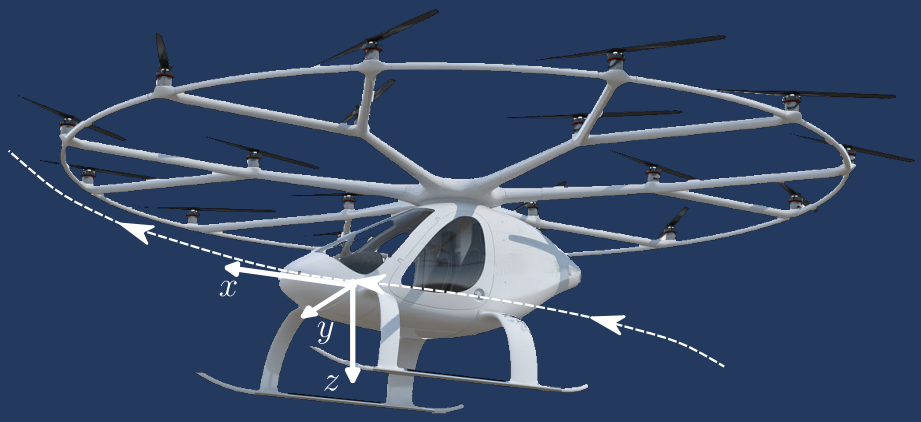
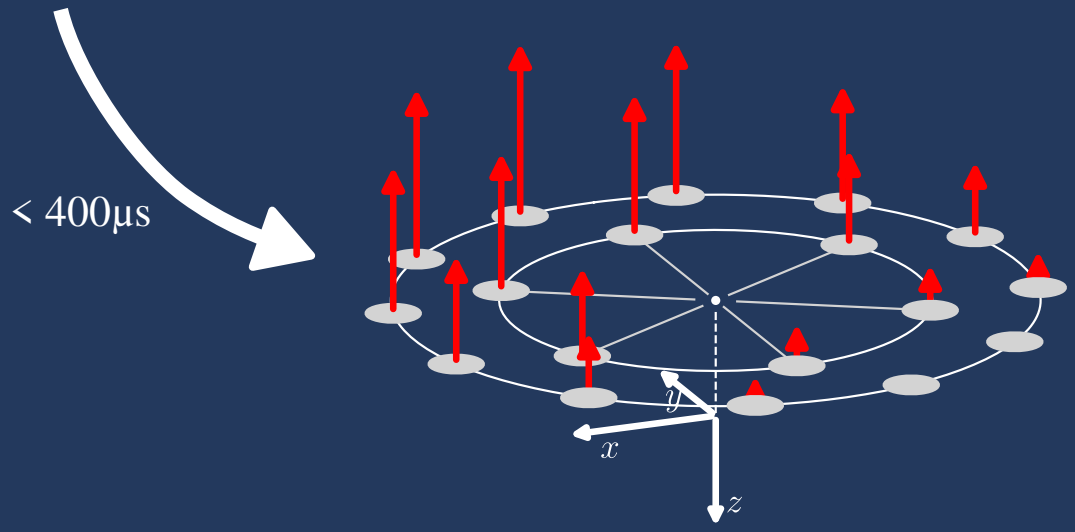
Computationally Efficient Control Allocation

Using Active-Set Algorithms

T.M. Blaha





TU Delft

Titleimage: Volocopter 2X. Reprinted with permission from https://www.cgtrader.com/3d-models/aircraft/other/autonomous-air-taxi-in-dubai-volocopter-2x-7d249f35-85a5-4fb1-bcf3-6b247072abc1

Computationally Efficient Control Allocation

Using Active-Set Algorithms

Thesis report

by

T.M. Blaha

to obtain the degree of Master of Science
at the Delft University of Technology
to be defended publicly on February 10, 2023 at 13:00

Thesis committee:	
Chair:	Dr C. de Visser
Supervisors:	Dr E.J.J. Smeur
	B.D.W. Remes
External examiner:	Dr J. Sodja
Place:	Faculty of Aerospace Engineering, Delft
Project Duration:	October, 2021 - February, 2023
Student number:	4597176

Faculty of Aerospace Engineering · Delft University of Technology

Preface

You, the reader, have opened my Master's Thesis report on "Computationally Efficient Control Allocation", which I have been working on for the past 16 months with the intention to graduate from the Delft University of Technology as a Master of Science. This research took me on a journey from books and journal articles, via some exploratory computer programming, all the way to fine-tuning the small puzzle pieces necessary to help fly a machine efficiently. Next to producing the computer code, and this document detailing its origins and performance, I gained valuable experience along the way that eventually top off my Master's education. Creating a pathway from initial experimentation to final validations of the product took me through months and months of writing, debugging, configuring, revising and optimising code in three programming languages, as well as soldering, measuring, flying, crashing and re-assembling a flying test-bench.

But my first steps in academic research have also shown me that, while it is sometimes beneficial and teaching to attempt things first-hand, a time has to come to instead focus on building on top of other people's smart and hard work. A common fallacy I have experienced is the urge to find reasons to come up with your own solution; "the problem is different", "it could be much simpler", "it could be much more general", while the real reason is "I take joy in doing it myself". To make meaningful contributions however, problems should only be solved where there are any, and the perfect solution never exists.

I would like to thank Dr Ewoud Smeur and Bart Remes, my supervisors, for their support and guidance. You have dived into the complex material with me and helped me solve issues, but also pushed me to focus on the relevant things and to finally complete the flight tests. I also want to thank Dr Coen de Visser for agreeing to chair the graduation committee. When I hit a dead-end on setting up my experiment flight platform, Erik van der Horst's experience and time were invaluable to finding the right settings and buttons. All my fellow Sim008 roommates and coffee corner regulars have made my time a lot more enjoyable, and helped with a suggestion here and there. Finally, I want to thank my family, friends and flatmates for being there for me, but also sharing the good things in life with me.

T.M. Blaha
Delft, January 2023

Contents

1 Introduction **1**
1.1 Flight Control and Control Allocation . . . 1
1.2 Research Formulation . . . 1
1.3 Structure of the Report . . . 2

I Preliminary Analysis **3**

2 Literature Review **4**
Executive Summary . . . 4
2.1 Introduction . . . 6
2.2 Aircraft Attitude Control Schemes . . . 9
2.3 Control Allocation . . . 12
2.4 Numerical Optimisation . . . 22
2.5 Comparative Testing and Benchmarks . . . 32
2.6 Conclusions and Future Work . . . 34

3 Preliminary Work **38**
3.1 Interior Point Methods for $\ell_2 - \ell_1$ Norms . . . 38

II Scientific Article **41**

4 Computationally Efficient Control Allocation Using Active-Set Algorithms **42**
4.1 Introduction . . . 42
4.2 Optimising control allocation formulations and solvers . . . 43
4.3 The Active-Set algorithm . . . 44
4.4 Methodology . . . 46
4.5 Results & Discussion . . . 49
4.6 Conclusions & Recommendations . . . 59
4.7 Appendices . . . 59

III Additional Results **67**

5 Additional Results **68**
5.1 MATLAB framework . . . 68
5.2 Additional Algorithms Partly Implemented . . . 69
5.3 Test Platform Specifications and Tuning . . . 70

IV Closure **73**

6 Conclusion **74**
6.1 Closing Remarks . . . 74
6.2 Research Questions . . . 74
6.3 Additional Conclusions . . . 75

7 Recommendations **76**

References **80**

Nomenclature

List of Abbreviations

(e)VTOL	(electric) Vertical Take-Off and Landing Vehicles
AMS	Attainable Moment Set
BLAS	Basic Linear Algebra Subroutines
DCA	Direct Control Allocation
DEP	Distributed Electric Propulsion
FLOPS	Floating Point Operations
ICE	Lockheed Martin Innovative Control Effectors aircraft
INDI	Incremental Nonlinear Dynamic Inversion control scheme
MAVLab	Micro Aerial Vehicle Lab (of the TU Delft)
MC/DC	Modified Condition/Decision Coverage
NDI	Nonlinear Dynamic Inversion control scheme
UAV	Unmanned Aerial Vehicle

Modifiers

$\dot{X}$	Time derivative of X
$\overline{X}$	Upper Bound for X
$\underline{X}$	Lower Bound for X
X^+	Pseudo-inverse of X
X^+, X^-	Positive part, negative part of X
X^+_W	Weighted Generalised Inverse of X with respect to W

Number sets

$\mathbb{R}$	Real numbers
d	Dimension of the pseudo-control vector ν
n	Dimension of the actuator position vector u
p	Dimension of the system output vector y
S	Feasible region of the actuator limits

List of Symbols

$\bar{J}$	3x3 Inertia matrix
β	Linear objective function coefficients of quadratic problem
ℓ_p	p-Norm
γ	Priority scalar for pseudo-control error.
$\hat{u}$	Estimated actuator positions
$\mathcal{L}$	Lagrangian
$\mathcal{L}_F a(x)$	Lie derivative of a(x) over the vector field F(x)
ν	Pseudo-control demanded from the actuators by the attitude control system.
ν_a	Pseudo-control achieved by the actuators.
Ω	Available Controls Set, subset of $\mathbb{R}^n$
ω	Rotational velocity
$\mathcal{P}(x)$	Projection operator
Φ	Attainable Moment Set, subset of $\mathbb{R}^d$
ρ	Pseudo-control scalar in DCA
$\mathcal{W}$	Working set of active-set algorithms
$\{A, B, C\}_{\text{sys}}$	LTI system matrices pertaining to system sys
$\{Q, R\}$	QR-decomposition
A	Linear equality constraint matrix

b	Linear equality constraint right hand sides
$B(.)$	Control effectiveness matrix. May be a constant or a function of system states.
$b(.)$	Control effectiveness intersect with $u = 0$ in affine model.
C	Linear inequality constraint matrix
c	Linear objective function coefficients
e	Pseudo-control error
e_i	Standard unit vector in direction i
$f(x)$	System evolution function
f_s	Sampling frequency
G	Ganging matrix
g	Linear inequality constraint right hand sides
$G(x)$	State-dependent input matrix
H	Hessian of quadratic problem
h	Vector of upper bounds
$h(x)$	Output function
i	Generic iteration variable
I, I_n	Identity matrix (of dimension n)
j	Generic iteration variable
k	Generic iteration variable
$L(s)$	Linear transfer function
N	Number of timesteps
n_f	Number unbounded (”free”) actuators
p	Scalar describing the ℓ_p norm
p_k	Search direction
s	Complex frequency
t	Time
u	Commanded actuator positions
u_p	Preferred actuator positions
$V(x, \nu)$	Control Lyapunov function
W_ν	Pseudo-control error weighting matrix
W_u	Actuator penalty weighting matrix
x	System state OR General variable to be solved for.
y	System output.

1

Introduction

The report at hand details and motivates the work done to improve the computational efficiency of optimising control allocation methods. This introduction serves to contextualise this work within the field of aircraft control, show the research objective and guide the reader through the rest of the contents.

1.1. Flight Control and Control Allocation

Historically, flight control of aeroplanes featured a one-to-one mapping between pilot inputs and actuators response; a pitch command on the stick would result in the elevator being moved. Over the last century, aircraft configurations have significantly increased in complexity, as redundancy to failures and performance for certain applications has been steadily enhanced. High performance fighter jets, helicopter, Vertical Take-off and Landing (VTOL) configurations and novel aeroplane configurations feature different control ”actuators”, such as rotors, motors, aerodynamic control surfaces, tilting rotors, and more. Therefore, this one-to-one mapping loses its meaning, and more complex and usually digital systems are required.

For most types of copters (vehicles such as helicopters that use vertical thrust to counter gravity), their attitude with respect to ground determines the flight path this enables the lift vector to have a horizontal component and accelerate the craft. For aeroplanes also the magnitude of the airspeed and the attitude with respect to the oncoming air determine the aerodynamic forces and flight path. Attitude control is thus necessary building block to controlling flight path and achieving meaningful tasks.

In many digital flight control systems (such as Snell et al., 1992 for high-performance aircraft, but also Bodson, 2002 for transport aircraft and Smeur et al., 2017 for multicopters), an attitude and path controller calculates a certain force and moment demand around the craft’s body axes (so maximally six values). These needs to be provided by the available collection of actuators (potentially much more than six) and so a translation step is needed, which is known as ”Control Allocation”.

1.2. Research Formulation

A literature review of available Control Allocation methods (see chapter 2) has revealed that of the many methods that have been developed, optimisation based approaches have been shown to best exploit the capabilities of a platform. However, it was also shown that these methods suffer from high computational demands from their solution algorithms, which make them unsuitable for larger platforms or platforms with small computational resources, such as UAVs.

Therefore, the research objective and main research question was formulated as follows:

Research Objective

Future research shall aim at finding relevant improvements of numerical optimisation algorithms for control allocation of UAVs and eVTOL, by implementing possible improvements over a baseline method and evaluating accuracy of the solutions, computational speed and balancing of the solution over the available actuators.

Research Question 1

What possible changes to the numerical algorithms for optimising control allocation can be shown to provide significant improvements over the current state-of-the-art?

1.3. Structure of the Report

Background information on flight control, control allocation and numerical solution methods as well as the reasoning and concrete approaches for the research questions above can be found in the literature review Chapter 2 and the preliminary work Chapter 3.

Part II shows the primary outcome of the thesis research as a standalone article. Complementing the paper, additional results that arose during the completion of the thesis are summarised in Chapter 5 and overarching conclusions and recommendations are shown in Part IV.

Part I

Preliminary Analysis

*This part has been assessed for the course AE4020 Literature Study.

2

Literature Review

Executive Summary

Presented in this review are investigation and analyses of literature related to Control Allocation, to identify possible improvements to enhance Control Allocation method for recent configurations of unmanned aerial vehicles (UAVs) and vertical takeoff and landing vehicles (VTOLs).

Preliminaries Control allocation is shown to arise naturally from common aircraft attitude control schemes (such as PID, (incremental) Nonlinear Dynamics Inversion) and is defined as the distribution of required forces and rotational torques over the available actuators. An important quantity in the control schemes is the effectiveness matrix B, which achieves the inverse task, calculating forces and torques from an actuator distribution.

An investigation of software conventions in commercial aircraft reveals that onboard software components that achieve these tasks would be required the highest safety considerations including a worst-case execution timing analysis and forbids the use of optimising compilers. Software onboard UAVs, especially consumer UAVs, usually require less strict analysis and rely more on testing.

The research in control allocation was found to best be split into three main aspects:

1. Different problem formulations to map the required control over the actuators.
2. Numerical real-time solvers if the problem formulation requires such a step to be done online.
3. Comparative testing procedures on benchmark problems.

Control Allocation Formulations Ganging, which assigns rigid groups is the simplest form of control allocation that explicitly maps any pseudo-control to an actuator distribution with a static mapping matrix. However, it is unsuitable to actuators encountered in UAVs and eVTOLs which cannot be explicitly grouped into groups that mainly act around a single rotation axis.

Weighted Generalised Inverses of the effectiveness matrix are attractive for their simplicity, as they provide a straightforward way to derive a static mapping matrix that fulfils a secondary objective to balance out the total control effort. This method can be expanded to ”Daisychaning”, which uses multiple decoupled effectiveness matrices to use certain actuators only when the primary actuators would exceed their limits (”saturate”). This however requires such a decoupling to be possible/appropriate which cannot be guaranteed for all UAVs.

Another class of methods that can redistribute control effort on saturation are ”optimising” control allocation schemes. The objective function is commonly the normed difference between the achieved pseudocontrol and the demanded pseudocontrol, which is to be minimised. Actuator limits are added as bound constraints and a secondary, de-prioritised, term is added to the objective to reduce the deflection of the actuators from a ”preferred” state. This is a powerful concept that results in high exploitation of the available actuators, while still balancing them and even allowing the prioritise some actuators for criteria such as drag or energy consumption.

The most powerful methods found to be able to control nonlinear and coupled actuators suites found in UAVs and eVTOLs are dynamic and nonlinear control allocation schemes, which extend linear optimising

control allocation by working with increments. Exact nonlinear optimisation was found not to be tractable for online control allocation, but not required even for challenging platforms, if the update frequency is high.

It should be noted that these three most powerful methods all rely on an identical bound-constrained optimisation routine, which was investigated next.

Optimisation Solvers Different norms can be chosen for the actuator deflection penalty and the pseudo-control error. The most popular in literature which gives the most naturally distributed solutions is the quadratic ℓ_2-norm that was also used to derive the Weighted Generalised Inverse solutions in the non-redistributing case. The most common solver is the active-set strategy which has been thought to be computationally tractable but has recently found its limits on UAV platforms with more than 15 actuators. A "Projected Gradient" strategy is used in literature on similar bound-constrained problems but has never been applied to control allocation.

However, different norms (sum of absolute value ℓ_1, ℓ_2, and the maximum element of the vector (supremum norm) ℓ_∞) for either term are possible and have been investigated. On top of that, different solvers can solve these different combinations. ℓ_1 is generally discouraged, although commonly theorised because it can lead to only very few actuators being used, as opposed to balancing the load, as would be ideal for vehicles with many redundant actuators. ℓ_∞ on the other hand *can* balance the actuators by minimising the most deflected actuator is attractive, while it can also be solved with fast linear programming methods. ℓ_2 for the actuator deflection, but ℓ_1 for the pseudo-control error has not been mentioned in literature before, but may be solved by interior-point methods with favourable scaling to many actuators.

Benchmark Testing Several comparative studies for control allocation methods and solvers have been analysed and it was found that the number of test cases used in these studies is generally low and closed-loop simulations are not commonly performed. This limits the explanatory power for the distribution balancing and the effect of sub-optimal solvers.

Future Research Planning The results are used to incite further research in the field by providing some relevant research questions centred around how to improve and compare different solvers. It was concluded that incremental nonlinear control allocation is a suitable allocation/control scheme for UAVs/VTOLs, but improvements on the part of the solvers are needed to enable allocation for platforms with many actuators.

2.1. Introduction

Control allocation is part of aircraft flight control software, with some components running on onboard digital flight computers. This chapter first introduces some preliminaries related to Automatic Flight Control and Flight Control Software in general.

2.1.1. An Overview of Automatic Flight Control

Automatic flight control denotes the system that enables aircraft to achieve a certain flight task (e.g. flying from one position to the next, executing a turn, landing) without the need for active intervention by a pilot. Frequently, this capability is implemented with a cascaded control system where the outer loops use knowledge of the aircraft's state and desired state to give setpoints to the inner loops. The inner-most loop can be an attitude controller, that finally uses the available hardware to achieve a desired attitude or rate command; be it commanded by a human pilot or one of the outer control loops.

Attitude Control

For most types of copters (vehicles that use vertical thrust to counter gravity), their attitude with respect to ground determines the flight path this enables the lift vector to have a horizontal component and accelerate the craft. For aeroplanes also the magnitude of the airspeed and the attitude with respect to the oncoming air determine the aerodynamic forces and flight path. Attitude control is thus necessary building block to achieving higher levels of autonomy in aircraft.

To achieve this, any aircraft requires a number of ”actuators” to influence the attitude by providing forces and rotational moments and a scheme to make use of the actuators sufficiently well to eventually achieve a desired attitude and flight path.

Control Actuators

The ”suite” of physical actuators on aircraft can be diverse. Classical planes, for example, most single-engine sport planes, make use of elevators, rudders, ailerons, flaps and propeller speed. Modern transport aeroplanes commonly feature more actuators to facilitate redundancy and enhance performance, extending the aforementioned basic suite by spoilers, leading edge slats and all-moving horizontal stabilisers. Also, control surfaces are often sectioned to allow them to be actuated by redundant drive systems. High-performance aircraft often add actuators to increase the bandwidth and control authority, such as moving foreplanes and thrust vectoring. More recently, spoiler slot deflectors and moving wingtips have been studied to effectively control tailless aircraft such as the Lockheed Martin Innovative Control Effector (ICE) aircraft (eg Matamoros, 2017).

In the realm of multi-rotors, the rotors are used not only as a means of propulsion but also to provide attitude control. In electric vertical take-off and landing vehicles (eVTOLs), it is common to combine many rotors and classical control surfaces to facilitate both vertical flight, efficient forward flight and the transition phase between the two (De Wagter et al., 2020, Nathen et al., 2021). Increasingly, Distributed Electric Propulsion (DEP) is studied for unmanned and manned flight. With DEP, many smaller engines are used instead of a few more powerful ones to increase redundancy and improve aerodynamic efficiency by exploiting jet-wing interaction and using the rotors for attitude control reducing empennage size (Nguyen Van et al., 2018, Zhang et al., 2021).

The actuator suite of an aircraft can encompass a multitude of types that show different levels of non-linearity, state dependency and coupling between different actuators. On many aircraft, the total number of actuators greatly exceeds the degrees of freedom of the control problem, which may be between 3 (rotational motion only) and 6 (rotational motion plus direct translational control).

Penalties The use of actuators is of course necessary but generally comes with a few penalties. Actuators of any type require power to drive them. Aerodynamics control surfaces have inherent drag and increase the radar cross-section of the vehicle. Depending on the use case, these penalties should be included in the control allocation scheme.

Control Allocation

Frequently in aircraft control, pseudo controls are computed by the attitude controller that represent the forces and moment required from the control actuator suite to move towards the desired attitude. NDI and

INDI naturally make use of this idea, see section 2.2.2. As the final step in these controllers, a mapping is needed that distributes these commands over the available actuators in the suite.

This can be formulated as finding solutions to equation 2.1, where ν is the vector of desired pseudo controls, u the vector of actuator commands, t denotes a time-dependency and B is a general known (or approximated) function that maps actuator states to achieved pseudo controls.

$$\vec{\nu} = B(\vec{x},\ \vec{u},\ t) \tag{2.1}$$

This is known as ”Control Allocation”, and a plurality of methods have been developed, tested and used (Johansen and Fossen, 2013, Härkegård, 2002).

2.1.2. Flight Control Software in Commercial Aircraft

The software onboard commercial aircraft is commonly assessed for airworthiness using the DO-178 guidelines[1]. Software the handles control allocation would likely be classified ”Level A” software in DO-178B terms, as failure of such a system ”prevent[s] continued safe flight and landing” (Brosgol and Comar, 2021). This has several important implications:

- It must be possible to formally verify the correctness of the code; this includes but is not limited to
  - an upper bound to the static worst-case execution timing must be known and within limits
  - fixed-point arithmetic may never overflow
- Testing must cover the source code according to the MC/DC coverage criterion
- Additional verification must be performed whenever object code does not directly trace back to source code (for instance when generating object code using an optimising compiler)

Certification, especially proving correct outputs within an execution time budget, poses a significant challenge to the use of complex, high-performing control allocation in future commercial aircraft (Cameron and Princen, 2000.

2.1.3. Flight Control Software in UAVs

Software in off-the-shelf UAVs and experimental or research vehicles is much less restricted. Usually, extensive flight tests suffice to convince of safety, reliability and performance. Limited size and costs restrict the available computational power greatly, which also affects the choice of algorithms used.

Popular automatic flight control solutions for most consumer UAVs have been using very simple forms of control allocation (see section 2.3.2) to reduce complexity and save computational time. The paparazzi UAV software project has incorporated more advanced forms of control allocation (see section 2.4.3) dealing with actuator saturation, based on Smeur et al. (2017), which however do not scale favourably with the number of actuators and are limited to around 10 to avoid taking up excessive amounts of processing time.

Much of the current literature in optimising control allocation (such as presented in Smeur et al. (2017)) only considers platforms with fewer actuators. However, as described above, many platforms of current or future VTOL UAVs feature more than ≈ 15 actuators, such that an investigation of the suitability of current methods to these platforms may be desired, as well as possible improvements to these methods to accomodate these larger suites.

2.1.4. Preliminary Research Questions and Aims

In the previous section, the premise is made that current methods are not equipped to deal with large actuator suites. This premise is to be verified by a literature review under the following preliminary research questions that have been derived from this premise. After the review of the literature, some of these research questions may be answered, need adjustment and new questions may be uncovered. Some of these refined research questions serve as the starting point of a TU Delft Master’s Thesis to be performed in the coming weeks. These questions and the associated research aim are shown in section 2.6.2.

[1] https://www.faa.gov/regulations_policies/advisory_circulars/index.cfm/go/document.information/documentID/1032046

1. What is the current academic state-of-the-art in UAV control allocation?
2. What requirements and benchmarks have been/can be established to accurately compare these methods for the given applications, also considering their typical hardware platforms?
   (a) Which requirements are hardest to achieve with current methods applied in UAVs?
   (b) Which requirements are hardest to achieve with any currently known control allocation method?
3. How can control allocation methods be improved in their performance with large actuator suites and low computational capabilities?
   (a) Which of these potential improvements may be addressed in the scope of a master's thesis at TU Delft's MAVLab?
   (b) Can the improvements be validated and demonstrated?

2.1.5. Organisation of this Review

This literature review first gives a brief overview of the most common Aircraft Control Schemes (chapter 2.2), which introduces where control allocation fits into the picture. Next, in chapter 2.3 several methods that have been used to solve the problem in the past are critically discussed and their applicability to UAVs / eVTOLs is argued (section 2.3.6).

The need for specialised types of real-time capable optimisation software is discovered in chapter 2.3 and the following chapter 2.4 is dedicated to the analysis of a plurality of published algorithms for this purpose, as well as their applicability to control allocation.

Chapter 2.5 comments on the use of comparative testing in literature and how it may be applied to further research in the field of UAV/eVTOL control allocation. Finally, conclusions and recommendations to starting a future Master's Thesis are given in chapter 2.6.

2.2. Aircraft Attitude Control Schemes

Before we analyse the different methods of control allocation, what follows is the context in which the need arises. Any aircraft autopilot system has a component to stabilise the crafts attitude or rotational rates at a given set-point, which may come from a higher-level controller to attempt to achieve a certain flight path or location.

Forms of Proportional-Integral-Derivative feedback control (section 2.2.1) are most prevalent in UAV and commercial aircraft, but better control in the presence of non-linearities can be achieved with Nonlinear Dynamics Inversion (NDI, section 2.2.2). To combat the difficulties in implementing NDI, incremental NDI (INDI, section 2.2.2) has been developed.

2.2.1. PID

Assuming a simple elevator-aileron-rudder aircraft, equipped with gyro sensors, a PI or PID loop may be used to regulate the control surface angles based on the error observed between the commanded rates and the measured rates (Deepa and Sudha, 2016). This lower level controller may be encapsulated in another feedback loop to regulate attitude based on an estimate of the current attitude.

Careful tuning and scheduling of the P, I and D gains are necessary to provide an adequate system response for all permissible combinations of configurations, airspeeds, altitudes and masses. An illustrative approach is documented in Willee, 2021 for the PX4 UAV autopilot system, which schedules feedback and feedforward gains with the dynamic pressure $\overline{q}$ and roll rate p to achieve uniform responses. If the aerodynamic model is accurate enough, stability and handling characteristics with PID control can be proven for a certain airframe, making certification possible.

In the above example, a separate controller can be used for each axis, since the actuators are decoupled and, for a large part, only provide a control moment around one axis. When there are more actuators or axis-coupled actuators, an additional step is needed to use those actuators correctly. In simple cases like elevons or a v-tail, this can be done by simply adding the elevator command to the aileron command or the rudder command to the elevator command, respectively. Such a static allocation will later be known as explicit ganging (2.3.2). However, how should this mixing be performed for multi-rotors where each motor affects all axes? In these cases, the feedback controllers are often giving a more abstract angular acceleration command known as the pseudo control vector ν, which is then *allocated* over the available actuators.

In some cases, PID controllers can control mildly non-linear systems by gain scheduling, especially when those non-linearities have large time constants (airspeed/mass/altitude commonly change slowly), but other methods have more explicit and exact capabilities to resolve non-linearities.

2.2.2. Nonlinear Dynamic Inversion Control (NDI)

NDI control was first applied to aircraft by Snell et al., 1992, who investigated supermaneuverable fighter aircraft. These platforms are characterised by high non-linearities, especially at high angles of attack. The basic principle of NDI is to derive a non-linear transformation from a system model, which links newly introduced pseudo controls ν (usually the desired time-derivatives of the output vector y) to actuator commands. Neglecting actuator dynamics, the transformed system can be treated as a decoupled linear system, because ν is directly linked to output derivatives. Simple PI or PID control can be applied to generate ν commands with only one set of gains which achieves the same linear response regardless of system state.

The following section is based on Sieberling et al., 2010, which gives a good introduction and derivation of both NDI and its incremental form.

Derivation of classic NDI

General control-affine non-linear systems are given by equation 2.2, where $x \in \mathbb{R}^m$ denotes the system state, $y \in \mathbb{R}^p$ (OR $\mathbb{R}^d$?) denotes the outputs that are desired to be steered towards a reference, $f(x)$ is the system evolution function, $h(x)$ the output function, $G(x)$ is the state-depended input matrix and $u \in \ltimes$ is the control input.

$$\begin{aligned} \dot{x} &= f(x) + G(x)u \\ y &= h(x) \end{aligned} \tag{2.2}$$

If the system has no internal dynamics, the inputs can be related directly to the output derivatives by simply differentiating y with respect to time. For some elements of y the inputs u may not "show up" after one derivation. In these cases, y should be derived at least once more until an algebraic relation between elements of u and a time derivative of y_i is found $\forall i \in p$. From here onward, we shall assume such a relation is found after just one differentiation for all outputs of the system. Equation 2.3 arises, where we introduce the Lie derivative $\mathcal{L}_F a(x) \equiv \nabla a(x) F(x)$ of a vector field $a(x)$ over a vector field $F(x)$.

$$\begin{aligned} \dot{y} &= \frac{\mathrm{d}h(x)}{\mathrm{d}t} = \nabla h(x)\dot{x} \\ &= \nabla h(x)(f(x) + G(x)u) = \nabla h(x) f(x) + \nabla h(x) G(x) u \\ &= \mathcal{L}_f h(x) + \mathcal{L}_G h(x) u \end{aligned} \tag{2.3}$$

As mentioned before, we shall assume that $\mathcal{L}_G h(x)$ has non-zero items in all rows. Even stronger, we shall assume it is square (implying $p = n$) and invertible for all x. Then we can invert and decouple the dynamics by assuming $\dot{y} = \nu$ is our new pseudo control input to the system which can be generated with a linear controller (Sieberling et al., 2010) and/or a reference model (Matamoros, 2017); both are aimed at prescribing some desired, linear, system dynamics that outer-loop controllers can more easily deal with.

Indeed, under the conditions above, the output derivatives are achieved with the inverted dynamics given in equation 2.4:

$$\begin{aligned} u &= \left[\mathcal{L}_G h(x)\right]^{-1} \left(\dot{y} - \mathcal{L}_f h(x)\right) \\ &= \left[\mathcal{L}_G h(x)\right]^{-1} \left(\nu - \mathcal{L}_f h(x)\right) \end{aligned} \tag{2.4}$$

Some remarks on this result: The accuracy of the computed control input depends on the accuracy of the system model $\mathcal{L}_f h(x)$ and the input effectiveness derivatives $\mathcal{L}_G h(x)$. Sieberling et al., 2010 shows how, for the case of aircraft rotational rate control, these matrices correspond to the aerodynamic moments generated by the airframe and the control effectors, respectively. However, these models tend to be costly to establish accurately and model mismatch can strongly influence the control performance (Snell et al., 1992).

A second drawback is the requirement to have as many inputs as controlled outputs; the need for allocation arises naturally when $p < n$ and the matrix $\mathcal{L}_G h(x) \in n \times p$, making the matrix equation 2.4 underdetermined.

Incremental Nonlinear Dynamic Inversion Control

Sieberling et al., 2010 and da Costa et al., 2003 present an variant of NDI that does not compute the entire actuator state u, but rather increments Δu that are to be added to the current state u.

da Costa et al., 2003 motivates this with the ability to deal with non-affine non-linear actuators, but Sieberling et al., 2010 shows how the model of the aerodynamic airframe moment can be neglected to first order and replaced with an measurement of the current achieved pseudo control $\nu_a = \dot{y} = \dot{\omega}$, ie the rotational accelerations in the case of aircraft inner-loop control. It should be noted that the assumption that enables this is the time-scale separation principle that states that when considering a small time-step in the flight control computer, the change in aerodynamic moments in response to a change in the system state are negligible compared to the change in moments arising from changing actuator states.

The incremental NDI (INDI) linearizing control law is shown in equation 2.5 (Sieberling et al., 2010, Eq 27), applied to aircraft where ω and $\dot{\omega}$ are the rotational rates and accelerations, respectively. $\overline{J}$ is the inertia matrix and M_c denotes the actuator effectiveness model matrix.

$$\dot{\omega} = \dot{\omega}_0 + \frac{\partial}{\partial u} \left[\overline{J} M_c\right]_{\omega_0, u_0} (\Delta u) \tag{2.5}$$

This alleviates the need to identify an aerodynamic model of the full platform; only moments that the actuators give rise to need to be estimated. However, as mentioned, the current values of ω_0 and $\dot{\omega}_0$ need to be estimated. Rotational accelerations are difficult to obtain, as rotational gyros would need to be differentiated. Different solutions have been found to address this issue, such as optimal filtering (Sieberling et al., 2010), incorporation of sensor processing low-pass filters (Smeur, 2018) into the controller to synchronise the control and measurement signal, as well as maintaining online estimates of actual actuator positions and using pseudo control hedging to account for unachieved pseudo control due to actuator delay (Matamoros, 2017).

Further extensions (Smeur, 2018) using adaptive implementations further reduce the need for accurate à priori knowledge of the actuator effectiveness, which is especially important when the effectiveness may change during flight (eg failures, different regimes of the flight envelope).

2.3. Control Allocation

Control Allocation generally defines finding suitable actuator commands for a given system state and pseudocommand. In aircraft control, as mentioned in section 2.2.2, these pseudo-controls $\vec{\nu}$ are commonly taken as the 3 rotational torques, or amended with 1-3 linear forces. Equation 2.1 is repeated below, where we specify the ”achieved” pseudo-controls using ν_a:

$$\vec{\nu_a} = B(\vec{x},\ \vec{u},\ t) \tag{2.1 revisited}$$

Methods can be classified based on a few criteria:

- Linear vs Nonlinear: Is B assumed linear or nonlinear in u?
- Static vs Dynamic: Is time dependency (for instance due to actuator dynamics) considered in the method?
- Redistributing: On exceeding the limits of an actuator, does the method compensate by adjusting the allocation accordingly?
- Optimising: Does the method work by numerically minimising an objective function?

In general, these attributes are relatively independent and any combination is possible in principle. To guide the reader, linear static methods are discussed first as they provide building blocks to some of the more advanced non-linear and/or dynamic methods.

2.3.1. Problem Analysis

Many authors attempt to provide intuition about the control allocation problem, by reducing it to a very simple and low-dimensional case. Some early literature (W. C. Durham, 1993; Glaze, 1998) describes the ”three-moment-problem”, in which the function B is time-independent, is not state-dependent, is linear, and the pseudo-controls ν_a are 3-dimensional (specific torques for pitch-roll-yaw): $\nu_a = Bu$.

Geometry of LTI Control Allocation

In that case, the problem can be visualised geometrically via the Attainable Moment Set $\Phi \subset \mathbb{R}^3$ (AMS) and its boundary $\partial(\Phi)$. The effectiveness matrix B provides a mapping from the admissible actuator control set $\Omega \subset \mathbb{R}^n$ to the AMS Φ. While the boundary of the control set $\partial(\Omega)$ is usually simply a box (since actuator limits are often independent), the boundary $\partial(\Phi)$ of the AMS is a convex polyhedron, which is only a box if each n actuators affect the rotation around a single axis only. Figure 2.1 (W. Durham et al., 2017, Figure 5.8) shows an example of such a polyhedron for an aircraft where some actuators affect multiple axes, leading to a general convex shape.

In cases where the attitude control system request a pseudo-control vector ($\nu = \begin{pmatrix}\dot{p} & \dot{q} & \dot{r}\end{pmatrix}$ in figure 2.1) that is inside the AMS, a *feasible* pseudo-control, the control allocation algorithm should return a control distribution $u \in \Omega$ such that $\nu = Bu$. On the other hand, when ν is outside the AMS, an *infeasible* pseudo-control, then the control allocation should return a feasible control distribution $u \in \Omega$, such that $Bu \in \partial(\Phi)$, ie the resulting achieved pseudo-control is on the boundary of the AMS.

It should be noted that simple one-shot allocation methods (such as using the pseudo-inverse of B with simple clipping to the boundary of $\partial(\Omega)$, see forward section 2.3.2), can neither fully exploit the AMS in feasible cases, nor achieve pseudo-controls on $\partial(\Phi)$ in infeasible cases (W. Durham et al., 2017).

Augmentations of the Control Allocation Problem

When the platform is over-actuated, the solution $u \in \Omega$ is generally not unique to each ν. A secondary wish to find a feasible control distribution u with the least control effort may be necessary to better define the problem.

In many vehicles, the AMS boundary changes over time, with vehicle state and/or actuator states. Furthermore, multiple projections of infeasible pseudo-controls onto $\partial(\Phi)$ are possible and for certain vehicles preserving some pseudo-control elements over others is preferred. In mathematical terms, there may be some requirements on the error $\nu - Bu$, when $\nu \notin \Phi$ leading to $\nu \neq Bu$. Some vehicles allow direct control of specific forces in some directions (vertical thrust in all types of copters; vehicles with thrust vectoring), which increases the dimension of ν and the AMS Φ.

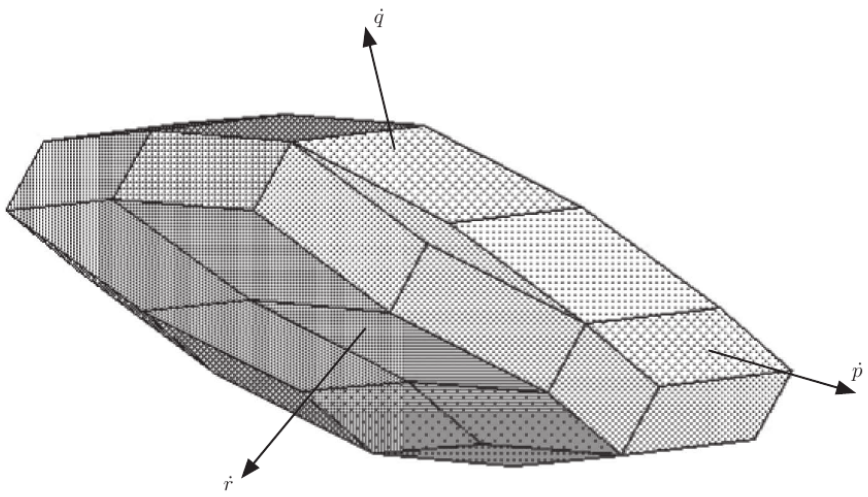


Figure 2.1: AMS of a Delta-Canard High Performance aircraft at low speed. Taken from W. Durham et al., 2017, Figure 5.8

When the platform features non-linear actuators with changing effectiveness over their range or due to usage of other actuators, the equation $\nu_a = Bu$ is not linear anymore and B must be seen as a non-linear function as in equation 2.1 (thrust vectoring changing the axis of thrust, see Blaha, 2021; upstream aerodynamic actuators changing the characteristics of downstream actuators as shown in Matamoros, 2017).

Finally, the time-dependent characteristics of the actuators may be considered in the choice of u, especially when some of the available actuators have different time constants. This is known as dynamic control allocation.

2.3.2. Linear Non-redistributing Allocation

The distribution procedures in this section are characterised by their simplicity, both during analysis and implementation. However, they fail to achieve a reachable pseudo-control as soon as any actuator saturates. This can strongly reduce the allocation-attainable moment set.

Ganging

The simplest form of control allocation would be to use an explicit heuristic map of the form $u = G\nu$ where G is an $n \times d$ matrixBodson and Frost, 2011. A classic example would be the control of a classical elevator-aileron-rudder aircraft with ailerons on both sides, making $n = 4$. If the actuators are assumed axis-decoupled and only affecting pitch, roll and yaw, respectively, the following ”ganging matrix” arises:

$$u = G\nu = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \nu \tag{2.6}$$

This scheme is static, linear, non-optimising and generally only used for actuator suites that can be grouped well to d groups with an explicit mapping. If a component of u exceeds its limit then it is simply clipped at that limit without redistribution. However, if the actuators are indeed axis-decoupled, then no form of redistribution on saturation will ever improve the situation. Grouping of non-identical actuators seems possible, but would likely lose performance, since they may saturate at different values, have different effectiveness or have different penalties associated with their use (eg drag, power).

Weighted Generalised Inverse

Assuming that the function $B(x, u, t)$ in equation 2.1 is linear in u and time-independent implies we can see it as a matrix-vector product $B(x)u$, where the effectiveness $B(x)$ depends on the state only. We

furthermore assume that the preferred state of all actuators in u is 0 and formulate the allocation problem as a constrained optimisation problem 2.7 whose solution reduces the square sum of control actuator deflections. Different norms in the objective function are possible and have practical significance, as we uncover later (see section 2.3.3, but the compact solution presented in this section is only possible with $p = 2$. Also, this norm will ensure that a solution of equal-weighted distances from the preferred actuator state is chosen over one with single "outliers", as a $p = 1$-norm would.

$$\begin{aligned} \min_{u} \quad & ||W_u^{\frac{1}{p}} u||_p \\ \text{s.t.} \quad & \nu = B(x)u \end{aligned} \tag{2.7}$$

Here, W_u is a diagonal, positive definite weighting matrix that enables penalising the use of certain actuators over others. As long as the equality constraint is consistent, the unique solution is given as $u^* = B^+_{W_u} \nu$, where we use a weighted generalized inverse $B^+_{W_u} \equiv W_u^{-1} B^T (B W_u^{-1} B^T)^{-1}$, see Eldén, 1982. Uniqueness arises from the fact that the objective is strictly convex (by our choice of a positive definite weighting matrix), so it is also strictly convex for any vector in the nullspace of the constraint matrix $B(x)$. For the case that $W_u = I$, the inverse reduces to the Moore-Penrose Pseudoinverse and is also referred to as the minimum-norm solution of an underdetermined linear problem.

In some cases the preferred actuator position is not 0 for all actuators, but rather the vector u_p; problem 2.8 arises.

$$\begin{aligned} \min_{u} \quad & ||W_u^{\frac{1}{p}} (u - u_p)||_p \\ \text{s.t.} \quad & \nu = B(x)u \end{aligned} \tag{2.8}$$

However, if we maintain $p = 2$, this only slightly complicates the calculation and the solution is given by the intuitive relation $u^* = u_p + B^+_{W_u}(\nu - B u_p)$.

This is an optimising control allocation scheme, but it is still linear, static and non-redistributing. In practical applications, the state dependency of the effectiveness matrix B is often neglected (for example for quadcopters), such that the generalized inverse $B^+_{W_u}$ and the preferred pseudo-control Bu_p can be pre-computed. In that case, every allocation takes two additions and one matrix-vector product; $n^2 + n + d$ floating-point operations.

It should be noted, however, that this scheme does not consider actuator limits and does not compensate for any errors in the actually achieved pseudo-control. In practice, the can lead to problems with vehicles where control over some axes is more important than others. A prime example is the multicopter for which the control authority is often limited in yaw, but yaw is also the least important axes to control as lift and linear accelerations are governed by pitch and roll. When yaw saturates following a large command relative to the low authority, pitch and roll may be distorted which can cause upsets (Smeur et al., 2017).

2.3.3. Linear Redistributing Allocation

Generally, every actuator has inherent limits. Literature considers position limits $\underline{u_i} \leq u_i \leq \overline{u}_i$ as well as rate limits $\underline{\dot{u}_i} \leq \dot{u}_i \leq \overline{\dot{u}_i}$ (Bordignon, 1996, p. 158, Härkegård, 2002).

For systems with axis-decoupled actuators (where each actuator only influences one pseudo-control), there is no merit in redistribution. We can establish the limit values for each pseudo-control directly from $B(x)$ and the AMS is a box in $\mathbb{R}^d$ on which we can project an unreachable pseudo-control to get the closest approximation without compromising the other pseudo-controls.

While this may hold approximately true for conventional elevator-aileron-rudder-thrust aircraft, cross-coupling in some or all axes exists for multirotors, helicopters and VTOLs, which implies that the AMS is not a box anymore and projection of a pseudo-control vector is not guaranteed to be the closest approximation. Furthermore, achieving some components of the pseudo-control may be more important to ensuring safe flight than others. All of this is to say that we require criteria that describes the desirability of an achievable pseudo-control on the interior or boundary of the AMS for every infeasible pseudo-control. The strategy should work for any AMS spanned by a linear effectiveness matrix.

Direct Control Allocation
One of the earliest control allocation methods to treat the problem with the geometric approach outlined in the introduction is direct control allocation (DCA), developed by W. C. Durham, 1993. The idea is to always retain the direction of the pseudo-control vector, which we shall call ”axis-preserving”. Equation 2.9 defines this idea mathematically (Bodson, 2002).

$$\begin{aligned} \max_{\rho, \vec{u}_1} \quad & \rho \\ \text{s.t.} \quad & B(x)u_1 = \rho\nu \\ & \underline{u} \leq u_1 \leq \overline{u} \\ & u = \frac{u_1}{\max(1,\ \rho)} \end{aligned} \tag{2.9}$$

Geometrically, this proposes that the most desirable feasible pseudo-control is always the intersection of the boundary of the AMS with a line from the origin to the commanded (infeasible) pseudo-control. It is not difficult to construct inputs or AMS boundaries that result in achieved pseudo-controls that are very small in magnitude even though closer (with regards to some norm) points on the boundary exist. The premise that axis-preservation is desirable is questionable, especially for vehicles where some axes are more important to stability than others, such as multicopters.

For the 3-moment problem with linear control effectiveness B, an efficient algorithm due to W. C. Durham, 2001 scales linearly with the number of actuators. However, the algorithm is very complex and does not guarantee convergence according to Bodson, 2002, who proposes a reformulation as a linear problem to be solved using the simplex method, which is fast in practice but shows very high theoretical upper bounds in the number iterations required (Hillier and Lieberman, 2009, sec. 4.9). If the AMS boundary can be precomputed offline and stored appropriately, finding the relevant AMS boundary facet and computing the intersection point can be done easily and efficiently (Petersen and Bodson, 1999).

This formulation is static, linear, optimising (for the maximum achievable axis-preserving control, not for some distance measure). A major benefit over all previously discussed schemes is that this method can fully exploit the entire AMS, however without prioritisation of axes or actuators and without the availability of algorithms with proven efficiency (unless the AMS boundaries can be pre-computed). Also, rate limits cannot be easily incorporated (Bodson, 2002).

Daisy-chaining
Daisy-chaining is similar to Ganging, but instead of scaling redundant actuators equally like in ganging, actuators can be prioritised. As an example, allocation is first performed using ganging matrix $G_1 \in n_1 \times d$ (where rank$G_1 = d$) and upon saturation of any element of u, the remaining actuators are invoked by allocating the residual ν according to $G_2 \in n_2 \times d$, Buffington and Enns (1996). Naturally, $n = n_1 + n_2$. Note that it is also possible to perform the two allocations using a generalised inverse, as suggested by Bordignon, 1996, p. 15, which is claimed to be superior to a plain generalized inverse allocation, but does not address the problem that some attainable pseudo-controls cannot be accurately solved.

An example of this could be using the spoilers on traditional transport aircraft only once the ailerons are fully deflected to achieve a certain roll command. This makes sure that the spoilers (with high lift and drag penalties) are only used when absolutely necessary. Like for Ganging, this requires that sensible heuristic choice separating G into G_1 and G_2 exists, which may not be the case for more intricate actuator suites that are not axis-decoupled. There is also no guarantee that the solution is optimal with respect to some criterion. It is a static, linear, non-optimising, but redistributing control scheme.

Redistributed Generalized Inverses
Instead of requiring a static partition of the actuators as for Daisy-chaining, one could consider creating the partitions online. A possible algorithm starts with the unconstrained general inverse solution of equation 2.7. Where actuator constraints are violated, those are fixed to either their upper or lower bounds yielding the first feasible iterate u_1 and residual $\nu_1 = \nu - Bu_1$. Then, the corresponding columns in the effectiveness matrix are set to zero, yielding B_2 and the difference to the commanded pseudo-control is allocated by solving $\nu_k = B_{k+1}\Delta u_{k+1}$, with the weighted generalized inverse of B_{k+1}. Then update $u_{k+1} = u_k + \Delta u_{k+1}$

and $\nu_{k+1} = \nu_k - B_k \Delta u$ and repeat until all actuators are fixed or the residual $\nu_k = 0$. This is one of the oldest schemes to incorporate limits into quadratic control allocation, see for instance Virnig and Bodden, 1994. Later we shall understand that this allocation scheme is better discussed as a possible solver for optimising control allocation and its analysis is deferred to section 2.4.3.

Heuristic Desaturation
Among consumer multirotor UAVs, generalized inverses (often simply pseudoinverses) are used to distribute pseudo-control-like commands from PID controllers (see section 2.2.1) over the actuators. Especially in drone racing, actuator limits are frequently encountered as maximum or minimum thrust is commanded to achieve maximum acceleration of the vehicle. To avoid losing angular control when all actuators are near or at their limit speeds, a number of heuristics are applied which are colloquially known as ”airmodes” (Willee and Grob, 2021). These aim to achieve roll, pitch and/or yaw commands by adjusting the overall thrust level (reducing when commanded thrust is high, increasing when commanded thrust is very low) up to some limit, thereby reducing the chance and impact of actuator saturation. This scheme is unlikely to lead to optimal solutions in any regard and is dependent on tuning the limits for permissible thrust reduction, which may be irrelevant for vehicles with large thrust-to-weight, but not for larger VTOLs that do not have that much thrust.

The industry-leading autopilot platform PX4 has very recently made advances to apply more flexible control allocation methods for fixed-wing and VTOL crafts to handle control surfaces, vectored actuators and rotors simultaneously [2]. Nevertheless, the allocation method is still only heuristic desaturation for multi-rotors or even just Pseudoinverse with clipping for other fixed-wing crafts.

Linear Optimising Control Allocation
With optimising control allocation, the problem is cast as a simple extension to the formulation used to derive the weighted generalised inverse allocation in section 2.3.2. However, the actuator position limits $\underline{u_i} \leq u_i \leq \overline{u_i}$ cannot be directly added as constraints to 2.8, since then the rank condition (rk$(B(x)) = n$) is not sufficient anymore to guarantee consistency of the constraints. Therefore, a natural formulation is a sequential minimisation problem, see equation 2.10. Intuitively, this asks to first establish the set K of all feasible controls that minimise the allocation error $B(x)u - \nu$ with respect to some norm weighted over the new pseudo-control-priority matrix W_ν, which we also assume diagonal and positive definite. From that set we then choose the control that minimises the control effort; the deviation from the preferred state u_p with respect to some weighted norm (Härkegård, 2002).

Geometrically, this obtains the closest pseudo-control (with respect to the chosen weighted norm p_ν and W_ν) on the boundary of the AMS and the corresponding minimum cost actuator control distribution $\vec{u}$ (minimum difference from the preferred control u_p with respect to the weighted norm p_u and W_u).

$$\begin{aligned} \min_{u \in K} \quad & ||W_u^{\frac{1}{p_u}}(u - u_p)||_{p_u} \\ K = & \arg\min_{u \in S} ||W_\nu^{\frac{1}{p_\nu}}(B(x)u - \nu)||_{p_\nu} \\ S = & \{u \mid \underline{u_i} \leq u_i \leq \overline{u_i} \quad \forall i = 1, 2, \ldots, n\} \end{aligned} \tag{2.10}$$

Solutions to the sequential problem Choosing $p = 2$ again, and assuming that $B(x)$ has full row rank and that $S = \mathbb{R}^n$, reduces equation 2.10 to 2.8 for which we have established the weighted generalised $B^+_{W_u}$ inverse as the unique solution. The redistributed generalised inverse method of section 2.4.3 can be seen as an approximate way to solve the full problem with actuator constraints (Härkegård, 2002).

Furthermore, Enns, 1998 has suggested replacing the rectangular constraint that determines the set S by a single elliptical approximation that contains the rectangle. When the solution obtained with the weighted generalised inverse is infeasible, it continues to find the closes point on the ellipse in the 2-norm sense, which appears to be solved quickly using bisection. This is only an approximation such that the solution may still exceed the actuator limits. According to (Bodson, 2002), the ellipse approach biases the

[2] https://github.com/PX4/PX4-Autopilot/pull/18776 Accessed 2022-01-13

solution towards the centre of the actuator limit, which is unfavourable for non-symmetric actuators such as propellers or spoilers.

Accurate direct solutions of 2.10 can be obtained by exhaustive search over all 3^n possibilities of actuator saturation behaviour (Bodson and Pohlchuck, 1998) and by a fixed-point algorithm due to Sabharwal and Potter, 1998, which is known to converge to the solution. However, both of these methods are slow in practice (Härkegård, 2002).

Weighted minimisation The problem 2.10 can be approximated well by forming a linear combination of the two objectives (reducing error and reducing actuator cost) as a single objective function that is to be minimised. The allocation error is then prioritised using a large scalar multiplier γ, establishing a hierarchy similar to the "Big M" for equality constraints in linear programming. For the original unconstrained problem 2.8 with $p = 2$, Björck, 1996, sec. 5.1.5. warns of ill-conditioning with too large values of γ, which has to be chosen as a tradeoff between arithmetical errors and approximation errors when this scalar is chosen too low.

The weighted minimisation formulation is given by equation 2.11 (Härkegård, 2002):

$$\begin{aligned} \min_{u} \quad & ||W_u^{\frac{1}{p_u}}(u - u_p)||_{p_u} + ||\gamma^{\frac{1}{p_\nu}} W_\nu^{\frac{1}{p_\nu}} (B(x)u - \nu)||_{p_\nu} \\ \text{s.t.} \quad & \underline{u}_i \leq u_i \leq \overline{u_i} \quad \forall i = 1, 2, \ldots, n \end{aligned} \tag{2.11}$$

Again, different choices of the norms are possible which lead to different solutions and different families of applicable numerical solvers, which are discussed further in chapter 2.4. According to Härkegård, 2002 and Bodson and Frost, 2011, real-time optimal solution of the weighted minimisation problem have become feasible with regards to onboard computational capabilities. However, Härkegård has only considered platforms with 8 actuators or less and Bodson, 2002 cites a runtime of 3.6ms or more for 16 actuators on a desktop processor, resulting in less than 300Hz update rate at full CPU usage for control allocation.

2.3.4. Dynamic Control Allocation

Any physical actuator is in itself a causal dynamical system which implies that any commanded deflection or rotational speed cannot be achieved instantly, and their rate of change is often limited as well. Even though hydraulic actuators are second-order systems, aircraft control design commonly treat them as rate-limited first-order systems with relatively low time constants (W. Durham et al., 2017, sec. 4.2).

Many authors suggest to exploit the digital nature of flight controllers to deal with actuator rate limits by restricting the actuator bounds to $\max(\underline{u}, u_{k-1} + \dot{u}_{\min}\Delta t) \leq u_k \leq \min(\overline{u}, u_{k-1} + \dot{u}_{\max}\Delta t)$, where Δt is the time-step and u_{k-1} is the value committed to the actuators at the previous time step (W. Durham et al., 2017, sec. 7.1, Bodson, 2002).

This however does not consider that some actuators may be better suited to high-frequency pseudo-control inputs than others. At the same time, some actuators may be preferred for low frequency or steady-state action (lower drag/energy consumption). Allocation schemes that handle this are desired to increase control bandwidth and reduce the steady-state allocation penalty.

Frequency Divided Control Allocation

A natural way to allow for this is to split the input signal into its high frequency and low-frequency components, allocate the two signals with different weights (W_{ul} and W_{uh}) to prioritise different actuators for high or low frequencies (Lallman et al., 2001). Then the resulting two control distribution is added to yield the final control $u = u_l + u_h$.

If using the unconstrained problem 2.7 with the 2-norm, the solution can be formulated as the linear combination of two differently weighted generalised inverses; the transfer function is shown in equation 2.12, where $L(s)$ denotes a linear low-pass filter

$$\frac{u(s)}{\nu(s)} = \left[B^+_{W_{ul}} L(s) + B^+_{W_{uh}} (1 - L(s))\right] \tag{2.12}$$

It is obvious that this scheme does not consider actuator limits at all and thus can be classified as linear, non-redistributing, dynamic, optimising.

Framewise Control Allocation and Dynamic Control Allocation due to Härkegård
An obvious way to adapt optimising control allocation to dynamic cases is to minimise an increment $\Delta u = u_k - u_{k-1}$, which will also find application in nonlinear allocation. One may consider the objective function $||W_u \Delta u|| + ||\gamma W_\nu (Bu - \nu)||$ to replace equation 2.11.

An important problem to solve when minimising increments is path dependency, which states that up-winding of the control distribution $u_{k+1} = u_k + \Delta u$ can occur, even when finding control solutions that achieve the desired pseudo-control ν with the smallest increments $||\Delta u||$. That is to say that if a steady-state flight condition is restored after a manoeuvre, the control distribution u may not be the same as before the manoeuvre (W. Durham et al., 2017, sec. 7.2). This is obviously undesired.

W. Durham et al., 2017, sec. 7.4.2 suggests to solve the problem of "Framewise Control Allocation" by finding and subtracting a term $u_\perp$ that is in the nullspace of $Bu_\perp = 0$ to reduce the norm of u. On the other hand, Härkegård, 2004 simply first solves an unconstrained minimisation problem $\min_{u_s} ||Bu_s - \nu||$ and uses the resulting u_s as the "desired" actuator state in the objective function $||W_{\Delta u} \Delta u|| + ||W_u (u - u_s)|| + ||\gamma W_\nu (Bu - \nu)||$.

Two decisive advantages result: First, the constrained optimisation problem can be solved with the same solvers as for any optimising control allocation problem 2.11. Second, u_s functions as the steady-state allocation that, over time, the system will converge to if the ν pseudo-command stays constant. The unconstrained optimisation can be performed offline with a weighted generalised inverse (see section 2.3.2), adding only $n*d$ operations to each iteration, allowing to prioritise the most energy-efficient actuators, and even allowing to factor in the actual desired actuator states u_p.

In the case illustrated in Härkegård, 2004, the Delta-Canard was designed to make use of the effective but inefficient canards only to improve the transient performance, without using them at all for steady-state while being much less prone to saturation than simple lead-lag pre-filtering of the canard signal.

Model Predictive Control Allocation
Model predictive control is an optimising control method that uses an internal model of the system dynamics and input effectiveness (often linear $x_{k+1} = Ax_k + Bu_k$) to maintain estimates of the future states in response to a finite series of inputs starting from the current time step. It steers the future state to some reference by minimising an objective function that includes state errors and control inputs over the finite horizon at every time step. Then the first element of the optimal control input series is submitted to the system and a measurement of the outputs is taken to observe and update the current states.

If no state or input constraints apply, the objective function is quadratic and the system dynamics are constant, then an infinite state horizon can be used and constant static state feedback gain matrix $u = Kx$ can be found offline. In that case, the scheme is coincident with LQR control. The benefit of online optimisation is thus to include varying system properties and crucially state and input constraints to be respected, at the cost of high online computational demand, especially if non-linear systems are considered. Another crucial disadvantage is that a dynamical system model must be known, which some aircraft control schemes like INDI explicitly avoid (Sieberling et al., 2010), citing the cost to establish a full aerodynamic model.

However, with Model Predictive Control Allocation the system to be controlled are the actuators themselves, which are often decoupled from the aerodynamic/kinematic aircraft model and more easily identified. The system is described as in equation 2.13 (Chatrath, 2019, sec. 3-5), where w is the actuator state (depending on the order of the actuator model, w may include derivatives of $u(t)$ or be simply $w = u$), $\{A, B, C\}_{\text{act}}$ are the matrices describing the linear actuator dynamics and δ is the input to be provided to the actuators.

$$\begin{aligned} w_{k+1} &= A_{\text{act}} w_k + B_{\text{act}} \delta_k \\ \nu_k &= B C_{\text{act}} w_k \end{aligned} \tag{2.13}$$

In the objective function of the MPC controller, the weighted square differences between the achieved pseudo-control $\nu_1, \nu_2, ..., \nu_{N_{\text{horizon}}}$ and the current reference value supplied from the higher level controller ν_{ref} is used. Also added are the weighted actuator deflections from the preferred state, similar to problem 2.8.

Recursively applying 2.13 for all time steps in the finite horizon of length N_{horizon} gives an expression for all ν_k depending only on the initial actuator state w_0, the effectiveness matrix B, the actuator dynamics A, C and the series of control inputs vectors $\delta_0, \delta_1, \ldots, \delta_{N-1}$. A quadratic objective function arises. However, in addition to problem 2.11, the constraints are not just bounds, but also linear constraints if w_k needs to be constrained which often results in large ($N \cdot n$) systems which can be solved with quadratic programming techniques similar to the ones used for problem 2.11.

2.3.5. Nonlinear Redistributing Control Allocation

Including general non-linearities (other than saturation) and coupling in the effectiveness of the actuators provide further challenges to actuation algorithms.

A natural approach that is found in early literature is simply expanding the problems of the previous section, equations 2.8, 2.9 or 2.11, to non-linear constraints or objective functions, but this requires more intricate and complex solvers. Another complication is that original problems are convex and thus only have one local and global optimum, but nonlinear functions can have multiple local optima.

Direct Nonlinear Control Allocation

This replaces the first linear constraint in 2.9 with a nonlinear equation $B(x, u_1)u_1 = \rho\nu$. This formulation has the same properties as DCA (axis-preserving, non-redistributing, static, optimising), but is much harder to solve (Matamoros, 2017, sec 2.3.1). Pre-computing the AMS is a possible solution (Doman and Sparks, 2002), but would then not allow online recalculation when parameters change, the actuators are dependent on the state (for example velocity) or actuators fail.

Optimising Nonlinear Control Allocation

The same approach can be followed for equation 2.11, resulting in a nonlinear program that may be nonconvex and generally slow to solve (Johansen et al., 2004). For the remainder of this section, we assume that all norms are $p = 2$, which seems prevalent in literature. Second-order solvers are preferred when the Hessian of $B(x, u)$ is cheap to compute, such as sequential quadratic programming or the Levenberg-Marquardt algorithm (Tol et al., 2014). Several simplifications to alleviate computational load based on linearisation are possible and outlined below.

Locally Affine Effector Model Typical real control actuators show decreased effectiveness near their limits (Oppenheimer et al., 2006). In cases where the effectiveness does not reverse, the actuator effectiveness may be approximated by a local multidimensional affine model around the current state x_0, u_0, see equation 2.14.

$$\begin{aligned} \nu_a &= b(x_0, u_0) + \left(\frac{\partial B}{\partial u}\right)_{x_0,u_0} \cdot (u - u_0) \\ \nu_a - b(x_0, u_0) + \left(\frac{\partial B}{\partial u}\right)_{x_0,u_0} \cdot u_0 &\equiv \tilde{\nu} \\ \tilde{\nu}_a &= \left(\frac{\partial B}{\partial u}\right)_{x_0,u_0} \cdot u \end{aligned} \tag{2.14}$$

Using the new model for the achieved pseudo-control $\tilde{\nu}_a$ and the local slopes $\left(\frac{\partial B}{\partial u}\right)_{x_0,u_0}$, the allocation u can be computed with standard linear methods as discussed in the previous chapters. For typical saturating actuator characteristics, this method produces more accurate results than a single linear fit of the actuator effectiveness matrix $B(x, u)$ (Oppenheimer et al., 2006), but that may not generalise to other actuator types. Sometimes, couplings between actuators are neglected and the local affine approximation

is done on each actuator separately (ie $\left(\frac{\partial B}{\partial u}\right)_{x_0,u_0}$ is diagonal, decoupling the equations; Oppenheimer et al., 2006, Matamoros, 2017, sec. 2.3.2), but the matrix algebra of equation 2.14 does not require this.

Higher accuracy may be obtained by not starting at just u_0, but at k starting values around u_0 and selecting the optimal $u^*_{0_k}$ that shows the lowest error $B(x_0, u^*_{0_k}) - \nu$ (Tol et al., 2014).

Piecewise Linear Effector Model The nonlinear functions describing the actuator effectiveness $B(x, u)$ may also be expressed as multidimensional piecewise linear functions that are derived à priori. Mixed-integer linear or quadratic programs can be formulated, using an integer or boolean to provide the solution algorithm with a degree of freedom to "select" the active region in the piecewise functions. "Branch-and-Bound" software to solve such problems is available and has been shown to accurately control nonlinear vehicles such as a simulated re-entry vehicle in the presence of failures (Bolender and Doman, 2004) and an underwater vehicle (Grechi and Caiti, 2016). However, in the first case, the timing requirements could not be met for real-time implementation and in the second case the vehicle had few actuators (7) and an update rate of only 10Hz, orders of magnitude lower than required for typical flight control.

However, in terms of the quality of the solution, the benefits of piecewise linear modelling have to be noted and provide large improvements over the affine modelling.

Sequential Affine Model An approach not mentioned in literature, but a natural extension of the approach that led to equation 2.14 is to treat the problem as an iterative procedure. $\tilde{\nu}$ and the linearisation $\left(\frac{\partial B}{\partial u}\right)_{x_0,u_0}$ is updated after each solution of the equation 2.14, by replacing u_0 with the new iterate u_k. This is essentially a sequential quadratic programming approach that may solve the problem exactly, if the global optimum is the only local optimum, or if the global optimum is at least a sufficiently close local optimum.

Incremental Nonlinear Control Allocation
Recall the local affine model, equation 2.14. If the upstream control system were to be incremental, providing the pseudo-control increment $\Delta\nu$, the equation 2.14 would simplify down to equation 2.15 dropping the need for knowing the intersect $b(x, u)$. This equation can be solved within the actuator constraints with the same methods as for the previous optimising control allocation schemes (Höppener, 2016, Equation (2), Matamoros, 2017, sec. 4.4).

$$\Delta\nu = \left(\frac{\partial B}{\partial u}\right)_{x_0,u_0} \cdot \Delta u \tag{2.15}$$

This idea was further developed by Matamoros, 2017 and successfully applied to the full ICE research aircraft. A similar idea was formulated in an internship report Blaha, 2021, sec. 4.6 taking inspiration from the "full INDI" module in the Paparazzi autopilot platform (Smeur, 2017, Höppener, 2016).

Analysis This scheme naturally handles the non-linear and coupled actuators, if an onboard jacobian model is used that includes these effects. The path dependency problem mentioned in section 2.3.4 has to be overcome by including an absolute desired state in the objective function of the optimisation (Matamoros, 2017, sec. 4.3) or using a version of dynamic control allocation (Härkegård, 2004) that should naturally translate to the fully incremental form 2.15. Originally, Matamoros, 2017 uses ℓ_2 norms, but other norms are possible, just as was the case for other optimising control allocation schemes.

Generally, the method does not find an allocation that fulfils the non-linear problem $\nu = B(x, u)$, which is in contrast to some of the non-linear methods mentioned in section 2.3.5. However, if the update frequency of the control system is high compared to the system and actuator dynamics, and the increment $\Delta\nu$ is not too large, then the changes in system state Δx, control distribution Δu and consequently of the effectiveness jacobian $\frac{\partial B(x,u)}{\partial u}$ are likely very small. Therefore, a solution to equation 2.15 is likely to be close to a local solution of the non-linear problem.

Advantages and Drawbacks Blaha, 2021, sec. 4.6.2 claims that $\frac{\partial B(x,u)}{\partial u}$ does not have to be re-computed at every time step if the actuator dynamics are much slower than the update rate of the Δu computations; in the case considered $20Hz$ vs $200Hz$. Depending on the choice of solver, this may significantly improve performance as also possible inverses of this matrix have to be computed less often. The opposite is also conceivable and the optimisation may be performed more than once per time step with updated $\frac{\partial B(x,u)}{\partial u}$, yielding a solver similar to the sequential affine model, discussed in section 2.3.5.

With highly coupled actuators, the scheme may struggle to find the global solution. This was encountered by Matamoros, 2017, sec. 5.2.4 and worked around by scheduling the actuator penalty weights of the affected actuator with domain knowledge to prefer their use in certain situations.

2.3.6. Requirement Generation and Method Candidate Selection

Bearing in mind the characteristics of UAVs and eVTOLs, a few requirements to control allocation can be derived that eliminate some of the above methods for our purposes.

The axis-decoupled assumption does not hold for the envisioned suites, especially for multicopters, where each rotor influences the lift force and at rotation about at least one axis. This rules out ganging and daisy-chaining, as they were enabled by these static maps.

Saturation handling can be very beneficial to increase flight envelope as we have seen that most of the non-redistributing schemes cannot reach every pseudo-control in the AMS. However, novel actuator design methodologies for new VTOL configurations with DEP include matching the AMS with a required moment set (Zhang et al., 2021). In that case, it would be instrumental to be able to actually achieve the AMS with the allocation strategy used. This rules out Generalised Inverse allocation and Frequency-Apportioned allocation.

Multicopters or eVTOLs in the multicopter-like flight phase would benefit from the ability to (de)prioritize some pseudo-controls over others to maintain safe and stable flight. It has been shown (Smeur et al., 2017) that prioritising roll/pitch/thrust over yaw improves the flying characteristics near the edge of the envelope. Therefore, direction control allocation and nonlinear DCA are recommended.

The envisioned actuator set may be large and diverse (propellers, surfaces, etc.), this means that daisy-chaining and other heuristic methods have to be carefully designed for every platform, vehicle or even flight phase. Other methods can deal with the non-linearities and dissimilarities better.

The computational load shall be low enough to enable the computation on small UAV platforms. Even for larger eVTOL aircraft, this may be a problem, as optimising compilers are problematic for commercial aviation, as the optimisations need to be formally verified. Optimising nonlinear control allocation is always slower than optimising linear CA, since the linear methods are often subroutines of the non-linear versions. We have seen indications in literature that these routines are too slow for the objective. The same holds for MPCA, which always results in much larger (albeit usually block-sparse) systems to be solved than linear CA.

Applicable control allocation schemes If the actuators are linear and not coupled, optimising control allocation can be used, provided the solver is accurate and real-time capable. Dynamic Control Allocation according to Härkegård can be considered if the actuators are linear and non coupled, but have significant differences in their dynamic behaviour. It relies on the same type of solver as optimising linear control allocation.

Incremental nonlinear control allocation (INCA) should be considered when actuators are non-linear and coupled and thus is the most powerful method shown above. Interestingly, it also uses the same solver type of solver as the optimisation problem at its core is the same. Any hardware that is capable of running linear optimising control allocation could also run INCA, provided that updating the effectiveness Jacobian is relatively cheap. However, recall that INCA can be assumed to be less precise than Optimising Nonlinear Control Allocation, but shown to work well in practice on a number of challenging platforms.

These methods are very flexible and should be able to control a large number of relevant platforms without the need for excessive platform-specific optimisation (like gain scheduling) if their numerical solvers are real-time capable for the number of actuators required. A closer look at such solvers is warranted.

2.4. Numerical Optimisation

In the previous chapter 2.3 we have seen the practical significance of efficiently solving constrained optimisation problems of the form encountered in 2.11. It finds application in many control allocation schemes (static and dynamic, linear and nonlinear) and together with a higher level control scheme such as INDI (see chapter 2.2) complex aircraft can be controlled.

$$\begin{aligned} \min_{u} \quad & ||W_u^{\frac{1}{p_u}}(u-u_p)||_{p_u} + ||\gamma^{\frac{1}{p_\nu}} W_\nu^{\frac{1}{p_\nu}} (B(x)u-\nu)||_{p_\nu} \\ \text{s.t.} \quad & \underline{u_i} \leq u_i \leq \overline{u_i} \quad \forall i = 1,2,\ldots,n \end{aligned} \tag{2.11 revisited}$$

In equation 2.11, we make the restriction that $W_u \in n \times n$ and $W_\nu \in d \times d$ be diagonal and positive definite, which is appropriate for comparative weighting matrices and will ensure the well-posedness of the presented algorithms. Also, B shall possess full row-rank and we shall denote the first term ”actuation penalty” and the second term ”pseudo control error”.

A crucial shortcoming of the current state of the art that has been identified is the lack of fast and precise solution methods that allow for control of a platform with a large number of actuators (≥ 15) without the need for artificial grouping, lowering the update rate or using large processing hardware.

2.4.1. Norms for the Optimisation Problem

In literature, three different norms have been proposed in the context of control allocation since the inception, ℓ_1, ℓ_2 and ℓ_∞ (Bodson and Pohlchuck, 1998, Pachter et al., 1995).

In general, ℓ_2 has the advantage that it provides a strictly convex objective function with a single optimal solution. The load is also naturally balanced between the actuators since large deviations in the distribution are avoided, even if the sum of the absolute errors of two possible distributions is equal (ie. their ℓ_1 norms are equal). However, quadratic solvers generally require more operations to find the unique solutions, whereas ℓ_1 and ℓ_∞ have the advantage that they can be formulated as a linear program that can be solved with fast methods such as the simplex algorithm.

In Bodson and Frost, 2011, the norms are mixed and ℓ_1 is applied to the pseudo control error while the supremum norm ℓ_∞ is used to balance the load between the actuators, addressing the problem of non-unique solutions with ℓ_1 norms for both terms. The resulting linear program can be solved with the simplex method, but has $2d+5n$ variables and $d+2n$ linear equality constraints, which is much larger than a comparable quadratic problem, but ”can still be solved very quickly on standard computing hardware” (Bodson and Frost, 2011). Using the supremum norm ℓ_∞ for both terms of equation 2.11 results in even bigger linear programs.

The question arises which combinations of norms are promising to return solutions that perform well for control allocation and also allow fast solvers.

Possible Combinations of Norms

From a control performance point of view, bearing in mind the knowledge gathered from the previous chapters, we can formulate the following expectations for the 3^2 possible combinations of norms. A basic premise is that ℓ_2 norms together with the restriction of positive definite weighting matrices ensure convexity, whereas ℓ_1 norms permit multiple distributions when $n > d$ and $Bu = \nu$ is feasible. For platforms with many identical actuators, this can result in maximal loading of $n-d$ actuators, while d actuators are at a lower level (Bodson, 2002) instead of balancing the load equally over all actuators, which is highly undesirable.

- Actuator Penalty: ℓ_1 – Pseudo control Error: ℓ_1

  Used frequently in literature (eg. Bodson, 2002 and Petersen and Bodson, 2005) and favoured for the availability of fast solution methods, but the problem does not have a unique optimum, especially when the allocation problem is feasible. This may lead to problems with certain solvers that first fully load some actuators while keeping others at their preferred state. According to Bodson, 2002, this may be regarded as an advantage, however for UAVs with many similar actuators it is highly undesirable to not use those similar actuators in similar ways.

- Actuator Penalty: ℓ_1 – Pseudo control Error: ℓ_2

  Not found in literature. The solution is still not unique, as with $\ell_1 - \ell_1$, but solvers are expected much slower since the problem is quadratic as opposed to linear.

  These two arguments combined lead to the conclusion to not further investigate the choice of norms.

- Actuator Penalty: ℓ_1 – Pseudo control Error: ℓ_∞

  Not found in literature and the same problems as $\ell_1 - \ell_2$ are expected, as it will lead to a larger system than $\ell_1 - \ell_1$.

  Therefore it is chosen to not further investigate the choice of norms.

- Actuator Penalty: ℓ_2 – Pseudo control Error: ℓ_2

  The classic bound-constrained least-squares problem. Used with success in many control allocation publications (Bodson, 2002; Härkegård, 2002; Smeur et al., 2017). It is a good candidate since its unconstrained version solves analytically to the weighted generalised inverse (see section 2.3.2) which has been shown to provide satisfactory solutions when actuator limits are not exceeded and is widely used in the drone industry.

  However, even though the problem formulation as constrained least-squares is known since the 1990s (see for example Virnig and Bodden, 1994), accurate numerical solution has only been considered computationally tractable with active-set algorithms (Härkegård, 2002) that even today are limited to not more than ≈ 15 actuators on commercial UAV hardware.

  Many solvers are available with different advantages and disadvantages.

- Actuator Penalty: ℓ_2 – Pseudo control Error: ℓ_1

  Not found in any research so far. Since ℓ_2 is used for the actuator penalty, convexity of the objective function is still maintained and a single unique solution would be returned from an accurate solver. It is unclear whether the use of ℓ_1 significantly reduces the quality of the solution in the context of control allocation, for instance, because of scaling issues between the dissimilar norms of the terms.

  Interior-point algorithms can be used to solve this problem.

- Actuator Penalty: ℓ_2 – Pseudo control Error: ℓ_∞

  Also not found in literature with equally uncertain advantages over $\ell_2 - \ell_1$. Additionally, reformulating ℓ_∞ norms for solution with conventional optimisation solvers always results in larger problems (Bodson and Frost, 2011).

  These two arguments combined lead to the conclusion to not further investigate the choice of norms.

- Actuator Penalty: ℓ_∞ – Pseudo control Error: ℓ_∞

  Minimising the supremum norm ℓ_∞ to balance control effort has been theorised in literature for use in control (Cadzow, 1971) and more specifically also control allocation (Bodson and Frost, 2011), where it has been reformulated as a linear program in $4n + 4d + 2$ variables and $2n + 3d$ linear equality constraints.

  This problem is thus larger and solution slower than $\ell_\infty - \ell_1$, whereas Bodson and Frost, 2011 note "no significant differences" when comparing the solutions $\ell_\infty - \ell_1$ and $\ell_\infty - \ell_\infty$.

  For these reasons, no further investigation for this choice of norms is performed.

- Actuator Penalty: ℓ_∞ – Pseudo control Error: ℓ_1

  Bodson and Frost, 2011 has used this combination of norms to address the issue that a simple ℓ_1 norm for the actuators does not incentivise balancing between similar actuators. By minimising the maximum deflection (also called min-max) a load-balancing action is achieved, at least between the heavily loaded actuators. It can be improved by weighting based on the actuator limits which returns identical solutions to Direct Control Allocation (equation 2.9) when the virtual control is feasible (Bodson and Frost, 2011).

- Actuator Penalty: ℓ_∞ – Pseudo control Error: ℓ_2

Not found in literature. Load-balancing can be achieved with the ℓ_∞ norm as seen above, but also with the ℓ_2 norm. Combining the two does not seem beneficial over $\ell_2 - \ell_2$, as quadratic solvers need to be used regardless of the ℓ_∞ norm which does not help improve the speed of the algorithm.

The remaining norm combinations are $\ell_1 - \ell_1$, $\ell_2 - \ell_2$, $\ell_2 - \ell_1$ and $\ell_\infty - \ell_1$. Numerical solvers for these norms are discussed next which focus on computational speed, accuracy and ease of implementation.

2.4.2. Solvers for $\ell_1 - \ell_1$ and $\ell_\infty - \ell_1$

In both cases, the problem is transferred to a standard linear programming problem of the form equation 2.16 (Bodson and Frost, 2011).

$$\begin{aligned} \min_x \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & 0 \leq x \leq h \end{aligned} \tag{2.16}$$

The decision vector x is not identical to the vector of actuator controls u; in both cases, the problem is larger to accommodate the positive and negative parts of the residuals in equation 2.11. This following paragraphs detailing this is a summary and discussion of the two important resources Bodson, 2002 and Bodson and Frost, 2011.

For the discussions below we need the definition of the function $s(x)$, equation 2.17.

$$s(x) \equiv \begin{cases} x \text{ if } x \geq 0 \\ 0 \text{ else} \end{cases} \tag{2.17}$$

Reformulation of $\ell_1 - \ell_1$ Consider the allocation error as $e = Bu - \nu$, the control allocation u, the preferred state u_p and the limits $\underline{u}$ and $\overline{u}$ with the requirement that $\underline{u} \leq u_p \leq \overline{u}$. The two terms in equation 2.11 can be split into the absolute values of their positive and negative parts using the transformations $u^+ = s(u - u_p)$, $u^- = s(-u + u_p)$ and $e^+ = s(e)$, $e^- = s(-e)$.

With these variables, the definition of an equivalent linear problem is possible: decision variable $x = \begin{pmatrix} u^+ & u^- & e^+ & e^- \end{pmatrix}^T$, objective function $c = \begin{pmatrix} \text{diag}(W_u) & \text{diag}(W_u) & \gamma\text{diag}(W_\nu) & \gamma\text{diag}(W_\nu) \end{pmatrix}^T$ and upper bound $h = \begin{pmatrix} \overline{u} - u_p & u_p - \underline{u} & Bu_p - \nu & Bu_p - \nu \end{pmatrix}^T$. The upper bounds on the error $e^\pm$ is not strictly necessary and therefore set to the a value that is always achievable, even when the most preferred actuator distribution is chosen.

The linear constraints $Ax = b$ need to implement the introduced variables $e^\pm$. Here we note that $u = u_p + u^+ - u^-$, $e = e^+ - e^-$ and consequently $e^+ - e^- = Bu_p + Bu^+ - Bu^- - \nu$. This is summarised see equation 2.18 (Bodson, 2002, eq. 43).

$$\begin{aligned} Ax &= b \\ \begin{pmatrix} -B & +B & +I_d & -I_d \end{pmatrix} x &= Bu_p - \nu \end{aligned} \tag{2.18}$$

Reformulation of $\ell_\infty - \ell_1$ Introduce the scalar variable $u^* > 0$ to the decision vector as an upper bound to the ℓ_∞ norm of the actuator penalty, such that $u^* = u^+ + \delta u^+$ and $u^* = u^- + \delta u^-$ with the slack variables $(\delta u^+, \delta u^-) > 0$. We redefine $x = \begin{pmatrix} u^+ & u^- & \delta u^+ & \delta u^- & u^* & e^+ & e^- \end{pmatrix}^T$ and include u^* in the objective function instead of the actual deflection: $c = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & \gamma\text{diag}(W_\nu) & \gamma\text{diag}(W_\nu) \end{pmatrix}^T$. The linear program solver will now decrease u^* as much as possible, until this upper bound on ℓ_∞ *becomes* the ℓ_∞ norm because it is bounded by any component of $\delta u^\pm$ hitting the lower limit 0.

Again, we introduce feasable upper bounds on the new variables, even though it would not be strictly necessary: $h = \begin{pmatrix} \overline{u} - u_p & u_p - \underline{u} & \overline{u} - u_p & u_p - \underline{u} & \overline{u^*} & Bu_p - \nu & Bu_p - \nu \end{pmatrix}^T$, where $\overline{u^*} = \max(|\overline{u} - u_p|, |u_p - \underline{u}|)$.

The newly introduced variables are implemented as $2n$ additional constraints, see equation 2.19 (Bodson and Frost, 2011, eq. 32).

$$Ax = b$$
$$\begin{pmatrix} -B & +B & 0 & 0 & 0 & +I_d & -I_d \\ 0 & 0 & I_n & 0 & -1 & I_n & 0 \\ 0 & 0 & 0 & I_n & -1 & 0 & I_n \end{pmatrix} x = \begin{pmatrix} Bu_p - \nu \\ 0 \\ 0 \end{pmatrix} \tag{2.19}$$

Simplex
A common class of methods to solve problems 2.16 is the simplex method, which traverses the facets of the feasible region until the facet that contains the constrained optimum is found. It is widely used and taught in operations research curricula and many software implementations exist (Hillier and Lieberman, 2009, sec. 4.1).

With application to control allocation, two main problems restrict the performance of the simplex algorithm. First, the method always maintains as many free ("basic") variables as there are constraints while all others are bound (Hillier and Lieberman, 2009, sec. 4.2). This implies that at the solution of an $\ell_1 - \ell_1$ formulation at most d variables in x are different from either 0 or their upper bounds h, which implies that at most d actuators are commanded to values other than their minimum, maximum or preferred state. Since the most advantageous actuator is used first until its saturation, this is somewhat reminiscent of daisy-chaining (see section 2.3.3) for its application to vehicles with many redundant axis-coupled actuators was already criticised. When there is only one optimum, the above is an intrinsic of linear programming in general, as mentioned by Bodson, 2002, but there can be cases where multiple adjacent corner points of the convex feasible region ("basic feasible solutions") have equally optimal objective function values. Without tie-breaking rules (Hillier and Lieberman, 2009, sec 4.5), any could be returned. This leads to problems in platforms with many similar and redundant actuators (eg. an octorotor) since it will load some motors more than others, even when no rotational acceleration is commanded.

Second, the simplex algorithm is fast in practice, especially on smaller problem sizes, but the theoretical upper bound on the iterations until convergence is exponential in the problem size (Hillier and Lieberman, 2009, sec. 4.9). This may provide obstacles when such an algorithm is used for certification, as a worst-case execution timing analysis is likely required (see section 2.1.2).

Interior-point
Algorithms of the interior-point class challenge the performance of the simplex algorithm, especially for large problems. Interior-point methods derive a Newton iteration scheme directly from qualified "Karush-Kuhn-Tucker" (KKT) conditions of problem 2.16 while limiting the iteration step size such that all iterates remain in the interior of the convex feasible region (Hillier and Lieberman, 2009, sec. 4.9). KKT conditions can be derived for quadratic objective functions as well, and the main derivations of the method for this more general case are outlined in appendix 3.1.

An advantage of the interior-point algorithms is that the iterations required until convergence are largely independent of problem size (as cited from Hillier and Lieberman, 2009, sec. 4.9, seen in practice eg. in Petersen and Bodson, 2006, Fig. 9). However, the cost per iteration is much higher than for the simplex algorithm as each iteration requires inversion of the jacobian 3.7. This implies that there for any type of problem there should be a problem size that forms a break-even point after which the interior-point method converges faster than the Simplex class. Owed to the special structure of the jacobian, the inversion should scale with $\mathcal{O}(n * d)$ which, in combination with the independence of iteration count after a certain n, would yield linear scaling of computational costs with actuator count n. However, an adaptation of the interior-point method been applied to $\ell_1 - \ell_1$ control allocation in Petersen and Bodson, 2005 has not reached the break-even point with $n = 11$ and $n = 16$.

A second advantage is that the method has an implicit way of dealing with the ambiguity mentioned for the Simplex algorithm. The centring parameter σ naturally chooses step directions that point towards the inside of the feasible region as much as possible. Preliminary experimentation with MATLAB®'s `linprog` has shown that when an entire edge or facet represents the optimal solution to the problem, the interior-point scheme is likely to converge to a point in the middle of this facet. However, methods of tuning σ for this purpose do not seem common in literature.

A severe disadvantage of the interior-point method is the stalling of the method (step size becomes very small) when an iterate is close to the boundary of the feasible region. The centring parameter prevents this during optimisation, but it implies that "warm-starting" the method from known close values (perhaps from a previous time step) is difficult. A possible approach is to maintain a set of active constraints that influence the path of the iterates the most and derive initial iterates from their combinations (δ-active set interior-point method, see Shahzad et al., 2010). For simplex algorithms, the choice of basic variables at the solution can simply be reused to start the iterations for the problem at the next time step.

2.4.3. Solvers for $\ell_2 - \ell_2$

Similar to the previous section on linear optimisation, problem 2.11 with ℓ_2 norms can be turned into a general quadratic programming 2.20 problem.

$$\begin{aligned} \min_u \quad & \frac{1}{2}u^T H u + \beta^T u \\ \text{s.t.} \quad & \underline{u_i} \leq u_i \leq \overline{u_i} \quad \forall i = 1, 2, \ldots, n \end{aligned} \tag{2.20}$$

As shown in equation 2.21, we use the definitions $H = \frac{1}{2}(W_u + B(x)^T \gamma W_\nu B(x))$ and $\beta = W_u u_p + \gamma B(x)^T W_\nu \nu$ and neglect the constant terms. By construction of H using the positive definite and diagonal weighting matrices, H remains positive definite and symmetric.

$$\begin{aligned} \min_u \quad & \frac{1}{2}||W_u^{\frac{1}{2}}(u - u_p)||_2^2 + \frac{1}{2}||\gamma^{\frac{1}{2}} W_\nu^{\frac{1}{2}} (B(x)u - \nu)||_2^2 \\ & = \frac{1}{2}(u - u_p)^T W_u (u - u_p) + \frac{1}{2}\gamma (B(x)u - \nu)^T W_\nu (B(x)u - \nu) \\ & = \frac{1}{2}u^T \left(W_u + B(x)^T \gamma W_\nu B(x)\right) u - u^T \left(W_u u_p + \gamma B(x)^T W_\nu \nu\right) + \frac{1}{2}u_p^T W_u u_p + \frac{1}{2}\nu^T \gamma W_\nu \nu \\ & = \frac{1}{2}u^T H u + u^T \beta + c \\ \text{s.t.} \quad & \underline{u_i} \leq u_i \leq \overline{u_i} \quad \forall i = 1, 2, \ldots, n \end{aligned} \tag{2.21}$$

Similar problems arise frequently in model predictive control and computational contact mechanics, both fields in which efficient algorithms are required. As such, many optimised method are described in literature (active-set Björck, 1996, sec 5.2. active-set with gradient projection Moré and Toraldo, 1989, conjugate gradient Vollebregt, 2014, interior-point Petersen and Bodson, 2005) and are discussed in the following.

Redistributed Generalised Inverse

As explained in section 2.3.3, the Redistributed Generalised Inverse allocation scheme can be seen as a solution procedure to the bound-constrained problem with the ℓ_2 norm. It relies on solving the unconstrained problem via a Generalised Inverse. When actuator constraints are violated, these actuators are fixed to their maximum or minimum values and the allocation is repeated with the remaining actuators and the remaining $\Delta\nu$ to allocate.

It is not guaranteed to find the optimal solution, or find an allocation for feasible pseudo controls within the AMS. Härkegård, 2002 has shown that the choice made in the RGI scheme to not permit "freeing" an actuator after it has saturated leads to inaccurate solutions. On top of that, it is non-smooth in the sense

that two arbitrarily close pseudo control input vectors can lead to largely different allocations and achieved pseudo controls.

Nevertheless, there is still considerable research interest in the method. It is quite fast, since it takes a maximum of $n - d$ iterations before an allocation is found and there is some indication that the quality of the solution can be significantly improved through improved bounding rules Jin, 2005, scaling and axis-preserving features similar to DCA (Zhang et al., 2018, Stephan and Fichter, 2017) that guarantee exact solution at least within the feasible domain.

Active-Set
Active-Set denotes a class of algorithms that follow a similar procedure to the Redistributed Generalised Inverse, but with the ability to free a constrained variable. As the name implies, it maintains a set of active constraints, turning some of the inequality constraints in 2.11 into equality constraints $Cu = d$ and neglecting all others that are not currently "active". This new and smaller "Equality-Constrained Least-Squares Problem" ("LSE" in Björck, 1996, p. 5.1.1.) can be solved efficiently using a variety of methods. After the exact "subspace minimisation" (Nocedal and Wright, 2006, p. 16.7), the active-set is updated by checking the constraints for feasibility and the Lagrange multipliers as a measure of optimality. This updated strategy is crucial to ensure finite convergence of the algorithm and in most cases, a simple rule is adopted that bounds the constraint that is violated first in the case of infeasibility, and that free the variable with the smallest Lagrange multiplier in case of non-optimality.

The active-set algorithm requires a feasible starting point which can be a non-trivial requirement for general linear inequality constraints, but for the bound-constraints at hand, we can simply use $u_0 = (\underline{u}+\overline{u})/2$. Härkegård, 2002 notes two decisive advantages with active-set algorithms for control allocation: (1) the solution u^* and corresponding active-set of the previous time step can be used as the starting point u_0 of the current time step and (2) each iterate is feasible and decreases the cost function which makes it possible to terminate the algorithm after a fixed number of iterations, expecting suboptimal results that are guaranteed to be better than the starting point. In the cases presented by Härkegård, 2002, applying (2) increased the maximum allocation error for infeasible controls, but reduced maximum computation time. Since this allows only a small number of changes in the working set, it is likely to induce a lag in the allocated control, the effect of which has not been characterised yet in closed-loop simulation.

It should be noted that while it is known that active-set terminates in finite steps, it is difficult to prove tight upper bounds on the number of active-set changes sufficient for convergence (Cimini and Bemporad, 2017).

Subspace Minimisation using QR decompositions Several methods are possible to solve the equality constrained subproblems. Björck, 1996 suggests the nullspace method for bound-constrained problems, which is outlined below.

1. Establish the equality constraint matrix A as the active set of the inequality constraints $Cx \leq g$

   Since we only have bound-constraints, this is simply a selection of the rows of $(I, -I)^T$.

2. QR decompose the equality constraint matrix A into Q_1, Q_2 and R_A. This also has trivial structure for bound-constraints.

   Now any vector that satisfies the equality constraints $Au = b$ can be represented as $u = u_1 + Q_2 y_2$ where u_1 is a particular solution of the equality constraints $u_1 = C^+ d = Q_1 R_A^T d$, by the properties of QR decomposition. This reduces the dimension of the least-squares system, since y_2 is of smaller dimension $n - n_k$ where n_k is the number of bound variables

3. Solve for y_2 using $(CQ_2)y_2 = g - Cu_1$ which can be solved via the QR decomposition of CQ_2.

   Most implementations (Härkegård, 2008, Höppener, 2016, Smeur et al., 2017) refactorise CQ_2 at every new iteration of the active set procedure.

Because of the simple structure of Q_1 and Q_2 which correspond to a selection of columns of C, it is advantageous to QR decompose C fully at the beginning (taking $1/3n^3$ operations) and then updating permutations of C where the first $n - n_k$ denote the free variables and the last n_k columns correspond to bound variables. When only a single variable is bound or freed each iteration, this can be done using efficiently in $2n^2$ operations using Givens rotations, with code available in LINPACK (Dongarra et al.,

1979, chapter 10) and is noted as a possibly significant improvement over existing control allocation implementations.

However, when many variables are already bound, the problem size becomes small and the updating permutations may not provide benefit over factorising the smaller matrix CQ_2 from scratch in $1/3(n-n_k)^3$ operations as opposed to updating in n^2 operations. Only representative experimentation will enable conclusions about this.

Subspace Minimisation using Conjugate Gradients The Conjugate Gradient (CG) method is an alternative to solving large linear problems $Ax = b$ if A is symmetric and positive definite. Unconstrained positive definite quadratic optimisation ($\min_u 1/2u^T Hu + \beta^T u$ can always be brought to this form, since we only require the derivatives to vanish: $0 = Hu + \beta \rightarrow Hu = -\beta$ (Nocedal and Wright, 2006, sec. 5.1).

The method relies on performing successive minimisation along linear search directions, which does not require any factorisation: $u_{k+1} = u_k - \frac{(Hu_k+\beta)p_k}{p_k^T H p_k} p_k$, where p_k is the chosen search direction and x_k are the iterates (Nocedal and Wright, 2006, eq. 5.6). It can be shown that if the first direction is simply the steepest descend $p_0 = -Hu_0 - \beta$ then an efficient, updating scheme can compute $n-1$ additional *conjugate* directions that lead to the solution in at most n steps (Nocedal and Wright, 2006, Theorem 5.3.), where n is the dimension of u_k.

Interestingly, when the k eigenvalues of H are identical, then the convergence completes in $n-k$ steps (Nocedal and Wright, 2006, Theorem 5.4). Preliminary experimentation seems to indicate that similar actuators (same thrust coefficients, symmetry, same distance components to the centre of gravity, same weights etc.) can be associated with identical eigenvalues in H. Since each iteration of CG takes $n^2 + 8n$ add/multiply and 2 division operations, this could reduce the total operations count for an unconstrained minimisation compared with a full QR solve ($1/3n^3 + 2n^2$) for platforms with many similar actuators and substitute the QR solver for the subspace minimisation of the active set algorithm.

Gradient Projection Method
Since projection onto the constraint set is computationally cheap for bound-constraints (only $2n$ comparison operations), specialised algorithms can exploit this fact. Gradient Projection Methods are said to more quickly converge to more quickly identify the set of active constraints at the solution by allowing more than 1 constraint per iteration (as for classical active-set) to be bound or freed (Björck, 1996, sec. 5.2.4.). The method can make use of subspace optimisations in between working set changes, but this is not strictly required and hence it is classified as a method separate from active-set in this document.

Gradient Projection essentially follows the steepest descend direction towards a local optimum on that line. When the constraint set is intersected along this line and before the local optimum, the path is *projected* onto that constraint and this is continued until a feasible local optimum is found or all variables are bound. In case a feasible local optimum along the path is found, it is called the "Cauchy Point" and refined by subspace optimisation. This is iterated by starting again with the direction of the steepest descent from the refined Cauchy point.

The basic steps, adapted from Nocedal and Wright, 2006, sec. 16.7. are shown below:

1. Define the projection operator $\mathcal{P}(u) = \text{median}(\underline{u}, u, \overline{u})$, where the median operator acts element-wise (Moré and Toraldo, 1989, eq 2.3).
2. Establish a starting point inside the interior of the feasible region. Set $k = 0$.

   If no better alternative, for instance from the solution of a previous time step, choose $u_0 = 0.5(\underline{u} + \overline{u})$).
3. Calculate the steepest decent vector $r_k = -Hu_k - \beta$. Set $p_0 = u_k$, $j = 0$
4. Perform a line minimisation along the *Projected Gradient* $p_j + t\left(\mathcal{P}(p + r_k) - p\right)$ the yielding the point u_k^j where $j = 0$.

   The positive definiteness of H implies that a minimum exists along any infinite line so an optimum is always found.
5. (a) If the optimum u_k^j is infeasible, set p_j to the first intersection of the constraint box with the the line $u_k + t(u_k^j - u_k)$, where t is a scalar. $j++$.
   If $j < n$, continue with 4. Else, set $u_{k+1} = p_j$, $k++$ and continue with 2.

   p_j is a feasible iterate with 1 more component at its maximum or minimum value than u_k.

(b) If the optimum u_k^j is feasible, the Cauchy Point is found. Proceed with a subspace optimisation over the current working set with j constraints, yielding u_k^*. Set $u_{k+1} = u_k^*$.
If $Hu_{k+1} + \beta == 0$ and the Lagrange Multipliers are ≥ 0 for all j active constraints, stop with solution u_{k+1}. Else, $k++$ and continue with 2.

The subspace optimisation can also be skipped, setting $u_{k+1} = u_k^j$ and returning directly to item 2., but would require more iterations.

Choice of Subspace Optimisation Method Just like for active-set, the subspace optimisation may be performed with any method, eg. nullspace method with QR factorisation or Conjugate Gradient. Since multiple variables are freed or bounded between subspace optimisations, it is possible that updating the QR factorisation does not improve the performance as much, since j updates with $2n^2$ computations have to be performed if j variables are bounded or freed at a given iteration and not only one.

Although no exclusive answer is given in literature, there are several reasons to prefer CG over the nullspace method. First, since subspace optimisation could be skipped entirely without affecting convergence guarantees, the problem does not need to be solved exactly. Therefore, it could be beneficial to only perform a few iterations of CG to improve the Cauchy Point, which is not possible with the direct solution of the nullspace method.

Second, if the subspace optimum is infeasible, then a new Projected Gradient iteration is necessary. Since CG generally makes rapid progress towards the subspace optimum in the first few iterations, infeasibility could be detected well before convergence (Nocedal and Wright, 2006, sec. 16.7). And if the subspace optimum *is* feasible, it is likely to be the global optimum or very close to the global optimum (after verifying the sign of the Lagrange multipliers). The same argument about similar eigenvalues for actuators suites with many similar actuators holds for this application of CG as well.

Third, the method is easier to implement, requiring no code for QR decomposition, only widely available and heavily optimised BLAS (Basic Linear Algebra Subroutines) level 1 and 2 functions and basic scalar arithmetic.

Use in other domains Derivatives of the method have been applied and optimised for computational contact mechanics (Vollebregt, 2014) in the software package CONTACT and for Model Predictive Control (Yu et al., 2021), however has never been applied to control allocation.

Interior-point
These algorithms are very close to the algorithms for linear problems presented in section 2.4.2. Petersen and Bodson, 2006 show a successful application to control allocation by introducing slack variables to the quadratic KKT system to derive an iteration scheme (see appendix 3.1, for a concise derivation).

The same argumentation for and against the interior-point algorithm can be continued here as well: The number of iterations increases only slowly with the problem size, such that a break-even point over active-set methods can be expected. For the ℓ_2 objectives, Petersen and Bodson, 2006 considered problems with very large numbers of actuators and found the break-even to be around $n = 15$. This behaviour makes the interior-point algorithm very attractive for general-purpose solvers, such that it is the default for many numerical software packages, The MathWorks, Inc., 2022 being a prime example. It should be noted, that $n = 15$ would be too ”late”, since naive and non-updating active-set already becomes intractable for $n \geq 15$.

The same downsides also hold: in contrast to active-set and gradient projection, the method cannot be easily restarted, which Petersen and Bodson, 2006 neglects to include in the comparison. How problematic this is in the context of control allocation is hard to estimate and is likely characterised best by representative closed-loop experiments.

Finally, each iteration requires inversion of an $n \times n$ matrix (called D in Petersen and Bodson, 2006) which is well-conditioned and positive definite and so can be factored using a Cholesky factorisation in $1/6n^3$ steps (Björck, 1996). However, D receives a full-rank update in every iteration of the interior-point as its diagonal terms change and thus needs to be refactored. This factorisation (as the only n^3 term) should quickly become the dominant contributor to the operations per iteration, but the plot of floating point operations over actuators (Petersen and Bodson, 2006, Fig. 10) seems quadratic in nature. The

authors also suggest using the Sherman-Morrison-Woodbury to perform an update of D, but this formula is constructed for rank-one updates or low-rank updates (Nocedal and Wright, 2006, App. A.1) which cannot represent full-rank diagonal additions. This is to say that the implementation is not entirely clear from the article and the results should be further researched/reproduced. Petersen contributed code to the `QCAT` project which can be examined to solve these doubts.

Fixed Point Solvers
The last category of solvers for quadratic programs are the fixed point iterative solvers that have been proved to converge to the global minimum (Burken et al., 2001) and are seemingly simple to implement. However, as found by Härkegård, 2002, Bodson, 2002 and Petersen and Bodson, 2006, tuning of the convergence parameter is difficult and so the solution may take many iterations depending on the pseudo control ν that is to be achieved. This unpredictability makes it generally unusable in practice.

Another recent approach is to formulate the iteration problem as a regulation problem on the residual and find a gain matrix K that enables finding a Lyapunov function to prove reduction of the pseudo control error residual after every iteration. This is dubbed ”control-theoretic iteration” in Cui et al., 2021, but mathematically congruent to fixed-point iterations. It is unclear whether the Control Lyapunov argumentation holds when the iterates are clipped, as is done in their algorithm. One variant presented is based on Conjugate Gradient scheme as above, however simple clipping iterates between iterations will not guarantee convergence to the global optimum, hence the introduction of active-sets that are carefully manipulated. Curiously, even when a pseudo control was feasible, their linear Conjugate Gradient scheme required more iterations to converge than the problem dimension, which should not be necessary by Nocedal and Wright, 2006, Theorem 5.4.

2.4.4. Solvers for ℓ_2-ℓ_1

This combination of norms requires adding a quadratic term to the linear objective while maintaining the bound-constraints and linear equality constraints. The problem 2.11 can be reformulated as shown in equation 2.22.

$$\begin{aligned} \min_u \quad & \frac{1}{2}||W_u^{\frac{1}{2}}(u-u_p)||_2^2 + |\gamma W_\nu \left(B(x)u-\nu\right)| \\ & = \frac{1}{2}(u-u_p)^T W_u (u-u_p) + |\gamma W_\nu \left(B(x)u-\nu\right)| \\ \text{s.t.} \quad & \underline{u_i} \leq u_i \leq \overline{u_i} \quad \forall i = 1,2,\ldots,n \end{aligned} \tag{2.22}$$

Now, to accommodate the ℓ_1 norm for the pseudo control error, we again split the allocation vector u into positive-only parts u^+ and u^- that denote the difference to the preferred state u_p. Also, we introduce the pseudo control error parts e^+ and e^- and we again use the decision vector $x = \begin{pmatrix} u^+ & u^- & e^+ & e^- \end{pmatrix}^T$. For a more rigorous discussion on these vectors, refer to section 2.4.2.

Note that with these definitions, the first term in equation 2.22 can be written $\frac{1}{2}(u^+ - u^-)^T W_u (u^+ - u^-)$ or equivalently $\frac{1}{2}u^{+T}W_u u^+ - u^{+T}W_u u^- + \frac{1}{2}u^{-T}W_u u^-$. Now, by construction, $u^+ = 0$ if $u^- > 0$ and vice versa, such that the product over a diagonal matrix $u^{+T}W_u u^- = 0$. Defining the augmented weighting matrix via $\text{diag}(\tilde{W}_u) = \begin{pmatrix} \text{diag}(W_u) & \text{diag}(W_u) & 0 & 0 \end{pmatrix}$ to match the definition of x, we can write the final form of the reformulated problem as equation 2.23:

$$\begin{aligned} \min_x \quad & \frac{1}{2}x^T \tilde{W}_u x + c^T x \\ \text{s.t.} \quad & Ax = b \\ & 0 \leq x \leq h \end{aligned} \tag{2.23}$$

The choices of A, b and h remains the same as in section 2.4.2, but the linear objective term for the actuator penalty is removed by setting $c = \begin{pmatrix} 0 & 0 & \gamma\text{diag}(W_\nu) & \gamma\text{diag}(W_\nu) \end{pmatrix}^T$.

The interior-point algorithm, as applied to the linear programming problems (section 2.4.2 and appendix 3.1) provides an interesting opportunity that may solve the ambiguity noticed in linear allocation problems with many similar actuators.

Interior-point
The modifications above seem complex, but result simply in the addition of a quadratic term to the objective function. As shown in appendix 3.1 this only adds $2n$ more divisions to the operations count per iteration, because of the fact that the Hessian $\tilde{W}_u$ is diagonal.

Despite the larger problem size than for an interior-point algorithm for the $\ell_2 - \ell_2$ norms (Petersen and Bodson, 2006), the largest matrix to be inverted is of dimension d and not n, whereas $d << n$ for highly over-actuated systems. It remains to be seen whether this addition requires the same or more iterations to converge to the solution, or if the break-even point with other solvers is only reached at a very high number of actuators as was the case for the quadratic interior-point. Nevertheless, it should be clear that this approach should outperform all ℓ_2 solvers in terms of asymptotical performance since the iterations complexity is bounded by $O(nd^2 + d^3)$ (presumably lowest for all iterative solvers) and the number of iterations for convergence is lowest for interior-point methods.

2.5. Comparative Testing and Benchmarks

Benchmark problems for implementations with the same or similar inputs/outputs can serve to (1) verify correctness (2) rank them for accuracy (3) rank them for computational speed. Since the goal of the thesis is to find the most suitable algorithms for UAVs with diverse effector suites, appropriate benchmark cases are beneficial to compare the candidate algorithms before the, presumably expensive, implementation on a few real-life platforms.

In literature, allocation algorithms have frequently been compared for accuracy and computational performance (e.g. Bodson, 2002; Härkegård, 2002; Jin, 2005; Sadien et al., 2019)

2.5.1. Testing Methodology in Literature

The goal of benchmark testing in the above works was to obtain estimates of accuracy and computational performance in a certain scenario. The common methodology of the testing can be summarised as follows: one (Härkegård, 2002; Sadien et al., 2019) or two (Bodson, 2002) effectiveness matrices B are chosen and a set of target pseudo controls was provided. Often, this target set is sectioned a priori or a posteriori into attainable and unattainable controls.

Closed-loop simulations are not performed and the discussion concludes with a comparison of error and mean/max convergence times without characterising the effect of the incurred inaccuracies or computational delay on the system under control. Furthermore, the chosen effectiveness matrices are either very specific to the applications of the authors (Sadien et al., 2019, Härkegård, 2002) or generally accepted and available benchmark systems, such as the Lockheed ICE (Bodson, 2002) and was not randomised or perturbed.

Testing Framework

In almost all studies MATLAB® has been used as the framework to implement the algorithms. However, MATLAB® was conceived as a framework for interacting with LINPACK and EISPACK routines (Moler, 2018) written in FORTRAN and also provides an interpreter to facilitate program flow control and data management. As such, the execution speed largely depends on the efficiency and the effective use of external routines and minimal use of the slow interpreted instructions. Some of these routines may benefit differently from vector instruction extensions common in desktop-x86 architectures (such as AVX2, Intel, 2014) to optimise the Basic Linear Algebra Subroutines (BLAS, used by MATLAB®, Moler, 2000).

It may therefore be unfair to compare the convergence times of two algorithms in MATLAB® that use different amounts of interpreted ”glue”-code and vectorised BLAS functions, that would not be present in a from-scratch interpretation in a system programming language such as C or C++ on microcontrollers common in UAVs.

An example is using `tic/toc` to compare the runtime of MATLAB®'s general function `quadprog` (The MathWorks, Inc., 2022) for solving quadratic optimisation problems with the purpose-built solver `QCAT` (Härkegård, 2008); `QCAT` ”outperforms” (Sadien et al., 2019) `quadprog`, but is the advantage with the algorithm or its implementation using less operations, or simply because of `quadprog`'s higher overhead?

Furthermore, Sadien et al., 2019 also notes that using `tic/toc` in the first place is inaccurate, as it includes processing time used for background activity on the CPU core.

2.5.2. Testing for the use-cases of this study

Already before thorough analysis, the algorithms discussed in chapter 2.4 seemingly have different weaknesses and strengths and may perform differently well depending on the properties of the effectiveness matrix, in cases when the solution is expected to be close to the solution of the previous time step, when the size of the problem is small or large or when there are different requirements on the accuracy of the solution.

Benchmark Generation

It seems indispensable to generate a number of benchmark problems that are representative of the many different configurations possible with diverse effector suites of UAVs and evaluate if the strengths or weaknesses of the algorithm align with relevant use-cases.

The requirements constrain the number of computational operations allowed for the generation of a control allocation vector.

Use of Testing Frameworks
Since MATLAB® provides a convenient high-level language with powerful functions that are commonly used in controls research, and considerable tools for control allocation are available for MATLAB®, such as `QCAT` (Härkegård, 2008), it makes sense to use it also for a future study to help understand the performance of the allocation algorithms.

However, its use shall be limited to the following objectives:

- Create reference implementations of the algorithms down to the LAPACK or BLAS level (only functions/operations present in LAPACK or BLAS may be used)

  This has several advantages: These implementations are easier to debug with the tools provided in the interpreted language and can later be traced line-by-line to a system-language implementation in C or C++. Furthermore, a framework such as Minka, 2021 can be used to count the number and types of computational operations that a non-optimising compiler would generate.
- Verify the correctness of the implementations against references (like `QCAT` and `quadprog`), if available
- Estimate the effect of inaccuracies in closed-loop simulations.

  If a dynamical model of the vehicle is also available, the allocation algorithm can be tested in a closed-loop scenario which also makes it possible to establish the true effect of inaccuracies by (for instance) truncating an iterative algorithm.
- Estimate the convergence time via the type and number of operations.

  The computational speed of the implementations will not be measured, to avoid the above pitfalls altogether and only operation count per iteration and the number of iterations required for convergence will be analysed. This is suggested by computer engineering literature outside of control allocation (see Lau et al., 2015, tab. 1) and also similar to the empirical approach used in W. C. Durham, 2001 to verify the scaling properties of the algorithm studied.

  The number of steps required for the iterative algorithms remains difficult to estimate for the warm-starting algorithms, as even closed-loop simulations may not include the effect of noise in the system state of actuator state measurements.

  The clock cycles of primitive operations (add, multiply, divide, square-root) for target platforms are available from manufacturer resources such as for the Cortex M4® architecture (STMicroelectronics, 2016), however, they should also be taken with care as pipelining makes predicting total execution time difficult for any full operation (including instruction decoding, loading data, executing the operation, saving data).

2.6. Conclusions and Future Work

In the previous chapters, we have seen how the need for control allocation arises naturally from attitude control schemes (chapter 2.2) and which formulations and solution approaches have been developed for the problem (chapter 2.3).

Requirements for UAVs Requirements for modern UAVs and (e)VTOLs were also established (section 2.3.6); now it is concluded that these requirements restrict the choice of applicable control allocation schemes from literature considerably. Only the Methods that require manual partitioning of actuators to certain axes (section 2.3.2, section 2.3.3) do not apply to multi-rotors. Coupled actuators are frequent (eg. tilting rotors), which rules out linear allocation methods (section 2.3.2). To exploit to Attainable Moment Set that is frequently used to design these crafts, a redistributing scheme is desired that deals with saturation (section 2.3.3). Furthermore, full aerodynamic characterisation of the crafts is often impractical, such that incremental control schemes (section 2.2.2) may be preferred.

It is concluded, that some of the most powerful and flexible control allocation schemes are available (dynamic control allocation, section 2.3.4; optimising non-linear control allocation, see section 2.3.5), and can indeed control effector suites of dissimilar, non-linear, coupled actuator suites with different dynamic responses with remarkably minor vehicle-specific tuning if the effector model is accurate (2.3.5).

It is difficult to quantify the computational load that may be appointed to control allocation and furthermore difficult to provide upper bounds to the execution time for most exact optimising control allocation schemes. Previous work has shown that execution time is indeed a practical problem (section 2.1.3) and UAV manufacturers compromise performance for low computational load (section 2.3.3) which indicates that this requirement is the most difficult to meet.

UAV autopilot state-of-the-art Curiously, the most popular UAV autopilot frameworks make use of Ganging (section 2.3.2) or simple pseudo-inverses (section 2.3.2), possibly improved with heuristic approaches to deal with actuators saturating (section 2.3.3). Optimising control allocation has only been found in one package and is mostly used for multi-rotor vehicles with few actuators. This points to a large difference between UAV control allocation and the academic state of the art. The reason may be that many consumer UAVs have high thrust-to-weight ratios, leaving sufficient overhead to make up for suboptimal control schemes. However, with the advance of VTOLs for unmanned flight, the current trend goes towards more sophisticated control allocation to control those dissimilar actuators more robustly (section 2.3.3).

2.6.1. Possible improvements for UAV and eVTOL control allocation

Central to most of the flexible schemes mentioned above is solving a constrained optimisation problem, whereas most authors seem to agree that non-linear optimisation (2.3.5) is not computationally tractable. Several simplifications based on linearisations (eg INCA, section 2.3.5) were found with the potential to provide adequate solutions for non-linear and coupled actuators, while only requiring the solution to least-norm optimisation problems where the terms are linear. This leaves the conclusion that a fast solution to equation 2.11 benefits all these allocation schemes.

Performance Indicators The performance indicators for the solvers of equation 2.11 are: (1) ability to balance the control effort over the actuator suite, (2) accuracy of the returned solution, (3) computational speed for a certain suite size n. It is not obvious from control allocation literature what an acceptable compromise between these indicators is. Closed-loop simulations and experimentation with real-world vehicles seem best to test the influence of (1) and (2) (section 2.5.2). The evaluation of computational load may be best performed by analysing operations per iteration and estimated the necessary iteration count from representative open-loop problems (section 2.5.2).

Current Performance and Research Trends From the available options presented in chapter 2.4, quadratic penalty functions with active-set solvers are considered state of the art for fulfilling (1) and (2) at tractable computational speed for small suites $n < 15$ (3) (section 2.4.3). Attempts to use interior-point algorithms have shown benefits over active-set for $n > 15$ (section 2.4.3), although comparison was problematic since the warm-start capability of the active-set algorithm was not used.

In the past, different choices of the norms in 2.11 have been made to attempt to sacrifice balancing (1) for speed (3) (section 2.4.2). However, even though steady-state allocation errors seem sufficiently small, it has not been conclusively shown with flight tests or closed-loop simulations how this affects the control distribution over time in a dynamic situation with saturation.

In early and recent literature, sacrificing accuracy (2) in exchange for speed (3) was considered. The iterative algorithms, such as active-set may be truncated after a fixed number of iterations, although this may result in a lag effect when changes are large which has not been quantified in dynamic closed-loop situations (section 2.4.3). Although rejected by early literature for their inability to return the exact solution and erratic control distributing, redistributed pseudo-inverse schemes have been improved in recent works (section 2.4.3) to make the solutions more stable and usable in control allocation.

Research Gaps

1. Surprisingly, active-set methods used in control allocation have not yet considered the use of updating factorisation to improve speed (3) and time complexity (section 2.4.3). This method would then be considered sufficiently optimised to serve as a baseline for comparing other solvers.
2. The combination of $\ell_2 - \ell_1$ norms in equation 2.11 seems to provide an appropriate compromise between speed (3) and balancing (1) (section 2.4.1) when solving with an interior-point algorithm (section 2.4.4), but is not found in literature.
3. The choice and design of solvers generally does not attempt to exploit certain special structures in the problem matrices to improve speed (3) while minimising inaccuracy (2). With many rotors pointing in the same direction and symmetry of the actuator suite, the eigenvalue structure of an $\ell_2 - \ell_2$ problem Hessian may be similar, of which a conjugate gradient based quadratic optimisation scheme can benefit (section 2.4.3). Despite the ease of implementation of conjugate gradient methods, this has never been performed in literature.
4. The method of Gradient Projection (section 2.4.3) has never been applied to control allocation despite its success in other comparable domains.
5. Comparative studies generally only consider one or very few similar use-cases. Especially for diverse platforms found in UAVs and eVTOL concepts, a comparative study with more test cases may be beneficial (chapter 2.5).

   Additionally, execution timings in recent control allocation literature can sometimes lack robustness when a desktop framework such as MATLAB® is used, with little attention to establishing an actual count of the arithmetic operations, even though that seems a common approach in other fields.
6. Closed-loop simulations or real-world experiments are rarely used to validate the balancing (1) behaviour or the effect of inaccurate solutions (2) (section 2.5.1).

2.6.2. Refined Research Questions

The introduction listed 3 preliminary research questions that were investigated with the literature review to be answered or refined. All but the extending subquestions of question 3. have been answered in the above conclusion.

The above research gaps constitute possible improvements over the current academic state-of-the-art in UAV and eVTOL control allocation. Since in principle methods exist that are powerful enough to deal with the challenging actuator suites of UAVs and eVTOLs, all of the gaps pertain to the improvement of the speed, accuracy and distribution balancing aspects of the optimisation procedure.

Research Objective ”Future research shall aim at finding relevant improvements of numerical optimisation algorithms for control allocation of UAVs and eVTOL, by implementing possible improvements over a baseline method and evaluating accuracy of the solutions, computational speed and balancing of the solution over the available actuators.”

Research Questions The different elements of the research objective have to be broken down and specified, to guide the research planning, the following Research Questions and subquestions have been formulated and partially answered.

1. What possible changes to the numerical algorithms for optimising control allocation can be shown to provide significant improvements over the baseline?
   (a) Can the speed of the active-set method for $\ell_2 - \ell_2$ norms be improved by iteratively updating its QR factors between iterations, using Given's rotations?
   (b) Can interior-point methods solve the $\ell_2 - \ell_1$ norm problem efficiently with accurate and balanced solutions?
   (c) Can the eigenvalue structure of the benchmark problems be exploited with Conjugate Gradient methods to make the active-set method for $\ell_2 - \ell_2$ norms more efficient?
   (d) Can the gradient projection method for the $\ell_2 - \ell_2$ norms improve speed?

Research Approach To answer the research questions with confidence, a number of secondary questions have been set up as pre-requisites to define the methodology. The more detailed project schedule will be based on these questions as well.

1. What performance parameters are relevant to UAVs and eVTOLs?

   As concluded above, the ability to balance control effort (1), the accuracy of the solutions (2), computational speed of the optimisation (3). However, a fair quantification and relative importance of these indicators for UAV/eVTOL flight performance are yet unclear.

   - How can the balancing action be quantified?
   - How can accuracy be quantified?

     Different norms of the allocation error $B(u^*, x_0, t) - \nu_a$ can be considered.
   - How can execution speed be quantified?

     As mentioned in section 2.5.2, floating-point operations count per iteration and number of iteration for convergence seem like a more robust option than empirical execution timing in MATLAB®.
   - How large is the budget of floating-point operations for one time step in the representative platforms (UAVs, eVTOLs)?
2. Which baseline method is appropriate to compare the improvements against?

   These improvements mentioned in the research question shall be compared to a baseline solver that is representative of the current state-of-the-art. Since the $\ell_2 - \ell_2$ choice of norms is known to balances the control distribution best, and the active-set method solves the problem accurately, the WLS solver described in Härkegård, 2002 and Smeur et al., 2017 will be used as the baseline.
3. How should the representative test problems be chosen?
   (a) Should the test suite consist of open-loop or closed-loop problems, or both?

   A major point of criticism to current methods of evaluating control allocation performance in this review has been the over-emphasis of speed (3) and (steady-state) accuracy (2). However, the effect of balancing the control effort (1) throughout dynamic manoeuvres can only be evaluated with closed-loop simulations and/or experimentation. Since these research items are more time consuming to execute, only a subset of the possible improvements should be included in this stage.

   (b) In either case, which control allocation formulation shall be used to derive the test problems?

   It has been established that INCA (section 2.3.5) is the most flexible control scheme that can be solved with linear-quadratic optimisation methods. It is flexible in the sense of supporting challenging platforms and also flexible enough to explore different update rates for the recomputation of the effectiveness matrix.

   Note that the optimisation problem at the heart of INCA, also forms the heart of methods in linear optimising control allocation and dynamic control allocation, which would all benefit from the prospective improvements.

(c) Which platforms should be used to generate the benchmark problems?

We have seen that literature usually only considers a few similar problems. To address the research objective, platforms should be representative of different types of UAVs and eVTOLs. A completely randomised platform can also give insights into the different strengths and weaknesses of the improvements.

(d) For closed-loop problems, what attitude controller will be used?

INCA implies the use of an incremental control scheme, so INDI (section 2.2.2) will be implemented.

3

Preliminary Work

During the Literature Study presented above (Chapter 2), some additional experimentation on developing the equations for an interior point solver to a specific constrained optimisation problem.

3.1. Interior Point Methods for $\ell_2 - \ell_1$ Norms

Notation in this chapter is standalone from the main document and self-contained. The present discussion is based on ideas from Nocedal and Wright, 2006, chapter 14 and The MathWorks, Inc., 2022, sec. Algorithms

A general minimisation problem be given as equation 3.1. It should be noted that for the case of non-zero lower bounds an equivalent problem of the form 3.1 can be derived.

$$\begin{aligned} \min_{u} \quad & \frac{1}{2}u^T H u + c^T u \\ \text{s.t.} \quad & Au = b \\ & 0 \leq u \leq h \end{aligned} \tag{3.1}$$

Slack variables s are introduced to convert the upper bounds to an equality constraints. The standard form equation 3.2 arises:

$$\begin{aligned} \min_{u} \quad & \frac{1}{2}u^T H u + c^T u \\ \text{s.t.} \quad & Au = b \\ & u + s = h \\ & 0 \leq u \\ & 0 \leq s \end{aligned} \tag{3.2}$$

The Karush-Kuhn-Tucker condition for the system can be derived via the Lagragian 3.3, where the vectors λ and η represent the lagrangian multipliers associated with the equality constraint and the upper bound constraint, respectively.

$$\mathcal{L}(u, \lambda, \eta) = \quad \frac{1}{2}u^T H u + c^T u - \lambda^T (Au - b) - \eta^T (u + s - h) \tag{3.3}$$

Any feasible and locally optimal solution satisfies the constraints and that the derivative of $dL/du = 0$. In addition to that, for every element u_i in u, either the associated slack variable s_i or the Lagrangian multiplier η_i must be 0. Intuitively, this corresponds to requiring that either a variable is at its positive limit $s_i = 0$ and cannot be increased without violating feasibility or the variable is at its unconstrained optimum and consequently $\eta_i = 0$. This can be formulated using the non-linear complementarity measure $\mu_i \equiv s_i \eta_i$.

The system of non-linear equations 3.4 (the "Karush-Kuhn-Tucker" conditions for the problem) is found.

$$\begin{aligned} Hu + c - A^T\lambda - \eta &= 0 \\ Au &= b \\ u + s &= h \\ s_i\eta_i &= 0 \end{aligned} \tag{3.4}$$

Using the definitions $S = \text{diag}(s)$ and $N = \text{diag}(\eta)$, the system 3.4 can be written in matrix form, see equation 3.5.

$$\begin{aligned} & & & & & & & & F\xi - \beta = 0 \\ \begin{pmatrix} H & -A^T & 0 & I \\ A & 0 & 0 & 0 \\ I & 0 & I & 0 \\ 0 & 0 & \frac{1}{2}N & \frac{1}{2}S \end{pmatrix} & \begin{pmatrix} u \\ \lambda \\ s \\ \eta \end{pmatrix} & - & \begin{pmatrix} -c \\ b \\ h \\ 0 \end{pmatrix} & = 0 \end{aligned} \tag{3.5}$$

By taking the jacobian of the matrix F, we can formulate a newton iteration step. For this, the definitions of the residuals (equation 3.6) simplify notation. We use the complementarity measure $\mu = s\eta$ defined above and the centring parameter $\sigma \in [0, 1)$ that steers the solution towards the central path to ensure the iterates only get close to the boundary of the constraints when approaching the solution (Nocedal and Wright, 2006, p. 396). $\vec{1}$ is a column vector of ones.

$$\begin{aligned} r_d &\equiv -A^T\lambda + \eta + c \\ r_p &\equiv Au - b \\ r_u &\equiv u + s - h \\ r_s &\equiv Ns + \sigma\mu\vec{1} \\ r &\equiv \begin{pmatrix} r_d & r_p & r_u & r_s \end{pmatrix}^T \end{aligned} \tag{3.6}$$

The newton iteration can now be written as $\Delta\xi = -J_F^{-1}r$. The iteration equation is summarised in equation 3.7.

$$\begin{aligned} & & & & & & & & J_F\Delta\xi = -r \\ \begin{pmatrix} H & -A^T & 0 & I \\ A & 0 & 0 & 0 \\ I & 0 & I & 0 \\ 0 & 0 & N & S \end{pmatrix} & \begin{pmatrix} \Delta u \\ \Delta\lambda \\ \Delta s \\ \Delta\eta \end{pmatrix} & = & \begin{pmatrix} -r_d \\ -r_p \\ -r_u \\ -r_s \end{pmatrix} \end{aligned} \tag{3.7}$$

Instead of directly factorising J_F, we notice the special structure and instead perform successive block row operations on equation 3.7 to eliminate variables. Update relations 3.8 are found (compare The MathWorks, Inc., 2022, sec. Algorithms):

$$\begin{aligned} (A\Gamma^{-1}DA^T)\Delta\lambda &= (A\Gamma^{-1})(Dr_d + r_u - N^{-1}r_s) - r_p \\ \Delta u &= \Gamma(DA^T\Delta\lambda - Dr_d - r_u + N^{-1}r_s) \\ \Delta s &= -r_u - \Delta u \\ \Delta\eta &= D^{-1}(-N^{-1} - \Delta s) \\ \text{where} & \\ D \equiv N^{-1}S, & \qquad \Gamma \equiv I + DH \end{aligned} \tag{3.8}$$

When $H = 0$ (no quadratic term in objective), then the first diagonal block element reduces to the identity and many of the inversions are trivial. However, since we required H to be diagonal, the inverses of N, D and Γ can always be computed in m divisions (m is the dimension of the state vector u), owed to their diagonal, positive-definite nature. The matrix $(A\Gamma^{-1}DA^T)$ is symmetric and of the same dimension as b (which will correspond to the pseudocommand ν when applied to control allocation, so only 3 to 6!). It can be solved in $1/6\text{dim}(b)^3 + 2\text{dim}(b)^2$ floating point operations using, for example, a Cholesky factorisation (Nocedal and Wright, 2006).

After computing the newton step, additional measures should be taken to make sure the step remains well within the interior of the feasible region and the problem remains numerically stable (see eg. Petersen and Bodson, 2005). Then the delta can be added to the current iterate and the iteration resumed.

Extensions like the predictor corrector algorithm that aim at reducing the effects of the linearisation of the complementarity measures are also possible before committing the update.

Part II

Scientific Article

Computationally Efficient Control Allocation Using Active-Set Algorithms

T.M. Blaha *, E.J.J. Smeur †, B.D.W. Remes ‡

* Master Student – MAVLab, TU Delft, Netherlands; t.m.blaha@student.tudelft.nl
† Assistant Professor – MAVLab, TU Delft, Netherlands; e.j.j.smeur@tudelft.nl
‡ Project Manager – MAVLab, TU Delft, Netherlands; b.d.w.remes@tudelft.nl

Abstract: An effective distribution of flight control commands over many aircraft actuators (engines, control surfaces, flaps, etc.) can be achieved with constrained optimisation. Active-Set methods solve these problems efficiently, but their computational time requirements are still prohibitive for aircraft with many actuators or slower digital flight control processors. This work shows how these methods can be improved in these regards, by updating the required matrix factorisations at lower computational costs, rather than solving a separate optimisation problem at every step of the iterative algorithm. Additionally, it is shown how the sparsity of the problem matrices can be exploited. Both open-loop simulations and flight tests have been performed, which show that worst-case timings for a 6-rotor multicopter UAV can be improved by 65% over a current Active-Set solver. Furthermore, methods are presented that remedy numerical stability issues occurring in micro-controller floating point arithmetic but introduce a small but measurable adverse effect on the flight behaviour.

Keywords: Control Allocation; Quadratic Optimisation; Active-Set algorithm

1. Introduction

In the context of aircraft and unmanned aerial vehicles, the flight control system comprises a set of controllers that ultimately send commands to a suite of actuators (propellers, aerodynamic control surfaces, etc.), to achieve a desired flight path and/or attitude. Oftentimes, cascaded control systems are used that compare references to estimated position, velocity, attitude and angular rates and output a set of desired forces and moments $\nu \in \mathbb{R}^d$ around the vehicle body. The number of degrees of freedom d depends on the location, orientation and types of actuators. For example, multicopters typically have all rotors axes aligned and can thus provide control around all three moment axes and one thrust axis ($d = 4$), such that tilting of the platform is used for position control. In general $3 \leq d \leq 6$.

The control allocation problem now poses that the desired "pseudo-controls" ν have to be allocated over the n available actuators. In other words, an actuator state vector $u \in \mathbb{R}^n$ must be found, which gives rise to ν. Incremental attitude control schemes, such as INDI [1,2], command increments $\Delta\nu$ instead; they are popular due to their good disturbance rejection while only requiring knowledge of the characteristics of the actuators. If the relation between actuator increment $\Delta u = u - u_c$ (with u_c the current actuator state) and the arising pseudo-controls increment $\Delta\nu_a$ is linear, the actuator effectiveness matrix $B \in \mathbb{R}^{d\times n}$ can be introduced linking $\Delta\nu_a$ to Δu as

$$\Delta\nu_a = B\Delta u \,. \tag{1}$$

Because any physical actuator has limits $\underline{u} \leq u \leq \overline{u}$ and $\text{rank}(B)$ may be less than n (especially since on many platforms $d < n$), this equation is not easily inverted to find an explicit formulation for Δu. Many methods have been devised to find suitable inverses to Equation 1, among which ganging [3], direct control allocation [4,5], frequency-divided control allocation [6,7], and optimising control allocation [5,8].

Optimising control allocation can satisfy important properties [8,9]. It is (1) able to find a Δu^* such that $B\Delta u^* = \Delta\nu$, whenever the desired $\Delta\nu$ is attainable by the actuator suite. Additionally, if $\Delta\nu_a$ in Equation 1 is over-determined, the scheme finds (2) a vector Δu^*

that is well balanced to make efficient use of the available suite of actuators with respect to some measure. Whenever $\Delta\nu$ is *not* attainable, the optimising control allocation can return (3) a vector Δu^* that gives rise to an achieved $\Delta\nu_a$ that is as close as possible to the demanded $\Delta\nu$ by some distance measure. Quadratic distance measures are preferred for accuracy and uniqueness of the solution, but this comes with the drawback of requiring significant computational resources for online optimisation [8].

This study presents methods to further reduce these computational requirements, with little to no impact on the flight behaviour, by investigating the popular Active-Set solver and evaluating improvements in both open-loop simulations and flight tests.

Section 2 of this work shows the motivation for the problem formulation used and possible solvers, while Section 3 details the chosen Active-Set algorithm and the considered improvements. Methods for comparing the improvements are described in Section 4, after which the results and conclusions are presented in Sections 5 and 6, respectively.

2. Optimising Control Allocation Formulations and Solvers

The three requirements mentioned in the introduction can be realised with a mathematical description as follows. Minimise the pseudo-control error $\arg\min_{\Delta u}|B\Delta u - \Delta\nu|_{\ell_\nu}$ (fulfilling requirements (1) and (3)). Out of all minimisers $\Delta u_s \in S \subset \mathbb{R}^n$, select the one that minimises $|u - u_p|_{\ell_u}$, where u_p is some preferred state of the actuators that may be chosen freely. Different choices of the distance measures ℓ_ν and ℓ_u are possible (see Blaha [10] for a detailed comparison), but weighted 2-norms are commonly used for adequate load-balancing, convexity, and availability of efficient solvers.

This sequential least squares problem can be solved as posed, but may also be converted into the combined weighted least squares problem [5,9]

$$\begin{aligned} \arg\min_{\Delta u} \quad & ||W_\nu(B\Delta u - \Delta\nu)||_2 + ||\gamma W_u(\Delta u - \Delta u_p)||_2 \\ s.t. \quad & \underline{\Delta u_i} \leq \Delta u_i \leq \overline{\Delta u_i} \quad \forall i \in \{1, \ldots, n\}\,, \end{aligned} \tag{2}$$

which is treated in this study. This penalty function includes both objectives; the first term we call "pseudo-control error" and the second one "actuator usage penalty". The introduced weighing matrices W_ν and W_u are diagonal and positive definite and can be used to weigh the relative importance of the components of the error terms. $\Delta\nu$ is commanded from a higher level controller and all other Δ's are relative to the current actuator state u_c, such that $\Delta\square = \square - u_c$.

Due to the choice of the quadratic norm and the restriction that W_u is positive definite (not semi-definite) and therefore full-rank, Equation 2 has one unique solution for each choice of $\Delta\nu$. In the case where $\Delta\nu$ is achievable within the bounds for Δu, the first term vanishes and a norm of $\Delta u - \Delta u_p = u - u_p$ is minimised to ensure efficient use of the actuators. When $\Delta\nu$ is *not* feasible, then the actuator usage penalty should be of low priority, which is achieved by a low factor $\gamma > 0$.

To provide good separation between the two objectives, a low enough value γ must be chosen. This can lead to low condition numbers of H, which can amplify numerical inaccuracies from floating point calculations on that matrix, such as matrix inversions, factorisation, or additions.

2.1. Equivalent Formulations

The advantage of this combined-objective formulation is, that Equation 2 can also be written as a standard least squares optimisation problem $\arg\min_{\Delta u}||A\Delta u - b||_2$ [9,11], by using the definitions $A = \begin{pmatrix} W_\nu B \\ \gamma W_u \end{pmatrix}$ and $b = \begin{pmatrix} W_\nu \Delta\nu \\ \gamma W_u \Delta u_p \end{pmatrix}$. While vector b is dense, Matrix A has a dense part $A_d = W_\nu B$ and a diagonal part below:

$$A = \left(\begin{array}{cccc} \multicolumn{4}{c}{A_d = W_v B} \\ \hline \gamma W_{u_1} & 0 & \dots & 0 \\ 0 & \gamma W_{u_2} & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & \gamma W_{u_n} \end{array}\right) \tag{3}$$

The problem can also be developed into the quadratic minimisation problem

$$\begin{aligned} \arg\min_{\Delta u} \quad & \frac{1}{2}||A\Delta u - b||_2^2 = \frac{1}{2}(A\Delta u - b)^T(A\Delta u - b) = \frac{1}{2}\Delta u^T A^T A \Delta u - \Delta u^T A^T b + \frac{1}{2} b^T b \\ \arg\min_{\Delta u} \quad & \frac{1}{2}\Delta u^T (A^T A)\Delta u - \Delta u^T A^T b \\ \arg\min_{\Delta u} \quad & \frac{1}{2}\Delta u^T H \Delta u + \Delta u^T \beta \quad \text{where } H \equiv A^T A,\ \beta \equiv -A^T b \\ & s.t. \quad \underline{\Delta u_i} \leq \Delta u_i \leq \overline{\Delta u_i} \quad \forall i \in \{1, \dots, n\}\ . \end{aligned} \tag{4}$$

2.2. Available Solvers

A possible class of solvers for the above problems are Interior Point Methods [12,13], which may be very advantageous in case that ℓ_v is chosen as the 1-norm [10] or when the number of actuators is large [14].

A more popular choice are Active-Set algorithms, where at every iteration, an unconstrained optimisation problem is solved on the subspace given by a set of enabled constraints (the "working set"). Then checks are performed for feasibility and optimality, which either terminate the algorithm or adapt the working set until the "Active Set" is identified which describes the active constraints at the solution [11]. Using simple adaptation rules, this algorithm is guaranteed to converge in finite time [9] where the quality of the solution (ie the value of the cost function) monotonically decreases with the iterations. Additionally, re-using the solution of a previous time step is much simpler and more effective for Active-Set methods than for interior point methods [8,15], as the active set can be re-used as the initial working-set for the next time step. This is known as "warm-starting".

3. The Active-Set algorithm

At the solution to problem 2, all components of Δu are either "bound" at the lower or upper limit, or "free" in between those. Active-Set algorithms maintain a "working set" $\mathcal{W}$ to keep track of which variables are free or bound; and if bound, at which of the two limits. To find the solution to the constrained problem, Active-Set algorithms solve a sequence of unconstrained problems over the subspace given by the free variables $\{i | i \notin \mathcal{W}\}$.

When one or more components of the solution to such an unconstrained problem violate a constraint, a variable is fixed at either bound and "added" to the working-set $\mathcal{W}$. When no constraints are violated in a subspace optimisation, or there are no more free variables left ($\neg\mathcal{W} = \emptyset$), the Lagrange multipliers for all bounds $\{i | i \in \mathcal{W}\}$ are computed. If any are negative, then the current solution is not optimal and a variable is removed from $\mathcal{W}$. This continues until the *active* set is found whose unconstrained subspace solution is feasible and optimal (ie no constraints are violated and all Lagrange multipliers are positive).

Algorithm 1 shows this in detail, shows the modularity for the subspace solver and provides the nomenclature for later analyses. n_f is the number of free actuators and the array *permutation* holds the indices of the free actuators in the first n_f positions and the indices of the bound variables in the last $n - n_f$ positions. The set $\mathcal{W}$ is implemented as an array $\mathcal{W}_a$ that contains $\mathcal{W}_a[i] \leftarrow +1$ whenever variable i is bound at its upper limit $\overline{\Delta u_i}$,

Algorithm 1: Active-Set overview. All indexing is end-point inclusive and 1-based.

Data:
n of INT, d of INT, $B[1:d][1:n]$ of FLOAT, $u_c[1:n]$ of FLOAT, $\nu[1:d]$ of FLOAT, $\underline{\Delta u}[1:n]$ of FLOAT, $\overline{\Delta u}[1:n]$ of FLOAT, $W_\nu[1:d][1:d]$ of FLOAT, $W_u[1:n][1:n]$ of FLOAT, $\mathcal{W}_a[1:n]$ of INT
Result: $\Delta u^* = \Delta u$

1. Problem setup

/* get A, b from B, $\nu, W_{u,\nu}$ and switch to Δ's */
$\overline{\Delta u} \leftarrow \overline{u} - u_c$; $\underline{\Delta u} \leftarrow \underline{u} - u_c$; $\Delta u_p \leftarrow u_p - u_c$; $\Delta u_0 \leftarrow \frac{1}{2}\left(\underline{\Delta u} + \overline{\Delta u}\right)$;
$A \leftarrow \begin{pmatrix} W_\nu B \\ \gamma W_u \end{pmatrix}$; $b \leftarrow \begin{pmatrix} W_\nu \nu \\ \gamma W_u \Delta u_p \end{pmatrix}$; $\Delta u \leftarrow \Delta u_0$;
$n_f \leftarrow COUNT(\mathcal{W}_a[:] = 0)$; `/* number of free actuators */`
$permutation[1:n_f] \leftarrow$ all indices i of free actuators ($\mathcal{W}_a[i] = 0$);
$permutation[n_f+1:n] \leftarrow$ all indices i of bound actuators ($\mathcal{W}_a[i] \; != 0$);

2. Subspace setup.

call SUBSPACE.INIT ; `/* Do matrix factorisation if required. */`
while *True* **do**

3. Subspace solver.

/* unconstrained optimum on the subspace of the $n - n_f$ bound variables */
$z[1:n] \leftarrow$ call SUBSPACE.SOLVE;

4. Active-Set evaluation.

/* check feasibility/optimality of z and change $\mathcal{W}_a, n_f, permutation$ or break */
if $\underline{\Delta u}[i] \le z[i] \le \overline{\Delta u}[i]$ **then**
/* all constraints are satisfied → check optimality */
$\Delta u \leftarrow z$;
if $n_f = n$ **then**
break; `/* optimal, since no constraints were present */`
end

5. Compute Lagrange Multipliers

$\lambda[1:(n-n_f)] \leftarrow$ call SUBSPACE.LAGRANGE_MULTIPLIERS;

4. Active-Set evaluation, cont'd

if $MIN(\lambda[:]) \ge 0$ **then**
break; `/* removing constraint would cause infeasibility */`
end
/* not optimal yet → remove variable with the biggest potential for improvement from the working set */
$p_f \leftarrow n_f +$ ARGMIN$(\lambda[:])$; `/* Position in` *permutation* `of the variable with biggest potential for improvement */`
$\mathcal{W}_a[permutation[c_f]] \leftarrow 0$; `/* free the variable in working set */`
$RIGHT\_CIRCSHIFT\Big(permutation\Big[(n_f+1):p_f\Big]\Big)$;
$n_f \leftarrow n_f + 1$;

3. Subspace solver. cont'd

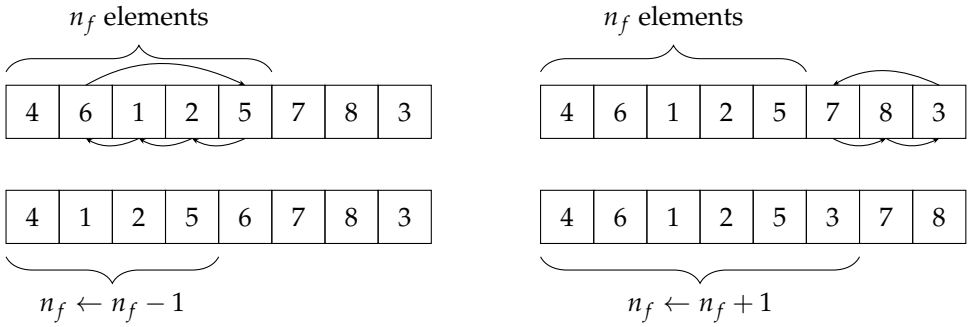
call SUBSPACE.FREE; `/* update factorisations if required */`
else
/* at least one constraint violated → add one actuator to working set */

4. Active-Set evaluation, cont'd

$\alpha \leftarrow$ largest scalar $\alpha \in [0,1]$ such that $\Delta u + \alpha(z - \Delta u)$ remains feasible;
$\Delta u \leftarrow \Delta u + \alpha(z - \Delta u)$;
$i_b \leftarrow$ index of a variable that would violate its bound at $\alpha + \epsilon$ (with $\epsilon > 0$);
$\mathcal{W}_a[i_b] \leftarrow$ +1 if $z[i_b] - u[i_b] > 0$, -1 otherwise;
$p_b \leftarrow$ position of i_b in $permutation$;
$LEFT\_CIRCSHIFT\Big(permutation\Big[p_b:n_f\Big]\Big)$;
$n_f \leftarrow n_f - 1$;

3. Subspace solver. cont'd

call SUBSPACE.BOUND; `/* update factorisations if required */`
end
end



(a) Bound the actuator in position 2. **(b)** Free the actuator in position 8.

Figure 1. Example of the changes in the *permutation* for freeing and bounding an actuator, represented by a partial circular and incrementing/decrementing n_f. Free actuators are on the left of the array, the bound ones are on the right.

$\mathcal{W}_a[i] \leftarrow -1$ when bound at the lower limit $\underline{\Delta u_i}$, and $\mathcal{W}_a[i] \leftarrow 0$ when it is free. Freeing and bounding an actuator i can be realised by a circular shift within *permutation* such that i is now at the n_f or $n_f + 1$ position, respectively, and then incrementing or decrementing n_f, as shown in Figure 1.

3.1. Subspace Solvers

Many methods are known to solve quadratic unconstrained optimisation, and any can be used to solve the subspace steps of algorithm 1 (denoted SUBSPACE.*). The baseline considered in this report is simply solving a completely separate optimisation problem, denoted "PPRZ" for its use in the Paparazzi Autopilot framework[1].

Using QR or Cholesky factorisations of A or $H \equiv A^T A \succ 0$, respectively, has advantages, since update routines exist to adapt the factors in case A or H only change by column permutation or appending/deleting a single row/column [11,16]. Finally, it can be shown that there may be only few distinct eigenvalues of H (Appendix A.4), Conjugate Gradient ("CG") may be advantageous for the unconstrained optimisation.

The working principles and the potential advantages of the considered algorithms are discussed in Appendix A and pseudo-code and implementation details are provided in appendix B.

4. Methodology

Three categories of improvements have been investigated in this study:

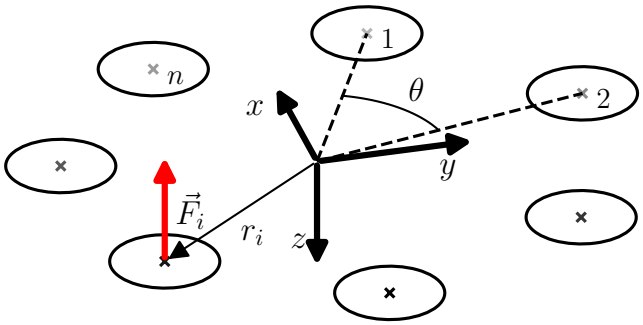
1. Subspace solvers
2. Exploiting the sparsity of the matrices
3. Warm-starting the algorithm with a previous solution

For the first two of these items, candidate algorithms and improvements have first been implemented in a high-level language (MATLAB®) for faster prototyping and to verify the implementation.

Based on this, a selection of algorithms are then implemented in C with the focus on minimum execution time. The implementation was verified against the MATLAB results which also makes it possible to compare the effect of the problems' condition numbers, 32 bit versus 64-bit floating point numbers, and complement the execution time estimations from the FLOPs counting.

Finally, three of these algorithms have been evaluated on multicopter test flights, to validate their suitability to the problem and evaluate possible adverse effects on flying characteristics. These test flights also make it possible to fairly evaluate the benefits of

[1] https://wiki.paparazziuav.org/



(a) Multicopter platform, nomenclature and body axis system.

(b) Nederdrone platform with 12 motors and 8 aerodynamic surfaces. Photo: TU Delft https://magazines.defensie.nl/allehens/2020/06/08_drones-op-waterstof

Figure 2. UAV platforms considered in the work.

warm-starting, which are highly dependent on how similar the conditions are at successive time steps.

4.1. High-Level Framework

Multicopters with n rotors and a randomised distance r_i have been generated as test platforms, see Figure 2a. Also, a 12-rotor winged eVTOL platform called the "Nederdrone" [17] (see Figure 2b) has been modelled, with and without considering its aerodynamic surfaces.

The locally linearised effectiveness matrix $\Delta\nu = B\Delta u$ is derived from the geometry of these platform under the assumption that $F_i = u_i F_{i,\,\max}$ for rotors and $\delta_i = u_i \delta_{i,\,\max}$ for rotating actuators, such as on the Nederdrone, where δ_i is the deflection angle of the i-th actuator.

4.1.1. Test Case Generation

The test cases on which the algorithms have been compared and verified consist of both feasible and infeasible choices of ν_{des}, where feasibility in the case implies that a choice $\underline{u} \leq u^* \leq \overline{u}$ exists, such that $\nu_{\text{des}} = Bu^*$.

To achieve a balanced generation of the desired ν_{des} set, the boundary of the Attainable Pseudo-control Set $\Phi \subset \mathbb{R}^d$ was approximated with a hyper-ellipsoid using the extreme points $B(e_i \odot \overline{\Delta u})$ and $B(e_i \odot \underline{\Delta u})$, where e_i are the standard unit vectors $e_i = \begin{pmatrix} 0_{1\times i-1} & 1 & 0_{1\times n-i} \end{pmatrix}^T$. The fitting algorithm presented in Kesaniemi and Virtanen [18], Fitzgibbon *et al.* [19] was applied. The direction of ν_{des} is then generated with uniform ellipsoidal sampling [20] and the magnitude is randomised with the expectation that 50% of the N test cases are within Φ and 50% are infeasible.

4.1.2. FLOPs Counting

For each operation in the high-level implementation, the floating point operations (FLOPs) were recorded, depending on the type of operation and the sizes of the vectors/-matrices, taking elements from [2]. Table 1 shows the operation count assumed for relevant operations.

4.2. Implementation in C

After verification in MATLAB, the relevant functions were re-implemented in C to make them usable on a real-time system such as a UAV flight controller. Additionally, modern MATLAB versions exclusively use 64-bit floating point ("double") arithmetics.

[2] https://github.com/tminka/lightspeed

Table 1. FLOPs per operation.

Operation	Input Sizes	FLOPs	Source
Scalar Addition $a + b$	$a \in \mathbb{R}, b \in \mathbb{R}$	1	[21]
Scalar Multiplication ab	$a \in \mathbb{R}, b \in \mathbb{R}$	1	[21]
Scalar Division a/b	$a \in \mathbb{R}, b \in \mathbb{R}$	14 on ARM32	[21]
Square Root $\sqrt{a}$	$a \in \mathbb{R}$	14 on ARM32	[21]
Dense multiplication AB	$A \in \mathbb{R}^{n \times m}$, $B \in \mathbb{R}^{m \times p}$	$np \cdot (2n - 1)$	From scalar operations
Solve system $y = Rx$	$R \in \mathbb{R}^{n \times n}$ triangular	$n^2 - n + 14n$	Counting from Forward/Backward substitution
QR factors $A = QR$ (Q as Householder factors $H_1 \dots H_n$)	$A \in \mathbb{R}^{n \times m}$	$2nm^2 - 2/3m^3$	[22]
Recover Q from Householder factors $H_1 \dots H_n$	$A \in \mathbb{R}^{n \times m}$	$2nm^2 - 2/3m^3$	[22]
Cholesky factors $H = LL^T$	$H \in \mathbb{R}^{n \times n} \succ 0$	$\frac{n^3 - n}{3} + n^2 + 2 \cdot 14n$	Counting from implementation

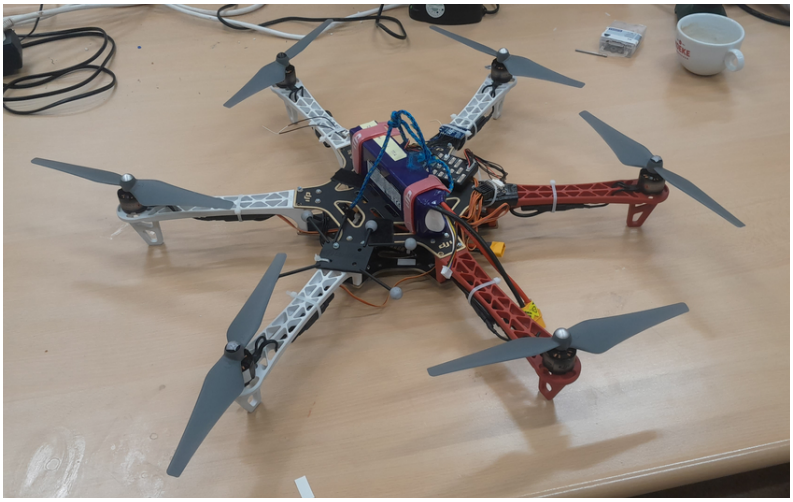


Figure 3. Hexacopter test platform with battery, Pixhawk flight controller and reflective infra-red markers.

However, many UAV flight controllers chips like the STM32 in the Pixhawk 4[3] flight controller, which only supports 32-bit floats ("singles") efficiently. Consequentially, it must be verified that the numerical stability of the algorithms is not affected by the switch from doubles to singles. The use of Cholesky factorisations has been criticised in this regard in literature [11,23].

4.3. Flight Tests

Indoor flight tests were conducted using the hexacopter platform shown in Figure 3, equipped with a Pixhawk[4] 4 running the Paparazzi autopilot framework, which itself runs on top of the Chibios real-time operating system[5]. Real-time position data from an IR camera positioning system was fed into the PID position control loop, which generates references to the INDI attitude controller, see Höppener [24]. The incremental pseudo-control commands $\Delta\nu$ from the INDI attitude controller are then used as input to Algorithm 1.

Chibios provides macros `chSysGetRealtimeCounterX()` and `RTC2US()`[6], which were used to obtain a time difference around Algorithm 1, with a resolution of `STM32_SYSCLK`, which is set to 216MHz. Values are rounded to μs. Note that significant bias and variance

[3] https://docs.px4.io/main/en/flight_controller/pixhawk4.html
[4] http://www.holybro.com/product/pixhawk-4/
[5] https://www.chibios.org/dokuwiki/doku.php
[6] https://chibios.sourceforge.net/docs3/rt/group__system.html#ga0fff5863f6cf801f414320b7fade3c85

Table 2. Overview of the algorithms.

Abbreviation	Description	MATLAB	C	Flight Test
PPRZ	The original control allocation algorithm used in the Paparazzi Autopilot [9].	yes, this study	yes, from [9]	yes
PPRZ-OPT	Like PPRZ, but optimised for sparsity.	yes, this study	no	no
QR	Active-Set with subspace solver based on updating QR factorisations.	yes, this study	yes, this study	yes
CHOL	Active-Set with subspace solver based on updating Cholesky factorisations.	yes, this study	yes, this study	yes
CG	Active-Set with Conjugate Gradient subspace solver.	yes, this study	yes, this study	no

Table 3. Verification of MATLAB implementations using the error $|\Delta u - \Delta u_{\text{PPRZ}}|_2$. 12-rotor multicopter.

	$\gamma = 8.8 \cdot 10^{-3}$		$\gamma = 88 \cdot 10^{-3}$		$\gamma = 884 \cdot 10^{-3}$	
	Max	**Mean**	**Max**	**Mean**	**Max**	**Mean**
PPRZ-OPT	5.835e-15	2.9683e-16	2.44e-15	2.6391e-16	3.2926e-15	1.794e-16
QR	3.3246e-11	1.0915e-12	5.2636e-13	1.5945e-14	9.0549e-14	5.6288e-15
CHOL	3.8767e-07	3.0245e-08	3.5664e-09	3.537e-10	4.5932e-11	1.2184e-11
Conj Grad	0.50445	0.00050445	3.2648e-09	1.1689e-10	2.7169e-09	3.4531e-11

are expected in these measurements, because the real-time operating system may interrupt the computations of Algorithm 1 to process higher-priority tasks.

The position reference was set as a step input, with a simultaneous 30 deg yaw angle change; for simplicity, repeatability and the ability to drive the actuators into saturation. The manoeuvre is shown in Figure 10. Nevertheless, the maximum RPMs of the platform have been artificially limited to ensure safe operation and that saturation occurs more quickly and for longer.

5. Results & Discussion

In total, five different algorithms are treated in this study, see Table 2. Their implementations and performances are discussed in the following sections, where Section 5.1 shows the high-level evaluation. Some algorithms are selected from these results, implemented in C (Section 5.2), and finally validated in a flight test (Section 5.3).

5.1. High-Level Verification and FLOPs Counting

Using MATLAB®, the 5 algorithms were compared to each other for different platforms and choices of the separation parameter γ. Table 3 shows the accuracy of the solutions as the 2-norm of the error $\Delta u - \Delta u_{\text{PPRZ}}$ over a test set of $N = 1000$ randomly chosen $\Delta\nu$ points, for a multicopter platform with $n = 12$ actuators. As a comparison, the machine precision for these computations was $\epsilon \approx 2.22 \cdot 10^{-16}$, so results are expected to never be below that and be only few orders of magnitude above that

It can be seen that the choice of γ may influence the accuracy of the results and that the conjugate gradient (CG) algorithm can become unstable at some choices of γ. Also, the Cholesky-based algorithm loses some accuracy for lower γ.

Figure 4 shows the mean FLOPs required per algorithm for a 12-rotor multicopter, split into how many actuators are saturated at the solution, and also stacked by algorithm subparts. Recall that these calculations are "cold-starts", ie the solver was initialised at $\Delta u_0 = \frac{1}{2}\left(\underline{\Delta u} + \overline{\Delta u}\right)$ on every execution.

As expected the "Problem Setup" step is equally expensive for all algorithms and almost negligible compared to the other subparts. The more iterations are performed, the more dominant the "Subspace Solver" step becomes for all algorithms, but especially for the updating algorithms (QR and Cholesky).

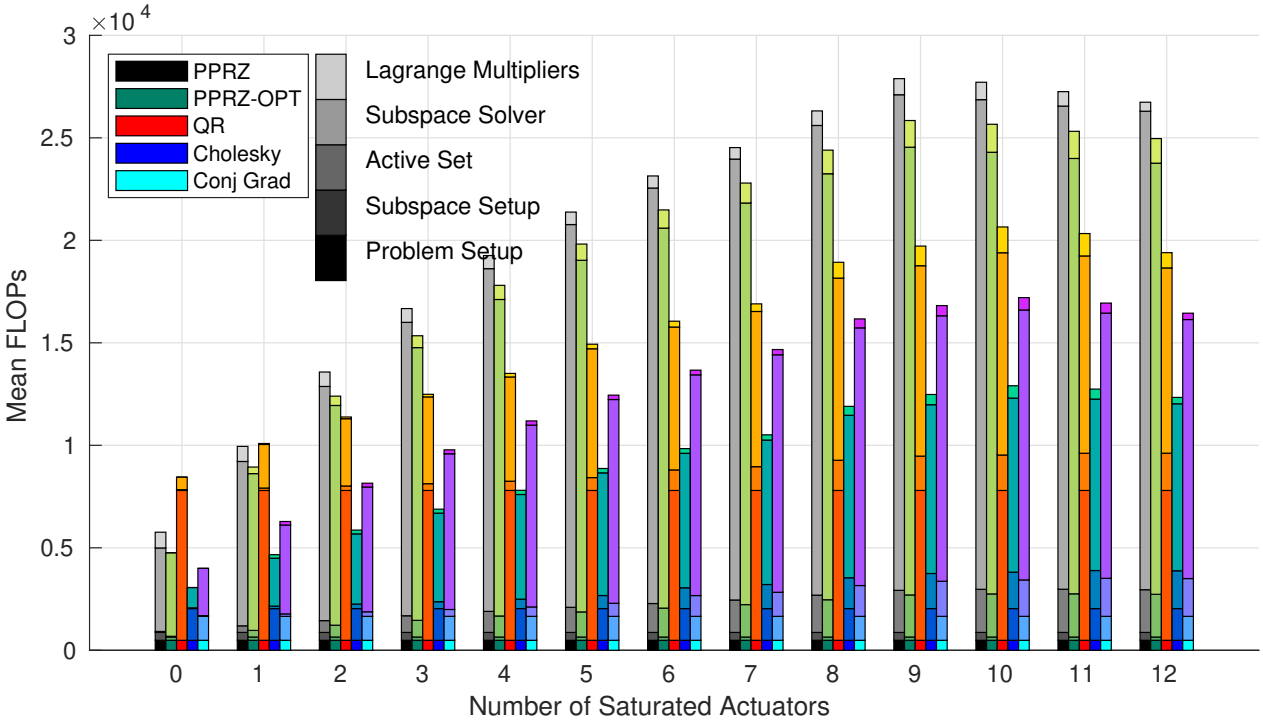


Figure 4. Theoretical algorithm performance, based on FLOPs counting, for a 12-rotor multicopter.

Overall, the expected computational expense decreases near total saturation because these cases typically require fewer iterations: imagine the unconstrained solution for a $\Delta\nu$ is a point $\Delta u^* \in \mathbb{R}^n$ that is very far outside the box given by the constraints. With every iteration, one actuator will be constrained at one of its limits and the algorithm likely terminates after $n+1$ iterations. However, if the point is much closer to the boundary of Φ, then some bounding *and* freeing iterations may be required before the correct active set is identified.

It can be seen how the improvements for the algorithms on updating factorisations (QR and Cholesky) become much more noticeable when more actuators saturate (and more iterations are required). Indeed, the large "subspace setup" component makes the QR perform slower than the PPRZ reference solvers, which do not have to perform the explicit recovery of the Q factor. Cholesky is faster than QR with a relatively constant difference since most improvements are gained in the initial factorisation, see Section A.3. For this platform, the theoretical improvements over the optimised PPRZ algorithm for the most expensive problems (9-10 saturating actuators) are around 20% for QR, 50% for Cholesky and 35% for CG. However, for a smaller multicopter with $n = 6$, QR, PPRZ and CG are around equal and Cholesky improves the worst-case runtime by only 25%.

Lastly, Figure 5 shows the scaling of the algorithm FLOPs with platform size for the *feasible* part of the test cases, for which a single iteration of the algorithms leads to the solution. All algorithms have an $O(n^3)$ component, so it is expected that they all scale cubically. For the Cholesky algorithm, a polynomial fit is presented.

5.1.1. Benefits from Sparse Multiplication Operations

Reduction of the floating point operation for matrix-matrix products $Z = AB$ is possible, when not all elements of Z have to be computed, for instance, because the sparsity of A or B dictates that some vector-vector products are 0, or because the resulting matrix Z is known to be symmetric and only a triangular part has to be computed and stored.

In particular, A in Equation 2 contains a dense $d \times n$ part and a diagonal $n \times n$ part below the dense part. The floating point operations of matrix-vector products with A can be reduced from $2n(n+d) - (n+d) = 2n^2 + 2nd - n - d$ to $2dn - d + n$, and the symmetric

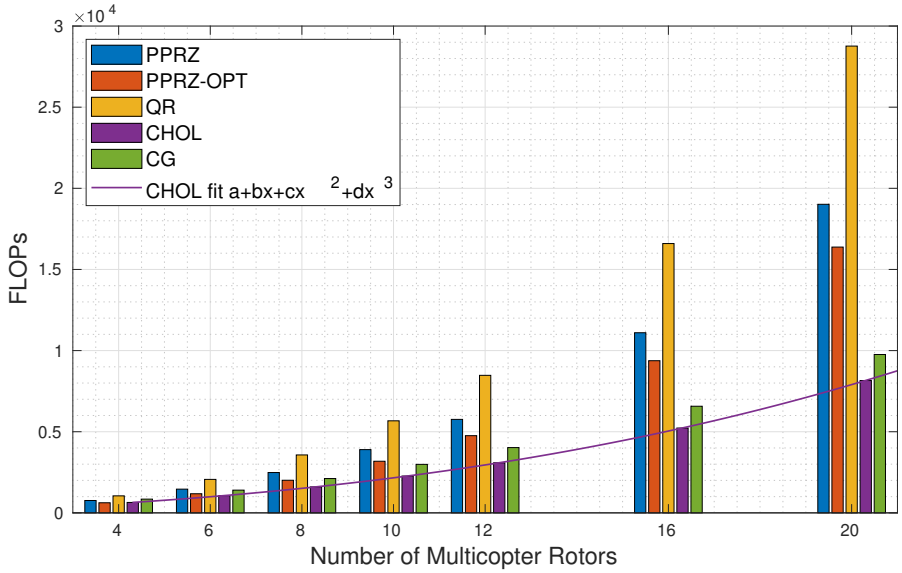


Figure 5. FLOPs scaling of the algorithms with multicopter platform size.

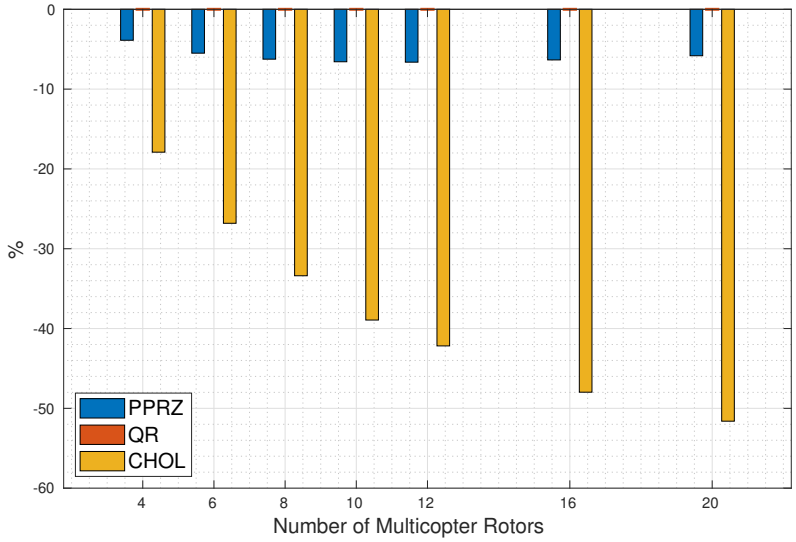


Figure 6. Reduction of operations by using sparse multiplication operations for Active-Set algorithms with different factorisations.

matrix A^TA can be s symmetric and reduces from the naive $2n^3 + 2n^2(d-2)$ down to $n^2(d-\frac{1}{2}) + n(d+\frac{1}{2})$. Sometimes, a permuted subset of the columns of A is needed, such that additional kn operations (with low factor k) are necessary to keep track of the locations of the non-zeros.

Replacing all vector-matrix and matrix-matrix products with optimised versions results in significant reduction of FLOPs for some algorithms, shown in Figure 6.

Since the QR-based subspace solver does not perform any products with A and only uses its factors Q and R for computation of the unconstrained subspace solutions, residual and Lagrange multipliers, there is no potential for optimisation. The initial A^TA product necessary for the Cholesky-based solver provides the largest opportunity for improvement.

5.2. C Package

For the C implementation, the control allocation computations are sectioned into two functions; `setup_wls` turns the components of Equation 2 into the components of Equation 4 and `solveActiveSet{PPRZ|QR|CHOL|CG}` solves the quadratic problem. See Appendix B for additional pseudo-code that describes the implementations of the subspace solvers.

However, most sparse matrix operations applied are not shown for brevity. The C code is available at https://www.github.com/tblaha/ActiveSetCtlAlloc.

The package uses a typical column major representation for matrices. The QR factorisations routines are taken from `qr_solve`[7], which is also used in the already existing PPRZ implementation. The Cholesky factorisation routine was adapted from[8] and the Paparazzi Autopilot code. All update routines for QR factorisations have been implemented by the authors based largely on Dongarra *et al.* [25, ch 10]. Cholesky updating has been implemented according to Stewart [26, sec 3.5/3.6] and Osborne [27, App B].

The following sections provide results of the verification of the *C* code using different choices for the separation parameter γ and the accuracy of the floating point arithmetic used. Furthermore, this section provides the verification results and estimated execution times on an x86 desktop platform. `gcc` version 9.4.0 was used throughout.[9]

5.2.1. Verification

The accuracy problems for Conjugate Gradient (already mentioned in Section 5.1) were so severe that it was discarded from further analysis.

Similarly, the Cholesky-based subspace solver also indeed exhibits numerical errors when 32-bit single floating points are used in the computations, but to lower severity. This was readily revealed by varying the γ parameter of Equation 2 and comparing the result of the C program to the reference output of MATLAB®.

Figure 7 shows a typical verification performed on the *C* code, in this case for a platform with 6 actuators, against `quadprog.m` at $cond(A^TA) = 10^9$. At higher γ (and consequently lower condition number of $H \equiv A^TA$), all algorithms have the same poor accuracy, due to solving a slightly different cost function. However, Cholesky with 32-bit single arithmetic cannot get close to the actual solution at $cond(A^TA) = 10^9$.

A notion of computational stability can be introduced as the difference in solutions when the input $\Delta\nu$ only changes slightly. At lower γ (and consequently higher condition numbers), the stability of the single Cholesky is highly compromised (even though the resulting $\Delta\nu$ may be close to the other algorithms. This may lead to erratic distributions in flight.

It would thus be advantageous to be able to trade-off between the two errors; introduced by a larger choices of γ (to achieve better conditioning) vs. numerical errors resulting in unstable allocations. In response to that, two methods are presented to reduce the inaccuracies; Condition Number Limiting and Cost Based Termination.

5.2.2. Condition Number Limiting

A correlate of the loss of accuracy seems to be the condition number of the matrix $H \equiv A^TA$ (Equation 4) that is factorised. Figure 7 shows the behaviour of the mean errors across the test set over the resulting condition number of A^TA.

Limiting the condition number of the matrix H through appropriate choice of the separation parameter γ has been found as a potential remedy. It is desired to change parameters in-flight, so this has to be done in real-time, which is challenging since precise condition number computations have $O(dn^2)$ time complexity (for instance with a QR factorisation).

For invertible square matrices, such as the positive definite H, the condition is the ratio of the highest and lowest eigenvalue [25]. Under some mild assumptions, an upper bound for the condition number can be found in $O(d^2)$ time. As shown in Equation A2, A can be split into the terms P and D, which have the properties $P^TP = A_d^TA_d$, where $A_d \in \mathbb{R}^{d\times n}$ are the dense rows of A and $D^TD = \gamma^2W_u^TW_u$. Since we desire some degree of separation between the objectives of Equation 2, we can postulate that the lowest eigenvalue

[7] John Burkardt: https://people.sc.fsu.edu/~jburkardt/c_src/qr_solve/qr_solve.html
[8] http://rosettacode.org/wiki/Cholesky_decomposition#C
[9] https://gcc.gnu.org/

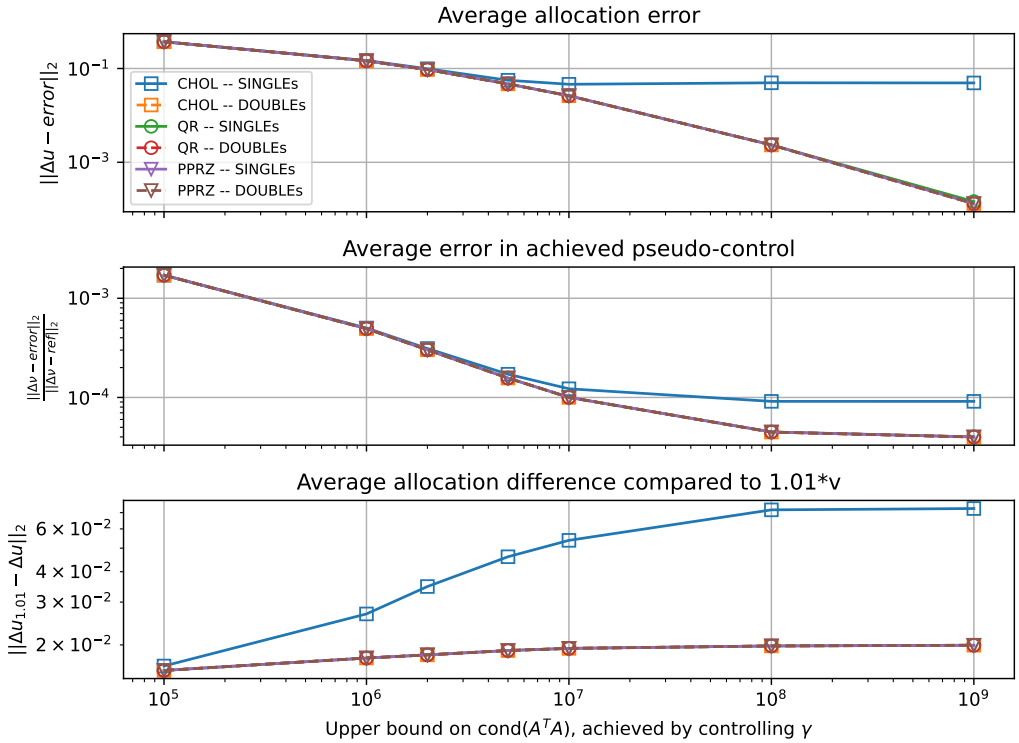


Figure 7. Effects of Condition Number on solver accuracy. Hexacopter platform.

be contained in D and given simply by $\sigma_{\min} = \gamma^2 \min_i W_{u_i}^2$. The largest eigenvalue is thus assumed to be contained in P and an upper bound $\sigma_{\max} \leq \bar{\sigma}_{\max}$ can be estimated using the Gershgorin disc with the largest upper bound [28] which can be computed in $O(d^2)$ time. In practice, this upper bound is close to the actual largest eigenvalue, with a maximum overestimate of 5.0% for a hexacopter platform and less for larger platforms.

Now, a minimum acceptable value for γ to maintain a condition number below $C_{\max}$ is simply given by

$$\gamma \geq \gamma_{\min} = \sqrt{\frac{\bar{\sigma}_{\max}}{\min_i W_{u_i}^2 \cdot C_{\max}}}. \tag{5}$$

Experimentation has to show appropriate choices of $C_{\max}$ that ensure numerical stability and accuracy, while not distorting the original problem. Furthermore, it is possible to apply a similar procedure to find a lower bound of the lowest eigenvalue in P which could be used to verify adequate separation between the objectives. Although not quantitatively investigated in this work, it should also be noted that the separation of the eigenvalues may only result in good separation of the two objectives when the actuators u and pseudo-controls ν are on similar scales.

5.2.3. Terminating Based on Cost

As an alternative or complementary measure to limiting the condition number, the iterations of an Active-Set algorithm may be truncated when no large improvements in the residual cost are observed. In addition to imprecise results, the Cholesky solver occasionally gets stuck in cycles where one or more actuators are bound and freed indefinitely with almost indistinguishable cost residuals, but without triggering the stopping conditions due to numerical inaccuracies. In the Cholesky subspace solver, the residual vector $r = Au - b$ is explicitly computed every time Lagrange multipliers are required (every time the actuator(s) are freed) and so the cost $||r||^2$ can be computed at negligible $2(n + d)$ FLOPs.

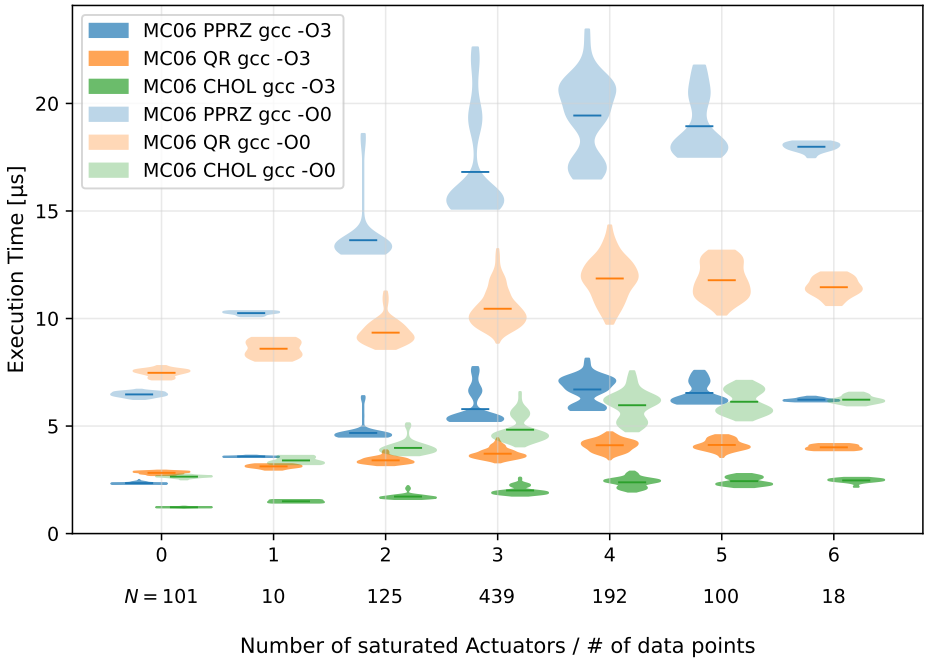


Figure 8. Desktop execution time measurements for 6-rotor multicopter.

A strategy to terminate can be proposed as such: a) terminate whenever the cost is below a certain absolute tolerance $\approx 10^{-7}$; b) terminate whenever the cost increases; c) terminate whenever the ratio cost reduction over cost is below a certain relative tolerance $\approx 10^{-7}$.

An indoor test flight confirmed that this strategy avoided all 519 occurring cycles. Further 21 executions that would have terminated, were terminated at equal or lower cost, but 22 were prematurely terminated at a maximum cost increase of 0.03, which is negligible (the cost at algorithm termination was between 0.51 and 1098 over the 138 361 data points).

5.2.4. x86 Execution Timing

The code has been compiled with and without the compiler optimisations present in `gcc`. The code was run single-core on a conventional mobile workstation with INTEL® `i7-4900MQ` processor at 3.8GHz and measured with `<time.h>`. To reduce the effects of the background activity, Ubuntu Linux 20.04 was used without a window manager, and minimum `niceness` was applied to the process. However, some extremely high spikes were still present, despite low or moderate iteration count, such that all data points above the 99.5th percentile were removed from each category.

For a hexacopter platform, Figure 8 arises, which shows very similar trends to the theoretical results from MATLAB® (Figure 4). Just as before, the Cholesky is faster than QR by an approximately constant margin, owing to the faster initial decomposition. Regardless of the algorithm, enabling the `-O3` optimisation flag yields an approximately three-fold increase in speed.

Figure 9 compares the scaling with platform size for the C-code to the theoretical polynomial curve found in MATLAB® (Figure 5). The floating point operations count (FLOPs) has been divided by the processor speed and multiplied by a coefficient that indicates the average number of clock cycles per floating point instruction. Without compiler optimisations (`-O0`), a value of $\sigma \approx 13.4$ has been found for the Clock per FLOP, which is much higher than would be expected for typical programs, which points to significant overhead in the program other than performing arithmetic operations, or that regular interrupts of context switches of the operating system distort the values. Their absolute magnitude should thus not be taken at face value.

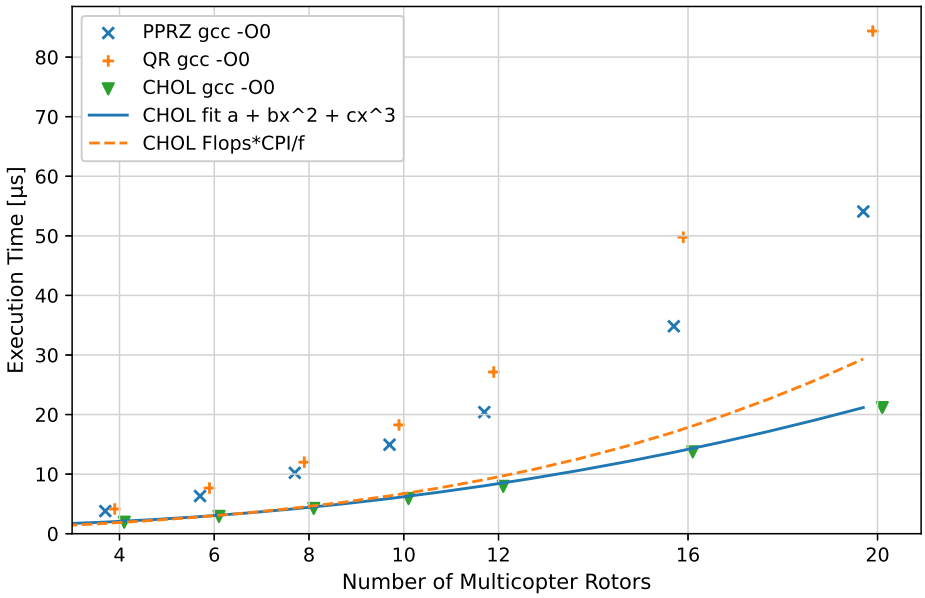


Figure 9. Desktop execution time measurements for feasible pseudo-controls (ie one iteration). The dashed line represents a scaled version of the fit in Figure 5.

5.3. Flight Testing

As explained in Section 4, an STM32F7 family microcontroller-based flight controller has been installed in a hexacopter platform to validate the results and provide a representative execution time comparison in a realistic scenario. The ARM processor design differs greatly from x86 architectures, and the real-time operating system environment differs from the desktop environment treated in the previous section. Therefore, different results are expected, both in magnitude and possibly also in comparison. The Paparazzi autopilot framework was compiled using the same `gcc` compiler and using the paparazzi-default `O2` optimisation flag, which is tuned to reduce executable size over execution speed.

Another objective of the test flights is to qualify if the choice of condition number (which may reduce the separation between the two objectives of the problem, Equation 2) affects the flying qualities. Furthermore, it should be verified that the numerical issues affecting the Cholesky-based solvers do not cause issues in-flight.

Figure 10 shows the manoeuvre flown 3 times for each algorithm and condition number setting. The large tilt angle command at the beginning, together with the yaw angle change successfully drives various actuators into saturation. Due to the saturation and priority setting between the axes, the multicopter loses altitude as some vertical thrust is sacrificed for achieving the rotation and levelling out the craft.

5.4. Flight Performance

Figure 11 shows an overview of the ≈ 15- minute test flight, where different algorithms and choices for the condition number bound have been configured. The pseudo-control error (first term in Equation 2) shows spikes for each of the manoeuvres like Figure 10 and is zero during hover, because the steady-state hover command is feasible. The spikes increases slightly with lower condition number bound, which indicates that the solution quality is worse.

Also, the termination reason of the algorithm shows how the cost plateau check described in Section 5.2.3 is triggered much more frequently at higher condition numbers for the Cholesky algorithm. In a different test flight, the algorithm even crashed with a `NaN` in the factorisation (found to be a square root of a negative number due to a round-off error). This shows that it is important to limit the condition number.

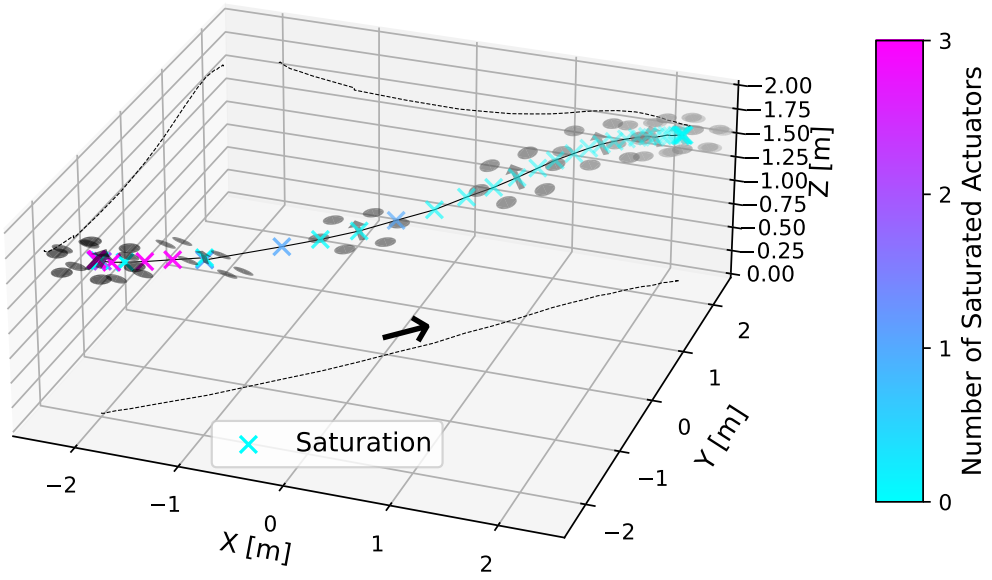


Figure 10. Flight Manoeuvre. Alternating between two holding points, using pitch, roll and yaw.

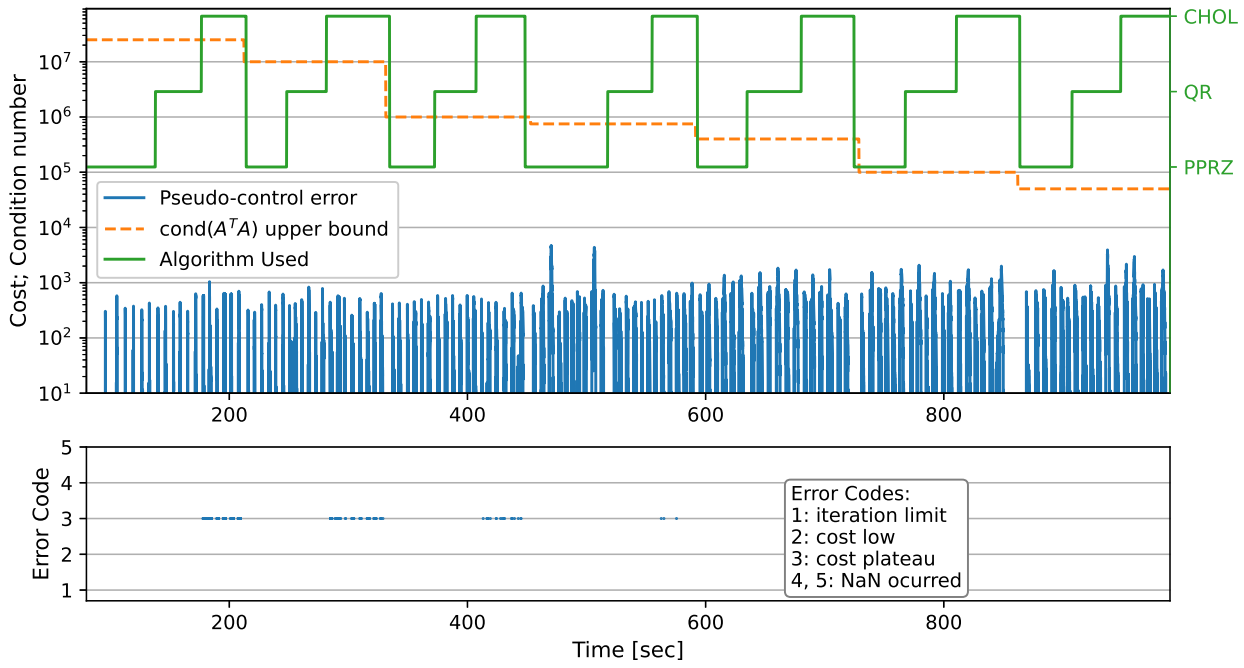


Figure 11. Test Flight algorithm behaviour. Cholesky has to be artificially terminated at higher condition numbers.

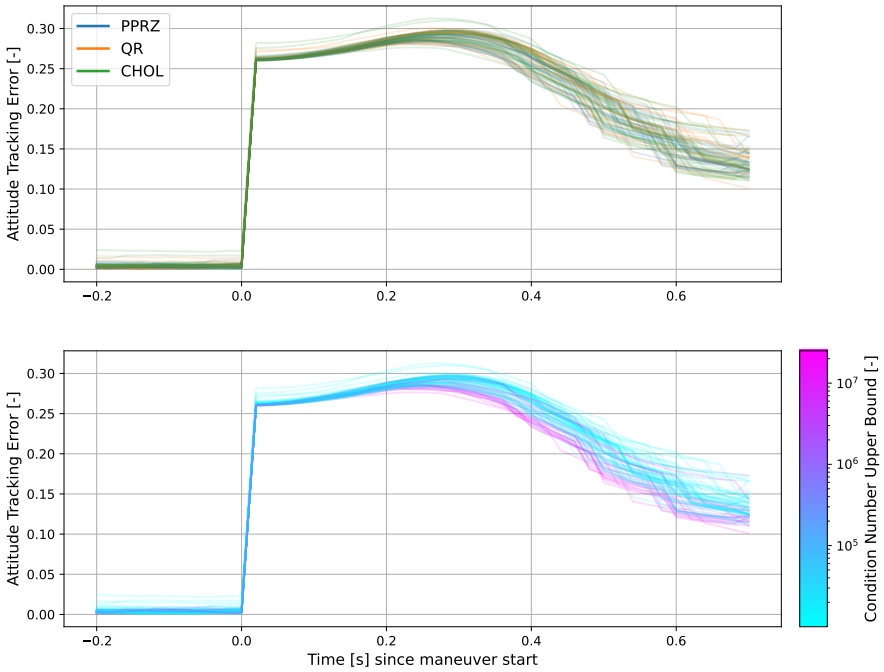


Figure 12. Attitude tracking errors. Setpoint vs onboard estimated attitude.

The effect on the attitude tracking performance is depicted in Figure 12 using a single-metric tracking performance indicator, defined as the Euclidean distance in $\mathbb{R}^4$, between the commanded and estimated attitude quaternion as shown in Huynh [29, function Φ_2].

The figure shows no correlation between the algorithms, but a small dependence on the condition number, whereas lower desired condition numbers cause slower attitude tracking. This is because the corresponding increase in the objective separator γ changes the penalty function (Equation 2) in such a way that the second term is not negligible when the first term is non-zero. Consequently, the actuator usage is reduced, despite that the desired pseudo-controls $\Delta\nu$ are not achieved, resulting in slower attitude tracking.

5.5. Execution Timings

The STM32F7 processor is configured to run at 216MHz, so it can be crudely expected that the execution is roughly 18 times slower than the x86 runs plotted in Section 5.2.4. The results in Figure 13 show that this is not the case and the difference is roughly a factor of 40, which is likely due to a higher load of the CPU than in the desktop experiments, or less efficient floating point arithmetic.

However, similar comparative trends are seen: Cholesky has a constant difference to QR and both become progressively more advantageous compared to PPRZ. Specifically, QR is around 50% faster, and Choleksy around 65% However, while in the flops counting and the x86 experiments, QR was slower than PPRZ at one iteration, they are now the same speed. Also for higher numbers of iterations, the PPRZ algorithm is slower than predicted, compared to QR and Cholesky. The cause could not be fully determined, but it may be that the QR-based least squares solver compiles to much less efficient bytecode on the ARM platform, or has been optimised for x86 in some way.

Note that Cholesky is the only algorithm to require more than 7 iterations, which may be due to its occasional inaccuracy at higher condition numbers.

5.6. Warm-Starting

Warm-starting an Active-Set algorithm refers to using the active set identified of a previous problem to solve a similar problem in fewer iterations. So far, all computations have assumed an empty working set $\mathcal{W}$ as a starting point (assuming no saturation is present). However, it is expected that the active set does not change rapidly between time

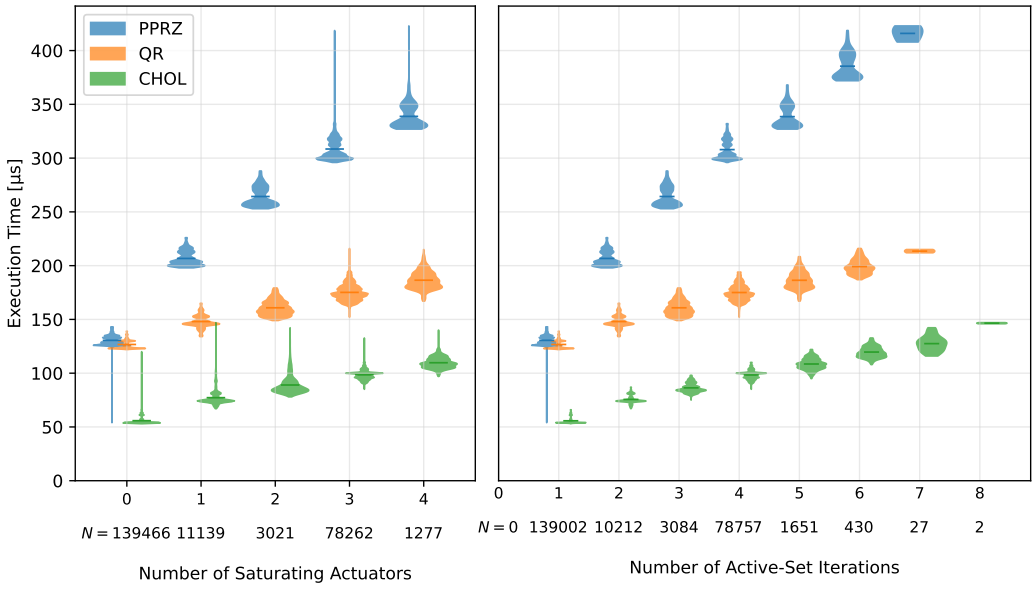


Figure 13. In-flight measurements. Outliers are likely due to processor interrupts. N refers to the datapoints present for the Cholesky dataset, but relative magnitude should be similar.

steps, owed to the high sampling frequency ($f_s = 500$Hz in the hexacopter experiments at hand) compared to the aircraft's dynamics.

Figure 14 shows the execution time of the Cholesky algorithm for 4 different manoeuvres; warm-starting on/off and high iteration limit of 100 vs. low iteration limit of 2. With large enough iteration limit, both warm-starting and cold-starting will return the same unique solution, however, the execution time for warm-starting is much lower *on average*. However, it cannot be guaranteed that the worst-case execution time is also lower, since there the active set may change significantly between time steps.

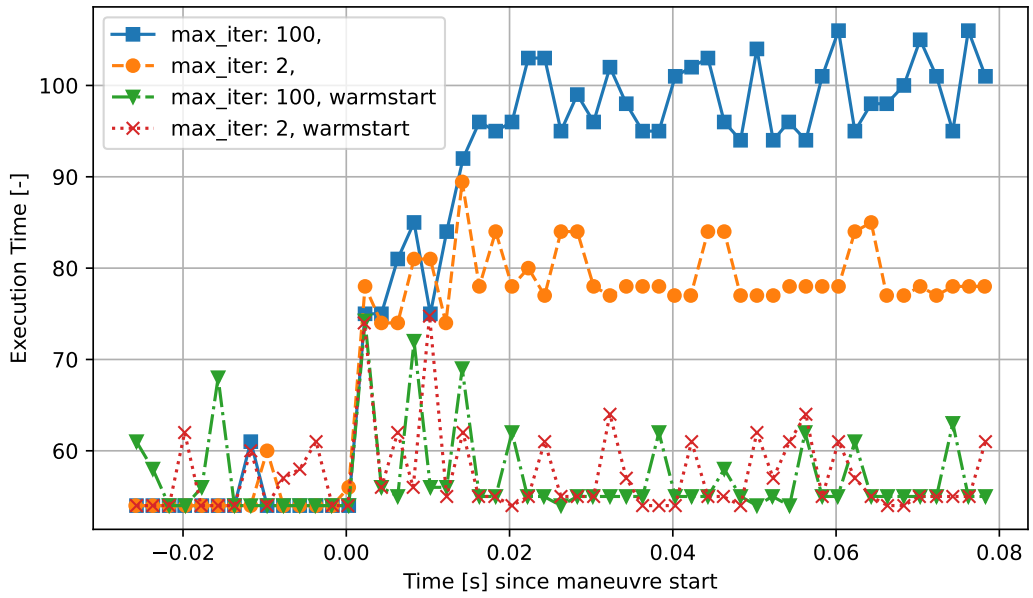


Figure 14. Warmstarting can significantly reduce computational time.

To provide guarantees on worst-case execution time, a low iteration limit can be placed. With cold- starting, this would result in significant allocation errors, whenever the active set requires more iterations. In Figure 14, 4 actuators are saturated which cannot be captured

with just 2 iterations. With warm-starting, this condition would only be present for the first few time steps following a large active set change, which may have little to no impact on the dynamics (potentially comparable to a very fast low pass filter). However, this impact was not quantified in this study.

6. Conclusions & Recommendations

This work shows that computational timing improvements over current Active-Set control allocation methods (with Smeur *et al.* [9] as a benchmark) are indeed possible.

Explicitly factorising one of the problem matrices (A into its QR factors, or H into its Cholesky factors) increases computational time whenever the optimisation terminates after one iteration. However, if these factors are updated with each Active-Set iteration, the worst-case computational time decreases for platforms with more than $n >= 6$ actuators. A recommendation can be made to investigate if the factors can be reused and the next time step, if the platform actuators' effectiveness does not change significantly between time steps.

The initial factorisation is approximately twice as fast with Cholesky than QR, owed to it intrinsically requiring fewer operations for the problem sizes. This holds true despite that $H \equiv A^T A$ has to be computed first, because this calculation can be optimised with the sparsity of A. Future research could produce a specialised QR factorisation algorithm (similar to Matstoms [30]) that takes into account the sparsity and could reduce computation times for QR and the benchmark [9]. It remains to be seen if such a method could approach the speed of the Cholesky-based solver.

Flight tests on an $n = 6$ hexacopter platform using an STM32F7 microprocessor confirmed that the runtime of Cholesky is twice as fast as QR for the first iterations and $\approx 25\%$ faster in the worst-case. However, it is revealed that in the most computationally demanding cases, QR is approximately twice as fast as the benchmark [9] (Cholesky thus $\approx 65\%$ faster), which was not expected from the theoretical floating point operations analysis and points to: a poor implementation of its subspace solver, additional runtime checks that were not considered in the operations counting analysis, or errors in that analysis.

A Conjugate Gradient-based subspace solver exhibits significant numerical problems, but may approach Cholesky speeds. Appropriate pre-conditioners (see Nocedal and Wright [31], Quirynen *et al.* [32]) may remedy this in future research.

If efficient 64-bit arithmetic becomes popular on micro-controllers in the future, the Cholesky-based algorithm could be used as is; however, to ensure safe flight on current 32-bit platforms, numerical stability issues must be resolved. An efficient method is presented that limits the condition number of H by controlling the objective separation parameters γ. The flight tests showed that the attitude tracking of a multicopter is measurably slower when the condition number is decreased, and a trade-off persists that may be quantitatively addressed in future research; the authors recommend a value of $5 \cdot 10^5$ for multicopters.

A simple cost-based pruning was shown to effectively avoid cycles in the Active-Set algorithm at insignificant extra computational costs.

Initialising the solver with the solution of the previous time-step ("warm-starting") can yield large benefits to the average run-time as the update frequency of flight control systems are typically much larger than the changes to the flight path and control commands. Nonetheless, ensuring the reduction of worst-case execution time still requires premature termination, introducing errors. Further research would have to quantify how placing an iteration limit affects the flying qualities.

Appendix A Subspace Solver Discussions

This section describes the working principles and possible advantages of the four subspace optimisers considered.

Appendix A.1 Solving Separate Systems – "PPRZ"

The algorithm described in Smeur *et al.* [9] (called PPRZ for its usage in the Paparazzi autopilot framework[10]) simply removes or adds a column in A_f and uses the residual $r = b - A\Delta u$, where Δu is the solution on the current subspace spanned by the currently active set of constraints. This way, the size of the problem size decreases with every actuator that is bound, but the least squares problem $\arg\min_{p_f} ||A_f p_f - r||$ still has to be solved from scratch at every iteration, even though A_f changes only slightly at every iteration.

Appendix A.2 Updating QR Factorisation – "QR"

The matrix A can also be QR-factored at the beginning of the algorithm into the full Q and R factors ($A = QR$, with Q orthogonal and R upper triangular), with which equality-constrained least square problems can be solved with the "nullspace method" [11, sec 5.1.3.]. This initial factorisation is more expensive than a single solution of the least squares problem, since Q is not usually explicitly recovered, but rather stored as Householder factors [25].

The resulting algorithm shown in Björck [11, ALGORITHM 5.2.1., page 203], bounding and freeing of an actuator is represented by a partial circular shift of the columns of A, such that the free actuators (ie the smaller system to be solved in the current iteration) are stored in the upper left corner of Q and R. The other parts of Q and R can be used to calculate Lagrange multipliers. After shifting the columns of R, it has to be re-triangularized using orthogonal Givens rotations, which can be performed in $O(n^2)$ time, as shown in Dongarra *et al.* [25, ch 10].

Appendix A.3 Updating Cholesky Factorisation – "CHOL"

When posed as a standard quadratic programming problem, Equation 4, the unconstrained optimum is the solution to

$$H_{f,f} u_f = \beta_f - H_{f,b} u_b, \tag{A1}$$

where subscript f denotes the rows of the currently free variables and b denotes the rows of the bound variables. The square matrix $H_{f,f}$ are the rows and columns H pertaining to free variables, and $H_{f,b}$ are the rows pertaining to free variables, but columns pertaining to bound variables.

Since $H \equiv A^T A$ is symmetric and positive definite, there exists a unique lower triangular matrix L, such that $H = LL^T$, which is called a Cholesky factor. When L is known, solving the linear system can thus be done efficiently in $O(n^2)$ time with forward-backward substitution [16]. Update schemes exist to efficiently augment/reduce L in $O(n^2)$ time, for the case that H_f only changes by the addition/removal of one row and one column [26].

Cholesky factorisations are generally faster than QR factorisations, but the initial computation of H may be expensive [23].

Appendix A.4 Conjugate Gradient – "CG"

Equation A1 can also be solved using Conjugate Gradients [31] since H_f is positive definite. In this method, we choose consecutive search directions that are optimised using a line search, which for positive definite matrices has unique and explicit solutions that do not require inversion of matrices. What makes this method attractive is that in general, it is guaranteed to terminate after $n_f + 1$ iterations, where n_f is the size of H_f. However, when H_f has only $\kappa < n_f$ distinct eigenvalues, the iterations terminate after $\kappa + 1$ iterations.

We can remember that A only has dense values in the first d rows, and an $n \times n$ diagonal part below that, which implies that $A = P + D$, where P holds the dense rows

[10] https://wiki.paparazziuav.org/

A and has zeros otherwise and $D = \begin{pmatrix} 0 & \gamma W_u \end{pmatrix}^T$ holds the diagonal part. Noting that the products $P^T D = D^T P = 0$ by the definitions of these two matrices, $A^T A$ can be written as

$$\begin{aligned} A^T A &= P^T P + P^T D + D^T P + D^T D \\ &= P^T P + D^T D \\ &= A_d^T A_d + \gamma^2 W_u^T W_u \, . \end{aligned} \tag{A2}$$

Here, $A_d \in \mathbb{R}^{d \times n}$ denotes the $d \times n$ dense part of A, as shown in Equation 3.

Since A_d has at most rank d, $P^T P$ has at most d distinct eigenvalues. When all or many weights in W_u are equal, $D^T D$ also only has few distinct eigenvalues, such that $A^T A$ may have only $d + 1$ distinct eigenvalues, depending on how many actuators weights in W_u are identical, which would mean that the Conjugate Gradient iterations may terminate with the exact solution much earlier than after $n + 1$ iterations.

Appendix B Subspace Solver Subroutines

Appendix B.1 INIT

Algorithm 2: PPRZ.INIT

$d \leftarrow b - Au$;

Algorithm 3: QR.INIT

R, $H \leftarrow$`DQRDC(`A`);` `/* Factorise into` R `and householder reflectors` H `*/`

$Q \leftarrow$`DORGQR(`H`);` `/* Recover` Q `factor from the columns of` H `*/`

Algorithm 4: CHOL.INIT

$H \leftarrow A^T A$;

$L \leftarrow$PPRZ_CHOLESKY_FLOAT(H);

Algorithm 5: CG.INIT

$H \leftarrow A^T A$;

Appendix B.2 SOLVE

Algorithm 6: PPRZ.SOLVE

Result: z

$p \leftarrow A\left[:,\ permutation[1:n_f]\right] \setminus d$;

$i \leftarrow 1$;

while $i \leq n_f$ **do**

  $z[permutation[i]] \leftarrow \Delta u[permutation[i]] + p[i]$;

  $i \leftarrow i + 1$;

end

Algorithm 7: QR.SOLVE [11]

Result: z

$c \leftarrow \left(Q[:,\ 1:n_f]\right)^T b$;

$c \leftarrow c - \left(R[:,\ n_f+1:n]\right)\ \Delta u\left[permutation[n_f+1:n]\right]$;

$q \leftarrow$ solve $\left(R[1:n_f,\ 1:n_f]\right)\ q = c[1:n_f]$ `/* Efficient, because` R `is upper triangular */`

$i \leftarrow 1$;

while $i \leq n_f$ **do**

  $z[permutation[i]] \leftarrow q[i]$;

  $i \leftarrow i + 1$;

end

while $i \leq n$ **do**

  $z[permutation[i]] \leftarrow \Delta u[permutation[i]]$;

  $i \leftarrow i + 1$;

end

Algorithm 8: CHOL.SOLVE

Result: z

$\beta \leftarrow A\left[:,\ permutation[1:n_f]\right]^T b\left[permutation[1:n_f]\right]$;

$\beta \leftarrow \beta - H\left[permutation[1:n_f],\ permutation[\left(n_f+1\right):n]\right]\Delta u\left[permutation[\left(n_f+1\right):n]\right]$;

$L_f \leftarrow L\left[1:n_f,\ 1:n_f\right]$;

$q \leftarrow$ solve $\left(L_f L_f^T\right) q = \beta$ `/* Efficient, because` L_f `and` L_f^T `are triangular */`

$i \leftarrow 1$;

while $i \leq n_f$ **do**

  $z[permutation[i]] \leftarrow q[i]$;

  $i \leftarrow i + 1$;

end

while $i \leq n$ **do**

  $z[permutation[i]] \leftarrow \Delta u[permutation[i]]$;

  $i \leftarrow i + 1$;

end

Algorithm 9: CG.SOLVE

$H_{ff} \leftarrow H[permutation[1:n_f],\ permutation[1:n_f]]$;

$H_{fb} \leftarrow H[permutation[1:n_f],\ permutation[\left(n_f+1\right):n]]$;

$\beta \leftarrow A\left[:,\ permutation[1:n_f]\right]^T b\left[permutation[1:n_f]\right]$;

$\beta \leftarrow \beta - H_{fb}\Delta u\left[permutation[\left(n_f+1\right):n]\right]$;

$q \leftarrow$ solve $H_{ff} q = \beta$; `/* Using [31, Algorithm 5.2] */`

$i \leftarrow 1$;

while $i \leq n_f$ **do**

  $z[permutation[i]] \leftarrow q[i]$;

  $i \leftarrow i + 1$;

end

while $i \leq n$ **do**

  $z[permutation[i]] \leftarrow \Delta u[permutation[i]]$;

  $i \leftarrow i + 1$;

end

Appendix B.3 FREE and BOUND

Algorithm 10: PPRZ.FREE

$d \leftarrow d - A\left[:,\ permutation[1 : n_f - 1]\right]\ p;$

Algorithm 11: QR.FREE

RIGHT_CIRCSHIFT_OF_COLUMNS$\left(R\left[:,\ \left(n_f + 1\right) : p_f\right]\right)$;
Re-triangularise R using Givens rotations $G_1,\ G_2,\ \ldots,\ G_M$; /* [25, ch 10] */
$i \leftarrow 1$;
while $i \leq M$ **do**
  $Q \leftarrow G_i^T Q$;
  $i \leftarrow i + 1$;
end

Algorithm 12: CHOL.FREE

/* Change L to represent appending a column/row to LL^T [27, App B.1] */
$H_{12} \leftarrow H[permutation[1 : n_f],\ permutation[p_f]]$;
$H_{22} \leftarrow H[permutation[p_f],\ permutation[p_f]]$;
$L_{21} \leftarrow$ solve $L\ L_{21}^T = H_{12}$;
$L_{22} \leftarrow \sqrt{H_{22} - L_{21} L_{21}^T}$;
Expand L using:
$L[n_f + 1][1 : n_f] \leftarrow L21$;
$L[n_f + 1][n_f + 1] \leftarrow L22$;

Algorithm 13: CG.FREE

/* NO OP */

Algorithm 14: PPRZ.BOUND

$d \leftarrow d - \alpha A\left[:,\ permutation[1 : n_f + 1]\right]\ p;$

Algorithm 15: QR.BOUND

LEFT_CIRCSHIFT_OF_COLUMNS$\left(R\left[:,\ p_b : n_f\right]\right)$;
Re-triangularise R using Givens rotations $G_1,\ G_2,\ \ldots,\ G_M$; /* [25, ch 10] */
$i \leftarrow 1$;
while $i \leq M$ **do**
  $Q \leftarrow G_i^T Q$;
  $i \leftarrow i + 1$;
end

Algorithm 16: CHOL.BOUND

/* Change L to represent removing column/row p_b from LL^T; [27, App B.4] */
$L_{33} \leftarrow L[(p_b + 1) : n_f,\ (p_b + 1) : n_f])$;
$L_{32} \leftarrow L[p_b : n_f,\ p_b]$;
$L[p_b : (n_f - 1),\ p_b : (n_f - 1)] \leftarrow \text{CHOL}(L_{33} L_{33}^T + L_{32} L_{32}^T)$; /* This is a rank 1 update of factor L_{33}; efficient using [26, Algo 3.9] */
Drop last column and row;

Algorithm 17: CG.BOUND

/* NO OP */

Appendix B.4 LAGRANGE_MULTIPLIERS

Algorithm 18: PPRZ.LAGRANGE_MULTIPLIERS

$r \leftarrow -A^T(d - A\left[:,\ permutation[1:n_f]\right]\ p)$;
$i \leftarrow n_f + 1$;
while $i \leq n$ **do**
  $\lambda[i] \leftarrow \mathcal{W}[permutation[i]]\ r[permutation[i]]$;
  $i \leftarrow i + 1$;
end

Algorithm 19: QR.LAGRANGE_MULTIPLIERS

$d \leftarrow Q\left[:,\ \left(n_f+1\right):n\right]^T b$;
$d \leftarrow d - R\left[\left(n_f+1\right):n,\ \left(n_f+1\right):n\right]\ \Delta u\left[permutation[\left(n_f+1\right):n]\right]$;
$r \leftarrow R\left[\left(n_f+1\right):n,\ \left(n_f+1\right):n\right]^T d$;
$i \leftarrow n_f + 1$;
while $i \leq n$ **do**
  $\lambda[i] \leftarrow \mathcal{W}[permutation[i]]\ r[permutation[i]]$;
  $i \leftarrow i + 1$
end

Algorithm 20: CHOL.LAGRANGE_MULTIPLIERS [11]

$r \leftarrow A[:,\ permutation[\left(n_f+1\right):n]]^T(A\Delta u - b)$;
$i \leftarrow 1$;
while $i \leq n - n_f$ **do**
  $\lambda[n_f + i] \leftarrow \mathcal{W}[permutation[n_f + i]]\ r[i]$;
  $i \leftarrow i + 1$;
end

Algorithm 21: CG.LAGRANGE_MULTIPLIERS

$r \leftarrow A[:,\ permutation[\left(n_f+1\right):n]]^T(A\Delta u - b)$;
$i \leftarrow 1$;
while $i \leq n - n_f$ **do**
  $\lambda[n_f + i] \leftarrow \mathcal{W}[permutation[n_f + i]]\ r[i]$;
  $i \leftarrow i + 1$;
end

References

1. Sieberling, S.; Chu, Q.P.; Mulder, J.A. Robust Flight Control Using Incremental Nonlinear Dynamic Inversion and Angular Acceleration Prediction. *Journal of Guidance, Control, and Dynamics* **2010**, *33*, 1732–1742. Publisher: American Institute of Aeronautics and Astronautics, https://doi.org/10.2514/1.49978.
2. Smeur, E. Incremental Control of Hybrid Micro Air Vehicles. PhD thesis, Delft University of technology, 2018. https://doi.org/10.4233/UUID:23C338A1-8B34-40A6-89E9-997ADBDAFD75.
3. Bodson, M.; Frost, S.A. Load Balancing in Control Allocation. *Journal of Guidance, Control, and Dynamics* **2011**, *34*, 380–387. Publisher: American Institute of Aeronautics and Astronautics, https://doi.org/10.2514/1.51952.
4. Durham, W.C. Constrained control allocation. *Journal of Guidance, Control, and Dynamics* **1993**. https://doi.org/10.2514/3.21072.
5. Bodson, M. Evaluation of Optimization Methods for Control Allocation. *Journal of Guidance, Control, and Dynamics* **2002**, *25*, 703–711. https://doi.org/10.2514/2.4937.
6. Lallman, F.; Davidson, J.; Bundick, W. Integrated reconfigurable control allocation. In Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit, 2001. _eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2001-4083, https://doi.org/10.2514/6.2001-4083.

7. Härkegård, O. Dynamic Control Allocation Using Constrained Quadratic Programming. *Journal of Guidance, Control, and Dynamics* **2004**, *27*, 1028–1034. https://doi.org/10.2514/1.11607.
8. Härkegård, O. Efficient active set algorithms for solving constrained least squares problems in aircraft control allocation. In Proceedings of the Proceedings of the 41st IEEE Conference on Decision and Control, 2002.; IEEE: Las Vegas, NV, USA, 2002; Vol. 2, pp. 1295–1300. https://doi.org/10.1109/CDC.2002.1184694.
9. Smeur, E.; Höppener, D.; de Wagter, C. Prioritized Control Allocation for Quadrotors Subject to Saturation. In Proceedings of the International Micro Air Vehicle Conference and Flight Competition 2017; H. D. P. J.-M. Moschetta G. Hattenberger: Toulouse, France, 2017.
10. Blaha, T. Literature Study: Computationally Efficient Control Allocation. Literature Study, Delft, 2022.
11. Björck, A. *Numerical methods for least squares problems*; SIAM: Philadelphia, 1996.
12. Hartley, E.N.; Jerez, J.L.; Suardi, A.; Maciejowski, J.M.; Kerrigan, E.C.; Constantinides, G.A. Predictive Control Using an FPGA With Application to Aircraft Control. *IEEE Transactions on Control Systems Technology* **2014**, *22*, 1006–1017. Conference Name: IEEE Transactions on Control Systems Technology, https://doi.org/10.1109/TCST.2013.2271791.
13. Lau, M.; Yue, S.; Ling, K.; Maciejowski, J. A Comparison of Interior Point and Active Set Methods for FPGA Implementation of Model Predictive Control. *Proc. European Control Conference* **2015**.
14. Petersen, J.; Bodson, M. Constrained quadratic programming techniques for control allocation. *IEEE Transactions on Control Systems Technology* **2006**, *14*, 91–98. Conference Name: IEEE Transactions on Control Systems Technology, https://doi.org/10.1109/TCST.2005.860516.
15. Shahzad, A.; Kerrigan, E.C.; Constantinides, G.A. A warm-start interior-point method for predictive control. In Proceedings of the UKACC International Conference on Control 2010, 2010, pp. 1–6. https://doi.org/10.1049/ic.2010.0409.
16. Madsen, K.; Nielsen, H.; Tingleff, O. *Optimization with Constraints*, 2 ed.; Technical University of Denmark, 2004.
17. De Wagter, C.; Remes, B.; Smeur, E.; Tienen, F.; Ruijsink, R.; Hecke, K.; Horst, E. The NederDrone: A hybrid lift, hybrid energy hydrogen UAV. Preprint, 2020.
18. Kesaniemi, M.; Virtanen, K. Direct Least Square Fitting of Hyperellipsoids. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **2018**, *40*, 63–76. https://doi.org/10.1109/TPAMI.2017.2658574.
19. Fitzgibbon, A.; Pilu, M.; Fisher, R. Direct least square fitting of ellipses. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **1999**, *21*, 476–480. https://doi.org/10.1109/34.765658.
20. Gammell, J.D.; Barfoot, T.D. The Probability Density Function of a Transformation-based Hyperellipsoid Sampling Technique **2014**. Publisher: arXiv Version Number: 2, https://doi.org/10.48550/ARXIV.1404.1347.
21. STMicroelectronics. Application note – Floating point unit demonstration on STM32 microcontrollers. Technical Report AN4044, 2016.
22. van de Geijn, R.A. Notes on Householder QR Factorization. Technical report, The University of Texas, 2014.
23. Lira, M.; Iyer, R.; Trindade, A.; Howle, V. QR versus cholesky: A probabilistic analysis. *International Journal of Numerical Analysis and Modeling* **2016**, *13*, 114–121.
24. Höppener, D. Actuator Saturation Handling using Weighted Optimal Control Allocation Applied to an INDI Controlled Quadcopter. Master's thesis, Delft University of Technology, Delft, 2016.
25. Dongarra, J.J.; Moler, C.B.; Bunch, J.R.; Stewart, G.W. *LINPACK Users' Guide*; SIAM, 1979. https://doi.org/10.1137/1.9781611971811.
26. Stewart, G.W. *Matrix algorithms*; Vol. Volume I: Basic Decompositions, Society for Industrial and Applied Mathematics: Philadelphia, 1998.
27. Osborne, M.A. Bayesian Gaussian processes for sequential prediction, optimisation and quadrature. PhD Thesis, Oxford University, UK, 2010.
28. Gershgorin, S. Über die Abgrenzung der Eigenwerte einer Matrix. *Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et naturelles* **1931**, pp. 749–754.
29. Huynh, D.Q. Metrics for 3D Rotations: Comparison and Analysis. *Journal of Mathematical Imaging and Vision* **2009**, *35*, 155–164. https://doi.org/10.1007/s10851-009-0161-2.
30. Matstoms, P. Sparse QR factorization in MATLAB. *ACM Transactions on Mathematical Software* **1994**, *20*, 136–159. https://doi.org/10.1145/174603.174408.
31. Nocedal, J.; Wright, S.J. *Numerical Optimization*; Springer Series in Operations Research and Financial Engineering, Springer New York, 2006. https://doi.org/10.1007/978-0-387-40065-5.
32. Quirynen, R.; Knyazev, A.; Di Cairano, S. Projected Preconditioning within a Block-Sparse Active-Set Method for MPC. In Proceedings of the IFAC-PapersOnLine. International Federation of Automatic Control, 2018, Vol. 51, pp. 28–34. ISSN: 2405-8963 Issue: 20, https://doi.org/10.1016/j.ifacol.2018.10.170.

Part III

Additional Results

5

Additional Results

5.1. MATLAB framework

For better ability to compare algorithm prototypes under representative conditions, a framework was built in MATLAB® that generates effectiveness matrices B for different platforms along with a set of feasible and infeasible test vectors ν, and records performance of the different algorithms.

Figure 5.1 shows a simplified block diagram.

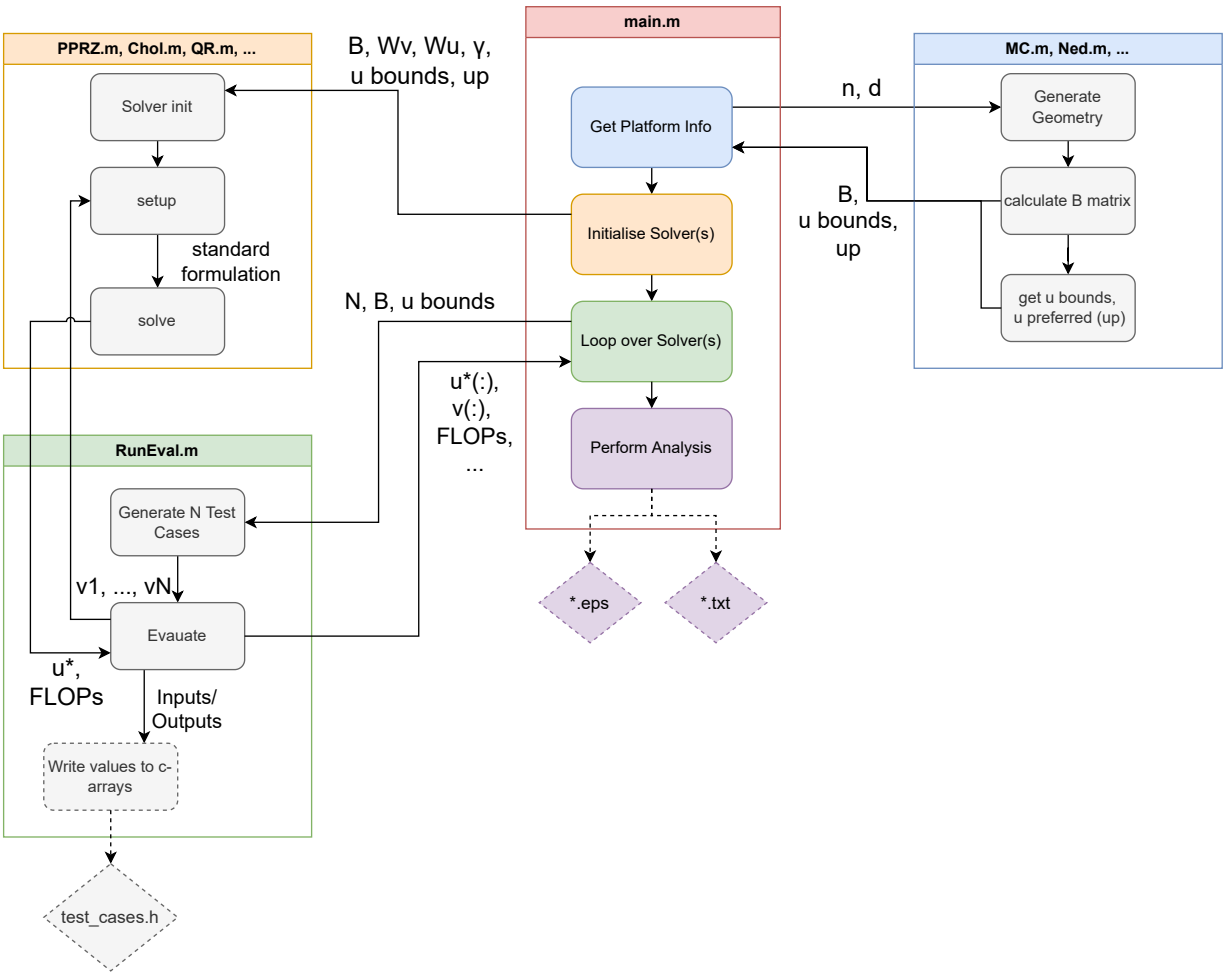


Figure 5.1: Modular framework to evaluate solvers and platforms.

Here, FLOPs stands for the floating pointing operations that were used to arrive at the solution of the

`setup` and `solve` blocks. This is implemented using a callback to a class instance which implements the counting and tracking of the FLOPS in the different algorithms and algorithm subparts, see example below, where `fadd` represents the counting callback passed to this lower level function.

forward_tri_solve.m

```
function x = forward_tri_solve(A, b, fadd)
    % solves Ax = b for x, iff A is lower triangular, square and
    % full-rank
    % see https://algowiki-project.org/en/Forward_substitution
    %if nargin < 3
    %    fadd = @(x)disp('');
    %end

    n = size(A, 1);
    if n == 0
        x = zeros(size(b));
        return
    end
    if n ~= size(A, 2)
        error("A is not square")
    end
    if rank(A) < n
        error("A must be full-rank")
    end
    if ~(norm(A - tril(A)) < sqrt(eps))
        error("A must be lower triangular")
    end

    x = zeros(n, 1);
    x(1) = b(1) / A(1, 1);
        fadd(flops_div_ARM());

    i = 2;
    while i <= n
        temp_sum = A(i, 1:i)*x(1:i);
            fadd(flops_mul(A(i, 1:i), x(1:i)))

        num = b(i) - temp_sum;
            fadd(1);
        x(i) = num / A(i, i);
            fadd(flops_div_ARM());
        i = i + 1;
    end

end
```

5.2. Additional Algorithms Partly Implemented

Although some equations for an Interior Point solver have been developed (Section 3.1), the implementation in the code presented big challenges and convergence could not be achieved. Especially the correct formulation of the complementarity measure μ_i and its treatment in the solution steps was perceived as very difficult. A future would likely require more extensive research into Interior Point methods, and starting from an existing implementation.

Gradient Projection was also attempted but it proved much slower than the active-set approaches, so it was discarded before it was completely debugged and integrated. A downside is that gradients are computed more often than just during the Lagrange multiplier step of active-set, and some form of

Aspect	Specification
Frame	DJI® Flame Wheel F550
Motors	23×12mm 920KV
Props	8×4.5”
ESC	PWM control, 15A
Battery	Flown on 5Ah 4S
Flight Controller	Holibro Pixhawk 4
Telemetry	XBee Pro RF
Receiver	FrSky R-XSR

Table 5.1: Components used for the flight test platform.

Figure 5.2: Flame Wheel Hexacopter, read-to-fly. Note the infrared markers one one of the arms and on some propellers. The centered attachment for a vertical safety rope is also visible.

subspace optimisation is still necessary to identify the saturated actuators and finally the optimum. The possibly reduced iteration count did not outweigh the downsides in terms of computational time.

5.3. Test Platform Specifications and Tuning

A DJI® Flamewheel F550 frame available at the MAVLab was used for the test flights. It was chosen for its availability and generous space for mounting of a Pixhawk 4[1] flight controller. The specs are listed in Table 5.1, and resulted in approximately 3 to 1 thrust to weight ratio and a hover-time in excess of 15 minutes.

The final configuration is shown in Figure 5.2.

5.3.1. Gain Tuning

Using a motor test bench, a quadratic relation from PWM percentage command to thrust percentage was found as $T = 0.616P^2 + 0.378P + 0.0056$. This relation was solved for P and used as the last scaling step in Paparazzi to ensure that a thrust command u maps linearly to the actual propeller thrust. Also, sound was recorded, from which the RPM response of the ESC/motor/propellor combination can be found. Figure 5.3 shows the Fourier transform of the audio of a step-input; we have selected the strongest harmonics

[1] http://www.holybro.com/product/pixhawk-4/

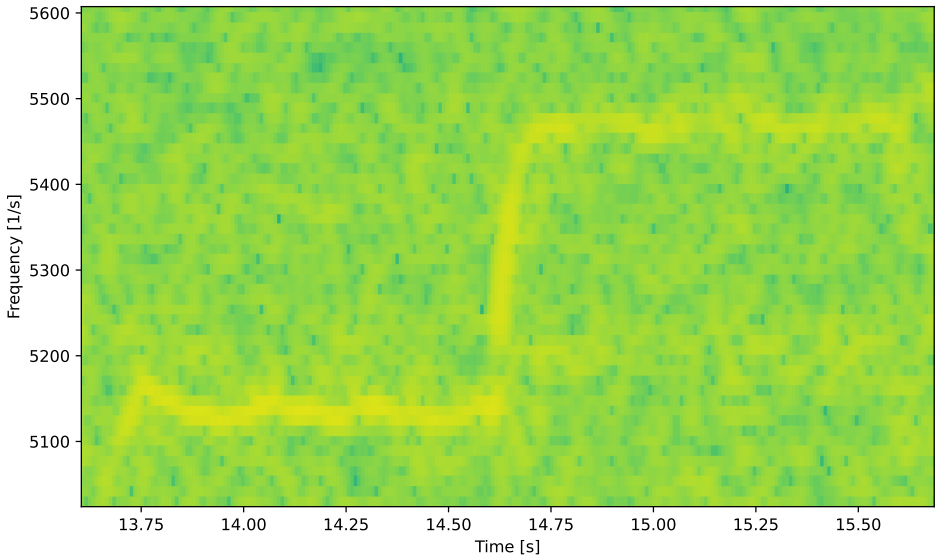


Figure 5.3: Fourier Transform of the sound signal during a motor test

(the propeller is spinning much slower than 5kHz) to estimate the time constant of a first order model at around $\tau = 0.06$sec, which is relatively slow.

The INDI controller was tuned by manually determining values, until the platform is somewhat stable on a safety rope. After that, logged data was used to improve the guess by matching the estimated actuator state to the accelerometer and gyro measurements. This resulted in the following effectiveness values:

```
<define name="G1_ROLL" value="{-6.5 , -13, -6.5, 6.5, 13, 6.5 }"/>
<define name="G1_PITCH" value="{+11.3 , 0, -11.3 , -11.3, 0, +11.3 }"/>
<define name="G1_YAW" value="{+0.18, -0.18, +0.18, -0.18, +0.18, -0.18}"/>
<define name="G1_THRUST" value="{-.4, -.4, -.4, -.4, -.4, -.4}"/>
<define name="G2" value="{30, -30, 30, -30, 30, -30}"/>
```

The final fits can be seen in Figure 5.4. Pitch and roll have good agreement, but yaw shows a slight lead in the predicted rotational jerk effect. Jerk was used to cancel out any constant offset effects, eg. from constant disturbances.

5.3.2. JSBSim Model

For better testing and debugging of the algorithms and flight plans, the JSBSim environment of Paparazzi was used to adapt an existing quadcopter model to an approximate hexacopter called `jsbsim/aircraft/flamewheel.xml`. Exact agreement of the motor dynamics and/or control effectiveness was not an objective; it is appropriate for sanity checks and for debugging purposes, but no gain tuning should be performed based on the simulator.

This approach was instrumental in preventing a number of crashes because even if the algorithm is verified, its integration with the rest of the flight control software had to be tested.

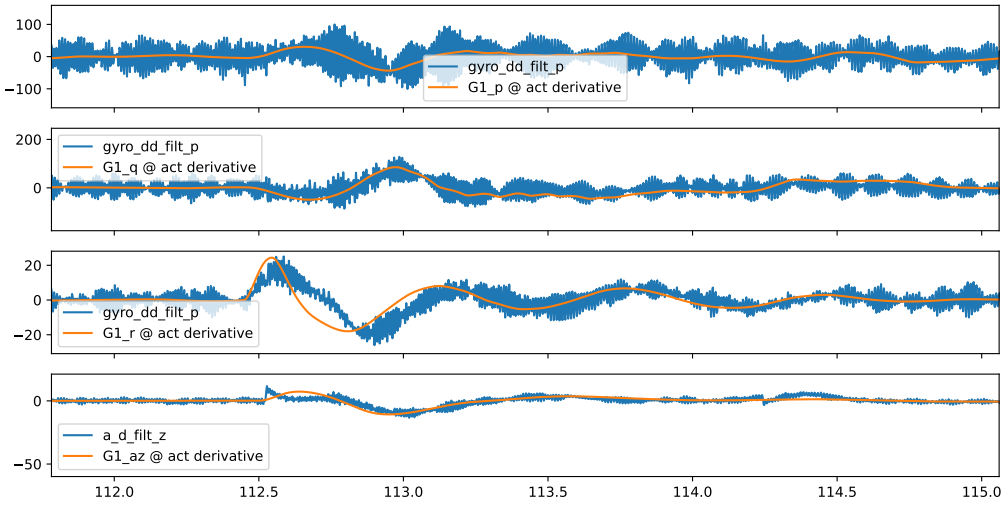


Figure 5.4: Rotational jerk comparison (first 3 plots) and linear jerk (last plot). Time on the x-axis in seconds.

Part IV

Closure

6 Conclusion

6.1. Closing Remarks

The work done in this project turned out to be quite detail-oriented and more in the direction of computer engineering, than expected at the start of the literature study. Often in engineering sciences, numerical solvers are slightly taken for granted, or simply selected, rather than programmed and/or optimised.

Some ideas that came up in the literature study and potentially held the answer to some parts of the research questions, had to be relegated to the recommendations (Chapter 7), in the interest of exploring the more relevant parts of the research questions sufficiently. I hope the interested reader will consider these!

6.2. Research Questions

Research Question 1

What possible changes to the numerical algorithms for optimising control allocation can be shown to provide significant improvements over the current state-of-the-art?

With this thesis it was demonstrated that there are unexplored algorithms and optimisation to algorithms in the field of constrained control allocation.

The literature study revealed possible approaches to the main research question repeated above, which at that point had not been investigated for the application in control allocation. These are restated below for convenience:

Research Question 1a

Can the speed of the active-set method for $\ell_2 - \ell_2$ norms be improved by iteratively updating its QR factors between iterations, using Given's rotations?

Research Question 1b

Can interior-point methods solve the $\ell_2 - \ell_1$ norm problem efficiently with accurate and balanced solutions?

Research Question 1c

Can the eigenvalue structure of the benchmark problems be exploited with Conjugate Gradient methods to make the active-set method for $\ell_2 - \ell_2$ norms more efficient?

Research Question 1d

Can the gradient projection method for the $\ell_2 - \ell_2$ norms improve speed?

Specifically, yes, it is possible to reduce the worst-case execution time of active-set algorithms using explicit factorisations that are updated during the iterations of the algorithm. QR factors are indeed updated efficiently using Given's rotations, improving runtime by 50% in a representative flight test. The first iteration of each algorithm is much slower, since the Q factor how to be recovered explicitly, but this has no significance for platforms with many actuators, as the worst-case execution time is seen when many actuators saturate and consequently many iterations are necessary.

When the distance measure for the penalty on not achieving the pseudo-control is linear, Interior Point may be very advantageous as the relevant matrix inverses can be computed very efficiently. However, this could not be confirmed in the present thesis work, as a stable implementation could not be achieved.

Gradient Projection methods were ruled out as they were not able to identify the active-set quicker and computing Lagrange Multipliers more often outweighed its computational benefits.

It was shown that Conjugate Gradient methods can indeed approach speeds of Cholesky factorisation and are easy to implement as they do not require factorisations and their update routines. However, the method is not numerically stable for the problems at hand, which may be addressed in future research using pre-conditioners.

6.3. Additional Conclusions

With more insights gained during the Thesis, additional conclusions were made.

Using a Cholesky decomposition directly on the unconstrained quadratic optimisation problem of an active-set algorithm is even more efficient than updating a QR decomposition, with -65% execution time over the baseline in the same flight test. The numerical issues occurring with this factorisation on current 32-bit flight controller hardware can be mitigated for multicopters by ensuring the problem is well-conditioned. It was shown how this can be achieved online with a computationally inexpensive method based on estimating the spectrum of a problem matrix online, at the disadvantage of slightly slower attitude tracking.

7

Recommendations

This chapter provides a brief overview of the primary recommendations for the future continuation of this research project.

Rec 1
Conditioning of the problem using the described approach (reducing the separation of the two objectives in the combined objective function) to avoid numerical problems with the Cholesky algorithm uncovered a tradeoff between accuracy and slower attitude tracking. A procedure to finding the correct parameter for different platforms effectiveness matrices may be developed in follow-up research.

Rec 2
Terminating an active-set algorithm sub-optimally may give much higher benefits than optimising the solution algorithms. In combination with warm-starting (ie using the previous solution as initial guess), the errors made by the premature termination may only persist for a few time steps. The effect this has on the attitude tracking may be small and should be investigated in future research.

Rec 3
The algorithm based on updating QR factorisation benefits from the simple problem structure (only bound constraints), but still a naive dense factorisation algorithm has been used. Future research could find a more specialised factorisation routine that avoids multiplying by 0 many times in the factorisation of the relevant matrix.

Rec 4
Future flight tests may be simplified with a smaller and more modern UAV platform. The hexacopter was used in this research mostly for availability, but provided challenges because its parameters had to be identified first and its power control electronics were relatively old and slow to respond.

Rec 5
The approach to generating a test set succeeded in generating pseudo-control vectors that result in different numbers of actuators to be saturated in both directions. However, the distribution of test vectors over these categories was not very uniform, such that the algorithms tests were biased either extremes: either many iterations or just a single iteration were necessary for most test cases.

Fair generation of these open-loop test cases may require different approach than the one described in the paper, which approximated the reachable control objective set using an ellipsoid, which was then randomly sampled inside (feasible) and outside (infeasible) its boundary. Future research could use the ellipsoid to generate a direction vector $\hat{\nu}$ whose magnitude is then scaled to ensure reaching saturation of 0 up to all actuators.

Rec 6
The analytical FLOPs counting approach shown only allows for estimating the relative execution time between algorithms, to some degree. Absolute predictions were not possible, likely due to the different levels of background activity on the real-time systems and some uncertainty in the clock-cycles per instruction factor. Also scaling of the execution time with problem size could not be reliably predicted. Only measuring the eventual algorithm on the eventual processing platform yielded definite answers with some variance, but clear ability to compare.

References

Björck, A. (1996). *Numerical methods for least squares problems*. SIAM.

Blaha, T. (2021). *Implementing embedded Drone Flight Controls for Offshore Surveillance VTOL Drone* (Internship Report AE5050). TU Delft. Delft.

Bodson, M. (2002). Evaluation of Optimization Methods for Control Allocation. *Journal of Guidance, Control, and Dynamics*, *25*(4), 703–711. https://doi.org/10.2514/2.4937

Bodson, M., & Frost, S. A. (2011). Load Balancing in Control Allocation. *Journal of Guidance, Control, and Dynamics*, *34*(2), 380–387. https://doi.org/10.2514/1.51952

Bodson, M., & Pohlchuck, W. A. (1998). Command Limiting in Reconfigurable Flight Control. *Journal of Guidance, Control, and Dynamics*, *21*(4), 639–646. https://doi.org/10.2514/2.4283

Bolender, M. A., & Doman, D. B. (2004). Nonlinear Control Allocation Using Piecewise Linear Functions. *Journal of Guidance, Control, and Dynamics*, *27*(6), 1017–1027. https://doi.org/10.2514/1.9546

Bordignon, K. A. (1996). *Constrained control allocation for systems with redundant control effectors* (Doctoral dissertation). Virginia Polytechnic Institute and State University. Blacksburg, VA. Retrieved December 9, 2021, from https://vtechworks.lib.vt.edu/handle/10919/28570

Brosgol, B., & Comar, C. (2021). DO-178C: A New Standard for Software Safety Certification.

Buffington, J. M., & Enns, D. F. (1996). Lyapunov stability analysis of daisy chain control allocation. *Journal of Guidance, Control, and Dynamics*, *19*(6), 1226–1230. https://doi.org/10.2514/3.21776

Burken, J. J., Lu, P., Wu, Z., & Bahm, C. (2001). Two Reconfigurable Flight-Control Design Methods: Robust Servomechanism and Control Allocation. *Journal of Guidance, Control, and Dynamics*, *24*(3), 482–493. https://doi.org/10.2514/2.4769

Cadzow, J. (1971). Algorithm for the minimum-effort problem. *IEEE Transactions on Automatic Control*, *16*(1), 60–63. https://doi.org/10.1109/TAC.1971.1099634

Cameron, D., & Princen, N. (2000). *Control allocation challenges and requirements for the Blended Wing Body*. https://doi.org/10.2514/6.2000-4539

Chatrath, K. (2019). *Vehicle Dynamics Control Using Control Allocation* (Master's thesis). Delft University of Technology. Delft. https://doi.org/10.13140/RG.2.2.35878.65608

Cimini, G., & Bemporad, A. (2017). Exact Complexity Certification of Active-Set Methods for Quadratic Programming. *IEEE Transactions on Automatic Control*, *62*(12), 6094–6109. https://doi.org/10.1109/TAC.2017.2696742

Cui, L., Zuo, Z., & Yang, Y. (2021). A Control-Theoretic Study on Iterative Solution to Control Allocation for Over-Actuated Aircraft. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *51*(6), 3429–3439. https://doi.org/10.1109/TSMC.2019.2924357

da Costa, R. R., Chu, Q. P., & Mulder, J. A. (2003). Reentry Flight Controller Design Using Nonlinear Dynamic Inversion. *Journal of Spacecraft and Rockets*, *40*(1), 64–71. https://doi.org/10.2514/2.3916

De Wagter, C., Remes, B., Smeur, E., Tienen, F., Ruijsink, R., Hecke, K., & Horst, E. (2020). *The NederDrone: A hybrid lift, hybrid energy hydrogen UAV* (Preprint). Retrieved January 14, 2022, from https://www.researchgate.net/publication/345654347_The_NederDrone_A_hybrid_lift_hybrid_energy_hydrogen_UAV

Deepa, S. N., & Sudha, G. (2016). Longitudinal control of aircraft dynamics based on optimization of PID parameters. *Thermophysics and Aeromechanics*, *23*(2), 185–194. https://doi.org/10.1134/S0869864316020049

Doman, D., & Sparks, A. (2002). Concepts for constrained control allocation of mixed quadratic and linear effectors. *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, *5*, 3729–3734 vol.5. https://doi.org/10.1109/ACC.2002.1024507

Dongarra, J. J., Moler, C. B., Bunch, J. R., & Stewart, G. W. (1979). *LINPACK Users' Guide*. SIAM. https://doi.org/10.1137/1.9781611971811

Durham, W., Bordignon, K. A., & Beck, R. (2017). *Aircraft control allocation*. Wiley.

Durham, W. C. (1993). Constrained control allocation. *Journal of Guidance, Control, and Dynamics*. https://doi.org/10.2514/3.21072

Durham, W. C. (2001). Computationally Efficient Control Allocation. *Journal of Guidance, Control, and Dynamics*, *24*(3), 519–524. https://doi.org/10.2514/2.4741

Eldén, L. (1982). A weighted pseudoinverse, generalized singular values, and constrained least squares problems. *BIT Numerical Mathematics*, *22*(4), 487–502. https://doi.org/10.1007/BF01934412

Enns, D. (1998). Control Allocation Approaches. *AIAA Guidance, Navigation, and Control Conference and Exhibit*. https://doi.org/10.2514/6.1998-4109

Glaze, M. L. (1998). *The Design and Implementation of a GUI-based Control Allocation Toolbox in the MATLAB environment* (Master's thesis). Virginia Polytechnic Institute and State University. Blacksburg, VA. Retrieved December 10, 2021, from https://vtechworks.lib.vt.edu/handle/10919/35625

Grechi, S., & Caiti, A. (2016). Comparison between Optimal Control Allocation with Mixed Quadratic & Linear Programming Techniques. *IFAC-PapersOnLine*, *49*(23), 147–152. https://doi.org/10.1016/j.ifacol.2016.10.335

Härkegård, O. (2002). Efficient active set algorithms for solving constrained least squares problems in aircraft control allocation. *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, *2*, 1295–1300. https://doi.org/10.1109/CDC.2002.1184694

Härkegård, O. (2004). Dynamic Control Allocation Using Constrained Quadratic Programming. *Journal of Guidance, Control, and Dynamics*, *27*(6), 1028–1034. https://doi.org/10.2514/1.11607

Härkegård, O. (2008). QCAT. Retrieved November 4, 2021, from https://www.mathworks.com/matlabcentral/fileexchange/4609-qcat

Hillier, F., & Lieberman, G. (2009). *Introduction to Operations Research*. McGraw-Hill Education.

Höppener, D. (2016). *Actuator Saturation Handling using Weighted Optimal Control Allocation Applied to an INDI Controlled Quadcopter* (Master's thesis). Delft University of Technology. Delft. http://resolver.tudelft.nl/uuid:3704b044-b9bf-454a-8678-0d140bd1d308

Intel. (2014). Optimizing Performance with Intel® Advanced Vector Extensions. Retrieved January 5, 2022, from https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/performance-xeon-e5-v3-advanced-vector-extensions-paper.pdf

Jin, J. (2005). Modified Pseudoinverse Redistribution Methods for Redundant Controls Allocation. *Journal of Guidance, Control, and Dynamics*, *28*(5), 1076–1079. https://doi.org/10.2514/1.14992

Johansen, T., & Fossen, T. (2013). Control allocation - A survey. *Autom.* https://doi.org/10.1016/j.automatica.2013.01.035

Johansen, T., Fossen, T., & Berge, S. (2004). Constrained nonlinear control allocation with singularity avoidance using sequential quadratic programming. *IEEE Transactions on Control Systems Technology*, *12*(1), 211–216. https://doi.org/10.1109/TCST.2003.821952

Lallman, F., Davidson, J., & Bundick, W. (2001). Integrated reconfigurable control allocation. *AIAA Guidance, Navigation, and Control Conference and Exhibit*. https://doi.org/10.2514/6.2001-4083

Lau, M., Yue, S., Ling, K., & Maciejowski, J. (2015). A Comparison of Interior Point and Active Set Methods for FPGA Implementation of Model Predictive Control. *Proc. European Control Conference*.

Matamoros, I. (2017). *Nonlinear Control Allocation for a High-Performance Tailless Aircraft with Innovative Control Effectors* (Master's thesis). Delft University of Technology. Delft.

Minka, T. (2021). Lightspeed matlab toolbox. Retrieved January 4, 2022, from https://github.com/tminka/lightspeed

Moler, C. B. (2000). MATLAB Incorporates LAPACK. Retrieved January 3, 2022, from https://nl.mathworks.com/company/newsletters/articles/matlab-incorporates-lapack.html

Moler, C. B. (2018). A Brief History of MATLAB. Retrieved January 3, 2022, from https://nl.mathworks.com/company/newsletters/articles/a-brief-history-of-matlab.html

Moré, J. J., & Toraldo, G. (1989). Algorithms for bound constrained quadratic programming problems. *Numerische Mathematik*, *55*(4), 377–400. https://doi.org/10.1007/BF01396045

Nathen, P., Strohmayer, A., Miller, R., Grimshaw, S., & Taylor, J. (2021). *Architectural performance assessment of an electric vertical take-off and landing (e-VTOL) aircraft based on a ducted vectored thrust concept* (Whitepaper). Retrieved January 14, 2022, from https://www.semanticscholar.org / paper / Architectural - performance - assessment - of - an - electric - Nathen - Strohmayer / 96113870bb6176a0818bf09330cb3125db44ed21

Nguyen Van, E., Troillard, P., Jézégou, J., Alazard, D., Pastor, P., & Döll, C. (2018). Reduction of Vertical Tail Using Differential Thrust: Influence on Flight Control and Certification. *Advanced Aircraft Efficiency in a Global Air Transport System (AEGATS'18)*, 1–8. Retrieved January 14, 2022, from https://hal.archives-ouvertes.fr/hal-02164023

Nocedal, J., & Wright, S. J. (2006). *Numerical Optimization*. Springer New York. https://doi.org/10.1007/978-0-387-40065-5

Oppenheimer, M. W., Doman, D. B., & Bolender, M. A. (2006). Control Allocation for Over-actuated Systems. *2006 14th Mediterranean Conference on Control and Automation*, 1–6. https://doi.org/10.1109/MED.2006.328750

Pachter, M., Chandler, P. R., & Mears, M. (1995). Reconfigurable tracking control with saturation. *Journal of Guidance, Control, and Dynamics*, *18*(5). https://doi.org/10.2514/3.21499

Petersen, J., & Bodson, M. (2006). Constrained quadratic programming techniques for control allocation. *IEEE Transactions on Control Systems Technology*, *14*(1), 91–98. https://doi.org/10.1109/TCST.2005.860516

Petersen, J., & Bodson, M. (1999). Fast control allocation using spherical coordinates. *AIAA Guidance, Navigation, and Control Conference and Exhibit*. https://doi.org/10.2514/6.1999-4215

Petersen, J., & Bodson, M. (2005). Interior-Point Algorithms for Control Allocation. *Journal of Guidance Control and Dynamics*, *28*, 471–480. https://doi.org/10.2514/1.5937

Sabharwal, A., & Potter, L. (1998). Convexly constrained linear inverse problems: Iterative least-squares and regularization. *IEEE Transactions on Signal Processing*, *46*(9), 2345–2352. https://doi.org/10.1109/78.709518

Sadien, E., Roos, C., Birouche, A., Carton, M., Grimault, C., Romana, L. E., & Basset, M. (2019). A detailed comparison of control allocation techniques on a realistic on-ground aircraft benchmark. *American Control Conference 2019*. Retrieved November 24, 2021, from https://hal.archives-ouvertes.fr/hal-02449260

Shahzad, A., Kerrigan, E. C., & Constantinides, G. A. (2010). A warm-start interior-point method for predictive control. *UKACC International Conference on Control 2010*, 1–6. https://doi.org/10.1049/ic.2010.0409

Sieberling, S., Chu, Q. P., & Mulder, J. A. (2010). Robust Flight Control Using Incremental Nonlinear Dynamic Inversion and Angular Acceleration Prediction. *Journal of Guidance, Control, and Dynamics*, *33*(6), 1732–1742. https://doi.org/10.2514/1.49978

Smeur, E. (2018). *Incremental Control of Hybrid Micro Air Vehicles* (Doctoral dissertation). Delft University of technology. https://doi.org/10.4233/UUID:23C338A1-8B34-40A6-89E9-997ADBDAFD75

Smeur, E. (2017). Working with INDI - PaparazziUAV. Retrieved January 14, 2022, from https://wiki.paparazziuav.org/wiki/Working_with_INDI#'Full'_INDI

Smeur, E., Höppener, D., & de Wagter, C. (2017). Prioritized Control Allocation for Quadrotors Subject to Saturation. *International Micro Air Vehicle Conference and Flight Competition 2017*. Retrieved November 4, 2021, from https://repository.tudelft.nl/islandora/object/uuid%3A54c4659a-3a6d-4d44-873d-aec6ae5e2376

Snell, S. A., Enns, D. F., & Garrard, W. L. (1992). Nonlinear inversion flight control for a supermaneuverable aircraft. *Journal of Guidance, Control, and Dynamics*, *15*(4), 976–984. https://doi.org/10.2514/3.20932

Stephan, J., & Fichter, W. (2017). Fast Exact Redistributed Pseudoinverse Method for Linear Actuation Systems. *IEEE Transactions on Control Systems Technology*, *PP*, 1–8. https://doi.org/10.1109/TCST.2017.2765622

STMicroelectronics. (2016). *Application note – Floating point unit demonstration on STM32 microcontrollers* (tech. rep. AN4044). Retrieved January 4, 2022, from https://www.st.com/resource/en/application_note/an4044-floating-point-unit-demonstration-on-stm32-microcontrollers-stmicroelectronics.pdf

The MathWorks, Inc. (2022). Quadratic programming - MATLAB quadprog. Retrieved January 4, 2022, from https://nl.mathworks.com/help/optim/ug/quadprog.html?s_tid=doc_ta#d123e135772

Tol, H., De Visser, C., Van Kampen, E.-J., & Chu, Q. (2014). Nonlinear Multivariate Spline Based Control Allocation for High Performance Aircraft. *Journal of Guidance Control and Dynamics*. https://doi.org/10.2514/1.G000065

Virnig, J., & Bodden, D. (1994). Multivariable control allocation and control law conditioning when control effectors limit. *Guidance, Navigation, and Control Conference*. https://doi.org/10.2514/6.1994-3609

Vollebregt, E. A. (2014). The Bound-Constrained Conjugate Gradient Method for Non-negative Matrices. *Journal of Optimization Theory and Applications*, *162*(3), 931–953. https://doi.org/10.1007/s10957-013-0499-x

Willee, H. (2021). Controller Diagrams | PX4 User Guide. Retrieved January 13, 2022, from https://docs.px4.io/master/en/flight_stack/controller_diagrams.html#airspeed-scaling

Willee, H., & Grob, M. (2021). Multicopter PID Tuning Guide (Manual/Advanced) | PX4 User Guide. Retrieved January 13, 2022, from https://docs.px4.io/master/en/config_mc/pid_tuning_guide_multicopter.html#airmode-mixer-saturation

Yu, Y., Elango, P., & Açıkmeşe, B. (2021). Proportional-Integral Projected Gradient Method for Model Predictive Control. *IEEE Control Systems Letters*, *5*(6), 2174–2179. https://doi.org/10.1109/LCSYS.2020.3044977

Zhang, J., Bhardwaj, P., Raab, S. A., Saboo, S., & Holzapfel, F. (2018). Control Allocation Framework for a Tilt-rotor Vertical Take-off and Landing Transition Aircraft Configuration. *2018 Applied Aerodynamics Conference*. https://doi.org/10.2514/6.2018-3480

Zhang, J., Söpper, M., & Holzapfel, F. (2021). Attainable Moment Set Optimization to Support Configuration Design: A Required Moment Set Based Approach. *Applied Sciences*, *11*(8), 3685. https://doi.org/10.3390/app11083685