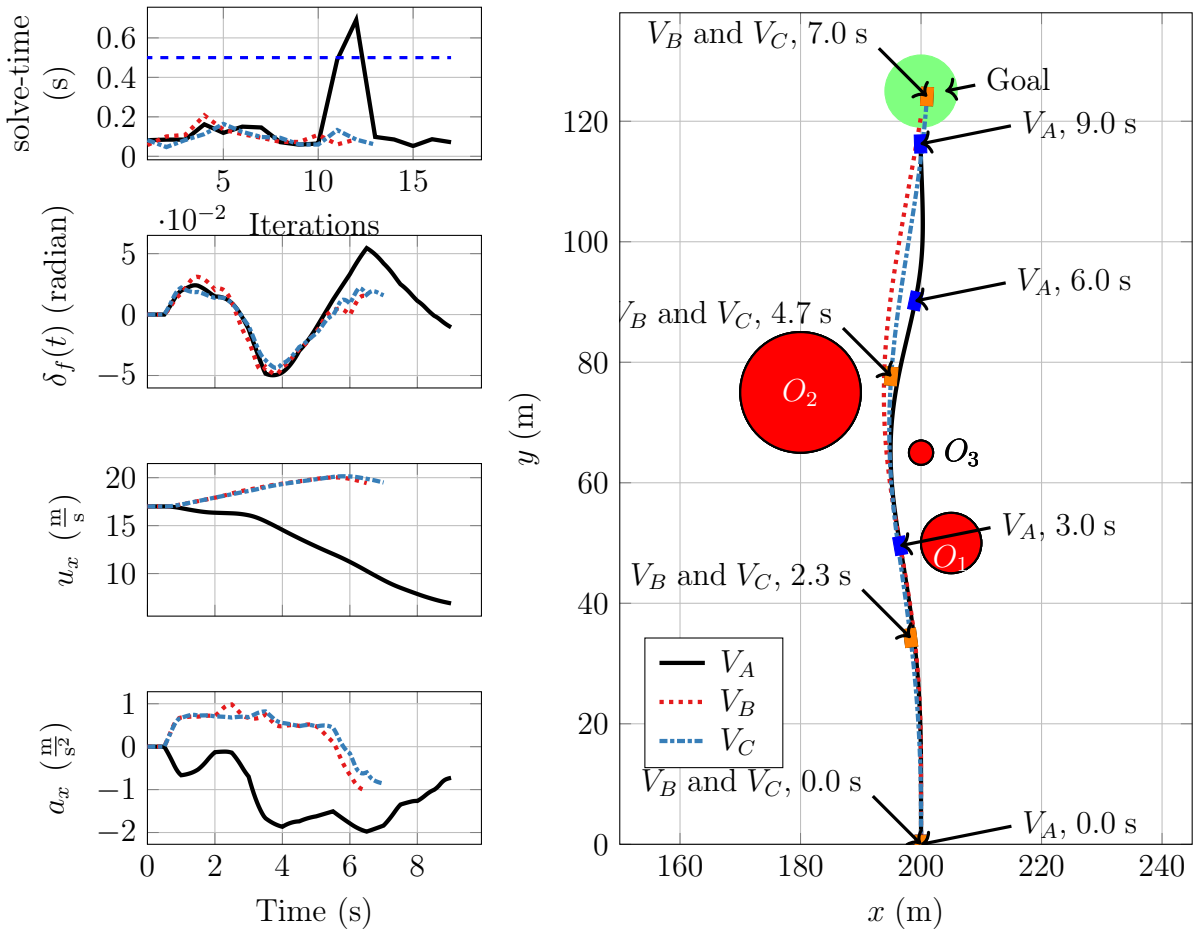


Figure 4.3: Closed-loop comparison of P_A, P_B, and P_C in E_A.

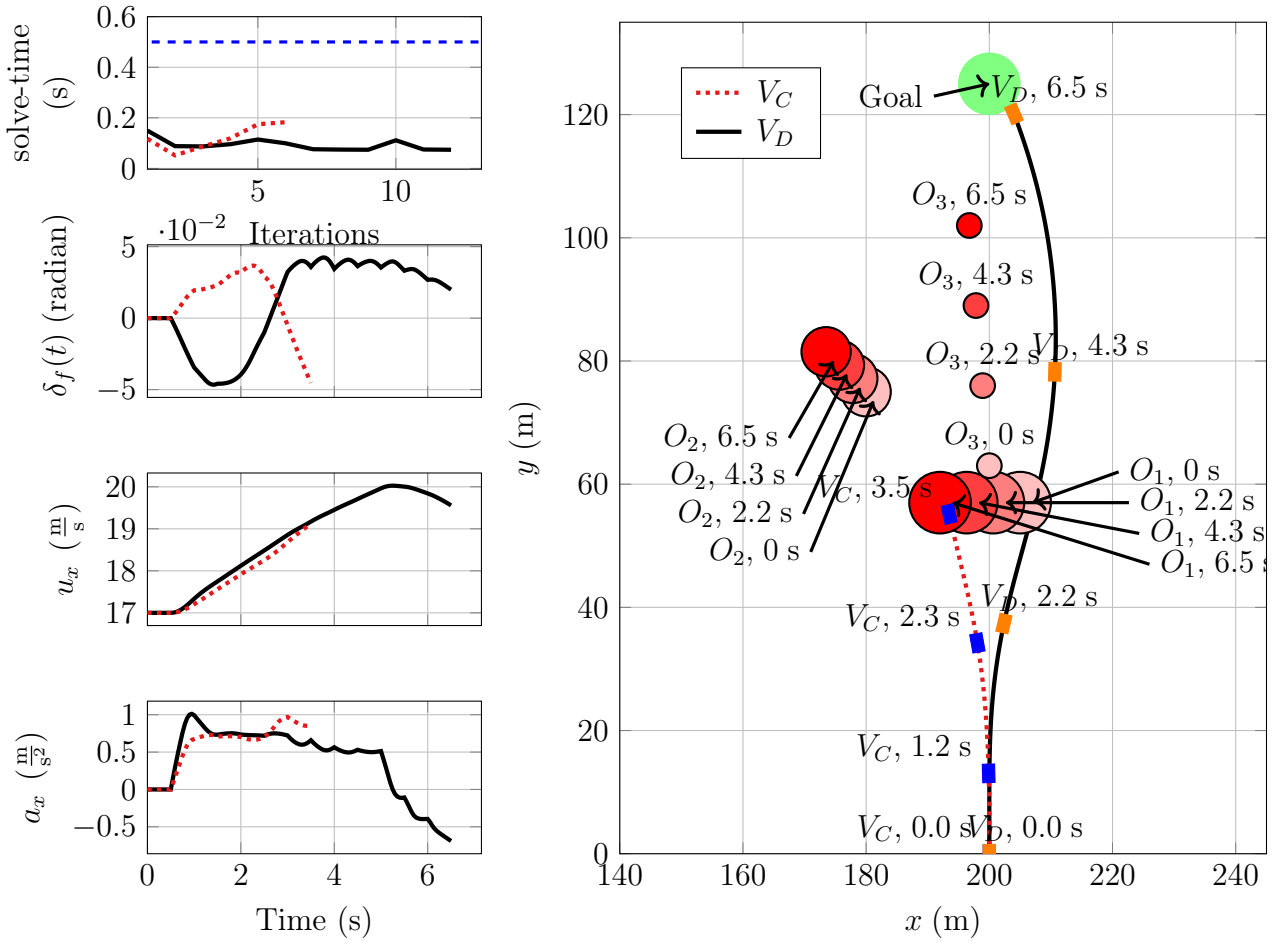


Figure 4.4: Closed-loop comparison of P_C and P_D in E_B.

the left and V_D to the right. P_C tries to avoid 0_1 to the left, which results in a crash at $t = 3.5$ s. On the other hand, by taking the obstacles' motion over the prediction horizon into account, P_D turns V_D to the right. This allows V_D to arrive safely at the goal at $t = 6.5$ s. Lastly, as seen in the top left trace in Fig. 4.4, the solve-times for both P_B and P_C are below the real-time threshold of 0.5 s.

4.4.3 Safety and Solve-Times within E_C

Similarly, planning with a moving obstacle avoidance specification can increase safety without significantly increasing the solve-times in a challenging, dynamic, structured environment. This is demonstrated by testing P_C and P_D within E_C (see Fig.

4.5 - Fig. 4.7). At the start of the test, the first lane-change maneuver is performed successfully for both V_C and V_D.

During this time, both vehicles accelerate aggressively to increase their speed from 17 $\frac{\text{m}}{\text{s}}$ at $t = 0$ s to 26.5 $\frac{\text{m}}{\text{s}}$ at $t = 19.5$ s. At this time, V_C crashes into 0_2 (see Fig. 4.6 for a zoomed in view of the crash) while V_D avoids 0_2 and eventually attains the goal. Additionally, P_D is able to avoid this collision with a solve-time that is only slightly higher than the one obtained with P_C's just before is causes V_C to crash. The next section discusses the larger solve-times encountered at 19.0 s. Finally, the solve-times for P_D are less than the real-time threshold of 0.5 s, despite the fact that this is a challenging environment (i.e., with 38 obstacles instead of 3).

4.4.4 Execution Horizon and Obstacle Speed Analysis within E_C

Including a moving obstacle avoidance specification increases safety over a range of execution horizons and obstacle speeds. To shown this V_C and V_D are tested within E_C for a range of execution horizons ($\mathbf{t_{ex}} = [0.01, 0.0621, \ldots, 1]$ s) and obstacle velocities ($\mathbf{v_y}[2] = [0, -2.11, \ldots, -20]$ $\frac{\text{m}}{\text{s}}$). The data from this parameter sweep are shown in Fig. 4.8, where a plotted point indicates a successful simulation. For instance, when the execution horizon is 0.01 s and the velocity of 0_2 is -2.11 $\frac{\text{m}}{\text{s}}$, both V_C and V_D attain the goal.

The data follow the expected trend: i.e., V_D is safer than V_C, and the results are statistically significant ($p = 2.2 \times 10^{-16}$), as shown by a Fisher Test, in Appendix 6.2. V_D accounts for the majority (87.1%) of the successful trials, and V_C accounts for the majority of (60.0%) of the trials that failed.

While making the execution horizon small creates a more reactive planner, which can more reliably avoid collisions with fast moving obstacles, it makes it more difficult to obtain the planning solutions in real-time. Fig. 4.9 depicts this issue, where the real-time-factor (RTF) and probability-of-safety (POS) are defined as follows

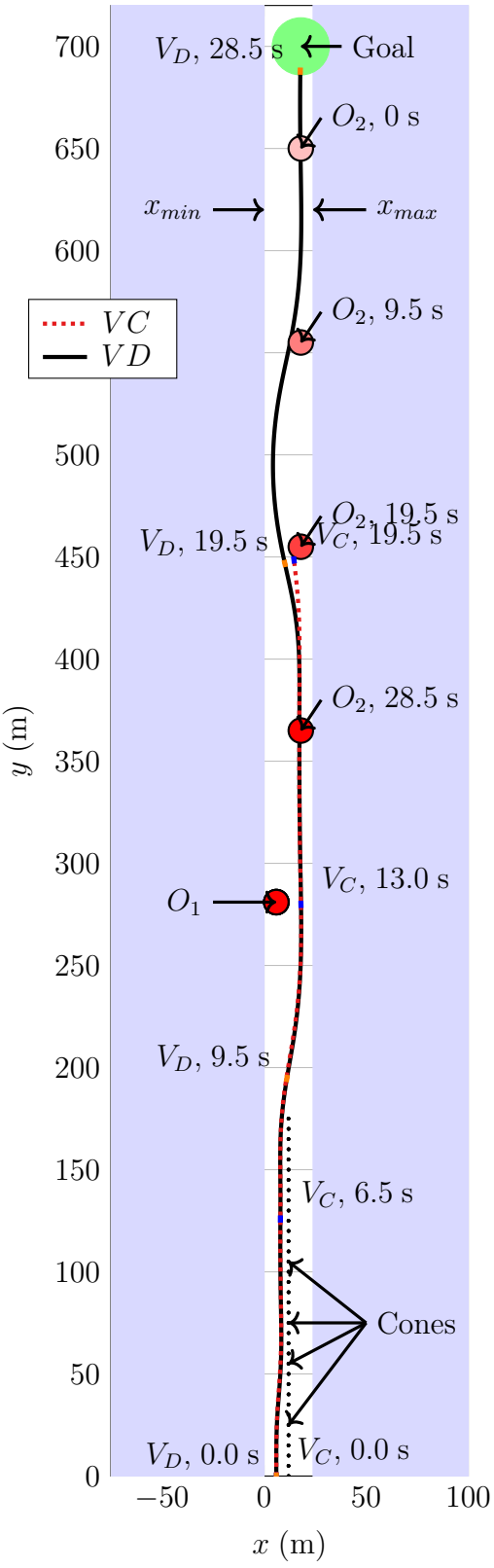


Figure 4.5: Closed-loop comparison of P_C and P_D in E_C.

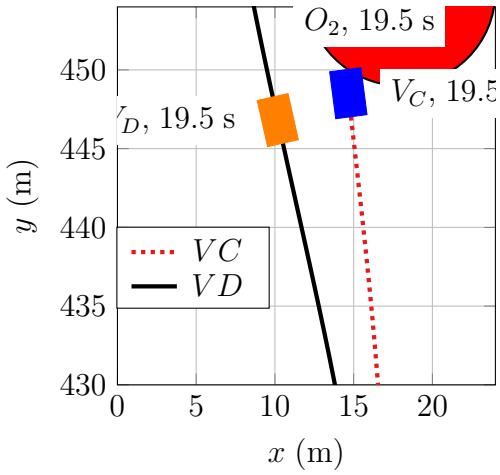


Figure 4.6: Zoomed in on Fig. 4.7 at 19.5 s

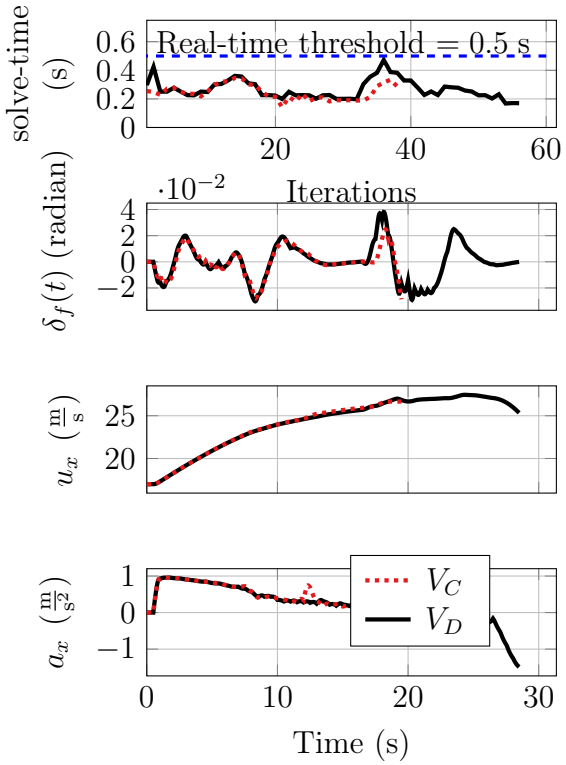


Figure 4.7: Closed-loop comparison of P_C and P_D in E_C.

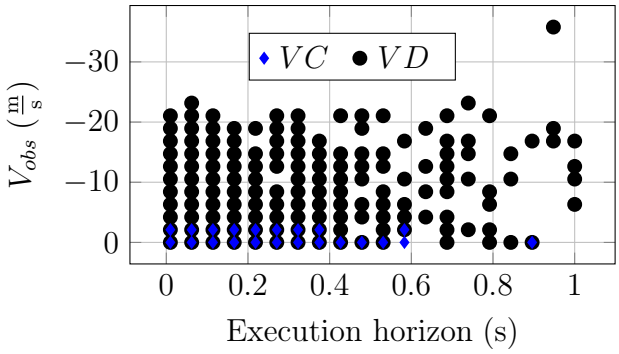


Figure 4.8: Effect that both the execution horizon and obstacle speed have on the vehicle attaining the goal in E_C for both P_C and P_D.

Definition IV.3. Real-time-factor (RTF): RTF $= \frac{\text{solve-times}_{\max}}{t_{ex}}$. To calculate $\text{solve-times}_{\max}$, the maximum value in a vector of solve-times for each test case (i.e., obstacle speed and execution horizon) is averaged across obstacle speeds.

Definition IV.4. Probability-of-safety (POS): The probability that the vehicle will attain the goal, which is calculated over the range of obstacle velocities for each execution horizon.

For both P_C and P_D, the RTF is very high at small execution horizons and drops for larger execution horizons, as shown in Fig. 4.9. For P_C, the POS is very low across the entire range of execution horizons. In contrast, P_D's POS is higher for smaller execution horizons and lower for larger execution horizons. Additionally, when using P_D there are two cases where the RFT is less than 1, namely when the execution horizon is either 0.687 or 0.790 s. In these cases the POS is 0.45 and 0.25, respectively.

4.5 Discussion

This chapter develops four NMPC-based trajectory planners, each with a different set of specifications. Comparisons among these planners, within three different environments, illuminate the potential effects of several key planner specifications on

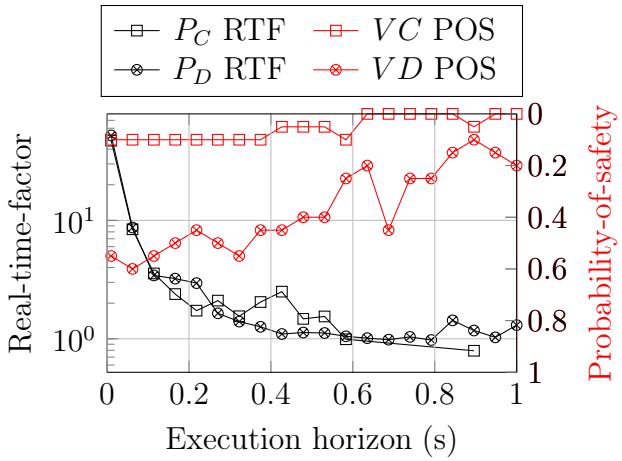


Figure 4.9: Effect of the execution horizon on both the maximum real-time factor (left axis) and the probability-of-safety (right axis) in E_C for both P_C and P_D.

UGV safety and performance. These comparisons provide the basis for this chapter's contributions.

This work was motivated by the assumption that including the set of specifications S_1-S_7 into a planner will improve both performance and safety, compared with less comprehensive sets. The results presented in this chapter support this assumption. In particular, the results show that including (a) minimum time-to-goal, (b) minimum control effort, and (c) moving obstacle avoidance specifications improves the closed-loop performance and safety for a UGV application.

Contrary to our expectations, adding several key planner specifications does not lead to larger solve-times. Specifically, the results show that adding (a) minimum time-to-goal, (b) minimum control effort, and (c) moving obstacle avoidance specifications does not lead to an increase in NLP solve-times.

In fact, adding a minimum time-to-goal specification actually reduces the solve-times within the simple, static, unstructured environment (see the top left trace in Fig. 5.13). The minimum time-to-goal specification helps balance the sixth term in Eqn. 4.5, which minimizes the area between the vehicle's position trajectory and a line that runs through the goal in the y-direction. To see this balancing effect, compare the

position trajectories of V_A to those of V_B and V_C in Fig. 5.13. The baseline planner, i.e., P_A, more effectively minimizes the area mentioned above for V_A than either P_B or P_C does for V_B and V_C, respectively. V_B and V_C have a larger area because both P_B and P_C have, in addition to the sixth term in Eqn. 4.5, a minimum time-to-goal specification. To reduce this area more effectively, V_A aggressively decelerates over the entire test and operates at lower speeds; these lower speeds allow V_A to return to the line that runs through the goal in the y-direction sooner than either V_B or V_C. These differences between the vehicle's trajectories may have led to the differences in the planners' solve-times, where P_A has longer solve-times than either P_B or P_C. Notice that at around 6 s, V_A's steering angle $\delta_f(t)$ and longitudinal deceleration $a_x(t)$ are large, and P_A's solve-time increases sharply. On the basis of such observations, this chapter speculates that planning aggressive deceleration and steering trajectories at low speeds may be more computationally expensive than planning less aggressive deceleration and steering trajectories at high speeds.

The results presented in this chapter show that `NLOptControl` can solve UGV OCPs in real-time, suggesting that `NLOptControl` can solve complex OCPs faster than *MATLAB* [92, 38]. Our latest UGV work [38] has a less complex OCP than this work, while using the same computer and the same class of collocation methods[1] as this work. Thus, the OCP solve-times obtained in this chapter and our previous work can be compared to help evaluate the ability of the respective software stacks to quickly solve complex OCPs. Our previous work uses *MATLAB* in conjunction with the *IPOPT* NLP solver to solve a UGV planning problem. To illustrate a shortcoming of this work, Table B.1, in Appendix 6.2, summarizes long solve-times obtained using this software stack and hard and soft constraints for obstacle avoidance to solve a single OCP in dynamic, unstructured environments with 3 and 17 obstacles. Similar research shows that solving real-time UGV planning problems using *MATLAB*

[1] In both cases local-collocation methods are used; this work uses the trapezoidal method and our previous work uses Euler's backward method.

and *IPOPT* is challenging [92] — planning problems are solved up to 30 times slower than real-time with a 2.90GHz Intel Xenon processor and a 0.5 s execution horizon. This chapter shows that solving UGV OCPs, using a direct-collocation method implemented in `NLOptControl` [37] in conjunction with the *KNITRO* NLP solver, makes real-time solutions feasible. Additionally, unreported tests in E_A-E_C indicate that using `NLOptControl` in conjunction with the open-source *IPOPT* NLP solver yields similar solve-times. Therefore, this step forward for real-time UGV planning can be attributed to the novel design specifications of `NLOptControl` and not the *KNITRO* NLP solver.

As the number of obstacles increase, the environment becomes more challenging, because the number of obstacles directly affects the computational load [20]. In the formulation developed in this work, the NLP dimensions grow linearly as the number of obstacles increases. Thus, increasing the number of obstacles from 3 (in E_A and E_B) to 38 (in E_C) enlarges the size of the NLP and the computational load. It is reasonable to assume that this increase is a major factor in the corresponding increase of solve-times. To see this increase in solve-times, compare Fig. 5.13 and Fig. 4.4 to Fig. 4.7. Increasing the number of obstacles may result in a loss of real-time solutions. However, several approaches may be taken to use the formulation presented in this chapter in an environment with many obstacles. These approaches include: developing a strategy that considers a smaller number of obstacles at a time, grouping several small, closely packed obstacles as a single obstacle, increasing computational power, or some combination of these.

For a given UGV, as obstacle speed increases, the environment becomes more challenging, because the vehicle is put in an increasingly difficult situation. The data plotted in Fig. 4.8 support this claim; even with a moving obstacle avoidance specification, it is not possible to reliably avoid the oncoming obstacle 0_2 in E_C when it is moving faster than 21.1 $\frac{\mathrm{m}}{\mathrm{s}}$.

A moving obstacle avoidance specification may be unnecessary if the planner is updating quickly and the obstacles are moving slowly. The data plotted in Fig. 4.8 also support this claim. The data reveals that if the obstacle is moving directly toward the vehicle at a speed less than 2.11 $\frac{\text{m}}{\text{s}}$, and if the execution horizon is less than 0.375 s, then a planner without a moving obstacle avoidance specification will safely attain the goal. Removing the moving obstacle avoidance specification will also eliminate the need for an algorithm to predict the speed of the obstacles. This simplification may be appropriate for some industrial applications, where the obstacles are known to move slowly.

In addition to a moving obstacle avoidance specification, as obstacle speed increases, a small execution horizon becomes increasingly crucial for safety. The data plotted in Fig. 4.8 supports this claim as well; over a range of obstacle speeds, planning with a smaller execution horizon makes it more likely that the vehicle attains the goal. It is therefore desirable to make the execution horizon as small as possible in order to create a more reactive and safer planner. Having a small execution horizon, however, makes it more difficult for the planner to obtain solutions in real-time.

In order to ensure that the planning solutions are obtained in real-time while maintaining safety, it may be necessary to operate the UGV within environments where the obstacles are traveling from low to moderate speeds. Particular sets of data plotted in Fig. 4.8 support this claim as well. Specifically, when disregarding the cases where the obstacle is traveling faster than 16.8 $\frac{\text{m}}{\text{s}}$, the RTF decreases to 0.933 and the POS increases to 0.889. Similarly, when disregarding the data where the obstacle is traveling faster than 4.21 $\frac{\text{m}}{\text{s}}$, the RTF is further reduced to 0.920 and the POS increases to 1.

In addition to reducing the execution horizon, planning in a dangerous situation can increase the RTF. This results from the fact that planning in a dangerous situation can lead to less feasible or even infeasible NLP constraints, which make it more

challenging or even impossible for the NLP solver to obtain a solution. The top trace of Fig. 4.7 supports this claim; it shows that solve-times increase sharply just before V_D avoids a collision with 0_2. It is important to consider these situations in terms of solve-time; if the planner cannot obtain a trajectory within the real-time limit, then the vehicle will not have a trajectory to follow and the situation status will go from dangerous to disastrous.

4.6 Conclusion

This chapter incorporates planner specifications S_1-S_7 (listed in Table 1.1) into an NMPC-based trajectory planner for a UGV. UGV safety and performance is tested within four simulation-based comparisons. The results show that

- planners with less comprehensive sets of specifications than S_1-S_7 reduce UGV safety and performance,
- if the planner is updating quickly, then a slowly moving obstacle can be safely avoided without a moving obstacle avoidance specification,
- to avoid faster obstacles, both the moving obstacle avoidance and small execution horizon specifications are necessary,
- a small execution horizon improves safety, but decreases the feasibility of obtaining trajectories in real-time, and
- planning in an environment with more obstacles increases OCP solve-times.

Contrary to our expectations, our results show that adding the minimum-time-to-goal, minimum control effort, and moving obstacle avoidance specifications does not lead to larger solve-times. In fact, adding a minimum-time-to-goal specification actually reduces planning solve-times in the simple, static, unstructured environment. For our

final research objective, the first three comparisons show that `NLOptControl` solves the OCP formulations, with a minimum-time-to-goal specification, in real-time, i.e., the solve-times are all less than the chosen execution horizon of 0.5 s. In contrast, previous work [38, 92] shows that *MATLAB* cannot solve OCP formulations that have a similar level of complexity in real-time. Therefore, `NLOptControl` is found to be a suitable tool for quickly solving complex OCPs. While this work tailors the NMPC-based trajectory planner for a UGV application, a variety of automated vehicle systems, e.g., UAVs and spacecraft, can also make use of the approach detailed here.

CHAPTER V

Real-time Trajectory Planning for Automated Vehicle Safety and Performance in Unknown Dynamic Environments

5.1 Introduction

Achieving safe and performance-based control of automated vehicles in unknown, dynamic environments is challenging. Part of the challenge is identifying and combining suitable planning and perception algorithms. For safety and performance, a suitable planning algorithm must fully exploit the vehicle's maneuvering capacity to avoid collisions, especially for large, high-speed, automated-vehicle platforms such as a racecar, HMMWV, or a passenger car in an emergency situation. Specifically for safety, a suitable perception algorithm must accurately predict obstacle motion using sensor data. While existing combinations of planning and perception algorithms allow operation of automated vehicles in unknown dynamic environments, a combination has yet to be established that ensures safety and performance.

Table 1.1 lists the specifications of the planner that are most critical for automated vehicle performance and safety in dynamic environments. For safety in dynamic environments, the planner should include a static and moving obstacle avoidance specification (S_1, Table 1.1). To improve the vehicle's ability to avoid collisions with obsta-

cles, the planner should include a dynamic vehicle model specification (S_2, Table 1.1), which helps ensure that the planned trajectories are dynamically feasible [44, 45]. For performance, the planner should include a minimum time-to-goal specification (S_3, Table 1.1), which in conjunction with a dynamic vehicle model specification, exploits the vehicle's maneuvering capacity to attain the goal as quickly as possible. For additional performance, a minimum control effort specification (S_4, Table 1.1) should be included to reduce both mechanical wear and fuel consumption. To further improve safety and performance, the planner should simultaneously optimize speed and steering (S_5, Table 1.1). This specification permits the planner to exploit the vehicle's maneuvering capacity better to avoid collisions with obstacles [92] and attain the goal as quickly as possible. Finally, for safety, the planner should have a fast update rate (i.e., small execution horizon, S_6, Table 1.1), which makes the planner responsive to sudden changes in the environment. However, formulating a planning algorithm with S_1–S_5 and solving it quickly is difficult. Fortunately, `NLOptControl`, the direct-collocation-based optimal control software tool developed in Chapter III, may enable comprehensive trajectory planning algorithms to be easily formulated and quickly solved. Thus, as an additional research task, this work evaluates `NLOptControl`'s ability to solve comprehensive trajectory planning problems in unknown environments quickly.

Safety- and performance-based planning algorithms [92, 38] have been combined with perception algorithms to operate in unknown static environments [92], but not in unknown dynamic environments. In [92], a relatively simple perception algorithm is developed to enable automated vehicle testing in an unknown static environment. Unfortunately, this simple perception algorithm cannot be easily modified to account for moving obstacles. Combinations of planning and perception algorithms exist that permit mobile robots [39] and manipulator arms [134] to operate in unknown dynamic environments. However, neither [39], nor [134] explicitly considers the vehicle's dy-

namics in the planning algorithm. As previously observed [44, 45], this simplification renders the planning algorithm unsafe, especially in the types of applications specified above.

The ultimate goal of this research is to ensure the safety and performance of automated vehicles operating in unknown, dynamic environments. To this end, this work combines a safety- and performance-based planning algorithm [91, 92, 38] with a suitable perception algorithm; which is a recently developed Kalman-filter-based algorithm that provides estimates of obstacle size, position and velocity using raw light detection and ranging (LiDAR) data [115]. Planning algorithms are typically developed under the assumption of a known environment. Therefore, as an additional goal, this research statistically quantifies various safety and performance factors that are expected to deteriorate when the assumption that the environment is known is relaxed. These factors are obstacle avoidance ability, tracking error, orientation error, planner solve times, time-to-goal, steering effort, throttle effort, and braking effort. For this quantification, this chapter assumes that

- the goal location is known,
- the vehicle state is known, and
- the terrain is flat.

The remainder of this chapter is structured as follows: Section 5.2 describes the automated vehicle framework developed to combine the planning and perception algorithms. Section 5.3 describes the test conditions under which the proposed automated vehicle system is evaluated. Section 5.4 compares the performance and safety factors of the automated vehicle system operating in a known to those of an unknown environment. The results of this comparison are discussed in Section 5.5. Section 5.6 summarizes the chapter and draws conclusions.

5.2 Methods

This chapter combines planning [91, 92, 38] and perception [115] algorithms to enable safety and performance for automated vehicles operating in unknown dynamic environments. To this end, this chapter develops a high-fidelity, simulation-based, proving ground framework. This framework is developed in ROS to move towards experimental testing. This section describes the design of this framework.

5.2.1 Automated vehicle framework

Many automated vehicle frameworks are tailored for urban settings, otherwise known as a structured environment. In a structured environment there specific constraints, such as traffic rules to follow, which are accounted for using hierarchical control schemes. However, the automated vehicle framework developed in this work is tested in an unstructured environment. In an unstructured environment, there are no traffic rules to follow. As such, this section first describes the framework developed in the context of a structured environment. This description illustrates how a hierarchical control scheme in a structured environment can use the framework developed in this work.

In urban settings, control of automated ground vehicles can be accomplished using a hierarchical structure with four layers [108]; route planning, behavioral layer, motion planning, and local feedback control. These layers are shown conceptually in Fig. 5.1a and are briefly described for completeness. Afterward, the reduced framework that is used in this work, which is depicted in Fig. 5.1a, is discussed. This reduced framework eliminates route planning and behavioral layers.

Given a goal location $\boldsymbol{\mathcal{G}}$ a route planning algorithm, such as the Transfer Pattern [5] technique[1], computes a route information $\boldsymbol{\mathcal{R}}$ and typically includes a sequence of way-points through a road network [6]. With $\boldsymbol{\mathcal{R}}$ and environment estimates $\boldsymbol{\mathcal{E}}_{est}$,

[1] which has been used in Google Maps since 2010 [6]

such as obstacle-free space, the behavioral layer identifies motion specifications $\boldsymbol{\mathcal{M}}$ and may identify if the vehicle should change lanes, obey traffic rules, or slow down near pedestrians, etc.

The motion planner is then provided motion specifications $\boldsymbol{\mathcal{M}}$, environment information estimates $\boldsymbol{\mathcal{E}}_{est}$, and plant state estimate **X0est**, to calculate either a reference signal for the local feedback control layer. This reference signal may be a reference path **Xref**, or state trajectory **Xref**(t), or control trajectory **Uref**(t) [5]. Depending on the design of the motion planner desirable properties of **Xref**/**Xref(t)** include collision-free, dynamically feasible, minimum-time, and minimal control effort. Path planning involves finding a feasible and optimal path which entails meeting the designer's constraints and minimizing the terms in the objective functional. Popular path planning methods include geometrical methods [19] and sampling based methods [75]. A limitation of path planners is that they do not tell the local feedback controller how to follow **Xref**, because no temporal information is included. Therefore, following a reference path **Xref** may not be safe as it does constrain the vehicle dynamics to be feasible and the environment may not be static. When small automated ground vehicles are operated in 3D static environments at low to moderate speeds with a conservative control input, incorporating kinematic constraints into path planner may be suitable [82]. However, in emergency situations, military missions, or racing applications, the environment is often dynamic and it may be necessary to operate automated ground vehicles at high-speeds with control inputs that push the vehicle to its dynamical limit. Trajectory planners are better suited for this type of application because **Xref(t)** provides a temporal component. This work focuses on military applications and develops a controller for an HMMWV vehicle. The HMMWV is a large vehicle that has a high center of gravity and operating it at high-speeds causes to rollover be a major issue. In addition to rollover, it is desirable to operate HMMVs in unstructured environments. Typically motion planners are designed for urban en-

vironments [76, 108] and relatively less research and development work specifically for unstructured environments [131, 22, 82]. As mentioned, in an unstructured environment, there are no roads to follow or traffic rules to obey. While in some ways this makes the motion planning problem easier, it also opens up new challenges. For instance, with no roads to follow the design space for identifying trajectories or paths is significantly larger

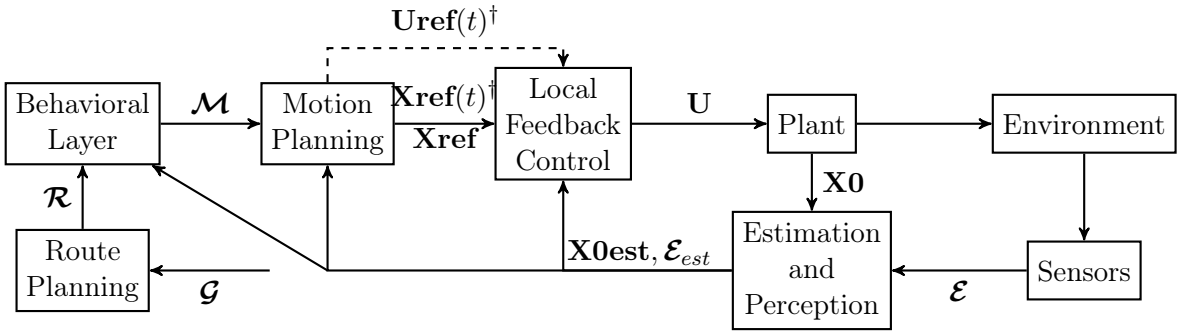
When this reference signal, a state estimate **X0est**, and an estimate of the environment information $\boldsymbol{\mathcal{E}}_{est}$, the local feedback control layer calculates the actuator control signals **U**. The goal of the local feedback control layer is to stabilize the reference signal to ensure robustness and correct for both modeling errors and uncertainties. Effective techniques used in the local feedback layer include pure pursuit [21], linear MPC, and nonlinear MPC controllers. Ultimately, the actuator control signal **U** induces the plant state **X0** to evolve in the environment.

The remaining blocks in Fig. 5.1a account for the perception and estimation aspects of the automated vehicle problem. The sensors produce the environmental data $\boldsymbol{\mathcal{E}}$, which includes both perception sensors, such as LiDARs and cameras, and state sensors, such as GPS, IMU, and wheel encoders. Then the estimation and perception algorithms process the environmental data $\boldsymbol{\mathcal{E}}$ to produce state estimates **X0est** and environment information estimates $\boldsymbol{\mathcal{E}}_{est}$.

5.2.2 Scope of this work

To narrow the scope of this work, route planning, and the behavioral layer in the framework in Fig. 5.1a are omitted and the focus is the rest of the framework. Additionally, this work considers that the automated ground vehicle is operating in an open area such as a field, so there are no streets to follow or traffic rules to obey. With that, both the route planning and behavioral layers are removed and the goal location $\boldsymbol{\mathcal{G}}$ is be passed directly to the motion planning layer. Finally, the actual

(a) Hierarchical control structure used for automated ground vehicles in urban environments. † designates trajectory planning inputs, as opposed to path planning input.



(b) Hierarchical control structure used in this work.

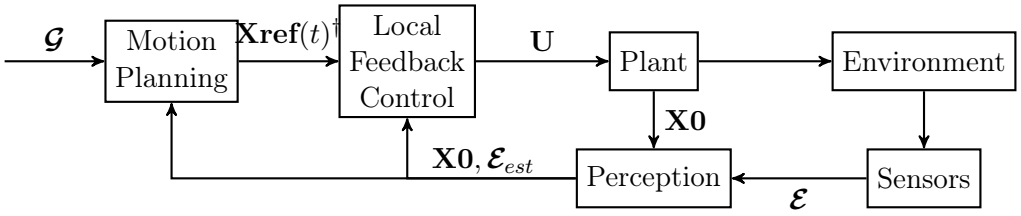


Figure 5.1: Hierarchical control structures used for automated vehicles

plant state **X0** is assumed to be known.

This reduced framework is depicted in Fig. 5.1b. The four major components of this framework include the plant, the local feedback controllers, the motion planning algorithm, the perception algorithm, and the environment. A 145 DOF HMMWV model is created using Project Chrono to realize the plant. For the feedback, controls consist of a pure pursuit algorithm and a PID controller. Together, these controllers send steering, throttle and brake control signals **U** to the plant that follow the position and speed trajectories, which are included in the reference trajectory **Xref(t)** and determined using the planning algorithm. The planning algorithm uses the plant state **X0** and estimates of the obstacles' size and speed $\boldsymbol{\mathcal{E}}_{est}$. The perception algorithm produces obstacle estimates $\boldsymbol{\mathcal{E}}_{est}$ using data collected in the environment $\boldsymbol{\mathcal{E}}$. The environment is modeled using the Gazebo simulator. This environment model includes a 2D LiDAR model and the vehicle's position size and orientation.

A scenario consists of a goal location $\boldsymbol{\mathcal{G}}$, an obstacle field, and an initial plant state. Provided a particular scenario, the framework in Fig. 5.1a runs until either the vehicle attains the goal or the test fails. The test fails if

- the vehicle crashes into an obstacle,
- two of the tires on the plant lift off of the ground,
- any of the solve times exceeds 10 s,
- the overall simulation time exceeds 500 s,
- the solution to the nonlinear programming problem (NLP) does not meet the Karush-Kuhn-Tucker conditions for optimality, or if
- the vehicle travels outside of the prescribed test track area in Chrono.

The remainder of this section further describes the perception algorithm, LiDAR model, collision detection, plant model, planning algorithm, and the local feedback controller.

5.2.3 Perception algorithm

The perception algorithm [115] used in this work processes raw 2D LiDAR data to detect and track obstacles. Detected obstacles are represented using circular models. A Kalman filter enables tracking of these circular models, which produces an estimate of the obstacle's velocity. Fig. 5.2 illustrates the accuracy of the position estimates determined by the perception algorithm. However, this figure illustrates that even though the obstacles are circles the estimated obstacle position is slightly forward of the actual obstacle position. This is an artifact the design of the algorithm which centers the obstacles position between the two furthest points for each obstacle.

This work uses an implementation of this perception algorithm [115] that has been tested in physical experimentation. This implementation that uses raw 2D LiDAR

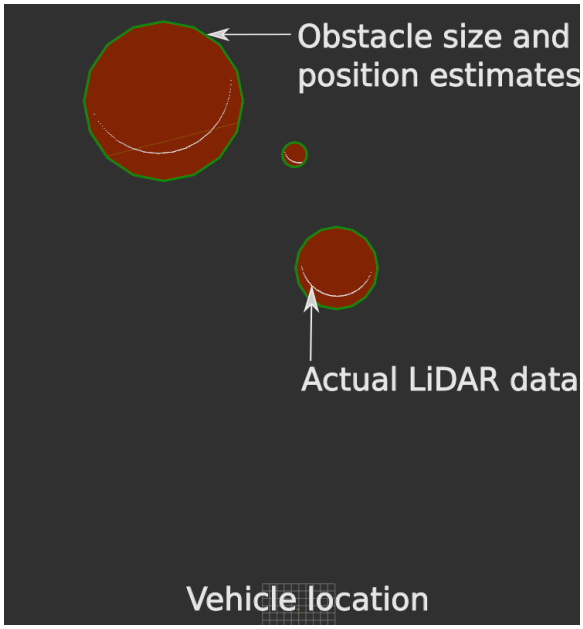


Figure 5.2: Estimating obstacle data using raw LiDAR data and the Kalman filter-based perception algorithm [115]

data to estimated environment information $\boldsymbol{\mathcal{E}}_{est}$. Unfortunately, at the time of experimentation, the author did not have access to an automated HMMWV vehicle testbed; thus, the raw 2D LiDAR data necessary for this implementation is generated in the simulation. Fortunately, this type of data is produced using the Gazebo simulation environment.

5.2.4 LiDAR model and collision detection

Fig. 5.3 illustrates the Gazebo simulation environment that models a 2D LiDAR device, the obstacles, and the vehicle's size, position (x, y), and orientation ψ. The state of the actual plant model is used to update the vehicle's position and orientation. This design permits the LiDAR model to accurately gather data while moving among the obstacles. The LiDAR has a range of 100 m and sweeps with 1875 samples at points 180 degrees in front of the vehicle. Gaussian noise is added to the LiDAR model with a mean of 0 m and a standard deviation of 0.01 m. The remaining LiDAR

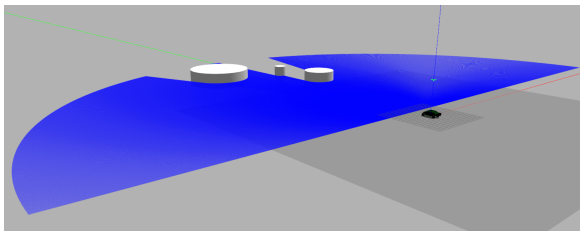


Figure 5.3: Modeling the LiDAR, the obstacles, and collisions between the vehicle and the obstacles using Gazebo

parameters are summarized in Table C.4, which is in Appendix 6.2. A contact sensor plug-in is added to Gazebo to enable collision detection between the vehicle and the obstacles.

5.2.5 Plant model

Fig. 5.4 depicts the high-fidelity HMMWV model adapted from Chrono [117] that models the plant. This plant simulation in Chrono consists of an FEA-based model for the terrain and the wheels that interact with a 145 degree of freedom vehicle dynamics model. This vehicle dynamics model is rear-wheel drive and includes a full double wishbone suspension, a pitman steering arm mechanism, and a model powertrain model. Steering $\in [-1, 1]$, throttle $\in [0, 1]$, and brake $\in [0, -1]$ commands $\mathbf{U}$ control the plant. The Chrono model and vehicle parameters are in Table C.2 and Table C.3, respectively, these tables are in Appendix 6.2. Unfortunately, the plant model simulation runs about two times slower than real-time. Therefore, the Gazebo simulation time is advanced based off of the Chrono time and the ROS time is advanced based off of the Gazebo time. Additionally, the planner is not provided this additional time. If the cumulative planner solve time exceeds the current simulation time, the planner is forced to wait until this condition is false before commencing another optimization.

Figure 5.4: High-fidelity HMMWV model in Chrono used to model the plant [128].

5.2.6 Planning algorithm

The optimal control problem formulation used in this work is tailored for a HMMWV and is developed in [91, 92, 38]. A 3DOF vehicle model [91] is used to realize the dynamic vehicle model specification. This model has eight states, two control, uses a pure-slip Pacejka tire model [107], considers the longitudinal load transfer effects when determining the vertical tire force, and constrains the vertical tire load to remain above a minimum vertical tire load limit $F_{z_{min}}$. Table C.4, which is in Appendix 6.2, contains the key planning parameters used in this chapter. Since the focus of this chapter is not to develop a planning algorithm, the optimal control problem developed in [91, 92, 38] is omitted here. However, this work does provide the most realistic test of this type of planning algorithm. This work tests a planning algorithm with specifications S_1-S_5 is tested (1) with a high-fidelity plant model, (2) on a distributed ROS-based system, (3) and most importantly in conjunction with a perception algorithm to enable operation in an unknown dynamic environment.

Fig. 5.5 shows the planning animation that can be displayed as the simulation is running. This animation shows the plant trajectories, the current planned trajectory, and the solve-times. This figure shows that planning solutions are obtained at around 0.5 Hz. Unfortunately, this update rate is not fast enough such that the planner algorithm can directly send control signals to the plant. Additionally, the plant is

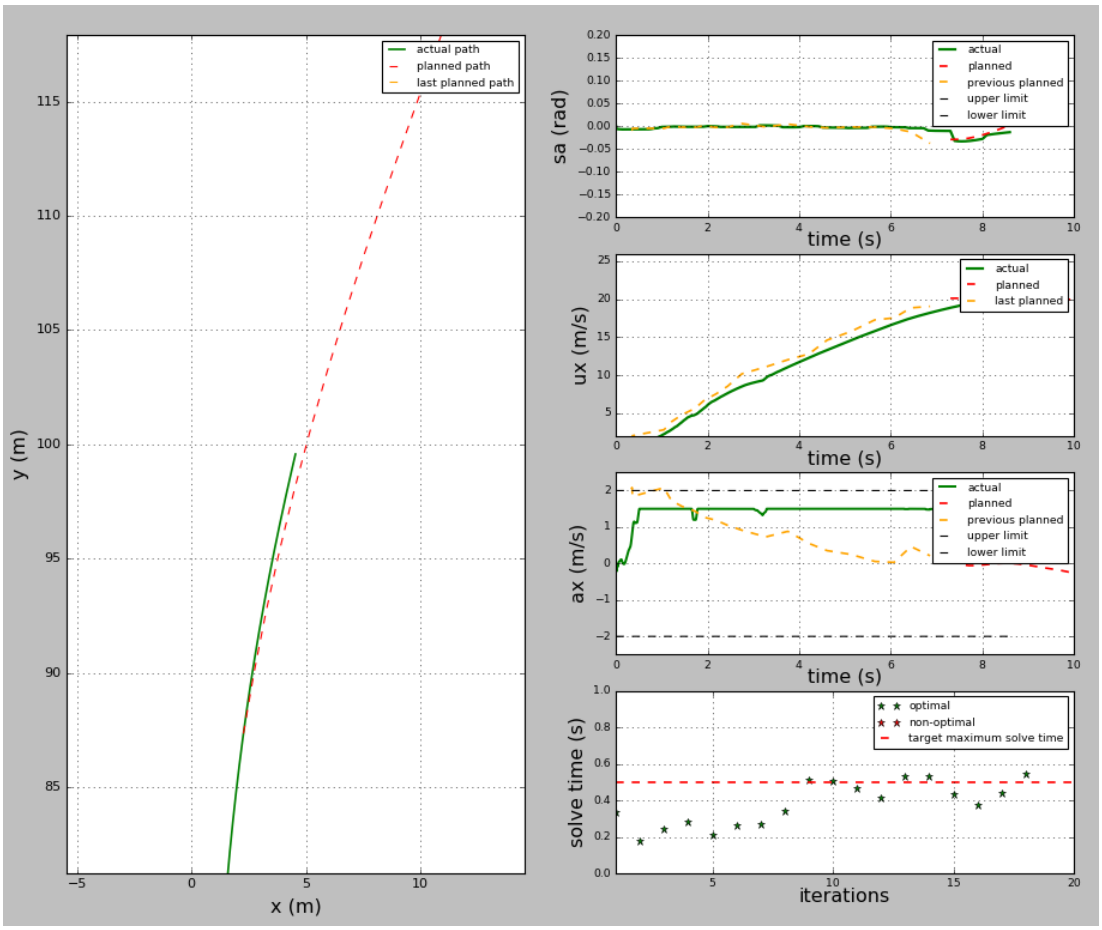


Figure 5.5: Animation showing (1) the current position, steering angle, longitudinal speed and acceleration trajectories calculated with the planning algorithm compared against the respective plant trajectories, and (2) the planning solve times.

controlled using a control signal **U** that consists of steering, throttle, and braking signals. Therefore, the position and speed trajectories **Xref(t)** are sent to a path tracking algorithm to quickly produces appropriate plant control signals **U**.

5.2.7 Local feedback controller

Two controllers process the latest trajectory **Xref(t)** to calculate the plant control signals **U**. A PID controller uses the speed trajectory to calculate appropriate throttle $\in [0, 1]$, and brake $\in [0, -1]$ commands to send to the plant. Additionally, a pure pursuit path tracking algorithm [21] processes the position trajectory **Xref(t)** to

determine the the steering command $\in [-1, 1]$ to send to the plant.

5.3 Evaluation Description

The next section evaluates the change in performance and safety factors when moving from a known environment to an unknown environment. This section describes the test track, the performance and safety factors, and the computer hardware and software specifications that produce the results presented in this chapter.

5.3.1 Test track

Fig. 5.6 shows the test track used to evaluate the perception algorithm's ability to enable the planning algorithm to operate in an unknown environment. This test track consists of four obstacles, which are denoted as 0_1, 0_2, 0_3, and 0_4, and a goal location. Table C.5, which is in Appendix 6.2, lists the base parameters of this test track.

To permit the perception algorithm to be evaluated under a wide range of conditions, this test track is parameterized by the radius r_1 and velocity vy_1 of 0_1. These test conditions are defined realized by 400 random combinations of r_1 and vy_1 are selected in the ranges of 1 to 10 m and -0 to -20 $\frac{\text{m}}{\text{s}}$, respectively. The negative on the obstacle's velocity indicates that it is in the negative y direction, coming directly at the vehicle's initial position, which can be seen in Fig. 5.6. Tests are performed for these test conditions in an environment that is known and then one that is unknown. The vehicles operating in the known and unknown environments are referred to as V_A and V_B, respectively. As the tests are conducted, the various safety and performance factors are recorded.

5.3.2 Performance and Safety Factors

These safety and performance factors are now described.

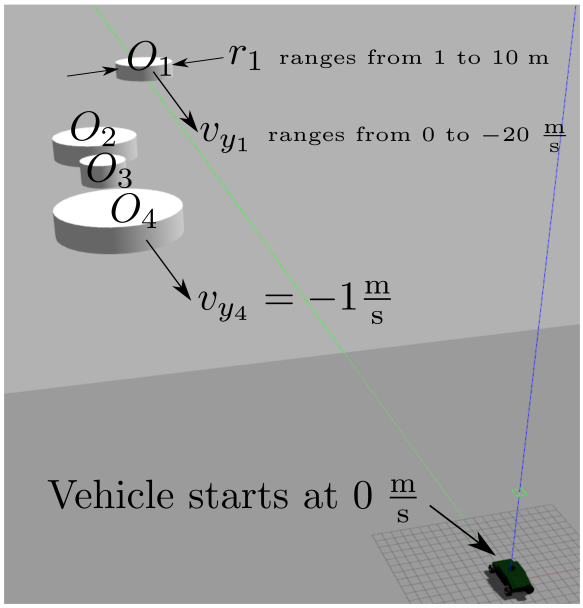


Figure 5.6: Test track that enables an evaluation of the Kalman-filter based perception algorithm [115]

obstacle avoidance ability This safety factor is either true or false, indicating if the vehicle collided into an obstacle or not.

tracking and orientation error These are both safety and performance factors. They are safety factors because, if the vehicle is not following the planned trajectory, then there is no guarantee that the obstacles will be avoided. They are performance factors because the trajectory that we determined for the vehicle has a minimum time to goal specification. By more closely following this trajectory the vehicle can be pushed to higher operating speeds.

Fig. 5.7 shows the way that the tracking and orientation error are calculated. This figure has three main variables, the orientation error Θ_ϵ, the tracking error χ_ϵ, the vehicle's current (x_v, y_v) and previous (x_{v_p}, y_{v_p}) (x, y) position state, and the closest two points on the current planned trajectory (x_{t_1}, y_{t_1}) and (x_{t_2}, y_{t_2}). As seen in the figure, these four points are used to calculate two lines. The magnitude of the angle

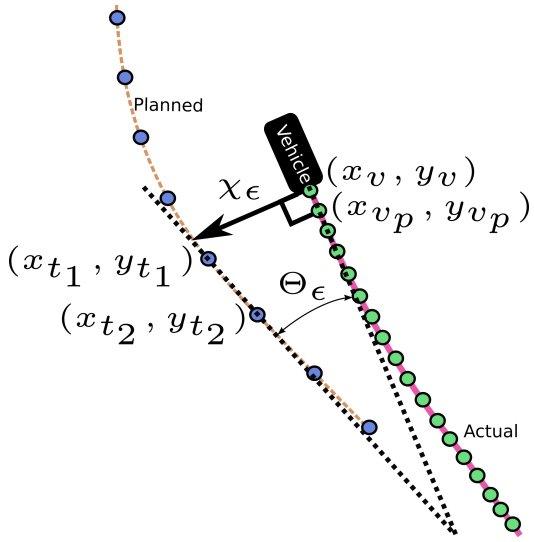


Figure 5.7: Calculating the orientation error Θ_ϵ and tracking error χ_ϵ

between these lines is calculated for the orientation error Θ_ϵ and the perpendicular distance from the vehicle's current state (x_v, y_v) to the planning line is the tracking error χ_ϵ. This algorithm is run at 5 Hz.

planner solve time Averaging all of the planners solve times determines the planner solve time.

time to goal The amount of time that it takes for the vehicle to attain the goal is the time to goal.

steering effort Integrating the square of the steering signal that is sent to the Chrono vehicle determines the steering effort.

throttle effort Integrating the square of the throttle signal that is sent to the Chrono vehicle determines the throttle effort.

braking effort Integrating the square of the braking signal that is sent to the Chrono vehicle determines the braking effort.

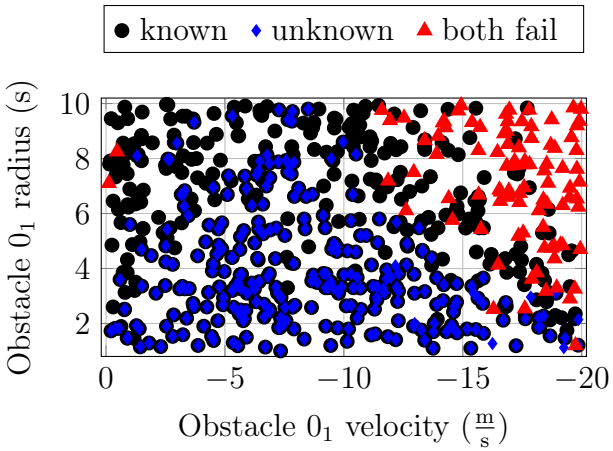


Figure 5.8: Effect of obstacle size and speed on attaining the goal in both an unknown and a known environment.

5.3.3 Hardware Platform and Software Stack

The results in this chapter are produced using a single machine running Ubuntu 16.04 with an Intel Core i7 − 3770MQ CPU @2.90GHz × 8, and 15.6GB of RAM. In this work `NLOptControl` 0.1.6+ [37] configured with the trapezoidal method [10, 78] direct-collocation solver is used with the *KNITRO* 10.3 NLP solver to solve the planning problems.

5.4 Results

Operating in an unknown environment reduces the vehicle's ability to attain the goal over a range of obstacle sizes and speeds. To illustrate this effect, tests are performed on the test track, which is described in the previous section, in both an unknown and a known environment for 600 random sizes and speeds of obstacle 0_1. Fig. 5.8 summarizes the the data from this parameter sweep indicating if the vehicle attains the goal in the known environment, the known environment, or not in either the known or unknown environment.

Out of the 600 cases tested, there are 264 cases where the vehicle attains the goal in both the known and unknown environments. Fig. 5.9 illustrates one of the cases

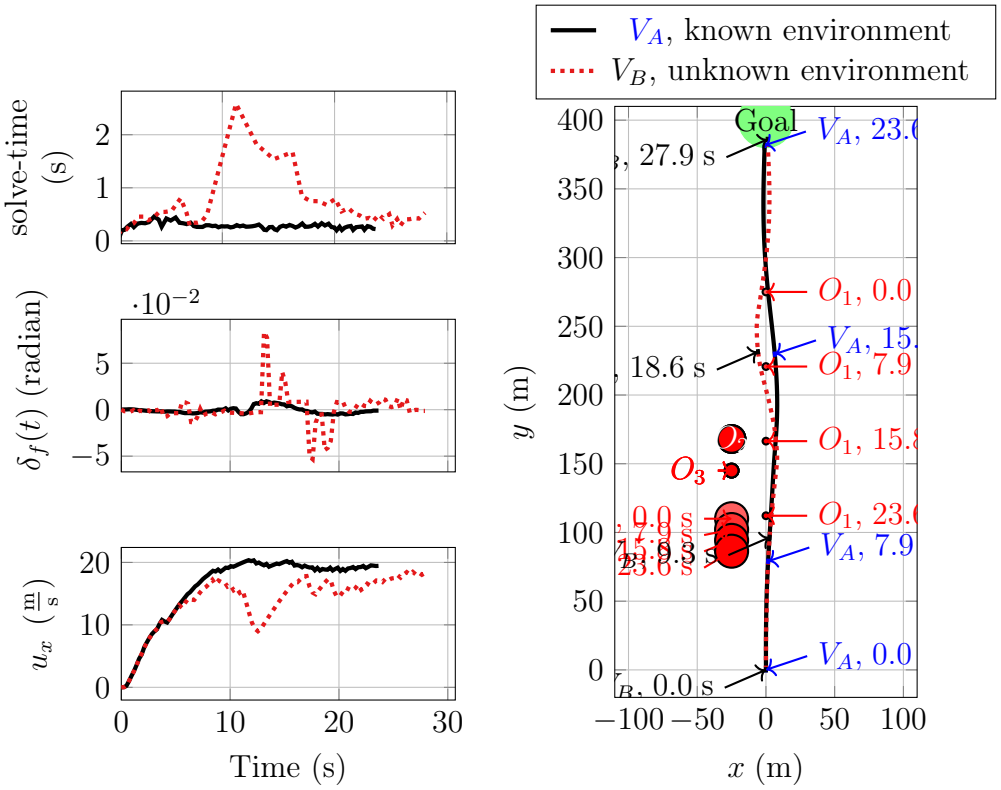


Figure 5.9: A case with a small obstacle traveling at moderate speeds, where the goal is attained in both known and unknown environments.

that the vehicle attains the goal in both the known and unknown environments. In this case, 0_1 has a 2.44 m radius and is traveling at a velocity of -6.89 $\frac{\text{m}}{\text{s}}$. While both vehicles attain the goal, the vehicles take different trajectories and have different solve times. For this case, operating in an unknown environment increases both solve times and steering effort.

Overall, the results from this parameter sweep indicate that operating in an unknown environment degrades all of the performance and safety factors. Fisher tests show that operating in an unknown environment significantly increases ($p \ll 1 \times 10^{-6}$) collisions, tracking error, orientation error, planner solve-times, time to goal, steering effort, throttle effort, and braking effort. Only the data from the cases where the vehicle attains the goal in both the unknown and known environment are used to calculate the safety and performance factors, except for the collision statistics.

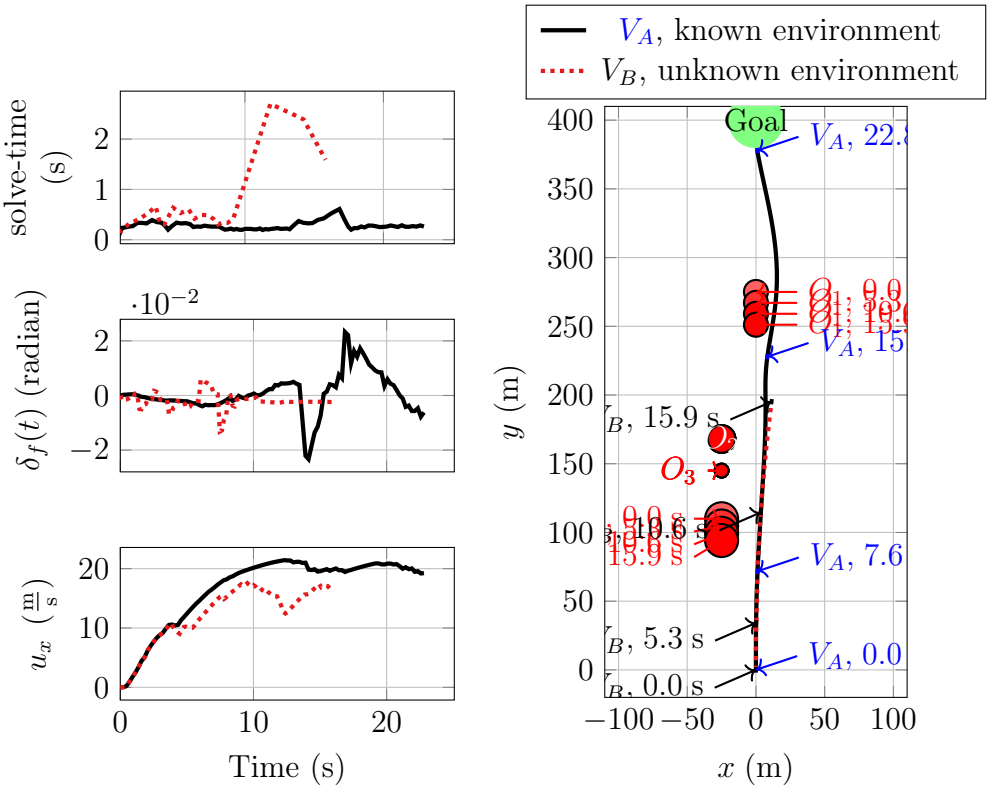


Figure 5.10: A case with a large obstacle traveling at slow speeds, where the vehicle attains the goal in an known environment, but not in an unknown environment, because a non-optimal planning solution is determined.

Fig. 5.8 indicates that it is harder to avoid large and slow obstacles in an unknown environment, than a known environment. Consider one of these cases in Fig. 5.10, where 0_1 has a radius of 8.72 m and is traveling at a velocity of -1.5 $\frac{\text{m}}{\text{s}}$.

Fig. 5.8 indicates that the vehicle operating an unknown environment with medium sized vehicles traveling slowly, does not attain the goal. Consider one of these cases in Fig. 5.11, where 0_1 has a radius of 1.89 m and is traveling at a velocity of -0.63 $\frac{\text{m}}{\text{s}}$. There is a large spike in the planner solve-times and the actual reason for failure is that the planner did not determine an optimal solution, as required. It is important to remember that as the vehicle is driving, the planner is calculating the next trajectory. Therefore, in Fig. 5.11, while it appears that the planner should be able to calculate a feasible trajectory, the optimization problem that is infeasible was initialized almost 2 s before the 19.2 s. It is also important to keep in mind

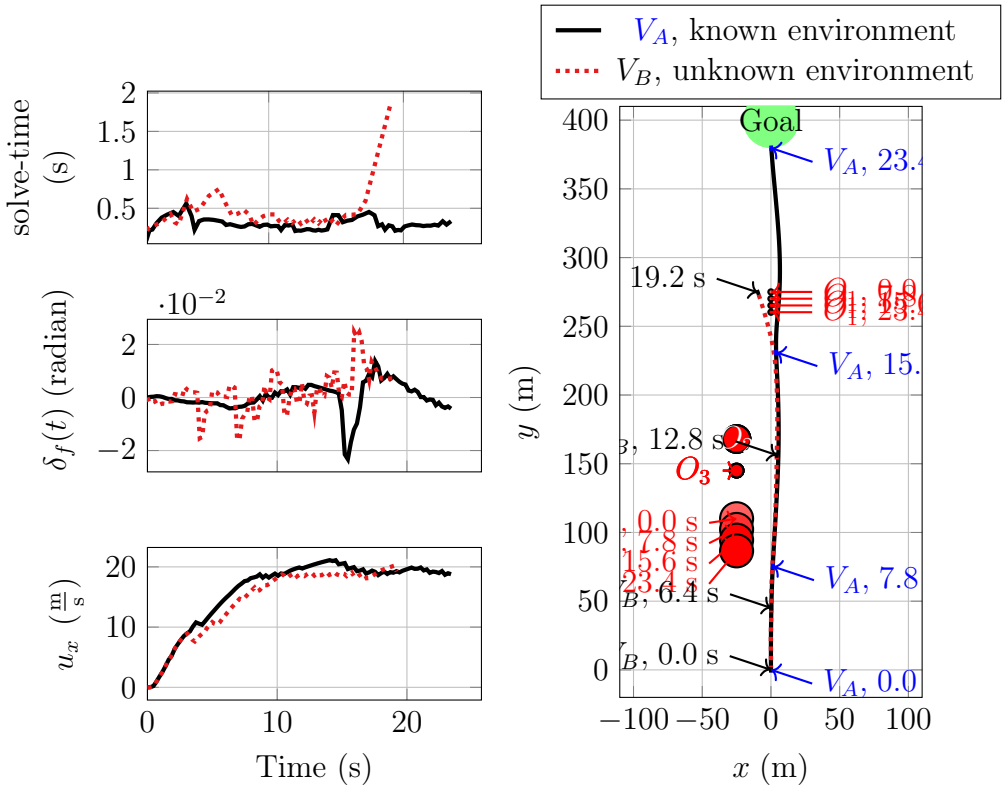


Figure 5.11: A case with a medium sized obstacle traveling slowly, where the goal is not attained in an unknown environment

that the optimization problems in the unknown environment are solved using estimates of the obstacle data, which are different than the actual obstacle data plotted in these figures. The following section provides a possible explanation for why the vehicle better attains the goal when the obstacles are moving at moderate to fast speeds in an unknown environment, than when they are either stationary or moving slowly.

Fig. 5.8 shows several anomalous cases where the vehicle does not attain the goal, when a medium sized obstacle is traveling at moderate speeds in an unknown environment. Consider one of these cases in Fig. 5.12, where 0_1 has a radius of 4.2 m and is traveling at a velocity of -7.78 $\frac{\text{m}}{\text{s}}$. As seen, the vehicle in the unknown environment misses the goal, which is likely due to the large planning solve-times. Ultimately, the reason for failure is that simulation is stopped because the actual time

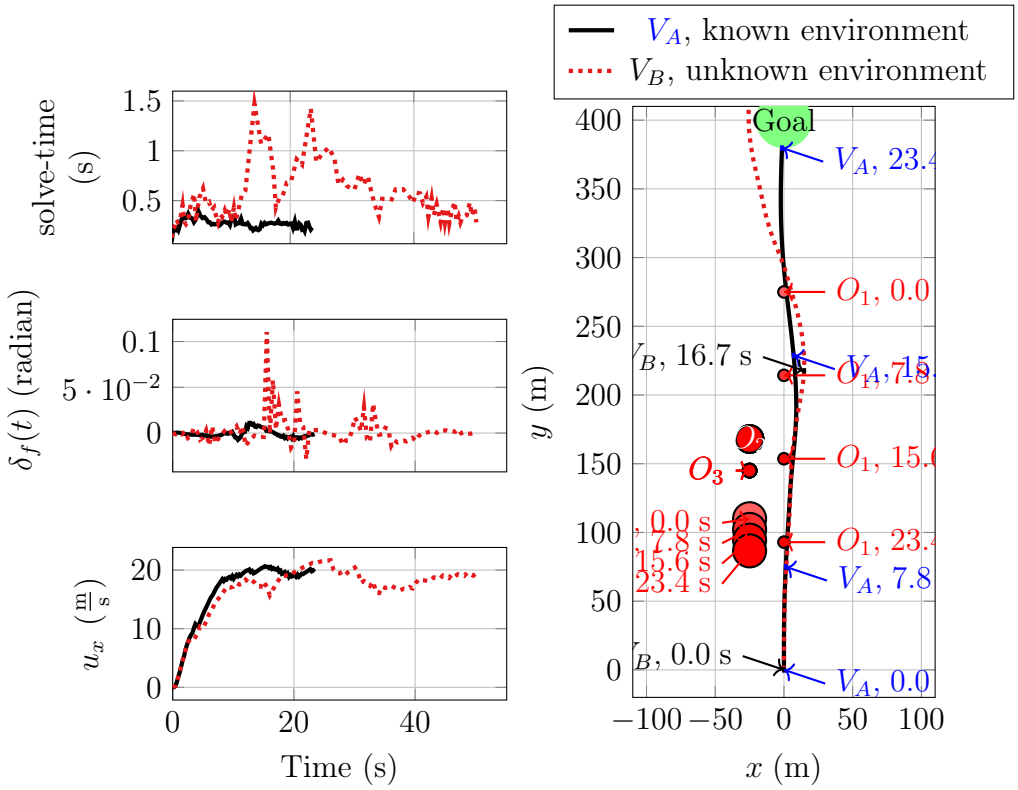


Figure 5.12: A case with a medium sized obstacle traveling at moderate seeds, where the goal is missed in an unknown environment, but attained in an known environment

exceeds 50 s. The other failure cases in this region fail in this way as well.

Fig. 5.8 shows two anomalous cases where the vehicle does not attain the goal, when a medium sized obstacle is moving very slowly in a known environment. Consider one of these cases in Fig. 5.13, where 0_1 has a radius of 7.1 m and is traveling at a velocity of -0.12 $\frac{\text{m}}{\text{s}}$. The reason for failure in both of these anomalous cases is that an optimal solution is not determined by the planner. Fig. 5.13 shows that, just before 15 s, there is a spike in solve-time and the final two optimal solutions take nearly 1 s to calculate. These planning delays put the vehicle in such a dangerous situation, that the planner is not able determine an optimal solution and then, the simulation is terminated.

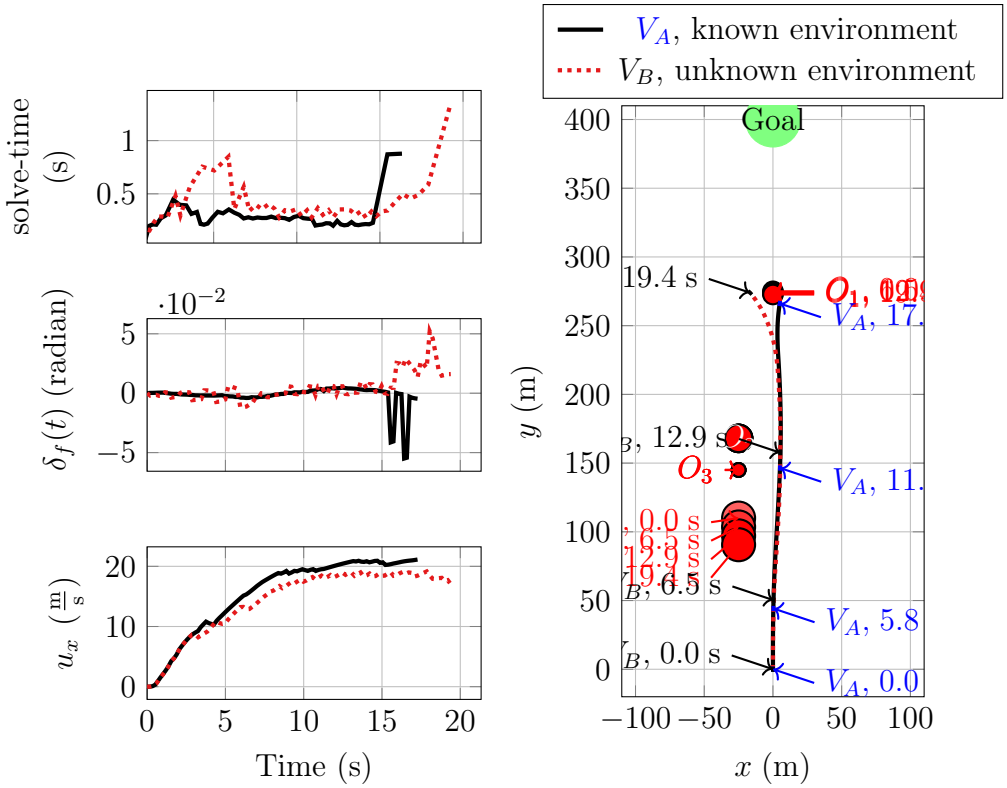


Figure 5.13: A case with a small sized obstacle is moving at very slow seeds, where an optimal solution is not determined in a known environment and the vehicle crashes in the unknown environment

5.5 Discussion

This work combines planning and perception algorithms to enable safe operation of large, high-speed, unmanned ground vehicles in unknown, dynamic environments.

Moving from a known to an unknown environment degrades various safety and performance factors. These factors include obstacle avoidance ability, tracking error, orientation error, planner solve-times, time to goal, and steering effort. The results in this work illustrate that moving from a known to an unknown environment degrades all of these safety and performance factors. This performance and safety gap will exist until perception algorithms can provide planning algorithms with better estimates of the size and speed of the obstacles in the environment. However, assuming that the environment is known makes it easier to develop and test new automated vehicles algorithms. Therefore, researchers assuming that the environment is known should anticipate that the performance and safety factors highlighted in this work will degrade. With this anticipation, special care should be taken to further improve these factors in known environments as well.

When obstacles are traveling close to the vehicle, the perception algorithm is more accurate at estimating the sizes of obstacles traveling at moderate speeds than those traveling at either low or high speeds. This claim is supported by Fig. 5.8, where it is illustrated that in an unknown environment there is a range of obstacles speeds, that does not start at zero, where the vehicle is most likely to attain the goal. However, the reason for most of these failure cases, as depicted in Fig. 5.10, is that the planner was not able to attain an optimal solution. Thus, the data in Fig. 5.8 and the results shown in Fig. 5.10 are not enough to make the claim that the perception algorithm is at fault for these cases. To make this claim, the perception algorithm's inability to send the planning algorithm accurate estimates of the obstacle data must be established.

To this end, the planning algorithm's ability to accurately estimate the radius of

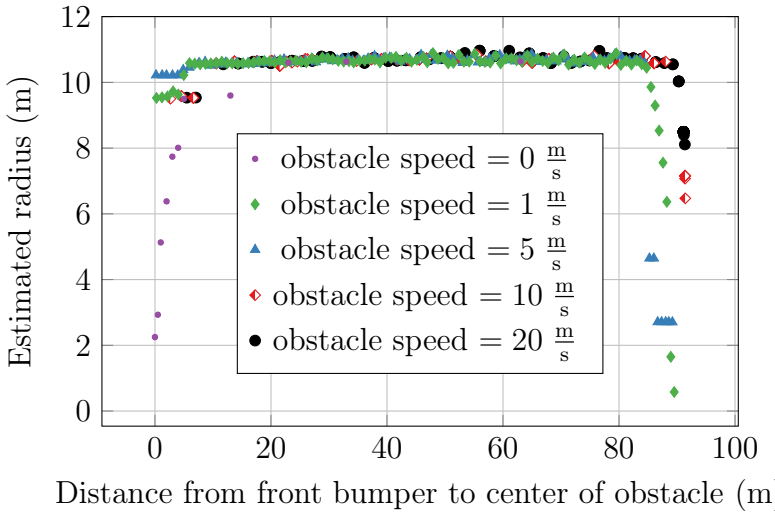


Figure 5.14: Effect that distance to the obstacle has on size estimates for various obstacle speeds

an obstacle with a 10 m radius is tested for five different obstacle speeds. Fig. 5.14 shows the results of these perception algorithm tests, which indicate that obstacle speed and distance both play an important role in the perception algorithm's ability to accurately determine obstacle data (i.e., size and speed). Recall that the perception algorithm processes raw LiDAR data to estimate obstacle data.

When the obstacle is stationary and close to the vehicle, obstacle size is underestimated. This underestimate is likely caused by the limited vantage provided by the radial LiDAR sensor, which is shown in Fig. 5.15. On the other hand, when obstacles are moving and close to the vehicle, obstacle size estimates are much more accurate. This increased accuracy is likely due to the fact that the perception algorithm is designed using a Kalman-filter, which uses obstacle size estimates that were determined using data when the obstacle was further away from the vehicle.

When the obstacles are just coming into the into view of radial LiDAR sensor, obstacle size is underestimated. This can also be understood by considering that the raw LiDAR data that is provided to the perception algorithm. As the obstacle enters the view of the LiDAR, it appears quite small because only a portion of the obstacle

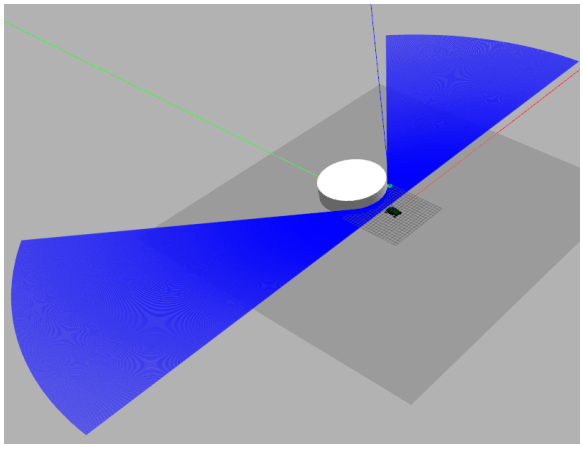


Figure 5.15: Limited data due to radial nature of LiDAR, which becomes more restricted as the obstacle gets closer

is actually within range of the sensor. Accordingly, as the obstacle gets closer to the vehicle, more LiDAR data is provided to the perception algorithm and thus more accurate obstacle size estimates are determined until the obstacle gets too close, as previously discussed. However, when the center of the obstacle is 17 m to 82 m away from the front bumper of the vehicle, the perception algorithm overestimates obstacle size.

Ultimately, the inaccurate estimates of obstacle size and speed make it less likely for the vehicle to attain the goal and decrease various safety and performance factors.

Large planning solve-times and non-optimal solutions are a major safety issue and strategies to mitigate and compensate for these spikes must be developed. Consider the two anomalous tests in Fig. 5.8 that failed in a known environment, when a medium-sized obstacle is moving at very slow speeds. These tests highlight how spikes in planning solve-times can lead to non-optimal solutions and ultimately safety issues. This work speculates that these failures are due to either the limited planning range, the limited level of discretization in the trajectory, a sampling-rate issue, or some combination of these factors. Therefore, these failure cases can likely be mitigated by tuning the planner's parameters. However, in addition to further parameter tuning,

safeguards need to be established to compensate for long planning solve-times and non-optimal solutions. For instance, in similar research [92], when a non-optimal solution is encountered, the next portion of the last optimal solution is sent to the vehicle controllers; such heuristics are not considered in this work. An alternative is having multiple planners running in parallel and using a management node to decide which trajectory to send to the vehicle controllers.

5.6 Conclusions

This work identifies and combines perception [115] and planning algorithms [91, 92, 38] to enable automated vehicle safety and performance in unknown dynamic environments. As expected the results illustrate that operating in an unknown dynamic environment, as opposed to a known dynamic environment, significantly increases[2] collisions, tracking error, orientation error, planner solve-times, time to goal, steering effort, throttle effort, and braking effort. Thus, this work identifies and quantifies significant gaps in safety and performance engendered when operating in an unknown environment and helps evaluate the known environment assumption. Contrary to expectations, results show that operating in an unknown environment among obstacles traveling at moderate speeds is safer than with obstacles moving at slow speeds. It is speculated that this result is in part due to the large underestimation of obstacle size when the obstacles are close to the vehicle and moving at slow speeds. To mitigate this safety issue, the use of more accurate perception systems should be explored. For instance, using more than one LiDAR sensor will enable to perception algorithm to more accurately estimate obstacle size. Ultimately, this work demonstrates that the real-time planning algorithm used in this work enables safety and performance in unknown environments among medium-sized and low-speed dynamic obstacles.

[2] $p \ll 1 \times 10^{-6}$

CHAPTER VI

Conclusion

This thesis develops a real-time trajectory planning algorithm to enable automated vehicle safety and performance in unknown dynamic environments.

Safety and performance are enabled by incorporating a dynamic vehicle model, simultaneous optimization of speed and steering, minimum control effort and time to goal objectives, both static and moving obstacle avoidance, and a small execution horizon into the planning algorithm. In a known environment, the moving obstacle avoidance specification is not needed for safety when the planner has a small execution horizon ($\leq$ 0.375 s), and the obstacles are moving slowly ($\leq 2.11 \frac{\text{m}}{\text{s}}$). However, a moving obstacle avoidance specification is needed when the obstacles are moving faster, and this specification improves the overall safety by a factor of 6.73 ($p = 2.2 \times 10^{-16}$) without, in most cases, increasing the solve-times. Therefore, safe and performance-based planning algorithms for automated vehicles should include the entire set of specifications mentioned above unless a static or low-speed environment permits a less comprehensive algorithm.

Real-time solutions to this comprehensive algorithm are obtained using `NLOpt-Control` [37], the open-source, direct-collocation-based optimal control modeling language that is created in this thesis. To generate this language, this thesis identifies and exploits a unique opportunity to leverage three recent developments (1) the

Julia computational language [13], (2) reverse automatic differentiation [121] with the acrylic coloring method [59] to exploit sparsity in the Hessian matrix, and (3) the *JuMP* optimization modeling language [30]. `NLOptControl` is shown to be both easier to use and faster than the commonly-used commercial optimal control solver called *PROPT* [112]. Ease of use is quantified by the number of lines and characters that it takes to formulate and solve the moon lander optimal control problem [99]. Additionally, `NLOptControl` was used to develop a new optimal-control-based learning algorithm [85] — accomplished independently of this thesis and without any help from the author, further supporting the claim regarding ease of use. Therefore, `NLOptControl` holds great potential for not only improving but also engendering a wide variety of off- and on-line control systems.

A Kalman-filter-based perception algorithm [115] that processes raw LiDAR data to estimate the sizes, positions, and velocities of obstacles is identified and combined with the proposed planning algorithm to enable operation in unknown environments. This combination of algorithms can safely control an automated vehicle among small obstacles (radii $<$1.46 m) traveling up to high speeds (20 $\frac{\text{m}}{\text{s}}$) in an unknown environment. However, unexpected results of tests in an unknown environment indicate that when obstacles are traveling at moderate speeds (≈ 7.5 $\frac{\text{m}}{\text{s}}$) they are avoided for the broadest range of obstacle sizes. On either side of this safety peak, safety can be achieved only as obstacle size is reduced. The failure cases for these slower speed obstacles may reflect the possibility that obstacle estimations (i.e., size and speed) made for these slow-speed cases are less accurate than those made for the moderate-speed cases. More accurate estimates are determined for dynamic obstacles because the Kalman-filter can exploit richer LiDAR data. Additionally, as the obstacles move closer to the vehicle, the vantage that radial LiDAR sensor has becomes increasingly limited, which makes the perception algorithm increasingly underestimate obstacle size.

To further quantify challenges encountered when operating in an unknown environment, this thesis identifies eight safety and performance factors: collision avoidance, tracking error, orientation error, planner solve-times, time-to-goal, steering effort, throttle effort, and braking effort. All of these factors degrade significantly when operating in an unknown environment, as opposed to a known environment. Thus, this work identifies and quantifies significant gaps in safety and performance engendered when running in an unknown environment and helps evaluate the known environment assumption. The use of more accurate perception systems should be explored to avoid problems associated with this deterioration.

While this work tailors a trajectory planning formulation for a UGV application, a variety of automated vehicle systems, e.g., UAVs and spacecraft, can also make use of the approach detailed here.

6.1 Contributions

The novel contributions of this thesis are:

Comparing soft constraints to hard constraints for realizing moving obstacle avoidance in trajectory planning algorithms

The choice between soft and hard constraints affects safety in terms of two factors (1) the planner's solve-times and (2) the vehicle's ability to avoid obstacles. Thus, it is critical to quantify how this choice affects safety. To this end, this thesis formulates two trajectory planning algorithms, which are tailored for an HMMWV, to permit moving obstacle avoidance; one of these formulations uses soft constraints, and the other uses hard constraints. Results indicate that, in terms of both solve-times and obstacle avoidance ability, hard constraints are safer than soft constraints. Therefore, for safety, trajectory planning algorithms should use hard constraints instead of soft constraints for avoiding obstacles [38].

Establishing an easy to use and fast direct-collocation-based optimal

control modeling language

Comprehensive trajectory planning algorithms can be reliably solved using a direct-collocation-based numerical optimal control method [92, 111, 125, 112, 73, 74, 64, 133, 44], but these methods often struggle to solve such algorithms in real-time [38, 92, 5, 55]. To accomplish **Contribution 2**, this thesis identifies and investigates an approach for developing an easy to use and fast optimal control modeling language. This language is called `NLOptControl` , and this approach is as follows:

Approach

- For ease of use and speed, `NLOptControl` is embedded in the fast, dynamically-typed *Julia* programming language [13].
- For increased ease of use, `NLOptControl` extends the *JuMP* optimization modeling language [30], which is written in *Julia*, to include a natural syntax for modeling optimal control problems in Bolza form.
- For increased speed, `NLOptControl` uses the acrylic-coloring method [59] to exploit sparsity in the Hessian matrix and reverse-automatic differentiation through the *ReverseDiffSparse* package [121], which is also written in *Julia*.

Results show that `NLOptControl` is both faster and easier to use than the commonly-used commercial *MATLAB*-based optimal control software package called *PROPT*. The speed benchmarking against *PROPT*, which is done using a trajectory planning problem for a UGV, illustrates that `NLOptControl` can solve the problem fast enough for use in an on-line NMPC application, while *PROPT* generally cannot. Therefore, `NLOptControl` is well-suited for formulating comprehensive trajectory planning problems that must be solved on-line, such as the general algorithms developed in [92] and [38]. With `NLOptControl` ready for use, the focus of this thesis shifts back to enabling a safe and performance-based trajectory planning algorithm for automated vehicles operating in dynamic environments.

Formulation of trajectory planning algorithm with specifications S_1–S_7 (listed in Table 1.1)

This thesis speculates that this set of specifications (i.e., S_1–S_7) is necessary for safety and performance in dynamic environments. Chapter IV details the mathematical formulation that incorporates S_1–S_7.

Quantification of the effect that different sets of planning specifications have on safety, performance, and solve-time

This contribution validates the speculation that S_1–S_7 should be incorporated into trajectory planning algorithms, for safety and performance in dynamic environments. To perform this quantification, three additional planning algorithms, each with a less comprehensive set of specifications, are formulated and compared. As expected, planners with less comprehensive sets of specifications than S_1-S_7 reduce UGV safety and performance. However, contrary to expectations, this work shows that adding the minimum time to goal, minimum control effort, and moving obstacle avoidance specifications does not lead to longer solve-times.

Quantification of the need to include a moving obstacle avoidance specification for a range of execution horizons and obstacle speeds

This contribution validates the speculation that a planner without a moving obstacle avoidance specification may be able to operate safely in a dynamic environment under certain conditions. As expected, if the planner is updating quickly, then a slowly moving obstacle can be reliably avoided without a moving obstacle avoidance specification. Additionally, to avoid faster obstacles, both the moving obstacle avoidance and small execution horizon specifications are necessary.

Evaluation of `NLOptControl`'s ability to solve the proposed formulation in real-time in a low-fidelity, known environment

Overall, `NLOptControl` solves the proposed formulation in real-time over a wide range of test cases. As expected, a little execution horizon improves safety but de-

creases the feasibility of obtaining trajectories in real-time. As is also shown in the literature [20], this thesis shows that planning in an environment with more obstacles increases solve-times. Overall, `NLOptControl` is found to be a suitable tool for solving complex NMPC problems on-line.

Evaluation of the developed algorithm's ability to maintain safety and performance in unknown dynamic environments, while being solved in real-time

This thesis develops a high-fidelity, simulation-based proving ground to test the planner developed in this work in an unknown environment. This proving ground is established in ROS and uses a 2D LiDAR model in Gazebo, a perception algorithm that handles dynamic obstacles [115], and a 145 degree of freedom model of the HMMWV in Chrono. Results from this proving ground demonstrate that the developed planning algorithm enables safety and performance in unknown dynamic environments with small obstacles that are moving at speeds up to high. As expected, the results also show that operating in an unknown dynamic environment, as opposed to a known dynamic environment significantly increases[1] collisions, tracking and orientation error, planner solve-times, time-to-goal, steering effort, throttle effort, and braking effort. Thus, this thesis identifies and quantifies significant gaps in safety and performance factors engendered by operating in an unknown environment and helps evaluate the known environment assumption. The use of more accurate perception systems should be explored to avoid problems associated with this deterioration.

The ultimate contribution of this thesis is consequently

Enabling real-time trajectory planning for automated vehicle safety and performance in unknown dynamic environments.

[1] $p \ll 1 \times 10^{-6}$

6.1.1 Publications, presentations, and workshops

Much of this thesis leverages papers that have either been published or are in the publication process. Accordingly, the author gratefully acknowledges the efforts of the co-authors of those papers that helped with them.

Conference papers Two conference papers have been published. The first publication is not directly related to the work in this thesis, but the second is.

1. Febbo, H.; Stein, Jeffrey; Ersal, Tulga; (2016). A Combined Plant/Controller Optimization Framework for hybrid vehicles with mpg, emissions and drivability concerns: ASME IDETC/CIE

2. Febbo, H.; L., Jiechao; Jayakumar, Paramsothy; Stein, Jeffrey; Ersal, Tulga; (2017). Moving Obstacle Avoidance for Large, High-Speed Autonomous Ground Vehicles. American Control Conference. Seattle, WA, USA.

Chapter V is based on a paper that will be submitted to the 2020 to American Control Conference. The current details of this future publication are:

- Febbo, H.; Jayakumar, Paramsothy; Stein, Jeffrey; Ersal, Tulga; (2019). Real-time trajectory planning for automated vehicle safety and performance in unknown dynamic environments

Journal papers Chapter IV is based on a preprint of a paper that is currently being tailored for ACM Transactions on Mathematical Software. The current details of this future publication are:

- Febbo, H.; Jayakumar, Paramsothy; Stein, Jeffrey; Ersal, Tulga; (2019). `NLOpt-Control`: A modeling language for dynamic optimization

Chapter III is based on a paper that was submitted to IEEE Transactions on Control Systems Technology Journal in December 2018, and is currently under review. The current details of this future publication are:

- Febbo, H.; Jayakumar, Paramsothy; Stein, Jeffrey; Ersal, Tulga; (2018). Real-time trajectory planning for automated vehicle safety and performance in dynamic environments

Finally, the algorithms developed in this thesis were modified and used to support human studies in a shared control research project. The current details of this future publication are:

- Bhardwaj, A.; Ghasemi, A.; Zheng, Y.; Febbo, H.; Jayakumar, Paramsothy; Stein, Jeffrey; Gillespie, B.; (2019). Who's the Boss? Arbitrating Control Authority Between a Human Driver and Automation System

Presentations

- "Vehicle-Dynamics-Conscious Real-Time Obstacle Avoidance in Autonomous Ground Vehicles," ARC Annual Review, May 25th, 2016.
- "Vehicle-Dynamics-Conscious Real-Time Obstacle Avoidance in Autonomous Ground Vehicles," ARC Annual Review, May 9th, 2017.
- "Vehicle Control and Design: With Applications To Powertrain Design And Control Autonomous Vehicle Control, And Teleoperated Vehicle Control," A seminar given at Auburn University, August. 28th, 2017

Workshops

- "`NLOptControl` - A High-Level and Fast Model Predictive Control Tool in julia," A workshop given at JuliaCon in Berkeley, June. 24th, 2017

6.2 Future research

The research provided by this thesis can be used as a basis for developing control schemes for dynamic systems. Specifically, it can be applied to nonlinear optimal control and model predictive control to applications such as automated vehicles, robotics systems, chemical applications, and medical applications.

Automated vehicles

This thesis proposes and develops a real-time trajectory planning algorithm that enables safe and performance-based operation of automated vehicles in unknown dynamic environments. As discussed, this algorithm is then tested in a high-fidelity, simulation-based proving ground. Test results indicate that the developed planning algorithm safely controls an HMMWV at high-speeds in an unknown, dynamic environment among small to medium sized obstacles traveling up to moderate speeds. Therefore, future work for this algorithm includes experimental validation and development. This necessary step will uncover practical issues that can introduce additional research questions.

Optimal control

This thesis develops a fast and easy to use optimal control tool called `NLOpt-Control`. While the current use of optimal control is typically limited to engineering applications, it holds great potential for improving and engendering a wide array of other applications. To this end, the remainder of this chapter provides an example of an application for which `NLOptControl` can be used to improve anesthesiology training in veterinary medicine.

Safely prescribing multimodal anesthetic drugs, e.g., injectable and inhalants is a dynamic task that is challenging to teach veterinary students. Part of the challenge is teaching the students how the drugs they prescribe (i.e., control variables) affect

the patient's vital parameters (i.e., state) over time. In the early stages of veterinary training, this type of feedback is difficult to provide without an actual patient. Currently, students assimilate knowledge gathered in core courses like anatomy, pharmacology, physiology, and anesthesiology to determine the drugs needed to induce a particular patient's state. While this is a critical skill that needs to be developed, it is a time-consuming and error-prone endeavor that can limit and skew learning. To improve learning, the students can be provided with a virtual model of the patient that can be treated and monitored.

This virtual pedagogical tool can be developed using a model predictive control approach, which has a mathematical model of the patient dictated by the state and control variables. A species-specific, mathematical model can be developed; the state of the patient can be quantified its by heart rate, respiratory rate, temperature, blood pressure, carbon dioxide level, blood oxygenation saturation, and electrocardiogram waveform (ECG); the controls are various injectable and inhalant anesthetic drugs; and sudden and expected drops/rises in the vital parameters can be used to identify when the patient is ready to be intubated/extubated. During the entire procedure, the student can monitor the patient's vitals and administer inhalant/injectable anesthesia in response to the patient's dynamic vitals. Each time the student prescribes a new dosage, an optimal control problem can be solved to show the optimal dosage at that time. The model of the patient can then be simulated using the student's dosage. This simulated anesthetic process can continue until either the patient dies or the procedure is completed. Thus, prior to hands-on training, i.e., clinical rotations, veterinary students can navigate potential outcomes to various aesthetic cases using the student's prescribed pharma.

APPENDICES

APPENDIX A

Supplementary Material for Chapter III

Fig. A.1 shows the analytic solutions for the states, control, as well as costates compared to the results obtained with `NLOptControl` using a single interval with 30 LGR nodes. `NLOptControl` calculates these trajectories reasonably well.

The NLP solver for this example is *IPOPT* and an hp-method in `NLOptControl` is used with 4 intervals and 10 LGR nodes.

A.1 Moon lander problem

A.1.1 Closed-loop

Fig. A.2 shows the closed-loop solution to the moon lander problem using `NLOptControl`. The closed-loop trajectory of the plant is very close to the analytic solution. Additionally, all of the solve-times are well below the chosen execution horizon t_{ex} of 0.2 s; thus `NLOptControl` solves this NMPC problem in real-time.

The NLP solver for this example is *IPOPT* and an hp-method in `NLOptControl` is used with 4 intervals and 10 LGR nodes.

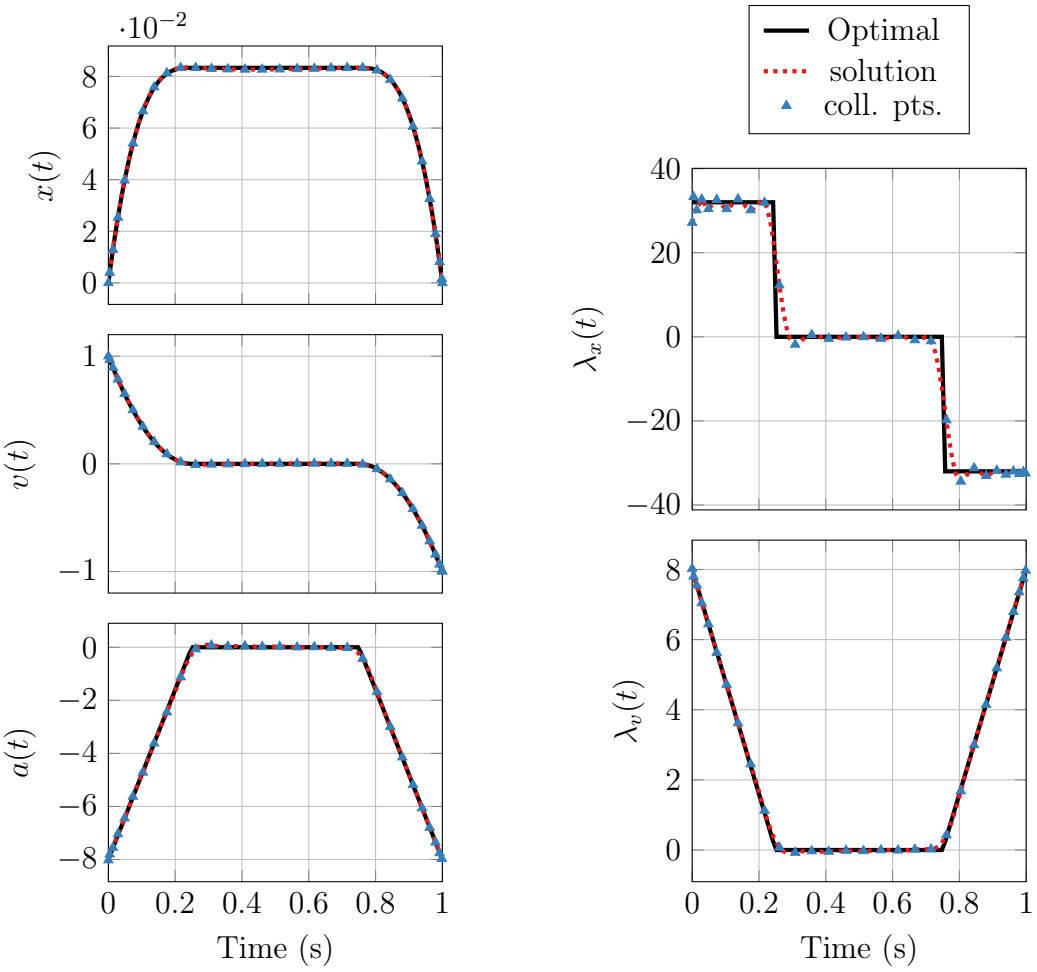


Figure A.1: State, control, and costate trajectories using `NLOptControl` (with 30 LGR nodes) compared to the analytical optimal solution for the Bryson Denham problem

A.1.2 Open-loop

In Fig. A.3, it can be seen that both `NLOptControl` and *PROPT* determine the analytic solution accurately, with 30 LGR and Chebyshev nodes, respectively. However, there is an overshoot in the solution of the control with both `NLOptControl` and *PROPT*. This is due to the bang-bang nature of the analytic solution. It is noted that this overshoot may be mitigated using either mesh refinement [109, 25, 70] or radial basis functions [100].

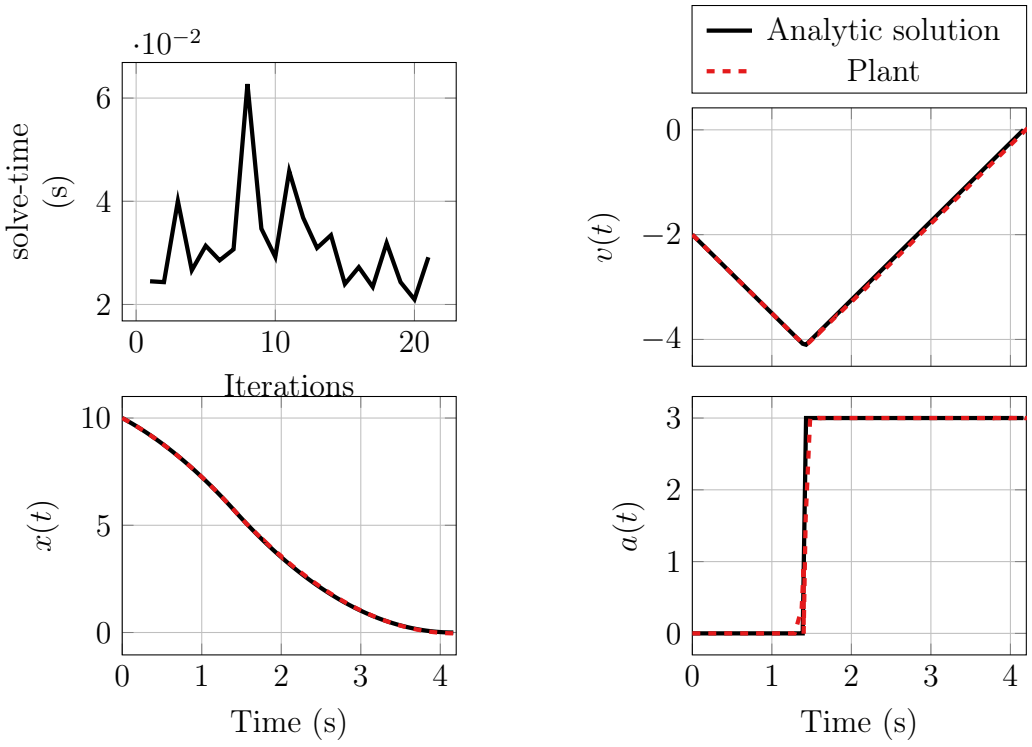


Figure A.2: Closed-loop trajectories for moon lander problem compared to the analytic solution

A.2 Benchmark problem

This section provides an example of the type of solutions that are obtained from the benchmark between `NLOptControl` and *PROPT*. For `NLOptControl`, the hp-method with LGR nodes and four intervals and 10 collocation points per interval is used. *PROPT* is set to use four phases and 10 collocation points per phase and Chebyshev nodes. Fig. A.4 compares the results of these solvers, where it can be seen that position trajectories are close. Starting at a speed of 15 $\frac{\text{m}}{\text{s}}$, the solutions obtained from both *PROPT* and `NLOptControl` apply maximum acceleration from $t_0 = 0$ s to $t_f =$ 5.1 s while avoiding collision with the obstacle and reaching the desired goal position. The trajectories for `NLOptControl` exhibit large oscillations in the α trajectory. These oscillations may be an artifact of the Runge phenomenon and seem to be reduced with *PROPT* as it uses Chebyshev nodes.

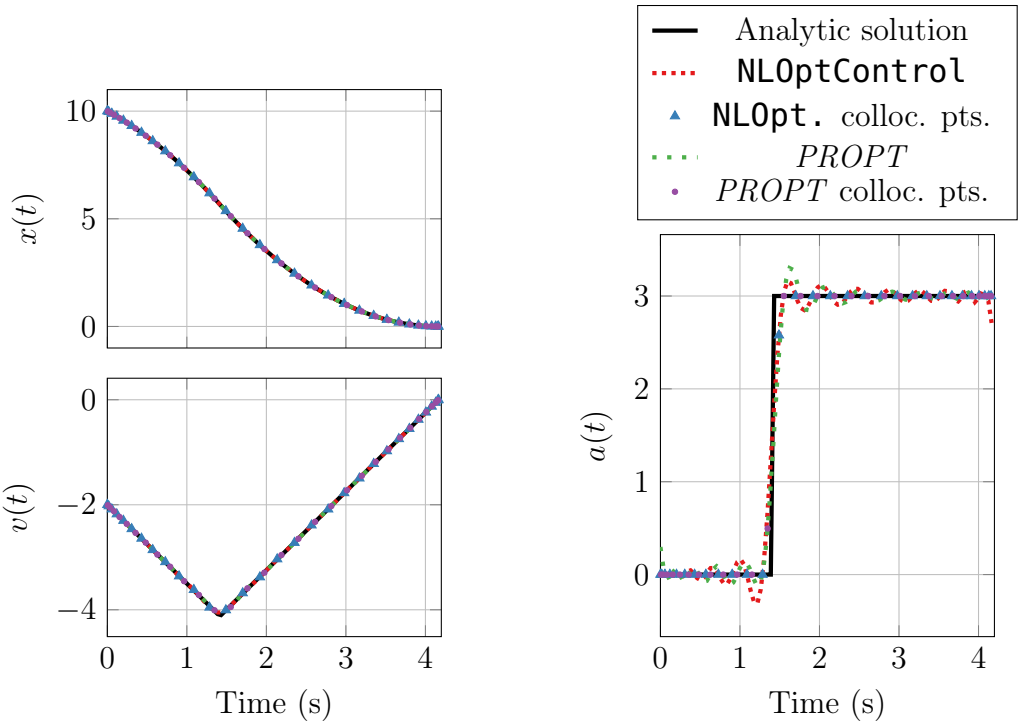


Figure A.3: State and control trajectories using `NLOptControl` (with 30 LGR nodes) and *PROPT* (with 30 Chebyshev nodes) compared to the analytic solution for moon lander problem

There is no analytic solution to this problem.

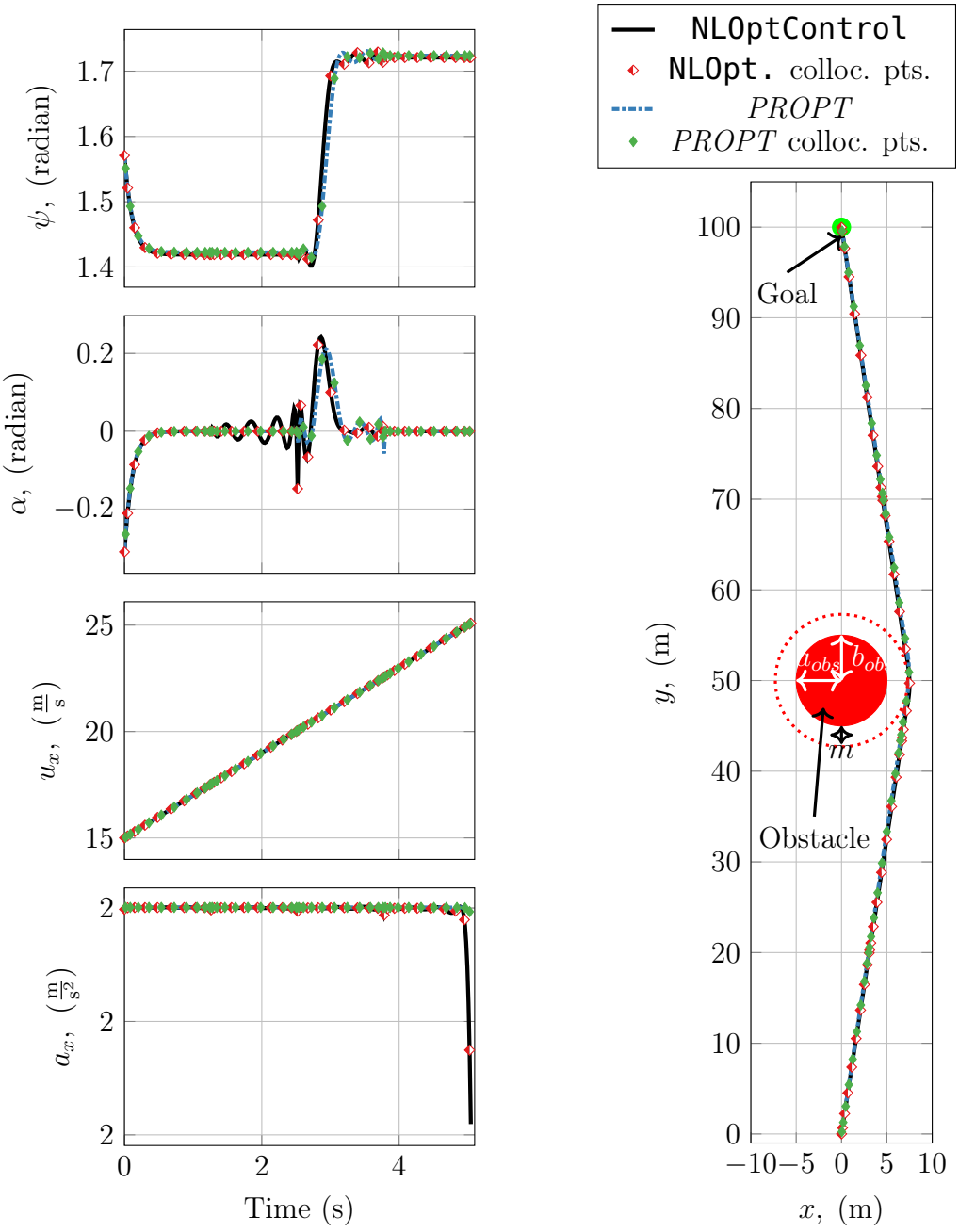


Figure A.4: State and control trajectories using **NLOptControl** (with 4 intervals and 10 LGR nodes) and *PROPT* (with 4 intervals and 10 Chebyshev nodes) for the kinematic ground vehicle problem

APPENDIX B

Supplementary Material for Chapter IV

Table B.1: Long *MATLAB* solve-times

	Solve-times	
Constraints	3 Obstacles	17 Obstacles
Hard constraints	44.4 s	193 s
Soft constraints	110 s	2.19×10^3 s

Table B.2: Vehicle Parameters

Variable	Value	Units
M_t	2689	kg
I_{zz}	4110	$\text{kg} - \text{m}^2$
L_f,L_r	1.58,1.72	m
K_{z_x},$K_{z_{yr}}$,$K_{z_{yf}}$	806,1076,675	$\frac{\text{N}}{\frac{\text{m}}{\text{s}^2}}$
$F_{z_{min}}$	1000	N
a,b	1300,100	-
$\psi_{\min}$,$\psi_{\max}$	$[-2\pi, 2\pi]$	$^\circ$
$\delta_{f,\min}$,$\delta_{f,\max}$	$[-30, 30]$	$^\circ$
$\gamma_{f,\min}$,$\gamma_{f,\max}$	$[-5, 5]$	$\frac{^\circ}{\text{s}}$
$J_{x,\min}$,$J_{x,\max}$	$[-5, 5]$	$\frac{\text{m}}{\text{s}^3}$
$U_{\min}$,$U_{\max}$	$[0.01, 29]$	$\frac{\text{m}}{\text{s}}$

Table B.3: Simulation Parameters for P_A

Variable or Conditions	Value and Units
t_{ex},N,L_{range},κ	0.5 s,10,50.0 m,5.0 m
sm_1, sm_2, sm	2.5 m,4 m,2 m
$\mathbf{X0}$	$[200 \text{ m}, 0 \text{ m}, 0, 0, 1.57\text{rad}, 0, 17 \ \frac{\text{m}}{\text{s}}, 0]$
$\mathbf{X0}_{tol}$	$[0.5 \text{ m}, 0.5 \text{ m}, 0.5, 0.005, 0.5, 0.25, 0.5 \ \frac{\text{m}}{\text{s}}, 0.5]$
$\mathbf{XF}_{tol}$	$[5.0 \text{ m}, 5.0 \text{ m}, NaN, NaN, NaN, \dots$ $NaN, NaN, NaN]$
w_{ic},w_{x0},w_{y0},w_{v0},w_{r0},$w_{\psi 0}$,w_{sa0},w_{ux0},w_{ax0},w_{xf}	100,1,1,10,10,10,2,0.1,0.1,100
w_g,w_t,w_{haf},w_{F_z},w_{ce},w_{sa},w_{sr},w_{ax},w_{jx}	10,0,1,0.5,0,0.1,1,0.1,0.01
moving obstacle avoidance constraint in Eqn. 4.7	$false$

Table B.4: Environment for E_A

Variable	Description	Value	Unit
$\mathbf{a_{obs}}$	array of the obstacles semi-major axes	$[5, 4, 2]$	m
$\mathbf{b_{obs}}$	array of the obstacles semi-minor axes	$[5, 4, 2]$	m
$\mathbf{x0_{obs}}$	array of the obstacles initial x positions	$[205, 180, 200]$	m
$\mathbf{y0_{obs}}$	array of the obstacles initial y positions	$[57, 75, 63]$	m
$\mathbf{v_x}$	array of obstacles speeds in x direction	$[0, 0, 0]$	$\frac{\text{m}}{\text{s}}$
$\mathbf{v_y}$	array of obstacles speeds in y direction	$[0, 0, 0]$	$\frac{\text{m}}{\text{s}}$
x_g	x position of goal location	200	m
y_g	x position of goal location	125	m
σ	tolerance on goal location	15	m
ψ_g	desired orientation at goal	$\frac{\pi}{2}$	rad

Table B.5: Environment for E_C
see Table. C.5 for Variable Descriptions

Variable	Value
$\mathbf{a_{obs}}$	$[6, 6, 0.387, 0.387, 0.387, 0.387, 0.387, 0.387, 0.387, 0.387, 0.387, 0.387, 0.387, 0.387, \dots$ $\dots 0.387, 0.387, 0.387, 0.387, 0.387, 0.387, 0.387, 0.387, 0.387, 0.387, 0.387, 0.387, \dots$ $\dots 0.387, 0.387, 0.387, 0.387, 0.387, 0.387, 0.387, 0.387, 0.387, 0.387, 0.387, 0.387]$
$\mathbf{b_{obs}}$	same as $\mathbf{a_{obs}}$
$\mathbf{x0_{obs}}$	$[6, 18, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, \dots$ $\dots 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12]$
$\mathbf{y0_{obs}}$	$[281, 650, 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, \dots$ $\dots 95, 100, 105, 110, 115, 120, 125, 130, 135, 140, 145, 150, 155, 160, 165, 170, 175]$
$\mathbf{v_x}$	$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$
$\mathbf{v_y}$	$[0, -10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$
x_g	18
y_g	700
σ	25
ψ_g	$\frac{\pi}{2}$

Table B.6: Control Effort

Effort Term	P_B	P_C	Decrease
Steering Angle	0.000586	0.000422	28.0 %
Steering Rate	0.00129	0.000922	28.5 %
Longitudinal Jerk	0.530	0.420	20.8 %
Total	0.532	0.421	20.9 %

Table B.7: Fisher's exact test for attaining the goal for P_C and P_D in E_C ($p = 2.2 \times 10^{-16}$)

	Fail	Pass	Total
P_C	378 (60.0 %)	22 (12.9 %)	400
P_D	252 (40.0 %)	148 (87.1 %)	400
Total	630 (100%)	170 (100 %)	800

APPENDIX C

Supplementary Material for Chapter V

Table C.1: LiDAR Parameters

Variable	Value	Units
Update rate	40	Hz
Minimum range	0.1	m
Maximum range	100	m
Samples	1875	
Mean	0	m
Standard deviation	1000	m

Table C.2: Chrono Model Parameters [117]

Variable	Value
Vehicle assembly	$HMMWVFull$
Contact method	NSC
Tire model	$TMEASY$
Powertrain model	$SHAFTS$
Drive type	RWD
Steering type	$PITMANARM$

Table C.3: Vehicle Parameters

Variable	Value	Units
M_t	2689	kg
I_{zz}	4110	$\text{kg} - \text{m}^2$
L_f,L_r	1.58,1.72	m
K_{z_x},$K_{z_{yr}}$,$K_{z_{yf}}$	806,1076,675	$\frac{\text{N}}{\frac{\text{m}}{\text{s}^2}}$
$F_{z_{min}}$	1000	N
a,b	1300,100	-
$\psi_{\min}$,$\psi_{\max}$	$[-2\pi, 2\pi]$	$^\circ$
$\delta_{f,\min}$,$\delta_{f,\max}$	$[-30, 30]$	$^\circ$
$\gamma_{f,\min}$,$\gamma_{f,\max}$	$[-5, 5]$	$\frac{^\circ}{\text{s}}$
$J_{x,\min}$,$J_{x,\max}$	$[-5, 5]$	$\frac{\text{m}}{\text{s}^3}$
$U_{\min}$,$U_{\max}$	$[0.01, 29]$	$\frac{\text{m}}{\text{s}}$

Table C.4: Planning Parameters

Variable or Conditions	Value and Units
N,L_{range},κ	35,60.0 m,5.0 m
sm_1, sm_2, sm	2.5 m,4 m,2 m
$\mathbf{X0}$	$[0\text{ m}, 0\text{ m}, 0, 0, 1.57\text{rad}, 0, 0\ \frac{\text{m}}{\text{s}}, 0]$
$\mathbf{X0}_{tol}$	$[0.5\text{ m}, 0.5\text{ m}, 0.5, 0.005, 0.5, 0.25, 0.5\ \frac{\text{m}}{\text{s}}, 0.5]$
$\mathbf{XF}_{tol}$	$[5.0\text{ m}, 5.0\text{ m}, NaN, NaN, NaN, NaN, NaN, NaN]$
w_{ic},w_{x0},w_{y0},w_{v0},w_{r0},$w_{\psi 0}$,w_{sa0},w_{ux0},w_{ax0},w_{xf}	100,50,50,10,20,20,2,0.1,0.1,100
w_g,w_t,w_{haf},w_{F_z},w_{ce},w_{sa},w_{sr},w_{ax},w_{jx}	10,250,1,0.5,1,0.1,1,0.1,0.01

Table C.5: Environment Parameters

Variable	Description	Value	Unit
$\mathbf{a_{obs}}$	array of the obstacles semi-major axes	$[r_1, 10, 5, 12]$	m
$\mathbf{b_{obs}}$	array of the obstacles semi-minor axes	$[r_1, 10, 5, 12]$	m
$\mathbf{x0_{obs}}$	array of the obstacles initial x positions	$[0, -25, -25, -25]$	m
$\mathbf{y0_{obs}}$	array of the obstacles initial y positions	$[275, 168, 145, 110]$	m
$\mathbf{v_x}$	array of obstacles speeds in x direction	$[0, 0, 0, 0]$	$\frac{\text{m}}{\text{s}}$
$\mathbf{v_y}$	array of obstacles speeds in y direction	$[vy_1, 0, 0, -1]$	$\frac{\text{m}}{\text{s}}$
x_g	x position of goal location	400	m
y_g	x position of goal location	0	m
σ	tolerance on goal location	10	m
ψ_g	desired orientation at goal	$\frac{\pi}{2}$	rad

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Milton Abramowitz. Stegun. *Handbook of Mathematical Functions*, 55:888, 1965.

[2] Jan Albersmeyer, Dörte Beigel, Christian Kirches, Leonard Wirsching, Hans Georg Bock, and Johannes P Schlöder. Fast nonlinear model predictive control with an application in automotive engineering. In *Nonlinear Model Predictive Control*, pages 471–480. Springer, 2009.

[3] Joel Andersson, Johan 12.0Akesson, and Moritz Diehl. Casadi: A symbolic package for automatic differentiation and optimal control. In *Recent Advances in Algorithmic Differentiation*, pages 297–307. Springer, 2012.

[4] Lorenz T. Biegler Andreas Wachter. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming, 2004.

[5] G Basset, Yunjun Xu, and OA Yakimenko. Computing short-time aircraft maneuvers using direct methods. *Journal of Computer and Systems Sciences International*, 49(3):481–513, 2010.

[6] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route planning in transportation networks. In *Algorithm Engineering*, pages 19–80. Springer, 2016.

[7] Craig Earl Beal and J. Christian Gerdes. Model predictive control for vehicle stabilization at the limits of handling. *IEEE Transactions on Control Systems Technology*, 21(4):1258–1269, 2013.

[8] Victor M Becerra. Solving complex optimal control problems at no cost with psopt. In *Computer-Aided Control System Design (CACSD), 2010 IEEE International Symposium on*, pages 1391–1396. IEEE, 2010.

[9] John T Betts. Survey of numerical methods for trajectory optimization. *Journal of Guidance Control and Dynamics*, 21(2):193–207, 1998.

[10] John T Betts. *Practical methods for optimal control and estimation using nonlinear programming*, volume 19. Siam, 2010.

[11] John T Betts, Stephen L Campbell, and NN Kalla. Initialization of direct transcription optimal control software. In *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, volume 4, pages 3802–3807. IEEE, 2003.

[12] John T Betts and William P Huffman. Exploiting sparsity in the direct transcription method for optimal control. *Computational Optimization and Applications*, 14(2):179–201, 1999.

[13] Jeff Bezanson, Stefan Karpinski, Viral B Shah, and Alan Edelman. Julia: A fast dynamic language for technical computing. *arXiv preprint arXiv:1209.5145*, 2012.

[14] Lorenz T Biegler. An overview of simultaneous strategies for dynamic optimization. *Chemical Engineering and Processing: Process Intensification*, 46(11):1043–1053, 2007.

[15] Hans Georg Bock and Karl-Josef Plitt. A multiple shooting algorithm for direct solution of optimal control problems. *IFAC Proceedings Volumes*, 17(2):1603–1608, 1984.

[16] F. Borrelli, P. Falcone, T. Keviczky, J. Asgari, and D. Hrovat. MPC-based approach to active steering for autonomous vehicle systems. *International Journal of Vehicle Autonomous Systems*, 3(2-4):265–291, 2005.

[17] NEAL E. BOUDETTE. G.m. says its driverless car could be in fleets by next year. *New York Times Magazine*, 2018.

[18] Richard H. Byrd, Jorge Nocedal, and Richard A. Waltz. Knitro: An integrated package for nonlinear optimization, 2006.

[19] John Canny and John Reif. New lower bound techniques for robot motion planning problems. In *Foundations of Computer Science, 1987., 28th Annual Symposium on*, pages 49–60. IEEE, 1987.

[20] Hao-Tien Lewis Chiang, Baisravan HomChaudhuri, Lee Smith, and Lydia Tapia. Safety, challenges, and performance of motion planners in dynamic environments.

[21] R Craig Coulter. Implementation of the pure pursuit path tracking algorithm. Technical report, Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, 1992.

[22] Hendrik Dahlkamp, Adrian Kaehler, David Stavens, Sebastian Thrun, and Gary R Bradski. Self-supervised monocular road detection in desert terrain. In *Robotics: Science and Systems*, volume 38. Philadelphia, 2006.

[23] Christopher L Darby. *hp-Pseudospectral method for solving continuous-time nonlinear optimal control problems*. University of Florida, 2011.

[24] Christopher L Darby, William W Hager, and Anil V Rao. An hp-adaptive pseudospectral method for solving optimal control problems. *Optimal Control Applications and Methods*, 32(4):476–502, 2011.

[25] Christopher L Darby and Anil V Rao. A mesh refinement algorithm for solving optimal control problems using pseudospectral methods. *Proceedings of the AIAA*, 2009.

[26] Moritz Diehl, H Georg Bock, Johannes P Schlöder, Rolf Findeisen, Zoltan Nagy, and Frank Allgöwer. Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations. *Journal of Process Control*, 12(4):577–585, 2002.

[27] Moritz Diehl, Hans Georg Bock, Holger Diedam, and P-B Wieber. Fast direct multiple shooting algorithms for optimal robot control. In *Fast Motions in Biomechanics and Robotics*, pages 65–93. Springer, 2006.

[28] Moritz Diehl, Hans Joachim Ferreau, and Niels Haverbeke. Efficient numerical methods for nonlinear mpc and moving horizon estimation. *Nonlinear Model Predictive Control*, 384:391–417, 2009.

[29] Moritz Diehl, Daniel B Leineweber, and Andreas AS Schäfer. *MUSCOD-II users' manual*. Universität Heidelberg. Interdisziplinäres Zentrum für Wissenschaftliches âĂę, 2001.

[30] Iain Dunning, Joey Huchette, and Miles Lubin. Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[31] Iain Robert Dunning. *Advances in robust and adaptive optimization: algorithms, software, and insights*. PhD thesis, Massachusetts Institute of Technology, 2016.

[32] Jorge J. More Elizabeth D. Dolan. Benchmarking optimization software with performance profiles. 2001.

[33] Gamal Elnagar, Mohammad A Kazemi, and Mohsen Razzaghi. The pseudospectral legendre method for discretizing optimal control problems. *IEEE Transactions on Automatic Control*, 40(10):1793–1796, 1995.

[34] Azim Eskandarian. *Handbook of Intelligent Vehicles*. Springer, 2012.

[35] Fariba Fahroo and I Michael Ross. Advances in pseudospectral methods for optimal control. In *AIAA Guidance, Navigation and Control Conference*, page 7309, 2008.

[36] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng, and D. Hrovat. Predictive active steering control for autonomous vehicle systems. *IEEE Transactions on Control Systems Technology*, 15(3):566–80, 2007.

[37] Huckleberry Febbo. NLOptControl. https://github.com/JuliaMPC/NLOptControl.jl, 2017.

[38] Huckleberry Febbo, Jiechao Liu, Paramsothy Jayakumar, Jeffrey L Stein, and Tulga Ersal. Moving obstacle avoidance for large, high-speed autonomous ground vehicles. In *American Control Conference*, pages 5568–5573, 2017.

[39] Sarah Ferguson, Brandon Luders, Robert C Grande, and Jonathan P How. Real-time predictive modeling and robust avoidance of pedestrians with uncertain, changing intentions. In *Algorithmic Foundations of Robotics XI*, pages 161–177. Springer, 2015.

[40] Robert Fourer. On the evolution of optimization modeling systems. *Optimization Stories*, pages 377–388, 2012.

[41] Robert Fourer, DM Gay, and BW Kernighan. Ampl: A modeling language for mathematical programming, 2002. *Duxbury Press*.

[42] Emmanuel Fragniere, Jacek Gondzio, Robert Sarkissian, and Jean-Philippe Vial. A structure-exploiting tool in algebraic modeling languages. *Management Science*, 46(8):1145–1158, 2000.

[43] J. V. Frasch, A. Gray, M. Zanon, H. J. Ferreau, S. Sager, F. Borrelli, and M. Diehl. An auto-generated nonlinear MPC algorithm for real-time obstacle avoidance of ground vehicles. In *European Control Conference*, pages 4136–41, 2013.

[44] Janick V Frasch, Andrew Gray, Mario Zanon, Hans Joachim Ferreau, Sebastian Sager, Francesco Borrelli, and Moritz Diehl. An auto-generated nonlinear mpc algorithm for real-time obstacle avoidance of ground vehicles. In *European Control Conference*, pages 4136–4141. IEEE, 2013.

[45] Emilio Frazzoli, Munther A Dahleh, and Eric Feron. Real-time motion planning for agile autonomous vehicles. *Journal of Guidance, Control, and Dynamics*, 25(1):116–129, 2002.

[46] Yiqi Gao, Theresa Lin, Francesco Borrelli, Eric Tseng, and Davor Hrovat. Predictive control of autonomous ground vehicles with obstacle avoidance on slippery roads. In *Dynamic Systems and Control Conference*, pages 265–272, 2010.

[47] Divya Garg. *Advances in global pseudospectral methods for optimal control*. PhD thesis, University of Florida USA, 2011.

[48] Divya Garg, William W Hager, and Anil V Rao. Pseudospectral methods for solving infinite-horizon optimal control problems. *Automatica*, 47(4):829–837, 2011.

[49] Divya Garg, Michael Patterson, William W Hager, Anil V Rao, David A Benson, and Geoffrey T Huntington. A unified framework for the numerical solution of optimal control problems using pseudospectral methods. *Automatica*, 46(11):1843–1851, 2010.

[50] Carl Friedrich Gauss. *Methodus nova integralium valores per approximationem inveniendi*. apvd Henricvm Dieterich, 1815.

[51] Assefaw H Gebremedhin, Arijit Tarafdar, Alex Pothen, and Andrea Walther. Efficient computation of sparse hessians using coloring and automatic differentiation. *INFORMS Journal on Computing*, 21(2):209–223, 2009.

[52] Assefaw Hadish Gebremedhin, Fredrik Manne, and Alex Pothen. What color is your jacobian? graph coloring for computing derivatives. *SIAM review*, 47(4):629–705, 2005.

[53] Mathieu Geisert and Nicolas Mansard. Trajectory generation for quadrotor based systems using numerical optimal control. In *International Conference on Robotics and Automation*, pages 2958–2964. IEEE, 2016.

[54] Jon Gertner. Tesla's dangerous sprint into the futture. *The New York Times Magazine*, 2017.

[55] Borna Ghannadi, Naser Mehrabi, Reza Sharif Razavian, and John McPhee. Nonlinear model predictive control of an upper extremity rehabilitation robot using a two-dimensional human-robot interaction model. In *International Conference on Intelligent Robots and Systems*, pages 502–507. IEEE, 2017.

[56] Philip E. Gill, Walter Murray, and Michael A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. 2002.

[57] J Gonzales, F Zhang, K Li, and F Borrelli. Autonomous drifting with onboard sensors. In *Advanced Vehicle Control*, page 133. CRC Press, 2016.

[58] Colin M Greatwood and Arthur G Richards. Implementation of fast mpc with a quadrotor for obstacle avoidance. In *AIAA Guidance, Navigation, and Control (GNC) Conference*, page 4790, 2013.

[59] Andreas Griewank, David Juedes, and Jean Utke. Algorithm 755: Adol-c: a package for the automatic differentiation of algorithms written in c/c++. *ACM Transactions on Mathematical Software (TOMS)*, 22(2):131–167, 1996.

[60] William W Hager. Runge-kutta methods in optimal control and the transformed adjoint system. *Numerische Mathematik*, 87(2):247–282, 2000.

[61] Nicholas Hale and Alex Townsend. Fast and accurate computation of gauss–legendre and gauss–jacobi quadrature nodes and weights. *SIAM Journal on Scientific Computing*, 35(2):A652–A674, 2013.

[62] Charles R Hargraves and Stephen W Paris. Direct trajectory optimization using nonlinear programming and collocation. *Journal of Guidance, Control, and Dynamics*, 10(4):338–342, 1987.

[63] Josef Heinen. Gr. https://github.com/jheinen/GR.jl, 2018.

[64] Anders Holmqvist and Fredrik Magnusson. Open-loop optimal control of batch chromatographic separation processes using direct collocation. *Journal of Process Control*, 46:55–74, 2016.

[65] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. ACADO toolkit: An open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011.

[66] Joey Huchette, Miles Lubin, and Cosmin Petra. Parallel algebraic modeling for stochastic optimization. In *Proceedings of the 1st First Workshop for High Performance Technical Computing in Dynamic Languages*, pages 29–35. IEEE Press, 2014.

[67] David G Hull. Conversion of optimal control problems into parameter optimization problems. *Journal of Guidance, Control, and Dynamics*, 20(1):57–60, 1997.

[68] Geoffrey Huntington, David Benson, and Anil Rao. A comparison of accuracy and computational efficiency of three pseudospectral methods. In *AIAA Guidance, Navigation and Control Conference*, page 6405, 2007.

[69] M Yousuff Hussaini and Thomas A Zang. Spectral methods in fluid dynamics. *Annual Review of Fluid Mechanics*, 19(1):339–367, 1987.

[70] Sachin Jain and Panagiotis Tsiotras. Trajectory optimization using multiresolution techniques. *Journal of Guidance, Control, and Dynamics*, 31(5):1424–1436, 2008.

[71] Christopher Jewison, R Scott Erwin, and Alvar Saenz-Otero. Model predictive control with ellipsoid obstacle constraints for spacecraft rendezvous. *IFAC-PapersOnLine*, 48(9):257–262, 2015.

[72] Eric Firing Michael Droettboom John Hunter, Darren Dale. matplotlib. https://github.com/matplotlib/matplotlib, 2018.

[73] Timothy Jorris, Christopher Schulz, Franklin Friedl, and Anil Rao. Constrained trajectory optimization using pseudospectral methods. In *AIAA Atmospheric Flight Mechanics Conference and Exhibit*, page 6218, 2008.

[74] Wei Kang and Naz Bedrossian. Pseudospectral optimal control theory makes debut flight, saves nasa 1 m in under three hours. 2007.

[75] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[76] Christos Katrakazas, Mohammed Quddus, Wen-Hua Chen, and Lipika Deka. Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C: Emerging Technologies*, 60:416–442, 2015.

[77] A. Katriniok and D. Abel. LTV-MPC approach for lateral vehicle guidance by front steering at the limits of vehicle dynamics. In *IEEE Conference on Decision and Control and European Control Conference*, pages 6828–33.

[78] Matthew Kelly. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, 59(4):849–904, 2017.

[79] Eric C Kerrigan and Jan M Maciejowski. Soft constraints and exact penalty functions in model predictive control. In *UKACC International Conference*, 2000.

[80] Christian Kirches, Leonard Wirsching, Sebastian Sager, and Hans Georg Bock. Efficient numerics for nonlinear model predictive control. In *Recent Advances in Optimization and its Applications in Engineering*, pages 339–357. Springer, 2010.

[81] Krisada Kritayakirana and J Christian Gerdes. Using the centre of percussion to design a steering controller for an autonomous race car. *Vehicle System Dynamics*, 50(sup1):33–51, 2012.

[82] Philipp Krüsi, Paul Furgale, Michael Bosse, and Roland Siegwart. Driving on point clouds: Motion planning, trajectory optimization, and terrain assessment in generic nonplanar environments. *Journal of Field Robotics*, 34(5):940–984, 2017.

[83] Viktor Leek. An optimal control toolbox for matlab based on casadi, 2016.

[84] Benoît Legat. SumOfSquares. https://github.com/JuliaOpt/SumOfSquares.jl, 2019.

[85] Laurent Lessard, Xuezhou Zhang, and Xiaojin Zhu. An optimal control approach to sequential machine teaching. *arXiv preprint arXiv:1810.06175*, 2018.

[86] A. Liniger, A. Domahidi, and M. Morari. Optimization-based autonomous racing of 1:43 scale RC cars. *Optimal Control Applications and Methods*, 36(5):628–47, 2015.

[87] Todd Litman. *Autonomous vehicle implementation predictions*. Victoria Transport Policy Institute, 2017.

[88] Jiechao Liu, Paramsothy Jayakumar, James L. Overholt, Jeffrey L. Stein, and Tulga Ersal. The role of model fidelity in model predictive control based hazard avoidance in unmanned ground vehicles using lidar sensors. volume 3 of *ASME Dynamic Systems and Control Conference*, page Dynamic Systems and Control Division. American Society of Mechanical Engineers, 2013.

[89] Jiechao Liu, Paramsothy Jayakumar, Jeffrey L. Stein, and Tulga Ersal. A multi-stage optimization formulation for mpc-based obstacle avoidance in autonomous vehicles using a lidar sensor. American Society of Mechanical Engineers, 2014.

[90] Jiechao Liu, Paramsothy Jayakumar, Jeffrey L. Stein, and Tulga Ersal. An mpc algorithm with combined speed and steering control for obstacle avoidance in autonomus ground vehicles. American Society of Mechanical Engineers, 2015.

[91] Jiechao Liu, Paramsothy Jayakumar, Jeffrey L Stein, and Tulga Ersal. A study on model fidelity for model predictive control-based obstacle avoidance in high-speed autonomous ground vehicles. *Vehicle System Dynamics*, 54(11):1629–1650, 2016.

[92] Jiechao Liu, Paramsothy Jayakumar, Jeffrey L Stein, and Tulga Ersal. Combined speed and steering control in high-speed autonomous ground vehicles for obstacle avoidance using model predictive control. *IEEE Transactions on Vehicular Technology*, 66(10):8746–8763, 2017.

[93] Sikang Liu, Nikolay Atanasov, Kartik Mohta, and Vijay Kumar. Search-based motion planning for quadrotors using linear quadratic minimum time control. In *International Conference on Intelligent Robots and Systems*, pages 2872–2879. IEEE, 2017.

[94] Sikang Liu, Kartik Mohta, Nikolay Atanasov, and Vijay Kumar. Search-based motion planning for aggressive flight in se (3). *IEEE Robotics and Automation Letters*, 3(3):2439–2446, 2018.

[95] Michael C Lo, Robert P Giffin, Kraig A Pakulski, W Sumner Davis, Stephen A Bernstein, and Daniel V Wise. High-mobility multipurpose wheeled vehicle rollover accidents and injuries to us army soldiers by reported occupant restraint use, 1992–2013. *Military medicine*, 182(5-6):e1782–e1791, 2017.

[96] Miles Lubin and Iain Dunning. Computing in operations research using julia. *INFORMS Journal on Computing*, 27(2):238–248, 2015.

[97] Miles Lubin, Yury Dvorkin, and Scott Backhaus. A robust approach to chance constrained optimal power flow with renewable generation. *IEEE Transactions on Power Systems*, 31(5):3840–3849, 2016.

[98] Charles C MacAdam. Development of driver/vehicle steering interaction models for dynamic analysis. Technical report, MICHIGAN UNIV ANN ARBOR TRANSPORTATION RESEARCH INST, 1988.

[99] J Meditch. On the problem of optimal thrust programming for a lunar soft landing. *IEEE Transactions on Automatic Control*, 9(4):477–484, 1964.

[100] Hossein Mirinejad and Tamer Inanc. An rbf collocation method for solving optimal control problems. *Robotics and Autonomous Systems*, 87(Supplement C):219 – 225, 2017.

[101] Kartik Mohta, Ke Sun, Sikang Liu, Michael Watterson, Bernd Pfrommer, James Svacha, Yash Mulgaonkar, Camillo Jose Taylor, and Vijay Kumar. Experiments in fast, autonomous, gps-denied quadrotor flight. *arXiv preprint arXiv:1806.07053*, 2018.

[102] Manfred Morari and Jay H Lee. Model predictive control: past, present and future. *Computers & Chemical Engineering*, 23(4):667–682, 1999.

[103] Yuuki Nishio, Kenichiro Nonaka, and Kazuma Sekiguchi. Moving obstacle avoidance control by fuzzy potential method and model predictive control. In *Asian Control Conference*, pages 1298–1303. IEEE, 2017.

[104] Toshiyuki Ohtsuka and Kohei Ozaki. Practical issues in nonlinear model predictive control: real-time optimization and systematic tuning. In *Nonlinear Model Predictive Control*, pages 447–460. Springer, 2009.

[105] Sheehan Olver. Fastgaussquadrature. https://github.com/ajt60gaibb/FastGaussQuadrature.jl, 2017.

[106] Mike R Osborne. On shooting methods for boundary value problems. *Journal of Mathematical Analysis and Applications*, 27(2):417–433, 1969.

[107] Hans Pacejka. *Tire and vehicle dynamics*. Elsevier, 2005.

[108] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, 2016.

[109] Michael A Patterson, William W Hager, and Anil V Rao. A ph mesh refinement method for optimal control. *Optimal Control Applications and Methods*, 36(4):398–421, 2015.

[110] Michael A Patterson and Anil V Rao. Exploiting sparsity in direct collocation pseudospectral methods for solving optimal control problems. *Journal of Spacecraft and Rockets*, 49(2):364–377, 2012.

[111] Michael A Patterson and Anil V Rao. Gpops-ii: A matlab software for solving multiple-phase optimal control problems using hp-adaptive gaussian quadrature collocation methods and sparse nonlinear programming. *ACM Transactions on Mathematical Software (TOMS)*, 41(1):1, 2014.

[112] Marcus M. Edvall Per E. Rutquist. *PROPT - Matlab Optimal Control Software*. Tomlab Optimization Inc., 1260 SE Bishop Blvd Ste E, Pullman, WA 99163, USA, 1 edition, June 2016.

[113] Anna Petrovskaya and Sebastian Thrun. Model based vehicle detection and tracking for autonomous urban driving. *Autonomous Robots*, 26(2-3):123–139, 2009.

[114] Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. CRC Press, 1987.

[115] Mateusz Przybyła. Detection and tracking of 2d geometric obstacles from lrf data. In *Robot Motion and Control (RoMoCo), 2017 11th International Workshop on*, pages 135–141. IEEE, 2017.

[116] Christopher Rackauckas and Qing Nie. Differentialequations. jl–a performant and feature-rich ecosystem for solving differential equations in julia. *Journal of Open Research Software*, 5(1), 2017.

[117] Dan Negrut Radu Serban. *PROJECT CHRONO*. http://projectchrono.org/, February 2019.

[118] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science and Business Media, 2011.

[119] Anil V Rao. User's manual for gpocs c version 1.0: A matlab® implementation of the gauss pseudospectral method for solving multiple-phase optimal control problems, 2007.

[120] Anil V Rao, David A Benson, Christopher Darby, Michael A Patterson, Camila Francolin, Ilyssa Sanders, and Geoffrey T Huntington. Algorithm 902: Gpops, a matlab software for solving multiple-phase optimal control problems using the gauss pseudospectral method. *ACM Transactions on Mathematical Software (TOMS)*, 37(2):22, 2010.

[121] Jarrett Revels. Reversediff. https://github.com/JuliaDiff/ReverseDiff.jl, 2017.

[122] Richard E Rosenthal. Gams–a user's guide. 2004.

[123] I Michael Ross. A beginner's guider to dido: A matlab application package for solving optimal control problem. *http://www. elissar. ziz*, 2007.

[124] 2.1 edn. RWTH Aachen University, Germany. *DyOS User Manual*. 2002.

[125] Joan Pau Sanchez and Daniel Garcia Yarnoz. Asteroid retrieval missions enabled by invariant manifold dynamics. *Acta Astronautica*, 127:667 – 677, 2016. Asteroid mission;Easily retrievable objects;Libration point orbits;Low thrust;Trajectory designs;.

[126] RWH Sargent. Optimal control. *Journal of Computational and Applied Mathematics*, 124(1-2):361–371, 2000.

[127] Pierre OM Scokaert and James B Rawlings. Feasibility issues in linear model predictive control. *AIChE Journal*, 45(8):1649–1659, 1999.

[128] Radu Serban, Michael Taylor, Dan Negrut, and Alessandro Tasora. Chrono:: Vehicle–template-based ground vehicle modeling and simulation. Technical report, Technical Report TR-2016-10, 2016.

[129] Levente L Simon, Zoltan K Nagy, and Konrad Hungerbuehler. Swelling constrained control of an industrial batch reactor using a dedicated nmpc environment: Optcon. In *Nonlinear Model Predictive Control*, pages 531–539. Springer, 2009.

[130] Matthew J Tenny, James B Rawlings, and Rahul Bindlish. Feasible real-time nonlinear model predictive control. In *AICHE SYMPOSIUM SERIES*, pages 433–437. New York; American Institute of Chemical Engineers; 1998, 2002.

[131] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *Journal of Field Robotics*, 23(9):661–692, 2006.

[132] Madeleine Udell, Karanveer Mohan, David Zeng, Jenny Hong, Steven Diamond, and Stephen Boyd. Convex optimization in julia. In *Proceedings of the 1st First Workshop for High Performance Technical Computing in Dynamic Languages*, pages 18–28. IEEE Press, 2014.

[133] Trygve Utstumo, Therese W Berge, and Jan Tommy Gravdahl. Non-linear model predictive control for constrained robot navigation in row crops. In *Industrial Technology (ICIT), 2015 IEEE International Conference on*, pages 357–362. IEEE, 2015.

[134] Rayomand Vatcha and Jing Xiao. Practical motion planning in unknown and unpredictable environments. In *Experimental Robotics*, pages 883–897. Springer, 2014.

[135] E. Velenis, P. Tsiotras, and Lu Jianbo. Modeling aggressive maneuvers on loose surfaces: The cases of trail-braking and pendulum-turn. In *European Control Conference*, pages 1233–40, 2007.

[136] E Velenis, P Tsiotras, and J Lu. Aggressive maneuvers on loose surfaces: Data analysis and input parametrization. In *Control & Automation, 2007. MED'07. Mediterranean Conference on*, pages 1–6. IEEE, 2007.

[137] Oskar von Stryk. Dircol. *Internet/WWW*, 2001.

[138] Peng Yao, Honglun Wang, and Zikang Su. Real-time path planning of unmanned aerial vehicle for target tracking and obstacle avoidance in complex dynamic environment. *Aerospace Science and Technology*, 47:269–279, 2015.

[139] Yongsoon Yoon, Jongho Shin, H Jin Kim, Yongwoon Park, and Shankar Sastry. Model-predictive active steering and obstacle avoidance for autonomous ground vehicles. *Control Engineering Practice*, 17(7):741–750, 2009.

[140] Xiaojing Zhang, Alexander Liniger, and Francesco Borrelli. Optimization-based collision avoidance. *arXiv preprint arXiv:1711.03449*, 2017.