Technical University of Munich

Institute of Flight System Dynamics

Evaluation of Solvers for Optimization-based Control Allocation

–

Evaluierung von Solvern für optimierungsbasierte Allokationsmethoden

Semester Thesis

Author: Vittor Braide Costa

Matriculation Number: 03783398

Supervisors: M.Sc. Simon Hafner

Examiner: Prof. Dr.-Ing. Florian Holzapfel

2025-04-10

Statutory Declaration

I, Vittor Braide Costa, declare on oath towards the Institute of Flight System Dynamics of Technical University of Munich, that I have prepared the present Semester Thesis independently and with the aid of nothing but the resources listed in the bibliography.

This thesis has neither as-is nor similarly been submitted to any other university.

Garching, 2025-04-10

[illegible]

Vittor Braide Costa

Abstract

Non-classical aircraft configurations, such as Vertical Take-Off Landing Vehicles (VTOLs), often rely on Control Allocation to fulfill pseudo-control commands using redundant control inputs. This project aims to evaluate freely available solvers for academic use that solve the Control Allocation problem using optimization-based approaches, specifically applying Active-Set and Interior-Point methods. Metrics such as elapsed time, allocation error, secondary objective error, and violations of C coding standards are captured and compared across the solvers.

To ensure reliable results, all solvers were implemented in Simulink and tested using Processor-in-the-loop (PIL) simulations on a hardware board. Different dataset sizes were used to test each solver, represented by a fighter aircraft, a multicopter, and a VTOL. The study's main finding is that solvers based on the Active-Set method offer a better balance between elapsed time and allocation error for the dataset dimensions applied. However, one of the Interior-Point solvers demonstrated strong robustness to increasing problem size and may be more suitable for large datasets.

Kurzfassung

Außergewöhnliche Flugzeugkonfigurationen, wie zum Beispiel Kampfflugzeuge oder Vertical Take-Off Landing Vehicle (VTOL), verlassen sich oft auf Control Allocation, um Pseudo Steuerungsbefehle mit redundanten Steuereingängen zu erfüllen. Dieses Projekt zielt darauf ab, frei verfügbare Solver für den akademischen Gebrauch zu evaluieren, die das Problem der Kontrollzuweisung mit Hilfe von optimierungsbasierten Ansätzen lösen, insbesondere unter Anwendung von Active-Set- und Interior-Point-Methoden. Metriken wie verstrichene Zeit, Zuweisungsfehler, sekundäre Zielfehler und Verstöße gegen C-Codierungsstandards werden erfasst und mit den verschiedenen Solvern verglichen.

Um zuverlässige Ergebnisse zu gewährleisten, wurden alle Solver in Simulink implementiert und mit Hilfe von PIL-Simulationen auf einer Hardwarekarte getestet. Zum Testen der einzelnen Solver wurden unterschiedliche Datensätze verwendet, die durch ein Kampfflugzeug, einen Multicopter und ein Vertical Take-Off Landing Vehicle (VTOL) repräsentiert wurden. Das Hauptergebnis der Studie ist, dass Löser, die auf der Active-Set-Methode basieren, ein besseres Gleichgewicht zwischen der verstrichenen Zeit und dem Zuweisungsfehler für die verwendeten Datensatzgrößen bieten. Einer der Interior-Point-Solver zeigte jedoch eine hohe Robustheit bei zunehmender Problemgröße und könnte für große Datensätze besser geeignet sein.

Table of Contents

List of Figures vii

List of Tables ix

List of Algorithms xi

Table of Acronyms xiii

Table of Symbols xv

1 Introduction 1
1.1 Objectives 2
1.2 Outline 3

2 State of the Art 5
2.1 Optimization Problems 5
2.2 Optimization Methods 6
2.2.1 Active-Set 6
2.2.2 Interior-Point 9
2.3 Control Allocation Framework 12

3 Solvers 15
3.1 Quadratic Programming Control Allocation Toolbox (QCAT) 15
3.1.1 Sequential Least-squares (SLS) 15
3.1.2 Weighted Least-squares (WLS) 17
3.2 Embedded Conic Solver (ECOS) 17
3.3 Code Generation for Convex Optimization (CVXGEN) 22

4 Implementation 25
4.1 Datasets 25
4.1.1 ADMIRE 25
4.1.2 12-copter 26
4.1.3 VTOL 26
4.2 Implementation in C and Testing 27

5 Results 31
5.1 FLOPs Counting 31
5.2 Elapsed Time and Errors 32
5.3 Code Compliance with C Standards 35

6 Conclusion and Outlook 39
6.1 Discussion of Results 39

6.2 Future Work 40

References I

List of Figures

1–1 F-15B from ACTIVE research project by NASA [5] 1

2–1 Simplified control system structure including control allocation 12
2–2 ADMIRE Dataset Attainable Moment Set 13

3–1 ECOS supported cones 18

4–1 ADMIRE model layout configuration [20] 25
4–2 Side View of the VTOL configuration [23] 26
4–3 ECOS PIL Block creation from Subsystem 27
4–4 Code Generation for Convex Optimization (CVXGEN) online interface and implementation in *Simulink* 28
4–5 Visual Comparison of Achievable and Attainable ν Values in ADMIRE 28

5–1 Estimated FLOPs for each Solver per Dataset 32
5–2 Comparison of Elapsed Time Across Solvers and Datasets 33
5–3 Comparison of Solvers Elapsed Time under different AMS Locations 33
5–4 Comparison of Solvers Allocation Error under different AMS Locations 34
5–5 Comparison of Normalized Allocation Errors Across Solvers and Datasets 34
5–6 Comparison of Solvers Secondary Objective Error under different AMS Locations . 35
5–7 Code Guideline Violations According to SEI CERT C and MISRA C:2012 36
5–8 Top 10 Most Frequently Violated Coding Standards in WLS 37
5–9 Top 10 Most Frequently Violated Coding Standards in SLS 37
5–10 Top 10 Most Frequently Violated Coding Standards in ECOS 38
5–11 Top 10 Most Frequently Violated Coding Standards in CVXGEN 38

List of Tables

4–1 ADMIRE Effectors Deflection Limits 26
4–2 Solvers Parameters 29

5–1 Floating Point Operations (FLOPs) for Common Operations 31

List of Algorithms

2–1 Active-Set pseudo-code [2] 8
2–2 Interior-Point pseudo-code 11
3–1 Sequential Least-squares [2] 16
3–2 Weighted Least-squares [2] 17
3–3 ECOS pseudo-code 21
3–4 CVXGEN pseudo-code 23

Table of Acronyms

Acronym	**Description**
ADMIRE	Aero-Data Model In a Research Environment
AMS	Attainable Moment Set
CVXGEN	Code Generation for Convex Optimization
ECOS	Embedded Conic Solver
FCS	Flight Control System
FLOPs	Floating Point Operations
INDI	Incremental Nonlinear Dynamic Inversion
IP	Interior-Point
KKT	Karush–Kuhn–Tucker
LP	Linear Program
PIL	Processor-in-the-loop
QCAT	Quadratic Programming Control Allocation Toolbox
QCQP	Quadratically Constrained Quadratic Program
QP	Quadratic Program
RPI	Redistributed Pseudo Inverse
SLS	Sequential Least-squares
SOCP	Second-order Cone Program
VTOL	Vertical Take-Off Landing Vehicle
WLS	Weighted Least-squares

Table of Symbols

Mathematical Notation - in the text, vector quantities are represented by bold lowercase letters $\boldsymbol{u}$, while matrices are represented in bold uppercase $\boldsymbol{B}$. Scalar quantities are represented in italics: α.

Symbol	**Description**
$\boldsymbol{B}$	control effectiveness matrix
$\boldsymbol{W}$	weighting matrices
$\boldsymbol{u}$	control input vector
$\boldsymbol{\nu}$	pseudo control vector
$\underline{\boldsymbol{u}}$	lower bound of effectors positions
$\overline{\boldsymbol{u}}$	upper bound of effectors positions
$\boldsymbol{u_d}$	desired effectors positions
$\boldsymbol{\mathcal{W}}$	working-set
$\boldsymbol{f}, \boldsymbol{g}, \boldsymbol{l}$	generic functions
$\boldsymbol{p}$	step direction
α	step length
δ	convergence tolerance
γ	weighting factor

1 Introduction

Modern fighter or VTOLs (Vertical Take-off and Landing) aircraft are composed of nonclassical configurations, and there are usually more control inputs than controlled variables, commonly called virtual controls. The redundancy offered by multiple effectors enhances maneuverability and fault tolerance, and therefore, much emphasis has been placed on the overactuated configuration [1]. As a consequence of having to distribute the control effort among effectors, control allocation methods are applied to find an optimal solution for the underdetermined and constrained equation system [2].

Concurrently, the previously mentioned aircraft offer a challenging control problem, as the dynamics are nonlinear, the aerodynamics are uncertain, and the effectors are constrained by position and rate limits. Despite all these challenges, the operation should guarantee maximum maneuverability and robustness and operate under different conditions. Thus, methods used should yearn for the full capabilities of the effectors suite.

The F-15 Advanced Controls Technology for Integrated Vehicles (ACTIVE) illustrates the problem. With the main goal to study flight control capabilities applying pitch and yaw thrust vectoring [3], the F-15 ACTIVE has control surfaces in the canards, horizontal tails, vertical stabilizers, wings (ailerons and flaps), and of course, the nozzle deflections from its axisymmetric thrust vectoring feature. Therefore, control redundancy is strongly present, and in low-speed flight, a clever control allocation should account for the effector's effectiveness loss [4].

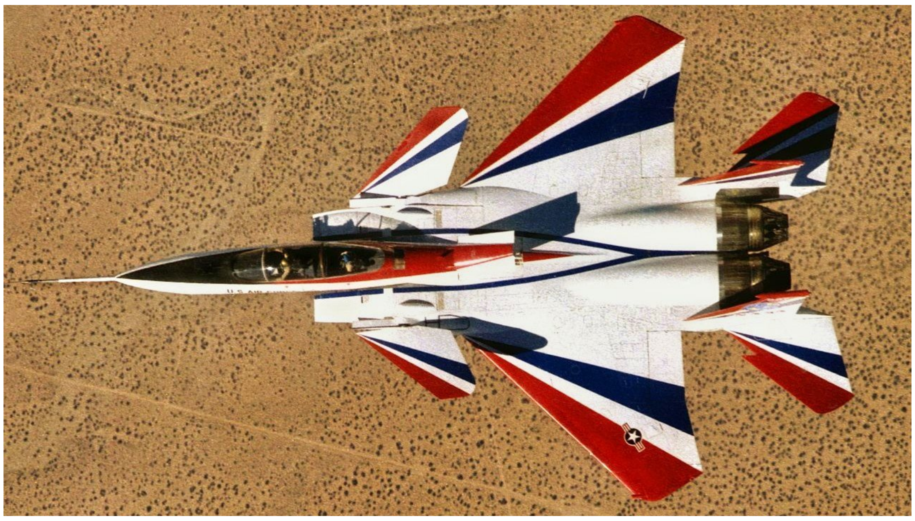


Figure 1–1: F-15B from ACTIVE research project by NASA [5]

Multiple solutions for this intricate problem are proposed as stated in Johansen et al. [6]. However, much emphasis has been given to optimization-based methods, as they usually outperform the alternatives, and since their computational complexity is well within the limits of today's embedded systems [6]. Therefore, the optimization-based design provides a powerful approach to handle the challenges mentioned above and can be realistically considered for real-time control allocation [7].

1.1 Objectives

Building on the motivation, this study advances Optimization-based control allocation evaluations to identify potential improvements. The existing literature already benchmarks some methods, as done by Bodson [7]. Still, the investigation usually focuses on a more high-level proposal, not specifically on Optimization-based control allocation, which may apply different solvers and have different implementations.

Some other limitations are that assessments usually take place in desktop applications, which do not provide a completely noise-free environment and may raise reliability concerns about the results [8]. Additionally, studies often do not evaluate how different implementations perform across diverse datasets. Lastly, in aircraft applications, software guidelines are crucial for building reliable and safe products. However, optimization benchmarking often neglects them.

To further explore these aspects, this study aims to implement different optimization solvers, each addressing distinct optimization problems and utilizing different optimization methods. These implementations are then evaluated based on Floating Point Operations (FLOPs) counting, elapsed time measurement, and allocation and secondary objective errors computation.

Additionally, certifiability is assessed using *Polyspace* tools to evaluate robustness, code coverage, and compliance with coding standards. The testing is conducted through processor-in-the-loop (PIL) simulations in Simulink, utilizing an STM32F767 board to minimize potential disturbances from background desktop tasks.

In light of this, this study's objectives can be summarized in the following questions:

1. Whether any solver is best-fitted for Optimization-based control allocation. Here, "best-fitted" means: (i) if the effectors can generate the commanded pseudo-controls, the commanded vector will be accurately produced; (ii) upon realizing commanded pseudo-controls, the secondary objective related to control effort minimization can be achieved; (iii) if actuator saturation prevents the effectors from generating the commanded pseudo-controls, the allocation scheme minimizes control error by maximizing control power; and (iv) the solver can achieve low elapsed time for real-time applications.
2. Whether the solver behaves similarly when changing dataset sizes and control allocation designs
3. Whether the embedded code is compliant with software standards, rules, and recommendations.

1.2 Outline

This thesis is structured as follows: Chapter 2 introduces the theoretical tools necessary to understand the classifications of the optimization problem and the methods used to solve it. Chapter 3 details the working principle of each solver and how they differ. Afterward, Chapter 4 presents the simulation setup, providing an overview of the datasets and detailing the solvers implementations. Chapter 5 outlines the results by showcasing the FLOPs counting, the elapsed time, and the allocation and secondary objective error. Lastly, Chapter 6 reviews the key findings from the evaluation and explores opportunities for further investigation.

2 State of the Art

To contextualize and analyze the topics relevant to this study, it is first necessary to discuss what constitutes an optimization problem. Subsequently, methods for solving these problems are introduced and linked to the solvers selected for evaluation. Finally, the application of optimization to control allocation in overactuated systems is explored.

2.1 Optimization Problems

According to Boyd et al. [9], a mathematical optimization problem has the form represented in Eq. (2–1), where $f_0(x) : \mathbb{R}^n \rightarrow \mathbb{R}$ is called the objective function, and the functions $f_i(x) : \mathbb{R}^n \rightarrow \mathbb{R}$ the constraint functions.

$$\begin{aligned} \min \quad & f_0(\boldsymbol{x}) \\ \text{s.t.} \quad & f_i(\boldsymbol{x}) \leq b_i,\ i = 1, \dots, m \end{aligned} \tag{2–1}$$

The generic problem can be divided into families, characterized by the objective and constraint functions forms [9]. If f_0 and f_i are convex (i.e., Equation (2–2) is satisfied), the problem is a convex optimization problem.

$$f_i(\alpha\boldsymbol{x} + \beta\boldsymbol{y}) \leq \alpha f_i(\boldsymbol{x}) + \beta f_i(\boldsymbol{y}) \tag{2–2}$$

Similarly, constrained least-squares is another optimization problem, defined in Eq. (2–3) [9]. Least-squares can be combined in different manners to solve broader problems.

$$\begin{aligned} \min \quad & f_0(\boldsymbol{x}) = \|\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}\|_2^2 \\ \text{s.t.} \quad & \underline{x}_i \leq x_i \leq \overline{x}_i,\ i = 1, \dots, n \end{aligned} \tag{2–3}$$

Quadratic Program (QP) is a generalization of least-squares, where the objective function is quadratic, and constraints are linear. This is the problem target from CVXGEN solver and its details are shown in the Equation (2–4) [9].

$$\begin{aligned} \min \quad & f_0(x) = \tfrac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{q}^\top \boldsymbol{x} + r \\ \text{s.t.} \quad & \boldsymbol{G}\boldsymbol{x} \preceq \boldsymbol{h}, \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}. \end{aligned} \tag{2–4}$$

One can see that the least-squares problem can be easily converted to QP (see Eq. (2–5)). Besides CVXGEN, this approach is also used in the Quadratic Programming Control Allocation Toolbox (QCAT) [2], which includes solvers based on weighted least-squares and sequential least-squares methods. Their working principles will be further elaborated in Chapter 3.

$$\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2^2 = (\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})^\top(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}) = \underbrace{\boldsymbol{x}^\top\boldsymbol{A}^\top\boldsymbol{A}\boldsymbol{x}}_{\boldsymbol{Q}=2\boldsymbol{A}^\top\boldsymbol{A}} - \underbrace{2\boldsymbol{b}^\top\boldsymbol{A}\boldsymbol{x}}_{\boldsymbol{q}=-2\boldsymbol{A}^\top\boldsymbol{b}} + \underbrace{\boldsymbol{b}^\top\boldsymbol{b}}_{r=\boldsymbol{b}^\top\boldsymbol{b}} \,. \tag{2–5}$$

Another optimization problem is the Second-order Cone Program (SOCP), defined in Eq. (2–6) [9]. It is used in the Embedded Conic Solver (ECOS) [10]. Notably, SOCP can be reduced to other problems, such as Quadratically Constrained Quadratic Program (QCQP) and Linear Program (LP), providing more flexibility in the problem definition when compared to QP. The conversion between QP and SOCP is further detailed in Section 3.2.

$$\begin{aligned} \min \quad & \boldsymbol{f}^\top\boldsymbol{x} \\ \text{s.t.} \quad & \|\boldsymbol{A}_i x + \boldsymbol{b}_i\|_2 \leq \boldsymbol{c}_i^\top x + d_i,\ i = 1,\ldots,m \\ & \boldsymbol{F}\boldsymbol{x} = \boldsymbol{g} \end{aligned} \tag{2–6}$$

2.2 Optimization Methods

As highlighted by Wong [11], QP optimization methods are typically grouped into Active-Set and Interior-Point (IP) methods. The following sections elaborate on each approach and their distinguishing features.

2.2.1 Active-Set

The Active-Set method solves an equality-constrained least squares problem (see Eq. (2–3)). The process is usually divided into two phases: the first focuses on feasibility, and the second focuses on optimality. Hence, every point $\boldsymbol{u}^{(k)}$ should have a so-called 'working-set' $\mathcal{W}^{(k)}$ that represents a subset of the active constraints.

Considering the mixed-constrained least squares problem from Eq. (2–8), the process of finding an optimal solution starts with an arbitrary feasible point $\boldsymbol{u}^{(0)}$ and its working-set $\boldsymbol{\mathcal{W}}^{(0)}$. Thereafter, the method computes a sequence of feasible iterates $\boldsymbol{u}^{(k)}$ such that $\boldsymbol{u}^{(k+1)} = \boldsymbol{u}^{(k)} + \alpha_k \boldsymbol{p}_k$ [11], where $\boldsymbol{p}_k$ is the search direction and α_k the step length. In this formulation, the constraint matrix and the variable upper $\overline{\boldsymbol{u}}$ and lower $\underline{\boldsymbol{u}}$ bounds can be defined as $\boldsymbol{C} = \begin{bmatrix}\boldsymbol{I}_m & -\boldsymbol{I}_m\end{bmatrix}^\top$ and $\boldsymbol{u}_{lim} = \begin{bmatrix}\underline{\boldsymbol{u}} & -\overline{\boldsymbol{u}}\end{bmatrix}^\top$.

$$\begin{aligned} \min_{u} \quad & f(\boldsymbol{u}) = \|\boldsymbol{A}\boldsymbol{u} - \boldsymbol{b}\| \\ \text{s.t.} \quad & \boldsymbol{B}\boldsymbol{u} = \boldsymbol{\nu} \\ & \boldsymbol{C}\boldsymbol{u} \geq \boldsymbol{u}_{lim} \end{aligned} \tag{2–7}$$

To do so, the process should proceed through the following steps: Solving for the direction $\boldsymbol{p}_k$, computing the step length α_k, and deleting and adding constraints. The transition between them relies on some conditions that will be explained later in this section.

Finding the direction $\boldsymbol{p_k}$ can be obviously represented by the following optimization problem, as it guarantees that the search direction respects the active constraints while minimizing the objective function.

$$\begin{aligned} \min_{p} \quad & f(\boldsymbol{u}^{(k)} + \boldsymbol{p}) = \|\boldsymbol{A}(\boldsymbol{u}^{(k)} + \boldsymbol{p}) - \boldsymbol{b}\| \\ \text{s.t.} \quad & \boldsymbol{B}\boldsymbol{p} = \boldsymbol{0} \end{aligned} \tag{2–8}$$

In this study, the solver that applies the Active-Set method calculates $\boldsymbol{p}_k$ directly by $\min_p \|\boldsymbol{A}\boldsymbol{p} - \boldsymbol{d}\|$, with $\boldsymbol{d} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{u}^{(k)}$ and guarantees that $\boldsymbol{u}^{(k)}$ remains in the feasible subspace by determining α_k. This unconstrained optimization problem is directly solved by the linear system $\boldsymbol{d} = \boldsymbol{A}\boldsymbol{p}$. Hence, after computing $\boldsymbol{p}_k$, the method checks the feasibility and optimality of $\boldsymbol{u}^{(k+1)} = \boldsymbol{u}^{(k)} + \boldsymbol{p}_k$.

The optimality is verified if $\boldsymbol{u}^{k+1}$ is feasible. As stated by Wong [11], this is achieved if the Lagrange multiplier associated with the inequality constraint (denoted here as $\boldsymbol{\lambda}$) is greater or equal to zero. To compute it, a Karush–Kuhn–Tucker (KKT) system as depicted in Eq. (2–9) is solved.

$$\boldsymbol{A}^\top(\boldsymbol{A}\boldsymbol{u} - \boldsymbol{b}) = \begin{pmatrix} \boldsymbol{B}^\top & \boldsymbol{C}_0^\top \end{pmatrix} \begin{pmatrix} \boldsymbol{\mu} \\ \boldsymbol{\lambda} \end{pmatrix} \tag{2–9}$$

In the case of $\boldsymbol{u}^{(k+1)}$ not being the optimum solution (i.e. any $\lambda_i < 0$ for $i = 1, \ldots, 2m$), the constraint associated with the most negative element from $\boldsymbol{\lambda}$ shall be removed from $\boldsymbol{\mathcal{W}}^{(k+1)}$. The Algorithm 2–1 presents the key steps of the Active-Set method, as previously discussed.

As noted, Active-Set needs an initialization defined by $\boldsymbol{u}^{(0)}$ and $\boldsymbol{\mathcal{W}}^{(0)}$, and, therefore, it is particularly effective when warm starts are available. This makes it well-suited for sequential optimization problems, where solving a series of related quadratic programs is required [11].

Algorithm 2–1 Active-Set pseudo-code [2]

```
Let $\boldsymbol{u}^{(0)}$ be a feasible starting point.
Let the working set $\boldsymbol{\mathcal{W}}$ contain a (subset of) the active inequality constraints at $\boldsymbol{u}^{(0)}$.
for $k = 0, 1, 2, \ldots, k_{max} - 1$ do
    Given $\boldsymbol{u}^{(k)}$, find the optimal perturbation $\boldsymbol{p}$, considering the constraints in the working set
    as equality constraints and disregarding the remaining inequality constraints.
    Solve:
        $$\begin{aligned} \min_{p} \quad & \|\boldsymbol{A}(\boldsymbol{u}^{(k)} + \boldsymbol{p}_k) - \boldsymbol{b}\| \\ \text{s.t.} \quad & \boldsymbol{B}\boldsymbol{p}_k = 0 \\ & \boldsymbol{p}_{k,i} = 0, \quad i \in \boldsymbol{\mathcal{W}} \end{aligned} \tag{2–10}$$
    if $\underline{\boldsymbol{u}} \leq \boldsymbol{u}^{(k)} + \boldsymbol{p}_k \leq \overline{\boldsymbol{u}}$ then
        Set $\boldsymbol{u}^{(k+1)} = \boldsymbol{u}^{(k)} + \boldsymbol{p}_k$ and compute the Lagrange multiplier $\boldsymbol{\lambda}$, where $\boldsymbol{\lambda}$ is associate with
        the second constraints from 2–8 respectively.
        if all $\boldsymbol{\lambda}_i \geq 0$ then
            $\boldsymbol{u}^{(k+1)}$ is the optimal solution to 2–8.
            break
        else
            Remove the constraint associated with the most negative $\boldsymbol{\lambda}$ from the working set.
        end if
    else
        Determine the maximum step length $\alpha_k$ such that $\boldsymbol{u}^{(k+1)} = \boldsymbol{u}^{(k)} + \alpha_k\boldsymbol{p}_k$ is feasible.
        Add the primary bounding constraint to the working set.
    end if
end for
```

2.2.2 Interior-Point

The main difference between the Interior-Point and the Active-Set is that IP compute iterates inside the problem feasible region rather than on the feasibility boundary [11]. Nevertheless, finding a solution is quite similar, as starting with a feasible point $\boldsymbol{u}^{(0)}$, the search or update direction shall be computed. Later, the step length and the iterate $\boldsymbol{u}^{(k+1)}$ are found, and lastly, the optimality is checked.

The same problem presented for Active-Set is used to describe the method in detail. However, this time, the objective function $f(\boldsymbol{u})$ and the inequality constraint function $g(\boldsymbol{u})$ are written in a generic form (see Eq. (2–12)) to shorten the following equations.

$$\begin{aligned} \min_{u} \quad & f(\boldsymbol{u}) = \|\boldsymbol{A}\boldsymbol{u} - \boldsymbol{b}\| \\ \text{s.t.} \quad & \boldsymbol{B}\boldsymbol{u} = \boldsymbol{\nu} \\ & \boldsymbol{g}(\boldsymbol{u}) = \boldsymbol{C}\boldsymbol{u} \geq \boldsymbol{u}_{lim} \end{aligned} \tag{2–11}$$

As mentioned, the IP method's iterations are computed inside the feasibility region. Hence, and in contrast with Active-Set, IP does not commit to a specific set of active constraints. Therefore, a constant monitoring of all KKT conditions and adjustments in the primal variable $\boldsymbol{u}$ and all associated Lagrange multipliers $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are needed.

Interior-point methods can be classified into Barrier methods and Primal-Dual methods [9]. In this study, only the Primal-Dual approach is applied, as it is used by the chosen solvers. The Lagrangian and the KKT conditions, as stated by Singh et al. [12], are depicted below:

$$L(\boldsymbol{u}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{u}) + \boldsymbol{\lambda}^\top \boldsymbol{g}(\boldsymbol{u}) + \boldsymbol{\mu}^\top (\boldsymbol{B}\boldsymbol{u} - \boldsymbol{\nu}) \tag{2–12}$$

$$\begin{aligned} \nabla f(\boldsymbol{u}) + \nabla \boldsymbol{g}(\boldsymbol{u})\boldsymbol{\lambda} + \boldsymbol{B}^\top \boldsymbol{\mu} &= \boldsymbol{0} \\ \boldsymbol{\Lambda}\boldsymbol{g}(\boldsymbol{u}) &= \boldsymbol{0}, \quad \boldsymbol{\Lambda} = \operatorname{diag}(\boldsymbol{\lambda}) \\ \boldsymbol{B}\boldsymbol{u} &= \boldsymbol{\nu} \\ \boldsymbol{\lambda}, -\boldsymbol{g}(\boldsymbol{u}) &\geq \boldsymbol{0}. \end{aligned} \tag{2–13}$$

The so-called Newton step is obtained by rewriting and linearizing the KKT conditions. Equation (2–14) illustrates it and represents the step related to finding the primal-dual search direction.

$$\begin{bmatrix} \nabla^2 f(\boldsymbol{u}) + \sum_{j=1}^{r} \boldsymbol{\lambda}_j \nabla^2 \boldsymbol{g}_j(\boldsymbol{u}) & \nabla \boldsymbol{g}(\boldsymbol{u})^\top & \boldsymbol{B}^\top \\ \boldsymbol{\Lambda} \nabla \boldsymbol{g}(\boldsymbol{u}) & \operatorname{diag}(\boldsymbol{g}(\boldsymbol{u})) & \boldsymbol{0} \\ \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{u} \\ \Delta \boldsymbol{\lambda} \\ \Delta \boldsymbol{\mu} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{r}_{dual} \\ \boldsymbol{r}_{cent} \\ \boldsymbol{r}_{prim} \end{bmatrix} \tag{2–14}$$

After computing the direction, the step length α_k should be found, and the current point $\boldsymbol{y} = (\boldsymbol{u}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ should be updated. To determine α_k, as outlined by Boyd et al. [9], the largest positive step length that ensures the dual variables remain non-negative must be computed. This is achieved by selecting α_k such that $\boldsymbol{\lambda} + \alpha \Delta \boldsymbol{\lambda} \succeq \boldsymbol{0}$ leading to the expression:

$$\alpha_{\max} = \min \left\{ 1, \min \left(-\frac{\boldsymbol{u}_i}{\Delta \boldsymbol{u}_i} : \Delta \boldsymbol{u}_i < 0 \right) \right\}. \tag{2–15}$$

With parameters $\gamma, \beta \in (0,1)$, the step length is initialized as $\alpha = 0,99\alpha_{\max}$ and a backtracking strategy is applied, where α is updated as $\alpha = \beta\alpha$ until all constraints are satisfied, i.e., $g_j(\boldsymbol{u}^{(k+1)}) \geq \boldsymbol{u}_{lim,j}, \quad j = 1, \ldots, 2m$. Additionally, α continues to be reduced as $\alpha = \beta\alpha$ until the residual norm satisfies:

$$\begin{aligned} &\|\boldsymbol{r}(\boldsymbol{u}^{(k+1)}, \boldsymbol{\lambda}^{(k+1)}, \boldsymbol{\mu}^{(k+1)})\| \leq (1 - \gamma\alpha) \|\boldsymbol{r}(\boldsymbol{u}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\mu}^{(k)})\| \\ &\text{with} \quad \boldsymbol{u}^{(k+1)} = \boldsymbol{u}^{(k)} + \alpha \Delta \boldsymbol{u}, \quad \boldsymbol{\lambda}^{(k+1)} = \boldsymbol{\lambda}^{(k)} + \alpha \Delta \boldsymbol{\lambda}, \quad \boldsymbol{\mu}^{(k+1)} = \boldsymbol{\mu}^{(k)} + \alpha \Delta \boldsymbol{\mu}. \end{aligned} \tag{2–16}$$

This process is repeated until convergence or until the maximum number of iterations is reached. The convergence criterion is stated in Eq. (2–17). A summary of the whole method can be visualized in the Algorithm 2–2.

$$\begin{aligned} &\eta(\boldsymbol{u}, \boldsymbol{\lambda}) = -\boldsymbol{g}(\boldsymbol{u})^\top \boldsymbol{\lambda} \\ &\eta^{(k)} \leq \delta \quad \text{and} \quad \left(\|\boldsymbol{r}_{prim}\|_2^2 + \|\boldsymbol{r}_{dual}\|_2^2 \right)^{1/2} \leq \delta. \end{aligned} \tag{2–17}$$

The Interior-Point method typically requires fewer iterations compared to the Active-Set method. However, each iteration tends to be more computationally expensive, as it involves solving linear systems that include all variables in the problem (i.e., primal and dual variables). On the other hand, Active-Set only considers a subset of the variables.

Moreover, a key advantage of incorporating all variables in the system is that the equations' dimensionality and the involved matrix's sparsity pattern remain constant throughout the process [11], facilitating an implementation without dynamic memory allocation.

Algorithm 2–2 Interior-Point pseudo-code

```
Let $\boldsymbol{u}^{(0)}$ be a feasible starting point $\boldsymbol{\lambda}^{(0)} \succ 0, \boldsymbol{\mu}^{(0)}$
Let $\eta(\boldsymbol{u}^{(0)}, \boldsymbol{\lambda}^{(0)}) = -\boldsymbol{g}(\boldsymbol{u}^{(0)})^\top \boldsymbol{\lambda}$
Let the parameters $\gamma, \beta \in (0,1)$
Let $\delta$ be the convergence tolerance
for $k = 0, 1, 2, \dots, k_{max} - 1$ do
    Given $u^{(k)}$, find the optimal search direction $\Delta y = [\Delta u \ \Delta\lambda \ \Delta\mu]^\top$:
        $$\begin{bmatrix} \nabla^2 f(\boldsymbol{u}) + \sum_{j=1}^{r} \boldsymbol{\lambda}_j \nabla^2 \boldsymbol{g}_j(\boldsymbol{u}) & \nabla \boldsymbol{g}(\boldsymbol{u})^\top & \boldsymbol{B}^\top \\ \boldsymbol{\Lambda} \nabla \boldsymbol{g}(\boldsymbol{u}) & \operatorname{diag}(\boldsymbol{g}(\boldsymbol{u})) & \mathbf{0} \\ \boldsymbol{A} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{u} \\ \Delta \boldsymbol{\lambda} \\ \Delta \boldsymbol{\mu} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{r}_{dual} \\ \boldsymbol{r}_{cent} \\ \boldsymbol{r}_{prim} \end{bmatrix} \quad (2–18)$$
    Find the step length $\alpha_k$:
        $$\alpha_{\max} = \min \left\{ 1, \min \left( -\frac{\boldsymbol{u}_i}{\Delta \boldsymbol{u}_i} : \Delta \boldsymbol{u}_i < 0 \right) \right\}, \quad \alpha = 0.99\alpha_{max} \quad (2–19)$$
    while $\|\boldsymbol{r}(\boldsymbol{u}^{(k+1)}, \boldsymbol{\lambda}^{(k+1)}, \boldsymbol{\mu}^{(k+1)})\| > (1 - \gamma\alpha)\|\boldsymbol{r}(\boldsymbol{u}^{(k)}, \boldsymbol{\lambda}^{(k)}, \boldsymbol{\mu}^{(k)})\|$ do
        $$\alpha \leftarrow \beta\alpha \quad (2–20)$$
    end while
    Compute the new point $\boldsymbol{y}^{(k+1)} = \boldsymbol{y}^{(k)} + \alpha_k \Delta \boldsymbol{y}$
    if $\eta^{(k+1)} \leq \delta$ and $\left(\|\boldsymbol{r}_{prim}\|_2^2 + \|\boldsymbol{r}_{dual}\|_2^2\right)^{1/2} \leq \delta$ then
        break
    end if
end for
```

2.3 Control Allocation Framework

The main goal of this section is to correlate the above-presented mathematical problems with the flight dynamics of aircraft with redundant control effectors. The dynamics model, as described in Johansen et al. [6], is presented below:

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{f}(\boldsymbol{x}, t) + \boldsymbol{g}(\boldsymbol{x}, t)\boldsymbol{\nu} \\ \boldsymbol{y} &= \boldsymbol{l}(\boldsymbol{x}, t) \end{aligned} \tag{2–21}$$

Where $\boldsymbol{f}$, $\boldsymbol{g}$, and $\boldsymbol{l}$ are functions, $\boldsymbol{x} \in \mathbb{R}^n$ is the state vector, and $\boldsymbol{\nu} \in \mathbb{R}^p$ the virtual control vector, which is usually formed by desired forces and moments. The control inputs are represented by $\boldsymbol{u} \in \mathbb{R}^m$, and therefore, the Control Allocation task is to compute actuator commands $\boldsymbol{u}$ which provide an overall control effort $\boldsymbol{\nu}$ [2]. The Fig. 2–1 shows the closed-loop system when control allocation is applied.

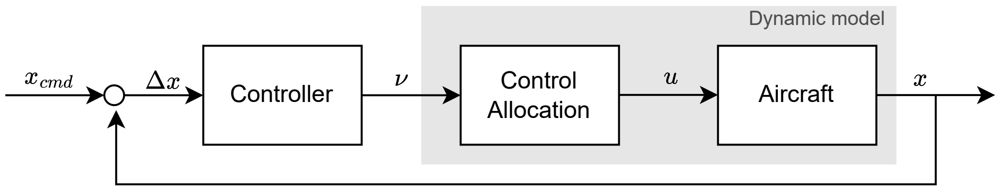


Figure 2–1: Simplified control system structure including control allocation

The correlation between $\boldsymbol{u}$ and $\boldsymbol{\nu}$ is defined by design, and it is usually represented by a static effector model: $\boldsymbol{\nu} = \boldsymbol{h}(\boldsymbol{u}, \boldsymbol{x}, t)$ [6]. However, the effectors model may be linear in $\boldsymbol{u}$, such that $\boldsymbol{\nu} = \boldsymbol{B}(\boldsymbol{x}, t)\boldsymbol{u}$ or shortly $\boldsymbol{\nu} = \boldsymbol{B}\boldsymbol{u}$.

Moreover, the inputs $\boldsymbol{u}$ are susceptible to saturation, meaning they are constrained by limit vectors: $\underline{\boldsymbol{u}} \leq \boldsymbol{u} \leq \overline{\boldsymbol{u}}$. Hence, if a control vector $\boldsymbol{u}$ exists for a particular pseudo-control $\boldsymbol{\nu}$, $\boldsymbol{\nu}$ is said to be attainable, and the set of attainable pseudo-controls is called Attainable Moment Set (AMS) [4]. Fig. 2–2 exemplifies the AMS from one of the datasets used in this study.

To compute $\boldsymbol{u}$, Durham et al. [4] and Oppenheimer et al. [1] present multiple methods. Direct Allocation, Daisy-chaining, Redistributed Pseudo Inverse (RPI), and Optimization-based control allocation are briefly discussed.

The Direct Allocation method is based on the scaling of the unconstrained optimization problem and can be quite complex to solve when the dimension of $\boldsymbol{u}$ is large [13]. Daisy-chaining is a method that assumes a hierarchy of control effectors [1]. The main idea is to utilize another control command to eliminate errors when one control or a group of controls saturates. While this approach is useful for improving performance by prioritizing one effector when another saturates (e.g., spoilers and ailerons for roll control), it does not guarantee an optimal solution

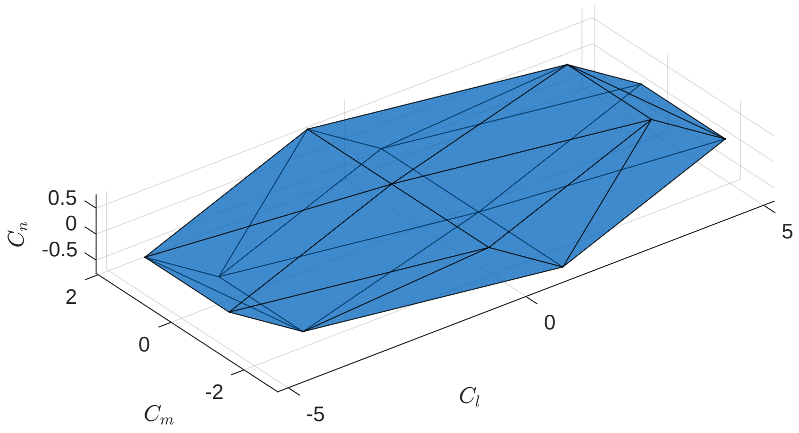


Figure 2–2: ADMIRE Dataset Attainable Moment Set

with respect to certain criteria [14].

As the name suggests, Redistributed Pseudo Inverse (RPI) works similarly to a simple pseudo-inverse algorithm. However, when an effector saturates, it is removed online from the subsequent pseudo-inverse solution [1]. Besides being a simple method, previous studies, as Bodson [7], have demonstrated that RPI can result in sub-optimal control allocation.

Lastly, the method here referred as Optimization-based control allocation, utilizes two different least-squares to allocate control commands [2] (see Eq. (2–22)). $\boldsymbol{W_u}$ and $\boldsymbol{W_v}$ are weighting matrices and $\boldsymbol{u_d}$ is refereed as desired control vector. As highlighted by Oppenheimer et al. [1], the main idea is to maximize the use of available degrees of freedom and, when adequate control power is available, achieve secondary objectives (i.e., control effort minimization).

$$\begin{aligned} \boldsymbol{u_S} &= \arg\min\nolimits_{\boldsymbol{u}\in\mathcal{K}} \|\boldsymbol{W_u}(\boldsymbol{u}-\boldsymbol{u_d})\| \\ \mathcal{K} &= \arg\min\nolimits_{\underline{\boldsymbol{u}}\leq\boldsymbol{u}\leq\overline{\boldsymbol{u}}} \|\boldsymbol{W_v}(\boldsymbol{B}\boldsymbol{u}-\boldsymbol{\nu})\| \end{aligned} \tag{2–22}$$

This problem can be directly solved using the methods presented in Section 2.2. However, there are still some claims about using optimization-based methods, especially related to online use in aircraft. However, as captured by Bodson [7], its use can observe noticeable performance gains, and computational times are mostly lower than a millisecond.

Therefore, based on the analysis of the aforementioned general methods and recognizing that the performance of the optimization-based approach is highly dependent on its numerical implementation (e.g., different solvers applied), this study aims to evaluate various implementations to benchmark control allocation solutions.

3 Solvers

As noted in Chapter 2, multiple solutions and different implementations are proposed for optimization problems. Solvers that use Active-Set or Interior-Point methods were applied to build a broad evaluation and consider different parameters in the compared solutions. A brief description of their implementation is provided in the sections below.

3.1 Quadratic Programming Control Allocation Toolbox (QCAT)

The QCAT implementation was done by Härkegård [2] and made freely available in *Matlab File Exchange*. The created toolbox has six implemented solvers, but this study uses only the Sequential Least-squares (SLS) and Weighted Least-squares (WLS) implementations. Both of them use the Active-Set described in Section 2.2.1.

3.1.1 Sequential Least-squares (SLS)

As illustrated in Eq. (2–22), the control allocation goals can be normally summarized in two constrained least-squares problems. The SLS algorithm solves the problem in two phases [2]. Firstly, it finds a feasible solution for $\boldsymbol{B}\boldsymbol{u} = \boldsymbol{\nu}$ using Active-Set and then uses the solution as a starting point for phase 2, which also uses Active-Set. The process is detailed in the Algorithm 3–1.

Algorithm 3–1 Sequential Least-squares [2]

```
Phase 1:
Let $\boldsymbol{u}^{(0)}$ and $\boldsymbol{\mathcal{W}}^{(0)}$ be the resulting solution and working set from the previous sampling instant.
Solve using Algorithm 2–1:
    $$\boldsymbol{u_1} = \arg\min_{\boldsymbol{u}} \|\boldsymbol{W}_v(\boldsymbol{B}\boldsymbol{u} - \boldsymbol{\nu})\| \qquad (3–1)$$
    $$\underline{\boldsymbol{u}} \leq \boldsymbol{u} \leq \overline{\boldsymbol{u}}$$
if $\boldsymbol{B}\boldsymbol{u_1} - \boldsymbol{\nu} \leq \delta$ then
    Move to Phase 2
else
    Set $\boldsymbol{u_S} = \boldsymbol{u_1}$.
end if
Phase 2:
Let $\boldsymbol{u}^{(0)}$ and $\boldsymbol{\mathcal{W}}^{(0)}$ be the resulting solution and working set from Phase 1.
Solve using Algorithm 2–1:
    $$\boldsymbol{u_S} = \arg\min_{\boldsymbol{u}} \|\boldsymbol{W}_{\boldsymbol{u}}(\boldsymbol{u} - \boldsymbol{u_d})\| \qquad (3–2)$$
    $$\text{s.t.} \quad \boldsymbol{B}\boldsymbol{u} = \boldsymbol{\nu}$$
    $$\underline{\boldsymbol{u}} \leq \boldsymbol{u} \leq \overline{\boldsymbol{u}}$$
```

3.1.2 Weighted Least-squares (WLS)

The same goals are applied for the WLS solver, but now, they are combined in only one least-square problem. The Equation (3–3) depicts it, with γ being the weighting factor, and a solution is more straightforward then (see Algorithm 3–2).

$$\left\|\boldsymbol{W_u}(\boldsymbol{u}-\boldsymbol{u_d})\right\|^2+\gamma\left\|\boldsymbol{W_v}(\boldsymbol{Bu}-\boldsymbol{\nu})\right\|^2=\left\|\underbrace{\begin{bmatrix}\gamma\boldsymbol{W_v B}\\ \boldsymbol{W_u}\end{bmatrix}}_{\boldsymbol{A}}\boldsymbol{u}-\underbrace{\begin{bmatrix}\gamma\boldsymbol{W_v \nu}\\ \boldsymbol{W_u u_d}\end{bmatrix}}_{\boldsymbol{b}}\right\|_2^2 \tag{3–3}$$

Algorithm 3–2 Weighted Least-squares [2]

```
Let $\boldsymbol{u}^{(0)}$ and $\boldsymbol{\mathcal{W}}^{(0)}$ be the resulting solution and working set from the previous sampling instant.
Solve the problem below with $\boldsymbol{A}$ and $\boldsymbol{b}$ defined in Eq. (3–3) and using Algorithm 2–1
    $\boldsymbol{u_W} = \arg\min_{\boldsymbol{u}} \|\boldsymbol{Au}-\boldsymbol{b}\|$    (3–4)
    $\underline{\boldsymbol{u}} \le \boldsymbol{u} \le \overline{\boldsymbol{u}}$
```

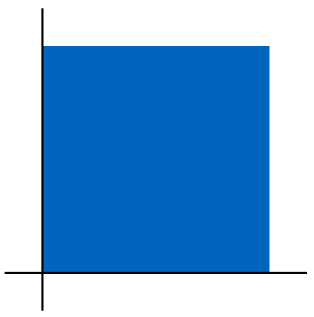
One can see that WLS solves an approximation problem parameterized by γ. Therefore, and as previously observed by Härkegård [2], a lower average allocation error is expected for SLS compared to WLS. However, this comes at the cost of increased elapsed time for SLS.

3.2 Embedded Conic Solver (ECOS)

ECOS is a solver built with a focus on embedded applications and uses different implementation techniques, such as sparse factorization, to save memory and execution time. It is designed to solve a SOCP problem, applying the Interior-Point method [10]. The standard form from the problem solved by ECOS can be visualized in the Eq. (3–4).

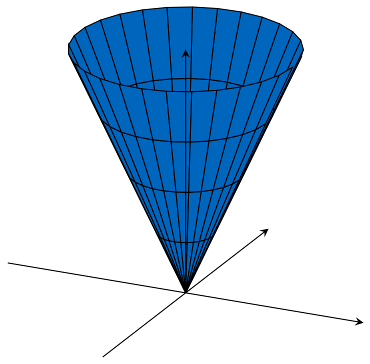
$$\begin{aligned}\min_{x}\quad & \boldsymbol{c}^\top\boldsymbol{x}\\ \text{s.t.}\quad & \boldsymbol{Ax}=\boldsymbol{b}\\ & \boldsymbol{Gx}+\boldsymbol{s}=\boldsymbol{h}\quad \boldsymbol{s}\in\boldsymbol{\mathcal{K}}\end{aligned}$$

Where $\boldsymbol{x}$ are the primal variables, $\boldsymbol{s}$ slack variables, $\boldsymbol{c}\in\mathbb{R}^n$, $\boldsymbol{G}\in\mathbb{R}^{M\times n}$, $\boldsymbol{h}\in\mathbb{R}^M$, $\boldsymbol{A}\in\mathbb{R}^{p\times n}$, $\boldsymbol{b}\in\mathbb{R}^p$ are the problem parameters, and $\boldsymbol{\mathcal{K}}$ is the cone. As stated by Domahidi [15], $\boldsymbol{\mathcal{K}}$ can be formed by a multiplication of multiple cones $\boldsymbol{\mathcal{K}}=\boldsymbol{\mathcal{K}_1}\times\cdots\times\boldsymbol{\mathcal{K}_N}$, and on top of that, the



$\mathbb{R}^n_+ = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x}_i \geq 0,\ \forall i\}$

(a) Positive orthant



$\boldsymbol{Q}^m = \left\{(x_0, \boldsymbol{x_1}) \in \mathbb{R} \times \mathbb{R}^{m-1} \mid \|\boldsymbol{x_1}\|_2 \leq x_0\right\}$

(b) Second-order cone

Figure 3–1: ECOS supported cones

cones can be classified in different types. Figure 3–1 illustrates the cones needed in the control allocation problem definition (i.e., $\boldsymbol{K_i} = \mathbb{R}^n_+$ and $\boldsymbol{K_i} = \boldsymbol{Q}^{m_i}$).

For control allocation purposes, the same approach used in the WLS is used for ECOS. Hence, the two constrained least-squares are combined in only one problem, and as a consequence, the same optimization goal depicted in Eq. (3–3) is applied. The problem below shall be then rewrited in the form from Eq. (3–4).

$$\begin{aligned} \min \quad & \|\boldsymbol{Au} - \boldsymbol{b}\|_2 \\ \text{s.t} \quad & \boldsymbol{Cu} \geq \boldsymbol{u_{lim}} \end{aligned} \tag{3–5}$$

This can be achieved by introducing a new variable t, redefining $\boldsymbol{x}$ as $\boldsymbol{x} = [\boldsymbol{u} \quad t]^\top$, and reformulating the problem accordingly. The vector $\boldsymbol{c}$ is then directly defined as $\boldsymbol{c} = [\boldsymbol{0}_{m\times 1} \quad 1]^\top$.

$$\begin{aligned} \min \quad & t \\ \text{s.t.} \quad & (t, \boldsymbol{Au} - \boldsymbol{b}) \in \boldsymbol{Q}^{m+1} \\ & \boldsymbol{Cu} \geq \boldsymbol{u_{lim}} \end{aligned} \tag{3–6}$$

One can see that two cones are applied: the first one is a second-order cone, here called quadratic constraint and given by $\|\boldsymbol{Au} - \boldsymbol{b}\|_2 \leq t$, and the second is a positive orthant called linear constraint. Therefore, by defining $\boldsymbol{G} = [\boldsymbol{G_{lin}} \quad \boldsymbol{G_{quad}}]^\top$ and $\boldsymbol{h} = [\boldsymbol{h_{lin}} \quad \boldsymbol{h_{quad}}]^\top$ in terms of the cones, the following formulation is obtained.

$$\boldsymbol{G_{quad}} = \begin{bmatrix} \mathbf{0}_{1\times m} & -1 \\ -\boldsymbol{A} & \mathbf{0}_{(m+p)\times 1} \end{bmatrix}, \qquad \boldsymbol{h_{quad}} = \begin{bmatrix} 0 \\ -\boldsymbol{b} \end{bmatrix} \tag{3–7}$$

$$\boldsymbol{G_{lin}} = \begin{bmatrix} \boldsymbol{I}_m & \mathbf{0}_{m\times 1} \\ -\boldsymbol{I}_m & \mathbf{0}_{m\times 1} \end{bmatrix}, \qquad \boldsymbol{h_{lin}} = \boldsymbol{u_{lim}} = \begin{bmatrix} \underline{\boldsymbol{u}} \\ -\overline{\boldsymbol{u}} \end{bmatrix} \tag{3–8}$$

With the problem defined, ECOS follows some steps until applying the IP method. As it is built to address multiple minimization and maximization problems, the first step is to write the extended self-dual problem. To do so, the variables τ, κ, and θ besides the primal variables $\boldsymbol{\chi} = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}, \boldsymbol{z})$ are added .

The Equation (3–9) illustrates the extended problem, where $M = \sum_{i=1}^{N} m_i$ (i.e., the sum of the cones sizes) [10]. One main advantage of this approach is that optimality and primal or dual infeasibility can be checked by computing τ and κ.

$$\begin{aligned} \min \quad & (M+1)\theta \\ \text{s.t.} \quad & 0 = \boldsymbol{A}^\top \boldsymbol{y} + \boldsymbol{G}^\top \boldsymbol{z} + \boldsymbol{c}\tau + \boldsymbol{q_x}\theta, \\ & 0 = -\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}\tau + \boldsymbol{q_y}\theta, \\ & \boldsymbol{s} = -\boldsymbol{G}\boldsymbol{x} + \boldsymbol{h}\tau + \boldsymbol{q_z}\theta, \\ & \kappa = -\boldsymbol{c}^\top \boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{y} - \boldsymbol{h}^\top \boldsymbol{z} + \boldsymbol{q_\tau}\theta, \\ & 0 = -\boldsymbol{q_x}^\top \boldsymbol{x} - \boldsymbol{q_y}^\top \boldsymbol{y} - \boldsymbol{q_z}^\top \boldsymbol{z} - \boldsymbol{q_\tau}\tau + M + 1, \\ & (\boldsymbol{s}, \boldsymbol{z}) \in \boldsymbol{\mathcal{K}}, \quad (\tau, \kappa) \geq 0. \end{aligned} \tag{3–9}$$

Subsequently, the next step is to search update directions by applying the Algorithm 2–2 (i.e. computing iterates $\boldsymbol{\chi}^{(k+1)} = \boldsymbol{\chi} + \alpha_k \Delta \boldsymbol{\chi}^{(k)}$). However, Domahidi et al. [10] demonstrate that the problem's central path is nonlinear and needs linearization. As a consequence, the search direction is divided into affine and centering-plus-corrector. Both of them are calculated by $\boldsymbol{K}\Delta\boldsymbol{\xi}_i = \boldsymbol{\beta}_i$, where $\boldsymbol{K}$ is the KKT matrix (see Eq. (3–10)) and $\boldsymbol{\beta}_1 \triangleq [-\boldsymbol{c}^\top \ \boldsymbol{b}^\top \ \boldsymbol{h}^\top]^\top$, $\boldsymbol{\beta}_2 \triangleq [\boldsymbol{d_x}^\top \ \boldsymbol{d_y}^\top \ \boldsymbol{d_z}^\top]^\top$.

$$\boldsymbol{K} \triangleq \begin{bmatrix} \mathbf{0} & \boldsymbol{A}^\top & \boldsymbol{G}^\top \\ \boldsymbol{A} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{G} & \mathbf{0} & -\boldsymbol{W}^2 \end{bmatrix} \tag{3–10}$$

$\boldsymbol{W}$ is the scaling matrix obtained by the Nesterov-Todd scaling, and it preserves the cone and central path [16]. It is calculated by $\boldsymbol{W} = \text{blkdiag}(\boldsymbol{W}_1, \boldsymbol{W}_2, \ldots, \boldsymbol{W_N})$, where i represent each cone. For the l linear cones (i.e. positive orthant and $s, z \in \mathcal{Q}^1$) $W_{+,i} = \sqrt{s/z}$, and for the $N - l$ second-order cones, $\boldsymbol{W_{SOC,i}}$ is obtained by the Eq. (3–11) [10].

$$\begin{aligned}
&\bar{\boldsymbol{z}} = \frac{\boldsymbol{z}}{(z_0^2 - \boldsymbol{z_1}^\top \boldsymbol{z_1})^{1/2}}, \quad \bar{\boldsymbol{s}} = \frac{\boldsymbol{s}}{(s_0^2 - \boldsymbol{s_1}^T \boldsymbol{s_1})^{1/2}}, \\
&\gamma = \left(\frac{1 + \bar{\boldsymbol{z}}^\top \bar{\boldsymbol{s}}}{2}\right)^{1/2}, \quad \bar{\boldsymbol{w}} = \frac{1}{2\gamma}\left(\bar{\boldsymbol{s}} + \begin{bmatrix} \bar{z}_0 \\ -\bar{\boldsymbol{z}}_{\boldsymbol{1}} \end{bmatrix}\right) \\
&\boldsymbol{W_{SOC}} \triangleq \eta \begin{bmatrix} \bar{w}_0 & \bar{\boldsymbol{w}}^\top \\ \bar{\boldsymbol{w}}_{\boldsymbol{1}} & \boldsymbol{I}_{m-1} + (1 + \bar{w}_0)\bar{\boldsymbol{w}}_{\boldsymbol{1}} \bar{\boldsymbol{w}}_{\boldsymbol{1}}^\top \end{bmatrix}, \quad \eta \triangleq \left(\frac{s_0^2 - \boldsymbol{s_1}^\top \boldsymbol{s_1}}{z_0^2 - \boldsymbol{z_1}^\top \boldsymbol{z_1}}\right)^{1/4}
\end{aligned} \tag{3–11}$$

Finally, the updates can be calculated as stated in Eq. (3–12) [10], where $\boldsymbol{\lambda} = \boldsymbol{W}^{-1}\boldsymbol{s}$. The whole process is summarized in the Algorithm 3–3.

$$\begin{aligned}
&\Delta\tau = \frac{d_\tau - d_\kappa/\tau + \begin{bmatrix} \boldsymbol{c}^\top & \boldsymbol{b}^\top & \boldsymbol{h}^\top \end{bmatrix} \Delta\boldsymbol{\xi_1}}{\kappa/\tau - \begin{bmatrix} \boldsymbol{c}^\top & \boldsymbol{b}^\top & \boldsymbol{h}^\top \end{bmatrix} \Delta\boldsymbol{\xi_2}}, \\
&\begin{bmatrix} \Delta\boldsymbol{x}^\top & \Delta\boldsymbol{y}^\top & \Delta\boldsymbol{z}^\top \end{bmatrix}^\top = \Delta\boldsymbol{\xi_1} + \Delta\tau\Delta\boldsymbol{\xi_2}, \\
&\Delta\boldsymbol{s} = -\boldsymbol{W}\left(\boldsymbol{\lambda}\backslash\boldsymbol{d_s} + \boldsymbol{W}\Delta\boldsymbol{z}\right), \\
&\Delta\kappa = -\frac{d_\kappa + \kappa\Delta\tau}{\tau}.
\end{aligned} \tag{3–12}$$

Algorithm 3–3 ECOS pseudo-code

```
Let $\boldsymbol{c}^\top$, $\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{G}$, and $\boldsymbol{h}$ be the problem definition data, as illustrated in Eq. (3–4)
Let $m$ be the cone sizes, $l$ be the number of linear cones, and $N$ the total number of cones
Let $\delta$ be the convergence tolerance
Compute $M = \sum_{i=1}^{N} m_i$ to do the problem extension
To initialize $\boldsymbol{\chi}^{(0)} = (\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}, \boldsymbol{s}^{(0)}, \boldsymbol{z}^{(0)})$ solve the following linear system [17]:
```

$$\begin{bmatrix} \mathbf{0} & \boldsymbol{A}^\top & \boldsymbol{G}^\top \\ \boldsymbol{A} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{G} & \mathbf{0} & -\boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{z} \end{bmatrix} = \begin{bmatrix} -\boldsymbol{c} \\ \boldsymbol{b} \\ \boldsymbol{h} \end{bmatrix}. \tag{3–13}$$

```
Then use $\boldsymbol{x}^{(0)} = \boldsymbol{x}$, $\boldsymbol{y}^{(0)} = \boldsymbol{y}$, and if $\boldsymbol{z}$ satisfy the cone constraints $\boldsymbol{z}^{(0)} = \boldsymbol{z}$ and $\boldsymbol{s}^{(0)} = -\boldsymbol{z}$
for $k = 0, 1, 2, \ldots, k_{max} - 1$ do
    for $i = 1, 2 \ldots N$ do
        if $i \leq l$ then
            $W_i = \sqrt{s/z}$
        else
            Compute $\boldsymbol{W_i}$ using Eq. (3–11)
        end if
    end for
    Compute $\boldsymbol{W} = \mathrm{blkdiag}(W_1, W_l, \boldsymbol{W_{l+1}}, \ldots, \boldsymbol{W_N})$
    Let $\boldsymbol{K}$ given by the Eq. (3–10) be the KKT matrix
    Solve for the affine and centering-plus-corrector directions:
```

$$\begin{aligned} \boldsymbol{K}\Delta\boldsymbol{\xi_1} &= \boldsymbol{\beta_1} \\ \boldsymbol{K}\Delta\boldsymbol{\xi_2} &= \boldsymbol{\beta_2} \end{aligned} \tag{3–14}$$

```
    Using Eq. (3–12) calculate the update direction $\Delta\boldsymbol{\chi}$
    Update variable $\boldsymbol{\chi}^{(k+1)} = \boldsymbol{\chi}^{(k)} + \alpha_k \Delta\boldsymbol{\chi}$. For a detailed computation of $\alpha_k$, see [17].
    Compute the residuals by:
```

$$\begin{bmatrix} \boldsymbol{r_x} \\ \boldsymbol{r_y} \\ \boldsymbol{r_z} \\ r_\tau \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \boldsymbol{s}^{(k+1)} \\ \kappa^{(k+1)} \end{bmatrix} - \begin{bmatrix} \mathbf{0} & \boldsymbol{A}^\top & \boldsymbol{G}^\top & \boldsymbol{c} \\ -\boldsymbol{A} & \mathbf{0} & \mathbf{0} & \boldsymbol{b} \\ -\boldsymbol{G} & \mathbf{0} & \mathbf{0} & \boldsymbol{h} \\ -\boldsymbol{c}^\top & -\boldsymbol{b}^\top & -\boldsymbol{h}^\top & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^{(k+1)} \\ \boldsymbol{y}^{(k+1)} \\ \boldsymbol{z}^{(k+1)} \\ \tau^{(k+1)} \end{bmatrix}. \tag{3–15}$$

```
    if $\|\boldsymbol{r}\|_2 \leq \delta$ and ${\boldsymbol{z}^{(k+1)}}^\top \boldsymbol{s}^{(k+1)} \leq \delta$ then
        Optimal solution found, return $\boldsymbol{\chi}$
    else if $\boldsymbol{h}^\top \boldsymbol{z}^{(k+1)} + \boldsymbol{b}^\top \boldsymbol{y}^{(k+1)} < 0$ then
        Return Primal Infeasibility
    else if $\boldsymbol{c}^\top \boldsymbol{x}^{(k+1)} < 0$ then
        Return Dual Infeasibility
    end if
end for
```

3.3 Code Generation for Convex Optimization (CVXGEN)

The last solver applied is called CVXGEN, and similarly to ECOS, it is built with a focus on embedded applications [18]. However, the target from CVXGEN is QP (see Eq. (3–16) [19]) instead of SOCP. This provides more simplicity in the implementation but less flexibility in the problem definition [18].

$$\begin{aligned} \min \quad & \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{q}^\top \\ \text{s.t.} \quad & \boldsymbol{G}\boldsymbol{x} \le \boldsymbol{h}, \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}. \end{aligned} \tag{3–16}$$

CVXGEN, in contrast to QCAT, utilizes the Interior-Point method and, therefore, it is a good choice to benchmark some alternatives. To find a solution, the process is nearly identical to the one applied for ECOS.

The first step consists of transforming the problem into a canonical form. This involves expanding via epigraphs and collecting all optimization variables into a single vector, as it was done for ECOS (see Eq. (3–6)). Since the control allocation problem already conforms to Eq. (3–16), this conversion is unnecessary, and the variables must be overridden accordingly (see Eq. (2–5)).

Again, this approach rewrites the problem in the extended self-dual form, and hence, the variable $\boldsymbol{\chi} = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}, \boldsymbol{z})$ is used. Next, the solver computes the affine and centering-plus-corrector directions. This is done by the KKT systems illustrated in Eq. (3–17) and Eq. (3–18).

$$\begin{bmatrix} \boldsymbol{Q} & \boldsymbol{0} & \boldsymbol{G}^\top & \boldsymbol{A}^\top \\ \boldsymbol{0} & \text{diag}(\boldsymbol{s}) & \text{diag}(\boldsymbol{z}) & \boldsymbol{0} \\ \boldsymbol{G} & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{x_1} \\ \Delta\boldsymbol{s_1} \\ \Delta\boldsymbol{z_1} \\ \Delta\boldsymbol{y_1} \end{bmatrix} = \begin{bmatrix} -(\boldsymbol{A}^\top\boldsymbol{y} + \boldsymbol{G}^\top\boldsymbol{z} + \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{q}) \\ -\text{diag}(\boldsymbol{s})\boldsymbol{z} \\ -(\boldsymbol{G}\boldsymbol{x} + \boldsymbol{s} - \boldsymbol{h}) \\ -(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \end{bmatrix} \tag{3–17}$$

$$\begin{gathered} \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{0} & \boldsymbol{G}^\top & \boldsymbol{A}^\top \\ \boldsymbol{0} & \text{diag}(\boldsymbol{s}) & \text{diag}(\boldsymbol{z}) & \boldsymbol{0} \\ \boldsymbol{G} & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{x_2} \\ \Delta\boldsymbol{s_2} \\ \Delta\boldsymbol{z_2} \\ \Delta\boldsymbol{y_2} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\sigma\mu 1} - \text{diag}(\Delta\boldsymbol{s_1})\Delta\boldsymbol{z_1} \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}, \\ \text{where} \quad \sigma = \left(\frac{(\boldsymbol{s} + \alpha\Delta\boldsymbol{s_1})^\top(\boldsymbol{z} + \alpha\Delta\boldsymbol{z_1})}{\boldsymbol{s}^\top\boldsymbol{z}}\right)^3 \quad \text{and} \quad \mu = \frac{\boldsymbol{s}^\top\boldsymbol{z}}{p}, \quad s \in \mathbb{R}^p \end{gathered} \tag{3–18}$$

A key point to emphasize is that CVXGEN explores a permuted $LDL^\top$ factorization to solve the KKT systems and speed up the solution finding. A dynamic regularization is applied to ensure numerical stability, and Hanger et al. [18] give its details.

Hereafter, the primal and dual variables are updated by $\boldsymbol{\chi}^{(k+1)} = \boldsymbol{\chi} + \alpha_k \Delta \boldsymbol{\chi}^{(k)}$, with α_k computed as depicted in Section 2.2.2. A summary of the entire process can be visualized in Algorithm 3–4.

Algorithm 3–4 CVXGEN pseudo-code

```
Let $\boldsymbol{Q} = 2\boldsymbol{A}^\top \boldsymbol{A}$, $\boldsymbol{q} = -2\boldsymbol{A}^\top \boldsymbol{b}$, $\boldsymbol{G} = -\boldsymbol{C}$, and $\boldsymbol{h} = -\boldsymbol{u_{lim}}$ with $\boldsymbol{A}$ and $\boldsymbol{b}$ defined in Eq. (3–3), and $\boldsymbol{C} = \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{I} \end{bmatrix}^T$ and $\boldsymbol{u_{lim}} = \begin{bmatrix} \underline{\boldsymbol{u}} & -\overline{\boldsymbol{u}} \end{bmatrix}^\top$.
Let $\delta$ be the convergence tolerance
To initialize $\boldsymbol{\chi}^{(0)} = (\boldsymbol{x}^{(0)}, \boldsymbol{s}^{(0)}, \boldsymbol{z}^{(0)})$ solve the following linear system [18]:
$$\begin{bmatrix} \boldsymbol{Q} & \boldsymbol{G}^\top \\ \boldsymbol{G} & -\boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{z} \end{bmatrix} = \begin{bmatrix} -\boldsymbol{q} \\ \boldsymbol{h} \end{bmatrix}. \qquad (3–19)$$
Then use $\boldsymbol{x}^{(0)} = \boldsymbol{x}$, and if $\boldsymbol{z}$ is feasible $\boldsymbol{z}^{(0)} = \boldsymbol{Gx} - \boldsymbol{h}$ and $\boldsymbol{s}^{(0)} = -\boldsymbol{z}$
for $k = 0, 1, 2, \ldots, k_{max} - 1$ do
    Find the affine and centering-plus-corrector directions as depicted in Eq. (3–17) and Eq. (3–18).
    See [18] for more details about dynamic regularization.
    Compute primal and dual variables updates with
    $\alpha = \min\{1,\ 0.99 \sup(\alpha \geq 0 \mid \boldsymbol{s} + \alpha\Delta\boldsymbol{s} \geq 0,\ \boldsymbol{z} + \alpha\Delta\boldsymbol{z} \geq 0)\}$:
    $$\begin{aligned} \Delta\boldsymbol{x} &= \Delta\boldsymbol{x_1} + \Delta\boldsymbol{x_2}, & \boldsymbol{x}^{(k+1)} &= \boldsymbol{x}^{(k)} + \alpha_k \Delta\boldsymbol{x}, \\ \Delta\boldsymbol{s} &= \Delta\boldsymbol{s_1} + \Delta\boldsymbol{s_2}, & \boldsymbol{s}^{(k+1)} &= \boldsymbol{s}^{(k)} + \alpha_k \Delta\boldsymbol{s}, \\ \Delta\boldsymbol{z} &= \Delta\boldsymbol{z_1} + \Delta\boldsymbol{z_2}, & \boldsymbol{z}^{(k+1)} &= \boldsymbol{z}^{(k)} + \alpha_k \Delta\boldsymbol{z}. \end{aligned} \qquad (3–20)$$
    Compute the residuals as follows with $\boldsymbol{S} = \mathrm{diag}(\boldsymbol{s})$ and $\boldsymbol{Z} = \mathrm{diag}(\boldsymbol{z})$:
    $$\begin{bmatrix} \boldsymbol{r_x} \\ \boldsymbol{r_s} \\ \boldsymbol{r_z} \end{bmatrix} = \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{0} & \boldsymbol{G}^\top \\ \boldsymbol{0} & \boldsymbol{S}^{-1}\boldsymbol{Z} & \boldsymbol{I} \\ \boldsymbol{G} & \boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^{(k+1)} \\ \boldsymbol{s}^{(k+1)} \\ \boldsymbol{z}^{(k+1)} \end{bmatrix} \qquad (3–21)$$
    if $\|r\|_2 \leq \delta$ then
        break
    end if
end for
```

4 Implementation

Three different datasets are used to evaluate each solver. They are constituted by different size problems and represented by different aircraft configurations. Therefore, the evaluation considers design variations and checks how the solver's performance accomplishes them. The details about each dataset are presented in the following sections.

Moreover, the implementation was done on an STM board to ensure accurate elapsed time measurement and minimize the influence of external system processes. The specifics will also be discussed in the upcoming sections.

4.1 Datasets

4.1.1 ADMIRE

The Aero-Data Model In a Research Environment (ADMIRE) models a single-seat fighter aircraft with a delta-canard configuration [20] (see Fig. 4–1). It became a commonly used model, as it provides a freely distributed model to the research community.

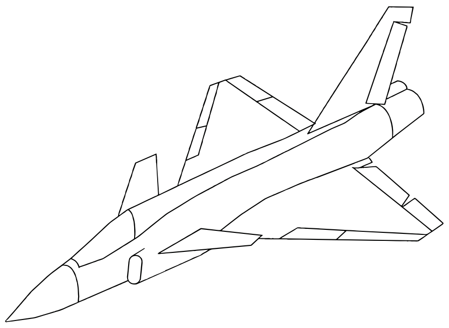


Figure 4–1: ADMIRE model layout configuration [20]

To put it concisely, the pseudo-control ν from ADMIRE is given by the control channels u_p, u_q, and u_β, representing the rolling, pitching, and yawing moments. Although the model's Flight Control System (FCS) uses seven control actuators [20], in this study, the actuators are grouped into four effectors: canard, left elevon, right elevon, and rudder. Hence, the dataset has the sizes $m = 4$ and $p = 3$.

The maximum and minimum deflections limit the effectors, which may vary with the Mach number. The values assumed in the dataset used here are displayed in Table 4–1.

Table 4–1: ADMIRE Effectors Deflection Limits

Control Surface	Min. [°]	Max. [°]
Canard	-55	25
Left Elevon	-30	30
Right Elevon	-30	30
Rudder	-30	30

4.1.2 12-copter

The 12-copter dataset represents a $600\ kg$ multicopter with twelve propellers mounted coaxially in a hexacopter layout. Accordingly, the problem dimensions are $n = 12$ and $p = 6$, with $\boldsymbol{\nu}$ representing the aircraft's three moments and three forces around the body-fixed frame. An Incremental Nonlinear Dynamic Inversion (INDI) control approach is used and, therefore, the input increments are obtained and allocated from the pseudo-control increments [21].

The effectors are composed of the squared propeller speeds, ω_i^2, and are again bounded by lower and upper positional limits. When working with actuators' rotational velocities, it is very likely to encounter differences in the order of magnitude, which may lead to numerical issues. To mitigate this, the problem matrices are normalized as shown in Eq. (4–1) [22], where $\boldsymbol{b}_i$ denotes the i-th column of $\boldsymbol{B}$.

$$\begin{aligned} \boldsymbol{V}_u &= \operatorname{diag}\left(\frac{\overline{\boldsymbol{u}} - \underline{\boldsymbol{u}}}{2}\right), \quad \boldsymbol{V}_\nu = \left[\operatorname{diag}\left(\sum_{i=1}^{m} |\boldsymbol{b}_i|\right)\right]^{-1} \\ \boldsymbol{B}_{norm} &= \boldsymbol{V}_\nu \boldsymbol{B} \boldsymbol{V}_u \end{aligned} \tag{4–1}$$

4.1.3 VTOL

The last dataset applied in the solver's evaluation is a VTOL. The configuration illustrated in Fig. 4–2 is applied to accomplish hover and cruise flight phases, where the aircraft is symmetric in the x_B axis. The pseudo-control ν is given by the incremental load factors Δn_x, Δn_y and Δn_z, and the incremental pitch, roll and yaw accelerations, $\Delta\dot{p}$, $\Delta\dot{q}$ and $\Delta\dot{r}$ respectively [23].

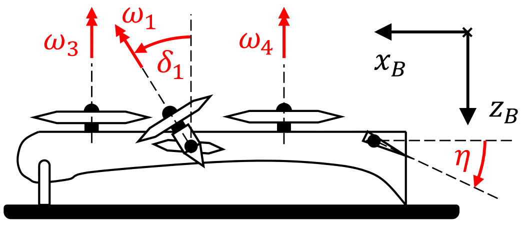


Figure 4–2: Side View of the VTOL configuration [23]

Furthermore, the aircraft features seven effectors: δ_1 and δ_2 represent the tilt servo deflection from the two propellers distributed symmetrically around the x axis [23]. ω_1 and ω_2 are the tilted propellers' rotational rates, and ω_3 and ω_4 the rotational rates from the front and rear propellers. Lastly, η is the elevator deflection.

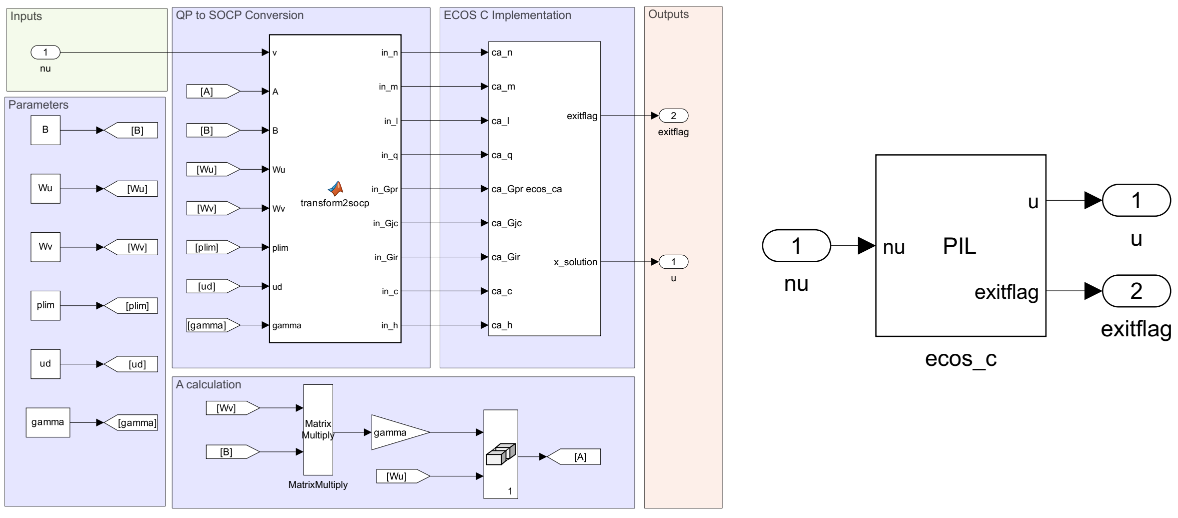
In summary, the model has dimensions $p = 6$ and $m = 7$, positioning it as an intermediate case in terms of size between the ADMIRE and the 12-copter datasets. Again, normalization of the inputs and pseudo-control is applied (i.e., $\boldsymbol{B}$ replaced by $\boldsymbol{B}_{norm}$) to handle order of magnitude differences.

4.2 Implementation in C and Testing

As anticipated from the references, the control allocation is expected to solve the problem within milliseconds. Therefore, the Matlab timer may not be the most adequate approach to measure elapsed time, as it is vulnerable to system interruptions and other phenomena in non-real-time systems [8].

Consequently, to avoid these problems, all solvers are made available in C code, and the simulations are run on an STM32F767 board. To facilitate the communication between the development computer and the board processor, *SIL/PIL Manager* App from *Simulink* is used.

Hence, all solvers are first implemented in *Simulink*, using either *Matlab Function* block or *C Caller* block. The Figure 4–3 illustrates the implementation of ECOS implementation, and the other solvers follow the same standard.



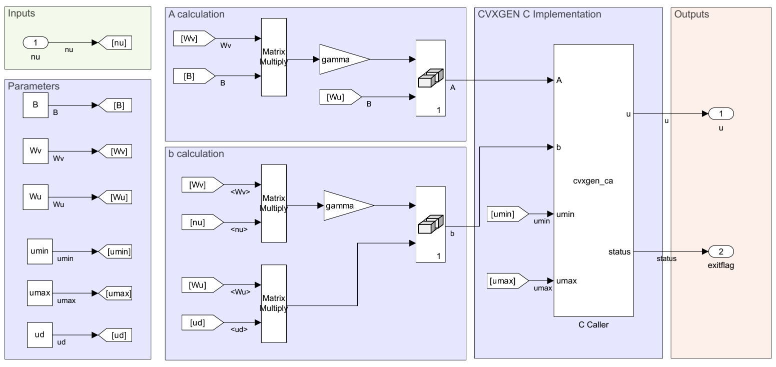
(a) ECOS *Simulink* Subsystem **(b) PIL Block created**

Figure 4–3: ECOS PIL Block creation from Subsystem

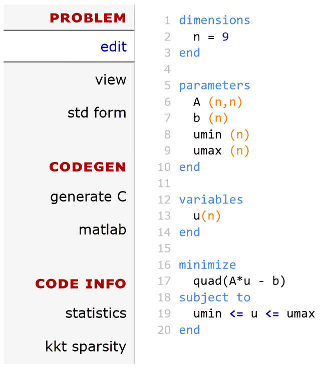
CVXGEN has its own online user interface[1] which requires a license to have access. However,

[1]CVXGEN Interface: https://cvxgen.com/docs/

the license is free to academic and non-commercial users. To generate the C code, the user should specify the problem dimensions, parameters, variables, and problem. The Figure 4–4 illustrates the VTOL specification and its implementation in *Simulink*.



(a) CXVGEN *Simulink* Subsystem



```
dimensions
  n = 9
end

parameters
  A (n,n)
  b (n)
  umin (n)
  umax (n)
end

variables
  u(n)
end

minimize
  quad(A*u - b)
subject to
  umin <= u <= umax
end
```

(b) VTOL specification

Figure 4–4: CVXGEN online interface and implementation in *Simulink*

The C code for ECOS is publicly available on *GitHub*[2], and unlike CVXGEN, the scripts are not problem-specific. ECOS is structured into three main functions: Setup, Solve, and Cleanup to handle dynamic memory allocation [10]. However, in this study, all three functions were used within the *Simulink C Caller* block, which may not benefit from implementation optimizations and can result in higher elapsed times to find a solution.

Lastly, QCAT WLS and SLS are both implemented in *Matlab* and made publicly available[3]. Their scripts are embedded in the *Matlab Function Simulink* block. Worth mentioning is that SLS was implemented using logical indexing, which requires dynamic memory allocation for C code generation. This may have also negatively impacted the elapsed time.

After implementing all solvers, testing is done by logging the elapsed time for each iteration and the corresponding computed solution. To cover a wide range of conditions and ensure sufficient test points, ν values are defined within the AMS, at its boundaries, and outside of it. Figure 4–5 illustrates the Attainable and Achievable moments for ADMIRE, where "Achievable" refers to values that lie within the limits defined by the minimum and maximum pseudo-controls.

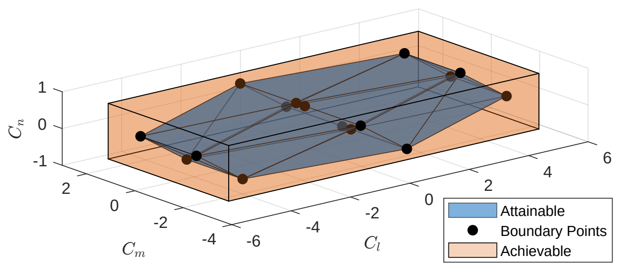


Figure 4–5: Visual Comparison of Achievable and Attainable ν Values in ADMIRE

[2]ECOS Repository: https://github.com/embotech/ecos.git
[3]QCAT Documentation: https://research.harkegard.se/qcat

The number of points on the boundaries corresponds to all binary combinations of upper and lower input limits. Hence, the number of points is 2^m, with m being the number of control inputs. After that, random points are sampled inside the AMS and between the AMS and the surrounding box, forming the so-called random and outside ν values, respectively. In this study, $10{,}000$ points were selected for each of them.

Finally, the solvers parameter, as convergence tolerance δ, weighting factor γ, maximum number of iterations k_{max} may also affect the results. For the tests, this parameters were kept fixed for all datasets and $\boldsymbol{\nu}$ values and they are defined in Table 4–2.

Table 4–2: Solvers Parameters

Solver	Weighting Factor γ	Convergence Tol. δ	Max. Iterations k_{max}
WLS	$1\text{E}3$	$1\text{E}-10$	100
SLS	N/A	$1\text{E}-10$	100
ECOS	$1\text{E}3$	$1\text{E}-8$	100
CVXGEN	$1\text{E}3$	$1\text{E}-6$	25

5 Results

The goal of this study, as highlighted before, is to find the best-fitted solver for the optimization-based control allocation. Therefore, to compare them, the next sections present an overview of the high-level performance with a low-level fidelity, a comparison between execution time and average error throughout all datasets, and an analysis of code certifiably using *Polyspace Bug Finder* and *Polyspace Code Prover*.

5.1 FLOPs Counting

To design and evaluate algorithms, assessments about the Floating Point Operations (FLOPs) are inevitable [24]. They provide an overview of performance and can also point to high-level improvements. This study estimates a FLOPs based on each operation involving scalar, vectors, or matrices, and Table 5–1 lists the number of FLOPs for the main operations used by the solvers.

Table 5–1: Floating Point Operations (FLOPs) for Common Operations

Operation	Input Sizes	FLOPs	Source
Scalar Addition $a + b$	$a \in \mathbb{R}, b \in \mathbb{R}$	1	[14]
Scalar Multiplication ab	$a \in \mathbb{R}, b \in \mathbb{R}$	1	[14]
Scalar Division a/b	$a \in \mathbb{R}, b \in \mathbb{R}$	14 on ARM32	[14]
Square Root $\sqrt{a}$	$a \in \mathbb{R}$	14 on ARM32	[14]
Dense multiplication AB	$A \in \mathbb{R}^{n\times m}, B \in \mathbb{R}^{m\times p}$	$np \cdot (2n-1)$	[14]
Moore–Penrose inverse A^+	$A \in \mathbb{R}^{n\times m}$	$14mn^2 + \frac{16}{3}n^3$	[25]
QR factors $A = QR$ (Householder factors)	$A \in \mathbb{R}^{n\times m}$	$2nm^2 - 2/3m^3$	[14]
Recover Q from Householder factors	$A \in \mathbb{R}^{n\times m}$	$2nm^2 - 2/3m^3$	[14]
Dense LDL Factorization	$A \in \mathbb{R}^{n\times n}$	$\frac{1}{3}n^3$	[26]

A more detailed breakdown of each solvers' operation can be found in their pseudo-codes in Chapter 3. Figure 5–1 shows the estimated FLOPs per dataset, obtained by dividing each solver's code into sections, recording the operations within each, and summing their FLOPs counts. The FLOP counting reflects a single iteration of each solver since the total number of iterations depends on the problem and cannot be known in advance.

The solvers based on the Interior-Point method (i.e., ECOS and CVXGEN) show a significantly higher FLOPs estimate compared to those using the Active-Set method. This aligns with the discussion in Chapter 2, as Active-Set typically solves a smaller problem by considering only the active constraints. However, since Active-Set often requires more iterations to converge, the observed advantage in FLOPs does not necessarily translate to reduced elapsed time.

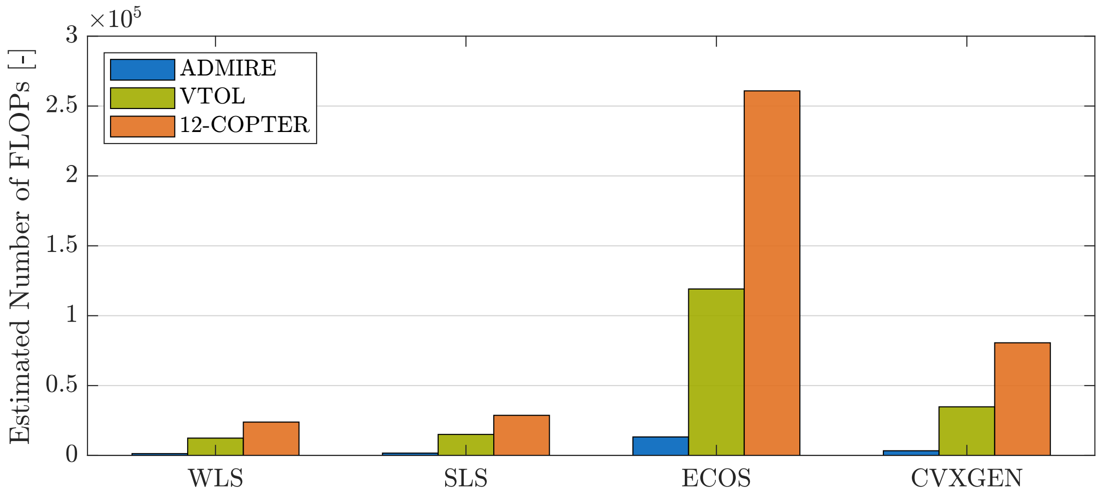


Figure 5–1: Estimated FLOPs for each Solver per Dataset

Moreover, it is important to emphasize that the estimations presented here assume dense matrix operations, which may lead to an overestimation of the FLOPs for ECOS and CVXGEN, as both solvers exploit sparse matrix structures. Although the control effectiveness matrix $\boldsymbol{B}$ is not sparse, the Control Allocation problem relies, for example, on effector limits and, therefore, involves highly sparse matrices such as $\boldsymbol{C} = [\boldsymbol{I} \quad -\boldsymbol{I}^\top]$. This highlights the importance of using sparse formats and efficient numerical methods to enable optimized implementations, such as the ones used by ECOS [10].

Lastly, it should also be noted that the estimated FLOPs for CVXGEN are lower than for ECOS, despite both using the same method and having similar implementations. This is because CVXGEN targets QP problems and, unlike ECOS, does not need to expand the problem. As a result, each iteration in CVXGEN involves one fewer variable due to the absence of the equality constraint $\boldsymbol{Ax} = \boldsymbol{b}$.

5.2 Elapsed Time and Errors

While the previous section focused on a theoretical assessment based on floating point operations, evaluating the solvers in practice is also essential. Therefore, this section presents a comparison of their actual elapsed time during execution, as well as the accuracy of their results.

Figure 5–2 shows an overview of each solver's performance across the datasets when using random $\boldsymbol{\nu}$ values within the AMS. The impact of implementation details becomes apparent, as ECOS underperforms in every case, even having a similar implementation to CVXGEN. This behavior may come from the current implementation, in which the *Setup*, *Solve*, and *Cleanup* routines are unnecessarily invoked at every iteration. This indicates a clear opportunity for future improvement.

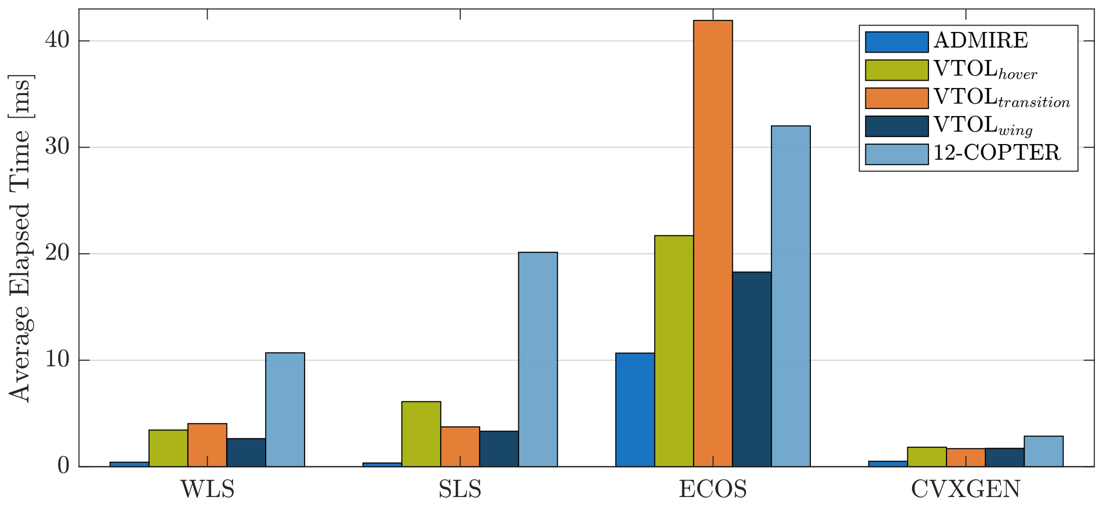


Figure 5–2: Comparison of Elapsed Time Across Solvers and Datasets

Another interesting observation is that CVXGEN performs well, even as the dataset size increases. This shows the strength of using the Interior-Point method and highlights the importance of complementing the evaluation with a real-case test, as the elapsed time from CVXGEN is lower despite the higher estimated FLOPs.

The elapsed time results also corroborate the statement from Wong [11], as although the FLOPs per iteration for CVXGEN may be higher, the time to find a solution is lower. Hence, this suggests that the solver reaches a solution with fewer iterations.

To further complement the analysis, Figure 5–3 were created to evaluate the solvers' behavior with ν values located within, on the boundary of, and outside the AMS. The points indicated with markers represent the mean values. These scenarios were explicitly designed to test the solvers under different levels of control demand and constraint activation, guaranteeing that the control allocation requirements are achieved.

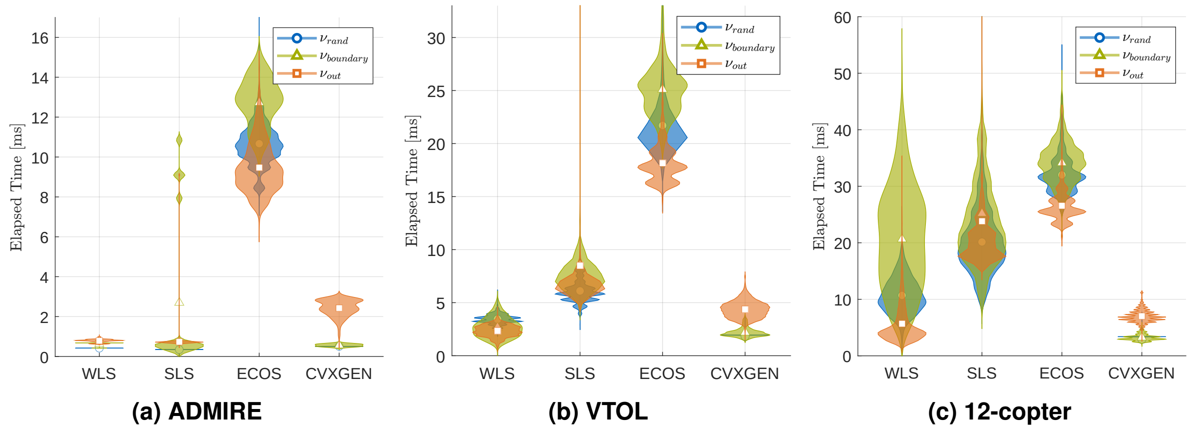
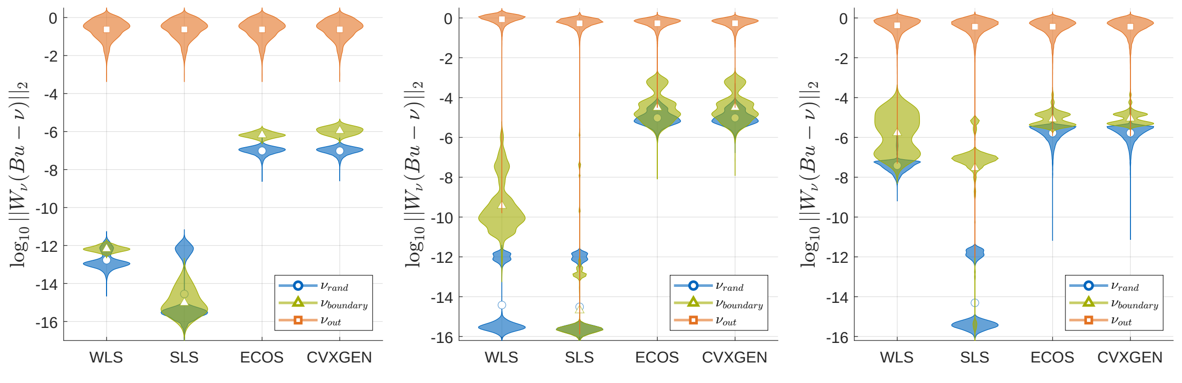


Figure 5–3: Comparison of Solvers Elapsed Time under different AMS Locations

Notably, having ν values outside the AMS does not necessarily result in higher elapsed times.

In fact, boundary moments are where most solvers tend to take the longest to find a solution. This may be explained by the increased number of iterations required as the solver approaches the boundary to reach the optimal solution.

Although the outside ν values do not require the most time to solve, they naturally result in higher allocation errors. Figure 5–4 illustrates this behavior and provides a general overview of each solver's performance in terms of allocation error.



(a) ADMIRE (b) VTOL (c) 12-copter

Figure 5–4: Comparison of Solvers Allocation Error under different AMS Locations

For completeness, a bar plot similar to Figure 5–2, but showing the allocation error instead, is presented in Figure 5–5. The errors are normalized by their respective averages to enable a meaningful comparison of solver behavior across different datasets.

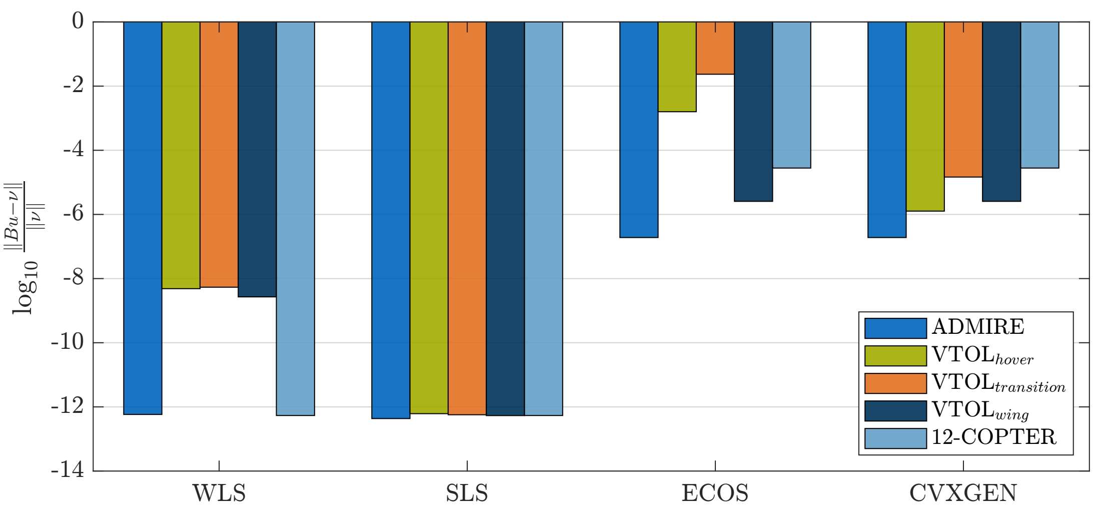
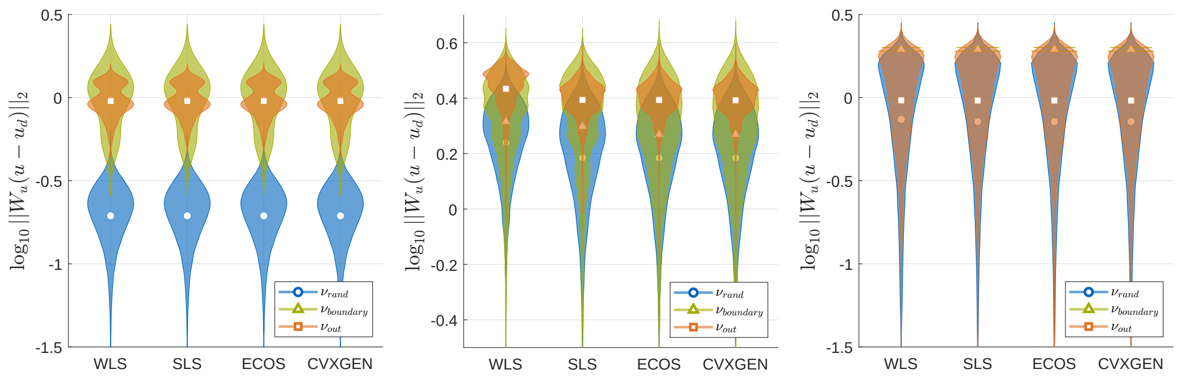


Figure 5–5: Comparison of Normalized Allocation Errors Across Solvers and Datasets

As previously mentioned, WLS, ECOS, and CVXGEN are implemented to solve an approximate version of the problem, weighted by the factor γ. As a result—and as expected—SLS generally performs better when considering only the allocation error. However, a clear trade-off emerges, as CVXGEN and WLS typically find solutions faster than SLS. Therefore, choosing a solver requires balancing these two key performance indicators.

While allocation error and computation time are important performance indicators, they do not provide a complete picture. Another relevant aspect of evaluating is the solver's ability to minimize the secondary objective related to control effort once the commanded pseudo-controls are achieved. This reflects how efficiently the solver distributes control inputs while meeting the primary control demand. Figure 5–6 illustrates the solvers' performance concerning this secondary objective error across different AMS locations and datasets.



(a) ADMIRE **(b) VTOL** **(c) 12-copter**

Figure 5–6: Comparison of Solvers Secondary Objective Error under different AMS Locations

The results indicate that all solvers exhibit comparable performance in minimizing the secondary objective error, with only minor variations across different ν scenarios. This suggests that no solver presents a clear advantage over the others in terms of control effort distribution.

5.3 Code Compliance with C Standards

Coding standards are key for aerospace applications, as they provide guidance to more safe, reliable, and secure systems [27]. Therefore, it is essential to assess the compliance of the solvers with some rules and recommendations, especially considering their application in control allocation aircraft systems.

Every solver evaluated in this study was reviewed about certifiability aspects by first implementing them in *Simulink* and generating the C code to be embedded on the STM board. *Polyspace Bug Finder* and *Polyspace Code Prover* are both applied to check compliance with *SEI CERT-C* and *MISRA C:2012* code standards.

The *SEI CERT C* is developed by CERT Division [27] and currently contains guidelines divided by 121 rules and 187 recommendations. Violating a rule means that the system is likely to result in a defect, which may affect safety, and violating recommendations suggests improvements for code quality may exist [27].

Similarly, *MISRA C:2012* is one of the standards developed by the MISRA Consortium and is primarily applied in automotive and aerospace applications. The guidelines are classified into rules and directives, which are subdivided into the categories "mandatory," "required," or "advisory" [28]. This study considers only mandatory and required guidelines, excluding those classified as advisory.

Figure 5–7 illustrates the results and highlights the advantages of using tools such as *Simulink* for deploying code to hardware. This is supported by the fact that both WLS and SLS are developed using *Matlab* functions, with C code generated according to user-selected guidelines and internally defined standards. As a result, fewer violations are observed for both solvers.

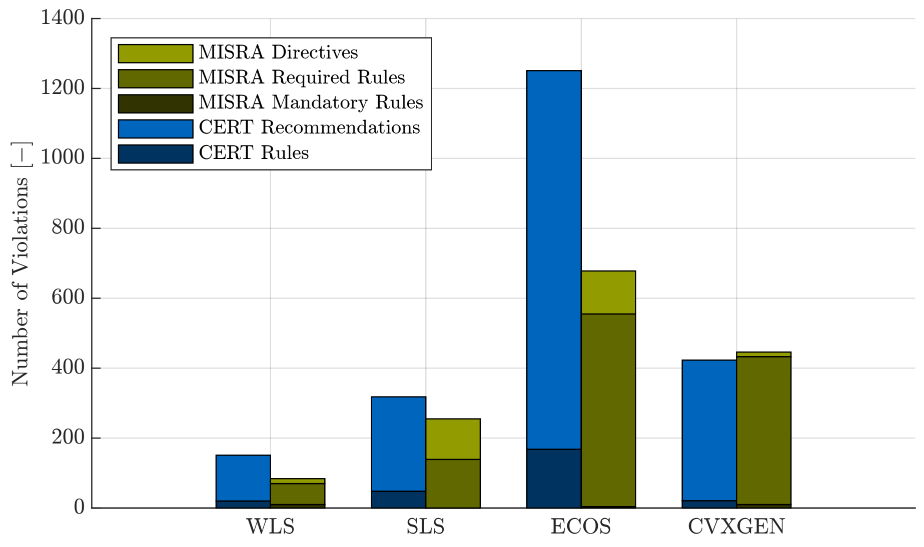


Figure 5–7: Code Guideline Violations According to SEI CERT C and MISRA C:2012

To further support this observation, Fig. 5–8 and Fig. 5–9 present the ten most violated coding standards identified for both WLS and SLS. The recurrence of some violations across both solvers suggests a consistent pattern followed by *Simulink* during code generation.

Furthermore, the difference observed between WLS and SLS points to the disadvantages of using dynamic memory allocation in SLS, as many coding standards require its absence [27]. This limitation may lead to additional violations and suggests that the implementation could be further optimized to enhance compliance.

Lastly, as mentioned in Section 4.2, CVXGEN generates code specifically tailored to the defined problem. As a result, the scripts are simplified and produced with fixed dimensions, parameters, and variables. In contrast, ECOS targets a broader class of problems (i.e., SOCP) and is designed to support a wide range of user-defined inputs and runtime environments.

This generality increases the complexity of the code and makes it more vulnerable to standard violations. One possible improvement would be refactoring the code by removing unnecessary components and retaining only those relevant to the control allocation problem. A clearer picture of this issue can be seen in Fig. 5–10 and Fig. 5–11, which present again the top 10 most frequently violated coding standards for ECOS and CVXGEN.

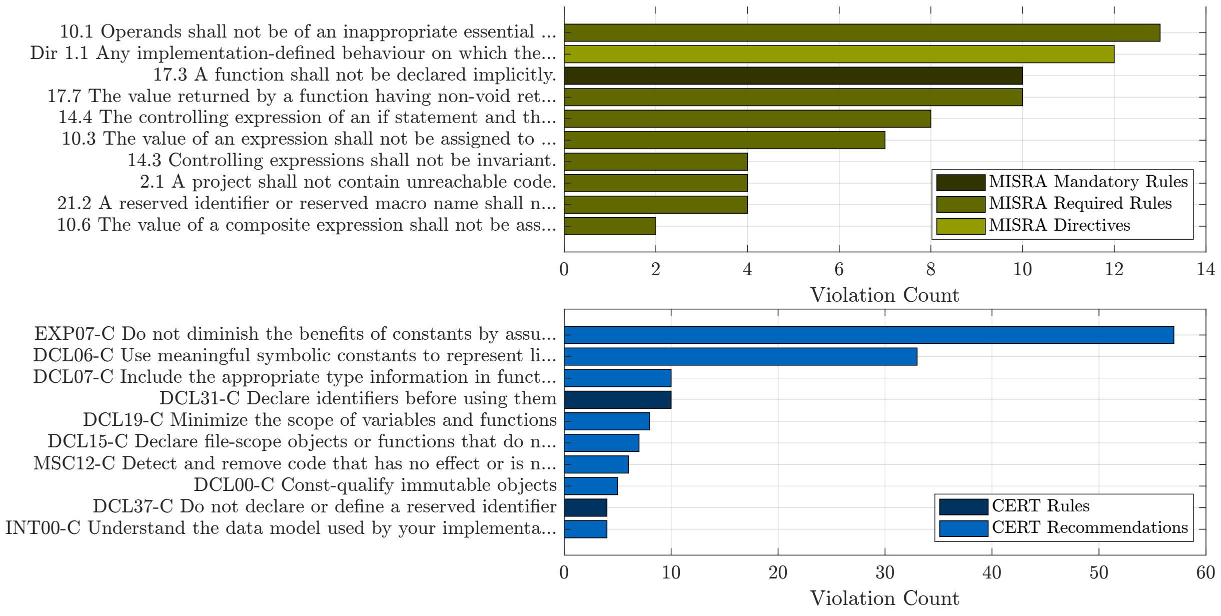


Figure 5–8: Top 10 Most Frequently Violated Coding Standards in WLS

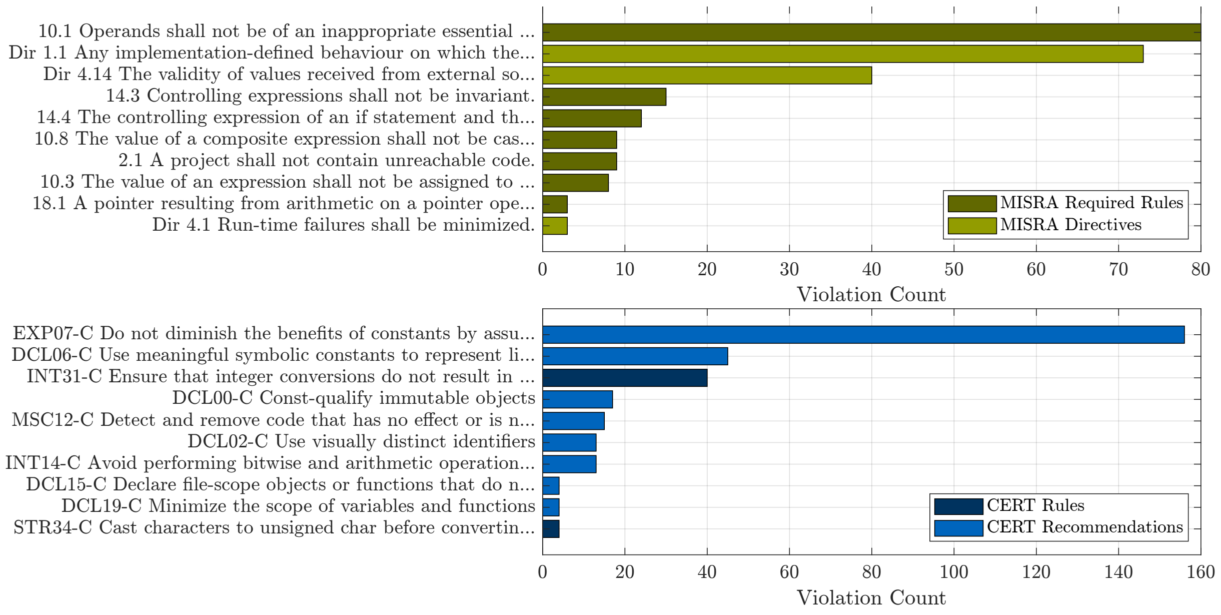


Figure 5–9: Top 10 Most Frequently Violated Coding Standards in SLS

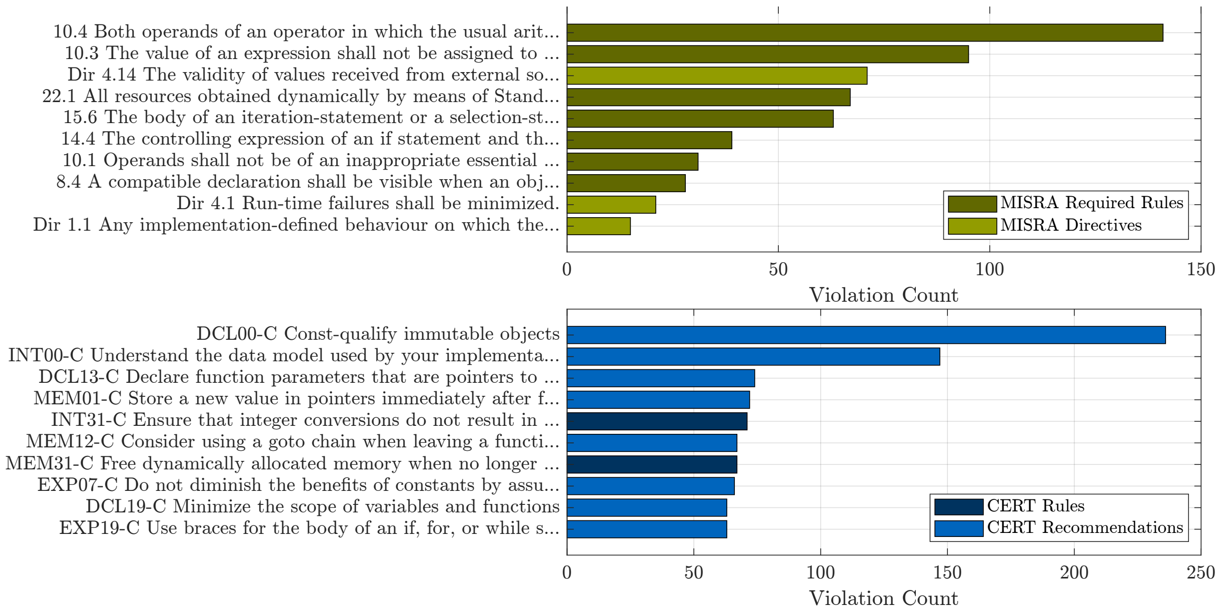


Figure 5–10: Top 10 Most Frequently Violated Coding Standards in ECOS

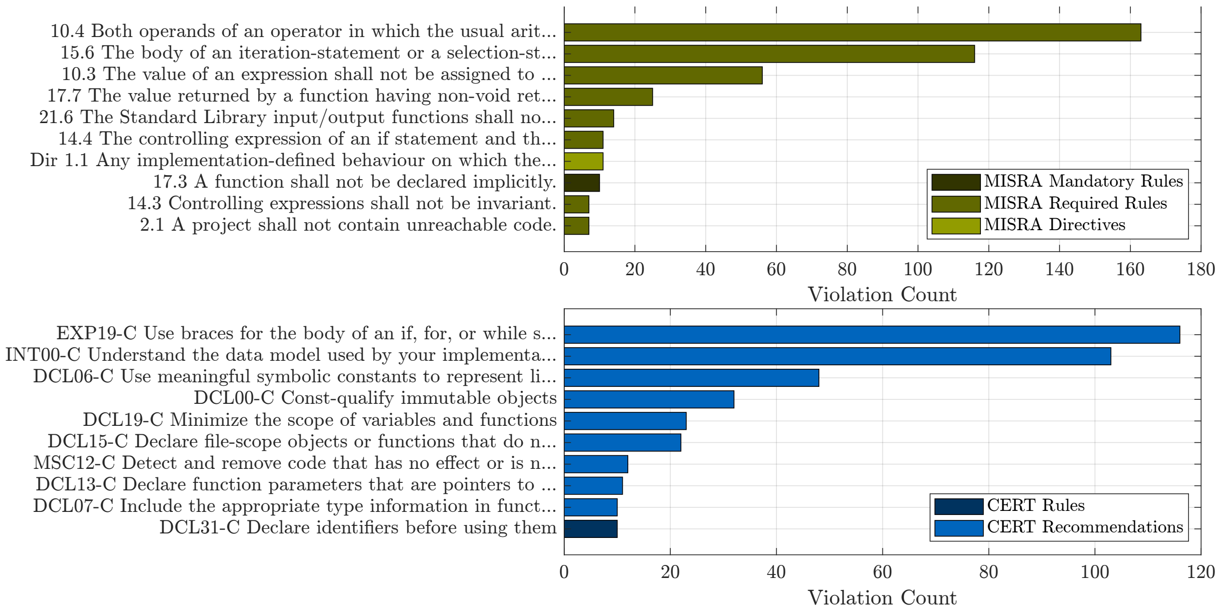


Figure 5–11: Top 10 Most Frequently Violated Coding Standards in CVXGEN

6 Conclusion and Outlook

Advancements in air mobility and fighter technologies enhance overall performance but also lead to more complex systems that must be robust and fully exploit their capabilities. Focusing on Control Allocation, this study evaluated different optimization implementations to solve the problem and benchmark available approaches.

Free solvers for academic users, applying Active-Set and Interior-Point methods, were used. To evaluate their performance, the first step was to estimate the required FLOPs for each solver. The solvers were then implemented in Simulink, and with the help of the *SIL/PIL Manager App*, each implementation was tested on a hardware board.

Execution time and computed solutions were logged. Average execution time, as well as allocation and secondary errors, were calculated to assess solver performance. Multiple datasets with varying sizes and configurations were used to support the analysis. Additionally, *Polyspace* was employed to evaluate the generated C code against coding standards.

6.1 Discussion of Results

The objective of this study was to evaluate if any of the solvers are best-fitted for Optimization-based control allocation. The results indicate that there is not a solver that stands out in all evaluated aspects (i.e., elapsed time, allocation error, secondary objective error, and conformity with C standards), but clearly, some of them have better performance in each aspect.

CVXGEN outperforms in terms of elapsed time and demonstrates less sensitivity to increases in problem dimension. This makes it a good fit for larger problems, a characteristic also discussed by Bodson [7] in the context of Interior-Point methods. However, the advantage is less significant for dimensions similar to those of the applied datasets, especially when compared to WLS.

The QCAT implementations of WLS and SLS appear to result in lower allocation errors, with WLS underperforming slightly due to it being a problem approximation. However, when considering the trade-off between error and elapsed time, WLS is a more favorable approach than SLS. It is essential to highlight the dynamic memory allocation required by SLS, and further investigation may lead to a more definitive conclusion.

Although ECOS implementation requires further improvement, the results showed that it is not the best suited for the Control Allocation problem compared to other solvers. A similar conclusion is obtained by Caron et al. [29], where QP problems are solved with multiple solvers, and their metrics are collected.

Finally, the certifiability aspect highlighted the power of using *Simulink* tools to generate C code, as WLS and SLS are the ones with fewer rules and recommendations violations. *Simulink* code generation includes adjustable parameters that can be aligned with *Polyspace* configuration settings.

6.2 Future Work

To further increase the reliability of the solver evaluation, the most immediate task is to improve the implementations of SLS and ECOS by avoiding dynamic memory allocation. Moreover, as stated earlier, CVXGEN appears to be less affected by increases in problem dimension. Therefore, this behavior could be further evaluated using a dataset with larger dimensions, and compared, for example, with WLS.

In addition to implementation improvements, ECOS has been found to be less effective for QP problems, and more recent solvers leveraging advanced techniques have outperformed it [29]. It may therefore be of interest to apply one of these solvers and assess whether it can outperform both CVXGEN and WLS.

Lastly, it is well known that Active-Set methods can benefit from previously computed solutions in certain cases. Hence, an evaluation using real and sequential ν values could help determine whether there is a performance gain when comparing Active-Set with Interior-Point methods.

References

[1] M. W. Oppenheimer, D. B. Doman, and M. A. Bolender, “Control allocation for over-actuated systems”, in *2006 14th Mediterranean Conference on Control and Automation*, 2006, pp. 1–6. DOI: 10.1109/MED.2006.328750.

[2] O. Härkegård, “Efficient active set algorithms for solving constrained least squares problems in aircraft control allocation”, in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, vol. 2, 2002, 1295–1300 vol.2. DOI: 10.1109/CDC.2002.1184694.

[3] J. W. Smolka et al., “F-15 active flight research program”, NASA Dryden Flight Research Center, Tech. Rep., 2005, Accessed: 2025-03-08. [Online]. Available: https://www.nasa.gov/image-article/f-15-active/.

[4] W. Durham, K. A. Bordignon, and R. Beck, *Aircraft Control Allocation*, en. Wiley, Nov. 2016. DOI: 10.1002/9781118827789. [Online]. Available: http://dx.doi.org/10.1002/9781118827789.

[5] National Aeronautics and Space Administration, *Multi-axis thrust-vectoring engine exhaust nozzles on f-15b*, Accessed: 2025-03-05, 1996. [Online]. Available: https://www.nasa.gov/image-article/multi-axis-thrust-vectoring-engine-exhaust-nozzles-f-15b/.

[6] T. A. Johansen and T. I. Fossen, “Control allocation—a survey”, *Automatica*, vol. 49, no. 5, pp. 1087–1103, 2013, ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2013.01.035.

[7] M. Bodson, “Evaluation of optimization methods for control allocation”, en, *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 4, pp. 703–711, Jul. 2002. DOI: 10.2514/2.4937. [Online]. Available: http://dx.doi.org/10.2514/2.4937.

[8] N. Thoma, *Implementation and Analysis of Optimization Based Control Allocation Methods*, Bachelor’s Thesis, 2023.

[9] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge: Cambridge University Press, 2004.

[10] A. Domahidi, E. Chu, and S. Boyd, *Ecos: An socp solver for embedded systems*, 2013. DOI: 10.23919/ECC.2013.6669541.

[11] E. Wong, “Active-set methods for quadratic programming”, Doctor of Philosophy in Mathematics, Ph.D. dissertation, University of California, San Diego, 2011.

[12] A. Singh and P. Ravikumar, *Primal-dual interior-point methods*, Lecture Notes for Convex Optimization 10-725/36-725, 2017.

[13] W. C. Durham, “Constrained control allocation - three-moment problem”, en, *J. Guid. Control Dyn.*, vol. 17, no. 2, pp. 330–336, Mar. 1994.

[14] T. Blaha, “Computationally efficient control allocation using active-set algorithms”, Master’s Thesis, Delft University of Technology, 2023. [Online]. Available: https://resolver.tudelft.nl/uuid:bffb47bf-5864-4b18-921b-588b3a664866.

[15] A. Domahidi, *Interior-point algorithms: Methods & tools, part ii: Conic ipms and ecos*, 2015.

[16] Emmanuel Candes, *Lecuture notes Advanced Topics in Convex Optimization: Interior Point Methods*, 2015.

[17] L. Vandenberghe, “The cvxopt linear and quadratic cone program solvers”, University of California, Los Angeles, Tech. Rep., Mar. 2010, Available online. [Online]. Available: https://www.seas.ucla.edu/~vandenbe/publications/coneprog.pdf.

[18] M. Hanger, T. A. Johansen, G. K. Mykland, and A. Skullestad, “Dynamic model predictive control allocation using cvxgen”, in *2011 9th IEEE International Conference on Control and Automation (ICCA)*, 2011, pp. 417–422. DOI: 10.1109/ICCA.2011.6137940.

[19] J. Mattingley, *Code generation for embedded convex optimization*, Available online, Oct. 2010. [Online]. Available: https://www.seas.ucla.edu/~vandenbe/publications/coneprog.pdf.

[20] L. Forssell and U. Nilsson, “Admire: The aero-data model in a research environment version 4.0, model description”, FOI – Swedish Defence Research Agency, Stockholm, Sweden, Tech. Rep. FOI-R–1624–SE, 2005, p. 36.

[21] T. Rupprecht, A. Steinert, C. Kotitschke, and F. Holzapfel, “INDI control law structure for a MEDEVAC eVTOL and its reference models: Feedforward, physical limitations, and innerloop dynamics for optimal tracking”, in *AIAA AVIATION Forum and ASCEND 2024*, AIAA, 2024. DOI: 10.2514/6.2024-4425. [Online]. Available: https://doi.org/10.2514/6.2024-4425.

[22] S. Hafner, S. Myschik, J. Bachler, and F. Holzapfel, “Fast pseudoinverse-based control allocation using a variant of the cholesky decomposition”.

[23] J. Zhang et al., “Control allocation framework for a tilt-rotor vertical take-off and landing transition aircraft configuration”, in *2018 Applied Aerodynamics Conference*, American Institute of Aeronautics and Astronautics (AIAA), 2018. DOI: 10.2514/6.2018-3480.

[24] R. Hunger, “Floating point operations in matrix-vector calculus”, Technische Universität München, Associate Institute for Signal Processing, Technical Report Version 1.3, 2007, First globally accessible version. [Online]. Available: mailto:hunger@tum.de.

[25] A. Smoktunowicz and I. Wróbel, “Numerical aspects of computing the Moore-Penrose inverse of full column rank matrices”, en, *BIT*, vol. 52, no. 2, pp. 503–524, Jun. 2012.

[26] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th. Baltimore, MD: Johns Hopkins University Press, 2013.

[27] CERT Division, “Sei cert c coding standard: Rules for developing safe, reliable, and secure systems”, Software Engineering Institute, Carnegie Mellon University, Tech. Rep., 2016. [Online]. Available: https://www.securecoding.cert.org/.

[28] *MISRA C:2012 – Guidelines for the Use of the C Language in Critical Systems*. Nuneaton, Warwickshire, UK: MIRA Limited, 2013, First published March 2013, ISBN: 978-1-906400-11-8. [Online]. Available: https://www.misra.org.uk.

[29] S. Caron et al., *qpbenchmark: Benchmark for quadratic programming solvers available in Python*, version 2.4.0, 2024. [Online]. Available: https://github.com/qpsolvers/qpbenchmark.