Advances in Quadratic Programming: Regularization, Learning-Based Enhancements, and Applications to Trajectory Planning and Nonlinear Programming

Jeremy BERTONCINI

Vollständiger Abdruck der von der Fakultät für Luft- und Raumfahrttechnik der Universität der Bundeswehr München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

angenommenen Dissertation.

Gutachter: 1. Prof. Dr. rer. nat. Matthias Gerdts
2. Prof. Dr. rer. nat. Enrico Bertolazzi

Die Dissertation wurde am 08.01.2026 bei der Universität der Bundeswehr München eingereicht und durch die Fakultät für Luft- und Raumfahrttechnik am 15.05.2026 angenommen. Die mündliche Prüfung fand am 05.06.2026 statt.

Declaration of Authorship

I, Jeremy Bertoncini, declare that this thesis titled, "Advances in Quadratic Programming: Regularization, Learning-Based Enhancements, and Applications to Trajectory Planning and Nonlinear Programming" and the work presented in it are my own. I confirm that:

- This work was completed in the course of my doctoral research at the University of the Bundeswehr Munich.
- All published work of others that has informed this research has been clearly cited.
- The parts of this dissertation that have been published by me are clearly indicated as such.

Signed: _____

Date: _____

Abstract

This dissertation develops numerical optimization methods for quadratic programming problems whose structural properties challenge solver reliability and computational efficiency. These methods are applied within optimal control problem formulations, with particular emphasis on autonomous unmanned aerial vehicles (UAVs). To support a practical deployment, a modular optimal control framework is developed and integrated with communication protocols and simulation environments.

A stabilized primal-dual interior-point solver for quadratic programming (QP) is presented, integrating proximal and augmented Lagrangian regularization to enhance both robustness and convergence. A novel proximal regularization for slack variables is proposed, along with safeguards to maintain trajectories near the central-path. Extensive numerical experiments on challenging benchmark problems, including ill-conditioned and poorly scaled cases, demonstrate the solver's accuracy, robustness, and competitiveness with leading commercial and open-source QP solvers. Moreover, a reinforcement learning approach is introduced to automatically tune proximal and augmented Lagrangian regularization weight dynamics. The policy, trained on a small number of random QP instances, adapts the solver parameters online, enhancing convergence speed while maintaining theoretical guarantees. The QP solver is further embedded into an in-house Sequential Quadratic Programming (SQP) framework, enabling the efficient solution of constrained nonlinear optimization problems and providing comprehensive background for the following part of the thesis.

A Model Predictive Control (MPC) framework is developed and applied to three types of vehicles: fixed-wing aircraft, quadrocopters, and ground vehicles, each presenting different dynamics and control challenges. The methods are validated through Software-In-The-Loop (SITL) simulations and real-world experiments, including flight tests with a very-light aircraft. A real-time trajectory visualization and guidance interface was implemented to assist pilots, demonstrating the practical relevance of the approach in both civil and military air traffic scenarios. Finally, interactions between cooperative autonomous systems are examined through the high-speed landing of a quadrocopter on an unmanned ground vehicle, demonstrating the ability of MPC-based methods to handle multi-agent coordination, high-speed maneuvers, and strict feasibility requirements. Overall, the dissertation shows that advanced optimization techniques, combined with data-driven parameter tuning, can provide reliable and efficient control of autonomous systems in complex, real-world environments.

Zusammenfassung

Diese Dissertation beschäftigt sich mit der Entwicklung numerischer Methoden für quadratische Optimierungsprobleme (QP), deren strukturelle Eigenschaften die bestehenden Solver vor eine Herausforderung in Bezug auf ihre Zuverlässigkeit und Recheneffizienz stellen. Die hier entwickelten Methoden werden auf diverse Optimalsteuerungsprobleme angewendet, wobei besonderes Augenmerk auf autonome unbemannte Luftfahrzeuge (UAVs) gelegt wird. Zur Gewährleistung der praktischen Anwendbarkeit wird ein modulares Framework für die Optimalsteuerung entwickelt und in bestehende Kommunikationsprotokolle sowie Simulationsumgebungen integriert.

Im Rahmen von dieser Arbeit wird ein stabilisierter primal-dualer Innere-Punkte-Solver für quadratische Optimierungsprobleme vorgestellt. Dabei werden proximale und Augmented-Lagrange-Methoden zur Regularisierung herangezogen, um sowohl die Robustheit als auch die Konvergenz des Solvers zu verbessern. Zudem wird eine neuartige proximale Regularisierung für Schlupfvariablen mit Absicherungsmechanismen präsentiert, um ein Verbleiben der Trajektorien in der Nähe des zentralen Pfades zu gewährleisten. Umfangreiche numerische Experimente mit anspruchsvollen Benchmark-Problemen (darunter schlecht konditionierte und schlecht skalierte) belegen die Genauigkeit, Robustheit und vergleichbare Leistungsfähigkeit des Solvers im Vergleich zu führenden kommerziellen und Open-Source-QP-Solvern. Um die Gewichtsdynamik der auf proximalen und Augmented Lagrange-Methoden basierten Regularisierung automatisch anzupassen, wird ein Reinforcement-Learning-Ansatz verwendet. Die auf einer kleinen Anzahl zufälliger QP-Probleme trainierte Policy passt die Solver-Parameter online an, was eine Erhöhung der Konvergenzgeschwindigkeit bei gleichzeitiger Beibehaltung der theoretischen Garantien zur Folge hat. Der QP-Solver wird zusätzlich in ein internes Framework für die Sequentielle Quadratische Programmierung (SQP) eingebettet. Dieses interne Programm dient zur effizienten Lösung von nichtlinearen Optimierungsproblemen mit Nebenbedingungen und bietet eine Grundlage für den folgenden Teil dieser Arbeit.

Darüber hinaus wurde ein Modellprädiktives Regelungs-Framework (MPC) entwickelt und auf drei Fahrzeugtypen angewendet: Starrflügler, Quadrocopter und Bodenfahrzeuge. Diese Fahrzeuge weisen jeweils unterschiedliche Dynamiken auf und bringen somit auch verschiedene Herausforderungen hinsichtlich der Steuerung mit sich. Die Methoden wurden durch Software-In-The-Loop-Simulationen (SIL) und reale Experimente validiert, darunter Flugtests mit einem Ultraleichtflugzeug. Zur Unterstützung der Piloten wurde eine Schnittstelle zur Echtzeit-Visualisierung und -Führung der Flugbahn implementiert, die die praktische Relevanz des Ansatzes sowohl in zivilen als auch militärischen Luftverkehrsszenarien demonstriert. Schließlich werden die Interaktionen zwischen kooperativen autonomen Systemen durch die Landung eines Quadrocopters bei hoher Geschwindigkeit auf einem unbemannten Bodenfahrzeug untersucht. Dadurch wird die Fähigkeit von MPC-basierten Methoden zur Multi-Agenten-Koordination bei hochdynamischen Manövern unter strengen Machbarkeitsanforderungen nachgewiesen. Insgesamt zeigt die Dissertation, dass fortschrittliche Optimierungstechniken in Kombination mit datengetriebener Parameteroptimierung eine zuverlässige und effiziente Steuerung diverser autonomer Systeme in realen komplexen Umgebungen ermöglichen.

Acknowledgement

I am deeply grateful to my supervisor, Prof. Dr. Matthias Gerdts, for his continuous mentorship, the freedom to pursue my research ideas, and his valuable time and feedback throughout this work. I am also sincerely grateful to my colleagues and friends whose contributions greatly supported and enriched this research. In particular, I would like to thank Dr. Alberto De Marchi for the mathematical rigorousness he brought to the development of `QProxIP`, and Dr. Simon Gottschalk for our common work on the reinforcement-learning extensions of the method. Our working sessions balanced energy and enthusiasm with a necessary dose of realism, preventing us from immediately designing the next ten modules before finishing the first one. I thank Prof. Dr. Enrico Bertolazzi for inviting me to the University of Trento, where his guidance led to great results in testing `QProxIP`.

I thank Thomas Rottmann Dipl.-Ing. for his support with flight-test preparation and GNSS interfaces, and Dr. Adrian Dudek and Johannes Ostler M.Sc. for their help with obstacle detection and flight monitoring, even when flight preparation stretched into late nights after a pizza. I also thank Dr. Martin Russ for his project management within the MissionLab framework, which made our flight tests possible. Their expertise, commitment, and collaboration were essential to the success of this work. I thank my teammate and fellow work companion, Viktoriya Nikitina M.Sc. for her energy and dedication in our joint work that led to a paper at the CDC conference, as well as for her enthusiasm in the MUVOpt project we advanced together throughout 2025. I gratefully acknowledge the support of DTEC and the European Union, whose funding and resources made this research possible.

Finally, I would like to thank my family and friends for their constant support, patience, and encouragement throughout the years. I thank my parents for always motivating me to stay curious. I thank my brother Alex for teaching me over the years to find simple and pragmatic solutions while keeping a sense of fun. I thank my friend Seb for inspiring me to stay relaxed when facing challenges and for bringing a touch of eccentricity to my perspectives. I thank the 5 Mousquetaires of Thann for the long evenings spent playing cards and debating the world, and sciences. This work would not have been possible without your support.

A mes parents.

$\mathcal{K}$

Contents

Declaration of Authorship iii

Abstract iv

Zusammenfassung v

Acknowledgement vi

Publications and Lectures xvii

Introduction 1

I A Modular QP Solver: Regularization, Convergence, Learning-based Enhancement 5

1 A Modular Regularized QP Solver 7
1.1 Introduction . . . 7
1.2 Convex optimization . . . 8
1.2.1 Problem statement . . . 8
1.2.2 Hyperplane separation . . . 9
1.2.3 Optimality . . . 11
1.2.4 Duality . . . 14
1.3 Convex Quadratic Programming . . . 18
1.3.1 Mathematical formulation . . . 19
1.3.2 Constraints analysis . . . 20
1.3.3 Convexity of QPs . . . 22
1.3.4 Duality of QPs . . . 23
1.3.5 Existence of the solution for convex QPs . . . 25
1.3.6 Optimality of QPs . . . 26
1.4 Algorithmic . . . 27
1.4.1 Merit function . . . 27
1.4.2 First order methods . . . 28
1.4.3 Newton Schemes . . . 28
1.4.4 Linesearch . . . 29
1.4.5 Termination . . . 32
1.4.6 Preconditioning . . . 33
1.5 Interior Point Methods . . . 34
1.5.1 History and background . . . 34
1.5.2 Enforcing inequality constraints . . . 34
1.5.3 Classical primal Interior Point . . . 36
1.5.4 Classical primal-dual Interior Point . . . 39
1.5.5 Primal and primal-dual KKT schemes linear equivalency . . . 44
1.5.6 Stabilized primal-dual Interior Point . . . 44

1.5.7 Proximal slack-stabilized primal-dual Interior-Point Method 50
1.6 Regularization interpretation 51
1.6.1 The Euclidean proximal operator 52
1.6.2 The primal-dual Gill-Robinson Augmented Lagrangian 54
1.7 Safe-guards, corrections and decompositions 55
1.7.1 Central path correction 56
1.7.2 Fraction to the boundary 57
1.7.3 Centering the barrier 58
1.7.4 Boundary control 59
1.7.5 Early termination amortization 59
1.7.6 Symbolic and numerical factorization 60
1.8 Convergence analysis 61
1.8.1 Outerloop - Proximal Point Algorithm 61
1.8.2 Innerloop - a regularized Newton method 63
1.9 Modularity and object oriented programming 64
1.10 Results 66
1.10.1 Monitoring and metrics 66
1.10.2 Maros-Mészáros test-set 67
1.10.3 Performance Variability in Computational Benchmarks 72
1.11 Applications and Study Case 74
1.11.1 Vehicle Routing Problem, a continuous-based optimal transport approach 74
1.11.2 Stability inside a Tokamak 76
1.11.3 Portfolio Allocation optimization 79
1.11.4 Quadrocopter Path-Following Optimal Control 85
1.12 Chapter conclusion 87

2 Learning to Enhance a QP-solver: A Reinforcement Learning Approach 89
2.1 Introduction 89
2.2 Hyperparameters in QP problems 91
2.3 Learning-enhanced methods 92
2.3.1 States and reward 93
2.3.2 Training procedures 95
2.3.3 Training problem set 96
2.4 Numerical investigations 99
2.4.1 Training results 100
2.4.2 Validation 101
2.4.3 Multistage learning and in-training validation 104
2.4.4 Problem dimensions scaling 104
2.4.5 Effective condition Number of the KKT matrix 105
2.4.6 Maros-Mészáros test-set 106
2.5 Chapter conclusion and outlook 107

3 Nonlinear Programming: Sequential Quadratic Programming (SQP) 109
3.1 Introduction 109
3.2 Lagrange Newton method 110
3.3 Sequential Quadratic Programming 111
3.4 Implementation 114
3.5 Numerical results 114
3.6 Chapter conclusion 116

Part I Conclusion 118

II Model Predictive Control and Applications to Path-Planning and Mission Management **119**

1 Optimal Control **121**
1.1 Introduction . . . 121
1.2 Model Predictive Controller . . . 123
1.3 Architecture and engineering . . . 125
1.4 Chapter conclusion . . . 126

2 Aerial and Ground Vehicle Path-Planning and Path-Tracking **127**
2.1 Introduction . . . 127
2.2 Coordinate systems . . . 128
2.3 Fixed-wing path-planning . . . 129
2.3.1 Optimal control problem definition . . . 129
2.3.2 Evaluation on randomized scenarios . . . 132
2.4 Quadrocopter path-planning . . . 133
2.4.1 Optimal control problem definition . . . 134
2.4.2 Evaluation on randomized scenarios . . . 135
2.5 Ground vehicles path-tracking . . . 137
2.5.1 Path-tracking optimal control . . . 137
2.5.2 Inter-agent collision avoidance . . . 138
2.5.3 Multi-UGV path-tracking and collision avoidance . . . 140
2.6 Chapter conclusion . . . 144

3 Man-Unmanned Navigation Assistance System **145**
3.1 Introduction . . . 145
3.2 Optimization problem statement . . . 147
3.2.1 Optimal control problem definition . . . 147
3.3 System design . . . 149
3.3.1 On-board INS . . . 150
3.3.2 Ground station . . . 150
3.3.3 Inflight mini-computer . . . 151
3.4 Flight experiments . . . 151
3.4.1 Scenario parameters . . . 151
3.4.2 Realization 1: multiple zone avoidance . . . 153
3.4.3 Realization 2: dense emergency maneuver . . . 155
3.5 Chapter conclusion . . . 156

4 UAV-UGV Interactions: The Case of Quadrocopter-UGV Dynamical Docking **159**
4.1 Introduction . . . 159
4.2 Optimal control . . . 161
4.2.1 UAV path-planning . . . 161
4.2.2 UGV path-tracking . . . 162
4.2.3 Hierarchical controllers . . . 162
4.3 Architecture . . . 163
4.3.1 Communication structure between controllers . . . 163
4.3.2 SQP solver . . . 163
4.4 Results . . . 164
4.4.1 Scenario parameters . . . 164
4.4.2 Realization . . . 165
4.5 Chapter conclusion . . . 168

Part II Conclusion **169**

Conclusion **171**

Appendix **172**

References **175**

List of Figures

1.1 Barrier penalty depending on $\nu = 100$ (left) $\nu = 10$ (right). 36
1.2 Merit function, derivatives and directions. 45
1.3 $\phi(rp, rd)$ mapping. 60
1.4 Indicator and Normal cone function for $s \geq 0$. 61
1.5 Modular architecture and flowchart. 65
1.6 Performance profile of `QProxIP` depending on $\alpha_{\square}$ - MM100 sub-testset. . . . 68
1.7 Performance profile of `QProxIP` depending on $\delta_{\square,0}$ - MM100 sub-testset . . . 69
1.8 Data profile of `QProxIP` with and without the s-prox regularization for MM50 without Ruiz equilibration (preconditioning). 70
1.9 Maros–Mészáros data profile, benchmarking of reference QP-solvers on the full test-set MM139. 72
1.10 Performance profile of `QProxIP` depending on the power supply (left) and CPU overload (right). 73
1.11 Maros–Mészáros data profile on the test-set MM50, solved 100 times in a row. 73
1.12 Routing with 3 bases, 2 targets and 3 obstacles 76
1.13 Stable tokamak configuration - Energy, States and Phase Portrait 78
1.14 Unstable tokamak configuration - Energy, States and Phase Portrait 79
1.15 Portfolio evolution (01/01/2020-01/10/2025), controlled vs. equally weighted. 81
1.16 Ratio controlled portfolio vs. equally weighted portfolio. 82
1.17 Actions and assets repartition. 83
1.18 Asset price evolution from 2010 to 2025. The shaded area marks the prediction period. 84
1.19 Covariance matrices per year with the rolling data window: $T = 10$ past years. 84
1.20 Visualization of the quadrocopter trajectory and position tracking. The subplots show (left to right): 3D trajectory with waypoints, and position versus waypoints over time, highlighting how the quadrocopter follows the planned path. 87
1.21 Visualization of velocity, control inputs, and constraint magnitudes over time. The subplots show (left to right): velocities, control inputs, and constraints magnitudes. 87

2.1 A QP solver is combined with a reinforcement learning (RL) agent that evaluates a policy to adapt the internal parameters of the QP solver. Based on its observation of the state of the environment, the RL action aims at improving the solver's performance. The RL policy is parameterized by a small artificial neural network, the state σ summarizes the internal state of the QP solver (at the beginning of each inner loop), the action α affects the parameters δ of the solver, and the reward $\mathcal{R}$ promotes faster convergence to a solution of the QP. Illustration inspired by [70, Fig. 1]. 90
2.2 Distribution of problem size (in terms of number of nonzero entries) and density for randomized training, validation, and scale 10× problems (sample of 10^3 instances) and for Maros-Mészáros problems (100 smallest instances). 99

2.3 Distributions of problem initial residuals (sample of 10^3 instances). Primal-dual values for different problem classes (left panel). Target vs actual values generated for the training procedure (right panel). 100
2.4 Distribution of the effective condition number κ_e (sample of 10^3 instances). Values for a log-uniform distribution spanning 10^{15} to 10^{20} (orange) when applied to random problem generation (blue). 101
2.5 Evolution of relevant metrics during the multistage training procedure.. . . . 102
2.6 Validation set: Data (left) and performance (right) profiles comparing RL and solver configurations with fixed $\alpha_x = 0.15$, α_λ and α_μ. 103
2.7 Mappings α_x (left), α_λ (middle) and α_μ (right): learned policy π_ζ evaluated with fixed $\nu = 10^{-2}$ and $\varepsilon = 10^{-2}$. 104
2.8 In-training validation: policy performance along the multistage process. Data (left) and performance (right) profiles comparing the influence of different RL policy. 104
2.9 Scale 10× set: Data (left) and performance (right) profiles comparing RL and solver configurations with fixed $\alpha_x = 0.15$, α_λ and α_μ. 105
2.10 Evolution of the effective condition number of the KKT matrix across iterations for RL-QProxIP (blue) and standard QProxIP (red) strategies. 105
2.11 Maros–Mészáros set, MM100: Data (left) and performance (right) profiles comparing RL and solver configurations with fixed $\alpha_x = \alpha_\lambda = \alpha_\mu$. Despite the lightweight training on random problems, the learned RL policy performs well on the previously unseen class of Maros–Mészáros instances. 107
2.12 Maros–Mészáros set: Data (left) and performance (right) profiles comparing the influence of the RL policy on solver configurations with different values for ξ. The results with the default solver setup ($\xi = 10^{-2}$) and with fixed $\alpha_x = \alpha_\lambda = \alpha_\mu$ are provided for comparison. The learned RL policy adapts well to different solver tuning, yielding consistent performance gains. 108

3.1 Visualization of the Rosenbrock function. Left: 3D surface illustrating the curved valley and local minima. Right: Contour plot on the (x_1, x_2) plane with the gradient field. 115
3.2 Weighted quadratic-quartic sum function. Left: $f(v_1)$ minimisation process. Right: Phase space $v_1 - \Delta v_1$. 116
3.3 Visualization of constrained saddle point problem. Left: Contour plot on the (x_1, x_2) plane with the gradient field. Right: 3D surface illustrating the curved valley and local minima. 117

1.1 Basic structure of the Model Predictive Controller feedback loop. 123
1.2 MPC scheme - State-Control couple with a predictions using a fixed rolling horizon. 124
1.3 Modular path-planner framework. 126

2.1 Evaluation of a fixed-wing path-planning optimal control problem on randomized scenarios. 133
2.2 Evaluation of a quadrocopter path-planning optimal control problem on randomized scenarios. 135
2.3 Trajectories of the quadrocopter in randomized scenarios. 136
2.4 Velocities of the quadrocopter in randomized scenarios. 136
2.5 Control inputs: thrust (u_N), roll rotational acceleration (u_γ), pitch rotational acceleration (u_ϕ) and yaw rotational acceleration (u_ψ). 136
2.6 Two-agent collision avoidance set-up and projections. 139
2.7 Trajectories of eight agents at $t = 5.0$ seconds, $t = 25.0$ seconds, $t = 50.0$ seconds and $t = 100.0$ seconds. 141

2.8 Collision avoidance of two agents at $t = 13.5$ seconds, $t = 14.5$ seconds, $t = 15.5$ seconds and $t = 17.0$ seconds. 142
2.9 States ψ and ψ_r of agents 1 and 5 during the collision avoidance phase. . . . 143
2.10 Controls u of agents one and five during the collision avoidance phase. . . . 143
2.11 Lateral deviation per iteration for each agent. 143

3.1 Very light aircraft. 146
3.2 Simulation Setup, Left: *xplane* simulator and pilot, Right: First Person View of the Human-Machine Interface. 147
3.3 Sensor-pod under the right wing. 147
3.4 Avoidance trajectory. 149
3.5 Inboard and ground station hardware setup. 150
3.6 Left: Digital twin of the very light aircraft, Right: Cockpit view inflight. . . . 153
3.7 Altitude evolution, aircraft and predicted points. 154
3.8 Multiple zone avoidance, trajectory and predicted positions. 155
3.9 Multiple zone avoidance in dense area, trajectory and predicted points. 156
3.10 Multiple zone avoidance in dense area, zoom on zone Z_4. 157

4.1 Trajectories UAV-UGV - Unreal engine. 160
4.2 Architecture UAV-UGV. 164
4.3 Trajectories UAV-UGV. 166
4.4 UAV Velocities: MPC vs. simulator. 166
4.5 UAV controls. 167
4.6 Distribution of the computational time required for solving the MPC optimal control problem at each control loop of the quadrocopter. 167
4.7 Computational time required at each control loop to solve the path-planning problem for the quadrocopter. 168

List of Tables

1.1 Applied regularization and safeguards. . . . 67
1.2 Augmented Lagrangian and proximal weights applied and their geometric decrease, see Figure 1.6. . . . 68
1.3 Augmented Lagrangian and proximal weights applied and their geometric decrease . . . 69
1.4 Applied regularization and safeguards . . . 70
1.5 Fixed parameters for the `QProxIP` solver. . . . 71
1.6 Augmented Lagrangian and proximal weights applied and their geometric decrease. . . . 71
1.7 Vehicle Routing parameters. . . . 75
1.8 Tokamak Stability Parameters. . . . 78
1.9 Portfolio optimization parameters . . . 81
1.10 Constituent companies of the European portfolio. . . . 81
1.11 OCP parameters and physical bounds. . . . 86

2.1 PPO hyperparameters used in training. . . . 96
2.2 Default solver setup. . . . 96
2.3 Setup to build random QPs for training and validation sets. . . . 98
2.4 Reward weights during the multistage training procedure. . . . 100
2.5 Applied regularization and safe-guards applied in the subsection. . . . 106

2.1 Parameters relevant for the path-tracking. . . . 140

3.1 Computational and system parameters values. . . . 152
3.2 States and controls box constraints' values. . . . 153

4.1 UAV and platform velocities in selected studies. . . . 161
4.2 MPC and system parameters values. . . . 164
4.3 UAV states and controls box constraints' values. . . . 165
4.4 Maros–Mészáros test-set results. . . . 172

List of Symbols

Part I — Quadratic Programming

Symbol	Name
$\mathbf{1}_n$	Vector of ones of dimension n
$\mathbf{0}_n$	Vector of zeros of dimension n
$\mathbb{I}_n$	Identity matrix of size $n \times n$
n, m, p	Number of variables, equality constraints, inequality constraints
x	Primal variable
s	Slack variable
$\theta_\lambda, \theta_\mu$	Lagrangian multiplicators In/Equality constraints
θ_x, θ_s	Euclidean Proximal estimators
λ, μ	Dual auxiliary variables
$\delta_\lambda, \delta_\mu$	Augmented Lagrangian weights
δ_x, δ_s	Proximal weights
$\mathcal{L}$	Lagrangian
ν	Barrier parameter
ϵ	Algorithm termination threshold
ε	Early-termination of the innerloop
$\phi, \mathcal{M}$	Merit function

Part II — Model Predictive Control and Path Planning

Symbol	Name	Unit
$x(t)$	Continuous state	-
$u(t)$	Continuous control	-
x_k	Discretized state	-
u_k	Discretized control	-
$f_c(x, u)$	Continuous dynamics	-
$f_d(x, u)$	Discretized dynamics	-
x, y, z	Position coordinates	m
$\dot{x}, \dot{y}, \dot{z}$	Velocity components in inertial frame	$\mathrm{m\,s^{-1}}$
$\ddot{x}, \ddot{y}, \ddot{z}$	Acceleration components in inertial frame	$\mathrm{m\,s^{-1}}$
N	Thrust magnitude	N
m	Mass of the vehicle	kg
g	Gravitational acceleration	$\mathrm{m\,s^{-2}}$
ψ	Roll angle	rad
ϕ	Pitch angle	rad
γ	Yaw angle	rad
ω_ψ	Roll angular rate	$\mathrm{rad\,s^{-1}}$
ω_ϕ	Pitch angular rate	$\mathrm{rad\,s^{-1}}$
ω_γ	Yaw angular rate	$\mathrm{rad\,s^{-1}}$

List of Operators

Part I — Quadratic Programming

Operator	Name
$\cdot$	Vector product
$< ., . >$	Vector product
$\circ$	Hadamard element-wise vector product
$\oslash$	Hadamard element-wise division

Publications and Lectures

Proceedings

- Fixed-Wing UAV Path Planning and Collision Avoidance using Nonlinear Model Predictive Control and Sensor-based Cloud Detection, October 2023, 2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC), Jeremy Bertoncini, Adrian Dudek, Martin Russ, Matthias Gerdts, Peter Stütz.
- MissionLab: A Next Generation Mission Technology Research Platform Based on a Very Light Aircraft, January 2024, AIAA SciTech Forum, Johannes Ostler, Adrian Dudek, Jeremy Bertoncini, Martin Russ, Peter Stütz.
- Human-Machine Pilot Navigation Method Using Nonlinear Model Predictive Control, Simulation and Inflight Testing, November 2024, 2024 AIAA DATC/IEEE 43rd Digital Avionics Systems Conference (DASC), Jeremy Bertoncini, Adrian Dudek, Johannes Ostler, Martin Russ, Matthias Gerdts.

Journal Publications

- Multi-Agent Dynamic Scheduling with A Posteriori Path Tracking and Collision Avoidance Using Model Predictive Control, December 2023, IEEE Control Systems Letters, Jeremy Bertoncini, Viktoriya Nikitina, Matthias Gerdts.
- Reinforcement learning for adaptive interior point methods in convex quadratic programming, Jeremy Bertoncini, Alberto De Marchi, Matthias Gerdts, Simon Gottschalk, https://arxiv.org/abs/2509.07404

Conferences with lectures

- October 2023, IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC, Barcelona, Spain),
- December 2023, Conference on Decision and Control (CDC, Singapore),
- June 2024, Learning and Intelligent OptimizatioN Conference (LION, Ischia, Italy),
- October 2024, IEEE/AIAA 43rd Digital Avionics Systems Conference (DASC, San Diego, USA),
- March 2025, Study Association of the German society for defence technology (DWT), Unmanned systems X (Bonn, Germany)

- June 2025, EUROPT conference, (Southampton, United-Kingdom)

Disclaimer: The English language of this thesis has been revised and improved with the assistance of artificial intelligence tools. The author remains responsible for the final content.

Introduction

Background and Context

This research was conducted between February 2022 and October 2025 as part of the Project *MissionLab* within the University of the Bundeswehr Munich in the research group *Ingenieur Mathematik* of Professor Gerdts. The project was funded by *dtec.bw*, the center for digitalization and technology research of the Bundeswehr, and aimed to address real-time trajectory optimization of aerial unmanned vehicles problems. Situated within the broader context of optimization and applied mathematics, the research reflects the collaborative efforts of a multidisciplinary team specializing in optimization, optimal control, sensors integration and flight mechanics.

Content of the Thesis

This dissertation is divided into two main parts. The first part presents both theoretical and numerical results for Quadratic Programs (QP) and Nonlinear Programs (NLP). We present a primal-dual interior-point solver for Quadratic Programs that combines proximal and augmented Lagrangian regularization to improve convergence speed and robustness. The modular solver efficiently handles equality and inequality constraints, is benchmarked against state-of-the-art QP solvers software, and demonstrates high accuracy and robustness on challenging problems, including ill-conditioned or poorly scaled cases. In addition, several case studies from optimal control, financial analysis, and industrial applications are presented to illustrate the performance and versatility of the proposed approach.

Moreover, we propose a reinforcement learning (RL) framework to automatically tune proximal and augmented Lagrangian regularization hyperparameters in the presented Quadratic Programming solver. Trained on random QP instances and tested on standard benchmarks, the RL policy adapts parameters dynamically, enhancing convergence speed and robustness while maintaining theoretical guarantees at a low training cost.

To extend the capabilities of our QP solver to nonlinear problems, we integrate it into a Sequential Quadratic Programming (SQP) framework. SQP is a widely used method for solving constrained nonlinear nonconvex optimization problems by iteratively solving a series of quadratic approximations. At each iteration, the nonlinear problem is linearized and a QP subproblem is solved to generate a search direction for the primal variables and the Lagrangian multiplicators. By embedding our stabilized QP solver within an inhouse SQP framework, we can efficiently tackle Nonlinear Programs (NLPs), including optimal control problems. This integration serves as the natural bridge to Part II of the dissertation, where

the focus shifts to practical applications. All optimal control problems presented are solved using this SQP-based approach, leveraging the robustness and efficiency of our QP solver.

In Part II, we apply the methods developed in Part I to a wide range of autonomous aeronautical applications. Optimal control is a mathematical approach for determining control inputs that guide a dynamic system toward desired objectives while satisfying physical constraints. It is widely used in engineering applications, from aerospace and robotics. With this foundation, we introduce rigorously an Optimal Control Problem (OCP) framework and outline the Model Predictive Control (MPC) that will be used throughout the subsequent chapters. A brief overview of stability considerations is given to ensure well-posed problems and obtain efficient trajectories. Building on this, Model Predictive Control (MPC) is presented in both continuous and discrete settings, establishing a framework that will be used consistently throughout the subsequent chapters of the thesis.

In this thesis, we consider three classes of vehicles: fixed-wing aircraft, quadrocopters, and ground vehicles. Each system exhibits distinct dynamics and control challenges. Fixed-wing aircraft are characterized by forward motion, limited maneuverability at low speeds, and aerodynamic constraints. Control focuses on maintaining desired velocity, altitude, and heading while respecting turn-radius limitations and aerodynamic feasibility. Quadrocopters are highly maneuverable, capable of vertical takeoff and hovering, and can execute rapid, precise and agile maneuvers. Control involves managing both translational and rotational dynamics to achieve desired positions and orientations, often in the presence of disturbances or obstacles. Ground vehicles operate on planar surfaces and are generally constrained by kinematic limitations such as steering angles and maximum velocities. Control strategies often aim to follow reference paths while ensuring comfort. This kinematic understanding provides the foundation for designing and controlling trajectories for all subsequent optimal control and MPC scenarios in the thesis.

After extensive simulations, a real-world use case was conducted based on a very-light aircraft flight tests. A graphical interface was developed to display trajectories computed in real-time to pilots, effectively acting as a low-level controller following them using a tunnel-in-the-sky method. This setup allowed validation of both the trajectory generation algorithms and the guidance interface under realistic operational conditions, bridging the gap between simulation and practical deployment. The developed tools provide guidance to pilots, acting as a decision support system for the air traffic management. Applications include military operations, such as no-fly zone avoidance, and civil air traffic scenarios, particularly in airports operating under Visual Flight Rules (VFR).

The automation of individual agents offers numerous future applications, but interactions between autonomous systems open new possibilities. The automatic landing of a drone on an unmanned ground vehicle (UGV) is presented in this research. The UGV performs path-tracking along a predefined trajectory and does not adapt to the drone approach, as human operators could be in the ground vehicle. The drone, having a path-planner based on Model Predictive Control (MPC), must land safely on the moving UGV while handling high velocities, reaching up to 15 m/s. This case demonstrates the ability of MPC-based methods to safely coordinate multi-agent interactions, handle high-speed maneuvers, and achieve precise alignment in dynamic environments, highlighting the potential of autonomous systems for complex cooperative missions.

Contributions

The first part of this work introduces the `QProxIP` solver, which leverages primal-dual regularization to address degenerate Quadratic Programming (QP) problems within a stabilized primal-dual Interior-Point Method (IPM) framework. A novel proximal regularization for slack variables is proposed, accompanied by safeguards for the central path, a concept well-established in theory but rarely applied in practical implementations. In addition, theoretical proofs are provided for cases where the literature is lacking, particularly regarding convergence when scaling the IPM-KKT matrix by the positive slack variables. This research delivers a competitive, robust implementation of a stabilized Primal-Dual IPM solver. Additionally, the presented precursor study on proximal-ALM hyperparameter tuning using reinforcement learning with a lightweight neural network demonstrated reduced sensitivity to initial solver parameters.

The second part of this work focused on the practical implementation of model predictive control (MPC) for both simulation and real hardware, including real flight tests. A modular mission-planning framework was developed, providing flexible integration with communication protocols for sensor and vehicle interaction. Furthermore, a concept for high-velocity docking maneuvers between a quadrocopter and an unmanned ground vehicle (UGV) was introduced, leveraging two decoupled systems to ensure robust coordination and trajectory feasibility.

Part I

A Modular QP Solver: Regularization, Convergence, Learning-based Enhancement

Chapter 1

A Modular Regularized QP Solver

1.1 Introduction

Quadratic programming (QP) is a key optimization problem that has applications in a broad spectrum of disciplines, including machine learning, finance, and control systems. The problem is a minimization of a quadratic objective function subject to linear equality and inequality constraints. Among many approaches developed to solve (QP) problems, interior-point methods have emerged as a particularly powerful and versatile class of techniques. At every iteration, interior-point methods relax the inequality constraints using barrier functions penalizing the Lagrangian. By progressively reducing the influence of the barrier term, the algorithm converges to an optimal solution that satisfies both the objective and constraints.

To enhance the performance and robustness of QP solvers, regularization techniques play a decisive role. The Augmented Lagrangian Method (ALM) and proximal-point regularizations introduce additional terms in order to stabilize the numerical scheme and linear algebra, particularly in ill-conditioned or degenerate cases. By adding a quadratic penalty term, the augmented Lagrangian method enforces dual updates and ensures better convergence properties. Proximal-point methods incorporate a neighborhood operator convexifying the original problem and ensuring super-linear convergence to the solution. A particular emphasis is placed on the Gill-Robinson primal-dual regularization [54], highlighting the respective roles of the primal and dual variables in the construction of the Lagrangian. Through this exploration, our research aims to provide a comprehensive understanding of these tools and methods as well as their applications in solving real-world QP problems.

Methods for the numerical solution of QPs have been extensively studied. These differ in how they balance the number of iterations and the computational cost for each iteration. First-order methods, such as OSQP [119] based on ADMM [23], build upon simple steps but typically require many iterations, so several acceleration schemes have been proposed [6, 37, 56, 70]. In contrast, Newton-type methods such as interior point (IP) [26, 50, 51], active set [41] and semismooth Newton [87] usually need fewer, yet more demanding, iterations to reach accurate solutions. Recently, proximal techniques [101] and the augmented Lagrangian (AL) framework [108] have seen a resurgent interest due to their inherent regularization properties, resulting in the development of solvers such as QPDO [34], QPALM [68] and ProxQP [11]. Along this vein, the integration of proximal regularization with the IP strategy led to IP-PMM [103], PS-IPM [29] and PIQP [114]. Among others, QPDO [34] and PS-IPM [29] have been developed in this spirit so that proximally regularized subproblems can be (efficiently and reliably) solved with Newton-type methods. Thanks to proximal regularization, these solvers can cope with matrix Q merely positive semi-definite and with linear

equality and inequality matrices A, G having linearly dependent rows. For further regularization, a Tikhonov-like proximal term is included to handle also non-strictly convex QPs, as in [11, 15, 119].

1.2 Convex optimization

This section is based on the work of Boyd and Vandenberghe [24][Chap. 1-5] and Gerdts and Lempio [49][Chap. 5].

1.2.1 Problem statement

The convex optimization problem is formulated as,

Problem 1.2.1 — **Convex problem**

$$\begin{aligned} \underset{x\in K}{\text{minimize}} \quad & f(x) \\ & h_i(x) = 0, \quad i = 1,...,m, \text{ be affine}, \\ & g_i(x) \leq 0, \quad i = 1,...,p, \text{ be convex}, \end{aligned} \tag{1.1}$$

$$\begin{aligned} \text{with} \quad & f : \mathcal{K} \to \mathbb{R}, \\ & h_i : \mathcal{K} \to \mathbb{R}^m, \quad i = 1,...,m, \\ & g_i : \mathcal{K} \to \mathbb{R}^p, \quad i = 1,...,p, \end{aligned}$$

with K a convex subset of a real vector space X, see Definition 1.2.1. On this set, we consider the convex functions f, g_i and the affine functions $h_i(x)$, with f an objective function, h_i a equality constraints and g_i inequality constraints, see Definition 1.2.2.

Definition 1.2.1 — **Convex set**

Let K be a subset of a real vector space X. We call K convex if for any $x, y \in K$ and any $\theta \in [0,1]$, the point

$$\theta x + (1-\theta)y$$

on the line segment between x and y also belongs to K.

Definition 1.2.2 — **Convex and affine functions**

A function $f : K \to \mathbb{R}$ is called convex if for all $x, y \in K$ and all $\theta \in [0,1]$,

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y). \tag{1.2}$$

The function is called strictly convex if inequality (1.2) is strict.

A function $f : K \to \mathbb{R}$ is called affine if for all $x, y \in K$ and all $\theta \in [0,1]$,

$$f(\theta x + (1-\theta)y) = \theta f(x) + (1-\theta)f(y).$$

Definition 1.2.3 — **Domain of a function**

The domain of a function $f\colon K \to \mathbb{R} \cup \{+\infty\}$ is defined as

$$\operatorname{dom}(f) := \{\, x \in K \mid f(x) < +\infty \,\}.$$

1.2.2 Hyperplane separation

To derive necessary optimality conditions, we require results on the separation of convex sets by hyperplanes. These results will be applied exclusively in finite-dimensional spaces, and thus we consider sets in $\mathbb{R}^n$. A hyperplane in $\mathbb{R}^n$ is an affine set that divides the space into two parts. Each hyperplane naturally defines two halfspaces. Formally, Definition 1.2.4 provides the precise formulation of a hyperplane.

Definition 1.2.4 — **Hyperplane and halfspace**

Let be $a \in \mathbb{R}^n$, $a \neq 0$ and $\gamma \in \mathbb{R}$. The set $H = \{x \in \mathbb{R}^n : a^\top x = \gamma\}$ is called an hyperplane. The sets $H_+ = \{x \in \mathbb{R}^n : a^\top x \geq \gamma\}$ and $H_- = \{x \in \mathbb{R}^n : a^\top x \leq \gamma\}$ are called positive/negative halfspace of H.

To analyze the convexity of sets, we introduce the concept of separation to characterizes how two sets can be distinguished by a hyperplane. Intuitively, a hyperplane separates two sets if one set lies entirely on one side and the other set on the opposite side. Depending on the strictness of this condition, we speak of separation, strict separation, or proper separation, see Definition 1.2.5.

Definition 1.2.5 — **Sets separation**

Let $A, B \subset \mathbb{R}^n$ be nonempty sets, and let $H = \{\, x \in \mathbb{R}^n : a^\top x = \gamma\}$, be a hyperplane with $a \neq 0$.

- The hyperplane H separates the sets A and B if $a^\top x \leq \gamma$ for all $x \in A$, $a^\top x \geq \gamma$ for all $x \in B$.
- The hyperplane H strictly separates A and B if $a^\top x < \gamma$ for all $x \in A$, $a^\top x > \gamma$ for all $x \in B$.
- The hyperplane H properly separates A and B if it separates them and not both sets lie entirely within H.

In convex optimization, distinguishing points and convex sets, as well as pairs of convex sets, allows one to derive optimality conditions and establish duality results. The following two theorems formalize these separation results. Theorem 1.2.1 establishes the separation of a single point from a convex set, guaranteeing the existence of a hyperplane that separates the origin from a nonempty convex set not containing it. If the set is additionally closed, the separation can be made strict. Theorem 1.2.2 generalizes this result to two disjoint convex sets, providing conditions under which a hyperplane separates them. If one set is closed and the other is compact, the sets can be strictly separated, ensuring a nonzero gap between them.

Theorem 1.2.1 — **Separation of a point and a convex set**

Let $A \subset \mathbb{R}^n$ be a nonempty convex set with $0 \notin A$. Then there exists $a \in \mathbb{R}^n \setminus \{0\}$ such that

$$a^\top x \geq 0 \quad \text{for all } x \in A.$$

If, in addition, A is closed, then $\{0\}$ can be strictly separated from A, that is, one can choose a such that

$$a^\top x > 0 \quad \text{for all } x \in A.$$

Theorem 1.2.2 — **Separation of two convex sets**

Let $A, B \subset \mathbb{R}^n$ be nonempty convex sets with $A \cap B = \emptyset$. Then there exists a hyperplane that separates A and B, i.e., there exist $a \in \mathbb{R}^n \setminus \{0\}$ and $\gamma \in \mathbb{R}$ such that

$$a^\top x \geq \gamma \quad \text{for all } x \in A,$$

$$a^\top x \leq \gamma \quad \text{for all } x \in B.$$

If, in addition, A is closed and B is compact, then A and B can be strictly separated, that is, one can choose a and γ such that

$$a^\top x > \gamma > a^\top y \quad \text{for all } x \in A,\ y \in B.$$

As introduced in Definition 1.2.6, an affine set is a set that contains all affine combinations of its points. The affine hull of a set is the minimal affine set containing it. The relative interior of a set consists of all points that possess a neighborhood contained within the set relative to its affine hull. These notions provide the geometric framework for the following results.

Definition 1.2.6 — **Affine set, affine hull, relative interior**

Let $A \subset \mathbb{R}^n$.

(a) Affine set: The set A is called *affine* if for all $x, y \in A$ and all $\theta \in \mathbb{R}$,

$$(1-\theta)x + \theta y \in A.$$

(b) Affine hull: The *affine hull* of A is the smallest affine set containing A, denoted by $\operatorname{aff}(A)$. It can be written as

$$\operatorname{aff}(A) = \left\{ \sum_{i=1}^{m} \theta_i x_i \;\middle|\; m \in \mathbb{N},\ \sum_{i=1}^{m} \theta_i = 1,\ \theta_i \in \mathbb{R},\ x_i \in A \right\}.$$

(c) Relative interior: The *relative interior* of A is defined as

$$\operatorname{relint}(A) = \{ x \in A \mid \exists\, \varepsilon > 0 \text{ such that } B_\varepsilon(x) \cap \operatorname{aff}(A) \subset A \},$$

where $B_\varepsilon(x)$ denotes the open Euclidean ball of radius ε centered at x.

Theorem 1.2.3 guarantees that every nonempty convex set has a nonempty relative interior. In other words, a convex set always contains points that are interior with respect to its affine hull. Theorem 1.2.4 establishes a precise condition for the proper separation of two convex sets. Indeed, two convex sets can be properly separated by a hyperplane if and only

if their relative interiors do not intersect.

Theorem 1.2.3 — **Non-emptiness of the relative interior**

Let $A \subset \mathbb{R}^n$ be nonempty and convex. Then the relative interior of A is nonempty, i.e.

$$\operatorname{relint}(A) \neq \emptyset.$$

Theorem 1.2.4 — **Proper separation**

Let $A, B \subset \mathbb{R}^n$ be nonempty and convex. There exists a hyperplane that properly separates A and B if and only if

$$\operatorname{relint}(A) \cap \operatorname{relint}(B) = \emptyset.$$

1.2.3 Optimality

To identify optimal solutions, we rely on optimality conditions giving a certificate of the solution's quality. Necessary optimality conditions characterize properties that must be satisfied by any optimal solution, while sufficient conditions provide criteria for identifying optimal solutions among feasible candidates. In the setting of convex optimization, these conditions are closely aligned. The theoretical foundations of the following results trace back to the work of John, Valentine, Slater, Karush, and Kuhn-Tucker, see [71, 76, 82, 116, 124].

In what follows, we restrict attention to finite-dimensional convex optimization problems and carefully distinguish between convex and affine inequality constraints. For the mathematical treatment of affine constraints, see [21]. This section draws primarily from the material presented in Lecture [131].

We denote an optimal point x^* of a convex constrained optimization problem if it satisfies all constraints and $f(x^*) \leq f(x), \quad \forall x$ feasible. We denote by X_{opt} the set of all such optimal points. Definition (1.2.7) introduces the concept of a locally optimal solution candidate within a certain neighborhood. This differs from global optimality, where a point must minimize the objective over the entire feasible set.

Definition 1.2.7 — **Local Optimality**

A feasible point x is called locally optimal for Problem (1.2.1) if there exists an $R > 0$ such that

$$f(x) \leq f(y)$$

for all feasible y satisfying $\|y - x\|_2 \leq R$. Equivalently, x solves

$$\begin{aligned} \text{minimize} \quad & f(z) \\ \text{subject to} \quad & h_i(z) = 0, \quad i = 1, \ldots, m, \\ & g_i(z) \leq 0, \quad i = 1, \ldots, p, \\ & \|z - x\|_2 \leq R. \end{aligned}$$

However, in the case of convex optimization problems, local optimality is sufficient to guarantee global optimality, as formalized in Theorem (1.2.5).

Theorem 1.2.5 — **Global solution**

Any locally optimal point of a convex optimization problem is also globally optimal.

Proof 1.2.1 — **Global solution**

Suppose x is locally optimal and let $y \neq x$ be a globally optimal point with

$$f(y) < f(x).$$

Since x is locally optimal, there exists $R > 0$ such that

$$\text{if } z \text{ is feasible and } \|z - x\|_2 \leq R, \quad \text{then } f(z) \geq f(x).$$

Consider the point

$$z = \theta y + (1-\theta)x, \qquad \theta = \frac{R}{2\|y - x\|_2}.$$

Because $\|y - x\|_2 > R$, we have $\theta \in (0, 1/2)$.
Since z is a convex combination of two feasible points, it is feasible.
Furthermore,

$$\|z - x\|_2 = \theta\|y - x\|_2 = \frac{R}{2},$$

so $\|z - x\|_2 \leq R$. By convexity of f,

$$f(z) \leq \theta f(y) + (1-\theta)f(x) < f(x),$$

which contradicts the assumption that x is locally optimal. Thus, no such point y can exist. □

Theorem (1.2.6) elaborate first-order conditions for optimality using the gradient of the objective function. Geometrically, the condition $\nabla f(x)^\top (y - x) \geq 0$ implies that there are no feasible descent directions in the neighborhood of x. In other words, all feasible directions locally point either uphill or lie tangent to a level set of f, and keep the objective function value constant.

Theorem 1.2.6 — **First-Order Condition for Convex Problems**

Let the objective function f of Problem (1.2.1) be differentiable, and let be X the feasible set. Then a point x is optimal if and only if $x \in X$ and

$$\nabla f(x)^\top (y - x) \geq 0 \qquad \forall\, y \in X.$$

Proof 1.2.2 — **First-Order Condition for Convex Problems**

Suppose $x \in X$. We want to show

$$f(x) \leq f(y) \quad \forall y \in X \iff \nabla f(x)^\top (y - x) \geq 0 \quad \forall y \in X.$$

First-Order Condition for Convex Problems

Suppose $\nabla f(x)^\top (y - x) \geq 0$ for all $y \in X$. Since f is convex, for all $y \in X$,

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x) \geq f(x),$$

so x is optimal.

Suppose x is optimal, but there exists $y \in X$ with $\nabla f(x)^\top (y - x) < 0$. Consider the point

$$z(t) = ty + (1 - t)x, \quad t \in [0, 1],$$

with $z(t) \in X$ for all $t \in [0, 1]$. Then

$$\lim_{t \to 0} \frac{f(z(t)) - f(x)}{t} = \nabla f(x)^\top (y - x) < 0.$$

For sufficiently small $t > 0$, this implies $f(z(t)) < f(x)$, which contradicts the assumption that x is optimal.
Therefore, $\nabla f(x)^\top (y - x) \geq 0$ for all $y \in X$. □

In unconstrained convex optimization, first-order conditions using the gradient of the objective are sufficient to characterize optimality. However, in the presence of constraints, feasible directions are restricted, and the gradient alone no longer captures all necessary information. The Fritz John Theorem (1.2.7) generalizes first-order optimality conditions to constrained optimization problems without assuming any regularity conditions, i.e., without requiring constraint qualifications such as linear independence of the active constraint gradients. It states that if a feasible point $\tilde{x}$ is a local minimum, then there exist multipliers (w, u, v), not all zero, satisfying the conditions in Theorem (1.2.7).

Geometrically, the Fritz John conditions can be interpreted as a balance of forces at the optimum. The gradient of the objective function at $\tilde{x}$ is expressed as a nonnegative linear combination of the gradients of the active constraints. This means that, at $\tilde{x}$, there are no feasible directions that decrease the objective while remaining within the constraints and every feasible direction either increases the objective or is neutral along the active constraint surfaces.

Theorem 1.2.7 **Fritz John Necessary Conditions**

Let $\tilde{x}$ be a feasible solution of optimization Problem (1.2.1). If $\tilde{x}$ is a local minimum, then there exist multipliers (w, μ, λ) such that

$$\begin{aligned} w\nabla f(\tilde{x}) + \sum_{i=1}^{p} \mu_i \nabla g_i(\tilde{x}) + \sum_{i=1}^{m} \lambda_i \nabla h_i(\tilde{x}) &= 0, \\ (w, \mu) &\geq 0, \\ (w, \mu, \lambda) &\neq 0, \\ \mu_i \cdot g_i(\tilde{x}) &= 0, \quad i = 1, \ldots, p. \end{aligned}$$

However, the Fritz John conditions are not always sufficient, because they allow the degenerate case in which the multiplier associated with the objective function is zero. In such situations, the optimality system does not provide a meaningful relationship between the objective and the active constraints, and the conditions may hold even at points that are not genuine candidates for optimality. The Karush-Kuhn-Tucker (KKT) conditions addresses

these limitations by imposing a regularity assumption, namely the linear independence of the gradients of the active constraints. Under this assumption, the multiplier associated with the objective is strictly positive, and the resulting KKT system provides a nondegenerate characterization of local optimality.

Theorem (1.2.8) formalizes these necessary conditions for a local minimum $\tilde{x}$ of a standard-form constrained optimization Problem (1.2.1). Geometrically, the KKT conditions imply that at the optimal point $\tilde{x}$, the gradient of the objective function lies in the cone generated by the gradients of the active inequality constraints together with the linear span of the equality constraints. In other words, no feasible descent directions exist: any infinitesimal move along a feasible direction either increases the objective or remains tangent to the active constraint surfaces, ensuring local optimality.

Theorem 1.2.8 **KKT Necessary Conditions**

Let $\tilde{x}$ be a feasible solution of the standard-form optimization problem, and let

$$I = \{\, i \mid g_i(\tilde{x}) = 0,\ i = 1, \dots, p \,\}, \text{ the active inequality index set.}$$

Suppose that the collection of vectors $\{\nabla g_i(\tilde{x}) \mid i \in I\} \cup \{\nabla h_i(\tilde{x}) \mid i = 1, \dots, m\}$ is linearly independent. If $\tilde{x}$ is a local minimum, then there exist multipliers (μ, λ) such that

$$\begin{aligned} \nabla f(\tilde{x}) + \sum_{i=1}^{p} \mu_i \nabla g_i(\tilde{x}) + \sum_{i=1}^{m} \lambda_i \nabla h_i(\tilde{x}) &= 0, \\ \mu &\geq 0, \\ \mu_i \cdot g_i(\tilde{x}) &= 0, \quad i = 1, \dots, p. \end{aligned}$$

1.2.4 Duality

The Lagrangian is a central tool in constrained optimization that incorporates both the objective function and the constraints into a single function.

Let $\mathcal{D} = \operatorname{dom}(f) \cap \bigcap_{i=0}^{m} \operatorname{dom}(h_i) \cap \bigcap_{i=1}^{p} \operatorname{dom}(g_i)$ denote the domain where all objective and constraint functions are defined.

Definition 1.2.8 **Lagrangian**

The Lagrangian is defined as,

$$\begin{aligned} &\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}, \quad \operatorname{dom}(\mathcal{L}) = \mathcal{D} \times \mathbb{R}^m \times \mathbb{R}^p, \\ &\mathcal{L}(x, \mu, \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i h_i(x) + \sum_{i=1}^{p} \mu_i g_i(x), \end{aligned} \tag{1.3}$$

where λ_i and μ_i are Lagrange multipliers associated with the equality constraints $h_i(x) = 0$ and the inequality constraints $g_i(x) \leq 0$, respectively.

As shown in Definition (1.2.8), the Lagrangian allows us to combine the objective and

constraints in a single function. We will now derive the dual problem and analyze its optimality conditions.

The Lagrangian dual function is obtained by minimizing the Lagrangian over the primal variables, producing a function of the multipliers alone. We formulate it as follow,

$$q : \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}, \quad q(\mu, \lambda) = \inf_{x \in \mathcal{D}} \mathcal{L}(x, \mu, \lambda), \tag{1.4}$$

where μ_i are the Lagrange multipliers associated with the inequality constraints $g_i(x) \leq 0$ and λ_i those associated with the equality constraints $h_i(x) = 0$. Explicitly, we have

$$\begin{aligned} q(\mu, \lambda) &= \inf_{x \in \mathcal{D}} \left\{ \mathcal{L}(x, \mu, \lambda) \right\}, \\ &= \inf_{x \in \mathcal{D}} \left\{ f(x) + \sum_{i=1}^{m} \lambda_i h_i(x) + \sum_{i=1}^{p} \mu_i g_i(x) \right\}. \end{aligned} \tag{1.5}$$

The dual function $q(\mu, \lambda)$ is always concave in (μ, λ), even if f is not convex, and it can take the value $-\infty$ for some choices of multipliers, see [131].

The following Theorem (1.2.9) formalizes a fundamental property of the Lagrangian dual function: every value it attains yields a valid lower bound on the primal optimal value. This property is the cornerstone of Lagrangian duality, as it provides the basis for constructing dual problems and analyzing optimality gaps.

Theorem 1.2.9 — **Lower Bounds of $q(\mu, \lambda)$**

For any $\mu \geq 0$ and any λ, the Lagrangian dual function satisfies

$$q(\mu, \lambda) \leq l^*,$$

where l^* is the optimal value of the primal problem.

Proof 1.2.3 — **Lower Bounds of $q(\mu, \lambda)$**

Let $\tilde{x}$ be any feasible point of the primal problem. Since $\mu \geq 0$, we have

$$\sum_{i=1}^{p} \mu_i g_i(\tilde{x}) + \sum_{i=1}^{m} \lambda_i h_i(\tilde{x}) \leq 0.$$

Hence,

$$\mathcal{L}(\tilde{x}, \mu, \lambda) = f(\tilde{x}) + \sum_{i=1}^{p} \mu_i g_i(\tilde{x}) + \sum_{i=1}^{m} \lambda_i h_i(\tilde{x}) \leq f(\tilde{x}).$$

Lower Bounds of $q(\mu, \lambda)$

Taking the infimum over all $x \in \mathcal{D}$, we obtain

$$q(\mu, \lambda) = \inf_{x \in \mathcal{D}} \left\{ \mathcal{L}(x, \mu, \lambda) \right\} \leq \mathcal{L}(\tilde{x}, \mu, \lambda) \leq f(\tilde{x}).$$

Since this holds for every feasible $\tilde{x}$, it follows that

$$q(\mu, \lambda) \leq l^*,$$

which proves the theorem. □

The lower bound property of the Lagrangian dual function naturally motivates the formulation of the dual problem, which seeks the best possible lower bound on the primal optimal value. By selecting multipliers that yield the tightest possible lower bound, one obtains a strong certificate of optimality. This leads to the formulation of the Lagrange dual problem, which consists of maximizing the dual function over all admissible multipliers.

Problem 1.2.2 — **Lagrange Dual Problem**

$$\begin{aligned} &\text{maximize} && q(\mu, \lambda) \\ &\text{subject to} && \mu \geq 0, \end{aligned} \tag{1.6}$$

where,

$$q(\mu, \lambda) = \inf_{x \in \mathcal{D}} \left\{ \mathcal{L}(x, \mu, \lambda) \right\}, \tag{1.7}$$

is the Lagrangian dual function.

The dual problem seeks the best possible lower bound d^* on the primal optimal value l^*. Formally, $d^* := \sup_{\mu \geq 0, \lambda} q(\mu, \lambda)$, and it represents the tightest lower bound on l^* obtainable by any choice of dual multipliers and corresponds to the optimal value of the dual optimization problem. A pair (μ, λ) is considered dual feasible if $\mu \geq 0$ and $(\mu, \lambda) \in \text{dom}(q)$, meaning that the dual function is finite, $q(\mu, \lambda) > -\infty$. The dual optimal multipliers are denoted by (μ^*, λ^*), and the difference between the primal and dual optimal values, $l^* - d^*$, is referred to as the optimal duality gap.

Weak duality, expressed as $d^* \leq l^*$, holds universally for all optimization problems, whether convex or nonconvex. In contrast, strong duality, meaning $d^* = l^*$, does not hold in general. However, for convex problems, strong duality typically holds, if constraint qualifications are satisfied.

We consider a convex optimization problem where the equality constraints are express as linear $Ax = b$, and written in the form,

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, p, \\ & Ax = b, \quad A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m \end{aligned} \tag{1.8}$$

with variable $x \in \mathbb{R}^n$ and domain $\mathcal{D}_g = \bigcap_{i=0}^{p} \text{dom}(g_i)$.

We introduce Slater's condition as a regularity assumption that strengthens the feasibility requirements of a constrained optimization problem by requiring the existence of a strictly feasible point.

Definition 1.2.9 **Slater's Condition**

The Slater's condition writes,

$$\exists\, x \in \operatorname{relint}(\mathcal{D}_g) \quad \text{with} \quad g_i(x) < 0,\ i = 1, \dots, p, \quad \text{and} \quad Ax = b. \tag{1.9}$$

where the interior is taken relative to the affine hull of $\mathcal{D}_g$. Under these assumptions, the following classical result, known as Slater's Theorem (1.2.10), guarantees strong duality.

Theorem 1.2.10 **Slater's Theorem**

As Problem (1.2.1) is convex, if Slater's condition holds, then strong duality holds, and the optimal dual value equals the optimal primal value.

Now suppose x^* is a primal optimal solution and (λ^*, μ^*) are dual optimal multipliers. Then x^* minimizes the Lagrangian

$$\mathcal{L}(x, \lambda^*, \mu^*) = f(x) + \sum_{i=1}^{p} \mu_i^* g_i(x) + \sum_{j=1}^{m} \lambda_j^* h_j(x), \tag{1.10}$$

and the complementary slackness conditions

$$\mu_i^* g_i(x^*) = 0, \quad i = 1, \dots, p, \tag{1.11}$$

hold. These conditions state that if an inequality constraint is strictly inactive ($g_i(x^*) < 0$), its corresponding multiplier vanishes ($\mu_i^* = 0$), and if the multiplier is positive ($\mu_i^* > 0$), then the constraint is active ($g_i(x^*) = 0$). Moreover, if strong duality holds, x is primal optimal, μ, λ are dual optimal, and g_i, h_i are differentiable, then the KKT conditions must hold, see Theorem (1.2.8).

Given the Lagrangian (1.10) the primal problem can be written in saddle-point form as

$$\inf_{x \in \mathcal{P}} \left\{ \sup_{\mu \geq 0,\, \lambda} \left\{ \mathcal{L}(x, \mu, \lambda) \right\} \right\}, \tag{P}$$

while the corresponding dual problem takes the form

$$\sup_{\mu \geq 0,\, \lambda} \left\{ \inf_{x \in \mathcal{P}} \left\{ \mathcal{L}(x, \mu, \lambda) \right\} \right\}, \tag{D}$$

$$\text{with } \mathcal{P} := \{\, x \in \mathbb{R}^n \mid g_i(x) \leq 0,\ i = 1, \dots, p, \quad h_j(x) = 0,\ j = 1, \dots, m \,\}.$$

the primal feasible set. This formulation provides a natural minimax interpretation where the primal variables x attempt to minimize the Lagrangian, while the dual variables (μ, λ) attempt to maximize it.

Weak duality follows directly from the minimax inequality, with

$$\sup_{\mu\geq 0,\,\lambda}\left\{\inf_{x\in\mathcal{P}}\left\{\mathcal{L}(x,\mu,\lambda)\right\}\right\} \;\leq\; \inf_{x\in\mathcal{P}}\left\{\sup_{\mu\geq 0,\,\lambda}\left\{\mathcal{L}(x,\mu,\lambda)\right\}\right\}. \tag{1.12}$$

This inequality implies that, for any feasible dual variables (μ,λ) and any feasible primal variable $x\in\mathcal{P}$, the value of the dual function provides a lower bound on the primal objective. In other words, the best lower bound achievable by the dual problem cannot exceed the optimal value of the primal problem. However, this inequality alone does not guarantee equality, and therefore does not ensure the existence of a saddle point. To have a saddle point, as in Definition (1.2.10), one requires a point (x^*,μ^*,λ^*) where the minimum and maximum meet exactly, and therefore having lower bound and upper bound coinciding at $\mathcal{L}(x^*,\mu^*,\lambda^*)$.

Definition 1.2.10 **Saddle Point of a Function**

Let $L : X\times Y\to\mathbb{R}$ be a real-valued function defined on the Cartesian product of two sets X and Y. A point $(x^*,y^*)\in X\times Y$ is called a saddle point of L if

$$L(x^*,y)\leq L(x^*,y^*)\leq L(x,y^*),\qquad \forall\, x\in X,\ y\in Y. \tag{1.13}$$

The existence of a saddle point at (x^*,μ^*,λ^*) guarantees strong duality [107][Theorem 28.3], and

$$f(x^*) \;=\; q(\mu^*,\lambda^*) \;=\; \inf_{x\in\mathcal{P}}\left\{\sup_{\mu\geq 0,\,\lambda}\left\{\mathcal{L}(x,\mu,\lambda)\right\}\right\} \;=\; \sup_{\mu\geq 0,\,\lambda}\left\{\inf_{x\in\mathcal{P}}\left\{\mathcal{L}(x,\mu,\lambda)\right\}\right\}, \tag{1.14}$$

the primal and dual optimal values coincide. The points x^* and (μ^*,λ^*) are optimal for the primal and dual problems, respectively.

The properties of strong and weak duality play a central role in certifying optimality in numerical optimization. When strong duality holds, Primal-dual algorithms exploit this saddle-point structure to converge reliably and provide meaningful stopping criteria. In contrast, if only weak duality holds, the dual problem yields merely a lower bound on the primal optimal value. Dual solutions in this case may not correspond to any primal optimum, and complementary slackness conditions cannot be applied. Consequently, primal-dual algorithms may converge to suboptimal points. Thus, strong duality is not only a theoretical guarantee but also provides certitude in identifying the true optimum, enabling robust and efficient numerical methods for constrained optimization.

1.3 Convex Quadratic Programming

In this section, we present the class of convex quadratic programming (QP) problems that form the foundation of our research.

Classical QP solvers typically rely on strong regularity assumptions, such as the Linear Independence Constraint Qualification (LICQ) and strict convexity, to guarantee uniqueness of the solution and fast convergence. When these assumptions are violated, however, their

numerical behavior can deteriorate rapidly, leading to poor convergence or even failure. In contrast, we propose a regularized formulation based on proximal and augmented Lagrangian techniques that fundamentally reduces sensitivity to problem degeneracy. By relaxing the reliance on LICQ, full-rank constraint sets, and strict convexity, our approach remains well posed even in the presence of degenerate, or weakly convex QP instances. The introduced regularization yields stable primal-dual solutions and consistent numerical behavior, thereby expanding the class of QP problems that can be solved reliably.

1.3.1 Mathematical formulation

Convex Quadratic Programming is a particular type of convex optimization problem where the objective is a quadratic function subject to linear constraints. One formulation of a QP problem is

Problem 1.3.1 — **QP**

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & \frac{1}{2}x^\top Qx + c^\top x \\ \text{subject to} \quad & Ax = b \\ & Gx \leq d \end{aligned} \qquad \text{with :} \quad \begin{aligned} & x \in \mathbb{R}^n, \\ & Q \in \mathbb{S}^n_+, \\ & A \in \mathbb{R}^{m\times n}, \\ & G \in \mathbb{R}^{p\times n}, \\ & c \in \mathbb{R}^n, \\ & b \in \mathbb{R}^m, \\ & d \in \mathbb{R}^p, \end{aligned} \qquad \text{(QP)}$$

with $\mathbb{S}^n_+$ the set of positive semi-definite matrices, see def (1.3.3), for some nonnegative integers n, m, and p. We consider Q to be symmetric and at least semi-positive definite, resulting in convex QP problems. In the case of Q positive definite, the problem is strictly convex.

Problem (1.3.1) can be rewritten using a semi-positive definite vector $s \in \mathbb{R}^p$ called slack variables. The slack variables enable the rewriting of the inequality equations as equality constraints while ensuring only the semi-positivity of s. Indeed, by maintaining the semi-positivity of s, the feasibility of x (the interior of the feasible region) is ensured, and the problem is equivalent. Variables x and s are called primal variables, see [24], [96].

Problem 1.3.2 — **reformulated QP**

$$\begin{aligned} \underset{x\in\mathbb{R}^n, s\in\mathbb{R}^p}{\text{minimize}} \quad & \frac{1}{2}x^\top Qx + c^\top x \\ \text{subject to} \quad & Ax = b \\ & Gx - d + s = 0 \\ & s \geq 0 \end{aligned} \qquad \text{with :} \quad \begin{aligned} & x \in \mathbb{R}^n, \\ & s \in \mathbb{R}^p, \\ & Q \in \mathbb{S}^n_+, \\ & A \in \mathbb{R}^{m\times n}, \\ & G \in \mathbb{R}^{p\times n}, \\ & c \in \mathbb{R}^n, \\ & b \in \mathbb{R}^m, \\ & d \in \mathbb{R}^p. \end{aligned} \qquad \text{(QP-s)}$$

1.3.2 Constraints analysis

In the Quadratic Programming (QP) problem formulation, the constraints define the feasible region of the optimization problem. Equality constraints are given by:

$$A \cdot x = b, \tag{1.15}$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. These constraints define a linear subspace of the solution space. The feasible set for the equality constraints, $\mathcal{F}_{\text{eq}}$, is defined as

$$\mathcal{F}_{\text{eq}} = \{x \in \mathbb{R}^n \mid Ax = b\}. \tag{1.16}$$

The feasible set $\mathcal{F}_{\text{eq}}$ is a convex set, as it is the intersection of affine hyperplanes, see Theorem (1.3.1).

Theorem 1.3.1 **Intersection of affine spaces**

Let $A_1, A_2, \ldots, A_n \subseteq \mathbb{R}^d$ be affine spaces. Then their intersection

$$A := \bigcap_{i=1}^{n} A_i$$

is either empty or an affine space.

Proof 1.3.1 **Intersection of affine spaces**

Assume that $A \neq \emptyset$, and let $x, y \in A$, so that $x, y \in A_i$ for all $i = 1, 2, \ldots, n$. Since each A_i is affine, for any $\theta \in \mathbb{R}$, the affine combination

$$z := \theta x + (1 - \theta) y \in A_i \quad \text{for all } i.$$

Therefore, $z \in \bigcap_{i=1}^{n} A_i = A$. Hence, for any $x, y \in A$ and any $\theta \in \mathbb{R}$, we have $\theta x + (1 - \theta) y \in A$, proving that A is an affine space. □

However, feasibility is not guaranteed. If the constraints are inconsistent, the convex feasible set $\mathcal{F}_{\text{eq}} = \emptyset$ leads to an infeasible problem.

Inequality constraints are represented as:

$$G \cdot x \leq d, \tag{1.17}$$

where $G \in \mathbb{R}^{p \times n}$ and $d \in \mathbb{R}^p$. These constraints define a feasible set

$$\mathcal{F}_{\text{ineq}} = \{x \in \mathbb{R}^n \mid Gx \leq d\}, \tag{1.18}$$

where $G \in \mathbb{R}^{p \times n}$ and $d \in \mathbb{R}^p$. The feasible set $\mathcal{F}_{\text{ineq}}$ is convex when the inequality constraints are linear, see Theorem (1.3.2).

Theorem 1.3.2 **Convexity of linear inequality constraints**

Let $G \in \mathbb{R}^{p \times n}$ and $d \in \mathbb{R}^p$. The feasible set

$$\mathcal{F}_{\text{ineq}} := \{x \in \mathbb{R}^n \mid Gx \leq d\}$$

defined by linear inequality constraints is convex.

Proof 1.3.2 **Convexity of linear inequality constraints**

Let $x, y \in \mathcal{F}_{\text{ineq}}$. Then

$$Gx \leq d \quad \text{and} \quad Gy \leq d.$$

For any $\theta \in [0, 1]$, consider the convex combination

$$z := \theta x + (1 - \theta)y.$$

By linearity of G, we have

$$Gz = \theta Gx + (1 - \theta)Gy \leq \theta d + (1 - \theta)d = d.$$

Thus, $z \in \mathcal{F}_{\text{ineq}}$, which shows that

$$\theta x + (1 - \theta)y \in \mathcal{F}_{\text{ineq}} \quad \text{for all } \theta \in [0, 1].$$

Hence, $\mathcal{F}_{\text{ineq}}$ is convex. □

The overall feasible set of the QP problem, $\mathcal{F}$, is the intersection of the feasible sets for equality and inequality constraints:

$$\mathcal{F} = \mathcal{F}_{\text{eq}} \cap \mathcal{F}_{\text{ineq}}. \tag{1.19}$$

A QP problem having a disjointed constraints set $\mathcal{F} = \emptyset$ is called infeasible. The infeasibility arises if the constraints are mutually inconsistent, such that no solution x satisfies all constraints simultaneously. Only feasible QPs are investigated in this thesis. Depending on $\mathcal{F} = \emptyset$, the problem may be unbounded, leading to no local nor global solution.

To assess the regularity and local structure of the constraint set near feasible points, additional constraints qualifications are commonly introduced. These conditions impose regularity properties on the description of the feasible set, ensuring that its local behavior can be accurately captured by first-order approximations of the defining constraint functions. In the context of quadratic programming, such qualifications are standard for classical primal methods with convergence guarantees. In this research, however, we introduce a family of regularized formulations that fundamentally relax these requirements, allowing robust analysis and computation without that the Linear Independence Constraint Qualification (LICQ) holds [118], see Definition 1.3.1 and the Slater condition (1.2.9).

Definition 1.3.1 — **LICQ**

The Linear Independence Constraint Qualification (LICQ) requires that the gradients of all active constraints at a feasible point be linearly independent. In the case of linear constraints, this condition reduces to linear independence of the rows corresponding to the active inequalities and all equalities.

1.3.3 Convexity of QPs

Convexity is a decisive property in optimization, and it influences the solvability of a QP problem. The convexity determines whether a local minimum is also a global minimum, significantly affecting the computational methods required to solve the problem, see Theorem (1.2.5).

Definition 1.3.2 — **Positive Definite Matrix**

A symmetric matrix $Q \in \mathbb{R}^{n\times n}$ is *positive definite* if

$$x^\top Qx > 0 \quad \forall x \in \mathbb{R}^n \setminus \{0\}.$$

We denote it $Q \succ 0$.

Moreover, all eigenvalues of $Q \succ 0$ are strictly positive, and the quadratic form $x^\top Qx$ defines a strictly convex function. Positive definiteness implies that the matrix is invertible and that the optimization cost

$$\min_x \frac{1}{2}x^\top Qx + c^\top x$$

has a unique minimizer when unconstrained, while holding a strong duality.

Definition 1.3.3 — **Positive Semidefinite Matrix**

A symmetric matrix $Q \in \mathbb{R}^{n\times n}$ is *positive semidefinite* if

$$x^\top Qx \geq 0 \quad \forall x \in \mathbb{R}^n.$$

We denote it $Q \succeq 0$.

Equivalently, all eigenvalues of $Q \succeq 0$ are non-negative. The associated quadratic form $x^\top Qx$ is convex but non-strictly convex. There may exist directions $x \neq 0$ such that $x^\top Qx = 0$, which corresponds to flat directions in the function surface.

Remark 1.3.1 nonconvex QP

The QP problem becomes non-convex if Q is not at least positive semi-definite. Such QP problems are generally more difficult to solve due to the potential presence of multiple local minima. Specialized algorithms are required to find a suitable solution. These considerations go further than the scope of this chapter and will be introduced in chapter 3.

1.3.4 Duality of QPs

The Lagrangian formulation resulting from Problem (1.3.2) enforces the linear equality and inequality constraints,

$$\mathcal{L}(x, s, \theta_\lambda, \theta_\mu, \theta_s) := \frac{1}{2} x^\top Q x + c^\top x + \theta_\lambda^\top (Ax - b) + \theta_\mu^\top (Gx - d + s) - \theta_s^\top s, \tag{1.20}$$

where $\theta_\lambda \in \mathbb{R}^m$ and $\theta_\mu \in \mathbb{R}^p$, $\theta_s \in \mathbb{R}^p$ are the Lagrange multipliers (also called dual variables) associated with the equality constraints, inequality constraints and slack variable positivity, respectively.

Taking first-order stationarity of the Lagrangian with respect to the slack variable s yields

$$\begin{aligned} & \nabla_s \mathcal{L} := 0 \quad \Rightarrow \theta_\mu - \theta_s := 0 \\ \Leftrightarrow \quad & \theta_\mu = \theta_s. \end{aligned} \tag{1.21}$$

Substituting the relation $\theta_\mu = \theta_s$ into the stationarity condition with respect to the primal variable x, we obtain

$$\begin{aligned} & \nabla_x \mathcal{L} := 0 \quad \Rightarrow Qx + c + A^\top \theta_\lambda + G^\top \theta_\mu := 0 \\ \Leftrightarrow \quad & c = -Qx - \theta_\lambda^\top A - \theta_\mu^\top G. \end{aligned} \tag{1.22}$$

We now write the dual problem $q(\theta_\lambda, \theta_\mu)$

$$q(\theta_\lambda, \theta_\mu) = \max_{\theta_\lambda, \theta_\mu \geq 0} \inf_x \left\{ \frac{1}{2} x^\top Q x + c^\top x + \theta_\lambda^\top (Ax - b) + \theta_\mu^\top (Gx - d) \right\}, \tag{1.23}$$

see (1.4) for its formulation in the convex case

Under the stationarity conditions, the dual Lagrangian is rewritten using the expression of c,

$$\begin{aligned}\mathcal{L}_D(x,\theta_\lambda,\theta_\mu) &= \frac{1}{2}x^\top Qx + c^{*\top}(x,\theta_\lambda,\theta_\mu)x + \theta_\lambda^\top(Ax-b) + \theta_\mu^\top(Gx-d)\\ &= \frac{1}{2}x^\top Qx + x^\top\left(-Qx-\theta_\lambda^\top A-\theta_\mu^\top G\right) + \theta_\lambda^\top(Ax-b) + \theta_\mu^\top(Gx-d)\\ &= -\frac{1}{2}x^\top Qx - b^\top\theta_\lambda - d^\top\theta_\mu,\end{aligned} \tag{1.24}$$

leading to the resulting *Wolfe* dual problem:

Problem 1.3.3 — **Wolfe Dual QP**

$$\begin{array}{ll}\max\limits_{\theta_\lambda,\theta_\mu} & -\frac{1}{2}x^\top Qx - b^\top\theta_\lambda - d^\top\theta_\mu\\ \text{s.t.} & Qx + c + A^\top\theta_\lambda + G^\top\theta_\mu = 0,\end{array} \quad \text{with :} \quad \begin{array}{l}x\in\mathbb{R}^n,\\ Q\in\mathbb{S}_+^n,\\ A\in\mathbb{R}^{m\times n},\\ G\in\mathbb{R}^{p\times n},\\ c\in\mathbb{R}^n,\\ b\in\mathbb{R}^m,\\ d\in\mathbb{R}^p.\end{array} \tag{1.25}$$

which is concave since the dual function is the pointwise infimum of a family of affine functions of $(\theta_\lambda,\theta_\mu)$, see [120][chapter 5], [10]. Therefore, the Wolfe dual Problem (1.3.3) provides a powerful alternative formulation that is a concave QP maximization, i.e. equivalent to a convex QP minimization.

We consider the convex QP (P)

$$\min_{x\in\mathbb{R}^n}\ \frac{1}{2}x^\top Qx + c^\top x \quad \text{s.t.} \quad Gx\le d,\ Ax=b,$$

with $Q\succeq 0$, and its associated dual problem (D) with Lagrange multipliers $\theta_\mu\ge 0$ for the inequalities and θ_λ for the equalities:

$$q(\theta_\mu,\theta_\lambda) = \inf_{x\in\mathbb{R}^n}\left\{\frac{1}{2}x^\top Qx + c^\top x + \theta_\mu^\top(Gx-d) + \theta_\lambda^\top(Ax-b)\right\}.$$

Weak Duality: For any primal feasible $\bar{x}$ and any dual feasible $(\bar{\theta_\mu},\bar{\theta_\lambda})$ with $\bar{\theta_\mu}\ge 0$,

$$q(\bar{\theta_\mu},\bar{\theta_\lambda}) \le \frac{1}{2}\bar{x}^\top Q\bar{x} + c^\top\bar{x}.$$

Thus, the dual objective provides a lower bound on the primal objective.

Strong Duality: If the primal problem is convex and satisfies Slater's condition (1.2.9) (strict feasibility of $Gx<d$ and $Ax=b$), then there exist dual optimal multipliers $(\theta_\mu^*\ge 0,\theta_\lambda^*)$ such that

$$q(\theta_\mu^*,\theta_\lambda^*) = \frac{1}{2}x^{*\top}Qx^* + c^\top x^*,$$

where x^* is primal optimal. In this case, the KKT conditions hold, and the primal and dual optimal values coincide.

1.3.5 Existence of the solution for convex QPs

The existence of solutions for Equality Constrained Quadratic Programs (ECQP) has been well established in the literature, see [61, 74]. When linear inequality constraints are introduced, additional conditions are required to ensure the existence of a solution. We recall a few fundamental concepts from set theory that play a central role in optimization theory. In particular, the notions of closedness, boundedness, and compactness describe the geometric properties of the feasible set that ensure minimizers exist.

Definition 1.3.4 **Closed Set**

A set $S \subseteq \mathbb{R}^n$ is called ***closed*** if it contains all its limit points. That is, if a sequence $\{x_k\} \subset S$ converges to x, then $x \in S$.

Definition 1.3.5 **Bounded Set**

A set $S \subseteq \mathbb{R}^n$ is called ***bounded*** if there exists a constant $M > 0$ such that $\|x\| \leq M$ for all $x \in S$.

Definition 1.3.6 **Compact Set**

A set $S \subseteq \mathbb{R}^n$ is called ***compact*** if it is both closed and bounded. Equivalently, every sequence $\{x_k\} \subset S$ has a subsequence that converges to a point in S.

By the Weierstrass extreme value theorem, any continuous function achieves its minimum over a compact set. Therefore, if the feasible set of a QP problem is compact (i.e., closed and bounded in the sense of Definitions 1.3.4-1.3.6) and the objective function is convex, the existence of at least one minimizer is guaranteed.

Theorem 1.3.3 **Existence and uniqueness of a solution for convex QP**

Let the Quadratic Program be given by

$$\min_{x \in \mathcal{F}} \; f(x) = \tfrac{1}{2} x^\top Q x + c^\top x, \tag{1.26}$$

where $Q \in \mathbb{R}^{n \times n}$ is symmetric and at least positive semi-definite, and the feasible set

$$\mathcal{F} = \{x \in \mathbb{R}^n \mid Ax = b, \; Gx \leq d\}$$

is nonempty and closed.

If Q is positive semidefinite and $\mathcal{F}$ is compact, there exists at least one global minimizer $x^* \in \mathcal{F}$.
If Q is positive definite, then the minimizer x^* is unique, see [107, 120].

In other words, convexity of both the objective function and the feasible set, together with the closedness and boundedness properties introduced in Definitions 1.3.4–1.3.6, is sufficient to guarantee the existence of an optimal solution to the Quadratic Program. If the objective function is strictly convex (i.e., $Q > 0$), the solution is unique. In the case of a non-strictly

convex objective function (i.e., $Q \succeq 0$), at least one but possibly multiple minimizers exist.

1.3.6 Optimality of QPs

The optimality of a candidate solution can be assessed through a set of well-defined metrics. These measures, typically derived from first- and second-order derivatives, quantify how closely the current point satisfies stationarity, feasibility, and complementarity requirements.

For a convex quadratic program, where $Q \in \mathbb{R}^{n\times n}$ with $Q \succeq 0$ and the constraints are linear, a constraint qualification is automatically satisfied at every feasible point, see [49][Chap. 5 page 277]. Consequently, the Karush-Kuhn-Tucker (KKT) conditions apply, and it is not necessary to consider the more general Fritz-John conditions. In this case, the KKT conditions are both necessary and sufficient for optimality. A candidate solution $x^* \in \mathbb{R}^n$ is optimal if and only if there exist Lagrange multipliers $\theta^*_\lambda \in \mathbb{R}^m$ and $\theta^*_\mu \in \mathbb{R}^p$ such that the following conditions hold,

$$
\begin{aligned}
&Qx^* + c + A^\top \theta^*_\lambda + G^\top \theta^*_\mu = 0, && \text{(Stationarity)} && \text{(1.27a)}\\
&Ax^* = b, && \text{(Primal feasibility - equalities)} && \text{(1.27b)}\\
&Gx^* \leq d, && \text{(Primal feasibility - inequalities)} && \text{(1.27c)}\\
&\theta^*_\mu \geq 0, && \text{(Dual feasibility)} && \text{(1.27d)}\\
&\theta^*_{\mu,i} \cdot (Gx^* - d)_i = 0, \quad i = 1, \ldots, p, && \text{(Complementarity).} && \text{(1.27e)}
\end{aligned}
$$

Under the stationarity condition (1.27a) the x-derivative ($\nabla_x(.)$) of the Lagrangian is zero at the optimum. The objective's gradient is exactly balanced by the gradients of the constraints weighted by their corresponding Lagrange multipliers. The optimality condition (1.27b) enforces the linear-equality constraints feasibility of a candidate solution, and equivalently for the linear-inequality constraints condition (1.27c). The dual feasibility (1.27d) (non-negativity of θ^*_μ) reflects that relaxing a satisfied constraint should not improve the objective; thus, the corresponding multipliers must remain non-negative. For each inequality constraint, the complementarity condition (1.27e) is satisfied at the optimum when the $i-th$ constraint is active ($G_i \cdot x - d_i = 0$), or the $i-th$ constraint is inactive and its corresponding Lagrange multiplier is zero. Thus, no inequality constraint contributes to the Lagrangian unless it is active.

Under the slack reformulation of the original QP problem, the complementarity condition (1.27e) can be equivalently rewritten,

$$
\theta^*_{\mu,i} \cdot s^*_i = 0, \quad \forall i = 1, \ldots, p, \quad \text{(Complementarity slackness).} \tag{1.28}
$$

The complementarity slackness (1.28) implies for each index i, that either the i-th inequality constraint is inactive ($s^*_i > 0 \Rightarrow \theta_\mu, i^* = 0$), or the constraint is active ($s^*_i = 0 \Rightarrow \theta_\mu, i^* \geq 0$), not simultaneously.

1.4 Algorithmic

In this section, we translate the theoretical foundations of convex optimization and convex quadratic programming into the algorithmic framework used throughout this work. Building on optimality conditions and duality theory, we introduce the numerical methods that form the core of the approach presented here. We begin with first-order methods as a brief reminder of iterative schemes, and then proceed to second-order Newton-type algorithms, emphasizing the role of regularization and merit functions in ensuring robustness. Key components such as linesearch strategies, termination criteria tailored to QPs, and preconditioning techniques are introduced.

1.4.1 Merit function

As presented in the previous section, showing the existence of the solution for Quadratic programs can be tedious, often requiring intricate analysis of the feasible set. Another strategy to circumvent the complexity of proving existence directly is to reformulate the problem into an equivalent but easier-to-solve problem. One such approach involves the use of regularizations [42, 51], by adding penalty terms to the original Lagrangian to make it easier to solve and analyze. In particular, regularized problems are transformed into equivalent unconstrained problems, where the objective and constraint violations are penalized simultaneously. In this framework, the (regularized) Lagrangian $\mathcal{L}$ constitutes a natural starting point for the construction of meaningful merit functions, denoted throughout this document by ϕ and equivalently $\mathcal{M}$ when regularized.

The Augmented Lagrangian Methods (ALM) penalize infeasibility by introducing a quadratic Euclidean measure of constraint violation, scaled by penalty weights [19, 109]. This penalty term enlarges the classical Lagrangian by adding components that grows quadratically with the degree of constraint violation, thus penalizing infeasible solutions during the optimization process. Using an iterative ALM-weight evolution framework, the constrained problem is effectively reformulated into a serie of subproblems, each progressively enforcing feasibility more strongly. The Euclidean second norm squared provides a smooth measure of unfeasibility, being compatible with gradient-based and Newton-type methods commonly used to solve these subproblems.

The **proximal** regularizations [109] introduce a penalty based on the primal variables' evolution by penalizing their deviations' velocity and reinforcing the convexity structure of the problem. An Euclidean measure qualifies the distance between successive iterates and a reference point (an estimator). Proximal methods promote stability and robustness, particularly for ill-conditioned or non-strictly convex problems. The proximal penalty assesses how far a given iterate is from satisfying optimality conditions (highlighted in the next § 1.3.6). The proximal reformulation of the Lagrangian is based on a sequence of regularized subproblems, each of which is better conditioned and easier to solve.

Gill and Robinson introduced a **primal-dual Augmented Lagrangian Method (ALM)** regularization [53], which extends the classical ALM framework by incorporating both primal and dual variables into the regularization process. Unlike the standard ALM, which primarily penalizes primal infeasibility through a Euclidean norm of the constraint residuals, the primal-dual ALM also directly regularizes the behavior of the dual variables by introducing additional terms into the augmented Lagrangian that simultaneously penalize deviations

in both the primal and dual variables from their expected optimal behavior using estimators. This dual regularization improves the robustness of the method, especially in situations where strong duality does not immediately hold or when the constraint qualifications are weak.

The combination of the introduced regularizations ensures the efficient solving of degenerated and ill-conditioned QP problems, and the Lagrangian formulation are carefully studied in § 1.5.4.

1.4.2 First order methods

Optimization algorithms are iterative methods that begin with an initial guess for the primal and dual variable $x, s, \theta_\lambda, \theta_\mu$ and generate a sequence of improved iterates until they terminate, ideally at a solution. Gradient-based optimization algorithms use the values of the objective function and constraints' first derivatives (gradient) to guide the search for an optimal solution. The strategy to update each iterate involves computing these gradients and adjusting the current iterate in the direction that reduces the objective function while staying feasible. Effective gradient-based algorithms can show three main qualities: robustness, efficiency, and accuracy. Robustness ensures that the algorithm performs well across various problems. Efficiency guarantees that computational time and data storage waste are avoided. Accuracy means that the algorithm can identify solutions with precision even with numerical errors.

Algorithm 1 presents a gradient-descent framework built around the merit function $\phi(x, \theta_\lambda, \theta_\mu)$. At each iteration, the concatenated vector of primal and dual variables $z = [x, \theta_\lambda, \theta_\mu]$ is updated by moving along the negative gradient of this merit function. The step size α_k is chosen to ensure a sufficient decrease of ϕ, thereby guaranteeing progress toward a stationary point that corresponds to a solution of the underlying quadratic program.

Algorithm 1 Gradient-descent iterative optimization framework.

1: **Input:** Initial guess $z_0 = [x_0, \theta_{\lambda_0}, \theta_{\mu_0}]$, tolerance ϵ
2: **while** ϵ-Termination not satisfied **do**
3: Compute $d_k = -\nabla(\phi(z_k))$
4: Find α_k
5: Update iterate: $z_{k+1} = z_k + \alpha_k \cdot d_k$
6: **end while**
7: **Output:** Approximate solution z^*

The strategy to choose α_k is presented in § 1.4.4 and the ϵ-termination structure is presented later in § 1.4.5.

1.4.3 Newton Schemes

Newton methods are powerful second-order optimization techniques that make use of both gradient and curvature information of the objective function. For convex quadratic programming (QP) problems, Newton-based methods are widely used due to their ability to achieve fast (typically quadratic) convergence near the optimal solution. These methods solve a sequence of linearized systems derived from the Karush-Kuhn-Tucker (KKT) conditions to find a primal-dual solution.

Algorithm (2) shows the basic Newton methods steps for continuous optimization. The goal is to iteratively solve the nonlinear system $\nabla(\phi)(z) = 0$, and we denote H the Hessian of the merit function $\phi(z)$ with respect to the primal-dual variables $z = [x, \theta_\lambda, \theta_\mu]$. The merit function encodes both optimality and feasibility, so its stationary points coincide with solutions of the underlying convex QP.

Algorithm 2 Usual Newton-based optimization algorithm for convex QPs.

Input: Initial guess $z_0 = [x_0, s_0, \theta_{\lambda 0}, \theta_{\mu 0}]$, tolerance ϵ
while ϵ-Termination not satisfied **do**
  Compute $R = \nabla(z_k)$
  Compute $H = \nabla(\nabla(\phi(z_k)))$
  Solve $H_k \cdot \Delta z_k = -R_k$
  Estimate line step α_k
  Update iterate: $z_{k+1} = z_k + \alpha_k \Delta z_k$
end while
Output: Approximate solution z^*

The algorithm begins with an initial guess for the primal and dual variables, denoted $z_0 = [x_0, s_0, \theta_{\lambda 0}, \theta_{\mu 0}]$, along with a termination tolerance ϵ. At each iteration, the residual vector R_k is computed to measure the violation of the optimality conditions. Then, the Jacobian of this residual, i.e., the Hessian H_k is evaluated. Solving the linear system $H_k \cdot \Delta z_k = -R_k$ provides the Newton direction Δz_k, which is the search direction for both primal and dual updates. A step size α_k is then chosen, possibly using a linesearch or a barrier-based strategy, to ensure that the updated iterate remains feasible and reduces the residual sufficiently. The new iterate is computed as $z^{k+1} = z^k + \alpha_k \Delta z_k$. This process repeats until the residual norm falls below the specified tolerance, indicating convergence to an approximate solution.

1.4.4 Linesearch

The purpose of the linesearch in the inner loop is to determine a suitable step length $0 < \alpha \leq 1$ along the Newton search direction Δ. While the Newton direction provides a locally optimal update, taking the full step ($\alpha = 1$) can lead to infeasibility, or insufficient reduction of the merit function $\phi(z)$, see [96]. The linesearch balances progress by sufficiently decreasing the cost and feasibility. When using proximal or augmented Lagrangian regularization the regularized Hessian H of $\phi(z)$ is positive definite, allowing the Newton direction to point downhill. The linesearch then finds the step length to achieve a controlled, stable descent ensuring the convergence of the regularized primal-dual method while exploiting the fast local convergence of Newton's method.

Armijo condition: A step size α is said to satisfy the *Armijo condition* if

$$\phi(z + \alpha \Delta z) \leq \phi(z) + c\alpha \nabla \phi(z)^\top \Delta z, \quad 0 < c < 1. \tag{1.29}$$

This condition guarantees a sufficient decrease of the merit function at each iteration.

Theorem 1.4.1 **Convergence of Newton-Armijo method**

Suppose that Algorithm (2) does not terminate. Let Δz be a descent direction of $\phi(z)$. Let z^* be an accumulation point of the sequence $\{z^k\}_{k\in\mathbb{N}}$ generated by Algorithm (2). Let $\phi : \mathbb{R}^n \to \mathbb{R}$ be twice continuous, differentiable and bounded below. Then it holds

$$\|\nabla\phi(z^*)\|_2 = 0.$$

Proof 1.4.1 **Convergence of Newton-Armijo method**

Assume that $\{z_{k_j}\}_{j\in\mathbb{N}}$ is a subsequence with

$$z_{k_j} \to z^* \quad \text{and} \quad \|\nabla\phi(z^*)\| \neq 0. \tag{1.30}$$

By construction, the sequence $\{\phi(z_{k_j})\}_{j\in\mathbb{N}}$ is monotonically non-increasing, and by continuity of ϕ and $\nabla\phi$ it holds

$$\begin{aligned} &\phi(z_{k_j}) \to \phi(z^*), \\ &\Delta_{k_j} = -H_{k_j}^{-1}\nabla\phi(z_{k_j}) \to -H_{k_j}^{-1}\nabla\phi(z^*) =: \Delta_{k_j}^*, \quad \text{with } H \text{ positive definite.} \end{aligned} \tag{1.31}$$

The Armijo condition at Newton step reads

$$\phi(z_{k_{j_\ell}} + \alpha^{k_{j_\ell}}\Delta_k) \leq \phi(z_{k_{j_\ell}}) + c\alpha^{k_{j_\ell}} v_k^\top \Delta_k = \phi(z_{k_{j_\ell}}) - c\alpha^{k_{j_\ell}}\|v_k\|^2_{H_k^{-1}}, \tag{1.32}$$

with j_ℓ the step where the Armijo condition is accepted, $v_{k_{j_\ell}} := \nabla\phi(z_{k_{j_\ell}})$ and the Newton direction

$$\Delta_{k_{j_\ell}} = -H_{k_{j_\ell}}^{-1} v_{k_{j_\ell}}, \quad \text{with } H_{k_{j_\ell}}, \Delta_{k_{j_\ell}}. \tag{1.33}$$

We rewrite (1.32),

$$0 \leq c\alpha^{k_j}\|v_{k_j}\|^2_{H_{k_j}^{-1}} \leq \phi(z_{k_j}) - \phi(z_{k_{j+1}}). \tag{1.34}$$

As

$$\phi(z_{k_j}) - \phi(z_{k_{j+1}}) \to 0,$$

two cases appear.

Case 1: $\|v_{k_j}\|^2_{H_{k_j}^{-1}} = 0$ and by definition Theorem (1.4.1) holds.

Case 2: $\alpha^{k_{j_\ell}} = 0$, and by rewriting (1.32) at backtracking step j_ℓ, before Armijo (1.32) holds, we have:

$$\phi(z_{k_j} + \frac{\alpha^{k_j}}{\beta}\Delta_{k_j}) > \phi(z_{k_j}) - c\frac{\alpha^{k_j}}{\beta}\|v_{k_j}\|^2_{H_{k_j}^{-1}}, \tag{1.35}$$

with the backtracking rule $\alpha^{k_{j+1}} = \beta\alpha^{k_j}, \quad 0 < \beta < 1.$

By rewriting (1.35) we get:

Convergence of Newton-Armijo method

$$\frac{\phi(z_{k_j} + \frac{\alpha^{k_j}}{\beta}\Delta_{k_j}) - \phi(z_{k_j})}{\frac{\alpha^{k_j}}{\beta}} > -c\|v_{k_j}\|^2_{H^{-1}_{k_j}}. \tag{1.36}$$

As $\frac{\alpha^{k_j}}{\beta} \xrightarrow{j} 0$, we have

$$\frac{\phi(z_{k_j} + \frac{\alpha^{k_j}}{\beta}\Delta_{k_j}) - \phi(z_{k_j})}{\frac{\alpha^{k_j}}{\beta}} = \Delta_{k_j} v_{k_j} = -\|v_{k_j}\|^2_{H^{-1}_{k_j}}, \quad \text{see (1.33)}, \tag{1.37}$$

and as

$$\|v_{k_j}\|^2_{H^{-1}_{k_j}} \xrightarrow{k_j} \|v^*_{k_j}\|^2_{H^{-1*}_{k_j}}, \tag{1.38}$$

the inequality reads,

$$\begin{aligned} & -\|v_{k_j}\|^2_{H^{-1}_{k_j}} \geq -c\|v_{k_j}\|^2_{H^{-1}_{k_j}}, \\ \Leftrightarrow \quad & 0 \leq \|v_{k_j}\|^2_{H^{-1}_{k_j}} \leq c\|v_{k_j}\|^2_{H^{-1}_{k_j}}, \quad \text{with: } 0 < c < 1, \\ \Rightarrow \quad & v^* = 0. \qquad \square. \end{aligned} \tag{1.39}$$

Non-monotone linesearch: While the classical Armijo rule enforces monotonic decrease of ϕ, this can be overly conservative in ill-conditioned or nearly singular regimes. A non-monotone linesearch relaxes the strict descent requirement by allowing temporary increases in the merit function, provided that an overall trend of decrease is maintained. A typical criterion is given by

$$\phi(z_{k+1}) \leq \max_{j \in [k-M,k]} \phi(z_j) + c\alpha_k \nabla\phi(z_k)^\top \Delta z_k, \tag{1.40}$$

where $M \geq 0$ controls the memory length of past merit values. This approach improves practical convergence when the regularized Newton direction provides locally accurate but globally aggressive updates, see [63].

Adding monotonicity, barrier coupling ν: To further enhance robustness, monotonicity can be adaptively restored using a strategy inspired by Li-Fukushima's work, see [84]. The acceptance threshold of the Armijo rule is linked to the barrier parameter ν, with $\nu_k \to 0$. Specifically, when ν is large (far from the central path), non-monotone steps are permitted to accelerate progress. As ν decreases monotonously, the criterion becomes stricter, enforcing monotonic decrease of the merit function $\phi(z)$. This can be expressed as

$$\phi(z_{k+1}) \leq \phi(z_k) + \alpha_k \nabla\phi(z_k)^\top \Delta z_k + \eta(\nu_k), \tag{1.41}$$

where $\eta(\nu_k) \in (0,1)$ is a decreasing function of ν_k, such as $\eta(\nu_k) = c_{f1} + c_{f2}\nu_k$ with $c_{f1}, c_{f2} > 0$. This coupling preserves fast convergence during early iterations and guarantees global convergence as the barrier parameter tends to zero.

Overall, this adaptive line-search framework balances global convergence guarantees of the Armijo condition with the quadratic convergence of the Newton descent.

1.4.5 Termination

As introduced in § 1.3.6, where the optimality conditions were presented, one can monitor how closely a candidate solution $[x, \theta_\lambda, \theta_\mu]$ satisfies the original QP problem. Based on these optimality conditions, a termination procedure can be defined. We recall the optimality conditions,

$$0 = Qx + q + A^\top \theta_\lambda + G^\top \theta_\mu, \quad 0 = \min\{s, \theta_\mu\}, \quad 0 = Ax - b, \quad 0 = Gx - d + s. \tag{1.42}$$

From these conditions, three termination criteria can be derived, see [83] for details.

Primal residual: the primal residual measures the violation of the problem's equality and inequality constraints, see (1.27b), at iteration k (1.27c),

$$\begin{aligned} r_p &:= \max\left[\|Ax^{(k)} - b\|_\infty,\ [Gx^{(k)} - d]_+\right], \\ r_p^{rel} &:= \max\left[\|Ax^{(k)}\|_\infty, \|b\|_\infty, \|Gx^{(k)}\|_\infty, \|d\|_\infty\right], \end{aligned} \tag{1.43}$$

where $[\xi]_+ = \max(\xi, 0)$ is applied componentwise. A solver declares primal feasibility when

$$r_p \leq \epsilon + r_p^{rel} \cdot \epsilon_{\text{rel}}, \tag{1.44}$$

where ϵ is an absolute tolerance and ϵ_{rel} is a relative tolerance relaxing the primal residual depending on the problem scaling.

Dual residual: the dual residual quantifies the deviation from the stationarity condition of the KKT system, which corresponds to the optimality of the Lagrangian gradient with respect to the primal variables, at iteration k

$$\begin{aligned} r_d &:= \|Qx^{(k)} + c + A^\top \theta_\lambda^{(k)} + G^\top \theta_\lambda^{(k)}\|_\infty, \\ r_d^{rel} &:= \max\left[\|Qx^{(k)}\|_\infty, \|A^\top \theta_\lambda^{(k)}\|_\infty, \|G^\top \theta_\mu^{(k)}\|_\infty\right], \end{aligned} \tag{1.45}$$

where dual feasibility holds when

$$r_d \leq \epsilon + \epsilon_{\text{rel}} \cdot r_d^{rel}, \tag{1.46}$$

where ϵ is an absolute tolerance and ϵ_{rel} is a relative tolerance relaxing the primal residual depending on the problem scaling.

Duality gap: the duality gap reflects the difference between the primal and dual objective values and provides a certificate of suboptimality . For convex QPs, a zero gap guarantees global optimality.

$$\begin{aligned} r_g &:= |(x^{(k)})^\top Q x^{(k)} + q^\top x^{(k)} + b^\top \theta_\lambda^{(k)} + d^\top \theta_\mu^{(k)}|, \\ r_g^{rel} &:= \max\left[|(x^{(k)})^\top Q x^{(k)}|,\ |q^\top x^{(k)}|,\ |b^\top \theta_\lambda^{(k)}|,\ |d^\top \theta_\mu^{(k)}|\right], \end{aligned} \tag{1.47}$$

Termination via the gap test requires:

$$r_g \le \epsilon + \epsilon_{\text{rel}} \cdot r_g^{rel} \tag{1.48}$$

where ϵ is an absolute tolerance and ϵ_{rel} is a relative tolerance relaxing the primal residual depending on the problem scaling.

The algorithm terminates when the combined primal, dual, and duality-gap residuals satisfy

$$\|r_p + r_d + r_g\|_\infty \le \epsilon, \tag{1.49}$$

ensuring that the solution is sufficiently close to optimal.

1.4.6 Preconditioning

The convergence behavior and numerical stability are heavily influenced by the conditioning of the linear system that must be solved iteratively. A poor scaling of the problem data can lead to slow convergence and numerical issues. To mitigate this, a diagonal preconditioning (*Ruiz Equilibration*) scales the problem before applying Newton steps, see [110, 119]. The scaling matrices $D \in \mathbb{R}^{n\times n}$, $E \in \mathbb{R}^{m\times m}$, are both positive definite and diagonal. Their goal is to rescale the primal and dual variables, resulting in a transformed optimization problem. Define the scaled variables $\tilde{x} = D^{-1}x$ and $\tilde{z} = Ez$. The scaled problem becomes

$$\begin{aligned} \min_{\tilde{x}} \ & \frac{1}{2}\tilde{x}^\top (DQD)\tilde{x} + (Dq)^\top \tilde{x} \\ \text{subject to} \quad & E_\lambda A D \tilde{x} = b \\ & E_\mu G D \tilde{x} \le d. \end{aligned} \tag{1.50}$$

The motivation behind choosing D and E is to balance the magnitudes of the rows and columns of A, and to ensure that all variables have roughly the same numerical scale. This equalization improves the spectral properties of the matrices involved in the linear system, see [119]. To assess the termination criterion (1.49), the original problem data are reconstructed, and the primal and dual candidate solutions are back-preconditioned. The complete procedure is described in [110, 119].

1.5 Interior Point Methods

1.5.1 History and background

Interior Point Methods (IPMs) have undergone a remarkable evolution since their origins. The development of Interior Point Methods (IPMs) is rooted in the foundational work on linear programming by George Dantzig, who introduced the simplex method and formalized much of the early theory in his classic books [31, 32]. For decades, the simplex method was the dominant approach, although it is not polynomial in the worst case. A major theoretical breakthrough occurred in 1979 when Khachiyan introduced the first polynomial-time algorithm for linear programming using the ellipsoid method [78]. However, it was Karmarkar's work in 1984 that truly revolutionized the field with a new polynomial-time algorithm based on projective geometry [75]. This algorithm inspired a wealth of subsequent research exploring the connections between barrier functions and projective methods, as seen in the work by Gill et al. [52]. The development of primal-dual interior point methods soon followed. Kojima and collaborators introduced a primal-dual framework in the late 1980s [81], and this was later extended to handle infeasible starting points [80]. Megiddo contributed significantly to understanding the geometric interpretation of the central path and convergence properties [92], helping bridge theory and algorithmic design. Over time, IPMs were no longer confined to linear programming but generalized to handle quadratic, semidefinite, and nonlinear programming problems. An extensive overview of the development and refinement of Interior Point Methods (IPMs) over the past decades is provided by Gondzio in his retrospective articles [58, 59]. His survey serves as both a historical record and a technical reference, reflecting on the sustained relevance and adaptability of IPMs in areas such as quadratic programming, conic optimization, and beyond.

1.5.2 Enforcing inequality constraints

Slack variables transform inequality constraints $Gx \leq d$ into equality constraints $Gx - d + s = 0$, where $s > 0$. The slack transformation does not alter the solution space but do not enforce satisfaction of the inequality constraints at each iteration [26]. This reformulation of the problem ensures compatibility with numerical solvers and provides insights into constraint activity at the optimal solution, see [96] [24] [51]. The resulting Lagrangian of problem (1.3.2)

$$\begin{aligned}\mathcal{L}_\delta(x,s) \coloneqq &\frac{1}{2}x^\top Qx + c^\top x \\ &+ \theta_\lambda^\top (Ax - b) \\ &+ \theta_\mu^\top (Gx - d + s) \\ &+ \sigma\nu\phi(s),\end{aligned} \tag{1.51}$$

with $\phi(s) = -\sum_{i=0}^{p} \ln(s_i)$, ν the barrier parameter, and $\sigma \in (0, 1]$ is a centering parameter.

Comparing to the Lagrangian of problem (1.3.1), the inequality constraints are relaxed by a log barrier sum $\phi(s)$. Given an early-termination tolerance $\epsilon > 0$, with $\hat{\epsilon} > 0$ a minimum acceptable threshold, the barrier parameter is updated using the heuristic rule from `IPOPT`

[126, Eq. 7]: starting from $\nu_0 = 1$, at each outer iteration it is reduced by taking the minimum of a linear scaling term $\kappa_\nu \nu_k$ and a superlinear reduction $\nu_k^{\theta_\nu}$, where $\kappa_\nu \in (0,1)$ and $\theta_\nu \in (0,1)$, and is floored at $\hat{\epsilon}/10$ to prevent ν_k from becoming excessively small.

$$\nu_{k+1} = \max\{\hat{\epsilon}/10, \min\{\kappa_\nu \nu_k, \nu_k^{\theta_\nu}\}\}. \tag{1.52}$$

This formulation perturbs the first-order optimality conditions, see § 1.3.6. Instead of enforcing the complementarity condition $\theta_\mu^i \cdot s^i = 0$ exactly (as required at optimality by the KKT conditions), a relaxed version $(\theta_\mu^i \cdot s^i = \nu, \quad \nu > 0, \quad \nu \to 0)$ is used and gradually driven to zero during the optimization process. The barrier parameter ν is decreased at each optimization following [130]. Moreover, this relaxation removes the need to explicitly determine which inequality constraints are active or inactive and therefore avoids numerical misclassification of active/inactive constraints, leading to a destruction of the solution, especially for high-precision solutions.

As the algorithm progresses and $\nu \to 0$ [130], the partitioning of the slack variables into zero and nonzero components naturally emerges, thereby revealing the active set at the solution. This approach provides a robust and systematic means of handling inequality constraints without the combinatorial complexity of active-set guessing. However, this approach makes the parameter ν central to the method's convergence, which complicates the warm-start process. Empirical results in [62] indicate that interior-point methods are generally much faster for large problems. However, when a warm start is required, active-set methods may be preferable. Although significant research has been dedicated to improving the warm-start capabilities of interior-point methods [44], also using learning-based methods, the full potential of these techniques remains largely unexplored. In practice IPMs are well suited for large problems and converge in a number of iterations that is almost constant independently of the problem dimension [30].

Example 1.5.1 **Barrier function**

We illustrate the use of the log-barrier penalty for relaxing a simple inequality constrained optimization problem. Considering the following optimization problem,

$$\begin{aligned} \min_{x_1, x_2} \quad & f(x_1, x_2) = x_1^2 - 2x_1x_2 + 4x_2^2, \\ \text{s.t.} \quad & 0.1x_1 - x_2 \geq 1, \\ & x_1 + x_2 \geq 4. \end{aligned} \tag{1.53}$$

The barrier method introduces a barrier term to the objective function to enforce the constraints. The barrier-enforced cost function becomes,
$f_\nu(x_1, x_2) = f(x_1, x_2) - \nu\left(\ln(0.1 \cdot x_1 - x_2 - 1) + \ln(x_1 + x_2 - 4)\right)$,
where ν is a sequence of decreasing positive values.
As ν approaches zero, the barrier term weakens, allowing the optimization process to approach the boundary of the feasible region. In Figure 1.1 the barrier of example 1 is shown as we gradually decrease ν.

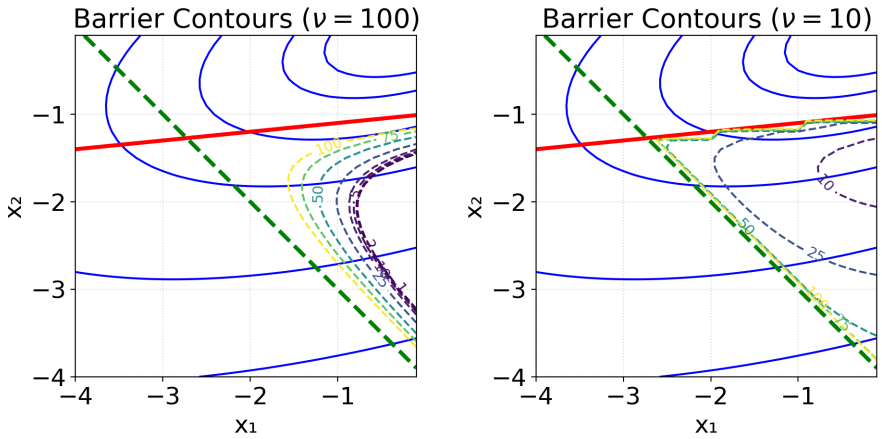


FIGURE 1.1: Barrier penalty depending on $\nu = 100$ (left) $\nu = 10$ (right).

1.5.3 Classical primal Interior Point

1.5.3.1 Formulation

Primal methods focus on minimizing the objective with respect to the primal variables x, while the dual variables $\theta_\lambda, \theta_\mu$ are updated along the optimization process according to an update rule. These methods work by iteratively improving the primal solution and adjusting the dual estimates to enforce the constraints, often leading to efficient convergence when applied to well-structured problems. In this section, the primal merit function is an augmented proximal Lagrangian formulated as,

$$\begin{aligned}\mathcal{L}_\delta(x, s; \theta) :=& \frac{1}{2} x^\top Q x + c^\top x \\ &+ \theta_\lambda^\top (Ax - b) + \frac{1}{2\delta_\lambda} \|Ax - b\|_2^2 \\ &+ \theta_\mu^\top (Gx - d + s) + \frac{1}{2\delta_\mu} \|Gx - d + s\|_2^2 \\ &+ \frac{\delta_x}{2} \|\theta_x - x\|_2^2 + \sigma \nu \phi(s),\end{aligned} \tag{1.54}$$

with $\theta = [\theta_\lambda, \theta_\mu, \theta_x]$ the Lagrangian multiplicators of the original Lagrangian (1.20) and the proximal estimator. The Lagrangian (1.54) is formulated using the cost function of problem (1.3.1), following by a constraints relaxation using Lagrangian-multiplicator $\theta_\lambda, \theta_\mu$ and an augmented Lagrangian regularization of penalty parameters $\delta_\lambda, \delta_\mu$. A proximal term of penalty parameters δ_x is added in order to handle non-strictly convex problems.

Let us consider the following general type of two-nested loops iterative method for solving the barrier Problem (1.51) using a Lagrangian merit function. The primal variables minimize the proximal augmented Lagrangian while the dual variables are updated such that,

$$
\begin{aligned}
x^+, s^+ &\approx \underset{x\in\mathbb{R}^n, s\in\mathbb{R}^p}{\arg\min}\ \mathcal{L}_\delta(x, s; \theta) \\
\theta_\lambda^+ &= \theta_\lambda + Ax - b \\
\theta_\mu^+ &= \theta_\mu + Gx - d + s \\
\theta_x^+ &= x.
\end{aligned}
\tag{1.55}
$$

The Lagrangian multiplicators $\theta_\mu, \theta_\lambda$ ensure the feasibility of the problem by regularizing the equality and inequality constraints. The proximal multiplicator θ_x increases the robustness and stability of the minimization process by penalizing a too great distance from previous solutions.

The augmented proximal Lagrangian's first derivatives in primal directions x and s are defined such as,

$$
\begin{aligned}
\nabla_x \mathcal{L}_\delta(x, s; \theta) = Qx + c + A^\top \theta_\lambda + \frac{1}{\delta_\lambda} A^\top (Ax - b) \\
+ G^\top \theta_\mu + \frac{1}{\delta_\mu} G^\top (Gx - d + s) + \delta_x (x - \theta_x),
\end{aligned}
\tag{1.56a}
$$

$$
\nabla_s \mathcal{L}_\delta(x, s; \theta) = \theta_\mu + \frac{1}{\delta_\mu}(Gx - d + s) - \sigma \nu s^{-1},
\tag{1.56b}
$$

and second derivatives as,

$$
\nabla^2_{x,x} \mathcal{L}_\delta(x, s; \theta) = Q + \frac{1}{\delta_\lambda} A^\top A + \frac{1}{\delta_\mu} G^\top G + \delta_x \mathbb{I}_n,
\tag{1.57a}
$$

$$
\nabla^2_{s,s} \mathcal{L}_\delta(x, s; \theta) = \frac{1}{\delta_\mu} \mathbb{I}_p + \sigma \nu s^{-2},
\tag{1.57b}
$$

$$
\nabla^2_{x,s} \mathcal{L}_\delta(x, s; \theta) = \frac{1}{\delta_\mu} G.
\tag{1.57c}
$$

The augmented proximal Lagrangian is a function of the primal variables $[x, s]$ and dual and proximal multiplicators $[\theta_\mu, \theta_\lambda, \theta_x]$. From this set of first and second derivatives, the associated primal Newton step can be derived as a two by two block-dense matrix following $\mathcal{H}_\delta(x, s; \theta)\Delta = -\nabla \mathcal{L}_\delta \cdot (x, s; \theta)$. The Hessian matrix $\mathcal{H}_\delta(x, s; \theta)$ is constructed using (1.57a), (1.57b), (1.57c) such that,

$$
\begin{bmatrix}
Q + \delta_x \mathbb{I}_n + \frac{1}{\delta_\lambda} A^\top A + \frac{1}{\delta_\mu} G^\top G & \frac{1}{\delta_\mu} G^\top \\
\frac{1}{\delta_\mu} G & \frac{1}{\delta_\mu} \mathbb{I}_p + \sigma \nu s^{-2}
\end{bmatrix}
\begin{bmatrix} \Delta x \\ \Delta s \end{bmatrix}
= -
\begin{bmatrix} \nabla_x \mathcal{L}_\delta(x, s; \theta) \\ \nabla_s \mathcal{L}_\delta(x, s; \theta) \end{bmatrix}.
\tag{1.58}
$$

As the linear system (1.58) would need to be solved multiple times in an optimization iterative process, its structure influences the whole solver performance. By introducing a sparse subsystem being bigger but easier to numerically inverse we introduce auxiliary variables $\mathfrak{u}$ and $\mathfrak{v}$. By substituting the auxiliary variables in the first and second block-lines of equations of system (1.58) we have

$$
\begin{aligned}
&(Q+\delta_x\mathbb{I}_n)\Delta x + A^\top\left(\frac{1}{\delta_\lambda}A\right)\Delta x + G^\top\left(\frac{1}{\delta_\mu}(G\Delta x+\Delta s)\right) = -\nabla_x\mathcal{L}_\delta(x,s;\theta),\\
&\frac{1}{\delta_\mu}G\Delta x + \frac{1}{\delta_\mu}\mathbb{I}_p + \sigma\nu s^{-2}\Delta s = -\nabla_s\mathcal{L}_\delta(x,s;\theta),\\
&\mathfrak{u} \triangleq \frac{A\Delta x}{\delta_\lambda},\\
&\mathfrak{v} \triangleq \frac{G\Delta x+\Delta s}{\delta_\mu}.
\end{aligned}
\tag{1.59}
$$

The Newton four-by-four block linear system can be constructed as follows,

$$
\begin{bmatrix}
Q+\delta_x\mathbb{I}_n & 0 & A^\top & G^\top\\
0 & \frac{\sigma\nu}{s^2} & 0 & \mathbb{I}_p\\
A & 0 & -\delta_\lambda\mathbb{I}_m & 0\\
G & \mathbb{I}_p & 0 & -\delta_\mu\mathbb{I}_p
\end{bmatrix}
\begin{bmatrix}\Delta x\\ \Delta s\\ \Delta\mathfrak{u}\\ \Delta\mathfrak{v}\end{bmatrix}
= -\begin{bmatrix}\nabla_x\mathcal{L}_\delta(x,s;\theta)\\ \nabla_s\mathcal{L}_\delta(x,s;\theta)\\ 0\\ 0\end{bmatrix}.
\tag{1.60}
$$

The linear system is primal-consistent, having block-lines 3 and 4 equal to zero, meaning that the descent direction minimizes the augmented proximal Lagrangian only according to variables x and s. The auxiliary variables $\mathfrak{u}$ and $\mathfrak{v}$ don't have any meaning so far in the development but will play a role later in the primal-dual study. Moreover, it can be observed that the term $\nabla^2_{s,s}\mathcal{L}_\delta$ contains a factor proportional to s^{-2}, which may destabilize the Newton iteration. In practice, the optimal slack variables s^* can become extremely large or extremely small, depending on the active constraints, thereby introducing severe scaling issues in the KKT system.

1.5.3.2 Algorithm

Algorithm 3 represents the classical single-loop formulation of a primal interior-point method, where primal and barrier updates occur within one iteration structure.

Algorithm 3 Classical primal Interior-Point Method

1: Initialize variables $(x, s, \theta_\lambda, \theta_\mu)$, and barrier parameter ν.
2: **while** $r_\mathrm{p} \le \epsilon$ and $r_\mathrm{d} \le \epsilon$ **do** {check termination}
3: Solve the primal KKT system for (x, s), $KKT.\Delta = -R$ (1.60)
4: Linesearch: find (x^k, s^k) such that $\left\|\nabla\mathcal{L}_k(x^k, s^k)\right\| \le \varepsilon_k$ and $s^k > 0$
5: Update the primal variables: $x \leftarrow x + \alpha\cdot\Delta x$, $s \leftarrow s + \alpha\cdot\Delta s$.
6: Update the Lagrangian and proximal multipliers: $\theta_\lambda, \theta_\mu, \theta_x$, see eq 1.55 .
7: Update the Lagrangian and proximal weights: $\delta_\lambda, \delta_\mu, \delta_x$.
8: Update the barrier parameter: ν.
9: **end while**
10: **Return** the optimal primal solution.

Step 3 poses the most computational requirements. The termination conditions (1.49) are checked at Step 2, before updating the incumbent Lagrangian multiplicators θ^{k+1} at Step 6. Of all steps, the choice of hyperparameters δ^k and ν_k at Steps 7,8 remains the most elusive, with minimal guidelines dictated by the theory.

The convergence of Algorithm 3 is guaranteed insofar as the regularization parameters δ^k remain positive and bounded, the barrier parameter ν_k vanishes from above, and the inner tolerance ε_k is summable.

The technical requirements and mild conditions for convergence are detailed, e.g., in [89] and investigated in §1.8.1. The convergence properties of Algorithm 3 can be analyzed with the lens of inexact proximal point algorithms; we refer the interested reader to [11, 15, 29, 34, 35, 68, 87, 103] for recent results established by exploiting this connection for regularized QP solvers.

The constraint violations, $(Ax - b)$ and $(Gx - d + s)$, steer the evolution of the multiplier updates $(\theta_\lambda^+, \theta_\mu^+)$ toward their optimal values $(\theta_\lambda^*, \theta_\mu^*)$ throughout the minimization process. For the primal variables, a linesearch method is used to find a safe gradient descent step-length α. In later developments, this scheme will be extended into an outer-inner loop framework, in which the outer loop updates the barrier parameter, estimators, and dual regularization terms, while the inner loop solves the corresponding KKT subsystem to convergence. The termination of the algorithm is governed by the evaluation of the primal and dual residuals, which are described in detail in § 1.4.5.

1.5.4 Classical primal-dual Interior Point

Unlike purely primal interior-point methods, primal-dual methods simultaneously handle both the primal variables (x, s) and the dual variables $(\theta_\lambda, \theta_\mu)$. The primal interior-point method solves a sequence of barrier-modified primal problems, focusing exclusively on the primal space and ensuring that iterates remain strictly within the feasible region. However, since the dual variables are not explicitly updated, convergence toward dual feasibility is generally weaker [128, § 3.3] for primal IPMs than primal-dual IPMs.

1.5.4.1 Formulation

The Primal-Dual IPM solves the primal Problem (1.3.1) and dual Problem (1.3.3) simultaneously by applying Newton's method to the perturbed Karush-Kuhn-Tucker conditions. This method follows the central path and ensures convergence to both primal and dual solutions [128]. The central path aspect of primal-dual IPMs will be investigated later in § 1.7.1. Primal-dual IPMs are more efficient in practice, particularly in large-scale settings, and supports infeasible starts [128]. Furthermore, they show, under suitable conditions, better-than-linear convergence rates [57].

A primal-dual merit function, denoted by $\mathcal{M}_\delta^{GR}$ (1.62), is constructed based on the previously defined primal Lagrangian (1.54). This formulation incorporates a primal-dual regularization term following the Gill-Robinson approach [54], leading to the following minimization problem,

$$\underset{x\in\mathbb{R}^n, s\in\mathbb{R}^p, \lambda\in\mathbb{R}^m, \mu\in\mathbb{R}^p}{\text{minimize}} \; \mathcal{M}_\delta^{GR}(x, s, \mu, \lambda; \theta_\mu, \theta_\lambda, \theta_x), \text{ with,} \tag{1.61}$$

$$
\begin{aligned}
\mathcal{M}_\delta^{GR} :=& \frac{1}{2}x^\top Qx + c^\top x \\
&+ \theta_\lambda^\top (Ax-b) + \frac{1}{2\delta_\lambda}\|Ax-b\|_2^2 \; + \theta_\mu^\top (Gx-d+s) + \frac{1}{2\delta_\mu}\|Gx-d+s\|_2^2 \\
&+ \frac{\delta_x}{2}\|\theta_x - x\|_2^2 + \sigma\phi_{\nu(s)} \\
&+ \frac{1}{2\delta_\lambda}\|Ax-b-\delta_\lambda(\hat{\theta}_\lambda-\theta_\lambda)\|_2^2 + \frac{1}{2\delta_\mu}\|Gx-d+s-\delta_\mu(\hat{\theta}_\mu-\theta_\mu)\|_2^2 \\
:=& \mathcal{L}_\delta(x,s;\theta_\mu,\theta_\lambda,\theta_x) + \frac{1}{2\delta_\lambda}\|Ax-b-\delta_\lambda(\hat{\theta}_\lambda-\theta_\lambda)\|_2^2 + \frac{1}{2\delta_\mu}\|Gx-d+s-\delta_\mu(\hat{\theta}_\mu-\theta_\mu)\|_2^2 \\
:=& \mathcal{L}_\delta(x,s;\theta_\mu,\theta_\lambda,\theta_x) + \mathcal{L}_\delta^{GR}(x,s,\mu,\lambda;\theta_\mu,\theta_\lambda).
\end{aligned}
\tag{1.62}
$$

The quantities $(\hat{\theta}_\lambda, \hat{\theta}_\mu)$ act as auxiliary estimators of the Lagrange multipliers $(\theta_\lambda, \theta_\mu)$, introduced to regulate and bound the evolution of the dual variables during the iterative process. For clarity of notation, we will henceforth denote $\hat{\theta}_\lambda$ by λ and $\hat{\theta}_\mu$ by μ, see remark below. Thus far, only the Gill–Robinson augmented Lagrangian $\mathcal{L}_\delta^{GR}$ includes the (μ,λ) terms. From this point onward, for numerical stability reasons, the dual variables are now (λ,μ). The primal-dual minimization process is then formulated as

$$
(x^+, s^+, \mu^+, \lambda^+) = \underset{x\in\mathbb{R}^n,, s\in\mathbb{R}^p,, \lambda\in\mathbb{R}^m,, \mu\in\mathbb{R}^p}{\arg\min} \mathcal{M}_\delta^{GR}(x,s,\mu,\lambda;\theta_\mu,\theta_\lambda,\theta_x). \tag{1.63}
$$

Remark 1.5.1 λ-μ-substitution

Used in an iterative minimization process, at convergence

$$
\begin{aligned}
& \nabla\mathcal{L}_\delta^{GR}(x^+,s^+,\mu^+,\lambda^+;\theta) = 0, && (1.64)\\
\Leftrightarrow \quad & Ax-b-\delta_\lambda(\lambda-\theta_\lambda) = 0, && (1.65)\\
\Leftrightarrow \quad & \lambda^* = \theta_\lambda + \frac{Ax^+-b}{\delta_\lambda}, && (1.66)\\
\Leftrightarrow \quad & Gx-d+s-\delta_\mu(\mu-\theta_\mu) = 0, && (1.67)\\
\Leftrightarrow \quad & \mu^* = \theta_\mu + \frac{Gx^+-d+s^+}{\delta_\mu}. && (1.68)
\end{aligned}
$$

Consequently, the auxiliary variables λ and μ can be interpreted as effective dual variables that guide the gradient-based minimization process. In this formulation, they serve both as approximations of the true Lagrange multipliers and as stabilizing terms that regulate the evolution of the dual iterates, ensuring consistent progression toward the primal-dual optimum. **Applying the λ-μ-substitutions causes the Gill–Robinson augmented Lagrangian to vanish.**

For clarity, the index $\square \in \{x,\lambda,\mu\}$ will be used throughout the remainder of this dissertation.

We derive $\mathcal{M}_\delta^{GR}(x,s,\lambda,\mu;\theta_\square)$ with respect to its arguments, obtaining the first-order derivatives,

$$\begin{aligned}
\nabla_x \mathcal{M}_\delta^{GR}(x,s,\lambda,\mu;\theta) =& Qx + c + A^\top\theta_\lambda + \frac{1}{\delta_\lambda}A^\top(Ax-b) + \frac{1}{\delta_\lambda}A^\top(Ax-b-\delta_\lambda(\lambda-\theta_\lambda)) \\
&+ G^\top\theta_\mu + \frac{1}{\delta_\mu}G^\top(Gx-d+s) + \frac{1}{\delta_\mu}G^\top(Gx-d+s-\delta_\mu(\mu-\theta_\mu)) \\
&+ \delta_x(x-\theta_x), && \text{(1.69a)} \\
\nabla_s \mathcal{M}_\delta^{GR}(x,s,\lambda,\mu;\theta) =& \theta_\mu + \frac{1}{\delta_\mu}(Gx-d+s) - \sigma\nu s^{-1} \\
&+ \frac{1}{\delta_\mu}(Gx-d+s-\delta_\mu(\mu-\theta_\mu)), && \text{(1.69b)} \\
\nabla_\lambda \mathcal{M}_\delta^{GR}(x,s,\lambda,\mu;\theta) =& -(Ax-b-\delta_\lambda(\lambda-\theta_\lambda)), && \text{(1.69c)} \\
\nabla_\mu \mathcal{M}_\delta^{GR}(x,s,\lambda,\mu;\theta) =& -(Gx-d+s-\delta_\mu(\mu-\theta_\mu)). && \text{(1.69d)}
\end{aligned}$$

The corresponding second-order derivatives are then expressed as

$$\begin{aligned}
\nabla^2_{x,x}\mathcal{M}_\delta^{GR}(x,s,\lambda,\mu;\theta) &= \nabla^2_{xx}\mathcal{L}_\delta(x,s;\theta) + \frac{1}{\delta_\lambda}A^\top A + \frac{1}{\delta_\mu}G^\top G, \\
&= Q + \delta_x\mathbb{I}_n + \frac{2}{\delta_\lambda}A^\top A + \frac{2}{\delta_\mu}G^\top G, && \text{(1.70a)} \\
\nabla^2_{s,s}\mathcal{M}_\delta^{GR}(x,s,\lambda,\mu;\theta) &= \nabla^2_{ss}\mathcal{L}_\delta(x,s;\theta) + \frac{1}{\delta_\mu}\mathbb{I}_p, \\
&= \frac{\sigma\nu}{s^2} + \frac{2}{\delta_\mu}\mathbb{I}_p && \text{(1.70b)} \\
\nabla^2_{\lambda,\lambda}\mathcal{M}_\delta^{GR}(x,s,\lambda,\mu;\theta) &= \delta_\lambda\mathbb{I}_m, && \text{(1.70c)} \\
\nabla^2_{\mu,\mu}\mathcal{M}_\delta^{GR}(x,s,\lambda,\mu;\theta) &= \delta_\mu\mathbb{I}_p, && \text{(1.70d)} \\
\nabla^2_{x,\lambda}\mathcal{M}_\delta^{GR}(x,s,\lambda,\mu;\theta) &= -A^\top, && \text{(1.70e)} \\
\nabla^2_{x,\mu}\mathcal{M}_\delta^{GR}(x,s,\lambda,\mu;\theta) &= -G^\top, && \text{(1.70f)} \\
\nabla^2_{x,s}\mathcal{M}_\delta^{GR}(x,s,\lambda,\mu;\theta) &= \frac{2}{\delta_\mu}G^\top. && \text{(1.70g)} \\
\nabla^2_{s,\lambda}\mathcal{M}_\delta^{GR}(x,s,\lambda,\mu;\theta) &= 0, && \text{(1.70h)} \\
\nabla^2_{s,\mu}\mathcal{M}_\delta^{GR}(x,s,\lambda,\mu;\theta) &= -\mathbb{I}_p. && \text{(1.70i)}
\end{aligned}$$

Based on the above derivatives, the Newton system can be constructed as follows, providing the linear approximation that drives the iterative primal-dual update,

$$\begin{bmatrix} Q+\delta_x\mathbb{I}_n+\frac{2}{\delta_\lambda}A^\top A+\frac{2}{\delta_\mu}G^\top G & -A^\top & -G^\top & \frac{2}{\delta_\mu}G^\top \\ -A & -\delta_\lambda^k\mathbb{I}_m & 0 & 0 \\ -G & 0 & -\delta_\mu^k\mathbb{I}_p & -\mathbb{I}_p \\ \frac{2}{\delta_\mu}G & 0 & -\mathbb{I}_{p\times p} & \frac{\sigma\nu}{s^2}+\frac{2\mathbb{I}_p}{\delta_\mu} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta\mu \\ \Delta s \end{bmatrix} = -\begin{bmatrix} \nabla_x \mathcal{M}_\delta^{GR}(x,s,\lambda,\mu;\theta_x) \\ \nabla_\lambda \mathcal{L}_\delta^{GR}(\lambda,\mu) \\ \nabla_\mu \mathcal{L}_\delta^{GR}(\lambda,\mu) \\ \nabla_s \mathcal{M}_\delta^{GR}(x,s,\lambda,\mu;\theta_x) \end{bmatrix}. \tag{1.71}$$

The primal Hessian associated with $\mathcal{L}\delta(x,s;\theta_\mu,\theta_\lambda,\theta_x)$ (cf. Eq. 1.60) exhibits significantly greater sparsity. This structural density increases computational complexity and time, particularly for large-scale problems. An equivalent, computationally efficient, and stabilized version of this Newton scheme is presented later in § 1.5.6. In this approach, a linear transformation is applied to improve the numerical properties of the KKT matrix. These modifications come at the expense of symmetry and no longer rely on the pure Hessian structure.

1.5.4.2 Algorithm

Algorithm 4 implements an outer-inner loop primal-dual interior-point method,

Algorithm 4 Outerloop-Innerloop primal-dual Interior-Point Method.

1: Initialize variables $z = [x_0^0, s_0^0, \lambda_0^0, \mu_0^0]^\top$, estimators $(\theta_{\lambda_0,\mu_0,x_0})$, proximal and augmented Lagrangian weight $(\delta_{\lambda_0,\mu_0,x_0})$ and barrier parameter ν_0.
2: **while** $r_{\text{prim}} \le \epsilon$ and $r_{\text{dual}} \le \epsilon$ **do** {Outer-termination (k)}
3: **while** $\left\|\nabla\mathcal{M}_\delta^{GR}(x_k^j, s_k^j, \lambda_k^j, \mu_k^j)\right\| \le \varepsilon_k \le \varepsilon$ **do** {Inner-termination (j)}
4: Solve the KKT system for $(x_k^j, s_k^j, \lambda_k^j, \mu_k^j, KKT.\Delta = -R$ (1.71)
5: Linesearch: find $(x_k^j, s_k^j, \lambda_k^j, \mu_k^j)$ such that $\left\|\nabla\mathcal{M}_\delta^{GR}(x_k^j, s_k^j, \lambda_k^j, \mu_k^j)\right\| \le \varepsilon_k$ and $s_k^j > 0$
6: Update the primal and dual variables: $z_k^{j+1} \leftarrow z_k^j + \alpha\cdot\Delta z$
7: **end while**
8: Update the estimators: $\theta_{\lambda_{k+1}}, \theta_{\mu_{k+1}}, \theta_{x_{k+1}}$, (1.55).
9: **If** $\rho_k > \eta_{cv}\rho_{k-1}$, update the weights $\delta_{\square,k+1}$ (1.73),
10: Update the barrier parameter: ν_{k+1}.
11: **end while**
12: **Return** the optimal primal-dual solution.

and have a nested outerloop-innerloop structure of indexes (k) and (j). In the outerloop the estimators $\theta_{\square_k}$, the proximal and augmented Lagrangian weights $\delta_{\square_k}$ and the barrier parameter ν_k are updated at Steps 8,9,10.

For the update of $\delta_{\square,k}$, let the constraint violation at iteration k be defined as

$$\rho_k := \|\max(0, Gx_k - h)\|_\infty + \|Ax_k - b\|_\infty, \tag{1.72}$$

where $Ax = b$ and $Gx \le h$ denote the equality and inequality constraints, respectively. Let $\eta_{cv} \in (0,1)$ be a relative constraint violation threshold. Specifically, if the relative decrease in constraint violation between outer iterations $k-1$ and k is seen as good enough, the next subproblem is deconvexified by applying the geometric update rule (1.73), resulting in $\delta_{k+1} < \delta_k$.

Indeed, the outerloop generates subproblems that are progressively less convexified, as the proximal and augmented Lagrangian weights $\delta_\square$ are updated non-monotonically across iterations (see §1.6 for the δ-dynamics analysis). To achieve superlinear convergence of the outer loop (see §1.8.1), the weights remain strictly positive while being reduced according to the update rule

$$\delta_{\square,k+1} = \max\left(\delta_{\min}, \alpha_{\square,k} \cdot \delta_{\square,k}\right), \qquad \square \in x, \lambda, \mu, \tag{1.73}$$

starting from some $\delta^0 = (\delta_{x,0}, \delta_{\lambda,0}, \delta_{\mu,0}) \in \mathbb{R}^3_{++}$, where the small threshold $\delta_{\min} > 0$ safeguards the numerical linear algebra for (1.81) and the decrease factors $\alpha_{\square,k} \in (0,1)$ are fixed.

In the subsequent innerloop, having fixed estimators, proximal and augmented Lagrangian weights, and barrier parameter, a subQP problem is solved until satisfying an early termination $\left\|\nabla \mathcal{M}_\delta^{GR}(x^k, s^k, \lambda^k, \mu^k)\right\| \le \varepsilon_k \le \varepsilon$ based on the gradient of the merit function. The early termination ε_k is updated at each outer-iteration using an amortized geometric rule, see § 1.4.5,1.7.5. By freezing the estimators, each iteration of the outer loop defines a convexified sub-QP problem in the primal-dual variables (x, s, λ, μ), which can be efficiently solved as a regularized quadratic program using the Newton system associated with $\mathcal{M}_\delta^{GR}$. The convexified KKT system is solved iteratively to obtain the search directions $(\Delta x, \Delta s, \Delta\lambda, \Delta\mu)$. A linesearch is applied to the primal and dual variables to guarantee sufficient decrease of the merit function and to maintain positivity of the slack variables s.

Upon convergence of the inner loop, i.e., when the gradient of the subproblem is sufficiently small, the outer loop updates the estimators $(\theta_\lambda, \theta_\mu, \theta_x)$ and the barrier parameter ν.

The inner tolerance ε evolves as

$$\varepsilon_{k+1} = \max\{\epsilon/10, \kappa_\varepsilon \alpha_\varepsilon^k \varepsilon_k\}, \tag{1.74}$$

with decay rate $\kappa_\varepsilon \in (0,1)$ and decay damping $\alpha_\varepsilon \in (0,1)$. The initial inner tolerance ε_0 is chosen adaptively, based on the primal-dual guess (x^0, s^0, y^0, z^0) according to

$$\varepsilon_0 = \xi \|\nabla \mathcal{M}_0(x^0, s^0, y^0, z^0)\|_2, \tag{1.75}$$

where $\xi \in (0,1)$ is the initial damping factor.

This nested outer-innerloop framework enables the algorithm to handle non-strict convexity while ensuring numerical stability, as each inner problem remains strictly convex and well-conditioned. Termination of the algorithm is achieved when both the primal and dual residuals satisfy the prescribed tolerances, thereby guaranteeing convergence to a primal-dual solution, see § 1.4.5.

Once the inner loop converges ,i.e., the gradient of the subproblem is sufficiently small, the outerloop updates the estimators and the barrier parameter, creating a new convexified subproblem for the next iteration. The convexification ensures that each inner problem is well-posed and tractable.

1.5.5 Primal and primal-dual KKT schemes linear equivalency

In this subsection, we investigate the relationship between the primal and primal-dual KKT systems by examining the equivalence of their associated Hessian matrices. Specifically, we aim to determine whether the dynamics of the primal and primal-dual minimization processes are similar, which would be reflected in structurally related Hessians. To this end, we seek an invertible transformation matrix T such that $\mathcal{H}(\mathcal{L}^{GR}(x,s,\lambda,\mu;\theta_x)) = T^{-1}\mathcal{H}(\mathcal{L}_\delta(x,s,\theta_\mu,\theta_\lambda;\theta_x))$.

Having the λ^*,μ^* substitution rule (earlier $\mathfrak{u},\mathfrak{v}$),

$$\begin{aligned}\lambda^* &\triangleq \frac{A\Delta x}{\delta_\lambda},\\ \mu^* &\triangleq \frac{G\Delta x+\Delta s}{\delta_\mu}.\end{aligned} \tag{1.76}$$

when substituting it in (1.60), we find $\mathcal{H}(\mathcal{M}_\delta^{GR}(x^*,s^*,\lambda^*,\mu^*;\theta_\square)) = \mathcal{H}(\mathcal{L}_\delta(x^*,s^*,\theta_\mu,\theta_\lambda;\theta_x))$, as:

$$\begin{aligned}&\left(\tilde{Q}+\frac{2}{\delta_\lambda}A^\top A+\frac{2}{\delta_\mu}G^\top G\right)\Delta x+\frac{2}{\delta_\mu}G^\top\Delta s-\frac{1}{\delta_\lambda}A^\top A-\frac{1}{\delta_\mu}G^\top G=\nabla_x\mathcal{L}^{GR}(x,s,\lambda,\mu;\theta_x),\\ &\left(\frac{2}{\delta_\mu}G^\top\right)\Delta x+\left(\frac{\sigma\nu}{s^2}+\frac{2\mathbb{I}_{p\times p}}{\delta_\mu}\right)\Delta s-\mathbb{I}_{p\times p}\frac{G\Delta x+\Delta s}{\delta_\mu}=\nabla_s\mathcal{L}^{GR}(x,s,\lambda,\mu;\theta_x).\end{aligned} \tag{1.77}$$

with $\tilde{Q} = Q+\delta_x\mathbb{I}_{n\times n}$.

Moreover, both primal and primal-dual symmetrized KKT matrices are linearly equivalent by finding a back transformation T^{-1} that is triangular and therefore invertible.

$$\begin{aligned}&\begin{bmatrix} Q+\delta_x\mathbb{I}_n+\frac{2}{\delta_\lambda}A^\top A+\frac{2}{\delta_\mu}G^\top G & \frac{2}{\delta_\mu}G^\top & -A^\top & -G^\top\\ \frac{2}{\delta_\mu}G & \frac{\sigma\nu}{s^2}+\frac{2\mathbb{I}_p}{\delta_\mu} & 0 & -\mathbb{I}_p\\ -A & 0 & \delta_\lambda^k\mathbb{I}_m & 0\\ -G & -\mathbb{I}_p & 0 & \delta_\mu^k\mathbb{I}_p\end{bmatrix}\\ &=\begin{bmatrix}\mathbb{I}_n & 0 & \frac{2}{\delta_\lambda}A^\top & \frac{2}{\delta_\mu}G^\top\\ 0 & \mathbb{I}_p & 0 & \frac{2}{\delta_\mu}\mathbb{I}_p\\ 0 & 0 & -\mathbb{I}_m & 0\\ 0 & 0 & 0 & -\mathbb{I}_p\end{bmatrix}\cdot\begin{bmatrix} Q+\delta_x^k\mathbb{I}_n & 0 & A^\top & G^\top\\ 0 & \frac{\sigma\nu}{s^2} & 0 & \mathbb{I}_p\\ A & 0 & -\delta_\lambda^k\mathbb{I}_m & 0\\ G & \mathbb{I}_p & 0 & -\delta_\mu^k\mathbb{I}_p\end{bmatrix}\end{aligned} \tag{1.78}$$

1.5.6 Stabilized primal-dual Interior Point

The primal-dual Newton system (1.71), when directly derived from the regularized merit function $\mathcal{M}_\delta^{GR}$, is largely block-dense and contains terms such as $A^\top A$ and $G^\top G$. These contributions can be computationally expensive and lead to dense matrices even when the linear constraint matrices A and G themselves are sparse. From a numerical perspective,

this formulation is inefficient and undermines the advantages of the more structured Gill-Robinson approach previously emphasized.

1.5.6.1 Formulation

The transformations are performed in two stages. First, the variables λ and μ are substituted into $\mathcal{M}_\delta^{GR}$, as shown in (1.66) and (1.68). Subsequently, the gradient with respect to s is scaled elementwise by s, that is, $s \circ \nabla_s \mathcal{M}_\delta^{GR}$. These operations can equivalently be interpreted as the sequential application of two linear transformations. Starting from the primal-dual Newton system (1.71), we introduce a transformation matrix $\mathcal{T}$ defined as

$$\mathcal{T} = \mathcal{T}_s \cdot \mathcal{T}_r = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & s \end{bmatrix} \cdot \begin{bmatrix} 1 & 2/\delta_\lambda A^\top & 2/\delta_\mu G^\top & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 2/\delta_\mu & 1 \end{bmatrix}, \tag{1.79}$$

which is applied to the gradient of the merit function $\nabla \mathcal{M}_\delta^{GR}(x, s, \lambda, \mu; \theta_\square)$ ((1.69a)-(1.69d)) such as

$$\mathcal{T} \cdot \nabla \mathcal{M}_\delta^{GR}(x, s, \lambda, \mu; \theta_\square) = \begin{bmatrix} Qx + c + A^\top \lambda + G^\top \mu + \delta_x (x - \theta_x) \\ Ax - b - \delta_\lambda (\lambda - \theta_\lambda) \\ Gx - d + s - \delta_\mu (\mu - \theta_\mu) \\ s \cdot \mu - \sigma \nu \end{bmatrix}. \tag{1.80}$$

Figure 1.2 depicts the sequential linear transformations applied to $\nabla \mathcal{M}_\delta^{GR}$, which collectively yield the Newton scheme direction $\tilde{\Delta}$.

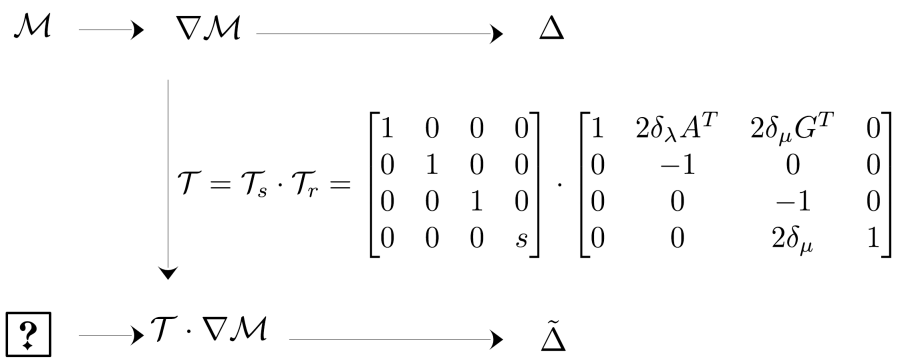


FIGURE 1.2: Merit function, derivatives and directions.

The transformed gradient in the s-direction is now given by $s \cdot \mu - \sigma \nu$, thereby eliminating the $1/s$ term that previously induced the unstable $1/s^2$ behavior in the original Hessian matrix (1.71). The second and third block rows are multiplied by -1 in order to obtain matrix properties that will be examined later.

Taking the derivative of this expression once more with respect to the primal and dual variables yields the transformed Newton system, $\nabla \left(\mathcal{T} \nabla \mathcal{M}_\delta^{GR} \right) \cdot \tilde{\Delta} = -\mathcal{T} \nabla \mathcal{M}_\delta^{GR}$,

$$\begin{bmatrix} Q+\delta_x & A^\top & G^\top & 0 \\ A & -\delta_\lambda \mathbb{I}_{m\times m} & 0 & 0 \\ G & 0 & -\delta_\mu \mathbb{I}_{p\times p} & \mathbb{I}_{p\times p} \\ 0 & 0 & diag(s) & diag(\mu) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta\mu \\ \Delta s \end{bmatrix} = - \begin{bmatrix} Qx + c + A^\top\lambda + G^\top\mu + \delta_x(x-\theta_x) \\ Ax - b - \delta_\lambda(\lambda-\theta_\lambda) \\ Gx - d + s - \delta_\mu(\mu-\theta_\mu) \\ s\cdot\mu - \sigma\nu\mathbf{1}_p \end{bmatrix}. \tag{1.81}$$

The resulting KKT matrix exhibits both sparsity and numerical stability, although it is not symmetric, as the second-order matrix does not correspond to the pure Hessian of the merit function $\mathcal{M}_\delta^{GR}$, see [29] for its original formulation.

However, an important question remains: from an optimality perspective, is it valid to apply the $\mathcal{T}_r$ transformation, and with respect to which merit function does the resulting scheme perform minimization? A formal justification and analysis of these properties are provided in the following subsection.

Remark 1.5.2 — **IPM algorithm**

The subsequent algorithm is the same as for the inner-outerloop classical primal-dual IPM, see Algorithm 4.

1.5.6.2 Merit function minimizer

A linear transformation is applied on the Newton equation (1.71) and the question remains; does system (1.81) minimize $\mathcal{M}_\delta^{GR}$ (see eq: 1.62) even after the $\mathcal{T}r$ transformation is applied to ∇M making an corresponding merit function non-trivial to find ? Figure 1.2 schematizes the transformation process.

To demonstrate that the transformed gradient $\mathcal{T}r\nabla\mathcal{M}^{GR}\delta$ yields a valid search direction $\tilde{\Delta}$ for minimizing $\mathcal{M}^{GR}\delta$, we establish the negativity of the inner product $\langle\nabla\mathcal{M}^{GR}\delta, \tilde{\Delta}\rangle < 0$ in Proof 1.5.1.

Theorem 1.5.1 — **Newton direction $\tilde{\Delta}$ invariance**

Under a gradient transformation $\mathcal{T}_s\cdot\mathcal{T}_r$, see (1.79), the Newton system

$$\nabla\left(\mathcal{T}_s\cdot\mathcal{T}_r\cdot\nabla\mathcal{M}_\delta^{GR}\right)\tilde{\Delta} = -\mathcal{T}_s\cdot\mathcal{T}_r\cdot\nabla\mathcal{M}_\delta^{GR} \tag{1.82}$$

yields the search direction $\tilde{\Delta}$ that minimizes the merit function $\mathcal{M}_\delta^{GR}$.

Proof 1.5.1 — **Newton direction $\tilde{\Delta}$ invariance**

We aim to show that the search direction $\tilde{\Delta}$ minimizes the merit function $\mathcal{M}_\delta^{GR}$ by showing,

$$\langle\nabla\mathcal{M}_\delta^{GR}, \tilde{\Delta}\rangle < 0, \tag{1.83}$$

with the Newton system

Newton direction $\tilde{\Delta}$ invariance

$$\nabla\left(\mathcal{T}_s \cdot \mathcal{T}_r \nabla \mathcal{M}_\delta^{GR}\right)\tilde{\Delta} = -\mathcal{T}_s \cdot \mathcal{T}_r \cdot \nabla \mathcal{M}_\delta^{GR}. \tag{1.84}$$

The transformations and back transformations $\mathcal{T}_s, \mathcal{T}_s^{-1}, \mathcal{T}_r, \mathcal{T}_r^{-1}$ are defined as

$$\mathcal{T}_r = \begin{bmatrix} 1 & {}^2/_{\delta_\lambda}A^\top & {}^2/_{\delta_\mu}G^\top & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & {}^2/_{\delta_\mu} & 1 \end{bmatrix} \quad \mathcal{T}_r^{-1} = \mathcal{T}_r \text{ as } \mathcal{T}_r^{-1} \cdot \mathcal{T}_r = \mathbb{I}_{n+m+2p}, \tag{1.85}$$

$$\mathcal{T}_s = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & s \end{bmatrix} \quad \mathcal{T}_s^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & {}^1/_s \end{bmatrix}, \quad s > 0. \tag{1.86}$$

Let $\mathcal{T} = \mathcal{T}_s \cdot \mathcal{T}_r$ and $\mathcal{T}^{-1} = \mathcal{T}_r^{-1} \cdot \mathcal{T}_s^{-1}$.

Having the Newton equations

$$\begin{aligned}
\nabla\left(\mathcal{T} \cdot \nabla \mathcal{M}_\delta^{GR}\right) \cdot \tilde{\Delta} &= -\mathcal{T} \cdot \nabla \mathcal{M}_\delta^{GR} \\
\nabla \mathcal{M}_\delta^{GR} &= -\mathcal{T}^{-1}\left[\nabla\left(\mathcal{T} \cdot \nabla \mathcal{M}_\delta^{GR}\right) \cdot \tilde{\Delta}\right] \\
\nabla \mathcal{M}_\delta^{GR} &= -\mathcal{T}^{-1} \cdot \left[\nabla\left(\mathcal{T} \cdot \begin{bmatrix} \nabla_x \mathcal{M}_\delta^{GR} \\ \nabla_\lambda \mathcal{M}_\delta^{GR} \\ \nabla_\mu \mathcal{M}_\delta^{GR} \\ \nabla_s \mathcal{M}_\delta^{GR} \end{bmatrix}\right)\right] \cdot \tilde{\Delta} \qquad \text{see (1.69a)-(1.69d)} \\
\nabla \mathcal{M}_\delta^{GR} &= -\mathcal{T}^{-1} \cdot \nabla\left(\begin{bmatrix} Qx + c + A^\top\lambda + G^\top\mu + \delta_x(x - \theta_x) \\ Ax - b - \delta_\lambda(\lambda - \theta_\lambda) \\ Gx - d + s - \delta_\mu(\mu - \theta_\mu) \\ \mu \circ s - \sigma\nu \end{bmatrix}\right) \cdot \tilde{\Delta}, \quad \text{see (1.80)} \\
\nabla \mathcal{M}_\delta^{GR} &= -\mathcal{T}^{-1} \cdot \begin{bmatrix} Q + \delta_x\mathbb{I}_n & A^\top & G^\top & 0 \\ A & -\delta_\lambda & 0 & 0 \\ G & 0 & -\delta_\mu & \mathbb{I}_p \\ 0 & 0 & diag(s) & diag(\mu) \end{bmatrix} \cdot \tilde{\Delta} \\
\nabla \mathcal{M}_\delta^{GR} &= -\begin{bmatrix} Q + \delta_x\mathbb{I}_n + {}^2/_{\delta_\lambda}A^\top A + {}^2/_{\delta_\mu}G^\top G & -A^\top & -G^\top & {}^2/_{\delta_\mu}G^\top \\ -A & \delta_\lambda\mathbb{I}_m & 0 & 0 \\ -G & 0 & \delta_\mu\mathbb{I}_p & -\mathbb{I}_p \\ {}^2/_{\delta_\mu}G & 0 & -\mathbb{I}_p & diag\left({}^2/_{\delta_\mu} + \mu/s\right) \end{bmatrix} \cdot \tilde{\Delta},
\end{aligned} \tag{1.87}$$

yielding in the product

Newton direction $\tilde{\Delta}$ invariance

$$
\begin{aligned}
&\langle \nabla \mathcal{M}_\delta^{GR}, \tilde{\Delta} \rangle = \langle -\mathcal{T}^{-1} \cdot \left[\nabla \left(\mathcal{T} \cdot \nabla \mathcal{M}_\delta^{GR} \right) \tilde{\Delta} \right] \cdot \tilde{\Delta} \rangle \\
&= -\tilde{\Delta}^\top \cdot \begin{bmatrix} Q + \delta_x \mathbb{I}_n & A^\top & G^\top & 0 \\ A & \delta_\lambda & 0 & 0 \\ G & 0 & \delta_\mu & \mathbb{I}_p \\ 0 & 0 & diag(s) & diag(\mu / s) \end{bmatrix} \cdot \tilde{\Delta}. \\
&= -\left[\Delta x^\top \tilde{Q} \Delta x + \Delta s^\top \frac{\mu}{s} \Delta s \right] \\
&\quad - \left[\frac{2}{\delta_\lambda} \Delta x^\top A^\top A \Delta x + \Delta \lambda^\top \delta_\lambda \Delta \lambda - 2 \Delta x^\top A^\top \Delta \lambda \right] \\
&\quad - \left[\frac{2}{\delta_\mu} \Delta x^\top G^\top G \Delta x - 2 \Delta x^\top G^\top \Delta \mu + \frac{4}{\delta_\mu} \Delta x^\top G^\top \Delta s + \Delta \mu^\top \delta_\mu \Delta \mu \right] \\
&\quad - \left[\Delta s^\top \frac{2}{\delta_\mu} \Delta s - 2 \Delta \mu^\top \Delta s \right].
\end{aligned} \tag{1.88}
$$

The terms in **black** are positive, and we investigate the sums of terms in **blue** and **orange**.

In **blue**,

$$
\begin{aligned}
&\frac{2}{\delta_\lambda} \Delta x^\top A^\top A \Delta x + \Delta \lambda^\top \delta_\lambda \Delta \lambda - 2 \Delta x^\top A^\top \Delta \lambda \\
&= \frac{2}{\delta_\lambda} \left[\left(\Delta x^\top A^\top A \Delta x - \delta_\lambda \Delta x^\top A^\top \Delta \lambda + \Delta \lambda^\top \frac{\delta_\lambda^2}{4} \Delta \lambda \right) + \Delta \lambda^\top \frac{\delta_\lambda^2}{4} \Delta \lambda \right] \\
&= \frac{2}{\delta_\lambda} \left[\left(\Delta x^\top A^\top - \frac{\delta_\lambda}{2} \Delta \lambda \right)^\top \cdot \left(\Delta x^\top A^\top - \frac{\delta_\lambda}{2} \Delta \lambda \right) + \Delta \lambda^\top \frac{\delta_\lambda^2}{4} \Delta \lambda \right] > 0.
\end{aligned} \tag{1.89}
$$

In **orange**,

$$
\begin{bmatrix} \Delta x \\ \Delta \mu \\ \Delta s \end{bmatrix}^\top \begin{bmatrix} {}^2/_{\delta_\mu} G^\top G & -G^\top & {}^2/_{\delta_\mu} G^\top & \\ -G & 0 & -\delta_\mu \mathbb{I}_p & -\mathbb{I}_p \\ {}^2/_{\delta_\mu} G & -\mathbb{I}_p & {}^2/_{\delta_\mu} - \mathbb{I}_p & \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \mu \\ \Delta s, \end{bmatrix} \tag{1.90}
$$

having a M matrix positive, shown using the Sylvester criterion, see [55],

$$
\begin{aligned}
&\frac{2}{\delta_\mu} G^\top G > 0 \\
&\frac{2}{\delta_\mu} G^\top G \delta_\mu + G^\top G > 0 \\
&\frac{2}{\delta_\mu} G^\top G \cdot (3 \cdot \mathbb{I}_p) - \frac{2}{\delta_\mu} G^\top G + \frac{2}{\delta_\mu} G^\top G > 0.
\end{aligned} \tag{1.91}
$$

As all terms from equation (1.88) are positive and negated, the transformation $\mathcal{T}$ applied on $\nabla \mathcal{M}_\delta^{GR}$ leads to a Newton scheme minimizing $\mathcal{M}_\delta^{GR}$. □

1.5.6.3 KKT matrix properties: Nesterov-Todd decomposition

The structure of the KKT system can be exploited to simplify the underlying linear algebra. The Nesterov-Todd (NT) decomposition [95] provides a framework enabling a transformation of the primal-dual variables that symmetrizes and decouples parts of the system. This

decomposition leads to a reduced and better-conditioned linear system of smaller dimension, making the linear algebra more efficient and stable.

For a quadratic program with equality constraints $Ax = b$, inequality constraints $Gx + s = d$, slack $s \geq 0$ and duals $\mu \geq 0$, the regularized KKT system is

$$\underbrace{\begin{bmatrix} Q + \delta_x \mathbb{I}_n & A^\top & G^\top & 0 \\ A & -\delta_\lambda \mathbb{I}_m & 0 & 0 \\ G & 0 & -\delta_\mu \mathbb{I}_p & \mathbb{I}_p \\ 0 & 0 & diag(s) & diag(\mu) \end{bmatrix}}_{\text{KKT}} \begin{bmatrix} \Delta x \\ \Delta \lambda \\ \Delta \mu \\ \Delta s \end{bmatrix} = - \begin{bmatrix} r_x \\ r_\lambda \\ r_\mu \\ r_s \end{bmatrix}, \tag{1.92}$$

where $r_s = s \circ \mu - \sigma \nu \mathbf{1}_p$.

The definitions of the residual terms correspond to those introduced in Equation (1.81). From the 4^{th} block equations (complementarity),

$$S\,\Delta\mu + \boldsymbol{\mu}\Delta s = -r_s \quad \Leftrightarrow \quad \Delta s = (-r_s - S\,\Delta\mu) \oslash (\mu), \tag{1.93}$$

Δs is isolated and substituted into the 3^{rd} block-row of the KKT such as,

$$\begin{aligned} & G\Delta x - \delta_\mu \Delta\mu + \Delta s = r_\mu \\ \Leftrightarrow & G\Delta x - \delta_\mu \Delta\mu + (-r_s - S\,\Delta\mu) \oslash (\mu) = r_\mu \\ \Leftrightarrow & G\Delta x - (W + \delta_\mu)\Delta\mu = -r_\mu + (r_s) \oslash (\mu) = -\tilde{r}_{\mu,s} \end{aligned} \tag{1.94}$$

with $\oslash$ the Hadamard component wise division and $W = (s) \oslash (\mu)$, leading to the reduced symmetric KKT system with variables $(\Delta x, \Delta\lambda, \Delta\mu)$,

$$\underbrace{\begin{bmatrix} Q + \delta_x \mathbb{I}_n & A^\top & G^\top \\ A & -\delta_\lambda \mathbb{I}_m & 0 \\ G & 0 & -(W + \delta_\mu \mathbb{I}_p) \end{bmatrix}}_{\text{KKT}_{NT}} \begin{bmatrix} \Delta x \\ \Delta \lambda \\ \Delta \mu \end{bmatrix} = - \begin{bmatrix} r_x \\ r_\lambda \\ \tilde{r}_{\mu,s} \end{bmatrix}. \tag{1.95}$$

where $Q + \delta_x \succ 0$, $\delta_\lambda > 0$, and $W + \delta_\mu > 0$. In the typical symmetric quasi-definite (Vanderbei [125]) form, it reads

$$\text{KKT}_{NT} = \begin{bmatrix} A & B^\top \\ B & -C \end{bmatrix}, \qquad A = \begin{bmatrix} Q + \delta_x \mathbb{I}_n & A^\top \\ A & \delta_\lambda \mathbb{I}_m \end{bmatrix} \succ 0, \qquad C = (W + \delta_\mu)\mathbb{I}_p \succ 0. \tag{1.96}$$

By classical results on quasi-definite matrices, KKT_{NT} is nonsingular and admits a block $\text{LDL}^\top$ factorization

$$\text{KKT}_{NT} = L \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} L^\top, \tag{1.97}$$

where L is unit lower triangular, with one of D_1 or D_2 positive definite and the other negative definite.

1.5.7 Proximal slack-stabilized primal-dual Interior-Point Method

In this subsection, we introduce a novel proximal slack regularization within the stabilized primal-dual interior-point framework. To the best of our knowledge, such a formulation does not appear in the existing literature. This represents a first attempt to enhance numerical stability and control the evolution of slack variables in a unified primal-dual setting.

We augment the generalized regularized primal-dual merit function $\mathcal{M}_{\delta}^{GR}$ with an additional Euclidean proximal term on the slack variables, yielding

$$\mathcal{M}_{\delta,\delta_s}^{GR}(x,y,\lambda,\mu;\theta_x,\theta_\lambda,\theta_\mu,\theta_s) = \mathcal{M}_{\delta}^{GR}(x,y,\lambda,\mu;\theta_x,\theta_\lambda,\theta_\mu) + \frac{1}{2\delta_s}\|\theta_s - s\|_2^2, \tag{1.98}$$

where $s \in \mathbb{R}^p$, $s > 0$, and θ_s denotes a proximal estimator. Applying the same transformation and derivation steps as in the stabilized KKT system (1.81), we obtain the following proximal-slack-regularized system:

$$\begin{bmatrix} Q+\delta_x & A^\top & G^\top & 0 \\ A & -\delta_\lambda \mathbb{I}_m & 0 & 0 \\ G & 0 & -\delta_\mu \mathbb{I}_p & \mathbb{I}_p \\ 0 & 0 & \operatorname{diag}(s) & \operatorname{diag}(\zeta) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta\mu \\ \Delta s \end{bmatrix} = - \begin{bmatrix} Qx + c + A^\top\lambda + G^\top\mu + \delta_x(x-\theta_x) \\ Ax - b - \delta_\lambda(\lambda-\theta_\lambda) \\ Gx - d + s - \delta_\mu(\mu-\theta_\mu) \\ s\circ\mu - \sigma\nu + s\cdot\delta_s(s-\theta_s) \end{bmatrix}, \tag{1.99}$$

with $\zeta = \mu + \delta_s(2s - \theta_s)$.

In the KKT matrix, the term $\nabla_{ss}^2 = \operatorname{diag}(\mu + \delta_s(2s-\theta_s))$ appears as a diagonal correction. When $s \approx \theta_s$, this simplifies to approximately $\operatorname{diag}(\mu + \delta_s s)$, providing a stabilizing penalty that scales with the magnitude of s. Similarly, in the residual, the term $s \cdot \delta_s(s-\theta_s)$ vanishes as s approaches θ_s.

Within the outer-inner loop algorithm, the estimators are updated at the outer iteration and remain fixed during the inner loop, which defines a sub-QP problem. For the augmented Lagrangian and proximal terms, the standard updates $\theta_\square^{k+1} = \theta_\square^k$ are employed (see Algorithm 4). A similar approach could be adopted for θ_s, yet we propose here an analytical update rule.

We define θ_s as the minimizer of the subproblem,

$$\theta_s^+ = \underset{s\in\mathbb{R}^p}{\arg\min}\ \mathcal{M}_{\delta}^{GR}(x,s,\mu,\lambda;\theta_\mu,\theta_\lambda,\theta_x,\theta_s), \tag{1.100}$$

whose stationary condition, after applying the transformation $\mathcal{T}_r$, reads

$$s\circ\nabla_s\mathcal{M}_{\delta}^{GR}(x,s,\mu,\lambda;\theta_\mu,\theta_\lambda,\theta_x,\theta_s) = \mu\circ s - \sigma\nu + s\cdot\delta_s(s-\theta_s) = 0. \tag{1.101}$$

This leads to a quadratic equation in θ_s for each component. The corresponding discriminant is

$$\Delta = (\mu + \delta_s s)^2 + 4\delta_s\sigma s > 0, \tag{1.102}$$

which guarantees the existence of real and positive roots. Selecting the positive root, the analytical update for θ_s in the outer loop becomes

$$\theta_s^{k+1} = \frac{2\sigma s}{(\mu + \delta_s s^k) + \sqrt{(\mu + \delta_s s^k)^2 + 4\delta_s \sigma s}}. \tag{1.103}$$

This expression ensures smooth and stable evolution of θ_s, maintaining both feasibility and numerical robustness in the proximal-slack regularized interior-point framework.

Concerning the Nesterov-Todd decomposition, the definitions of the residual terms follow those introduced in Equation (1.81). From the fourth block of equations (the complementarity conditions), $-\tilde{r}_{\mu,s}$ is derived from,

$$\nabla_s \mathcal{M}^{GR}_{\delta,\delta_s} = s \circ \mu - \sigma\nu + s \cdot \delta_s(s - \theta_s), \tag{1.104}$$

where the additional term $s \cdot \delta_s(s - \theta_s)$ does not alter the form of the Nesterov–Todd decomposition; it only results in a redefinition of W and $\tilde{r}_{\mu,s}$.

1.6 Regularization interpretation

This section interprets the used regularization techniques through their impact on problem convexity and their influence on the merit function, gradients, and Hessian structure. As introduced in § 1.4.1, an optimization problem can be regularized using various techniques to enhance stability, convergence, and numerical robustness. Regularizations transform the original problem into a more tractable form while preserving its solution and providing convergence guarantees. In this research, we investigate proximal, augmented Lagrangian, and primal-dual augmented Lagrangian methods.

The analysis is based on the proximal augmented Lagrangian and primal-dual Gill-Robinson terms in equation (1.62).

Definition 1.6.1 **Diagonal shift and Positivity**

Let $M \in \mathbb{R}^{n\times n}$ be a symmetric matrix positive semi-definite with $M \succeq 0$. For any scalar $\delta > 0$, the shifted matrix

$$M_\delta := M + \delta \mathbb{I}_n \tag{1.105}$$

is *positive definite*, i.e.

$$x^\top M_\delta x > 0 \quad \forall\, x \neq 0, \tag{1.106}$$

and hence invertible. We denote this property as

$$M_\delta \succ 0. \tag{1.107}$$

Definition 1.6.1 establishes a simple regularization principle. By adding a small multiple on the diagonal of any symmetric positive semi-definite matrix M, it becomes strictly positive definite. This diagonal shift ensures numerical stability and always guarantees the invertibility of otherwise singular or ill-conditioned systems. In terms of optimization, the constrained problem becomes strictly convex while keeping the same solution.

Proof 1.6.1 — **Diagonal shift and Positivity**

Let $M \in \mathbb{R}^{n\times n}$, $M \geq 0$, we have $x^\top Mx \geq 0$ for all $x \in \mathbb{R}^n$. Then, for any $x \neq 0$,

$$x^\top M_\delta x = x^\top Mx + \delta\|x\|^2 \geq \delta\|x\|^2 > 0, \tag{1.108}$$

and M_δ is positive definite. □

1.6.1 The Euclidean proximal operator

In the merit function (1.62), the proximal regularization is expressed as an Euclidean norm term. This term acts as a damping mechanism centered at the current iterate x_k, preventing the next iterate x_{k+1} from moving too far away (update $\theta_{k+1} = x_k$). The regularization penalizes deviations from the estimator θ_x and can be seen as a local quadratic approximation that convexifies the objective function.

Let $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ be a continuous, smooth, and convex function. The operator $\text{prox}_{\delta_x f}(\theta_x) = \frac{\delta_x}{2}\|\theta_x - x\|_2^2$, see definition 1.6.2. The operator shifts x in the direction of the minimum of f, with the amount of movement controlled by the parameter δ_x, while keeping it near θ_x. A full derivation, interpretation, and theoretical discussion can be found in [101].

Definition 1.6.2 — **Proximal operator**

Let $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ be a convex function. The proximal operator of f at a point $\theta_x \in \mathbb{R}^n$ with parameter $\delta_x > 0$ is defined as

$$\text{prox}_{\delta_x f}(x) := \arg\min_x \left\{ f(x) + \frac{\delta_x}{2}\|x - \theta_x\|_2^2 \right\}. \tag{1.109}$$

This operator returns the point that best balances minimizing f and remaining close to the reference point θ_x in the Euclidean sense.

As stated in definition 1.6.3, x^* is a fixed point of the proximal operator if and only if it is a minimizer of f. Therefore, the solution x^* can be interpreted both as a proximal fixed point and as the solution of the optimization problem.

Definition 1.6.3 — **Proximal fixed-point operator**

A point $\hat{x} \in \mathbb{R}^n$ is called a proximal fixed point if

$$\hat{x} = \text{prox}_{\delta_x f}(\hat{x}). \tag{1.110}$$

Theorem 1.6.1 links a proximal fixed point to the $\arg\min_x f(x)$.

Theorem 1.6.1 — **Proximal fixed-point and minimizer of $f(x)$**

A proximal fixed point,

$$\hat{x} = \text{prox}_{\delta_x f}(\hat{x}), \tag{1.111}$$

coincide with the minimizers of f, $\hat{x} = x^*$.

Proof 1.6.2 — **Proximal fixed-point and minimizer of $f(x)$**

The formal proof of Theorem 1.6.1 can be found in [101, 120].

The Moreau envelope $M_{\delta_x f}$, see definition 1.6.4, acts not only as a smoothing operator for nonsmooth functions but also as a curvature regularizer for smooth cost functions. It produces a surrogate objective whose gradient is globally Lipschitz continuous with constant δ_x, regardless of the conditioning of $\nabla^2 f$. The gradient of the Moreau envelope $\nabla M_{\delta_x f}(x)$ is Lipschitz continuous [101]. Indeed, for any $x, x' \in \mathbb{R}^n$,

$$\begin{aligned} \|\nabla M_{\delta_x f}(x) - \nabla M_{\delta_x f}(x')\|_2 &= \|\nabla f(x) - \nabla f(x') + \delta_x(x-\theta_x) - \delta_x(x'-\theta_x)\|_2 \\ &= \|\nabla f(x) - \nabla f(x')\|_2 + \delta_x\|x-x'\|_2 \\ &\leq \delta_x\|x-x'\|_2. \end{aligned} \tag{1.112}$$

so that $L = \delta_x$ is the Lipschitz constant of $\nabla M_{\delta_x f}$ by definition. As a result, the neighborhood of θ_x is better conditioned, and as $\delta_x \to 0$, the regularization decreasingly convexifies the problem until reaching the original QP problem, see [89] for a formal proof.

In other words, even for non-strictly convex QPs, where the gradients may vanish in certain regions, the Lipschitz continuity of the gradient of the Moreau–Yosida envelope ensures that descent directions remain well-scaled, effectively providing a momentum-like effect that guides the iterates steadily toward the solution.

Definition 1.6.4 — **Moreau–Yosida regularization**

The Moreau–Yosida regularization (Moreau envelope) of f is defined as

$$M_{\delta_x f}(x) := \inf_{x\in\mathbb{R}^n} \left\{ f(x) + \frac{\delta_x}{2}\|x-\theta_x\|_2^2 \right\}. \tag{1.113}$$

The function $M_{\delta_x f}$ is a smooth, regularized approximation of f: it is finite-valued and continuously differentiable on $\mathbb{R}^n$, even when f is not. The sets of minimizers of f and $M_{\delta_x f}$ coincide, so minimizing $M_{\delta_x f}$ is equivalent to minimizing f while yielding a smooth optimization landscape.

Modified residuals: when deriving the merit function with the proximal Euclidean regularization, the gradient in the x-direction can be expressed, for example using an unconstrained optimization problem with a proximal regularization term, as

$$\nabla_x \left(f(x) + \frac{\delta_x}{2}\|\theta_x - x\|_2^2 \right) = \nabla_x f(x) + \delta_x(x-\theta_x). \tag{1.114}$$

The proximal regularization not only stabilizes the residual of $f(x)$ but also ensures a globally Lipschitz gradient, improving both the numerical conditioning and the convergence behavior of Newton or quasi-Newton updates.

Modified Hessian matrix: thus, after deriving the gradient of the regularized merit function or Lagrangian, the corresponding Hessian can be computed, and in the x-direction becomes

$$\nabla_x^2\left(f(x)+\frac{\delta_x}{2}\|\theta_x-x\|_2^2\right)=\nabla^2 f(x)+\delta_x\mathbb{I}_n\succ 0. \tag{1.115}$$

If the original Hessian $\nabla^2 f(x)$ is positive semidefinite and possibly singular, the diagonal shift term $\delta_x\mathbb{I}_n$ guarantees that the modified Hessian is positive definite (in the symmetric case before the transformation, see def (1.79)), see Definition 1.6.1. This ensures that the regularized Newton system is always solvable and well-conditioned. As a result, even originally non-strictly convex problems can be handled robustly, and Newton or quasi-Newton steps remain well-defined, enabling stable innerloop iterates in primal-dual algorithms.

1.6.2 The primal-dual Gill-Robinson Augmented Lagrangian

The Gill-Robinson primal-dual augmented Lagrangian [53] extends the classical augmented Lagrangian approach by incorporating both primal and dual regularization terms. This formulation simultaneously stabilizes the Newton system and enforces feasibility, providing a well-conditioned framework for solving convex quadratic programs, even in the presence of rank-deficient or linearly dependent constraints. The classical augmented Lagrangian and Gill-Robinson regularization can be expressed as an additive operator in the form

$$\begin{aligned}
P_{A,b,\theta_\lambda,\delta_\lambda}(x,\lambda)&:=\frac{1}{2\delta_\lambda}\|Ax-b\|_2^2+\frac{1}{2\delta_\lambda}\|Ax-b-\delta_\lambda(\lambda-\theta_\lambda)\|_2^2,\\
P_{G,d,\theta_\mu,\delta_\mu}(x,s,\mu)&:=\frac{1}{2\delta_\mu}\|Gx-d+s\|_2^2+\frac{1}{2\delta_\mu}\|Gx-d+s-\delta_\mu(\mu-\theta_\mu)\|_2^2.
\end{aligned} \tag{1.116}$$

We call the augmented Lagrangian operators $P_{A,b,\theta_\lambda,\delta_\lambda}=P_\lambda$ and equivalently, $P_{G,d,\theta_\mu,\delta_\mu}=P_\mu$. The operators are additive on the merit function level, and we introduce $P=P_\lambda+P_\mu$. The gradient of the operators P_λ,P_μ with respect to x,λ,μ reads

$$\begin{aligned}
\nabla_x P_\lambda+\nabla_x P_\mu&=\frac{1}{\delta_\lambda}A^\top(Ax-b)+\frac{1}{\delta_\mu}G^\top(Gx-d+s)\\
&\quad+\frac{1}{\delta_\lambda}A^\top(Ax-b-\delta_\lambda(\lambda-\theta_\lambda))+\frac{1}{\delta_\mu}G^\top(Gx-d+s-\delta_\mu(\mu-\theta_\mu)),\\
\nabla_\lambda P_\lambda+\nabla_\lambda P_\mu&=-(Ax-b-\delta_\lambda(\lambda-\theta_\lambda))\\
\nabla_\mu P_\lambda+\nabla_\mu P_\mu&=-(Gx-d+s-\delta_\mu(\mu-\theta_\mu)).
\end{aligned} \tag{1.117}$$

In the direction x, the gradient of the operators in (1.116) establishes a regularized coupling between the primal and dual variables. Viewed as penalty operators, $P\lambda$ and $P\mu$ enforce feasibility of $Ax=b$ and $Gx-d+s=0$. Unlike classical augmented Lagrangian quadratic penalty methods, this formulation incorporates dual variables through the correction terms $\lambda-\theta_\lambda$ and $\mu-\theta_\mu$. The gradients of the operators vanish only when both the constraint violations and the dual variable corrections approach zero, ensuring satisfaction of the KKT optimality conditions. The weighting terms $\frac{1}{\delta_\lambda}$ and $\frac{1}{\delta_\mu}$ regulate the strength of the primal-dual enforcement and directly influence the local curvature of the constrained problem.

When derivating the gradients, the Hessian of the primal-dual operator (1.116) can be written

$$\nabla^2 P(x,\lambda,\mu) = \begin{bmatrix} \frac{2}{\delta_\lambda}A^\top A + \frac{2}{\delta_\mu}G^\top G & -A^\top & -G^\top \\ -A & \delta_\lambda \mathbb{I}_m & 0 \\ -G & 0 & \delta_\mu \mathbb{I}_p \end{bmatrix}, \tag{1.118}$$

On the full diagonal, the regularization terms introduce a positive shift, ensuring positive-definiteness (see 1.6.1). This effect is analogous to the convexification induced by the proximal operator on the cost function, but here it acts on the feasible set $\mathcal{F}$. For the variable x, the term, $\frac{2}{\delta_\lambda}A^\top A + \frac{2}{\delta_\mu}G^\top G$ is positive semi-definite and acts as a regularized primal curvature, enforcing local convexity along the feasible directions defined by the constraints. When some equality or inequality constraints are linearly dependent, the matrices A and G lose full rank, leading to potential singularities in the Newton system. The inclusion of the regularization terms $\delta_\lambda, \delta_\mu > 0$, in the second and third block lines, effectively breaks these linear dependencies, restoring invertibility. For sufficiently small values of δ_λ and δ_μ, this modification preserves the original optimal solution while ensuring a stable and well-conditioned Newton update.

The presence of the off-diagonal blocks $-A, -G, -A^\top, -G^\top$ couples the primal and dual variables and prevents singularities in the Newton system. Using the Schur complement with $D = \operatorname{diag}(\delta_\lambda I_m, \delta_\mu I_p) \succ 0$, the block matrix in (1.118) is positive semidefinite because its Schur complement is

$$S = \frac{1}{\delta_\lambda}A^\top A + \frac{1}{\delta_\mu}G^\top G \succeq 0. \tag{1.119}$$

Equivalently, its determinant factors as

$$\det\begin{bmatrix} \frac{2}{\delta_\lambda}A^\top A + \frac{2}{\delta_\mu}G^\top G & -A^\top & -G^\top \\ -A & \delta_\lambda I_m & 0 \\ -G & 0 & \delta_\mu I_p \end{bmatrix} = (\delta_\lambda)^m \cdot (\delta_\mu)^p \cdot \det\Big(\frac{1}{\delta_\lambda}A^\top A + \frac{1}{\delta_\mu}G^\top G\Big). \tag{1.120}$$

The regularization adds some monotonicity to the optimization problem. As the Newton scheme converges and $\delta_\lambda, \delta_\mu \to 0$, both the constraint violations and the associated regularization terms vanish. Conversely, as $\frac{1}{\delta_{\lambda,\mu}}$ increases, the constraints are enforced more strictly. This leads to larger augmented penalty weights but smaller residuals $|Ax-b|$ and $|Gx-d+s|$, thereby driving the iterates closer to feasibility.

1.7 Safe-guards, corrections and decompositions

In practical implementations of primal-dual interior-point methods, theoretical convergence guarantees may be challenged by numerical errors, ill-conditioning, or proximity to the *log*-barrier. Safeguards and correction mechanisms are introduced to enhance robustness in these situations. These techniques, including central path corrections, fraction-to-the-boundary rules, centering barriers, boundary control, and early-termination amortization, ensure robustness and reliability where theory and numerical practice sometimes collide. Importantly,

they are designed to be activated if needed, as unnecessary application can slow down the process.

1.7.1 Central path correction

Formulation: The central path is defined as the trajectory of primal-dual iterates $(x(\sigma),\ s(\sigma),\ \lambda(\sigma),\ \mu(\sigma))$ satisfying the perturbed KKT system with barrier parameter $\sigma > 0$. The idea behind this process is to update the estimators $\theta_x, \theta_\lambda, \theta_\mu$ of the primal-dual IPM to maintain proximity to the central path neighborhood. This approach is less aggressive than directly modifying the variables x, λ, μ, s, yet the estimators, by defining a damping neighborhood, guide the solution trajectory along the central path. Using the stable Newton-KKT system (1.81), we define the slack-dual operator F as

$$F(s, \mu, \Delta s, \Delta\mu) := \operatorname{diag}(s)\Delta\mu + \operatorname{diag}(\mu)\Delta s + s \circ \mu - \sigma\nu\mathbf{1}_p. \tag{1.121}$$

The last block of the Newton system can then be compactly written as

$$F(s, \mu, \Delta s, \Delta\mu) - s \circ \mu + \sigma\nu = 0, \tag{1.122}$$

which represents the deviation from the central path. The central path itself is obtained by solving

$$F(s, \mu, 0, 0) = s \circ \mu - \sigma\nu\mathbf{1}_p = 0, \tag{1.123}$$

so that the slack and dual variables evolve along a strictly positive trajectory satisfying $s > 0$, $\mu > 0$. The central path guarantees convergence to the optimal KKT solution as $\sigma \to 0$, while the Newton step updates $(\Delta x, \Delta\lambda, \Delta s, \Delta\mu)$ enforce complementarity. Staying close to the central path ensures that the iterates maintain a balance between feasibility and complementarity, which provides the best convergence behavior in primal-dual interior-point methods. Deviations from the central path can lead to slower convergence, numerical instability, or stepping outside the feasible region, motivating the use of central path correction as a safeguard to guide iterates back toward the central trajectory when necessary.

The correction can be formulated as [128]§5,

$$\begin{bmatrix} Q + \delta_x & A^\top & G^\top & 0 \\ A & -\delta_\lambda I_m & 0 & 0 \\ G & 0 & -\delta_\mu I_p & I_p \\ 0 & 0 & \operatorname{diag}(s) & \operatorname{diag}(\mu) \end{bmatrix} \begin{bmatrix} \partial x \\ \partial\lambda \\ \partial\mu \\ \partial s \end{bmatrix} = - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \tag{1.124}$$

leading to the corrections,

$$
\begin{aligned}
\theta_{\lambda}^{k} &\leftarrow \theta_{\lambda}^{k} + \left(\nu^{k} - \nu^{k+1}\right) \cdot \partial\lambda, \\
\theta_{\mu}^{k} &\leftarrow \theta_{\mu}^{k} + \left(\nu^{k} - \nu^{k+1}\right) \cdot \partial\mu, \\
\theta_{x}^{k} &\leftarrow \theta_{x}^{k} + \left(\nu^{k} - \nu^{k+1}\right) \cdot \partial x,
\end{aligned}
\tag{1.125}
$$

under assumptions $\nu^{k} > \nu^{k+1}$ and the distance to the central path $s \circ \mu$ is large enough.

Algorithm: The central path correction is applied after updating the estimators whenever the new solutions deviate too far from the central path,

Algorithm 5 `QProxIP`, central path correction safeguard.

```
Init
while Outerloop do
    while Innerloop do
        Solve (1.81)
        Linesearch
    end while
    Update the estimators: θ_{□k+1}
    Update the weights: δ_{□k+1}
    Update the barrier parameter: ν_{k+1}
    Compute centrality measure: d_c = ‖s ∘ μ − ν_{k+1} 1‖_2
    Compute threshold: τ = τ_0 + τ_1 ν_{k+1}
    if d_c > τ then
        Solve (1.124) and apply (1.125)
    end if
end while
Return the optimal primal-dual solution (x*, s*, λ*, μ*),
```

with $\tau_0 = 10^{-5}, \tau_1 = 10.0$. As shown in Algorithm 5, after each update of the estimators $(\theta_{\square})$, weights $(\delta_{\square})$ and the barrier parameter ν, the algorithm measures the distance of the current iterate from the central path using the quantity $d_c = |s \circ \mu - \nu \mathbf{1}|_2$. If this distance exceeds an affine threshold $\tau = \tau_0 + \tau_1 \nu$, a correction is applied by solving the system (1.124) and updating the estimators according to (1.125). This correction incurs the computational cost of solving a KKT system; however, the symbolic and numerical factorizations are retained, so only the system solve using the existing factorization is required. The centrality measure and the Mehrotra-style corrector-predictor was used in a proximal QP primal-dual IPM in [114].

1.7.2 Fraction to the boundary

Formulation: In Interior Point framework, the fraction-to-the-boundary rule ensures that the next iterate remains strictly feasible with respect to the inequality constraints $s \geq 0$ and $\mu \geq 0$. Given a search direction $(\Delta s, \Delta\mu)$ from the Newton step, the maximal step sizes $\alpha_{s}^{\max}$ and $\alpha_{\mu}^{\max}$ that maintain strict positivity are

$$\alpha_{\max} = \min\left(\max\{\alpha \in [0,1] \mid s + \alpha\Delta s \geq 0\}, \;\; \max\{\alpha \in [0,1] \mid \mu + \alpha\Delta\mu \geq 0\} \right). \quad (1.126)$$

The fraction-to-the-boundary rule then chooses the step size as a fraction $\tau \in (0,1)$ of the maximum $\alpha = \tau\alpha^{\max}$, with

$$\tau = \max(\tau_{\min}, 1-\nu), \qquad \alpha_{\max} = \min\left(-\tau\frac{s_i}{\Delta s_i} \quad | \quad \Delta s_i < 0, ; i = 1, \ldots, p \right), \quad (1.127)$$

where the minimum is taken only over the components of Δs that would decrease s_i, i.e., those with $\Delta s_i < 0$, to ensure that the updated slack variables remain strictly positive. Typically, τ is chosen between 0.9 and 0.999 to prevent the iterates from getting too close to the boundary, being numerically destabilizing. This approach ensures that the iterates remain in the interior of the feasible set while allowing the Newton method to make meaningful progress along the search direction. The fraction to the boundary method was introduced first in [126].

Algorithm: Algorithm 6 describes the integration of the a fraction-to-the-boundary rule in the innerloop. The rule is enforced to determine the maximal step size $\alpha_{\max}$ that keeps the slacks s and duals μ strictly positive. The maximal step α is then used as the starting point of the linesearch backtracking.

Algorithm 6 Central path correction.

1: **while** Innerloop **do**
2: Solve $(x_k^j, s_k^j, \lambda_k^j, \mu_k^j, KKT.\Delta = -R$ (1.81)
3: Fraction to the boundary: find $\alpha_{\max}$
4: Linesearch: find $(x_k^j, s_k^j, \lambda_k^j, \mu_k^j)$ such that $\left\| \nabla \mathcal{M}_\delta^{GR}(x_k^j, s_k^j, \lambda_k^j, \mu_k^j) \right\| \leq \varepsilon_k$ and $s_k^j > 0$
5: Update: $[x, \lambda, \mu, s] \leftarrow [x, \lambda, \mu, s] + \alpha[\Delta x, \Delta\lambda, \Delta\mu, \Delta s]$
6: **end while**

1.7.3 Centering the barrier

To guide the balance between centering and progress in a Mehrotra-style predictor-corrector [96], [103] interior-point method, we compute a centering parameter σ based on the predicted complementarity gap. The current average complementarity is given by $\bar{\mu} = \frac{s^\top \mu}{p}$.

After the linesearch, the predicted average complementarity becomes

$$\mu_{\text{avg}} = \frac{(s + \alpha\Delta s)^\top (\mu + \alpha\Delta\mu)}{p}. \quad (1.128)$$

To measure the relative change in complementarity, we define the ratio

$$\eta = \frac{(s + \alpha\Delta s)^\top (\mu + \alpha\Delta\mu)}{s^\top \mu}. \tag{1.129}$$

The centering parameter σ is then computed by applying the cubic transformation,

$$\sigma = \Big(\min\big(1.0, \max(0.1,\ \eta)\big)\Big)^3. \tag{1.130}$$

This cubic transformation ensures that σ remains small when the linesearch update already predicts strong progress and approaches 1 when additional centering is required. The resulting σ is subsequently used to define the right-hand side of the corrector system in the predictor-corrector iteration, see [103].

1.7.4 Boundary control

In interior-point methods, the slack variables s must remain strictly positive to ensure the logarithmic barrier terms well-definiteness. During early iterations, however, some components of s may approach zero, potentially causing numerical instability or breakdown of the method [103]. To prevent this, we monitor the smallest component of the slack vector $s_{min} = \min_i s_i$ and if it falls below a critical threshold $s_{min} < \frac{\varepsilon_{\text{early}}}{100}$, where ε is the early termination. All slack variables are slightly shifted to avoid division by zero or logarithmic singularities, $s \leftarrow s + 10^{-10}\mathbf{1}_p$. A candidate barrier parameter ν_{try} is then estimated from the current slack s and the inequality feasibility violation, $r = d - Gx, \quad r_+ = \max(r, 0), \quad \nu_{\text{try}} = \frac{r_+^\top s}{p}$. The barrier parameter is updated conservatively as $\nu = \min\left(5 \cdot \nu, \nu_{\text{try}}\right)$, ensuring that ν is not increased too aggressively even in cases of near-infeasibility, and the new value is stored in the problem state $\nu_{\text{try}} \leftarrow \nu$. This heuristic safeguards the interior-point method from instability when iterates approach the boundary of the feasible set prematurely, while gently restoring a higher barrier to guide iterates back toward the central path.

1.7.5 Early termination amortization

To improve the robustness of early termination in our solver, we introduce an amortization factor based on the relative residual mismatch $\phi = \frac{|r_{\text{p}} - r_{\text{d}}|}{\max(r_{\text{p}}, r_{\text{d}})} + 1$, where $(r_{\text{p}}, r_{\text{d}}) > (0, 0)$. The early termination tolerance is applied by $\varepsilon \leftarrow \varepsilon/\phi\varepsilon$, effectively damping the scheduled early termination. When the primal and dual residuals are balanced ($r_{\text{p}} \approx r_{\text{d}}$), $\phi \approx 1$ and termination proceeds as fast as planned; when they are unbalanced, $\phi > 1$ and the method slows down slightly, allowing additional iterations to reduce the larger residual. This mechanism dampens the expected early termination speed when either the primal or dual residual lags behind, ensuring the solver continues iterating until both residuals are adequately reduced.

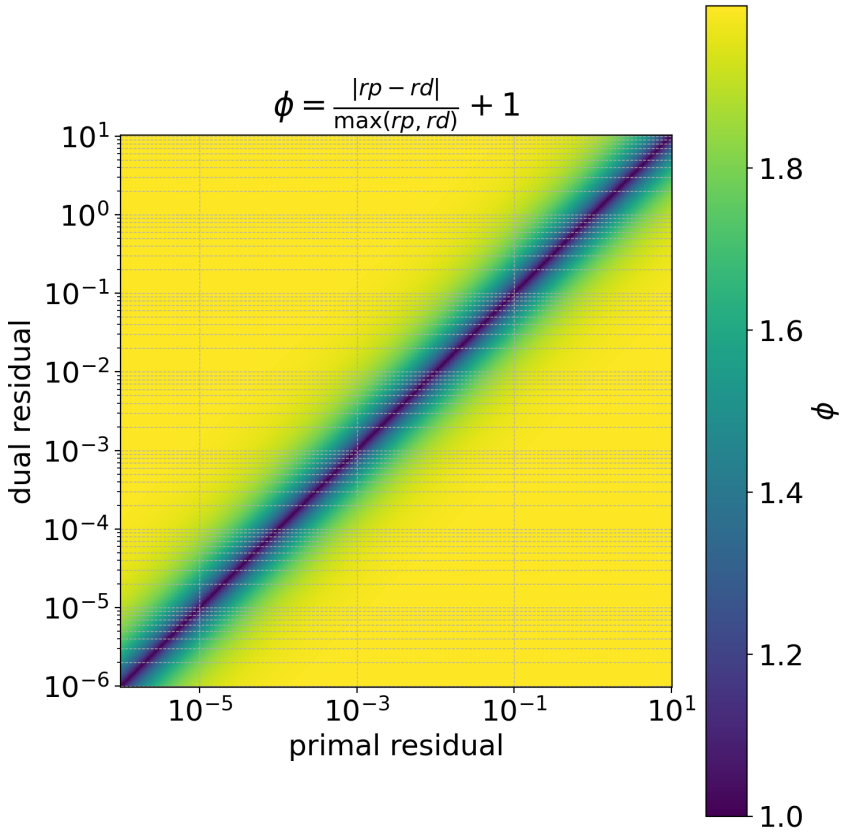


FIGURE 1.3: $\phi(rp, rd)$ mapping.

1.7.6 Symbolic and numerical factorization

At each Newton step of the primal-dual interior-point method, the linearized KKT system must be assembled and solved. We distinguish between two algebraic formulations: the full 4×4 block system, suitable for LU factorization, and the Nesterov-Todd reduced 3×3 symmetric system, suitable for $\mathrm{LDL}^\top$ factorizations.

Full system (LU): The Newton step in the full space of variables $(\Delta x, \Delta\lambda, \Delta\mu, \Delta s)$ is given by (1.81). This system explicitly includes the diagonal slack and dual matrices $diag(s)$, $diag(\mu)$ in the last block row.

Reduced system ($\mathrm{LDL}^\top$): Eliminating Δs from (1.81) yields the symmetric Nesterov–Todd reduced system (1.95) in variables $(\Delta x, \Delta\lambda, \Delta\mu)$. Here the scaling matrix $W = S\,M^{-1}$ appears in the dual block, and the residual is modified to account for the elimination.

Inner-loop updates: During inner iterations, the coefficient blocks Q, A, and G remain constant as well as the full structure; the symbolic factorization needs to be computed only

once. Only the diagonal matrices $\text{diag}(s)$ and $\text{diag}(\mu)$ (and consequently W in the Nesterov–Todd decomposition) require updates at each inner-loop iteration, triggering the corresponding numerical factorization.

1.8 Convergence analysis

1.8.1 Outerloop - Proximal Point Algorithm

We recall Problem (1.3.2) formulation, and strictly enforce the slack variable positivity via the indicator function, see Definition 1.8.1. In the following development, we will relate this formulation to the logarithmic penalty method used in the IPM.

Definition 1.8.1 — **Indicator function**

$$\text{IND}(s) := \begin{cases} 0 & \text{if } s \geq 0, \\ +\infty & \text{otherwise.} \end{cases}$$

Definition 1.8.2 — **Normal Cone**

$$NC_{s\geq 0}(s) := \begin{cases} \{0\} & \text{if } s > 0, \\ (-\infty, 0] & \text{if } s = 0, \\ \emptyset & \text{if } s < 0. \end{cases}$$

The first derivative of the indicator function is a normal cone vector defined on $s \geq 0$ having a free length, see Definition 1.8.2. This term ensures that any candidate solution s that violates the inequality $s \geq 0$ is automatically excluded from the feasible region of the Lagrangian. Figure 1.4 presents the indicator function $IND(s)$ for $s \geq 0$ and its first derivative, the normal cone $NC_{s\geq 0}(s)$ for s positive.

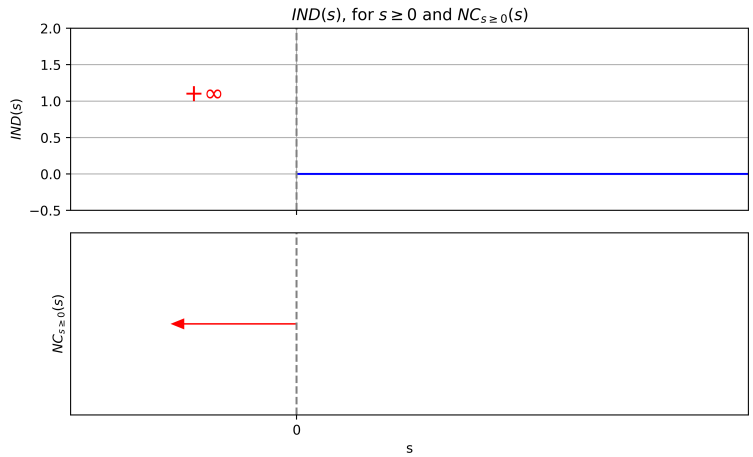


FIGURE 1.4: Indicator and Normal cone function for $s \geq 0$.

The indicator function is a theoretical tool but is used in order to study the interior-point framework without the inconvenience of a barrier penalty.

Let us consider the outer loop from Algorithm 4. It follows an inexact proximal point scheme [109], so that global and local convergence of the outer iterations can be established following the results in [89]. By construction, the regularization parameters are positive, non-increasing, and bounded away from zero. Furthermore, by the early-termination criterion ε_k, the sequence of tolerances is summable, i.e.,

$$\sum_{k=0}^{\infty} \varepsilon_k \leq \varepsilon_0 \sum_{k=0}^{\infty} K_\varepsilon^k = \frac{\varepsilon_0}{1-K_\varepsilon} < +\infty, \text{ see (1.74)} \tag{1.131}$$

The goal of this convergence analysis is to rewrite the outerloop process as a Proximal Point Algorithm (PPA), with $z = [x, \lambda, \mu, s]$ as, PPA $= T(z) + \Sigma(z - \theta_z) \ni 0$.

From optimization Problem QP-s we formulate a Lagrangian having all the previously emphasized regularization, where the quadratic cost is denoted as $f(x)$,

$$\begin{aligned}
\mathcal{L}_{PPA}(x,\lambda,\mu,s) \coloneqq\ & f(x) + \theta_\lambda^\top (Ax-b) + \theta_\mu^\top (Gx-d+s) \\
& + \frac{1}{2\delta_\lambda}\|Ax-b\|_2^2 + \frac{1}{2\delta_\lambda}\|Ax-b-\delta_\lambda(\lambda-\theta_\lambda)\|_2^2 \\
& + \frac{1}{2\delta_\mu}\|Gx-d+s\|_2^2 + \frac{1}{2\delta_\mu}\|Gx-d+s-\delta_\mu(\mu-\theta_\mu)\|_2^2 \\
& + \frac{\delta_x}{2}\|\theta_x - x\|_2^2 + \frac{\delta_s}{2}\|\theta_s - s\|_2^2 \\
& + IND(s)
\end{aligned} \tag{1.132}$$

where the indicator function enforces the linear inequality constraints exactly, as introduced in Definition 1.8.1, together with its subdifferential being the slack-variable normal cone $NC_{s\geq 0}$ defined in Definition 1.8.2 (see Figure 1.4). Using this formulation, the KKT conditions can be expressed as follows

$$0 \ni \begin{bmatrix} \partial_x \mathcal{L}_{PPA} \\ \\ \partial_\lambda \mathcal{L}_{PPA} \\ \partial_\mu \mathcal{L}_{PPA} \\ \partial_s \mathcal{L}_{PPA} \end{bmatrix} = \begin{bmatrix} Qx + c + A^\top\theta_\lambda + \frac{1}{\delta_\lambda}A^\top(Ax-b) + \frac{1}{\delta_\lambda}A^\top(Ax-b-\delta_\lambda(\lambda-\theta_\lambda)) \\ +G^\top\theta_\mu + \frac{1}{\delta_\mu}G^\top(Gx-d+s) + \frac{1}{\delta_\mu}G^\top(Gx-d+s-\delta_\mu(\mu-\theta_\mu)) \\ +\delta_x(x-\theta_x) + \delta_s(s-\theta_s) \\ -(Ax-b-\delta_\lambda(\lambda-\theta_\lambda)) \\ -(Gx-d+s-\delta_\mu(\mu-\theta_\mu)) \\ NC_{s\geq 0}(s) + \mu \end{bmatrix},$$

where $\partial_\square$ a subdifferential operator being consistent with the use of the normal cone. Using the λ,μ substitutions (1.66), (1.68) we can write,

$$\begin{bmatrix} \partial_x \mathcal{L}_{PPA} \\ \partial_\lambda \mathcal{L}_{PPA} \\ \partial_\mu \mathcal{L}_{PPA} \\ \partial_s \mathcal{L}_{PPA} \end{bmatrix} = \begin{bmatrix} Qx + c + A^\top\lambda + G^\top\mu \\ -(Ax-b) \\ -(Gx-d+s) \\ NC_{s\geq 0}(s) + \mu \end{bmatrix} + \Sigma \begin{bmatrix} x-\theta_x \\ \lambda-\theta_\lambda \\ \mu-\theta_\mu \\ s-\theta_s \end{bmatrix}, \tag{1.133}$$

with $\Sigma = diag(\delta_x, \delta_\lambda, \delta_\mu, \delta_s)$ and according to Luques [89], as the optimization problem can be written as a PPA the outerloop converges. It can be noted, that without the slack variable proximal regularization, the problem cannot be cast in the PPA form with all its variables, and convergence of the outer loop would no longer be guaranteed in this form.

In practice, the indicator function $\text{IND}(s)$ is relaxed by a smooth log-barrier relaxation, $\sigma\nu\phi(s)$, with $\phi(s) = -\sum_{i=1}^{p} \ln(s_i), \quad \nabla\phi(s) = -s^{-1}$,

where s^{-1} denotes the elementwise inverse of s. The parameter $\nu > 0$ determines the barrier strength, while $\sigma \in (0, 1]$ is a centering factor. This substitution maintains feasibility ($s > 0$) while introducing a smooth approximation of the non-smooth $IND(s)$.

When $IND(s)$ is replaced by $\sigma\nu\phi(s)$, the outerloop algorithm no longer represents an exact PPA of the form $0 \in T(z) + \Sigma(z - \theta_z)$. Instead, the process can be viewed as an inexact Proximal Point Algorithm (iPPA), where the deviation acts as a structured approximation error controlled by the barrier parameter ν, with $\sum_k \nu_k < \infty$, see [117]. As $\nu \to 0$, the barrier term approaches the indicator function, and the scheme tends to the exact PPA form. The work by Solodov and Svaiter [117] provides a convergence framework when the iterates satisfy $0 \in T(z^{k+1}) + \Sigma(z^{k+1} - z^k) + e^k$, with e^k denoting a residual error that is either summable or bounded relatively to $\|z^{k+1} - z^k\|$. Under suitable control of ν and σ, the log-barrier formulation can thus be interpreted as an inexact PPA, ensuring asymptotic convergence of the outerloop.

1.8.2 Innerloop - a regularized Newton method

Consider the innerloop of a primal-dual Algorithm (4), where a merit function $\phi(z)$ is minimized along a search direction Δ. We assume the following,

Assumptions 1.8.1 — **Innerloop convergence**

(a) The linear system (1.81) is always solvable.

(b) The search direction Δ exists and is unique.

(c) There exists a unique optimal step size $\alpha^* > 0$ along Δ that minimizes $\phi(z + \alpha\Delta z)$ [24].

(d) The merit function ϕ is continuously differentiable and strictly convex.

Under these assumptions, we can prove convergence of the inner loop using an Armijo backtracking linesearch.

In the upcoming study, the merit function $\phi(z) = \mathcal{M}_\delta^{GR}(x, s, \mu, \lambda; \theta_\mu, \theta_\lambda, \theta_x, \theta_s)$ and $z = [x, \lambda, \mu, s]^\top$. The merit function is strictly convex due to the proximal and augmented Lagrangian regularization. The associated Newton scheme is always solvable due to its KKT matrix properties, see (1.81), leading to a unique Δ descent direction.

By strict convexity and differentiability of ϕ, the directional derivative along Δx exists and is unique

$$\langle \nabla\phi^\top \cdot \Delta \rangle < 0, \text{ see Proof (1.5.1)}, \tag{1.134}$$

thus, Δ is a descent direction unless $z = z^*$ is already the unique minimizer.

The Armijo condition requires a step size α satisfying,

$$\phi(z_k + \alpha\Delta z_k) \leq \phi(z_k) + c\alpha\nabla\phi(z_k)^\top\Delta z_k, \quad 0 < c < 1, \tag{1.135}$$

and there exists a sequence of non-zero step sizes $(\alpha_{k_j})_{j\in\mathbb{N}}$ satisfying the Armijo condition (1.32) at each iteration. This ensures a decrease in ϕ along the Newton direction unless $\nabla\phi(z_{k_j}) = 0$, see Proof 1.4.1.

Since ϕ is strictly convex, the function $\alpha \mapsto \phi(x_k + \alpha\Delta z_k)$ is strictly convex. By assumption 1.8.1[c], there exists a unique $\alpha^* > 0$ minimizing ϕ along Δx_k, and the Armijo condition is satisfied for some $\alpha \leq \alpha^*$.

Each iteration produces

$$\phi(z_{k+1}) = \phi(z_k + \alpha_k\Delta x_k) \leq \phi(z_k) + c\alpha_k\nabla\phi(z_k)^\top\Delta z_k < \phi(z_k), \tag{1.136}$$

so the sequence $\{\phi(x_k)\}$ is strictly decreasing.

Being strictly convex, ϕ has a unique minimizer z^* and $\phi^* = \phi(z^*)$, which bounds the sequence from below. Hence,

$$\phi(z_k) \to \phi^*, \qquad \nabla\phi(z_k) \to 0, \quad z_k \to z^*. \tag{1.137}$$

Under the aforementioned assumptions, the innerloop using a regularized Newton method with Armijo backtracking converges globally to the unique minimizer of ϕ of the defined sub-QP$(\delta_\square, \theta_\square)$, providing a numerically robust framework for the primal-dual algorithm.

1.9 Modularity and object oriented programming

Modular architecture: This section outlines the solver's design, emphasizing how modularity is achieved through object-oriented programming principles. Each major component—such as the KKT solver, linesearch strategy, and termination criterion—is encapsulated within its own class. Components implement common interfaces or inherit from abstract base classes, enabling seamless interchangeability without modifying other parts of the solver.

To further enhance modularity, the solver employs a factory pattern. This allows users to select among different strategies, for example, a dense or sparse KKT solver, at runtime without the need for recompilation.

The architecture promotes code reuse and simplifies future development. Developers can introduce a new solver backend or implement a custom stopping condition by extending the relevant base class and registering the new component through the factory. This design ensures the solver remains highly adaptable, maintainable, and extensible.

Figure 1.5 shows the architecture of the object oriented program. The main routine is composed of 3 sequential blocks, the construct of the KKT scheme, the linesearch and the

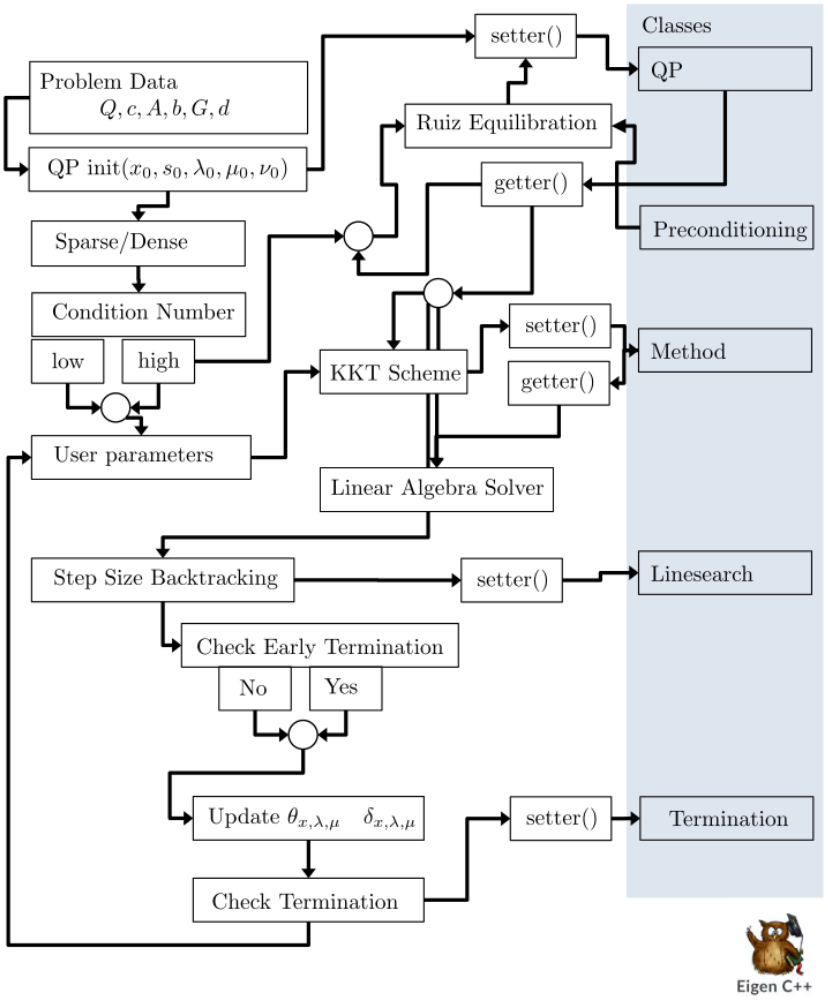


FIGURE 1.5: Modular architecture and flowchart.

termination. These 3 blocks are customizable in terms of regularization, safeguards and are easy to get in for machine learning tools using object instances for each configuration. An automatical preconditioning method can be activated, and the core structure is made using the *Eigen* package, having methods for dense and sparse matrices. The results presented in the next section were made using the sparse version.

The solver is designed in programming language *C++* for efficiency reasons and flexibility of development. Using CMake for modern build and package management, the system can be easily installed and compiled across platforms. The main solver class maintains references to user-selected modules, including different linesearch conditions, KKT schemes and termination conditions. This architecture minimizes coupling, allowing components, such as other and new backtracking methods or filters to be replaced without modifying the solver core. Class composition is implemented via smart-pointers, ensuring computational efficiency and good memory management. A lightweight Python library with a C++ wrapper further enables integration with other software platforms, facilitating scripting and experimentation.

Linear algebra: The solver is integrated with multiple open-source and academic linear algebra packages. For the LU decomposition of non-symmetric KKT matrices, users can

choose between the *UMFPACK* package for optimized performance, *SuperLU* for a simple Eigen-based solution without additional dependencies, or *MUMPS* when CPU parallelization is required. For the $\mathrm{LDL}^{\mathrm{T}}$ decomposition, employed in the Nesterov-Todd scaling scheme (see §(1.93)), the most competitive option is the *MA97* solver, which is available free of charge only for academic use. To this end, we provide a competitive Eigen-based implementation of MA97 for sparse matrices that minimizes unnecessary data copies—constituting, as no equivalent implementation currently exists within the Eigen ecosystem. The Eigen library also includes an integrated *LDLT-Cholesky* routine, while the *CHOLMOD* linear algebra solver was also integrated. However, *CHOLMOD* can only be applied when the KKT matrix, and not merely Q, is guaranteed to be positive definite.

1.10 Results

In all presented results, the standard `QProxIP` solver is depicted in red for clarity and consistency when compared against other solvers and parameter configurations.

1.10.1 Monitoring and metrics

To evaluate the presented results, it is necessary to define appropriate metrics and monitoring criteria. These metrics provide a consistent basis for comparing solver performance, tracking convergence behavior, and assessing numerical stability across different problem instances and parameter configurations.

Profiles: For P the set of problems and S the set of solvers, let $t_{s,p}$ denote the (wall-clock) runtime required by solver $s \in S$ to solve instance $p \in P$ (for a given tolerance $\epsilon > 0$). We will monitor the computational performance of different solver settings and graphically summarize our results by means of two profiles. When solver $s \in S$ fails to solve $p \in P$, we set $t_{s,p} = \infty$.

Definition 1.10.1 — **Data Profile**

A *data profile* is the graph of the cumulative distribution function $f_s \colon [0, \infty) \to [0, 1]$ of the runtime, namely $f_s(t) := \frac{|\{p \in P \,|\, t_{s,p} \leq t\}|}{|P|}$, where $|P|$ denotes the cardinality of set P. As such, each data profile reports the fraction of problems $f_s(t)$ that can be solved by s with a computational budget t, and therefore it is independent of the other solvers.

Definition 1.10.2 — **Performance Profile**

A *performance profile* is the graph of the cumulative distribution function of the performance ratio $r_{s,p} := \frac{t_{s,p}}{\min_{s' \in S} t_{s',p}}$. As such, it displays the fraction of problems solved within a time factor compared to the fastest solver of each problem. Therefore, the performance profile of a solver also depends on the performance of all other solvers in S.

1.10.2 Maros-Mészáros test-set

Throughout the results, the `QProxIP` solver is consistently represented in red when compared to other solvers and configurations. The benchmark problems, detailed in the Annex, are ordered according to their storage size (in bits per problem). For example, the subset denoted as `MM100` refers to the reference test set comprising 100 representative problems and `MM139` is the full test set.

1.10.2.1 Test set characteristics

The Maros–Mészáros test set is a widely used benchmark collection of convex quadratic programming (QP) and linear programming (LP) problems. It was originally designed to evaluate the robustness, efficiency, and numerical stability of optimization solvers across a broad spectrum of problem characteristics. The diversity of the test set makes it a standard reference in the evaluation of both academic and industrial QP solvers.

The collection comprises 139 problem instances of varying sizes and structures. The number of variables and constraints ranges from a few hundred up to several tens of thousands. The constraint matrices are typically sparse, with a low density of nonzero elements, which poses challenges for storage efficiency and matrix factorization. Many problems are also ill-conditioned, and some include nearly singular or degenerate constraint structures. In several instances, the Hessian matrix is only positive semidefinite or close to singular, which tests the numerical robustness of solvers under challenging conditions. These properties make the Maros–Mészáros test set particularly suitable for assessing solver performance beyond average behavior, including worst-case convergence, sensitivity to conditioning, and scalability. Solvers that are able to successfully handle all or nearly all instances in this benchmark, under tight tolerances (e.g., 10^{-6}), are generally considered to exhibit a high level of robustness and numerical reliability.

1.10.2.2 Proximal and Augmented Lagrangian weight geometric decrease influence

Depending on the dynamics of the parameter $\alpha_{\square}$, the convergence speed of the solver varies, as illustrated in the performance profile. Nevertheless, the solver remains robust even when the augmented Lagrangian proximal weights are not decreased. Since parameter tuning can be a tedious task for the user, it is desirable that the solver demonstrate robustness across a wide range of regularization settings.

In the following studies, the solver is configured with the regularization and safeguard settings summarized in Table 1.1. All options listed in the table are enabled or applied as indicated, ensuring a consistent solver setup across the experiments in this subsection.

ID	ALM	x-prox	s-prox	CPC	NML-LS	Ruiz	Amort.	Bound. Ctrl
All	yes	yes	no	yes	yes	yes	yes	yes

TABLE 1.1: Applied regularization and safeguards.

$\alpha_\square$ **influence:** In the first experiment, all proximal weights share the same initial value $\delta_\square = 1$, while we vary their geometric decrease coefficient $\alpha_\square$. The solver settings used in this study are summarized in Table 1.2. The corresponding results are presented in the performance profile shown in Figure 1.6.

ID	δ_x	δ_λ	δ_μ	$\overline{\delta_x}$	$\overline{\delta_\lambda}$	$\overline{\delta_\mu}$	α_{δ_x}	α_{δ_λ}	α_{δ_μ}
QProxIP	1	1	1	10^{-12}	10^{-12}	10^{-12}	0.15	0.2	0.2
QProxIP 0.10	1	1	1	10^{-12}	10^{-12}	10^{-12}	0.1	0.1	0.1
QProxIP 0.25	1	1	1	10^{-12}	10^{-12}	10^{-12}	0.25	0.25	0.25
QProxIP 0.50	1	1	1	10^{-12}	10^{-12}	10^{-12}	0.5	0.5	0.5
QProxIP 1e-9	10^{-9}	10^{-9}	10^{-9}	10^{-12}	10^{-12}	10^{-12}	1	1	1

TABLE 1.2: Augmented Lagrangian and proximal weights applied and their geometric decrease, see Figure 1.6.

The performance profile compares five QProxIP solver configurations as shown in Table 1.2.

The default setting and constant augmented Lagrangian and proximal weight $\delta_\square = 10^{-9}$ settings demonstrate superior performance, solving approximately between 60% and 80% of problems twice as fast as the solver with other settings. The intermediate settings, 0.1 and 0.25, achieve comparable robustness but are slower. In contrast, the 0.5 setting significantly underperforms in terms of robustness and solving time at convergence. The results indicate that, although the convergence speed varies depending on the dynamics of the parameter α, the solver remains robust: even under slower or suboptimal settings, it successfully solves all or more than 90% of the problems.

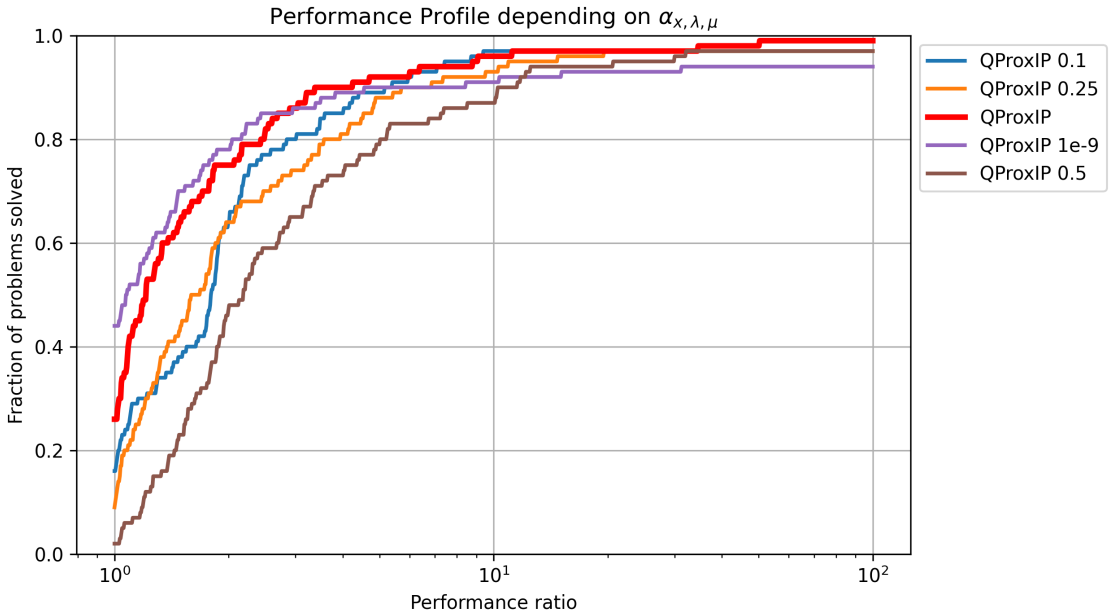


FIGURE 1.6: Performance profile of QProxIP depending on $\alpha_\square$ - MM100 subtestset.

$\delta_\square$ **influence:** In a second experiment, we modify the scaling of the initial augmented Lagrangian and proximal weights $\delta_\square$, decreasing at a geometric scale $\alpha_\square = 0.25$, further illustrating that the solver maintains stable performance over a wide range of regularization

settings, thereby reducing the need for extensive manual tuning. The settings presented in Figure 1.7 are presented in Table 1.3.

ID	δ_x	δ_λ	δ_μ	$\overline{\delta_x}$	$\overline{\delta_\lambda}$	$\overline{\delta_\mu}$	α_{δ_x}	α_{δ_λ}	α_{δ_μ}
QProxIP	1	1	1	10^{-12}	10^{-12}	10^{-12}	0.15	0.2	0.2
QProxIP 1e0	1	1	1	10^{-12}	10^{-12}	10^{-12}	0.25	0.25	0.25
QProxIP 1e-1	0.1	0.1	0.1	10^{-12}	10^{-12}	10^{-12}	0.25	0.25	0.25
QProxIP 1e-2	0.01	0.01	0.01	10^{-12}	10^{-12}	10^{-12}	0.25	0.25	0.25
QProxIP 1e-3	0.001	0.001	0.001	10^{-12}	10^{-12}	10^{-12}	0.25	0.25	0.25

TABLE 1.3: Augmented Lagrangian and proximal weights applied and their geometric decrease

The performance profile compares five solver configurations with initial penalty parameters $\delta_{\square,0}$. The setting $\delta_{\square,0} = 10^{-2}$ achieves the best performance, solving about 65% of problems at the fastest time. The default, 10^{-3} settings show intermediate performance, solving 50–60% of problems optimally and 95–97% within a factor of 10. In contrast, the 10^{-1} and 10^{0} settings underperform, solving fewer problems optimally and requiring significantly more time, indicating that larger initial penalty values slow convergence. Overall, $\delta = 10^{-2}$ provides the best balance for general use.

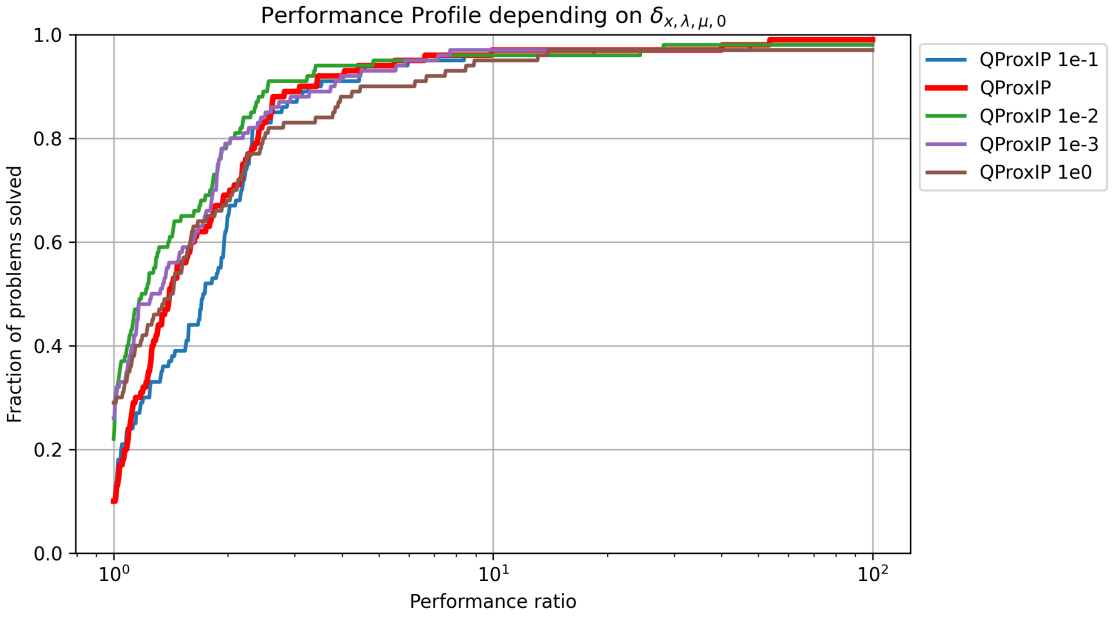


FIGURE 1.7: Performance profile of QProxIP depending on $\delta_{\square,0}$ - MM100 sub-testset

The results show that, over a wide range of solver settings and various combinations of safeguards, the QProxIP solver remains robust across the Maros–Mészáros 100 (MM100) test set. From the perspective of the user, this robustness has important implications: it allows the tuning process to be carried out coarsely without compromising reliability, and it ensures that the solver performs consistently across different problem instances. At the same time, achieving optimal performance requires careful consideration of specific metrics; strategies for effectively leveraging these metrics will be discussed in detail in Chapter 2.

ID	ALM	x-prox	s-prox	CPC	NML-LS	Ruiz	Amort.	Bound. Ctrl
`QProxIP`	yes	yes	yes	yes	yes	no	yes	yes

TABLE 1.4: Applied regularization and safeguards

1.10.2.3 S-prox regularization influence

Figure 1.8 compares the performance of `QProxIP` with and without the s-prox regularization on the MM50 test set without Ruiz equilibration (preconditioning). Table 1.4 summarizes the regularization techniques and safeguard mechanisms implemented in `QProxIP`.

For smaller problems solved within 10^{-1} seconds, the s-prox regularization yields comparable or slightly improved results. However, a noticeable loss of robustness appears for larger problems, which may stem from insufficient parameter tuning of the s-prox framework.

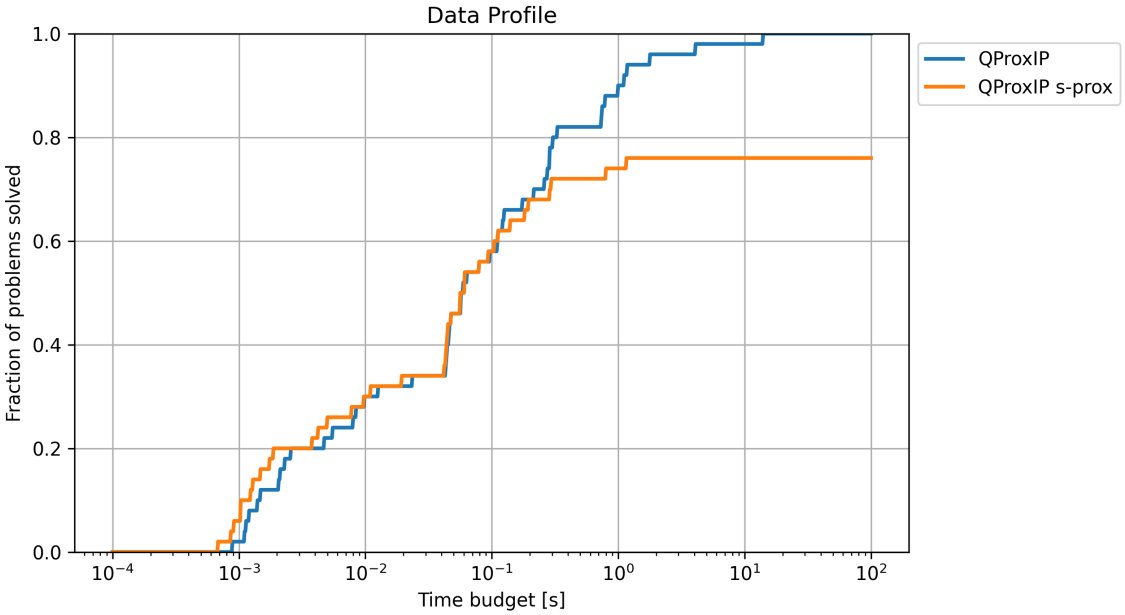


FIGURE 1.8: Data profile of `QProxIP` with and without the s-prox regularization for MM50 without Ruiz equilibration (preconditioning).

1.10.2.4 Benchmarking reference solvers

To assess the performance of `QProxIP`, we compare it against several widely used quadratic programming solvers representing distinct algorithmic paradigms.

PIQP is a modern primal-dual interior-point solver designed specifically for quadratic programs. It employs a Mehrotra predictor-corrector strategy to achieve high numerical robustness and efficiency, particularly for high termination tolerance, see [114]. The **OSQP** solver implements the alternating direction method of multipliers (ADMM) for convex quadratic programs. It is well suited for large, sparse problems, offering predictable performance and simple warm-start capabilities, though it may require more iterations to reach high-accuracy solutions, see [119]. **SCS** (Splitting Conic Solver) is a first-order operator-splitting method for solving large-scale conic problems, including QPs expressed in conic form. It sacrifices high-accuracy optimality for scalability and robustness, making it a reliable baseline for very

large or ill-conditioned problems, [97]. **ProxQP** is a proximal-based quadratic programming solver leveraging iterative refinement and proximal regularization. It provides robust convergence behavior and predictable numerical performance, particularly suitable for embedded or control-oriented applications, [11]. **Gurobi** is a state-of-the-art commercial optimization solver implementing advanced barrier, simplex, and mixed-integer algorithms. It is known for its exceptional performance, extensive presolving, and highly optimized parallelization, providing reference-level accuracy and speed. **Mosek** is another leading commercial solver that focuses on large-scale convex optimization with no fixed limits on the number of variables and constraints, using a highly efficient primal-dual interior-point method. It provides strong numerical stability, high-precision solutions, and robust handling of conic and quadratic problems.

The solver `QProxIP` have the following hyperparameters:

Parameter	Value	Eq. ref. see
ϵ	10^{-6}	(1.49)
κ_ε	0.5	(1.74)
α_ε	0.98	(1.74)
ξ	10^{-2}	(1.75)
κ_ν	0.2	(1.52)
θ_ν	1.5	(1.52)
$\delta_{\min}$	10^{-12}	(1.73)
$\delta_{x,0}$	1	(1.73)
$\delta_{y,0}$	10	(1.73)
$\delta_{z,0}$	10	(1.73)
η_{cv}	0.75	(1.72)
max-outer-it	25	–
max-inner-it	100	–

TABLE 1.5: Fixed parameters for the `QProxIP` solver.

And have the following regularization and safeguards,

ID	ALM	x-prox	s-prox	CPC	NML-LS	Ruiz	Amort.	Bound. Ctrl
`QProxIP`	yes	yes	no	yes	yes	yes	yes	yes

TABLE 1.6: Augmented Lagrangian and proximal weights applied and their geometric decrease.

The author would like to emphasize that the same solver configuration is used for all problem instances and all benchmarked solvers are initialized at the point $x = 0$. No problem-specific adaptation or tuning is performed based on characteristics such as size or structure, ensuring that the integrity of the performance profiles is fully preserved.

Figure 1.9 illustrates the benchmarking results, which show the data profile for the full MM139 Maros-Mészáros test set against previously listed solvers. The optimization tolerances are set to $\varepsilon = 10^{-8}$ and $\varepsilon_{\text{rel}} = 10^{-9}$ for all solvers to ensure a fair and consistent comparison. The full results table for `QProxIP` is provided in the appendix. For the other solvers, benchmarks were generated using the PIQP benchmark routine [114] and run locally on the same machine, but these results are omitted from the appendix for the sake of brevity.

`QProxIP` ranks among the most robust QP-solvers, successfully converging on 137 out of 139 problems within the time limit of 1000 seconds. Among the remaining two cases, one

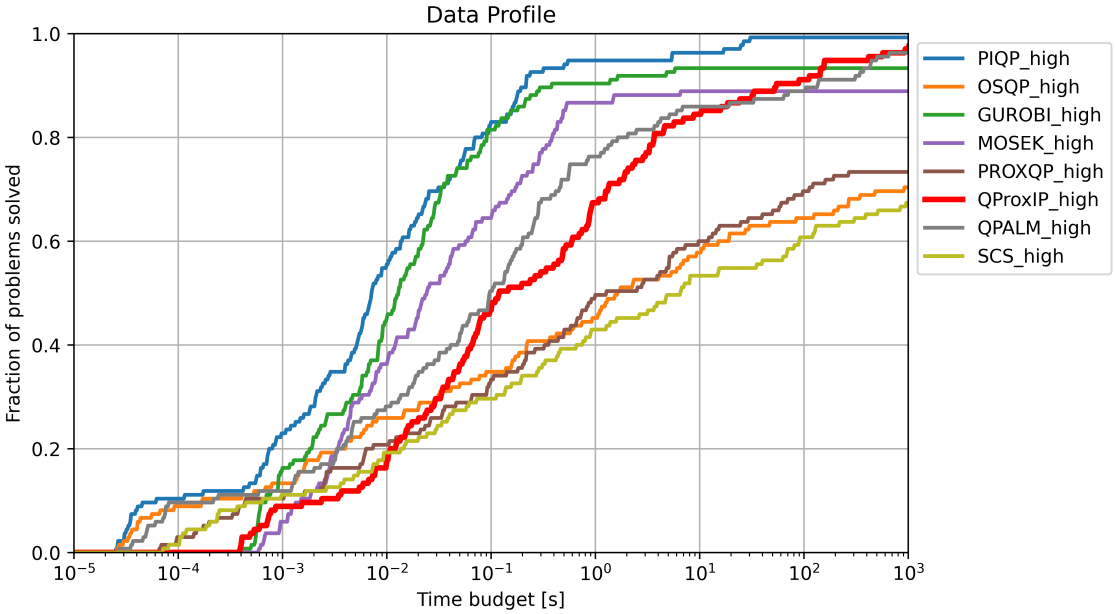


FIGURE 1.9: Maros–Mészáros data profile, benchmarking of reference QP-solvers on the full test-set MM139.

problem converges but at a slower rate, while the other exhibits stagnation beyond a tolerance of 10^{-3}.

For problems solved within the millisecond range, the solver is less competitive compared to the fastest state-of-the-art methods but remains within the same order of magnitude as most established solvers.

Beyond 10 milliseconds, first-order methods such as OSQP and SCS begin to struggle due to the stringent optimization tolerances, as does PROX-QP, which is primarily designed for real-time robotic control.

For larger-scale problems requiring computation times on the order of seconds (typically involving around $10^3 - 10^6$ constraints and variables) `QProxIP` becomes highly competitive, achieving performance comparable to QPALM. Finally, for the most challenging instances, `QProxIP` remains within one order of magnitude of the fastest solver, PIQP.

All problems are included in the plot, however, `EXDATA`, `BOYD1`, and `BOYD2` could not be loaded using Eigen and are therefore marked as failed.

All results are provided in the annexes (see Appendix II).

1.10.3 Performance Variability in Computational Benchmarks

Benchmarking provides useful metrics to gauge the performance level of a solver across different settings, but it should not be interpreted as a universal truth. As demonstrated, solver performance can vary significantly depending on benchmarking configuration, system environment, and concurrent computational load. Therefore, results should always be reported and interpreted in the context of the specific experimental conditions.

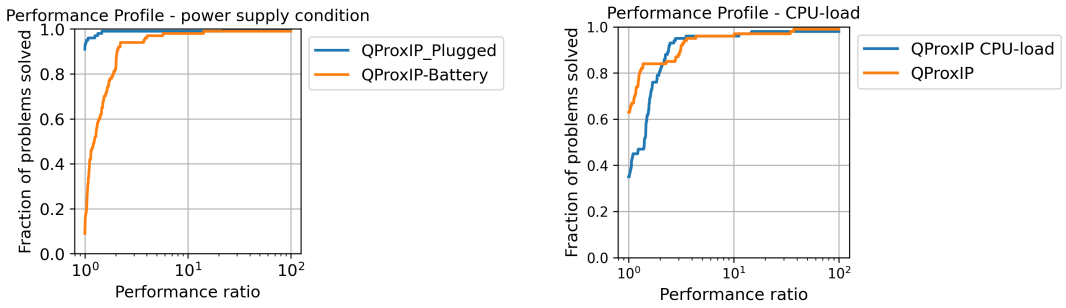


FIGURE 1.10: Performance profile of `QProxIP` depending on the power supply (left) and CPU overload (right).

Measured computational performance is inherently variable, influenced by both hardware and software factors. Background processes, operating system scheduling, and transient system states introduce minor inconsistencies even when identical problems are executed repeatedly. Cache utilization, memory access patterns, and thread scheduling can also contribute to measurable deviations in runtime. Power management features affect performance, with low-power or laptop modes typically yielding slower runtimes than high-performance settings.

System load has a significant impact on reproducibility. Background services, browser activity, or system updates consume CPU cycles and memory bandwidth, while large-scale problems may suffer from limited RAM bandwidth. The software environment introduces additional variability: differences in compiler optimization levels, numerical libraries (e.g., BLAS or LAPACK), or thread management can meaningfully shift results. Even small changes, such as using `-O2` instead of `-O3` arguments when compiling.

Figures 1.10 illustrate these effects. The figure on the left shows the performance profile of `QProxIP` under different power supply conditions (battery vs plugged), while the figure on the right shows the effect of CPU overload, where an additional program uses 12 threads to repeatedly invert random matrices, simulating high computational load.

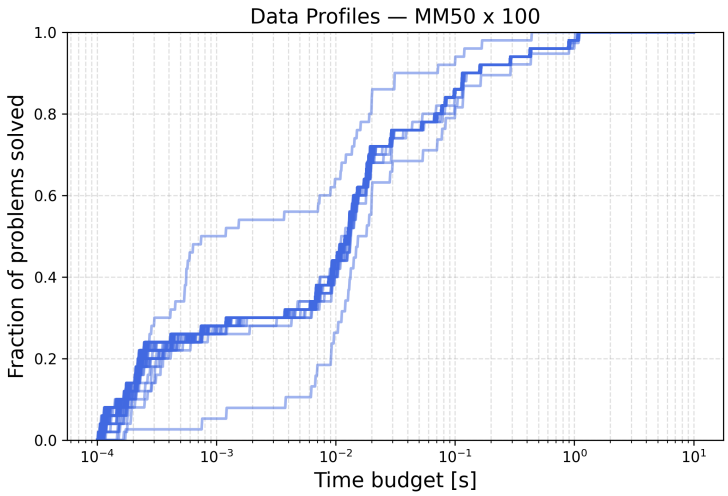


FIGURE 1.11: Maros–Mészáros data profile on the test-set MM50, solved 100 times in a row.

Figure 1.11 shows the data profiles of `QProxIP` evaluated over 100 independent batches of the MM50 sub-test-set. Each blue curve corresponds to one batch. Overall, the profiles

exhibit a consistent trend across batches, suggesting stable solver performance over different test sets. All batches achieve full success within roughly 10^{-1} to 10^{0} seconds, while the spread at smaller time scales (10^{-4}–10^{-2} s) reveals variability in the convergence speed for simpler instances. The overlapping blue curves indicate low dispersion, confirming that the solver maintains reliable computational efficiency and robustness across diverse problem realizations. However, two batches stand out from the mass: one performing significantly better and one considerably worse, with runtime differences of approximately one order of magnitude compared to the majority. This deviation likely reflects variability in execution time due to uncontrolled background system activity during the experiments, despite no user interaction and the wireless network being disabled throughout the runs.

1.11 Applications and Study Case

1.11.1 Vehicle Routing Problem, a continuous-based optimal transport approach

We consider a resource allocation problem in which a set of mobile agents (e.g., drones, ground vehicles, robots) stationed at multiple bases must be assigned to a number of target locations. The assignment is subject to spatial exclusion zones, which impose penalties based on the geometric intersection of each path with circular obstacles. This problem is formulated as a structured Quadratic Program (QP) with linear equality constraints representing supply and demand and a diagonal cost matrix incorporating distance and obstacle penalties. Let there be n_b bases and n_t targets, indexed by $i \in \{1, \dots, n_b\}$ and $j \in \{1, \dots, n_t\}$. Define binary assignment variables $x_{ij} \in \{0, 1\}$ to indicate whether base i is assigned to target j. Each target must be covered by exactly one base, while each base may serve multiple targets.

The subsequent quadratic program is formulated as,

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^N} \quad & \frac{1}{2}\mathbf{x}^\top \mathbf{Q}\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}_{\text{supply}}\mathbf{x} = \mathbf{1}_{n_b}, \\ & \mathbf{A}_{\text{demand}}\mathbf{x} = \mathbf{1}_{n_t}, \\ & \mathbf{x} \in [0, 1]^N. \end{aligned} \tag{1.138}$$

Having a decision variable vector is formulated as,

$$\mathbf{x} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n_t} & x_{21} & \dots & x_{n_b n_t} \end{bmatrix}^\top \in \mathbb{R}^N. \tag{1.139}$$

Let $\mathbf{D} \in \mathbb{R}^{n_b \times n_t}$ denote the Euclidean distance matrix between bases and targets, and $\mathbf{C} \in \mathbb{R}^{n_b \times n_t}$ denote the matrix of obstacle-induced penalties. The total cost per base-target path is $Q_{ij} = D_{ij} + \lambda C_{ij}$, where $\lambda \geq 0$ weights the collision penalty.

Each base provides one unit of supply, and the corresponding supply constraint is defined as follows,

$$\sum_{j=1}^{n_t} x_{ij} = 1, \quad i = 1, \dots, n_b. \tag{1.140}$$

Equivalently, $\mathbf{A}_{\text{supply}} x = b_{supply}$ with $\mathbf{A}_{\text{supply}} \in \mathbb{R}^{n_b \times N}$,

$$(\mathbf{A}_{\text{supply}})_{i,k} = \begin{cases} 1, & \text{if } k = (i-1)n_t + j \text{ for some } j, \\ 0, & \text{otherwise.} \end{cases} \tag{1.141}$$

Each target must be served by agents and the demand constraint is expressed as,

$$\begin{aligned} &\sum_{i=1}^{n_b} x_{ij} = 1, \quad j = 1, \ldots, n_t, \\ &\sum_{i=1}^{n_b} \alpha_i = n_b. \end{aligned} \tag{1.142}$$

Equivalently, $A_{\text{demand}} x = b_{demand}$, with $\mathbf{A}_{\text{demand}} \in \mathbb{R}^{n_t \times N}$,

$$\mathbf{A}_{\text{demand}} = \begin{bmatrix} \alpha_1 \mathbf{I}_{n_t} & \alpha_2 \mathbf{I}_{n_t} & \cdots & \alpha_N \mathbf{I}_{n_t} \end{bmatrix}, \tag{1.143}$$

where the weights α_i represents a priority factor for each target. The sum of all α's is normalized by one.

Each exclusion zone is modeled as a circle with center $z_k \in \mathbb{R}^2$ and radius r_k. For the line segment from base i to target j, denoted by $[p_i, q_j]$, the collision penalty is

$$C_{ij} = \sum_{k=1}^{n_z} \text{Length}\Big([p_i, q_j] \cap B(z_k, r_k)\Big), \tag{1.144}$$

where $B(z_k, r_k)$ is the disk corresponding to the k-th exclusion zone. Intersection lengths are computed analytically from the quadratic form of the line-circle intersection.

Table 1.7 presents the problem settings, and figure 1.12 shows in on a map as well as the optimization results. The studied case shows 3 bases and 2$targets$ with 3 avoidances zones in between them. Target 1 is prioritized over target 2 and $\alpha_1 + \alpha_2 = n_b$.

TABLE 1.7: Vehicle Routing parameters.

Parameter	**Value**
λ	1
n_b	3
n_t	2
n_z	3
α_1	1.75
α_2	1.25

Figure 1.12 presents the solution of the aforementioned use case, and each base deploys all its available agents, as indicated by the sum of each row in Figure 1.12. The demand constraint is satisfied since $1.23 + 1.77 = 3$, and the prioritization accuracy is within 1% (with 1.75 and 1.25 proposed by the user). The optimization converged with a termination tolerance of 10^{-8}.

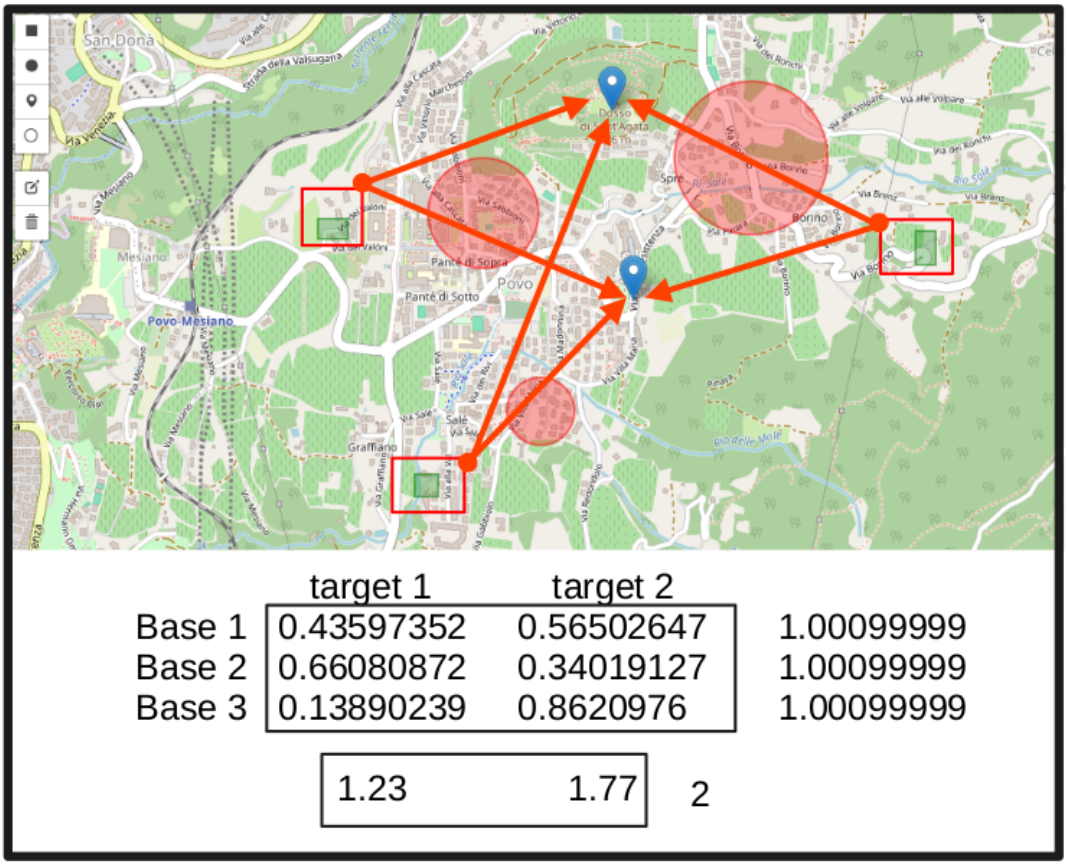


FIGURE 1.12: Routing with 3 bases, 2 targets and 3 obstacles

1.11.2 Stability inside a Tokamak

We consider a minimal two-dimensional dynamical system described by the state vector $\mathbf{x} = [x_1, x_2]^\top \in \mathbb{R}^2$. The system's evolution is governed by energy minimization, where at each discrete time step t, the state evolves toward a configuration that minimizes the system's quadratic energy subject to physical constraints.

The potential energy perturbation around equilibrium is defined as,

$$\Delta E(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{Q}\mathbf{x} = \frac{1}{2}kx_1^2 + \frac{1}{2}x_2^2 + d\,x_1x_2, \tag{1.145}$$

where the Hessian matrix $\mathbf{Q}$ captures the system's energetic structure:

$$\mathbf{Q} = \begin{bmatrix} k & d \\ d & 1 \end{bmatrix}. \tag{1.146}$$

Here, $k \in \mathbb{R}$ represents the self-energy coefficient of the first state variable, while $d \in \mathbb{R}$ quantifies the coupling strength between the two states. The second state variable is assigned unit self-energy without loss of generality. For a simplified plasma mode analysis,x_1, x_2 denote the amplitudes of two modes (e.g. the kick and ballooning modes in tokamak plasmas). The parameter k characterizes the magnetic field energy and mode coupling, serving as a measure of the stability margin relative to critical thresholds.

The stability of the system is determined by the definiteness properties of the Hessian $\mathbf{Q}$. We analyze this through the eigenvalue decomposition. The eigenvalues of $\mathbf{Q}$ satisfy the characteristic equation,

$$\det(\mathbf{Q} - \lambda \mathbf{I}) = \lambda^2 - (k+1)\lambda + (k - d^2) = 0, \tag{1.147}$$

yielding:

$$\lambda_{\pm} = \frac{(k+1) \pm \sqrt{(k+1)^2 - 4(k-d^2)}}{2} = \frac{(k+1) \pm \sqrt{(k-1)^2 + 4d^2}}{2}. \tag{1.148}$$

The system exhibits distinct dynamical regimes:

- **Stable Configuration** ($k > d^2$): Both eigenvalues are positive, $\lambda_{\pm} > 0$, rendering $\mathbf{Q}$ positive definite. The energy functional possesses a unique global minimum, and the system converges to an equilibrium state under perturbations. The basin of attraction is the entire feasible domain.
- **Unstable Configuration** ($k < d^2$): At least one eigenvalue is negative, typically $\lambda_- < 0$ for $k < 0$. The energy functional exhibits a saddle point structure, and perturbations grow exponentially along the unstable eigenmode direction. No bounded attractor exists.
- **Critical Configuration** ($k = d^2$): The system resides at the stability boundary, with $\lambda_- = 0$. This represents a degenerate case with neutral stability along one direction.

For the specific configurations examined:

$$\text{Stable:} \quad k = 2.0,\ d = 0.1 \quad \Rightarrow \quad \lambda_+ \approx 2.005,\ \lambda_- \approx 0.995, \tag{1.149}$$

$$\text{Unstable:} \quad k = -1.0,\ d = 0.1 \quad \Rightarrow \quad \lambda_+ \approx 1.005,\ \lambda_- \approx -1.005. \tag{1.150}$$

In the tokamak context, the stable configuration ($k > 0$) corresponds to operation above critical stability thresholds, when the safety factors are above the kick instability limit or operating below the ballooning mode β-limit—where magnetic field energy provides natural confinement and perturbations decay through resistive dissipation [133]. Conversely, the unstable configuration ($k < 0$) represents conditions where destabilizing pressure gradients or current-driven effects exceed stabilizing magnetic forces, analogous to exceeding operational limits where modes like kink instabilities or tearing modes grow exponentially, leading to rapid loss of plasma confinement [133].

At each time step t, the free-system evolves to it's minimal energy configuration, formulated as a quadratic program,

$$\mathbf{x}^{t+1} = \arg\min_{\mathbf{x}} \tfrac{1}{2}\mathbf{x}^\top \mathbf{Q}\mathbf{x} \quad \text{s.t.} \quad l \geq x \geq u, \tag{1.151}$$

where l, u imposes physical bounds such as amplitude saturation and material limits.

The system evolves in time according to the recursion:

$$\mathbf{x}^{t+1} = \alpha\, \tilde{\mathbf{x}}^{t+1} + (1-\alpha)\mathbf{x}^\top, \tag{1.152}$$

where $\tilde{\mathbf{x}}^{t+1}$ is the solution to the QP problem and $\alpha = 0.7$ is a relaxation parameter that ensures numerical stability. To assess the tokamak stability, we introduce a perturbation at time $t_{\text{pert}} = 30s$ as,

$$\mathbf{x}^{t_{\text{pert}}} \leftarrow \mathbf{x}^{t_{\text{pert}}} + \boldsymbol{\delta}, \quad \boldsymbol{\delta} = [2.0, 1.0]^\top. \tag{1.153}$$

Stable systems exhibit exponential decay: $\|\mathbf{x}^\top - \mathbf{x}^*\| \sim e^{-\gamma t}$ with $\gamma > 0$, where $\mathbf{x}^*$ is the equilibrium and the energy perturbation ΔE converges to zero. Unstable systems show exponential growth: $\|\mathbf{x}^\top\| \sim e^{\gamma t}$ with $\gamma > 0$ along unstable directions and the energy diverges, i.e. $|\Delta E|$ diverges.

The stability is quantified using the energy growth rate γ. Post-perturbation $[t_{\text{pert}}, T]$ γ is defined as $\gamma = \frac{1}{T - t_{\text{pert}}} \frac{d}{dt} \ln E(t) \approx \frac{\ln E(T) - \ln E(t_{\text{pert}})}{T - t_{\text{pert}}}$ and characterizes the net amplification or decay of perturbation energy. The system is classified as stable for $\gamma < -0.05$ and unstable for $\gamma > 0.05$.

TABLE 1.8: Tokamak Stability Parameters.

Parameter	Value
k_{stable}	2
k_{unstable}	-1.0
d_{stable}	0.1
d_{unstable}	0.1
Perturbation amplitude	2
l	-2.6
u	2.6

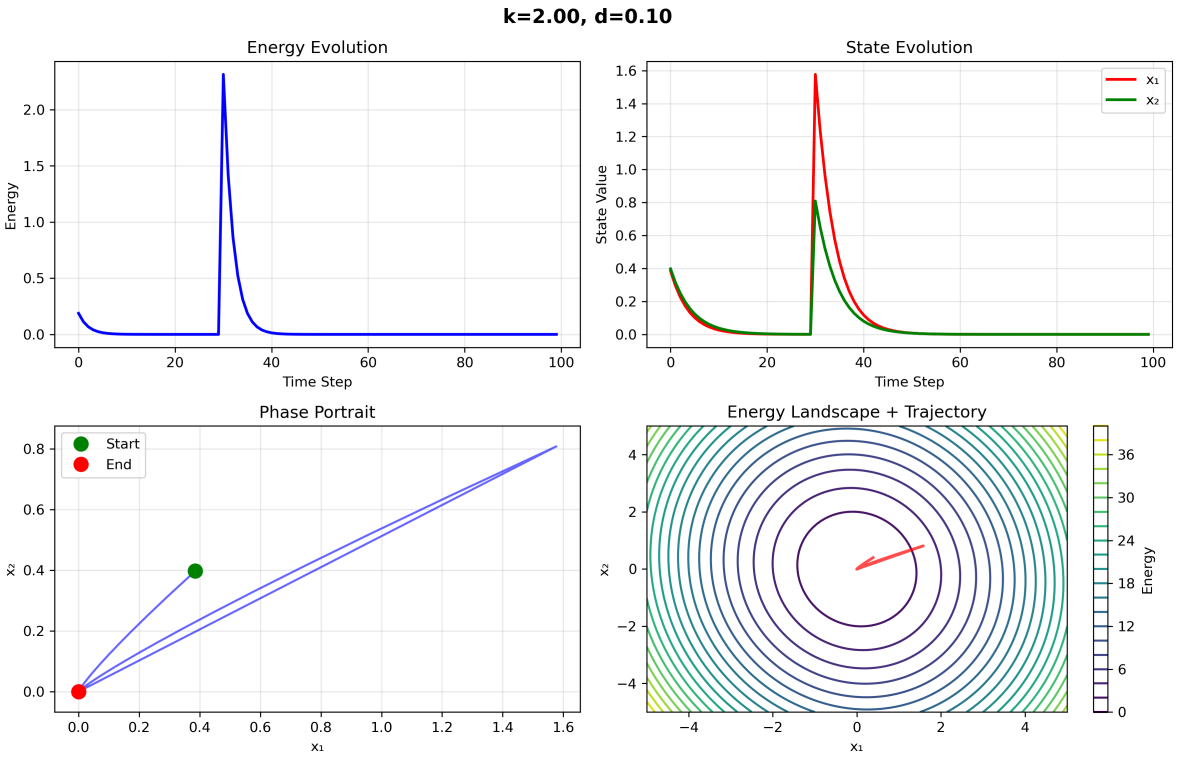


FIGURE 1.13: Stable tokamak configuration - Energy, States and Phase Portrait

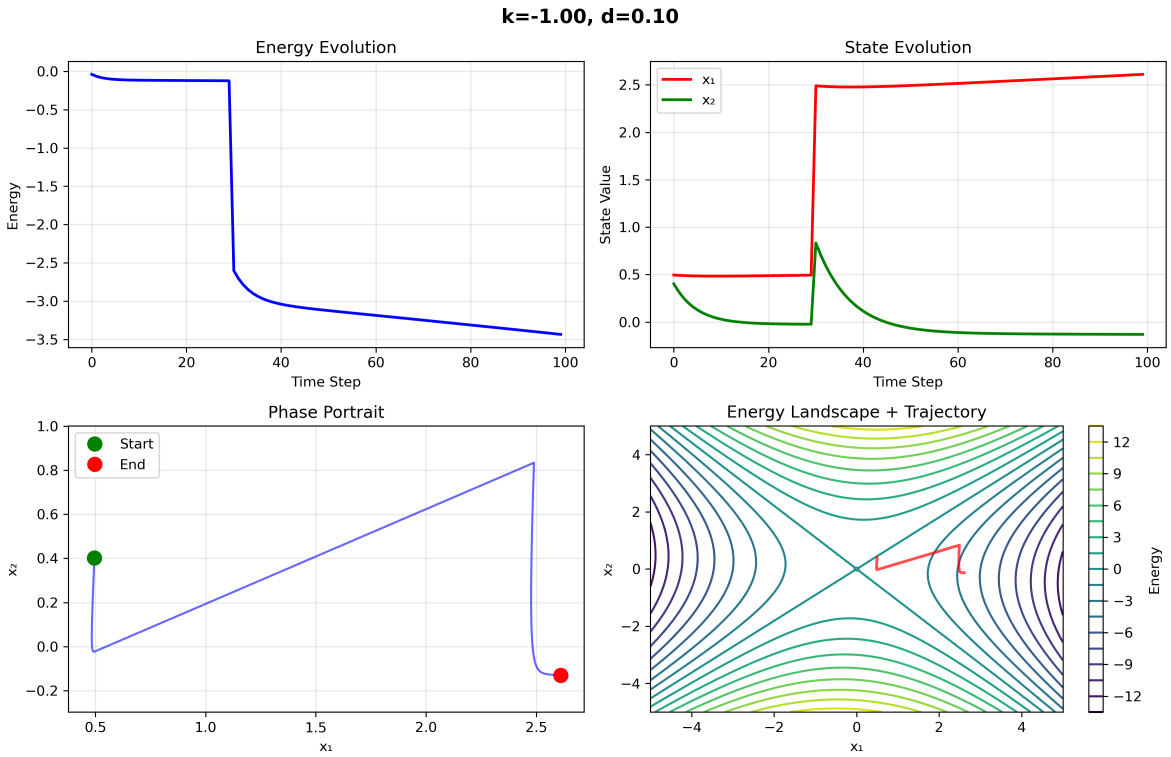


FIGURE 1.14: Unstable tokamak configuration - Energy, States and Phase Portrait

Figures 1.13,1.14, present the stable and unstable systems response to a perturbation in terms of energy and states. In the stable configuration the energy is positive over T. Small disturbances increase the system's energy, and restoring forces act to return it toward equilibrium. Thus, $\Delta E = 0$ reflects stability, where perturbations decay over time instead of growing. The phase portrait (x_1, x_2) exhibits a stable spiral converging to the origin.

In the unstable configuration the perturbation energy growth indicates that the equilibrium is not a minimum but a saddle point of the potential energy landscape. The perturbation energy grows until states approach the constraint boundaries $l \geq x \geq u$. When the energy grows until the states reach the constraint boundaries, the system may either saturate if well controlled or exceed these limits, leading to a loss of plasma confinement.

1.11.3 Portfolio Allocation optimization

In the presented investment framework, asset allocations are determined using a static Optimal Control (OC) approach, as the future market dynamic is not known. At the beginning of each year, 100 currency units (USD or Euro) or assets are bought, and the allocation weights are obtained by solving a quadratic programming (QP) problem that jointly considers the trade-off between expected portfolio returns and associated risk. The optimization is based on statistical estimates of expected returns and the covariance matrix of asset returns, computed over a rolling historical window. The presented problem does not allow the sell of some assets and is presented as a risk mitigation tool to help human decision making. Theoretical aspects of optimal control problems are not introduce here, as the portfolio study serves only in the current chapter to illustrate QP-problems applications.

From the estimation window of T past trading years, we compute over n assets:

$$\mu = \mathbb{E}_T[r] \in \mathbb{R}^n \qquad \text{(annualized mean returns)}, \tag{1.154}$$

$$\Sigma = \mathrm{Cov}_T(r) \in \mathbb{R}^{n\times n} \qquad \text{(annualized covariance matrix)}, \tag{1.155}$$

where r denotes the vector of daily returns.

At the beginning of year k, we seek portfolio weights $w \in \mathbb{R}^n$, where w_i is the fraction of wealth invested in asset i. The QP-problem can be written as,

$$\min_{w\in\mathbb{R}^n} \quad \frac{1}{2} w^\top Q w + c^\top w, \tag{1.156}$$

$$\text{s.t.} \quad \mathbf{1}^\top w = 1, \tag{1.157}$$

$$l \le w \le u, \tag{1.158}$$

where,

$$Q = \xi\Sigma, \tag{1.159}$$

$$c = -\lambda\mu. \tag{1.160}$$

Here $\lambda > 0$ is a risk-aversion parameter. The quadratic term $w^\top\Sigma w$ measures portfolio variance (risk), while the linear term $-\lambda\mu^\top w$ rewards expected returns. The constraints ensure the portfolio is fully invested (eq: (1.157)), while the budget is not spent on too many or too few assets (eq: (1.158)). The covariance captures the degree to which asset returns move together and thus quantifies the diversification potential within the portfolio. A lower covariance between assets implies greater diversification benefits, allowing the model to allocate weights that minimize overall portfolio variance for a given level of expected return. The diagonal entries of the covariance matrix represent each asset's individual variance (its own risk), while the off-diagonal terms capture pairwise covariances indicating how assets co-move. Larger diagonal values correspond to higher individual volatility, whereas smaller or negative off-diagonal values reflect assets that diversify each other more effectively.

At each decision period k, the controller framework updates portfolio allocations by estimating the expected returns μ_k and covariance matrix Σ_k from the most recent T years of data, solving the quadratic program to obtain the optimal weights w_k^*, allocating the annual contribution C proportionally to these weights, and holding the resulting portfolio until the next rebalancing date. The covariance matrix Σ_k summarizes the joint risk structure of the assets during the estimation window. The controller policy adapts the weights each year, reflecting updated risk-return characteristics. By construction, $\sum_{i=1}^n w_{k,i}^* = 1, \quad w_{k,i}^* \ge 0$, ensuring the portfolio is normalized and feasible.

Results: Figure 1.15 illustrates the evolution of the portfolio (see Table 1.10) value under optimal controlled and equal-weighted strategies over the period 2020-2025. The portfolio exhibits a characteristic stepwise growth pattern corresponding to the yearly investment schedule. Each January, a new contribution is invested, leading to discrete upward shifts in the cumulative capital curve. Between 2020 and 2023, both strategies evolved similarly. However, the controlled allocation systematically achieves higher portfolio values, reflecting its adaptive optimization based on updated covariance estimates and expected returns. From 2023 onward, more pronounced divergences appear. Notably, the peaks observed in late 2023 and mid-2024 coincide with strong price variation in the **ASML**'s stock, which

Table 1.9: Portfolio optimization parameters

Parameter	Value
T	10 years
t_{begin}	2010
t_{end}	2020
$T_{prediction}$	2020-2026
$t_{prediction}$	1 year
u	0.4
l	0.0
λ	2
ξ	2

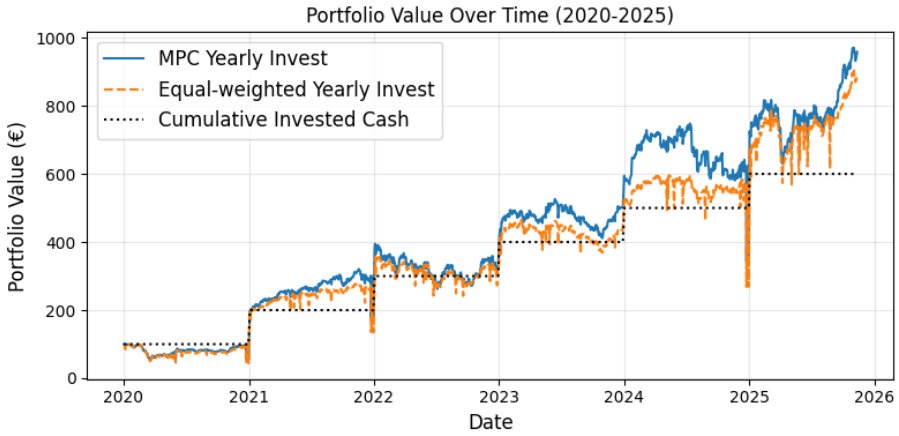


Figure 1.15: Portfolio evolution (01/01/2020-01/10/2025), controlled vs. equally weighted.

represents a substantial portion of the MPC portfolio during those years. This concentration arises from ASML's high expected return and relatively low correlation with other industrial and defense-oriented assets.

Table 1.10: Constituent companies of the European portfolio.

Company	Ticker (Yahoo Finance)
ASML Holding N.V.	ASML.AS
Shell plc	SHEL.L
ArcelorMittal S.A.	MT.AS
Deutsche Lufthansa AG	LHA.DE
International Airlines Group	IAG.L
Nokia Oyj	NOKIA.HE
Siemens AG	SIE.DE
Mercedes-Benz Group AG	MBG.DE

The performance ratio between the two strategies is shown in Figure 1.16. The curve remains almost always above 1 throughout the period 2020-2025, indicating a consistent outperformance of the optimizer strategy relative to the equal-weighted benchmark, reaching only once a minimal ratio of 0.971.

The actions and the repartition of the portfolio are presented in Figure 1.17. From 2020 onward, ASML maintains a dominant share—typically between 25% and 45%, reflecting its strong historical performance and low covariance and favorable profit profile relative to the

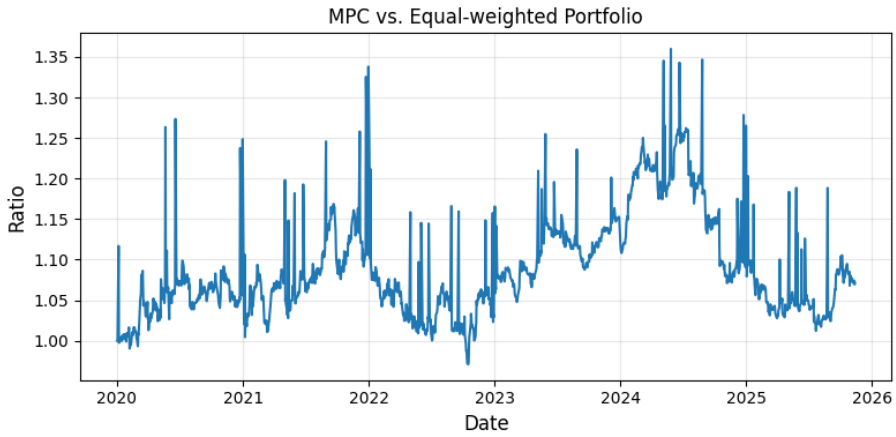


FIGURE 1.16: Ratio controlled portfolio vs. equally weighted portfolio.

other assets. In contrast, other assets such as ArcelorMittal (MT.AS), Lufthansa (LHA.DE), and International Airlines Group (IAG.L) receive significantly lower allocations, consistent with their higher volatility and weaker expected returns. Industrial assets such as Siemens (SIE.DE) and Mercedes-Benz Group (MBG.DE) appear recurrently as stabilizing components, particularly in later years (2023-2025), when market conditions increased their relative attractiveness due to moderate covariance and steady returns. The accompanying circular charts summarize the cumulative portfolio composition up to each year. Overall, the optimizer framework dynamically reallocates capital in response to shifting return and covariance structures, yet the optimization consistently emphasizes technology and industrial leaders, capturing market momentum while exposing the portfolio to idiosyncratic risk from dominant positions.

Figure 1.18 illustrates the historical evolution of selected asset prices from 2010 to 2025. The period from 2010 to 2020 corresponds to the data used for estimating the covariance matrices, while the subsequent interval, highlighted in green, represents the algorithm's operation phase, including one-year-ahead predictions and a rolling covariance matrix re-update.

Figure 1.19 presents covariance matrices estimated over rolling previous ten-year windows. First, **ArcelorMittal (MT.AS)** and **International Airlines Group (IAG.L)** consistently exhibit the highest intra-sector covariance values, reflecting their exposure to macroeconomic factors and price fluctuations. In contrast, **ASML Holding N.V. (ASML.AS)** shows low covariance with most other assets, particularly industrials such as **Siemens (SIE.DE)** and **Mercedes-Benz Group (MBG.DE)**. This low correlation explains its repeated overweighting in the asset allocations, as the optimizer leverages ASML's strong historical returns combined with its diversification potential.

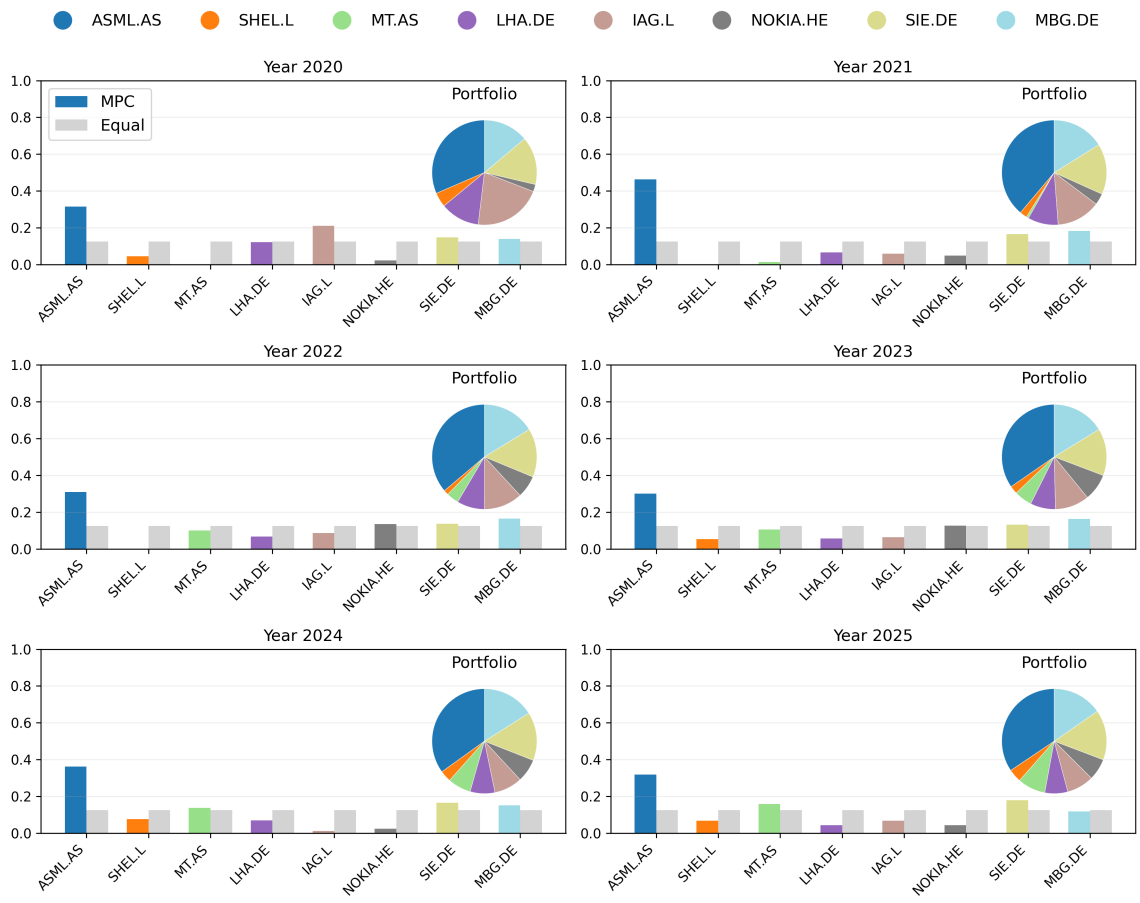


FIGURE 1.17: Actions and assets repartition.

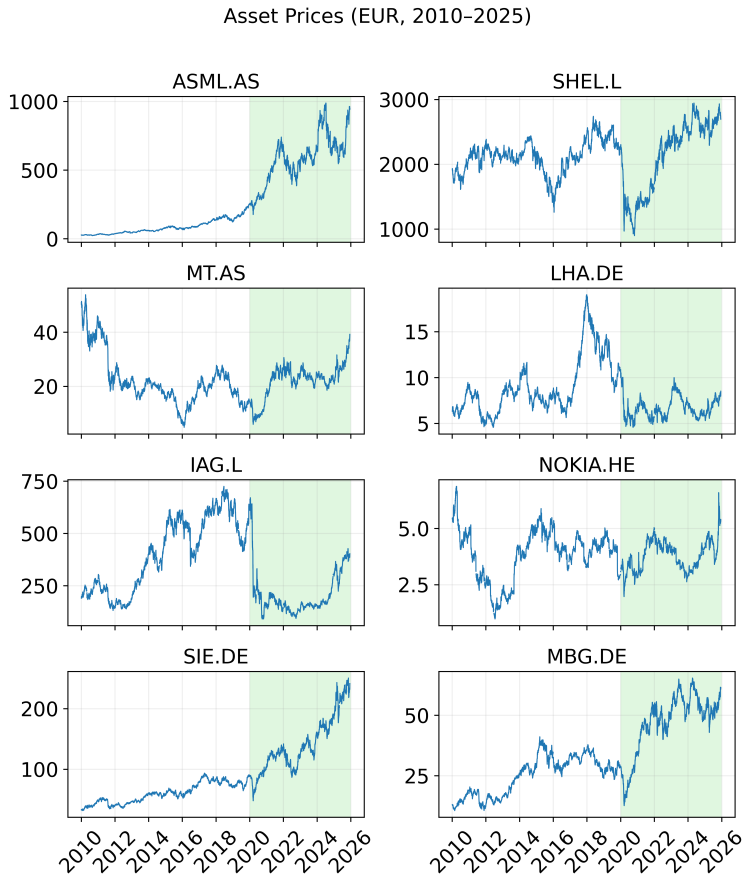


FIGURE 1.18: Asset price evolution from 2010 to 2025. The shaded area marks the prediction period.

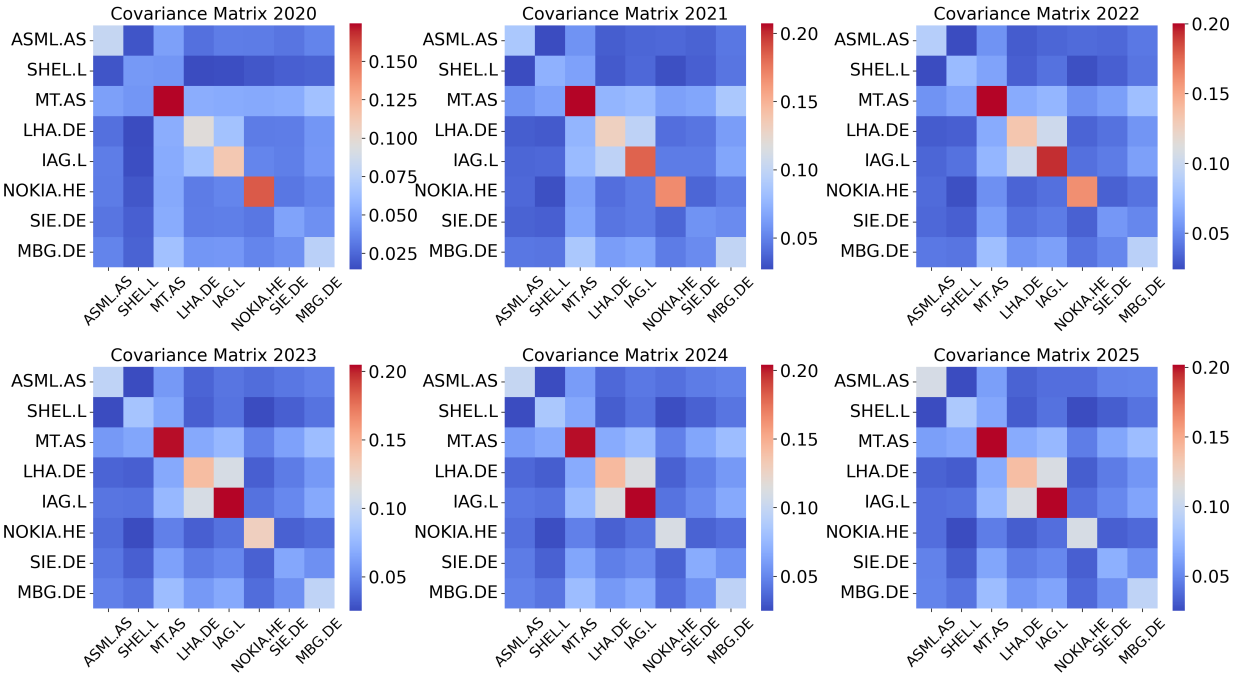


FIGURE 1.19: Covariance matrices per year with the rolling data window: $T = 10$ past years.

1.11.4 Quadrocopter Path-Following Optimal Control

In this study, the quadrocopter is modeled as a point mass with no orientation dynamics, under the assumption that attitude control is handled by an innerloop stabilizing low-level controller. Consequently, only the translational motion is considered in optimal control formulation. The vehicle dynamics are described by the position and velocity of the center of mass, while the control inputs correspond to the commanded accelerations along the three spatial axes. The state vector is defined as $\mathbf{x}_k = [p_x, p_y, p_z, v_x, v_y, v_z]^\top \in \mathbb{R}^6$, comprising position and velocity components, while the control vector is $\mathbf{u}_k = [a_x, a_y, a_z]^\top \in \mathbb{R}^3$, representing acceleration inputs. The quadrotor is modeled as a double integrator along each axis, such that

$$\dot{p}_i = v_i, \quad \dot{v}_i = a_i, \quad i \in \{x, y, z\}. \tag{1.161}$$

In compact form, the continuous-time dynamics can be written as

$$\dot{x}(t) = \begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{p}_z \\ \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} 0_{3\times3} & I_3 \\ 0_{3\times3} & 0_{3\times3} \end{bmatrix} x(t) + \begin{bmatrix} 0_{3\times3} \\ I_3 \end{bmatrix} u(t). \tag{1.162}$$

Applying Euler discretization with a sampling period Δt yields the discrete-time dynamics

$$x_{k+1} = Ax_k + Bu_k, \tag{1.163}$$

where,

$$A = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \Delta t & 0 & 0 \\ 0 & \Delta t & 0 \\ 0 & 0 & \Delta t \end{bmatrix}. \tag{1.164}$$

The finite-horizon optimal control problem is formulated by minimizing the quadratic cost function

$$\min_{x,u} J = \sum_{k=0}^{N-1} (x_k^\top Q x_k + u_k^\top R u_k) + x_N^\top Q_f x_N, \tag{1.165}$$

where $Q \succ 0$ and $R \succ 0$ are the state and control diagonal weighting matrices, and Q_f is the terminal cost matrix. By concatenating all state and control variables into a single decision vector

$z = \begin{bmatrix} x_0^\top & x_1^\top & \cdots & x_N^\top & u_0^\top & u_1^\top & \cdots & u_{N-1}^\top \end{bmatrix}^\top$, and the cost (1.165) can be written in compact quadratic form:

$$J = \frac{1}{2} z^\top H z, \tag{1.166}$$

where H is a block-diagonal matrix composed of the stage and terminal weights,

$$H = \text{bloc}(Q, \ldots, Q_f, R, \ldots, R). \tag{1.167}$$

The parameters of the controller and the quadrocopter system are presented in table 1.11.

TABLE 1.11: OCP parameters and physical bounds.

Parameter	**Value**
Prediction horizon T	4.5 s
Sampling time Δt	0.01 s
Sampling points N	450
Velocity limit $v_{\max}$	5.0 m/s
Acceleration limit $a_{\max}$	5.0 m/s^2
Control input limit $u_{\max}$	0.25 m/s^2
State weight matrix $\mathbf{Q}$	$\mathrm{diag}(1.0)^{6\times6}$
Control weight matrix $\mathbf{R}$	$\mathrm{diag}(0.01)^{3\times3}$
Reference trajectory $\mathbf{x}_k^{\mathrm{ref}}$	$\mathbf{x}_k^{\mathrm{ref}} = \begin{bmatrix} 25\,t_k + 2.0\,\sin(2\pi t_k) \\ 25\,t_k + 2.0\,\cos(2\pi t_k) \\ 10\,\sin(2\pi t_k)\,(1-t_k) \end{bmatrix}$, $t_k = \frac{k}{N-1}, \quad k = 0,\ldots,N-1.$ Cubic Spline interpolated

The results in Figures 1.20 and 1.21 demonstrate that the controller effectively tracks the smooth nonlinear reference trajectory while satisfying the imposed physical limits. The 3D trajectory plot shows the trajectory solution closely follows the desired oscillating path, with minimal deviation at sharp curvature regions. The velocity and acceleration profiles remain well within their respective bounds $v_{\max} = 5.0$ m/s and $a_{\max} = 5$ m/s^2, indicating that the control effort is appropriately regulated. The control input signals exhibit smooth variations with amplitudes below the specified saturation limit $u_{\max} = 0.25$ m/s^2, confirming stable actuation behavior. The position tracking plots reveal that all three spatial components converge to the reference endpoints with small transient errors, particularly around turning points where the reference curvature changes. Finally, the constraint magnitude plot verifies that the controller maintains dynamic feasibility throughout the prediction horizon, ensuring consistent compliance with velocity, acceleration, and control constraints as long as there is a reference to track. Overall, the OCP formulation provides accurate and dynamically consistent trajectory tracking performance for the simplified quadrocopter model. The quadratic program (QP) requires approximately 0.005 s of computation time per iteration to plan the 20 s reference trajectory at an optimization termination tolerance of 10^{-3}, demonstrating real-time feasibility of the proposed control scheme inside a predictive controller.

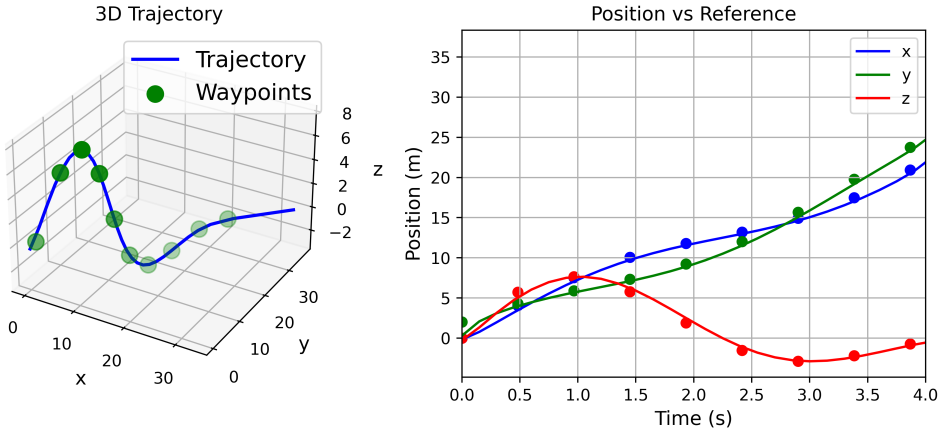


FIGURE 1.20: Visualization of the quadrocopter trajectory and position tracking. The subplots show (left to right): 3D trajectory with waypoints, and position versus waypoints over time, highlighting how the quadrocopter follows the planned path.

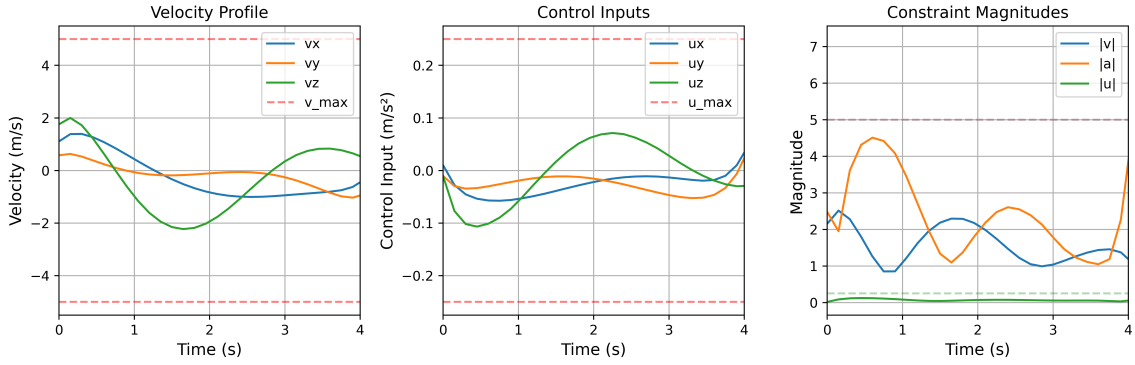


FIGURE 1.21: Visualization of velocity, control inputs, and constraint magnitudes over time. The subplots show (left to right): velocities, control inputs, and constraints magnitudes.

1.12 Chapter conclusion

This chapter presented a comprehensive overview of continuous constrained QP optimization problems using Interior Point Methods (IPM). We discussed classical approaches, also including proximal methods, augmented Lagrangian techniques, and quadratic primal-dual penalty Gill-Robinson terms, emphasizing how the convergence of the outer and innerloop can be formally described. By introducing a stabilized Newton scheme formulation (1.81), we ensured it remains robust to numerical errors and efficiently solvable by state-of-the-art linear solvers. The stable formulation employs a scaling of the gradients (s) and we provided a formal proof that the transformed directions minimize a regularized merit function. Beyond the mathematical perspective, we provided interpretations of regularization effects on the optimization process and the behavior of candidate solution trajectories.

A novel s-proximal regularization was introduced on the slack variables and relaxed inequality constraints. The Euclidean $s-$proximal term effectively limits excessive slack steps while maintaining positivity, and an analytical formula for estimating θ_s serves as a proximal anchor, enabling the study of the IPM outer loop as an inexact proximal point algorithm.

Additional safeguards, including central path corrections where estimators $\theta_{\square}$ are adjusted, ensure that solution trajectories remain close to the IPM central path. This framework was implemented in the solver `QProxIP`, with careful analysis of data profiles and δ-dynamics, highlighting their influence on computational speed and robustness.

The proposed approach was successfully validated on the Maros–Mészáros test set, achieving performance comparable to the best academic and commercial solvers in terms of robustness and computational efficiency. Performance variability was also observed depending on factors such as CPU core allocation, temperature, and other uncontrolled hardware and system conditions. This highlights that benchmarking is important for understanding solver performance at scale, but results cannot be interpreted as universal truth, particularly for complex solvers with intricate heuristics and tuned for specific test sets. Multiple toy problems were also presented, demonstrating applicability across different fields, from finance to control. The results illustrate the effectiveness of the proposed theoretical developments and methodological innovations, providing a solid foundation for the construction and solution of more complex nonlinear, non-convex optimal control problems in Part 2, where real-time control is the primary goal.

Chapter 2

Learning to Enhance a QP-solver: A Reinforcement Learning Approach

2.1 Introduction

The convergence behavior of quadratic programming (QP) solvers is often highly sensitive to the choice of algorithmic hyperparameters, particularly those related to regularization. The performance and robustness of a solver can vary significantly across problem instances, as shown in §1.10.3 making manual tuning both problem-dependent and computationally burdensome. Developing methods capable of autonomously adapting these parameters through lightweight learning-based mechanisms can provide a substantial advantage in engineering and control applications, where robustness and speed are critical.

Although convergence guarantees can be established under minimal requirements on the problem data and algorithmic parameters, there are often numerous hyperparameters that must be tuned to speed up convergence, and few clues on how to adjust them. Furthermore, in proximally regularized methods, the stepsize parameters not only affect the convergence rate but they also impact on the stability of linear algebra operations; cf. [29, §4].

Some theoretical works seek optimal values for the stepsize parameters, but they rely on solving auxiliary problems that are harder than the original QP itself [56, 60]. Another offline tuning approach is that of OPAL [7], which takes an algorithm as input and returns a recommendation on parameter values that maximize some user-defined performance metric. Alternatively, online heuristics introduce "feedback" by monitoring progress and adapting the parameters along the optimization process; see [20, §2], [119, §5.2], [68, §5.3] for some common instances.

Remark 2.1.1 RL-QProxIP

Our goal is twofold and not to demonstrate state-of-the-art performance: (i) to keep numerical noise and imperfections unfiltered in order to stress the solver, and (ii) to provide a clean baseline that isolates and clarifies the effect of reinforcement learning on performance.

After the seminal work [85] on "learning to optimize" for automated algorithm design, several researchers approached acceleration, warm-starting and hyperparameter tuning as machine learning tasks. A modern view on this computational approach is presented in the

tutorial [5] on "amortized optimization" methods, which use learning to predict the solutions to problems by exploiting the shared structure between similar problem instances.

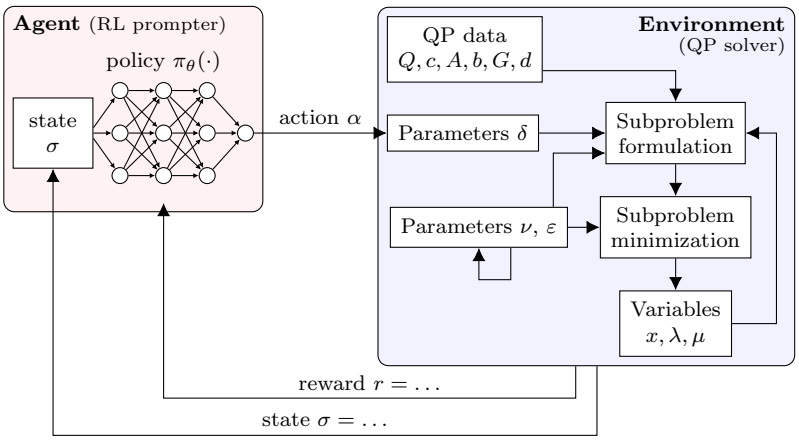


FIGURE 2.1: A QP solver is combined with a reinforcement learning (RL) agent that evaluates a policy to adapt the internal parameters of the QP solver. Based on its observation of the state of the environment, the RL action aims at improving the solver's performance. The RL policy is parameterized by a small artificial neural network, the state σ summarizes the internal state of the QP solver (at the beginning of each inner loop), the action α affects the parameters δ of the solver, and the reward $\mathcal{R}$ promotes faster convergence to a solution of the QP. Illustration inspired by [70, Fig. 1].

Despite the flexibility of learning-based methodologies, considerable attention was devoted to accelerating *first-order* optimization methods, driven by their scalability and applications in machine learning. Exemplarily, the architecture proposed in [111] consists of a neural network mapping from the QP problem data to warm-starts, learned across families of QPs. Dedicated to proximal methods for image processing, the tuning-free approach of [127] formulates the online hyperparameter selection as a sequential decision-making problem and implements a policy network for automated parameter search. Closer to our focus here is RLQP [70], which couples the OSQP solver with a reinforcement learning (RL) agent: by learning a policy to adapt the internal parameters of the ADMM algorithm, RLQP can outperform OSQP (with its fine-tuned heuristic update rules) on a variety of problems.

In contrast, the combination of learning techniques and *second-order*, or Newton-type, methods remains relatively unexplored. We investigate the coupling of a regularized IP solver for QPs with a policy for online hyperparameter tuning, parametrized with a simple neural network and learned by RL. Compared to the single-loop structure of OSQP, the architecture arising from an IP scheme can be represented with two nested loops, where each inner loop corresponds to the approximate solution of a subproblem (by executing some iterations of Newton method); see Figure 2.1. While ubiquitous in second-order methods and with well established convergence theory, the two-loop structure does not fit the RL framework developed for RLQP, because of endogenous parameter sequences controlling the solution process (and affecting the MDP formalization). We aim to fill this gap by exploring a machine learning framework suitable for accelerating the tuning and solution process of QP solvers beyond first-order methods.

We formulate a RL framework to learn a policy for effective and reliable hyperparameter tuning, with the QP solver retaining provable convergence guarantees. The definitions of state, action and reward are intended (i) to yield a policy that is invariant to problem size,

scaling and permutations, and (ii) to expose other internal parameters that drive the solver's convergence.

Experimental results on random and benchmark QPs demonstrate that the learned policy performs well on previously unseen problem sizes, scales and classes, where tedious hand-tuning is usually needed. Moreover, these capabilities emerge after a lightweight training of a relatively small neural network, running on a standard laptop, compared to the prohibitive training for RLQP.

Convex quadratic programming is introduced in chapter 1 with a stabilized IP numerical scheme and a discussion on the challenges and opportunities of hyperparameter tuning. An overview of reinforcement learning is given in Section 2.3, with the tools needed to properly interact with the QP solver. Training procedures and numerical results for three evaluation settings are then presented in Section 2.4. We conclude with some final remarks in Section 2.5.

2.2 Hyperparameters in QP problems

The behavior and performance of Algorithm 7 depend on the sequences of hyperparameters $\{\delta^k\}$, $\{\nu_k\}$ and $\{\varepsilon_k\}$. As depicted in Figure 2.1, we deliberately consider the barrier parameter ν and inner tolerance ε as endogenous hyperparameters; their values are selected by evaluating common heuristic rules. Conversely, the regularization parameters δ will be subject to the action of the external RL agent and updated according to its policy.

The evolution of the barrier and inner tolerances follows the adaptive schemes introduced in § 1.5.2. Both parameters decrease according to problem-dependent update rules combining linear and superlinear decay, with lower bounds proportional to the prescribed accuracy ϵ. The initial inner tolerance is selected adaptively from the magnitude of the initial primal–dual residual, scaled by a damping factor.

While parameters ν and ε should become sufficiently small, there is no clear way to choose the weights δ; in fact, convergence is guaranteed even if they are kept constant. This motivated us to experiment with machine learning techniques to discover a policy for updating δ^k.

Our approach is to define the update rule as

$$\delta_{\square,k+1} = \max\{\delta_{\min}, \alpha_{\square,k}\delta_{\square,k}\} \qquad \text{for } \square \in \{x,\lambda,\mu\}, \tag{2.1}$$

starting from some $\delta^0 = (\delta_{x,0}, \delta_{\lambda,0}, \delta_{\mu,0}) \in \mathbb{R}^3_{++}$, where the small threshold $\delta_{\min} > 0$ safeguards the numerical linear algebra for (1.81) and the decrease factors $\alpha_{\square,k} \in (0,1)$ are outputs of the RL policy; see Figure 2.1. To fix ideas, the RL action takes the form

$$\alpha^k = \begin{pmatrix} \alpha_{x,k} \\ \alpha_{\lambda,k} \\ \alpha_{\mu,k} \end{pmatrix} = \alpha\left(r_{\mathrm{p}}(x^k,\lambda^k,\mu^k), r_{\mathrm{d}}(x^k,\lambda^k,\mu^k), \nu_k, \varepsilon_k\right), \tag{2.2}$$

where the mapping α is learned by training a neural network π_ζ. The choice of defining this mapping with residuals as inputs, as opposed to primal-dual decision variables (x,λ,μ), is

intended to make the mapping invariant to the problem dimensions (n, m, p); subsequently we will make it also scale-invariant.

Algorithm 4 is modified into Algorithm 7 where the policy π_ζ is called for the hyperparameter tunning, with the differences highlighted in blue for clarity,

Algorithm 7 Regularized primal-dual Interior-Point Method for Convex QPs - hyperparameter tuning.

1: Initialize variables $z_0 = [x_0^0, s_0^0, \lambda_0^0, \mu_0^0]^T$, estimators $(\theta_{\lambda_0,\mu_0,x_0})$, proximal and augmented Lagrangian weight $(\delta_{\lambda_0,\mu_0,x_0})$, and barrier parameter ν_0.
2: **while** Outer-termination (k): $r_\mathrm{p} \leq \epsilon$ and $r_\mathrm{d} \leq \epsilon$ **do**
3: **while** Inner-termination (j): $\|\nabla \mathcal{M}_\delta^{GR}(x_k^j, s_k^j, \lambda_k^j, \mu_k^j)\| \leq \varepsilon_k \leq \varepsilon$ **do**
4: Solve the primal KKT system for $(x_k^j, s_k^j, \lambda_k^j, \mu_k^j)$: $KKT \cdot \Delta = -R$ (1.71).
5: Perform linesearch to find $(x_k^j, s_k^j, \lambda_k^j, \mu_k^j)$ such that $\|\nabla \mathcal{M}_\delta^{GR}(x_k^j, s_k^j, \lambda_k^j, \mu_k^j)\| \leq \varepsilon_k$ and $s_k^j > 0$.
6: Update the primal and dual variables: $z_k^{j+1} \leftarrow z_k^j + \alpha \Delta z$.
7: **end while**
8: Update estimators: $\theta_\square^{k+1} \leftarrow (x^k, \lambda^k, \mu^k)$
9: Update δ: $\delta_\square^{k+1} \leftarrow \pi_\zeta$.
10: **end while**.

Compared to Algorithm 4, the evolution of the weights $\delta_\square$ in Step 9 is governed by the policy π_ζ, whereas in Algorithm 4 the weights are decreased only when the constraint violation exceeds a predefined threshold, as in Step 9.

We conclude this overview on the modified-QP solver by remarking that common safeguards and tricks for numerical stability were intentionally left out. Therefore, features such as boundary control [103], centrality correction [114], presolving and Ruiz-based preconditioning [119], nonmonotone linesearch [63, 84], infinity pruning for the inequality constraints [119], and adaptive early-termination strategies are deactivated for all numerical tests.

2.3 Learning-enhanced methods

In the following, we aim at finding a good strategy to choose the parameters during the optimization. So far, the choice of these parameters is up to the operators, and thus, the optimization highly depends on their experiences. Reinforcement learning [122] makes it possible to automate this choice and avoid manual hyperparameter tuning. Therefore, reinforcement learning approaches benefit from their ability to train a policy by interacting with the environment in a trial-and-error manner. In recent years, the use of machine learning methods in order to improve optimization strategies has increased. For example, neural networks are used to warm-start either QPs [111] or Fixed-Point Optimization Algorithms [112].

In [70] the authors apply a Twin-Delayed Deep Deterministic Policy Gradient approach to adapt step size parameters for solving QP problems. But instead of improving the Alternating Direction Method of Multiplier (ADMM) approach, we focus on the regularized interior point approach, which was discussed earlier. Thus, the goal is to automate the choice of parameters $\alpha_{\square,k}$ for $\square \in \{x, \lambda, \mu\}$ in iteration k. Thereby, the choice should be made with respect to the primal and dual residuals, the barrier parameter and the inner tolerance as in

(2.2). Figure 2.1 shows the embedding of the RL policy into the optimizer which is summarized in Algorithm 4. In an inner loop, the primal-dual augmented Lagrangian function $\mathcal{M}_\delta(x, s, \lambda, \mu; \theta, \nu)$ is minimized for fixed θ^k, barrier parameter ν_k and weights $\delta_{\square,k}$ until a tolerance ε_k is reached. Once this has been achieved, in an outer loop θ^k, ν_k and ε_k are updated for the next iteration of the inner loop. In addition, the hyperparameters $\delta_{\square,k}$ are adapted by factors $\alpha_{\square,k}$, which are determined by the RL policy. Thus, the RL policy acts at Step 8 in Algorithm 4, where δ^k is updated. This procedure is repeated until the primal and dual residuals are small enough. During training, the policy to be learned is improved step by step by interacting with the described framework, which in RL context is also called environment, while trying to solve randomly selected QPs.

2.3.1 States and reward

We continue with the mathematical background, which is needed for Reinforcement Learning. RL is based on the Markov decision process (MDP) [40], which is represented by the quadruple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$. The state space $\mathcal{S}$ comprises all possible configurations of the environment, while the action space $\mathcal{A}$ represents all actions which allow the policy to influence the environment. The probability transition $\mathcal{P}$ describes the probability of reaching a state under the condition that the previous state and action are given. Finally, the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ rates the state transition and enables to compare and improve solutions. While the state and action space as well as the reward function need to be defined by the operator, the probability transition does not need to be given explicitly. It is compensated by the interaction of the policy with the environment. In the following, we will define this framework for our present task.

States: Since we are convinced that the choice of factors $\alpha_{\square,k}$ for $\square \in \{x, \lambda, \mu\}$ depends mainly on the primal and dual residuals of the QPs, we decide to make them part of the state definition. Moreover, since the residuals (r_p and r_d) can span several orders of magnitude during the solution process, causing scaling issues while training the policy networks, we encode the environment state with residuals in logarithmic scale:

$$\sigma(x, \lambda, \mu, \nu, \varepsilon) := \begin{pmatrix} -\log\left(\sigma_\mathrm{p}(x, \lambda, \mu) + 10^{-9}\right) \\ -\log\left(\sigma_\mathrm{d}(x, \lambda, \mu) + 10^{-9}\right) \\ -\log\left(\nu + 10^{-17}\right) \\ -\log\left(\varepsilon + 10^{-9}\right) \end{pmatrix} \tag{2.3}$$

where the normalized residuals σ_p and σ_d, defined in terms of the primal and dual residuals r_p and r_d (see (1.43) and (1.45)), are given by

$$\sigma_\mathrm{p}(x, \lambda, \mu) := \frac{r_\mathrm{p}(x, \lambda, \mu)}{\max\{\|A\|_\infty, \|b\|_\infty, \|G\|_\infty, \|d\|_\infty\}} \tag{2.4a}$$

$$\sigma_\mathrm{d}(x, \lambda, \mu) := \frac{r_\mathrm{d}(x, \lambda, \mu)}{\max\{\|Q\|_\infty, \|q\|_\infty, \|A\|_\infty, \|G\|_\infty\}} \tag{2.4b}$$

are intended to make the state definition scale-invariant.

Remark: The inclusion of the barrier parameter ν and tolerance ε into the state vector guarantees the Markov property of our framework. Note that at this point our choice of the state vector differs from the one chosen in [70], which did not have any additional parameters that need to be updated for an inner loop. In our case, omitting them would have made it necessary to introduce a partially observable MDP [73]. However, the preliminary numerical tests reported in Section 2.4 indicate that the RL training remains manageable and efficient in practice. Nevertheless, an extension to partially observed MDPs associated with a change of RL formulation seems of interest for an even leaner training procedure, but it comes with challenges and is left for future research.

Action: The action of our task is the choice of $\alpha_{\square,k}$ in each iteration step k for $\square \in \{x, \lambda, \mu\}$. Since $\alpha_{\square,k}$ is the factor that is used to sequentially decrease the regularization parameter $\delta_{\square,k}$, according to (2.1), it must be strictly positive and smaller than or equal to one.

Reward: Finally, we define the reward function. We remember that our goal was to accelerate the interior point approach. To achieve this, we define a reward function, which consists of three parts

$$\mathcal{R}(\sigma, \sigma') := \mathcal{R}_1(\sigma, \sigma') + \mathcal{R}_2(\sigma, \sigma') + \mathcal{R}_3(\sigma, \sigma') \quad \text{for } \sigma, \sigma' \in \mathcal{S}, \tag{2.5}$$

where each term depends on the scaled residuals $\sigma_{\mathrm{p}}(x, \lambda, \mu), \sigma_{\mathrm{d}}(x, \lambda, \mu)$ and the inner-loop iteration count N required to meet the early-termination tolerance.

The first component, $\mathcal{R}_1$, penalizes imbalance between primal and dual residuals. As shown in Figure 2.1 and Algorithm 7, the method combines inner- and outer-loop updates. To validate early termination at tolerance ε, both residuals must be simultaneously small, which enforces a tube along the axis $r_{\mathrm{p}} = r_{\mathrm{d}}$. The tube behavior is shown in Figure 2.7. We define

$$\mathcal{R}_1(\sigma, \sigma') := -\tanh\Big(\Big|\log\big(\sigma'_{\mathrm{p}} + 10^{-9}\big) - \log\big(\sigma'_{\mathrm{d}} + 10^{-9}\big)\Big| \cdot c_g\Big). \tag{2.6}$$

The definition of c_g and other constants that appear in the reward are listed in Table. 2.4.

The second component, $\mathcal{R}_2$, rewards large reductions in the residuals between consecutive outer loops, thereby promoting consistent improvement in feasibility and optimality before updating the proximal and augmented Lagrangian parameter $\delta_{\square}$. It is given as

$$\begin{aligned}\mathcal{R}_2(\sigma, \sigma') := \tanh\Big(\Big(\log\big(\sigma_{\mathrm{p}} + 10^{-9}\big) - \log\big(\sigma'_{\mathrm{p}} + 10^{-9}\big)\Big) \cdot c_p\Big) \\ + \tanh\Big(\Big(\log\big(\sigma_{\mathrm{d}} + 10^{-9}\big) - \log\big(\sigma'_{\mathrm{d}} + 10^{-9}\big)\Big) \cdot c_p\Big).\end{aligned} \tag{2.7}$$

The third component, $\mathcal{R}_3$, provides a performance bonus scaled by the iteration budget c_{perf}, incentivizing rapid convergence within a limited number of steps. For the inner-loop iteration count $N \in \mathbb{N}_{>0}$, the third component reads

$$\mathcal{R}_3(\sigma, \sigma') := 1 - \exp\left(\frac{N}{c_{perf}}\right). \tag{2.8}$$

Note that the terms $\mathcal{R}_1$ and $\mathcal{R}_2$ are smoothed by hyperbolic tangent functions, bounding them in $[-c_g, c_g]$ and $[-c_p, c_p]$, respectively. The parameter c_{perf} determines the maximum negative value attainable by $\mathcal{R}_3$.

Neural network: After we have defined the MDP, all that remains is to define the policy in order to have an RL framework. Our parameterized policy π_ζ is represented by a feedforward neural network, whose weights and biases form the parameters ζ. We determined the size of the network by a grid search and decided on a net with two hidden layers with 25 neurons in each layer. The size of the input layer (four) and output layer (three) are given by the dimension of the state and action space. For each layer, the hyperbolic tangent is used as the activation function.

Based on these definitions, RL aims to maximize the objective with respect to the weights and biases of the neural network:

$$\min_{\zeta} \quad \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i \mathcal{R}(\sigma_i, \sigma_{i+1})\right]$$

with discount factor $\gamma \in (0,1)$ and $\sigma_i \in \mathcal{S}$ for $i \in \mathbb{N}$. Note that the expected value, which appears in the objective function, is defined over all trajectories $(\sigma_0, \alpha_0, \sigma_1, \alpha_1, \dots)$ for $\alpha_i \in \mathcal{A}$.

2.3.2 Training procedures

Based on the MDP, which was defined in the previous subsection, we run the RL approach. We chose Proximal Policy Optimization (PPO) [113], red robust and stable and is capable of handling continuous action spaces. It is an actor-critic RL approach, which means that the policy π_ζ and the value function V_Φ with parameters Φ are trained simultaneously. Note that in our case, both the policy and the value function share the majority of the neural network, which was described before. Typically, PPO uses an approximation of the advantages function $\hat{A}_k$ in iteration $k \in \mathbb{N}$, which has the form:

$$\hat{A}_k := \sum_{i=0}^{\infty} (\gamma\lambda)^i \left(\mathcal{R}(\sigma_{k+i}, \sigma_{k+i+1}) + \gamma V_\Phi(\sigma_{k+i+1}) - V_\Phi(\sigma_{k+i})\right).$$

For the parameter, which is used for the general advantage estimation $\hat{A}_k$, it holds $\lambda \in [0,1]$. Now, the parameters of the policy can be updated. For the update, a gradient ascent of the following objective function for $\tau \in [0.1, 0.3]$ is computed:

$$L_\pi(\zeta) = \mathbb{E}_k\left[\min\left(\frac{\pi_\zeta(\alpha_k|\sigma_k)}{\pi^{old}(\alpha_k|\sigma_k)}\hat{A}_k, \operatorname{clip}\left(\frac{\pi_\zeta(\alpha_k|\sigma_k)}{\pi^{old}(\alpha_k|\sigma_k)}, 1-\tau, 1+\tau\right)\hat{A}_k\right)\right].$$

It is based on the idea that the quotient of the new policy π_ζ and the previous policy π^{old} becomes larger in the case of a positive advantage value and smaller otherwise. Furthermore, the objective function is clipped if the policy update becomes too intense, leading to a more robust training. After the policy update, the approximation of the value function V_Φ is updated by a gradient descent of the loss function:

$$L_V(\Phi) := \frac{1}{2}\mathbb{E}_k\left[\left(V_\Phi(\sigma_k) - \mathcal{R}(\sigma_k, \sigma_{k+1}) - \gamma V_\Phi(\sigma_{k+1})\right)^2\right].$$

Our technical implementation is based on the RL library `RLlib` [86, 129]. The choice of hyperparameters for the described PPO approach can be seen in Table 2.1.

As metric to monitor the training, we observe the policy loss, the value function loss and

Learning rate schedule	$(0, 5 \cdot 10^{-4})$ $(500\text{K}, 3 \cdot 10^{-4})$ $(1\text{M}, 10^{-4})$ $(2\text{M}, 5 \cdot 10^{-5})$ $(3\text{M}, 10^{-5})$ $(4\text{M}, 5 \cdot 10^{-6})$ $(5\text{M}, 10^{-6})$ $(7\text{M}, 5 \cdot 10^{-7})$ $(9\text{M}, 10^{-7})$ $(20\text{M}, 10^{-8})$	Entropy coeff. schedule	$(0, 5 \cdot 10^{-4})$ $(1\text{M}, 3 \cdot 10^{-4})$ $(2\text{M}, 10^{-4})$
Batch size	4096	γ	0.99 (default)
λ	1.0 (default)	ϵ	0.2 (default)
Gradient clip	25	KL coeff.	0.05
KL target	0.01	Network layers	[25, 25] (tanh)
Num. CPUs (local worker)	20	Num. CPUs per worker	1
System configuration	Kubuntu 22.04.5 LTS Intel Core i9-12950HX		

TABLE 2.1: PPO hyperparameters used in training.

the entropy in each episode. The loss value trends show the training behavior of the policy and the value function. The entropy is monitored, since it represents the exploration and uncertainty of the policy during the training.

For the training and numerical experiments reported in this work, the solver is configured as summarized in Table 2.2. The superlinear barrier decrease is governed by $\kappa_\nu = 0.2$ and $\theta_\nu = 1.5$, as in Ipopt [126]. The tolerance ϵ for the termination conditions (1.42) is set to 10^{-6}. The early-termination decay rate is initialized at $\kappa_\varepsilon = 0.5$ and progressively damped at each outer iteration by a factor $\alpha_\varepsilon = 0.98$. An additional initialization parameter is set to $\xi = 0.01$.

ϵ	κ_ε	α_ε	ξ	κ_ν	θ_ν	max-outer-it
10^{-6}	0.5	0.98	10^{-2}	0.2	1.5	25
$\delta_{\min}$	$\delta_{x,0}$	$\delta_{y,0}$	$\delta_{z,0}$	$\alpha_{\min}$	$\alpha_{\max}$	max-inner-it
10^{-12}	1	10	10	0.05	0.95	25

TABLE 2.2: Default solver setup.

2.3.3 Training problem set

Random QP problems are generated to train the Reinforcement Learning loop illustrated in Figure 2.1. Each problem of the form of Problem (1.3.2) is defined by matrices Q, A, G and vectors q, b, d, with properties such as convexity, sparsity, LICQ (Linear Independence Constraint Qualification), and conditioning carefully controlled to produce meaningful and diverse problem batches. The considered problem class includes non-strict convexity of the objective, constraint linear dependencies, and ill-conditioning. These issues can lead to singular or poorly conditioned KKT systems, causing Newton-type solvers to struggle or converge slowly without appropriate regularization.

Non strict convexity: Matrix Q encodes the convexity and conditioning of the problem and is constructed as $Q = W^\top W$, where W is a random matrix that always has rank deficiency. The sparsity of Q is determined by the sparsity of W. Specifically, the number of nonzeros in Q depends on how the nonzero rows and columns of W interact. The density of W is $\frac{\text{nnz}(W)}{hn} \in (0,1)$. For each batch of problems, an average number of nonzero entries per row, nnz_{row}, for W is sampled from a bounded uniform distribution, see Table 2.3. Note that the density of Q is always greater than the one of W and may vary depending on the structure of W. This approach introduces variability in sparsity patterns across the batch, covering a wide range of problem constructions. With $Q = W^T W$, due to the rank deficiency of $W \in \mathbb{R}^{h\times n}$, $h < n$, the $Q \in \mathbb{R}^{n\times n}$ matrices of the batches are always symmetric positive semi-definite, and not positive definite, resulting in non-strictly convex QPs.The used rank deficiency of W is set to $n/2$.

Constraints linear dependencies: Numerical challenges arise for Newton-type solvers when dealing with equality and inequality constraints that are linearly dependent, because of singular linear systems. In practice, we generate m equality and p inequality constraints and then a fraction of constraints, $\frac{m}{2}$ and $\frac{p}{2}$, is duplicated and thus linearly dependent, increasing the total number of constraints to $\frac{3m}{2}$ and $\frac{3p}{2}$. These linear dependencies make the KKT matrix singular and, when solving (1.81) the linear system solver struggles due to an ill-posed linear system at each iteration. The augmented Lagrangian formulation using δ_λ, δ_μ breaks this linear dependency and regularizes the problem.

Ill-conditioning control: QP problems can become numerically challenging when Q is ill-conditioned, as this directly affects the conditioning of the KKT matrix (1.81) and the accuracy of the linear solver in each Newton step. When the condition number of the linear systems (1.81) is high, small numerical errors are amplified during LU or Cholesky factorization, making the Newton directions inaccurate and slowing down convergence, or breaking it altogether.

In our training set, the condition number of the Q matrices is explicitly controlled. Since $Q = W^\top W$, we have $\lambda_{\max}(Q) = \sigma^2_{\max}(W)$ and $\lambda_{\min,+}(Q) = \sigma^2_{\min,+}(W)$, with $\sigma_{\max}(W)$ and $\sigma_{\min,+}(W)$ the largest and the smallest positive singular values of W. Consequently, the effective condition number of Q can be expressed as

$$\kappa_e(Q) = \left(\frac{\sigma_{\max}(W)}{\sigma_{\min,+}(W)}\right)^2. \tag{2.9}$$

Overall, to control $\kappa_e(Q)$, a target condition number $\kappa_Q^{\text{target}} \sim \mathcal{U}^{\log}\left(10^{15}, 10^{20}\right)$ for Q is drawn. Afterwards, we manipulate the singular values of W by scaling its columns up to the drawn κ_Q^{target} according to a geometric sequence, see Definition 2.3.1.

Particularly, the scaling is applied to each column $W_{:,j}$ as

$$W_{:,j} \leftarrow \left(\frac{1}{\sqrt{\kappa_Q^{\text{target}}}}\right)^{\frac{j-1}{n-1}} W_{:,j}, \quad \forall j = 1,\dots,h, \quad h < n, \tag{2.10}$$

thereby imposing an approximate singular value decay on Q and controlling its ill-conditioning to the desired level.

Definition 2.3.1 **Geometric grid**

Let $a, b \in \mathbb{R}_{>0}$ with $a > b$, and let $n \in \mathbb{N}$. A geometric space of n positive values from a to b is a sequence $(s_j)_{j=1}^{n} \subset \mathbb{R}_{>0}$ such that

$$s_j = a \cdot \left(\frac{b}{a}\right)^{\frac{j-1}{n-1}}, \quad j = 1, \ldots, n.$$

Each term is spaced geometrically so that the ratio between consecutive elements is constant on the logarithmic scale.

Initial guess control: It is essential that the training dataset covers a broad range of the state space to ensure that Algorithm 7 encounters diverse scenarios during convergence. Coverage can be improved by strategically selecting initial points while keeping generation costs low. If initial points are consistently chosen near feasibility and stationarity, the residuals $(r_{\text{primal}}, r_{\text{dual}})$ will exhibit limited variation, biasing the training distribution toward nearly optimal states. To avoid this, candidate initial solutions (x, y, z) are selected such that the initial residuals satisfy $r_{\text{prim}}^{\text{target}}, r_{\text{dual}}^{\text{target}} \sim \mathcal{U}^{\log}(10^{-2}, 10^{2})$ and are found by solving a least-squares problem. The variable x is randomly generated, and the vectors b and d are constructed to enforce the primal residual as

$$b = Ax + r_{\text{prim}}^{\text{target}} \quad \text{and} \quad d = Gx + r_{\text{prim}}^{\text{target}} + \iota, \tag{2.11}$$

where ι is a random positive value ensuring the inequality direction. Then, with fixed x, the unregularized dual residual $r_{\text{d}}^{\text{target}}$ can be obtained by solving the least-squares problem

$$\arg\min_{\lambda,\mu} \left\| Qx + q + A^{\top}\lambda + G^{\top}\mu - r_{\text{d}}^{\text{target}} \right\|_2^2 \quad \text{subject to} \quad \mu \geq 0 \tag{2.12}$$

from which the multipliers λ and μ are computed. In practice, solving the constrained problem (2.12) is computationally expensive for large-scale dataset generation. Therefore, the unconstrained counterpart is solved and the inequality multipliers μ are then projected onto the nonnegative orthant.

The training and validation datasets are summarized in Table 2.3, which reports the ranges and distributions of the problem dimensions, the controlled effective condition number of Q, and the sparsity pattern per row of the matrix W. These settings are used to generate random QPs for both training and validations.

Setup	n
Training	$\mathcal{U}(20, 30)$
Validation	$\mathcal{U}(20, 30)$
Scale 10×	$\mathcal{U}(200, 300)$

m	$\frac{n}{\mathcal{U}(2,5)}$
p	$n\,\mathcal{U}(2, 5)$
$\text{nnz}_{\text{row}}(W)$	$\mathcal{U}(2, 5)$
$\kappa_e(Q)$	$\mathcal{U}^{\log}(10^{15}, 10^{20})$

TABLE 2.3: Setup to build random QPs for training and validation sets.

Figure 2.2 presents the actual problem sizes and sparsity statistics for the generated datasets, including the number of variables, constraints, and the distribution of nonzeros per

row in the constraint matrices. For reference, the figure also includes a comparison with the problem sizes and sparsity patterns from the Maros-Mészáros benchmark set. The training random problems exhibit a broad range of nonzero values in Q and $[A^\top G^\top]$, which are further scaled by a factor of 100× in the scale 10× validation set. In terms of sparsity, these generated problems maintain a consistent patterns across the dataset. In contrast, the Maros-Mészáros benchmark test set displays a wider diversity in both the number of nonzeros and the sparsity patterns, making it challenging to solve as a batch using a single solver configuration.

The targeted versus actual residuals for the generated problems are illustrated in Figure 2.3, highlighting the effectiveness of the initial candidate generation procedure in achieving the desired residual ranges. While the primal and dual residuals exhibit larger errors in the range $[10^{-2}, 1]$ (i.e., closest to the solution), the control performs well for larger residual values. The objective of the method is not high precision, but rather to ensure a diverse coverage of initial residuals at low computational cost.

The targeted effective condition number ($\kappa_e(Q) \sim \mathcal{U}^{\log}(10^{15}, 10^{20})$ – orange) versus the actual effective condition number (blue) for the generated training problems is illustrated in Figure 2.4. The effective condition numbers are generated up to 10^8–10^{20}, instead of the targeted 10^15–10^{20} having most of the values between 10^16–10^{20}. This range is sufficient and provides greater diversity for training. Rather than aiming for high precision, the method is designed to efficiently produce a broad, nearly log-uniform distribution of effective condition numbers.

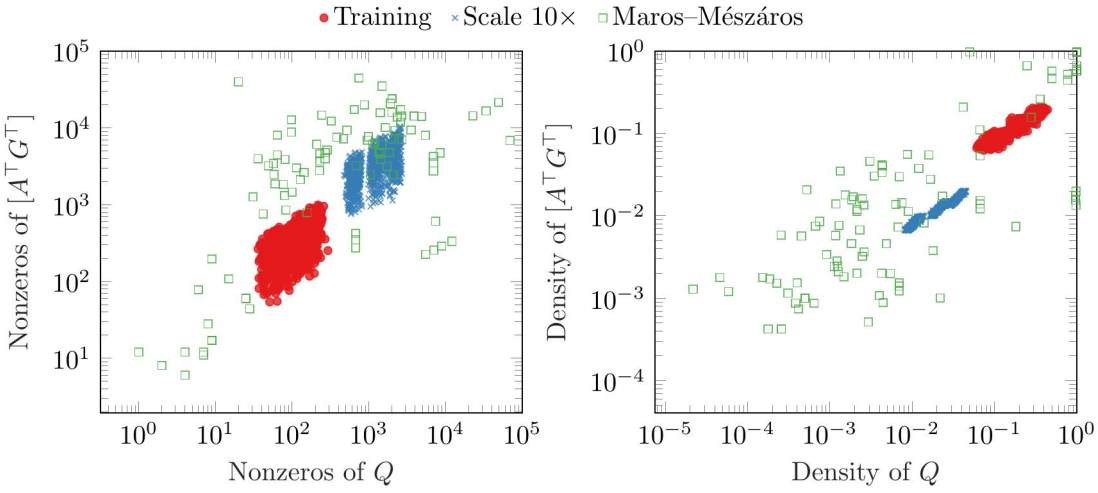


FIGURE 2.2: Distribution of problem size (in terms of number of nonzero entries) and density for randomized training, validation, and scale 10× problems (sample of 10^3 instances) and for Maros-Mészáros problems (100 smallest instances).

2.4 Numerical investigations

We report the outcomes of the training process and corresponding validation, followed by an evaluation on larger randomly generated problems to assess scalability. Finally, we benchmark the approach on the Maros-Mészáros test set to examine its zero-shot generalization on challenging real-world problem instances. The analysis is based on performance and data profiles, consistently comparing the RL-enhanced solver with the typical fixed-α strategy. The robustness and acceleration of the different methods will be discussed in the analysis.

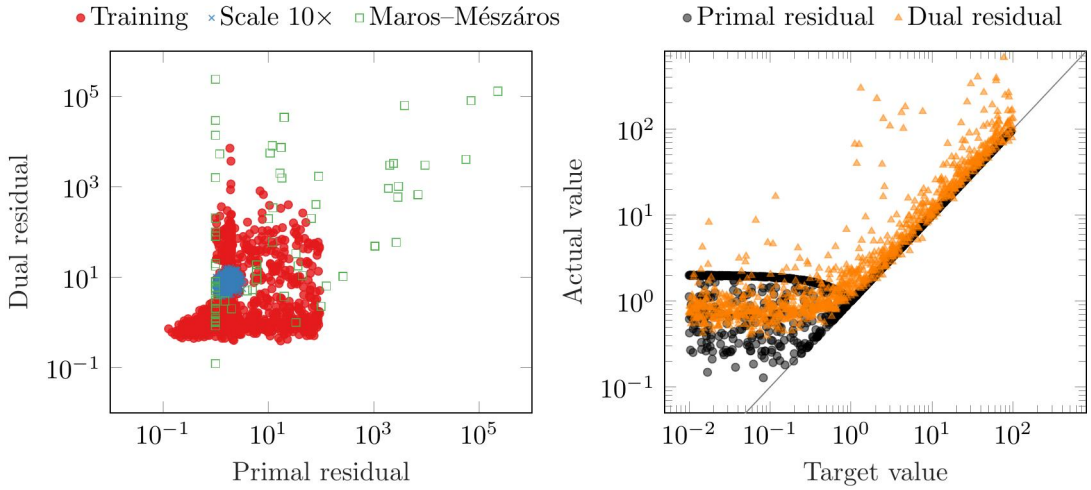


FIGURE 2.3: Distributions of problem initial residuals (sample of 10^3 instances). Primal-dual values for different problem classes (left panel). Target vs actual values generated for the training procedure (right panel).

Note that updated numerical results, obtained after the completion of this work, are available in [17].

2.4.1 Training results

The reward parameters are presented in Table 2.4 from stage 1 to 6, respectively displayed with different colors in Figure 2.5. The first training stage have only $\mathcal{R}_1$ and $\mathcal{R}_2$ and the goal here is to create a sane structure for the neural network and a smooth mapping $\sigma(x, \lambda, \mu, \nu, \varepsilon) \rightarrow \alpha$. The second stage incorporate all rewards term and introduce first the performance term, acting on the number of inner-loops which represent most of the solving time. The first and second reward term are required to ensure convergence of the primal–dual solver by promoting sufficiently large outer-loop steps, preventing premature vanishing of $\delta_\square$, and maintaining similarity between primal and dual residuals. Relying only on the performance term $\mathcal{R}_3$ would bias the training toward easier problems in average, where rapid decrease is sufficient and highly rewarded, while neglecting harder instances that demand a nontrivial $\delta_\square$ dynamics. In the final stages 3–6, only the performance term is retained, albeit at the risk of destabilizing the previously trained and structured neural network. The value of c_{perf} is set to 23 as a fast-tuning to bound $\mathcal{R}_3$ around $[0, -2]$ with a maximum allowed number of inner iteration of 25 and a maximum number of outer iteration set at 25; see Table 2.2 for the complete solver setup used for the training and validation set. The performance differences of the method after training stages 1, 2, 3 and 6 are shown in Figure 2.8. Stages 3, 4, and 6 use the same reward formulation and are each run for 2 million steps. Although these last three stages are identical, they are reported separately in the results to reflect how the experiments were conducted.

Training stage	c_g	c_p	c_{perf}
1	1	1	0
2	1	1	23
3 – 6	0	0	23

TABLE 2.4: Reward weights during the multistage training procedure.

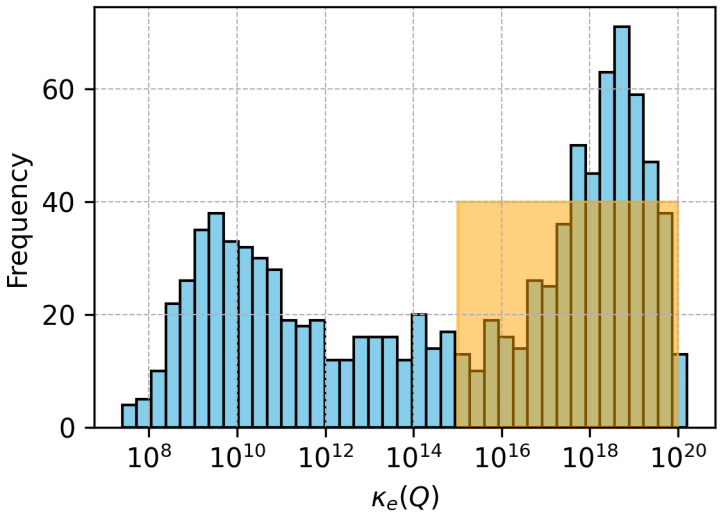


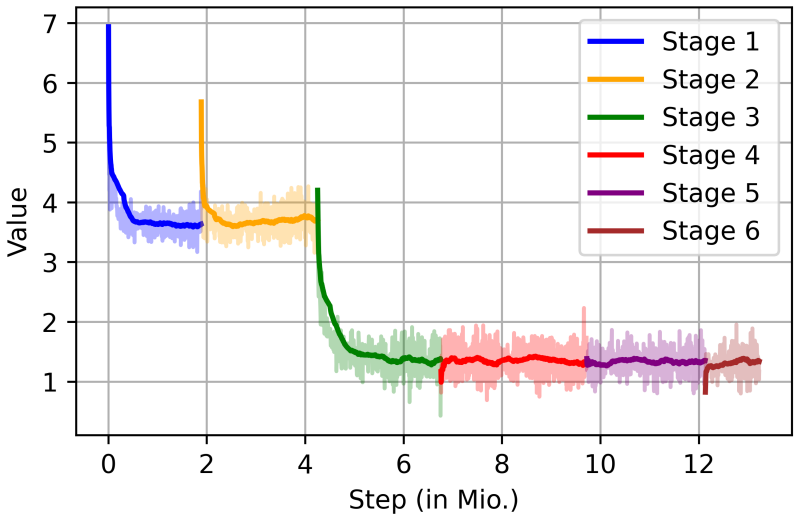
FIGURE 2.4: Distribution of the effective condition number κ_e (sample of 10^3 instances). Values for a log-uniform distribution spanning 10^{15} to 10^{20} (orange) when applied to random problem generation (blue).

The training progress can be observed in Fig. 2.5. In these figures, the different training phases (see Table 2.4) are indicated by different colors. In Fig. 2.5a it can be observed how the loss of the critic evolves.

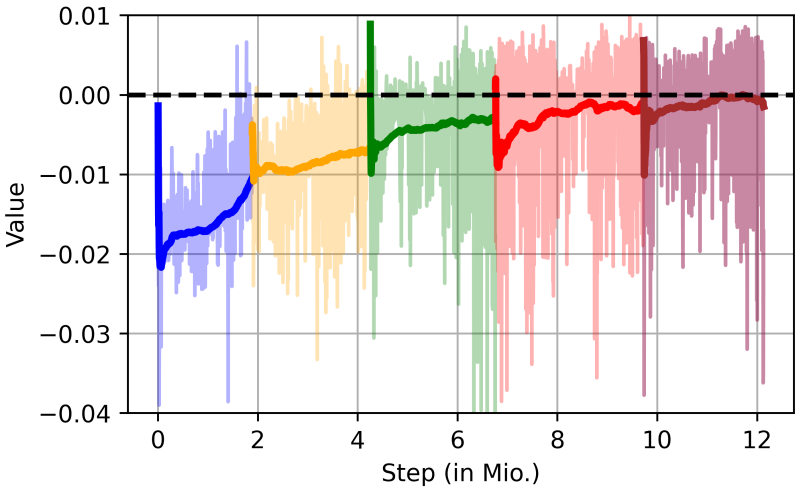
Especially in the first three phases, a significant reduction of the loss can be noticed. This indicates that the approximation of the value function improves. In the last three phases, the PPO algorithm fine-tunes this approximation. The reason for the jumps of the loss function between phases lies in the change of the reward function, which is typical for curriculum learning. In addition to the loss function, Fig. 2.5b shows the policy loss, which is an indicator of the improvement of the policy. The mostly negative values show that the policy improves, since advantageous actions, which means that they have a positive advantageous function approximation, are reinforced, and disadvantageous actions are weakened. Furthermore, we plot the entropy during training in Fig. 2.5c. Entropy can be used to determine how uncertain or exploratory the training is. We see that the entropy increases in the first and second phases, which is a typical behavior, since exploration is needed at the beginning of the training. Afterwards, the entropy decreases and converges to a certain level. Although this behavior is common for RL algorithms in general, it is noticeable that the final level of the entropy is relatively high. This means that the final policy is relatively uncertain and is not close to a deterministic policy. The investigation of whether an enhanced state space or even longer training phases can make the policy more certain is left for future works. In any case, the plots indicate a successful training, and the influence of the RL policy on the QP solver is discussed in the remainder of this manuscript.

2.4.2 Validation

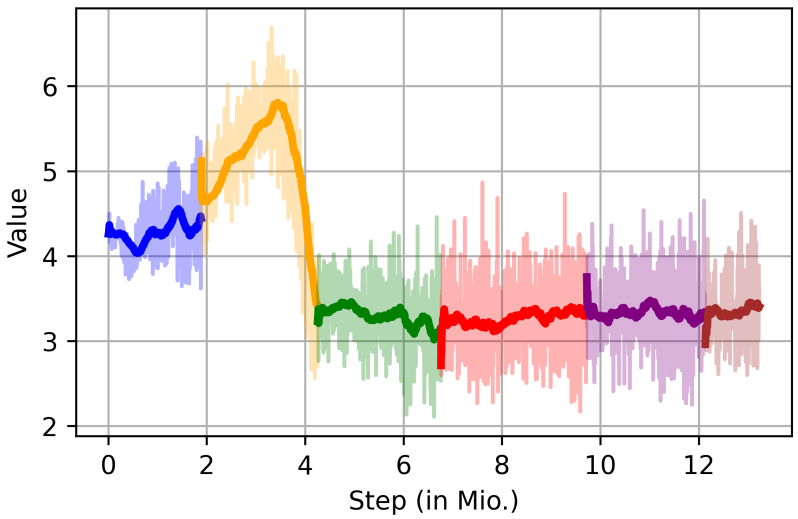
After the multi-staged training, the performance of the method is evaluated on a batch of 4096 problems having the same properties as the problems used for the training, (see Table 2.3). In Figure 2.6, the performance and data profiles of the RL-enhanced solver are compared against different solver configurations with fixed $\alpha_\square$ values, which were obtained



(A) Value function loss.



(B) Policy loss.



(C) Entropy.

FIGURE 2.5: Evolution of relevant metrics during the multistage training procedure..

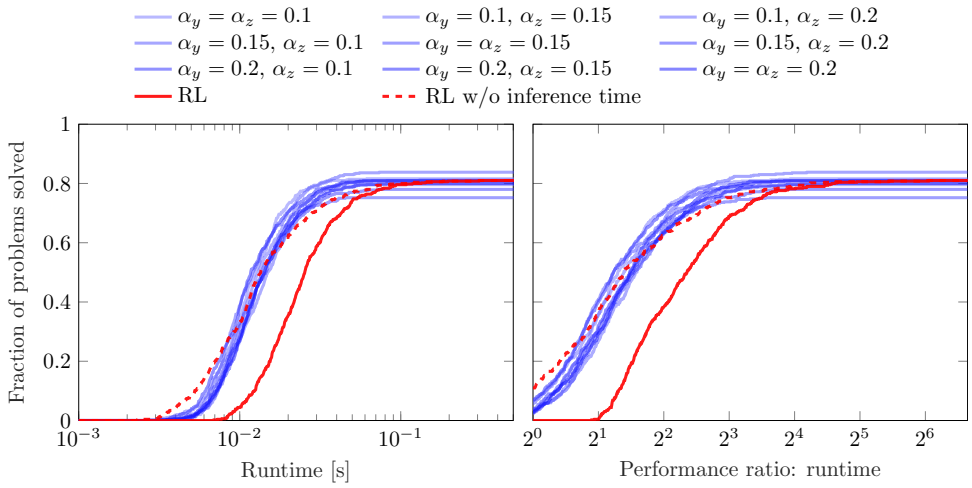


FIGURE 2.6: Validation set: Data (left) and performance (right) profiles comparing RL and solver configurations with fixed $\alpha_x = 0.15$, α_λ and α_μ.

by extensive hand-tuning to achieve best performance. The fixed values of $\alpha_\square$ correspond to combinations of the values $\{0.1, 0.15, 0.2\}$. For these fixed settings, the robustness difference is about 10%, a significant margin for QPs, where state-of-the-art methods are already highly optimized and studied. The RL-enhanced solver is shown with its total runtime (red solid curve) and excluding the neural network inference time (red dashed line). Since the training problems are very small (see Table 2.3), the inference time of up to 0.1 seconds is non-negligible. The data profile (left panel) illustrates solving times ranging from 10^{-3} to 0.5 seconds, highlighting the diversity of problem complexity in the batch. In terms of performance, considering solving time only (dashed red line), the RL approach identifies sequences of $\alpha_{\square,k}$, $k \in \{0, 1, \ldots, \text{max-outer-it}\}$ (and correspondingly $\delta\square$), enabling the solver to achieve the fastest solving times on approximately 25% of the test problems. In terms of robustness, the RL-enhanced method lies in the upper range of the tested solvers.

In Figure 2.7, the mapping $\sigma(x, \lambda, \mu, \nu, \varepsilon) \to \alpha_\square$ is shown for fixed values $\nu = 10^{-2}$ and $\varepsilon = 10^{-2}$. The mappings are scaled by $-\log_{10}(\cdot)$, and can be read from the bottom-left corner to the upper-right corner, with a termination tolerance of $\epsilon = 10^{-6}$. The actions $\alpha_\square$ span the interval $[0.05, 0.95]$, ensuring a consistent decrease; see (2.1). This mapping still represents an early phase of the optimization process, and is typically observed after 4–5 outer iterations with good convergence (see κ_ν and θ_ν in Table 2.2). Consequently, noise is present for larger primal and dual residuals, since little to no data is available in these regions with the used early termination and the barrier parameter schedule. For the proximal weight δ_x decrease (left), very small values are obtained in the quadrant where the primal residual is smaller than the dual residual, while high values are observed in the opposite case. A smooth transition appears across the diagonal $r_{\text{prim}} = r_{\text{dual}}$. Within this region ("tube") where primal and dual residuals are of similar magnitude, the values of $\alpha_\square$ fall within $[0.2, 0.5]$, which corresponds to the typical range used for fixed α-decrease schemes and serves as a useful benchmark for validating the RL-enhanced solver. For the equality-constraint weight δ_λ (center), a simple pattern emerges, with α_λ taking minimal values almost everywhere. This reflects the comparatively lower complexity of handling equality constraints relative to inequality constraints. For the inequality-constraint weight δ_μ (right), a more intricate pattern is observed. The gradient forms a valley along the tube, where the values change more gradually than in the case of the proximal weight. Additionally, the symmetry along the "tube" of the α_x mapping is reduced but still discernible.

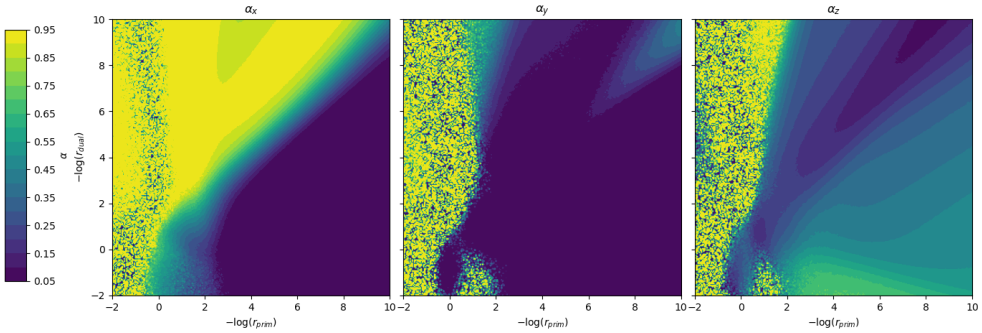


FIGURE 2.7: Mappings α_x (left), α_λ (middle) and α_μ (right): learned policy π_ζ evaluated with fixed $\nu = 10^{-2}$ and $\varepsilon = 10^{-2}$.

2.4.3 Multistage learning and in-training validation

In Figure 2.8, the learned policies at training stages 1, 2, 3 and 6 performances are compared on the validation set (see Table 2.4). Between the first (black) and second (grey) stages, a substantial acceleration of the solving process is observed: the policy obtained after stage 2 consistently achieves faster solving times across all problems compared to stage 1. This improvement results from the introduction of the performance term in the reward, building on the stable neural network structure established during the first stage. On stage 3, where only the performance term is retained, the policy yields a further reduction in solving time but at the cost of approximately a 3% decrease in robustness. Continued training up to stage 6 produces the best results, with the final policy achieving both the fastest solving times and the highest robustness among the tested configurations.

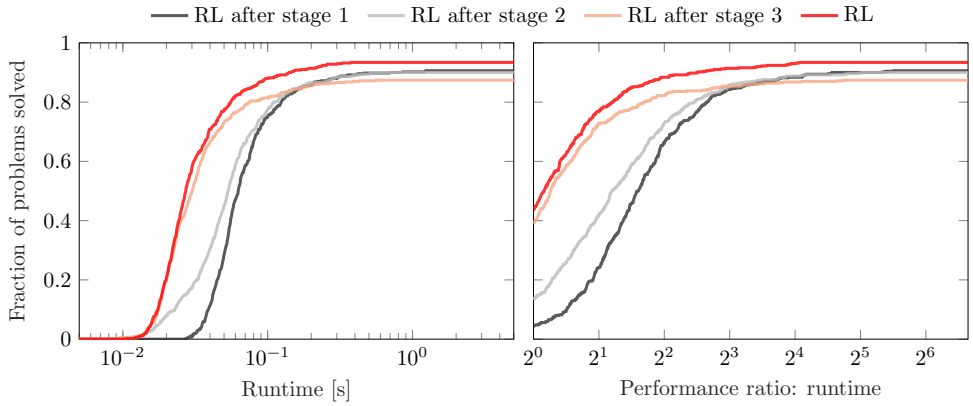


FIGURE 2.8: In-training validation: policy performance along the multistage process. Data (left) and performance (right) profiles comparing the influence of different RL policy.

2.4.4 Problem dimensions scaling

When evaluating the RL-policy on the validation set, the observed improvements over fixed $\alpha_\square$ strategies remain modest. This is largely due to the relatively small size of the validation problems, which, although ill-conditioned, non-strictly convex, and subject to redundant constraints, still exhibit limited complexity. However, the choice of random problem sizes during training (see Figure 2.2 for a comparison of problem properties) was primarily motivated by the need to keep the computational training cost low enough to complete on a

personal computer within a single day; see the system configuration in Table 2.1. A question then arises: how does the policy π_ζ, trained at low cost, perform on larger problem instances?

Figure 2.9 presents the results for a batch of problems whose dimensions and sparsity are scaled by a factor of 10 relative to the validation set. As in the previous performance and data profiles, the solid red curve corresponds to the RL-enhanced solver, while the dashed red curve excludes the neural-network inference time. With larger problems, the inference overhead becomes negligible for the vast majority of cases (up to a performance ratio of 1.1). On these scaled problems, the RL-enhanced method solves most convergent instances faster than the standard geometric strategies across all tested $\alpha_\square$-decrease schemes. Notably, its performance ratio reaches 2^4 for 80% of the problems (and 98% of those solved), underscoring the method's ability to generalize and provide substantial gains on larger-scale instances.

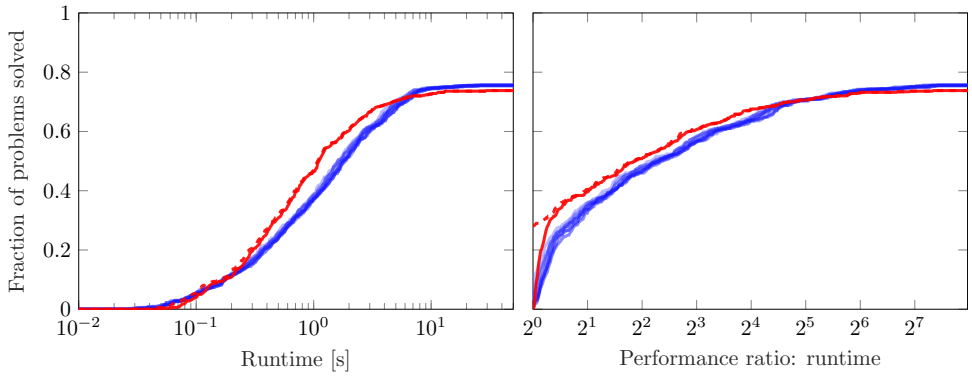


FIGURE 2.9: Scale 10× set: Data (left) and performance (right) profiles comparing RL and solver configurations with fixed $\alpha_x = 0.15$, α_λ and α_μ.

2.4.5 Effective condition Number of the KKT matrix

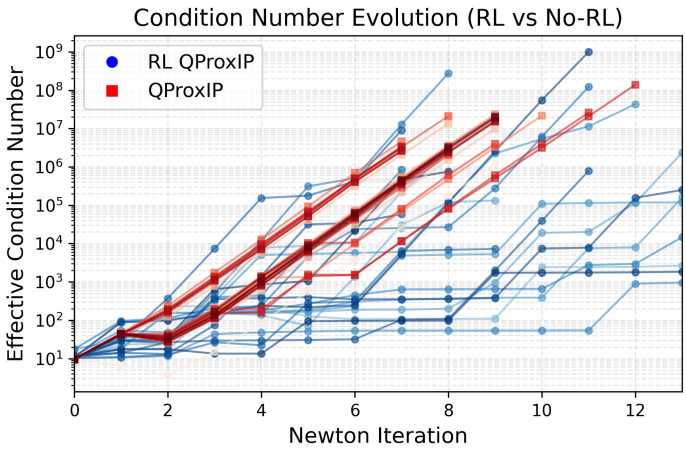


FIGURE 2.10: Evolution of the effective condition number of the KKT matrix across iterations for RL-QProxIP (blue) and standard QProxIP (red) strategies.

Figure 2.10 illustrates that the reinforcement learning (RL)-enhanced strategy effectively mitigates the growth of the effective condition number of the KKT matrix throughout the iterations in most cases. The random problem instances are sampled from the same distribution as those used during training. Although the first iteration occasionally exhibits higher condition numbers due to initial aggressive weight updates of the RL strategy, the subsequent

evolution remains markedly better conditioned. Overall, the RL-based enhancement leads to condition numbers that are most often better, or at least equivalent, to those of the standard approach, underscoring its stabilizing effect in limiting the numerical ill-conditioning typically observed otherwise.

2.4.6 Maros-Mészáros test-set

In the following, two variants of the `QProxIP` solver are considered to evaluate the impact of reinforcement learning (RL) acceleration. The first, referred to as *ablated* `QProxIP`$^-$ solver, where all additional safeguards are removed to isolate the pure effect of RL acceleration within a simplified setting. The second corresponds to the standard `QProxIP` solver, presented in the previous chapter, which includes the full set of safeguards and algorithmic protections to ensure robust convergence.

ID	ALM	x-prox	s-prox	CPC	NML-LS	Ruiz	Amort.	Bound. Ctrl
RL–`QProxIP`$^-$	yes	yes	no	no	no	no	no	no
RL–`QProxIP`	yes	yes	no	yes	yes	yes	yes	yes

TABLE 2.5: Applied regularization and safe-guards applied in the subsection.

Table 2.5 summarizes the differences between RL–`QProxIP`$^-$ and RL–`QProxIP`, including no central-path correction, no non-monotonous linesearch, no Ruiz precodnitioning and no Boundary control for RL–`QProxIP`$^-$.

2.4.6.1 Ablated RL–`QProxIP`$^-$ Solver (without Safeguards)

Training was carried out on randomly generated problem instances, but an important question is how well the method generalizes to established benchmarks. To this end, we evaluate on the Maros–Mészáros test set [90], a widely used benchmark collection of challenging quadratic programs. This suite contains many large-scale and ill-conditioned problems and is considered a standard for assessing the robustness and efficiency of convex QP solvers. For evaluation, we consider the problem in a zero-shot generalization setting and focus on the 100 smallest instances (out of 138), where problem size is determined by folder size. A comparison of these instances with the randomly generated problems used for training is shown in Figure 2.2. As before, the Maros–Mészáros set is used in its raw form, meaning that inequality constraints with "infinite" bounds (represented as values of 10^{20} in OSQP [119]) are retained in the problem data, see Problem 1.3.2. To accommodate the increased difficulty of these benchmarks, both the maximum number of inner and outer iterations are increased to 50.

The corresponding performance and data profiles are reported in Figure 2.11. The RL-enhanced solver is compared against fixed choices $\alpha_{x,\lambda,\mu} = 0.1$, $\alpha_{x,\lambda,\mu} = 0.2$, and $\alpha_{x,\lambda,\mu} = 0.5$, both with and without accounting for neural-network inference time. The RL-enhanced solver demonstrates improvements in robustness of approximately 2%, 4%, and 7% relative to the fixed-α curves. As previously observed in the scale 10× experiments, the RL-enhanced solver achieves speed-ups for problems requiring at least one second of solving time. When excluding inference time, its performance on the data profile matches the best fixed-α values for smaller problems solved in under 0.1 seconds. Although the Maros–Mészáros problems are not part of the training set, the RL solver identifies meaningful actions. Depending on the

problem and solver configuration, finding an effective set of fixed $\alpha_\square$ values can be tedious, and both robustness and solution speed are highly sensitive to this choice. The test set is particularly diverse, with some problems solved in milliseconds and others requiring up to 30 seconds, making it challenging to select a single fixed $\alpha_\square$ that performs well across the entire set. In contrast, the RL-enhanced method exhibits versatility, maintaining consistently high performance across the benchmark.

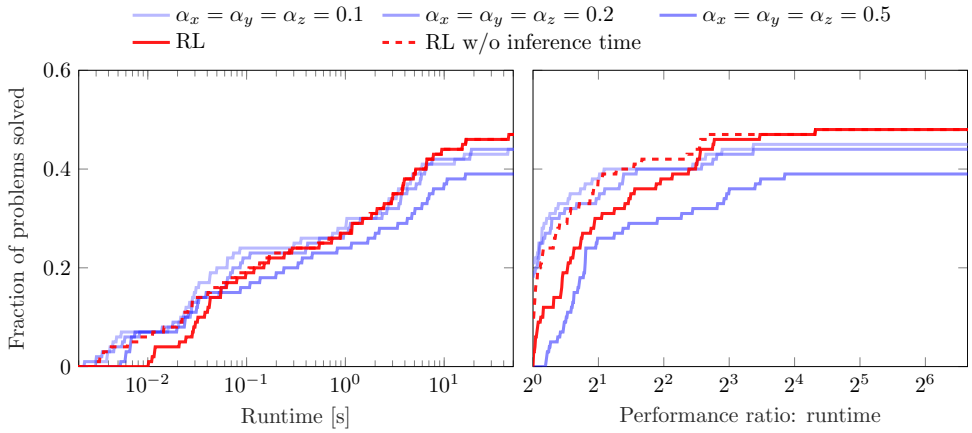


FIGURE 2.11: Maros–Mészáros set, MM100: Data (left) and performance (right) profiles comparing RL and solver configurations with fixed $\alpha_x = \alpha_\lambda = \alpha_\mu$. Despite the lightweight training on random problems, the learned RL policy performs well on the previously unseen class of Maros–Mészáros instances.

Depending on the problem's conditioning and the choice of initial candidate, it may be beneficial for the user to tune the early-termination scaling factor ξ. This parameter determines the starting point for a fixed geometric decrease of ε_k and should be chosen consistently with the geometric decay rate $\alpha_\square$ to achieve the best performance.

Numerical results illustrating this effect are presented in Figure 2.12, where the RL-enhanced solver is evaluated for $\xi \in \{10^{-3}, 10^{-5}, 10^{-7}\}$ along with the default value $\xi = 10^{-2}$ for which the policy was trained. Even under a poorly chosen solver setup, the RL-enhanced solver adapts to maintain performance comparable to that obtained under a well-configured setup. Furthermore, all RL curves remain more robust than the fixed $\alpha_\square$ strategies, even when the latter are used with the best solver configuration ($\xi = 10^{-2}$). This versatility makes the solver more robust and easier to use, as it can automatically adapt to a wide range of problem instances. Examples include problems with large variations such as financial portfolio optimization, model predictive control under high perturbations and sudden noise and changes, large-scale power system optimization, trajectory planning for robotics in dynamic environments, and parameter estimation in ill-conditioned or highly constrained systems.

2.5 Chapter conclusion and outlook

We presented a method based on reinforcement learning (RL) to speed up and harden the convergence of a (regularized Newton-based) solver for convex quadratic programming. We used RL to learn a policy aimed at adapting the solver's hyperparameters to reduce the number of iterations needed to obtain a solution. For the RL training, we generated a framework

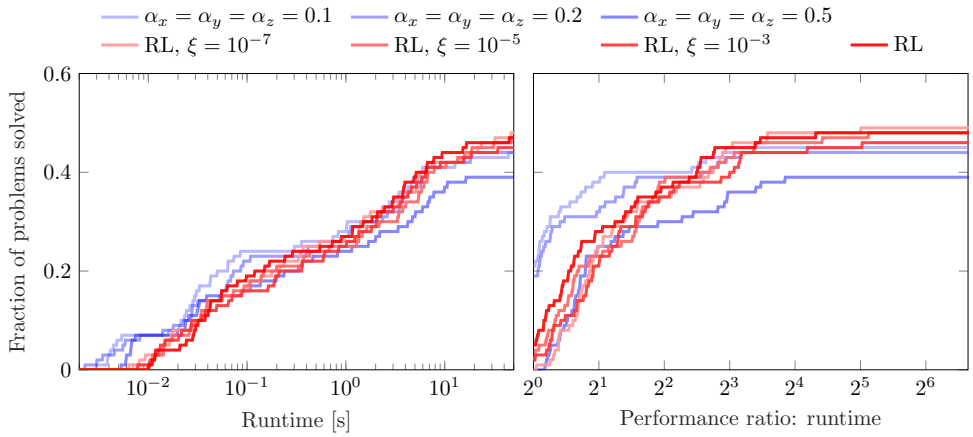


FIGURE 2.12: Maros–Mészáros set: Data (left) and performance (right) profiles comparing the influence of the RL policy on solver configurations with different values for ξ. The results with the default solver setup ($\xi = 10^{-2}$) and with fixed $\alpha_x = \alpha_\lambda = \alpha_\mu$ are provided for comparison. The learned RL policy adapts well to different solver tuning, yielding consistent performance gains.

to design small random QPs, which serves as good training problems. Then the parameterized RL policy, which is constructed as an artificial neural network, is trained on these QPs. In experiments, the results suggest that a single policy can improve convergence rates for a broad class of problems. In particular, we would like to emphasize the fact that the policy can successfully be used for problems with different sizes, scales and structures. Compared to previous approaches, like RLQP [70, §6], our lightweight training achieves similar validation performance with a fraction of time and memory (13 hours on a laptop vs. days on a high-end computer). Only for QPs that converge after few iterations, and thus require no hyperparameters adaptation, having a good neural policy is unnecessary at best. In these cases, the inference time it takes to evaluate the RL policy may reduce the performance benefit of faster convergence.

Despite these promising results, some questions remain for future works. For example, we will investigate whether additional RL policy options can further accelerate the convergence rate. Thereby, one aspect of investigation could be to measure the influence of the barrier parameter ν and the tolerance ε on the policy or the effect of including additional states, like the regularization parameter δ.

Furthermore, we will investigate the policy performance to other QP problem classes, such as large-scale sparse problems arising in power systems, optimal control problems showing particular three-banded structures or financial optimization with highly volatile market data. Moreover, for even more adaptive RL approaches, the policy could continue learning and improving during solver deployment. Lastly, it would also be valuable to benchmark the method against other state-of-the-art solvers, by extending with RL the in-house solver capable of solving the entire Maros–Mészáros test set.

Chapter 3

Nonlinear Programming: Sequential Quadratic Programming (SQP)

3.1 Introduction

Sequential Quadratic Programming (SQP) is a powerful framework for solving constrained nonlinear nonconvex optimization problems. By iteratively solving QP subproblems that locally approximate the nonlinear program, SQP methods exhibit fast local convergence and a well-established theoretical foundation, [47, 96].

Beyond their success in general nonlinear programming, SQP methods play a central role in optimal control and have been widely integrated into Model Predictive Control (MPC) frameworks [64, 106]. The SQP problem formulation matches the structure of optimal control problems, which include dynamic system as equality constraints and stage-wise costs, making SQP methods well suited for control implementation.

Early practical implementations include the real-time and large-scale variants proposed in [12], which address computational efficiency and robustness. Subsequent developments have focused on ensuring global convergence and numerical stability, often through the incorporation of globalization strategies such as filter methods, merit functions, or trust-region mechanisms [47, 54].

More recently, hybrid schemes combining SQP with interior-point techniques have been studied to leverage the advantages of both approaches—particularly the ability of interior-point methods (IPMs) to handle inequality constraints efficiently and maintain feasibility [4, 13, 98]. Despite their promising convergence properties, such IP-SQP algorithms still face challenges related to warm starting, especially when adapting the barrier parameter between successive solves. Yet, an effective heuristic for adjusting the barrier parameter between SQP-iterations remains unknown, often causing instability or slow progress between iterations.

At each iteration, SQP methods locally approximate the nonlinear programming (NLP) problem by a quadratic programming (QP) subproblem, whose solution provides a search direction for both primal and dual variables. Formally, we consider the general nonlinear program:

Problem 3.1.1 NLP

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & h(x) = 0, \\ & g(x) \le 0, \end{aligned} \tag{NLP}$$

where

$$\begin{aligned} f &: \mathbb{R}^n \to \mathbb{R} && \text{objective function,} \\ h &: \mathbb{R}^n \to \mathbb{R}^m && \text{equality constraints,} \\ g &: \mathbb{R}^n \to \mathbb{R}^p && \text{inequality constraints.} \end{aligned}$$

We assume throughout that f, h, g are twice continuously differentiable.

The feasible set is:

$$\mathcal{F} = \{x \in \mathbb{R}^n \mid h(x) = 0,\ g(x) \le 0\}. \tag{3.1}$$

3.2 Lagrange Newton method

We begin by considering a nonlinear optimization problem with equality constraints only,

Problem 3.2.1 NLP-EQ

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & h_i(x) = 0, \quad i = 1, \dots, m. \end{aligned} \tag{3.2}$$

A point $(x^*, \lambda^*) \in \mathbb{R}^n \times \mathbb{R}^m$ is called a Karush–Kuhn–Tucker (KKT) point of Problem (3.2.1) if it satisfies

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = 0 \in \mathbb{R}^n, \tag{3.3}$$

$$h_i(x^*) = 0, \quad i = 1, \dots, m, \tag{3.4}$$

where

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^\top h(x) \tag{3.5}$$

denotes the Lagrangian function and $\nabla_x \mathcal{L}$ its gradient with respect to the primal variable x.

Conditions (3.3)–(3.4) form a nonlinear system of equations in the variables (x, λ). Defining the mapping

$$F : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^{n+m}, \qquad F(x, \lambda) = \begin{pmatrix} \nabla_x \mathcal{L}(x, \lambda) \\ h(x) \end{pmatrix}, \tag{3.6}$$

the KKT conditions can be written compactly as

$$F(x^*, \lambda^*) = 0. \tag{3.7}$$

The Lagrange-Newton method applies Newton's method to the nonlinear system (3.7) to determine a null-space F.

Given a current iterate $(x^{(k)}, \lambda^{(k)})$, the Newton step $(\Delta x^{(k)}, \Delta\lambda^{(k)})$ is obtained by solving

$$\begin{pmatrix} \nabla^2_{xx}\mathcal{L}(x^{(k)}, \lambda^{(k)}) & \nabla h(x^{(k)})^\top \\ \nabla h(x^{(k)}) & 0 \end{pmatrix} \begin{pmatrix} \Delta x^{(k)} \\ \Delta\lambda^{(k)} \end{pmatrix} = -\begin{pmatrix} \nabla_x \mathcal{L}(x^{(k)}, \lambda^{(k)}) \\ h(x^{(k)}) \end{pmatrix}. \tag{3.8}$$

Equation (3.8) corresponds to the optimality system of an equality-constrained quadratic program arising. This interpretation directly links the Lagrange–Newton method to equality-constrained quadratic programming and allows for a straightforward extension to inequality constraints.

3.3 Sequential Quadratic Programming

Sequential Quadratic Programming (SQP) can be interpreted as an extension of the Lagrange–Newton method to optimization problems with both equality and inequality constraints. SQP methods have been extensively studied in the literature, see [49, 96]. We therefore consider Problem (3.1.1) as an extension of Problem (3.2.1) and write the Lagrangian

$$\mathcal{L}_\rho(x, \lambda, \mu) = f(x) + \lambda^\top h(x) + \mu^\top g(x). \tag{3.9}$$

We construct the local approximation of the Problem (3.1.1) at iteration k by a Quadratic Problem (QP), see [12]. At each SQP iteration a local QP subproblem is defined and solved, see the SQP algorithm overview (8).

Algorithm 8 Sequential Quadratic Programming (SQP): Overview.

1: Initialize x^0, multipliers (λ^0, μ^0), set $k = 0$
2: **while** not converged **do**
3: Solve the quadratic subproblem at k
4: Apply the search directions Δ
5: $k \leftarrow k + 1$
6: **end while**
7: **Return** $(x^{k+1}, \lambda^{k+1}, \mu^{k+1})$

The Lagrangian of Problem (3.1.1) around the current iterate $x^{(k)}$ is expanded to second order as

$$\begin{aligned} \mathcal{L}(x^{(k)} + \Delta x, \lambda^{(k)}, \mu^{(k)}) \approx & \mathcal{L}(x^{(k)}, \lambda^{(k)}, \mu^{(k)}) + \nabla_x \mathcal{L}(x^{(k)}, \lambda^{(k)}, \mu^{(k)})^\top \Delta x \\ & + \tfrac{1}{2} \Delta x^\top \nabla^2_{xx} \mathcal{L}(x^{(k)}, \lambda^{(k)}, \mu^{(k)})\, \Delta x. \end{aligned} \tag{3.10}$$

where only the primal variables are expanded and which serves as the quadratic model of the local subproblem objective. To ensure consistency with the current iterate, the nonlinear constraints are replaced by their first-order linearizations:

$$h(x^{(k)}+\Delta x) \approx h(x^{(k)}) + A^{(k)}\Delta x, \qquad A^{(k)} := \nabla h(x^{(k)})^\top,$$
$$g(x^{(k)}+\Delta x) \approx g(x^{(k)}) + G^{(k)}\Delta x, \qquad G^{(k)} := \nabla g(x^{(k)})^\top.$$

The resulting Sequential Quadratic Programming (SQP) subproblem can thus be written as in Problem (3.11).

Problem 3.3.1 — **SQP-QP**

$$\begin{aligned} \underset{\Delta x\in\mathbb{R}^n}{\text{minimize}} \quad & \tfrac{1}{2}\Delta x^\top Q^{(k)}\Delta x + c^{(k)\top}\Delta x \\ \text{subject to} \quad & A^{(k)}\Delta x + h(x^{(k)}) = 0, \\ & G^{(k)}\Delta x + g(x^{(k)}) \le 0, \end{aligned} \tag{3.11}$$

where $Q^{(k)} = \nabla^2_{xx}\mathcal{L}(x^{(k)},\lambda^{(k)},\mu^{(k)})$ and $c^{(k)} = \nabla_x f(x^{(k)})$.

This QP provides a search direction $\Delta x^{(k)}$ toward a new iterate $x^{(k+1)} = x^{(k)} + \Delta x^{(k)}$. The SQP iterations correspond to updates of the nonlinear problem, while the QP iterations correspond to the solution process of this QP subproblem. For consistency purposes with chapter 1, this QP can equivalently be written in a compact form as

Problem 3.3.2 — **SQP-QP - reformulation**

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & \tfrac{1}{2}x^\top Qx + c^\top x \\ \text{subject to} \quad & l \le Ax \le u. \end{aligned} \tag{SQP-QP}$$

Theorem 3.3.1 provides a fundamental local characterization of SQP subproblems in a neighborhood of a KKT point,see [96][Thm 18.1]. It states that, under standard regularity assumptions (LICQ, strict complementarity, and second-order sufficient conditions) the quadratic programming subproblem associated with SQP admits a local solution whose active set coincides with that of the original nonlinear program. This property is essential for the local convergence theory of SQP methods, as it ensures that, sufficiently close to the solution, the SQP subproblem accurately reflects the local structure of the NLP and enables fast local convergence.

Theorem 3.3.1 **Local Solution of SQP Subproblems**

Suppose that x^* is a local solution of a NLP at which the KKT conditions are satisfied for some multiplier λ^*, μ^*. Assume that the following conditions hold at (x^*, λ^*, μ^*):

- the Linear Independence Constraint Qualification (LICQ),
- the strict complementarity condition,
- and the second-order sufficient conditions.

Then, if $(x^{(k)}, \lambda^{(k)}, \mu^{(k)})$ is sufficiently close to (x^*, λ^*, μ^*), there exists a local solution of the QP subproblem whose active set coincides with the active set of the original NLP at x^*, see [96], Thm 18.1. For a detailed discussion of the QP optimality conditions, see § 1.3.6.

Algorithm 9 Sequential Quadratic Programming (SQP).

Initialize primal variables x^0, multipliers (λ^0, μ^0), and set tolerances $\varepsilon > 0$
for $k = 0, 1, 2, \ldots$ **do**
  Evaluate derivatives at x^k: $c^k := \nabla f(x^k)$, $A^k := \nabla h(x^k)^\top$, $G^k := \nabla g(x^k)^\top$
  Globalization: Form BFGS Update, see [47]
  Solve the QP subproblem (SQP-QP) to obtain $(\Delta x^k, \Delta \lambda^k, \Delta \mu^k)$
  Select step size $\alpha^k \in (0, 1]$ (filter method or linesearch)
  Update iterates: $x^{k+1} \leftarrow x^k + \alpha^k \Delta x^k$, $\lambda^{k+1} \leftarrow \lambda^k + \alpha^k \Delta \lambda^k$, $\mu^{k+1} \leftarrow \max\{0, \mu^k + \alpha^k \Delta \mu^k\}$
  if $\|\nabla_x \mathcal{L}(x^{k+1}, \lambda^{k+1}, \mu^{k+1})\| \leq \varepsilon$ **and** constraints satisfied within tolerance **then**
    Return $(x^{k+1}, \lambda^{k+1}, \mu^{k+1})$
  end if
end for

Algorithm 9 presents the Sequential Quadratic Programming (SQP) method for solving smooth nonlinear nonconvex constrained optimization problems, see [47] for the formal methods and implementations. The procedure begins with an initialization of the primal variables x^0, the Lagrange multipliers (λ^0, μ^0), and a user-defined termination tolerance ε. At each iteration, the algorithm evaluates the gradient of the objective function, denoted by $c^k = \nabla f(x^k)$, and the Jacobians of the equality and inequality constraints, $A^k = \nabla h(x^k)^\top$ and $G^k = \nabla g(x^k)^\top$, respectively. These derivatives provide the local linear approximation of the constraint set around the current iterate.

Next, instead of computing the exact Hessian of the Lagrangian, a BFGS update is used to construct a positive definite approximation $B^k \approx \nabla^2_{xx} \mathcal{L}(x^k, \lambda^k, \mu^k)$, see [43]; which captures the local curvature information of the problem [47]. Using BFGS is more robust in practice, because it guarantees a positive definite approximation even when the true Hessian is singular or nearly singular, preventing numerical instabilities in the QP subproblem. However, the BFGS globalization is only suitable for non-sparse matrices. With these quantities, a quadratic programming (QP) subproblem is formulated that approximates the original nonlinear problem near x^k. Solving this QP provides the search directions $(\Delta x^k, \Delta \lambda^k, \Delta \mu^k)$, which approximate a Newton step for the Karush–Kuhn–Tucker (KKT) system. If the Hessian is negative definite or singular, globalization methods can be applied like the BFGS update [43, 96].

A step size $\alpha^k \in (0, 1]$ is then chosen, typically using a linesearch or a filter strategy, to guarantee sufficient progress toward optimality while maintaining feasibility. The iterates are

subsequently updated according to $x^{k+1} = x^k + \alpha^k \Delta x^k$, $\lambda^{k+1} = \lambda^k + \alpha^k \Delta \lambda^k$, and $\mu^{k+1} = \max\{0, \mu^k + \alpha^k \Delta \mu^k\}$ to ensure nonnegativity of the inequality multipliers. This process is repeated until the norm of the Lagrangian gradient is below the tolerance ε and the constraints are satisfied within acceptable limits. Upon convergence, the algorithm returns the triplet $(x^{k+1}, \lambda^{k+1}, \mu^{k+1})$, which approximates a KKT point of the original problem.

Remark 3.3.1 — **L-1 norm regularization**

In practice the Lagrangian (3.9) is regularized using an L-1 norm of the form

$$\mathcal{L} \leftarrow \mathcal{L} + \frac{\rho}{2} \left(\|h(x)\|_1 + \|g(x)_+\|_1 \right).$$

3.4 Implementation

In practice, Sequential Quadratic Programming (SQP) methods differ primarily in how they approximate second-order information, treat inequality constraints, and ensure global convergence. The Hessian of the Lagrangian, $Q^{(k)} = \nabla^2_{xx} \mathcal{L}(x^{(k)}, \lambda^{(k)}, \mu^{(k)})$, may be obtained from exact second derivatives or approximated using quasi-Newton updates such as BFGS [12, 51, 54, 96]. These updates provide a positive-definite approximation to the true curvature, thereby maintaining local superlinear convergence while avoiding the computational expense of forming exact Hessians.

Different strategies exist for managing inequality constraints. Traditional active-set methods explicitly estimate the active set of constraints at each iteration, whereas interior-point SQP variants maintain feasibility by introducing slack variables and barrier term, see chapter 1.

3.5 Numerical results

Only a limited set of unconstrained and constrained nonconvex nonlinear problems is considered, serving as illustrative examples to give an overview of the SQP method used with a stabilized IPM-based QP solver.

Unconstrained NLP: Rosenbrock function: The function is defined as

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2, \tag{3.12}$$

where $x = [x_1, x_2]^\top \in \mathbb{R}^2$. The global minimum is located at $x^* = (1, 1)$, where $f(x^*) = 0$.

Despite its simple formulation, the Rosenbrock function presents a significant challenge for many optimization methods. Its narrow, curved valley makes the gradient smalls, which can cause slow convergence for gradient-based algorithms. The landscape is characterized by a flat region near the minimum and steep walls along the sides of the valley, leading to ill-conditioning and strong nonlinearity in the optimization trajectory.

Figure 3.2(left) illustrates the three-dimensional shape of the function, showing the characteristic curved valley and local minima. The gradient field shown in Figure 3.2(right) highlights the x_1, x_2 nonlinearity, demonstrating optimization oscillation in SQP method before converging to the global optimum.

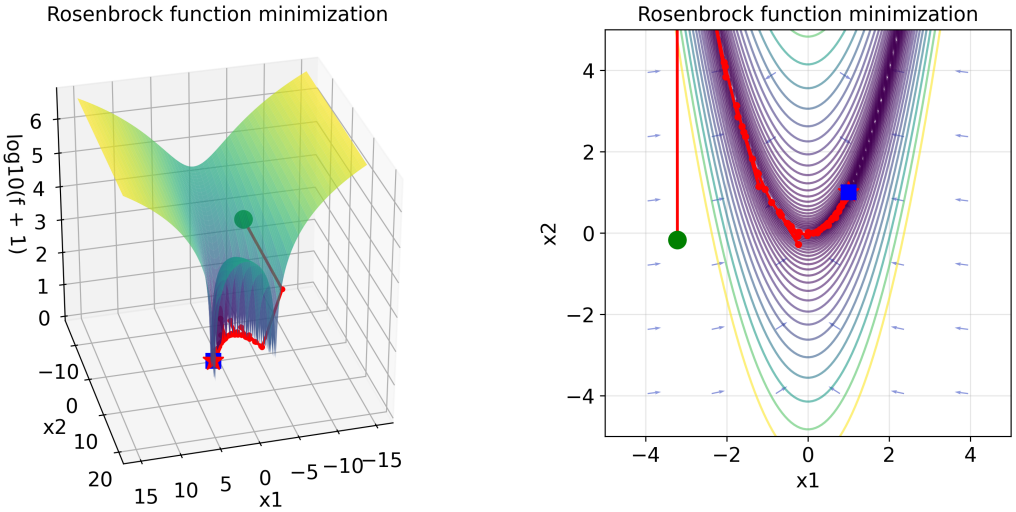


FIGURE 3.1: Visualization of the Rosenbrock function. Left: 3D surface illustrating the curved valley and local minima. Right: Contour plot on the (x_1, x_2) plane with the gradient field.

Unconstrained NLP: weighted quadratic-quartic sum function: This test problem is an unconstrained nonlinear programming (NLP) problem in 100 variables. The objective function is designed to combine a simple quadratic term with a nonlinear quartic term that couples all variables, introducing anisotropy and mild nonconvexity along a dominant direction. It serves as a benchmark to evaluate the performance of gradient-based and high-dimensional optimization algorithms, particularly in situations where certain directions in the search space contribute more strongly to the objective.

$$\begin{aligned} \min_{x\in\mathbb{R}^{100}} \quad & f(x) = 10^{-6}\left(\sum_{i=1}^{100} x_i^2 + \left(\frac{1}{2}\sum_{i=1}^{100} i x_i\right)^2 + \left(\frac{1}{2}\sum_{i=1}^{100} i x_i\right)^4\right) \\ \text{st:} \quad & x \in \mathbb{R}^{100} \end{aligned} \tag{3.13}$$

The function can be expressed more compactly as

$$f(x) = 10^{-6}\left(\sum_{i=1}^{100} x_i^2 + v_1^2 + v_1^4\right), \quad v_1 = \frac{1}{2}\sum_{i=1}^{100} i x_i, \tag{3.14}$$

where $x = [x_1, \ldots, x_{100}]^\top \in \mathbb{R}^{100}$. The global minimum is located at $x^* = 0$, where $f(x^*) = 0$.

The first term, $\sum x_i^2$, is a standard L2-norm squared penalty that encourages all variables to remain small, forming a spherical bowl in 100-dimensional space. The second and third terms depend on the weighted sum v_1, which gives higher-index variables more influence. The quadratic term v_1^2 introduces additional curvature along this direction, while the quartic

term v_1^4 creates a nonlinear ridge that increases steeply for large values of v_1. This combination makes the optimization landscape anisotropic: some directions are more sensitive than others, and the problem becomes nonconvex along the v_1 direction.

Although the problem is high-dimensional, it can be understood intuitively in a low-dimensional analogy as a "bowl with a ridge": the quadratic term forms a smooth basin, while the weighted quartic term creates a steep, elongated feature along a specific linear combination of variables. Such structures test optimization algorithms' ability to handle smooth but coupled and anisotropic landscapes efficiently.

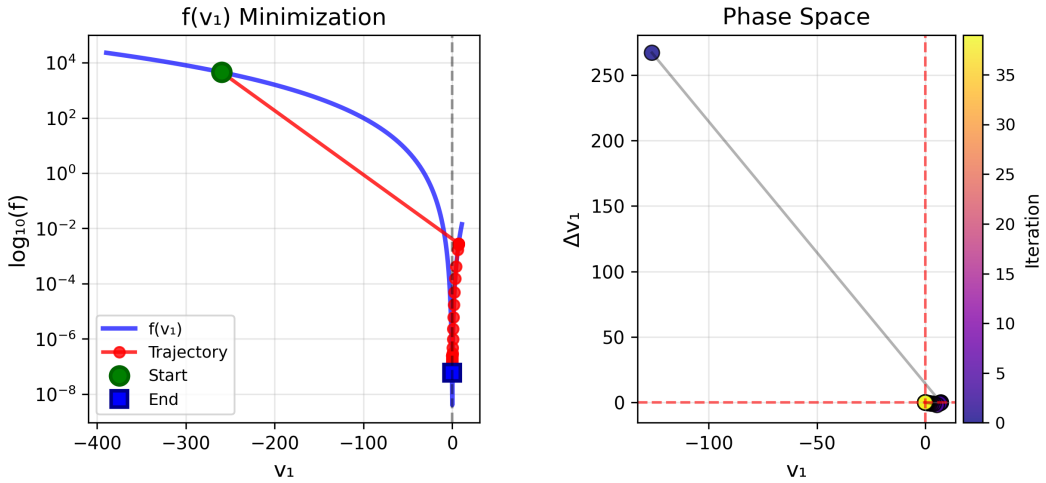


FIGURE 3.2: Weighted quadratic-quartic sum function. Left: $f(v_1)$ minimisation process. Right: Phase space $v_1 - \Delta v_1$.

Constrained NLP: degenerate saddle problem: The nonlinear optimization problem is defined as

$$\begin{aligned} \min_{x_1,x_2} \quad & f(x_1,x_2) = x_1^2 - x_2^2, \\ \text{s.t.} \quad & g(x_1,x_2) = x_1^2 + x_2^2 \le 1, \\ & -10 \le x_1 \le 10, -10 \le x_2 \le 10. \end{aligned} \tag{3.15}$$

where $x = [x_1, x_2]^\top \in \mathbb{R}^2$.

The SQP algorithm converges to a point on the boundary of the unit circle $x_1^2 + x_2^2 = 1$, as illustrated in Figure 3.3. The left panel shows the 2D projection with the feasible region and trajectory approaching the boundary, while the right panel displays the 3D surface of the objective function, highlighting the curved valley and the location of the constrained minimum. At the converged point, $x_1 \approx 0$ and $x_2 \approx \pm 1$, the objective function $f(x_1,x_2) = x_1^2 - x_2^2$ attains a local minimum $f_{\min} = -1$. The trajectory starts outside the feasible region and progressively satisfies the nonlinear constraint while minimizing the objective.

3.6 Chapter conclusion

In this chapter, we provided a brief overview of the theory and implementation of nonlinear programming (NLP) methods and illustrated their application by solving both unconstrained and constrained problems. These examples highlighted key concepts and typical solution

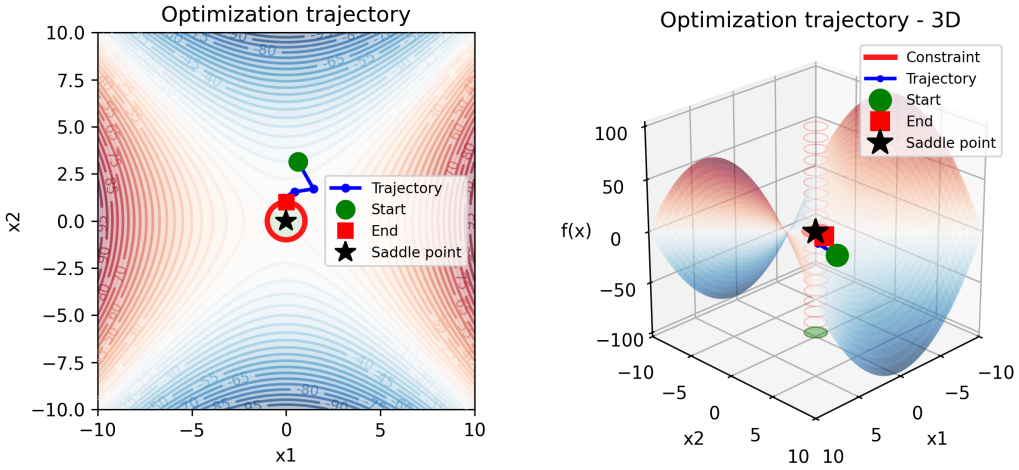


FIGURE 3.3: Visualization of constrained saddle point problem. Left: Contour plot on the (x_1, x_2) plane with the gradient field. Right: 3D surface illustrating the curved valley and local minima.

behavior, serving as a foundation for the construction and solution of more complex NLP problems. The insights gained here will guide the development of nonlinear, nonconvex optimal control problems in Part 2, where the focus will be on real-world application and real-time control implementation.

Conclusion Part I

This work provided a comprehensive overview of a competitive QP solver framework. In Chapter 1, we presented Interior Point Methods (IPM) for quadratic programming, including regularization, such as Proximal methods, classical augmented Lagrangian techniques, and primal-dual Gill-Robinson penalties. We introduced a stabilized Newton formulation (1.81), employing gradient scaling and formal proof that the transformed directions minimize a regularized merit function. Theoretical interpretations of regularization effects were provided, offering insight into the behavior of candidate solution trajectories. Chapter 3 complemented this discussion with an overview of nonlinear programming (NLP) methods, illustrated through unconstrained and constrained problems, laying the foundation for constructing more complex, nonconvex optimal control problems in Part 2, with a focus on real-time control applications.

On the methodological side in the first chapter, we introduced a novel s-proximal regularization to control slack variables and relax inequalities, with an analytical estimator θ_s serving as a proximal anchor. Central-path corrections and additional safeguards were implemented to maintain solution trajectories near the IPM central path. Chapter 2 extended these methods by incorporating reinforcement learning (RL) to adapt solver hyperparameters, accelerating convergence for convex QPs. The RL policy, trained on randomly generated small QPs and implemented as a lightweight neural network, is capable of accelerating the process of solving problems with varying sizes, scales, and structures.

The proposed frameworks and algorithms were validated through extensive experiments. In Chapter 1, `QProxIP` successfully solved the Maros–Mészáros test set with robustness and computational efficiency comparable to the best academic and commercial solvers. Performance variability was observed due to factors such as CPU core assignment, temperature, and other uncontrolled system conditions, underscoring the importance of large-scale benchmarking while highlighting that results cannot be universally generalized for solvers with complex heuristics. RL-enhanced QP solvers further demonstrated improved convergence rates across the Maros–Mészáros test-set using an ablated `QProxIP` solver without any safeguards and preconditioning. Together, these results establish a foundation for tackling the nonlinear, nonconvex optimal control problems explored in Part II.

Part II

Model Predictive Control and Applications to Path-Planning and Mission Management

Chapter 1

Optimal Control

1.1 Introduction

Optimal control theory provides a systematic framework for determining how a dynamical system should be influenced over time to achieve a desired goal while minimizing a prescribed performance criterion. The objective may reflect different aspects of system behavior, such as minimizing energy consumption, maximizing efficiency, or achieving a specific terminal state. In this sense, optimal control links the design of decision-making policies to the mathematical description of a system's evolution, allowing one to determine the best possible control input that balances performance and effort.

At its core, an optimal control problem couples optimization with the underlying differential equations that describe the system dynamics. The control inputs influence how the system evolves, and the optimization seeks those inputs that yield the most desirable behavior according to the chosen objective. Because most real-world systems are continuous in time, their dynamics are governed by ordinary or partial differential equations, which must be respected exactly or approximately during optimization.

Solving an optimal control problem numerically requires discretizing time into a finite grid and approximating the system dynamics through numerical integration. The continuous-time formulation is thereby transformed into a finite-dimensional optimization problem, often referred to as a nonlinear program. The grid resolution introduces a fundamental trade-off: finer discretization improves the accuracy of the solution but increases computational cost, while coarser grids reduce complexity at the expense of precision. Efficient algorithms therefore balance these competing aspects, ensuring reliable convergence within practical computation times.

The framework of optimal control has found widespread application across a diverse range of domains. In automotive systems, it underpins model predictive control for energy-efficient driving and autonomous vehicle motion planning. In aerospace engineering, optimal control governs trajectory optimization, guidance, and attitude control of aircraft and spacecraft. In the field of energy management, it enables optimal scheduling of distributed resources and battery systems to balance supply and demand. Even in finance, similar principles guide portfolio optimization and risk-sensitive decision-making over time. Across all these domains, the unifying goal remains the same: to determine optimal decisions that respect the dynamics of the system while achieving the most efficient or desirable performance possible.

A general continuous-time optimal control problem (OCP) is often formulated in the Bolza form:

$$\begin{aligned} \min_{x,u} \quad & J(x,u) = E_c(x(T)) + \int_0^T \ell_c(x(t),u(t))\,dt, \\ \text{subject to} \quad & \dot{x}(t) = f_c(x,u), \quad x(0) = \hat{x}, \\ & x(t) \in \mathcal{X}, \quad u(t) \in \mathcal{U}, \quad \forall t \in [0,T], \end{aligned} \tag{1.1}$$

where the state vector $x(t) \in \mathbb{R}^n$ and control vector $u(t) \in \mathbb{R}^m$ are bounded by a set of differential equations $f : \mathbb{R}^n \times \mathbb{R}^m \times [0,T] \to \mathbb{R}^n$ describing the system dynamic over a defined time $t \in [0,T]$. The cost function presents two terms, $\ell_c(x(t),u(t))$ the running cost over $t \in [0,T]$, and $E_c(x(T))$ the terminal cost evaluated at $t = T$. The initial state at $x(0)$ is denoted as $\hat{x}$.

The control can be made in **open-loop**, where the control function $u(t)$ is determined only based on the initial condition and does not adapt to disturbances or the real system, or in **closed-loop** (feedback), where the control is given by a function $u(t) = \kappa(x(t),t)$, which depends on the current state vector $x(t)$. Real-life problems require feedback loops as real-systems are never perfect and are often not fully described by the differential equations.

Optimal control problem (1.1) is written as a continuous system but needs to be discretized to be numerically solved. A standard discrete-time OCP is written as,

$$\min_{x_k,u_k} \quad J = E_d(x_N) + \sum_{k=0}^{N-1} \ell_d(x_k,u_k), \tag{1.2}$$

$$\text{subject to} \quad x_{k+1} = f_d(x_k,u_k), \quad x_0 = \hat{x}, \tag{1.3}$$

$$x_k \in \mathcal{X}, \quad u_k \in \mathcal{U}, \quad \forall k = 0,\dots,N-1. \tag{1.4}$$

The discretized ODEs are obtained using numerical integration methods (e.g., implicit Euler, explicit Euler, Runge-Kutta etc.). The discretization schemes are explicit and implicit. For example, for the forward Euler scheme, the next state is computed directly from the current state and control input,

$$x_{k+1} = x_k + h f_c(x_k,u_k), \tag{1.5}$$

where h is the time step. Explicit methods are simple and computationally efficient but can suffer from stability issues, especially for stiff systems or large time steps.

In contrast, implicit schemes, such as the backward Euler scheme, involve solving an equation where the next state appears on both sides

$$x_{k+1} = x_k + h f_c(x_{k+1},u_k). \tag{1.6}$$

Implicit methods are more stable, particularly for nonlinear dynamics, but they require solving a set of equations at each time step, which increases computational complexity.

1.2 Model Predictive Controller

Although the Model Predictive Control problem solved at each time-iteration is mathematically similar to optimal control problem (1.2), it should fundamentally be interpreted as a feedback controller. The fixed prediction horizon is, in general, not long enough to reach the goal state in a single optimization step, and therefore multiple iterations are required until convergence of the closed-loop system. Using MPCs, the optimal control problem is repeatedly solved online using the current measured or estimated state as the initial condition, either on a real system or on a simulator. Only the first control action of the optimized input sequence is applied to the system, while the remaining predicted states and control inputs are shifted and reused as a warm start for the next optimization problem. This receding-horizon strategy embeds the optimal control problem into a closed-loop feedback structure, enabling the controller to react to disturbances, model mismatch, and changes in the environment.

Figure 1.1 illustrates the basic structure of the MPC feedback. The optimizer computes a predicted state vector x^* and control vector input u based on the system model with a user-defined time horizon and time-grid. The first control input is applied to the simulator or real-world process, which output a real system state vector $\hat{x}$. For each control iteration, changes in the environment or external parameters are updated prior to trajectory optimization and fed back to the optimizer.

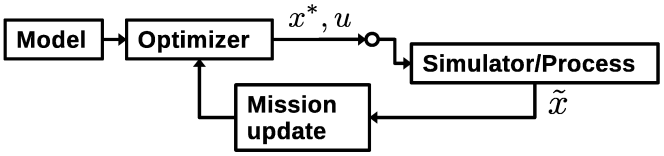


FIGURE 1.1: Basic structure of the Model Predictive Controller feedback loop.

At each time iteration $k = 0, ..., N-1$, an OCP problem of the form (1.2) is solved using the current state $\hat{x}$ as initial state condition. The problem is defined over a fixed prediction horizon T having N time grid points. The solution of the MPC problem is the couple states-controls defined as, $\{x_1^*, \ldots, x_N^*\}, \{u_0^*, u_1^*, \ldots, u_{N-1}^*\}$, with each $x_{i+1}, u_i, i \in [0, N-1]$ being a vector. At the next time step, the horizon shifts forward in time, the new state is measured or estimated, and the optimization is repeated. This procedure is known as the receding horizon or rolling window approach. By continuously updating the control input based on the latest state information, MPC provides closed-loop performance, inherently incorporating feedback.

In Figure 1.2 a MPC-scheme is presented. Its model is not representative of any existing system, but it serves as an example to picture the state-control couple as well as predictions using a rolling fixed horizon. In orange a random control u_k is plotted in the range $[0, 1]$ and in blue the state x_k reacting to u_k with some delays. In red the controller prediction is plotted.

To reduce computational effort and improve convergence speed, MPC solvers often employ warm starting, where the state and controls from the previous time step are used as initializations points for the current step. This exploits the similarity of successive optimization problems, especially when the ODEs are smooth and the controller work at a high-enough frequency. In the next subsection two types of MPC are presented: the linear MPC (LMPC) and nonlinear MPC (NMPC).

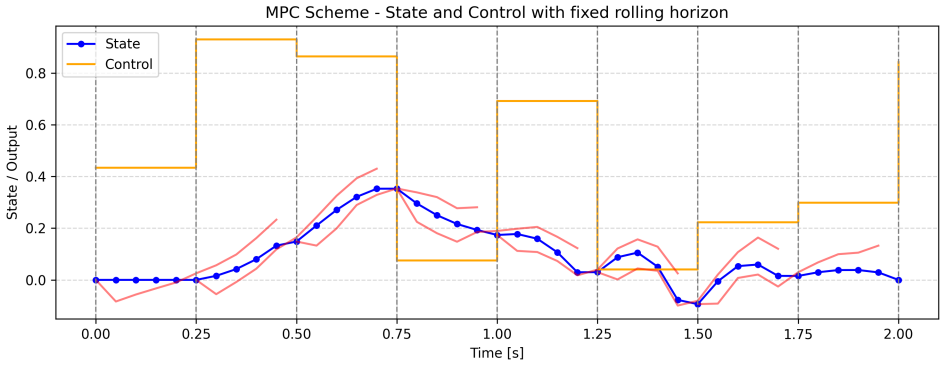


FIGURE 1.2: MPC scheme - State-Control couple with a predictions using a fixed rolling horizon.

The LMPC case: In the special case of linear or linearized dynamics, the optimal control problem reduces to a quadratic program (QP) or a linear program (LP), see sections 1,2. Let us consider the discrete-time linear system,

$$x_{k+1} = Ax_k + Bu_k, \tag{1.7}$$

and the cost function:

$$J = \sum_{k=0}^{N-1} (x_k^\top Q x_k + u_k^\top R u_k) + x_N^\top Q_f x_N, h \tag{1.8}$$

where $Q \succeq 0$, $R \succ 0$, and $Q_f \succeq 0$ are weight matrices.

The corresponding QP can be formulated as,

$$\min_{x_0,\dots,x_N,u_0,\dots,u_{N-1}} \quad \sum_{k=0}^{N-1} (x_k^\top Q x_k + u_k^\top R u_k) + x_N^\top Q_f x_N \tag{1.9}$$

$$\text{subject to} \quad x_{k+1} = Ax_k + Bu_k, \quad k = 0,\dots,N-1, \tag{1.10}$$

$$x_0 = \hat{x}, \tag{1.11}$$

$$x_k \in \mathcal{X}, \quad u_k \in \mathcal{U}, \quad \forall k \in [0,1,...,N]. \tag{1.12}$$

This convex QP can be efficiently solved using interior-point or active-set methods, see section 1, and it forms the basis of linear MPC algorithms used in industrial control.

The NMPC case: When the system dynamics $f_c(x(t),u(t))$ and/or cost function $\ell_c(x(t),u(t))$ are nonlinear, the OCP becomes a nonlinear program (NLP), see section 3. The resulting problem may also be nonconvex. The general NMPC optimal control problem at each MPC iteration is expressed in Bolza form, as given in (1.1). Solving these problems in real-time presents a computational challenge. Despite this, NMPC has become increasingly used due to advances in numerical solvers and hardware. Theoretical foundations and algorithms are thoroughly discussed in [64] and [106].

This thesis focuses on a special class of NMPC problems where a quadratic cost (on the states and controls) is multiplied by a bounded sum of Gaussian distributions. The equality

constraints may be nonlinear depending on the used dynamical system, but the inequality constraints are linear.

$$\begin{aligned} \min_{x,u} \quad & \int_0^T \gamma(x(t)) \cdot \left(\tfrac{1}{2} x(t)^\top Q x(t) + \tfrac{1}{2} u(t)^\top R u(t)\right) dt \\ & \qquad + \gamma(x(T)) \cdot \tfrac{1}{2} x(T)^\top P x(T), \\ \text{subject to} \quad & \dot{x}(t) = f_c(x,u), \quad x(0) = \hat{x}, \\ & x(t) \in \mathcal{X}, \quad u(t) \in \mathcal{U}, \quad \forall t \in [0,T], \end{aligned} \tag{1.13}$$

with the state-dependent weight function $\gamma(x(t))$ defined as,

$$\gamma(x(t)) = 1 + \sum_{i=1}^{M} \alpha_i \exp\left(-\tfrac{1}{2}(x(t) - \mu_i)^\top \Sigma_i^{-1} (x(t) - \mu_i)\right), \tag{1.14}$$

with μ_i fixed $\forall t \in [0,T]$.

According to Formulation (1.1) we denote:

$$\ell_c(x(t), u(t)) = \tfrac{1}{2} x(t)^\top Q x(t) + \tfrac{1}{2} u(t)^\top R u(t), \text{ the stage cost,} \tag{1.15}$$

$$E(x(t), u(t)) = \tfrac{1}{2} x(T)^\top P x(T), \text{ the terminal cost.} \tag{1.16}$$

1.3 Architecture and engineering

The path-planner is implemented as a modular mission manager, with the optimal control solver integrated into a framework that provides interfaces for communication with simulators and forecasting results over a local network using the Robot Operating System middleware (ROS2). As illustrated in Figure 1.3, the modular path-planner first constructs the optimal control problem for each time step, then integrates it in time using the in-house OCPIDDAE tool, see [46], and finally solves it as an NLP problem using an inhouse SQP routine (see §3.11). The solver operates at a user-defined frequency, depending on the system to be controlled and the available computational resources. A parallel thread is employed to forecast simulator data at high frequency across the local network via ROS2. Simulator data is accessed through the MAVLink protocol when using PX4 (Pixhawk autopilot) or via the proprietary Crazyflie protocol for Crazyflie platforms.

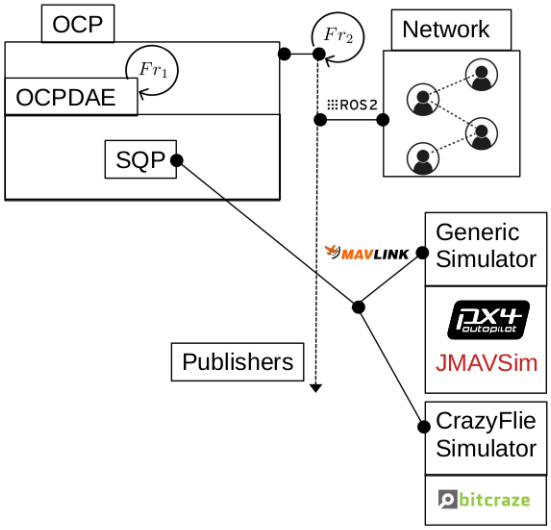


FIGURE 1.3: Modular path-planner framework.

1.4 Chapter conclusion

This chapter provides an overview of optimal control problem (OCP) formulations and a symbolic framework for model predictive control, where both linear (LMPC) and nonlinear (NMPC) variants are introduced. The presented OCP framework lays the foundation for subsequent chapters, particularly Chapters 3 and 4, which focus on real-life scenarios and practical implementations. The chapter introduces architectural and engineering aspects relevant for implementing predictive controllers in practical systems with multiple operators in a network, simulators, and possibly hardware. This design foundation is used for the following results presented in this document.

Chapter 2

Aerial and Ground Vehicle Path-Planning and Path-Tracking

2.1 Introduction

This chapter presents a Nonlinear Model Predictive Control (NMPC) approach for real-time path-planning and collision avoidance for aerial and ground vehicles (UAV,UGV). The method is applicable to quadrocopters, fixed-wing aircraft, and UGVs. At each discrete time iteration, a Nonlinear Programming (NLP) problem is solved using an in-house Sequential Quadratic Programming (SQP) solver for path-planning, while a Quadratic Programming (QP) problem is solved for path-tracking. The proposed NMPC framework enables optimal trajectory computation towards user-defined targets while dynamically avoiding obstacles in the environment. Collision avoidance is implemented using potential field methods, allowing vehicles to maintain safe distances from static circular obstacles.

The mission tasks considered here involve reaching one or multiple designated targets while navigating through a cluttered environment. Users can define or update target positions in real time, and the NMPC planner computes trajectories that respect the vehicle dynamics and collision constraints. For quadrocopters, a reduced-order model capturing translational and rotational dynamics is used to ensure real-time performance, whereas for UGVs, a kinematic vehicle model with nonholonomic constraints is employed for trajectory tracking. A reduced order point mass vehicle is used to model fixed-wings aircraft, permitting real-time path-planning, see [69],[121]. The very well-known Dubin's aircraft kinematic model is used as fixed-wings UAV model.

In a representative scenario, a vehicle is assigned an initial position and multiple user-defined targets. Obstacles in the environment are modeled using circular potential fields. The NMPC planner generates trajectories that guide the vehicle toward the targets while avoiding collisions, dynamically adjusting the path in response to changes in the environment. This approach builds upon existing work in NMPC for collision avoidance [88, 94].

The chapter is organized as follows: first with fixed-wing UAV path-planning, covering the optimal control formulation, cost function, vehicle kinematics, constraints, and evaluation on randomized scenarios. Next, quadrocopter path-planning is presented, including the corresponding optimal control problem, kinematics, constraints, and scenario evaluations. Finally, ground vehicle path-tracking is discussed and introduced for a use case presented in chapter 4.

2.2 Coordinate systems

Coordinate systems are essential for modeling and controlling aerial and ground vehicles. Transformations between different frames are needed in order to construct a modular framework working with different simulators and real-world systems. This section introduces the coordinate systems used in this research.

Body coordinate system: The body coordinate system is attached to the vehicle and moves with it, with its origin located at the center of mass. The x_b axis points forward along the nose of the vehicle, the y_b axis points toward the right wing, and the z_b axis points downward, perpendicular to the wing plane for a fixed-wing aircraft, forming a right-handed coordinate system. This frame is convenient for expressing control inputs such as thrust and angular velocities, as they naturally align with the vehicle's orientation. Measurements from onboard IMUs are expressed here in this frame.

Inertial coordinate system: The inertial coordinate system is a fixed, global reference frame used to describe the absolute position and orientation of a vehicle. Two commonly used local coordinate frames are the ENU (East–North–Up) and NED (North–East–Down) systems, both defined with respect to the Earth's surface. In the ENU convention, the x-axis points East, the y-axis points North, and the z-axis points upward. The ENU frame is standard in many robotics and automotive) applications. The NED convention is traditional in aerospace and aviation, commonly adopted in autopilot systems and flight controllers. Both ENU and NED are right-handed coordinate systems. Transformations between these coordinate frames require rotation matrices that depend on the vehicle's orientation, expressed through Euler and Cardan angles or quaternions in order to avoid singularities in the system.

The combined rotation matrix for a ENU coordinate system is given by

$$R_{\text{ENU}}(\psi, \phi, \gamma) = R_z(\gamma)\, R_x(\phi)\, R_y(\psi), \tag{2.1}$$

with

$$R_{\text{ENU}}(\psi, \phi, \gamma) = \begin{bmatrix} \cos\phi\cos\gamma & \cos\phi\sin\gamma & -\sin\phi \\ \sin\psi\sin\phi\cos\gamma - \cos\psi\sin\gamma & \sin\psi\sin\phi\sin\gamma + \cos\psi\cos\gamma & \sin\psi\cos\phi \\ \cos\psi\sin\phi\cos\gamma + \sin\psi\sin\gamma & \cos\psi\sin\phi\sin\gamma - \sin\psi\cos\gamma & \cos\psi\cos\phi \end{bmatrix}. \tag{2.2}$$

This matrix transforms any vector from the body frame to the inertial frame as

$$\mathbf{v}_{\text{inertial}} = R_{\text{ENU}}(\psi, \phi, \gamma)\, \mathbf{v}_{\text{body}}. \tag{2.3}$$

To transform in the opposite direction (from the inertial frame to the body frame), the transpose of the rotation matrix is used:

$$\mathbf{v}_{\text{body}} = R_{\text{ENU}}^{T}(\psi, \phi, \gamma)\, \mathbf{v}_{\text{inertial}}. \tag{2.4}$$

2.3 Fixed-wing path-planning

Fixed-wing aircraft exhibit dynamic behavior that is inherently stable under steady aerodynamic conditions when sufficient thrust is maintained. In trimmed flight condition, even in the absence of control inputs, the vehicle naturally converges to a flight state at equilibrium, though not necessarily to a zero state, as the equilibrium depends on the prevailing wind conditions and the corresponding velocity vector. This passive stability distinguishes fixed-wing platforms from underactuated ($card(u) < DOF, u = \{\text{controls}\}$) aerial systems such as some multirotors, which are unstable without continuous feedback control. Despite this inherent stability, path-planning and path-tracking for fixed-wing vehicles remain challenging. Aggressive maneuvers require large curvature and rapid deceleration, which may violate the lift–drag equilibrium or exceed the available control bounds. Consequently, optimal control formulations for fixed-wing systems must account for both mechanical feasibility and actuation limits to ensure realizable, stable trajectories.

2.3.1 Optimal control problem definition

The aim, as presented in [48], [38], [64], [106], is to construct a feedback control law $\mu : \mathbb{N} \times \mathbb{X} \to \mathbb{U}$ for the constrained control system in discrete time:

$$\begin{aligned} & x^{i+1} = f(x^i, u^i) \\ & \quad x^i \in \mathbb{X} \subseteq \mathbb{R}^n, \quad i = 0, 1, 2, ..., \\ & \quad u^i \in \mathbb{U} \subseteq \mathbb{R}^m, \quad i = 0, 1, 2, ..., \\ & \quad x^0 = x_0 \\ & \quad x^i \text{ denotes the state vector at discrete time } i \\ & \quad u^i \text{ denotes the control vector at discrete time } i \end{aligned} \quad . \tag{2.5}$$

The path is described in discrete time by the couple states-controls x^i, u^i at time $t_i = ih$ with the sampling time $h > 0$. Finding acceptable paths require to solve an optimal control problem on some time horizon $[k, k + N_h]$ like

Problem 2.3.1 — **MPC path-planning**

$$\min_{x,u} \quad \left(\mathcal{J}_{distance}(x^i(N_h)) + \sum_{k=1}^{N_h} \mathcal{J}_{energy}(x^i(k), u^i(k)) \right) \cdot \tag{2.6a}$$
$$\prod_{l=1}^{N_g} \left(1 + \mathcal{N}_l(\mu_l^i(k),\, \sigma_l^i(k))\right)$$
$$\text{at discrete time } i,\ k = 1, ..., N_h, l = 1, ..., N_g$$

subject to

$$\dot{x}^i(k) = f(x^i(k), u^i(k)), \tag{2.6b}$$
$$x^i(0) = x_0^i (= x^{i-1}(1) \text{or } \hat{x}), \tag{2.6c}$$
$$u_{min} \leq u^i(k) \leq u_{max}, \tag{2.6d}$$
$$x_{min} \leq x^i(k) \leq x_{max}. \tag{2.6e}$$

Herein, the index k is a local time index at step i of the NMPC. Solutions are sets of $N_h + 1$ states $[x^i(k), x^i(k+1), ..., x^i(k+N_h)]$ and controls $[u^i(k), u^i(k+1), ..., u^i(k+N_h)]$ predictions, at discrete time i, with a rolling horizon N_h. The index l iterates the N_g obstacles detected or present in the scenario.

The expression to minimize at discrete time i includes the quadratic functions $\mathcal{J}_{distance}(x^i(N_h))$ and $\mathcal{J}_{energy}(x^i(k), u^i(k))$ multiplied by a product of N_g multi-dimensional Normal distributions $\mathcal{N}_l(\mu_l^i,\, \sigma_l^i)$ (2.6a) of mean μ_l^i and standard variation σ_l^i, plus 1, which models a potential field centered on each detected obstacle gravity center. Note that the Normal potential fields $\mathcal{N}_l(\mu_l^i,\, \sigma_l^i)$ are zero anywhere far away from the obstacles' vicinity. Therefore an additive value of 1 is added to each Normal potential field.

The normal distribution in dimension n associated with obstacle l at the NMPC discrete time step i is defined as

$$\mathcal{N}_l\left(\mu_l^i(k),\, \sigma_l^i(k)\right) = c_G \frac{1}{(2\pi)^{\frac{n}{2}}\,(\sigma_l^i)^n} \exp\left(-\frac{1}{2} \sum_{j=1}^{n} \left(\frac{x_j^i(k) - \mu_{l,j}^i}{\sigma_l^i} \right)^2 \right), \tag{2.7}$$

In this study, we restrict our attention to the cases $n = 2$ and $n = 3$, which are relevant for the considered applications. For $n = 2$, the Gaussian potential defines a cylindrical obstacle on the ground plane, corresponding to an obstacle that is invariant along the vertical direction. For $n = 3$, the formulation yields a spherical obstacle, where the penalty increases as the system state approaches the center of the obstacle, providing a smooth repulsive penalty. The weight c_G defines the amplitude of the penalty and must be tuned depending on the application.

For a two-dimensional Gaussian penalty the normal distribution associated with obstacle l at NMPC time step i is given by

$$\mathcal{N}_l(\mu_l^i(k),\, \sigma_l^i(k)) = c_G \frac{1}{\sigma_l^i \sqrt{2\pi}} \cdot \left(\left(\frac{x^i(k) - \mu_{l,x}^i}{\sigma_l^i} \right)^2 + \left(\frac{y^i(k) - \mu_{l,y}^i}{\sigma_l^i} \right)^2 \right) \quad . \tag{2.8}$$

For a three-dimensional position, the normal distribution associated with obstacle l at NMPC time step i is given by

$$\mathcal{N}_l\big(\mu_l^i(k),\, \sigma_l^i(k)\big) = c_G \frac{1}{(2\pi)^{\frac{3}{2}} (\sigma_l^i)^3} \cdot \left(\left(\frac{x^i(k) - \mu_{l,x}^i}{\sigma_l^i} \right)^2 + \left(\frac{y^i(k) - \mu_{l,y}^i}{\sigma_l^i} \right)^2 + \left(\frac{z^i(k) - \mu_{l,z}^i}{\sigma_l^i} \right)^2 \right) \cdot \tag{2.9}$$

The kinematic model of the aircraft is defined in (2.6b), coupled with the initial condition at step i (2.6c) or states given by a simulator $\hat{x}$. Constraints (2.6e) and (2.6d) bundle the states $x^i(k)$ and controls $u^i(k)$ between constants.

Cost function: The cost function minimized at discrete time step i is given in (2.6a) and is composed of a single distance term, $\mathcal{J}_{\text{distance}}(x^i(N_h))$, which penalizes the deviation between the predicted vehicle position at the final prediction horizon t_{i+N_h} and the desired target position x_{target}^i. It is defined as

$$\mathcal{J}_{\text{distance}}(x^i(N_h)) = \left\| x^i(N_h) - x_{\text{target}}^i \right\|_W^2 , \tag{2.10}$$

where W is a positive definite weighting matrix defining the relative importance of each position component in the optimization.

The second term of the cost function $\mathcal{J}_{energy}$ is an energy term, constructed like:

$$\mathcal{J}_{energy}(u^i(k)) = c_e u^i(k)^T Q u^i(k) , \tag{2.11}$$

with Q a symmetric and positive semi-definite matrix of size $m \times m$.

Fixed-wing UAV kinematics: The optimal control Problem 2.3.1 incorporates the kinematic behavior of the UAV in (2.6b). This study uses a generic Dubin aircraft kinematic model, as described in [102], [14], [3], composed of seven states and three controls:

$$x = [x, y, z, \theta, \gamma, \psi, \phi]^T, u = [a, \omega_\phi, \omega_\gamma]^T \quad . \tag{2.12}$$

The seven states are composed by the UAV's position x, y, z [m], orientation (roll, pitch, yaw angles) θ, γ, ψ [rad] and the rudder angle ϕ. The three controls are composed of the UAV acceleration a, the rudder change rate ω_ϕ and the pitch change rate ω_γ. The rudder angle change rate ω_ϕ controls the UAV's rudder position over time. The pitch rate change ω_γ controls aileron movements over time. The rolling behavior is controlled by the back elevator's movement over time. The kinematic system is defined as,

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{v} \\ \dot{\psi} \\ \dot{\phi} \\ \dot{\gamma} \\ \dot{\omega}_\phi \\ \dot{\omega}_\gamma \end{bmatrix} = \begin{bmatrix} v \cdot \cos(\gamma) \cdot \cos(\psi) \\ v \cdot \cos(\gamma) \cdot \sin(\psi) \\ -v \cdot \sin(\gamma) \\ a \\ v \cdot \tan(\phi)/L \\ \omega_\phi \\ \omega_\gamma \\ u_\phi \\ u_\gamma \end{bmatrix}. \tag{2.13}$$

In a purely kinematic fixed-wing differential system the roll angle can be expressed as linearly dependent to the time derivative of the yaw angle [16] as:

$$\theta = \beta \cdot \dot{\psi}. \tag{2.14}$$

Assumption 2.3.1 — **Roll angle of fixed-wings aircraft**

A fixed-wings aircraft having a straight trajectory has a zero roll angle. However, while doing curves, the roll angle becomes non-null. The roll angle may be modeled as linearly dependent on the heading angle velocity.

Note that the parameter β is deduced depending on the aircraft capacities. The scenarios are considered as a square having sides of less than ten kilometers. Therefore, scenario areas don't consider the curvature of the earth. The *East North Up* (ENU) coordinate system is used.

States and controls box constraints: In order to be physically meaningful, a solution path states-controls needs to respect box constraints. Box constraints are introduced in Problem 2.3.1: (2.6e),(2.6d) and the bounds are directly dependent on the UAV mechanical properties. As we consider a fixed-wing UAV flying at low velocity without strong accelerations, the bounds on $x^i(k)$ and $u^i(k)$ are constants.

2.3.2 Evaluation on randomized scenarios

To evaluate the performance of the obstacle-aware NMPC, we conducted a series of simulations with randomly generated initial conditions and obstacle configurations. Each simulation instance consisted of an initial state x_0 sampled from a predefined distribution over the feasible state space, a fixed goal position x_g representing the target, and a set of obstacles

characterized by their positions (o_x, o_y) and radii, randomly placed around the target position while avoiding infeasible initial states. For each initial state, the NMPC controller was based on Python with *SciPy's* optimization routines. This implementation focuses on validating the problem formulation with a simple, easy-to-implement solver rather than achieving real-time performance. Importantly, it provides a clear indication that this formulation of the problem is solvable, allowing us to confirm feasibility and correctness before investing significant effort into developing a more sophisticated, time-intensive tailored method. The next chapters will present engineering application we achieve real-time performance and efficiency.

Trajectories were labeled as successful if the goal was reached without collisions and failed otherwise. This approach allows for flexible offline analysis, including plotting of trajectories, evaluation of cost function evolution, and assessment of control inputs across a wide variety of scenarios. By iterating over multiple randomized initial states, we are able to statistically quantify the robustness and performance of the proposed NMPC scheme under diverse environment configurations.

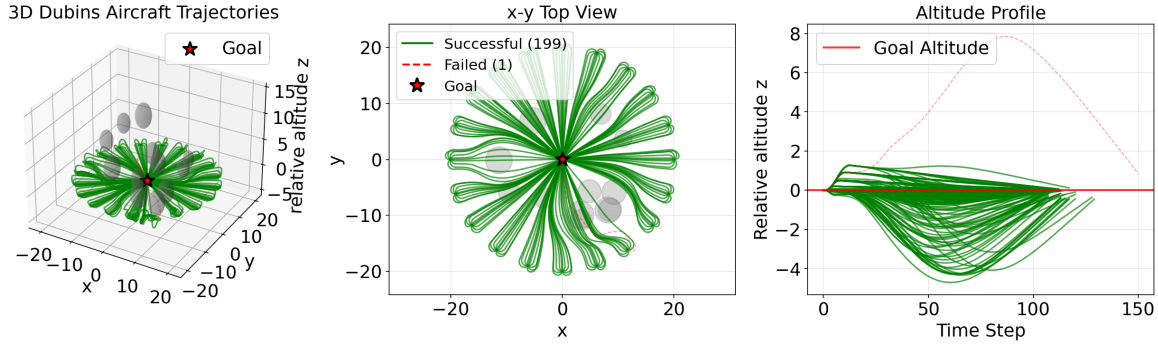


FIGURE 2.1: Evaluation of a fixed-wing path-planning optimal control problem on randomized scenarios.

Figure 2.1 illustrates a set of randomized collision-avoidance scenarios with varying initial positions and yaw angles, while maintaining a constant initial velocity. Out of 200 simulated trajectories, only one resulted in failure, where the UAV was unable to reach its target without collision. As shown in the right panel, the MPC controller increased altitude but failed to recover a feasible trajectory back to the target.

The turning radii of the fixed-wings aircraft are given by

$$R_{\text{yaw}} = \frac{v\cos\phi}{|u_\theta|}, \quad R_{\text{pitch}} = \frac{v}{|u_\phi|}, \tag{2.15}$$

which, for level flight ($\phi = 0$), reduces to $R_{\text{yaw,min}} = v/u_{\gamma,\max}$. Both radii scale linearly with constant velocity and inversely with the angular rate limits. The dynamical model represents a small, agile RC aircraft with low turning radii, using parameters $v = 1$ m/s, $u_{\gamma,\max} = 1.0$ rad/s, and $u_{\phi,\max} = 0.5$ rad/s.

2.4 Quadrocopter path-planning

Quadrocopters are inherently underactuated aerial systems, possessing six degrees of freedom (three translational and three rotational) but only four independent control inputs generated by the propellers. This coupling between attitude and translational motion means that

horizontal movement can only be achieved through tilting, which reorients the total thrust vector.

Unlike fixed-wing aircraft, quadrocopters can hover, take off, and land vertically, making them highly versatile for complex environments. However, their nonlinear and coupled dynamics pose challenges for trajectory optimal control, particularly when aggressive maneuvers or tight obstacle constraints are involved. The optimal control formulation used here ensures dynamically feasible trajectories that respect actuator limits while maintaining stability during flight.

2.4.1 Optimal control problem definition

For path-planning, the problem structure is identical for both quadrocopters and fixed-wing aircraft, as formulated in Problem 2.3.1. The cost function retains its shortest-path formulation, where obstacle avoidance is incorporated through a Gaussian relaxation term.

Quadrocopter UAV kinematics: The quadrocopter kinematic model describes the nonlinear coupling between translational and rotational dynamics of an underactuated aerial vehicle. The state vector is defined in (2.16) and includes the position coordinates x, y, z, linear velocities v_x, v_y, v_z, the total thrust N, the Euler angles ψ, ϕ, γ representing roll, pitch, and yaw, and their corresponding angular velocities $\omega_\psi, \omega_\phi, \omega_\gamma$.

$$\dot{\mathbf{X}} = [x, y, z, v_x, v_y, v_z, N, \psi, \phi, \gamma, \omega_\psi, \omega_\phi, \omega_\gamma] \tag{2.16}$$

The full kinematic system is expressed in (2.17). The translational motion depends on the projection of the total thrust N onto the inertial axes, introducing nonlinearities in the dynamics. The vertical component additionally includes the gravitational term g.

The control inputs u_N, u_ψ, u_ϕ, and u_γ act as generalized accelerations on the thrust and angular rates, respectively. This formulation allows the computation of smooth trajectories by controlling accelerations rather than direct position or velocity commands, ensuring dynamically feasible paths suitable for real-time NMPC implementations.

$$\dot{\mathbf{X}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \\ \dot{N} \\ \dot{\omega}_\psi \\ \dot{\omega}_\phi \\ \dot{\omega}_\gamma \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ \frac{1}{m} N \left(\sin\phi \cos\psi \cos\gamma + \sin\psi \sin\gamma \right) \\ \frac{1}{m} N \left(\sin\phi \cos\psi \sin\gamma - \sin\psi \cos\gamma \right) \\ \frac{1}{m} N \cos\phi \cos\psi - g \\ u_N \\ u_\psi \\ u_\phi \\ u_\gamma \end{bmatrix} \tag{2.17}$$

Remark 2.4.1 Low-level control

When interfaced with a *PX4* simulator (see flowchart 1.3), the MPC provides a velocity vector to the PID-based low-level controller, which tracks it at a high update rate. The MPC itself controls the acceleration level using the nonlinear kinematic model, ensuring a smooth evolution of the velocity vector delivered to the low-level controller.

States and control constraints: As formulated in Problem 2.3.1, bounds are imposed on all states and control inputs, reflecting the physical limitations of the quadrocopter. A detailed analysis of these constraints, including typical numerical values and their influence on trajectory feasibility, is provided in Chapter 4.

2.4.2 Evaluation on randomized scenarios

The performance of the NMPC path-planning controller for the quadrocopter was evaluated across 25 randomized scenarios, as illustrated in Figures 2.2–2.4.

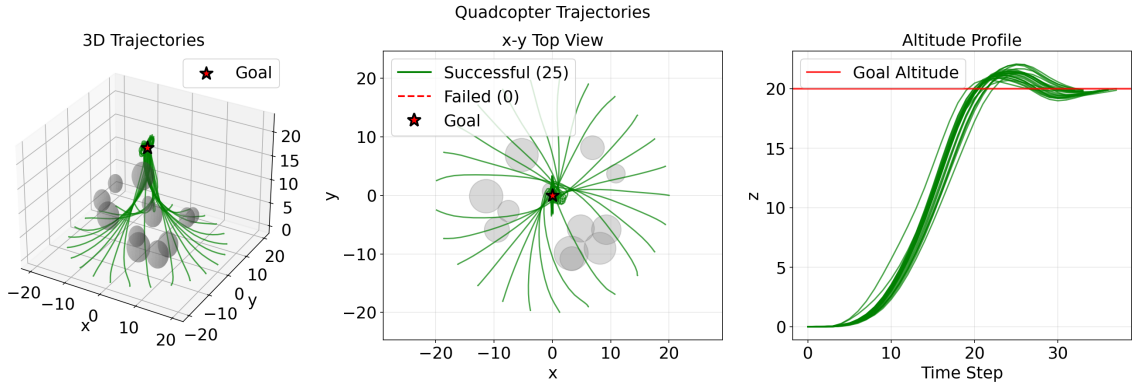


FIGURE 2.2: Evaluation of a quadrocopter path-planning optimal control problem on randomized scenarios.

Figure 2.3 provides an overview of the scenario setup and the success of the controller in reaching the target while avoiding randomly placed spherical obstacles. The resulting trajectories of the quadrocopter are shown in Figure 2.4, demonstrating smooth and dynamically feasible paths from the initial positions to the target. Figure 2.4 depicts the corresponding velocity profiles, confirming that the controller maintains gradual acceleration and deceleration, ensuring a controlled approach and stable hover at the target location. These results highlight the effectiveness of the NMPC scheme in generating collision-free, smooth, and physically realizable trajectories in diverse environments.

Figure 2.5 depicts the corresponding control inputs. The thrust control is generally smooth but tends to exhibit bang-bang behavior near its limits. For the rotational accelerations, rapid changes are common due to the agile dynamics of the quadrocopter, and these variations can be moderated by tuning the corresponding energy terms in the objective function (2.11).

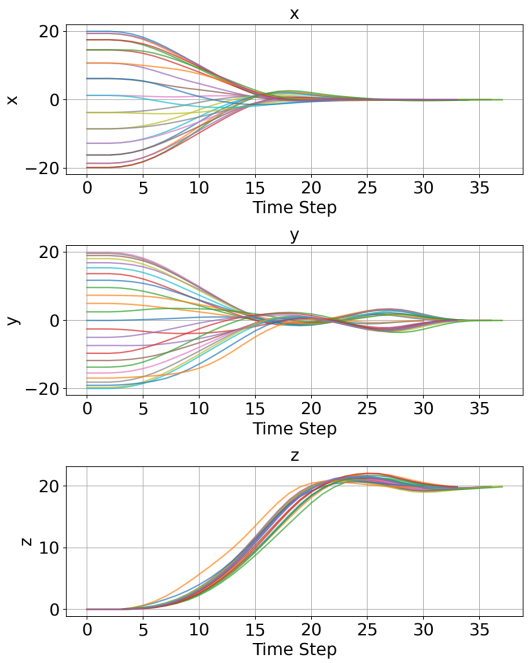


FIGURE 2.3: Trajectories of the quadrocopter in randomized scenarios.

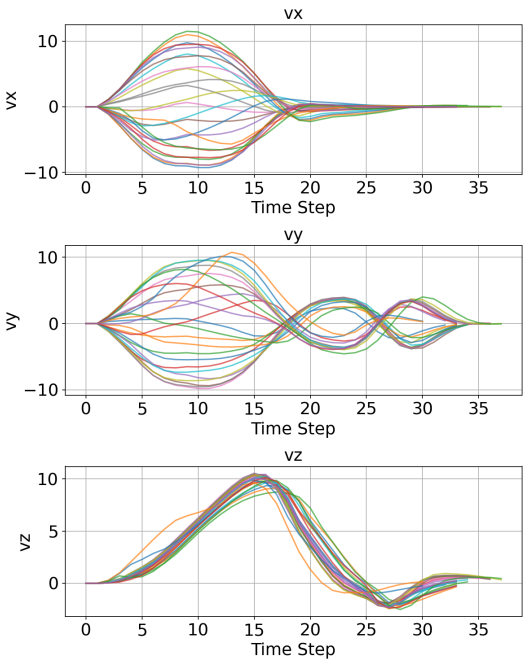


FIGURE 2.4: Velocities of the quadrocopter in randomized scenarios.

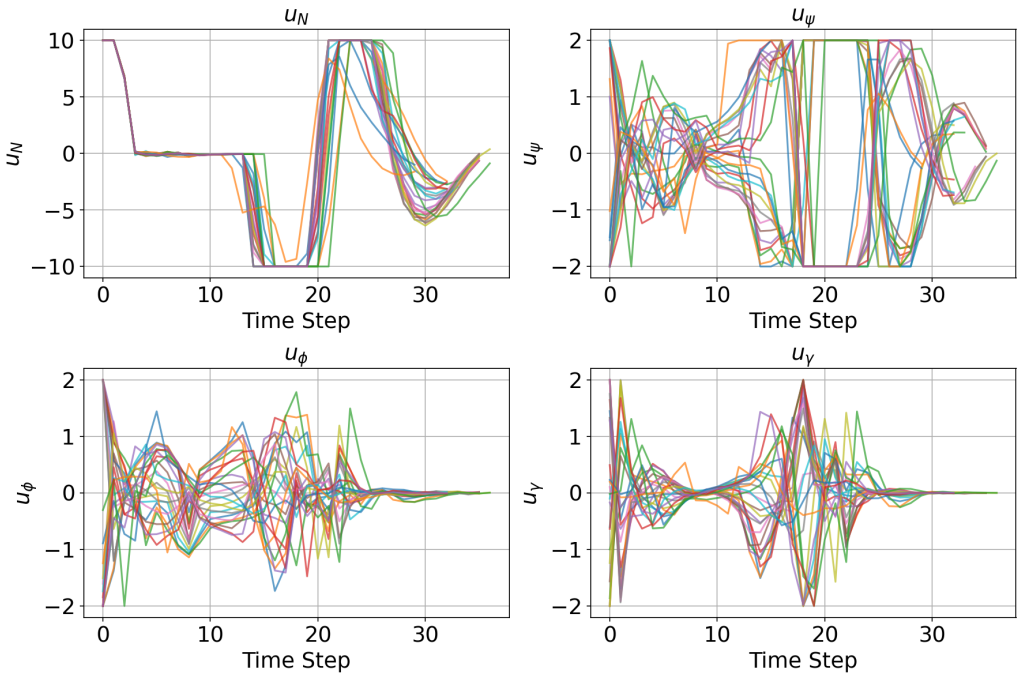


FIGURE 2.5: Control inputs: thrust (u_N), roll rotational acceleration (u_γ), pitch rotational acceleration (u_ϕ) and yaw rotational acceleration (u_ψ).

2.5 Ground vehicles path-tracking

Path-tracking for unmanned ground vehicles (UGVs) focuses on accurately following a predefined reference trajectory rather than generating a new path from scratch. This is particularly relevant in applications where the environment is partially structured, such as roads network and where precise adherence to a trajectory is critical for safety and operational efficiency. Unlike aerial vehicles, which must consider three-dimensional maneuverability and obstacle avoidance, UGVs can operate in constrained, two-dimensional environments where maintaining alignment with the reference path is a challenge, while the vertical dimension primarily affects the vehicle's dynamic stability.

2.5.1 Path-tracking optimal control

Accurate ground-vehicle path-tracking requires modeling the vehicle's nonlinear kinematics. However, for path-tracking tasks, linearizations around the reference trajectory are often possible, allowing the optimal control problem to be formulated using simplified linear models where the OCP can be formulated as,

Problem 2.5.1 — **OCP$(\mathbf{i}, \mathbf{x}_0^{\mathbf{i}})$ - Path-Tracking UGV**

$$\min_{x,u} \quad H = \sum_{k=0}^{N} (x_k^\top Q x_k + u_k^\top R u_k) \tag{2.18a}$$

subject to

$$x^i(k+1) = Ax^i(k) + Bu^i(k), \tag{2.18b}$$
$$x^i(0) = x_0^i, \tag{2.18c}$$
$$u_{min} \leq u^i(k) \leq u_{max}, \tag{2.18d}$$
$$x_{min} \leq x^i(k) \leq x_{max}. \tag{2.18e}$$

Here, the linear dynamic equations describe the vehicle's motion in curvilinear coordinates with respect to the reference path. The resulting linear time-varying (LTV) model can then be discretized, allowing for a fast and reliable numerical solution suitable for real-time implementation. Indeed, the tracking controller operates in a purely planar setting following a reference curve $\gamma_r : [0, L] \to \mathbb{R}^3$ of length $L > 0$. In Frenet coordinates [67], the system is conveniently represented as,

$$\begin{cases} \dot{s}(t) = \frac{v \cdot \cos(\psi(t) - \psi_r(t))}{1 - r(t)\kappa_r(s(t))}, \\ \dot{r}(t) = v \cdot \sin(\psi(t) - \psi_r(t)), \\ \dot{\psi}(t) = v \cdot \kappa(t), \\ \dot{\kappa}(t) = u(t), \\ \dot{\psi}_r(t) = \dot{s}(t) \cdot \kappa_r(s(t)). \end{cases} \tag{2.19}$$

Herein, t denotes the time, and $r(t)$ corresponds to the lateral deviation of the agent

position $(x(t), y(t))^\top$ from the reference curve γ_r. Moreover, $\psi(t)$ and $\kappa(t)$ are the agent's yaw angle and its curvature, respectively. Similarly, $\psi_r(t)$ and $\kappa_r(t)$ denote the reference curve's yaw angle and its curvature, respectively. Note that the curvature of the reference curve is given by $\kappa_r(s(t)) = \dot{x}_r(t) \cdot \ddot{y}_r(t) - \dot{y}_r(t) \cdot \ddot{x}_r(t)$ and its yaw angle by $\psi_r(s(t)) = \arctan\left(\frac{\dot{y}_r(s(t))}{\dot{x}_r(s(t))}\right)$.

The UGV is steered by controlling the time derivative of its curvature, implying $\dot{\kappa}(t) = u(t)$. An observable state vector $y(t) = [r(t), \psi(t) - \psi_r(t)]^\top$ is introduced in order to linearize the equations of motion (2.19). Indeed, for a given constant flight velocity v, the approximations

$$\dot{s} \approx v, \quad \dot{r}(t) \approx v(\psi(t) - \psi_r(t)) \tag{2.20}$$

are acceptable under small deviations, i.e., $y(t) \approx (0,0)$. Finally, we obtain the following linear time-variant (LTV) system:

$$\begin{cases} \dot{X}(t) = A \cdot X(t) + B \cdot u(t) + d(t), \\ y(t) = C \cdot X(t), \end{cases} \tag{2.21}$$

with,

$$\begin{aligned} X(t) &= \begin{bmatrix} r(t) \\ \psi(t) \\ \kappa(t) \\ \psi_r(t) \end{bmatrix}, \\ A &= \begin{bmatrix} 0 & v & 0 & -v \\ 0 & 0 & v & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \\ C &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}, \quad d(t) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ v \cdot \kappa_r(t) \end{bmatrix}. \end{aligned} \tag{2.22}$$

This linear time-varying approximation provides a tractable representation of the nonlinear path-tracking dynamics, enabling an efficient formulation of the UGV path-following problem as a quadratic program suitable for real-time control. This kinematic model was extensively used in [18] for multi-UGV collision avoidance, while maintaining a decoupled linearized kinematic model for each agent.

2.5.2 Inter-agent collision avoidance

When multiple UGVs perform path-tracking in close proximity, the risk of inter-agent collisions necessitates the development of dedicated safe-avoidance strategies. In the considered setting, each agent is assigned an individual reference path to follow. Each curve uses a spline approximation. We denote by $(V)^i_j$ the projection of any vector $\vec{V}$ related to agent i projected on spline j. While working with multiple agents, inter-agent collisions may occur. For each agent, any other one is viewed as a possible obstacle. Obstacles are modeled by

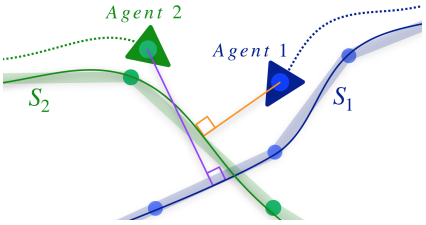


FIGURE 2.6: Two-agent collision avoidance set-up and projections.

imposing linear state constraints on the lateral deviation $r(t_n)$ at a discretized time step t_n such that $\bar{r}(t_n) \geq r(t_n) \geq r(t_n)$, where $\bar{r}(t_n)$ and $r(t_n)$ are the maximal and minimal lateral deviation bounds, respectively. The inter-agent collision avoidance is linear in order to keep the multi-agent problem in the QP form. The bounds of $r(t_n)$ may be deformed using an elastic Gaussian mode as indicated in [25]:

$$r_j^i(t_n) \leq r_j^i(t_n) + \sum_{\forall j \neq i} \delta_j^i - \sum_{\forall j \neq i} \bar{\delta}_j^i \leq \bar{r}_j^i(t_n), \tag{2.23}$$

where δ_j^i and $\bar{\delta}_j^i$ have the form:

$$\delta_j^i(t_n) = 0.9 \begin{cases} R_j^i(t_n) \cdot \exp\left(\frac{-(s_i^i - \epsilon(t_n) - s_j^i)^{c_2}}{c_{11}}\right) & \text{if } s_i^i \leq \epsilon(t_n) - s_j^i, \\ R_j^i(t_n) \cdot \exp\left(\frac{-(s_i^i + \epsilon(t_n) - s_j^i)^{c_2}}{c_{12}}\right) & \text{if } s_i^i \geq s_j^i + \epsilon(t_n), \\ R_j^i(t_n) & \text{else.} \end{cases} \tag{2.24}$$

Herein, s_j^i indicates the arc length of agent i projected on spline j with $1 < i, j < N$. The exponent c_2 is a positive even integer whereas the coefficients c_{11} and c_{12} are some positive constants. The term R_j^i defines a characteristic corridor width along spline i for a possible avoidance of agent j. The term $\epsilon(t_n)$ denotes the length of the avoidance trajectory $\delta_j^i(t_n)$ and is computed via $\epsilon(t_n) = c_\epsilon \cdot \frac{\|v_j^i(t_n)\|_2}{v}$ using the relative flight velocity rate of agents $i, j \in \mathcal{N}$ and a multiplicator c_ϵ. $R_j^i(t_n)$ is a shape function defined as $R_j^i(t_n) = c_\rho \cdot \left(1 + \frac{\psi_{r_j}{}^i}{\psi_{r_i}{}^j}\right)$ with all angles lying between $[0, 2\pi]$. As the kinematic model is equivalent to a point mass model, c_ρ is tuned depending on the characteristic size of the simulated agents.

A *Frenet* projection method and its operator Proj_{S_i} is used to link the position of each agent to all splines. The operator Proj_{S_i} employs a *Newton* method to determine a minimal orthogonal distance (hereinafter referred to as d_j^i). Figure 2.6 visualizes the idea behind this projection method for agent 1 (blue) and agent 2 (green) moving along the respective splines S_1 and S_2.

For $i, j \in \mathcal{N}$, let the set $\mathcal{M}_i = \{(x_i^j, y_i^j), d_j^i \leq r_m\}$ contain all positions of agent j in a neighborhood of radius r_m around (x_i^i, y_i^i). The elastic mode becomes then active when $(x_i^j, y_i^j) \in \mathcal{M}_i$, but $s_i^j \geq s_i^i$.

Remark 2.5.1 — **First comes first served**

In a scenario where agent i would come earlier to contact than agent j, s_j^i would lie behind s_i^i. In this case, a *first-come first-served rule* would apply.

2.5.3 Multi-UGV path-tracking and collision avoidance

We consider a multi-UGV application, following the framework presented in [18], where a dynamic scheduling problem is solved to assign both a starting time and a reference path to each of the N agents. In this approach, M dynamic obstacles are handled during the scheduling preprocessing stage. However, the resulting plan does not explicitly account for inter-agent collision avoidance during execution. In table 2.1, all relevant parameters are given. The first part of the numerical experiments were conducted on an Intel(R) Core i7 computer with a 3.60GHz-CPU and an 16GB-RAM.

The time horizon is $t_f = 5[\text{s}]$ with 100 time points using a uniform grid. The frequency of the LMPC is therefore equal to $20[\text{Hz}]$. The length of all curves γ^i ranges from $290[\text{m}]$ to $390[\text{m}]$. The weight matrices $Q(t) \in \mathbb{R}^{4\times4}$ and $R(t) \in \mathbb{R}$ presented in Problem (2.5.1) are chosen as

$$Q = \begin{bmatrix} h\cdot 10^5 & 0 & 0 & 0 \\ 0 & h\cdot 10^4 & 0 & -h \\ 0 & 0 & 0 & 0 \\ 0 & -h & 0 & h \end{bmatrix}, \; R = \begin{bmatrix} h\cdot 10^4 \end{bmatrix}.$$

The scenario considered by our coupled approach and some results are roughly summarized in Figure 2.7. The plots visualize positions of all eight agents and their collision-free trajectories at a *snapshot* time t with $t = 5, 25, 50, 100[\text{s}]$. The obstacles shown as orange circles move periodically following circular trajectories. As it can be seen, the splines followed by agents 1 and 5 cross. However, they are prevented from colliding with each other by the coordinated LMPC. The collision avoidance effects are clearly shown in Figure 2.8. After the collision avoidance phase, their trajectories converge to the respective reference paths. A full video of the scenario may be found at `https://youtu.be/RNDx6I0ezPs`.

TABLE 2.1: Parameters relevant for the path-tracking.

Parameter	Value	Parameter	Value
N	8	v	$10\left[\frac{\text{m}}{\text{s}}\right]$
M	6	T	2
t_f	$5[\text{s}]$	N_H	100
$h = \frac{t_f}{N_H}$	$0.05[\text{s}]$	$\bar{r}$ / r	$\pm 3[\text{m}]$
r_m	$22.5[\text{m}]$	R_{max} / R_{min}	$\pm 0.25[\text{m}]$
u_{max} / u_{min}	$\pm 0.3\left[\frac{\text{rad}}{\text{s}^2}\right]$	κ_{max} / κ_{min}	$\pm 0.3\left[\frac{\text{rad}}{\text{s}}\right]$
c_{11} / c_{12}	50 / 50	c_2	4
c_ϵ	11	c_ρ	0.045

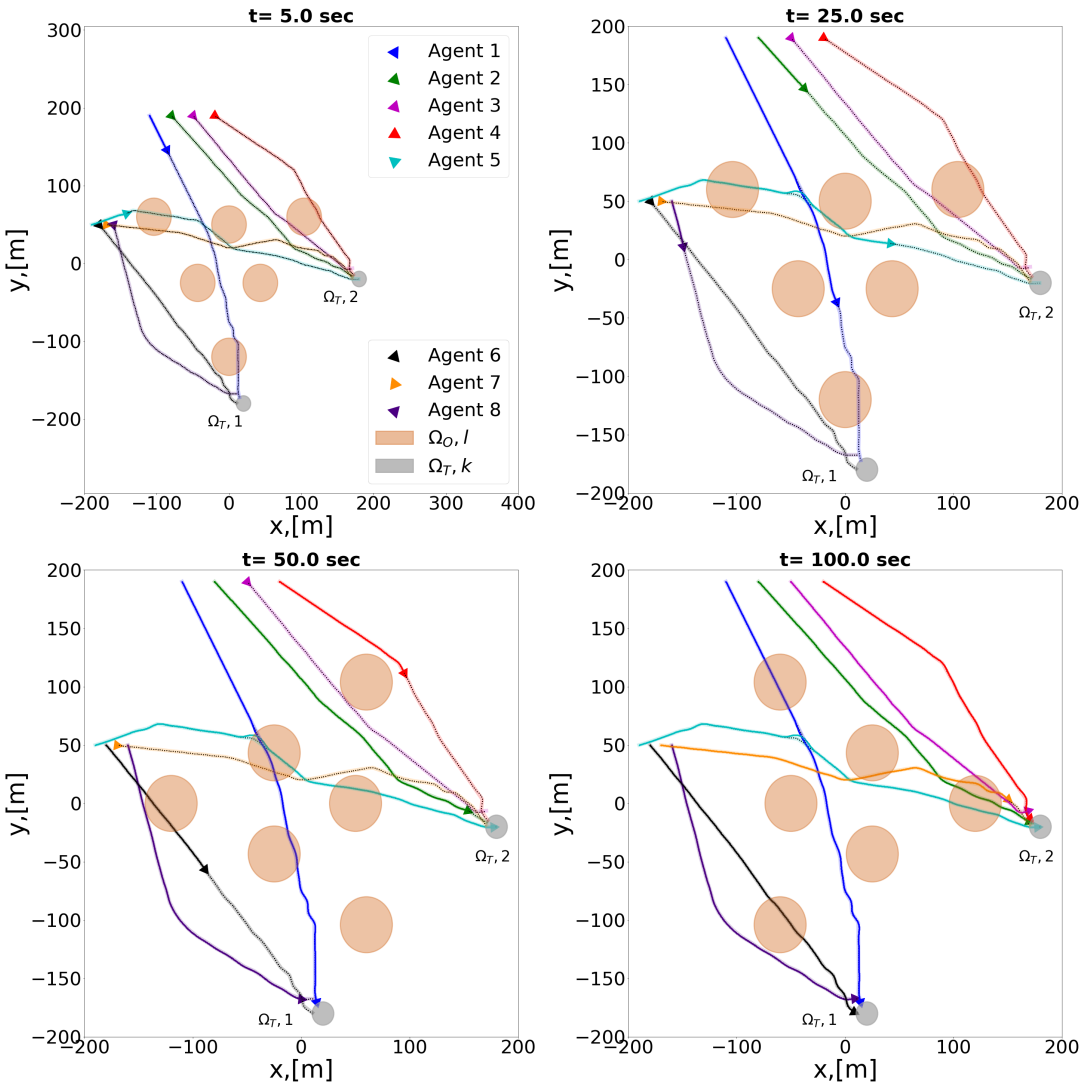


FIGURE 2.7: Trajectories of eight agents at $t = 5.0$ seconds, $t = 25.0$ seconds, $t = 50.0$ seconds and $t = 100.0$ seconds.

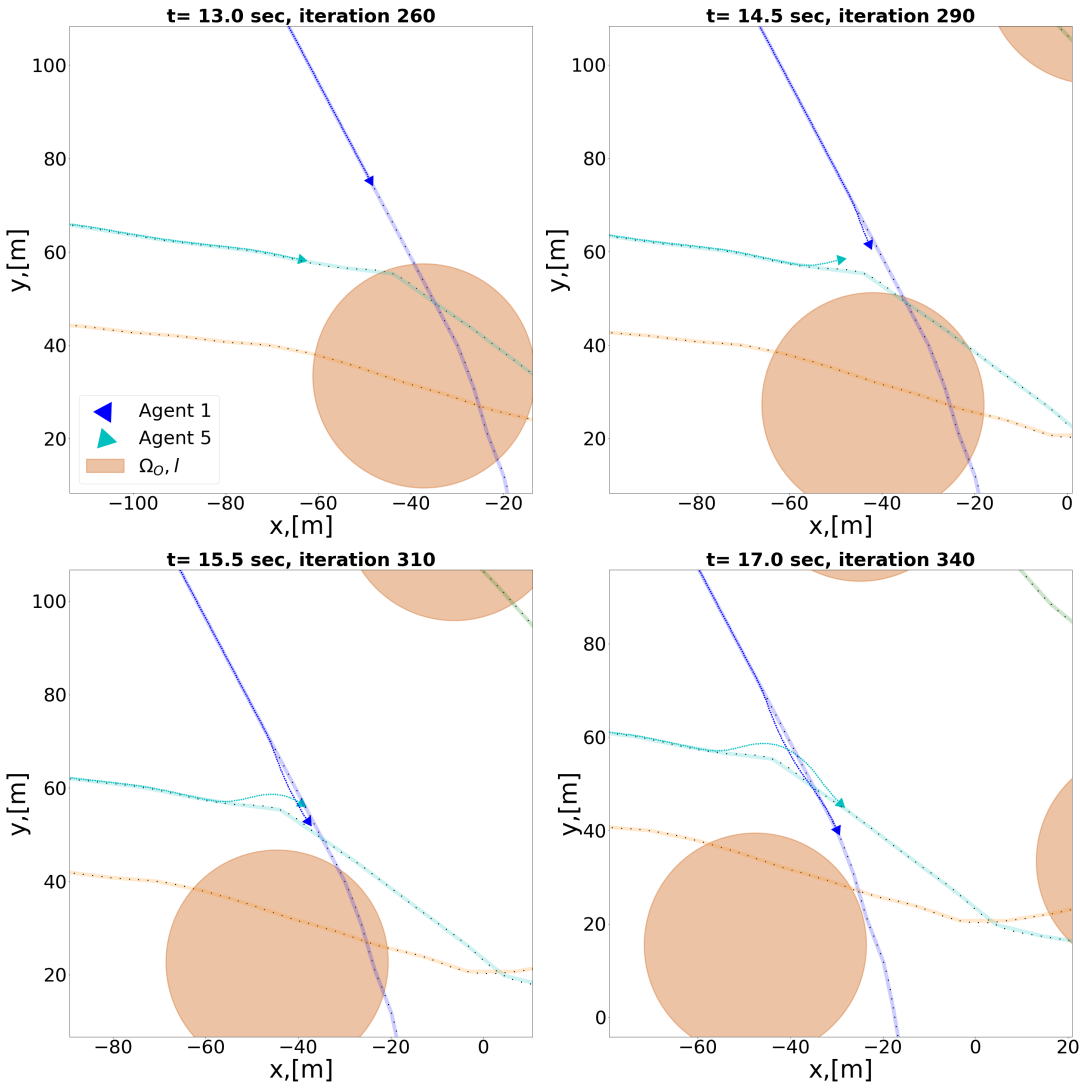


FIGURE 2.8: Collision avoidance of two agents at $t = 13.5$ seconds, $t = 14.5$ seconds, $t = 15.5$ seconds and $t = 17.0$ seconds.

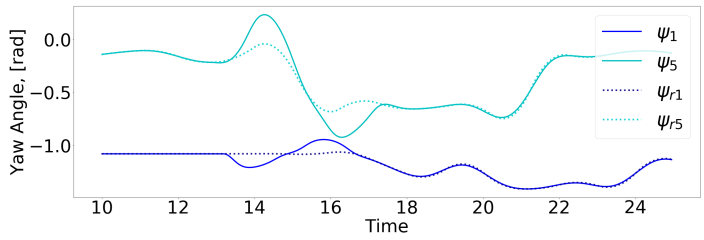


FIGURE 2.9: States ψ and ψ_r of agents 1 and 5 during the collision avoidance phase.

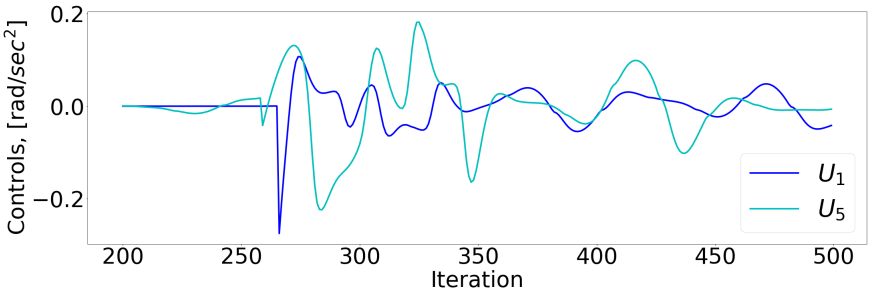


FIGURE 2.10: Controls u of agents one and five during the collision avoidance phase.

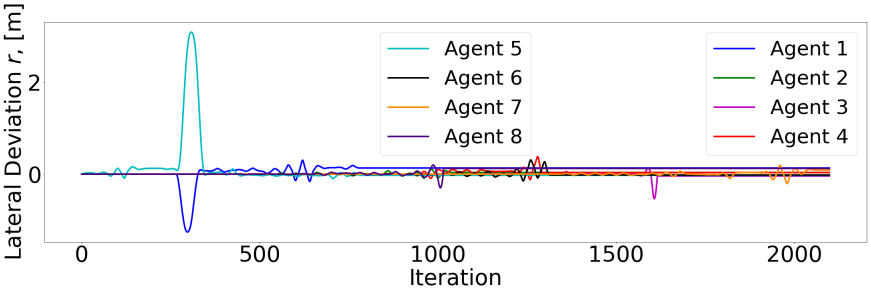


FIGURE 2.11: Lateral deviation per iteration for each agent.

In Figure 2.9, the states $\psi(t)$ and $\psi_r(t)$ are plotted for t ranging from 12.7[s] to 17.7[s]. Yaw angles of both agents 1 and 5 are characterized by a smooth evolution and oscillations around the respective reference yaw angles because of $u_1(t)$ and $u_5(t)$, respectively, see Figure 2.10.

Figure 2.11 shows the lateral deviation $r(t)$. Outside of the collision avoidance neighborhood, the maximum deviation's absolute amplitude is about 0.53[m] and occurs for agent 3. During collision avoidance, the maximum absolute deviation attained is equal to 3.09[m] and 1.25[m] for agent 1 and 5, respectively.

For all time iterations of the discrete LMPC-OCP method, the required computational time per agent was under the LMPC limit of 0.05[s] (i.e. 20[Hz]).

2.6 Chapter conclusion

In this chapter, various path-planning and path-tracking methodologies have been presented and analyzed for different classes of autonomous vehicles, including quadrocopters, fixed-wing aircraft, and ground vehicles. Each system's optimal control problem (OCP) was formulated within a model predictive control (MPC) framework, enabling consistent handling of dynamic constraints, control limits, and trajectory objectives. The approaches discussed here provide a unified foundation for implementing advanced control strategies in practical, real-world scenarios. In the following chapters, these formulations will serve as the basis for multiple real-life applications, demonstrating their flexibility, scalability, and performance in diverse operational environments.

Chapter 3

Man-Unmanned Navigation Assistance System

3.1 Introduction

Background and motivation: In this chapter, we present a pilot navigation method based on nonlinear model predictive control. During real flights, the pilot has access to a digital twin of the aircraft in a virtual environment displayed on a screen in the cockpit. The digital twin relates the actual position, orientation and topography of the environment using a precise GNSS system. The ground station communicates with the aircraft and, using a simple graphical interface, can define a new waypoint destination and zones to avoid (dangerous areas on the ground, closed airspace, etc.). The ground station can add or delete a new waypoint or avoidance zone at any time. The NMPC repeatedly finds a new set of feasible controls and states that form a fixed time horizon trajectory that the pilot can follow. Indeed, the NMPC set of differential equations and constraints are tuned to obtain trajectories depending on the aircraft's speed range and agility using previous flight data. The pilot visualizes the actual states of the aircraft (position, orientation, velocity) in first person projected in a virtual environment. The solution points are represented by a set of doors forming a tunnel-in-the-sky in the virtual environment, see figure 3.2 in first-person view.

In addition, depending on the conditions (wind, disturbances, external causes), the pilot and the ground station can adjust agility coefficients during the flight based on the NMPC box constraints limits, such as pitch and yaw acceleration. The NMPC method works in real-time and is embedded in the aircraft, updating the route to be followed by the pilot, considering new areas to avoid or a new destination. The objective function of the optimal control problem consists of a minimum distance term and a control effort sum of quadratic terms (energy consumption, maximum angular acceleration, etc.). Gaussian potential fields are automatically constructed and relate the zones defined by the ground station to be avoided. The potential field is multiplied by the quadratic objective function and ensures zone avoidance. The ground station also defines an altitude to be maintained for the pilot, and the NMPC automatically adjusts the tunnel-in-the-sky view when the aircraft needs to fly upwards or downwards. The prototyping of both components (NMPC and Human-Pilot-Interface) and their communication was developed and tested within a two-staged testbed environment.

The first step was to develop a simulation tool based on the flight simulator *xplane* and the virtual environment *Carmenta*. Initial simulations were carried out directly with the pilot, adapting the interface to feedback e.g. usability, delays, look-ahead-distances etc. For

testing under demanding real-world flight conditions, an ultra-light aircraft was equipped with two payload containers, see figures 3.1 and 3.3. The sensors, computers, and batteries are carried within these containers, which are suspended under the wings. For inflight and onboard computation of the trajectories, an Intel NUC [2] is connected to the ground station and the pilot interface using the ROS2 middleware and a Silvus Technologies data link. The architecture and communication technologies have already been presented in detail in [100]. The system has been successfully tested in real flights with different avoidance scenarios. Scenarios where areas to avoid appear and change positions frequently, requiring agile, fast and robust path-planning.

Pilot flight assistance is a full research topic, and many different technologies and approaches are developed and published. Tunnel-in-the-sky methods have been developed inside simulators for special maneuvers, see [115], [99], [65], [79] and to decrease the pilot workload while increasing the situational awareness of the navigating crew [123]. According to the authors best knowledge, none of the found methods include zone avoidance and is NMPC based.



FIGURE 3.1: Very light aircraft.

The MPC and optimal control methods build upon the notations, definitions, and equations established in the preceding chapters.

Contributions: The contribution of the article is the engineering and real flight testing of the presented method in order to help pilots navigate in dense, dangerous, and crowded airspace.

Outlines: The chapter first recalls the optimization problem, then presents the system design, highlighting the on-board computation, ground station, and digital twin. Two flight realizations are discussed: one demonstrating zone avoidance and another testing a dense scenario with multiple zones and an emergency avoidance. The chapter concludes with a summary of the results.



FIGURE 3.2: Simulation Setup, Left: *xplane* simulator and pilot, Right: First Person View of the Human-Machine Interface.



FIGURE 3.3: Sensor-pod under the right wing.

3.2 Optimization problem statement

3.2.1 Optimal control problem definition

The optimal control problem is a zone-avoidance path-planning problem, as introduced previously with Problem 2.3.1. The trajectory is described in discrete time by the pair of states and controls (X^i, u^i) at time $t_i = ih$ with sampling time $h > 0$ and the dynamic of the aircraft is modeled using an aircraft Dubin model as in (2.13). To find trajectories that satisfy the constraints defined by the aircraft and are able to adapt to the air environment, an optimal control problem can be solved on a given time horizon $[k, k+N_h]$.

We recall Problem 2.3.1, which formulates the path-planning task as a nonlinear program (NLP) minimizing the trajectory length while penalizing control effort and angular rate changes. Quadratic terms are applied on the states and controls at the end of the horizon and along the trajectory, with obstacle avoidance implemented via Gaussian potential fields centered on zones-to-avoid, which decay to zero far from the obstacles.

As a reminder, the multi-dimensional normal distribution of a zone l at an NMPC's discrete time step i is defined like:

$$
\begin{aligned}
&\mathcal{N}_l(\mu_l^i,\, \sigma_l^i) = c_G \frac{1}{\sigma_l^i \sqrt{2\pi}} \cdot \\
&\left(\left(\frac{x^i(k) - \mu_{l,x}^i}{\sigma_l^i} \right)^2 + \left(\frac{y^i(k) - \mu_{l,y}^i}{\sigma_l^i} \right)^2 \right), \\
&\text{with: } x\text{: east [m], } y\text{: north [m] see (2.13)}, \\
&\mu_l^i = [\mu_{l,x}^i, \mu_{l,y}^i].
\end{aligned}
\tag{3.1}
$$

The coefficient c_G is a scaling parameter, defining maximal value of the normal distributions. By modulating its value, the user can prioritize some zones over other ones. The more c_G increases, the more important becomes the avoidance in the overall objective function and the more complicated the problem becomes.

The path-planning task is formulated as a shortest-path problem with a relaxed collision avoidance term. The distance component of the objective function, $\mathcal{J}_{distance}$, penalizes the deviation between the aircraft's predicted position at the end of the horizon, t_{i+N_h}, and the user-defined target waypoint P_{target}^i,

$$
\begin{aligned}
\mathcal{J}_{distance}(X^i(N_h)) = \|P^i(N_h) - P_{target}^i\|_2^2, \\
P \in \mathbb{P} \subseteq \mathbb{X} \subseteq \mathbb{R}^n,
\end{aligned}
\tag{3.2}
$$

where P denotes the position of the aircraft (east, north, up [m]), which is part of the state vector X.

The energy term of the cost function $\mathcal{J}_{energy}$ is an energy term applied to all discrete time between i and $i + N_h$ such that,

$$
\begin{aligned}
&\mathcal{J}_{energy}(X^i(k), u^i(k)) = c_q u^i(k)^T Q u^i(k) \\
&+ c_w X^i(k)^T W X^i(k) + c_y X^i(k)^T Y X^i(k)
\end{aligned}
\quad . \tag{3.3}
$$

The matrices Q, W and Y are symmetric and positive semi-definite of size $m \times m$, $n \times n$ and $n \times n$. The matrices Q, W and Y will be presented later in §3.4.1.

Kinematic model: The optimal control in Problem 2.3.1 enforces the aircraft kinematics through the equality constraints (2.6b), integrating the discrete dynamics over the horizon $[i, i + N_h]$. A generic Dubins aircraft model with nine states and three controls is used, as introduced in (2.13). The states include the aircraft position (x, y, z), orientation angles (θ, γ, ψ), rudder angle ϕ, and their respective rates $(\omega_\phi, \omega_\gamma)$. The controls correspond to linear acceleration a, rudder angular acceleration u_ϕ, and pitch angular acceleration u_γ. The system is defined in the East-North-Up (ENU) frame.

Box constraints : In order to be physically meaningful, a solution path states-controls needs to respect box constraints. Box constraints are introduced in Problem 2.3.1: (2.6d), (2.6e) and the bounds are directly dependent on the aircraft mechanical properties. As we

consider a fixed-wing aircraft flying at low velocity without strong accelerations, the bounds on $X^i(k)$ and $u^i(k)$ are constants.

Zone avoidance: As formulated in Problem 2.3.1, the cost functions are multiplied by a one plus a product of normal distribution being null everywhere outside the vicinty of some circular zones at discrete time i, defined by the centers $[xc_l^i, yc_l^i] \approx [\mu_{l,x}^i, \mu_{l,y}^i]$. A scalar one is added to the multiplied normal distribution in order to retain only the quadratic terms of the cost functions when far enough away from zones to avoid. The zones to avoid are user-defined by the ground station and are scaled according to context intelligence and pilot flight information. In figure 3.4 the zone to avoid can be seen as the red cylinder and the target as a thin blue one.

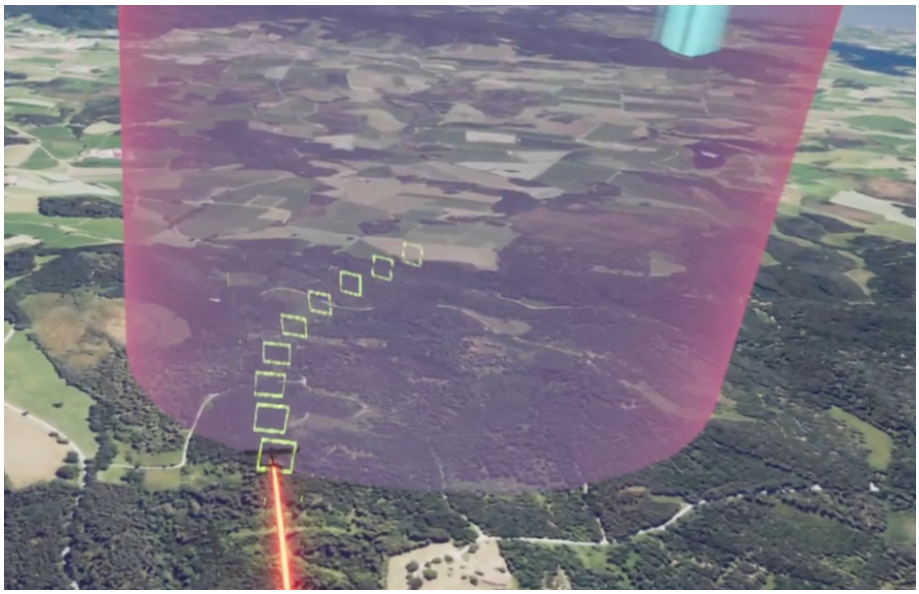


FIGURE 3.4: Avoidance trajectory.

3.3 System design

Figure 3.5 shows two blocks, the inboard and the ground station computing and communication devices. Inboard, embedded in the payload pods (see figure 3.3) the NMPC computer is connected to the First Person View (FPV) tunnel-in-the-sky display, the Inertial Navigation System (INS), a solid-state drive (SSD), sensors for other experiments realized in parallel, and a data-link. The inboard architecture was extensively presented in [100]. The ground station consists of a command center directly connected to the inboard computational framework. From the ground station, the mission can be managed by defining avoidance zones and target zones and tuning the controller according to the pilots' requirements.

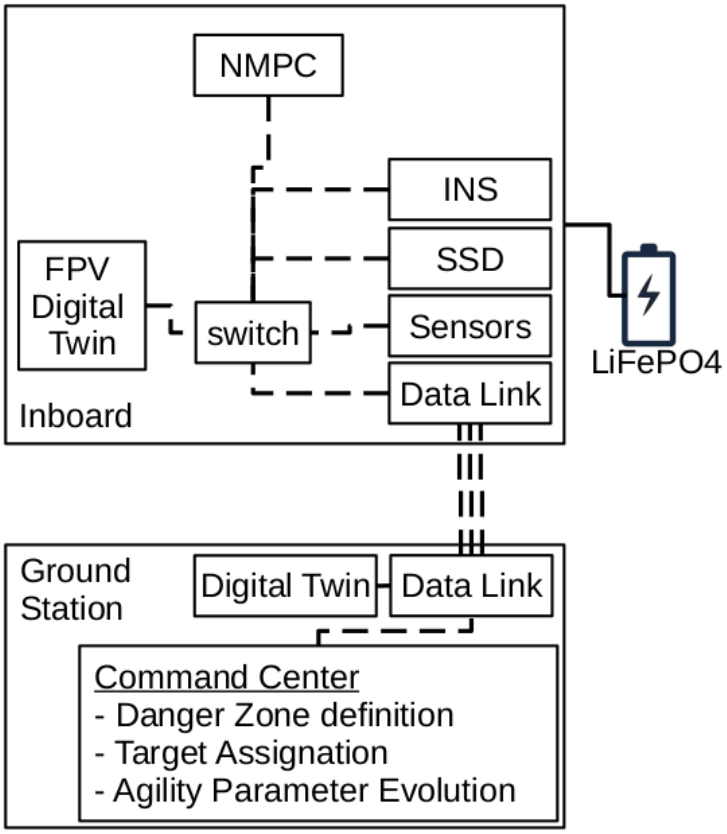


FIGURE 3.5: Inboard and ground station hardware setup.

3.3.1 On-board INS

The pilot can follow the computed trajectories thanks to a realistic First Person View (FPV) digital twin installed in the cockpit, see figure 3.2 and 3.6. The position and orientation of the cockpit in the digital twin is derived from a high-precision (around the centimeter) and high-frequency (100 Hz) Inertial Navigation System (INS). The INS is installed in the sensor pod and is interfaced via the Robot Operation System 2 (ROS2) middleware. In figure 3.5 the inboard architecture and the ground station is presented. The INS is directly linked to the NMPC giving a position and orientation feedback at each i time iteration. The smooth rendering and high-frequency update of the digital twin are needed for the pilot to follow the computed trajectories.

3.3.2 Ground station

The ground station can follow in real-time the trajectory calculated by the NMPC and followed by the pilot. Using a user-friendly GUI (graphical user interface: here an interactive map), it is possible to define targets for the pilot to reach and circular zones to avoid (see figure 3.6). The ground station can add as many avoidance zones of different radii as required. By aggregating different circular zones, complex scenarios can be modeled using a simple but robust GUI. Moreover, using only circular zones, but with the possibility of using many of them, inherently makes the problem easier to solve than with equivalent complex unique zones to avoid.

As shown in subsection 3.2.1, the dynamical system used is simplified for the sake of computational speed and does not take into account aerodynamic effects such as deflection angles and lift losses. However, the cost function presented in subsection 3.2.1 does include energy terms such as the agility of the aircraft. With feedback from the pilot and depending on the environmental conditions (wind, temperature, altitude, etc.), the ground station can tune the aggressiveness of the trajectory in real-time by changing the coefficients of the matrices Y and W.

Depending on the electromagnetic activity and obstacles in the vicinity of the data link antenna, the connection may be lost. From the ground station, the NMPC receives just a new target and possibly new zones-to-avoid. In the event of a loss of data link, the calculation would continue unaffected and the new mission information would have to be retransmitted from the ground station in order to avoid old messages perturbing the experiment in the future.

3.3.3 Inflight mini-computer

The NMPC structure is carried by an Intel NUC Pro Mini of 12^{th} generation [2] having a i5-1240P processor with 16GB of RAM and 12 cores.

3.4 Flight experiments

In this section two real flight experiments are presented. The first realization has a size of six kilometers by six kilometers and has five zones to avoid. The second realization presents a limit case where the zones to avoid are closer to each other and where a zone appears in front of the aircraft with very little time for the pilot to react.

3.4.1 Scenario parameters

The table 4.2 contains the values of the aircraft model and the NMPC parameters. The NMPC time horizon is fixed and set to 5 seconds. For the pilot, for example, at a speed of 50 [m.s^{-1}] (180 [km.h^{-1}]), the projected future trajectory will be 250 [m] long. On the 5 seconds trajectory, 10 time grid points are computed, see target windows for the pilot in figures 3.4 and 3.6. The update frequency of the NMPC solution is 2 Hz and 0.5 seconds is the time to fly from one window to the next at a constant speed.

Regarding the objective function, $\mathcal{J}_{distance}$ is minimized during the flight while moving toward the target waypoint. The energy part of the objective function is scaled according to its value, having a weight c_e equal to 20 % of the distance part of the objective, thus about 16 % of the total cost function at the beginning of the optimization. After optimizing the trajectory, these percentages can evolve. The coefficients $[c_q, c_w, c_y]$ represent the weight given to the energy associated with the terms $[a, u_\phi, u_\gamma]$, $[\phi, \gamma]$, and $[\omega_\phi, \omega_\gamma]$, respectively. The matrices Q, W and Y are all diagonal and select the states and controls to weight as shown in the equation (3.3). For the sake of simplicity, the size of the block matrices is given

TABLE 3.1: Computational and system parameters values.

Parameter	**Value**	**Parameter**	**Value**
Horizon	5 [s]	c_e	$0.2 \times \mathcal{J}_{distance}$
N_h	10	$[c_q, c_w, c_y]$	$[0.5, 0.1, 0.5]$
Q	$\begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	β	8 $[\mathrm{s}^{-1}]$
W	$\begin{bmatrix} 0_{4\times4} & 0 & 0 & 0 & 0 \\ 0 & \xi_{11} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \xi_{22} & 0 \\ 0 & 0 & 0 & 0 & 0_{2\times2} \end{bmatrix}$	Y	$\begin{bmatrix} 0_{7\times7} & 0 \\ 0 & \epsilon_{jj} Id_{2\times2} \end{bmatrix}$
R	500 [m]	μ_l^i	xc_l^i, yc_l^i, zc_l^i [m]
L	4.88 [m]	${\sigma_l}^i$	R_l [m]

for the diagonal terms. All other terms are written as 0 but are null submatrices, forming the square matrices W and Y.

Matrix Q penalize the linear acceleration to 0.01 and weight the rudder and pitch angular acceleration to 1.

As explained before, it is possible to tune the NMPC online during the flight, depending on the conditions and the pilot sensation. The weights ξ_{pp} and ϵ_{jj} with $p = 1, 2$ and $j = 1, 2$ can be changed depending on the desired trajectory, with $\xi_{pp}, \epsilon_{jj} \in]0, 1]$. The coefficient ξ_{pp} is associated with the rudder angle and pitch angle values and the coefficient ϵ_{jj} is associated with the pitch angle and rudder angle change rates. The more ξ_{pp} is high, the more the turn and the avoidance will be expensive in terms of the objective function, and the more ϵ_{jj} is high, the more the curves will be smooth, bringing less flexibility in dense areas. Finally, the circular zones to avoid are set to $R = 500$ [m] and the aircraft length is set to 4.88 [m].

Table 4.3 presents the states and controls constraints. For the states, the velocity v ranges between 40 and 60 $[\mathrm{m\,s}^{-1}]$ and the coefficient β is equal to 8 $[\mathrm{s}^{-1}]$. The coefficient β must be adjusted according to the flight capacities of the selected aircraft, and previous flight statistics have been used.

The rudder and pitch angles rate of change ϕ, γ are bounded between -0.9 [rad] and 0.9 [rad] multiplied by the agility factors ξ_{11}, ξ_{22} for the angles of the rudder and the pitch, respectively. The roll angle is computed using relation (2.14) and therefore depends on ϵ_{11}. The rudder and pitch angular velocities are bounded between -0.1 $[\mathrm{rad\,s}^{-1}]$ and 0.1 $[\mathrm{rad\,s}^{-1}]$ multiplied by the agility factors ϵ_{11}, ϵ_{11}. In the flight experiments presented in the next section $\xi_{11} = \xi_{22}$ and $\epsilon_{11} = \epsilon_{22}$. Their values may be changed online during the flight.

TABLE 3.2: States and controls box constraints' values.

Lower limit	Variable	Upper limit
40 $[\mathrm{m\,s^{-1}}]$	v	60 $[\mathrm{m\,s^{-1}}]$
-5 $[\mathrm{m\,s^{-2}}]$	a	10 $[\mathrm{m\,s^{-2}}]$
$-0.9 \cdot \xi_{11}$ [rad]	ϕ	$0.9 \cdot \xi_{11}$ [rad]
$-0.9 \cdot \xi_{22}$ [rad]	γ	$0.9 \cdot \xi_{22}$ [rad]
$-0.1 \cdot \epsilon_{11}$ $[\mathrm{rad\,s^{-1}}]$	ω_ϕ	$0.1 \cdot \epsilon_{11}$ $[\mathrm{rad\,s^{-1}}]$
$-0.1 \cdot \epsilon_{22}$ $[\mathrm{rad\,s^{-1}}]$	ω_γ	$0.1 \cdot \epsilon_{22}$ $[\mathrm{rad\,s^{-1}}]$
8 $[\mathrm{s^{-1}}]$	β	8 $[\mathrm{s^{-1}}]$

3.4.2 Realization 1: multiple zone avoidance

The flight experiment lasts 162 seconds and covers a distance of approximately six kilometers. Five zones-to-avoid are present at a fixed position during all the experiments, named $Z_1, ..., Z_5$, as shown in figure 3.8. The plane trajectory is plotted in blue, and the predicted states are plotted in orange. The target is plotted as a green disk. The blue-to-red isolines plots the used potential field, see (3.1). The potential field has values between one (no isolines) and 3.3 (red isolines). In the case of zones-to-avoid far from each others, the peaks of the potential field are defined in the center of the circular zone to avoid. When two zones-to-avoid are close (see Z_4, Z_5) the product of $\mathcal{N}_4(\mu_4^i, \sigma_4^i)$ and $\mathcal{N}_5(\mu_5^i, \sigma_5^i)$ creates a noncircular Gaussian distribution. It ensures the impossibility of flying through too narrow corridors. The format of this plot is used throughout the article.



FIGURE 3.6: Left: Digital twin of the very light aircraft, Right: Cockpit view inflight.

It can be noted that in both flight experiments the optimization solver solved Problem 2.3.1 faster than the defined NMPC frequency (2 Hz according to table 4.2) for each time

iteration despite the nonconvexity and nonlinearity of the constrained problem.

For the sake of clarity, all results are given in meters. The NMPC is also based on a local East, North, Up (ENU) system but works with the INS and global data using the WGS84 model [1]. The agility parameters are set to $\xi_{pp}, \epsilon_{jj} = 0.5 \; \forall p, j$. The trajectory is smooth, and the pilot manages to follow the planned trajectory exactly. In figure 3.7 the altitude of the aircraft is plotted in blue above sea level (ASL). Each thinner line represents the altitude evolution of the predicted points (green windows in figure 3.6) during the flight, light green for the first, orange for the second, and so on. The total loss of altitude during the flight is only 20 meters, while a constant altitude of 1260 meters was targeted. During avoidance maneuvers, the aircraft has a loss of lift and loses altitude. This effect is minimized by the pilot accelerating during turns.

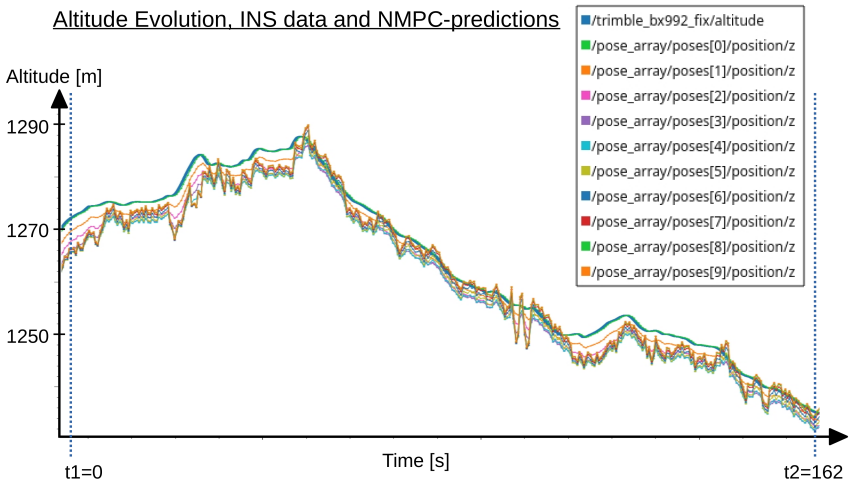


FIGURE 3.7: Altitude evolution, aircraft and predicted points.

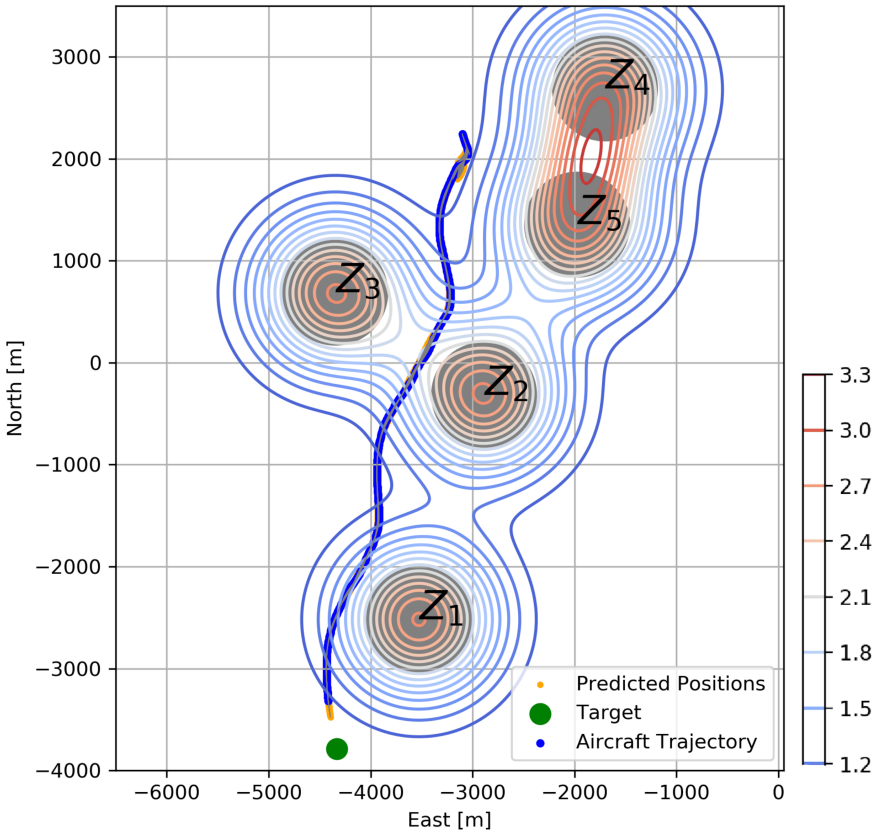


FIGURE 3.8: Multiple zone avoidance, trajectory and predicted positions.

3.4.3 Realization 2: dense emergency maneuver

The second flight experiment presents a denser scenario of four kilometers by four kilometers, with three zones to avoid initially and one added later in the scenario directly in front of the aircraft. In figure 3.9 the trajectory is plotted at two different time stamps. On the top subplot, the aircraft is between two avoidance zones Z_1 and Z_2, flying towards the target waypoint. On the lower subplot a new avoidance zone Z_4 appears and the pilot has to change the trajectory before reaching the target waypoint. The agility parameters are set to $\xi_{pp}, \epsilon_{jj} = 1$ $\forall p, j$.

In figure 3.10 the avoidance zone Z_4 is highlighted. The aircraft position is plotted in blue and the prediction point in orange. To emphasize the quality of the pilot following the predicted trajectory, a gray line connects the orange prediction points. The avoidance is smooth, even with strong turns. The pilot was able to follow the trajectory even with an agility parameter scaled to the maximum. In fact, the blue curve and the orange trajectories overlap for most of the experiment. At the very end of the experiment, the orange predicted trajectories show some oscillations as a new waypoint was set further away. When new waypoints are set, a jump in the objective function is to be expected, as the term $\mathcal{J}_{distance}$ suddenly increases strongly. These artifacts are also visible on the figures 3.7 and 3.9 at the beginning of the scenario.

A video displaying the successful flight-tests and the tunnel-in-the-sky method is available under the link:

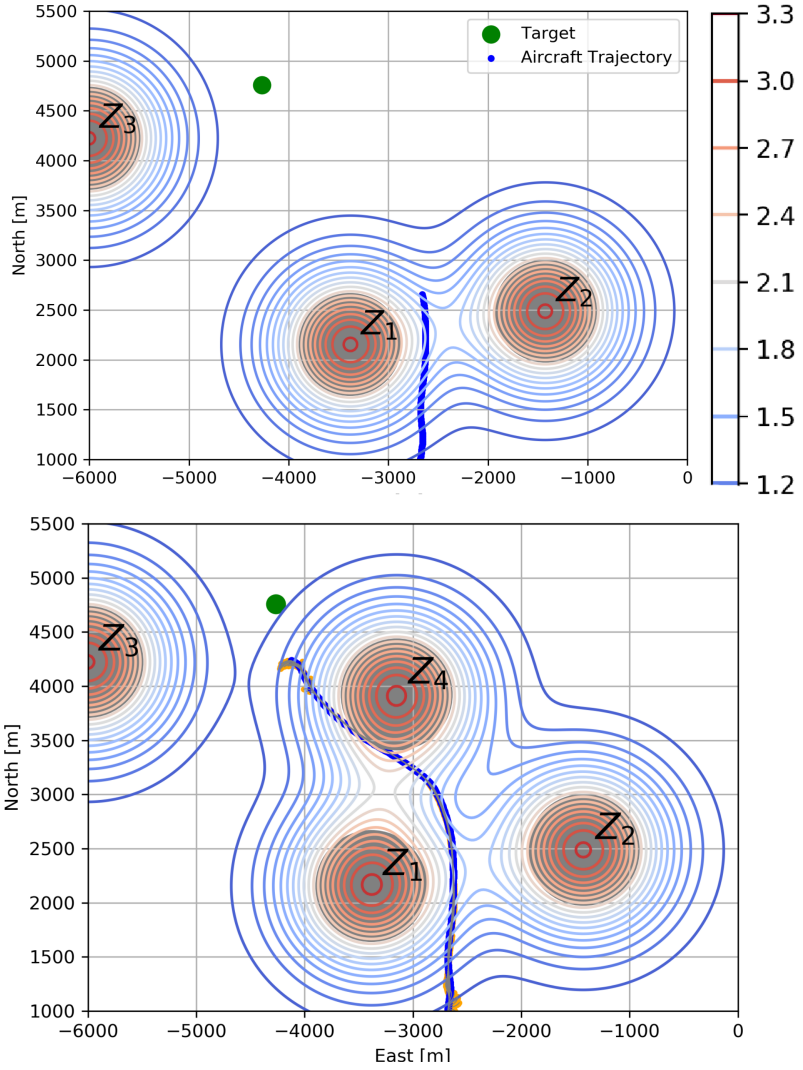


FIGURE 3.9: Multiple zone avoidance in dense area, trajectory and predicted points.

https://youtu.be/XQsf2f5p4ZY .

The video format emphasize the dynamic of the trajectories and the ability for pilots to follow real-time computed paths, especially when new zones-to-avoid appear.

3.5 Chapter conclusion

In this chapter, we proposed a human-machine interface to help pilots navigate into dangerous or congested areas by indicating trajectories to follow. Using the described NMPC system architecture, the predicted routes were followed by the pilot with very little delay. Two flight experiments were proposed and we demonstrated the robustness and adaptability of the presented NMPC path-planner and digital twin immersive display. Thanks to an online fine-tuning of the aircraft agility, the pilot was able to avoid several multiple zones even in a small flight area.

The presented method with a pilot can be extended to the case of multi-agent systems

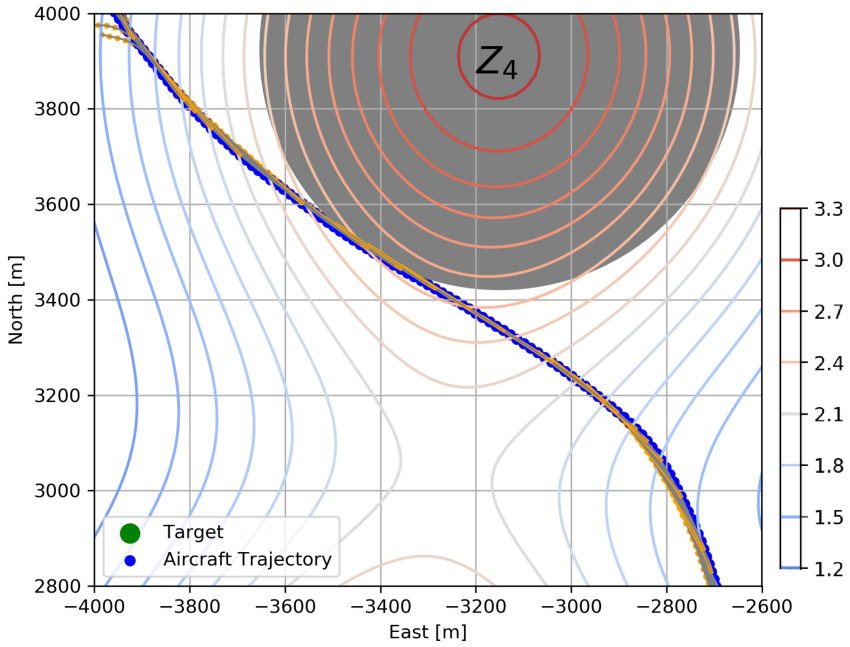


FIGURE 3.10: Multiple zone avoidance in dense area, zoom on zone Z_4.

and manned-unmanned teaming or even unmanned-unmanned teaming, ensuring successful missions and air traffic management, as presented in the next chapter. Finally, since the aircraft is already equipped with sensors, detect and avoid experiences would be beneficial towards more real applications.

Chapter 4

UAV-UGV Interactions: The Case of Quadrocopter-UGV Dynamical Docking

4.1 Introduction

Background and motivation: The aim of this research is Cooperative Rendezvous of an Unmanned Aerial Vehicle (UAV) and an automated Unmanned Ground Vehicle (UGV) equipped with a rooftop platform. The growing use of quadrocopters has changed the way surveillance, rescue and military operations are conducted. Their inherent agility and multi-engine design have opened up new flight paths and feasible missions. However, there are several prerequisites for their efficient use, such as a trained pilot and to taking off in the vicinity of the theater of operations due to its lack of endurance. The objective is to enable high operational utility by deploying a quadrocopter from an automated UGV equipped with a rooftop platform. Moreover, it is intended to largely avoid sensitive and expensive additional structures such as robotic gripper arms. As an aid for landing, only a landing surface equipped with electromagnets on the roof of the UGV is provided.

Using MPC, on a fixed time horizon and at a fixed frequency, a solution trajectory and its associated controls are recomputed and updated. The computed solution is a set of states and controls defined on the chosen time horizon. The controls and states are related by a set of differential equations that relate the dynamics of the agents. Constraints are applied to the controls and states in order to obtain robust and reliable trajectories. The aim of our work is to achieve moving docking maneuvers while decoupling the dynamics of both agents. Each one shares its real-time data with the other one without knowing each other's dynamics. This has the advantage of being platform flexible and allows an operator to drive the UGV while the quadrocopter acts in an automated way.



FIGURE 4.1: Trajectories UAV-UGV - Unreal engine.

Contributions: This work proposes a framework for cooperative aerial-ground dynamic docking between a quadrotor UAV and a UGV equipped with a rooftop landing platform. A hierarchical control architecture is developed, in which a Nonlinear Model Predictive Controller (NMPC) handles the UAV's path-planning, while a Linear MPC (LMPC) manages trajectory tracking. The UAV and UGV coordinate their actions in real time using the ROS2 middleware, enabling decentralized operation without any physical coupling. To maintain a lightweight and cost-effective design, the system avoids the use of complex mechanical structures such as robotic grippers, relying instead on optimal control and velocity-based actuation to achieve precise docking. The proposed approach is validated through hybrid simulations, where the UGV operates on a real-world circuit while the UAV is simulated in a high-fidelity physics environment. The results demonstrate accurate tracking performance, robust constraint satisfaction, and the successful execution of dynamic landing maneuvers, even under environmental perturbations.

State-of-the-art: Recent advancements in UAV guidance and control have enabled precise interactions with dynamic platforms across diverse scenarios. Procházka et al. [104] propose a model predictive control (MPC) framework tailored for agile drone landings on fast-moving boats, explicitly handling nonlinear ship motion dynamics. Keipour and Zarzhitsky [77] adopt a visual servoing method, relying solely on onboard vision and GPS-free estimation to achieve robust landing on ground vehicles. Gao and Yang [45] introduce a learning-based adaptive tracking and perching strategy, where the quadrotor dynamically adjusts its trajectory in real-time using perception cues. Borowczyk et al. [22] address high-speed landings by combining state estimation and predictive control to compensate for relative motion between UAV and the ground vehicle. Babu and Kumar [8] focus on landing accuracy, developing a method that fuses inertial and visual inputs to enable stable touchdown on erratic moving targets. Cao et al. [27] leverage the YOLO object detection framework to visually track and guide drones toward high-precision landings on moving platforms. Jong and Kim [72] explore a unique "never-landing" strategy where the UAV autonomously soars in front of moving obstacles, inspired by bird flight behavior. Zeng and Liu [132] employ reinforcement learning to train a quadrotor to land on moving targets without explicit modeling, using simulation-based policies that generalize across speeds and directions. Hadiwardoyo et al. [66] contribute a UAV mobility model that supports dynamic communication links between UAVs and cars, critical for vehicular coordination in mobile networks. Chen and Xu [28] develop a hierarchical Proximal Policy Optimization (PPO) algorithm to manage

multi-stage maneuvering tasks, such as approach, alignment, and landing. Together, these works demonstrate the diversity of control strategies, from classical MPC to reinforcement learning, and a wide spectrum of applications, including maritime logistics, urban tracking, autonomous inspection, and communication relay.

TABLE 4.1: UAV and platform velocities in selected studies.

Paper	Platform	Platform Speed	UAV Speed
[104], 2024	Boat	~1 m/s	Matched
[77], 2021	Car	~4 m/s	Matched
[22], 2016	Car	~15 m/s	Matched
Our Research, 2025	Car	~15 m/s	Matched

Outlines: This last chapter is structured as follows. Section 4.2 presents the optimal control problems formulations and a stability analysis of the asynchronous MPCs. Section 4.3 describes the system architecture and communication methodology. Hybrid-simulation results of a docking maneuver are analyzed in Section 4.4. Finally, conclusions and future directions are provided in Section 4.5.

4.2 Optimal control

4.2.1 UAV path-planning

Path-planning consists of computing a feasible and efficient trajectory that respects the system dynamics and constraints, ensuring the quadrocopter can physically dock smoothly on the UGV. While the baseline UAV navigation control follows the formulation of Problem 2.3.1, the docking maneuver requires a specialized control formulation, which is introduced as follows:

Problem 4.2.1 — **OCP$(\mathbf{i}, \mathbf{x}_0^{\mathbf{i}}, \mathbf{J}_k^{\mathrm{UGV}*})$ – Path-planning UAV for UGV-docking**

$$\min_{x_k, u_k} \quad J = \sum_{k=0}^{N} \left[(\eta_k - c_\eta \cdot \eta_k^{\mathrm{UGV}*})^2 + u_k^\top R u_k \right] \tag{4.1a}$$

at discrete time step $k = 0, \ldots, N-1$.

Subject to:

$$x_{k+1} = f(x_k, u_k), \tag{4.1b}$$
$$x_0 = \hat{x}, \tag{4.1c}$$
$$x_k \in \mathcal{X}, \quad u_k \in \mathcal{U}, \tag{4.1d}$$
$$\eta_k = [x, y, z, v_x, v_y, v_z], \tag{4.1e}$$
$$\eta_k^{\mathrm{UGV}*} = \mathrm{proj}_k(\eta^{\mathrm{UGV}}, \eta_k) \tag{4.1f}$$

with the pseudo-state η a subset of the state vector x presented in (2.17). The maneuver involves two asynchronous MPCs and a concept of Euclidean distance between their finite-horizon trajectory is introduced. We compute the distance between each point on the NMPC trajectory, solution of problem 4.2.1, and its corresponding time-aligned projection onto the

finite-horizon trajectories of the LMPC, solution of problem (2.5.1). The projected points on the LMPC trajectory corresponding to the NMPC points at time index k is denoted as $\eta_k^{\mathrm{UGV}*} = \mathrm{proj}_k(\eta^{\mathrm{UGV}}, \eta_k)$, where $\mathrm{proj}_k(\cdot,\cdot)$ represents a projection operator that aligns the NMPC trajectory point at time step k with its temporally equivalent point on the LMPC trajectory. Practically the projection is based on a B-spline routine. The constant $c_\eta \geq 1$ serves as an offset, so the UAV trajectory minimizes its relative velocity vector (with the UAV) but comes fast enough to actually land. Box constraints are applied to controls and states, see Table 4.3, and the second norm of the velocity vector is bounded. The dynamic (2.17) is defined by the nonlinear set of equations presented in chapter 2, and discretized using an explicit Euler scheme. The position $[x, y, z]$ is expressed in the ENU (*east, north, up*) coordinate system and is derived in order to get the velocities $[v_x, v_y, v_z]$ and their derivatives (translational accelerations) hold the nonlinear behavior. The angles (and their derivatives) $[\psi, \phi, \gamma]$ (*roll, pitch, yaw*) are expressed in the *Cardan* referential. The thrust derivatives are controlled as well as the rotational accelerations.

4.2.2 UGV path-tracking

The path-tracking controller ensures the UGV follows a desired reference trajectory as accurately as possible while respecting system constraints and control efforts. It involves real-time correction of deviations caused by disturbances and model mismatch.

Problem (2.5.1) is formulated as a quadratic program (QP) optimal control problem (OCP), as introduced in Chapter 2. The objective function (2.18a) is strictly convex due to the positive definiteness of the weighting matrices $Q \succ 0$ and $R \succ 0$, ensuring the existence and uniqueness of a global optimum, see [24, 96]. The dynamics (2.18b) are linear, and the box constraints (2.18d) and (2.18e) define compact sets $\mathcal{U} \subset \mathbb{R}^m$ and $\mathcal{X} \subset \mathbb{R}^n$, respectively. This structure guarantees well-posedness of the finite-horizon problem and allows for efficient online solutions using standard QP solvers.

Although both the reference path and UAV states are defined in 3D, the LMPC controller operates in a 2D plane. The reference trajectory in the 2D plane $\gamma_r(s) := (x_r(s), y_r(s))^\top$ is parameterized by its arc length $s \in [0, L]$, as detailed in [18, 39].

4.2.3 Hierarchical controllers

LMPC stability: The LMPC formulation (2.5.1) is a strictly convex quadratic program (QP) having a global unique solution [96, 120]. Given a feasible control sequence $u_0, \ldots, u_N$, the corresponding state trajectory $x_0, \ldots, x_N$ is smooth due to the LTV system dynamics and the continuous reference curve. Furthermore, a Lyapunov function can be defined as the terminal cost at discrete time N and its stability can be proved using the solution of a discrete-time algebraic Riccati equation, which guarantees asymptotic stability of the closed-loop system under standard LMPC assumptions [91, 106]. The stabilizing terminal region and feedback law ensure that the value function acts as a Lyapunov function, thus establishing the recursive feasibility and stability of the LMPC.

MPCs sychronization: The coordination of the time-asynchronous MPCs (LMPC for UAV and NMPC for UGV) is achieved by projecting the UGV's predicted trajectory onto the

UAV's spatial reference domain. This is performed via a B-spline fitting procedure, which yields a smooth and Lipschitz continuous mapping. The projection operator $\text{proj}_k(\eta^{\text{UGV}}, \eta_k)$ ensures consistent synchronization across time prediction points, and preserves continuity and bounded derivatives [33].

NMPC stability: The UAV's nonlinear MPC controller solves a nonlinear program (NLP) at each sampling step, based on a nonlinear, non-convex dynamical model. The cost function is quadratic in states and controls, and a local quadratic Lyapunov candidate is constructed from this stage cost J. However, due to non-convexity, the NLP-solver only guarantees convergence to a local minimum. To ensure practical stability, it is shown that the Lyapunov function $V(x)$ satisfies $\dot{V}(x(t)) < 0$ in a neighborhood of the optimal trajectory, provided small enough discretization steps and well-tuned penalty weights. This implies that the closed-loop system exhibits local asymptotic stability around the reference trajectory, as supported by theoretical results in [36, 64] for nonlinear MPC under appropriate regularity and feasibility assumptions.

4.3 Architecture

4.3.1 Communication structure between controllers

As stated, the UGV broadcast its position and orientation in real-time to the UAV as well as its prediction points from the LMPC. From a set of points defining the middle-line of a circuit, a B-spline is computed and serves as reference γ_r. The real-time position and orientation of the UAV is measured using an Inertial Navigation System based on a Inertial Measure Unit (INS $\rightarrow$ GNSS and IMU). The reference point the GNSS localization is in the middle of the roof platform, with a vertical offset to the ground of h_{UGV}. It can be noted that the docking is considered achieved if the vertical distance UAV-platform is less than ten centimeters everywhere on the platform. The communication is based on the middleware ROS2. The drone evolves in a simulated environment (JMavSim [9]) and the MPC-Simulator communication is based on the MAVLink protocol, see [93]. The UAV is controlled, see dynamical system (2.17), using second time derivative for smooth trajectories. However, the velocity vector is given to the simulator as a command. The PX4, see [105], low-level autopilot serves as a tracker. The velocity tracking results are presented in Figure 4.4. The data structure and flux are presented in Figure 4.2.

4.3.2 SQP solver

An in-house Sequential Quadratic Programming (SQP) method [46] and multiple integration schemes are used in order to solve Problem 4.2.1. The documentation is available at : `https://www.unibw.de/ingmathe/research/software`. An in-house Newton Semi-Smooth method is used to solve Problem (2.5.1), see [39]. The Karush-Kuhn-Tucker (KKT) tolerance [96] was set to 10^{-3}.

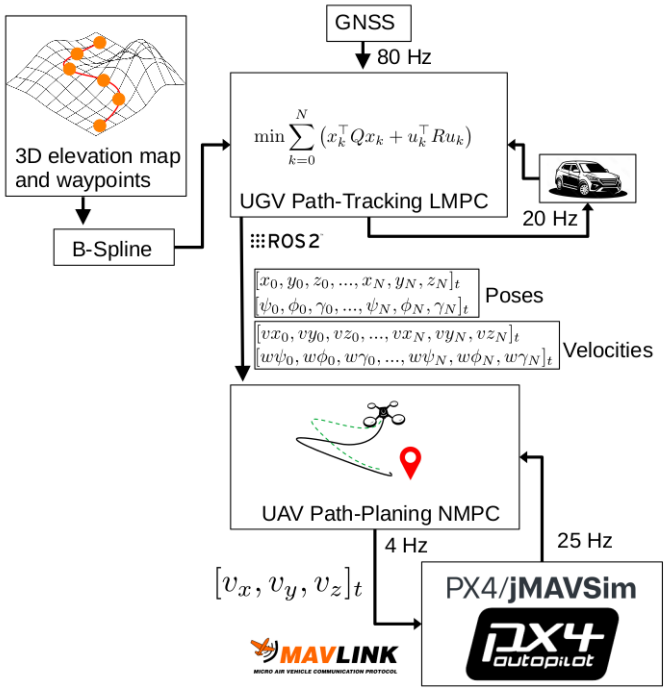


FIGURE 4.2: Architecture UAV-UGV.

4.4 Results

4.4.1 Scenario parameters

Table 4.2 lists the main parameters used for the MPC formulation and the system models of both UAV and UAV.

TABLE 4.2: MPC and system parameters values.

Parameter	**Value**	**Parameter**	**Value**
Horizon UAV	3.5 [s]	N UAV	10
Horizon UGV	20 [s]	N UGV	80
m_{UAV}	4kg	h_{UGV}	1.98 [m]
ϵ_p	0.1m	R	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.5 \end{bmatrix}$

Table 4.3 summarizes the box constraints imposed on the UAV states and control inputs.

TABLE 4.3: UAV states and controls box constraints' values.

Lower limit	Variable	Upper limit
$0\ [\mathrm{m\,s^{-1}}]$	$\|\|v\|\|_2$	$15\ [\mathrm{m\,s^{-1}}]$
$-5\ [\mathrm{m\,s^{-2}}]$	a	$5\ [\mathrm{m\,s^{-2}}]$
$-1.4\ [\mathrm{rad}]$	ψ	$1.4\ [\mathrm{rad}]$
$-1.4\ [\mathrm{rad}]$	ϕ	$1.4\ [\mathrm{rad}]$
$-2.5\ [\mathrm{rad\,s^{-1}}]$	ω_ψ	$2.5\ [\mathrm{rad\,s^{-1}}]$
$-2.5\ [\mathrm{rad\,s^{-1}}]$	ω_ϕ	$2.5\ [\mathrm{rad\,s^{-1}}]$
$-1\ [\mathrm{rad\,s^{-1}}]$	ω_γ	$1\ [\mathrm{rad\,s^{-1}}]$
$0\ [\mathrm{kg\,m/s^{-2}}]$	N	$10 \cdot g \cdot m_{UAV}\ [\mathrm{kg\,m/s^{-2}}]$
$-1.75\ [\mathrm{rad\,s^{-2}}]$	u_ψ	$1.75\ [\mathrm{rad\,s^{-2}}]$
$-1.75\ [\mathrm{rad\,s^{-2}}]$	u_ϕ	$1.75\ [\mathrm{rad\,s^{-2}}]$
$-0.15\ [\mathrm{rad\,s^{-2}}]$	u_γ	$0.15\ [\mathrm{rad\,s^{-2}}]$
$-3\ [\mathrm{rad\,s^{-1}}]$	u_N	$3\ [\mathrm{rad\,s^{-1}}]$

4.4.2 Realization

Figure 4.3 presents the trajectories of the UAV and UGV during the docking maneuver over a time window of fifty seconds within an area of a square kilometer. The top subplot shows the 2D-plan $[east, north]$ trajectories, where the UAV approaches the vicinity of the UGV before minimizing their relative position and velocity, considering a C_η offset. The middle subplot displays the altitude evolution of both vehicles. The UAV starts at an altitude of ten meters, as the takeoff phase lies outside the scope of this work. The docking is interpreted as a mission termination phase. During the initial seconds, the UAV's altitude drops rapidly due to high pitch and roll rates, which momentarily diminish the vertical thrust component (see Figure 4.5). As the altitude is bounded but not penalized in the cost function of Problem 4.2.1 this behavior is expected. The bottom subplot provides a zoomed-in view of the final docking phase, highlighting the initial MPC prediction points. Some UAV (blue) points are missing due to data loss or logging limitations. Despite this, the UAV's predicted trajectories (in grey) closely follow the actual flight path, demonstrating the high accuracy of the MPC strategy coupled with the PX4 autopilot in the JMavSim simulator.

The presented results are also available as a video in the *Unreal Engine 5* virtual environment:

`https://youtu.be/RRu12HCWzMQ?si=pQ_ZhCIEkP6-KIWV` ,

as shown in Figure 4.1.

Figure 4.4 compares the UAV velocity components between the MPC-predicted $v_{x,y,z}(k+1)$ and the velocities executed by the PX4 autopilot in simulation. The profiles exhibit a high

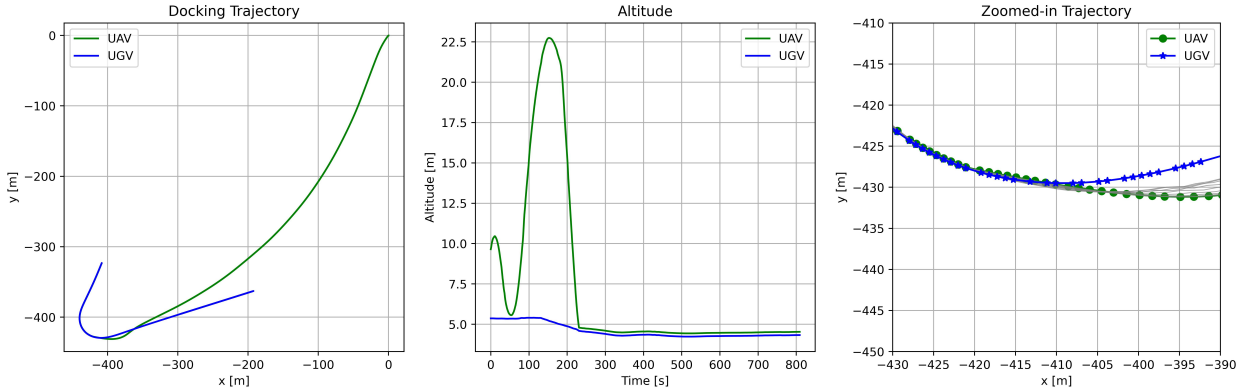


FIGURE 4.3: Trajectories UAV-UGV.

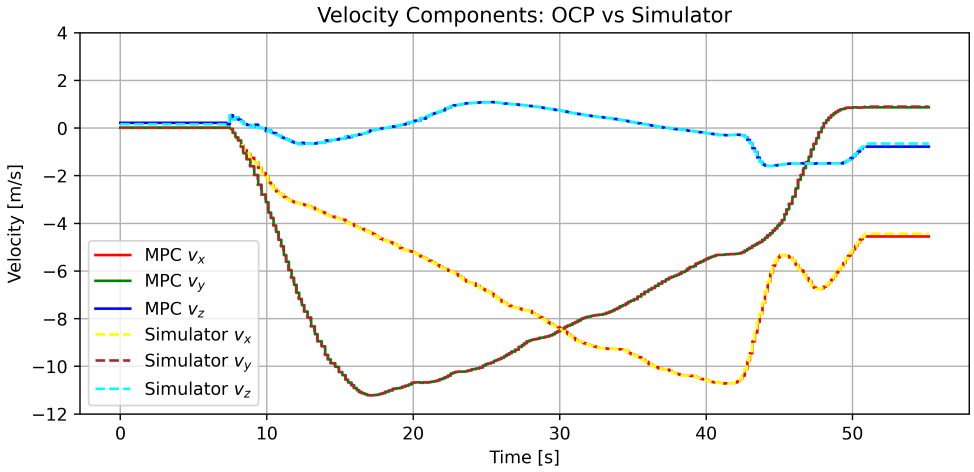


FIGURE 4.4: UAV Velocities: MPC vs. simulator.

level of compatibility throughout the maneuver, proving the followability of the predicted trajectories by the autopilot, even during sharp transitions and dynamic phases of the trajectory.

In Figure 4.5, the control inputs generated by the MPC are shown, specifically the angular accelerations in yaw (u_ψ), roll (u_ϕ), and pitch (u_γ), as well as the thrust acceleration (u_N). Although these control inputs are not directly applied to the UAV, since a velocity control strategy is used, they govern the evolution of the velocity commands, resulting in the smooth velocity profiles observed in Figure 4.4. The angular accelerations display sharp variations, including bang-bang behavior during aggressive reorientation phases, yet they remain within the predefined constraint bounds imposed by the optimal control problem. Of particular interest is the thrust acceleration (u_N), which, while generally centered around zero, exhibits several clearly negative phases. These intervals correspond to a deliberate reduction in thrust during descent, culminating in periods of zero thrust. While this behavior is consistent with the solution of the optimal control problem, minimizing control effort while staying within admissible bounds, it introduces a critical issue from a practical standpoint. In real-world UAV operations, a zero-thrust descent leads to a loss of control.

Figure 4.6 shows a histogram of the computational time required to solve the NMPC optimization problem at each control loop. The red vertical line indicates the nominal MPC sampling period, defined as the prediction horizon divided by the number of discretization

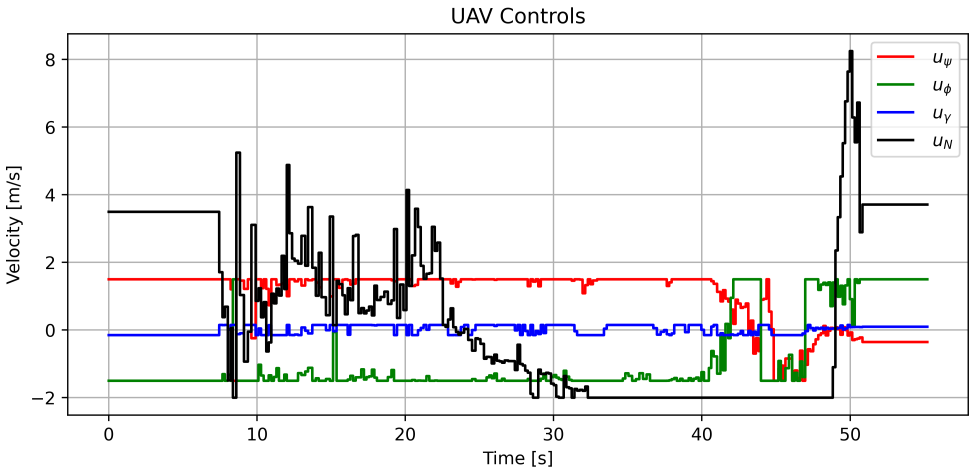


FIGURE 4.5: UAV controls.

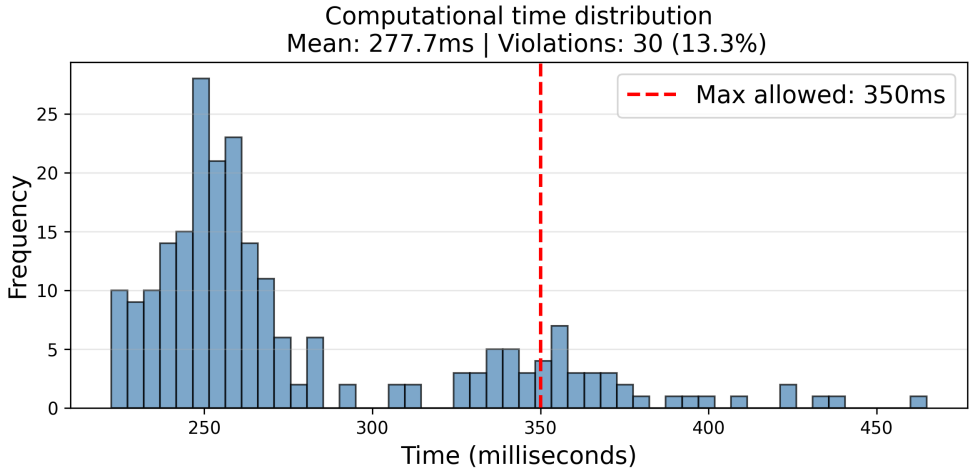


FIGURE 4.6: Distribution of the computational time required for solving the MPC optimal control problem at each control loop of the quadrocopter.

points (see Table 4.2). Approximately 13% of the control iterations exceed this time limit, which is non-negligible. However, the majority of these violations remain within 15% of the nominal MPC period (350ms), and only a small number of iterations require up to 450ms. Occasional overruns are not critical for system safety, as the PX4 low-level controller continues to track the previously issued velocity command. Nevertheless, such events should be kept infrequent to avoid performance degradation. One possible mitigation strategy is to enforce a hard time limit on the optimization and return the best available solution candidate when the limit is reached, even if the termination criteria have not yet been satisfied.

Figure 4.7 illustrates the computational time required to solve the MPC problem over the course of the simulation. The control loop violations are approximately uniformly distributed over time. Moreover, no violations occur during the final 10 seconds of the simulation, corresponding to the terminal docking phase, where the risk of collision is at the highest. This indicates that the computational burden does not increase during the critical docking maneuver for the presented scenario.

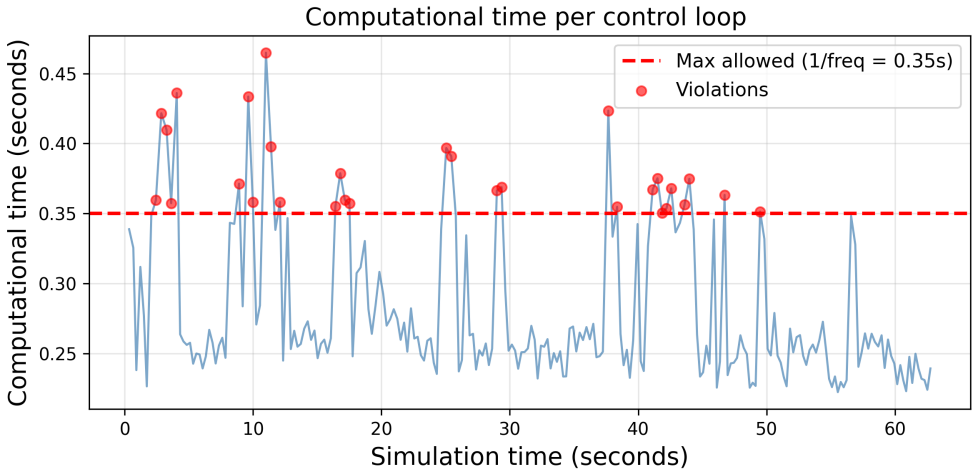


FIGURE 4.7: Computational time required at each control loop to solve the path-planning problem for the quadrocopter.

4.5 Chapter conclusion

The asynchronous MPC-based docking maneuver achieves real-time smooth docking, while both controllers are separated but communicate hierarchically using the ROS2 protocol. The UAV NMPC control quality was shown to be effectively traceable by a common PX4 autopilot using a velocity vector command. The pipeline runs in real-time, on a standard laptop for UAV simulation coupled to a real autonomous UGV, as presented in the hybrid-simulation results. Using the standard ROS2 and MAVLink protocols, this setup ensures both reproducibility and easy deployment in real UAV platforms.

Part II conclusion

This work has presented a comprehensive framework for model predictive control of autonomous vehicles, spanning from theoretical foundations to practical implementations. Chapter 1 introduced the optimal control problem (OCP) formulations and a symbolic MPC framework, including both linear (LMPC) and nonlinear (NMPC) variants, together with considerations on stability, architecture, and implementation in networked systems with simulators and hardware-in-the-loop setups.

Chapter 2 extended this framework to various classes of autonomous vehicles, including quadrocopters, fixed-wing aircraft, and ground vehicles. Path-planning and path-tracking problems were formulated as MPC tasks, incorporating system dynamics, state and control constraints, and trajectory objectives, providing a unified foundation for consistent and scalable control strategies in real-world environments.

In Chapter 3, a human-machine interface was developed to guide pilots in navigating congested or hazardous areas, demonstrating the robustness and adaptability of the NMPC-based path-planner. Real-time trajectory guidance, combined with fine-tuning of aircraft agility, enabled successful avoidance of multiple dynamic zones and illustrated the potential for manned-unmanned or multi-agent teaming scenarios.

Finally, Chapter 4 presented a hierarchical MPC approach for UAV-UGV docking maneuvers. The asynchronous controllers communicate via ROS2 and MAVLink protocols, achieving real-time, smooth docking trajectories. The UAV NMPC controller effectively interfaced with a standard PX4 autopilot, while the overall simulation and hybrid setup demonstrated reproducibility, real-time performance, and practical deployment potential for autonomous vehicle operations.

Together, these chapters establish a complete pipeline from theory to application, demonstrating that the proposed MPC and OCP frameworks are flexible, scalable, and effective across multiple autonomous platforms and operational scenarios.

Conclusion

This dissertation demonstrates the development of robust and efficient quadratic programming (QP) solvers and their application to autonomous vehicle control. The first part focused on the design of a stabilized primal-dual interior-point method (IPM) enhanced with proximal and primal-dual augmented Lagrangian regularization and safeguarding strategies. These contributions address degenerated QPs, ensure convergence, and reduce sensitivity to solver initialization. Additionally, nonlinear programming (NLP) methods, including sequential quadratic programming (SQP), were explored to solve constrained and unconstrained problems. Lightweight reinforcement learning techniques were also applied for hyperparameter tuning, showing that solver performance can be significantly improved with minimal computational overhead. The combination of these solver advances provides a solid foundation for handling complex optimal control problems in real-time applications.

Building on this theoretical foundation, the second part explored a range of real-world applications involving quadrocopter, fixed-wing aircraft, and ground vehicles. Model predictive control (MPC) frameworks were consistently formulated for path planning and tracking, ensuring dynamic feasibility, obstacle avoidance, and smooth trajectory generation. The hierarchical integration of multiple controllers enabled coordinated UAV–UGV interactions, including a real-time docking maneuver. Experiments with hybrid simulations and hardware-in-the-loop validated both the performance and reproducibility of the proposed methods. The human-machine interface demonstrated how predictive control can support operator decision-making, enhancing safety and agility in congested or hazardous environments.

Overall, the dissertation demonstrates that advances in solver design directly enable high-performance control in complex, multi-agent autonomous systems. By unifying theoretical development with applied implementation, it establishes a framework that is both rigorous and adaptable. While current results focus on single- and few-agent scenarios, the methodologies are extendable to larger fleets, multi-agent coordination, and real-time decision-making under uncertainty. Future research may explore further integration with sensor-based detection, stochastic disturbances, and human-in-the-loop supervision to expand operational capabilities.

In summary, this work provides a comprehensive contribution from robust QP solver theory to practical, real-time control of heterogeneous autonomous vehicles, highlighting both scientific innovation and practical applicability in modern robotics and unmanned systems.

Appendix - Maros–Mészáros test-set results table

TABLE 4.4: Maros–Mészáros test-set results.

name	status	epsilon	run_time	iter	obj_val	n	m	p	nnz
TAME	optimal	6.9×10^{-9}	0.000486	9	0	2	1	2	15
HS21	optimal	4.1×10^{-11}	0.001534	37	0.0400202	2	0	5	19
ZECEVIC2	optimal	8.2×10^{-11}	0.000407	9	-4.125	2	0	6	24
QPTEST	optimal	1×10^{-9}	0.000399	9	4.37208	2	0	5	23
HS35MOD	optimal	3.4×10^{-9}	0.000852	20	-8.75	3	1	3	23
HS35	optimal	5.1×10^{-11}	0.000406	10	-8.88889	3	0	4	23
HS51	optimal	5.1×10^{-9}	0.000776	25	-6	5	3	0	26
HS52	optimal	2.5×10^{-9}	0.000393	12	-0.673352	5	3	0	26
HS53	optimal	1.5×10^{-9}	0.000605	10	-1.90698	5	3	10	56
HS76	optimal	6.4×10^{-9}	0.000735	18	-4.68145	4	0	7	43
S268	optimal	2×10^{-9}	0.000725	15	-14463	5	0	5	80
HS268	optimal	2×10^{-9}	0.000708	15	-14463	5	0	5	80
GENHS28	optimal	2.2×10^{-9}	0.000528	13	0.927174	10	8	0	84
LOTSCHD	optimal	4.3×10^{-11}	0.002365	32	2398.42	12	7	12	163
HS118	optimal	4.4×10^{-16}	0.003418	42	664.82	15	0	59	260
QAFIRO	optimal	9.4×10^{-9}	0.010217	109	-1.59069	32	8	51	327
CVXQP2_S	optimal	1.2×10^{-9}	0.012523	16	8120.94	100	25	200	1445
CVXQP1_S	optimal	3.6×10^{-9}	0.015939	20	11590.7	100	50	200	1618
CVXQP3_S	optimal	5.5×10^{-9}	0.024335	37	11943.4	100	75	200	1791
QADLITTL	optimal	4.3×10^{-9}	0.02917	66	480319	97	15	138	1350
QPCBLEND	optimal	2.5×10^{-9}	0.015318	36	-0.00784236	83	43	114	1388
QSCAGR7	optimal	7.3×10^{-12}	0.063946	230	26865900	140	84	185	1563
QSHARE2B	optimal	4.1×10^{-11}	0.010655	65	11703.7	79	13	162	1890
QRECIPE	optimal	9.2×10^{-9}	0.03375	31	-266.616	180	91	249	2404
QSC205	optimal	8.8×10^{-9}	0.033167	27	-0.00581351	203	91	317	2139
DUAL4	optimal	8.6×10^{-9}	0.012263	16	0.746091	75	1	150	6124
DPKLO1	optimal	9.8×10^{-9}	0.059211	61	0.370096	133	77	0	3360
DUAL1	optimal	1.4×10^{-9}	0.015041	23	0.0350133	85	1	170	7712
QSHARE1B	optimal	9.5×10^{-9}	0.397607	871	720078	225	89	253	3361
QPCBOEI2	optimal	2.3×10^{-10}	0.074506	221	8171960	143	4	378	3485
DUALC2	optimal	2.9×10^{-11}	0.005887	27	3551.31	7	1	242	3526
DUAL2	optimal	1.2×10^{-10}	0.006652	19	0.0337339	96	1	192	9689
VALUES	optimal	9.3×10^{-9}	0.005475	14	-1.38829	202	1	404	9059
QBORE3D	optimal	7.2×10^{-10}	0.097782	218	3100.2	315	215	344	4484
DUALC1	optimal	9.3×10^{-10}	0.007726	39	6155.25	9	1	232	4220
PRIMALC2	optimal	1×10^{-12}	0.007182	33	-3551.31	231	0	236	4159
QSCORPIO	optimal	9.1×10^{-10}	0.017345	36	1880.51	358	280	466	4708
QBRANDY	optimal	8×10^{-9}	0.059737	121	28375.1	249	166	303	5610
DUALC5	optimal	1.9×10^{-11}	0.003621	14	427.232	8	1	293	4838
DUAL3	optimal	4.9×10^{-9}	0.007715	17	0.135762	111	1	222	12994
PRIMALC1	optimal	1.7×10^{-12}	0.0103	40	-6155.25	230	0	224	5024
QGROW7	optimal	2.4×10^{-10}	0.486897	239	-42798700	301	140	581	8062
QSCAGR25	optimal	4.2×10^{-11}	0.907618	1082	201738000	500	300	671	5779
QCAPRI	optimal	1.9×10^{-9}	1.0674	1114	66793300	353	158	583	7244
GOULDQP2	optimal	7.7×10^{-9}	0.052949	49	0.000184276	699	349	1398	8032
QISRAEL	optimal	2.5×10^{-10}	0.11642	156	25347800	142	0	316	6592

name	status	epsilon	run_time	iter	obj_val	n	m	p	nnz
PRIMALC5	optimal	2.7×10^{-13}	0.009835	18	-427.232	287	0	286	5721
QBEACONF	optimal	2×10^{-9}	0.07791	85	164712	262	140	295	7989
QSCTAP1	optimal	4.9×10^{-9}	0.013623	13	0.000186835	480	120	660	5838
QBANDM	optimal	6.7×10^{-9}	0.078547	57	16352.3	472	305	472	7213
GOULDQP3	optimal	2.2×10^{-9}	0.055108	46	-29647.8	699	349	1398	8730
QE226	optimal	1.2×10^{-9}	0.070919	54	205.54	282	33	472	8301
QSCSD1	optimal	6.1×10^{-9}	0.043445	35	8.66667	760	77	760	9275
QSCFXM1	optimal	5.2×10^{-9}	0.437595	381	16882700	457	187	600	8690
DUALC8	optimal	4.3×10^{-9}	0.023147	32	18309.4	8	1	518	8663
QFORPLAN	optimal	2.8×10^{-9}	0.337152	240	7456630000	421	93	511	12129
CVXQP2_M	optimal	1.3×10^{-11}	0.85688	21	820155	1000	250	2000	14716
QPCBOEI1	optimal	7.8×10^{-9}	0.100532	88	11503900	384	9	971	10082
QGFRDXPN	optimal	4.5×10^{-8}	4.47931	3000	100791000000	1092	548	1418	10728
QPCSTAIR	optimal	6.8×10^{-11}	0.294724	207	6204390	467	291	532	9936
QSTAIR	optimal	3.4×10^{-9}	3.31269	2044	7985450	467	291	532	11840
CVXQP1_M	optimal	7.1×10^{-11}	2.258	36	1087510	1000	500	2000	16464
QETAMACR	optimal	1.9×10^{-9}	1.89118	71	86760.4	688	354	869	16513
PRIMALC8	optimal	9.6×10^{-13}	0.02011	27	-18309.4	520	0	511	10357
QSTANDAT	optimal	1×10^{-9}	0.038436	41	6411.84	1075	176	1362	12365
QSCRS8	optimal	1×10^{-8}	0.235707	182	904.562	1169	384	1275	11706
QGROW15	optimal	7.3×10^{-10}	1.24178	506	-101694000	645	300	1245	16844
QSEBA	optimal	2.8×10^{-9}	0.880118	491	81481800	1028	507	1550	15989
CVXQP3_M	optimal	1.1×10^{-9}	16.0836	90	1362830	1000	750	2000	18212
MOSARQP2	optimal	1.4×10^{-9}	0.043645	31	-1597.48	900	0	1500	10150
PRIMAL1	optimal	9×10^{-9}	0.009872	10	0.85455	325	0	86	12043
QSCSD6	optimal	7.2×10^{-9}	0.118964	41	50.8082	1350	147	1350	16795
QSHIP04S	optimal	4.9×10^{-9}	0.067739	33	2424990	1458	354	1506	15022
QSCFXM2	optimal	6.3×10^{-9}	1.68477	780	27776200	914	374	1200	16796
LASER	optimal	5.7×10^{-13}	0.066097	29	2409600	1002	0	2000	19460
QFFFFF80	optimal	1.3×10^{-9}	5.6809	1176	873147	854	350	1028	19670
QGROW22	optimal	1.2×10^{-9}	2.04462	679	-149629000	946	440	1826	24942
PRIMAL2	optimal	2.7×10^{-9}	0.026979	10	0.00916266	649	0	97	16832
QSHIP04L	optimal	4.9×10^{-9}	0.101924	36	2420020	2118	354	2166	21622
QSCFXM3	optimal	8.8×10^{-9}	3.30084	1016	30816400	1371	561	1800	24292
MOSARQP1	optimal	3.7×10^{-9}	0.075125	23	-952.874	2500	0	3200	17634
QSCTAP2	optimal	9.6×10^{-9}	0.048462	13	0.00027406	1880	470	2500	23310
QSIERRA	failed	1.6×10^{0}	24.5736	3000	23750500	2036	528	4751	30145
QSCSD8	optimal	7.6×10^{-9}	0.145267	33	940.764	2750	397	2750	33305
YAO	optimal	4.6×10^{-9}	0.086078	53	-75.8073	2002	2	2001	16011
AUG3DQP	optimal	2.3×10^{-9}	0.880864	18	-661.463	3873	1000	3873	29584
QSHIP08S	optimal	2.9×10^{-9}	2.39056	32	2385730	2387	698	2467	46832
QPILOTNO	optimal	9.9×10^{-10}	51.8031	1006	4728590	2172	905	2582	37579
AUG3DCQP	optimal	1.8×10^{-10}	0.737214	19	-976.692	3873	1000	3873	29584
AUG3DC	optimal	7×10^{-9}	0.109027	22	-1165.24	3873	1000	0	17965
QSCTAP3	optimal	1.2×10^{-10}	0.038779	16	0.0000252413	2480	620	3340	30870
CONT-050	optimal	7.5×10^{-9}	1.2311	28	-4.56385	2597	2401	5194	44590
QSHELL	optimal	9.9×10^{-9}	112.482	1004	1572640000000	1775	784	1644	83869
QSHIP12S	optimal	1.6×10^{-10}	0.890167	38	3056960	2763	1045	2869	61281
PRIMAL4	optimal	5.7×10^{-9}	0.028608	11	-0.719484	1489	0	76	33629
STADAT2	optimal	7.2×10^{-9}	0.264966	71	-32.6267	2001	0	5999	35994
STADAT1	optimal	2.5×10^{-9}	0.196224	50	-28526900	2001	0	5999	35994
KSIP	optimal	6.2×10^{-9}	0.034288	11	0.590361	20	0	1001	40817
PRIMAL3	optimal	5.4×10^{-9}	0.049551	10	0.0160099	745	0	112	43953
STCQP1	optimal	5.3×10^{-9}	6.94977	17	155144	4097	2052	8194	102419
STCQP2	optimal	6.1×10^{-9}	8.76875	25	22327.3	4097	2052	8194	102419
QSHIP08L	optimal	9.5×10^{-10}	10.3317	39	2376040	4283	698	4363	111564
Q25FV47	optimal	3.5×10^{-9}	31.4227	165	13744400	1571	515	1876	146010
STADAT3	optimal	3.8×10^{-9}	0.695469	92	-35.7795	4001	0	11999	71994
HUESTIS	optimal	5.5×10^{-11}	1.91551	188	348245000000	10000	2	10000	80002
HUES-MOD	optimal	5.9×10^{-12}	1.36318	131	34824500	10000	2	10000	80002
QSHIP12L	optimal	2.9×10^{-10}	18.2528	41	3018880	5427	1045	5533	175609
CVXQP2_L	optimal	2.3×10^{-9}	155.355	35	81842500	10000	2500	20000	147466

name	status	epsilon	run_time	iter	obj_val	n	m	p	nnz
CVXQP1_L	optimal	3.1×10^{-8}	1002.55	127	108705000	10000	5000	20000	164964
POWELL20	failed	1.8×10^{3}	26.9837	3000	29157000000	10000	0	10000	60000
CVXQP3_L	failed	3.7×10^{2}	1023.73	53	18706100	10000	7500	20000	182462
CONT-101	optimal	1×10^{-11}	415.725	436	0.195527	10197	10098	20394	180675
LISWET9	optimal	1.4×10^{-7}	144.279	3000	-1121.33	10002	0	10000	80002
LISWET10	optimal	1×10^{-8}	54.0601	2423	-2484.57	10002	0	10000	80002
LISWET12	optimal	1.1×10^{-7}	143.917	3000	-1275	10002	0	10000	80002
LISWET11	optimal	1×10^{-8}	32.8516	722	-2478.55	10002	0	10000	80002
LISWET8	optimal	2.3×10^{-8}	89.8799	3000	-1920.17	10002	0	10000	80002
LISWET4	optimal	9.5×10^{-9}	0.59442	68	-714.516	10002	0	10000	80002
LISWET3	optimal	6.4×10^{-9}	0.487589	56	-1000.26	10002	0	10000	80002
LISWET2	optimal	8.1×10^{-9}	0.930057	57	-1666.99	10002	0	10000	80002
LISWET7	optimal	3.7×10^{-8}	157.294	3000	-2291.05	10002	0	10000	80002
LISWET1	optimal	9.9×10^{-9}	3.04173	100	-2497.13	10002	0	10000	80002
LISWET5	optimal	8.4×10^{-9}	0.480122	56	-15976.2	10002	0	10000	80002
LISWET6	optimal	9.3×10^{-9}	0.600388	55	-2162.22	10002	0	10000	80002
CONT-100	optimal	1.6×10^{-10}	24.3538	38	-4.6444	10197	9801	20394	179190
DTOC3	optimal	9.3×10^{-9}	1.3245	73	235.262	14999	10000	0	94989
UBH1	optimal	8×10^{-10}	0.525039	21	1.116	18009	12012	12006	162063
AUG2DQP	optimal	3.1×10^{-9}	4.56839	33	6227110	20200	10000	20200	170800
AUG2DCQP	optimal	1.8×10^{-10}	3.59775	24	6487020	20200	10000	20200	170800
AUG2D	optimal	8.8×10^{-9}	2.83725	28	1.7×10^{6}	20200	10000	0	110200
AUG2DC	optimal	6.7×10^{-9}	3.65336	28	1.8×10^{6}	20200	10000	0	110200
EXDATA	failed	–	–	–	–	3000	6001	12000	2278500
CONT-201	optimal	8.7×10^{-9}	928.232	33	0.0720147	40397	40198	80794	721375
CONT-200	optimal	2.2×10^{-9}	570.564	23	-4.72063	40397	39601	80794	718390
CONT-300	optimal	6.7×10^{-9}	3.65336	28	1.8×10^{6}	90597	0	181194	204294
BOYD1	failed	–	–	–	–	93261	93279	186522	932029
BOYD2	failed	–	–	–	–	93263	279794	559570	1449663
AUG3D	optimal	4×10^{-9}	0.823629	20	-782.432	3873	1000	0	17965

Bibliography

[1] *Geolib library, https://deltares.github.io/GEOLib/latest/user/intro.html*.

[2] Intel® nuc, programming for the custom solutions header, 2016, https://www.intel.com/content/dam/support/us/en/documents.

[3] Min Adhikari and Anton Ruiter. Real-time autonomous obstacle avoidance for fixed-wing uavs using a dynamic model. *Journal of Aerospace Engineering*, 33:04020027, 07 2020.

[4] Joao S. Albuquerque, Vipin Gopal, George H. Staus, Lorenz T. Biegler, and B. Erik Ydstie. Interior point sqp strategies for structured process optimization problems. *Computers and Chemical Engineering*, 21:S853–S859, 1997. Supplement to Computers and Chemical Engineering.

[5] Brandon Amos. Tutorial on amortized optimization. *arXiv:2202.00665*, 2023.

[6] David Applegate, Mateo Diaz, Oliver Hinder, Haihao Lu, Miles Lubin, Brendan O' Donoghue, and Warren Schudy. Practical large-scale linear programming using primal-dual hybrid gradient. In *Advances in Neural Information Processing Systems*, volume 34, pages 20243–20257. Curran Associates, Inc., 2021.

[7] Charles Audet, Kien-Cong Dang, and Dominique Orban. Optimization of algorithms with OPAL. *Mathematical Programming Computation*, 6(3):233–254, 2014.

[8] S. Kiran Babu and Ashok Kumar. A robust and accurate landing methodology for drones on moving targets. *Drones*, 6(4):98, 2022.

[9] Alexey Babushkin. jMAVSim: A simple multirotor simulator. Online; accessed 2025-07-09, 2013. Available at `https://github.com/PX4/jMAVSim`.

[10] Siva Balakrishnan. Convex optimization: Lecture 11 - qp duality. 2023.

[11] Antoine Bambade, Fabian Schramm, Sarah El-Kazdadi, Stéphane Caron, Adrien Taylor, and Justin Carpentier. Proxqp: an efficient and versatile quadratic programming solver for real-time robotics applications and beyond. 09 2023.

[12] Alex Barclay, Philip E. Gill, and J. Ben Rosen. Sqp methods and their application to numerical optimal control. In Werner H. Schmidt, Knut Heier, Leonhard Bittner, and Roland Bulirsch, editors, *Variational Calculus, Optimal Control and Applications*, pages 207–222. Birkhäuser Basel, Basel, 1998.

[13] Hassan Bashir and Ximing Liang. Interior point method based sequential quadratic programming algorithm with quadratic search for nonlinear optimization. *International Journal of Computer Information Systems and Industrial Management Applications*, 3, 12 2011.

[14] Randal Beard, Derek Kingston, Morgan Quigley, Deryl Snyder, Reed Christiansen, Walt Johnson, Tim McLain, and Michael Goodrich. Autonomous vehicle technologies for small fixed-wing uavs. *Journal of Aerospace Computing, Information, and Communication*, 2, 02 2005.

[15] Alberto Bemporad. A numerically stable solver for positive semidefinite quadratic programs based on nonnegative least squares. *IEEE Transactions on Automatic Control*, 63(2):525–531, 2018.

[16] Jeremy Bertoncini, Adrian Dudek, Martin Russ, Matthias Gerdts, and Peter Stütz. Fixed-wing uav path planning and collision avoidance using nonlinear model predictive control and sensor-based cloud detection. In *2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC)*, pages 1–10, 2023.

[17] Jeremy Bertoncini, Alberto De Marchi, Matthias Gerdts, and Simon Gottschalk. Reinforcement learning for adaptive interior point methods in convex quadratic programming, 2025.

[18] Jeremy Bertoncini, Viktoriya Nikitina, and Matthias Gerdts. Multi-Agent Dynamic Scheduling with A Posteriori Path Tracking and Collision Avoidance Using Model Predictive Control. *IEEE Control Systems Letters*, 2023.

[19] Dimitri P. Bertsekas. Constrained optimization and lagrangian multiplier methods. *Athena Scientific, Belmont, Massachusetts*, 1996.

[20] Ernesto G. Birgin and José M. Martínez. Augmented Lagrangian method with nonmonotone penalty parameters for constrained optimization. *Computational Optimization and Applications*, 51(3):941–965, 2012.

[21] E. Blum and W. Oettli. *Mathematische Optimierung: Grundlagen und Verfahren*. Oekonometrie und Unternehmensforschung. Springer-Verlag, 1975.

[22] Michał Borowczyk, Jan Piskorz, Mateusz Wnuk, and Jurek Sasiadek. Autonomous landing of a multirotor micro air vehicle on a high velocity ground vehicle. *arXiv preprint arXiv:1611.07329*, 2016.

[23] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[24] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[25] Andreas Britzelmeier and Matthias Gerdts. A nonsmooth newton method for linear model-predictive control in tracking tasks for a mobile robot with obstacle avoidance. *IEEE Control Systems Letters*, 4(4):886–891, 2020.

[26] Richard Byrd, Jorge Nocedal, and Richard Waltz. Feasible interior methods using slacks for nonlinear optimization. *Computational Optimization and Applications*, 26, 03 2002.

[27] Qiang Cao, Yifan Liu, and Jun Wang. High-precision landing on a moving platform based on drone vision using yolo algorithm. *Drones*, 9(4):261, 2023.

[28] Haoran Chen and Yujia Xu. Research on unmanned aerial vehicle intelligent maneuvering method based on hierarchical ppo. *Processes*, 13(2):357, 2025.

[29] Stefano Cipolla and Jacek Gondzio. Proximal stabilized interior point methods and low-frequency-update preconditioning techniques. *Journal of Optimization Theory and Applications*, 197(3):1061–1103, 2023.

[30] Marco Colombo and Jacek Gondzio. Further development of multiple centrality correctors for interior point methods. *Computational Optimization and Applications*, 41:277–305, 12 2008.

[31] G.B. Dantzig. *Linear Programming and Extensions*. Princeton landmarks in mathematics and physics. Princeton University Press, 1998.

[32] GEORGE B. DANTZIG. *Linear Programming and Extensions*. Princeton University Press, 1991.

[33] Carl de Boor. *A Practical Guide to Splines*, volume 27 of *Applied Mathematical Sciences*. Springer, 1978.

[34] Alberto De Marchi. On a primal-dual Newton proximal method for convex quadratic programs. *Computational Optimization and Applications*, 81(2):369–395, 3 2022.

[35] Alberto De Marchi. Regularized interior point methods for constrained optimization and control. *IFAC-PapersOnLine*, 56(2):1247–1252, 2023.

[36] Moritz Diehl, Hans Georg Bock, Jörg P. Schlöder, Rolf Findeisen, Zoltán Nagy, and Frank Allgöwer. A real-time iteration scheme for nonlinear optimization in optimal control. *SIAM Journal on Control and Optimization*, 43(5):1714–1736, 2005.

[37] Alexandre d'Aspremont, Damien Scieur, and Adrien Taylor. Acceleration methods. *Foundations and Trends® in Optimization*, 5(1–2):1–245, 2021.

[38] M Emam and M Gerdts. Sensitivity updates for linear-quadratic optimization problems in multi-step model predictive control. *Journal of Physics: Conference Series, International Workshop on Mathematical Modeling and Scientific Computing*, 2514, 2022.

[39] Mostafa Emam, Thomas Rottmann, and Matthias Gerdts. Efficient real-time obstacle avoidance using multi-objective nonlinear model predictive control and semi-smooth newton method. In *Proceedings of the 10th International Conference on Vehicle Technology and Intelligent Transport Systems - VEHITS*, pages 178–189. INSTICC, SciTePress, 2024.

[40] Eugene A. Feinberg and Adam Shwartz, editors. *Handbook of Markov Decision Processes: Methods and Applications*, volume 40 of *International Series in Operations Research & Management Science*. Springer, Boston, MA, 2002.

[41] Hans Joachim Ferreau, Christian Kirches, Andreas Potschka, Hans Georg Bock, and Moritz Diehl. qpOASES: a parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.

[42] R Fletcher. *Practical Methods of Optimization*. John Wiley and Sons, Ltd, 2000.

[43] R. Fletcher and M. J. D. Powell. A rapidly convergent descent method for minimization. *The Computer Journal*, 6(2):163–168, 08 1963.

[44] A. Forsgren. On warm starts for interior methods. In F. Ceragioli, A. Dontchev, H. Futura, K. Marti, and L. Pandolfi, editors, *System Modeling and Optimization*, pages 51–66, Boston, MA, 2006. Springer US.

[45] Fei Gao and Jie Yang. Adaptive tracking and perching for quadrotor in dynamic scenarios. *arXiv preprint arXiv:2312.11866*, 2023.

[46] M. Gerdts. Ocpid-dae1 optimal control and parameter identification with differential-algebraic equations of index 1. 2013.

[47] Matthias Gerdts. *Optimal Control of ODEs and DAEs*. De Gruyter Textbook. De Gruyter, Berlin, Boston, 2011. 1st edition.

[48] Matthias Gerdts. Numerical experiments with multistep model-predictive control approaches and sensitivity updates for the tracking control of cars. 09 2018.

[49] Matthias Gerdts and Frank Lempio. *Mathematische Optimierungsverfahren des Operations Research*. De Gruyter, 2011.

[50] Alexandre Ghannad, Dominique Orban, and Michael A. Saunders. Linear systems arising in interior methods for convex optimization: a symmetric formulation with bounded condition number. *Optimization Methods and Software*, 37(4):1344–1369, 2022.

[51] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, 1981.

[52] Philip E. Gill, Walter Murray, Michael A. Saunders, J. A. Tomlin, and Margaret H. Wright. On projected newton barrier methods for linear programming and an equivalence to karmarkar's projective method. *Mathematical Programming*, 1986.

[53] Philip E. Gill and Daniel P. Robinson. A primal-dual augmented Lagrangian. *Computational Optimization and Applications*, 51(1):1–25, 1 2012.

[54] Philip E. Gill and Daniel P. Robinson. A globally convergent stabilized sqp method. *SIAM Journal on Optimization*, 23(4):1983–2010, 2013.

[55] Giorgio Giorgi. Various proofs of the sylvester criterion for quadratic forms. *Journal of Mathematics Research*, 9:55–55, 10 2017.

[56] Pontus Giselsson and Stephen Boyd. Linear convergence and metric selection for Douglas–Rachford splitting and ADMM. *IEEE Transactions on Automatic Control*, 62(2):532–544, 2017.

[57] Pradeep Gohil. Lecture: Primal-dual interior-point methods, 2017.

[58] Jacek Gondzio. Interior point methods 25 years later. *European Journal of Operational Research*, 218(3):587–601, 2012.

[59] Jacek Gondzio. Interior point methods in the year 2025. *EURO Journal on Computational Optimization*, 13:100105, 2025.

[60] Baptiste Goujaud, Céline Moucer, François Glineur, Julien M. Hendrickx, Adrien B. Taylor, and Aymeric Dieuleveut. Pepit: computer-assisted worst-case analyses of first-order optimization methods in python. *Mathematical Programming Computation*, 16(3):337–367, 2024.

[61] Nicholas I. M. Gould. On practical conditions for the existence and uniqueness of solutions to the general equality quadratic programming problem. *Mathematical Programming*, 1985.

[62] Philippe L. and Kočvara Michal Gould, Nicholas I. M.and Toint. *Numerical Methods for Large-Scale Non-Convex Quadratic Programming*, pages 149–179. Springer US, Boston, MA, 2002.

[63] L. Grippo, F. Lampariello, and S. Lucidi. A nonmonotone line search technique for newton's method. *SIAM Journal on Numerical Analysis*, 23(4):707–716, 1986.

[64] L. Gruene and J. Pannek. *Nonlinear Model Predictive Control, Theory and Algorithms, Second Edition*. 2017.

[65] B. I. Gursky, W. F. J. Olsman, and N. Peinecke. Development of a tunnel-in-the-sky display for helicopter noise abatement procedures. *CEAS Aeronautical Journal*, 5(2):199–208, Jun 2014.

[66] Setia Hadiwardoyo, Jose Ordonez-Lucena, and Christos Tranoris. Uav mobility model for dynamic uav-to-car communications. In *Proceedings of the 2019 Workshop on New Frontiers in IoT Security and Privacy*, 2019.

[67] Hao Su, Justin Solomon. Curves: Parametrization, curvature, frenet frame. https://geoml.github.io/Lectures/L3_Curves_I.pdf.

[68] Ben Hermans, Andreas Themelis, and Panagitis Patrinos. QPALM: a proximal augmented lagrangian method for nonconvex quadratic programs. *Mathematical Programming Computation*, 14(3):497–541, 2022.

[69] Michael Hurni, Pooya Sekhavat, and Michael I. Ross. Autonomous trajectory planning using real-time information updates, 08 2008.

[70] Jeffrey Ichnowski, Paras Jain, Bartolomeo Stellato, Goran Banjac, Michael Luo, Francesco Borrelli, Joseph E. Gonzalez, Ion Stoica, and Ken Goldberg. Accelerating quadratic optimization with reinforcement learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 21043–21055. Curran Associates, Inc., 2021.

[71] Fritz John. Extremum problems with inequalities as subsidiary conditions. *Studies and Essays, Courant Anniversary Volume*, pages 187–204, 1948.

[72] Dongmin Jong and Soohyun Kim. Never landing drone: Autonomous soaring of a unmanned aerial vehicle in front of a moving obstacle. *International Journal of Micro Air Vehicles*, 13, 2021.

[73] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, 1998.

[74] A. Kamara, M. Koroma, and M. M.-Ali. Improved conditions for the existence and uniqueness of solutions to the general equality constrained quadratic programming problem. *Open Journal of Optimization*, 2012.

[75] Narendra Karmarkar. A new polynomial-time algorithm for linear programming-ii. *Combinatorica*, 4:373–395, 12 1984.

[76] William Karush. Minima of functions of several variables with inequalities as side conditions, 1939.

[77] Arian Keipour and Dimitri Zarzhitsky. Visual servoing approach for autonomous uav landing on a moving vehicle. *arXiv preprint arXiv:2104.01272*, 2021.

[78] L.G. Khachiyan. A polynomial algorithm in linear programming. 1979.

[79] FUNABIKI Kohei. Design of tunnel-in-the-sky display and curved trajectory. In *24TH INTERNATIONAL CONGRESS OF THE AERONAUTICAL SCIENCES*.

[80] M. Kojima, Nimrod Megiddo, and Shinji Mizuno. A primal—dual infeasible-interior-point algorithm for linear programming. *Mathematical Programming*, 1993.

[81] M. Kojima, S. Mizuno, and A. Yoshise. *A primal-dual interior point algorithm for linear programming*, page 29–47. Springer-Verlag, Berlin, Heidelberg, 1988.

[82] H. W. Kuhn and A. W. Tucker. *Nonlinear Programming*. University of California Press, Berkeley, 1951.

[83] Swaenen Lense. Optimality conditions and numerical tolerances in qp solvers. 2021.

[84] Dong-hui Li and Masao Fukushima. A derivative-free line search and global convergence of broyden-like method for nonlinear equations. *Optimization Methods and Software*, 13, 04 1999.

[85] Ke Li and Jitendra Malik. Learning to optimize. In *International Conference on Learning Representations*, 2017.

[86] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3053–3062. PMLR, 2018.

[87] Dominic Liao-McPherson and Ilya Kolmanovsky. FBstab: A proximally stabilized semismooth algorithm for convex quadratic programming. *Automatica*, 113:108801, 2020.

[88] Björn Lindqvist, Sina Sharif Mansouri, Ali-akbar Agha-mohammadi, and George Nikolakopoulos. Nonlinear mpc for collision avoidance and control of uavs with dynamic obstacles. *IEEE Robotics and Automation Letters*, 5(4):6001–6008, 2020.

[89] Fernando Javier Luque. Asymptotic convergence analysis of the proximal point algorithm. *SIAM Journal on Control and Optimization*, 22(2):277–293, 1984.

[90] István Maros and Csaba Mészáros. A repository of convex quadratic programming problems. *Optimization Methods and Software*, 11(1-4):671–681, 1999.

[91] David Q Mayne, James B Rawlings, Christopher V Rao, and Peter O M Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.

[92] N. Megiddo. Pathways to the optimal set in linear programming. In *On Progress in Mathematical Programming: Interior-Point and Related Methods*, page 131–158, Berlin, Heidelberg, 1989. Springer-Verlag.

[93] Lorenz Meier. MAVLink: Micro air vehicle communication protocol. Header-only message serialization library; initial release 2009, 2009. Used extensively in PX4, ArduPilot, QGroundControl.

[94] S Meimei, L Huixia, H Jinwen, and et al. Path planning based on improved mpc for fixed wing uavs with collision avoidance. In *Advances in Guidance, Navigation and Control*, pages 2287–2296. Springer Singapore, 2022.

[95] Yu. E. Nesterov and M. J. Todd. Primal-dual interior-point methods for self-scaled cones. *SIAM Journal on Optimization*, 8(2):324–364, 1998.

[96] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer, 2. ed. edition, 2006.

[97] Brendan O'Donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, 2016.

[98] Takayuki Okuno and Masao Fukushima. An interior point sequential quadratic programming-type method for log-determinant semi-infinite programs. *Journal of Computational and Applied Mathematics*, 376:112784, 2020.

[99] Reiner Onken and Anton Walsdorf. Assistant systems for aircraft guidance: cognitive man-machine cooperation. *Aerospace Science and Technology*, 5(8):511–520, 2001.

[100] Johannes Ostler, Adrian Dudek, Jeremy Bertoncini, Martin Russ, and Peter Stütz. Missionlab: A next generation mission technology research platform based on a very light aircraft. In *AIAA SCITECH 2024 Forum*.

[101] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

[102] Cheng Peng, James Keller, and Vijay Kumar. Time-optimal uav trajectory planning for 3d urban structure coverage. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2750–2757, 2008.

[103] Spyridon Pougkakiotis and Jacek Gondzio. An interior point-proximal method of multipliers for convex quadratic programming. *Computational Optimization and Applications*, 78(2):307–351, 2021.

[104] Jan Procházka, Jan Jarkovský, Miroslav Jirgl, and Lukáš Polák. Model predictive control-based trajectory generation for agile landing of unmanned aerial vehicle on a moving boat. *Ocean Engineering*, 2024.

[105] PX4 Development Team and Community. Px4 autopilot. `https://px4.io`, 2012–`present`. Open-source, BSD-3-Clause licensed flight control software for drones and other unmanned vehicles.

[106] James B Rawlings, David Q Mayne, and Moritz Diehl. *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2nd edition, 2017.

[107] Ralph Tyrell Rockafellar. *Convex Analysis*. Princeton University Press, Princeton, 1970.

[108] Ralph Tyrrell Rockafellar. Augmented Lagrangians and applications of the proximal point algorithm in convex programming. *Mathematics of operations research*, 1(2):97–116, 1976.

[109] Ralph Tyrrell Rockafellar. Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, 14(5):877–898, 1976.

[110] Daniel Ruiz. A scaling algorithm to equilibrate both rows and columns norms in matrices. Technical Report TR/PA/2001/40, CERFACS, 2001.

[111] R. Sambharya, G. Hall, B. Amos, and B. Stellato. End-to-end learning to warm-start for real-time quadratic optimization. In N. Matni, M. Morari, and G. J. Pappas, editors, *Proceedings of the 5th Annual Learning for Dynamics and Control Conference*, volume 211 of *Proceedings of Machine Learning Research*, pages 220–234. PMLR, 6 2023.

[112] R. Sambharya, G. Hall, B. Amos, and B. Stellato. Learning to warm-start fixed-point optimization algorithms. *Journal of Machine Learning Research*, 25(166):1–46, 2024.

[113] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. 2017.

[114] Roland Schwan, Yuning Jiang, Daniel Kuhn, and Colin N. Jones. PIQP: A proximal interior-point quadratic programming solver. In *2023 62nd IEEE Conference on Decision and Control (CDC)*, pages 1088–1093, 2023.

[115] Daniel Simon. Fighter aircraft maneuver limiting using mpc, theory and application. In *PhD Dissertation*, volume ISBN 978-91-7685-450-1, 2017.

[116] M. Slater. Lagrange multipliers revisited: A contribution to nonlinear programming. November 1950.

[117] M. V. Solodov and B. F. Svaiter. A unified framework for some inexact proximal point algorithms *. *Numerical Functional Analysis and Optimization*, 22(7-8):1013–1035, 2001.

[118] Mikhail V. Solodov. *CONSTRAINT QUALIFICATIONS*. Instituto de Matematica Pura e Aplicada, Rio de Janeiro, Brazil.

[119] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. Osqp: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

[120] Boyd Stephen and Vandenberghe Lieven. *A MOOC on convex optimization*. Cambridge University Press, first edition, 2014.

[121] Liang Sun. *Dynamic Modeling, Trajectory Generation and Tracking for Towed Cable Systems*. PhD thesis, 12 2012.

[122] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[123] Mehling Tim Oliver. Augmented reality pilot assistance system for helicopter shipboard operations. In *Doctoral Thesis TUM*, 2022.

[124] F. A. Valentine. The problem of lagrange with differential inequalities as added side conditions. *Contrib. Calculus Variations 1933–1937*, pages 403–447, 1937.

[125] Robert Vanderbei. Symmetric quasidefinite matrices. *Siam Journal on Optimization - SIAM J OPTIMIZATION*, 5, 02 1995.

[126] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 3 2006.

[127] Kaixuan Wei, Angelica Aviles-Rivero, Jingwei Liang, Ying Fu, Hua Huang, and Carola-Bibiane Schönlieb. TFPnP: Tuning-free plug-and-play proximal algorithms with applications to inverse imaging problems. *Journal of Machine Learning Research*, 23(16):1–48, 2022.

[128] Stephen J. Wright. *Primal-dual interior-point methods*. Society for Industrial and Applied Mathematics, USA, 1997.

[129] Zhanghao Wu, Eric Liang, Michael Luo, Sven Mika, Joseph E. Gonzalez, and Ion Stoica. RLlib flow: Distributed reinforcement learning is a dataflow problem. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[130] Andreas Wächter and Lorenz Biegler. Line search filter methods for nonlinear programming: Motivation and global convergence. *SIAM Journal on Optimization*, 16:1–31, 01 2005.

[131] Xiaohui Xie. Lecture cs295: Convex optimization, 2025. Course materials.

[132] Yuxuan Zeng and Yujie Liu. Reinforcement learning based autonomous multi-rotor landing on moving platforms. *Autonomous Robots*, 2024.

[133] Hartmut Zohm. Magnetohydrodynamic stability of tokamaks. 2014.