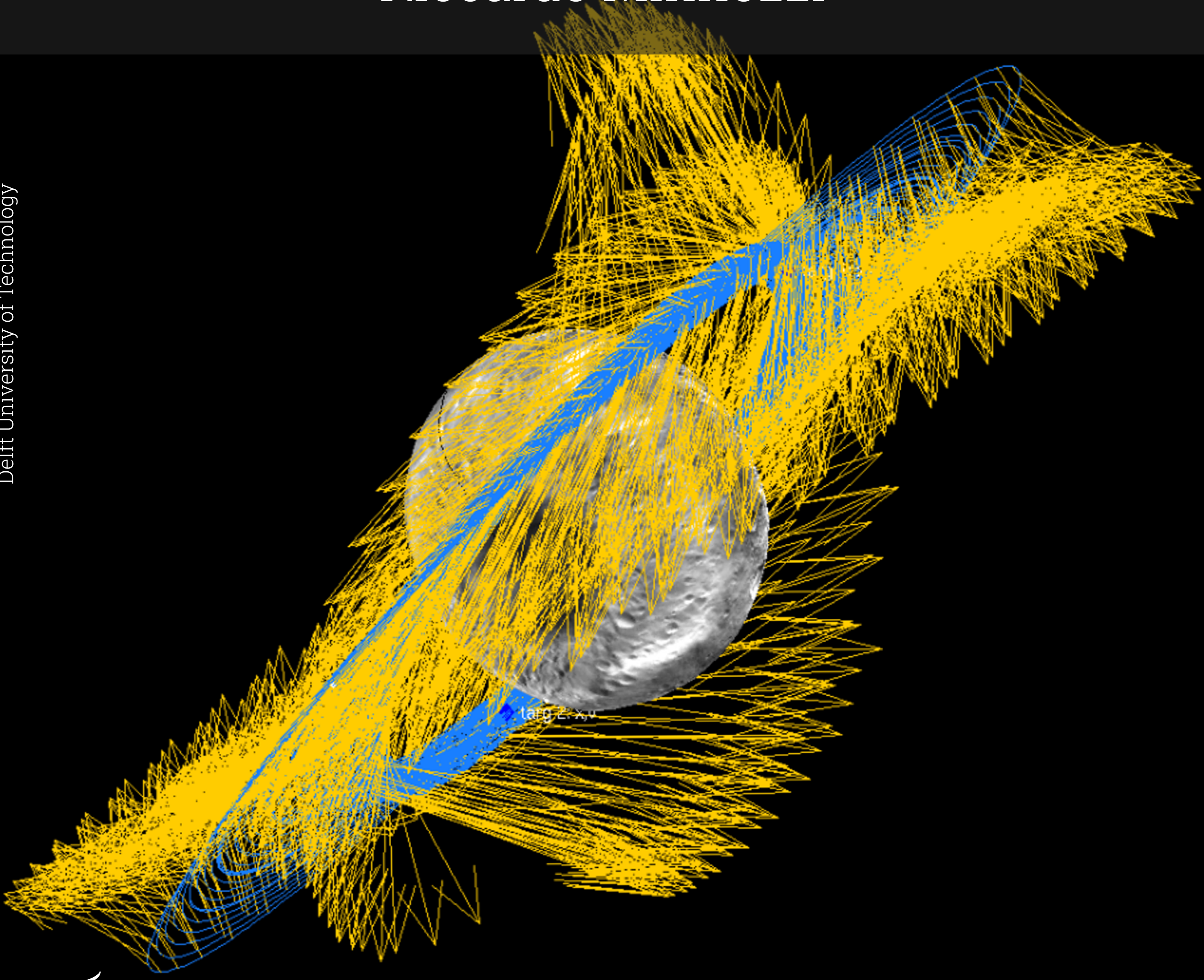
Differential Dynamic Programming for the Optimization of Many-Revolution Solar-Sail Transfers

Riccardo Minnozzi

Delft University of Technology



TUDelft

Differential Dynamic Programming for the Optimization of Many-Revolution Solar-Sail Transfers

by

Riccardo Minnozzi

Student Name	Student Number
Minnozzi Riccardo	5856418

Supervisor:	Dr. Ir. J. Heiligers
Co-Supervisor:	Ir. F. Gamez Losada
Project Duration:	April, 2024 - March, 2025
Faculty:	Faculty of Aerospace Engineering, Delft

Cover: Thrust direction for Vesta proximity operations [1]

Preface

Well ...
it feels a bit unreal to be at the end of this long journey. First of all a heartfelt thank you to my supervisors, Jeannette and Fernando, for the support, kindness, and interest shown throughout the whole thesis. I sincerely believe in the potential of this work and I hope it will be useful in future studies.

I would really like to have the words to describe how grateful I am for all the friends I have made throughout these years: being able to go almost anywhere and have the opportunity to feel at home is a huge privilege, and you all are the reason why I have this.

Thank you to my friends here in Delft, for the support and the shared memories in the past years. Thank you to my friends from Turin for being there like we were never separated, I am looking forward to hugging you all again (yes, I have discovered your ”sneaky” trip to Delft). Thank you to my friends from Camerino, who have inexplicably managed to be here despite the ample availability of sleeping accommodations (my floor). As you can see I am not good with words, and I would much rather tell you all in person how much your presence means to me and celebrate this moment together.

Finally, I guess it’s time to thank my parents, who have made all of this possible, who have supported me throughout these years away from home, and who have always been a beacon of safety and a point of trust. I don’t think I’ll ever be able to thank you enough, but I hope to have made you proud,

Thank you

Riccardo Minnozzi
Delft, March 2025

Summary

Planet-centered applications of solar sails, though less studied in literature than interplanetary applications, offer promising perspectives for active debris removal, in-orbit servicing, and scientific exploration missions. The dynamics and operations of solar sails around a planet differ considerably from those in an interplanetary environment. Namely, the small magnitude of attainable solar-sail thrust with respect to central body gravity implies reduced orbit control authority, which is further affected by frequent eclipsing phenomena. These features result in the need for many orbital revolutions to accomplish a single transfer.

Optimization problems resulting from such orbital transfers imply many decision variables. Direct optimization methods poorly scale with the size of the decision variables vector. Indirect optimization methods, while not showing similar numerical limitations, are highly sensitive to initial guesses, which are difficult to produce due to the generic lack of knowledge on optimal planet-centered solar sail transfers. Recently a Q-law approach, consisting of an objective-informed Lyapunov controller, has been successfully used to investigate many-revolution transfers using solar sails: this technique does not enforce optimality conditions, thus only generating near-optimal solutions.

An alternative approach is therefore identified in Differential Dynamic Programming (DDP). The DDP algorithm offers promising numerical properties, showing a wide convergence basin, numerical stability, and only linear scaling in computational requirements with problem size. While current DDP approaches have been demonstrated to effectively address computational limitations caused by high-dimensional problems, expansions are required to properly address optimal sail-powered transfers: these expansions include reductions in the algorithm's number of hyper-parameters, the capability to handle problems where travel time is a decision variable (i.e., rendezvous scenarios), and the ability to reliably and efficiently enforce path constraints (i.e., dynamical limitations of). The objective of this thesis is to address the main limitations of the DDP algorithm by introducing novel methodologies for the flexible-final-time formulation of the DDP algorithm, the enforcement of path constraints up to second order, and the automatic tuning of key hyper-parameters. The devised algorithm is implemented following Object-Oriented Programming (OOP) paradigms to guarantee a modular, readable, and re-usable implementation: by providing a reliable optimization algorithm, with insights into its tuning process, this work aims at enabling future efforts into the application of DDP to high-dimensional optimal control problems. The developed algorithm is used to identify and analyze optimal many-revolutions solar sail transfers around Earth.

The capability to optimize flexible-final-time problems is introduced through time-dilation: the problem duration is parameterized and handled as a static decision variable. The technique is implemented by modifying the variational equations approach to embed information on variable final time directly within the state transition maps used in the DDP algorithm. Path constraints are enforced by analytically solving for optimality conditions on a second-order approximation of each DDP stage sub-problem. The DDP back-propagation of a quadratic cost model is extended to the constraint violation partial derivatives, enabling an automatic tuning of the penalty parameter. A relaxation technique for the DDP trust-region procedure is also introduced, reducing sensitivity to hyper-parameters related to the accuracy of the DDP quadratic cost model. The devised algorithm is implemented in MATLAB® and integrated with an automatic differentiation package.

The resulting DDP solver is applied to a simplified dynamical model for the identification of optimal co-planar circular-to-circular solar sail transfers. The dynamical model considers the effects of central-body gravity, solar radiation pressure, and eclipses. Numerical stability is improved through scaling and a variable transformation. The algorithm is successfully verified and validated against a state-of-the-art direct optimization solver. The algorithm behavior according to its hyper-parameters is characterized through several factorial analyses: key findings are summarized to provide a 'rule-set' for future DDP tuning. The algorithm convergence properties over increasing problem dimensionality are also investigated.

The validated algorithm is applied to time-optimal many-revolution solar-sail transfers under different conditions. Two approaches to time-optimal solutions are defined: a fixed-time formulation, where the increase in orbital radius over a specified transfer duration is maximized, and a flexible-final-time formulation, where the transfer time to achieve a specified gain in orbital radius is minimized. The fixed-time formulation is used to analyze solutions at two different altitudes: the diverse illumination conditions, influenced by the orbital altitudes and transfer duration, result in distinct control regimes and optimal orbital geometries, resulting in considerably different orbit-raising performances. The optimized transfers are used to derive power regression models for the solar sail in both the analyzed cases. The performance prediction from the regression model is exploited to initialize the flexible-final-time formulation, successfully identifying a time-optimal circular-to-circular transfer at low orbital altitudes.

The developed algorithm was observed to reliably achieve convergence to optimal solutions even in complex and high-dimensional optimal control problems. The validated implementation and OOP architecture provide a solid starting point for further development of the DDP algorithm. While the obtained results shed light on optimal solar-sail many-revolutions transfers, higher-fidelity dynamical models shall be considered in further analyses. The software designed as part of this thesis is made available to encourage future works on high-dimensional optimal control problems.

Contents

List of Figures

List of Tables

Nomenclature

List of acronyms

ACS	Attitude Control System
ACS3	Advanced Composite Solar Sail System
ADR	Active Debris Removal
AMT	Active Mass Translator
C2C	Circular To Circular
COV	Calculus of Variations
CRTBP	Circular Restricted Three Body Problem
DDP	Differential Dynamic Programming
EOM	Equations of Motion
EP	Electric Propulsion
GEO	Geostationary Equatorial Orbit
GTO	Geostationary Transfer Orbit
GTOC	Global Trajectory Optimisation Competition
HDDP	Hybrid Differential Dynamic Programming
IEP	Ion Electric Propulsion
IKAROS	Interplanetary Kite-craft Accelerated by Radiation Of the Sun
JAXA	Japanese Aerospace Exploration Agency
KKT	Karush-Kuhn-Tucker
LEO	Low Earth Orbit
MBH	Monotonic Basin Hopping
MPBVP	Multi-Point Boundary Value Problem
NASA	National Aeronautics and Space Administration
NEA	Near Earth Asteroid
NLP	Non-Linear Programming
OCP	Optimal Control Problem
OOP	Object-Oriented Programming
PDE	Partial Differential Equations
QP	Quadratic Programming
RCD	Reflective Control Device
RCS	Reaction Control System
SBS	Sail-Boom System
SDC	Static Dynamic Control
SRP	Solar Radiation Pressure
SSA	Solar Sail Acceleration
SST	Solar Sail Thrust
SSO	Sun-Synchronous Orbit
STM	State Transition Map
SWEEP	Space-Waste Elimination around Earth by Photon propulsion
TLE	Two-Line Element
TOP	Trajectory Optimization Problem
TRAC	Triangular Rollable and Collapsible
TRQP	Trust-Region Quadratic Programming
UAV	Unmanned Aerial Vehicle
WBS	Work-Breakdown Structure

Greek symbols

α	Small update to the Hessian-shifted trust-region sub-problem model
β	Sail lightness number
Δ	Trust-region radius
Δ_σ	Penalty update restriction parameter
Δ_{best}	Trust-region radius best estimate
ρ	Quadratic model validity metric
ϵ_1	Quadratic model validity threshold
ϵ_{opt}	Optimality threshold
ϵ_{feas}	Feasibility threshold
$\mathbf{\Gamma}$	Initial conditions parametrization
λ	Hessian shift parameter
γ	Hessian correction term
μ	Gravitational parameter
φ	Terminal cost function
$\tilde{\varphi}$	
$\mathbf{\Psi}$	Terminal constraints function
σ	Penalty parameter
$\boldsymbol{\lambda}$	Lagrange multipliers (terminal constraints)
$\boldsymbol{\mu}$	Lagrange multipliers (path constraints)
Φ^1	State Transition Matrix
Φ^2	State Transition Tensor
$\tilde{\sigma}$	Sail loading
κ_ϵ	Quadratic model validity threshold update parameter
κ_{easy}, κ_{hard}	Convergence tolerances in different cases for the robust trust-region solver

κ_d	Trust-region radius update parameter
$\Delta_{\min}$, $\Delta_{\max}$	Limits to the trust-region radius update
κ_σ	Parameter to perform penalty updates
Δt	Step-size for the stage collocation function
$\delta \boldsymbol{u}^*$	Trust-region sub-problem model minimizer

λ^L, λ^U	Hessian shift upper and lower bounds
λ_1	lowest Hessian eigenvalue
θ	Parameter to update Hessian shift λ
ρ_{best}, Δ_{best}	Best accuracy estimates used in the quadratic model validity adaptive enlargement
α	Sail cone angle
δ	Sail clock angle

Latin symbols

au	Astronomical unit
a_0	Solar sail characteristic acceleration
A	Surface area
m	Mass
U	Control feedback matrix (for path-constrained problems)
c	Speed of light
L	Running cost function
W	Energy flux per unit of time
$\mathcal{L}_s$	Solar luminosity
$\mathcal{L}_s$	Sun's luminosity
$R_{\oplus\odot}$	Distance between Earth and Sun
$r_{s\odot}$	Distance between generic point and Sun
ΔE	Energy carried by energy flux
Δt	Time interval
P	Pressure
Δp	Momentum transported by particles
$P_\oplus$	Solar radiation pressure at 1 au from the Sun
$W_\oplus$	Solar energy flux at 1 au from the Sun
J	Cost functional
t	Independent variable
$\boldsymbol{x}$	State vector
$\boldsymbol{u}$	Control inputs
$\mathcal{U}$	Set of admissible controls
$\mathcal{W}$	Set of admissible parameters
$\boldsymbol{f}$	Dynamics function
$\dot{\boldsymbol{x}}$	Time derivative of the state vector
$\boldsymbol{g}$	Path constraints function
$\mathcal{R}$	Real numbers set
n_u	Dimension of the control inputs vector
n_x	Dimension of the state vector

n_w	Dimension of the static parameters vector
n_λ	Dimension of the Lagrange multipliers vector
n_g	Dimension of the path constraints function
n_q	Dimension of the active path constraints set
$N_i + 1$	Number of optimization stages
$\delta \boldsymbol{x}$	State vector variation
$\delta \boldsymbol{u}$	Control update
$\bar{\boldsymbol{x}}$	Reference trajectory state vector
$\bar{\boldsymbol{u}}$	Reference trajectory control input
$\hat{r}_{\odot s}$	Unit vector in the direction from Sun to satellite (the Sun-line)
$\hat{n}$	Sail normal unit vector
f	Feasibility metric
h	Optimality metric
$\boldsymbol{X}$	Augmented state vector
ER	Expected cost-to-go reduction
AR	Actual cost reduction
$\mathcal{P}$	Generic penalty function
M	Number of phases
H	Problem Hessian matrix
H_R	Reduced Hessian matrix
Z	Null space of the constraints Jacobian
$\mathcal{M}$	Matrix in the affine-term equations for the path-constrained stage sub-problem
$\mathcal{H}$	Hessian in the affine-term equations for the path-constrained stage sub-problem
A_k, B_k, C_k, D_k	Controls feedback law terms
$A_{\boldsymbol{\lambda}_+}$, $C_{\boldsymbol{\lambda}_+}$	Lagrange multipliers feedback law terms

$A_{\boldsymbol{w}_+}$, $B_{\boldsymbol{w}_+}$, $C_{\boldsymbol{w}_+}$, $D_{\boldsymbol{w}_+}$	Parameters feedback law terms
$\mathcal{D}$	Trust-region scaling matrix
$\tilde{\boldsymbol{w}}$	Measure of gradient of the secular equation
L_c	Cholesky factorization matrix
$\mathcal{G}_{\updownarrow}$, $\mathcal{L}_l$, $\mathcal{F}_l$, $\mathcal{N}_l$	Different sub-sets of the Hessian shift available range
d_c, v_c	Results of the partial Cholesky factorization
$\boldsymbol{q}$	Active set of constraints in a stage sub-problem
$\boldsymbol{Y}$	Set of non-controls decision variables
I_{opt}	Optimality improvement gained from mesh refinement
$\hat{\boldsymbol{b}}$	Improvement direction for trust-region sub-problem model computed using LINPACK

Subscripts

f	Final quantity
0	Initial quantity
k	Quantity at generic stage k
N	Quantity at final optimization stage N
i	Quantity at generic phase i
$+$	Quantity at inter-phase on the first stage of the upstream phase
$-$	Quantity at inter-phase on the last stage of the previous phase
$\odot$	Sun
$\oplus$	Earth

1 Introduction

Solar sailing is a novel spacecraft propulsion method that makes use of a thin mirror-like structure (the sail) to harness the momentum carried by Sun-emitted photons, applying a continuous acceleration on the spacecraft. The concept was first theorized by Johannes Kepler in the 17^{th} century, who noticed that comet tails are always pointed away from the Sun, and proposed that the future of space travel could be based on the ability to make use of this ”breeze” to propel ships. The first real application of the concept can be traced back to 1975 when the National Aeronautics and Space Administration (NASA) used it to address the reduced propellant margin of the Mariner 10 spacecraft [2].

Thanks to its propellant-less nature, solar sailing has been the focus of multiple studies for applications in interplanetary environments. Growing interest in planet-centered applications of the solar sail concept has been noticed in recent years, ranging from scientific observation [3], to on-orbit servicing and active debris removal [4]. Operating solar sails around a planet, however, presents significant challenges. Namely, the small magnitude of the attainable solar sail thrust results in reduced orbit control authority, which is further affected by the frequent eclipsing phenomena: performing orbital transfer maneuvers thus requires long transfer times, resulting in many revolutions being necessary to accomplish the maneuver.

This thesis work focuses on developing and testing an optimization framework to efficiently tackle the optimal control problem resulting from many-revolution solar-sail transfers. The document is structured as follows. First, a literature review is presented in Chapter 2, diving into works on solar sailing, optimal control, and the Differential Dynamic Programming algorithm (chosen as the thesis focus), ultimately defining a research objective and related questions. Then, the developed methodology is illustrated in Chapter 3, with technical information regarding the full optimization solver and its implementation. The novel methodologies and obtained results are presented in the form of a journal article in Chapter 4. Finally, the outcome of the work is summarized in Chapter 5, with insight into key findings, limitations, and further research opportunities.

2 Literature Review

This chapter provides an overview of the relevant literature on the topics of solar sailing and optimal control, ultimately leading to the research objective formulation. First, the current state of solar sailing technology is presented in Section 2.1, focusing on flown missions and applications within planet-centered environments. Then, works on optimal control applied to low-thrust trajectory optimization transfers are displayed in Section 2.2, providing a brief overview of constrained optimization techniques, illustrating the features and limitations of state-of-the-art direct, indirect, and heuristics-based solution methods. Subsequently, constrained optimization approaches are summarized in Section 2.3, presenting their features and applicability to the desired optimal control problem. Finally, the algorithm chosen to carry out this study is presented in further detail in Section 2.4. The derived research objective and questions are stated in Section 2.5.

2.1. Solar sailing

Solar sails exploit the momentum carried by Sun-emitted photons to generate an acceleration on the spacecraft [2]. The energy flux per unit of time carried by solar radiation is described as:

$$W = \frac{\mathcal{L}_S}{4\pi R_{\oplus\odot}^2}(\frac{R_{\oplus\odot}}{r_{s\odot}})^2 \tag{2.1}$$

where $\mathcal{L}_S$ is the solar luminosity, $R_{\oplus\odot}$ is the Sun-Earth distance ($R_{\oplus\odot} = 1\ au = 149597870700\ m$), and $r_{s\odot}$ is the distance between the Sun and a generic object s in space. Evaluating Eq. 2.1 at the Sun-Earth distance ($r_{s\odot} = R_{\oplus\odot}$) yields the solar flux constant $W_{\oplus} = 1368\ \frac{J}{m^2 s}$ [5]. The solar luminosity, and consequently also the solar flux constant $W_{\oplus}$, can vary depending on solar weather conditions (such as the solar cycle and coronal mass ejections) [5].

When interacting with finite objects, the energy flux induces a net force $F = PA_{eff}$ which depends on the effective area A_{eff} exposed to the flux and a pressure term P, referred to as Solar Radiation Pressure (SRP). If the black-body assumption is introduced (all radiation is absorbed), and incident radiation is approximated as parallel light beams, the SRP acting on a generic object s is:

$$P = \frac{W}{c} = \frac{\mathcal{L}_S}{4\pi c R_{\oplus\odot}^2}(\frac{R_{\oplus\odot}}{r_{s\odot}})^2 \tag{2.2}$$

where c is the speed of light ($c = 299792000 m/s$).

An ideal model for the SRP acting on a solar sail is obtained by assuming parallel incoming sunlight, infinitely rigid, and perfectly reflecting sail membrane [6], and will be now referred to as the ideal sail model. For a fully reflective object, the pressure computed in Eq. 2.2 doubles, since photon-flux momentum is transferred from both absorbed and reflected radiation. Under these assumptions, the solar-sail performance can be characterized using different parameters: the sail loading, which is a representation of the sailcraft design parameters, the characteristic acceleration, which provides an intuitive metric for the sail thrusting capabilities at a specified distance from the Sun, and the lightness number, which quantifies the sail performance regardless of its distance from the Sun. The definitions of each parameter are now provided:

$\tilde{\sigma}$: the sail loading $\tilde{\sigma}$ corresponds to the total spacecraft mass to area ratio:

$$\tilde{\sigma} = \frac{m}{A} \tag{2.3}$$

where m is the total spacecraft mass and A is the sail surface. Higher-performing sailcraft exhibit low sail loading $\tilde{\sigma}$ values, as it implies low mass and/or large sail area, resulting in higher attainable acceleration.

a_0 : the characteristic sail acceleration a_0 is defined as the acceleration produced by an ideal solar sail, oriented perpendicularly to the incoming Sunlight, at $1\ au$ distance from the Sun, thus:

$$a_0 = 2\frac{P_{\oplus}A}{m} = 2\frac{P_{\oplus}}{\tilde{\sigma}} \tag{2.4}$$

where $P_{\oplus}$ is the SRP value at $1\ au$ ($P_{\oplus} = \frac{W_{\oplus}}{c} = 4.56 \cdot 10^{-6}\ \frac{N}{m^2}$). The value for a_0 provides an intuitive metric for the performance of a sail at the Earth's distance from the Sun.

β : the lightness number β is defined as the dimensionless ratio between maximum SRP and Sun gravitational acceleration: since both quantities are assumed to scale with the inverse square of spacecraft distance from the Sun $r_{s\odot}$, the lightness number is a design constant (i.e.: independent from mission scenario). Its mathematical formulation is:

$$\beta = \frac{a_0 R^2_{\oplus\odot}}{\mu_{\odot}} = \frac{a_0}{5.93 \cdot 10^{-6}\ m/s^2} \tag{2.5}$$

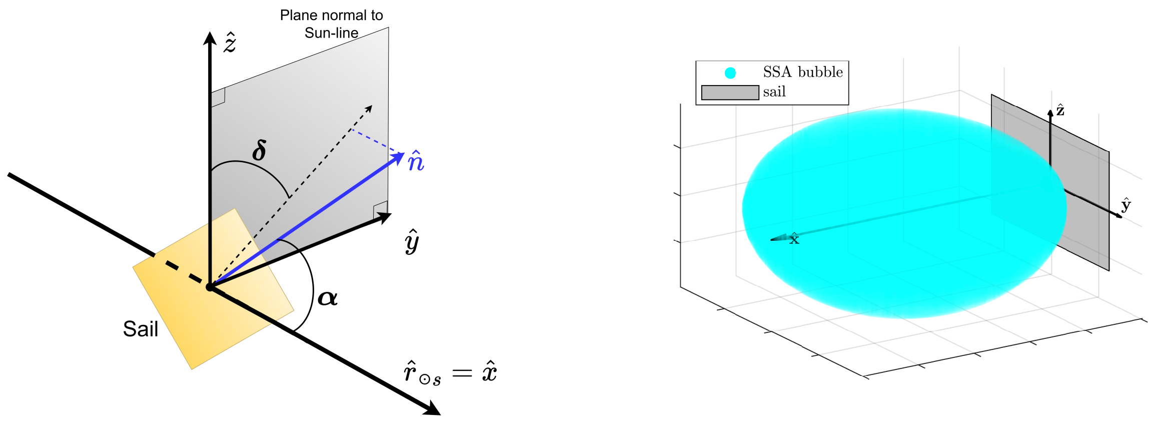
where $\mu_{\odot}$ is the Sun's gravitational parameter ($\mu_{\odot} = 1.327 \cdot 10^{20}\ \frac{m^3}{s^2}$). These three parameters provide interchangeable ways to quantify the sailcraft performance.

Under the assumption of an ideal sail model, the generated Solar Sail Thrust (SST) is directed along the sail-normal vector (small directional variations can occur if these assumptions are dropped [7]) and cannot have a component in the direction of the Sun [6]. The sailcraft orientation with respect to the Sun-line (i.e., the direction connecting the Sun to the satellite) can be described through the cone-clock angle pair $\alpha,\ \delta$. The cone angle α is the angle between the sail-normal vector and the Sun-line and represents the sail illumination condition, while the clock angle δ is the angle, defined in a plane normal to the Sun-line, between a reference direction and the in-plane projection of the sail-normal vector. An illustration of these angles is provided in Figure 2.1a: the sail-normal vector $\hat{\boldsymbol{n}}$ is defined in the Sun-light frame $\mathcal{S}(\hat{\boldsymbol{x}},\ \hat{\boldsymbol{y}},\ \hat{\boldsymbol{z}})$, with origin on the sailcraft center of mass, the $\hat{\boldsymbol{x}}$ axis directed along the Sun-line, the $\hat{\boldsymbol{z}}$ axis normal to the ecliptic plane (in the direction of Earth's North pole), and the $\hat{\boldsymbol{y}}$ axis completing the right-handed reference frame.

Sailcraft orbit control is exerted by modifying its attitude: altering the sail-normal vector $\hat{\boldsymbol{n}}$ leads to variations in effective area A_{eff} exposed to SRP, as well as the resulting SSA direction. Coupling of these effects determines the so-called SSA bubble: Figure A.2 depicts the contours of all attainable SSA vectors $\boldsymbol{a}$ (the SSA bubble) in the Sun-light frame $\mathcal{S}(\hat{\boldsymbol{x}},\ \hat{\boldsymbol{y}},\ \hat{\boldsymbol{z}})$. The bubble shape of the SSA contour translates to significant losses in thrusting capability when the sailcraft is oriented in a direction significantly different from the Sun-line (i.e., large cone angle α values). While higher-fidelity dynamical models imply variations to these considerations [7], it holds that solar sail control authority is inherently limited, with constrained magnitude and direction.

2.1.1. Solar-sail technology

At the Sun-Earth distance, the SRP has small magnitude (i.e., from Eq. 2.2 a value $P_{\oplus} = 4.56 * 10^{-6}\ \frac{N}{m^2}$ is obtained). To produce significant SSA, a small sail loading $\tilde{\sigma}$ is required: large sail area A ensures the generation of significant SST, while a small mass enables high acceleration. Current design practices consist of fabricating the solar sail itself (in the form of a membrane) from highly reflecting, lightweight materials, which require specific mechanisms/configurations to maintain the sail shape and transfer the generated SST to the satellite [6]. The full structure is tightly stored within the spacecraft bus to



(a) **(b)**

Figure 2.1: Schematic illustrations of the solar-sail-normal vector $\hat{\boldsymbol{n}}$ orientation defined using the cone angle α and clock angle δ (left side) and of the SSA bubble (right side). Both illustrations use the Sun-light frame $\mathcal{S}(\hat{\boldsymbol{x}},\ \hat{\boldsymbol{y}},\ \hat{\boldsymbol{z}})$

comply with the tight volume capacity of current launch vehicles, only deploying the solar sail after orbit injection. Different solar-sail designs have been theorized (i.e., square sails, heliogyros [8], disc sails, and solar photon thrusters [9]), but few actual technological demonstrations have been achieved. An overview of flown solar-sailing missions is now provided and summarized in Table 2.1, with the goal of outlining technological trends, state-of-the-art, and future perspectives in solar sail missions.

IKAROS

The Interplanetary Kite-craft Accelerated by Radiation Of the Sun (IKAROS) mission was the first milestone in the field of solar sailing. It was developed by the Japanese Aerospace Exploration Agency (JAXA) and successfully launched in 2010 on a Venus transfer trajectory, with the ultimate goal of performing a Venus fly-by [10]. The objectives accomplished by this mission are:

- Demonstrate the deployment mechanism for the solar sail;
- Generate electric power using the thin flexible solar arrays attached to the sail;
- Demonstrate the possibility of navigating using SRP in the interplanetary environment;
- Estimate direction and magnitude of the SST.

The square sail used in the IKAROS mission had an area of $200\ m^2$ and a mass of $15\ kg$, yielding a total spacecraft mass of approximately $300\ kg$ [11]. Attitude control was achieved through spin-stabilization, aided by an Attitude Control System (ACS) which included a cold-gas thrust Reaction Control System (RCS) and a Reflective Control Device (RCD) (i.e., liquid crystals with variable optical properties to shift the sail center of (radiation) pressure [12]). After the two-step deployment process, the sailcraft proceeded on its trajectory to Venus: the registered velocity increase ($100\ m/s$ within the first 6 months [13]) verified the viability of SRP as a propulsion method in the interplanetary environment [11]. Analyses performed on the IKAROS mission data were the first bases for the development of accurate solar-sail dynamical models [12, 14].

NanoSail-D

On a smaller scale, NASA's NanoSail-D aimed at demonstrating sail deployment and the concept of a drag-sail as a passive de-orbiting method. After a launch vehicle failure during the first attempt, the replica NanoSail-D2 was successfully launched and deployed in 2011 [15]. The NanoSail-D2 sailcraft consisted of a $3U$ cubesat platform (meaning a volume of $10\ cm \times 10\ cm \times 30\ cm$), mounting a square sail with a surface of $10\ m^2$ and total mass of $4\ kg$ [16]. The sailcraft remained in orbit for 243 days without active control of the sail and managed to maintain ground communications for only 3 days before running out of power. Two-Line Element (TLE) data showed that the sail, even if uncontrolled, was successfully serving its de-orbiting purpose [15], making it the first technology demonstration for space debris mitigation enabled by a drag-sail and also the first sailcraft mission in Earth-orbit.

LightSail-1

The LightSail-1 mission was launched in 2015 on a ride-share mission to Low Earth Orbit (LEO), targeting a perigee altitude of $356\ km$ [17]). The sailcraft consisted of a $3U$ cubesat bus, hosting a square sail with a surface of $32\ m^2$, for a total mass of $4.93\ kg$. Given the low perigee altitude (i.e., dominance of aerodynamics over SRP), the mission objectives were limited to demonstrating the cubesat functionality and sail deployment [18]. Its ACS therefore only included magnetic torque rods (while a momentum-wheel was deemed unnecessary given the mission objectives [18]). LightSail-1 successfully achieved its goals and re-entered Earth's atmosphere only a week later due to its low perigee altitude and the large area-to-mass ratio [18].

LightSail-2

After the successful demonstration of LightSail-1, the follow-up mission LightSail-2 aimed at demonstrating controlled sailing in LEO and was launched into a $720\ km$ altitude near-circular orbit [17]. The spacecraft-bus design was shared with its predecessor, augmented by introducing a momentum wheel to enable active attitude control for sail steering [17]. After successful deployment, LightSail-2 began implementing its "On-Off" control logic (with "On" corresponding to the sail-normal vector being pointed away from the Sun, maximizing energy gain, and "Off" corresponding to the sail being oriented edgewise to minimize drag-induced losses) [19]. The observed increase in apogee altitude and the reduced decay rate in semi-major axis ($19.9\ m/day$ on average, against the decay rate of $34.5\ m/day$ for uncontrolled operations) demonstrated the feasibility of Earth-bound solar sailing in LEO [20].

NEA Scout

Launched in 2022, the NEA Scout mission aimed at demonstrating sail controllability in an interplanetary environment, by performing a close fly-by of a Near Earth Asteroid (NEA) [21]. The solar-sail propulsion was chosen as it proved to be the only concept capable of ensuring mission accomplishment in a reasonable time (2 years) even under high uncertainties in both launch date and NEA target positions [21]. Despite the failure to establish communications after orbit injection, the mission is still considered a relevant endeavor for its ambitious objective. The sailcraft was designed by scaling the NanoSail-D concept, leading to a $6U$ bus, storing a sail with a surface of $86\ m^2$ and a total mass of $15.9\ kg$. The sailcraft ACS consisted of a cold-gas RCS for de-tumbling and an Active Mass Translator (AMT) system to steer and trim the sailcraft by shifting its center of mass [21].

ACS3

Launched in 2024, NASA's Advanced Composite Solar Sail System (ACS3) mission aims at demonstrating new composite materials for the sail-boom system, characterizing SST and flexible dynamics for future low-cost cubesat missions (as opposed to other poorly scalable propulsion methods, such as Ion Electric Propulsion (IEP)) [22]. The sailcraft uses a $12U$ cubesat platform, hosting a sail with a surface of $\simeq 80\ m^2$ and mass of $12\ kg$, steered through reaction wheels and magnetic torque rods [22]. The sailcraft is injected into a midnight-noon Sun-Synchronous Orbit (SSO) at an altitude of $1000\ km$ [22], where energy gain can be maximized by slewing the sail by 90° per orbit [6], with additional steering laws used to estimate the sail optical properties [23].

A summary of the presented missions, together with their objectives, design choices, and performance metrics is provided in Table 2.1.

Table 2.1: Summary of mission characteristics

Mission	β [−]	Launch date	ACS	SBS	Objective
IKAROS	0.001	2010	Spin-stabilization, cold-gas RCS, RCD	Square sail, centrifugal deployment and tensioning	Demonstate deployment, power production and navigation capability of the sail, characterize SRP, perform Venus fly-by
NanoSail-D	0.003	2011	No ACS	Square sail, TRAC booms deployed by electric motors	Demonstrate sail de-orbiting capabilities and deployment
LightSail 1/2	0.099	2015/2019	Magnetic torque rods, momentum wheel	Square sail, TRAC booms deployed by electric motors	Demonstrate sail deployment and controlled solar sailing
NEA-Scout	0.01	2022	Cold-gas RCS, AMT	Square sail, TRAC booms with redundant spooling and electric motor deployment	Perform close fly-by of a NEA, demonstrate controlled solar sailing in interplanetary environment
ACS3	0.008	2024	Magnetic torque rods, reaction wheels	Square sail, composite material booms	Demonstrate new deployment mechanism and orbit raising capabilities

Trends observed from Table 2.1 indicate a clear preference towards Earth-bound missions, as they imply lower costs and a well-established operational framework. Sailcraft designs (limited to the square sail concept) are observed to move towards low-cost cubesat-scale platforms [22], implying that near-future applications for Earth-bound solar sailing will maintain similar performance metrics to those included in Table 2.1.

2.1.2. Solar sailing in the Earth environment

Due to the small values of attainable SSA and reduced steering authority, solar sails are best suited for missions in interplanetary environments, where slow attitude adjustments, constant exposure to SRP, and relatively minor gravitational effects (with respect to planet-centered missions) are implied. For these reasons early studies on solar sailing mainly focused on interplanetary missions, avoiding the challenges/drawbacks of solar sailing around a planet [24]. However, interest in planet-centered solar sailing missions has recently increased, thanks to the growing mission heritage (as observed in Table 2.1), the benefits of propellant-free continuous thrust, and concerning evolutions of the Earth orbit environment (namely, the acknowledgment of space debris as an environmental threat [25]). Studies on potential solar-sail planet-centered applications are now introduced, followed by considerations on the dynamics and operations of solar sails in such environments.

Science missions

In Reference [26], spacecraft with high area-to-mass ratios (e.g., solar sails) are identified for application to Earth's imaging studies. Thanks to dynamical perturbations induced by Earth's oblateness and SRP, the study identifies a set of 'quasi-frozen' orbits (i.e., orbits where long-term perturbations are minimized) with Sun-pointing apogee, achievable by low-cost spacecraft with small reflective devices. The

Sun-pointing apogee implies long exposure to the sunlight, thus enabling constant telecommunication support or imaging of Earth's illuminated side.

In Reference [27] (and later in Reference [28]) the SST is identified as a solution to define SSOs around Mercury. While spacecraft around Earth can benefit from its oblateness to induce nodal precession, thus enabling SSOs, the spherical shape of Mercury does not allow such practices: solar sails in highly elliptical polar orbits are shown to provide a suitable effect to enable SSOs. Under such conditions, the sailcraft orbit can be slightly shifted away from the solar terminator line (i.e., the line corresponding to sunset/sunrise conditions) such that fully illuminated orbital arcs can be exploited for effective planetary imaging.

In Reference [3], a small low-cost solar sail is identified as a candidate solution for the exploration of Earth's magnetic tail: leveraging the sail's continuous thrust, the mission can benefit from long exposure to the plasma within the magnetic tail, enabling the characterization of long-term behaviors.

Active Debris Removal

Solar sail technology is being investigated as a potential propulsion method for Active Debris Removal (ADR) missions. Due to the increasing number of objects being launched into orbit, combined with the lack of measures to dispose of non-responsive or fragmented satellites (referred to as space-debris), the Earth orbit environment is reaching capacity [25]. As debris continues to collide and fragment, the cascading effect may cause the debris population to grow uncontrollably (i.e., the Kessler syndrome), making some orbital regions unusable for future missions. The adoption of ADR measures is seen as a promising solution to the space debris problem [29]. Thanks to the propellant-less nature and typically long mission lifetimes, solar sails provide advantageous platforms for this endeavor, enabling the targeting and de-orbiting of multiple debris objects without the need for replacement or refurbishment. Studies such as the TugSat concept [4] and the Space-Waste Elimination around Earth by Photon propulsion (SWEEP) project [30] focus on this application.

Operational considerations

Despite the many potential applications, solar sailing around Earth also presents significant challenges. The Earth orbit environment is characterized by considerable perturbing forces, meaning that sailcraft dynamics are governed not only by Earth's gravity and SRP, but also aerodynamic effects, third-body perturbations caused by the Sun and Moon, and frequent eclipsing phenomena. Most noticeably, the shape and size of solar sails make them particularly sensitive to aerodynamic drag, defining a so-called operational altitude, below which the sail loses its orbit-increase capabilities ($450 - 600\ km$, depending on current solar activity) [31]. Additionally, large sail areas imply increased collision risk with the dense debris objects population. Given such dynamical considerations, the sail dynamics are highly non-linear.

The SST nature implies that it is constrained (and coupled) in magnitude and direction. The resulting orbit control authority is considerably reduced (with respect to other spacecraft propulsion methods) and implies long orbit-transfer durations, leading to many revolutions being necessary to accomplish a transfer. In the planetary environment, solar sailing requires frequent attitude adjustments [24]. These features significantly impact the definition and solution of solar-sail transfer optimization problems.

2.2. Optimal control

Given the high operation and development costs, space missions are designed to optimize certain performance metrics while fulfilling their objectives. As a result, the mission design problem is often formulated as a Trajectory Optimization Problem (TOP), consisting of determining the trajectory (i.e., the time history of vehicle states) that minimizes certain quantity/quantities, while satisfying some initial and final conditions (if required) [32]. While in most studies ”trajectory optimization” and ”optimal control” are used interchangeably, it is to be noted that the TOP is usually restricted to optimizing a set of parameters, allowing a simplified representation of candidate trajectories, and a later reconstruction of the corresponding controls, if any exist [33]. Conversely, the Optimal Control Problem (OCP) aims at identifying dynamic control inputs that minimize certain metrics: by implicitly accounting for vehicle dynamics, the optimal control approach provides more reliable and representative solutions [34].

Solutions to both TOPs and OCPs have been the focus of multiple studies in the aerospace field, ranging from orbital transfers, to planetary entries, to rendezvous/docking maneuvers [35]. More relevant towards the scope of this work are studies on low-thrust propulsion transfers [33] (i.e., orbital transfers enabled by limited thrust magnitudes for prolonged duration and with high fuel efficiency). While chemically propelled trajectories can be approximated to a finite number of impulsive shot maneuvers, the low-thrust OCP requires finding a continuous steering/thrusting law, significantly increasing problem dimensionality (number of decision variables) [36]. The low control authority of low-thrust propelled spacecraft around a planet causes many local minima and stationary points in the resulting OCP [33]. The mentioned numerical challenges, combined with highly non-linear spacecraft dynamics, make for considerably difficult OCPs [37]. The solution of low-thrust Electric Propulsion (EP)-based transfer OCPs has drawn considerable interest and led to numerous studies [33], while only in later years similar concepts and approaches started being applied to solar sailing problems [38, 39, 40].

Given the complexity and dimensionality of OCPs resulting from planet-centered low-thrust orbital transfers, it is often difficult to derive analytical solutions, therefore numerical optimization methods are employed. Depending on the technique adopted to tackle the OCP, these methods can be classified as indirect, direct, and dynamic programming approaches [33]. An additional class of algorithms, based on heuristics, is also presented: despite not enforcing optimality conditions, these methods adopt specific rules/policies to steer solutions to near-optimal conditions [41]. The following sub-sections provide an overview of the mentioned optimization methods, highlighting their advantages and disadvantages for applications in solar sail many-revolutions transfer OCPs and presenting relevant related literature. Numerical optimization techniques shall be able to properly address the main challenges of solar-sail many-revolution transfer OCPs: a wide convergence basin is required to identify optimal solutions starting from poor initial guesses under highly non-linear dynamics [33], enforcing optimality conditions is known to improve solution accuracy and robustness against local minima and stationary points [42], while favorable scaling of the algorithm's computational requirements with problem dimensionality is required, given the high-dimensional nature of the considered OCPs [43]. A summary of the main features of the presented numerical optimization methods is provided in Table 2.2, with colors indicating their ability to properly address the challenges of solar-sail many-revolution transfer OCPs.

2.2.1. Indirect methods

Indirect optimization is based on Calculus of Variations (COV) theory: by enforcing optimality conditions (such as Pontryagin's minimum principle), the OCP is formulated as a set of Hamiltonian equations satisfying Euler-Lagrange theorem [44]. The Hamiltonian is a function of the problem states, co-states (representing the cost function sensitivity with respect to the states) and controls [44]. The set of Hamiltonian equations is transcribed into a Multi-Point Boundary Value Problem (MPBVP) to be solved numerically [44]. This approach ensures high-quality solutions and the satisfaction of first-order optimality conditions. The numerical solution of the resulting MPBVP is highly sensitive to initial guesses. The difficult interpretability of the co-states furthermore implies several challenges in the generation of accurate initial guesses for this approach, affecting its robustness and convergence properties [33]. Additionally, indirect methods require the analytical derivation of optimality conditions for every specific problem formulation (i.e., different dynamical models, constraints, or objectives), restricting their adaptability to various OCP definitions [44].

The applicability of indirect methods to the many-revolutions transfer OCP is mainly limited by their robustness and convergence issues [45]. Several studies to address such issues have been carried out: homotopy methods, consisting of mapping between the 'complete' OCP and a similar, well behaved, formulation, represent the state-of-the-art in such context [46].

A variety of studies have explored the application of indirect optimization methods to low-thrust trajectory design. Initial approaches to the many-revolutions OCP leveraged Edelbaum's approach [47], later extended to the many-revolution problem [48], to obtain initial guesses for EP-powered spacecraft transfers. A homotopy method was used in Reference [46] to identify fuel-optimal EP-powered transfers up to 754 revolutions, while in Reference [45] a Q-law approach was adopted to generate initial guesses, yielding up to 228-revolution transfers.

Indirect optimization methods have also been investigated for applications to solar-sail transfers. Adopting orbit-averaging techniques, [49] attempted to identify optimal many-revolutions Earth-centered solar sail transfers, with limited results due to convergence issues. More recent studies analytically derived locally optimal control laws that maximize rates of change of specified orbital elements, using indirect optimization to blend them and apply them to planet-centered mission scenarios [28, 50]. Similar approaches are applied to scenarios including atmospheric drag [51] and planetary radiation pressure [52]. Overall, literature on indirect optimization offers a promising outlook for future studies but highlights the need for approximations (such as orbit averaging) or very accurate initial guesses. Given the limited knowledge and literature on optimal solar-sail many-revolution transfers, indirect optimization methods are considered unfit for the scope of this work.

2.2.2. Direct methods

Direct optimization approaches adopt transcription techniques to reformulate the OCP as a unique Non-Linear Programming (NLP) problem, which is then solved numerically [53]. The transcription process (also referred to as direct collocation) consists of interpolating the system dynamics and controls through polynomial functions at different collocation points: coefficients for the defined polynomials become decision variables for the resulting NLP [54]. Thanks to the established NLP solution methodologies, direct optimization methods are widely employed in commercially available software [33], making them a popular choice in studies across different scientific domains. NLP solvers display generally wider convergence basins with respect to MPBVP solvers used in indirect methods, however, solutions typically show sensitivity to the chosen collocation scheme [33]. Most noticeably, the transcription process implies a quadratic growth of NLP problem dimensionality with transfer duration (i.e., the 'curse of dimensionality'), quickly leading the resulting OCP to numerically untractable size [54, 55].

In the context of many-revolutions orbital transfers, the 'curse of dimensionality' typically limits the applicability of direct methods. Different techniques are adopted to mitigate these limitations. Sparsity patterns in the problem Hessian matrix are exploited to reduce the computational intensity of the solution process [56]. Adaptive meshing techniques are also defined, with the goal of minimizing the number of collocation points required to accurately discretize the problem [57].

Thanks to their flexibility and the availability of related state-of-the-art software, direct methods have successfully been applied to several studies on many-revolution low-thrust transfers. Most relevant towards the scope of this work are the studies in Reference [56], where a direct solver using mesh refinement and sparsity exploitation was used for fuel-optimal transfers up to 578 revolutions, and in Reference [58], where the GPOPS-II [57] direct optimization software is used to identify time-optimal Earth-bound transfers (using an IEP engine) up to 1023 revolutions. Many-revolution solar-sail transfers are investigated in Reference [59] using the GPOPS-II software, combined with orbit averaging and control parametrization to reduce problem dimensionality, successfully solving for time-optimal Geostationary Transfer Orbit (GTO) to Geostationary Equatorial Orbit (GEO) transfers up to $\simeq$ 300 revolutions. Direct optimization methods have proven effective for many-revolution transfer OCPs but remain limited for longer transfers, as existing techniques only partially mitigate the "curse of dimensionality".

2.2.3. Heuristic methods

Heuristic methods apply specific policies and strategies to assess and improve the optimality of a candidate solution, without mathematically enforcing optimality conditions [33]. Examples of heuristic methods can be identified in genetic-inspired optimization algorithms, which apply evolution policies inspired by natural laws to 'evolve' candidate solutions towards optimality. A more relevant example is that of the Q-Law [60], a Lyapunov orbit-control approach that captures the proximity to the desired objective, and uses locally optimal control laws to steer the solution towards optimality. Despite their 'blindness' to optimality conditions, heuristic methods see frequent application in low-thrust transfer optimization studies due to their flexibility and/or low computational requirements [35].

While requiring minimal effort in terms of implementation and problem formulation, genetic-inspired algorithms typically imply significant computational effort, as numerous candidate solutions need to be evaluated simultaneously: the large search space spanned by these algorithms makes them promising candidates for use in early design studies (in combination with trajectory parametrization techniques) [35]. In the context of many-revolution transfers optimization, [61] uses a Differential Evolution algorithm to blend locally optimal control laws [6], producing a preliminary mission design for a sail-powered debris removal mission.

The Q-law approach is widely applied in studies regarding the optimization of many-revolution transfers using low-thrust electric engines: in [60, 62] multiple Earth- and Vesta-centered transfers are analyzed, showcasing promising results both in terms of computational and optimality performance. The Mystic software presented in [1] also includes a Q-law approach for the generation of initial guesses or approximate solutions. More relevant to the scope of this thesis is the modified Q-law approach derived in [38], specifically tailored for applications to the many-revolutions Earth-centered solar sail transfer problem, displaying promising results with reduced computational efforts in both Earth orbit and Earth-moon transfers. A combination also including genetic-inspired optimization is observed in [63], where the Lyapunov control gains are optimized for robustness through the particle-swarm optimizer, and the resulting control law is applied to the GEO space debris removal problem using solar sail propulsion. Heuristic methods offer promising perspectives for preliminary studies into the complex nature of many-revolution transfer optimization: the lack of information and guarantees of solution optimality determine the need for more accurate approaches.

2.2.4. Dynamic programming

While direct and indirect optimization methods enforce first-order optimality conditions, the dynamic programming approach is based on Bellman's principle of optimality: ”An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision” [64]. While the stated principle refers to discrete sets of decisions, analogous continuous time theory is provided by the Hamilton-Jacobi-Bellman equations, a system of first-order Partial Differential Equations (PDE) equivalent to Pontryagin's minimum principle.

Thanks to this optimality principle, dynamic programming techniques focus on converting complex problems into a series of simpler sub-problems, which are then solved recursively [64]. The recursive approach allows dynamic programming to explore the full design space (i.e., all possible combinations of control inputs): despite guaranteeing global optimality, the approach leads to the problem dimensionality increasing exponentially with the number of decision variables, incurring in the ”curse of dimensionality” [35]. For this reason, the application of dynamic programming to OCPs is extremely limited.

A different approach, with favorable numerical properties, is identified in the DDP algorithm. The DDP algorithm [43] avoids the 'curse of dimensionality' by only exploring the search space around a certain reference solution: the OCP is discretized in a series of smaller sub-problems (referred to as stages), which are sequentially solved thanks to Bellman's optimality principle [64], iteratively improving the reference solution towards optimality. Since each stage sub-problem is solved individually, the computational requirements of DDP only scale linearly with the number of discretization points [65]. DDP algorithms share similar convergence basin and speed to direct solvers [66], without the related dimensionality issues. Given the sequential approach to the solution of stage sub-problems, DDP algorithms are expected to incur in longer runtimes than other solver classes. DDP algorithms have been successfully applied to several studies on optimal many-revolution transfers (both using IEP and solar sailing as propulsion methods): a thorough overview of relevant literature is provided in Subsection 2.4.3.

2.2.5. Summary and considerations

The summary in Table 2.2 provides valuable insight into the applicability of each numerical optimization method to the solar-sail many-revolution transfer OCP. It is noticed that the computational requirements implied by high-dimensional OCPs are the main limiting factor for various solver classes: orbit-averaging techniques have been introduced to address numerical issues caused by problem dimensionality [49, 59], but they introduce approximations to the dynamical model and are therefore inherently inaccurate. Recent studies in solar-sail trajectory optimization tend to exploit locally optimal laws to maximize the rate of change in specific orbital elements at every instant [28, 50, 51, 52]. While consisting of analytical solutions (thus free from computational limitations), these control laws only satisfy optimality conditions at isolated time instants: application to extended duration problems such as orbital transfers requires numerical optimization techniques to blend these control laws over the full problem duration [28]. In general, applying an optimal control solver to the defined OCP yields more optimal solutions with respect to those provided by locally optimal control laws [39], which are therefore not considered for the scope of this work.

Table 2.2: Main features of the different numerical optimization methods.

Solver class	**Computational requirements**	**Convergence properties**	**Optimality conditions**
Indirect	Linear increase with the number of decision variables	Narrow convergence basin, highly sensitive to co-states guess	First-order optimality
Direct	Quadratic increase with the number of decision variables	Wide convergence basin, quadratic convergence order	First- and second-order optimality
Heuristics (Q-law)	Being implemented through a control law, only numerical propagation is performed	Not applicable	No guarantees on optimality
Heuristics (genetic algorithms)	Depends on the chosen population size, very computationally demanding [33]	Global search space	No guarantees on optimality
Dynamic programming	Exponential increase with the number of decision variables	Global search space	Global optimality
Differential Dynamic Programming	Linear increase with the number of decision variables	Wide convergence basin, quadratic convergence order	First- and second-order optimality

Numerical optimization methods based on an indirect, Q-law, or DDP approach do not suffer from the ”curse of dimensionality”. Indirect methods, however, are highly sensitive to initial guesses in terms of both states and co-states, which are difficult to produce for the rather unexplored many-revolution solar-sail transfer problem. The Q-law approach provides a promising method for the quick generation of near-optimal trajectories but does not enforce optimality conditions, thus not providing the desired guarantees of accuracy and reliability inherent with optimization solvers [33]. The DDP algorithms class is observed to provide the required features for applications to high-dimensional OCPs, and it is therefore illustrated in further detail in Section 2.4.

2.3. Constrained optimization

The optimization of low-thrust transfers implies enforcing constraints of different kinds. The orbital transfer OCP can include both terminal constraints (i.e., applied to the spacecraft's final conditions), and path constraints (i.e., enforced throughout the full OCP duration). Numerical optimization methods are typically defined on unconstrained problem formulations and require integration with constrained optimization techniques for effective applications.

Different methodologies are available to enforce constraints on an OCP. ”Trivial” methodologies such as control projection [67] are not taken into account, as they ”artificially” impose feasibility conditions without integrating them into the optimization process. The methods considered for the scope of this work are limited to the broader class of Image Space Analysis methods [68], which benefit from a noticeable amount of related literature and applications. These methods can be classified into three categories: Lagrange multipliers methods, penalty methods, and augmented Lagrangian methods. For notational clarity. The following subsections provide an overview of each of the mentioned classes. For notational clarity, the following methods are illustrated on a generic problem with cost function $L(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})$, where t is the independent variable, $\boldsymbol{u}$ are control inputs, and $\boldsymbol{w}$ are static parameters, subject to the equality constraint function $\boldsymbol{g}(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})$.

2.3.1. Lagrange multipliers

The Lagrange multipliers approach to constraint optimization consists in the introduction of a new cost functional through an additional set of variables, named Lagrange multipliers $\boldsymbol{\lambda}$). The OCP is then reformulated as the minimization of the new cost functional (referred to as Lagrangian $\mathcal{L}$):

$$\mathcal{L} = L(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) + \boldsymbol{\lambda}^T \boldsymbol{g}(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}). \tag{2.6}$$

Inequality constraints are typically integrated within this framework through the slack-variables approach [44], which allows to convert inequality constraints of the form $\boldsymbol{g}(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) \leq 0$ into equality constraints by adding a slack variable s resulting in:

$$\begin{aligned} \boldsymbol{g}(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) + s &= 0, \\ s &\geq 0. \end{aligned} \tag{2.7}$$

The necessary optimality conditions for such problems require the gradients of constraints and Lagrangian to be parallel at the solution [44]. In a more general way, the optimality conditions for the constrained problem in Eq. 2.6 are expressed through the Karush-Kuhn-Tucker (KKT) conditions [69], consisting in:

1. *Stationarity*: the gradient of the problem Lagrangian $\mathcal{L}$ with respect to the decision variables is null;
2. *Primal feasibility*: the solution satisfies all the constraints (both equality and inequality)
3. *Dual feasibility*: Lagrange multipliers associated to inequality constraints are non-negative
4. *Complementary slackness*: for active inequality constraints, the corresponding multipliers are positive, while they are null for inactive constraints

Constrained optimization methods belonging to this class can be solved through different approaches.

Elimination methods

Elimination methods consist of estimating the set of active constraints (i.e., the components of the constraints function $\boldsymbol{g}$ that violate the feasibility condition $\boldsymbol{g} = 0$) to define a set of dependent decision variables: the problem is therefore reduced to only optimizing the independent decision variables, while the dependent ones are computed from the active constraints.

An example of an elimination method is the clamping approach [42, 70, 71]: after estimating the active set of constraints, the decision variables that violate those constraints are fixed (hence the name ”clamping”), while an unconstrained optimization is performed on the independent variables. The clamping technique is inherently iterative, as the estimation of the active set of constraints can only be trusted if consecutive iterations do not introduce additional active constraints [70].

Range-space methods

Range-space methods also require the estimation of an active set of constraints: the Jacobian of those constraints is used to construct the ”range space”, corresponding to the set of decision variables that affect the constraint violation. The complementary approach is the null-space method, which operates along the null-space of the Jacobian of the active constraints, corresponding to the set of decision variables that have no effect on constraint violations.

These methodologies combine the two sets to identify the decision variables affecting the solution feasibility (i.e., the range space) and those affecting only optimality (i.e., the null space) [42]. Since these methods perform purely algebraic steps, they can benefit from efficient numerical methods and/or approximations to obtain fast convergence [72].

Min-max techniques

An iterative approach to the solution of such systems is the ”min-max” method, consisting of an inner loop where optimal decision variables are computed, and an outer loop where maximal Lagrange multipliers $\boldsymbol{\lambda}$ are defined (i.e., Lagrange multipliers that maximize the Lagrangian $\mathcal{L}$). These methods guarantee robust convergence to an optimal solution [73], but are typically slow, especially in highly nonlinear problems [42], and computationally expensive as they require two nested loops.

2.3.2. Penalty methods

The penalty methods approach constitutes a simple way to integrate constraints into an OCP without major re-formulation effort. The cost functional is augmented with a (scalar and non-negative) penalty function $\mathcal{P}$, resulting in:

$$J = L(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) + \sigma \mathcal{P}(\boldsymbol{g}(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})) \tag{2.8}$$

where σ is the penalty parameter. This formulation allows to tackle the constrained problem as an unconstrained OCP, as the constraint violation is automatically minimized together with the cost functional. The shape of the penalty term $\mathcal{P}$ determines how the constraints interact with the OCP solution: a common choice is a quadratic penalty function, which tends to steer the solution towards feasibility thanks to the smooth gradients. The penalty parameter σ affects the feasibility of the final solution: an infinitely large value for the penalty parameter σ results in an optimal solution that is inherently feasible ($\mathcal{P}(\boldsymbol{g}(t,\boldsymbol{x},\boldsymbol{u},\boldsymbol{w}))=0$). The penalty parameter also impacts convergence speed: higher σ values lead the optimization algorithm to prioritize feasibility over costs, but also introduce the risk of ill-conditioning [44].

Interior-point methods

A particular set of penalty methods is the class of interior-point methods. Interior-point methods augment the cost functional to enforce inequality constraints through a barrier function, a specific type of penalty term P which assumes high values only when the solution is close to the constraint boundary. The typical choice is the logarithmic function $\mathcal{P}=-\log(-\boldsymbol{g}(t,\boldsymbol{x},\boldsymbol{u},\boldsymbol{w}))$, which also gives the name to this class of methods (since the logarithm of constraint violations is undefined for unfeasible solutions, where $-\boldsymbol{g}(t,\boldsymbol{x},\boldsymbol{u},\boldsymbol{w})<0$, the iterates are restricted to only points inside the feasible region). In interior-point methods, the penalty parameter σ is typically decreased iteratively to allow the solution to get closer to the constraint boundary. While these methods allow to easily and efficiently tackle highly-constrained problems without the need to estimate the active set of constraints, the delicate interaction between σ and the high values of barrier functions makes these methods likely to encounter ill-conditioning and/or slow convergence [44].

2.3.3. Augmented Lagrangian

The augmented Lagrangian approach combines the two classes of methods explained previously. The method augments the cost functional (referred to as augmented Lagrangian) by adding both the Lagrange multipliers and a penalty term, resulting in:

$$J=L(t,\boldsymbol{x},\boldsymbol{u},\boldsymbol{w})+\boldsymbol{\lambda}^T\boldsymbol{g}(t,\boldsymbol{x},\boldsymbol{u},\boldsymbol{w})+\sigma\mathcal{P}(\boldsymbol{g}(t,\boldsymbol{x},\boldsymbol{u},\boldsymbol{w})) \tag{2.9}$$

The techniques to tackle this formulation correspond to those introduced for the Lagrange multipliers approach introduced in subsection 2.3.1. The augmented Lagrangian approach benefits from the favorable convergence properties of penalty functions but addresses the ill-conditioning risks by combining it with Lagrange multipliers, ensuring convergence even under reduced σ values. A relevant feature of the augmented Lagrangian technique is the ability to solve the ”min-max” technique only approximately, by updating decision variables and Lagrange multipliers in a single loop, without the loss of convergence properties [74].

2.3.4. Approaches comparison

Having presented the available classes of constrained optimization approaches, their applicability to the enforcement of both path and terminal constraints is investigated. Recent works have introduced the practice of separately handling path and terminal constraints in low-thrust-related OCPs [42, 71], as the two classes of constraints present different characteristics and requirements:

- *Path constraints*: these constraints are applied to each discretization point of the trajectory. When introduced in a sequential solver such as DDP, techniques used to handle these constraints shall prioritize computational efficiency. Additionally, it is noticed that path constraints typically represent physical limitations of the dynamical system that cannot be violated, thus requiring a reliable solution method (i.e., a method that is unlikely to yield violations of such constraints due to instabilities).
- *Terminal constraints*: terminal constraints are only applied to the final point of a trajectory and typically represent the target of the specified OCP. Since an error margin is allowed, the enforcement of terminal constraints can be achieved through approximate techniques. It is to be noticed that terminal constraints are affected by all the OCP decision variables, thus implying significant limitations to the applicability of algebraic methods (e.g., elimination and range-space methods).

The mentioned classes of constrained optimization methods are analyzed in Table 2.3 for applications to both path and terminal constraints. The Lagrange multipliers methods included in Table 2.3 mainly refer to non-iterative techniques (such as range-space methods) as they provide the desired accuracy without major computational overhead. Terminal constraints can be robustly enforced through the augmented Lagrangian method, exploiting the approximate ”min-max” technique [74] to efficiently obtain accurate results.

Table 2.3: Trade-off table of the considered classes of constrained optimization methods

	Lagrange multipliers	**Penalty methods**	**Augmented Lagrangian**
Path constraints	Immediate convergence through KKT conditions, need first/second-order information and active-set estimation, no ill-conditioning	Fast convergence but require iterations, prone to ill-conditioning	Fast convergence but require iterations, numerically stable
Terminal constraints	Slow convergence on high-dimensional OCPs (min-max approach), no ill-conditioning	Fast convergence regardless of dimensionality, prone to ill-conditioning	Fast convergence regardless of dimensionality, robust to ill-conditioning

2.4. Differential Dynamic Programming

The DDP approach is identified as a promising methodology to tackle the high-dimensional OCPs resulting from many-revolution solar-sail transfers, thanks to its linear scaling in computational requirements over problem dimensionality, the wide convergence basin, and optimality guarantees. This section provides a more detailed illustration of the DDP approach to optimal control, followed by an overview of relevant theoretical studies that expand on the DDP framework, and finally a list of relevant applications of DDP algorithms to many-revolutions low-thrust OCPs.

2.4.1. Algorithm formulation

The main DDP steps are now introduced. First, the algorithm is initialized by providing an initial guess, consisting of a (discrete) time history of control inputs and static parameters to forward-propagate the trajectory, obtaining an initial reference solution. The algorithm initialization requires the definition of parameter values for the different solvers used to tackle the optimization sub-problems.

The core step of DDP is the backward induction, illustrated in Figure 2.2: the $\bar{\square}$ symbol indicates quantities belonging to the reference trajectory, the subscript k represents a generic discrete-time stage (with N being the number of stages), while the $\delta\square$ symbol refers to small variations used to evaluate feedback laws. Following the classic OCP notation, the state vector is represented as $\boldsymbol{x}$, while control inputs are indicated using $\boldsymbol{u}$. According to Bellman’s optimality principle, optimal control decisions taken at a certain stage shall be optimal regardless of decisions taken at previous steps. During the backward induction, this property is exploited by computing an optimal feedback control law at each stage: since each stage sub-problem is independent of previous stages, the backward induction starts from the final stage by computing the optimal control policy in the form of a feedback law.

The methods used to compute the optimal feedback law rely on a quadratic model of the cost function around the current reference trajectory: each stage results in a quadratic optimization problem, solved using a trust-region procedure to enforce problem convexity [75]. Partial derivatives computed on the optimized stage sub-problem are mapped backward to the previous stage, constituting the induction step: the ”regular” DDP algorithm performs this step by numerically propagating (backward) the cost partial derivatives following an Euler scheme [43]. Each optimization stage coincides with the collocation points used for numerical propagation, ensuring a smooth approximation of the optimal control law. Since all optimized upstream stages are solved through a control feedback law which is stored as part of the current solution, each stage sub-problem only requires the identification of the related optimal control policy: upstream control policies are stored and included in the quadratic cost model through the induction step, while Bellman’s optimality guarantees that previous stages do not affect the current solution. Once all the stage sub-problems are solved, an expected reduction in the objective function is also retrieved.

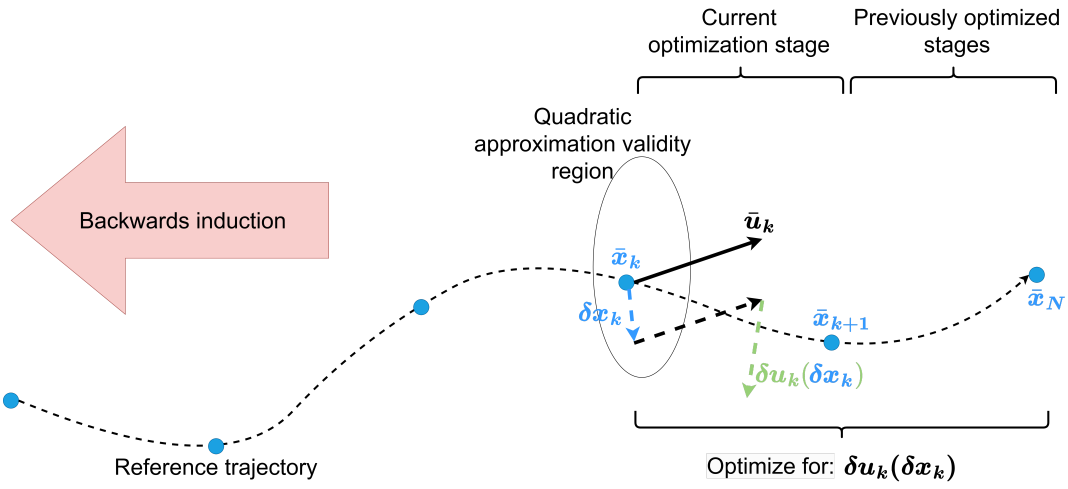

Current optimization stage
Previously optimized stages
Quadratic approximation validity region
Backwards induction
$\bar{u}_k$
$\bar{x}_k$
δx_k
$\bar{x}_{k+1}$
$\bar{x}_N$
$\delta u_k(\delta x_k)$
Reference trajectory
Optimize for: $\delta u_k(\delta x_k)$

Figure 2.2: Illustration of the backwards induction in the DDP algorithm.

The second-order cost model used to solve each stage sub-problem allows the algorithm to enforce both first- and second-order optimality conditions [42]. First-order optimality is tested through the expected cost reduction computed from the last successful backward induction (i.e., the expected reduction is below a certain threshold). Second-order optimality is checked on the positive definiteness of the Hessian matrices computed for each stage sub-problem. Since the backward induction produces optimal feedback laws for each stage sub-problem, DDP-based frameworks offer a promising foundation for the implementation of robust optimization strategies.

If convergence is not detected, the following step is the forward pass, illustrated in Figure 2.3. The forward pass procedure consists of propagating the trajectory forward in time, progressively updating the control inputs according to the feedback laws computed during the backward induction. The forward pass procedure outputs a so-called ”trial iterate”: acceptance of this trial iterate is determined by the proximity of the expected and actual cost reductions. Since the validity of the backward induction quadratic model cannot be guaranteed a-priori, DDP algorithms follow a trust-region-like procedure: the solution of each stage sub-problem is restricted to a certain region around the reference control profile (i.e., the trust-region), which is restricted or enlarged according to the accuracy of the ”trial-iterate” [75].

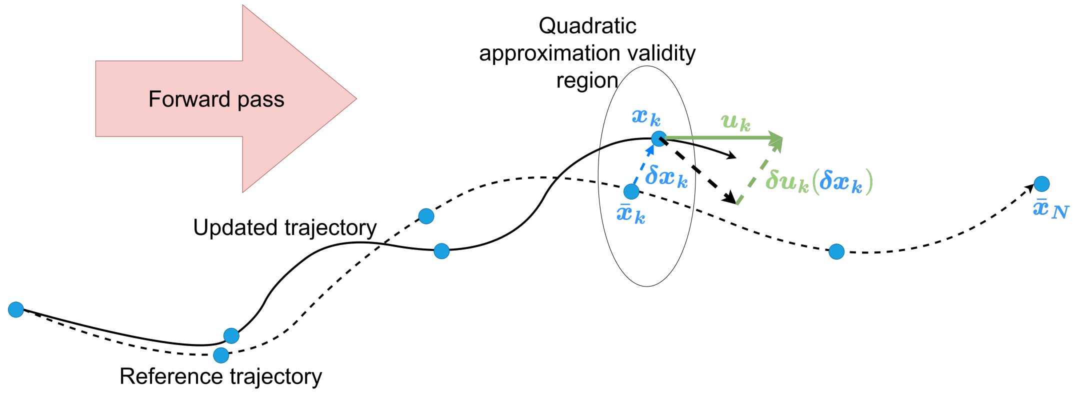


Figure 2.3: Illustration of the forward pass in the DDP algorithm.

There are, however, limitations implicit with DDP. Revolving around a quadratic cost model, DDP requires first- and second-order partial derivatives of the full problem formulation (i.e., dynamical models and objectives). Defining a problem using two-times differentiable functions can be a limiting factor, especially when considering dynamical models implemented through tabulated data (hence nondifferentiable), such as atmospheric density [51]. The definition of new partial derivatives for every change to problem formulation greatly reduces the flexibility of the approach [33]. The multiple subtasks solved as part of the DDP algorithm (backward induction, forward pass, convergence test, etc.) require the definition of many algorithm parameters. Tuning and sensitivity to these hyper-parameters has been identified as a critical limitation in several works [36, 39, 42]. Finally, in their most basic formulation, DDP methods are limited to unconstrained and fixed-duration problems.

2.4.2. Theory

Having presented the working principles of DDP algorithms, the focus is now shifted towards relevant studies that expand on the DDP framework for the solution of OCPs. The rationale behind each of the presented studies is first introduced, followed by an overview of the relevant expansions to the general DDP framework presented in Subsection 2.4.1. A summary of the main characteristics of each relevant source introduced in this subsection is presented in Table 2.4, with a color scheme indicating the applicability of each of the mentioned features to the many-revolution solar-sail transfer OCP class. The rationale behind the color schemes is explained in deeper detail in Section 2.5.

Augmented Lagrangian Differential Dynamic Programming

The DDP approach, in its most basic form, is limited to unconstrained OCPs. The limitation is addressed in Reference [76] by introducing an augmented Lagrangian technique to enforce terminal constraints (i.e., constraints applied on the final discretization stage). The devised DDP algorithm implements the approximate ”min-max” procedure mentioned in subsection 2.3.3, ensuring robust convergence [74] with minor computational overhead thanks to the single loop used by the approximate ”min-max” technique.

High-fidelity discretization in Differential Dynamic Programming

The discrete problem formulation used in DDP typically requires a fixed problem discretization, which can introduce significant approximation errors or unnecessary computational overhead: Reference [77] introduces a variable-step discretization into the DDP logic, enabling higher fidelity during the backward induction and forward pass procedures. The devised approach performs an estimation of the integration error implied by the current discretization and consequently re-arranges the optimization stages to meet a defined tolerance. Terminal constraints are enforced through the ”min-max” (pure) Lagrangian approach presented in subsection 2.3.1, thus requiring two nested loops [77].

Control-limited Differential Dynamic Programming

The DDP methodology is expanded to also consider path constraints (i.e., constraints applied throughout the full problem duration) in Reference [70]. This expansion extends the applicability of DDP to a considerably broader and realistic class of OCPs, despite being limited to path constraints in the form of linear control bounds [70]. The devised technique to enforce path constraints is a clamped line-search approach: the clamping logic presented in subsection 2.3.1 is combined with a line-search to optimize the independent control variables at every iteration.

Tube Stochastic Differential Dynamic Programming

The DDP algorithm solves each stage sub-problem in the form of a control feedback law, inherently providing robustness against uncertainties [43]. Reference [78] introduces the Tube Stochastic DDP algorithm, where the robust optimization capabilities of DDP are fully exploited by introducing an unscented transform [79] to deterministically propagate uncertainties. The study adopts a chance-constraints approach (i.e., only satisfying constraints from a probabilistic standpoint) to solve uncertain and constrained OCPs using DDP [78].

Quadratic Differential Dynamic Programming

In Reference [72], a constrained optimization methodology based on the KKT conditions is introduced. The approach relies on a second-order approximation of the constraints (both terminal and path constraints), which is combined with the second-order cost model used by the DDP algorithm to efficiently enforce KKT optimality. The technique first estimates the set of active constraints by solving the unconstrained problem, then performs a second-order expansion of the KKT conditions around the current reference solution: the resulting system is a set of linear equations that yield a second-order feedback law, with improved numerical stability over the linear feedback terms in ”regular” DDP implementations [72].

Static/Dynamic Control algorithm

The state-of-the-art in DDP algorithms is represented by the Static Dynamic Control (SDC) algorithm [1]. While SDC itself implies several key differences, it adopts the same core concepts of Bellman optimality explained in Subsection 2.4.1. The SDC expands on the DDP framework by allowing the solution a generic OCP, including both dynamic controls and static parameters (i.e., constant values throughout a trajectory arc), under arbitrary Equations of Motion (EOM) and constraints, enforced through an augmented Lagrangian approach. Additionally, the SDC formulation accommodates multi-phase OCPs, that is OCPs where the trajectory is split into multiple legs characterized by different dynamics and/or objectives, enhancing its applicability to mission-design scenarios [1]. A Hessian-shift technique is used to enforce positive definiteness at every stage sub-problem [75], ensuring solution optimality [42]. The SDC algorithm performs the induction step through a high-fidelity propagation of the quadratic cost model: while the technique implies considerable computational overhead, its high-fidelity capabilities prove advantageous in realistic applications. The technique also enables time-dilation techniques to handle OCPs with variable problem durations [1].

Hybrid Differential Dynamic Programming

A major expansion on the DDP framework is introduced in Reference [42] through the HDDP algorithm. The HDDP algorithm combines the DDP approach with the logic of direct transcription methods: the fine problem discretization required by DDP is substituted by a coarser mesh of discretization stages, reducing the computational requirements to solve an OCP. The HDDP induction step is performed through an State Transition Map (STM) approach, which enables the accurate mapping of partial derivatives between consecutive stages: the stage-wise STMs are precomputed on the reference solution and applied during the backward induction, further improving the HDDP computational efficiency. The HDDP framework enables the optimization of both dynamic controls and static parameters applied to multi-phase and constrained OCPs. An augmented Lagrangian approach is used to enforce terminal constraints, while generic path constraints are implemented through a range-space method applied to a first-order expansion of each stage sub-problem [42]. The study also recognizes the algorithm's sensitivity to its hyper-parameter and addresses the limitation by introducing a simple adaptive procedure to update the penalty parameter [42].

Following up on the HDDP framework, Reference [80] presents minor improvements and revisions for a more robust framework. The study introduces safeguards to the backward induction process to increase its robustness. The range-space method introduced in Reference [42] is shown to be inefficient (i.e., does not fully exploit the available trust region) and to cause chattering (i.e., the first-order approximation can cause the updated controls to violate the constraints when applying the optimal feedback laws during the forward pass), therefore different methodologies are presented. An alternative is identified in the combination of a clamping logic (limited to one iteration) with the proposed range-space method, registering an increase in accuracy at the cost of computational overhead. An interior-point method to scale the trust region [81] is also presented, consisting of the use of barrier functions to enforce path constraints in the forms of linear control bounds [80].

Multiple-shooting Differential Dynamic Programming

Reference [71] integrates the HDDP algorithm within a multiple-shooting framework. The multiple-shooting technique consists of splitting the full OCP duration into shorter sub-intervals and solving the resulting multi-phase problem, enforcing continuity constraints between the sub-intervals. The multiple-shooting DDP adopts an NLP solver to optimize the initial conditions of each shooting phase, while the dynamic controls over each leg are obtained through HDDP [71]. The HDDP algorithm adopted in Reference [71] follows the original HDDP formulation provided in Reference [42, 80]. with a modified approach to path constraints: the clamped range-space method from Reference [80] is extended to a fully iterative procedure, fully exploiting the clamping technique introduced in Reference [70].

Higher-order Hybrid Differential Dynamic Programming

The second-order cost approximation used in DDP is expanded to higher orders in Reference [82]. A differential algebra framework is introduced within the HDDP framework, allowing the efficient computation of the STMs to higher orders for a more accurate propagation of the cost function model across successive states.

Flexible Final Time Differential Dynamic Programming

The expansions on the DDP framework presented until this point are inherently limited to fixed problem durations. Reference [67] addresses this limitation by incorporating the OCP duration as a decision variable, treating it as a dynamic control input applied to the last stage. Optimal adjustments to the time of flight are implemented by appending or removing discretization stages to the end of the trajectory: being based on a linear truncation of the dynamics, the technique is not expected to perform accurately in the context of long-duration orbital transfers. The Flexible Final Time DDP adopts an augmented Lagrangian approach to enforce constraints, while control projection is used to address control bounds [67].

Table 2.4: Main features of relevant expansions to the DDP algorithmic framework

Ref.	Runtime effort	Terminal constraints	Path constraints	Multi-phase	Flexible final time
[76]	High	Augmented Lagrangian (single loop)	No	No	No
[77]	High	Lagrangian (nested loops)	No	No	No
[70]	High	Augmented Lagrangian (single loop)	Clamped line-search (fully iterative, only control bounds)	No	No
[78]	High	Penalty	Chance constraints (probabilistic)	No	No
[72]	High	Quadratic KKT model	Quadratic KKT model	No	No
[1]	High	Penalty	Penalty	Yes	Time dilation
[42]	Low	Augmented Lagrangian (single loop)	Range space	Yes	No
[80]	Low	Augmented Lagrangian (single loop)	Clamped range space (two iterations)	Yes	No
[71]	Medium	Augmented Lagrangian (single loop)	Clamped range space (fully iterative)	Yes	No
[82]	Medium	Augmented Lagrangian (single loop)	Clamped range space (two iterations)	Yes	No
[67]	High	Augmented Lagrangian (single loop)	Control projection	Yes	Append stages

2.4.3. Applications

Having presented relevant expansions to the DDP framework, the focus is now shifted towards applications to relevant OCPs. The following studies are grouped by the related dynamical system used to define the corresponding OCP (i.e., EP-powered spacecraft, solar-sail-powered spacecraft, and different systems). An overview of the objectives, adopted methodologies, and main findings is provided for each study. The main features are grouped in Table 2.5, with a color scheme indicating the applicability of each of the mentioned features to the many-revolution solar-sail transfer OCP class. The rationale behind the color schemes follows the one in Table 2.3 regarding constrained optimization methodologies, while other features (the multi-phase and flexible-final-time capabilities) are evaluated based on the flexibility that they provide for the formulation of an OCP.

Electric-propulsion powered spacecraft

The high-fidelity discretization in DDP is used to identify fuel-optimal low-thrust transfers from Earth to rendezvous with a NEA in a fixed transfer time: the variable-step discretization is exploited to accurately handle both the Earth-orbit and NEA arrival phases in a single optimization procedure [77].

The Tube Stochastic DDP is applied to a fuel-optimal IEP-powered Earth-Mars transfer: Monte Carlo analyses show its robustness against uncertainties as well as the improved performance with respect to traditional duty-cycle methods [78]. The robust control policy is shown to outperform the ”deterministic” DDP approach on a long-duration transfer: under reduced engine performance, the robust policy is shown to successfully satisfy the OCP objectives and constraints, while a deterministic DDP approach to the solution fails to converge [78].

The SDC algorithm from Reference [1] is implemented in the Mystic software for the optimization of EP-powered spacecraft [83], enhanced by a Q-law approach for the generation of initial guesses [62]. Given the computational costs of the SDC procedure, the Q-law approach is also used to generate a near-optimal solution in case runtime limits are exceeded [83]. The Mystic software was successfully applied to the DAWN mission design, successfully identifying fuel-optimal transfers around asteroid Vesta using a high-fidelity dynamical model and complex non-linear constraints [84].

The HDDP algorithm is initially applied to a series of test cases in Reference [80]. The algorithm is applied to an Earth-Mars transfer mission using a low-thrust IEP engine, and results are successfully validated against commercially available direct and indirect optimization tools [80]. More relevant to the scope of this work, a test case on an Earth-centered multi-revolutions transfer (using the same low-thrust engine) is illustrated: the HDDP algorithm shows convergence when starting from trivial initial guesses, identifying a fuel-optimal 17-revolution transfer while showing better runtime performance than off-the-shelf sparse direct optimization solvers [80]. Finally, the multi-phase capabilities of the algorithm are tested on the Global Trajectory Optimisation Competition (GTOC) 4 problem (multi-asteroid rendezvous mission), showcasing comparable performance (in terms of both accuracy and runtime) with other solution approaches adopted during GTOC [80].

The HDDP algorithm is applied to both time- and fuel-optimal GTO to GEO transfers in Reference [36], successfully identifying up to 1500-revolution solutions. Comparison of HDDP performance against direct, indirect, and Q-law approaches highlights the algorithm's improved computational performance [36].

The HDDP framework is also applied to simplified/approximated dynamical models. Fuel-optimal rendezvous maneuvers within the Circular Restricted Three Body Problem (CRTBP) dynamical setting are investigated in Reference [85]: thanks to the simplified dynamical model (available thanks to the CRTBP assumption), the HDDP approach is applied to problems with variable duration by parameterizing the targeted state as well as the problem duration, improving the effectiveness in optimizing rendezvous maneuvers [85]. Similar features are implemented in Reference [86] and [87], respectively using a Sims-Flanagan approximation (i.e., trajectory discretization where perturbing accelerations are modeled as impulsive shots and sailcraft dynamics are propagated according to the Kepler model [88]) and the Stark model (i.e., trajectory discretization where perturbing forces are embedded in a uniform force field and the dynamics are propagated through a semi-analytical model [89]). The two simplified models are applied to both interplanetary rendezvous and multi-revolution Earth-centered applications, greatly improving the runtime performance of HDDP at the cost of minor accuracy decrease [87].

The multiple-shooting DDP algorithm is applied to multi-revolution fuel-optimal EP transfers around Earth in Reference [90]. The devised algorithm is used on a fuel-optimal 40-revolution transfer: while retractable using a single-shooting approach (i.e., solving the full transfer as a single trajectory), dividing into up to 20 shooting phases reduces the OCP sensitivity, significantly improving the algorithm convergence properties [90].

The higher-order HDDP from Reference [82] is successfully applied to a fuel-optimal Earth-Mars interplanetary transfer. Despite the improved accuracy granted by the higher-order STMs, the approach is shown to introduce no considerable improvements over the regular HDDP methodology: the increased runtime required to obtain higher-order STM does not outweigh the convergence benefits gained from more accurate dynamical approximations, while the higher-order control feedback laws are prone to instabilities [82].

Solar-sail powered spacecraft

The state-of-the-art Mystic software [83] is also applied to the NEA Scout mission design [91]. The software is modified to accommodate high-fidelity modeling and optimization of solar-sail transfers: the resulting Mystic version is integrated within an automated procedure to quickly generate cis-lunar transfer trajectories to NEAs with minimal cone-angles (as the feature yields robust trajectories), streamlining the mission design process and accommodating the considerable variability in mission launch date [91].

Reference [39] applies a simplified version of the HDDP approach to the solar-sail orbit raising problem. The chosen implementation uses a penalty method to enforce terminal constraints, a logistic function is used to enforce control bounds, and a fixed Hessian-shift approach is used to solve stage subproblems [39]. The devised algorithm is combined with Monotonic Basin Hopping (MBH) to identify globally optimal control laws for orbit raising around Earth, successfully optimizing up to 500-revolution raising maneuvers and highlighting the algorithm's sensitivity to its hyper-parameters [39]. The same HDDP algorithm is also applied to interplanetary Earth-Mars transfers in Reference [40].

Additional applications

Many of the presented works on DDP [70, 72, 80, 90] include a verification case based on linear-quadratic OCPs (i.e., problems with linear dynamics and quadratic cost function). The verification is performed by checking for convergence in a single iteration, as augmented Lagrangian methods (which are used in the considered studies) are known to possess such property when applied to linear-quadratic problems [92].

The control-limited DDP from Reference [70] is applied to different applications in robotics, ranging from the car-parking problem to the full control of a humanoid robot. The clamping technique introduces considerable overhead due to its iterative nature, but it is shown to outperform simple constrained optimization techniques, namely control projection (i.e., setting control variables on their boundary when they exceed the constraints) or logistic functions (i.e., artificial restrictions of the achievable control values), which are known to be inefficient and can cause ill-conditioning [70].

The Flexible-Final-Time DDP introduced in Reference [67] is successfully applied to a fuel-optimal Unmanned Aerial Vehicle (UAV) guidance problem. The algorithm displays improved convergence properties over a direct optimization approach [57] based on time-dilation.

2.5. Research objective and research questions

Previous sections provided an overview of relevant studies, concepts, and approaches in the context of planetocentric solar sailing, focusing on the numerical solution of many-revolutions transfers OCPs. This section summarizes relevant information leading to the identified research gap, presenting the formulation of the research objective and the consequent research questions.

Solar sailing provides a very appealing propulsion method for various applications thanks to its propellant-free nature. As shown in subsections 2.1.1 and 2.1.2, solar sailing in the Earth-orbit environment provides promising opportunities for planetary imaging, telecommunications, targeted measurements, and debris removal. The constrained nature of solar-sail thrust implies reduced orbit control authority: in the planet-centered environment, this results in many revolutions being needed to accomplish a single orbital transfer [24].

Optimization of orbital transfers consists of identifying the control inputs applied to the spacecraft for the trajectory duration while minimizing a certain performance metric and satisfying specified constraints. The problem is therefore formulated as an OCP. Solar-sail transfers around a planet result in high-dimensional and constrained OCPs [38]. Direct optimization methods scale poorly with problem dimensionality, limiting their applicability to the many-revolutions transfer problem [43]. Indirect solvers require accurate initial guesses (on both states and co-states) to achieve reliable convergence: limited knowledge on the nature of many-revolution solar-sail transfers implies the need for a more robust

Table 2.5: Main features of relevant applications of DDP algorithms

Ref.	Propulsion method	Problem	Objective	Additional notes
[77]	EP	Interplanetary transfers	Fuel-optimal	
[78]	EP	Interplanetary transfer	Fuel-optimal	
[84]	EP	Vesta-centered transfers	Fuel-optimal	
[80]	EP	Interplanetary transfer	Fuel-optimal	
[80]	EP	Earth-centered transfer	Fuel-optimal	17-revolution transfer
[36]	EP	Earth-centered transfers	Fuel- and time-optimal	Up to 1500-revolution transfers
[85]	EP	Interplanetary transfers	Fuel-optimal	CRTBP
[87]	EP	Earth-centered transfers	Fuel-optimal	Stark model
[90]	EP	Planet-centered transfers	Fuel-optimal	40-revolution transfer
[82]	EP	Interplanetary transfer	Fuel-optimal	
[91]	Solar sail	Interplanetary transfer	Minimum cone angle	
[39]	Solar sail	Earth-centered orbit-raising	Time-optimal	Up to 500-revolution orbit raising

approach [44]. Heuristics-based approaches, while feasible, do not enforce optimality conditions [60], therefore DDP is identified as the final candidate.

Constrained optimization methods are required to enforce constrained on the defined OCP. The features of the three main approaches (Lagrange multiplier methods, penalty methods, and augmented Lagrangian methods) are summarized in Table 2.3, considering the different features and requirements implied by path and terminal constraints.

Having identified DDP as a promising algorithm for the solution of many-revolutions transfers OCPs, the algorithm formulation itself is introduced in Section 2.4.1. Its main limitations in terms of flexibility (i.e., fixed-time and unconstrained formulations) and parameter tuning are presented. Relevant literature on DDP is presented in Section 2.4: expansions to the algorithmic framework and their main features are presented in Table 2.4, while relevant applications are illustrated in Table 2.4.3.

The optimization of solar-sail many-revolution transfers requires an efficient optimization algorithm, capable of handling problem formulations which include:

- *Path constraints*: these can represent the physical limitation implicit with the solar-sail thrust (e.g., the sail cannot produce sunward thrust), artificial constraints dictated by the chosen dynamical model (e.g., unit-norm constraints if the control inputs are represented using a unit vector), or operational constraints (e.g., a minimum operational altitude [51] and attitude-control limitations [93]).
- *Terminal constraints*: these can represent any target in a specified OCP, ranging from achieving a certain orbital regime to rendezvous constraints [85].
- *Multi-phase capabilities*: these can enable the use of the devised algorithm for missions with multiple phases, such as debris removal (i.e., a first phase to rendezvous with the debris object, followed by a disposal phase). The multi-phase capabilities of the algorithm also enable its integration within a multiple-shooting framework [71].

- *Flexible-final-time capabilities*: given the propellant-less nature of solar sails, investigating minimum-time solutions to specified problems is typically the primary aim of the optimization process, thus requiring a methodology to tackle variable-duration OCPs.

The DDP-related works presented in Table 2.4 and Table 2.5 fail to successfully address the previously mentioned points. Additionally, the analyzed literature depicts the following trends:

- Most studies refer to EP, with little interest towards solar sailing applications;
- Given the interest on EP-powered applications, most works focus on fuel-optimal transfers;
- While methodologies regarding the implementation of terminal constraints is well established, few studies focus on enforcing generic path constraints;
- Minor efforts are present to expand DDP and/or HDDP to variable-duration problems.

Noticeably, while some of the presented works investigate the effects of specific algorithm hyper-parameters on optimized solutions [36, 39], no effort is dedicated to the implementation of adaptive procedures to speed up the tuning process.

Given the restricted knowledge on optimal planet-centered many-revolution transfers using solar sails [37], a flexible, robust, and efficient optimization framework is desired: the HDDP algorithm is identified as the most promising baseline implementation. To achieve such a framework, the defined solver shall robustly tackle a wide range of OCPs: multi-phase, path- and terminal-constrained, variable-duration formulations shall be accommodated. Expansions to existing methodologies related to the handling of path constraints and variable-duration problems are therefore required. Given the propellant-less nature of solar sails, time-optimal solutions to the orbital transfer problem are of great interest. To properly investigate the nature of optimal solar-sail many-revolution transfers, such an optimization framework should also enable quick iterations on problem formulation.

2.5.1. Research objective

> Realize an optimization framework based on HDDP that is capable of handling high-dimensional, path-constrained, and variable-duration optimal control problems, and apply the optimization algorithm to investigate time-optimal many-revolution solar-sail transfers around Earth.

2.5.2. Research questions

The presented research objective leads to the following research questions and sub-questions:

RQ1 - How can a robust and flexible optimization framework, aimed at optimizing many-revolutions solar sail transfers, be defined using HDDP?

- **RQ1.1** - How can time of flight be included as a decision variable within the HDDP optimization algorithm?
- **RQ1.2** - How can path constraints be enforced reliably and efficiently within the HDDP optimization framework?
- **RQ1.3** - How can HDDP sensitivity to its hyper-parameters be reduced?

RQ2 - How does the defined optimization algorithm perform, when applied to the identification of time-optimal many-revolution solar-sail transfers?

- **RQ2.1** - What are the effects of different orbital regimes on optimal many-revolution circular-to-circular transfers?
- **RQ2.2** - How do variable-time solutions compare to fixed-time circular-to-circular transfers?

3

Methodology

This chapter presents the methodology chosen and developed to achieve the defined research objective. First, the mathematical notation and general formulation of an OCP are introduced in Section 3.1. Then, the chosen HDDP formulation is thoroughly explained in Section 3.2, individually illustrating the different building blocks of the algorithm, including the original methods developed to handle path-constrained, variable-duration problems, and to perform adaptive parameter tuning. The resulting software implementation is illustrated in Section 3.3.

3.1. Problem formulation

As in most numerical optimization approaches, a discrete-time problem formulation is introduced, where each finite point is referred to as a *stage*. Additionally, in its most generic form, an OCP also encompasses cases including different definitions of the system dynamics: each of these periods is referred to as a *phase*. Throughout this work, M and i represent, respectively, the total number and a generic phase. Analogously, the total number of stages in phase i is indicated as $N_i + 1$, while k stands for a generic stage. For the remainder of this section, quantities referred to a generic phase and stage are indicated through subscripts i, k. For notational clarity, when considering quantities at the interface between two phases i and $i + 1$, the subscripts $-$ and $+$ are used to represent, respectively, the final quantities of phase i and the initial quantities of phase $i+1$. For the extremal phases $i = 1, M$, the same inter-phase formulation using subscripts $-$ and $+$ is adopted without loss of generality, only assuming "artificial" quantities (i.e., related to fictitious phases $i = 0, M + 1$) to be dummy variables, with no effect on the problem formulation itself. Relations and quantities referred to all the stages in a phase (e.g., the dynamics equation or the static parameters vector) directly omit the k subscript.

The definition of an OCP requires the identification of a dynamical system: $\boldsymbol{x}$ represents the state vector, $\boldsymbol{u}$ the control inputs, and $\boldsymbol{w}$ the vector of static parameters. The quantities n_x, n_u, and n_w are used to indicate the sizes of, respectively, the state, controls, and static parameters vectors. Newton derivative notation is adopted, such that derivatives with respect to the independent variable t are represented by the overhead dot operator ($\dot{\square} = \frac{d\square}{dt}$). The dynamics function $\dot{\boldsymbol{x}} = \boldsymbol{f}_i(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})$ expresses the systems evolution according to an independent variable t, typically representing time. The initial conditions are defined through the parametrization $\boldsymbol{x}_{i,1} = \Gamma_i(\boldsymbol{w}_i)$. While the state vector is implicitly limited to physically feasible values thanks to the dynamics function, controls and parameters are limited, respectively, to subsets $\mathcal{U}$ (i.e., admissible controls) and $\mathcal{W}$ (i.e., admissible static parameters).

Typical OCPs also include constraints, both on the final state of the system as well as throughout its trajectory. The former are referred to as terminal constraints, and can be imposed at the end of every phase: to allow the definition of continuity conditions between consecutive phases, terminal constraints are represented by the vector function $\boldsymbol{\Psi}_i(\boldsymbol{x}_{i,N_i+1}, \boldsymbol{w}_i, \boldsymbol{x}_{i+1,1}, \boldsymbol{w}_{i+1}) = \boldsymbol{\Psi}_i(\boldsymbol{x}_-, \boldsymbol{w}_-, \boldsymbol{x}_+, \boldsymbol{w}_+)$. The latter are called path constraints and indicated by the vector function $\boldsymbol{g}_i(t_{i,k}, \boldsymbol{x}_{i,k}, \boldsymbol{u}_{i,k}, \boldsymbol{w}_i)$: being enforced on all points in a certain phase, path constraints typically represent physical limitations of the system which are not implicit within the dynamical formulation, or artificial bounds posed to the specific problem (e.g., operational altitude constraints in the case of solar sails [51]). Both classes can be defined as equality or inequality constraints. In this Section, only equality constraints are explicitly introduced: Section 2.3 presents methods to also handle inequality constraints, therefore the notation used in this Section does not result in a loss of generality. The dimensions of the constraints functions are, respectively, n_{Ψ_i} and n_{g_i}.

Finally, the definition of an OCP requires a (scalar) cost functional J to be minimized. The cost functional can include different terms: the running costs $L_i(t_{i,k}, \boldsymbol{x}_{i,k}, \boldsymbol{u}_{i,k}, \boldsymbol{w}_i)$ applied to every point in the trajectory, and the terminal cost $\varphi_i(\boldsymbol{x}_-, \boldsymbol{w}_-, \boldsymbol{x}_+, \boldsymbol{w}_+)$ defined for inter-phase conditions. The cost functional can always be represented using any combination of the two terms [44]: for the scope of this study, the general 'Bolza' form is adopted, where both terms are added together.

The full mathematical formulation of the OCP then becomes:

$$\begin{aligned} \text{find } \boldsymbol{u}_{i,k}, \boldsymbol{w}_i &= \text{argmin } J,\ \forall i = 1, ..., M,\ \forall k = 1, ..., N_i + 1 \\ J &= \sum_{i=1}^{M} \left[\varphi_i(\boldsymbol{x}_-, \boldsymbol{w}_-, \boldsymbol{x}_+, \boldsymbol{w}_+) + \sum_{k=1}^{N_i+1} L_i(t_{i,k}, \boldsymbol{x}_{i,k}, \boldsymbol{u}_{i,k}, \boldsymbol{w}_i) \right] \end{aligned} \tag{3.1}$$

subject to the conditions:

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) \\ \boldsymbol{u} &\in \mathcal{U} \\ \boldsymbol{w} &\in \mathcal{W} \\ \boldsymbol{g}_i(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) &= 0,\ \forall i = 1, ..., M \\ \boldsymbol{\Psi}_i(\boldsymbol{x}_-, \boldsymbol{w}_-, \boldsymbol{x}_+, \boldsymbol{w}_+) &= 0,\ \forall i = 1, ..., M \end{aligned} \tag{3.2}$$

Multiple terms appear in the generic formulation presented in Eq. 3.1 and 3.2. The following sections provide more detailed overviews of the methodologies used to define and solve each part of the OCP. Section 2.3 focuses on the implementation and solution of the last four conditions in Eq. 3.2. Then, Section 3.2 illustrates the numerical method defined to solve Eq. 3.1, while the dynamical model (the first sub-equation in Eq. 3.2) is introduced directly in Chapter 4.

3.2. Hybrid Differential Dynamic Programming

The research objective stated in Section 2.5 includes the development of a robust and generic HDDP algorithm, expanding on the framework defined in Reference [42]. A high-level block diagram for the full algorithm procedure is provided in Figure 3.1 for clarity.

The following subsections provide detailed insight into each step of the modified HDDP algorithm developed as part of this thesis work. The devised algorithm aims at solving the general OCP formulation presented in Section 3.1. Since an augmented Lagrangian approach is adopted for terminal constraints, the terminal cost function φ is reformulated as:

$$\tilde{\varphi}_i(\boldsymbol{x}_-, \boldsymbol{w}_-, \boldsymbol{x}_+, \boldsymbol{w}_+, \boldsymbol{\lambda}_i, \sigma) = \varphi_i(\boldsymbol{x}_-, \boldsymbol{w}_-, \boldsymbol{x}_+, \boldsymbol{w}_+) + \boldsymbol{\lambda}_i^T \boldsymbol{\Psi}_i(\boldsymbol{x}_-, \boldsymbol{w}_-, \boldsymbol{x}_+, \boldsymbol{w}_+) + \sigma \| \boldsymbol{\Psi}_i(\boldsymbol{x}_-, \boldsymbol{w}_-, \boldsymbol{x}_+, \boldsymbol{w}_+) \|^2 \tag{3.3}$$

where the $\tilde{\varphi}$ represents the augmented terminal cost, $\boldsymbol{\lambda}_i \in \mathcal{R}^{n_\lambda \times 1}$ is the Lagrange multipliers vector (referred to the terminal constraints $\boldsymbol{\Psi}_i$) of size n_λ, and σ is the penalty parameter. The cost functional J defined in Eq. 3.1 is adjusted by replacing the terminal cost φ with its augmented counterpart $\tilde{\varphi}$: the HDDP solver aims at minimizing the resulting augmented cost functional J.

3.2.1. State Transition Maps

As with other HDDP implementations, the one adopted for this work is initialized by providing a control guess (the initial reference solution) as well as parameters defining the different solvers used throughout the procedure.

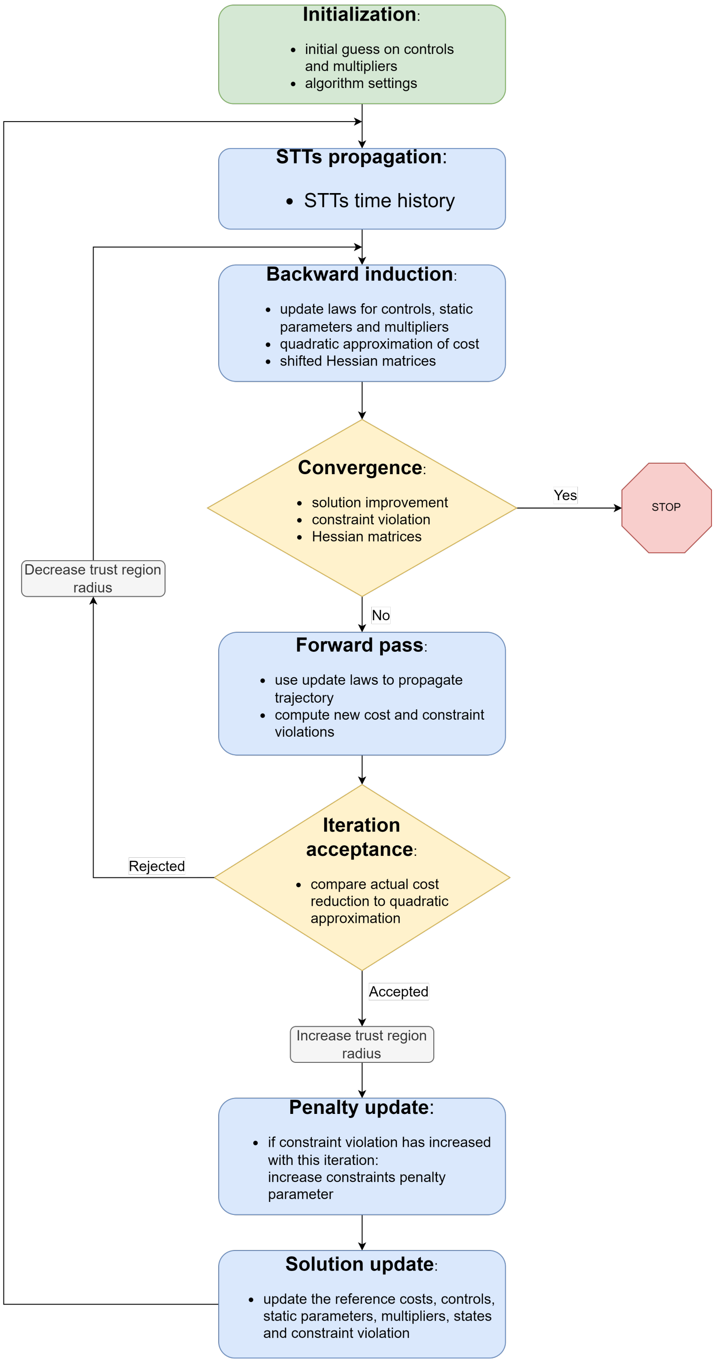

Initialization:
initial guess on controls and multipliers
algorithm settings
STTs propagation:
STTs time history
Backward induction:
update laws for controls, static parameters and multipliers
quadratic approximation of cost
shifted Hessian matrices
Convergence:
solution improvement
constraint violation
Hessian matrices
Yes
STOP
No
Decrease trust region radius
Forward pass:
use update laws to propagate trajectory
compute new cost and constraint violations
Iteration acceptance:
compare actual cost reduction to quadratic approximation
Rejected
Accepted
Increase trust region radius
Penalty update:
if constraint violation has increased with this iteration: increase constraints penalty parameter
Solution update:
update the reference costs, controls, static parameters, multipliers, states and constraint violation

Figure 3.1: Block-diagram representation of the HDDP algorithm presented in [42].

An algorithm iteration starts by obtaining the STMs between stages of the reference trajectory. The conventional approach to STMs variation mapping is illustrated in Figure 3.2. Throughout this document, STMs refers to the first- and second-order variation mappings from one stage to the next one. The STMs notation used throughout this study adopts an augmented state convention: the augmented state $\boldsymbol{X}$ is introduced by concatenating the state, controls, and static parameters vectors, resulting in:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{u} \\ \boldsymbol{w} \end{bmatrix} \tag{3.4}$$

A column vector convention is used, meaning that $\boldsymbol{X} \in \mathcal{R}^{n_X \times 1}$, with $n_X = n_x + n_u + n_w$. The stage and phase subscripts introduced in Section 3.1 are extended also to the augmented state. The functions used for the problem formulation also benefit from the augmented state notation, hence the dynamics, path constraints, and running cost functions are interchangeably defined using either the extended $(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})$ or the augmented state $(t, \boldsymbol{X})$ notations. While the dynamics function for state and augmented state are both represented by the same symbol $\boldsymbol{f}$, the augmented state convention requires the introduction of additional terms:

$$\boldsymbol{f}(t, \boldsymbol{X}) = \begin{bmatrix} \boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) \\ \mathbb{0}^{n_u \times 1} \\ \mathbb{0}^{n_w \times 1} \end{bmatrix} \tag{3.5}$$

where the $\mathbb{0}$ symbol is used to represent the null matrix in the real space $\mathcal{R}$ of the corresponding dimension (in this case, $\mathcal{R}^{n_u \times 1}$ and $\mathcal{R}^{n_w \times 1}$). As expected, the dynamics function is augmented using null derivatives for the static parameters (by definition, $\dot{\boldsymbol{w}} = \mathbb{0}^{n_w \times 1}$), while the stage-wise constant controls are an assumption introduced by the HDDP algorithm, allowing the algorithm to exploit the STMs approach for extended trajectory arcs [42]. Note that, without loss of generality, the stage-wise constant controls can also represent the coefficients for a parametrized representation of the control inputs to the dynamical system.

Using the augmented state notation, the first order STM (also named state transition matrix) is defined as:

$$\Phi_k^1 = \frac{\partial \boldsymbol{X}_{k+1}}{\partial \boldsymbol{X}_k} \tag{3.6}$$

where $\Phi_k^1 \in \mathcal{R}^{n_X \times n_X}$ indicates the first-order STM from stage k to $k+1$. Similarly, the second order map (also referred to as state transition tensor) is indicated as:

$$\Phi_k^2 = \frac{\partial^2 \boldsymbol{X}_{k+1}}{\partial \boldsymbol{X}_k^2} \tag{3.7}$$

where $\Phi_k^2 \in \mathcal{R}^{n_X \times n_X \times n_X}$ is the second-order STM from stage k to $k+1$. Using this notation, the augmented state variation mapping, also illustrated in Figure 3.2, is implemented through:

$$\delta \boldsymbol{X}_{k+1} = \Phi_k^1 \delta \boldsymbol{X}_k + \frac{1}{2} \delta \boldsymbol{X}_k^T \cdot \Phi_k^2 \cdot \delta \boldsymbol{X}_k \tag{3.8}$$

where the δ symbol indicates a variation from the reference value of the corresponding quantity (in this case the augmented states at stages k and $k+1$).

Partial derivatives and tensor notations

The computation of STMs is numerically carried out by propagating variational equations [94]. Before focusing on the formulation of variational equations, some additional notation is introduced for clarity and conciseness. The partial derivatives of a scalar quantity with respect to vector variables are indicated using subscripts. Considering the example of the partial derivatives of the cost functional J with respect to a dummy variable $q = [q_1 \;\; q_2 \;\; \cdots \;\; q_n]^T$, the notation is:

$$\begin{aligned} \nabla_q J = J_q &= \begin{bmatrix} \frac{\partial J}{\partial q_1} & \frac{\partial J}{\partial q_2} & \cdots & \frac{\partial J}{\partial q_n} \end{bmatrix} \\ \nabla_{qq} J = J_{qq} &= \begin{bmatrix} \frac{\partial^2 J}{\partial q_1 \partial q_1} & \frac{\partial^2 J}{\partial q_1 \partial q_2} & \cdots & \frac{\partial^2 J}{\partial q_1 \partial q_N} \\ \frac{\partial^2 J}{\partial q_2 \partial q_1} & \frac{\partial^2 J}{\partial q_2 \partial q_2} & \cdots & \frac{\partial^2 J}{\partial q_2 \partial q_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 J}{\partial q_N \partial q_1} & \frac{\partial^2 J}{\partial q_N \partial q_2} & \cdots & \frac{\partial^2 J}{\partial q_N \partial q_N} \end{bmatrix} \end{aligned} \tag{3.9}$$

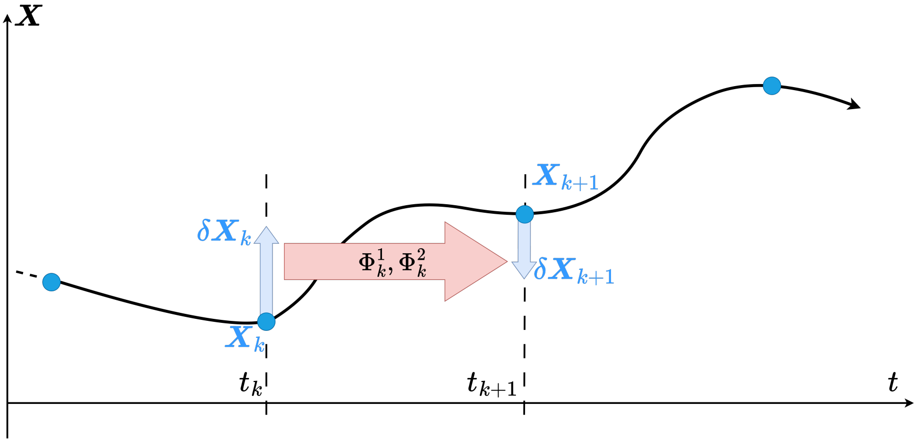


Figure 3.2: Illustration of the augmented state variations mapping between stages k and $k+1$.

where the ∇_q operator indicates gradient with respect to variable q, and ∇_{qq} indicates the Hessian matrix with respect to the same variable q. It is noticed that first-order partials are defined by concatenating partial derivatives with respect to each component along the second dimension, meaning that $\frac{\partial J}{\partial q_v} = J_q(1, v), \ \forall v = 1, ..., n$. Similarly, the second-order partials are obtained by concatenating the partial derivatives of the gradient along the second dimension, resulting in $\frac{\partial^2 J}{\partial q_v \partial q_w} = J_{qq}(v, w), \ \forall v, w = 1, ..., n$. The convention is that the indexing logic used to access the matrix with the partial derivatives reflects the order of differentiation. According to this reasoning, second-order derivatives for vector quantities (in this example the initial conditions function $\boldsymbol{\Gamma} = [\boldsymbol{\Gamma}_1, \ ..., \ \boldsymbol{\Gamma}_p, \ ..., \ \boldsymbol{\Gamma}_{n_x}, \ \forall p = 2, ..., n_x - 1]$ is considered) are defined as $\frac{\partial^2 \boldsymbol{\Gamma}_p}{\partial q_v \partial q_w} = \boldsymbol{\Gamma}_{qq}(p, v, w), \ \forall p = 1, ..., n_x, \ \forall v, w = 1, ..., n.$

While methodologies and notations for matrix-products are well established, the HDDP incorporates higher-dimensional quantities (e.g., second-order STMs or second derivatives of vector functions). The tensor products convention adopted throughout this work defines matrix-tensor products as:

$$(A \cdot B)(:, :, p) = A(:, :) \cdot B(:, :, p), \ \forall p = 1, ..., dim(B, 3) \tag{3.10}$$

where A is a generic matrix, B is a generic tensor and $dim(B, 3)$ indicates the size of the third dimension of B. Similarly, vector-tensor products are defined as:

$$(A \cdot B)(:, :, p) = A(:) \cdot B(:, :, p), \ \forall p = 1, ..., dim(B, 3) \tag{3.11}$$

where A in this case is a generic vector.

Variational equations

The STMs define the sensitivity of the dynamical system at a certain moment with respect to variations at earlier times. Several approaches can be adopted to compute their values (e.g., finite differences or complex-step differences [95]): the variational equations provide a generic and widely accepted methodology to compute STMs [94, 95, 96]. By definition, the variational equations describe the evolution of small perturbations in a dynamical system: to derive their mathematical expression, the explicit form of the (augmented) state at a specific stage is expressed as:

$$\boldsymbol{X}_{k+1} = \boldsymbol{X}_k + \int_{t_k}^{t_{k+1}} \boldsymbol{f}(t, \boldsymbol{X}(t))dt, \tag{3.12}$$

which is referred to as the dynamics transition function. Given the STMs definition (Eq. 3.6 and Eq. 3.7), the following step is to compute the derivative of Eq. 3.12 with respect to $\boldsymbol{X}_k$ up to two times to obtain the first- and second-order STMs. While the first term in Eq. 3.12 is trivial, the second term requires Leibniz's rule to differentiate the integral term, resulting in the equations:

$$\begin{aligned}\Phi_k^1 &= \mathbb{1} + \int_{t_k}^{t_{k+1}} \boldsymbol{f}_{\boldsymbol{X}}(t, \boldsymbol{X}(t))\Phi^1(t)dt \\ \Phi_k^2 &= \int_{t_k}^{t_{k+1}} \left[\boldsymbol{f}_{\boldsymbol{X}}(t, \boldsymbol{X}(t)) \cdot \Phi^2(t) + \Phi^1(t) \cdot \boldsymbol{f}_{\boldsymbol{X}\boldsymbol{X}}(t, \boldsymbol{X}(t)) \cdot \Phi^1(t)\right] dt.\end{aligned} \tag{3.13}$$

The integrals in Eq. 3.13 are solved numerically, by propagating together the augmented dynamics function and the two integrands starting from the initial conditions:

$$\begin{aligned}\boldsymbol{X}(t_k) &= \boldsymbol{X}_k \\ \Phi^1(t_k) &= \mathbb{1}^{n_X \times n_X} \\ \Phi^2(t_k) &= \mathbb{0}^{n_X \times n_X \times n_X}\end{aligned} \tag{3.14}$$

where the $\mathbb{1}$ symbol indicates the identity matrix in the real space of the corresponding dimension (in this case $\mathcal{R}^{n_X \times n_X}$).

The presented approach allows to retrieve STMs between fixed time instants by numerically integrating Eq. 3.13. In order to accommodate formulations with a free time of flight, the OCP problem discretization needs to become variable to be included in the decision process. Literature on DDP algorithms accommodating variable-duration problems is limited (as noticed from Table 2.4 and Table 2.5) and two main approaches can be identified:

- *Vary the number of discretization points*: introduced in Reference [67], this technique consists in computing an optimal variation in time of flight at the start of the backward induction process. The computed time of flight update is applied as part of the forward pass procedure by appending (or removing) stages from the end of the trajectory. This technique offers a relatively simple and understandable implementation, with minimal impact on the algorithm's runtime performance. The sensitivities of the cost functional with respect to the time of flight (required to initialize the backward induction) are computed through a truncated expansion of the dynamics:

  $$\boldsymbol{X}_{N+1+e} = \boldsymbol{f}(t_{N+1}, \boldsymbol{X}_{N+1})(t_{N+1+e} - t_{N+1}) \tag{3.15}$$

  where e indicates the index of the appended stages (i.e., $e = 1$ indicates the first extra stage, $e = -1$ indicates the second-to-last stage). The $\boldsymbol{X}_{N+1+e}$ value is used to initialize the quadratic cost model.

  The approach is appealing thanks to its straightforward derivation and implementation, but also implies several limitations. Appending extra stages at the end of the trajectory requires the use of a guess control input: given the assumption in Eq. 3.15, it is reasonable to extend the control inputs from the final stage to also the appended ones [67]. The technique is envisioned for applications within the ”regular” DDP framework, where each optimization stage corresponds to a collocation point for numerical integration [43]. Additionally, the technique does not accommodate the multi-phase problem formulation tackled in this thesis. Applying the update in time of flight in a single ”chunk” at the end of the trajectory implies that the algorithm heavily relies on the linear approximation of Eq. 3.15. The first-order approximation is expected to perform poorly in the context of long-duration orbital transfers, where the highly non-linear spacecraft dynamics imply that such an approximation is only reliable for short durations: given the long time scale of many-revolution transfers, the feature translates to undesirably slow convergence.

- *Vary the collocation of discretization points*: adopted in References [85, 86], this time-dilation approach integrates time of flight as part of the static parameters vector $\boldsymbol{w}$. The computed correction in problem duration is distributed evenly among the available stages, causing ”accordion-like” changes to the discretization [85]. The procedure is achieved by augmenting the STMs with sensitivity terms with respect to the time of flight parameter. Previous works exploit semi-analytical propagation schemes (tailored to the problem-specific dynamical models [85, 86]) to compute these sensitivities without the need to numerically propagate the variational equations. The computed corrections in time of flight are distributed among the stages as:

  $$\frac{\partial t_k}{\partial t_{flight}} = \frac{k}{N+1} \tag{3.16}$$

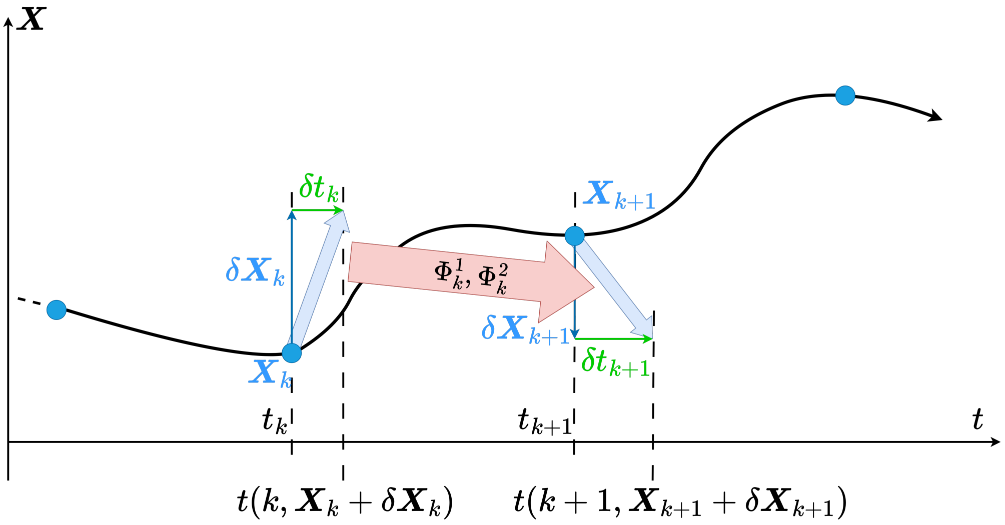

$\boldsymbol{X}$
δt_k
$\boldsymbol{X}_{k+1}$
$\delta \boldsymbol{X}_k$
Φ_k^1, Φ_k^2
$\delta \boldsymbol{X}_{k+1}$
$\boldsymbol{X}_k$
δt_{k+1}
t_k
t_{k+1}
t
$t(k, \boldsymbol{X}_k + \delta \boldsymbol{X}_k)$
$t(k+1, \boldsymbol{X}_{k+1} + \delta \boldsymbol{X}_{k+1})$

Figure 3.3: Illustration of the (augmented) state variations mapping between consecutive stages when accounting for variable stage collocation

where t_{flight} is the time of flight. For a generic dynamical model, however, numerical integration is required to propagate the system dynamics, thus requiring a different approach to embed sensitivities to the time of flight within the STMs.

The presented approaches show promising features but do not satisfy the general purposes of the algorithm developed as part of this work. This thesis proposes a novel approach to the definition of the variational equations, with the goal of providing a general methodology to incorporate variable-time formulations within the STMs framework. This approach leverages Leibniz's rule of differentiation for integral terms and applies it to a generalized version of Eq. 3.12, where the integration bounds are allowed to vary according to an arbitrary function, referred to as stage-collocation function $t_k = t(k, \boldsymbol{X}_k)$. The variable stage collocation approach is illustrated in Figure 3.3. The stage collocation function uniquely defines the OCP discretization. Notice that the stage index k can be exploited to "reformulate" the stage collocation approach into an adaptive step-size approach, since:

$$\frac{\partial t(k, \boldsymbol{X})}{\partial k} = \Delta t(k, \boldsymbol{X}) \tag{3.17}$$

where Δt is the duration of a stage. The technique can be exploited to define a dynamically evolving mesh, aimed at accurately capturing control switch events and/or for the definition of a computationally efficient adaptive mesh procedure.

Adopting this approach, the augmented state propagation is defined as:

$$\boldsymbol{X}_{k+1} = \boldsymbol{X}_k + \int_{t(k,\boldsymbol{X}(t))}^{t(k+1,\boldsymbol{X}(t))} \boldsymbol{f}(t, \boldsymbol{X}(t))dt. \tag{3.18}$$

The application of Leibniz's rule to Eq. 3.18 one time yields the variational equation for the first-order STM:

$$\begin{aligned} \Phi_k^1 = \mathbb{1} + &\int_{t(k,\boldsymbol{X}_k)}^{t(k+1,\boldsymbol{X}_{k+1})} \boldsymbol{f}_{\boldsymbol{X}}(t, \boldsymbol{X}(t))\Phi^1(t)dt \\ &+ \boldsymbol{f}(t(k+1, \boldsymbol{X}_{k+1}), \boldsymbol{X}_{k+1}) \left[t_{\boldsymbol{X}}(k+1, \boldsymbol{X}_{k+1}) - t_{\boldsymbol{X}}(k, \boldsymbol{X}_k)\right]. \end{aligned} \tag{3.19}$$

The last term in Eq. 3.19 is the contribution caused by the variation in integration step size. Applying Leibniz's rule once more to differentiate Eq. 3.19 yields the variational equation for the second-order

STM:

$$\begin{aligned}\frac{\partial^2 \boldsymbol{X}_{k+1}}{\partial \boldsymbol{X}_k^2} = \Phi_k^2 = \int_{t(k,\boldsymbol{X}_k)}^{t(k+1,\boldsymbol{X}_{k+1})} & \left[\boldsymbol{f}_{\boldsymbol{X}}(t,\boldsymbol{X}(t)) \cdot \Phi^2(t) + \Phi^1(t)^{\mathrm{T}} \cdot \boldsymbol{f}_{\boldsymbol{X}\boldsymbol{X}}(t,\boldsymbol{X}(t)) \cdot \Phi^1(t)\right] dt \\ & + \left[\boldsymbol{f}_{\boldsymbol{X}}(t(k+1,\boldsymbol{X}_{k+1}),\boldsymbol{X}_{k+1}) \cdot \Phi^1(t(k+1,\boldsymbol{X}_{k+1}))\right] \cdot \left[t_{\boldsymbol{X}}(k+1,\boldsymbol{X}_{k+1}) - t_{\boldsymbol{X}}(k,\boldsymbol{X}_k)\right] \\ & + \dot{\boldsymbol{f}}(t(k+1,\boldsymbol{X}_{k+1}),\boldsymbol{X}_{k+1}) \left[t_{\boldsymbol{X}}(k+1,\boldsymbol{X}_{k+1}) - t_{\boldsymbol{X}}(k,\boldsymbol{X}_k)\right]^2 \\ & + \boldsymbol{f}(t(k+1,\boldsymbol{X}_{k+1}),\boldsymbol{X}_{k+1}) \left[t_{\boldsymbol{X}\boldsymbol{X}}(k+1,\boldsymbol{X}_{k+1}) - t_{\boldsymbol{X}\boldsymbol{X}}(k,\boldsymbol{X}_k)\right].\end{aligned} \tag{3.20}$$

The extra terms in Eq. 3.20 also represent the contributions of variable integration step size. Similarly to the more ”conventional” (i.e., fixed time duration) approach to variational equations, the integrals in Eq. 3.19 and Eq. 3.20 are solved numerically.

The devised technique allows the incorporation of variable-time information directly within the STMs and the variable stage collocation is then integrated into the forward pass procedure, as explained in Subsection 3.2.4. No additional adjustments are required to the rest of the HDDP formulation. The devised approach follows considerations introduced in Reference [95], where the implications of variable step integration on the propagation of variational equations are investigated: the study highlights the accuracy improvement obtained using a similar variational equations approach directly within STM numerical integration. While it doesn’t translate directly to the context of this thesis, the work provides a mathematical foundation for the introduction of Eq. 3.19 and Eq. 3.20 within HDDP with variable discretization.

Performance considerations

Conversely from more common applications of STMs approach [94, 96], where the mapping is implemented between the initial condition and an arbitrary point in the trajectory, the HDDP formulation defines STMs between two consecutive stages. The STMs are computed on a pre-defined reference trajectory (provided either as the initial guess or resulting from the forward pass), meaning that all stages are fully known before propagating the variational equations. This feature allows for the a-synchronous parallel computation of the STMs, greatly leveraging the parallel computing capabilities of modern computer architectures to speed up the algorithm. Computing the STMs is, in fact, the most computationally intensive step of HDDP, as it involves the numerical propagation of $n_X^3 + n_X^2 + n_X$ dimensional quantities (n_X entries for the second-order STMs, n_X^2 for the first-order STMs, and n_X for the augmented state vector).

Methods to leverage symmetries in the STMs can also be introduced to reduce the numerical propagation size from $n_X^3 + n_X^2 + n_X$ to $n_X^3/8 + n_X^2/2 + n_X$: this results, however, in the loss of matrix notation and therefore requires tedious implementation of the full expressions for the variational equations [95]. Additionally, it is to be noted that relevant DDP use cases imply $N >> n_X$, meaning that runtime performance is typically limited by the number of numerical integration steps to be taken, rather than their size, therefore no major performance improvement is expected from symmetry exploitation. Conversely, semi-analytical propagation schemes (for both state and STMs) are expected to provide substantial performance improvements but are restricted to specific simplifying assumptions and thus not considered.

3.2.2. Backwards induction

Having defined values for all stage-wise STMs, the backward induction takes place. This procedure is the main step of DDP algorithms and consists of reducing the full-scale OCP to a series of smaller and simpler sub-problems, which are solved sequentially. The backward induction procedure takes its name from the direction used to define and solve these sub-problems, which is indeed backward (i.e., from the last stage until the first one). The mathematical foundation for this practice is Bellman’s optimality principle, stating:

> An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision [64]

Leveraging this concept, the backward induction consists of the solution of each stage sub-problem by minimizing its corresponding cost-to-go. The cost-to-go at a specified stage J_k is defined as the cost incurred from the specified stage until the end of the trajectory: assuming a single-stage formulation, the cost-to-go is defined as

$$J_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \ldots, \boldsymbol{u}_N, \boldsymbol{w}, \boldsymbol{\lambda}) = \sum_{j=k}^{N} L(t_j, \boldsymbol{x}_j, \boldsymbol{u}_j, \boldsymbol{w}) + \tilde{\varphi}(\boldsymbol{x}_{N+1}, \boldsymbol{w}, \boldsymbol{\lambda}). \tag{3.21}$$

According to Bellman's optimality principle, all upstream control policies shall be optimal with respect to the current state vector $\boldsymbol{x}_k$: since the devised HDDP algorithm also includes Lagrange multipliers $\boldsymbol{\lambda}$ and static parameters $\boldsymbol{w}$, the optimal control policy shall be in the form $\boldsymbol{u}_k(\boldsymbol{x}_k, \boldsymbol{w}, \boldsymbol{\lambda})$ [42]. According to this logic, the optimized cost-to-go J_k^* can be expressed as:

$$J_k^*(\boldsymbol{x}_k, \boldsymbol{w}, \boldsymbol{\lambda}) := \min_{\boldsymbol{u}_k, \ldots, \boldsymbol{u}_N} J_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \ldots, \boldsymbol{u}_N, \boldsymbol{w}, \boldsymbol{\lambda}) = J_k(\boldsymbol{x}_k, \boldsymbol{u}_k(\boldsymbol{x}_k, \boldsymbol{w}, \boldsymbol{\lambda}), \ldots, \boldsymbol{u}_N(\boldsymbol{x}_N, \boldsymbol{w}, \boldsymbol{\lambda})) \tag{3.22}$$

where it is noticed that only controls until the stage N appear, as the stage discretization implies that controls for a generic stage $\boldsymbol{u}_k$ are applied to the interval $[t_k,\ t_{k+1})$, meaning that controls applied to the final stage $N+1$ do not affect the state of the dynamical system. Leveraging Bellman's optimality principle, Eq. 3.22 can be transformed into the recursive relation:

$$\begin{aligned} J_k^*(\boldsymbol{x}_k, \boldsymbol{w}, \boldsymbol{\lambda}) &= \min_{\boldsymbol{u}_k} \left[L(t_k, \boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}, \boldsymbol{\lambda}) + \min_{\boldsymbol{u}_{k+1}, \ldots, \boldsymbol{u}_N} J_{k+1}(\boldsymbol{x}_{k+1}, \boldsymbol{u}_{k+1}, \ldots, \boldsymbol{u}_N, \boldsymbol{w}, \boldsymbol{\lambda}) \right] \\ &= \min_{\boldsymbol{u}_k} \left[L(t_k, \boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}, \boldsymbol{\lambda}) + J_{k+1}^*(\boldsymbol{x}_{k+1}, \boldsymbol{w}, \boldsymbol{\lambda}) \right]. \end{aligned} \tag{3.23}$$

The recursive Eq. 3.23 is the basis of all dynamic programming techniques: by storing solutions to previously solved sub-problems (i.e., the optimized cost-to-go from upstream stages J_{k+1}^*), the computational effort is considerably reduced [43]. Traditional dynamic programming techniques solve Eq. 3.23 by discretizing all decision variables (i.e., states, controls, static parameters, and Lagrange multipliers), thus guaranteeing global optimality: the "curse of dimensionality" results in unfeasible storage requirements, as the number of stored solutions to the upstream subproblems quickly reaches retractable size [33]. The DDP methodology sacrifices global optimality by restricting the state space to a quadratic validity region around the current reference solution, drastically reducing the dimensionality of the search space and only yielding local optimality guarantees [89].

The backward induction consists in the recursive solution of Eq. 3.23: starting from the last stage, each stage cost-to-go J_k is minimized through a feedback control policy $\boldsymbol{u}_k(\boldsymbol{x}_k, \boldsymbol{w}, \boldsymbol{\lambda})$, resulting in an optimized cost-to-go J_k^* which is then carried to the previous stage to repeat the process. The backward induction procedure, enabled by Bellman's optimality principle, allows the efficient optimization of control inputs over long transfer times: the multi-phase capability of the algorithm is implemented by propagating the cost-to-go information across phases. This subsection provides further details on the quadratic expansion and update processes performed on each stage and inter-phase problem (while the solution of each sub-problem is presented in detail in subsection 3.2.3).

Inter-phase quadratic expansion

As previously mentioned, DDP algorithms sequentially solve a quadratic cost model to optimize a full OCP. The process requires the definition of a quadratic cost model: this step is initialized in the backward induction by computing first- and second-order partial derivatives of the (augmented) terminal cost $\tilde{\varphi}$. Given the chosen algorithm architecture, this step corresponds exactly to an inter-phase quadratic expansion, where no upstream costs J^* are present and where "dummy" variables are used for the upstream phase. The inter-phase quadratic expansion (i.e., between phases i and $i+1$) is carried out by combining the second-order derivatives of the upstream costs $J_{i+1,1}^*$ (referred to as J_1^* in Eq. 3.24 for notational simplicity) with the partial derivatives of the terminal costs for the current phase $\tilde{\varphi}_i$ (indicated as $\tilde{\varphi}$ for notational simplicity), resulting in:

$$
\begin{aligned}
&J_{\boldsymbol{x}_+} = J^*_{\boldsymbol{x},1} + \tilde{\varphi}_{\boldsymbol{x}_+}, \quad J_{\boldsymbol{x}_+\boldsymbol{x}_+} = J^*_{\boldsymbol{x}\boldsymbol{x},1} + \tilde{\varphi}_{\boldsymbol{x}_+\boldsymbol{x}_+},\\
&J_{\boldsymbol{x}_+\boldsymbol{w}_+} = J^*_{\boldsymbol{x}\boldsymbol{w},1} + \tilde{\varphi}_{\boldsymbol{x}_+\boldsymbol{w}_+}, \quad J_{\boldsymbol{x}_+\boldsymbol{\lambda}_+} = J^*_{\boldsymbol{x}\boldsymbol{\lambda},1}, \quad J_{\boldsymbol{x}_+\boldsymbol{x}_-} = \tilde{\varphi}_{\boldsymbol{x}_+\boldsymbol{x}_-},\\
&J_{\boldsymbol{x}_+\boldsymbol{w}_-} = \tilde{\varphi}_{\boldsymbol{x}_+\boldsymbol{w}_-}, \quad J_{\boldsymbol{x}_+\boldsymbol{\lambda}_-} = \tilde{\varphi}_{\boldsymbol{x}_+\boldsymbol{\lambda}_-},\\
&J_{\boldsymbol{w}_+} = J^*_{\boldsymbol{w},1} + \tilde{\varphi}_{\boldsymbol{w}_+}, \quad J_{\boldsymbol{w}_+\boldsymbol{w}_+} = J^*_{\boldsymbol{w}\boldsymbol{w},1} + \tilde{\varphi}_{\boldsymbol{w}_+\boldsymbol{w}_+}, \quad J_{\boldsymbol{w}_+\boldsymbol{\lambda}_+} = J^*_{\boldsymbol{w}\boldsymbol{\lambda},1},\\
&J_{\boldsymbol{w}_+\boldsymbol{x}_-} = \tilde{\varphi}_{\boldsymbol{w}_+\boldsymbol{x}_-}, \quad J_{\boldsymbol{w}_+\boldsymbol{w}_-} = \tilde{\varphi}_{\boldsymbol{w}_+\boldsymbol{w}_-}, \quad J_{\boldsymbol{w}_+\boldsymbol{\lambda}_-} = \tilde{\varphi}_{\boldsymbol{w}_+\boldsymbol{\lambda}_-},\\
&J_{\boldsymbol{\lambda}_+} = J^*_{\boldsymbol{\lambda},1}, \quad J_{\boldsymbol{\lambda}_+\boldsymbol{\lambda}_+} = J^*_{\boldsymbol{\lambda}\boldsymbol{\lambda},1},\\
&J_{\boldsymbol{x}_-} = \tilde{\varphi}_{\boldsymbol{x}_-}, \quad J_{\boldsymbol{x}_-\boldsymbol{x}_-} = \tilde{\varphi}_{\boldsymbol{x}_-\boldsymbol{x}_-}, \quad J_{\boldsymbol{x}_-\boldsymbol{w}_-} = \tilde{\varphi}_{\boldsymbol{x}_-\boldsymbol{w}_-}, \quad J_{\boldsymbol{x}_-\boldsymbol{\lambda}_-} = \tilde{\varphi}_{\boldsymbol{x}_-\boldsymbol{\lambda}_-},\\
&J_{\boldsymbol{w}_-} = \tilde{\varphi}_{\boldsymbol{w}_-}, \quad J_{\boldsymbol{w}_-\boldsymbol{w}_-} = \tilde{\varphi}_{\boldsymbol{w}_-\boldsymbol{w}_-}, \quad J_{\boldsymbol{w}_-\boldsymbol{\lambda}_-} = \tilde{\varphi}_{\boldsymbol{w}_-\boldsymbol{\lambda}_-},\\
&J_{\boldsymbol{\lambda}_-} = \tilde{\varphi}_{\boldsymbol{\lambda}_-}, \quad J_{\boldsymbol{\lambda}_-\boldsymbol{\lambda}_-} = 0
\end{aligned}
\tag{3.24}
$$

where the upstream optimized cost-to-go J^*_1 and its related partials assumed null values when initializing the backward induction. Introducing the initial condition parametrization $\boldsymbol{\Gamma}_i(\boldsymbol{w}_i)$ (indicated as $\boldsymbol{\Gamma}(\boldsymbol{w})$ for notational simplicity), the inter-phase quadratic expansion is updated as:

$$
\begin{aligned}
\tilde{J}_{\boldsymbol{w}_+} &= J_{\boldsymbol{w}_+} + J_{\boldsymbol{x}_+}\Gamma_{\boldsymbol{w}}\\
\tilde{J}_{\boldsymbol{w}_+\boldsymbol{w}_+} &= J_{\boldsymbol{w}_+\boldsymbol{w}_+} + J_{\boldsymbol{x}_+}\Gamma_{\boldsymbol{w}\boldsymbol{w}} + \Gamma^T_{\boldsymbol{w}} J_{\boldsymbol{x}_+\boldsymbol{x}_+}\Gamma_{\boldsymbol{w}} + \Gamma^T_{\boldsymbol{w}} J_{\boldsymbol{x}_+\boldsymbol{w}_+} + J^T_{\boldsymbol{x}_+\boldsymbol{w}_+}\Gamma_{\boldsymbol{w}}\\
\tilde{J}_{\boldsymbol{w}_+\boldsymbol{\lambda}_+} &= J_{\boldsymbol{w}_+\boldsymbol{\lambda}_+} + \Gamma^T_{\boldsymbol{w}} J_{\boldsymbol{x}_+\boldsymbol{\lambda}_+}\\
\tilde{J}_{\boldsymbol{w}_+\boldsymbol{x}_-} &= J_{\boldsymbol{w}_+\boldsymbol{x}_-} + \Gamma^T_{\boldsymbol{w}} J_{\boldsymbol{x}_+\boldsymbol{x}_-}\\
\tilde{J}_{\boldsymbol{w}_+\boldsymbol{w}_-} &= J_{\boldsymbol{w}_+\boldsymbol{w}_-} + \Gamma^T_{\boldsymbol{w}} J_{\boldsymbol{x}_+\boldsymbol{w}_-}\\
\tilde{J}_{\boldsymbol{w}_+\boldsymbol{\lambda}_-} &= J_{\boldsymbol{w}_+\boldsymbol{\lambda}_-} + \Gamma^T_{\boldsymbol{w}} J_{\boldsymbol{x}_+\boldsymbol{\lambda}_-}
\end{aligned}
\tag{3.25}
$$

where the $\tilde{J}$ indicates the cost functional after accounting for parametrization in initial conditions.

Stage quadratic expansion

Each stage quadratic expansion combines quadratic model information coming from upstream stages J_{k+1} with dynamics and running costs defined on the specified stage L_k, as specified in Eq. 3.23. A second-order Taylor expansion of the running cost L_k and upstream cost-to-go J^*_{k+1} is performed, using the STMs mapping in Eq. 3.8 to relate differentials at stage k with stage $k+1$: the resulting expansion equations are:

$$
\begin{aligned}
\begin{bmatrix} J_{\boldsymbol{x},k} \\ J_{\boldsymbol{u},k} \\ J_{\boldsymbol{w},k} \end{bmatrix}^T &= \begin{bmatrix} L_{\boldsymbol{x},k} \\ L_{\boldsymbol{u},k} \\ L_{\boldsymbol{w},k} \end{bmatrix}^T + \begin{bmatrix} J^*_{\boldsymbol{x},k+1} \\ \mathbf{0}^{n_u} \\ J^*_{\boldsymbol{w},k+1} \end{bmatrix}^T \Phi^1_k,\\
\begin{bmatrix} J_{\boldsymbol{x}\boldsymbol{x},k} & J_{\boldsymbol{x}\boldsymbol{u},k} & J_{\boldsymbol{x}\boldsymbol{w},k} \\ J_{\boldsymbol{u}\boldsymbol{x},k} & J_{\boldsymbol{u}\boldsymbol{u},k} & J_{\boldsymbol{u}\boldsymbol{w},k} \\ J_{\boldsymbol{w}\boldsymbol{x},k} & J_{\boldsymbol{w}\boldsymbol{u},k} & J_{\boldsymbol{w}\boldsymbol{w},k} \end{bmatrix} &= \begin{bmatrix} L_{\boldsymbol{x}\boldsymbol{x},k} & L_{\boldsymbol{x}\boldsymbol{u},k} & L_{\boldsymbol{x}\boldsymbol{w},k} \\ L_{\boldsymbol{u}\boldsymbol{x},k} & L_{\boldsymbol{u}\boldsymbol{u},k} & L_{\boldsymbol{u}\boldsymbol{w},k} \\ L_{\boldsymbol{w}\boldsymbol{x},k} & L_{\boldsymbol{w}\boldsymbol{u},k} & L_{\boldsymbol{w}\boldsymbol{w},k} \end{bmatrix}\\
&\quad + \Phi^{1T}_k \begin{bmatrix} J^*_{\boldsymbol{x},k+1} & \mathbf{0}^{n_x\times n_u} & J^*_{\boldsymbol{x},k+1} \\ \mathbf{0}^{n_u\times n_x} & \mathbf{0}^{n_u\times n_u} & \mathbf{0}^{n_u\times n_x} \\ J^{*T}_{\boldsymbol{w},k+1} & \mathbf{0}^{n_x\times n_u} & J^*_{\boldsymbol{w},k+1} \end{bmatrix} \Phi^1_k + \begin{bmatrix} J^*_{\boldsymbol{x},k+1} \\ \mathbf{0}_{n_u} \\ J^*_{\boldsymbol{w},k+1} \end{bmatrix}^T \cdot \Phi^2_k,\\
J_{\boldsymbol{\lambda},k} &= J^*_{\boldsymbol{\lambda},k+1},\\
J_{\boldsymbol{\lambda}\boldsymbol{\lambda},k} &= J^*_{\boldsymbol{\lambda}\boldsymbol{\lambda},k+1},\\
\begin{bmatrix} J_{\boldsymbol{x}\boldsymbol{\lambda},k} \\ J_{\boldsymbol{u}\boldsymbol{\lambda},k} \\ J_{\boldsymbol{w}\boldsymbol{\lambda},k} \end{bmatrix} &= \begin{bmatrix} J^*_{\boldsymbol{x}\boldsymbol{\lambda},k+1} \\ \mathbf{0}^{n_u} \\ J^*_{\boldsymbol{w}\boldsymbol{\lambda},k+1} \end{bmatrix}^T \Phi^1_k.
\end{aligned}
\tag{3.26}
$$

where the $\mathbf{0}$ terms indicate partial derivatives of the upstream costs with respect to the control inputs at the corresponding stage: from Bellman's optimality principle, optimized stage sub-problems are solved through an optimal policy depending on states, static parameters, and Lagrange multipliers, thus leaving no degree of freedom to the control inputs themselves.

Stage quadratic update

The stage Quadratic Programming (QP) sub-problem defined in Eq. 3.26 is solved using a trust-region method (hence the name Trust-Region Quadratic Programming (TRQP) sub-problem). Solution approaches to the TRQP sub-problem are illustrated in detail in subsection 3.2.3. Following Bellman's optimality, the solution to the TRQP sub-problem is a control feedback law in the form:

$$\boldsymbol{\delta u}_k = A_k + B_k \boldsymbol{\delta x}_k + C_k \boldsymbol{\delta w}_k + D_k \boldsymbol{\delta \lambda}_k \tag{3.27}$$

where the $\boldsymbol{\delta}$ operator indicates small variations in the respective variables with respect to the reference solution: these variations are evaluated during the forward pass. The defined feedback law is used to update the cost-to-go J_k expansion accounting for the optimal control policy, yielding the quadratic model of optimized cost-to-go J_k^* as:

$$\begin{aligned}
\mathrm{ER}_k &= \mathrm{ER}_{k+1} + J_{\boldsymbol{u},k}^T A_k + \frac{1}{2} A_k^T J_{\boldsymbol{uu},k} A_k, \\
J_{\boldsymbol{x},k}^* &= J_{\boldsymbol{x},k} + J_{\boldsymbol{u},k}^T B_k + A_k^T J_{\boldsymbol{uu},k} B_k + A_k^T J_{\boldsymbol{ux},k}, \\
J_{\boldsymbol{xx},k}^* &= J_{\boldsymbol{xx},k} + B_k^T J_{\boldsymbol{uu},k} B_k + B_k^T J_{\boldsymbol{ux},k} + J_{\boldsymbol{ux},k}^T B_k, \\
J_{\boldsymbol{xw},k}^* &= J_{\boldsymbol{xw},k} + B_k^T J_{\boldsymbol{uu},k} C_k + B_k^T J_{\boldsymbol{uw},k} + J_{\boldsymbol{ux},k}^T C_k, \\
J_{\boldsymbol{x\lambda},k}^* &= J_{\boldsymbol{x\lambda},k} + B_k^T J_{\boldsymbol{uu},k} D_k + B_k^T J_{\boldsymbol{u\lambda},k} + J_{\boldsymbol{ux},k}^T D_k, \\
J_{\boldsymbol{w},k}^* &= J_{\boldsymbol{w},k} + J_{\boldsymbol{u},k}^T C_k + A_k^T J_{\boldsymbol{uu},k} C_k + A_k^T J_{\boldsymbol{uw},k}, \\
J_{\boldsymbol{ww},k}^* &= J_{\boldsymbol{ww},k} + C_k^T J_{\boldsymbol{uu},k} C_k + C_k^T J_{\boldsymbol{uw},k} + J_{\boldsymbol{uw},k}^T C_k, \\
J_{\boldsymbol{w\lambda},k}^* &= J_{\boldsymbol{w\lambda},k} + C_k^T J_{\boldsymbol{uu},k} D_k + C_k^T J_{\boldsymbol{u\lambda},k} + J_{\boldsymbol{uw},k}^T D_k, \\
J_{\boldsymbol{\lambda},k}^* &= J_{\boldsymbol{\lambda},k} + J_{\boldsymbol{u},k}^T D_k + A_k^T J_{\boldsymbol{uu},k} D_k + A_k^T J_{\boldsymbol{u\lambda},k}, \\
J_{\boldsymbol{\lambda\lambda},k}^* &= J_{\boldsymbol{\lambda\lambda},k} + D_k^T J_{\boldsymbol{uu},k} D_k + D_k^T J_{\boldsymbol{u\lambda},k} + J_{\boldsymbol{u\lambda},k}^T D_k
\end{aligned} \tag{3.28}$$

where the ER_k terms indicate the expected reduction in cost-to-go (defined on stage k) between the reference solution and current algorithm iteration. Notice that in Eq. 3.28 no partial derivatives with respect to control inputs $\boldsymbol{u}$ are present, since the controls for stage k are entirely defined by variations in state $\boldsymbol{x}$, parameters $\boldsymbol{w}$, and multipliers $\boldsymbol{\lambda}$ through the feedback law in Eq. 3.27, consistently with Bellman's optimality principle and with Eq. 3.26. The expected reduction ER is set to 0 when initializing the backward induction on the final stage.

Inter-phase quadratic update

The inter-phase TRQP sub-problems are solved using the same trust-region method. The chosen augmented Lagrangian technique implies a "min-max" approach to update Lagrange multipliers (i.e., computing optimal controls and parameters, while solving for multipliers that maximize the cost functional) [74]. The Lagrange multipliers $\boldsymbol{\lambda}$ are therefore computed by applying the trust-region method to the "opposite" problem, by feeding the opposite Hessian matrix $-J_{\boldsymbol{\lambda}_+\boldsymbol{\lambda}_+}$ and opposite gradient $-J_{\boldsymbol{\lambda}_+}$ to the trust-region solver. Similarly to the process followed to update the controls, the update law to the Lagrange multipliers is defined as:

$$\boldsymbol{\delta\lambda}_+ = A_{\boldsymbol{\lambda}_+} + C_{\boldsymbol{\lambda}_+} \boldsymbol{\delta w}_+ \tag{3.29}$$

where the $\boldsymbol{\delta}$ values are computed during the forward pass as variations with respect to the reference solution. The feedback law in Eq. 3.29 is applied to the quadratic expansion of the inter-phase cost-to-go (indicated as J_1 for notational clarity). The procedure yields the pre-optimized cost-to-go $\widehat{J}$, with the second-order expansion:

$$
\begin{aligned}
\mathrm{ER}_{i,0} &= \mathrm{ER}_{i,1} + J^T_{\boldsymbol{\lambda}+} A_{\boldsymbol{\lambda}+} + \frac{1}{2} A^T_{\boldsymbol{\lambda}+} J_{\boldsymbol{\lambda}+\boldsymbol{\lambda}+} A_{\boldsymbol{\lambda}+}, \\
\widehat{J}_{\boldsymbol{w}+} &= \widetilde{J}_{\boldsymbol{w}+} + J^T_{\boldsymbol{\lambda}+} C_{\boldsymbol{\lambda}+} + A^T_{\boldsymbol{\lambda}+} J_{\boldsymbol{\lambda}+\boldsymbol{\lambda}+} C_{\boldsymbol{\lambda}+} + A^T_{\boldsymbol{\lambda}+} J_{\boldsymbol{\lambda}+\boldsymbol{w}+}, \\
\widehat{J}_{\boldsymbol{w}+\boldsymbol{w}+} &= \widetilde{J}_{\boldsymbol{w}+\boldsymbol{w}+} + C^T_{\boldsymbol{\lambda}+} J_{\boldsymbol{\lambda}+\boldsymbol{\lambda}+} C_{\boldsymbol{\lambda}+} + C^T_{\boldsymbol{\lambda}+} J_{\boldsymbol{\lambda}+\boldsymbol{w}+} + J^T_{\boldsymbol{\lambda}+\boldsymbol{w}+} C_{\boldsymbol{\lambda}+}, \\
\widehat{J}_{\boldsymbol{w}_+\boldsymbol{x}_-} &= \widetilde{J}_{\boldsymbol{w}_+\boldsymbol{x}_-}, \\
\widehat{J}_{\boldsymbol{w}_+\boldsymbol{w}_-} &= \widetilde{J}_{\boldsymbol{w}_+\boldsymbol{w}_-}, \\
\widehat{J}_{\boldsymbol{w}_+\boldsymbol{\lambda}-} &= \widetilde{J}_{\boldsymbol{w}_+\boldsymbol{\lambda}-}
\end{aligned}
\tag{3.30}
$$

where the $\tilde{J}$ partials are those resulting from Eq. 3.25, while $ER_{i,1}$ is the expected reduction in cost-to-go obtained from the first stage of phase i. Since the ”min-max” approach computes maximal Lagrange multipliers, the update law in Eq. 3.29 is expected to yield a increase in cost-to-go, meaning that $ER_{i,0} \geq ER_{i,1}$. The pre-optimized cost-to-go $\widehat{J}$ quadratic model is used to define the new TRQP sub-problem to update the static parameters of the upstream phase $\boldsymbol{w}_+$, through the update law:

$$
\boldsymbol{\delta w}_+ = A_{\boldsymbol{w}_+} + B_{\boldsymbol{w}_+} \boldsymbol{\delta x}_- + C_{\boldsymbol{w}_+} \boldsymbol{\delta w}_- + D_{\boldsymbol{w}_+} \boldsymbol{\delta \lambda}_-
\tag{3.31}
$$

where it is noticed that updates in static parameters are computed from variations registered in the previous phase: for the particular case of the initial phase $i = 1$, this implies that only the feed-forward term $A_{\boldsymbol{w}_+}$ is relevant. The feedback law in Eq. 3.31 is applied to the pre-optimized cost-to-go $\widehat{J}$ expansion, yielding the quadratic expansion of the optimized cost-to-go (indicated as J^* for notational clarity):

$$
\begin{aligned}
\mathrm{ER}_{i,0} &= \mathrm{ER}_{i,0} + \widehat{J}^T_{\boldsymbol{w}_+} A_{\boldsymbol{w}_+} + \frac{1}{2} A^T_{\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{w}_+} A_{\boldsymbol{w}_+}, \\
J^*_{\boldsymbol{x}_-} &= J_{\boldsymbol{x}_-} + \widehat{J}^T_{\boldsymbol{w}_+} B_{\boldsymbol{w}_+} + A^T_{\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{w}_+} B_{\boldsymbol{w}_+} + A^T_{\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{x}_-}, \\
J^*_{\boldsymbol{x}_-\boldsymbol{x}_-} &= J_{\boldsymbol{x}_-\boldsymbol{x}_-} + B^T_{\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{w}_+} B_{\boldsymbol{w}_+} + B^T_{\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{x}_-} + \widehat{J}^T_{\boldsymbol{w}_+\boldsymbol{x}_-} B_{\boldsymbol{w}_+}, \\
J^*_{\boldsymbol{x}_-\boldsymbol{w}_-} &= J_{\boldsymbol{x}_-\boldsymbol{w}_-} + B^T_{\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{w}_+} C_{\boldsymbol{w}_+} + B^T_{\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{w}_-} + \widehat{J}^T_{\boldsymbol{w}_+\boldsymbol{x}_-} C_{\boldsymbol{w}_+}, \\
J^*_{\boldsymbol{x}_-\boldsymbol{\lambda}_-} &= J_{\boldsymbol{x}_-\boldsymbol{\lambda}_-} + B^T_{\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{w}_+} D_{\boldsymbol{w}_+} + B^T_{\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{\lambda}_-} + \widehat{J}^T_{\boldsymbol{w}_+\boldsymbol{x}_-} D_{\boldsymbol{w}_+}, \\
J^*_{\boldsymbol{w}_-} &= J_{\boldsymbol{w}_-} + \widehat{J}^T_{\boldsymbol{w}_+} C_{\boldsymbol{w}_+} + A^T_{\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{w}_+} C_{\boldsymbol{w}_+} + A^T_{\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{w}_-}, \\
J^*_{\boldsymbol{w}_-\boldsymbol{w}_-} &= J_{\boldsymbol{w}_-\boldsymbol{w}_-} + C^T_{\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{w}_+} C_{\boldsymbol{w}_+} + C^T_{\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{w}_-} + \widehat{J}^T_{\boldsymbol{w}_+\boldsymbol{w}_-} C_{\boldsymbol{w}_+}, \\
J^*_{\boldsymbol{w}_-\boldsymbol{\lambda}_-} &= J_{\boldsymbol{w}_-\boldsymbol{\lambda}_-} + C^T_{\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{w}_+} D_{\boldsymbol{w}_+} + C^T_{\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{\lambda}_-} + \widehat{J}^T_{\boldsymbol{w}_+\boldsymbol{w}_-} D_{\boldsymbol{w}_+}, \\
J^*_{\boldsymbol{\lambda}_-} &= J_{\boldsymbol{\lambda}_-} + \widehat{J}^T_{\boldsymbol{w}_+} D_{\boldsymbol{w}_+} + A^T_{\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{w}_+} D_{\boldsymbol{w}_+} + A^T_{\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{\lambda}_-}, \\
J^*_{\boldsymbol{\lambda}_-\boldsymbol{\lambda}_-} &= J_{\boldsymbol{\lambda}_-\boldsymbol{\lambda}_-} + D^T_{\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{w}_+} D_{\boldsymbol{w}_+} + D^T_{\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{\lambda}_-} + \widehat{J}^T_{\boldsymbol{w}_+\boldsymbol{\lambda}_-} D_{\boldsymbol{w}_+}.
\end{aligned}
\tag{3.32}
$$

The full process is interrupted once the first inter-phase problem (that is, the TRQP sub-problems between phase $i = 1$ and the dummy phase $i = 0$, defined with null costs) is solved, yielding a final expected reduction value $ER_{1,0}$.

3.2.3. Trust region quadratic sub-problem

It was mentioned in subsection 3.2.2 that the stage and inter-phase quadratic expansions result in QP sub-problems. These problems are solved backward sequentially until the very first stage (and therefore the full trajectory) has been optimized, following Bellman's optimality principle. The validity of the quadratic models and the method used to solve them remain open matter: both points are addressed through a trust-region method, consisting of restricting updates to the reference solution until the quadratic model shows acceptable accuracy.

The trust-region approach used by DDP algorithms consists in restricting the solution of each TRQP sub-problem to a bounded set of values (i.e., the trust region). For the remainder of this subsection, the notation only refers to stage QP sub-problems resulting from Eq. 3.26, noting that the same formulation and methods are also used for inter-phase TRQP sub-problems (i.e., the multipliers expansion in Eq.

3.24 and the parameters problem in Eq. 3.30), without loss of generality. The trust region restriction transforms the TRQP sub-problem in:

$$\min_{\delta \boldsymbol{u}_k} \frac{1}{2}\delta \boldsymbol{u}_k^T J_{\boldsymbol{uu},k}\delta \boldsymbol{u}_k + J_{\boldsymbol{u},k}\delta \boldsymbol{u}_k, \text{ such that } \|\mathcal{D}\delta \boldsymbol{u}_k\| \leq \Delta \tag{3.33}$$

where Δ is the trust region radius and $\mathcal{D}$ is the scaling matrix: the former represents the width of the trust region, while the latter defines its shape. The norm operator $\|\square\|$ refers by default to the 2-norm. The Taylor expansion used to define the second-order cost model only holds for solutions that are "close enough" to the reference trajectory: by reducing the trust-region radius Δ, the accuracy of the quadratic model is increased. The scaling matrix $\mathcal{D}$ is essential in poorly scaled problems, where the cost function is much more sensitive to a restricted set of decision variables: in such cases, the matrix $\mathcal{D}$ shall restrict the trust region to an ellipse, with minor axis along the direction of the corresponding decision variables. The TRQP problem defined in Eq. 3.33 is the focus of the methodologies introduced in this subsection.

The control updates (and equivalently also the multipliers and parameter updates) computed from the feedback law in Eq. 3.27 are guaranteed to provide a global descent direction only if the sub-problem Hessian matrix $J_{\boldsymbol{uu},k}$ is positive-definite [42]. The positive definiteness of the Hessian matrix is also required to solve the corresponding TRQP sub-problems, which would otherwise have undefined solutions. Since general OCPs do not guarantee the positive-definiteness of all Hessian matrices (a feature that is in practice very rare [42]), a regularization is introduced by performing a Hessian shift. The Hessian-shift technique consists of adding a positive term to the diagonal of the Hessian matrix (or its scaled version), effectively increasing the curvature of the TRQP sub-problem model ensuring both positive-definiteness of the Hessian matrix as well as restricting control updates to be within the trust region. A geometric representation of Hessian shifting (in a 1D control setting) is provided in Figure 3.4. The red color indicates non-accepted TRQP models (i.e., the non-shifted problem implies a negative-definite Hessian as well as a model minimizer outside the trust region), while the gradient towards green indicates progressively more satisfying solutions: as the Hessian-shift parameter is adjusted, the Hessian matrix moves towards positive definiteness, and the TRQP model minimizer $\delta \boldsymbol{u}^*$ approaches the trust-region boundary. The TRQP model gradient $J_{\boldsymbol{u},k}$ is highlighted as a dashed black line, as it is never modified by the trust-region solver and it is therefore a common feature of all the shifted TRQP models generated when adjusting the shift parameter.

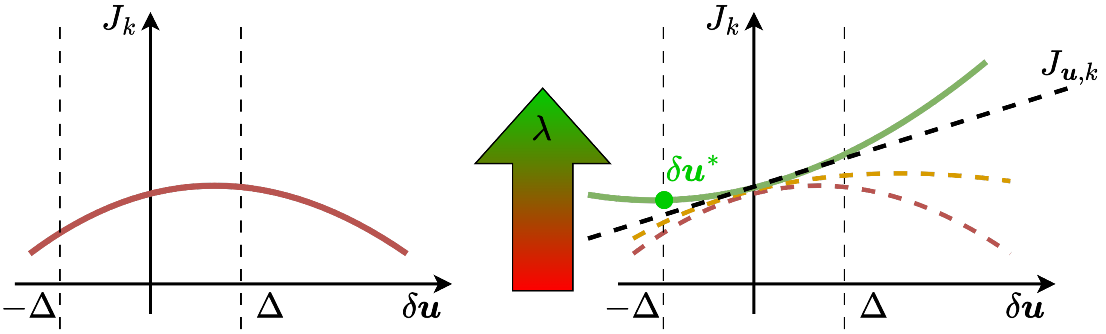


Figure 3.4: Schematic representation of the Hessian shift technique in a 1D control case, with shift λ

This work presents two algorithms to perform the Hessian shift procedure: by iteratively adjusting the Hessian shift parameter, these techniques are known to be more robust and efficient than arbitrary Hessian shifting [73]. For notational clarity, the stage subscripts k are dropped when illustrating these algorithms.

Basic trust region algorithm

First, a basic trust region algorithm is presented, which trivially enforces the trust-region constraint. An outline of the basic algorithm is provided in Alg. 1. Before performing any operation, a check is carried out to ensure that the Hessian is not identically zero, avoiding TRQP sub-problems without any control authority, which have no influence on the solution. The first step is the computation of a stationary point to the TRQP model, that is:

$$\delta \boldsymbol{u}^* = -J_{\boldsymbol{uu}}^{-1} J_{\boldsymbol{u}} \tag{3.34}$$

where $\delta \boldsymbol{u}^*$ is the candidate solution to Eq. 3.33, and the -1 exponent indicates matrix inversion. Equation 3.34, however, does not guarantee the satisfaction of the trust region constraint, nor enforces the positive definiteness of the Hessian.

First, the trust region constraint is addressed. This basic algorithm does not account for trust region scaling, meaning that the scaling matrix is assumed to be the identity matrix of corresponding size ($\mathcal{D} = \mathbb{1}^{n_u \times n_u}$). Additionally, the 2-norm trust region constraint $\|\mathcal{D}\delta \boldsymbol{u}^*\| \leq \Delta$ is distributed equally among the $\delta \boldsymbol{u}^*$ components, leading to:

$$\delta \boldsymbol{u}^*(p) \leq \frac{\Delta}{n_u}, \ \forall p = 1, ..., n_u. \tag{3.35}$$

This assumption removes the need to search for a descent direction within the algorithm, making it computationally lighter but also ”blind” to optimal descent directions. The reformulated trust-region constraint is then enforced by computing a correction term $\gamma(p)$ for each control component as:

$$\boldsymbol{\gamma}(p) = \left[\frac{\Delta}{n_u} - \boldsymbol{u}^*(p)\right] / J_{\boldsymbol{u}}(p), \ \forall p = 1, ..., n_u. \tag{3.36}$$

The Hessian correction γ is then applied to the inverted Hessian as:

$$\tilde{J}_{\boldsymbol{uu}}^{-1}(p,p) = J_{\boldsymbol{uu}}^{-1}(p,p) - \boldsymbol{\gamma}(p), \ \forall p = 1, ..., n_u \tag{3.37}$$

where $\tilde{J}_{\boldsymbol{uu}}^{-1}$ is the inverse of the candidate shifted Hessian matrix.

The positive-definiteness of the shifted Hessian is then to be enforced. Since this check requires more elaborate steps, the basic algorithm only implements a check to ensure that the new control update direction coincides with the ”unrestricted” update. The condition is implemented as:

$$\delta \boldsymbol{u}^*(p) \left[-\tilde{J}_{\boldsymbol{uu}}^{-1}(p,:) J_{\boldsymbol{uu}}\right] > 0, \ \forall p = 1, ..., n_u. \tag{3.38}$$

For components where Eq. 3.38 is not satisfied, the sign of the corresponding (shifted) inverted Hessian row $\tilde{J}_{\boldsymbol{uu}}^{-1}(p,:)$ is flipped. The TRQP model minimizer $\delta \boldsymbol{u}^*$ is then obtained by re-computing the stationary point as in Eq. 3.34, that is:

$$\delta \boldsymbol{u}^* = -\tilde{J}_{\boldsymbol{uu}}^{-1} J_{\boldsymbol{u}} \tag{3.39}$$

where the shifted Hessian now guarantees the satisfaction of the trust-region constraint $\|\delta \boldsymbol{u}^*\| \leq \Delta$. The basic trust-region solver is summarized in Alg. 1.

Algorithm 1 Basic trust region algorithm

```
Ensure: $J_{\boldsymbol{uu}} \neq \mathbb{0}^{n_u \times n_u}$
  for all $p = 1, ..., n_u$ do
    compute $\boldsymbol{\gamma}(p)$, Eq. 3.36
    shift inverted Hessian $\tilde{J}_{\boldsymbol{uu}}^{-1}$, Eq. 3.37
    if update direction is opposite, Eq. 3.38 then
      flip sign of shifted inverted Hessian row $\tilde{J}_{\boldsymbol{uu}}^{-1}(p,:)$
    end if
  end for
  recompute TRQP model minimizer $\delta \boldsymbol{u}^*$, Eq. 3.39
```

The computed shifted Hessian matrix $\tilde{J}_{\boldsymbol{uu},k}$ is used to define the feedback laws in Eq. 3.27, Eq. 3.29, and Eq. 3.31. For stage TRQP sub-problems, the update law terms are computed as:

$$\begin{aligned} A_k &= \delta \boldsymbol{u}_k^*, \\ B_k &= -\widetilde{J}^{-1}_{\boldsymbol{uu},k} J_{\boldsymbol{ux},k}, \\ C_k &= -\widetilde{J}^{-1}_{\boldsymbol{uu},k} J_{\boldsymbol{uw},k}, \\ D_k &= -\widetilde{J}^{-1}_{\boldsymbol{uu},k} J_{\boldsymbol{u\lambda},k}. \end{aligned} \tag{3.40}$$

where the $\widetilde{J}_{\boldsymbol{uu},k}$ indicates the shifted Hessian computed by the trust-region algorithm. Analogously, the Lagrange multipliers update is defined as:

$$\begin{aligned} A_{\boldsymbol{\lambda}_+} &= -\widetilde{J}^{-1}_{\boldsymbol{\lambda}_+\boldsymbol{\lambda}_+} J_{\boldsymbol{\lambda}_+} \\ C_{\boldsymbol{\lambda}_+} &= -\widetilde{J}^{-1}_{\boldsymbol{\lambda}_+\boldsymbol{\lambda}_+} J_{\boldsymbol{\lambda}_+\boldsymbol{w}_+} \end{aligned} \tag{3.41}$$

noticing that, in this case, the Hessian shift guarantees negative definiteness of the $\widetilde{J}_{\boldsymbol{\lambda}_+\boldsymbol{\lambda}_+}$ matrix, resulting in cost increase (in accordance with the "min-max" logic of the augmented Lagrangian approach). Finally, the parameter update law is assembled as:

$$\begin{aligned} A_{\boldsymbol{w}_+} &= -\widetilde{\widehat{J}}^{-1}_{\boldsymbol{w}_+\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+}, \\ B_{\boldsymbol{w}_+} &= -\widetilde{\widehat{J}}^{-1}_{\boldsymbol{w}_+\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{x}_-}, \\ C_{\boldsymbol{w}_+} &= -\widetilde{\widehat{J}}^{-1}_{\boldsymbol{w}_+\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{w}_-} \\ D_{\boldsymbol{w}_+} &= -\widetilde{\widehat{J}}^{-1}_{\boldsymbol{w}_+\boldsymbol{w}_+} \widehat{J}_{\boldsymbol{w}_+\boldsymbol{\lambda}_-} \end{aligned} \tag{3.42}$$

where the $\widetilde{\widehat{J}}_{\boldsymbol{w}_+\boldsymbol{w}_+}$ is the shifted Hessian resulting from the parameters TRQP sub-problem. Using the shifted Hessian (instead of the "natural" problem Hessian matrix) guarantees the restriction of all the terms in the feedback laws, as well as the global descent direction towards optimality. Despite this restriction, OCPs with very low control authority, or badly scaled problem formulations, can lead to numerical instabilities, thus requiring additional safeguards. While not implemented in this work, further developments shall investigate/implement safeguards such as those introduced in Reference [80].

Robust trust region algorithm

The basic trust-region algorithm offers a simple and easily interpretable implementation. The algorithm is also limited, as it assumes an even distribution of the trust-region restriction and has no mathematical guarantees for convergence. For these reasons, a more elaborate and robust trust-region algorithm is required. The identified candidate is Alg. 7.3.4 from Reference [75], which has successfully been integrated into other HDDP implementations [36, 42]. A thorough explanation of the algorithm, together with the underlying theory, is provided in Reference [75]. In the following paragraphs, the main steps of this algorithm are outlined, together with the modifications introduced as part of this work. The robust algorithm is outlined in Alg. 2.

As also done in Alg. 1, a check on the Hessian matrix is performed before computing the required Hessian shifts. If the Hessian matrix is null ($J_{\boldsymbol{uu}} = \mathbb{0}^{n_u \times n_u}$), an additional check on the gradient is performed: TRQPs where also the gradient is null are left untouched (as they have no influence on the solution), while TRQPs with non-zero gradients are solved by imposing a feed-forward control update of magnitude Δ along the negative gradient direction, that is:

$$\delta \boldsymbol{u}^* = -\mathcal{D}^{-1} \frac{J_{\boldsymbol{u}}}{\Delta}. \tag{3.43}$$

The robust trust-region solver also accommodates scaling, that is, a non-identity scaling matrix $\mathcal{D}$ in Eq. 3.33. The scaling is applied before initializing the iterative procedure of Alg. 2, effectively modifying the TRQP sub-problem to be solved, through:

$$J_{\boldsymbol{uu}} = \mathcal{D}^{-1,T} J_{\boldsymbol{uu},k} \mathcal{D}^{-1} \qquad J_{\boldsymbol{u},k} = \mathcal{D}^{-1,T} J_{\boldsymbol{u},k}. \tag{3.44}$$

The modified sub-problem model is provided as input to Alg. 2: the corresponding outputs, consisting of the optimal feed-forward model minimizer $\delta\boldsymbol{u}^*$ and the shifted Hessian matrix $\tilde{J}_{\boldsymbol{uu},k}$ are re-scaled back through:

$$\tilde{J}_{\boldsymbol{uu},k} = \mathcal{D}^T \tilde{J}_{\boldsymbol{uu},k} \mathcal{D} \qquad \delta\boldsymbol{u}^* = \mathcal{D}^{-1}\delta\boldsymbol{u}^* \tag{3.45}$$

For notational clarity, the following overview of the robust trust region algorithm assumes a non-scaled problem, noting that Eq. 3.44 and Eq. 3.45 can be used to relate any scaled TRQP sub-problem to an equivalent non-scaled formulation.

The algorithm proposed in Reference [75] aims at identifying the Hessian shift λ which satisfies both Hessian positive-definiteness and the trust region requirements. In this context, the shifted Hessian $\tilde{J}_{\boldsymbol{uu}}$ is obtained as:

$$\tilde{J}_{\boldsymbol{uu}}(\lambda) = J_{\boldsymbol{uu}} + \lambda \mathbb{1}^{n_u \times n_u} \tag{3.46}$$

The shift λ applied in Eq. 3.46 is uniform over the Hessian $J_{\boldsymbol{uu}}$ diagonal (conversely from the Hessian correction γ defined in Eq. 3.36): this allows the algorithm to operate along (and preserve) the optimal descent directions of the QP problem. The TRQP model minimizer is still defined by Eq. 3.39 and therefore depends on the chosen Hessian shift parameter λ. The key idea adopted in Reference [75] is to substitute the trust-region restriction $\|\delta\boldsymbol{u}^*\| \leq \Delta$ with a more numerically well-behaved formulation, referred to as the secular equation:

$$\frac{1}{\|\delta\boldsymbol{u}^*(\lambda)\|} - \frac{1}{\Delta} = 0. \tag{3.47}$$

It is immediately noticed that Eq. 3.47 substitutes the trust region restriction inequality with the equality sign: the algorithm accounts for cases where the model minimizer $\delta\boldsymbol{u}^*(\lambda)$ lies within the trust region (not on its boundary) by checking for interior convergence, as explained in following paragraphs. The Newton method used to root-solve the secular equation requires safeguards to guarantee convergence [73]. These safeguards come in the form of bounds to the search space for the Hessian shift λ, initialized as:

$$\begin{aligned}
\lambda^U &= \max\left\{0, -\min J_{\boldsymbol{uu}}(p,p), \frac{\|J_{\boldsymbol{u}}\|}{\Delta} - \min\left[\max\left(J_{\boldsymbol{uu}}(p,p) + \sum_{p\neq q}|J_{\boldsymbol{uu}}(p,q)|\right), \|J_{\boldsymbol{uu}}\|_F, \|J_{\boldsymbol{uu}}\|_\infty\Bigg]\right]\right\} \\
\lambda^L &= \max\left\{0, \frac{\|J_{\boldsymbol{u}}\|}{\Delta} + \min\left[\max\left(-J_{\boldsymbol{uu}}(p,p) + \sum_{p\neq q}|J_{\boldsymbol{uu}}(p,q)|, \|J_{\boldsymbol{uu}}\|_F, \|J_{\boldsymbol{uu}}\|_\infty\Bigg)\right)\Bigg]\right]\right\}
\end{aligned} \tag{3.48}$$

where $p,\ q$ are matrix indices (ranging from 1 to n_u), $|\square|$ is the absolute value operator, and $\|\square\|_F$, $\|\square\|_\infty$ are the Frobenius and infinity norm operators. The shift values λ generated by every iterate are limited by lower (λ^L) and upper (λ^U) bounds: for clarity, the Hessian-shift values λ are classified into different sets. The sets $\mathcal{F}_l$ and $\mathcal{N}_l$ contain, respectively, the feasible and unfeasible shift parameters λ. The feasible set $\mathcal{F}_l$ is further divided into the sets $\mathcal{L}_l$ and $\mathcal{G}_l$, which represent shifts λ which are, respectively, lower and higher than the solution to Eq. 3.47 (which would put the model minimizer $\delta\boldsymbol{u}^*(\lambda)$ exactly on the trust-region boundary).

Having initialized the bounds to the Hessian shit, Newton-method iterations begin by attempting a Cholesky factorization of the (shifted) Hessian matrix $\tilde{J}_{\boldsymbol{uu}}(\lambda) = L_c L_c^T$ (where L_c is the resulting lower triangular matrix) using MATLAB® `chol` function. If the factorization is successful, the Hessian is positive definite and $\lambda \in \mathcal{F}_l$. Then, a trial TRQP model minimizer $\delta\boldsymbol{u}^*(\lambda)$ is computed: if it doesn't satisfy the trust region constraint, then $\lambda \in \mathcal{L}_l$. If the trust region constraint is satisfied, $\lambda \in \mathcal{G}_l$ and a check for interior convergence is performed, consisting of an evaluation of the trust region constraint using $\lambda = 0$: since the Hessian shift is positive by definition, if $\lambda \in \mathcal{G}_l$ and $\lambda = 0$ is feasible the null Hessian shift $\lambda = 0$ is necessarily the solution. This condition is referred to as interior convergence since it corresponds to a model minimizer lying inside the trust-region, instead of the boundary imposed by the secular equation. Note that the only case for which $\delta\boldsymbol{u}^*(\lambda)$ does not lie on the trust region constraint boundary corresponds to the $\lambda = 0$ case. If the factorization fails, the corresponding shift λ is unfeasible and $\lambda \in \mathcal{N}_l$.

Having determined the current iteration set, the Hessian-shift bounds are updated accordingly through:

$$\lambda \in \mathcal{G}_l \Rightarrow \lambda^U = \lambda, \quad \lambda \in \mathcal{L}_l \Rightarrow \lambda^L = \lambda. \tag{3.49}$$

If the current shift λ is feasible, additional measures to restrict its bounds are introduced. First, the Cholesky factorization is used to compute a trial shift parameter λ^+ as:

$$\begin{aligned} \bar{\boldsymbol{w}} &= L_c^{-1}\delta\boldsymbol{u}^*(\lambda), \\ \lambda^+ &= \lambda + \left(\frac{\|\delta\boldsymbol{u}^*(\lambda) - \Delta\|}{\Delta}\right)\frac{\|\delta\boldsymbol{u}^*(\lambda)\|^2}{\|\bar{\boldsymbol{w}}\|^2}. \end{aligned} \tag{3.50}$$

where the $\bar{\boldsymbol{w}}$ vector is a measure of the gradient of Eq. 3.47 around the current Hessian shift λ (refer to Reference [75] for the complete explanation). If the vector $\bar{\boldsymbol{w}}$ is null, an additional safeguard is introduced, bypassing Eq. 3.50 and directly imposing $\lambda^+ = \lambda$.

Additional measures can be taken to improve the Hessian shift λ bounds in the case $\lambda \in \mathcal{G}_l$. To shift the Hessian matrix $\tilde{J}_{\boldsymbol{uu}}(\lambda)$ until positive-definite, the shift parameter λ is required to be opposite to the lowest eigenvalue of the "natural" Hessian $J_{\boldsymbol{uu}}$ (such eigenvalue is referred to as λ_1). It is shown [75] that the quantity $\hat{\boldsymbol{b}} \times \left(\tilde{J}_{\boldsymbol{uu}}(\lambda)\hat{\boldsymbol{b}}\right)$ (where $\hat{\boldsymbol{b}}$ is a generic unit vector) is guaranteed to be greater than the smallest eigenvalue of the shifted Hessian $\tilde{J}_{\boldsymbol{uu}}(\lambda)$ (that is, $\lambda + \lambda_1$). The unit vector $\hat{\boldsymbol{b}}$ that minimizes $\hat{\boldsymbol{b}} \times \left(\tilde{J}_{\boldsymbol{uu}}(\lambda)\hat{\boldsymbol{b}}\right)$ can be identified using the `LINPACK` method (thoroughly illustrated in Reference [75]). The resulting value $\hat{\boldsymbol{b}}$ is used to improve the lower bound, moving it closer to the minimum required shift $\lambda = -\lambda_1$, through:

$$\lambda^L = \max\left[\lambda^L, \lambda - \hat{\boldsymbol{b}} \times \left(\tilde{J}_{\boldsymbol{uu}}(\lambda)\hat{\boldsymbol{b}}\right)\right]. \tag{3.51}$$

The unit vector $\hat{\boldsymbol{b}}$ obtained via the `LINPACK` method is also adopted as the search direction to improve the model minimizer $\delta\boldsymbol{u}^*(\lambda)$. This is achieved by introducing a constant $\tilde{\alpha}$, whose value is found by root-solving:

$$\|\delta\boldsymbol{u}^*(\lambda) + \tilde{\alpha}\hat{\boldsymbol{b}}\| = \Delta. \tag{3.52}$$

The root-solving process is carried out by the MATLAB® `fsolve` function: `fsolve` is initialized twice, using search intervals with opposite signs ($[0;\ \Delta]$, $[-\Delta;\ 0]$), and out of the two solutions the chosen model minimizer $\delta\boldsymbol{u}^*(\lambda) = \delta\boldsymbol{u}^*(\lambda) + \tilde{\alpha}\hat{\boldsymbol{b}}$ is the one which solves (i.e., minimizes) the TRQP problem model in Eq. 3.33. Conversely, if the current shift λ is not feasible, a partial Cholesky factorization (refer to Reference [75] for further details) is used to compute two additional quantities (a scalar d_c and a vector $\boldsymbol{v}_c$) to approximate the $-\lambda_1$ lower bound as:

$$\lambda^L = \max\left[\lambda^L, \lambda + \frac{d_c}{\|\boldsymbol{v}_c\|^2}\right]. \tag{3.53}$$

With the Newton-method bounds defined, a new initial guess is generated. If $\lambda \in \mathcal{L}_l$ and the model gradient $J_{\boldsymbol{u}}$ is non-zero, the λ^+ becomes the new trial value, hence $\lambda = \lambda^+$. In case $\lambda \in \mathcal{G}_l$, an attempt at a Cholesky factorization of $\tilde{J}_{\boldsymbol{uu}}(\lambda^+) = L_c L_c^T$ is made: in case of success, $\lambda^+ \in \mathcal{L}_l$ and $\lambda = \lambda^+$. If the factorization does not succeed $\lambda^+ \in \mathcal{N}_l$: the lower bound is updated as $\lambda^L = \max\left(\lambda^L,\ \lambda^+\right)$, while the new trial shift λ is obtained as:

$$\lambda = \max\left[\sqrt{\lambda^L\lambda^U},\ \lambda^L + \theta(\lambda^U - \lambda^L)\right] \tag{3.54}$$

where the parameter θ is a small constant ($\theta \in (0;\ 1)$, with suggested values around $\theta = 0.01$ [75]). Equation 3.54 shows a bias towards values closer to the lower bound λ^L (thanks to the small θ value and due to the definition of geometric mean). By prioritizing smaller Hessian shifts, the algorithm introduces minimal "artificial effects" into the model, thus ensuring more accurate results [75]. If the current trial shift is unfeasible (i.e., $\lambda \in \mathcal{N}_l$), the same Eq. 3.54 is used to perform the update. Iterations produced by this algorithm are mathematically proven to generate a converging sequence of Hessian shifts λ [75], guaranteeing robustness.

The number of iterations is limited by a set of stopping criteria. First, a hard ceiling on the number of iterations is imposed. Then, a first convergence check is performed, using a defined constant $\mathcal{K}_{easy}$, for feasible shift values λ that produce reasonably acceptable results. This convergence check is implemented through:

$$\begin{aligned} &\lambda \in \mathcal{F}_l, \\ &|\|\delta\boldsymbol{u}^*(\lambda)\| - \Delta| \leq \mathcal{K}_{easy}\Delta. \end{aligned} \tag{3.55}$$

If the two conditions in Eq. 3.55 are satisfied, the retrieved solution to the TRQP sub-problem consists of the model minimizer $\delta\boldsymbol{u}^*(\lambda)$ and the shifted Hessian $\tilde{J}_{\boldsymbol{uu}}(\lambda)^{-1}$. The algorithm parameter $\mathcal{K}_{easy}$ is typically defined with values around $\mathcal{K}_{easy} = 0.0001$ [75]. The convergence check in Eq. 3.55 is referred to as the "easy" case, where the current shift λ is reasonably close to the actual solution. There are situations, however, where Eq. 3.55 is not satisfied and no significant improvement can be achieved by acting exclusively on the Hessian shift λ (refer to Reference [75] for the list and explanations of these situations). A stopping criterion for this class of problems, the so-called "hard" case, is implemented as:

$$\begin{gathered}\lambda \in \mathcal{G}_l, \\ \tilde{\alpha}^2\left[\hat{\boldsymbol{b}} \times \left(\tilde{J}_{\boldsymbol{uu}}(\lambda)\hat{\boldsymbol{b}}\right)\right] \leq \mathcal{K}_{hard}\left[\delta\boldsymbol{u}^*(\lambda) \times \left(\tilde{J}_{\boldsymbol{uu}}(\lambda)\delta\boldsymbol{u}^*(\lambda)\right) + \lambda\Delta^2\right]\end{gathered} \tag{3.56}$$

where the $\mathcal{K}_{hard}$ is a solver parameter, defined with values typically around $\mathcal{K}_{hard} = 0.0002$ [75], while the quantities $\hat{\boldsymbol{b}}$, $\tilde{\alpha}$ correspond to those in Eq. 3.52. If the current iterate does not pass the convergence test in Eq. 3.55, the stopping criterion in Eq. 3.56 is used, with the solution being defined as the model minimizer $\delta\boldsymbol{u}^*(\lambda) + \tilde{\alpha}\hat{\boldsymbol{b}}$ and the shifted Hessian $\tilde{J}_{\boldsymbol{uu}}(\lambda)^{-1}$. If none of the two tests are passed, and the number of iterations has hit its limit, a null solution $\delta\boldsymbol{u}^*(\lambda) = \mathbb{0}^{n_u \times 1}$, $\tilde{J}_{\boldsymbol{uu}}(\lambda)^{-1} = \mathbb{0}^{n_u \times n_u}$ is generated. The full algorithm is summarized in Alg. 2.

The robust trust-region solver outputs a shifted Hessian matrix exactly as the basic solver mentioned above. Consequently, the update laws are assembled in the same manner, with Eq. 3.40 for stage TRQP sub-problems, Eq. 3.41 for multipliers TRQP sub-problems, and Eq. 3.42 for parameters TRQP sub-problems.

Constrained trust region algorithm

The two trust region algorithms presented above introduce a Hessian shift to the TRQP sub-problem model. The Hessian shift has the double effect of enforcing positive definiteness of the Hessian matrix and restricting the TRQP model minimizer within the trust region. However, most OCPs also include path constraints, implying that TRQP sub-problems are also to be constrained. The approach chosen to constrain the TRQP sub-problem expands on the methodology introduced in Reference [72], based on a second-order approximation of the KKT conditions. The technique is now introduced, noting that path constraints are, by definition, enforced to all and only trajectory stages: multipliers and parameters updates are still defined by Eq. 3.29 and Eq. 3.31, obtained through the robust trust-region solver and assembled using, respectively, Eq. 3.41 and Eq. 3.42.

The chosen approach relies on a quadratic solution to the path-constrained TRQP sub-problem, hence assuming the control update to be of the corresponding order. The control feedback law, previously defined through Eq. 3.27, is instead computed as:

$$\begin{aligned}\delta\boldsymbol{u} = \delta\boldsymbol{u}_0 &+ \begin{bmatrix} U_{\boldsymbol{x}} & U_{\boldsymbol{w}} & U_{\boldsymbol{\lambda}} \end{bmatrix}\begin{bmatrix}\delta\boldsymbol{x} \\ \delta\boldsymbol{w} \\ \delta\boldsymbol{\lambda}\end{bmatrix} \\ &+ \frac{1}{2}\sum_{a=1}^{n_x}\left(U_{\boldsymbol{x}^a\boldsymbol{x}}\delta\boldsymbol{x} + U_{\boldsymbol{x}^a\boldsymbol{w}}\delta\boldsymbol{w} + U_{\boldsymbol{x}^a\boldsymbol{\lambda}}\delta\boldsymbol{\lambda}\right)\delta\boldsymbol{x}^a \\ &+ \frac{1}{2}\sum_{b=1}^{n_w}\left(U_{\boldsymbol{w}^b\boldsymbol{x}}\delta\boldsymbol{x} + U_{\boldsymbol{w}^b\boldsymbol{w}}\delta\boldsymbol{w} + U_{\boldsymbol{w}^b\boldsymbol{\lambda}}\delta\boldsymbol{\lambda}\right)\delta\boldsymbol{w}^b \\ &+ \frac{1}{2}\sum_{c=1}^{n_\lambda}\left(U_{\boldsymbol{\lambda}^c\boldsymbol{x}}\delta\boldsymbol{x} + U_{\boldsymbol{\lambda}^c\boldsymbol{w}}\delta\boldsymbol{w} + U_{\boldsymbol{\lambda}^c\boldsymbol{\lambda}}\delta\boldsymbol{\lambda}\right)\delta\boldsymbol{\lambda}^c\end{aligned} \tag{3.57}$$

where, for notational clarity, the feedback matrices/tensors are defined using the partial derivatives notation (for instance, $U_{\boldsymbol{x}}$ is the first-order feedback term with respect to the state x). The indexes corresponding to specific variables are represented using superscripts, for conciseness (i.e., $U_{\boldsymbol{x}^a\boldsymbol{x}} = U_{\boldsymbol{xx}}(:, a, :)$).

The devised approach relies on Alg. 2 for the enforcement of Hessian positive-definiteness and trust-region constraints. Path constraints are therefore applied to a TRQP model which is already convex, hence the KKT conditions are both necessary and sufficient for (constrained) optimality. The approach

Algorithm 2 Single iteration of the robust trust region algorithm

```
Attempt to factorize $\tilde{J}_{\boldsymbol{uu}}(\lambda) = L_c L_c^T$
if factorization succeeds then
  $\lambda \in \mathcal{F}_l$
  Solve $L_c L_c^T \delta\boldsymbol{u}^* = -J_{\boldsymbol{u}}$
  if $\|\delta\boldsymbol{u}^*\| < \Delta$ then
    $\lambda \in \mathcal{G}_l$; check for interior convergence.
  else
    $\lambda \in \mathcal{L}_l$.
  end if
else
  $\lambda \in \mathcal{L}_l$.
end if
Apply Eq. 3.49
if $\lambda \in \mathcal{F}_l$ then
  Solve Eq. 3.50
else if $\lambda \in \mathcal{G}$ then
  Use LINPACK method to find a unit vector $\hat{\boldsymbol{b}}$ making $\hat{\boldsymbol{b}} \times \left(\tilde{J}_{\boldsymbol{uu}}(\lambda)\hat{\boldsymbol{b}}\right)$ small.
  Apply Eq. 3.51.
  Root-solve Eq. 3.52 and update model minimizer $\delta\boldsymbol{u}^* + \tilde{\alpha}\hat{\boldsymbol{b}}$
else
  Perform partial Cholesky factorization of $\tilde{J}_{\boldsymbol{uu}}(\lambda^+)$, yielding $d_c$ and $\boldsymbol{v}_c$.
  Apply Eq. 3.53.
end if
Check for termination through Eq. 3.55, Eq. 3.56, and the limit on iteration count.
if $\lambda \in \mathcal{L}_l$ then
  Replace $\lambda$ with $\lambda^+$.
else
  if $\lambda \in \mathcal{G}_l$ then
    Attempt to factorize $\tilde{J}_{\boldsymbol{uu}}(\lambda^+) = L_c L_c^T$.
    if factorization succeeds then
      $\lambda^+ \in \mathcal{G}_l$
      Replace $\lambda$ with $\lambda^+$.
    else
      Otherwise, $\lambda^+ \in \mathcal{N}_l$.
      Apply Eq. 3.54.
    end if
  end if
else
  $\lambda \in \mathcal{N}_l$
  Apply Eq. 3.54.
end if
```

is initialized with a positive-definite Hessian matrix $\tilde{J}_{\boldsymbol{uu}}$ and a control update $\delta\boldsymbol{u}^*$ that is already within the trust region bounds. Solving a constrained stage TRQP sub-problem consists of:

$$\begin{gathered}\min_{\delta\boldsymbol{u}^*} \frac{1}{2}\delta\boldsymbol{u}^{*T}\tilde{J}_{\boldsymbol{uu}}\delta\boldsymbol{u}^* + J_{\boldsymbol{u}}\delta\boldsymbol{u}^*, \text{ such that:} \\ \|\mathcal{D}\delta\boldsymbol{u}^*\| \leq \Delta \\ \boldsymbol{g}(t_k, \boldsymbol{x}_k, \boldsymbol{u}_k + \delta\boldsymbol{u}^*, \boldsymbol{w}) = \mathbb{0}^{n_g \times 1}\end{gathered} \tag{3.58}$$

where n_g is the number of path constraints (corresponding to the length of $\boldsymbol{g}$). Inequality constraints are transformed into equality form through slack variables (as in Eq. 2.7): without loss of generality [44]. To avoid ill-conditioned systems resulting from KKT conditions, the model minimizer resulting $\delta\boldsymbol{u}^*$ used for initialization is exploited to compute the active set of constraints for the current stage: this is considered a reliable estimate, since $\delta\boldsymbol{u}^*$ is already a minimizer of the unconstrained problem [42]. The set of active constraints is condensed in a single constraint violation vector $\boldsymbol{q} \in \mathcal{R}^{n_q \times 1}$, where n_q corresponds to the number of active path constraints. The active set of constraints is adjoined to the TRQP sub-problem model of Eq. 3.58 through a set of Lagrange multipliers $\boldsymbol{\mu} \in \mathcal{R}^{n_q \times 1}$, which is assumed to be in the same quadratic form as Eq. 3.57 (and is represented using the same partial derivatives notation to indicate feedback law matrices/tensors). The KKT conditions for the resulting problem become:

$$\begin{aligned} J_{\boldsymbol{u}} + \sum_{j=1}^{n_q} \boldsymbol{\mu}^j \boldsymbol{q}^j_{\boldsymbol{u}} &= 0 \\ \boldsymbol{q} &= \mathbb{0}^{n_q \times 1} \end{aligned} \tag{3.59}$$

In general, Eq. 3.59 will not be satisfied due to the nonlinearity of path constraints functions, therefore a truncated expansion around the controls $\boldsymbol{u}$, multipliers $\boldsymbol{\lambda}$ and static parameters $\boldsymbol{w}$ is introduced. A first-order expansion of the first term in Eq. 3.59 is introduced: since this expansion also yields second-order terms, the second term in Eq. 3.59 needs to be approximated up to second order. The full procedure yields:

$$\begin{aligned} 0 &= J_{\boldsymbol{u}} + J_{\boldsymbol{ux}}\delta\boldsymbol{x} + J_{\boldsymbol{uw}}\delta\boldsymbol{w} + J_{\boldsymbol{u\lambda}}\delta\boldsymbol{\lambda} + J_{\boldsymbol{uu}}\delta\boldsymbol{u} \\ &+ \sum_{j=1}^{n_q} \boldsymbol{\mu}^j \left(\boldsymbol{q}^j_{\boldsymbol{u}} + \boldsymbol{q}^j_{\boldsymbol{ux}}\delta\boldsymbol{x} + \boldsymbol{q}^j_{\boldsymbol{uw}}\delta\boldsymbol{w} + \boldsymbol{q}^j_{\boldsymbol{u\lambda}}\delta\boldsymbol{\lambda} + \boldsymbol{q}^j_{\boldsymbol{uu}}\delta\boldsymbol{u} \right) \\ 0 &= \boldsymbol{q}^j + \boldsymbol{q}^j_{\mathbf{u}}{}^{\mathrm{T}}\delta\mathbf{u} + \frac{1}{2}\delta\mathbf{u}^{\mathrm{T}}\boldsymbol{q}^j_{\mathbf{uu}}\delta\mathbf{u} + \frac{1}{2}\delta\mathbf{x}^{\mathrm{T}}\boldsymbol{q}^j_{\mathbf{xu}}\delta\mathbf{u} \\ &+ \frac{1}{2}\delta\mathbf{u}^{\mathrm{T}}\boldsymbol{q}^j_{\mathbf{ux}}\delta\mathbf{x} + \frac{1}{2}\delta\mathbf{x}^{\mathrm{T}}\boldsymbol{q}^j_{\mathbf{xx}}\delta\mathbf{x} + \boldsymbol{q}^j_{\mathbf{w}}{}^{\mathrm{T}}\delta\mathbf{w} \\ &+ \frac{1}{2}\delta\mathbf{w}^{\mathrm{T}}\boldsymbol{q}^j_{\mathbf{pu}}\delta\mathbf{u} + \frac{1}{2}\delta\mathbf{x}^{\mathrm{T}}\boldsymbol{q}^j_{\mathbf{xp}}\delta\mathbf{w} + \frac{1}{2}\delta\mathbf{w}^{\mathrm{T}}\boldsymbol{q}^j_{\mathbf{pp}}\delta\mathbf{w} \\ &+ \boldsymbol{q}^j_{\mathbf{x}}{}^{\mathrm{T}}\delta\mathbf{x} + \frac{1}{2}\delta\mathbf{x}^{\mathrm{T}}\boldsymbol{q}^j_{\mathbf{xp}}\delta\mathbf{w} + \frac{1}{2}\delta\mathbf{w}^{\mathrm{T}}\boldsymbol{q}^j_{\mathbf{w}}\delta\mathbf{x} \end{aligned} \tag{3.60}$$

Expressing the control update $\delta\boldsymbol{u}$ through Eq. 3.57, Eq. 3.60 can be solved by grouping together coefficients for the unknown first- and second-order variations in states, multipliers, and static parameters. The affine terms (i.e., the feed-forward updates in controls$\delta\boldsymbol{u}_0$ and multipliers $\boldsymbol{\mu}$) can be obtained from:

$$0 = \begin{bmatrix} J_{\mathbf{u}} \\ q^1 \\ \vdots \\ q^{n_q} \end{bmatrix} + (\mathcal{M} + \frac{1}{2}\mathcal{H}) \begin{bmatrix} \delta\mathbf{u}_0 \\ \boldsymbol{\mu} \end{bmatrix} \tag{3.61}$$

where the $\mathcal{M}$ and $\mathcal{H}$ matrices are defined as:

$$\mathcal{M} = \begin{bmatrix} J_{\mathbf{uu}} & q^1_{\mathbf{u}} & \cdots & q^{n_q}_{\mathbf{u}} \\ q^{1\ \mathrm{T}}_{\mathbf{u}} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \\ q^{n_q\ \mathrm{T}}_{\mathrm{u}} & 0 & \cdots & 0 \end{bmatrix} \quad \mathcal{H} = \begin{bmatrix} \sum_{j=1}^{n_q} \boldsymbol{\mu}^j q^j_{\mathbf{uu}} & q^1_{\mathrm{uu}}\delta\mathbf{u}_0 & \cdots & q^{n_q}_{\mathrm{uu}}\delta\mathbf{u}_0 \\ \delta\mathbf{u}_0^{\mathrm{T}} q^1_{\mathrm{uu}} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \delta\mathbf{u}_0^{\mathrm{T}} q^{n_q}_{\mathrm{uu}} & 0 & \cdots & 0 \end{bmatrix} \tag{3.62}$$

It is noticed that Eq. 3.61 is a non-linear equation in the unknowns $\delta\boldsymbol{u}_0,\ \boldsymbol{\mu}$: its solution is retrieved using MATLAB® root-finding `fsolve` function (adopting a trust-region Levenberg-Marquadt algorithm), adjoining the trust-region constraint in Eq. 3.33 to prevent `fsolve` from exploring diverging solutions (i.e., outside the current trust region). Once the affine terms are known, the linear feedback terms are obtained by solving the linear systems:

$$0 = \begin{bmatrix} J_{\boldsymbol{ux}} \\ \boldsymbol{q}_{\boldsymbol{x}}^{1\mathrm{T}} \\ \vdots \\ \boldsymbol{q}_{\boldsymbol{x}}^{n_q\mathrm{T}} \end{bmatrix} + \begin{bmatrix} \sum_{j=1}^{n_q} \boldsymbol{\mu}^j \boldsymbol{q}_{\boldsymbol{ux}}^j \\ \delta\boldsymbol{u}_0 \boldsymbol{q}_{\boldsymbol{ux}}^1 \\ \vdots \\ \delta\boldsymbol{u}_0 \boldsymbol{q}_{\boldsymbol{ux}}^{n_q} \end{bmatrix} + (\mathcal{M}+\mathcal{H}) \begin{bmatrix} U_{\boldsymbol{x}} \\ \boldsymbol{\mu}_{\boldsymbol{x}} \end{bmatrix} \quad 0 = \begin{bmatrix} J_{\boldsymbol{uw}} \\ \boldsymbol{q}_{\boldsymbol{w}}^{1\mathrm{T}} \\ \vdots \\ \boldsymbol{q}_{\boldsymbol{w}}^{n_q\mathrm{T}} \end{bmatrix} + \begin{bmatrix} \sum_{j=1}^{n_q} \boldsymbol{\mu}^j \boldsymbol{q}_{\boldsymbol{uw}}^j \\ \delta\boldsymbol{u}_0 \boldsymbol{q}_{\boldsymbol{uw}}^1 \\ \vdots \\ \delta\boldsymbol{u}_0 \boldsymbol{q}_{\boldsymbol{uw}}^{n_q} \end{bmatrix} + (\mathcal{M}+\mathcal{H}) \begin{bmatrix} U_{\boldsymbol{w}} \\ \boldsymbol{\mu}_{\boldsymbol{w}} \end{bmatrix}$$

$$0 = \begin{bmatrix} J_{\boldsymbol{u\lambda}} \\ 0 \\ \vdots \\ 0 \end{bmatrix} + (\mathcal{M}+\mathcal{H}) \begin{bmatrix} U_{\boldsymbol{\lambda}} \\ \boldsymbol{\mu}_{\boldsymbol{\lambda}} \end{bmatrix}. \tag{3.63}$$

The affine and linear terms are then used to compute the second-order feedback terms, through the linear systems:

$$0 = \begin{bmatrix} 2\sum_{k=1}^{n_q} \boldsymbol{\mu}_{\boldsymbol{x}^j}^k (\boldsymbol{q}_{\boldsymbol{ux}}^k + \boldsymbol{q}_{\boldsymbol{uu}}^k U_{\boldsymbol{x}}) \\ U_{\boldsymbol{x}^j}^T \boldsymbol{q}_{\boldsymbol{ux}}^1 + \boldsymbol{q}_{\boldsymbol{x}^j\boldsymbol{u}}^1 U_{\boldsymbol{x}} + \boldsymbol{q}_{\boldsymbol{x}^j\boldsymbol{x}}^1 + U_{\boldsymbol{x}^j}^T \boldsymbol{q}_{\boldsymbol{uu}}^1 U_{\boldsymbol{x}} \\ \vdots \\ U_{\boldsymbol{x}^j}^T \boldsymbol{q}_{\boldsymbol{ux}}^{n_q} + \boldsymbol{q}_{\boldsymbol{x}^j\boldsymbol{u}}^{n_q} U_{\boldsymbol{x}} + \boldsymbol{q}_{\boldsymbol{x}^j\boldsymbol{x}}^{n_q} + U_{\boldsymbol{x}^j}^T \boldsymbol{q}_{\boldsymbol{uu}}^{n_q} U_{\boldsymbol{x}} \end{bmatrix} + (\mathcal{M}+\mathcal{H}) \begin{bmatrix} U_{\boldsymbol{x}^j\boldsymbol{x}} \\ \boldsymbol{\mu}_{\boldsymbol{x}^j\boldsymbol{x}} \end{bmatrix} \tag{3.64}$$

$$0 = \begin{bmatrix} 2\sum_{k=1}^{n_q} \boldsymbol{\mu}_{\boldsymbol{w}^j}^k (\boldsymbol{q}_{\boldsymbol{uw}}^k + \boldsymbol{q}_{\boldsymbol{uu}}^k U_{\boldsymbol{w}}) \\ U_{\boldsymbol{w}^j}^T \boldsymbol{q}_{\boldsymbol{uw}}^1 + \boldsymbol{q}_{\boldsymbol{w}^j\boldsymbol{u}}^1 U_{\boldsymbol{w}} + \boldsymbol{q}_{\boldsymbol{w}^j\boldsymbol{w}}^1 + U_{\boldsymbol{w}^j}^T \boldsymbol{q}_{\boldsymbol{uu}}^1 U_{\boldsymbol{w}} \\ \vdots \\ U_{\boldsymbol{w}^j}^T \boldsymbol{q}_{\boldsymbol{uw}}^{n_q} + \boldsymbol{q}_{\boldsymbol{w}^j\boldsymbol{u}}^{n_q} U_{\boldsymbol{w}} + \boldsymbol{q}_{\boldsymbol{w}^j\boldsymbol{w}}^{n_q} + U_{\boldsymbol{w}^j}^T \boldsymbol{q}_{\boldsymbol{uu}}^{n_q} U_{\boldsymbol{w}} \end{bmatrix} + (\mathcal{M}+\mathcal{H}) \begin{bmatrix} U_{\boldsymbol{w}^j\boldsymbol{w}} \\ \boldsymbol{\mu}_{\boldsymbol{w}^j\boldsymbol{w}} \end{bmatrix} \tag{3.65}$$

$$0 = \begin{bmatrix} 2\sum_{k=1}^{n_q} \boldsymbol{\mu}_{\boldsymbol{\lambda}^j}^k (\boldsymbol{q}_{\boldsymbol{uu}}^k U_{\boldsymbol{\lambda}}) \\ 0 \\ \vdots \\ 0 \end{bmatrix} + (\mathcal{M}+\mathcal{H}) \begin{bmatrix} U_{\boldsymbol{\lambda}^j\boldsymbol{\lambda}} \\ \boldsymbol{\mu}_{\boldsymbol{\lambda}^j\boldsymbol{\lambda}} \end{bmatrix} \tag{3.66}$$

$$0 = \begin{bmatrix} 2\sum_{k=1}^{n_q} \boldsymbol{\mu}_{\boldsymbol{w}^j}^k (\boldsymbol{q}_{\boldsymbol{ux}}^k + \boldsymbol{q}_{\boldsymbol{uu}}^k U_{\boldsymbol{x}}) \\ U_{\boldsymbol{w}^j}^T \boldsymbol{q}_{\boldsymbol{ux}}^1 + \boldsymbol{q}_{\boldsymbol{w}^j\boldsymbol{u}}^1 U_{\boldsymbol{x}} + \boldsymbol{q}_{\boldsymbol{w}^j\boldsymbol{x}}^1 + U_{\boldsymbol{w}^j}^T \boldsymbol{q}_{\boldsymbol{uu}}^1 U_{\boldsymbol{x}} \\ \vdots \\ U_{\boldsymbol{w}^j}^T \boldsymbol{q}_{\boldsymbol{ux}}^{n_q} + \boldsymbol{q}_{\boldsymbol{w}^j\boldsymbol{u}}^{n_q} U_{\boldsymbol{x}} + \boldsymbol{q}_{\boldsymbol{w}^j\boldsymbol{x}}^{n_q} + U_{\boldsymbol{w}^j}^T \boldsymbol{q}_{\boldsymbol{uu}}^{n_q} U_{\boldsymbol{x}} \end{bmatrix} + (\mathcal{M}+\mathcal{H}) \begin{bmatrix} U_{\boldsymbol{w}^j\boldsymbol{x}} \\ \boldsymbol{\mu}_{\boldsymbol{w}^j\boldsymbol{x}} \end{bmatrix} \tag{3.67}$$

$$0 = \begin{bmatrix} 2\sum_{k=1}^{n_q} \boldsymbol{\mu}_{\boldsymbol{x}^j}^k (\boldsymbol{q}_{\boldsymbol{uw}}^k + \boldsymbol{q}_{\boldsymbol{uu}}^k U_{\boldsymbol{w}}) \\ U_{\boldsymbol{x}^j}^T \boldsymbol{q}_{\boldsymbol{uw}}^1 + \boldsymbol{q}_{\boldsymbol{x}^j\boldsymbol{u}}^1 U_{\boldsymbol{w}} + \boldsymbol{q}_{\boldsymbol{x}^j\boldsymbol{w}}^1 + U_{\boldsymbol{x}^j}^T \boldsymbol{q}_{\boldsymbol{uu}}^1 U_{\boldsymbol{w}} \\ \vdots \\ U_{\boldsymbol{x}^j}^T \boldsymbol{q}_{\boldsymbol{uw}}^{n_q} + \boldsymbol{q}_{\boldsymbol{x}^j\boldsymbol{u}}^{n_q} U_{\boldsymbol{w}} + \boldsymbol{q}_{\boldsymbol{x}^j\boldsymbol{x}}^{n_q} + U_{\boldsymbol{x}^j}^T \boldsymbol{q}_{\boldsymbol{uu}}^{n_q} U_{\boldsymbol{w}} \end{bmatrix} + (\mathcal{M}+\mathcal{H}) \begin{bmatrix} U_{\boldsymbol{x}^j\boldsymbol{w}} \\ \boldsymbol{\mu}_{\boldsymbol{x}^j\boldsymbol{w}} \end{bmatrix} \tag{3.68}$$

(3.69)

The full solution to Eq. 3.60 consists in update laws for both controls $\delta\boldsymbol{u}$ and path-constraints multipliers $\boldsymbol{\mu}$. The updated path constraints multipliers $\boldsymbol{\mu}$ can also be used in NLP solvers to quickly re-converge a full-trajectory solution [42].

The quadratic update law defined in Eq. 3.57 is used to obtain the partial derivatives of the optimized cost-to-go J_k^* after applying the devised control update law. The stage quadratic update, normally implemented by Eq. 3.28, is adjusted to:

$$
\begin{aligned}
J^*_{\mathbf{x}} =& J_{\mathbf{x}} + U_{\mathbf{x}}^{\mathrm{T}} J_{\mathbf{u}} + J_{\mathbf{xu}}\delta\mathbf{u}_0 + U_{\mathbf{x}}^{\mathrm{T}} J_{\mathbf{uu}}\delta\mathbf{u}_0 \\
J^*_{\mathbf{w}} =& J_{\mathbf{w}} + U_{\mathbf{w}}^{\mathrm{T}} J_{\mathbf{u}} + J_{\mathbf{wu}}\delta\mathbf{u}_0 + U_{\mathbf{w}}^{\mathrm{T}} J_{\mathbf{uu}}\delta\mathbf{u}_0 \\
J^*_{\boldsymbol{\lambda}} =& J_{\boldsymbol{\lambda}} + U_{\boldsymbol{\lambda}}^{\mathrm{T}} J_{\mathbf{u}} + J_{\boldsymbol{\lambda}\boldsymbol{\lambda}}\delta\mathbf{u}_0 + U_{\boldsymbol{\lambda}}^{\mathrm{T}} J_{\mathbf{uu}}\delta\mathbf{u}_0 \\
J^*_{\mathbf{xx}} =& J_{\mathbf{xx}} + 2J_{\mathbf{xu}}U_{\mathbf{x}} + U_{\mathbf{x}}^{\mathrm{T}} J_{\mathbf{uu}}U_{\mathbf{x}} + \sum_{j=1}^{n_u} U^j_{\mathbf{xx}}\left(J_{\mathbf{u}} + J_{\mathbf{uu}}\delta\mathbf{u}_0\right)^j \\
J^*_{\mathbf{ww}} =& J_{\mathbf{ww}} + 2J_{\mathbf{wu}}U_{\mathbf{w}} + U_{\mathbf{w}}^{\mathrm{T}} J_{\mathbf{uu}}U_{\mathbf{w}} + \sum_{j=1}^{n_u} U^j_{\mathbf{ww}}\left(J_{\mathbf{u}} + J_{\mathbf{uu}}\delta\mathbf{u}_0\right)^j \\
J^*_{\boldsymbol{\lambda}\boldsymbol{\lambda}} =& J_{\boldsymbol{\lambda}\boldsymbol{\lambda}} + 2J_{\boldsymbol{\lambda}\boldsymbol{u}}U_{\boldsymbol{\lambda}} + U_{\boldsymbol{\lambda}}^{\mathrm{T}} J_{\mathbf{uu}}U_{\boldsymbol{\lambda}} + \sum_{j=1}^{n_u} U^j_{\boldsymbol{\lambda}\boldsymbol{\lambda}}\left(J_{\mathbf{u}} + J_{\mathbf{uu}}\delta\mathbf{u}_0\right)^j \\
J^*_{\mathbf{xw}} =& J_{\mathbf{xw}} + J_{\mathbf{xu}}U_{\mathbf{w}} + U_{\mathbf{x}}^{\mathrm{T}} J_{\mathbf{uu}}U_{\mathbf{w}} + \sum_{j=1}^{n_u} U^j_{\mathbf{xw}}\left(J_{\mathbf{u}} + J_{\mathbf{uu}}\delta\mathbf{u}_0\right)^j \\
J^*_{\boldsymbol{x}\boldsymbol{\lambda}} =& J_{\boldsymbol{x}\boldsymbol{\lambda}} + J_{\boldsymbol{x}\boldsymbol{u}}U_{\boldsymbol{\lambda}} + U_{\boldsymbol{x}}^{\mathrm{T}} J_{\boldsymbol{u}\boldsymbol{u}}U_{\boldsymbol{\lambda}} + \sum_{j=1}^{n_u} U^j_{\boldsymbol{x}\boldsymbol{\lambda}}\left(J_{\boldsymbol{u}} + J_{\boldsymbol{u}\boldsymbol{u}}\delta\boldsymbol{u}_0\right)^j \\
J^*_{\boldsymbol{w}\boldsymbol{\lambda}} =& J_{\boldsymbol{w}\boldsymbol{\lambda}} + J_{\boldsymbol{w}\boldsymbol{u}}U_{\boldsymbol{\lambda}} + U_{\boldsymbol{w}}^{\mathrm{T}} J_{\boldsymbol{u}\boldsymbol{u}}U_{\boldsymbol{\lambda}} + \sum_{j=1}^{n_u} U^j_{\boldsymbol{w}\boldsymbol{\lambda}}\left(J_{\boldsymbol{u}} + J_{\boldsymbol{u}\boldsymbol{u}}\delta\boldsymbol{u}_0\right)^j
\end{aligned}
\tag{3.70}
$$

3.2.4. Forward Pass

The feedback laws computed during the backward induction are then applied in the forward pass. The procedure is carried out by forward propagating the trajectory, using the update laws specified in Eq. 3.31, Eq. 3.29, and Eq. 3.27 in the corresponding stages. Notice that update laws are referred to specific stages, hence the numerical integration bounds are defined by the stage collocation function $t(k, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})$. Variations with respect to the reference trajectory (indicated by the overhead $\bar{\square}$ operator) are defined as:

$$
\boldsymbol{\delta x}_k = \boldsymbol{x}_k - \bar{\boldsymbol{x}}_k \qquad \boldsymbol{\delta w} = \boldsymbol{w} - \bar{\boldsymbol{w}} \quad \boldsymbol{\delta\lambda} = \boldsymbol{\lambda} - \bar{\boldsymbol{\lambda}} \tag{3.71}
$$

The forward pass is summarized in Alg. 3. The new reference solution obtained from the forward pass is used to compute cost and feasibility metrics of the current iteration: for later use, it is important to note the actual cost reduction AR, computed as the cost difference between the trial iterate J_{new} and the reference iterate $\bar{J}$, as well as the feasibility metric f, computed as:

$$
f := \sqrt{\frac{1}{M}\sum_{i=1}^{M}\left[\| \, \boldsymbol{\Psi}_i\left(\boldsymbol{x}_{i,N_i+1}, \boldsymbol{w}_i, \boldsymbol{x}_{i+1,1}, \boldsymbol{w}_{i+1}\right)\|^2\right]} \tag{3.72}
$$

3.2.5. Trust region update

The trust-region solvers defined in Subsection 3.2.3 constitute the inner-most part of a full trust region iterative procedure. The trust-region radius Δ used to restrict a trial iterate is to be updated according to the quadratic model accuracy. The quadratic model validity metric ρ is defined by the expected cost reduction $ER_{1,0}$ and the actual cost reduction $AR = J - \bar{J}$, through:

$$
\rho = \frac{AR}{ER_{1,0}} \tag{3.73}
$$

For the quadratic model to be accurate, its validity metric ρ shall have values close to 1. A quadratic model tolerance ϵ_1 is therefore introduced, leading to the trust region acceptance criterion:

$$
|\rho - 1| \leq \epsilon_1 \tag{3.74}
$$

Algorithm 3 Forward pass procedure

```
for all phases $i = 1, ..., M$ do
    compute $\delta \boldsymbol{x}_-,\ \delta \boldsymbol{w}_-,\ \delta \boldsymbol{\lambda}_-$
    update $\boldsymbol{w}$ through Eq. 3.31
    update $\boldsymbol{\lambda}$ through Eq. 3.29
    compute $\boldsymbol{X},\ \delta \boldsymbol{x}_1,\ \delta \boldsymbol{w},\ \delta \boldsymbol{\lambda}$
    for all stages $k = 1, ..., N_i$ do
        compute $t_k = t(k, \boldsymbol{X})$
        update $\boldsymbol{u}_k$ through Eq. 3.27
        update $\boldsymbol{X}$, store its value as new reference $\bar{\boldsymbol{X}}_k$
        compute $t_{k+1} = t(k+1, \boldsymbol{X})$
        propagate Eq. 3.18
        update new $\boldsymbol{x}_{k+1}$ and compute related $\delta \boldsymbol{x}_{k+1}$
    end for
    set last stage reference $\bar{\boldsymbol{X}}_{N_i+1}$ using the reference control $\boldsymbol{u}_{N_i}$ and $\boldsymbol{x}_{N_i+1}$ obtained from the last step of the previous loop
end for
```

Typical approaches to the definition of an acceptance criterion tend to be relaxed towards 'more optimistic' iterates, that is cases where $AR \geq ER_{1,0} \Rightarrow \rho \geq 1$. The algorithm developed as part of this work accommodates this relaxed tolerance: to minimize the number of hyper-parameters, only Eq. 3.74 is used to check for the quadratic model validity.

The trust-region radius is updated according to the trust-region acceptance in Eq. 3.74. Several methods for this update step have been investigated in recent studies [97], mostly aimed at increasing convergence speed by favoring larger Δ values. These methods, however, rely on gradient/Hessian information regarding the optimized model [98, 99, 100]: since the HDDP algorithm sequentially solves numerous TRQP sub-problems with different quadratic models, state-of-the-art approaches to increase the convergence speed in trust-region methods are inapplicable. For the scope of this work, the trust region radius is updated through a simple non-monotone strategy [101]:

$$\Delta_{p+1} = \begin{cases} \min\left((1+\kappa_d)\Delta_p, \Delta_{\max}\right) & \text{if } |\rho - 1| \leq \epsilon_1 \\ \max\left((1-\kappa_d)\Delta_p, \Delta_{\min}\right) & \text{otherwise} \end{cases} \tag{3.75}$$

where p and $p+1$ are indices for consecutive HDDP trust-region iterates. The strategy requires an initial trust-region radius value Δ_0 and a trust-region update parameter $\kappa_d \in (0;\ 1)$. When an iterate is accepted, the quadratic model is considered valid and the trust region radius is increased to favor convergence speed. Conversely, rejected iterates imply inaccurate models, therefore the trust region radius is decreased to favor smaller and more accurate updates. The minimum trust-region radius Δ_{min} controls the identification of "stalled" iterations (i.e., iterations where accurate quadratic models are only obtained if controls are excessively restricted), thus interrupting the optimization. The maximum trust-region radius Δ_{max} is used to limit excessively "confident" iterates but it is typically considered an uninfluential parameter in previous works [39, 86]).

3.2.6. Convergence test

The iterative optimization process is interrupted once a set of convergence criteria (or a maximum runtime limit) is met. The quadratic cost model defined by the HDDP process enables the enforcement of both first- and second-order optimality conditions. First-order optimality is enforced through the expected cost decrease between consecutive (accepted) iterates as:

$$ER_{1,0} \leq \epsilon_{opt} \qquad \& \qquad f \leq \epsilon_{feas} \tag{3.76}$$

where ϵ_{opt} is the optimality threshold, while ϵ_{feas} is the feasibility threshold. Second-order optimality is checked through the positive-definiteness of all the "natural" Hessian matrices (i.e., before applying the corresponding Hessian shift). For stage and static parameters TRQP sub-problems, the corresponding Hessians $J_{\boldsymbol{uu},k}$ and $\widehat{J}_{\boldsymbol{w}_+\boldsymbol{w}_+}$ are required to be positive definite, while the Lagrange multipliers TRQP sub-problem requires positive-definite (opposite) Hessian $-\tilde{J}_{\boldsymbol{\lambda}_+\boldsymbol{\lambda}_+}$, according to the "min-max" logic.

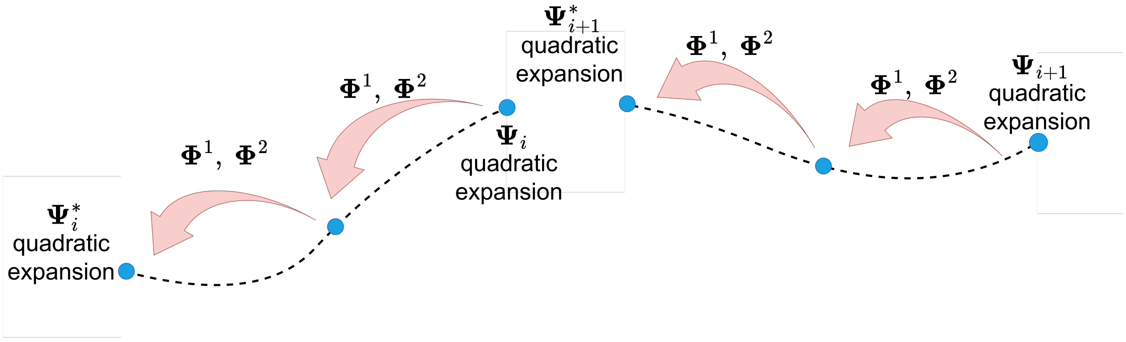

$\boldsymbol{\Psi}_{i+1}^*$
quadratic
expansion
$\boldsymbol{\Phi}^1$, $\boldsymbol{\Phi}^2$
$\boldsymbol{\Phi}^1$, $\boldsymbol{\Phi}^2$
$\boldsymbol{\Psi}_{i+1}$
quadratic
expansion
$\boldsymbol{\Phi}^1$, $\boldsymbol{\Phi}^2$
$\boldsymbol{\Psi}_i$
quadratic
expansion
$\boldsymbol{\Phi}^1$, $\boldsymbol{\Phi}^2$
$\boldsymbol{\Psi}_i^*$
quadratic
expansion

Figure 3.5: Backward propagation of the second-order model of the constraints violation $\boldsymbol{\Psi}_i$ for multiple phases

When introducing path constraints, the convergence test is adjusted accordingly. Positive definiteness is only enforced on the reduced Hessians (i.e., cost Hessians with respect to constraint-satisfying controls) [42]. The Jacobian of the active constraints is readily available, as the devised technique estimates the active set of constraints and computes its quadratic expansion, therefore reduced Hessians $H_{R,k}$ can be computed with minimal computational overhead as $H_{R,k} = Z_k^{\mathrm{T}} J_{\boldsymbol{uu},k} Z_k$, where Z_k is the null space of each stage k active set of constraints. The null space is efficiently computed through MATLAB® `null` routine.

3.2.7. Penalty update

The augmented Lagrangian approach is introduced in Section 2.3, and integrated within HDDP in Section 3.2. This approach employs both Lagrange multipliers as well as a penalty term: the solution sensitivity to the penalty parameter σ values is a well-documented issue [39, 42]. This problem is typically tackled by hand-tuning parameter values until the solution is deemed acceptable. The hand-tuning process, however, implies trial-and-error iterations which can become considerably time-consuming when in the context of long/high-dimensional OCPs (such as many-revolutions transfers). To address such limitations, this work proposes a novel algorithm for adaptive tuning of the penalty parameter σ, with the aim of robustly achieving solutions where feasibility closely meets the desired threshold ϵ_{feas}. The feature is desired as it allows greater control over OCP solutions, specifically on the balance between optimality and feasibility, which are typically conflicting [80].

The novel adaptive approach

In Subsection 3.2.2, expressions for the quadratic expansions of the optimized cost-to-go J^* have been introduced after solving the stage and inter-phase sub-problems (Eq. 3.28 and Eq. 3.32, respectively). In this work, the same concept is extended to constraint violations, allowing a closed-form expression for the sensitivity of the feasibility metric f to the constraint violations $\boldsymbol{\Psi}_i$. An additional closed-form relation between the constraint violations $\boldsymbol{\Psi}_i$ and the penalty parameter σ is also derived. The chain rule is then leveraged to link the two relations, obtaining a linear model for the feasibility metric f as a function of the penalty parameter σ: this linear model is finally used to compute updates in penalty parameter σ to match the required feasibility threshold ϵ_{feas}.

The quadratic model of the constraint violations is propagated backward using the same equations as the stage and inter-phase sub-problems, by treating each component of the terminal constraints function $\boldsymbol{\Psi}_i$ individually. Being single-variable functions, each of the $\boldsymbol{\Psi}_i$ components can be mapped to previous stages using Eq. 3.26, while substituting the J terms with individual components of the constraint violation vectors $\boldsymbol{\Psi}_i$ and removing running cost terms L. Similarly to the cost-to-go backward induction, the process is initialized by analytically computing the partial derivatives of the constraint violations $\boldsymbol{\Psi}_i$ at the final stage and then propagating backward until the first stage. Since different terminal constraints are defined for each phase, the backward mapping stops after every inter-phase update, and it is re-initialized using the new partial derivatives of the constraints from the previous phase. An illustration of the procedure is provided in Figure 3.5. The backward mapping is carried out along the backward induction, producing an array of quadratic-expansion objects for each terminal constraint violation function $\boldsymbol{\Psi}_i$.

The feasibility metric f is defined as a function of constraint violations $\boldsymbol{\Psi}_i$ in Eq. 3.72. The multi-phase problem formulation is accommodated by concatenating the constraint violation vectors $\boldsymbol{\Psi}_i$ in a single $\boldsymbol{\Psi}$, defined as:

$$\boldsymbol{\Psi} = \begin{bmatrix} \boldsymbol{\Psi}_1 \\ \vdots \\ \boldsymbol{\Psi}_M \end{bmatrix}. \tag{3.77}$$

Equation 3.72 is therefore differentiated with respect to the full constraint violation vector $\boldsymbol{\Psi}$, resulting in:

$$\begin{aligned} \frac{\partial f}{\partial \boldsymbol{\Psi}} &= \frac{\frac{\partial}{\partial \boldsymbol{\Psi}}\left(\frac{1}{M}\|\boldsymbol{\Psi}\|^2\right)}{2\sqrt{\frac{1}{M}\|\boldsymbol{\Psi}\|^2}}, \\ \frac{\partial}{\partial \boldsymbol{\Psi}}\left(\frac{1}{M}\|\boldsymbol{\Psi}\|^2\right) &= \frac{2}{M}\boldsymbol{\Psi}^T, \\ \frac{\partial f}{\partial \boldsymbol{\Psi}} &= \frac{\boldsymbol{\Psi}^{\boldsymbol{T}}}{\sqrt{M}\|\boldsymbol{\Psi}\|}. \end{aligned} \tag{3.78}$$

The sensitivity of the constraint violations with respect to the penalty parameter is now derived. The desired term $\frac{\partial \boldsymbol{\Psi}}{\partial \sigma}$ is split into intermediate terms through the chain rule. To achieve this, the term $\boldsymbol{Y} = [\boldsymbol{x};\ \boldsymbol{w}; \boldsymbol{\lambda}]$ is introduced: the quantities figuring in $\boldsymbol{Y}$ are the same appearing in the sensitivities of the optimized cost-to-go (and therefore also in each constraint violation component) after stage and inter-phase quadratic updates (e.g., in Eq. 3.28 or in Eq. 3.32). The introduction of the chain rule in the term $\frac{\partial \boldsymbol{\Psi}}{\partial \sigma}$ results in:

$$\frac{\partial \boldsymbol{\Psi}}{\partial \sigma} = \frac{\partial \boldsymbol{\Psi}}{\partial \boldsymbol{Y}}\frac{\partial \boldsymbol{Y}}{\partial J}\frac{\partial J}{\partial \sigma}. \tag{3.79}$$

The three terms appearing in Eq. 3.79 are tackled individually. The first term $\frac{\partial \boldsymbol{\Psi}}{\partial \boldsymbol{Y}}$ represents the sensitivity of the constraint violations with respect to the quantities $\boldsymbol{Y}$, hence corresponding to the final sensitivities obtained at the end of the backward induction over a single phase: indicating such quantities with $\boldsymbol{\Psi}^*_{\boldsymbol{Y}}$ (following the same convention as the optimized cost-to-go), it follows that:

$$\frac{\partial \boldsymbol{\Psi}}{\partial \boldsymbol{Y}} = \boldsymbol{\Psi}^*_{\boldsymbol{Y}}. \tag{3.80}$$

The second term in Eq. 3.79 can be obtained similarly from the backward induction results by leveraging the inverse function differential rule. Using the same $\boldsymbol{Y}$ notation, the full quadratic model of the optimized cost-to-go at the beginning of each phase is condensed as $J^*_{\boldsymbol{Y},i}$. Following the same notation introduced for the constraint violations, the sensitivities of the optimized cost-to-go are concatenated in a single vector $J^*_{\boldsymbol{Y}}$ as:

$$J^*_{\boldsymbol{Y}} = \begin{bmatrix} J^*_{\boldsymbol{Y},1,0} \\ \vdots \\ J^*_{\boldsymbol{Y},M,0} \end{bmatrix} \tag{3.81}$$

Under this convention, it follows that:

$$\begin{aligned} \frac{\partial J}{\partial \boldsymbol{Y}}(\boldsymbol{Y}) &= J^*_{\boldsymbol{Y}}, \\ \frac{\partial \boldsymbol{Y}}{\partial J} &= \frac{1}{\frac{\partial J}{\partial \boldsymbol{Y}}(\boldsymbol{Y}(J))} = \frac{1}{J^*_{\boldsymbol{Y}}}. \end{aligned} \tag{3.82}$$

Finally, the last term in Eq. 3.79 is obtained by differentiating the augmented Lagrangian definition introduced within HDDP in Eq. 2.9 with respect to the penalty parameter σ, yielding:

$$\frac{\partial J}{\partial \sigma} = \|\boldsymbol{\Psi}\|^2 \tag{3.83}$$

It is noticed that only a first-order model was considered in Eq. 3.79: adding any second-order terms would yield no effect due to the $\frac{\partial^2 J}{\partial \sigma^2} \equiv 0$. Substituting the derived equations in Eq. 3.79 yields:

$$\frac{\partial \boldsymbol{\Psi}}{\partial \sigma} = \boldsymbol{\Psi}^*_{\boldsymbol{Y}}\frac{1}{J^*_{\boldsymbol{Y}}}\|\boldsymbol{\Psi}\|^2 \tag{3.84}$$

The sensitivities in Eq. 3.78 and Eq. 3.84 can be combined using the chain rule, yielding:

$$\frac{\partial f}{\partial \sigma} = \frac{\boldsymbol{\Psi^T}}{\sqrt{M}\|\boldsymbol{\Psi}\|}\boldsymbol{\Psi}^*_{\boldsymbol{Y}}\frac{1}{J^*_{\boldsymbol{Y}}}\|\boldsymbol{\Psi}\|^2 = \frac{\boldsymbol{\Psi}^T\boldsymbol{\Psi}^*_{\boldsymbol{Y}}\|\boldsymbol{\Psi}\|}{\sqrt{M}J^*_{\boldsymbol{Y}}}. \tag{3.85}$$

At every iteration, the penalty parameter is updated through:

$$\sigma_{p+1} = \sigma_p + \min\left(\max\left(\frac{\epsilon_{feas} - f_p}{\frac{\partial f}{\partial \sigma}}\kappa_\sigma, -\Delta_\sigma\right), \Delta_\sigma\right) \tag{3.86}$$

where p and $p+1$ indicate two consecutive HDDP iterations. The κ_σ and Δ_σ are algorithm parameters that introduce some margin in the target feasibility threshold and limit excessive updates in penalty parameter, respectively. The algorithm requires the initialization of the penalty parameter σ_0.

The update law in Eq. 3.86 requires the definition of a restriction parameter Δ_σ. While being considerably less influential than the penalty parameter σ itself, the restriction parameter Δ_σ requires tuning and thus does not entirely satisfy the posed objective. A nested trust-region approach, outlined in Alg. 4, is introduced to adaptively compute the restriction parameter Δ_σ. The procedure outlined for the derivation of Eq. 3.86 yields an expected reduction in feasibility metrics $ER_f = \frac{\partial f}{\partial \sigma}(\sigma_{p+1} - \sigma_p)$: the nested trust-region approach consists in the comparison of the expected and actual feasibility reductions (respectively ER_f and AR_f). This comparison is carried out in an outer loop, where the penalty parameter σ is updated according to Eq. 3.86, while the restriction parameter Δ_σ is updated following the same logic as Eq. 3.75. The approach is robust to poor initial guesses for the penalty parameter σ_0 and removes all tuning efforts, at the cost of major computational overhead due to the nested trust-region loops, greatly affecting the HDDP runtime performance.

Algorithm 4 Nested trust-region approach for the adaptive penalty parameter tuning

```
Initialize $\sigma,\ \Delta_\sigma$
while $|\frac{ER_f}{AR_f} - 1| \geq \epsilon_1$ do
  while Eq. 3.74 not satisfied do
    Perform backward induction (also including the constraint violation expansions)
    Perform forward pass
    Compute $\rho$ through Eq. 3.73
  end while
  Compute $\frac{ER_f}{AR_f} - 1$
  Update $\Delta_\sigma$ according to Eq. 3.75
  Compute $\sigma$ according to Eq. 3.86
end while
```

Heuristic approach

An additional method to compute penalty parameter updates is also considered. The method is described in Reference [42] and implements a heuristic rule to adaptively increase the penalty parameter σ when consecutive iterates result in increased constraint violations. The update law is:

$$\begin{aligned} h &= \frac{1}{M}\sum_{i=1}^{M}\left[\sum_{k=1}^{N_i}\left(L_i\left(t_k, \boldsymbol{x}_{i,k}, \boldsymbol{u}_{i,k}, \boldsymbol{w}_i\right)\right) + \varphi_i\left(\boldsymbol{x}_-, \boldsymbol{w}_-, \boldsymbol{x}_+, \boldsymbol{w}_+\right)\right], \\ \sigma_{p+1} &= \max\left(\min\left(\frac{h}{2f^2},\ \kappa_\sigma\sigma_p\right),\ \sigma_p\right) \end{aligned} \tag{3.87}$$

where h represents a metric for optimality, and the subscripts p and $p+1$ indicate consecutive HDDP iterations. It is noticed that the provided update rule only increases the σ values, as it is aimed at steering the solution toward feasibility when the algorithm starts prioritizing optimality, and the authors themselves point out its unfeasibility to adaptively balance optimality and feasibility according to the user-specified tolerances [42].

3.2.8. Quadratic model tolerance relaxation

The algorithm sensitivity to the quadratic model validity threshold ϵ_1 is also addressed in this study. The sensitivity is addressed in two steps, one where an adaptive enlargement of the validity region is applied to avoid ”stalling” trust-region iterates (outlined in Alg. 5), and another where the ϵ_1 threshold is relaxed after convergence to improve solution convergence with minor runtime effort.

Adaptive enlargement of the validity threshold

The trust region radius update in Eq. 3.75 depends on several parameters, including a minimum value for the trust region radius $\Delta_{\min}$. This parameter prevents the trust region from ”stalling” on ineffective iterates. There are situations, however, where the defined minimum trust-region radius $\Delta_{\min}$ does not ensure the satisfaction of the validity threshold ϵ_1. To avoid force-interrupting the optimization, a safeguard is introduced. Across each trust-region trial iterate, the algorithm keeps track of the trust-region radius Δ corresponding to the most accurate quadratic model: if a stalling point is encountered (i.e., no trial iterates manage to satisfy the validity tolerance ϵ_1 with a $\Delta \geq \Delta_{\min}$), the stored radius Δ is used, and the validity tolerance ϵ_1 is enlarged accordingly. Failure to converge is only detected if also the most accurate iterate cannot meet the specified $\epsilon_{1,\max}$. The algorithm is summarized in Alg. 5.

Algorithm 5 Adaptive enlargement of the quadratic model validity region

```
Initialize best $\rho_{best},\ \Delta_{best}$ pair
while Eq. 3.74 not satisfied do
    Perform backward induction
    Perform forward pass
    Compute validity $\rho$ through Eq. 3.73
    if $|\rho - 1| \leq |\rho_{best} - 1|$ then
        Update best $\rho_{best},\ \Delta_{best}$ pair
    end if
    Update trust-region radius $\Delta$ through Eq. 3.75
    if $\Delta = \Delta_{min}$ then
        Compute $\epsilon_1 = |\rho_{best} - 1|$
        if $\epsilon_1 \leq \epsilon_{1,\max}$ then
            Set $\Delta = \Delta_{best}$
        else
            Interrupt HDDP process (due to trust-region stalling)
        end if
    end if
end while
```

Quadratic validity relaxation

The relaxation technique is now presented. For the quadratic model to be accurate, a small validity threshold $\epsilon_1 << 1$ is required implying $\rho \simeq 1$. Large values for the threshold ϵ_1 can cause iterates to diverge [75], while low values determine slow convergence. The following relaxation approach is based on observations derived throughout the tuning and optimization process. More specifically, early iterations demonstrate satisfying cost reductions even under strict tolerance ϵ_1 values, while exhibiting undesirable oscillations with larger tolerances ϵ_1. Conversely, later iterations are observed to significantly progress towards optimality only under larger tolerances ϵ_1. To maintain algorithm accuracy, while better exploiting the trust region in later iterations, the validity threshold ϵ_1 is relaxed according to:

$$\epsilon_1 = \min(\epsilon_1 \kappa_\epsilon,\ \epsilon_{1,\max}) \tag{3.88}$$

where $\epsilon_{1,\max}$ is a user-defined maximum value for the quadratic accuracy threshold, and κ_ϵ is the relaxation coefficient. The update is performed once convergence is detected: meeting first- and second-order optimality conditions, it is mathematically ensured that solution improvements, if any, do not significantly diverge from the identified optimum. Accepting less accurate iterates enables more effective exploitation of ”less-predictable” behavior: since the solution is already in a region of local optimality, it is guaranteed that newly generated iterates do not diverge.

3.2.9. Mesh refinement

The HDDP approach is favorable for high-dimensional OCPs, where numerous stages are required for accurate problem discretization. In general, increasing the number of stages leads to higher resolution but also longer runtime. Adaptive meshing techniques are used in some direct optimization approaches [56, 57] to reduce the number of stages, significantly improving runtime performance. This work adopts a simple mesh refinement procedure, outlined in Alg. 6, to tackle very large OCPs.

The devised approach solves problems with sequentially increasing resolution until the optimality improvement I_{opt} obtained from increasing the resolution matches the defined optimality tolerance ϵ_{opt}. The chosen refinement technique consists of doubling the number of stages after successful convergence. While requiring minimal implementation efforts, the chosen approach causes the number of stages to quickly rise to untractable size. Further work shall investigate more efficient strategies, such as introducing additional stages only where control inputs vary substantially (i.e., above a specified threshold) between consecutive stages. Given the independence of such an approach from the full HDDP framework, it is omitted from the software architecture illustration in Section 3.3.

Algorithm 6 Mesh refinement technique

```
Initialize discretization (number of stages), optimality improvement $I_{opt}$
while $I_{opt} \geq \epsilon_{opt}$ do
    Solve OCP through HDDP
    Double the number of stages
end while
```

3.3. Software design

The resulting HDDP algorithm is fully implemented in MATLAB ® and available in Reference [102]. The methodology and software are developed with further use as a main objective, therefore software design aims at accommodating modular changes to each sub-algorithm block, as well as guaranteeing a generic implementation, fit for a wide range of OCP applications. The defined software architecture is now illustrated, outlining the adopted design principles as well as key components of the full implementation. The integration of automatic differentiation within the framework is also presented.

3.3.1. Object-Oriented Programming

The HDDP algorithm presented in Section 3.2 incorporates numerous quantities, partial derivatives and different solvers. To ensure that the resulting implementation is readable, maintainable, and scalable, OOP principles are used, namely inheritance, abstraction, and encapsulation:

- **Inheritance**: allows a class to inherit properties and behaviors from another (super)class, enabling code reuse and hierarchical relationships. It is used to expand algorithm blocks, allowing growing mathematical complexity without requiring unnecessary implementation efforts.
- **Abstraction**: focuses on defining essential features while hiding unnecessary details. It is exploited to separately define 'rigid' interfaces between algorithmic blocks while leaving detailed implementation to specialized classes. This procedure allows to seamlessly swap different solvers for the same task without modifying the underlying code.
- **Encapsulation**: binds data and methods that manipulate it within the same class (or its related super-class). This philosophy ensures controlled interaction between an object's internal state and the mathematical manipulations performed throughout the HDDP process.

Following such principles, each of the classes introduced below inherits from a corresponding abstract super-class, which implements the basic interfaces (i.e., properties and methods) required to reliably instantiate it within the HDDP framework. These super-classes, identified by the '`abstract_`' prefix, are implemented as MATLAB® `Abstract` classes, which implicitly require concrete classes to inherit from them and implement specialized behavior to function properly. A full set of conventions used to define the following software diagrams is now provided:

- light-blue color: abstract classes;

- light-green color: concrete classes, implementing the adjacent super-class behaviors;
- light-orange color: classes defining solution quantities, which are updated iteratively throughout the HDDP loop;
- light-gray color: methods belonging to the adjacent class
- `solve`: method which operates on its class properties;
- `perform`: method operating on external classes;
- `build`: method used to generate the required data to start the HDDP optimization;
- Unified Modeling Language (UML) required interface block: fixed interface between objects and/or methods;
- gear icon: object/method implementing one or more of the HDDP core equations presented in Section 3.2;
- solid triangle: concrete implementation of a property/method defined in the corresponding super-class;
- dashed list: class properties defined in a higher-level block of the software hierarchy;
- dashed arrows: input/output relations;
- bold arrows: algorithm information flow;
- hollow arrows: inheritance;

The full software is described at a high level, aiming at illustrating the OOP paradigms applied to its design. The detailed implementation of each class and related interfaces is accessible through the source code, available in Reference [102].

First, an overview of the class structure is provided in Figure 3.6.

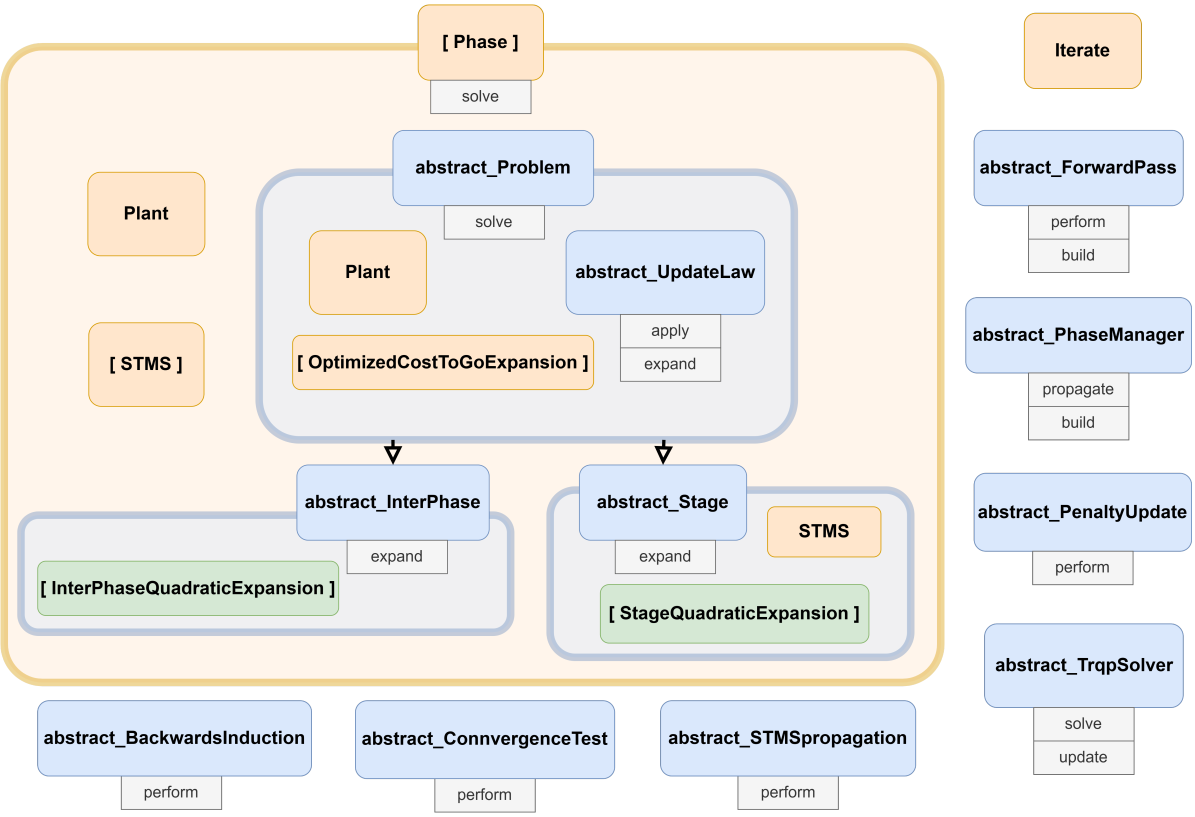


Figure 3.6: Overview of the OOP hierarchy

Class names follow a camel case convention (i.e.: ClassName), while methods are defined using snake case (i.e.: method_name). Class names are chosen to be as representative as possible. Less obvious name choices are the `Plant` class (storing the full values of the problem states $\boldsymbol{x}$, controls $\boldsymbol{u}$, parameters $\boldsymbol{w}$, and multipliers $\boldsymbol{\lambda}$), `Iterate` class (which stores the relevant metrics for the current iterate, such as augmented cost J and constraint violation f), and `PhaseManager` class (handling calls to the user-provided problem formulation).

The `Phase` class represents the algorithm solution and is updated with every algorithm iteration. By combining most of the HDDP interface quantities (i.e., quadratic expansions, STMs, plant, and update law objects), it is ensured that methods operating on the candidate solution do not accidentally modify/break interfaces with externally defined solvers (*encapsulation*).

Most high-level blocks of the HDDP procedure are implemented as individual entities, without significant dependencies to other super/sub-classes. The choice aims at providing maximum flexibility on the implementation of specific solvers. The illustrated `abstract_` classes pre-define the interfaces required for a fully operational algorithm, while their detailed implementation is left to specialized (and replaceable) classes (*abstraction*). A clear example is provided by the different `TrqpSolver` implementations considered throughout the thesis work: starting from the basic trust-region algorithm in Alg. 1, the more complex and robust version in Alg. 2 was later implemented by the `TrqpSolver_Conn` class (replacing the basic `TrqpSolver`). The integration of path constraints was achieved through the `TrqpSolver_quadraticConstrained` class: this class inherits the unconstrained TRQP sub-problem solution (used to estimate the active set of constraints) from the `TrqpSolver_Conn` class, while specialized methods are introduced to implement Eq. 3.57, Eq. 3.61, Eq. 3.62, Eq. 3.63, Eq. 3.64, and Eq. 3.70.

High-level architecture

The illustrated classes implement the high-level logic of the HDDP algorithm as in Figure 3.7.

The high-level implementation matches exactly the algorithm definition provided in Figure 3.1, increasing code interpretability. The `PhaseManager` class is the only ”additional” object (i.e., does not implement specific steps of the HDDP procedure). This class handles the interfaces between user-provided functions (i.e.: dynamics, cost function, constraints, and stage collocation) and the general-purpose HDDP solver. Further detail into the `PhaseManager` class is provided in Subsection 3.3.2.

ForwardPass class

Going into further detail into the individual classes, the main features of the `ForwardPass` class are illustrated in Figure 3.8. The dynamical feasibility of the initial guess is guaranteed by only requiring static parameters and dynamic controls, while the `ForwardPass` propagates the system dynamics. The HDDP forward-pass procedure is performed by re-initializing numerical integration at every stage (according to Alg. 3), computing a new control/parameter/multiplier update through the corresponding `UpdateLaw` object. It is noticed how the `UpdateLaw.apply` method is first defined in its corresponding super-class, allowing different implementations of such `UpdateLaw` objects. This is the case, for instance, of the `QuadraticUpdateLaw` class, which implements the update to the optimized cost-to-go through Eq. 3.70 instead of the corresponding unconstrained linear counterpart in Eq. 3.28.

StmsPropagation class

The `StmsPropagation` class implementation is provided in Figure 3.9. The `ForwardPass` provides a completely defined `Plant` object: the `StmsPropagation` can be performed in an asynchronous manner on each trajectory point, by propagating the variational equations (Eq. 3.19 and Eq. 3.20) using individual stages as initial conditions. This step is enabled by MATLAB® Parallel Computing capabilities.

BackwardsInduction class

The `BackwardsInduction` procedure is implemented as in Figure 3.10. The algorithm flow, indicated by the thick arrows, outlines the loop procedure, consisting of a backward solution of each `Phase` object. It is noticed that there is no apparent point of initialization for the backward induction procedure. To maintain the implementation as generic as possible, each phase is initialized and concluded exactly as if it were a generic inter-phase problem. Initialization is carried out on the final phase M by evaluating the inter-phase expansion in Eq. 3.24 between said phase (on the point $\boldsymbol{x}_-, \boldsymbol{w}_-$) and an ”artificial” later phase (with dummy x_+, w_+ values). Similarly, the backward induction is concluded by performing an inter-phase optimization (solving the TRQP sub-problems defined by the expansion in Eq. 3.25 and by the updated Eq. 3.30) between the initial phase $i = 1$ and a "dummy" `Phase` object with null costs. This architecture enables a unified formulation for all problem-defining functions (namely terminal costs and constraints).

Throughout the backwards induction, `Stage` and `InterPhase` objects are solved backwards. These classes implement their respective `solve` methods as illustrated in Figure 3.11 and Figure 3.12. The *inheritance* OOP paradigm is mainly observed in the different implementations of `Stage` and `InterPhase` objects. Both classes inherit properties (namely the `OptimizedCostToGoExpansion` defining the outputs of Eq. 3.283.32, `UpdateLaw`, and `Plant` objects, as well as the `expand` and `solve` methods) from the `abstract_Problem` class, ensuring compatibility with the full `Phase.solve` procedure.

The `Stage` super-class also introduces the need for `STMs` objects, required to perform the stage quadratic expansion in Eq. 3.26. The `UpdateLaw` objects are only defined as outputs of the `TrqpSolver.solve` method: depending on specific implementations, different `UpdateLaw` objects can be defined (both during problem setup as well as directly at runtime): compatibility is ensured by the required `apply` and `expand` methods, defined in the corresponding super-class. When solving OCPs with no path constraints, the `UpdateLaw.apply` method implements Eq. 3.27, Eq. 3.29, and Eq. 3.31, while `UpdateLaw.expand` defines the optimized cost-to-go expansion after the update law object is defined by the trust-region solver, through Eq. 3.28, Eq. 3.30, and Eq. 3.32.

Finally, while different quadratic expansions/cost-to-go objects are defined, it is specified that there is no limitation to the size of such objects, allowing the backward propagation of additional quantities (on top of the default cost-to-go value). This feature is leveraged when implementing the adaptive penalty parameter update: replacing the default `Stage` and `InterPhase` objects, the backward induction procedure is augmented by concatenating multiple quadratic-expansion objects (one for the cost-to-go and others for each of the constraint violation components). The previously highlighted *abstraction* and *encapsulation* paradigms imply that no further modifications to the implementation are required.

The `ConvergenceTest` class directly implements the checks defined in Eq. 3.76 and the positive-definiteness check for the Hessian matrices.

The `PenaltyUpdate` class implements all parameter updates performed after the acceptance of a trust-region trial iterate. The default implementation applies the heuristic method to update the penalty parameter (i.e., Eq. 3.87). This class can be swapped with the `PenaltyUpdatePsi` class to implement the novel update technique defined in Eq. 3.86 (with the required class-replacements to the `Stage` and `InterPhase` classes to perform the backward induction of the constraint violation partials). The `PenaltyUpdate.perform` method also implements the quadratic model validity threshold relaxation in Eq. 3.88, as well as the adaptive enlargement in Alg. 5.

3.3.2. Automatic differentiation

The OCP formulation and solution through HDDP requires the definition of multiple functions and their partial derivatives up to second-order. The HDDP algorithm, conversely from other DDP frameworks, decouples partial derivatives of the dynamics function $\boldsymbol{f}$ and cost/constraints functions φ, L, $\boldsymbol{\Psi}$, $\boldsymbol{g}$, favoring more flexibility when solving/formulating an optimization problem [42]. For the scope of this work, the flexibility of the approach is further enhanced by automatic differentiation, implemented through the ADiGator package [103] in MATLAB®. This practice proved particularly advantageous throughout the work, as it enabled quick iterations between different representations of the controls and state vector. The devised software implements all evaluations of the so-called problem-definition functions (i.e., dynamics, costs, constraints, stage collocation, and initial conditions) and their partial derivatives through the `PhaseManager` class. An overview of the `PhaseManager` class is provided in Figure 3.13.

The problem-definition functions are defined following the notation presented throughout this paper (states $\boldsymbol{x}$, controls $\boldsymbol{u}$, ...). The `PhaseManager.build` method re-arranges the user-provided functions such that they can be easily accessed by the HDDP algorithm and ADiGator routines: terminal costs φ and constraints $\boldsymbol{\Psi}$ are combined into the augmented Lagrangian cost function $\tilde{\varphi}$, the dynamics and stage collocation functions (as well as their partial derivatives) are combined into the variational equations (Eq. 3.19 and Eq. 3.20), and all functions are re-defined using the augmented state $\boldsymbol{X}$ convention.

If not already available, the build method generates all required first- and second-order partial derivatives (the red blocks in Figure 3.13) through the ADiGator routines. When evaluating partial derivatives or user-provided functions (i.e.: to perform stage/inter-phase quadratic expansions, propagate system dynamics or variational equations), the PhaseManager class manages function calls and array indexing required to access specific function partials. It is specified that the illustrated class is the ”default” implementation of the PhaseManager class, with automatically differentiated partial derivatives. The automatic differentiation framework provides easy access to first- and second-order information up to machine precision. Different approaches can also be adopted and implemented in analogous classes, with different implementations of the interfaces defined by the abstract_PhaseManager class.

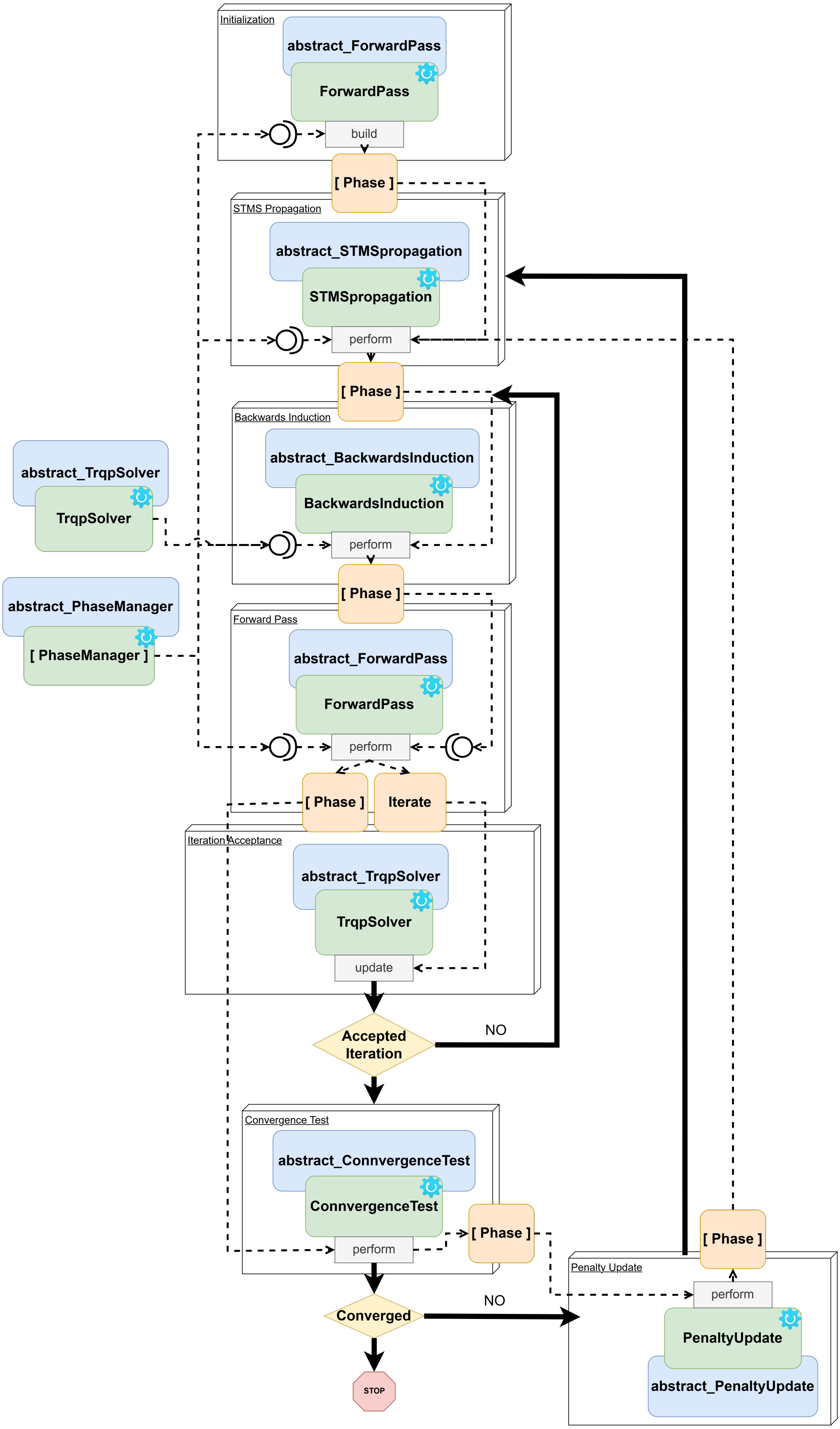


Figure 3.7: High-level overview of the HDDP algorithm implementation

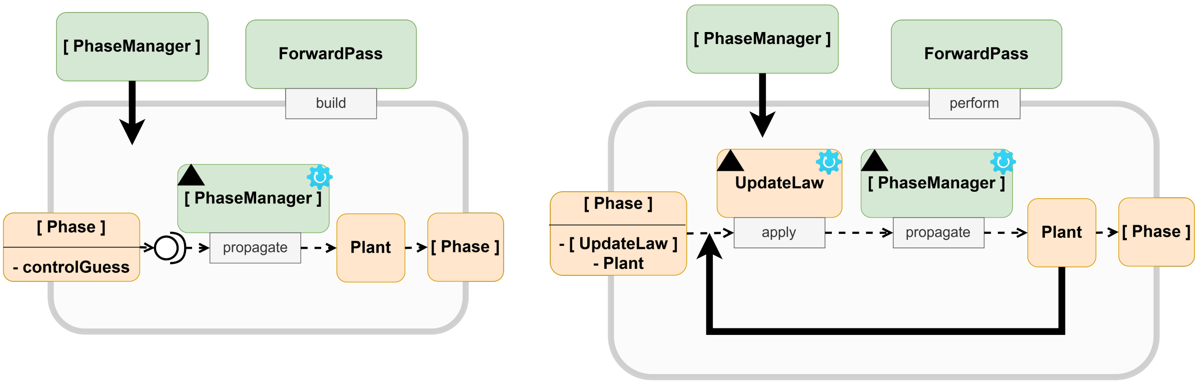


Figure 3.8: Overview of the `ForwardPass` class implementation

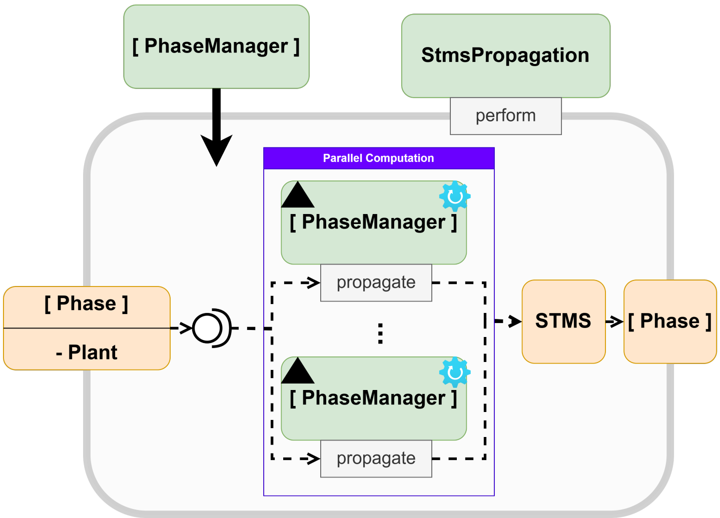


Figure 3.9: Overview of the `StmsPropagation` class implementation

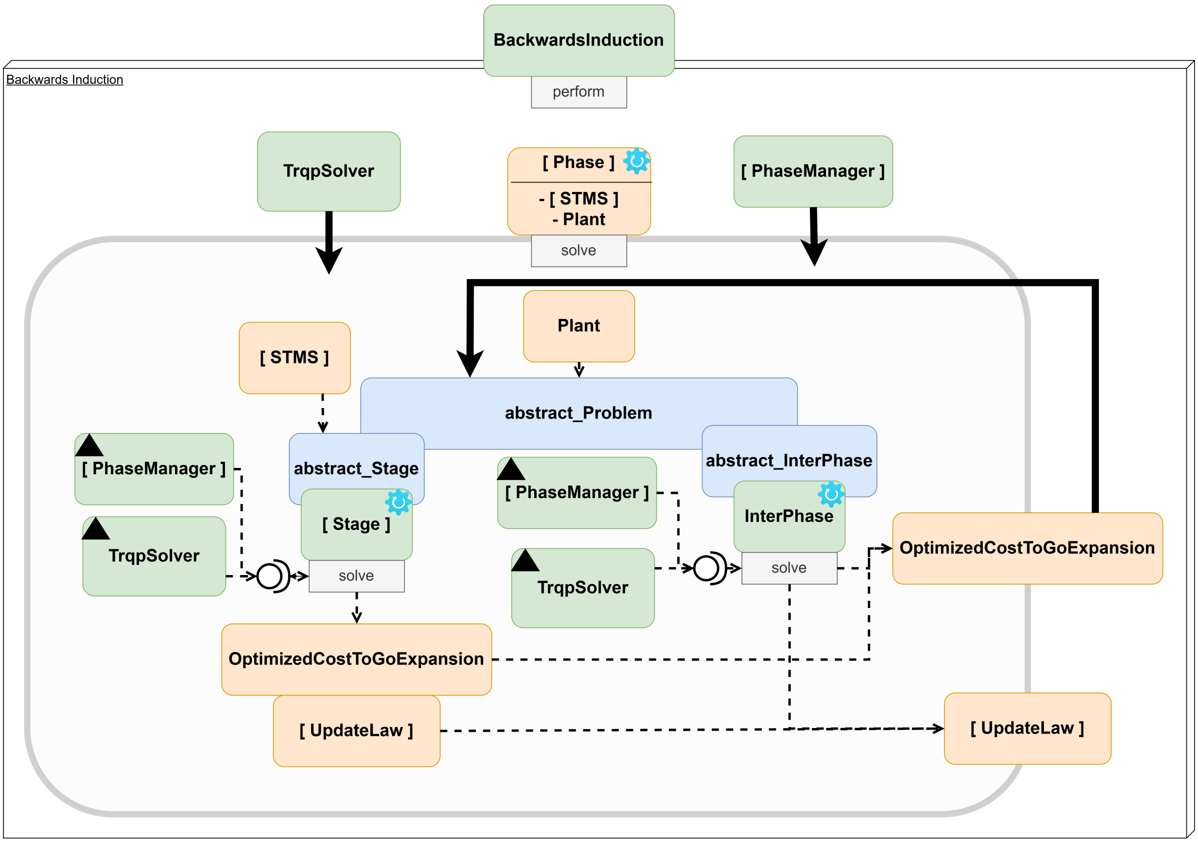


Figure 3.10: Overview of the `BackwardsInduction` implementation

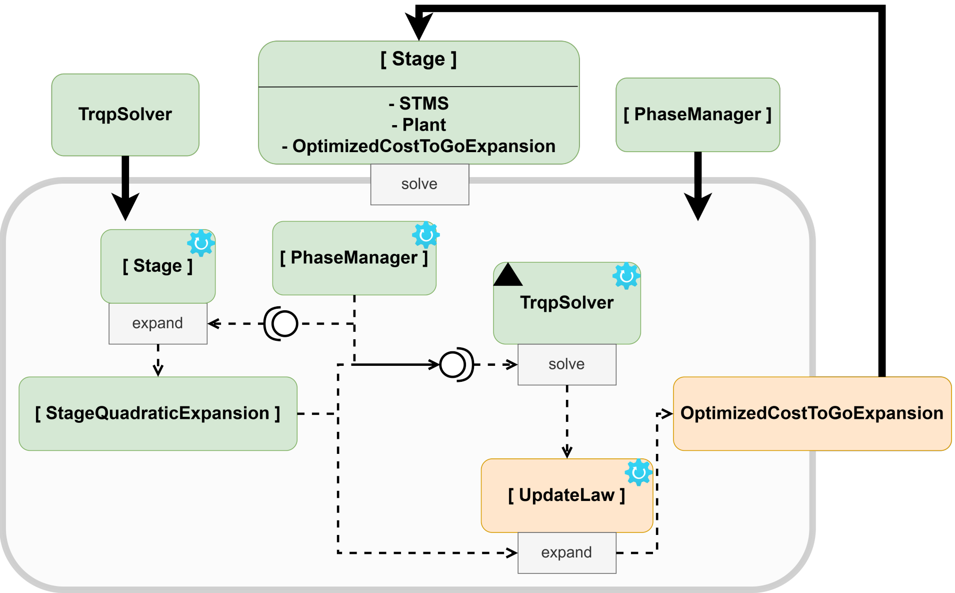


Figure 3.11: Overview of the `Stage.solve` method implementation

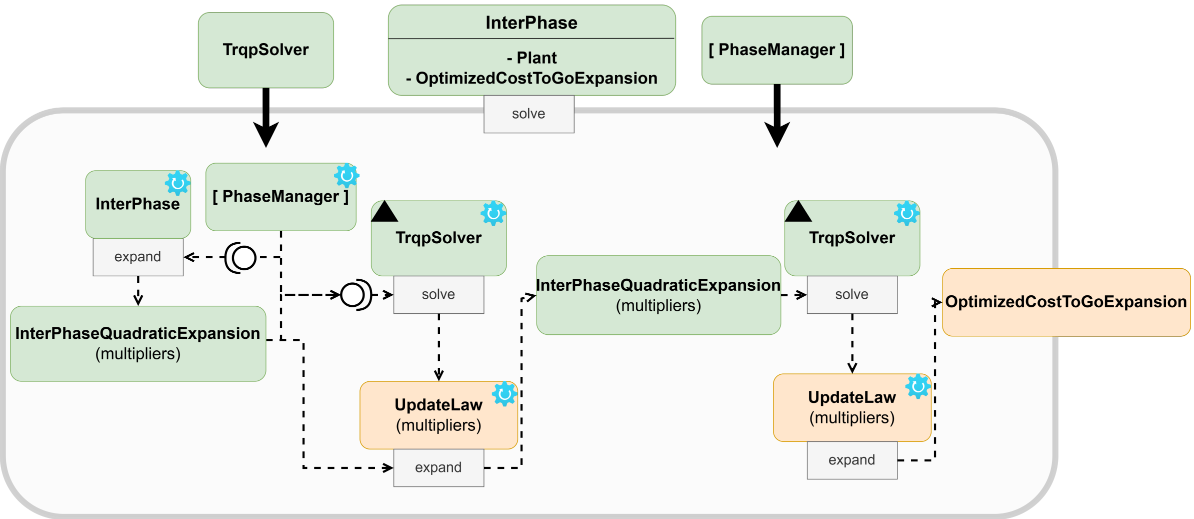


Figure 3.12: Overview of the `InterPhase.solve` method implementation

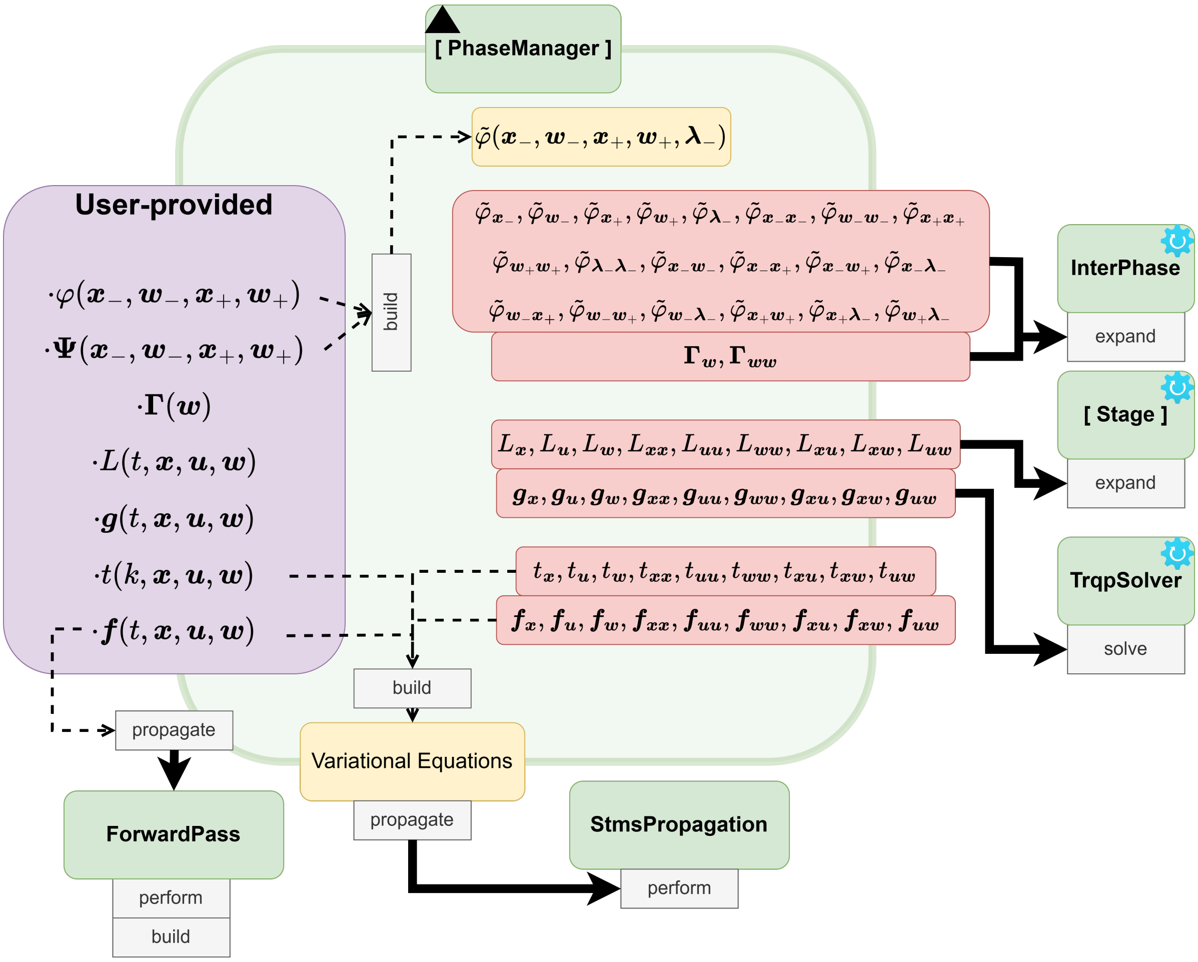

[PhaseManager]
$\tilde{\varphi}(\boldsymbol{x}_-, \boldsymbol{w}_-, \boldsymbol{x}_+, \boldsymbol{w}_+, \boldsymbol{\lambda}_-)$
User-provided
$\cdot\varphi(\boldsymbol{x}_-, \boldsymbol{w}_-, \boldsymbol{x}_+, \boldsymbol{w}_+)$
$\cdot\boldsymbol{\Psi}(\boldsymbol{x}_-, \boldsymbol{w}_-, \boldsymbol{x}_+, \boldsymbol{w}_+)$
$\cdot\boldsymbol{\Gamma}(\boldsymbol{w})$
$\cdot L(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})$
$\cdot\boldsymbol{g}(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})$
$\cdot t(k, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})$
$\cdot\boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})$
build
$\tilde{\varphi}_{\boldsymbol{x}_-}, \tilde{\varphi}_{\boldsymbol{w}_-}, \tilde{\varphi}_{\boldsymbol{x}_+}, \tilde{\varphi}_{\boldsymbol{w}_+}, \tilde{\varphi}_{\boldsymbol{\lambda}_-}, \tilde{\varphi}_{\boldsymbol{x}_-\boldsymbol{x}_-}, \tilde{\varphi}_{\boldsymbol{w}_-\boldsymbol{w}_-}, \tilde{\varphi}_{\boldsymbol{x}_+\boldsymbol{x}_+}$
$\tilde{\varphi}_{\boldsymbol{w}_+\boldsymbol{w}_+}, \tilde{\varphi}_{\boldsymbol{\lambda}_-\boldsymbol{\lambda}_-}, \tilde{\varphi}_{\boldsymbol{x}_-\boldsymbol{w}_-}, \tilde{\varphi}_{\boldsymbol{x}_-\boldsymbol{x}_+}, \tilde{\varphi}_{\boldsymbol{x}_-\boldsymbol{w}_+}, \tilde{\varphi}_{\boldsymbol{x}_-\boldsymbol{\lambda}_-}$
$\tilde{\varphi}_{\boldsymbol{w}_-\boldsymbol{x}_+}, \tilde{\varphi}_{\boldsymbol{w}_-\boldsymbol{w}_+}, \tilde{\varphi}_{\boldsymbol{w}_-\boldsymbol{\lambda}_-}, \tilde{\varphi}_{\boldsymbol{x}_+\boldsymbol{w}_+}, \tilde{\varphi}_{\boldsymbol{x}_+\boldsymbol{\lambda}_-}, \tilde{\varphi}_{\boldsymbol{w}_+\boldsymbol{\lambda}_-}$
$\boldsymbol{\Gamma}_{\boldsymbol{w}}, \boldsymbol{\Gamma}_{\boldsymbol{ww}}$
$L_{\boldsymbol{x}}, L_{\boldsymbol{u}}, L_{\boldsymbol{w}}, L_{\boldsymbol{xx}}, L_{\boldsymbol{uu}}, L_{\boldsymbol{ww}}, L_{\boldsymbol{xu}}, L_{\boldsymbol{xw}}, L_{\boldsymbol{uw}}$
$\boldsymbol{g}_{\boldsymbol{x}}, \boldsymbol{g}_{\boldsymbol{u}}, \boldsymbol{g}_{\boldsymbol{w}}, \boldsymbol{g}_{\boldsymbol{xx}}, \boldsymbol{g}_{\boldsymbol{uu}}, \boldsymbol{g}_{\boldsymbol{ww}}, \boldsymbol{g}_{\boldsymbol{xu}}, \boldsymbol{g}_{\boldsymbol{xw}}, \boldsymbol{g}_{\boldsymbol{uw}}$
$t_{\boldsymbol{x}}, t_{\boldsymbol{u}}, t_{\boldsymbol{w}}, t_{\boldsymbol{xx}}, t_{\boldsymbol{uu}}, t_{\boldsymbol{ww}}, t_{\boldsymbol{xu}}, t_{\boldsymbol{xw}}, t_{\boldsymbol{uw}}$
$\boldsymbol{f}_{\boldsymbol{x}}, \boldsymbol{f}_{\boldsymbol{u}}, \boldsymbol{f}_{\boldsymbol{w}}, \boldsymbol{f}_{\boldsymbol{xx}}, \boldsymbol{f}_{\boldsymbol{uu}}, \boldsymbol{f}_{\boldsymbol{ww}}, \boldsymbol{f}_{\boldsymbol{xu}}, \boldsymbol{f}_{\boldsymbol{xw}}, \boldsymbol{f}_{\boldsymbol{uw}}$
InterPhase
expand
[Stage]
expand
TrqpSolver
solve
build
propagate
Variational Equations
propagate
StmsPropagation
perform
ForwardPass
perform
build

Figure 3.13: Overview of the `PhaseManager` class

4

Journal Article

Differential Dynamic Programming for the Optimization of Many-Revolution Solar-Sail Transfers

Riccardo Minnozzi*
Delft University of Technology, Delft The Netherlands, 2629 HS

Fernando Gámez Losada † and Jeannette Heiligers‡
Delft University of Technology, Delft The Netherlands, 2629 HS

Solar sailing is a propellant-free propulsion method, leveraging the momentum of Sun-emitted photons to generate thrust, which it them promising for both interplanetary and Earth-bound applications. In Earth-orbit, the small magnitude of the solar-sail thrust with respect to the Earth's gravity implies the need for many revolutions to accomplish an orbital transfer. Solving the resulting optimization problem requires algorithms capable of handling very large sets of optimization variables. This study focuses on the use of Differential Dynamic Programming (DDP), expanding the algorithm formulation to handle problems with variable duration and path constraints. The resulting algorithm is validated against a state-of-the-art direct solver, and its sensitivity to hyper-parameters is investigated. The devised algorithm is applied to Earth-centered circular-to-circular planar solar-sail transfers, successfully optimizing up to 1000 revolutions transfers in LEO, and 180 revolutions in GEO. Regression models for the sail performance are derived and used to solve the circular-to-circular transfer problem through a flexible-final-time formulation. While the obtained results shed light onto optimal solar-sail many-revolution transfers, higher-fidelity dynamical models shall be considered in further analyses. The software developed as part of this work is made available for future studies, with the aim of enabling the optimization of high-fidelity mission scenarios, with variable time of flight and arbitrary operational constraints.

I. Introduction

Solar sailing is a form of spacecraft propulsion that uses the radiation pressure exerted by sunlight on large, reflective sails to generate thrust. Unlike conventional propulsion systems that rely on onboard fuel, solar sails harness the momentum of photons, offering the potential for long-duration missions with minimal resource consumption [1]. Its feasibility has been demonstrated in recent missions, in interplanetary environment by JAXA's IKAROS mission (2010)

*M.Sc. student, Aerospace engineering
†PhD Candidate, Aerospace Engineering, Astrodynamics & Space Missions
‡Associate Professor, Aerospace Engineering, Astrodynamics & Space Missions

[2], in Earth's orbit by NASA's NanoSail-D (2010) [3], the Planetary Society's LightSail 1 and 2 (2015, 2019) [4, 5], and NASA's ACS3 missions [6].

While solar sailing in the interplanetary environment has been widely explored [7–9], its unique features establish it as a promising propulsion method for planet-centered applications. Ongoing research and development efforts validate this potential: solar sails have been identified as effective solutions for science missions (e.g., planetary imaging [10, 11] and exploration of Earth's magnetic tail [12]) or commercial applications [13]. Due to the urgency of the matter, solar-sail-powered Active Debris Removal (ADR) missions have also attracted considerable attention [14–16].

The dynamics of solar sails in planetary environments differ considerably from those in interplanetary conditions. In planetary orbits, the small magnitude of the attainable solar-sail thrust compared to other forces (i.e., gravity and aerodynamic perturbations) causes low orbit control authority, which is further affected by eclipsing phenomena [17]. Notably, many orbital revolutions are required to accomplish a transfer, making their design and optimization a challenging aspect of solar-sailing mission planning [18]. Multiple optimization strategies for such transfers have been explored in the literature, spanning both electric propulsion and solar sailing missions.

Direct optimization techniques leverage collocation methods to transcribe the Optimal Control Problem (OCP) into a single Non-Linear Programming (NLP) problem [19–22]. However, their computational complexity scales quadratically with the number of decision variables [23], limiting their applicability to many-revolution transfers. Indirect methods enforce Pontryagin's minimum principle (PMP) to transform the OCP into a Multi-Point Boundary Value Problem (MPBVP) [24–27]. Solving MPBVPs numerically is challenging due to their high sensitivity to initial guesses for both states and co-states [23]: The limited literature on optimal solar-sail many-revolution transfers complicates the generation of accurate initial guesses [28]. The Q-law is a heuristics-based approach that employs a Lyapunov controller to quickly generate near-optimal trajectories [18, 29, 30]: since it does not enforce nor guarantee the solution optimality, it is typically used to generate initial guesses [26, 31].

A promising alternative is Differential Dynamic Programming (DDP), which uses Bellman's optimality principle to decompose high-dimensional OCPs into smaller sub-problems solved iteratively, incurring linear performance scaling with increasing problem dimensionality [32]. The cost function is approximated up to second order around a reference trajectory, providing both first and second-order optimality information, thus ensuring better convergence properties than gradient-based solvers [33]. The state-of-the-art in DDP-based solvers is the Static Dynamic Control (SDC) algorithm, which solves multi-phase and constrained OCPs using Riccati-like equations, enabling the high-fidelity mapping of the quadratic cost model [34]. The SDC algorithm is combined with a Q-law approach for initial guesses and a time-dilation technique to handle problems with variable duration: the resulting Mystic software was successfully applied to the trajectory optimization of the DAWN [35], PSYCHE [36], and NEA Scout [9] missions. A key advancement in the development of DDP methodologies is the Hybrid Differential Dynamic Programming (HDDP) algorithm, which uses first- and second-order State Transition Maps (STMs) to propagate the quadratic cost model on a reduced number of collocation points (stages), enabling parallel computing for enhanced performance [33]. Thanks to its computational efficiency, the HDDP algorithm has drawn considerable interest for applications to many-revolution transfers, including up to 1500 revolutions time- and fuel-optimal transfers around Earth [37], rendezvous within the Circular Restricted Three Body Problem (CRTBP) [38], and optimal Earth-centered orbit raising for 500 revolutions [39]. Improvements to the HDDP framework include the expansion of a cost model to higher orders [40], the integration within a multiple-shooting framework [41], and the incorporation of semi-analytical propagation schemes based on simplified dynamical models [42, 43]. Numerous works on DDP algorithms aimed at expanding their constrained optimization capabilities [33, 41, 44, 45].

Despite the promising features, works on the HDDP algorithm still include a range of limitations, which become particularly noticeable in solar-sailing applications. First, the algorithm's sensitivity to hyper-parameter tuning is identified as its main drawback in several works [33, 37, 39, 40]. The delicate tuning process becomes particularly time-consuming when tackling high-dimensional OCPs, such as those resulting from planet-centered solar-sail transfers [23]. Second, while the augmented Lagrangian approach [44] is a well-established methodology to handle terminal constraints, the enforcement of path constraints receives less consideration in literature. Given the inherently constrained nature of solar sailing (e.g., coupling of the sail thrust direction and magnitude [1] and minimum operational altitude constraints [46]), a DDP solver shall be able to tackle generic path constraints efficiently and robustly [17]. Studies on the topic, however, are mainly limited to the enforcement of linear bounds to the control inputs [39, 45, 47], while works addressing generic constraints (i.e., state-dependent and non-linear functions) employ linearized models, inducing chattering and reducing computational efficiency [41, 48]. Finally, studies adopting the HDDP framework are limited to fixed-duration OCPs, with expansions to variable duration problems being limited to simplified dynamical models [38, 42]. Variable-duration problems are particularly relevant in solar-sailing OCPs, as these often consist of identifying solutions meeting specified constraints in the minimum amount of time [23].

This study aims at expanding the HDDP framework through original techniques to handle OCPs with path constraints and variable time of flight. Furthermore, the sensitivity to hyper-parameters is thoroughly characterized and addressed through adaptive tuning approaches. The devised algorithm is validated against a state-of-the-art direct optimization solver and applied to time-optimal solar-sail many-revolution transfers.

The paper is structured as follows. First, the modified HDDP algorithm is presented in Section II, introducing the novel methodologies. The benchmark solar-sail-transfer problem is outlined in Section III. The algorithm behavior is characterized in Section IV, both in terms of hyper-parameter sensitivity as well as convergence properties. Relevant results are summarized in Section V, starting from simplified test problems and then moving to computationally challenging many-revolution transfers. Finally, conclusions and recommendations for future work are presented in Section VI.

II. Differential Dynamic Programming

This section presents the designed DDP implementation. The devised algorithm builds upon the HDDP framework defined in Ref. [33], as it provides an efficient and flexible solution method. The full algorithm is intended for further use as a general-purpose solver, designed to tackle a wide range of OCPs (solving for both dynamic controls and static parameters across multiple phases, accommodating constrained, FFT formulations). The reader is referred to Ref. [33] for the complete background on HDDP, while only a high-level overview is provided here.

The HDDP algorithm requires a discrete problem formulation, as well as first- and second-order partial derivatives for all problem definition functions (namely dynamics, costs, and constraints). Each discretization point is referred to as a stage. The procedure is summarized in Fig. 1, while a high-level overview of each step is presented below.

- **Initialization**: a guess for the control history and static parameters is used to forward-propagate the system dynamics, resulting in a first reference solution.
- **STMs propagation**: first- and second-order STMs between each consecutive stage of the reference solution are obtained. This work uses variational equations to perform this step, as it is a generic and widely accepted methodology [49, 50].
- **Backward induction**: this step is initialized by computing a quadratic cost model around the final stage of the reference trajectory. The resulting Quadratic Programming (QP) problem is solved, obtaining an optimal control law (in the form of state-dependent feedback [51]). The feedback law and the pre-computed STMs, are used to iteratively map the quadratic model of each stage to its predecessor until the first trajectory stage is encountered. A trust-region algorithm is used to solve each QP problem (hence referred to as Trust-Region Quadratic Programming (TRQP) problem), by artificially modifying the quadratic model and restricting control updates within a certain (trust) region. The trust-region restriction is controlled by the trust-region radius Δ. In this work, the adopted trust-region solver is the Hessian-shift Algorithm 7.3.4 from Ref. [52].
- **Forward pass**: the optimal feedback laws are applied in a forward-propagation of the reference trajectory. The resulting cost reduction is compared to the one predicted from the backward induction: if the values match up to a pre-defined tolerance ϵ_1 (quadratic model tolerance), the iterate is deemed acceptable, otherwise, the trust region is adjusted accordingly, and the backward induction re-starts. The HDDP algorithm enables the re-use of the pre-computed STMs around the reference trajectory, considerably speeding up the process with respect to other DDP frameworks (e.g., the SDC algorithm [34]). The trust-region radius update defines bounds to the trust-region radius Δ_{min} and Δ_{max} to avoid ”stalling” on ineffective or over-confident iterates.
- **Convergence test**: a convergence test is performed by evaluating both first- and second-order optimality conditions. First-order optimality is checked on the expected cost reduction registered by accepted iterates (i.e., those that meet the ϵ_1 threshold), conversely from Ref. [33] where the convergence test is performed before ensuring the validity of the current iteration. Second-order optimality is checked through the positive definiteness of the Hessian

matrices at every stage. The solution is controlled by the optimality and feasibility thresholds, ϵ_{opt} and ϵ_{feas}.

- **Parameter updates**: if the solution does not meet optimality conditions, the penalty parameter σ is updated to ensure convergence to a feasible solution, and the procedure starts from a new computation of the STMs.

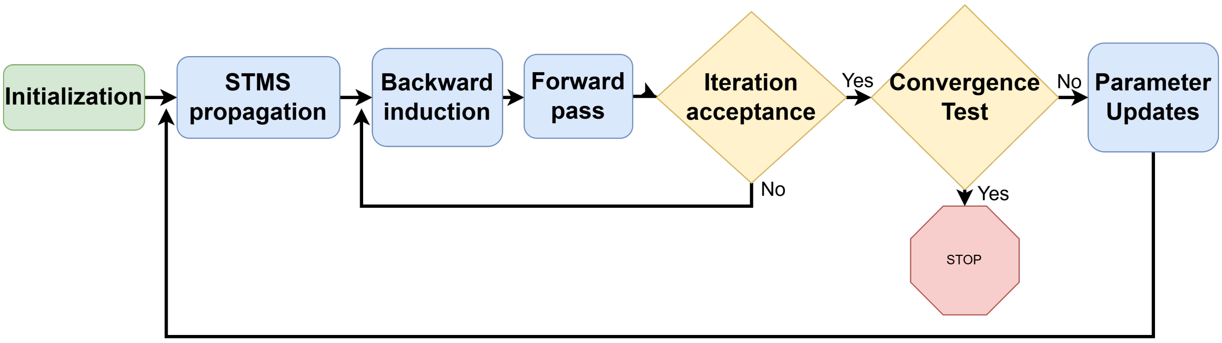


Fig. 1 High-level overview of the HDDP algorithm

The full algorithm is implemented in MATLAB® and made available through Ref. [53], following object-oriented programming paradigms to ensure that the software is maintainable and flexible. The devised implementation exploits automatic differentiation (through the ADiGator package [54]) to enable quick iterations between different problem formulations, by automatically computing exact first- and second-order partial derivatives for the problem-definition functions. The STMs propagation step is carried out in parallel through MATLAB® Parallel Computing functionalities.

The general OCP formulation considered for this work is now provided, introducing key terms, notation, and conventions. The provided formulation represents a discrete, multi-phase OCP, with parametrized initial conditions, subject to both path and terminal constraints. Terminal constraints are adjoined to the terminal costs using an augmented Lagrangian approach [44], hence the complete formulation:

$$\text{find } \boldsymbol{u}_{i,k}, \boldsymbol{w}_i = \text{argmin } J,\ \boldsymbol{\lambda}_i = \text{argmax } J,\ \forall i = 1, \dots, M,\ \forall k = 1, \dots, N_i + 1$$

$$J = \sum_{i=1}^{M} \left[\tilde{\varphi}_i(\boldsymbol{x}_{i,N_i+1}, \boldsymbol{w}_i, \boldsymbol{x}_{i+1,1}, \boldsymbol{w}_{i+1}, \boldsymbol{\lambda}_i, \sigma) + \sum_{k=1}^{N_i+1} L_i(t_{i,k}, \boldsymbol{x}_{i,k}, \boldsymbol{u}_{i,k}, \boldsymbol{w}_i) \right] \tag{1a}$$

$$\tilde{\varphi}_i(\boldsymbol{x}_{i,N_i+1}, \boldsymbol{w}_i, \boldsymbol{x}_{i+1,1}, \boldsymbol{w}_{i+1}, \boldsymbol{\lambda}_i, \sigma) = \varphi_i(\boldsymbol{x}_{i,N_i+1}, \boldsymbol{w}_i, \boldsymbol{x}_{i+1,1}, \boldsymbol{w}_{i+1}) + \boldsymbol{\lambda}_i^T \boldsymbol{\Psi}_i(\boldsymbol{x}_{i,N_i+1}, \boldsymbol{w}_i, \boldsymbol{x}_{i+1,1}, \boldsymbol{w}_{i+1}) \tag{1b}$$

$$+ \sigma \boldsymbol{\Psi}_i^T(\boldsymbol{x}_{i,N_i+1}, \boldsymbol{w}_i, \boldsymbol{x}_{i+1,1}, \boldsymbol{w}_{i+1}) \boldsymbol{\Psi}_i(\boldsymbol{x}_{i,N_i+1}, \boldsymbol{w}_i, \boldsymbol{x}_{i+1,1}, \boldsymbol{w}_{i+1}),$$

such that:

$$\boldsymbol{\Gamma}_i(\boldsymbol{w}_i) = \boldsymbol{x}_{i,1} \tag{1c}$$

$$\dot{\boldsymbol{x}}_i = \frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}_i(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) \tag{1d}$$

$$\boldsymbol{u} \in \mathcal{U} \tag{1e}$$

$$\boldsymbol{w} \in \mathcal{W} \tag{1f}$$

$$\boldsymbol{g}_i(t,\boldsymbol{x},\boldsymbol{u},\boldsymbol{w}) = 0,\ \forall i = 1,...,M \tag{1g}$$

$$\boldsymbol{\Psi}_i(\boldsymbol{x}_{i,N_i+1},\boldsymbol{w}_i,\boldsymbol{x}_{i+1,1},\boldsymbol{w}_{i+1}) = 0,\ \forall i = 1,...,M \tag{1h}$$

The i and k indexes subscripts indicate a generic phase (out of the M total phases) and a generic stage (out of the total $N_i + 1$ stages in a phase i), respectively. Hereinafter, for conciseness, the generic phase i subscript is omitted for relations that hold for all phases. The problem decision variables are the dynamic controls $\boldsymbol{u}$ and static parameters $\boldsymbol{w}$, restricted to the admissible set of controls $\mathcal{U}$ and parameters $\mathcal{W}$, respectively. Lagrange multipliers $\boldsymbol{\lambda}$ are adjoined to the terminal constraint violations $\boldsymbol{\Psi}$ and solved through a "min-max" approach, where optimal decision variables are computed together with maximal multipliers [55]. The initial conditions are parametrized through the function $\boldsymbol{\Gamma}$. $\dot{\boldsymbol{x}} = \frac{dx}{dt} = \boldsymbol{f}$ defines the dynamical evolution of the system over the independent variable t (which typically represents time, although advantageous formulations can be obtained by adopting a different independent variable [37]). The objective is the minimization of cost functional J, defined as a sum of running costs L and terminal costs φ: following the augmented Lagrangian formulation, the augmented terminal cost is indicated as $\tilde{\varphi}$. Finally, each stage is subject to the path constraints $\boldsymbol{g}$.

For notational simplicity, the augmented state $\boldsymbol{X}$ notation is used for the propagation of the STMs, where the augmented state vector is defined as $\boldsymbol{X} = \begin{bmatrix}\boldsymbol{x}^{\mathrm{T}} & \boldsymbol{u}^{\mathrm{T}} & \boldsymbol{w}^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$. To propagate stage-wise STMs, the HDDP algorithm assumes constant controls over a single stage [33], hence the augmented state derivative is introduced as $\boldsymbol{f}(t,\boldsymbol{X}) = \begin{bmatrix}\boldsymbol{f}(t,\boldsymbol{x},\boldsymbol{u},\boldsymbol{w}) & \boldsymbol{0}^{n_u\times 1} & \boldsymbol{0}^{n_w\times 1}\end{bmatrix}^{\mathrm{T}}$. The partial derivatives convention follows the one in Ref. [33], with bold subscripts indicating the differentiation variable (double subscripts indicate two-times differentiation). Sizes of each quantity are indicated through n with related subscript (i.e., the size of $\boldsymbol{x}$ is n_x).

A. Flexible-final-time handling

Optimal control solvers typically tackle variable-duration OCPs through time-dilation, consisting in a parametrization of the problem duration [34, 56]. Introducing time-dilation within the HDDP framework requires a technique to integrate the problem discretization as part of the decision process: Ref. [47] adopts a first-order truncation of the dynamics to estimate the number of stages to add or remove from the reference solution, while in Ref. [38, 42] the effects of variable duration are equally distributed among the discretization stages. The latter approach is enabled by the semi-analytical propagation schemes adopted to solve the related OCPs, where sensitivities to the time of flight can be easily computed by analytically differentiating the dynamics transition function and embedded within the STMs information [38].

This work aims at developing a generic HDDP algorithm, independent from the problem-specific dynamical model. For this reason, the approach from Ref. [38, 42] is extended to any general formulation of the system dynamics. However, The general STMs definition is inherently limited to mapping variations between fixed time instants [57], thus not accommodating time-dilation techniques. This work extends the variational equations framework to variable problem discretizations, enabling both time-dilation and adaptive mesh techniques.

The devised method starts from a generic stage collocation function $t_k = t(k, \boldsymbol{X})$, which uniquely defines the OCP discretization. Notice that the stage index k can be exploited to ”reformulate” the stage collocation approach into an adaptive step-size approach, through:

$$\frac{\partial t(k, \boldsymbol{X})}{\partial k} = \Delta t(k, \boldsymbol{X}). \tag{2}$$

The dynamics transition (i.e., the mapping of the state vector between consecutive stages) is therefore also dependent on the stage collocation function, as:

$$\boldsymbol{X}_{k+1} = \boldsymbol{X}_k + \int_{t(k, \boldsymbol{X}(t))}^{t(k+1, \boldsymbol{X}(t))} \boldsymbol{f}(t, \boldsymbol{X}(t)) dt. \tag{3}$$

Adopting such an approach enables adaptive problem discretization. Depending on the chosen stage collocation function, one can prioritize denser discretization around control switching points [48], or define a variable-duration OCP by defining the time of flight as part of the static parameter $\boldsymbol{w}$.

According to the re-formulated dynamics transition in Eq. 3, the STMs shall be adjusted to capture the dependency of the integration bounds with respect to the augmented states $\boldsymbol{X}$. Equation 3 is differentiated with respect to $\boldsymbol{X}_k$, accounting for variable integral bounds through Leibniz's rule, resulting in:

$$\begin{aligned} \frac{\partial \boldsymbol{X}_{k+1}}{\partial \boldsymbol{X}_k} = \Phi_k^1 = \mathbf{1} + \int_{t(k, \boldsymbol{X}_k)}^{t(k+1, \boldsymbol{X}_{k+1})} & \boldsymbol{f}_{\boldsymbol{X}}(t, \boldsymbol{X}(t)) \Phi^1(t) dt \\ & + \boldsymbol{f}(t(k+1, \boldsymbol{X}_{k+1}), \boldsymbol{X}_{k+1}) \left[t_{\boldsymbol{X}}(k+1, \boldsymbol{X}_{k+1}) - t_{\boldsymbol{X}}(k, \boldsymbol{X}_k) \right] \end{aligned} \tag{4}$$

$$\begin{aligned} \frac{\partial^2 \boldsymbol{X}_{k+1}}{\partial \boldsymbol{X}_k^2} = \Phi_k^2 = \int_{t(k, \boldsymbol{X}_k)}^{t(k+1, \boldsymbol{X}_{k+1})} & \left[\boldsymbol{f}_{\boldsymbol{X}}(t, \boldsymbol{X}(t)) \cdot \Phi^2(t) + \Phi^1(t)^{\mathrm{T}} \cdot \boldsymbol{f}_{\boldsymbol{X}\boldsymbol{X}}(t, \boldsymbol{X}(t)) \cdot \Phi^1(t) \right] dt \\ & + \left[\boldsymbol{f}_{\boldsymbol{X}}(t(k+1, \boldsymbol{X}_{k+1}), \boldsymbol{X}_{k+1}) \cdot \Phi^1(t(k+1, \boldsymbol{X}_{k+1})) \right] \cdot \left[t_{\boldsymbol{X}}(k+1, \boldsymbol{X}_{k+1}) - t_{\boldsymbol{X}}(k, \boldsymbol{X}_k) \right] \\ & + \dot{\boldsymbol{f}}(t(k+1, \boldsymbol{X}_{k+1}), \boldsymbol{X}_{k+1}) \left[t_{\boldsymbol{X}}(k+1, \boldsymbol{X}_{k+1}) - t_{\boldsymbol{X}}(k, \boldsymbol{X}_k) \right]^2 \\ & + \boldsymbol{f}(t(k+1, \boldsymbol{X}_{k+1}), \boldsymbol{X}_{k+1}) \left[t_{\boldsymbol{X}\boldsymbol{X}}(k+1, \boldsymbol{X}_{k+1}) - t_{\boldsymbol{X}\boldsymbol{X}}(k, \boldsymbol{X}_k) \right] \end{aligned} \tag{5}$$

where Φ_k^1 and Φ_k^2 are the first- and second-order STMs between stages k and $k+1$. The $\cdot$ operator here defines matrix-tensor products, using the same convention as in Ref. [57]. The integral terms in Eq. 3-5, represent the ”conventional” variational equations [50], solved through numerical propagation with initial conditions:

$$\begin{aligned} & \boldsymbol{X}(t(k, \boldsymbol{X}_k)) = \boldsymbol{X}_k, \\ & \Phi^1(t(k, \boldsymbol{X}_k)) = \mathbf{1}^{n_X \times n_X}, \\ & \Phi^2(t(k, \boldsymbol{X}_k)) = \mathbf{0}^{n_X \times n_X \times n_X}. \end{aligned} \tag{6}$$

The constant terms after the integral in Eq. 4 and Eq. 5 correspond to truncated expansions of the variational equations around the integration bounds. These terms embed information regarding the variable discretization directly within the STMs propagation, thus requiring minimal changes to the overall HDDP architecture and maintaining the second-order convergence properties of the algorithm [42].

B. Path constraints enforcement

Previous works on DDP introduce path constraints only as linear bounds on the control inputs through clamping [45], control projection [39, 47], or scaling of the trust region [48]. Non-linear path constraints are tackled in Ref. [33] through linearization, while Ref. [47] adopts a penalty function to enforce state-dependent constraints: the former approach suffers from chattering and inefficiency (i.e., does not fully exploit the available trust region) [48], while the latter is known to introduce inaccuracies and/or ill-conditioning [24].

The HDDP algorithm is typically applied to long-duration OCPs, thus solving numerous stage sub-problems, thus an efficient methodology to enforce path constraints is required. Additionally, solar-sail problems are typically constrained using quadratic functions (e.g., unit-norm constraints on the control inputs [15]). To address these points, this works proposes a methodology to enforce the Karush-Khun-Tucker (KKT) conditions on a second-order expansion of the constrained TRQP sub-problem. The technique introduced in this subsection extends the quadratic approach from Ref. [58] to the HDDP framework.

The proposed methodology starts by solving the unconstrained TRQP sub-problem, yielding the QP model minimizer $\delta\boldsymbol{u}^*$ (i.e., the feed-forward control input which minimizes the unconstrained QP problem) and the shifted Hessian $J_{\boldsymbol{uu}}$. The QP model minimizer $\delta\boldsymbol{u}^*$ is exploited to estimate the active set of constraints $\boldsymbol{q}$, which then yields the KKT conditions:

$$\begin{gathered} J_{\boldsymbol{u}} + \sum_{j=1}^{n_q} \boldsymbol{\mu}^j \boldsymbol{q}_{\boldsymbol{u}}^j = 0 \\ \boldsymbol{q} = \nvdash^{n_q \times 1} \end{gathered} \tag{7}$$

where $\boldsymbol{\mu}$ is the Lagrange multipliers vector referred to the active path constraints, n_q is the size of active path constraints, and the superscript j refers to the j-th component of the specified quantity. Following Ref. [58], Eq. 7 is expanded up to second order around the current states $\boldsymbol{x}$, controls $\boldsymbol{u}$, parameters $\boldsymbol{w}$, and multipliers $\boldsymbol{\lambda}$ (note that the $\boldsymbol{\lambda}$ multipliers refer to the HDDP terminal constraints). Following Bellman's optimality principle [51], the constrained TRQP sub-problem is solved through a feedback control policy, assumed to be in the quadratic form:

$$\begin{aligned}\delta\boldsymbol{u} = \delta\boldsymbol{u}_0 + \begin{bmatrix} U_{\boldsymbol{x}} & U_{\boldsymbol{w}} & U_{\boldsymbol{\lambda}} \end{bmatrix} \begin{bmatrix} \delta\boldsymbol{x} \\ \delta\boldsymbol{w} \\ \delta\boldsymbol{\lambda} \end{bmatrix} \\ + \frac{1}{2}\sum_{a=1}^{n_x} \left(U_{\boldsymbol{x}^a\boldsymbol{x}}\delta\boldsymbol{x} + U_{\boldsymbol{x}^a\boldsymbol{w}}\delta\boldsymbol{w} + U_{\boldsymbol{x}^a\boldsymbol{\lambda}}\delta\boldsymbol{\lambda} \right) \delta\boldsymbol{x}^a \\ + \frac{1}{2}\sum_{b=1}^{n_w} \left(U_{\boldsymbol{w}^b\boldsymbol{x}}\delta\boldsymbol{x} + U_{\boldsymbol{w}^b\boldsymbol{w}}\delta\boldsymbol{w} + U_{\boldsymbol{w}^b\boldsymbol{\lambda}}\delta\boldsymbol{\lambda} \right) \delta\boldsymbol{w}^b \\ + \frac{1}{2}\sum_{c=1}^{n_\lambda} \left(U_{\boldsymbol{\lambda}^c\boldsymbol{x}}\delta\boldsymbol{x} + U_{\boldsymbol{\lambda}^c\boldsymbol{w}}\delta\boldsymbol{w} + U_{\boldsymbol{\lambda}^c\boldsymbol{\lambda}}\delta\boldsymbol{\lambda} \right) \delta\boldsymbol{\lambda}^c,\end{aligned} \tag{8}$$

where the superscripts a, b and c indicate generic indexes of the $\boldsymbol{x}$, $\boldsymbol{w}$ and $\boldsymbol{\lambda}$ vectors respectively, while the δ symbol indicates small variations with respect to the reference trajectory. A feedback law in the same form as Eq. 8 is also assumed for the Lagrange multipliers $\boldsymbol{\mu}$.

The KKT conditions are enforced by substituting the quadratic optimal policy in Eq. 8 into the second-order expansion of the KKT system in Eq. 7. The second-order expansion of Eq. 7 is omitted for conciseness, thus the reader is referred to Ref. [58] for its complete expression. Since the small variations (i.e., $\delta\boldsymbol{x}$, $\delta\boldsymbol{w}$, and $\delta\boldsymbol{\lambda}$) are unknown, the solution is retrieved by grouping together the coefficients of the corresponding variations [58]. These terms can be split into feed-forward (i.e., do not refer to any variation element), linear feedback (i.e., coefficients for a single variation), and second-order feedback terms (i.e., coefficients for second-order variations). Grouping each of the components results in a set of algebraic systems of equations [58]. Notice that this classification corresponds exactly to the terms in Eq. 8, which includes the feed-forward term $\delta\boldsymbol{u}_0$, the linear feedback terms $U_{\boldsymbol{x}}, \ldots$ and the quadratic feedback terms $U_{\boldsymbol{x}^a\boldsymbol{x}}, \ldots$, thus justifying the assumed form of the optimal control policy [58].

For notational clarity, variable indexes in the following equations are defined using superscripts, following the convention $\boldsymbol{\mu}^k_{\mathbf{x}^j} = \frac{\partial \mu^k}{\partial \mathbf{x}^j}$, and $U_{\mathbf{x}^j\mathbf{x}} = \frac{\partial^2 U}{\partial \mathbf{x}^j \partial \mathbf{x}}$ [58]. The system resulting from the feed-forward terms is:

$$0 = \begin{bmatrix} J_{\mathbf{u}} \\ q^1 \\ \vdots \\ q^{n_q} \end{bmatrix} + (\mathcal{M} + \frac{1}{2}\mathcal{H}) \begin{bmatrix} \delta \mathbf{u}_0 \\ \boldsymbol{\mu} \end{bmatrix}, \text{ where:} \tag{9a}$$

$$\mathcal{M} = \begin{bmatrix} J_{\mathbf{uu}} & q^1_{\mathbf{u}} & \cdots & q^{n_q}_{\mathbf{u}} \\ q^{1\,\mathrm{T}}_{\mathbf{u}} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \\ q^{n_q\,\mathrm{T}}_{\mathrm{u}} & 0 & \cdots & 0 \end{bmatrix} \quad \mathcal{H} = \begin{bmatrix} \sum_{j=1}^{n_q} \boldsymbol{\mu}^j q^j_{\mathbf{uu}} & q^1_{\mathrm{uu}}\delta \mathbf{u}_0 & \cdots & q^{n_q}_{\mathrm{uu}}\delta \mathbf{u}_0 \\ \delta \mathbf{u}_0^{\mathrm{T}} q^1_{\mathrm{uu}} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \delta \mathbf{u}_0^{\mathrm{T}} q^{n_q}_{\mathrm{uu}} & 0 & \cdots & 0 \end{bmatrix}. \tag{9b}$$

The non-linear system is adjoined to the trust-region constraint (i.e., $||\delta \boldsymbol{u}_0|| \leq \Delta$) and solved through MATLAB®'s `fsolve` routine, whit a function value tolerance ϵ_{path}. The obtained affine terms $\delta \boldsymbol{u}_0$ and $\boldsymbol{\mu}$ are then used to solve the algebraic systems which yield both the linear and quadratic feedback terms for Eq. 8: being linear systems, they can be efficiently solved through MATLAB®'s \ command. For conciseness, only systems for first-order feedback terms are shown here, while second-order terms can be found in Appendix A:

$$0 = \begin{bmatrix} J_{\boldsymbol{ux}} \\ \boldsymbol{q}^{1\mathrm{T}}_{\boldsymbol{x}} \\ \vdots \\ \boldsymbol{q}^{n_q\mathrm{T}}_{\boldsymbol{x}} \end{bmatrix} + \begin{bmatrix} \sum_{j=1}^{n_q} \boldsymbol{\mu}^j \boldsymbol{q}^j_{\boldsymbol{ux}} \\ \delta \boldsymbol{u}_0 \boldsymbol{q}^1_{\boldsymbol{ux}} \\ \vdots \\ \delta \boldsymbol{u}_0 \boldsymbol{q}^{n_q}_{\boldsymbol{ux}} \end{bmatrix} + (\mathcal{M} + \mathcal{H}) \begin{bmatrix} U_{\boldsymbol{x}} \\ \boldsymbol{\mu}_{\boldsymbol{x}} \end{bmatrix} \quad 0 = \begin{bmatrix} J_{\boldsymbol{uw}} \\ \boldsymbol{q}^{1\mathrm{T}}_{\boldsymbol{w}} \\ \vdots \\ \boldsymbol{q}^{n_q\mathrm{T}}_{\boldsymbol{w}} \end{bmatrix} + \begin{bmatrix} \sum_{j=1}^{n_q} \boldsymbol{\mu}^j \boldsymbol{q}^j_{\boldsymbol{uw}} \\ \delta \boldsymbol{u}_0 \boldsymbol{q}^1_{\boldsymbol{uw}} \\ \vdots \\ \delta \boldsymbol{u}_0 \boldsymbol{q}^{n_q}_{\boldsymbol{uw}} \end{bmatrix} + (\mathcal{M} + \mathcal{H}) \begin{bmatrix} U_{\boldsymbol{w}} \\ \boldsymbol{\mu}_{\boldsymbol{w}} \end{bmatrix} \quad 0 = \begin{bmatrix} J_{\boldsymbol{u}\lambda} \\ 0 \\ \vdots \\ 0 \end{bmatrix} + (\mathcal{M} + \mathcal{H}) \begin{bmatrix} U_{\lambda} \\ \boldsymbol{\mu}_{\lambda} \end{bmatrix}. \tag{10}$$

The quadratic update law in Eq. 8 is then used to update the partial derivatives of the optimized cost-to-go J^* across the TRQP sub-problems through:

$$
\begin{aligned}
J^*_{\mathbf{x}} =& J_{\mathbf{x}} + U^{\mathrm{T}}_{\mathbf{x}} J_{\mathbf{u}} + J_{\mathbf{xu}}\delta\mathbf{u}_0 + U^{\mathrm{T}}_{\mathbf{x}} J_{\mathbf{uu}}\delta\mathbf{u}_0 \\
J^*_{\mathbf{w}} =& J_{\mathbf{w}} + U^{\mathrm{T}}_{\mathbf{w}} J_{\mathbf{u}} + J_{\mathbf{wu}}\delta\mathbf{u}_0 + U^{\mathrm{T}}_{\mathbf{w}} J_{\mathbf{uu}}\delta\mathbf{u}_0 \\
J^*_{\lambda} =& J_{\lambda} + U^{\mathrm{T}}_{\lambda} J_{\mathbf{u}} + J_{\lambda\lambda}\delta\mathbf{u}_0 + U^{\mathrm{T}}_{\lambda} J_{\mathbf{uu}}\delta\mathbf{u}_0 \\
J^*_{\mathbf{xx}} =& J_{\mathbf{xx}} + 2J_{\mathbf{xu}}U_{\mathbf{x}} + U^{\mathrm{T}}_{\mathbf{x}} J_{\mathbf{uu}}U_{\mathbf{x}} + \sum_{j=1}^{n_u} U^j_{\mathbf{xx}} \left(J_{\mathbf{u}} + J_{\mathbf{uu}}\delta\mathbf{u}_0\right)^j \\
J^*_{\mathbf{ww}} =& J_{\mathbf{ww}} + 2J_{\mathbf{wu}}U_{\mathbf{w}} + U^{\mathrm{T}}_{\mathbf{w}} J_{\mathbf{uu}}U_{\mathbf{w}} + \sum_{j=1}^{n_u} U^j_{\mathbf{ww}} \left(J_{\mathbf{u}} + J_{\mathbf{uu}}\delta\mathbf{u}_0\right)^j \\
J^*_{\lambda\lambda} =& J_{\lambda\lambda} + 2J_{\lambda\boldsymbol{u}}U_{\lambda} + U^{\mathrm{T}}_{\lambda} J_{\mathbf{uu}}U_{\lambda} + \sum_{j=1}^{n_u} U^j_{\lambda\lambda} \left(J_{\mathbf{u}} + J_{\mathbf{uu}}\delta\mathbf{u}_0\right)^j \\
J^*_{\mathbf{xw}} =& J_{\mathbf{xw}} + J_{\mathbf{xu}}U_{\mathbf{w}} + U^{\mathrm{T}}_{\mathbf{x}} J_{\mathbf{uu}}U_{\mathbf{w}} + \sum_{j=1}^{n_u} U^j_{\mathbf{xw}} \left(J_{\mathbf{u}} + J_{\mathbf{uu}}\delta\mathbf{u}_0\right)^j \\
J^*_{\boldsymbol{x}\lambda} =& J_{\boldsymbol{x}\lambda} + J_{\boldsymbol{xu}}U_{\lambda} + U^{\mathrm{T}}_{\boldsymbol{x}} J_{\boldsymbol{uu}}U_{\lambda} + \sum_{j=1}^{n_u} U^j_{\boldsymbol{x}\lambda} \left(J_{\boldsymbol{u}} + J_{\boldsymbol{uu}}\delta\boldsymbol{u}_0\right)^j \\
J^*_{\boldsymbol{w}\lambda} =& J_{\boldsymbol{w}\lambda} + J_{\boldsymbol{wu}}U_{\lambda} + U^{\mathrm{T}}_{\boldsymbol{w}} J_{\boldsymbol{uu}}U_{\lambda} + \sum_{j=1}^{n_u} U^j_{\boldsymbol{w}\lambda} \left(J_{\boldsymbol{u}} + J_{\boldsymbol{uu}}\delta\boldsymbol{u}_0\right)^j
\end{aligned}
\tag{11}
$$

When introducing path constraints, the convergence test (see Fig. 1) is adjusted accordingly. Positive definiteness is only enforced on the reduced Hessians H_R (i.e., Hessian matrices of the cost-to-go with respect to constraint-satisfying controls) [33]. The Jacobian of the active constraints is readily available, as the technique introduced here estimates the active set of constraints and computes its quadratic expansion. The reduced Hessians H_R can be computed with minimal computational overhead through $H_R = Z^{\mathrm{T}}HZ$, where Z is the null space of the active set of constraints on each stage, while H is the (unrestricted) Hessian.

C. Mesh refinement

The DDP approach is favorable for high-dimensional OCPs, where numerous stages are required for accurate problem discretization. In general, increasing the number of stages leads to higher resolution but also longer runtime. Adaptive meshing techniques are used in some direct optimization approaches [19, 56] to reduce the number of stages and improve runtime performance. This work adopts a simple mesh refinement procedure to tackle very large OCPs.

The devised approach solves problems with sequentially increasing resolution until the optimality improvement obtained from higher resolution matches the defined optimality tolerance ϵ_{opt}. By starting on a coarse mesh, the algorithm can efficiently identify low-resolution results: the solution is used to re-initialize the algorithm with a doubled number of stages. Thanks to the guarantees of local optimality of the low-resolution control policy, the refinement procedure typically consists of few iterations, minimizing the computational overhead introduced by the fine discretization. While requiring minimal implementation efforts, the chosen approach causes the number of stages to quickly rise to untractable size. Further work shall investigate more efficient strategies, such as introducing additional stages only the where control inputs vary substantially between consecutive stages.

D. Parameter updates

As previously mentioned, one of the main HDDP limitations is its sensitivity to hyper-parameters [33]. This section introduces adaptive techniques to tune the penalty parameter σ and the model validity threshold ϵ_1: the former controls the weight attributed to constraint violations introduced in the augmented cost functional, while the latter defines the desired accuracy of the second-order cost model used in the HDDP procedure.

1. Adaptive penalty parameter update

Penalty methods and, to a lesser extent, augmented Lagrangian approaches, introduce sensitivity to the chosen penalty parameter σ [44]. While hand-tuning is possible [37, 39], extensive trial and error is required to achieve acceptable balance between solution optimality and feasibility. Trial and error iterations become very time-consuming in the context of high-dimensional OCPs (such as many-revolutions transfers). Reference [33] introduces an heuristics-based approach to perform the penalty parameter update, as:

$$\sigma_{p+1} = \max\left(\min\left(\frac{h}{2f^2},\ \kappa_\sigma \sigma_p\right),\ \sigma_p\right), \tag{12}$$

where h is a metric for optimality (refer to Ref. [33] for the exact formulation), while p and $p+1$ represent consecutive HDDP iterations. It is noticed that the provided update rule only increases the σ values, as it is aimed at steering the solution toward feasibility when the algorithm excessively prioritizes optimality [33]. This work introduces an adaptive strategy to adjust the penalty parameter σ, with the goal of guaranteeing solutions that closely meet the desired feasibility threshold ϵ_{feas}, hence providing accurate control over the balance between optimality and feasibility through the corresponding tolerance values.

The chosen approach exploits the backward induction procedure to propagate a quadratic model of the constraint violations. The stage and inter-phase update equations (defined in Ref. [33] and expanded in Eq. 11 for path-constrained problems) implement the mapping of cost function partials after solving a TRQP sub-problem: the same procedure is adopted for the backward propagation of the partial derivatives of the constraint violations. For notational clarity, the quantities $\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{x}^T & \boldsymbol{w}^T & \boldsymbol{\lambda}^T \end{bmatrix}^T$ and $\boldsymbol{\Psi} = \begin{bmatrix} \boldsymbol{\Psi}_1^T & \dots & \boldsymbol{\Psi}_M^T \end{bmatrix}^T$ are introduced. Leveraging the chain rule, the relation between constraint violations and penalty parameter can be expressed as:

$$\frac{\partial \boldsymbol{\Psi}}{\partial \sigma} = \frac{\partial \boldsymbol{\Psi}}{\partial \boldsymbol{Y}} \frac{\partial \boldsymbol{Y}}{\partial J} \frac{\partial J}{\partial \sigma}. \tag{13}$$

The term $\frac{\partial \boldsymbol{\Psi}}{\partial \boldsymbol{Y}}$ is the sensitivity of the constraint violations with respect to the quantities $\boldsymbol{Y}$ after the complete backward induction. The sensitivity terms of each constraint violation vector $\boldsymbol{\Psi}_i$ are retrieved once the backward induction of the corresponding phase i is completed, and are indicated as $\boldsymbol{\Psi}_i^*$. Analogously, sensitivities of the optimized cost-to-go $J_{\boldsymbol{Y}}^*$ are retrieved after every inter-phase problem and used to compute the $\frac{\partial \boldsymbol{Y}}{\partial J}$ term through the inverse-function-differential rule. The term $\frac{\partial J}{\partial \sigma}$ is obtained by differentiating the augmented Lagrangian formulation in Eq. 1b. Combining the terms we obtain:

$$\begin{aligned}
\frac{\partial \boldsymbol{\Psi}}{\partial \boldsymbol{Y}} &= \boldsymbol{\Psi}_{\boldsymbol{Y}}^*, \\
\frac{\partial \boldsymbol{Y}}{\partial J} &= \frac{1}{\frac{\partial J}{\partial \boldsymbol{Y}}(\boldsymbol{Y}(J))} = \frac{1}{J_{\boldsymbol{Y}}^*}, \\
\frac{\partial J}{\partial \sigma} &= ||\boldsymbol{\Psi}||^2, \\
\frac{\partial \boldsymbol{\Psi}}{\partial \sigma} &= \boldsymbol{\Psi}_{\boldsymbol{Y}}^* \frac{1}{J_{\boldsymbol{Y}}^*} ||\boldsymbol{\Psi}||^2.
\end{aligned} \tag{14}$$

Then, the feasibility metric f is defined according to Ref. [33] through:

$$f := \sqrt{\frac{1}{M} \sum_{i=1}^{M} \left[\left|\left| \, \boldsymbol{\Psi}_i \left(\boldsymbol{x}_{i,N_i+1}, \boldsymbol{w}_i, \boldsymbol{x}_{i+1,1}, \boldsymbol{w}_{i+1} \right) \right|\right|^2 \right]}. \tag{15}$$

Its sensitivity with respect to the constraint violations is derived analytically as:

$$\frac{\partial f}{\partial \boldsymbol{\Psi}} = \frac{\boldsymbol{\Psi}^{\boldsymbol{T}}}{\sqrt{M}||\boldsymbol{\Psi}||} \tag{16}$$

Leveraging the chain rule, Eq. 14 and Eq. 16 are combined, yielding:

$$\frac{\partial f}{\partial \sigma} = \frac{\boldsymbol{\Psi}^{\boldsymbol{T}}}{\sqrt{M}||\boldsymbol{\Psi}||} \boldsymbol{\Psi}_{\boldsymbol{Y}}^* \frac{1}{J_{\boldsymbol{Y}}^*} ||\boldsymbol{\Psi}||^2 = \frac{\boldsymbol{\Psi}^{\boldsymbol{T}} \boldsymbol{\Psi}_{\boldsymbol{Y}}^* ||\boldsymbol{\Psi}||}{\sqrt{M} J_{\boldsymbol{Y}}^*} \tag{17}$$

The sensitivity resulting from Eq. 17 is used to update the penalty parameter σ according to:

$$\sigma_{p+1} = \sigma_p + \min \left(\max \left(\frac{\epsilon_{feas} - f_p}{\frac{\partial f}{\partial \sigma}} \kappa_\sigma, -\Delta_\sigma \right), \Delta_\sigma \right) \tag{18}$$

where p is an algorithm iteration and ϵ_{feas} is the desired value of the feasibility metric. Δ_σ and κ_σ are user-defined parameters: the Δ_σ limits the update to maintain the validity of the model in Eq. 16, while κ_σ introduces a margin to the "target" feasibility value to prevent iterations from chattering around the specified threshold ϵ_{feas}. Matching the HDDP logic, the devised penalty update is only performed if the expected reduction in constraint violation matches the actual reduction registered during the forward pass. Since the penalty update is only performed on trust-region iterates where the quadratic cost model is deemed accurate, this check is often successful.

The adaptive update rule in Eq. 18 requires the definition of a restriction parameter Δ_σ: while it is ensured that small values yield accurate updates of the penalty parameter, larger values favor convergence speed: to avoid the time-consuming tuning process, a nested trust-region approach is introduced and outline in Alg. 1.

The procedure outlined for the derivation of Eq. 18 yields an expected reduction in feasibility metrics $ER_f = \frac{\partial f}{\partial \sigma}(\sigma_{p+1} - \sigma_p)$: the nested trust-region approach consists in the comparison of the expected and actual reductions in the feasibility metric (respectively ER_f and AR_f). This comparison is carried out in an outer loop, where the penalty parameter σ is updated according to Eq. 18, while the restriction parameter Δ_σ is updated according to the trust-region logic (summarized in Eq. 20) of increasing the value after successful iterates and reducing it after unsuccessful ones. The approach is robust to poor initial guesses for the penalty parameter σ and removes all tuning efforts, at the cost of major computational overhead due to the nested trust-region loops, greatly affecting the HDDP runtime performance.

Algorithm 1 Nested trust-region approach for the adaptive penalty parameter tuning

```
Initialize $\sigma$, $\Delta_\sigma$
while $|\frac{ER_f}{AR_f} - 1| \geq \epsilon_1$ do
    while $|\rho - 1| \geq \epsilon_1$ do
        Perform backward induction (also including the constraint violation expansions)
        Perform forward pass
        Compute $\rho$ as the ratio between expected and actual cost reductions
    end while
    Compute $\frac{ER_f}{AR_f} - 1$
    Update $\Delta_\sigma$ according to the logic in Eq. 20
    Compute $\sigma$ according to Eq. 18
end while
```

2. Relaxation of the quadratic model validity threshold

The accuracy of the quadratic model is defined by the validity threshold ϵ_1, with small values $\epsilon_1 << 1$ being required to guarantee robust convergence and avoiding diverging iterates (i.e., iterates that ”escape” the local search space) and large values favoring faster convergence [52]. This work introduces a relaxation technique for the accuracy threshold ϵ_1, aimed at improving the robustness of the solution to the user-defined parameter value. The approach is based on observations derived during the parameter tuning process. More specifically, early iterations showed satisfying cost reductions even under strict ϵ_1 values, while exhibiting partially diverging iterates (i.e., the control updates escaping the local search space on some trajectory stages, resulting in undesirably oscillating/jittery optimal control profiles) with larger ϵ_1 tolerances. Conversely, later iterations progressed optimality only under larger ϵ_1 values. To maintain the accuracy of the algorithm, while better exploiting the trust region in later iterations, the validity threshold ϵ_1 is relaxed according to:

$$\epsilon_1 = \min(\epsilon_1 \kappa_\epsilon,\ \epsilon_{1,\max}) \tag{19}$$

where $\epsilon_{1,\max}$ is a user-defined maximum value for the quadratic accuracy threshold and κ_ϵ is the user-provided relaxation coefficient (with $\kappa_\epsilon \geq 1$). The update is performed once convergence is detected: meeting first- and second-order optimality conditions, it is mathematically ensured that the improvements to the solution, if any, do not significantly diverge from the identified optimum, while accepting less accurate iterates enables a more effective exploitation of ”less-predictable” behavior.

3. Additional safeguards

The HDDP algorithm relies on a trust region procedure to sequentially optimize quadratic models of the cost function. The quadratic model validity threshold ϵ_1 is used to update the trust region radius Δ according to:

$$\Delta_{p+1} = \begin{cases} \min\left((1+\kappa_d)\Delta_p, \Delta_{\max}\right) & \text{if } |\rho - 1| \leq \epsilon_1 \\ \max\left((1-\kappa_d)\Delta_p, \Delta_{\min}\right) & \text{otherwise} \end{cases} \tag{20}$$

where p indicates an algorithm iteration, ρ is the ratio between expected and actual cost reductions, κ_d is a user-defined parameter for the trust region radius update, and $\Delta_{\min}$ is the minimum trust region radius value [33]. Equation 20 restricts the trust region for rejected iterates and expands it after an iterate is accepted (the non-monotone behavior of the trust-region radius is known to improve convergence speed [59]).

The trust-region radius update in Eq. 20 depends on several parameters, including the minimum value $\Delta_{\min}$, which prevents the trust-region loop from ”stalling” on ineffective iterates. A premature interruption of the optimization is triggered in cases where the quadratic cost model can meet the desired accuracy threshold ϵ_1 only by applying insignificant changes (limited by the trust-region radius $\Delta_{\min}$) to its control law, typically yielding no effect on the optimality (i.e., resulting in ”ineffective iterates”). The interruption indicates that the chosen validity threshold ϵ_1 is too strict, thus requiring additional tuning efforts.

To avoid the unnecessary interruption of the optimization, a safeguard is introduced, as outlined in Alg. 2. Around a specified reference solution, the trust-region loop keeps track of the most accurate quadratic cost model (i.e., the trust-region radius Δ which results in the accuracy metric ρ being closest to 1). The corresponding parameter values are stored as Δ_{best} and ρ_{best}. If a premature optimization is encountered (i.e., too strict validity tolerance ϵ_1) the validity tolerance ϵ_1 is enlarged to accommodate the recorded ”most accurate cost model”. The enlargement of the validity region is limited by the user-provided maximum value $\epsilon_{1,\max}$. Following this approach, the algorithm is ensured to maintain the user-provided value for the validity tolerance ϵ_1, ensuring the robust convergence of the trust-region method [52], while only increasing the tolerance value if required. Definitive failure to converge is detected only once the quadratic model cannot satisfy even the largest allowed validity region $\epsilon_{1,\max}$.

III. Solar-sail transfer problem

The devised algorithm is applied to solar-sail orbital transfers. Due to the lack of literature on optimal solar-sail many-revolution transfers, this work investigates a simple co-planar circular-to-circular (C2C) transfer [60]. The

Algorithm 2 Safeguard to avoid ”stalling” during trust-region iterations

```
Initialize best $\rho_{best}$, $\Delta_{best}$
while $|\rho - 1| \geq \epsilon_1$ do
    Perform backward induction
    Perform forward pass
    Compute validity $\rho$ as ratio between expected and actual cost reductions
    if $|\rho - 1| \leq |\rho_{best} - 1|$ then
        Update $\rho_{best}$, $\Delta_{best}$ pair
    end if
    Update trust-region radius $\Delta$ through Eq. 20
    if $\Delta = \Delta_{min}$ then
        Compute $\epsilon_1 = |\rho_{best} - 1|$
        if $\epsilon_1 \leq \epsilon_{1,\max}$ then
            Set $\Delta = \Delta_{best}$
        else
            Interrupt HDDP process (due to trust-region stalling)
        end if
    end if
end while
```

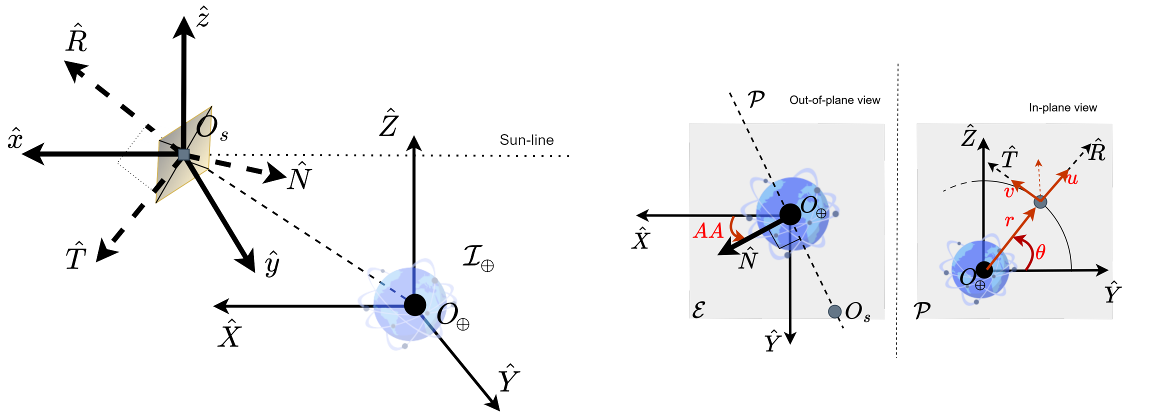
mathematical model of the chosen OCP is introduced in this section.

A. Reference frames

The reference frames defined throughout this work are illustrated in Fig. 2, with Fig. 2a depicting the ECI and SLF frames and Fig. 2b depicting the RTN frame. The definition of each reference frame is presented below:

- **Earth-Centered Inertial** (ECI): denoted by $\mathcal{I}_{\oplus}(\hat{X}, \hat{Y}, \hat{Z})$, the ECI frame is centered on Earth's barycenter $O_{\oplus}$. The $\hat{X}$ and $\hat{Y}$ axes lie in the ecliptic plane $\mathcal{E}$, with the $\hat{X}$ axis pointing in the direction of the vernal equinox at the $J2000$ epoch. The $\hat{Z}$ axis is perpendicular to the ecliptic plane pointing towards the North pole, with the $\hat{Y}$ axis completing the right-handed frame. As no ephemeris models are considered in this work, all OCPs assume initial time $t_0 = J2000$ for simplicity.
- **Sun-Light Fixed** (SLF): denoted as $\mathcal{S}(\hat{x}, \hat{y}, \hat{z})$, the SLF frame is centered on the sailcraft position O_s. The $\hat{x}$ axis is aligned with the Sun-Earth direction (i.e., at $J2000$ it is aligned with $\hat{X}$), while $\hat{z}$ is aligned with $\hat{Z}$, and $\hat{y}$ completes the right-handed reference frame.
- **Radial Tangential Normal** (RTN): denoted as $O(\hat{R}, \hat{T}, \hat{\boldsymbol{n}})$, the RTN frame has its origin on the sailcraft position O_s. The $\hat{R}$ axis is directed along the radial direction (away from the central body), $\hat{\boldsymbol{n}}$ is aligned with the angular momentum vector, and $\hat{T}$ completes the right-handed triplet. Assuming the sailcraft is only subject to in-plane effects, the orbital plane $\mathcal{P}$ is fixed.

The angle between the Sun-line $\hat{x}$ and the direction $\hat{\boldsymbol{n}}$ of orbit's angular momentum vector is referred to as aspect angle AA and shown in Fig. 2b. Limiting the analyses to polar orbits, varying aspect angles can describe all possible orbit illumination conditions ($AA = 0°$ and $AA = 90°$ indicate, respectively, orbits perpendicular and parallel to the incoming Sun-light).



(a) Reference frames

(b) Orbital plane definition and coordinates system

Fig. 2 Reference frames and coordinate systems representation

B. Dynamical model

The simplified transfer considered in this work accounts for Earth's point-mass gravity and the solar-sail acceleration (including eclipses). The following assumptions aim at replicating a Hohmann-transfer-like dynamical model to obtain easily interpretable results while reducing computational complexity. Further works shall investigate the features of orbital transfers under more representative dynamical models (i.e., including higher-order gravity terms, planetary radiation pressure, aerodynamics, and third-body gravitational effects).

An ideal sail model is adopted for this work, assuming a planar, fully reflective, and rigid surface [1]. Non-ideal effects (such as the ptical properties of the sail film, billowing under external loads, and long-term degradation [61, 62]) are neglected in favor of dynamical model simplicity, though they have been shown to have a significant impact on the sail performance [7] and shall consequently be considered in future works. Incoming radiation is assumed to be uniformly directed along the Sun-line $\hat{x}$ (i.e., the Sun is infinitely far). Time-dependent variations in the Sun-Earth configuration are accounted for by introducing a rotation of the SLF frame around its $\hat{z}$ axis of $0.9863°/day$ (corresponding to 360° per Earth year). The distance to the Sun is assumed to be constant, as the sailcraft is bound to planetary orbits: the sail performance is described by its characteristic acceleration a_0 (i.e., the magnitude of the maximum acceleration attainable at Earth's distance from the Sun using the considered solar sail). Under the previous assumptions, the solar-sail acceleration only depends on the orientation of the sail, which thus serves as the control input to the OCP.

Different representations of the sail orientation can be adopted. Most literature [39, 46] adopts the cone-clock angle pair (respectively α and δ) to describe the orientation of the sail's normal vector, $\hat{\boldsymbol{n}}_S$, with respect to the SLF frame, hence the control input to the OCP is $\boldsymbol{u} = [\alpha \quad \delta]^{\mathrm{T}}$. The cone angle α represents the current illumination conditions of the sail (angle between Sun-line and sail-normal vector), while δ defines its azimuthal orientation around the Sun-line. Since the solar sail cannot produce a sunward acceleration, the set of admissible cone angle values is restricted $\alpha \in (0°;\ 90°)$. The solar-sail acceleration vector $\boldsymbol{a}_{\mathcal{S}}$ is defined in the SLF frame as:

$$\boldsymbol{a}_{\mathcal{S}} = \zeta a_0 \cos^2 \alpha \begin{bmatrix} \cos\alpha & \sin\alpha\sin\delta & \sin\alpha\sin\delta \end{bmatrix}^{\mathrm{T}} \tag{21}$$

where ζ is the shadow factor. The condition $\zeta = 0$ represents an umbra region (i.e., no sunlight illuminating the solar sail), while the condition $\zeta = 1$ indicates full illumination. The shadow factor in the penumbra region is computed through a Heaviside step function, according to the model in Ref. [63].

An alternative representation of the OCP control inputs $\boldsymbol{u}$ is achieved through the components of the sail-normal vector in the SLF frame, thus yielding $\boldsymbol{u} = \begin{bmatrix} \hat{n}_x & \hat{n}_y & \hat{n}_z \end{bmatrix}^{\mathrm{T}}$. Physical solutions are guaranteed by constraining the components of the normal vector $\hat{n}_{\mathcal{S}}$ $\hat{n}_y,\ \hat{\boldsymbol{n}}_z \in (-1;\ 1)$, the sunward acceleration $\hat{n}_x \in (0;\ 1)$, and the unit-vector norm $||\hat{\boldsymbol{n}}|| = 1$. The solar-sail acceleration vector is computed in the SLF frame as:

$$\begin{aligned} &\boldsymbol{a}_{\mathcal{S}} = \zeta a_0 \hat{n}_x^2 \begin{bmatrix} \hat{n}_x & \hat{n}_y & \hat{n}_z \end{bmatrix}^{\mathrm{T}}, \text{ such that :} \\ &-1 \leq \hat{n}_y, \hat{n}_z \leq 1 \\ &0 \leq \hat{n}_x \leq 1 \\ &||\hat{\boldsymbol{n}}|| = 1 \end{aligned} \tag{22}$$

Both the cone-clock angles and the normal components formulations imply a restricted set of admissible controls $\mathcal{U}$, enforced through path constraints. Note that the set of admissible sail orientations (expressed in terms of normal vector components) represents a bubble in 3D space [1]. A different formulation, based on the Pontryagin's Minimum Principle is now introduced, with the aim of providing an OCP formulation that does not require path constraints. The primer-vector direction $\boldsymbol{\gamma}$, representing the direction along which the solar-sail acceleration is to be maximized for an optimal solution, thus it is considered as a control variable [15, 60]. Under the 2D dynamics assumption, the primer-vector direction $\boldsymbol{\gamma}$ necessarily lies in the orbital plane, and is uniquely described by its yaw angle A (defined with respect to the tangential direction), which becomes the control input to the OCP $\boldsymbol{u} = A$ [60]. The cone angle α that maximizes the sail acceleration along the primer-vector is computed using the locally optimal law from Ref. [1], while the corresponding δ is chosen so that the resulting acceleration lies in the orbital plane. The solar-sail acceleration vector is computed in the SLF frame through

$$
\begin{aligned}
\boldsymbol{\gamma}_S &= R_O^S \begin{bmatrix} \cos A & \sin A & 0 \end{bmatrix}^{\mathrm{T}} \\
\phi &= \tan^{-1}\left(\frac{\|\boldsymbol{\gamma}_S \times \hat{x}\|}{\boldsymbol{\gamma}_S^T \hat{x}}\right) \\
\alpha &= \frac{1}{2}\left[\phi - \sin^{-1}\left(\frac{\sin\phi}{3}\right)\right] \quad [1] \\
\delta &= \tan^{-1}\left(\frac{\gamma_y}{\gamma_z}\right) \\
\boldsymbol{a}_S &= \zeta a_0 \cos^2\alpha \begin{bmatrix} \cos\alpha & \sin\alpha\sin\delta & \sin\alpha\sin\delta \end{bmatrix}^{\mathrm{T}}
\end{aligned}
\tag{23}
$$

where R_O^S is the rotation from the RTN to the SLF frame and ϕ is the primer-vector cone angle (i.e.: the angle between the primer-vector $\boldsymbol{\gamma}$ and the Sun-line $\hat{x}$). The formula used to compute the primer-vector cone angle ϕ offers more numerical stability than the traditionally used arc-cosine function for small angles.

C. Coordinate systems

The modeled dynamics are limited to only in-plane effects (2D dynamics assumption). Under such assumption, the sailcraft state can be efficiently represented using polar coordinates, which provide advantageous numerical properties while being singularity-free [64]. The polar coordinates are included in Figure 2b. The polar coordinates set is $\begin{bmatrix} r & \theta & u & v \end{bmatrix}$, where r (radius) is the distance from the central body, θ (true anomaly) is the angle between the spacecraft position vector and a chosen reference direction, u is the radial velocity, and v is the tangential velocity. The chosen reference direction is the intersection line between the orbital plane $\mathcal{P}$ and the ecliptic plane $\mathcal{E}$, with $\theta = 0°$ towards the positive $\hat{Y}$ direction (see Fig. 2b).

The numerical stability and discretization quality is improved by introducing a Sundman transform [65], consisting of an independent variable transformation from time t to true anomaly θ. The change of variable ensures an equal angular spacing between stages and improves numerical stability in many-revolutions transfers [66]. The sailcraft state vector after the Sundman transform is defined as:

$$
\boldsymbol{x} = \begin{bmatrix} r & t & u & v \end{bmatrix}^T \tag{24}
$$

D. Scaling

The defined OCP is scaled to improve numerical stability and convergence properties. The chosen scaling approach follows the one from Ref. [24] (Appendix C.2). The chosen scaling factors are the initial radius r_0, its corresponding circular velocity v_0, and the time of flight t_f. The true anomaly θ is scaled according to a θ_{scale} value, which is varied depending on the OCP sensitivity to variations in the independent variable (unless specified otherwise, $\theta_{scale} = \theta_f$). The full EoMs are defined as:

$$\dot{\boldsymbol{x}} = \frac{d\boldsymbol{x}}{d\bar{\theta}} = \begin{bmatrix} \dot{\bar{r}} \\ \dot{\bar{t}} \\ \dot{\bar{u}} \\ \dot{\bar{v}} \end{bmatrix} = \begin{bmatrix} \bar{u}\frac{\eta}{\chi} \\ \frac{1}{\chi} \\ \left[\left(\frac{\bar{v}^2}{\bar{r}} - \frac{1}{\bar{r}^2}\right)\eta + a_R\frac{t_f}{v_0}\right]/\chi \\ \left[-\frac{\bar{u}\bar{v}}{\bar{r}}\eta + a_T\frac{t_f}{v_0}\right]/\chi \end{bmatrix}, \text{ where } \eta = \frac{v_0 t_f}{r_0}, \; \chi = \frac{\bar{v}\eta}{r\theta_{scale}} \tag{25}$$

where the $\bar{\square}$ symbol indicates scaled quantities, while a_R and a_T are the radial and tangential components of the solar-sail acceleration. Under the defined scaling, it is noticed that the solar-sail performance can be defined through its relative performance metric $\psi = \frac{a_0}{g_0}$, where g_0 is the planet's gravitational pull at distance r_0 from its center [60]. The sailcraft dynamics are propagated by MATLAB® `ode45` variable-step solver, using relative and absolute tolerances of 10^{-5} up to two-revolution transfers, while decreasing tolerances to 10^{-8} for many-revolution problems.

E. Problem formulation

The objective of the analysis is to identify time-optimal solutions to the C2C transfer problem, that is, solutions that accomplish the best C2C performance (defined as the increase in orbital radius) in the minimum amount of time, while resulting in circular final orbits. Due to the Sundman transformation, the minimum-time objective is substituted by a minimum-angle objective (i.e., minimum final true anomaly θ): since the Sundman transform only holds if the relation between time and true anomaly is monotonically increasing, the minimum-time and minimum-angle terms can be used equivalently [37]. To solve the C2C transfer problem, two possible formulations are available: the fixed-time formulation consists of maximizing the increase in orbital radius over a fixed transfer duration, while the variable-time formulation aims at achieving a certain increase in orbital radius in the minimum amount of time. Depending on the chosen control representation, two formulations are available: the C2C-NC formulation adopts the components of the sail-normal vector as control inputs (i.e., $\boldsymbol{u} = \begin{bmatrix}\hat{n}_x & \hat{n}_y & \hat{n}_z\end{bmatrix}^{\mathrm{T}}$), while the C2C-PY formulation defines the control input as the yaw angle of the primer-vector (i.e., $\boldsymbol{u} = A$).

The fixed-time problem formulation implies maximizing the final orbital radius while satisfying the terminal constraint on null eccentricity, resulting in:

$$
\begin{aligned}
&\min_{\boldsymbol{u}} \varphi(\boldsymbol{x}_{N+1}) = -r_f, \text{ such that:} \\
&\dot{\boldsymbol{x}} = \text{Eq. 25 with } \boldsymbol{a}_O = R^O_S \boldsymbol{a}_S, \text{ where } \boldsymbol{a}_S = \begin{cases} \text{Eq. 23, if C2C-PY} \\ \text{Eq. 22, if C2C-NC} \end{cases} \\
&\boldsymbol{x}_0 = \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix}^T \\
&\boldsymbol{\Psi}(\boldsymbol{x}_{N+1}) = \begin{bmatrix} \bar{u}_f \\ \bar{v}_f - \frac{1}{\sqrt{\bar{r}_f}} \end{bmatrix} = \boldsymbol{0} \\
&t(k) = t_0 + \frac{\theta_f - \theta_0}{N} k
\end{aligned}
\tag{26}
$$

where the θ_0 and θ_f indicate the initial and final true anomaly.

The algorithm capability of handling variable-duration OCPs (introduced in subsection II.A) are tested on the variable-time problem. This formulation is defined through:

$$
\begin{aligned}
&\min_{\boldsymbol{w}} \varphi(\boldsymbol{x}_{N+1}, \boldsymbol{w}) = \theta_f, \text{ such that:} \\
&\dot{\boldsymbol{x}} = \text{Eq. 25 with } \boldsymbol{a}_O = R^O_S \boldsymbol{a}_S, \text{ where } \boldsymbol{a}_S = \begin{cases} \text{Eq. 23, if C2C-PY} \\ \text{Eq. 22, if C2C-NC} \end{cases} \\
&\boldsymbol{x}_0 = \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix}^T \\
&\boldsymbol{\Psi}(\boldsymbol{x}_{N+1}) = \begin{bmatrix} \bar{u}_f \\ \bar{v}_f - \frac{1}{\sqrt{\bar{r}_f}} \\ r_{target} - r_f + s \end{bmatrix} = \boldsymbol{0} \\
&t(k, \boldsymbol{w}) = \theta_0 + \frac{\theta_f - \theta_0}{N} k
\end{aligned}
\tag{27}
$$

where r_{target} is the target minimum final radius and s is a slack variable used to transform the inequality constraint $r_{target} \leq r_f$ into its equality form (using $s \geq 0$) [24]. The chosen stage collocation function implements time-dilation by treating the transfer duration θ_f as a static parameter for the chosen OCP (i.e., $\boldsymbol{w} = \theta_f$) equally distributing changes in θ_f among all stages.

F. Initial guess

To initialize the HDDP solver, a feasible initial guess is needed: the initial guess is provided in the form of a set of control inputs, used to forward propagate the sailcraft dynamics. Two different initial guesses are used throughout this paper:

- **Locally optimal** guess: the OCP control input, regardless of its formulation, is defined such that the solar-sail acceleration is maximized along the velocity direction (following the locally optimal orbit raising law derived in Ref. [1]).
- **Trivial** guess: the OCP control input is defined with a constant trivial values, that is a primer-vector yaw angle $A \equiv 0°$ for the C2C-PY problem (i.e.: acceleration maximized along the tangential direction) and $\hat{\boldsymbol{n}} \equiv [1 \quad 0 \quad 0]^{\mathrm{T}}$ for the C2C-NC (i.e.: solar-sail acceleration always directed away from the Sun)

Unless specified otherwise, showcased results are generated using the locally optimal initial guess, as it provides faster convergence [39].

IV. Algorithm characterization

The following analysis aims to address the sensitivity of the algorithm to its hyper-parameters by providing a set of observations and interpretations, ultimately allowing informed decisions when tuning the algorithm during further analyses. Results presented in this section are obtained using the C2C-NC formulation (see Section III). The chosen benchmark problem has a duration of two revolutions ($n_{revs} = 2$), each discretized in 100 stages ($nSt_{rev} = 100$) for high-resolution results (using a $\theta_{scale} = 1$). The chosen solar sail has a characteristic acceleration $a_0 = 1\ mm/s^2$ and initial conditions corresponding to GEO altitude $r_0 = 42164\ km$. Given the short transfer duration, no Sun-line motion is considered, and the shadow factor ζ is kept constant $\zeta = 1$ (i.e., no eclipses). The benchmark problem is analyzed for two aspect angle values: $AA = 10°$ and $AA = 40°$. As a reference, Fig. 3 shows 3D views of the resulting trajectories, together with projections on the normal planes to the axes in the ECI frame, the Sun-line direction, and the time-histories of the primer vector and the solar-sail acceleration vector.

The choice in aspect angles stems from conclusions drawn in Ref. [60], where a critical aspect angle of $AA_c = 18.43°$ is identified: angles below AA_c fall in the so-caaled *ion-drive* regime (where optimal sail controls span the full admissible set $\mathcal{U}$), while angles above AA_c belong to the *solar sail* regime (where optimal sail control is restricted to only a subset of $\mathcal{U}$), which provides reduced performance in the C2C problem [60].

Bi-dimensional factorial analyses are carried out on different parameter groups, analyzing four different figures of merit: the non-augmented cost function J, the feasibility metric f, the number of iterations p, and the Total Variation of the First derivative (TVF). The TVF provides a measure for the smoothness of the optimal control profile, and it is

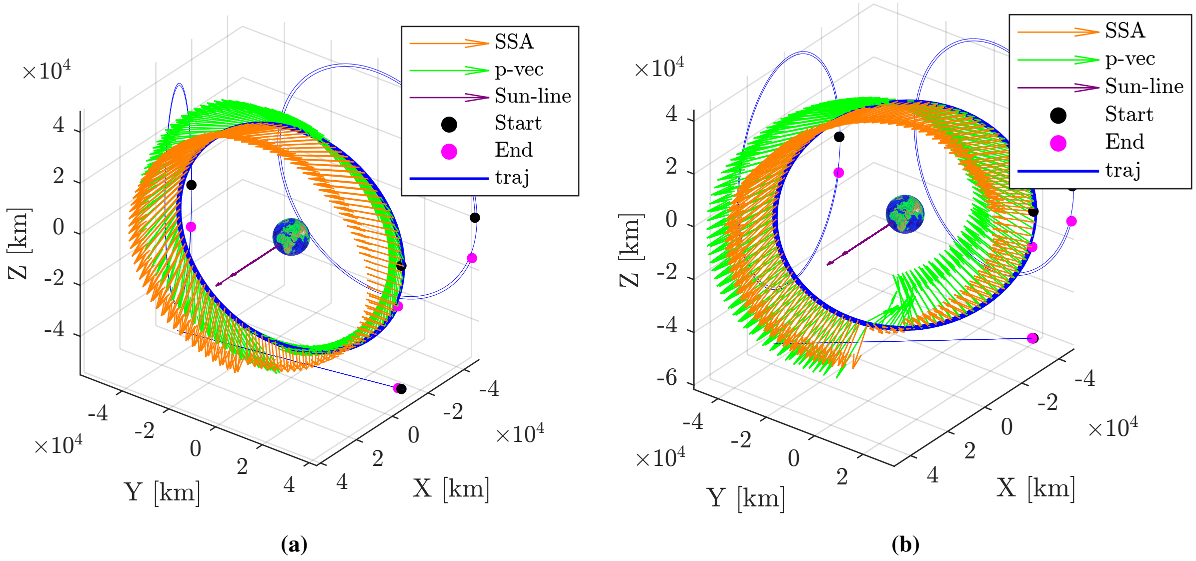


Fig. 3 3D views of the optimal trajectories in the considered benchmark problems for $AA = 10°$ (left) and $AA = 40°$(right), with trajectory projections along the ECI axes directions, orange arrows to indicate the history of the sail acceleration vectors, green arrows for the history of the primer-vector direction, and a purple arrow for the Sun-line

defined as:

$$TVF = \sum_{k=1}^{N-1} |\dot{\boldsymbol{u}}_{k+1} - \dot{\boldsymbol{u}}_k| \tag{28}$$

where the control derivatives $\dot{\boldsymbol{u}}$ at each stage are computed using a finite differences method. The TVF metric is used to investigate whether a solution has encountered diverging iterates (i.e., iterates where the control updates escape the local-optimality region, causing jittery control profiles).

The following analyses are performed on a high-performance computing platform using 48 cores (each assigned 1GB of RAM) using 2 `Intel Xeon E5-6248R 24C 3.0GHz` processors. The default values for hyper-parameters in the benchmark case are provided in Table 1, and are always used unless specified otherwise. The rationale behind the values in Table 1 is provided in the following subsections when analyzing the corresponding hyper-parameter. The objective of the following analyses is the definition of a reduced rule set to enable informed decisions throughout the HDDP tuning process, extrapolating information to generic OCPs.

Table 1 Default HDDP hyper-parameters for benchmark problem

ϵ_{opt}	ϵ_{feas}	ϵ_1	$\epsilon_{1,\max}$	Δ_0	σ_0	κ_d	κ_σ	κ_ϵ	nSt_{rev}	ϵ_{path}	Δ_σ
10^{-6}	10^{-5}	10^{-3}	10^{-2}	0.01	10^4	0.05	1.5	2.2	100	10^{-6}	50

A. Convergence tolerances

First, the optimality and feasibility tolerances ϵ_{opt} and ϵ_{feas} are considered. The effects of the factorial analysis are summarized in Fig. 4, displaying the solution optimality (with lighter colors corresponding to more optimal solutions) and feasibility (with smaller markers indicating more feasible solutions (i.e., less constraints violation)) as a function of the varied tolerances. The marker sizes are linearly scaled between the feasibility values indicated in the subtitles. The pink color indicates parameter combinations that failed to converge due to hitting the allocated runtime limit of 2 hours. A positive correlation between the tolerance values and the corresponding quantities is expected: increasing the optimality tolerance ϵ_{opt} should lead to a less optimal solution (i.e., higher cost value J), while increasing the feasibility tolerance ϵ_{feas} should yield less feasible solution (i.e., higher feasibility metric f, thus larger markers).

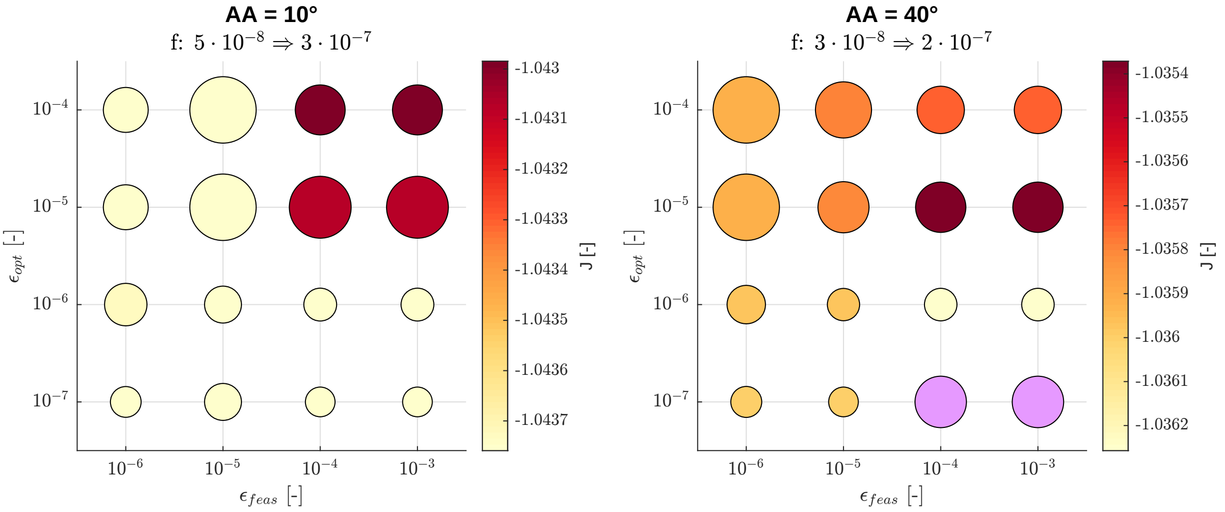


Fig. 4 Effects of ϵ_{opt} and ϵ_{feas} factorial analysis on optimality (J, indicated by marker color) and feasibility (f, indicated by marker size and linearly scaled between the denoted values). The pink color indicates failure to converge

Figure 4 only partly reflects the expected trends. Following the augmented Lagrangian approach, the optimality tolerance ϵ_{opt} inherently controls both the optimality and feasibility of the solution (embedded in the cost functional through Eq. 1b): as a result, a positive correlation between optimality tolerance and constraint violation can also be observed in Fig. 4, most noticeably for the $AA = 40°$ case (right). The expected correlation between solution feasibility and the corresponding tolerance can be observed in the $AA = 10°$ case, while the $AA = 40°$ case exhibits an opposite behavior. The counter-intuitive feature is caused by the relaxation technique for the validity threshold ϵ_1 in Eq. 19: solutions with higher feasibility tolerance ϵ_{feas} detect earlier convergence, immediately enlarging the validity threshold ϵ_1 and successfully terminating the optimization process, while stricter ϵ_{feas} tolerances allow the algorithm more "time" (i.e., more iterations) to trade-off feasibility for improved optimality (the additional iterations become detrimental to the HDDP performance in the 2 failed runs, where the algorithm is prematurely interrupted).

The tolerance values displayed in Table 1 are chosen as a trade-off between accuracy (i.e., solutions that are close to the minimal cost value J found by the different parameter combinations) and robustness (i.e., ability to reliably converge in the allocated runtime, thus excluding $\epsilon_{feas} = 10^{-3}, 10^{-4}$. Note that this analysis aims to investigate and summarize the algorithm's behavior affected by its hyper-parameters: while the cost value J exhibits only minor variations, these are considered relevant as they exceed the defined optimality tolerance ϵ_{opt} and provide insight into the underlying effects of hyper-parameter tuning on the HDDP procedure.

B. Quadratic model definition

The analysis is then extended to parameters defining the quadratic model accuracy, namely the number of stages per revolution nSt_{rev} and quadratic model accuracy threshold ϵ_1. To avoid masking out features, the ϵ_1 relaxation is not employed (i.e., $\epsilon_{1,\max} = \epsilon_1$). This parameter combination fully defines the accuracy of the quadratic cost model used to solve the OCP, with more stages (i.e., higher nSt_{rev} values) resulting in a finer discretization, while a stricter validity tolerance ϵ_1 resulting in only accepting accurate trust-region iterations. Figure 5 showcases the effects of this parameter combination on convergence speed (through the color-coded number of iterations p) and solution smoothness (through the TVF, with smaller markers corresponding to smoother solutions). As done in Fig. 4, subtitles indicate the range used to scale the marker sizes, and the pink color indicates a failure to converge. The validity tolerance ϵ_1 is expected to significantly affect the convergence speed, as large values imply that trust-region iterations are accepted more often, thus requiring fewer HDDP iterations, while the finer discretization is expected to yield smoother solutions, as it provides a better approximation of the ”true” optimal control policy.

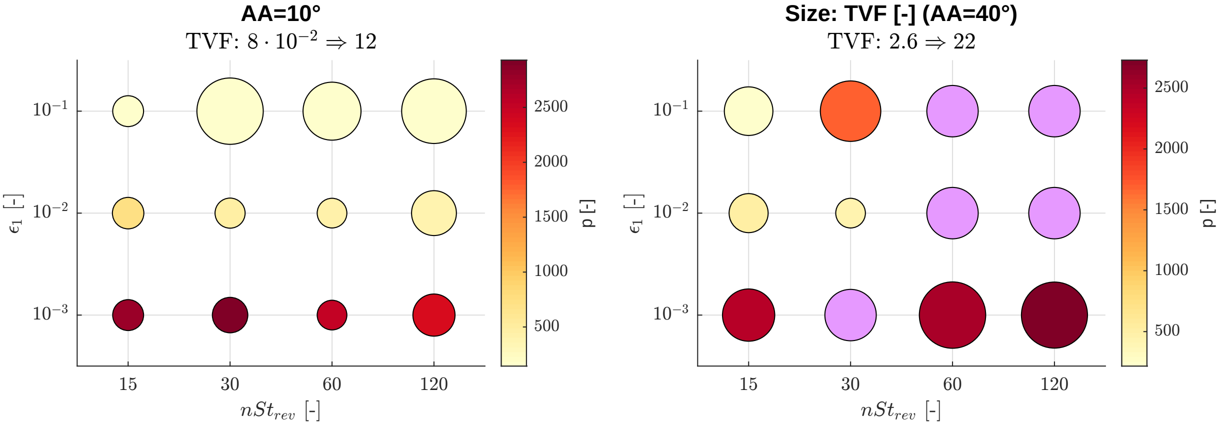
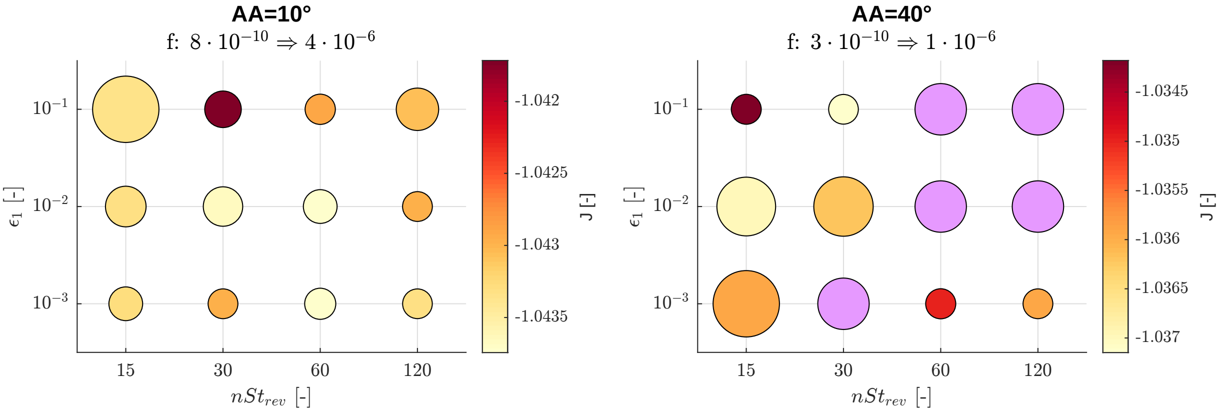


Fig. 5 Effects of nSt_{rev} and ϵ_1 factorial analysis on the number of iterations (p, indicated by marker color) and smoothness (TVF, indicated by marker size and linearly scaled between the denoted values). The pink color indicates failure to converge

Figure 5 depicts the expected correlation between validity tolerance ϵ_1 and convergence speed (i.e.: iterates are accepted more easily and therefore convergence is achieved in fewer iterations). No clear correlation can be identified between the discretization quality and the convergence speed. Focusing on the $AA = 40°$ case, it is clear that this parameter combination has a significant effect on the algorithm's reliability (i.e., its ability to converge to an optimal solution even in complex OCPs), as shown by the several failures to converge.

Conversely from the analysis in subsection IV.A (where failures were caused by "too slow" convergence), the failed cases in Fig. 5 are caused by the algorithm diverging from its local-optimality region. Smoothness and discretization quality display an opposite correlation with respect to the expected behavior: finer discretizations result in less smooth solutions, ranging from only "jittery" control profiles (e.g., the $nSt_{rev} = 120$, $\epsilon_1 = 10^{-3}$ combination) to the failed cases where TVF values are even higher than scale provided in Fig. 5. The feature is caused by the reduced sensitivity of the OCP with respect to its control inputs: the reduced control authority of the solar sail under the *solar sail* control regime [60], coupled with the short duration of the trajectory arc associated to a certain stage, causes the control inputs of each stage sub-problem to have a reduced effect on the solution of the sub-problem instead. Since the resulting TRQP sub-problems are almost insensitive to the controls (i.e., gradient/Hessian close to 0), the trust-region procedure is likely to accept iterates with diverging control inputs (even if in only some stages), as they have only a minor impact on the actual cost value J and thus do not reflect in the accuracy metric ρ used in Eq. 20. The $AA = 10°$ case highlights the expected correlation between validity tolerance ϵ_1 and the smoothness of the solution: this trend is expected since a strict tolerance ϵ_1 causes the trust-region procedure to only accept very accurate trial iterations, thus reducing the risk of introducing diverging controls into the reference solution.

Effects of the factorial analysis on solution optimality-feasibility (similarly to the ones displayed in Fig. 4) are presented in Fig. 6. While the validity tolerance ϵ_1 is not expected to correlate with optimality/feasibility specifically, a finer discretization is expected to provide a more accurate approximation of the "true" optimal control policy, thus resulting in an improved solution.



AA=10°

f: $8 \cdot 10^{-10} \Rightarrow 4 \cdot 10^{-6}$

AA=40°

f: $3 \cdot 10^{-10} \Rightarrow 1 \cdot 10^{-6}$

Fig. 6 Effects of nSt_{rev} and ϵ_1 factorial analysis on optimality (J, indicated by marker color) and feasibility (f, indicated by marker size and linearly scaled between the denoted values). The pink color indicates failure to converge

Figure 6 exhibits no clear trends in terms of optimality and feasibility. A minor exception is observed for the coarse discretization (i.e., $nSt_{rev} = 15$): considering the combined effect of feasibility and optimality, the strict tolerance ϵ_1 is shown to outperform larger values, displaying reduced constraint violations for identical values of optimality in the $AA = 10°$ case. It is noticed that a finer discretization (i.e.: higher nSt_{rev}) does not necessarily lead to improvements in terms of optimality or feasibility, conversely from the expected behavior. The feature is linked back to the issues caused by trust-region solvers for problems with very low control authority: addressing such limitations requires the coupling of a fine discretization with a strict validity tolerance ϵ_1, allowing the algorithm to accurately and reliably approximate the "true" optimum. The alternative route is to adopt a coarse discretization using a larger tolerance ϵ_1, allowing the exploitation of "less predictable" behavior (i.e., more distant from the reference solution used to derive the quadratic cost model): while resulting in faster convergence (due to the larger tolerance ϵ_1 and reduced number of decision variables), the approach is also less robust, as noticed from the numerous failed cases registered for large tolerances ϵ_1.

The chosen parameter combination displayed in Table 1 corresponds to a balance between the aim for high-resolution results (desired to understand the main features of the candidate solutions throughout the tuning process) and reliable convergence, thus falling into the fine discretization (i.e., $nSt_{rev} = 100$) and strict tolerance (i.e., $\epsilon_1 = 10^{-3}$) logic.

C. Penalty parameter

Finally, the effectiveness of the automatically-tuned penalty parameter update is assessed. The procedure aims at iteratively adjusting the penalty parameter to accurately meet the specified feasibility tolerance. For the scope of this analysis, three approaches to update the penalty parameter σ are considered: $penaltyUpdate$ 0 corresponds to the heuristics-based approach in Eq. 12, $penaltyUpdate$ 1 is the newly introduced update-rule in Eq. 18, and $penaltyUpdate$ 2 is the nested trust-region approach defined in Alg. 1. Different guesses for the penalty parameter σ are provided, resulting in solutions with different balances between optimality and feasibility: the effects are shown in Fig. 7.

It is immediately noticed how the adaptive procedures introduced in this work yield the desired effects, with the corresponding solutions displaying constraint violations that closely meet the defined feasibility tolerance $\epsilon_{feas} = 10^{-5}$ while resulting in more optimal solutions. Additionally, the nested trust-region procedure is shown to be particularly robust with respect to the provided initial guess on the penalty parameter σ. As expected, the heuristics-based approach from Eq. 12 typically displays more feasible and less optimal solutions: the feature, justified by the conflicting nature of the OCP objective and constraints, is caused by Eq. 12 only causing the penalty parameter σ to increase, thus excessively prioritizing feasibility.

The adaptive procedures (i.e., $penaltyUpdate$ $1, 2$) are shown to accurately meet the required feasibility tolerance ϵ_{feas}. This feature constitutes a major benefit in terms of parameter tuning, as it removes (or reduces) the sensitivity of HDDP to the tuning of the penalty parameter, which is known to have a noticeable impact on OCP solutions

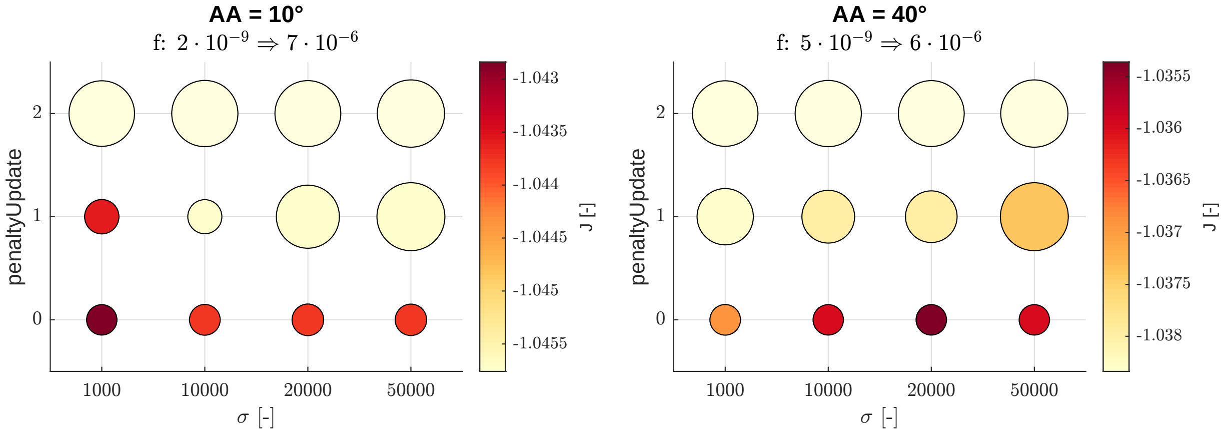


Fig. 7 Effects of σ and $PenaltyUpdate$ factorial analysis on optimality (J, indicated by marker color) and feasibility (f, indicated by marker size and linearly scaled between the denoted values). The pink color indicates failure to converge

[33, 39]. These approaches, however, introduce significant computational overhead caused by the propagation of multiple second-order models (one for each component of the constraint violation vector $\mathbf{\Psi}$) along with the quadratic cost model required by HDDP, with the nested trust-region approach in Alg. 1 requiring an extension to the allocated 2 hours to reliably converge in all the considered cases. Since the optimization of many-revolution transfer OCPs is an already computationally demanding task, the overhead is minimized by adopting the ”default” procedure (i.e., $penaltyUpdate$ 0) in Eq. 12.

D. Performance scaling

The advantageous feature of DDP algorithms is the linear memory scaling with problem dimensionality [32]. This property is verified for the benchmark C2C problem, using the C2C-PY formulation to minimize the computational effort (i.e., reduce the number of decision variables and path constraints) by gradually increasing the number of orbital revolutions. The analysis is performed on a single `AMD Ryzen 5 5600H` processor, with 6 cores dedicated to parallel computing. Three cases were analyzed, with $AA = 10°$, $AA = 40°$, and $AA = 90°$: since identical results and conclusions were drawn, only the $AA = 90°$ case is shown in Fig. 8, displaying the average runtime required per HDDP iteration and the estimated algorithm convergence order. The convergence order estimates the rate of descent of the solution accuracy (with respect to the final ”true” minimum cost J_f) over the algorithm iterations, computed as the best approximation of:

$$(J_{p+1} - J_f) = coeff(J_p - J_f)^{order} \tag{29}$$

where $order$ is the convergence order, p is a generic HDDP iteration, and $coeff$ is a finite real number. The convergence order is determined by performing an exponential fit of Eq. 29 over the HDDP iterations required to solve an OCP. Previous works state that the DDP algorithm has a quadratic convergence order [32].

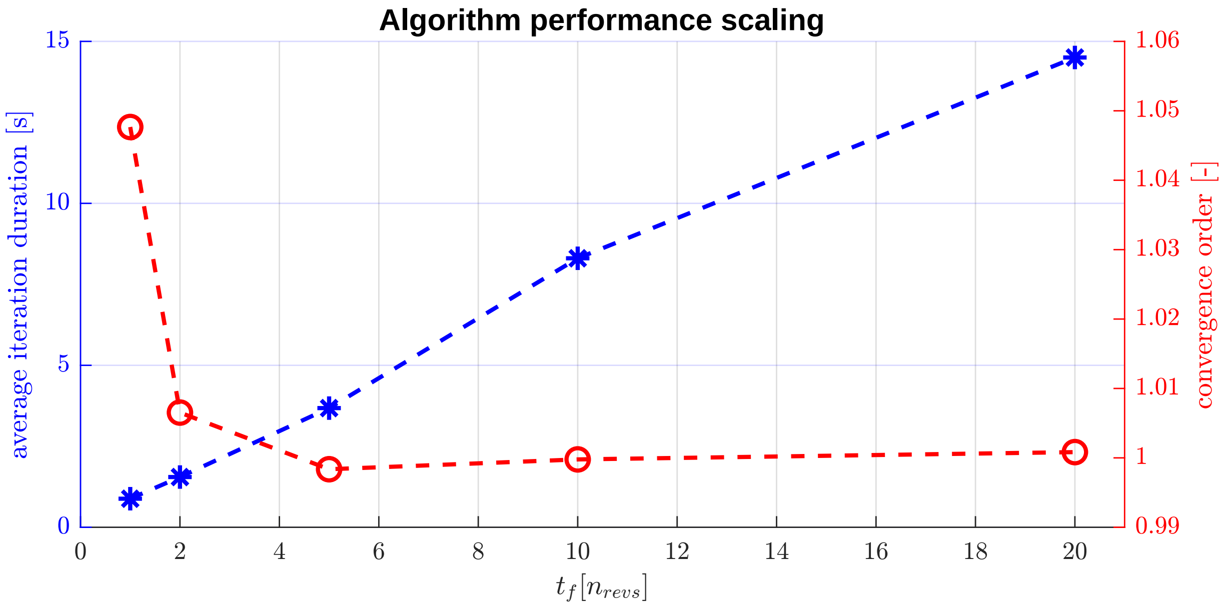


Fig. 8 Performance scaling of the modified HDDP over increasing transfer durations, in terms of average iteration duration and estimated convergence order

As expected, Fig 8 shows linearly increasing computational cost over transfer duration t_f: an exponential fit determines that the average iteration duration (i.e., iteration effort) increases as $O(t_f^{1.2})$. The slightly super-linear trend is caused by initialization overhead.

Conversely from theoretical guarantees, the convergence order is shown to be close to linear (i.e., close to 1) for the considered time of flight values. The undesirable feature is caused by the need to enforce positive definiteness on the Hessian matrices: since the property is artificially enforced through a Hessian-shifting algorithm [52], the convergence order drops to only linear [67]. It is observed that longer transfer durations further reduce the convergence order, meaning that an increased number of iterations requires the Hessian-shifting algorithm to enforce positive definiteness.

E. Considerations and summary

Results and features observed throughout the parameter analysis and algorithm tuning are now summarized, with the goal of outlining a 'guide' to HDDP tuning. Considerations regarding each parameter (or group of parameters) are presented.

- ϵ_{opt}, ϵ_{feas}: the tolerances, as expected, control their respective metrics, although the optimality tolerance indirectly affects feasibility through the penalty parameter σ (for large values for σ, ϵ_{opt} can have a dominant effect over ϵ_{feas}).
- ϵ_1: the validity threshold controls the acceptance of the trial iterates, significantly affecting solutions and convergence properties. High values for ϵ_1 should be prioritized for faster convergence, at the risk of inducing "jittery" controls or "escaping" the current local-optimality region, resulting in failure to converge. Low ϵ_1 values

(coupled with the devised relaxation technique) are preferable for smoother solutions. If a high-resolution is desired, the validity tolerance ϵ_1 shall be strict enough to avoid diverging iterates.

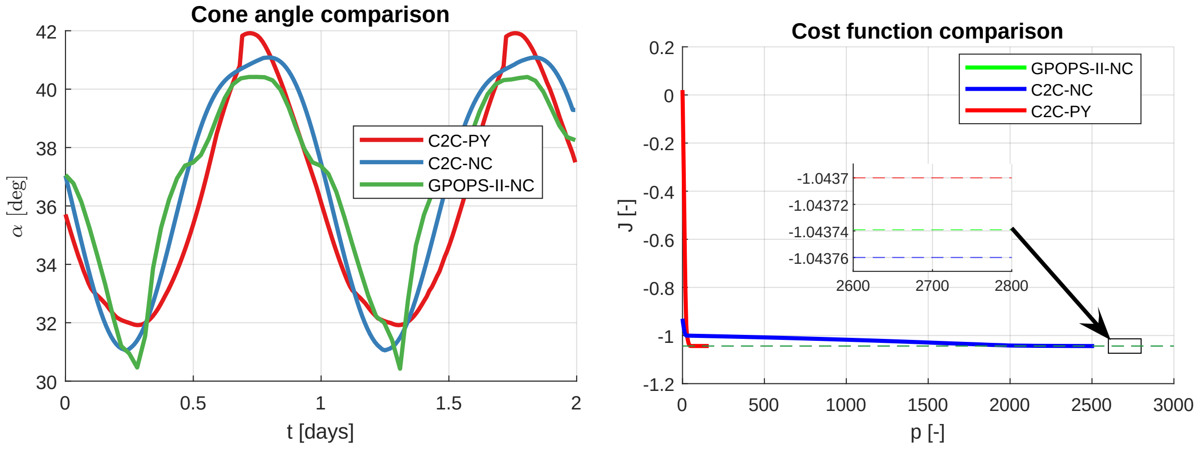
- nSt_{rev}: a fine discretization ensures high-resolution solutions. The convergence properties of the HDDP algorithm are improved under a coarse mesh, as the increased duration of the trajectory arc associated with each stage implies that each TRQP sub-problem is more sensitive to the corresponding control inputs: in these conditions, the trust-region procedure can reliably identify and discard iterations where control updates are diverging or sub-optimal. Expectedly, a coarse mesh leads to faster convergence, although the interaction between mesh collocation and problem dynamics (i.e., presence of control switches, uncontrollable trajectory arcs) is to be considered: the devised algorithm allows to dynamically address such concerns through the stage collocation function.
- σ: the penalty parameter controls the balance between solution optimality and feasibility. The augmented Lagrangian approach ensures that even under low σ values feasibility can be met. Lower values shall be preferred for solution quality, while higher σ ensures faster convergence to feasible solutions. The adaptive tuning technique successfully removes the need to manually tune the value for σ, greatly adding to the HDDP framework's reliability at the cost of considerable computational overhead, thus requiring further investigation.
- Δ_0: the initial value for the trust-region radius mainly controls the initial convergence speed of the algorithm, until a trust-region radius Δ is reached such that iterates are only accepted for similar trust-region radius. Initializing using a high Δ_0 requires many trial iterates to achieve a successful trust-region step, but results in a maximization of the available optimality gain in early iterations, and is therefore recommended.
- κ_d, Δ_{min}: the trust-region radius enlargement coefficient κ_d and the minimum trust-region radius Δ_{min} control the adaptive update of the trust-region radius in Eq. 20. The definition of a reliable Δ_{min} depends on problem knowledge and convergence requirements (i.e., a Δ_{min} close to machine precision ensures uninterrupted optimization, but also does not detect trust-region stalling). The adaptive enlargement of the validity threshold ϵ_1 was found to be a reliable way to address trust-region stalling without requiring very low Δ_{min}. Low values for κ_d are recommended ($\kappa_d = 0.05$ was found to work reliably in all considered problems).

V. Results

The devised algorithm is applied to different solar-sail transfer problems. First, validation against a state-of-the-art direct optimization solver is performed. The correct implementation of the HDDP framework is verified in Appendix B. The validated algorithm is then applied to different many-revolution C2C problems, characterizing minimum-time C2C solutions across multiple transfer durations and ultimately solving a variable-duration many-revolution C2C transfer.

A. Validation

The modified HDDP algorithm is validated against the state-of-the-art GPOPS-II solver, which adopts direct collocation to transcribe the OCP into an NLP, solved with IPOPT [56]. The validation is performed successfully on different problem formulations (from orbit raising to transfers, adopting different control representations), using the hyper-parameter settings from Table 1. The results shown in Figure 9 refer to the $AA = 10°$ 2-revolution C2C problem, comparing the results of the C2C-PY and C2C-NC formulations, both solved using the modified HDDP algorithm and benchmarked against the GPOPS-II solution to the C2C-NC problem (referred to as GPOPS-II-NC). All problems are solved starting from a trivial initial guess. Results are shown in Fig. 9: Fig. 9a depicts the optimal profiles of cone angle α, Fig. 9b showcases the decrease in cost function decrease over the HDDP algorithm iterations, with dashed lines representing the final cost values (the GPOPS-II final result is added as a reference, without displaying the evolution of its candidate solutions.



(a)

(b)

Fig. 9 Results from the path-constrained optimization validation on the $AA = 10°$ C2C benchmark problem: comparison of the cone angle α profiles (left) and of the decrease in cost function over HDDP iterations (right), with dashed lines indicating the final value of the cost function (the GPOPS-II result is added as a reference)

The identified solutions display minor differences in terms of optimal cone angle profiles while matching in terms of solution optimality. The final values of the cost function shown in Fig. 9b show the C2C-NC formulation to be advantageous in terms of optimality. A potential explanation of the phenomenon is in the nature of the solar-sail acceleration, which depends on the Sun-line direction (i.e., the $\hat{x}$ axis of the SLF reference frame, see Fig. 2a). Since HDDP assumes control inputs to be constant over each stage, the C2C-NC can inherently ”track” the inertial Sun-line direction better than the C2C-PY formulation, where the control inputs are defined in the (local) RTN reference frame. Further work shall investigate the hypothesis by thoroughly comparing different state-control representations of the problem.

Figure 9b shows that the trivial guess in the C2C-NC formulation performs better than its counterpart using the

C2C-PY formulation. The feature is caused by the different definitions of the adopted (trivial) initial guesses: the C2C-PY initial guess constantly thrusts along the $\hat{T}$ direction, significantly increasing the eccentricity and thus yielding major constraint violations $\mathbf{\Psi}$ (affecting the augmented cost functional through Eq. 1b). Thanks to the augmented Lagrangian formulation, the C2C-PY quickly reduces its constraint violation, achieving fast convergence. Conversely, the C2C-NC initial guess remains close to circular due to the $AA = 10°$ configuration, leading the trivial initial guess to an almost entirely out-of-plane acceleration (which is neglected by the 2D dynamics assumption).

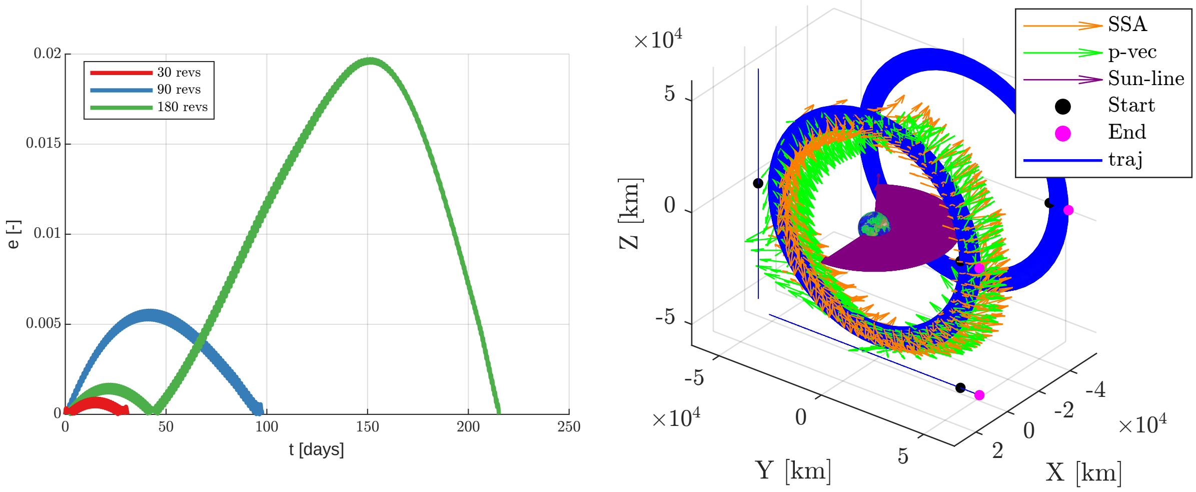
B. Many-revolution transfers

The validated algorithm is applied to time-optimal many-revolution transfers. The analysis is first carried out on fixed-duration problems, investigating the solution features in different settings. A high-control-authority case is defined at GEO altitude, where eclipses are almost negligible and the relative effect from Earth's gravity is reduced: the resulting OCP is highly sensitive to its control inputs, thus the problem also serves as a way to certify the algorithm's capability to reliably converge in complex problems (without additional overhead introduced by techniques such as multiple-shooting [41, 68]). The low-control-authority case is defined at LEO altitude to test the algorithm's computational limitations by optimizing very large OCPs. The gathered information is exploited to derive a power-regression model to characterize the time-optimal C2C performance, which is then used to define an initial guess for the FFT formulation. Computational complexity is minimized by adopting the C2C-PY formulation, removing the need for path-constraints enforcement and reducing the number of control variables. The adopted HDDP hyper-parameters follow the values from Table 1, with only the penalty parameter being changed to $\sigma = 10^6$, proving the robustness of the tuning process outlined in Section IV.

1. High control authority transfer

First, solutions to the C2C problem in a high-control-authority setting are investigated. The chosen scenario corresponds to a GEO-altitude orbit ($r_0 = 42164\ km$) and an ACS3-like solar sail [6] ($a_0 = 0.05\ mm/s^2$), resulting in a relative performance $\psi = \frac{a_0}{g_0} = 2.64 \cdot 10^{-4}$. The transfer starts at $AA = 0°$ but gradually increases due to the time variation of the orientation of the SLF frame. The mesh refinement procedure is initialized with $nSt_{rev} = 15$, and ultimately converges in all cases to $nSt_{rev} = 120$, as the high control authority implies that the solution is sensitive to the chosen discretization scheme. Results for 30, 90, and 180- revolution transfers are summarized in Fig. 10, with Fig. 10a displaying the eccentricity evolution for each of the analyzed transfers, while Fig. 10b depicts the 3D view of the optimized 180-revolution transfer (following the same convention as Fig 3). The observed trends are analyzed on a macroscopic scale, ignoring oscillations that occur within a single orbital period.

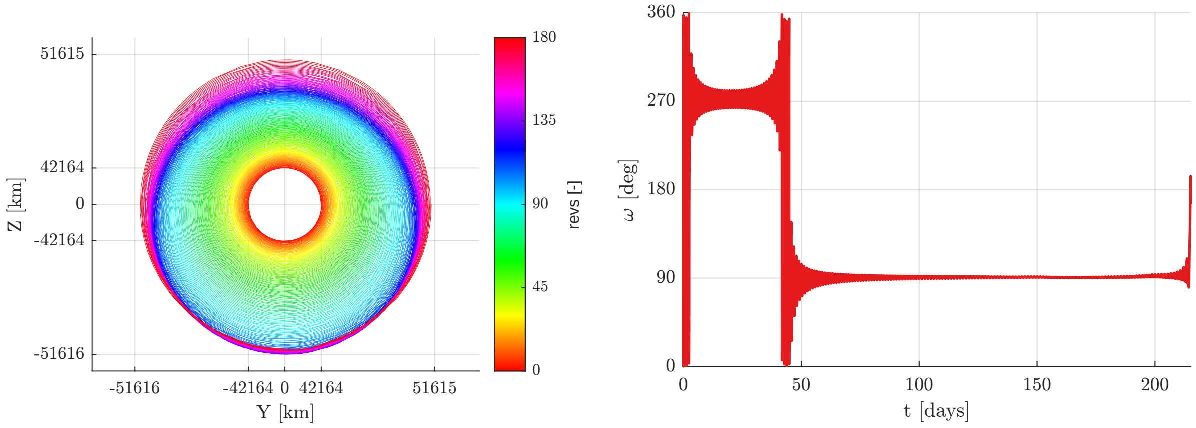
Two different trends can be identified in Figure 10a: for shorter transfers (i.e., 30 and 90 revolutions) the trajectory goes through one cycle of eccentricity increase and decrease, whereas for longer durations two cycles are observed.



(a) **(b)**

Fig. 10 Results for the many-revolution C2C transfers in a high-control-authority setting (ACS3-like solar sail at GEO altitude): time-history of the eccentricity (left) and 3D view of the optimal 180-revolution transfer (right)

Further analyses showed that the transition between the two regimes occurs at a ≃120-revolution transfer duration. To explain this behavior, Fig. 11 is introduced, with Fig. 11a showing the orbital-plane view of the optimal trajectory (with exaggerated proportions for easier visualization) and Fig. 11b displaying the time-evolution of the argument of periapsis ω (defined using the same convention as the true anomaly θ, with respect to the positive $\hat{Y}$ direction of the ECI frame using a counterclockwise rotation, see Fig. 2a).



(a) **(b)**

Fig. 11 Orbital-plane view of the 180-revolution C2C transfer (left), with exaggerated proportions, and evolution of the argument of periapsis (right) for the 180-revolution C2C transfer

The oscillating behavior in Fig. 11b (also later observed in Fig. 13b) is caused by the singularity in the argument of periapsis in circular orbits. Figure 11b clearly shows the presence of an apogee reversal: the argument of periapsis is defined through a positive rotation (i.e. counterclockwise) from the positive $\hat{Y}$ axis, thus Fig. 11b describes an orbit that grows towards the positive $\hat{Z}$ direction (i.e., $\omega = 270°$), followed by the apogee reversal which causes the orbit to extend towards the negative $\hat{Z}$ direction (i.e., $\omega = 90°$), as also visible from Fig. 11a. The apogee-reversal maneuver can be justified considering that the long duration of the transfer introduces noticeable variations in the orbit illumination conditions. Due to the time-varying nature of the aspect angle AA, the Sun-line rotates around the $\hat{z}$ axis of the SLF frame (which is aligned with the $\hat{Z}$ axis of the ECI frame, see Fig. 2a): the in-plane projection of the Sun-line is always directed towards the positive $\hat{Y}$ direction, with an increasing magnitude for aspect angles close to $AA = 90°$. Considering only this in-plane component, each revolution of the transfer trajectory can be split into two phases: an acceleration phase, where the sailcraft moves in the Sun-line direction (i.e., the bottom part of Fig. 11a), and a drift phase, where the sailcraft moves "against" the Sun-line (i.e., the top portion of Fig. 11a) [69]. By initially placing the perigee in the acceleration phase, the transfer trajectory maximizes the gain in orbital energy by thrusting during the trajectory arc where the instantaneous velocity is maximal. The following apogee-reversal maneuver causes the apogee to fall under the acceleration phase, exploiting the solar-sail acceleration to increase its apogee velocity thus circularizing the final orbit (performing an "apogee-burn-like" maneuver, as also found in Ref. [70]).

A complete overview of the optimal control profile is provided in Fig. 12, where Fig. 12a displays the optimal cone angle α (in the blue to red color scale) over the many revolutions, with the solid green line indicating perigee, the dashed green line for apogee, black dots for eclipses, and the shaded area corresponding to $AA \leq AA_c$ conditions (verified for both $AA \in [0°; 18.43°]$ and $AA \in [161.57°; 198.43°]$). Figure 12b highlights the maximum slew rate $||\dot{\hat{\mathbf{n}}}||$ required by the analyzed many-revolution transfers.

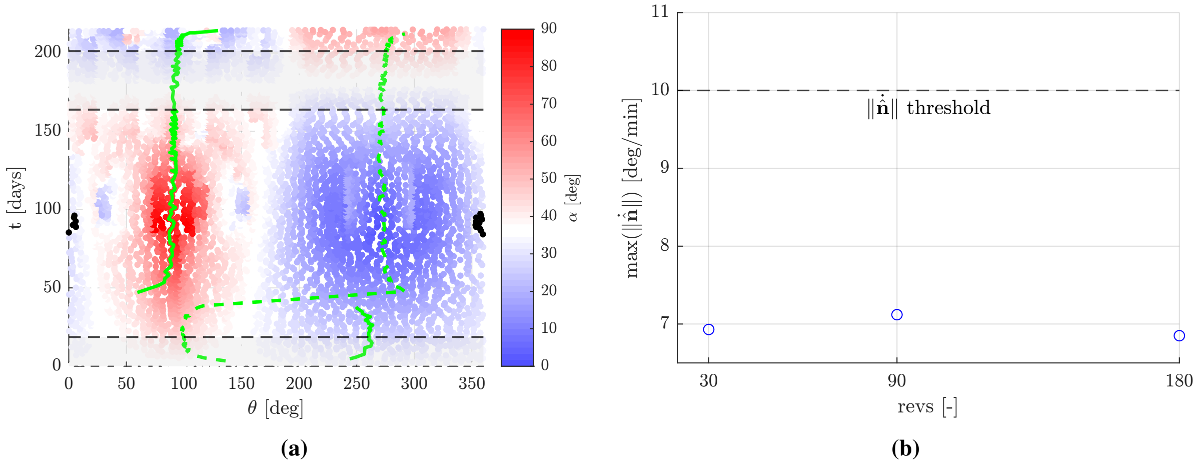
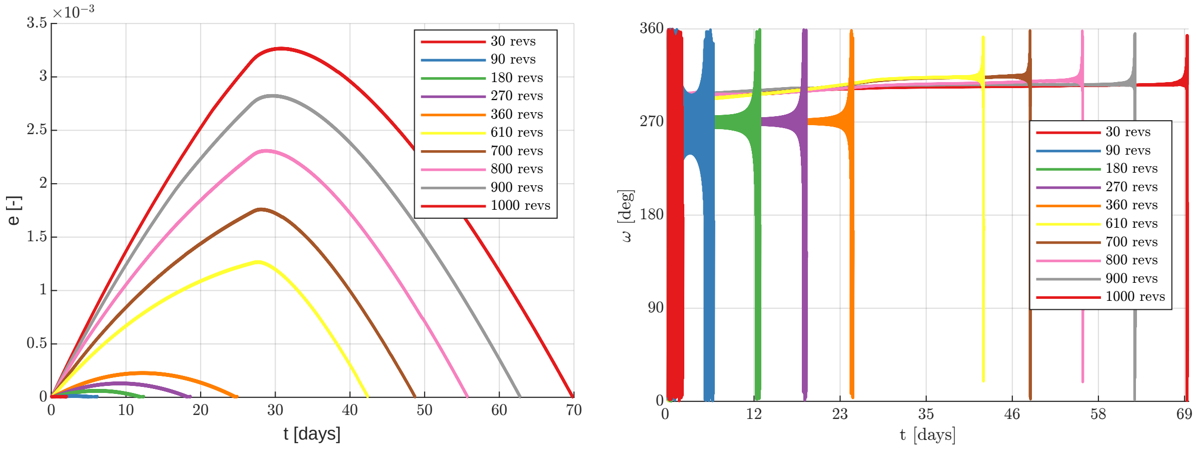


Fig. 12 Optimal control profile over the 180-revolution transfer (left), with the solid and dashed green lines to represent perigee and apogee, black dots for eclipsed conditions, and shaded areas for $AA \leq AA_c$. Maximum slew rate required by the analyzed many-revolution transfers (right)

The cone angle profile in Figure 12a portrays a clear distinction between the two control regimes identified in Ref. [60]: for $AA \leq AA_c$ the *ion-drive* regime yields a cone angle $\alpha \simeq 35.26°$, where $\alpha = 35.26°$ maximizes the in-plane solar-sail acceleration when $AA = 0°$. Conversely, the *solar-sail* regime requires periodic slewing of the solar-sail: during the acceleration phase (for $\theta \simeq 270°$), the effect of solar-sail acceleration is maximized by adopting a cone angle $\alpha \simeq 0°$, while the drift phase (for $\theta \simeq 90°$) corresponds to a cone angle $\alpha \simeq 90°$, which minimizes the acceleration component in the opposite direction of the velocity vector. The apogee reversal is also noticed in Fig. 12a, with the acceleration phase (represented by the blue region) first being associated with the perigee (solid green line) and then switching to apogee (dashed green line). Given the long orbital periods of the analyzed transfers, the required sail slew rates in Fig.12b are, expectedly, well within solar-sail attitude-control limitations (in Ref. [71] a $\simeq 10°/min$ threshold is defined for ACS3-like sails, specifying that limitations are typically introduced by the attitude control system, rather than the sail structure).

2. *Low control authority transfer*

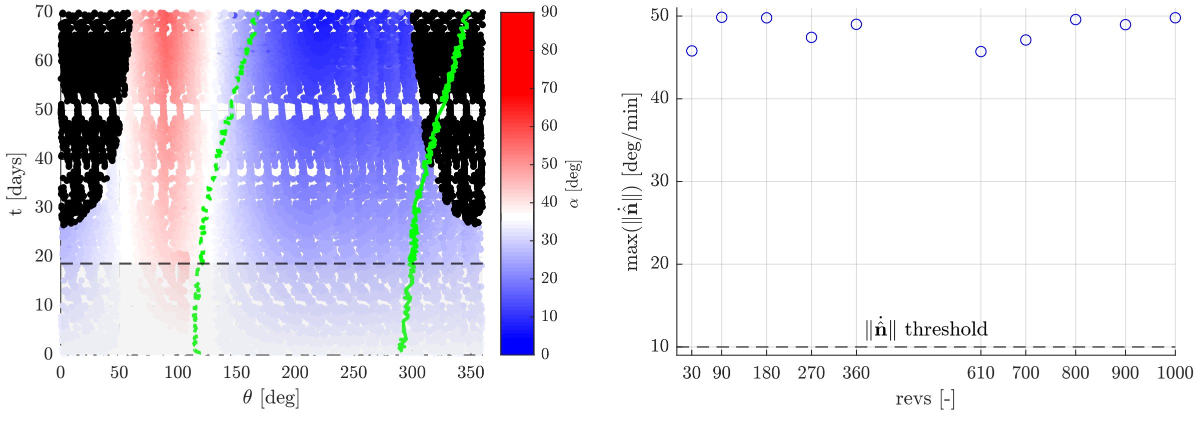
Many-revolution transfers are now investigated in a setting with much lower control authority, caused by the increased magnitude of Earth's gravitational pull as well as the frequent eclipsing phenomena. The chosen scenario considers an orbit at 700 km altitude and the same ACS3-like sail with relative performance of $\psi = 7.45 \cdot 10^{-6}$. As for the high-control-authority case, the transfer starts at $AA = 0°$. The mesh refinement procedure is initialized with $nSt_rev = 20$ and converges at its first iteration with $nSt_{rev} = 40$, as the low control authority leads to negligible solution improvement under higher resolutions. The algorithm hyper-parameters are the same as the high-control-authority transfer (i.e., Table 1 with penalty parameter $\sigma = 10^6$). The results for up to 1000-revolution transfers are summarized in Fig. 13, with Fig. 13a showing the eccentricity evolution and Fig. 13b showing the argument of periapsis for all analyzed transfers.



(a) **(b)**

Fig. 13 Results for the many-revolution C2C transfers in a low-control-authority setting (ACS3-like solar sail at 700 km altitude): time-history of the eccentricity (left) and of the argument of periapsis (right)

The eccentricity evolution in Fig. 13a displays the expected trend of a single eccentricity increase-decrease cycle for all analyzed cases. The argument of periapsis in Fig. 13b highlights two ”regions”: short-duration transfers (up to 360 revolutions) show fixed values $\omega = 270°$, while the longer-duration transfers display a slight increase and irregularity in terms of argument of periapsis $\omega \geq 270$. The feature is caused by the presence of eclipsing phenomena only in the long-duration transfers, as the time-varying Sun-line direction implies that the orbital plane intersects the umbra/penumbra regions only after ≃ days. More detailed insight into the optimal control profiles and orbital geometry is provided in Fig. 16, displaying the cone angle evolution for the 1000-revolution transfer in Fig. 14a (following the same conventions as Fig. 12a) and the required slew rates for all analyzed cases in Fig. 14b.



(a) **(b)**

Fig. 14 Optimal control profile over the 1000-revolution transfer (left), with the solid and dashed green lines to represent perigee and apogee, black dots for eclipsed conditions, and shaded areas for $AA \leq AA_c$. Maximum slew rate required by the analyzed many-revolution transfers (right)

The cone angle profile depicted in Fig. 14a shows the transition between the *ion-drive* regime (where the optimal cone angle is $\alpha \simeq 35.26°$) and *solar-sail* control regime. Under the *solar-sail* control regime, the division between the acceleration phase (i.e., $\alpha \simeq 0°$) and the drift phase (i.e., $\alpha \simeq 90°$) is different from the one in the high-control-authority case (see Fig. 12a). The feature is caused by the presence of significant eclipsing phenomena: since only a limited trajectory arc per revolution enables effective use of the solar-sail thrust, the optimal control profile maximizes the portion of the acceleration phase spent under illuminated conditions. This logic causes the drift phase (red region in Fig. 14a) to be considerably shorter than the acceleration phase (the blue region in Fig. 14a). The increased argument of periapsis $\omega \geq 270°$ observed in Fig. 13b is also a result of this feature, as the optimal control law gradually shifts the apogee of the transfer trajectory towards the acceleration phase (i.e., $\theta \simeq 90°$), prioritizing the orbit’s circularization at the end of the transfer. The required slew rate shown in Fig. 14b displays considerable violations of the threshold defined in Ref. [71], implying that future works shall investigate the effects of attitude-control limitations on optimal trajectories. It is noticed that both in Fig. 12b and Fig. 14b the required slew rate $||\dot{\hat{\mathbf{n}}}||$ does not depend on the transfer duration: since the optimal control laws require to periodically slew the sail, the maximum slew rate is only dependent on the shortest orbital period observed during the transfer (i.e., from its initial conditions), thus explaining the considerable differences observed between the attitude control requirements of GEO and LEO transfers.

3. *Power-regression models*

Having investigated the main features of the optimal control profiles for fixed-duration time-optimal many-revolution transfers, the overall solar-sail performance in the considered C2C cases is examined. Following the approach of Ref. [21], a power regression model $\Delta r = An_{rev}^b$, predicting the orbital radius increase Δr as a function of the number of revolutions n_{rev}, is fitted to the optimized C2C transfers (with a coefficient of determination R^2). The resulting models are shown in Fig. 15, displaying the data points obtained from the optimized transfers against the model prediction and coefficients, as well as the trajectory arc spent in umbra/penumbra conditions through the vertical gray bars.

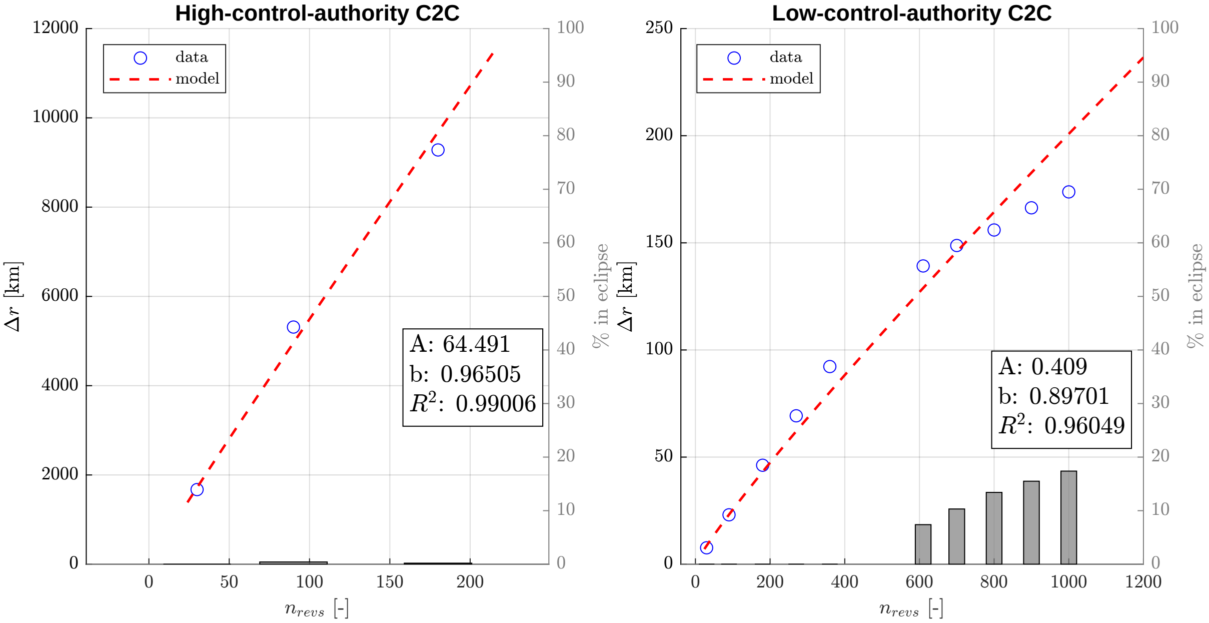


Fig. 15 Power regression models in the form $\Delta r = An_{rev}^b$ for the considered C2C transfers, combined with trajectory percentage spent in umbra/penumbra conditions indicated by the vertical gray bars, the resulting model coefficients A and b, and the coefficient of determination R^2

The regression on the high-control-authority transfers results in a quasi-linear model. While its high coefficient of determination R^2 can be attributed to the few available data points, no significant variations from the linear trend are expected within the considered range (0 – 180 revolutions). As expected, eclipsing is negligible in such conditions and yields no effects on the regression model. Conversely, the low-control-authority C2C analysis displays a sub-linear trend, with significant portions of the transfers spent under umbra/penumbra conditions. The elbow-like feature causing the sub-linear trend starts in correspondence with trajectory durations which imply significant eclipsed arcs (i.e., transfer durations higher than $n_{rev} = 600$): as the sailcraft enters shadowed regions, its orbit-raising capabilities decrease, causing the sub-linear trend. While the shadowed portion of the trajectory is observed to increase linearly with the transfer duration (after the sailcraft has entered eclipsed conditions), a decreasing trend is expected for transfers that reach higher aspect angles than $AA \geq 90°$, where the umbra/penumbra regions exit the orbital plane: the power regression model is therefore unreliable for extrapolation.

4. Flexible-final-time solution

The devised power regression model is used to estimate the time-optimal duration for a C2C transfer with a given orbital radius increase Δr. For the example in this subsection, a target orbital radius increase of $\Delta r = 150\ km$ is chosen, as it corresponds to a region where the defined model displays high accuracy (as shown in Fig. 15). Using the low-control-authority model coefficients $A = 0.409$ and $b = 0.89701$, an estimate of $\simeq 720$ revolutions is obtained to perform the 150 km orbit increase. Solving the fixed-time 720-revolution C2C transfer yields a $\Delta r = 147.25\ km$ (i.e., the model overestimates the sail performance). The transfer is subsequently reformulated as a variable-duration problem through Eq. 27, using the optimized 720-revolution transfer as the initial guess. The scaling factor on the independent variable θ_{scale} is reduced to 0.1 in this OCP, as the value was found to sufficiently reduce the high sensitivity of the problem to its discretization. The adopted HDDP parameters are summarized in Table 2.

Table 2 Default HDDP hyper-parameters for the variable-time C2C problem

ϵ_{opt}	ϵ_{feas}	ϵ_1	$\epsilon_{1,\max}$	Δ_0	σ_0	κ_d	κ_σ	κ_ϵ	nSt_{rev}	ϵ_{path}	Δ_σ
10^{-6}	10^{-5}	10^{-2}	10^{-1}	0.01	10^6	0.05	1.5	2.2	20	10^{-6}	50

The identified solution is a 745-revolution C2C transfer: the optimal control profile of the variable-time solution is shown in Fig. 16a, while the result of the 745-revolution fixed-time transfer is shown in Fig. 16b (both figures follow the convention introduced in Fig. 12a).

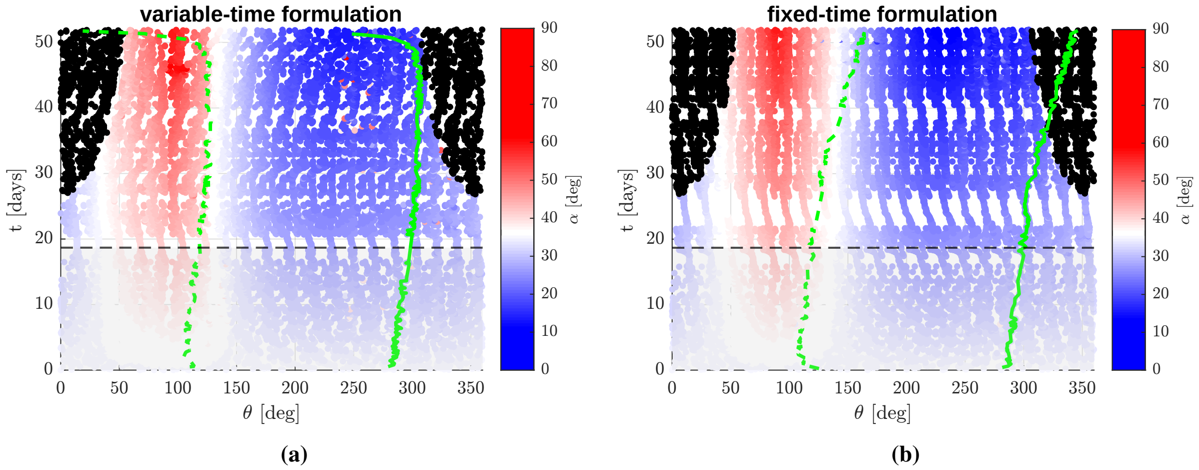


Fig. 16 Optimal control profile over the 745-revolution transfer using the variable-time (left) and fixed-time (right) formulations, with the solid and dashed green lines to represent perigee and apogee, black dots for eclipsed conditions, and shaded areas for $AA \leq AA_c$

The fixed-time formulation slightly outperforms the variable-time one, registering an increase in orbital radius of

$\Delta_r = 150.2\ km$ (against the increase $\Delta_r = 150\ km$ of the variable-time formulation). Both solutions display the features observed for the 1000-revolution solution (see Fig. 14a), namely the uneven distribution of the acceleration and drift phases, with the drift phase being ”allocated” to eclipsed trajectory arcs. The control profiles in Fig. 16 are overall very similar, certifying the validity of the time-dilation approach introduced in subsection II.A. A minor difference can be identified in the final portion of the transfer trajectory: while the fixed-time formulation manages to ”steer” the apogee (dashed green line) towards the acceleration phase (blue region), thus efficiently circularizing its orbit, the variable-time approach appears to follow an opposite policy. The feature can be attributed to the different formulations in terms of cost and constraints (see Eq. 26 and Eq. 27): the variable-time formulation sacrifices the solution feasibility in favor of a shorter-duration transfer, while the fixed-time formulation manages to identify a more feasible and optimal solution (note that both results meet the specified tolerance ϵ_{feas}). While it is expected that the performance of the variable-time formulation can be improved by accurately tuning the scaling factor θ_{scale}, it holds that the fixed-time formulation provides a more reliable framework, consistently with other time-dilation approaches applied to different optimization methods [21].

VI. Conclusions

This work presented a modified Hybrid Differential Dynamic Programming (HDDP) algorithm, enhanced with novel techniques to address some of its limitations: path constraints are enforced using a second-order method, variable-duration problems are accommodated through time-dilation and a modified framework for the propagation of State Transition Maps (STM), and the sensitivity to its hyper-parameters is reduced through adaptive techniques to tune the penalty parameter and the validity threshold of the quadratic cost model.

A simplified solar-sail circular-to-circular transfer problem was used to characterize the sensitivity of the algorithm to its hyper-parameters. Key findings include the coupled effects of the optimality tolerance ϵ_{opt} on both optimality and feasibility, caused by the augmented Lagrangian approach used in HDDP, the numerical instabilities induced by the highly influential quadratic cost model (both in terms of discretization and imposed accuracy threshold ϵ_1), and the satisfying performance of the devised adaptive parameter algorithm (which, however, induces undesirable computational overhead). The convergence properties of the modified algorithm were also verified, displaying the expected linear increase in computational effort with OCP dimensionality and a reduced convergence order, caused by the complex nature of the analyzed OCP. The full algorithm characterization was reduced to a small set of key takeaways, enabling informed decisions when tuning the algorithm in future works.

The newly introduced methodologies were successfully validated against a state-of-the-art direct optimization solver in the context of simplified solar sail transfer problems. The devised algorithm was then applied to more complex many-revolution transfers: solving such transfers at GEO altitude (where the solar sail has a higher control authority) highlighted unexpected features in the solutions, namely the presence of an apogee-reversal maneuver to address the time-varying Sun-line direction in long-duration transfers. Optimal transfers at LEO altitude were shown to address the significant eclipsing phenomena by shifting the orbital geometry, causing the drifting phase (i.e., the trajectory arc where the sail moves towards the Sun) to occur in the eclipsed region. The characterized solutions were used to derive a power regression model for the solar-sail performance in the analyzed transfer OCPs: the observed trends were (close to) linear but significantly affected by the presence of eclipsing. The regression model was used to initialize a variable-duration OCP, identifying a 745-revolution duration as the minimum time required to transfer between circular orbits from 700 km to 850 km altitudes.

Some limitations were also observed throughout the work. Further work shall improve the adaptive tuning of the penalty parameter introduced in subsection II.D to prioritize more computationally efficient approaches (e.g., the one from Ref. [72]). The obtained results were limited to a simplified dynamical model, thus future studies shall leverage the proposed HDDP implementation (available in Ref. [53]) to investigate higher-fidelity dynamical models and/or different OCPs, exploiting the flexibility of the approach provided by the automatic differentiation framework.

The showcased methodology and results constitute a first step towards the broader understanding of optimal sail-powered many-revolutions transfers around planetary bodies. The integration of variable-duration and path-constrained problems into the HDDP framework expands its optimization capabilities, enabling its use in relevant solar sail OCPs, from preliminary solutions to the space debris removal problem to trajectory optimization for long-duration missions.

Appendix A

Second-order feedback terms

The second-order terms for the path-constrained TRQP sub-problem solution introduced in Subsection II.B are:

$$0 = \begin{bmatrix} 2\sum_{k=1}^{n_q} \boldsymbol{\mu}^k_{\boldsymbol{x}^j}(\boldsymbol{q}^k_{\boldsymbol{ux}} + \boldsymbol{q}^k_{\boldsymbol{uu}} U_{\boldsymbol{x}}) \\ U^T_{\boldsymbol{x}^j}\boldsymbol{q}^1_{\boldsymbol{ux}} + \boldsymbol{q}^1_{\boldsymbol{x}^j\boldsymbol{u}} U_{\boldsymbol{x}} + \boldsymbol{q}^1_{\boldsymbol{x}^j\boldsymbol{x}} + U^T_{\boldsymbol{x}^j}\boldsymbol{q}^1_{\boldsymbol{uu}} U_{\boldsymbol{x}} \\ \vdots \\ U^T_{\boldsymbol{x}^j}\boldsymbol{q}^{n_q}_{\boldsymbol{ux}} + \boldsymbol{q}^{n_q}_{\boldsymbol{x}^j\boldsymbol{u}} U_{\boldsymbol{x}} + \boldsymbol{q}^{n_q}_{\boldsymbol{x}^j\boldsymbol{x}} + U^T_{\boldsymbol{x}^j}\boldsymbol{q}^{n_q}_{\boldsymbol{uu}} U_{\boldsymbol{x}} \end{bmatrix} + (\mathcal{M} + \mathcal{H}) \begin{bmatrix} U_{\boldsymbol{x}^j\boldsymbol{x}} \\ \boldsymbol{\mu}_{\boldsymbol{x}^j\boldsymbol{x}} \end{bmatrix}$$

$$0 = \begin{bmatrix} 2\sum_{k=1}^{n_q} \boldsymbol{\mu}^k_{\boldsymbol{w}^j}(\boldsymbol{q}^k_{\boldsymbol{uw}} + \boldsymbol{q}^k_{\boldsymbol{uu}} U_{\boldsymbol{w}}) \\ U^T_{\boldsymbol{w}^j}\boldsymbol{q}^1_{\boldsymbol{uw}} + \boldsymbol{q}^1_{\boldsymbol{w}^j\boldsymbol{u}} U_{\boldsymbol{w}} + \boldsymbol{q}^1_{\boldsymbol{w}^j\boldsymbol{w}} + U^T_{\boldsymbol{w}^j}\boldsymbol{q}^1_{\boldsymbol{uu}} U_{\boldsymbol{w}} \\ \vdots \\ U^T_{\boldsymbol{w}^j}\boldsymbol{q}^{n_q}_{\boldsymbol{uw}} + \boldsymbol{q}^{n_q}_{\boldsymbol{w}^j\boldsymbol{u}} U_{\boldsymbol{w}} + \boldsymbol{q}^{n_q}_{\boldsymbol{w}^j\boldsymbol{w}} + U^T_{\boldsymbol{w}^j}\boldsymbol{q}^{n_q}_{\boldsymbol{uu}} U_{\boldsymbol{w}} \end{bmatrix} + (\mathcal{M} + \mathcal{H}) \begin{bmatrix} U_{\boldsymbol{w}^j\boldsymbol{w}} \\ \boldsymbol{\mu}_{\boldsymbol{w}^j\boldsymbol{w}} \end{bmatrix} \qquad 0 = \begin{bmatrix} 2\sum_{k=1}^{n_q} \boldsymbol{\mu}^k_{\lambda^j}(\boldsymbol{q}^k_{\boldsymbol{uu}} U_{\lambda}) \\ 0 \\ \vdots \\ 0 \end{bmatrix} + (\mathcal{M} + \mathcal{H}) \begin{bmatrix} U_{\lambda^j\lambda} \\ \boldsymbol{\mu}_{\lambda^j\lambda} \end{bmatrix}$$

$$0 = \begin{bmatrix} 2\sum_{k=1}^{n_q} \boldsymbol{\mu}_{\boldsymbol{w}^j}^k (\boldsymbol{q}_{\boldsymbol{ux}}^k + \boldsymbol{q}_{\boldsymbol{uu}}^k U_{\boldsymbol{x}}) \\ U_{\boldsymbol{w}^j}^T \boldsymbol{q}_{\boldsymbol{ux}}^1 + \boldsymbol{q}_{\boldsymbol{w}^j\boldsymbol{u}}^1 U_{\boldsymbol{x}} + \boldsymbol{q}_{\boldsymbol{w}^j\boldsymbol{x}}^1 + U_{\boldsymbol{w}^j}^T \boldsymbol{q}_{\boldsymbol{uu}}^1 U_{\boldsymbol{x}} \\ \vdots \\ U_{\boldsymbol{w}^j}^T \boldsymbol{q}_{\boldsymbol{ux}}^{n_q} + \boldsymbol{q}_{\boldsymbol{w}^j\boldsymbol{u}}^{n_q} U_{\boldsymbol{x}} + \boldsymbol{q}_{\boldsymbol{w}^j\boldsymbol{x}}^{n_q} + U_{\boldsymbol{w}^j}^T \boldsymbol{q}_{\boldsymbol{uu}}^{n_q} U_{\boldsymbol{x}} \end{bmatrix} + (\mathcal{M} + \mathcal{H}) \begin{bmatrix} U_{\boldsymbol{w}^j\boldsymbol{x}} \\ \boldsymbol{\mu}_{\boldsymbol{w}^j\boldsymbol{x}} \end{bmatrix} \tag{27}$$

$$0 = \begin{bmatrix} 2\sum_{k=1}^{n_q} \boldsymbol{\mu}_{\boldsymbol{x}^j}^k (\boldsymbol{q}_{\boldsymbol{uw}}^k + \boldsymbol{q}_{\boldsymbol{uu}}^k U_{\boldsymbol{w}}) \\ U_{\boldsymbol{x}^j}^T \boldsymbol{q}_{\boldsymbol{uw}}^1 + \boldsymbol{q}_{\boldsymbol{x}^j\boldsymbol{u}}^1 U_{\boldsymbol{w}} + \boldsymbol{q}_{\boldsymbol{x}^j\boldsymbol{w}}^1 + U_{\boldsymbol{x}^j}^T \boldsymbol{q}_{\boldsymbol{uu}}^1 U_{\boldsymbol{w}} \\ \vdots \\ U_{\boldsymbol{x}^j}^T \boldsymbol{q}_{\boldsymbol{uw}}^{n_q} + \boldsymbol{q}_{\boldsymbol{x}^j\boldsymbol{u}}^{n_q} U_{\boldsymbol{w}} + \boldsymbol{q}_{\boldsymbol{x}^j\boldsymbol{x}}^{n_q} + U_{\boldsymbol{x}^j}^T \boldsymbol{q}_{\boldsymbol{uu}}^{n_q} U_{\boldsymbol{w}} \end{bmatrix} + (\mathcal{M} + \mathcal{H}) \begin{bmatrix} U_{\boldsymbol{x}^j\boldsymbol{w}} \\ \boldsymbol{\mu}_{\boldsymbol{x}^j\boldsymbol{w}} \end{bmatrix}$$

$$\dots$$

Appendix B

The modified HDDP algorithm validation presented in Subsection V.A represents one of the different performed tests. First, a linear-quadratic (LQ) problem is chosen as a test case, as augmented Lagrangian methods are known to converge in a single iteration in this problem class [73]. Different variations in terms of objectives, formulation, and initial conditions were tested: the LQ problem formulated here follows the test case in Ref. [48]:

$$\begin{aligned} \min_{\boldsymbol{u}_{i,k},\, \boldsymbol{w}_i} \sum_{i=1}^{2} \sum_{k=1}^{5} \|\boldsymbol{u}_{i,k}\|^2,&\ \text{such that,, for } i = 1,2 \;\&\; k = 1, ..., 5: \\ \boldsymbol{\Gamma}_1 &= \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T \\ \boldsymbol{\Gamma}_2 &= \boldsymbol{w}_2 \\ \boldsymbol{x}_{i,k+1} &= \begin{bmatrix} \boldsymbol{x}_{i,k}(1:3) + \boldsymbol{x}_{i,k}(4:6) \\ \boldsymbol{x}_{i,k}(4:6) + \boldsymbol{u}_{i,k} \end{bmatrix} \\ \boldsymbol{\Psi}_1(\boldsymbol{x}_{1,6}, \boldsymbol{w}_1, \boldsymbol{x}_{2,1}, \boldsymbol{w}_2) &= \boldsymbol{x}_{1,6} - \boldsymbol{w}_2 \\ \boldsymbol{\Psi}_2(\boldsymbol{x}_{2,6}, \boldsymbol{w}_2) &= \boldsymbol{x}_{2,6} - \begin{bmatrix} 1 & -1 & 0 \end{bmatrix}^T \end{aligned} \tag{30}$$

The algorithm is observed to achieve single-iteration convergence (as also in all considered LQ-problem variations), with results summarized in Fig. 17: visual comparison with the results from Ref. [48] confirms the correct implementation.

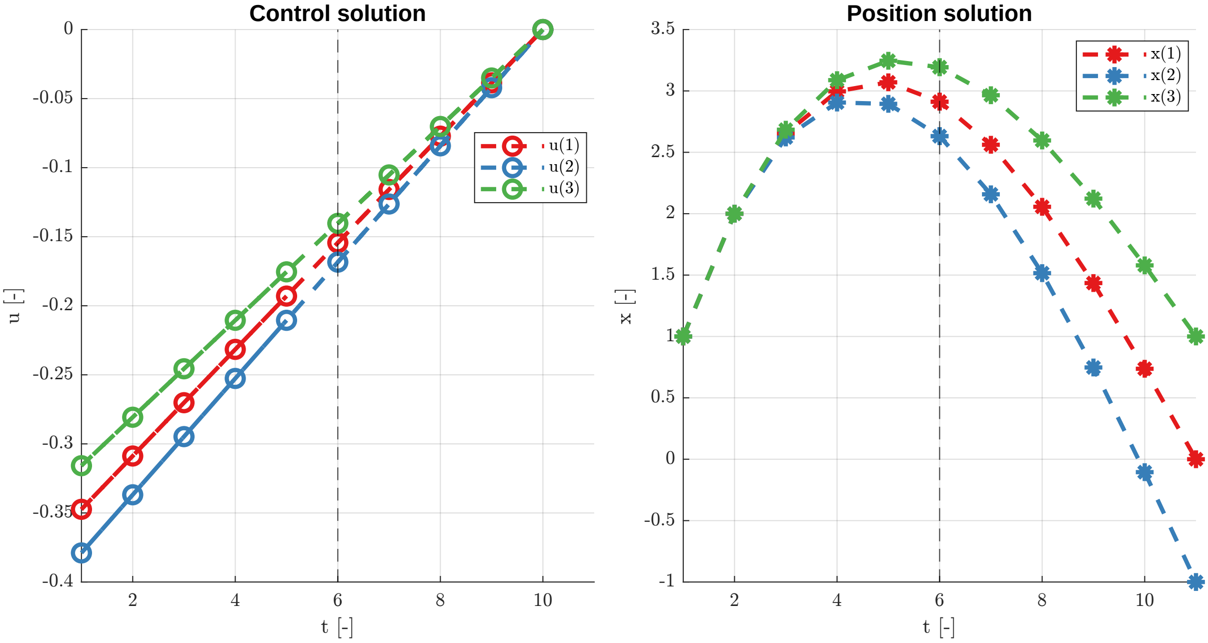


Fig. 17 Position states and controls resulting from the optimized LQ-problem (defined also in Ref. [48])

Acknowledgments

The authors acknowledge the use of computational resources of the DelftBlue supercomputer, provided by Delft High Performance Computing Centre (`https://www.tudelft.nl/dhpc`).

References

[1] McInnes, C. R., *Solar sailing. Technology, dynamics and mission applications.*, 1999. URL `https://ui.adsabs.harvard.edu/abs/1999sstd.book.....M`, publication Title: Solar sailing. Technology ADS Bibcode: 1999sstd.book.....M.

[2] Sawada, H., Mori, O., Okuizumi, N., Shirasawa, Y., Miyazaki, Y., Natori, M., Matunaga, S., Furuya, H., and Sakamoto, H., "Mission Report on The Solar Power Sail Deployment Demonstration of IKAROS," *52nd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, American Institute of Aeronautics and Astronautics, Denver, Colorado, 2011. https://doi.org/10.2514/6.2011-1887, URL https://arc.aiaa.org/doi/10.2514/6.2011-1887.

[3] Alhorn, D., Casas, J., Agasid, E., Adams, C., Laue, G., Kitts, D. C., and O'Brien, S., "NanoSail-D: The Small Satellite That Could!" Logan, UT, 2011.

[4] Ridenoure, R. W., Spencer, D. A., Stetson, D. A., Betts, B., Munakata, R., Wong, S. D., Diaz, A., Plante, B., Foley, J. D., and Bellardo, J. M., "Status of the Dual CubeSat LightSail Program," *AIAA SPACE 2015 Conference and Exposition*, American Institute of Aeronautics and Astronautics, Pasadena, California, 2015. https://doi.org/10.2514/6.2015-4424, URL https://arc.aiaa.org/doi/10.2514/6.2015-4424.

[5] Spencer, D. A., Betts, B., Bellardo, J. M., Diaz, A., Plante, B., and Mansell, J. R., "The LightSail 2 solar sailing technology

demonstration," *Advances in Space Research*, Vol. 67, No. 9, 2021, pp. 2878–2889. https://doi.org/10.1016/j.asr.2020.06.029, URL https://linkinghub.elsevier.com/retrieve/pii/S027311772030449X.

[6] Wilkie, W. K., "Overview of the NASA Advanced Composite Solar Sail System (ACS3) Technology Demonstration Project," *AIAA Scitech 2021 Forum*, American Institute of Aeronautics and Astronautics, VIRTUAL EVENT, 2021. https://doi.org/10.2514/6.2021-1260, URL https://arc.aiaa.org/doi/10.2514/6.2021-1260.

[7] Mengali, G., and Quarta, A. A., "Optimal Three-Dimensional Interplanetary Rendezvous Using Non-Ideal Solar Sail," *Journal of Guidance, Control, and Dynamics*, Vol. 28, No. 1, 2005, pp. 173–177. https://doi.org/10.2514/1.8325, URL https://arc.aiaa.org/doi/10.2514/1.8325.

[8] Quarta, A. A., Mengali, G., Bassetto, M., and Niccolai, L., "Optimal interplanetary trajectories for Sun-facing ideal diffractive sails," *Astrodynamics*, Vol. 7, No. 3, 2023, pp. 285–299. https://doi.org/10.1007/s42064-023-0158-4, URL https://link.springer.com/10.1007/s42064-023-0158-4.

[9] Lantoine, G., Cox, A., Sweetser, T., Grebow, D., Whiffen, G., Garza, D., Petropoulos, A., Oguri, K., Kangas, J., Kruizinga, G., and Castillo-Rogez, J., "Trajectory & maneuver design of the NEA Scout solar sail mission," *Acta Astronautica*, Vol. 225, 2024, pp. 77–98. https://doi.org/10.1016/j.actaastro.2024.08.039, URL https://www.sciencedirect.com/science/article/pii/S0094576524004788.

[10] Leipold, M. E., and Wagner, O., "Mercury sun-synchronous polar orbits using solar sail propulsion," *Journal of Guidance, Control, and Dynamics*, Vol. 19, No. 6, 1996, pp. 1337–1341. https://doi.org/10.2514/3.21791, URL https://doi.org/10.2514/3.21791, publisher: American Institute of Aeronautics and Astronautics _eprint: https://doi.org/10.2514/3.21791.

[11] Macdonald, M., and McInnes, C. R., "Analytical Control Laws for Planet-Centered Solar Sailing," *Journal of Guidance, Control, and Dynamics*, Vol. 28, No. 5, 2005, pp. 1038–1048. https://doi.org/10.2514/1.11400, URL https://arc.aiaa.org/doi/10.2514/1.11400.

[12] McInnes, C. R., Macdonald, M., Angelopolous, V., and Alexander, D., "GEOSAIL: Exploring the Geomagnetic Tail Using a Small Solar Sail," *Journal of Spacecraft and Rockets*, Vol. 38, No. 4, 2001, pp. 622–629. https://doi.org/10.2514/2.3727, URL https://arc.aiaa.org/doi/10.2514/2.3727.

[13] Colombo, C., Lücking, C., and McInnes, C. R., "Orbital dynamics of high area-to-mass ratio spacecraft with $J2$ and solar radiation pressure for novel Earth observation and communication services," *Acta Astronautica*, Vol. 81, No. 1, 2012, pp. 137–150. https://doi.org/10.1016/j.actaastro.2012.07.009, URL https://www.sciencedirect.com/science/article/pii/S009457651200272X.

[14] Kelly, P. W., Bevilacqua, R., Mazal, L., and Erwin, R. S., "TugSat: Removing Space Debris from Geostationary Orbits Using Solar Sails," *Journal of Spacecraft and Rockets*, Vol. 55, No. 2, 2018, pp. 437–450. https://doi.org/10.2514/1.A33872, URL https://arc.aiaa.org/doi/10.2514/1.A33872.

[15] Kelly, P., and Bevilacqua, R., “Geostationary debris mitigation using minimum time solar sail trajectories with eclipse constraints,” *Optimal Control Applications and Methods*, Vol. 42, No. 1, 2021, pp. 279–304. https://doi.org/10.1002/oca.2676, URL https://onlinelibrary.wiley.com/doi/10.1002/oca.2676.

[16] Delft, T., “SWEEP,” , ???? URL https://www.tudelft.nl/en/ae/sweep.

[17] Johnson, L., Young, R., Barnes, N., Friedman, L., Lappas, V., and McInnes, C., “Solar Sails: Technology And Demonstration Status,” *International Journal of Aeronautical and Space Sciences*, Vol. 13, No. 4, 2012, pp. 421–427. https://doi.org/10.5139/IJASS.2012.13.4.421, URL http://koreascience.or.kr/journal/view.jsp?kj=HGJHC0&py=2012&vnc=v13n4&sp=421.

[18] Oguri, K., Lantoine, G., Petropoulos, A. E., and McMahon, J. W., “Solar Sailing Q-Law for Planetocentric, Many-Revolution Sail Orbit Transfers,” *Journal of Guidance, Control, and Dynamics*, Vol. 46, No. 10, 2023, pp. 2005–2014. https://doi.org/10.2514/1.G007103, URL https://doi.org/10.2514/1.G007103, publisher: American Institute of Aeronautics and Astronautics _eprint: https://doi.org/10.2514/1.G007103.

[19] Betts, J. T., “Very low-thrust trajectory optimization using a direct SQP method,” *Journal of Computational and Applied Mathematics*, Vol. 120, No. 1, 2000, pp. 27–40. https://doi.org/10.1016/S0377-0427(00)00301-0, URL https://www.sciencedirect.com/science/article/pii/S0377042700003010.

[20] Topputo, F., and Zhang, C., “Survey of Direct Transcription for Low-Thrust Space Trajectory Optimization with Applications,” *Abstract and Applied Analysis*, Vol. 2014, 2014, pp. 1–15. https://doi.org/10.1155/2014/851720, URL http://www.hindawi.com/journals/aaa/2014/851720/.

[21] Graham, K. F., and Rao, A. V., “Minimum-Time Trajectory Optimization of Multiple Revolution Low-Thrust Earth-Orbit Transfers,” *Journal of Spacecraft and Rockets*, Vol. 52, No. 3, 2015, pp. 711–727. https://doi.org/10.2514/1.A33187, URL https://arc.aiaa.org/doi/10.2514/1.A33187.

[22] Fitzgerald, R. M., “Characterizing Minimum-Time Solar Sail Geostationary Orbit Transfers Using Pseudospectral Optimal Control,” *Journal of Spacecraft and Rockets*, Vol. 58, No. 4, 2021, pp. 997–1009. https://doi.org/10.2514/1.A34950, URL https://arc.aiaa.org/doi/10.2514/1.A34950.

[23] Morante, D., Sanjurjo Rivo, M., and Soler, M., “A Survey on Low-Thrust Trajectory Optimization Approaches,” *Aerospace*, Vol. 8, No. 3, 2021, p. 88. https://doi.org/10.3390/aerospace8030088, URL https://www.mdpi.com/2226-4310/8/3/88.

[24] Longuski, J. M., Guzmán, J. J., and Prussing, J. E., *Optimal Control with Aerospace Applications*, Springer New York, New York, NY, 2014. https://doi.org/10.1007/978-1-4614-8945-0, URL https://link.springer.com/10.1007/978-1-4614-8945-0.

[25] Haberkorn, T., Martinon, P., and Gergaud, J., “Low Thrust Minimum-Fuel Orbital Transfer: A Homotopic Approach,” *Journal of Guidance, Control, and Dynamics*, Vol. 27, No. 6, 2004, pp. 1046–1060. https://doi.org/10.2514/1.4022, URL https://arc.aiaa.org/doi/10.2514/1.4022, publisher: American Institute of Aeronautics and Astronautics.

[26] E, Z., and Guzzetti, D., “Multi-revolution low-thrust trajectory optimization using symplectic methods,” *Science China Technological Sciences*, Vol. 63, No. 3, 2020, pp. 506–519. https://doi.org/10.1007/s11431-019-9511-7, URL http://link.springer.com/10.1007/s11431-019-9511-7.

[27] Barles, A., Ceriotti, M., Ciampa, F., and Felicetti, L., “An optimal steering law for sailing with solar and planetary radiation pressure,” *Aerospace Science and Technology*, Vol. 118, 2021, p. 107051. https://doi.org/10.1016/j.ast.2021.107051, URL https://linkinghub.elsevier.com/retrieve/pii/S1270963821005617.

[28] Sackett, L. L., “Optimal Solar Sail Planetocentric Trajectories,” Tech. rep., The Charles Stark Draper Laboratory, Inc., Cambridge, Massachusetts, Sep. 1977.

[29] Petropoulos, A. E., “REFINEMENTS TO THE Q-LAW FOR LOW-THRUST ORBIT TRANSFERS,” *15th AAS/AIAA Space Flight Mechanics Conference*, Vol. 120, 2005, pp. 963–982.

[30] Shannon, J. L., Ozimek, M. T., and Atchison, J. A., “Q-LAW AIDED DIRECT TRAJECTORY OPTIMIZATION FOR THE HIGH-FIDELITY, MANY-REVOLUTION LOW-THRUST ORBIT TRANSFER PROBLEM,” *Journal of Spacecraft and Rockets*, Vol. 57, No. 4, 2020, pp. 672–682. https://doi.org/10.2514/1.A34586.

[31] Whiffen, G., “Mystic: Implementation of the Static Dynamic Optimal Control Algorithm for High-Fidelity, Low-Thrust Trajectory Design,” *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, American Institute of Aeronautics and Astronautics, Keystone, Colorado, 2006. https://doi.org/10.2514/6.2006-6741, URL https://arc.aiaa.org/doi/10.2514/6.2006-6741.

[32] Mayne, D. H., and Jacobson, D. Q., *Differential dynamic programming*, American Elsevier Pub. Co., New York, NY, 1970.

[33] Lantoine, G., and Russell, R. P., “A Hybrid Differential Dynamic Programming Algorithm for Constrained Optimal Control Problems. Part 1: Theory,” *Journal of Optimization Theory and Applications*, Vol. 154, No. 2, 2012, pp. 382–417. https://doi.org/10.1007/s10957-012-0039-0, URL http://link.springer.com/10.1007/s10957-012-0039-0.

[34] Whiffen, G. J., “OPTIMIZING A USEFUL OBJECTIVE,” , Dec. 2002.

[35] Whiffen, G. J., “THRUST DIRECTION OPTIMIZATION: SATISFYING DAWN’S ATTITUDE AGILITY CONSTRAINTS,” Kauai, Hawaii, 2013.

[36] Hart, W., Brown, G. M., Collins, S. M., De Soria-Santacruz Pich, M., Fieseler, P., Goebel, D., Marsh, D., Oh, D. Y., Snyder, S., Warner, N., Whiffen, G., Elkins-Tanton, L. T., Bell, J. F., Lawrence, D. J., Lord, P., and Pirkl, Z., “Overview of the spacecraft design for the Psyche mission concept,” *2018 IEEE Aerospace Conference*, 2018, pp. 1–20. https://doi.org/10.1109/AERO.2018.8396444, URL https://ieeexplore.ieee.org/document/8396444.

[37] Aziz, J. D., Parker, J. S., Scheeres, D. J., and Englander, J. A., “Low-Thrust Many-Revolution Trajectory Optimization via Differential Dynamic Programming and a Sundman Transformation,” *The Journal of the Astronautical Sciences*, Vol. 65, No. 2, 2018, pp. 205–228. https://doi.org/10.1007/s40295-017-0122-8, URL http://link.springer.com/10.1007/s40295-017-0122-8.

[38] Aziz, J. D., Scheeres, D. J., and Lantoine, G., “Hybrid Differential Dynamic Programming in the Circular Restricted Three-Body Problem,” *Journal of Guidance, Control, and Dynamics*, Vol. 42, No. 5, 2019, pp. 963–975. https://doi.org/10.2514/1.G003617, URL https://arc.aiaa.org/doi/10.2514/1.G003617.

[39] Leemans, G., Carzana, L., and Heiligers, J., “Many-Revolution Earth-Centred Solar-Sail Trajectory Optimisation Using Differential Dynamic Programming,” *AIAA SCITECH 2022 Forum*, American Institute of Aeronautics and Astronautics, San Diego, CA & Virtual, 2022. https://doi.org/10.2514/6.2022-1776, URL https://arc.aiaa.org/doi/10.2514/6.2022-1776.

[40] Maestrini, M., “Hybrid Differential Dynamic Programming Algorithm for Low-Thrust Trajectory Design Using Exact High-Order Transition Maps,” 2018. URL https://www.semanticscholar.org/paper/Hybrid-Differential-Dynamic-Programming-Algorithm-Maestrini/6199f8e4be9350542bf46b6f1f4ea44da4121970.

[41] Pellegrini, E., and Russell, R. P., “A multiple-shooting differential dynamic programming algorithm. Part 1: Theory,” *Acta Astronautica*, Vol. 170, 2020, pp. 686–700. https://doi.org/10.1016/j.actaastro.2019.12.037, URL https://linkinghub.elsevier.com/retrieve/pii/S0094576519314705.

[42] Lantoine, G., and Russell, R. P., “A FAST SECOND-ORDER ALGORITHM FOR PRELIMINARY DESIGN OF LOW-THRUST TRAJECTORIES,” 2008.

[43] Lantoine, G., and Russell, R., “The Stark Model: An Exact, Closed-Form Approach to Low-Thrust Trajectory Optimization,” 2011. URL https://www.semanticscholar.org/paper/The-Stark-Model%3A-An-Exact%2C-Closed-Form-Approach-to-Lantoine-Russell/c4e846ef74b58060d384452c0b7265a79fbc55a8.

[44] Lin, T. C., and Arora, J. S., “Differential dynamic programming technique for constrained optimal control,” *Computational Mechanics*, 1991.

[45] Tassa, Y., Mansard, N., and Todorov, E., “Control-limited differential dynamic programming,” *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, Hong Kong, China, 2014, pp. 1168–1175. https://doi.org/10.1109/ICRA.2014.6907001, URL http://ieeexplore.ieee.org/document/6907001/.

[46] Carzana, L., Visser, P., and Heiligers, J., “Locally optimal control laws for Earth-bound solar sailing with atmospheric drag,” *Aerospace Science and Technology*, Vol. 127, 2022, p. 107666. https://doi.org/10.1016/j.ast.2022.107666, URL https://linkinghub.elsevier.com/retrieve/pii/S1270963822003406.

[47] Zheng, X., He, S., and Lin, D., “Constrained Trajectory Optimization With Flexible Final Time for Autonomous Vehicles,” *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 58, No. 3, 2022, pp. 1818–1829. https://doi.org/10.1109/TAES.2021.3121668, URL https://ieeexplore.ieee.org/document/9582791/.

[48] Lantoine, G., and Russell, R. P., “A Hybrid Differential Dynamic Programming Algorithm for Constrained Optimal Control Problems. Part 2: Application,” *Journal of Optimization Theory and Applications*, Vol. 154, No. 2, 2012, pp. 418–442. https://doi.org/10.1007/s10957-012-0038-1, URL http://link.springer.com/10.1007/s10957-012-0038-1.

[49] Bani Younes, A., "Exact Computation of High-Order State Transition Tensors for Perturbed Orbital Motion," *Journal of Guidance, Control, and Dynamics*, Vol. 42, No. 6, 2019, pp. 1365–1371. https://doi.org/10.2514/1.G003897, URL https://arc.aiaa.org/doi/10.2514/1.G003897.

[50] Boone, S., and McMahon, J., "Directional State Transition Tensors for Capturing Dominant Nonlinear Effects in Orbital Dynamics," *Journal of Guidance, Control, and Dynamics*, Vol. 46, No. 3, 2023, pp. 431–442. https://doi.org/10.2514/1.G006910, URL https://arc.aiaa.org/doi/10.2514/1.G006910.

[51] Bellman, R. E., *Dynamic Programming*, Princeton University Press, Princeton, 1957.

[52] Conn, A. R., Gould, N. I. M., and Toint, P. L., *Trust-region methods*, MPS-SIAM series on optimization, Society for Industrial and Applied Mathematics [u.a.], Philadelphia, Pa, 2000.

[53] Minnozzi, R., "https://github.com/ rikiminno/RiccardoMinnozzi_MScThesis," , Apr. 2025. URL https://github.com/rikiminno/RiccardoMinnozzi_MScThesis.

[54] Weinstein, M. J., and Rao, A. V., "A Source Transformation via Operator Overloading Method for the Automatic Differentiation of Mathematical Functions in MATLAB," *ACM Transactions on Mathematical Software*, Vol. 42, No. 2, 2016, pp. 1–44. https://doi.org/10.1145/2699456, URL https://dl.acm.org/doi/10.1145/2699456.

[55] Bertsekas, D. P., *Constrained Optimization and Lagrange Multiplier Methods*, Athena Scientific, 1996.

[56] Rao, A. V., Benson, D. A., Darby, C., Patterson, M. A., Francolin, C., Sanders, I., and Huntington, G. T., "Algorithm 902: GPOPS, A MATLAB software for solving multiple-phase optimal control problems using the gauss pseudospectral method," *ACM Trans. Math. Softw.*, Vol. 37, No. 2, 2010, pp. 22:1–22:39. https://doi.org/10.1145/1731022.1731032, URL https://dl.acm.org/doi/10.1145/1731022.1731032.

[57] Pellegrini, E., and Russell, R. P., "On the Computation and Accuracy of Trajectory State Transition Matrices," *Journal of Guidance, Control, and Dynamics*, Vol. 39, No. 11, 2016, pp. 2485–2499. https://doi.org/10.2514/1.G001920, URL https://arc.aiaa.org/doi/10.2514/1.G001920, publisher: American Institute of Aeronautics and Astronautics.

[58] Patel, P., and Scheeres, D. J., "A second-order optimization algorithm using quadric control updates for multistage optimal control problems," *Optimal Control Applications and Methods*, Vol. 30, No. 6, 2009, pp. 525–536. https://doi.org/10.1002/oca.876, URL https://onlinelibrary.wiley.com/doi/10.1002/oca.876.

[59] Saeidian, Z., Aminifard, Z., and Babaie–Kafaki, S., "A nonmonotone adaptive trust region technique with a forgetting factor," *International Journal of Computer Mathematics*, Vol. 101, No. 5, 2024, pp. 512–523. https://doi.org/10.1080/00207160.2024.2350447, URL https://www.tandfonline.com/doi/full/10.1080/00207160.2024.2350447.

[60] Gamez Losada, F., Visser, P., and Heiligers, M., "Fundamentals of Solar-Sail Transfers Around Planetary Bodies," *Proceedings of the 29th International Symposium on Space Flight Dynamics*, 2024.

[61] Mengali, G., Quarta, A. A., Circi, C., and Dachwald, B., "Refined Solar Sail Force Model with Mission Application," *Journal of Guidance, Control, and Dynamics*, Vol. 30, No. 2, 2007, pp. 512–520. https://doi.org/10.2514/1.24779, URL https://arc.aiaa.org/doi/10.2514/1.24779.

[62] Dachwald, B., Mengali, G., Quarta, A. A., and Macdonald, M., "Parametric Model and Optimal Control of Solar Sails with Optical Degradation," *Journal of Guidance, Control, and Dynamics*, Vol. 29, No. 5, 2006, pp. 1170–1178. https://doi.org/10.2514/1.20313, URL https://arc.aiaa.org/doi/10.2514/1.20313.

[63] Aziz, J., Scheeres, D., Parker, J., and Englander, J., "A Smoothed Eclipse Model for Solar Electric Propulsion Trajectory Optimization," *TRANSACTIONS OF THE JAPAN SOCIETY FOR AERONAUTICAL AND SPACE SCIENCES, AEROSPACE TECHNOLOGY JAPAN*, Vol. 17, 2019. https://doi.org/10.2322/tastj.17.181.

[64] Hintz, G. R., "Survey of Orbit Element Sets," *Journal of Guidance, Control, and Dynamics*, Vol. 31, No. 3, 2008, pp. 785–790. https://doi.org/10.2514/1.32237, URL https://arc.aiaa.org/doi/10.2514/1.32237, publisher: American Institute of Aeronautics and Astronautics.

[65] Sundman, K. F., "Mémoire sur le problème des trois corps," *Acta Mathematica*, Vol. 36, No. none, 1913, pp. 105–179. https://doi.org/10.1007/BF02422379, URL https://projecteuclid.org/journals/acta-mathematica/volume-36/issue-none/M%c3%a9moire-sur-le-probl%c3%a8me-des-trois-corps/10.1007/BF02422379.full, publisher: Institut Mittag-Leffler.

[66] Aziz, J. D., Scheeres, D., and Lantoine, G., "Differential Dynamic Programming in the Three-Body Problem," *2018 Space Flight Mechanics Meeting*, American Institute of Aeronautics and Astronautics, Kissimmee, Florida, 2018. https://doi.org/10.2514/6.2018-2223, URL https://arc.aiaa.org/doi/10.2514/6.2018-2223.

[67] Yakowitz, S., and Rutherford, B., "Computational aspects of discrete-time optimal control," *Applied Mathematics and Computation*, Vol. 15, No. 1, 1984, pp. 29–45. https://doi.org/10.1016/0096-3003(84)90051-1, URL https://www.sciencedirect.com/science/article/pii/0096300384900511.

[68] Pellegrini, E., and Russell, R. P., "A multiple-shooting differential dynamic programming algorithm. Part 2: Applications," *Acta Astronautica*, Vol. 173, 2020, pp. 460–472. https://doi.org/10.1016/j.actaastro.2019.12.038, URL https://www.sciencedirect.com/science/article/pii/S0094576519314717.

[69] Leipold, M., Eiden, M., Garner, C. E., Herbeck, L., Kassing, D., Niederstadt, T., Krüger, T., Pagel, G., Rezazad, M., Rozemeijer, H., Seboldt, W., Schöppinger, C., Sickinger, C., and Unckenbold, W., "Solar sail technology development and demonstration," *Acta Astronautica*, Vol. 52, No. 2, 2003, pp. 317–326. https://doi.org/10.1016/S0094-5765(02)00171-6, URL https://www.sciencedirect.com/science/article/pii/S0094576502001716.

[70] Oguri, K., and Lantoine, G., "Indirect trajectory optimization via solar sailing primer vector theory: Minimum solar-angle transfers," *Acta Astronautica*, Vol. 211, 2023, pp. 405–415. https://doi.org/10.1016/j.actaastro.2023.06.032, URL https://www.sciencedirect.com/science/article/pii/S0094576523003314.

[71] Marshall, M. A., and Pellegrino, S., “Slew Maneuver Constraints for Agile Flexible Spacecraft,” *Journal of Guidance, Control, and Dynamics*, Vol. 46, No. 12, 2023, pp. 2300–2314. https://doi.org/10.2514/1.G007430, URL https://arc.aiaa.org/doi/10.2514/1.G007430.

[72] Mceowen, S., Calderone, D. J., Tiwary, A., Zhou, J. S. K., Kim, T., Elango, P., and Acikmese, B., “Auto-tuned Primal-dual Successive Convexification for Hypersonic Reentry Guidance,” , Nov. 2024. https://doi.org/10.48550/arXiv.2411.08361, URL http://arxiv.org/abs/2411.08361, arXiv:2411.08361 [math].

[73] Powell, M. J. D., “Algorithms for nonlinear constraints that use lagrangian functions,” *Mathematical Programming*, Vol. 14, No. 1, 1978, pp. 224–248. https://doi.org/10.1007/BF01588967, URL http://link.springer.com/10.1007/BF01588967.

5 Conclusions and Recommendations

This thesis aimed at developing a flexible Hybrid Differential Dynamic Programming optimization framework, integrating flexible-final-time handling and path constraints into algorithm's formulation, and reducing/assessing the effects of algorithm hyper-parameters. Additionally, an initial characterization of circular-to-circular many-revolutions solar sail transfers around Earth was also tackled. Conclusions drawn from the performed research are summarized in Section 5.1, while Section 5.2 outlines possible paths for future research.

5.1. Conclusions

The conclusions are presented in the form of answers to the research questions presented at the end of Chapter 2, as well as a final reflection on the research objective.

- RQ1: *How can a robust and flexible optimization framework, aimed at optimizing many-revolutions solar sail transfers, be defined using HDDP?*

  The optimization algorithm presented in this thesis expands on the HDDP framework of [42]. Following such formulation, the solver is verified to work for multi-phase, constrained optimal control problems, with parametrized initial conditions. By expanding the algorithm formulation to also account for variable-duration, path-constrained problems, the resulting solver can be applied to a broad class of optimal control problems. The flexibility of the approach is enhanced through Object-Oriented Programming-based software architecture, accommodating modular changes/replacements for task-specific solvers. The HDDP algorithm also de-couples partial derivatives of dynamics and cost/constraints functions: by leveraging automatic differentiation, quick iterations between different problem formulations are greatly simplified. The introduction of an arbitrary stage collocation function also enables further control over problem formulation.
  The optimization framework is observed to be robust to poor control and hyper-parameter guesses, namely in terms of penalty parameter σ and quadratic model validity threshold ϵ_1. The introduced safeguard on trust-region ”stalling” in Alg. 5, as well as the adoption of a robust trust-region solver in Alg. 2, ensure convergence under poorly tuned hyper-parameters, alleviating the time-consuming tuning process. Numerical instabilities are observed when tackling OCPs with high-resolution and very low control authority: additional safeguards are suggested to address this shortcoming.

  - RQ1.1: *How can time of flight be included as a decision variable within the HDDP optimization algorithm?*

    As introduced in Subsection 3.2.1, two approaches where considered for the implementation of a flexible-final-time HDDP. The choice, driven by accuracy and flexibility requirements, lies on a time-dilation approach: parametrizing the problem discretization, its duration can be included as a (static) decision variable. This practice requires the State Transition Map approach to also account for changes in stage collocation, therefore variational equations are modified accordingly (resulting in Eq. 3.19 and Eq. 3.20). The technique is successfully applied to time-optimal many-revolution Optimal Control Problems in Chapter 4. The devised stage collocation approach also enables further studies into adaptive mesh techniques.

- RQ1.2: *How can path constraints be enforced reliably and efficiently within the HDDP optimization framework?*

  Section 2.3 presents a trade-off between different approaches to implement constrained optimization within Trust-Region Quadratic Programming stage sub-problems (i.e., path constraints). Given the driving requirements of accuracy and efficiency, the chosen approach is a quadratic approach to the Karush-Kuhn-Tucker conditions: leveraging a quadratic path-constraints model, these are enforced up to second-order around the current solution point. Since the resulting KKT system is linear, efficient solution techniques can be leveraged to quickly solve constrained TRQP sub-problems. Using a quadratic approximation, the algorithm offers exact convergence for both linear (i.e.: control bounds) and quadratic (i.e.: distance or vector norm constraints) constraint functions.

- RQ1.3: *How can HDDP sensitivity to its hyper-parameters be reduced?*

  Two approaches were devised to reduce HDDP hyper-parameter sensitivities, as introduced in Section 3.2. The trust-region relaxation technique in Eq. 3.88 was shown to efficiently 'push' final algorithm iterations towards more optimal and smooth solutions. Sensitivity to the $\Delta_{min},\ \epsilon_1$ coupling is also reduced through the quadratic model validity adaptive enlargement in Alg. 5. Effects caused by penalty parameter tuning were shown to be mitigated by the automatic-tuning procedure in Eq. 3.86. Due to its formulation, the novel approach also implies tuning efforts (although with considerably reduced sensitivity): a solution was identified in the nested-trust-region approach in Alg. 4. Despite its effectiveness, the technique is deemed too computationally demanding for application to high-dimensional OCPs, thus further work on the adaptive penalty parameter tuning is encouraged. Sensitivity to coarse discretization was also reduced by introducing a mesh refinement procedure. On top of the active measures mentioned above, a "rule-set" (found in Chapter 4) for HDDP hyper-parameter sensitivity was derived, enabling informed decisions for algorithm tuning.

- RQ2: *How does the defined optimization algorithm perform, when applied to the identification of time-optimal many-revolution solar-sail transfers?*

  The convergence properties of the algorithm are verified on a set of many-revolutions transfer cases. The computational cost of every iteration is shown to increase linearly, as expected, certifying the algorithm viability in the optimization of high-dimensional optimal control problems. A convergence order analysis of the algorithm is also performed, highlighting linear convergence properties dictated by the trust-region technique. Using the devised algorithm, many-revolutions transfers are successfully optimized in different settings, up to 180 revolutions in a high-control-authority environment (i.e., GEO altitude) and up to 1000 revolutions in a low-control-authority setting (i.e., LEO altitude). Power regression models are derived to link orbit-increase performance with transfer duration: as expected, regression coefficients reflect the orbit illumination conditions (i.e., linearity is noticed in fully illuminated arcs, while sub-linear trends are introduced by eclipsing conditions).

  - RQ2.1: *What are the effects of different orbital regimes on optimal many-revolution circular-to-circular transfers?*

    Both many-revolutions solutions at GEO and LEO altitudes are showcased. The different time scales imply that the solutions adjust differently to the control requirements dictated by orbital motion and changes in the Sun-Earth configuration. In both cases, two different control regimens are identified, depending on the current orbit illumination conditions (referred to as *ion-drive* and *solar sail* regimes). The long time scale of transfers at high altitudes (in the high-control-authority setting) determines considerable variations in the Earth-Sun configuration throughout a transfer: the control profile is shown to adapt to such changes, by modifying orbital geometry such that the full power of SSA can be exploited at apogee, efficiently performing orbit-circularization (similarly to the Hohmann transfer "apogee-burn" maneuver). In the lower-control-authority setting, the dominance of eclipses determines a different adjustment to the changing illumination condition, with orbital geometry maximizing the illuminated region close to apogee to efficiently circularize the final orbit. The high-control-authority solution is shown to satisfy attitude-rate constraints (despite not being included in the problem formulation), while the low-control-authority setting is shown to exceed such limitations, hence requiring further investigation into enforcing control-limited solutions.

– RQ2.2: *How do variable-time solutions compare to fixed-time circular-to-circular transfers?*

The power regression model derived from the identified many-revolutions solutions is used to initialize the search for a time-optimal solution to circular-to-circular transfers in the low-control-authority setting. The time-optimal solution displays similar features to its fixed-time counterparts (with distinct control regimes and exceeding of slew-rate constraints), further certifying the validity of the flexible-final-time approach derived as part of this thesis. The minimum time identified to perform the transfer is close to the power regression model prediction, certifying its applicability. The accuracy of the power regression model can be significantly improved by introducing more solutions in the eclipsed portion of the model, as the solar-sail performance in the circular-to-circular transfer problem is greatly reduced in eclipsed regions.

5.2. Recommendations

The outcome of this work is a general-purpose HDDP solver, enhanced with an automatic differentiation framework to enable quick and easy iterations between different problem formulations. The optimization solver accommodates multi-phase, constrained, and variable-duration OCPs. As such, it is envisioned for extensive further use: given the high-dimensionality of solar-sail many-revolution transfers, the devised algorithm provides an appealing technique to investigate optimal solutions to this class of problems.

The dynamical model adopted to investigate time-optimal many-revolutions transfers is limited to in-plane effects and includes several additional simplifying assumptions. It is natural for further works to dive into higher-fidelity solutions, starting from 3D dynamics and expanding the dynamical model to include additional effects (i.e., higher-order gravity terms, aerodynamic perturbations, third-body gravitational effects). As HDDP requires two times differentiable functions, the differentiability of aerodynamical models (which typically depend on tabulated data) is to be addressed.

Given the flexibility of the approach, more relevant/realistic mission scenarios can be defined in further studies. The most documented many-revolution transfer scenario in previous works (though limited to EP-powered spacecraft) is the GTO to GEO orbital transfer case: although already characterized in [59], the HDDP algorithm would allow a higher-fidelity orbit representation by removing the need for orbit-averaging and control parametrization techniques. Additional relevant mission scenarios include active debris removal missions, which can be defined as multi-phase OCPs with rendezvous constraints, hence greatly benefiting from the algorithm's flexible-final-time handling capabilities.

The advantageous flexibility properties of the devised framework greatly enable changes to dynamical model formulation. The property can be leveraged to perform an extensive analysis of the effects of different state/control representations on both convergence properties and solution optimality on a specified many-revolutions solar-sail-powered benchmark problem. While similar comparisons are available for EP-powered many-revolution transfers, solar-sail problems have not been characterized to such an extent.

While the devised HDDP framework can be applied to a wide class of OCPs, the main focus of this thesis is the algorithm development itself. For this reason, the following subsections present potential improvements to address limitations in the developed algorithm, concluding with insights into potential research outlook to greatly advance the HDDP algorithm capabilities.

5.2.1. Differential Dynamic Programming algorithm

Several recommendations for future developments are already defined in Section 3.2 when illustrating the algorithm's full formulation. More specifically, these include the further expansion of the adaptive penalty parameter tuning approach in Subsection 3.2.7 (beyond the nested-trust-region approach in

Alg. 4), the extension of the flexible-final-time handling technique of [67] to the HDDP framework, the introduction of uncertainty information (which can be easily propagated thanks to the available STMs) to define a robust optimization framework, and the introduction of safeguards when defining the TRQP sub-problems optimal feedback laws. Additional recommendations for specific elements of the HDDP algorithm are now introduced.

State transition maps

Being the most computationally demanding step for the full HDDP procedure, the STMs propagation is an essential element for further works. Following [86], semi-analytical propagation schemes can be introduced, greatly reducing the runtime burden of HDDP optimization. Available semi-analytical propagators for low-thrust spacecraft dynamics are the Kepler [104] and Stark [87] models, both known to be twice differentiable and applied to solar-sail transfer problems.

The HDDP algorithm is limited to a second-order model of the cost function at every stage. The expansion to higher-order models (by exploiting STMs of higher-orders with respect to the second-order ones adopted by HDDP) offers a promising perspective for future works: it is known that higher-order derivative information improves algorithms convergence properties [42], but also introduces computational overhead. The higher-order STMs can be leveraged to obtain not only more precise TRQP sub-problem update laws, but also to provide a more accurate representation of the system's dynamics, enabling larger trust-region radius and therefore faster convergence. Different techniques are available for the propagation of higher-order cost models, from differential algebra to complex-step differentiation.

Scaling

When solving variable-duration OCPs, the problem sensitivity to updates in static parameters (i.e., time of flight) is considerably higher than stage-wise control sensitivities. The resulting OCP can suffer from scaling issues, with the quadratic model validity being entirely dominated by static parameters: in these conditions, strict trust-region radius Δ values are required to maintain reasonable validity in the quadratic model, consequently limiting control updates to be within the same Δ range, thus leading to slow convergence. An automatic scaling logic, aimed at balancing cost sensitivities throughout the multiple HDDP steps, is recommended to define a more robust solver, reducing the tuning efforts required to balance cost sensitivities among decision variables.

Performance improvements

The HDDP framework developed as part of this work is entirely implemented in MATLAB ®. The chosen platform provides a solid foundation for prototyping and initial developments, however lower-level programming languages (i.e., `C++` or hardware-accelerated `Python`) can be adopted to speed up computations, thus addressing the runtime limitations from which HDDP suffers. Switching to such lower-level languages could also enable more effective exploitation of parallel computing capabilities, even on low-performance computing platforms such as personal laptops.

Mesh refinement

The HDDP runtime limitations can also be addressed through a mesh refinement technique. The mesh refinement introduced in this work is very basic and quickly becomes ineffective if initialized with an untested number of stages. A significantly more promising approach is to perform mesh refinement only between stages where optimal controls differ significantly, hence increasing resolution only where needed. Thanks to the stage collocation function introduced in Subsection 3.2.1, a similar result can be achieved through specific formulations of the stage collocation function, with the advantage of not introducing additional stages to the discretization.

The HDDP algorithm implicitly assumes the controls over each stage to be constant, thus enabling the use of STMs to efficiently perform the backward induction. The limitations of this assumption can be reduced by introducing a control parametrization over each stage, where the dynamic controls become the coefficients for the defined parametrization. The approach can be combined with an adaptive-mesh algorithm, similar to Reference [57], to trade-off the complexity introduced by higher-order parametrizations with the gains in computational efficiency obtained by reducing the number of stages.

Guess generation capabilities

While the HDDP algorithm can be used as an effective standalone OCP solver, its features make it a promising tool for initial-guess generation. More specifically, thanks to the chosen path-constraints enforcement approach, Lagrange multipliers values for all constraints (both path and terminal constraints) are available as part of the algorithm solution and can be used to quickly re-converge a full solution using an NLP solver. The algorithm also provides the potential to generate initial guesses for indirect optimization methods: since the backward induction propagates a quadratic cost model, the initial cost sensitivities (obtained from Eq. 3.32 at the first stage of the first phase) can be used as an initial guess for the problem co-states [44].

5.2.2. Research outlook

Despite the preliminary nature of the work, the devised HDDP implementation provides the potential for a complete and efficient optimization solver. More specifically, integrating the defined framework with an adaptive mesh technique (similarly to the GPOPS-II algorithm in Reference [57]) can result in a solver capable of tackling high-dimensional OCPs (avoiding the ”curse of dimensionality” that affects direct solvers [43]) while addressing the long runtime required for HDDP iterations. The proposed approach is an expansion of the higher-order cost model concept previously mentioned.

The theorized approach shall leverage higher-order derivatives of the cost model, exploiting the more accurate dynamical prediction model to considerably speed up the forward pass (by applying the STMs similarly as in Eq. 3.8) as well as the STMs propagation. Performing the STMs propagation leveraging entirely STMs information implies the loss of one order of derivative information (i.e., using a third-order STMs model, only STMs up to second-order can be computed). Repeating the procedure allows to conduct several HDDP iterations without the need for numerical integration, greatly speeding up the algorithm. An adaptive mesh procedure, similar to the $h - p$ technique described in Reference [57], is to be defined to establish the STMs order required to achieve a specified degree of accuracy. The research outcome is an HDDP-based OCP solver with the advantageous scaling properties typical of DDP algorithms, and the promising runtime performance of direct transcription methods with adaptive collocation.

References

[1] Whiffen, G. J. (2002, December). *OPTIMIZING A USEFUL OBJECTIVE* (US 6,496,741 B1).

[2] Hollerman, W. A. (2003). The Physics of Solar Sails.

[3] McInnes, C. R., Macdonald, M., Angelopolous, V., & Alexander, D. (2001). GEOSAIL: Exploring the Geomagnetic Tail Using a Small Solar Sail. *Journal of Spacecraft and Rockets*, *38*(4), 622–629. https://doi.org/10.2514/2.3727

[4] Kelly, P. W., Bevilacqua, R., Mazal, L., & Erwin, R. S. (2018). TugSat: Removing Space Debris from Geostationary Orbits Using Solar Sails. *Journal of Spacecraft and Rockets*, *55*(2), 437–450. https://doi.org/10.2514/1.A33872

[5] Gulkis, S., & de Pater, I. (2003, January). Radio Astronomy, Planetary. In R. A. Meyers (Ed.), *Encyclopedia of Physical Science and Technology (Third Edition)* (pp. 687–712). Academic Press. https://doi.org/10.1016/B0-12-227410-5/00637-2

[6] McInnes, C. R. (1999, January). *Solar sailing. Technology, dynamics and mission applications.* [Publication Title: Solar sailing. Technology ADS Bibcode: 1999sstd.book.....M]. Retrieved April 17, 2024, from https://ui.adsabs.harvard.edu/abs/1999sstd.book.....M

[7] Dachwald, B., Mengali, G., Quarta, A. A., & Macdonald, M. (2006). Parametric Model and Optimal Control of Solar Sails with Optical Degradation. *Journal of Guidance, Control, and Dynamics*, *29*(5), 1170–1178. https://doi.org/10.2514/1.20313

[8] MacNeal, R. H. (1967, March). *The Heliogyro - an Interplanetary Flying Machine* (tech. rep. No. NASA-CR-84460) (NTRS Author Affiliations: Astro Research Corp. NTRS Document ID: 19670018298 NTRS Research Center: Headquarters (HQ)). Retrieved April 18, 2024, from https://ntrs.nasa.gov/citations/19670018298

[9] Fieseler, P. D. (1998). A method for solar sailing in a low earth orbit. *Acta Astronautica*, *43*(9-10), 531–541. https://doi.org/10.1016/S0094-5765(98)00175-1

[10] Sawada, H., Mori, O., Okuizumi, N., Shirasawa, Y., Miyazaki, Y., Natori, M., Matunaga, S., Furuya, H., & Sakamoto, H. (2011). Mission Report on The Solar Power Sail Deployment Demonstration of IKAROS. *52nd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*. https://doi.org/10.2514/6.2011-1887

[11] Gong, S., & Macdonald, M. (2019). Review on solar sail technology. *Astrodynamics*, *3*(2), 93–125. https://doi.org/10.1007/s42064-019-0038-x

[12] Saiki, T., Tsuda, Y., Funase, R., Mimasu, Y., Shirasawa, Y., & Ikaros Demonstration Team. (2012). Attitude Operation Results of Solar Sail Demonstrator IKAROS. *TRANSACTIONS OF THE JAPAN SOCIETY FOR AERONAUTICAL AND SPACE SCIENCES, AEROSPACE TECHNOLOGY JAPAN*, *10*(ists28), To_4_1–To_4_6. https://doi.org/10.2322/tastj.10.To_4_1

[13] Mori, O., Shirasawa, Y., Mimasu, Y., Tsuda, Y., Sawada, H., Saiki, T., Yamamoto, T., Yonekura, K., Hoshino, H., Kawaguchi, J., & Funase, R. (2014). Overview of IKAROS Mission. In M. Macdonald (Ed.), *Advances in Solar Sailing* (pp. 25–43). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-34907-2_3

[14] Tsuda, Y., Okano, Y., Mimasu, Y., & Funase, R. (2012). On-orbit sail quality evaluation utilizing attitude dynamics of spinner solar sailer IKAROS. *ResearchGate*, *143*, 1609–1625. Retrieved March 6, 2025, from https://www.researchgate.net/publication/291323306_On-orbit_sail_quality_evaluation_utilizing_attitude_dynamics_of_spinner_solar_sailer_IKAROS

[15] Alhorn, D., Casas, J., Agasid, E., Adams, C., Laue, G., Kitts, D. C., & O'Brien, S. (2011). NanoSail-D: The Small Satellite That Could!

[16] Krebs, G. D. (n.d.). NanoSail D. Retrieved April 22, 2024, from https://space.skyrocket.de/doc_sdat/nanosail-d.htm

[17] Spencer, D. A., Betts, B., Bellardo, J. M., Diaz, A., Plante, B., & Mansell, J. R. (2021). The LightSail 2 solar sailing technology demonstration. *Advances in Space Research*, *67*(9), 2878–2889. https://doi.org/10.1016/j.asr.2020.06.029

[18] Ridenoure, R. W., Spencer, D. A., Stetson, D. A., Betts, B., Munakata, R., Wong, S. D., Diaz, A., Plante, B., Foley, J. D., & Bellardo, J. M. (2015). Status of the Dual CubeSat LightSail Program. *AIAA SPACE 2015 Conference and Exposition*. https://doi.org/10.2514/6.2015-4424

[19] Betts, B., Spencer, D., Nye, B., Munakata, R., Bellardo, J., Wong, S. D., Diaz, A., Ridenoure, R., Plante, B., Foley, J., & Vaughn, J. (2016). LightSail 2 : Controlled Solar Sailing Using a CubeSat. Retrieved April 18, 2024, from https://www.semanticscholar.org/paper/LightSail-2-%3A-Controlled-Solar-Sailing-Using-a-Betts-Spencer/5f2e8819b1dc9cccc38dd895abd37c75479b93d5

[20] Mansell, J., Spencer, D. A., Plante, B., Diaz, A., Fernandez, M., Bellardo, J., Betts, B., & Nye, B. (2020, January). Orbit and Attitude Performance of the LightSail 2 Solar Sail Spacecraft. In *AIAA Scitech 2020 Forum*. American Institute of Aeronautics; Astronautics. https://doi.org/10.2514/6.2020-2177

[21] Lockett, T. R., Castillo-Rogez, J., Johnson, L., Matus, J., Lightholder, J., Marinan, A., & Few, A. (2020). Near-Earth Asteroid Scout Flight Mission. *IEEE Aerospace and Electronic Systems Magazine*, *35*(3), 20–29. https://doi.org/10.1109/MAES.2019.2958729

[22] Wilkie, W. K. (2021). Overview of the NASA Advanced Composite Solar Sail System (ACS3) Technology Demonstration Project. *AIAA Scitech 2021 Forum*. https://doi.org/10.2514/6.2021-1260

[23] Carzana, L., Minervino Amodio, A., Visser, P., Wilkie, W. K., & Heiligers, M. (2024). Calibration Steering Laws to Estimate the Optical Properties of NASA's ACS3 Solar Sail: 29th International Symposium on Space Flight Dynamics. *Proceedings of the 29th International Symposium on Space Flight Dynamics*.

[24] Johnson, L., Young, R., Barnes, N., Friedman, L., Lappas, V., & McInnes, C. (2012). Solar Sails: Technology And Demonstration Status. *International Journal of Aeronautical and Space Sciences*, *13*(4), 421–427. https://doi.org/10.5139/IJASS.2012.13.4.421

[25] Office, E. S. D. (2024, July). ESA Space Environment Report 2024. Retrieved December 5, 2024, from https://www.esa.int/Space_Safety/Space_Debris/ESA_Space_Environment_Report_2024

[26] Colombo, C., Lücking, C., & McInnes, C. R. (2012). Orbital dynamics of high area-to-mass ratio spacecraft with $J2$ and solar radiation pressure for novel Earth observation and communication services. *Acta Astronautica*, *81*(1), 137–150. https://doi.org/10.1016/j.actaastro.2012.07.009

[27] Leipold, M. E., & Wagner, O. (1996). Mercury sun-synchronous polar orbits using solar sail propulsion. *Journal of Guidance, Control, and Dynamics*, *19*(6), 1337–1341. https://doi.org/10.2514/3.21791

[28] Macdonald, M., & McInnes, C. R. (2005a). Analytical Control Laws for Planet-Centered Solar Sailing. *Journal of Guidance, Control, and Dynamics*, *28*(5), 1038–1048. https://doi.org/10.2514/1.11400

[29] Bonnal, C., Ruault, J.-M., & Desjean, M.-C. (2013). Active debris removal: Recent progress and current trends. *Acta Astronautica*, *85*, 51–60. https://doi.org/10.1016/j.actaastro.2012.11.009

[30] Delft, T. (n.d.). SWEEP. Retrieved April 22, 2024, from https://www.tudelft.nl/en/ae/sweep

[31] Carzana, L., Visser, P., & Heiligers, M. (2021). Solar-sail control laws for perturbed Earth-bound trajectories: 72nd International Astronautical Conference. *72nd International Astronautical Conference*.

[32] Conway, B. A. (2012). A Survey of Methods Available for the Numerical Optimization of Continuous Dynamic Systems. *Journal of Optimization Theory and Applications*, *152*(2), 271–306. https://doi.org/10.1007/s10957-011-9918-z

[33] Morante, D., Sanjurjo Rivo, M., & Soler, M. (2021). A Survey on Low-Thrust Trajectory Optimization Approaches. *Aerospace*, *8*(3), 88. https://doi.org/10.3390/aerospace8030088

[34] Gardi, A., Sabatini, R., & Ramasamy, S. (2016). Multi-objective optimisation of aircraft flight trajectories in the ATM and avionics context. *Progress in Aerospace Sciences*, *83*, 1–36. https://doi.org/10.1016/j.paerosci.2015.11.006

[35] Chai, R., Savvaris, A., Tsourdos, A., Chai, S., & Xia, Y. (2019). A review of optimization techniques in spacecraft flight trajectory design. *Progress in Aerospace Sciences*, *109*, 100543. https://doi.org/10.1016/j.paerosci.2019.05.003

[36] Aziz, J. D., Parker, J. S., Scheeres, D. J., & Englander, J. A. (2018). Low-Thrust Many-Revolution Trajectory Optimization via Differential Dynamic Programming and a Sundman Transformation.

The Journal of the Astronautical Sciences, *65*(2), 205–228. https://doi.org/10.1007/s40295-017-0122-8

[37] Oguri, K., & Lantoine, G. (2023). Indirect trajectory optimization via solar sailing primer vector theory: Minimum solar-angle transfers. *Acta Astronautica*, *211*, 405–415. https://doi.org/10.1016/j.actaastro.2023.06.032

[38] Oguri, K., Lantoine, G., Petropoulos, A. E., & McMahon, J. W. (2023). Solar Sailing Q-Law for Planetocentric, Many-Revolution Sail Orbit Transfers [Publisher: American Institute of Aeronautics and Astronautics _eprint: https://doi.org/10.2514/1.G007103]. *Journal of Guidance, Control, and Dynamics*, *46*(10), 2005–2014. https://doi.org/10.2514/1.G007103

[39] Leemans, G., Carzana, L., & Heiligers, J. (2022). Many-Revolution Earth-Centred Solar-Sail Trajectory Optimisation Using Differential Dynamic Programming. *AIAA SCITECH 2022 Forum*. https://doi.org/10.2514/6.2022-1776

[40] Martens, R. (2023, February). *Differential Dynamic Programming applied to Interplanetary Solar-Sail Trajectory Optimization* [Doctoral dissertation, TU Delft].

[41] Shannon, J. L., Ozimek, M. T., & Atchison, J. A. (2020). Q-LAW AIDED DIRECT TRAJECTORY OPTIMIZATION FOR THE HIGH-FIDELITY, MANY-REVOLUTION LOW-THRUST ORBIT TRANSFER PROBLEM. *Journal of Spacecraft and Rockets*, *57*(4), 672–682. https://doi.org/10.2514/1.A34586

[42] Lantoine, G., & Russell, R. P. (2012a). A Hybrid Differential Dynamic Programming Algorithm for Constrained Optimal Control Problems. Part 1: Theory. *Journal of Optimization Theory and Applications*, *154*(2), 382–417. https://doi.org/10.1007/s10957-012-0039-0

[43] Mayne, D. H., & Jacobson, D. Q. (1970). *Differential dynamic programming*. American Elsevier Pub. Co.

[44] Longuski, J. M., Guzmán, J. J., & Prussing, J. E. (2014). *Optimal Control with Aerospace Applications*. Springer New York. https://doi.org/10.1007/978-1-4614-8945-0

[45] E, Z., & Guzzetti, D. (2020). Multi-revolution low-thrust trajectory optimization using symplectic methods. *Science China Technological Sciences*, *63*(3), 506–519. https://doi.org/10.1007/s11431-019-9511-7

[46] Haberkorn, T., Martinon, P., & Gergaud, J. (2004). Low Thrust Minimum-Fuel Orbital Transfer: A Homotopic Approach [Publisher: American Institute of Aeronautics and Astronautics]. *Journal of Guidance, Control, and Dynamics*, *27*(6), 1046–1060. https://doi.org/10.2514/1.4022

[47] Edelbaum, T. N. (1971). Optimal Nonplanar Escape from Circular Orbits [Publisher: American Institute of Aeronautics and Astronautics]. *AIAA Journal*, *9*(12), 2432–2436. https://doi.org/10.2514/3.50047

[48] Wiesel, W. E., & Alfano, S. (1985). Optimal many-revolution orbit transfer. *Journal of Guidance, Control, and Dynamics*, *8*(1), 155–157. https://doi.org/10.2514/3.19952

[49] Sackett, L. L. (1977, September). *Optimal Solar Sail Planetocentric Trajectories* (tech. rep.). The Charles Stark Draper Laboratory, Inc. Cambridge, Massachusetts.

[50] Macdonald, M., & McInnes, C. R. (2005b). Realistic Earth Escape Strategies for Solar Sailing. *Journal of Guidance, Control, and Dynamics*, *28*(2), 315–323. https://doi.org/10.2514/1.5165

[51] Carzana, L., Visser, P., & Heiligers, J. (2022). Locally optimal control laws for Earth-bound solar sailing with atmospheric drag. *Aerospace Science and Technology*, *127*, 107666. https://doi.org/10.1016/j.ast.2022.107666

[52] Barles, A., Ceriotti, M., Ciampa, F., & Felicetti, L. (2021). An optimal steering law for sailing with solar and planetary radiation pressure. *Aerospace Science and Technology*, *118*, 107051. https://doi.org/10.1016/j.ast.2021.107051

[53] Rao, A. (2010). A Survey of Numerical Methods for Optimal Control. *Advances in the Astronautical Sciences*, *135*.

[54] Topputo, F., & Zhang, C. (2014). Survey of Direct Transcription for Low-Thrust Space Trajectory Optimization with Applications. *Abstract and Applied Analysis*, *2014*, 1–15. https://doi.org/10.1155/2014/851720

[55] Betts, J. T. (1998). Survey of Numerical Methods for Trajectory Optimization [Publisher: American Institute of Aeronautics and Astronautics]. *Journal of Guidance, Control, and Dynamics*, *21*(2), 193–207. https://doi.org/10.2514/2.4231

[56] Betts, J. T. (2000). Very low-thrust trajectory optimization using a direct SQP method. *Journal of Computational and Applied Mathematics*, *120*(1), 27–40. https://doi.org/10.1016/S0377-0427(00)00301-0

[57] Rao, A. V., Benson, D. A., Darby, C., Patterson, M. A., Francolin, C., Sanders, I., & Huntington, G. T. (2010). Algorithm 902: GPOPS, A MATLAB software for solving multiple-phase optimal control problems using the gauss pseudospectral method. *ACM Trans. Math. Softw.*, *37*(2), 22:1–22:39. https://doi.org/10.1145/1731022.1731032

[58] Graham, K. F., & Rao, A. V. (2015). Minimum-Time Trajectory Optimization of Multiple Revolution Low-Thrust Earth-Orbit Transfers. *Journal of Spacecraft and Rockets*, *52*(3), 711–727. https://doi.org/10.2514/1.A33187

[59] Fitzgerald, R. M. (2021). Characterizing Minimum-Time Solar Sail Geostationary Orbit Transfers Using Pseudospectral Optimal Control. *Journal of Spacecraft and Rockets*, *58*(4), 997–1009. https://doi.org/10.2514/1.A34950

[60] Petropoulos, A. (2004). Low-Thrust Orbit Transfers Using Candidate Lyapunov Functions with a Mechanism for Coasting. *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*. https://doi.org/10.2514/6.2004-5089

[61] Bianchi, C., Niccolai, L., Mengali, G., & Ceriotti, M. (2024). Preliminary design of a space debris removal mission in LEO using a solar sail. *Advances in Space Research*, *73*(8), 4254–4268. https://doi.org/10.1016/j.asr.2024.01.024

[62] Petropoulos, A. E. (2005). REFINEMENTS TO THE Q-LAW FOR LOW-THRUST ORBIT TRANSFERS. *15th AAS/AIAA Space Flight Mechanics Conference*, *120*, 963–982.

[63] Kelly, P., & Bevilacqua, R. (2019). An optimized analytical solution for geostationary debris removal using solar sails. *Acta Astronautica*, *162*, 72–86. https://doi.org/10.1016/j.actaastro.2019.05.055

[64] Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press.

[65] Liao, L.-Z., & Shoemaker, C. (1993). Advantages of Differential Dynamic Programming Over Newton's Method for Discrete-Time Optimal Control Problems.

[66] Yakowitz, S. (1989). Algorithms and Computational Techniques in Differential Dynamic Programming. In *Control and Dynamic Systems* (pp. 75–91, Vol. 31). Elsevier. https://doi.org/10.1016/B978-0-12-012731-3.50008-1

[67] Zheng, X., He, S., & Lin, D. (2022). Constrained Trajectory Optimization With Flexible Final Time for Autonomous Vehicles. *IEEE Transactions on Aerospace and Electronic Systems*, *58*(3), 1818–1829. https://doi.org/10.1109/TAES.2021.3121668

[68] Giannessi, F. (2005). *Constrained Optimization and Image Space Analysis: Volume 1: Separation of Sets and Optimality Conditions*. Springer US. https://doi.org/10.1007/0-387-28020-0

[69] Kuhn, H. W., & Tucker, A. W. (1951, January). Nonlinear Programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability* (pp. 481–493, Vol. 2). University of California Press. Retrieved April 25, 2024, from https://projecteuclid.org/ebooks/berkeley-symposium-on-mathematical-statistics-and-probability/Proceedings-of-the-Second-Berkeley-Symposium-on-Mathematical-Statistics-and/chapter/Nonlinear-Programming/bsmsp/1200500249

[70] Tassa, Y., Mansard, N., & Todorov, E. (2014). Control-limited differential dynamic programming. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 1168–1175. https://doi.org/10.1109/ICRA.2014.6907001

[71] Pellegrini, E., & Russell, R. P. (2020a). A multiple-shooting differential dynamic programming algorithm. Part 1: Theory. *Acta Astronautica*, *170*, 686–700. https://doi.org/10.1016/j.actaastro.2019.12.037

[72] Patel, P., & Scheeres, D. J. (2009). A second□order optimization algorithm using quadric control updates for multistage optimal control problems. *Optimal Control Applications and Methods*, *30*(6), 525–536. https://doi.org/10.1002/oca.876

[73] Conn, A. R., Gould, N. I. M., & Toint, P. (1991). A Globally Convergent Augmented Lagrangian Algorithm for Optimization with General Constraints and Simple Bounds [Publisher: Society for Industrial and Applied Mathematics]. *SIAM Journal on Numerical Analysis*, *28*(2), 545–572. https://doi.org/10.1137/0728030

[74] Bertsekas, D. P. (1996). *Constrained Optimization and Lagrange Multiplier Methods*. Athena Scientific.

[75] Conn, A. R., Gould, N. I. M., & Toint, P. L. (2000). *Trust-region methods*. Society for Industrial; Applied Mathematics [u.a.]

[76] Lin, T. C., & Arora, J. S. (1991). Differential dynamic programming technique for constrained optimal control. *Computational Mechanics*.

[77] Colombo, C., Vasile, M., & Radice, G. (2009). Optimal low-thrust trajectories to asteroids through an algorithm based on differential dynamic programming. *Celestial Mechanics and Dynamical Astronomy*, *105*(1-3), 75–112. https://doi.org/10.1007/s10569-009-9224-3

[78] Ozaki, N., Campagnola, S., & Funase, R. (2020). Tube Stochastic Optimal Control for Nonlinear Constrained Trajectory Optimization Problems. *Journal of Guidance, Control, and Dynamics*, *43*(4), 645–655. https://doi.org/10.2514/1.G004363

[79] Julier, S., & Uhlman, J. K. (1996). *A General Method for Approximating Nonlinear Transformations of Probability Distributions* (tech. rep.). University of Oxford. Oxford.

[80] Lantoine, G., & Russell, R. P. (2012b). A Hybrid Differential Dynamic Programming Algorithm for Constrained Optimal Control Problems. Part 2: Application. *Journal of Optimization Theory and Applications*, *154*(2), 418–442. https://doi.org/10.1007/s10957-012-0038-1

[81] Coleman, T. F., & Li, Y. (1996). An Interior Trust Region Approach for Nonlinear Minimization Subject to Bounds [Publisher: Society for Industrial and Applied Mathematics]. *SIAM Journal on Optimization*, *6*(2), 418–445. https://doi.org/10.1137/0806023

[82] Maestrini, M. (2018). Hybrid Differential Dynamic Programming Algorithm for Low-Thrust Trajectory Design Using Exact High-Order Transition Maps. Retrieved April 12, 2024, from https://www.semanticscholar.org/paper/Hybrid-Differential-Dynamic-Programming-Algorithm-Maestrini/6199f8e4be9350542bf46b6f1f4ea44da4121970

[83] Whiffen, G. (2006). Mystic: Implementation of the Static Dynamic Optimal Control Algorithm for High-Fidelity, Low-Thrust Trajectory Design. *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*. https://doi.org/10.2514/6.2006-6741

[84] Whiffen, G. J. (2013). THRUST DIRECTION OPTIMIZATION: SATISFYING DAWN'S ATTITUDE AGILITY CONSTRAINTS.

[85] Aziz, J. D., Scheeres, D. J., & Lantoine, G. (2019). Hybrid Differential Dynamic Programming in the Circular Restricted Three-Body Problem. *Journal of Guidance, Control, and Dynamics*, *42*(5), 963–975. https://doi.org/10.2514/1.G003617

[86] Lantoine, G., & Russell, R. P. (2008). A FAST SECOND-ORDER ALGORITHM FOR PRELIMINARY DESIGN OF LOW-THRUST TRAJECTORIES. *7*.

[87] Lantoine, G., & Russell, R. (2011). The Stark Model: An Exact, Closed-Form Approach to Low-Thrust Trajectory Optimization. Retrieved April 12, 2024, from https://www.semanticscholar.org/paper/The-Stark-Model%3A-An-Exact%2C-Closed-Form-Approach-to-Lantoine-Russell/c4e846ef74b58060d384452c0b7265a79fbc55a8

[88] Sims, J., & Flanagan, S. (2000). Preliminary Design of Low-Thrust Interplanetary Missions. *103*.

[89] Lantoine, G., & Russell, R. P. (2011). Complete closed-form solutions of the Stark problem. *Celestial Mechanics and Dynamical Astronomy*, *109*(4), 333–366. https://doi.org/10.1007/s10569-010-9331-1

[90] Pellegrini, E., & Russell, R. P. (2020b). A multiple-shooting differential dynamic programming algorithm. Part 2: Applications. *Acta Astronautica*, *173*, 460–472. https://doi.org/10.1016/j.actaastro.2019.12.038

[91] Lantoine, G., Cox, A., Sweetser, T., Grebow, D., Whiffen, G., Garza, D., Petropoulos, A., Oguri, K., Kangas, J., Kruizinga, G., & Castillo-Rogez, J. (2024). Trajectory & maneuver design of the NEA Scout solar sail mission. *Acta Astronautica*, *225*, 77–98. https://doi.org/10.1016/j.actaastro.2024.08.039

[92] Powell, M. J. D. (1978). Algorithms for nonlinear constraints that use lagrangian functions. *Mathematical Programming*, *14*(1), 224–248. https://doi.org/10.1007/BF01588967

[93] Marshall, M. A., & Pellegrino, S. (2023). Slew Maneuver Constraints for Agile Flexible Spacecraft. *Journal of Guidance, Control, and Dynamics*, *46*(12), 2300–2314. https://doi.org/10.2514/1.G007430

[94] Bani Younes, A. (2019). Exact Computation of High-Order State Transition Tensors for Perturbed Orbital Motion. *Journal of Guidance, Control, and Dynamics*, *42*(6), 1365–1371. https://doi.org/10.2514/1.G003897

[95] Pellegrini, E., & Russell, R. P. (2016). On the Computation and Accuracy of Trajectory State Transition Matrices [Publisher: American Institute of Aeronautics and Astronautics]. *Journal of Guidance, Control, and Dynamics*, *39*(11), 2485–2499. https://doi.org/10.2514/1.G001920

[96] Boone, S., & McMahon, J. (2023). Directional State Transition Tensors for Capturing Dominant Nonlinear Effects in Orbital Dynamics. *Journal of Guidance, Control, and Dynamics*, *46*(3), 431–442. https://doi.org/10.2514/1.G006910

[97] Li, X. (n.d.). Overview of Trust□region Methods.

[98] Sang, Z., & Sun, Q. (2009). A self-adaptive trust region method with line search based on a simple subproblem model. *Journal of Computational and Applied Mathematics*, *232*(2), 514–522. https://doi.org/10.1016/j.cam.2009.06.027

[99] Cui, Z., & Wu, B. (2012). A new modified nonmonotone adaptive trust region method for unconstrained optimization. *Computational Optimization and Applications*, *53*(3), 795–806. https://doi.org/10.1007/s10589-012-9460-4

[100] Saeidian, Z., Aminifard, Z., & Babaie–Kafaki, S. (2024). A nonmonotone adaptive trust region technique with a forgetting factor. *International Journal of Computer Mathematics*, *101*(5), 512–523. https://doi.org/10.1080/00207160.2024.2350447

[101] Lin, C.-J., & Moré, J. J. (1999). Newton's Method for Large Bound-Constrained Optimization Problems [Publisher: Society for Industrial and Applied Mathematics]. *SIAM Journal on Optimization*, *9*(4), 1100–1127. https://doi.org/10.1137/S1052623498345075

[102] Minnozzi, R. (2025, April). Https://github.com/ rikiminno/RiccardoMinnozzi_mscthesis. https://github.com/%20rikiminno/RiccardoMinnozzi_MScThesis

[103] Weinstein, M. J., & Rao, A. V. (2016). A Source Transformation via Operator Overloading Method for the Automatic Differentiation of Mathematical Functions in MATLAB. *ACM Transactions on Mathematical Software*, *42*(2), 1–44. https://doi.org/10.1145/2699456

[104] Pellegrini, E., Russell, R. P., & Vittaldev, V. (2014). F and G Taylor series solutions to the Stark and Kepler problems with Sundman transformations. *Celestial Mechanics and Dynamical Astronomy*, *118*(4), 355–378. https://doi.org/10.1007/s10569-014-9538-7

[105] Gamez Losada, F., Visser, P., & Heiligers, M. (2024). Fundamentals of Solar-Sail Transfers Around Planetary Bodies. *Proceedings of the 29th International Symposium on Space Flight Dynamics*.

A

Software Verification

A consistent part of this thesis work is composed by software development, which is to be verified. Section 4 describes how the developed optimization algorithm is verified (single-iteration convergence in a linear-quadratic problem) and validated (comparing results against state-of-the-art direct optimization solver). The results shown in Section 4 to verify and validate the algorithm are also integrated within the software framework through unit tests: throughout the development and testing of different solvers/approaches, the unit tests where used to continuously certify these implementation.

Separate verification steps are carried out on the dynamical models, and to justify numerical integration tolerances. MATLAB ® utilities are widely used to perform multiple steps, including numerical integration, parallel computation, and coordinate transformations: being part of the MATLAB ® suite, these functions do not require verification.

A.1. Dynamical models

Verification of the dynamical model is performed incrementally. First, the dynamics under point mass gravity are verified by propagating a single orbital revolution neglecting any SSA (i.e.: $a_0 = 0\ m/s^2$). Given an initial circular orbit, the spacecraft state is expected to be exactly unchanged. The property is verified by propagating the scaled dynamics, while re-scaled results (assuming initial $r_0 = 42164\ km$) are shown in Figure A.1. The same analysis is performed using both time-dependent and Sundman-transformed dynamics, yielding identical results. All the state vector components are observed to match the respective initial conditions (i.e.: all Δ values are null) except for the true anomaly θ, which exhibits the expected $360°$ difference since a full revolution was completed.

The ideal SSA model is verified by plotting its resulting acceleration bubble (i.e.: the contour of all attainable SSA vectors for any α, δ combination) for a sail with $a_0 = 1\ mm/s^2$. The bubble is plotted both in 2 and 3 dimensions in Figure A.2, observing that the resulting profiles match those shown in [6] and [105].

The chosen eclipse model is validated by propagating the sailcraft dynamics on a $AA = 90°$ orbit, using the locally optimal orbit raising law from [6] (the sail characteristic acceleration is kept at $a_0 = 1\ mm/s^2$. To better visualize the presence of eclipsing, results are shown for a trajectory at $r_0 = 20000\ km$. The locally optimal orbit raising law is used to verify that the eclipsing phenomenon has an impact on the sail performance: a comparison in drawn between the semi-major axis increase obtained from the eclipse and non-eclipse models. Results from the analysis are shown in Figure A.3. As expected, the presence of eclipsing manifests in overall 'delayed' orbit raising with respect to the 'no eclipse' case, as shown in Figure A.3a. The 3D representation in Figure A.3b confirms that the eclipse occurs in the correct arc of the orbit.

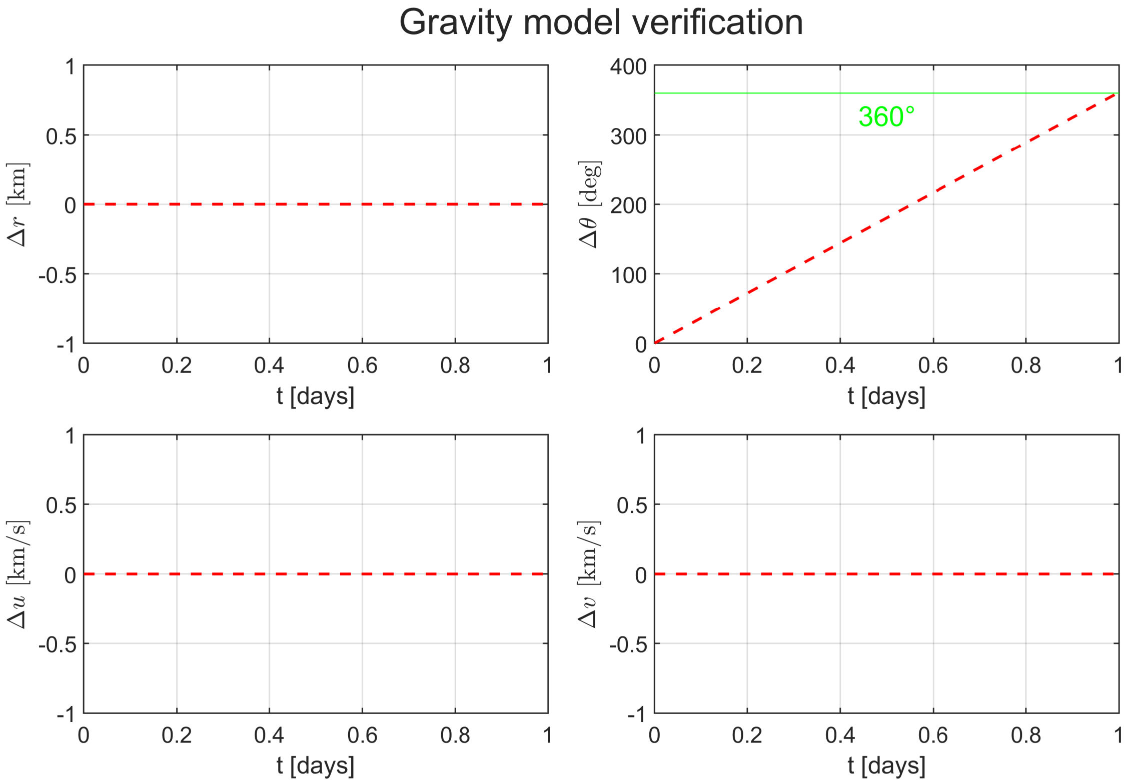


Figure A.1: Verification of the point-mass gravity model

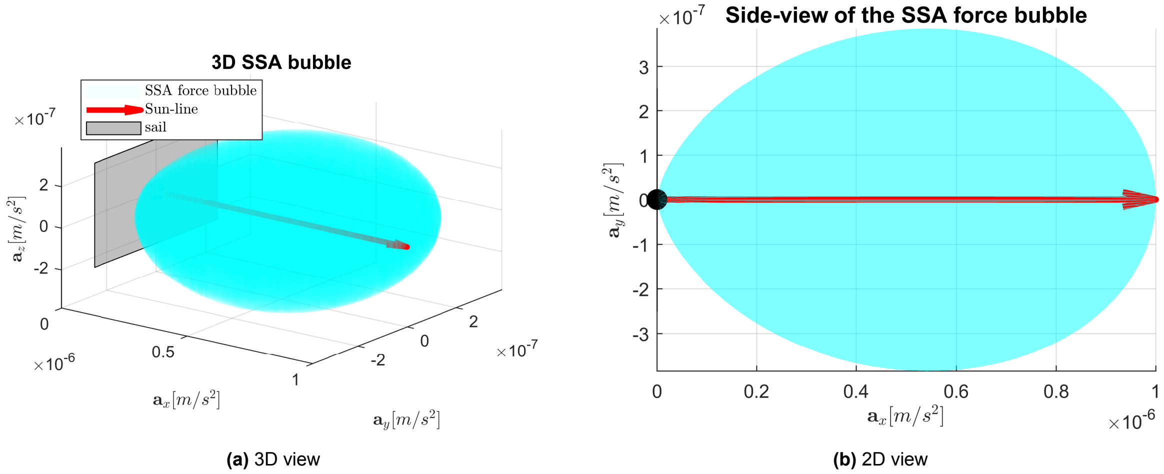
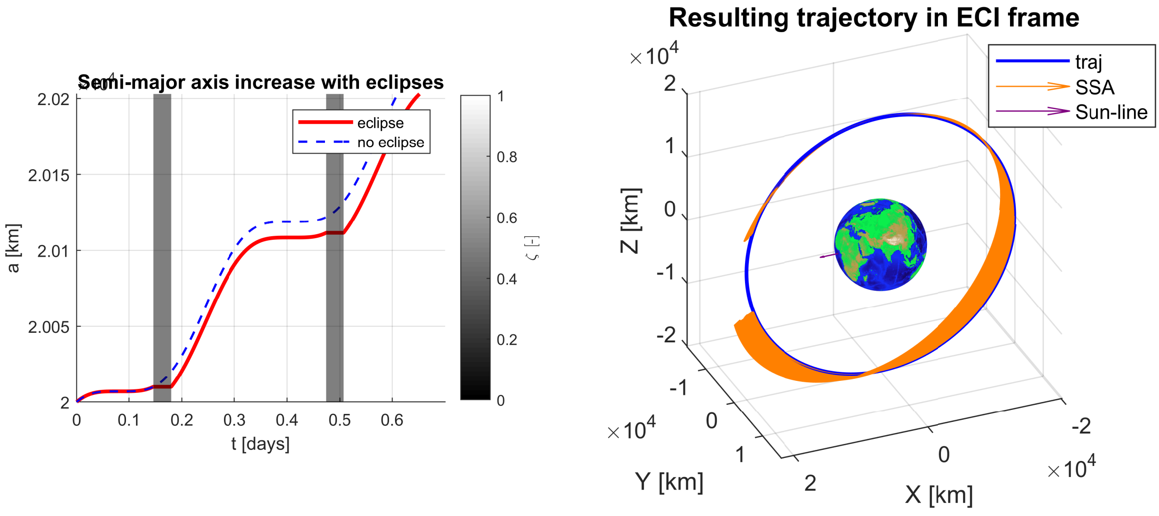


Figure A.2: Solar-sail acceleration bubble representation

A.2. Numerical integration

The numerical integration solver choice is now justified. The analysis is performed directly on the scaled dynamical model (over the default 2 revolutions propagation), to better justify the choices in accuracy requirements. A benchmark integrator is first defined. Given the default optimization tolerance values presented in Section 4, the benchmark results are deemed acceptable if the numerical integration error is 2 (or more) orders of magnitude smaller than the defined $\epsilon_{opt} = 10^{-6}$, ensuring that the benchmark choice does not significantly affect later results on integrator analysis. The integrator accuracy is assessed by computing the relative error $rel\ err$ between solutions with incrementally doubled step sizes Δt. The relative error is computed on the state vector norm. The chosen benchmark integrator is the `ode8` (8^{th} order Runge-Kutta integrator) implemented by MATLAB ®. Results are shown in Figure A.4.



(a) Semi-major axis comparison

(b) Full trajectory under eclipses

Figure A.3: Verification of the eclipse model

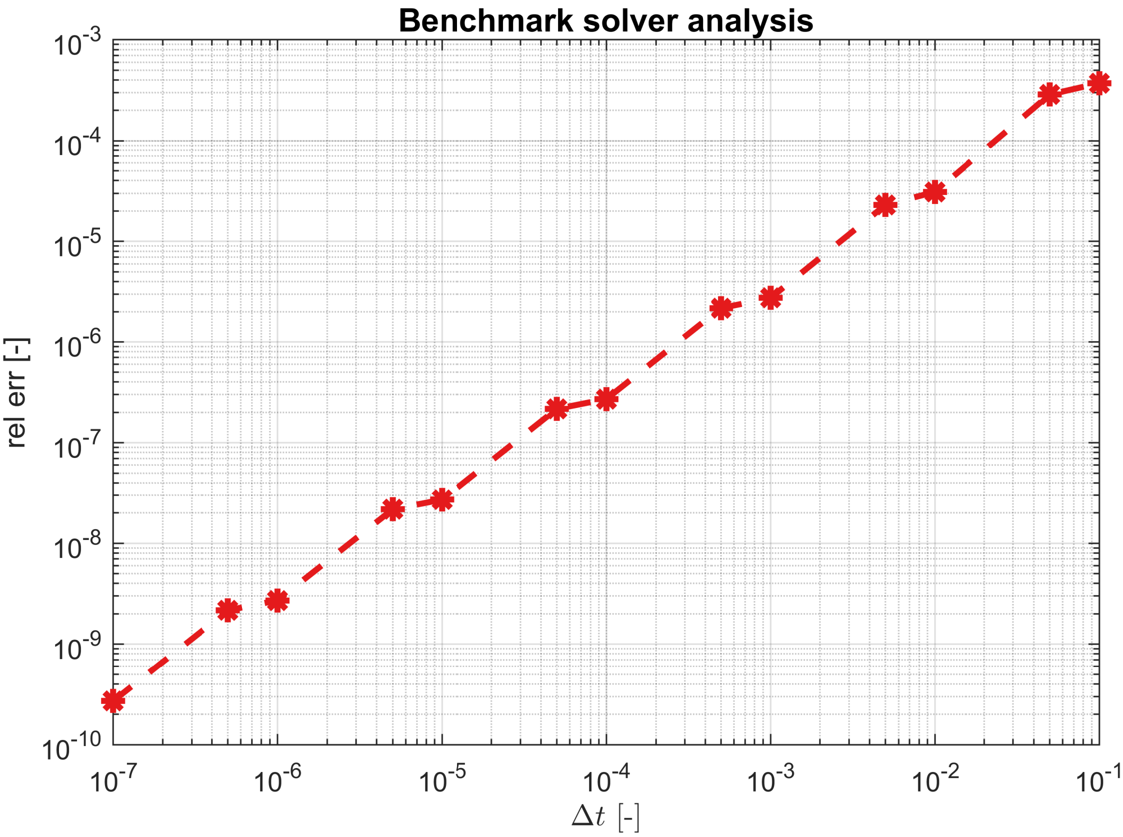
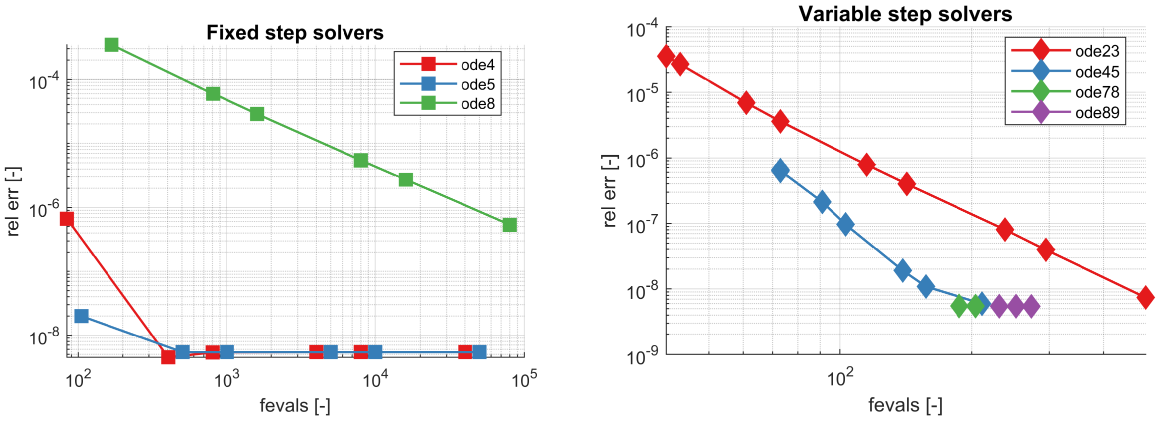


Figure A.4: Benchmark integrator analysis

The $\Delta t = 10^{-6}$ satisfies the relative error requirement of $rel\ err = 10^{-8}$ and is therefore chosen. The defined benchmark is then used to compare integrators of different orders and step-sizes/tolerances. The comparison aims at identifying the combination which achieves a relative error that is 1 (or more) order of magnitude below the defined ϵ_{opt} tolerance, while minimizing the number of function evaluations required to propagate the trajectory. The choice aims at minimizing the numerical propagation overhead to speed up the algorithm (as both the HDDP forward pass and STMs propagation require numerical integration), while avoiding interference between numerical integration errors and optimization tolerances. The full set of results is provided in Figure A.5, while the chosen step-sizes and tolerances (shown are both relative and absolute values) are summarized in Table A.1.

Table A.1: Swept step sizes and tolerances

Step Sizes	10^{-4}	$5 \cdot 10^{-4}$	10^{-3}	$5 \cdot 10^{-3}$	10^{-2}	$5 \cdot 10^{-2}$			
Tolerances	10^{-8}	$5 \cdot 10^{-8}$	10^{-7}	$5 \cdot 10^{-7}$	10^{-6}	$5 \cdot 10^{-6}$	10^{-5}	$5 \cdot 10^{-5}$	10^{-4}



(a) Fixed-step solvers

(b) Variable-step solvers

Figure A.5: Full integrator analysis

The fixed-step solvers in Figure A.5a generally have a larger number of function evaluations with respect to their variable counterparts. The variable-step solvers are therefore preferred: Figure A.5b shows the `ode45` solver, with tolerances of 10^{-5}, to meet the specified accuracy requirements.

When moving to the many-revolutions transfer cases, integration errors are expected to increase. Since a full integrator analysis on the long transfer cases is deemed too time-consuming, a different approach is chosen. The `ode45` solver relative and absolute tolerances are reduced to 10^{-8}, as it was found to provide acceptable computational overhead to perform the optimization. All generated solutions are then benchmarked against a more accurate solver (`ode45` with tolerances of 10^{-12}), by fully propagating the sailcraft dynamics with the optimal control law. The full set of results presented in Section 4 was observed to match the new benchmark accuracy to tolerances below the defined ϵ_{opt}, and are therefore deemed valid. As an example, a comparison of the 180 revolutions Circular To Circular (C2C) case at GEO altitude is shown in Figure A.6, in terms of semi-major axis increase radius increase and eccentricity.

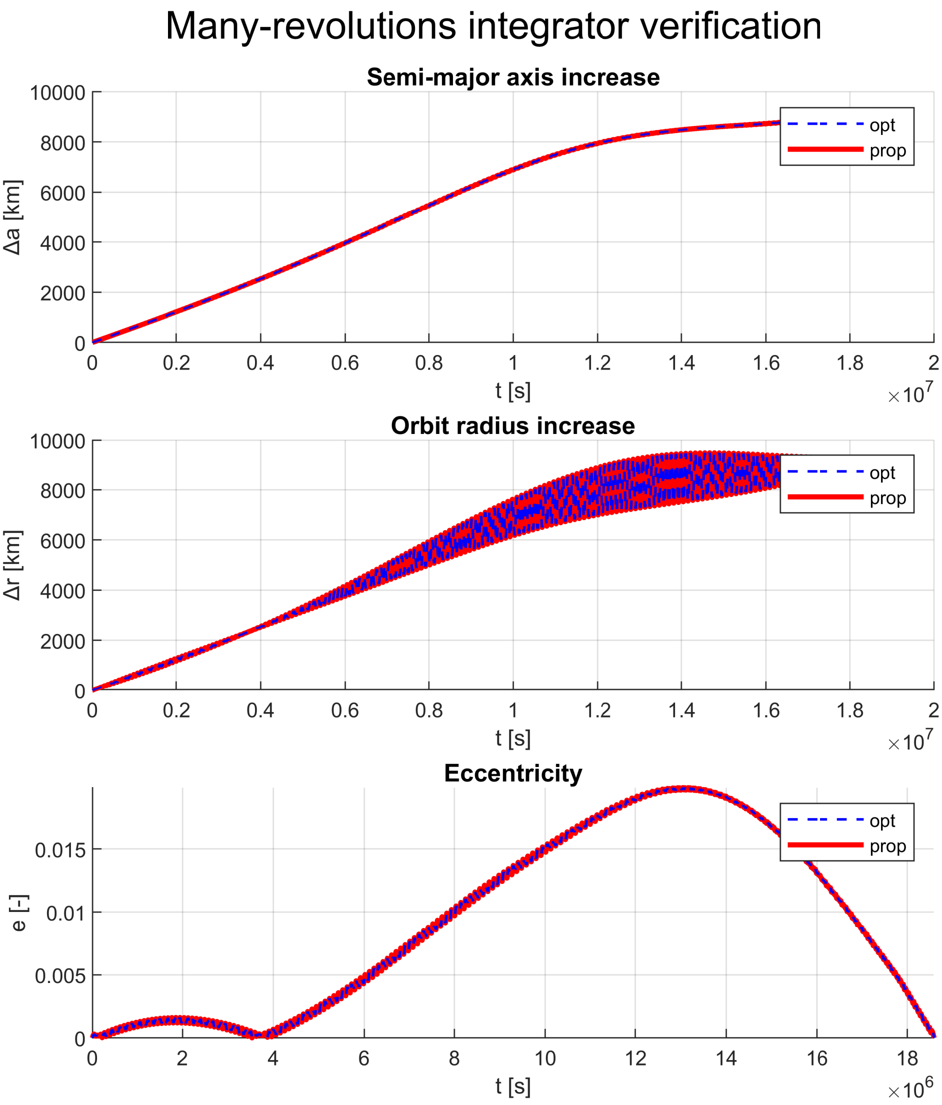


Figure A.6: 180 revolutions integrator verification

B
Project Management

The project management practices followed throughout the thesis are provided in this Section. First, the Work-Breakdown Structure (WBS) outlined after the literature review process is provided. The Gantt chart followed throughout the thesis work is also included, highlighting that red boxes indicate non-working periods. The project plan devised after the literature review was followed successfully throughout the whole thesis duration, with only minor setbacks. Additional (out of the thesis scope) activities were also pursued: as these were not known at the time of project-planning, the thesis duration had to be adjusted accordingly.

B.1. Work breakdown structure

WP1 (14 days) - Time optimal HDDP

Objective - reformulate the HDDP algorithm to include the time of flight as an optimization variable

1. (3 days) Write down mathematical formulation of HDDP explicitly
2. (2 days) Write down mathematical formulation for flexible final time HDDP explicitly
3. (5 days) Integrate and test formulation within the full HDDP framework

Margin (2 days) - While the first part of this task purely consists in establishing notation and conventions, a time optimal formulation for HDDP is not yet known and therefore margin is introduced to allow deeper insight into derivations and trade-offs between approaches (namely [67] and [85, 86]).

WP2 (10 days) - High-level software architecture

Objective - Design and implement a high-level software architecture that enables modular changes to the various components of HDDP

1. (4 days) Design high-level algorithm structure
   a. (1 days) Define required algorithm initialization settings
   b. (3 days) Define comprehensive HDDP block diagram structure
2. (1 days) Define interfaces for the high-level algorithm blocks
3. (1 days) Define and implement external software interface (MATLAB or Python)
4. (3 days) Implement high-level HDDP architecture
5. (1 days) Report process and results into thesis document (relates to WP8)

Margin (2 days) - The flexibility and modularity requirements imply the need to foresee and accommodate non-trivial features, such as the multi-phase formulation and/or the introduction of Sundman transforms, which can complicate the architecture definition and implementation.

WP3 (10 days) - **Propagator**

Objective - Implement the routine to define and propagate the dynamics and STMs

2. (0 days) Define a flexible way to change integration scheme (comes from WP2)
2. (3 days) Define and implement flexible method for dynamical model implementation
3. (3 days) Derive the variational equations
    a. (2 days) Investigate if automatic differentiation is feasible and effective in this context
    b. (1 days) Otherwise, use an automated Maple workflow
    c. (1 days) Implement the chosen approach for the generation of the variational equations
4. (2 days) Implement parallel propagation for the STMs
5. (1 days) Verify propagator implementation
6. (1 days) Report findings in thesis document (relates to WP8)

Margin (2 days) - The propagator implementation is relatively straightforward, thanks to the availability of multiple off-the-shelf tools for numerical integration, however, some uncertainty on the implementation of a modular approach to define the dynamical model as well as automatic differentiation and parallelized propagation is present.

WP4 (10 days) - **TRQP solver**

Objective - Investigate and implement an algorithm to solve the TRQP in the backwards sweep step of HDDP

1. (1 days) Implement interface for the TRQP solver block
2. (5 days) Research into the theory of Trust Reqion methods (available in [75])
    a. (2 days) Investigate the different choices of Trust Region algorithms
    b. (3 days) Investigate how the trust region radius update parameter can be generalized (answering RQ1.3)
3. (2 days) Implement the chosen TRQP solver with the pre-defined interface
4. (1 days) Verify the implementation
5. (1 days) Report findings and results into thesis document (relates to WP8)

Margin (2 days) - The generalization of the trust region radius update is an uncertain step, and might require longer research or testing than what is foreseen.

WP5 (13 days) - **HDDP algorithm integration**

Objective - Integrate the whole HDDP algorithm within the high-level software architecture defined in WP2

1. (1 days) Check that no conflicts between the interfaces defined in WP2 and those required by the lower level blocks defined in WP3 and WP4 are compliant
2. (0.5 days) Integrate the propagator into its interface (requires WP2 and WP3)
3. (0.5 days) Integrate the TRQP solver into its interface (requires WP2 and WP4)
4. (1 days) Implement convergence tests
5. (4 days) Implement the iteration acceptance tests
    a. (3 days) Investigate if the iteration acceptance threshold update can be generalized (answering RQ1.3)
    b. (1 days) Implement the iteration acceptance threshold update
6. (4 days) Implement the constraint violations check
    a. (3 days) Investigate if the constraint penalty update can be generalized (answering RQ1.3)
    b. (1 days) Implement the constraint penalty parameter update

7. (1 days) Verify the implementation (mostly achieved in WP6)
8. (1 days) Report findings into thesis document (relates to WP8)

Margin (3 days) - The uncertainty on how to generalize parameter updates on constraints and iteration acceptance means that time allocations defined a-priori are rather unreliable. Additionally, software integration is generally known to present unforeseen challenges, therefore time allocations defined for this work package are already generous.

WP6 (6 days) - **Basic validation case**

Objective - The HDDP applied to a linear-quadratic problem is known to converge in a single iteration (as illustrated, for instance, in [80]), therefore the first validation of the algorithm is to be performed on such a benchmark

1. (0 days) Define the dynamical model (already available in [80])
2. (1 days) Implement the dynamical model into the tool (requires WP5)
3. (0 days) Define objective function and constraints (already available in [80])
4. (1 days) Implement objective function and constraints into the tool (requires WP5)
5. (3 days) Perform optimization
6. (1 days) Report findings into thesis document (relates to WP8)

Margin (3 days) - Being the first integral application of the tool, it is expected for most inconsistencies and errors to show up at this step, therefore a considerable margin is allocated for debugging and code corrections.

WP7 (27 days) - **Many-revolutions transfers**

Objective - The HDDP framework is implemented with the main objective of identifying and characterizing optimal solar-sail transfers (see RQ2)

1. (14 days) First validation case: many-revolutions transfer using simple dynamical model (answering RQ2)
   - a. (0 days) Define the simplified dynamical model
   - b. (2 days) Implement the simplified dynamical model (requires WP5, mostly achieved in WP3)
   - c. (1 days) Define objective function and constraints
   - d. (1 days) Implement objective function and constraints into the tool (requires WP5)
   - e. (1 days) Define trivial and non-trivial initial guesses
   - e. (5 days) Perform optimization (answering RQ2.1)
   - f. (3 days) Analyze results and compare with alternative approach (answering RQ2.1)
   - g. (1 days) Report findings into thesis document (relates to WP8)
2. (13 days) Second validation case: many-revolutions transfer using more complex dynamical model (answering RQ2)
   - a. (0 days) Define the complex dynamical model
   - b. (3 days) Implement the complex dynamical model (requires WP5, partly achieved in WP7-1)
   - c. (0 days) Define objective function and constraints (done in WP7-1)
   - d. (0 days) Implement objective function and constraints into the tool (done in WP7-1)
   - e. (1 days) Define trivial and non-trivial initial guesses (can technically benefit from the solution available in WP7-1)
   - e. (5 days) Perform optimization (answering RQ2.1)

f. (3 days) Analyze results and compare with solution from simplified dynamical model in WP7-1 (answering RQ2.1)

g. (1 days) Report findings into thesis document (relates to WP8)

Margin (5 days) - The amount of effort required in order to achieve convergence in the optimization of both dynamical models is uncertain, and heavily depends on the quality of the framework resulting from the previous work packages. Additionally, part of the tasks (namely the comparison against an alternative optimization method and definition of the first non-trivial initial guess) depend on external factors and therefore some additional time margin is accounted for.

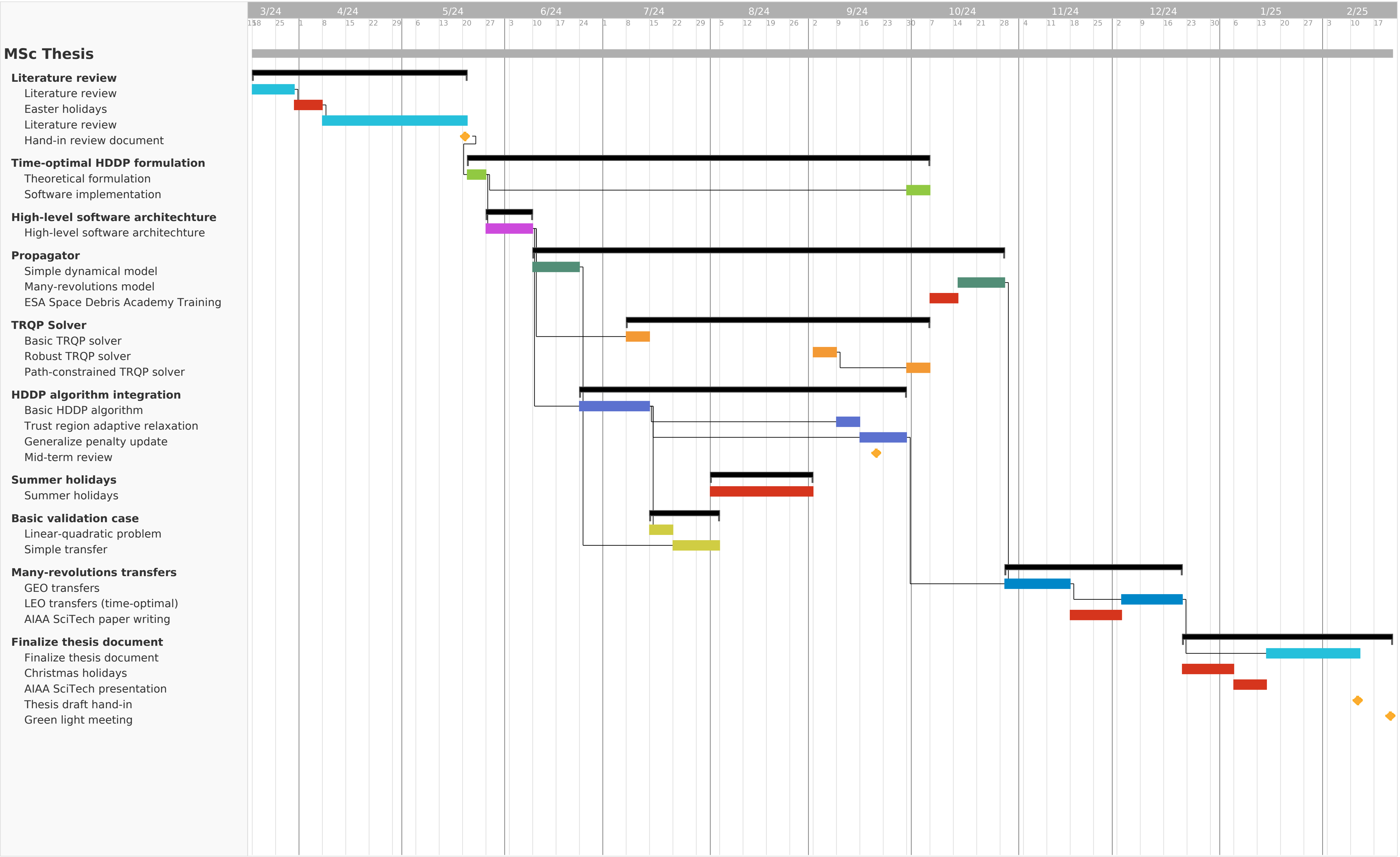
WP8 (17 days) - Thesis document

Objective - Finalize the thesis document by reporting the methodology, results and conclusions from all the steps.

1. (2 days) Report findings from WP1
2. (1 days) Report findings from WP2
3. (1 days) Report findings from WP3
4. (1 days) Report findings from WP4
5. (2 days) Report findings from WP5
6. (1 days) Report findings from WP6
7. (2 days) Report findings from WP7
8. (1 days) Final adjustments for thesis document
9. (6 days) Receive and implement feedback

Margin (3 days) - The time margin is allocated here mainly to account for potential delays with the writing process, as well as delays in the feedback.

B.2. Time allocation


3/24
4/24
5/24
6/24
7/24
8/24
9/24
10/24
11/24
12/24
1/25
2/25
MSc Thesis
Literature review
Literature review
Easter holidays
Literature review
Hand-in review document
Time-optimal HDDP formulation
Theoretical formulation
Software implementation
High-level software architechture
High-level software architechture
Propagator
Simple dynamical model
Many-revolutions model
ESA Space Debris Academy Training
TRQP Solver
Basic TRQP solver
Robust TRQP solver
Path-constrained TRQP solver
HDDP algorithm integration
Basic HDDP algorithm
Trust region adaptive relaxation
Generalize penalty update
Mid-term review
Summer holidays
Summer holidays
Basic validation case
Linear-quadratic problem
Simple transfer
Many-revolutions transfers
GEO transfers
LEO transfers (time-optimal)
AIAA SciTech paper writing
Finalize thesis document
Finalize thesis document
Christmas holidays
AIAA SciTech presentation
Thesis draft hand-in
Green light meeting