Constrained Spacecraft Attitude Control Using Sequential Convex Programming

Nicholas Melville

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Aeronautics & Astronautics

University of Washington

2022

Reading Committee:

Mehran Mesbahi, Chair

Behçet Açikmeşe

Program Authorized to Offer Degree:
William E. Boeing Department of Aeronautics & Astronautics

University of Washington

Abstract

Constrained Spacecraft Attitude Control Using
Sequential Convex Programming

Nicholas Melville

Chair of the Supervisory Committee:
Professor Mehran Mesbahi
William E. Boeing Department of Aeronautics & Astronautics

This thesis describes how sequential convex programming can be used to guide and control the attitude of a spacecraft which is subject to pointing constraints. The constraints are expressed as convex contraints, and the dynamics are linearized and discretized using direct collocation and a piecewise linear approximation of the control to allow the problem to be solved quickly using sequential convex programming. The sequential convex program is initialized using the analytical solution to a simplified problem so that convergence is achieved in fewer subproblem iterations compared to other initialization methods. In addition, the collocation points are redistributed at each iteration to increase the density of collocation points near active constraints, preventing constraint violation. Finally, a simulation setup that includes disturbance torques and sensor and actuator dynamics is used demonstrate that a satellite can be controlled reliably using sequential convex programming without ever violating a constraint in a real-world environment.

TABLE OF CONTENTS

LIST OF FIGURES

NOMENCLATURE

AACT: The University of Washington Aeronautics and Astronautics Cubesat Team

CAC: Constrained Attitude Control

MEKF: Multiplicative Extended Kalman Filter

SCP: Sequential Convex Programming

SOAR: SOC-i's Optimal Attitude Reorientation

SOC-I: The Satellite for Optimal Control and Imaging

UW: The University of Washington

ACKNOWLEDGMENTS

To mom and dad, thank you for giving me the freedom to focus on my education and for always supporting and encouraging me. I am so lucky to have you as my parents.

To my siblings and friends: Will, Alex, Madison, Cam, Allison, Gordon, Tim, Matt, Chase, Ryan, Wes, and ye Olde Knights of Thirt, thank you for making me feel lighthearted despite the unrelenting stress of school and the crazy world we live in.

To the members of the cubesat team, past and present: Taylor, Charlie K, Sophia, Derek, Henry, Josh, Jay, Rithu, Gordon, Mitch, Charlie N, Tristan, Will, Hamzah, Nick, Alan, Evelyn, and so many others. Thank you for showing me all the best parts of teamwork and engineering. It is always a good day if I get to work on the cubesat. I cannot wait to see SOC-i go to space!

To my labmates in the RAIN Lab: Shahriar, Aditya, Mengyuan, Niyousha, Spencer, and Eddie, thank you for inspiring me to push myself to learn more than I could have ever imagined.

To Taylor Reynolds, thank you for being a role model for me, for meeting with me week after week for half a year, and for making and showing me how SOAR works. You have been one of the best mentors I've ever had.

To my advisor, Mehran Mesbahi, thank you for welcoming me into the RAIN lab, guiding me through this process, and helping me write this thesis.

To professor Dan Calderone and professor Sam Burden, thank you for teaching my favorite courses. I really like your teaching styles and your instruction gave me the skills I needed to write this thesis.

Lastly, thank you to the University of Washington, to the Air Force, and to my teachers,

my classmates, and any else who has helped me along the way. Coming to the University of Washington was a dream come true and it has been everything I hoped it would be.

Chapter 1
INTRODUCTION

The objective of this thesis is to demonstrate the effectiveness of real-time constrained spacecraft attitude control using sequential convex programming (SCP). A convex programming (or convex optimization) approach has only recently become feasible for real-time attitude control due to improvements in convex solvers and processors [8, 9], and offers many advantages over other methods. These advantages include the ability to add or change constraints on the fly, the ability to minimize a cost function, and the fact that many convex optimization solvers are guaranteed to converge to a solution in a certain amount of time [10, 11]. In contrast with other methods, these characteristics enable a greater level of autonomy for spacecraft in a wider range of environments, without sacrificing reliability [12, 13, 14]. Despite these advantages, this is a relatively new technology which has yet to be used on a real satellite, and there is still work to be done to enable future satellites to take full advantage of this approach.

1.1 Outline of Thesis

There are six chapters in this thesis: Chapter 1 introduces the constrained attitude control (CAC) problem, provides relevant background information, and describes some of the methods needed to discretize, convexify, and solve the CAC problem with SCP. Chapter 2 introduces SOAR[1] (SOC-i's (Satellite for Optimal Control and Imaging) Optimal Attitude Reorientation), a convex-optimization based algorithm which was made for a cubesat technology demonstration. Chapter 3 discusses a method to initialize the SCP which reduces the number of iterations required for convergence compared to other methods. Chapter 4

[1] Yes, SOAR is an acronym within an acronym.

discusses how the collocation points can be redistributed to avoid inter-sample constraint violation without without greatly increasing the solve time and without inflating or adding new constraints. Chapter 5 introduces a Simulink-based satellite model which is used to demonstrate the reliability of SOAR even when sensor and actuator dynamics and disturbance torques reduce the pointing accuracy of the satellite. Finally, Chapter 6 concludes the thesis and discusses areas of future work.

1.2 The Constrained Attitude Control Problem

The attitude constraints considered in this thesis are inclusion cones and exclusion cones. In other words, body frame vectors are defined which are required to stay within a certain angle of an inertial vector (inclusion constraint), or are required to stay a certain angle away from an inertial vector (exclusion constraint). Examples of these cone shaped constraints are shown in figure 1.1. These types of constraints are relevant, for example, for sun sensors, which need to have a view of the sun to provide a sun vector to be used for attitude determination, and for sensitive instruments such as optical payloads, which might be damaged if pointed at the sun or some other bright object. Other attitude constraints can be used, as long as they are convexified (converted into a convex form), but only cone constraints are considered in this thesis.

Perhaps the first spacecraft which had to solve the CAC problem was the Cassini spacecraft, which had sensitive instruments onboard that could be damaged by bright objects [15]. The Cassini engineers solved the CAC problem by commanding the spacecraft to "circumnavigate" the constrained attitude until a great circle arc was found that would take the spacecraft to its commanded attitude. For added safety, a constraint monitor algorithm would detect if the spacecraft was about to violate a constraint then redirect the spacecraft to a safe attitude. This method was computationally efficient enough to operate in real time on the Cassini spacecraft despite the computational limitations of the time.

In addition to the use of a constraint monitoring algorithm for CAC, four other types of CAC problem algorithms have been identified by Kim et al[16]. These include geometric

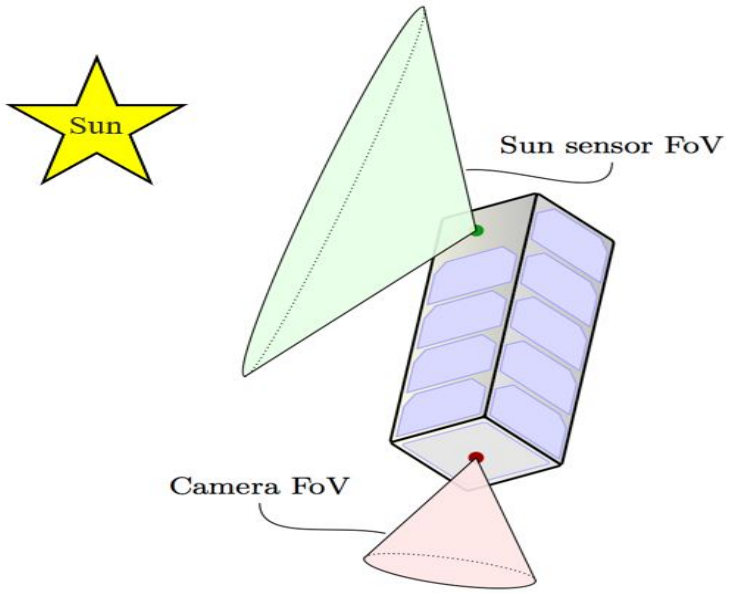


Figure 1.1: Illustration of Inclusion and Exclusion Constraints [1]

methods, randomized methods, potential function methods, and semidefinite programming methods. Geometric algorithms work by finding waypoints which will take the spacecraft to its commanded attitude without violating constraints. Randomized methods work by dividing the feasible state space into a graph, and using a path finding algorithm to find an efficient path to the commanded state [17]. Potential function methods use gradients of a potential function to find a path from the current state to the commanded state [18]. Each constraint has a potential function which creates a repellent "force" to keep the constraint from being violated. The constraint potential functions are combined with another potential function, which moves the state towards the commanded state [18]. Lastly, semidefinite methods (including SOAR) transcribe the problem into a convex optimization problem, specifically a semidefinite program. This method relies on the realization that an attitude cone constraint can be expressed as a quadratic constraint [19, 16], which can be solved using semidefinite programming (or second order cone programming, as shown in chapter 2). This is also why the problem has been called the quadratically constrained attitude

control problem. Although the dynamics of a satellite are nonlinear, this can be overcome by linearizing the dynamics. As a result, the CAC problem can be solved quickly using semidefinite programming.

Kim et al [16] also discuss four specific types of quadratic attitude constraints (cone constraints). These are static hard constraints (i.e. a camera cannot point at a fixed celestial body for any amount of time), static soft constraints (i.e. a camera cannot point at a celestial body for more than a certain amount of time), dynamic constraints (i.e. a camera cannot pointing at a moving object, such as another spacecraft), and mixed constraints (combinations of the other three constraints). It will be demonstrated that SOAR can handle all four of these types of constraints.

1.3 Optimal Control

The CAC problem can be expressed as an optimal control problem. The solution to an optimal control problem is a control function, which affects a given dynamical system in such a way to globally minimize (or maximize) a given cost function, subject to constraints on the control and the state of the system. The equations below describe a general optimal control problem:

The dynamics:

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t)),\ t \in [t_0, t_f] \tag{1.1}$$

The cost:

$$J = \boldsymbol{\phi}(\boldsymbol{x}(t_f), t_f) + \int_{t_0}^{t_f} L(\boldsymbol{x}(t), \boldsymbol{u}(t), t)dt \tag{1.2}$$

The constraints:

$$\boldsymbol{x}(t) \in \mathcal{X}(t),\ \boldsymbol{u}(t) \in \mathcal{U}(t) \tag{1.3}$$

Optimization problems are often written in standard form, shown below:

$$\begin{aligned} \min_{x,u} \quad & J(x,u) \\ \text{s.t.} \quad & \dot{\boldsymbol{x}}(t) = f(\boldsymbol{x(t)}, \boldsymbol{u}(t)) \\ & \boldsymbol{x}(t) \in \mathcal{X}(t),\ \boldsymbol{u}(t) \in \mathcal{U}(t) \end{aligned} \tag{1.4}$$

These equations are very general, and can be used to describe many different optimization problems. For example, 1.4 can be used to describe the optimization of pushing a mass from one point to another in one dimension in the shortest possible time[2]:

$$\begin{aligned} \min \quad & t_f \\ \text{s.t.} \quad & \dot{x}_1(t) = x_2,\ \dot{x}_2(t) = u \\ & x_1(t_0) = 0, x_2(t_0) = 0 \\ & x_1(t_f) = x_f, x_2(t_f) = 0 \\ & u_{min} < u < u_{max} \end{aligned} \tag{1.5}$$

Even the most complicated optimal control problems can be expressed in this format [4, 21, 22, 23, 24, 25, 26]. This format will be used to define the CAC problem in chapter 2.

1.4 The Maximum Principle

A milestone in optimal control theory was reached when Lev Pontryagin and his students invented the Maximum Principle (this principle will be leveraged in chapter 3 to initialize SOAR). The principle provides the necessary conditions for an optimal control, and is derived using the calculus of variations [27]. Essentially, the maximum principle states that given an optimal control and some performance parameter which needs to be maximized (or minimized, depending on the problem), any perturbation in the optimal control should result in a lower performance parameter. This means that the derivative of the performance

[2]The first optimal control problem ever solved was also a shortest time, or "Brachistochrone" problem [20].

parameter with respect to the perturbation should be zero, if the given control is in fact optimal. This requirement is fundamental to the maximum principle. Constraints and dynamics are also taken into account using Lagrange multipliers. A derivation of this principle can be found in chapter 3 of the Optimal Control textbook by Lewis and Syrmos [28], and the result is a set of equations which are the necessary conditions for optimal control. They are:

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}, t) \tag{1.6}$$

$$J = \boldsymbol{\phi}(\boldsymbol{x}(t_f), t_f) + \int_{t_0}^{t_f} L(\boldsymbol{x}(t), \boldsymbol{u}(t), t)dt \tag{1.7}$$

$$\Psi(\boldsymbol{x}(t_f), t_f) = 0 \tag{1.8}$$

$$H(\boldsymbol{x}, \boldsymbol{u}, t) = L(\boldsymbol{x}, \boldsymbol{u}, t) + \lambda^\top f(\boldsymbol{x}, \boldsymbol{u}, t) \tag{1.9}$$

$$\dot{\boldsymbol{x}} = \frac{\partial H}{\partial \lambda} = f \tag{1.10}$$

$$-\dot{\lambda} = \frac{\partial H}{\partial \boldsymbol{x}} = \frac{\partial f^\top}{\partial \boldsymbol{x}}\lambda + \frac{\partial L}{\partial \boldsymbol{x}} \tag{1.11}$$

$$0 = \frac{\partial H}{\partial \boldsymbol{u}} = \frac{\partial L}{\partial \boldsymbol{u}} + \frac{\partial f^\top}{\partial \boldsymbol{u}}\lambda \tag{1.12}$$

Where equation 1.8 uses a new variable Ψ to defines a final-time constraint, $H(\boldsymbol{x}, \boldsymbol{u}, t)$ is the Hamiltonian, equation 1.10 defines the "state equation," equation 1.11 defines the costate equation, and equation 1.12 defines the stationarity equation. With the maximum principle, solving an optimal control problems is done by finding state, costate (λ), and control functions which satisfy all of these equations. The maximum principle can be used

to solve many optimal control problems, including 1.5![3] Even problems which cannot be solved directly using the maximum principle, like the CAC problem, still make use of the maximum principle, using a property called duality. Duality is the principle that optimal control problems can be viewed from two perspectives, one using the state equation, and one using the costate equation. Respectively, these two perspectives are know as the primal and dual problem, and provide upper and lower bounds on the optimal solution[2]. Many optimization algorithms, including the convex optimization solvers used by SOAR, make use of this property.

1.5 Convex Optimization

Convex optimization is a type of optimization (see equation 1.4) that involves a convex cost function and convex constraints, which can be solved quickly. One critical step in sequential convex programming is the transcription of the original problem into a convex problem [11, 10]. This section describes how convex optimization works using simple examples and information from the textbook "Convex Optimization" by Boyd and Vanderberghe [2].

A convex function is a function such that the line between any two points on the graph of the function, is above the graph of the function. This is illustrated below:

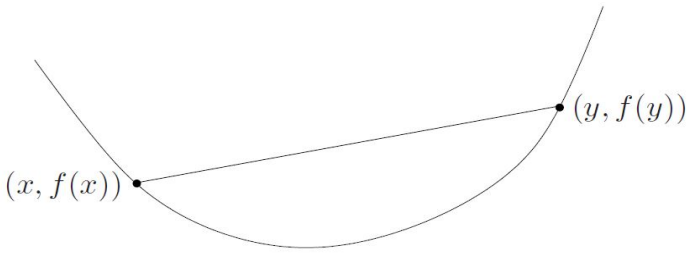


Figure 1.2: A convex function [2]

[3]This is left as an excercise to the reader, but is a great application of the maximum principle for someone just learning about optimal control. Many derivations of the solution to the double-integrator minimum time problem can be found online.

Another way to describe a convex function, would be to say that its second derivative is nonnegative, or that it curves upwards everywhere. A local minimum of a convex function can be found quickly by randomly picking a starting point, determining the direction in which the function decreases, and "stepping" in that direction. Eventually, assuming the step size converges to zero at an appropriate rate, a minimum of the function will be found (this is known as gradient descent). Since the function is convex, the only local minimum is also the global minimum. Importantly, it can also be shown that the global minimum will be found in less than a certain amount of time (polynomial time) [2]. Many convex solvers, which are algorithms that find the global minimum of convex functions, take advantage of these properties. Therefore, if an optimal control problem can be formulated as the minimization of a convex function, it can be solved quickly.

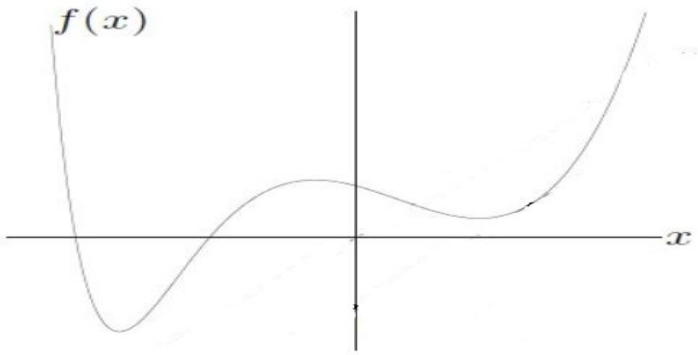


Figure 1.3: A non-convex function [2]

In contrast, figure 1.3 shows an example of a non-convex function[2]. A gradient descent method for this non-convex function, depending on the initial point, would not necessarily find the global minimum. For some non-convex functions it might take an infinite amount of time for a solver to find the global minimum.

The addition of constraints can also complicate an attempt to find the minimum value of a convex function. However, if the constraints form a convex set, then the solution can still be found quickly. A convex set, similar to a convex function, is a set such that the line

connecting any two points in the set is also in the set, as shown in figure 1.4.

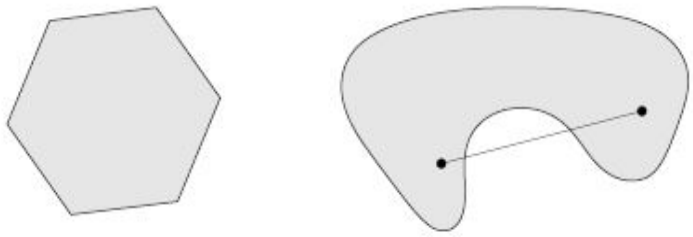


Figure 1.4: A convex and a non-convex set [2]

Just like with a non-convex function, if the constraints are non-convex, then a gradient descent algorithm could get stuck in a non-convex part of the set, and thus fail to find the global minimum.

SOAR utilizes these properties to find an optimal solution to the CAC problem quickly. The CAC problem is transcribed into a convex optimization problem using properties of convex sets and functions. Many of these properties can be found in Boyd and Vandenberghe's textbook, but the relevant ones for SOAR are that the intersection of two convex sets are convex (in other words, multiple convex constraints create a convex set), and that certain constraints, such as affine constraints, quadratic constraints, second order cone constraints, and norm inequality constraints, are convex. All of the constraints in the CAC problem can be converted into an equivalent convex constraint. In addition, the dynamics of the problem, which are essentially just another constraint, can be linearized about a reference trajectory, resulting in another convex constraint. Thus, given a cost function which is also convex, the CAC problem is converted into a convex optimization problem.

1.6 Direct Collocation

Many optimal control problems (including the CAC problem) are continuous, which means that the control at infinitely many moments in time must be determined to solve the problem.

While such a control can sometimes be found analytically, using the maximum principle for example, some problems are not easily solved analytically. In this case, the direct collocation method can be used [3, 29]. The key to this method is to convert the infinite dimensional control into a finite dimensional approximation of the control. For example, a curve can be approximated using a set of straight lines, as shown below:

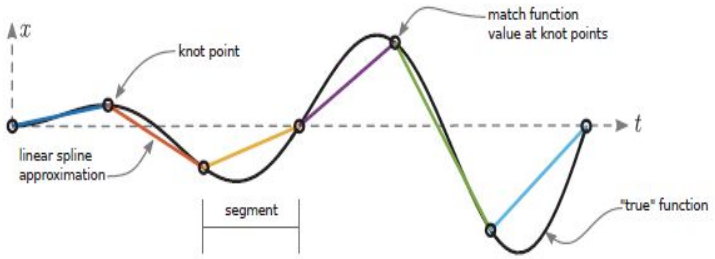


Figure 1.5: A piecewise linear approximation of a function [3]

This approximation is known as a piecewise linear approximation, or a linear spline approximation. Higher and lower order approximations are possible, but SOAR uses a piecewise linear approximation, so that method will be presented here. If the control function is approximated using N-1 line segments, then the solver just needs to find N points in control space, and linearly interpolate between the points to approximate the optimal continuous control function. In addition, the constraints and dynamics are also discretized based on the affine control approximation. The discretized points are called collocation points. Using this method for N evenly spaced (in time) collocation points, problem 1.5 becomes:

$$
\begin{aligned}
\min \quad & t_f \\
\text{s.t.} \quad & x_{1,n+1} = x_{1,n} + \tfrac{t_f}{2N}(x_{2,n} + x_{2,n+1}) \\
& x_{2,n+1} = x_{2,n} + \tfrac{t_f}{2N}(u_n + u_{n+1}) \\
& x_{1,1} = x_0,\ x_{2,1} = 0 \\
& x_{1,N} = x_f,\ x_{2,N} = 0 \\
& u_{min} < u < u_{max}
\end{aligned}
\tag{1.13}
$$

For $n \in [1, 2, ..., N]$. This problem can formulated as a convex optimization problem, and solved quickly. While the solution to this problem will not necessarily be the same as the actual optimal solution due to the discretization, it is often close enough, and the solution can be approximated with arbitrary accuracy by increase the number of collocation points.

1.7 Sequential Convex Programming

As was mentioned in 1.5, the CAC problem can be approximated as a convex optimization problem, which can be solved quickly, but this solution is not necessarily the solution to the original CAC problem. However, if solution to the convex problem is used as the reference trajectory about which the dynamics are linearized to create a new convex optimization problem, and the solutions to these iterations of convex optimization problems converge (similar to gradient descent), then the solution to the original CAC problem has been found. In other words, since the linear approximation of the dynamics is accurate near the reference trajectory, if the solution to the convex problem is not too different from the original reference trajectory, then this convex problem solution is also a solution to the CAC problem. This process is known as sequential convex programming (SCP) and it is used in a wide variety of aerospace applications [4, 22, 24, 25, 30]. In SCP, each convex optimization problem is called a subproblem. One caveat of SCP is that the solution to these iterations of subproblems is not necessarily the global minimum, since the original CAC problem is not convex, but these locally optimal solutions are typically good enough.

There are two common problems that arise during SCP. These are artificial unbound-

edness, and artificial infeasibility (infeasibility means a solution which satisfies all the constraints does not exist). These are illustrated by figure 1.6 from [4]. The dashed blue line is an affine approximation of the blue quadratic constraint. Assuming no other constraints, the minimum cost (on the dashed blueline) would be negative infinity, even though the actual optimal value (on the solid blue line) is zero. This is artificial unboundedness. Also, if the green constraint and red inequality constraint are also considered, the convex approximation appears to have no solution, since the dashed blue line and green line do not intersect to the left of the red inequality constraint, even though the blue line and the green line actually do intersect to the left of the red inequality constraint. This is artificial infeasibility.

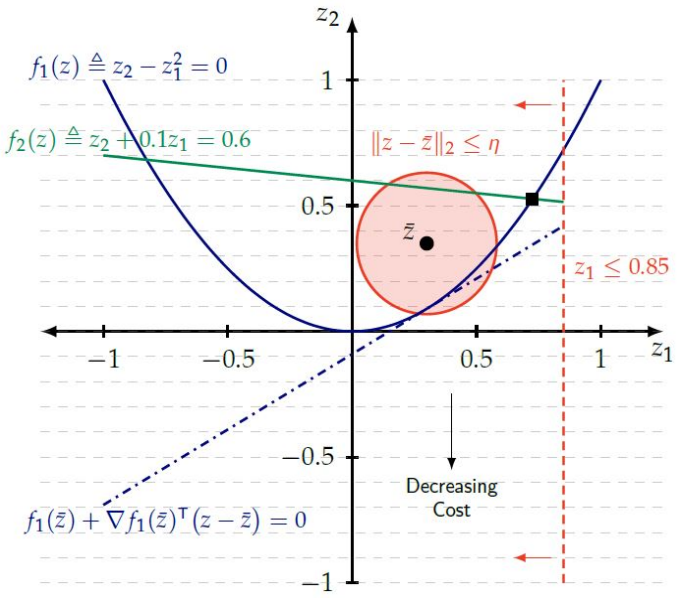


Figure 1.6: Illustration of Artificial Infeasibility and Unboundedness [4]

One solution to unboundedness is to limit the amount that the solution is allowed to change between iterations, using a trust region, which is analagous to reducing the step size of a gradient descent to achieve convergence. This has the added benefit of keeping the solution to each subproblem close to the reference trajectory, where the affine approximation of the dynamics is more accurate, and can also sometimes speed up the convergence. One

solution to infeasibility is to use virtual control, which allows any trajectory to be feasible, but the trajectories which use virtual control are heavily penalized, so that the solution not require any virtual control. The use of virtual control can also speed up convergence [4].

In SCP, similar to gradient descent, the initial guess is important. Different initial guesses will result in different solutions if the function has more than one minimum, which might be the case for a non-convex optimization problem. In addition, guesses which are closer to the optimal solution will result in faster convergence. This idea will be explored further in chapter 4.

1.8 SOC-i

Much of this work was inspired by the Satellite for Optimal Control and Imaging (SOC-i), which is being built by the UW Aeronautics and Astronautics Cubesat Team (AACT), and will go to space as early as June 2022. SOC-i aims to be the first satellite to demonstrate convex optimization based attitude control in space using SOAR [1, 30]. The SOAR algorithm is described in greater detail in the next chapter.

Chapter 2

SOAR

SOC-i's Optimal Attitude Reorientation (SOAR) uses direct collocation and sequential convex programming to control the attitude of satellite with cone inclusion and exclusion contraints. In addition, SOAR also accounts for constraints on reorientation time, torque, and rotation rate, and minimizes the energy required for the reorientation [5]. This chapter describes how it is formulated as a sequential convex program, and solved with a second order cone solver called ECOS [31].

2.1 Satellite Dynamics

The first step is to define the dynamics of the satellite. The state and control vectors are defined as

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{q}(t) \\ \boldsymbol{h}_B(t) \\ \boldsymbol{h}_W(t) \end{bmatrix}_{10\times 1} \qquad \boldsymbol{\tau} = \begin{bmatrix} \tau_1(t) \\ \tau_2(t) \\ \tau_3(t) \end{bmatrix}_{3\times 1} \tag{2.1}$$

The state includes the attitude quaternion $\boldsymbol{q} \in \boldsymbol{R^4}$, the angular momentum of the satellite bus $\boldsymbol{h}_B \in R^3$, and the net angular momentum of the reaction wheels $\boldsymbol{h}_W \in R^3$. The control vector is the x-y-z components of the net torque[1]. The dynamics of the state are defined as

[1]Using the net torque of the reaction wheel assembly instead of the torque from each reaction wheel allows SOAR to be used with satellites with any number of reaction wheels. The net torque just needs to be allocated to the individual wheels using any allocation method.

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}) = \begin{bmatrix} \frac{1}{2}\boldsymbol{q} \bigotimes (\boldsymbol{J}^{-1}\boldsymbol{h}_B) \\ -\boldsymbol{\tau} + (\boldsymbol{h}_B + \boldsymbol{h}_W)^{\times}(\boldsymbol{J}^{-1}\boldsymbol{h}_B) \\ \boldsymbol{\tau} \end{bmatrix} \tag{2.2}$$

Additionally, the energy $\boldsymbol{E}$ can be approximated in terms of the control input as

$$\boldsymbol{E} = \int_{t_0}^{t_f} \boldsymbol{\tau}^{\top}\boldsymbol{\tau} dt \tag{2.3}$$

Equation 2.3 will serve as a cost function for SOAR. This approximation is based on the fact that reaction wheel torque is approximately proportional to current, and power is approximately proportional to current squared. Thus, the total energy used during a reorientation is proportional to the integral of the torque squared. Note that this approximation is not necessarily accurate for all configurations of reaction wheels, but equation 2.3 still serves as a sufficient cost function for SOAR.

2.2 Standard Form

Now that the dynamics have been defined, the problem can be formulated as an optimization problem in standard form:

$$\begin{aligned}
\min \quad & \int_{t_0}^{t_f} \boldsymbol{\tau}^\top \boldsymbol{\tau} dt \\
\text{s.t.} \quad & \dot{\boldsymbol{q}} = \tfrac{1}{2}\boldsymbol{q} \bigotimes (\boldsymbol{J}^{-1}\boldsymbol{h}_B) \\
& \dot{\boldsymbol{h}}_B = -\boldsymbol{\tau} + (\boldsymbol{h}_B + \boldsymbol{h}_W)^\times (\boldsymbol{J}^{-1}\boldsymbol{h}_B) \\
& \dot{\boldsymbol{h}}_W = \boldsymbol{\tau} \\
& \|\boldsymbol{J}^{-1}\boldsymbol{h}_B\|_\infty \leq \boldsymbol{\omega}_{max} \\
& \|\boldsymbol{\tau}\|_\infty \leq \boldsymbol{\tau}_{max} \\
& \theta(t) < \theta_{max} \\
& \phi(t) > \phi_{max} \\
& t_{min} \leq t_f \leq t_{max} \\
& \boldsymbol{q}(t_0) = \boldsymbol{q}_0, \boldsymbol{h}_b(t_0) = \boldsymbol{h}_{b,0}, \boldsymbol{h}_w(t_0) = \boldsymbol{h}_{w,0} \\
& \boldsymbol{q}(t_f) = \boldsymbol{q}_f, \boldsymbol{h}_b(t_f) = \boldsymbol{h}_{b,f}
\end{aligned} \tag{2.4}$$

Where the vectors θ and ϕ are the angles between the constrained body frame vector and the corresponding inertial frame vector and define the inclusion and exclusion cone constraints respectively, and $\boldsymbol{q}_0, \boldsymbol{h}_{b,0}, \boldsymbol{h}_{w,0}, \boldsymbol{q}_f$, and $\boldsymbol{h}_{b,f}$ define the initial and final conditions for the attitude, body angular momentum, and reaction wheel net angular momentum.

This problem is not yet easily solveable because the dynamics are nonlinear, some of the constraints are not convex, and the continuous dynamics of the problem mean that infinitely many points need to be optimized. The following sections describe how this problem can be converted into a sequential convex program which can be solved quickly.

2.3 Quadratic Attitude Constraints

The pointing cone constraints can be defined as a quadratic inequality constraint in terms of the attitude quaternion with the following equation:

$$\boldsymbol{q}^\top \boldsymbol{M_{e/i}} \boldsymbol{q} < 2 \tag{2.5}$$

where M is computed with the following equations:

$$\boldsymbol{M}_i = \left(2\boldsymbol{I}_{4\times4} - \begin{bmatrix} \boldsymbol{y}^\top \boldsymbol{x} & \boldsymbol{y}^\times x \\ \boldsymbol{y}^\times \boldsymbol{x} & \boldsymbol{y}\boldsymbol{x}^\top + \boldsymbol{x}\boldsymbol{y}^\top - \boldsymbol{y}^\top \boldsymbol{x}\boldsymbol{I}_{3\times3} \end{bmatrix} + cos(\theta)\boldsymbol{I}_{4\times4} \right)^{\frac{1}{2}} \tag{2.6}$$

$$\boldsymbol{M}_e = \left(2\boldsymbol{I}_{4\times4} + \begin{bmatrix} \boldsymbol{y}^\top \boldsymbol{x} & \boldsymbol{y}^\times \boldsymbol{x} \\ \boldsymbol{y}^\times x & \boldsymbol{y}\boldsymbol{x}^\top + \boldsymbol{x}\boldsymbol{y}^\top - \boldsymbol{y}^\top \boldsymbol{x}\boldsymbol{I}_{3\times3} \end{bmatrix} - cos(\theta)\boldsymbol{I}_{4\times4} \right)^{\frac{1}{2}} \tag{2.7}$$

Where $\boldsymbol{x}$ is the unit inertial vector and $\boldsymbol{y}$ is the unit body-fixed vector. $\boldsymbol{M}_i$ is used for an inclusion constraint in equation 2.5, and $\boldsymbol{M}_e$ is used for an exclusion constraint in equation 2.5[19, 16].

2.4 Time Normalization

This section shows how the problem can be solved as a fixed final time problem, which is easier to solve, without limiting the actual reorientation to take place in a fixed amount of time. To achieve this, the reorientation is assumed to take place in a fixed amount of time, which is called the normalized time $\tau \in [0, 1]$. A new solution variable $s = \frac{dt}{d\tau}$ is added which linearly scales the dynamics of the problem. Therefore, the dynamics are redefined as

$$\dot{\boldsymbol{x}} = \frac{d\boldsymbol{x}}{dt} = \frac{d\boldsymbol{x}}{d\tau}\frac{d\tau}{dt} \quad \Rightarrow \quad \frac{d\boldsymbol{x}}{d\tau} = \frac{dt}{d\tau} f(\boldsymbol{x}, \boldsymbol{u}) = s\, f(\boldsymbol{x}, \boldsymbol{u}) \tag{2.8}$$

The variable s is equivalent to the final time t_f since the normalized time is defined between $[0, 1]$, and is included in the problem as a solution variable to be optimized. As a result the problem is essentially solved as a fixed-final time problem, yet the solver is still able to change the maneuver time.

2.5 Linearization

The dynamics are linearized about a reference trajectory so that they are an affine approximation of the solution variables. Given a reference trajectory $\bar{z}(\tau)$, the equations become

$$\frac{d\boldsymbol{x}}{d\tau} \approx \bar{s}f(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}) + \boldsymbol{A}(\bar{\boldsymbol{z}}(\tau))(\boldsymbol{x}(\tau) - \bar{\boldsymbol{x}}(\tau)) + B(\bar{\boldsymbol{z}}(\tau))(\boldsymbol{u}(\tau) - \bar{\boldsymbol{u}}(\tau)) + S(\bar{\boldsymbol{z}}(\tau))(s - \bar{s}) \quad (2.9)$$

$$= \boldsymbol{A}(\bar{\boldsymbol{z}})\boldsymbol{x} + \boldsymbol{B}(\bar{\boldsymbol{z}})\boldsymbol{u} + \boldsymbol{S}(\bar{\boldsymbol{z}})s + \boldsymbol{R}(\bar{\boldsymbol{z}}) \quad (2.10)$$

where,

$$\boldsymbol{A}(\bar{\boldsymbol{z}}) = \frac{d}{d\boldsymbol{x}} sf(\boldsymbol{x}, \boldsymbol{u}) = s \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} & \boldsymbol{0}_{4\times3} \\ \boldsymbol{0}_{3\times4} & \boldsymbol{A}_{22} & \boldsymbol{A}_{23} \\ \boldsymbol{0}_{3\times4} & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times4} \end{bmatrix} \quad (2.11)$$

$$\boldsymbol{B}(\bar{\boldsymbol{z}}) = \frac{d}{d\boldsymbol{u}} sf(\boldsymbol{x}, \boldsymbol{u}) = s \begin{bmatrix} \boldsymbol{0}_{4\times3} \\ -\boldsymbol{I}_{3\times3} \\ \boldsymbol{I}_{3\times3} \end{bmatrix} \quad (2.12)$$

$$\boldsymbol{S}(\bar{\boldsymbol{z}}) = \frac{d}{ds} sf(\boldsymbol{x}, \boldsymbol{u}) = f(\boldsymbol{x}, \boldsymbol{u}) \quad (2.13)$$

$$\boldsymbol{R}(\bar{\boldsymbol{z}}) = -\boldsymbol{A}(\bar{\boldsymbol{z}})\bar{\boldsymbol{x}} - \boldsymbol{B}(\bar{\boldsymbol{z}})\bar{\boldsymbol{u}} - \boldsymbol{S}(\bar{\boldsymbol{z}})\bar{s} \quad (2.14)$$

and given $\boldsymbol{\omega}_B = \boldsymbol{J}^{-1}\boldsymbol{h}_B$,

$$\boldsymbol{A}_{11} = \frac{1}{2} \begin{bmatrix} 0 & -\boldsymbol{\omega}_B^\top \\ \boldsymbol{\omega}_B^\top & \boldsymbol{\omega}_B^\times \end{bmatrix} \quad (2.15)$$

$$\boldsymbol{A}_{12} = \frac{1}{2} \begin{bmatrix} -\boldsymbol{q}_v^\top \\ \boldsymbol{q}_v^\times + q_4 \boldsymbol{I}_{3\times3} \end{bmatrix} \quad (2.16)$$

$$\boldsymbol{A}_{22} = (\boldsymbol{h}_B^\times + \boldsymbol{h}_W^\times)\boldsymbol{J}^{-1} - \boldsymbol{\omega}_B^\times \quad (2.17)$$

$$\boldsymbol{A}_{23} = -\boldsymbol{\omega}_B^\times \quad (2.18)$$

2.6 Discretization

Direct collocation and a piecewise linear approximation of the control is used to discretize the problem. Another example of a linear piecewise function is illustrated in figure 2.2.

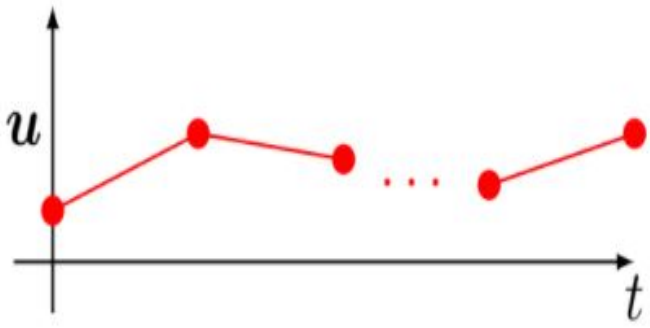


Figure 2.1: Illustration of Linear Piecewise Control [5]

With this approximation, the number of parameters that need to be optimized is reduced from infinitely many to finitely many, since the solver just needs to find a handful of points in control space. During the maneuver, the continuous torque is calculated as a linear interpolation between the two nearest points, with the following equation:

$$\boldsymbol{u}(\tau) = \frac{t_{k+1} - t}{t_{k+1} - t_k}\boldsymbol{u}_k + \frac{t - t_k}{t_{k+1} - t_k}\boldsymbol{u}_{k+1} = \lambda_-(\tau)\boldsymbol{u}_k + \lambda_+(\tau)\boldsymbol{u}_{k+1} \tag{2.19}$$

Where $\{\boldsymbol{u}_k\}_{k=1}^N$ is the set of control points which the solver needs to find at each point $\tau_k \in [0, 1]$ in time. While collocation points are typically evenly distributed in time, chapter 4 will investigate adjusting the collocation point distribution for better performance. With this discretization, the dynamics can be rewritten as

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}_{d,k}\boldsymbol{x}_k + \boldsymbol{B}^-_{d,k}\boldsymbol{u}_k + \boldsymbol{B}^+_{d,k}\boldsymbol{u}_{k+1} + \boldsymbol{S}_{d,k}s + \boldsymbol{R}_{d,k} \tag{2.20}$$

Where $\boldsymbol{A}_{d,k}$, $\boldsymbol{B}^-_{d,k}$, $\boldsymbol{B}^+_{d,k}$, $\boldsymbol{S}_{d,k}$, and $\boldsymbol{R}_{d,k}$ are

$$\boldsymbol{A}_{d,k} = \boldsymbol{\Phi}(\tau_{k+1,\tau_k}) \tag{2.21a}$$

$$\boldsymbol{B}^{-}_{d,k} = \boldsymbol{A}_{d,k} \int_{\tau_k}^{\tau_{k+1}} \boldsymbol{\Phi}^{-1}(\tau_{k+1}, \tau_k)\boldsymbol{B}(\tau)\lambda_{-}(\tau)d\tau \tag{2.21b}$$

$$\boldsymbol{B}^{+}_{d,k} = \boldsymbol{A}_{d,k} \int_{\tau_k}^{\tau_{k+1}} \boldsymbol{\Phi}^{-1}(\tau_{k+1}, \tau_k)\boldsymbol{B}(\tau)\lambda_{+}(\tau)d\tau \tag{2.21c}$$

$$\boldsymbol{S}_{d,k} = \boldsymbol{A}_{d,k} \int_{\tau_k}^{\tau_{k+1}} \boldsymbol{\Phi}^{-1}(\tau_{k+1}, \tau_k)\boldsymbol{S}(\tau)d\tau \tag{2.21d}$$

$$\boldsymbol{R}_{d,k} = \boldsymbol{A}_{d,k} \int_{\tau_k}^{\tau_{k+1}} \boldsymbol{\Phi}^{-1}(\tau_{k+1}, \tau_k)\boldsymbol{R}(\tau)d\tau \tag{2.21e}$$

and the discrete state transition matrix $\boldsymbol{\Phi}(\tau_{k+1}, \tau_k)$ is related to the continuous state transition matrix by

$$\boldsymbol{\Phi}(\tau_{k+1}, \tau_k) = \boldsymbol{I} + \int_{\tau_k}^{\tau} \boldsymbol{A}(t)\boldsymbol{\Phi}(t, \tau_k)dt \tag{2.22}$$

for a continuous state transition matrix $\boldsymbol{\Phi}(t, \tau)$, which is the matrix which relates the state at time t to the state at time τ. Thus, equation 2.20 defines the discrete dynamics of the problem [14]. For N collocation points, the dynamics of the problem are enforced by stacking equation 2.20 $N - 1$ times,

$$\boldsymbol{X} = \boldsymbol{A}\boldsymbol{X} + \boldsymbol{B}\boldsymbol{U} + \boldsymbol{S}s + \boldsymbol{R} \tag{2.23}$$

The concatenated state and control vectors $\boldsymbol{X}$ and $\boldsymbol{U}$ are

$$\boldsymbol{X} = [\boldsymbol{x}_1^\top \; \boldsymbol{x}_2^\top \; ... \; \boldsymbol{x}_N^\top] \tag{2.24}$$

$$\boldsymbol{U} = [\boldsymbol{u}_1^\top \; \boldsymbol{u}_2^\top \; ... \; \boldsymbol{u}_N^\top] \tag{2.25}$$

2.7 Virtual Control

Virtual control is used to deal with artificial infeasibility and to speed up convergence. A new virtual control parameter is added for every state at every collocation point. As a result, any infeasible state can be changed by the solver using these new control parameters to make it feasible. These new virtual control parameters do not follow any dynamics and are unconstrained. Since it is already necessary to solve this linearized subproblem multiple times (see section 1.7), it is okay if virtual control is used as long as the last iterate uses no virtual control. The virtual control is heavily penalized in the cost to encourage it to go to zero. Only solutions with near zero virtual control should be used. The dynamics equation becomes

$$\boldsymbol{X} = \boldsymbol{A}\boldsymbol{X} + \boldsymbol{B}\boldsymbol{U} + \boldsymbol{S}s + \boldsymbol{R} + \boldsymbol{V} \tag{2.26}$$

Where $\boldsymbol{V} \in R^{10N}$ is the virtual control.

2.8 The Convex Subproblem

As a result of sections 2.3 - 2.7, the problem can be written as a convex optimization problem:

$$\begin{aligned}
\min \quad & \gamma + \boldsymbol{\omega}_v^\top \boldsymbol{\eta} \\
\text{s.t.} \quad & \boldsymbol{X} = \boldsymbol{A}\boldsymbol{X} + \boldsymbol{B}\boldsymbol{U} + \boldsymbol{S}s + \boldsymbol{R} + \boldsymbol{V} \\
& \|\boldsymbol{H}_h \boldsymbol{x}_k\|_\infty \leq \omega_{max},\ k = 1, ..., N \\
& \|\boldsymbol{u}_k\|_\infty \leq u_{max},\ k = 1, ..., N \\
& \boldsymbol{x}_k^\top \boldsymbol{H}_q^\top \boldsymbol{M}_e \boldsymbol{H}_q \boldsymbol{x}_k \ \leq 2, k = 1, ..., N \\
& \boldsymbol{x}_k^\top \boldsymbol{H}_q^\top \boldsymbol{M}_i \boldsymbol{H}_q \boldsymbol{x}_k \ \leq 2, k = 1, ..., N \\
& t_{f,min} \leq s \leq t_{f,max} \\
& -\eta_j \leq V_j \leq \eta_j, j = 1, ..., 10N \\
& \boldsymbol{U}^\top \boldsymbol{H}_t^\top \boldsymbol{H}_t \boldsymbol{U} \leq \gamma \\
& \boldsymbol{x}_1 = [\boldsymbol{q}_0^\top \ \boldsymbol{h}_{B,0}^\top \ \boldsymbol{h}_{w,0}^\top]^\top \\
& \boldsymbol{H}_f \boldsymbol{x}_N = [\boldsymbol{q}_f^\top \ \boldsymbol{0}_{1\times 3}^\top]^\top
\end{aligned} \tag{2.27}$$

Where $\boldsymbol{H}_h$ converts the state to angular velocity:

$$\boldsymbol{H}_h = [\boldsymbol{0}_{4\times 4} \ \boldsymbol{J}^{-1} \ \boldsymbol{0}_{3\times 3}] \tag{2.28}$$

$\boldsymbol{H}_q$ selects the quaternion of the state:

$$\boldsymbol{H}_q = [\boldsymbol{I}_{4\times 4} \ \boldsymbol{0}_{4\times 6}] \tag{2.29}$$

$\boldsymbol{H}_f$ selects the quaternion and momentum of the body

$$\boldsymbol{H}_f = [\boldsymbol{I}_{7\times 7} \ \boldsymbol{0}_{7\times 3}] \tag{2.30}$$

And $\boldsymbol{H}_t$ is used to achieve trapezoidal integration of the control:

$$\boldsymbol{H}_t = [1 \ 1 \ 1] \bigotimes \left[\sqrt{\frac{1}{2}} \ 1 \ ... \ 1 \ \sqrt{\frac{1}{2}}\right]_{1\times N} \tag{2.31}$$

Where $\bigotimes$ represents the Kronecker product. Note that equation 2.31 comes from trapezoidal integration of each term of the control vector.

The new slack variables γ and $\boldsymbol{\eta}$ are used to make the cost linear (this is known as a relaxation [4]), and to allow negative virtual control values to penalized (if just the virtual control values were used in the cost function, then negative values would reduce the cost, and virtual control would be encouraged). The virtual control penalty $\boldsymbol{\omega}_v$ can be increased until the final iteration has sufficiently low virtual control.

2.9 Transcription to Second Order Cone Program

The quadratic constraints ($\boldsymbol{U}^\top \boldsymbol{H}_t^\top \boldsymbol{H}_t \boldsymbol{U} \leq \gamma$, $\boldsymbol{x}_k^\top \boldsymbol{H}_q^\top \boldsymbol{M}_i \boldsymbol{H}_q \boldsymbol{x}_k < 2$, and $\boldsymbol{x}_k^\top \boldsymbol{H}_q^\top \boldsymbol{M}_e \boldsymbol{H}_q \boldsymbol{x}_k < 2$) can be written as cone constraints:

$$\|\boldsymbol{H}_t U\|_2 < \bar{\gamma} \tag{2.32}$$

$$\|\boldsymbol{M}_i^{1/2} \boldsymbol{H}_q \boldsymbol{x}_k\|_2 < 2^{1/2} \tag{2.33}$$

$$\|\boldsymbol{M}_e^{1/2} \boldsymbol{H}_q \boldsymbol{x}_k\|_2 < 2^{1/2} \tag{2.34}$$

Where γ is now actually proportional to the square root of the energy used during the maneuver, but still works as a slack variable. All of the remaining constraints are equality constraint or inequality constraints, and the cost is affine. Therefore, by concatenating all of the constraints together, the convex subproblem can described compactly as:

$$\begin{aligned} \min \quad & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{s.t.} \quad & \|\boldsymbol{A}_c(i)\boldsymbol{x} - \boldsymbol{b}_c(i)\|_2 \leq \boldsymbol{c}_c(i)^\top \boldsymbol{x} - d_c(i) \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h} \end{aligned} \tag{2.35}$$

Where each i defines a cone constraint. This can be solved with any second order cone solver. For this thesis ECOS [31] is used because of its fast solve time. Coneprog [32] from Matlab's Optimization Toolbox also works, but takes longer.

2.10 Extension For Other Constraints

The first version of SOAR, which will operate in space on SOC-i, only includes hard pointing constraints. However, soft pointing constraints, dynamic constraints, and mixed constraints can be used as well. Soft constraints can be defined as second order cone constraints:

$$\|\boldsymbol{M}_s\boldsymbol{H}_s\boldsymbol{X}\|_2 < d_s \tag{2.36}$$

Where $\boldsymbol{M}_s$ is a diagonal matrix containing N sets of $\boldsymbol{M}_e$ or $\boldsymbol{M}_i$, $\boldsymbol{H}_s$ is a diagonal matrix containing N sets of $\boldsymbol{H}_q$:

$$\boldsymbol{M}_s = \boldsymbol{I}_{N\times N} \otimes \boldsymbol{M}_{e/i} \tag{2.37}$$

$$\boldsymbol{H}_s = \boldsymbol{I}_{N\times N} \otimes \boldsymbol{H}_q \tag{2.38}$$

$\boldsymbol{X}$ is the state trajectory in a column vector, and d_s is calculated so that whatever physical constraint that this discrete soft constraint is based on is satisfied. Note that d_s can be updated at each iteration to account for discretization and convexification error. Alternatively, weights could be applied to each block entry ($\boldsymbol{M}_{e/i}$) in $\boldsymbol{M}_s$ to better approximate the physical constraint as a convex function, but simply updating d_s is preferred due to its simplicity.

Dynamic pointing constraints can be considered simply by predicting how the inertial vector will change throughout the course of the maneuver. Lastly, mixed constraints can be used too since second order cone solvers work with arbitrarily many constraints as long as the convex problem remains feasible.

2.11 Summary of SOAR

The algorithm below summarizes SOAR:

Algorithm 1 SOAR

1: Compute initial state X_0, control U_0 and maneuver time s_0 ▷ More info in ch. 3
2: **while** $\Delta \boldsymbol{Z} < \Delta \boldsymbol{Z}_{min}$ and $\|\boldsymbol{V}\|_1 < V_{max}$ **do**
3: Integrate equations 2.21a-2.21e
4: Set up Constraint Matrices $\boldsymbol{A}_c$,$\boldsymbol{b}_c$,$\boldsymbol{c}_c$,$\boldsymbol{d}_c$,$\boldsymbol{A}$,$\boldsymbol{b}$,$\boldsymbol{G}$,$\boldsymbol{h}$
5: Calculate new $\boldsymbol{Z}$ with ECOS
6: Calculate $\Delta \boldsymbol{Z}$
7: **end while**

Where Z is the solution vector:

$$\boldsymbol{Z} = [\boldsymbol{X}^\top \boldsymbol{U}^\top \gamma \, s \, \boldsymbol{V}^\top \boldsymbol{\eta}^\top] \in R^{33N+2} \tag{2.39}$$

In addition to computing open loop trajectories and control, SOAR also uses feedback to more closely follow trajectories [33]. Matlab code for SOAR can be downloaded from Github [1].

SOAR is a very capable guidance algorithm, but there is still room for improvements. As is the case with many optimization problems, the initial guess has a big effect on the time it takes for the solver to converge. The next chapter shows how SOAR can be initialized to greatly reduce solve time.

Chapter 3
INITIALIZATION

As was mentioned in the previous chapter, an initial reference trajectory of is needed to linearize the dynamics. While SOAR will converge to a feasible solution for almost any initial reference trajectory, the number of subproblem iterations required and overall solve time vary depending on the initial reference trajectory [34]. This chapter shows how a reference trajectory can be found by analytically solving a one degree of freedom (1DOF) satellite reorientation using the maximum principle (see 1.4), and that this reference trajectory results in a lower overall solve time compared to an alternative method in which the initial and commanded states are interpolated to create a reference trajectory.

3.1 Finding an Initial Trajectory Using the Maximum Principle

The goal is to find an initial reference trajectory which is as close as possible to the optimal solution, without using a significant amount of computation time. This goal relies on the assumption that the closer the initial guess is to the optimal solution, the faster the optimal solution will be found. There is no known analytical solution to the CAC problem. However, the CAC problem is very similar to a one degree of freedom double-integrator system. This is the case if the satellite is constrained to only rotate about one axis, the exclusion/inclusion constraints are ignored, and the stored angular momentum is assumed to be zero. With these assumptions, the problem reduces to a one degree of freedom (1DOF) double-integrator system which can be controlled optimally using the maximum principle. Luckily, most of the assumptions are not totally unfounded, since the shortest rotation from one attitude to another is just a rotation about a single axis, and a satellite will typically use magnetorquers or some other actuators to keep the reaction wheel momentum near zero. Ignoring the

inclusion/exclusion constraints is not really a good assumption, but the solution to the 1DOF problem is still closer to the ultimate solution compared to other methods. One final assumption is that the maneuver will take the maximum allowed time $t_{f,max}$, which is also a good assumption since longer maneuvers require less torque which will minimize the cost function. The simplified problem is shown below:

$$\begin{aligned} \min \quad & \int_0^{t_f} u(t)^2 dt \\ \text{s.t.} \quad & \dot{\theta}(t) = \ \omega, \ \ \dot{\omega}(t) = u(t) \\ & \theta(t_0) = 0, \ \omega(t_0) = 0 \\ & \theta(t_f) = \theta_f, \ \omega(t_f) = 0 \\ & -\ u_{max} < u(t) < \ u_{max} \\ & -\ \omega_{max} < \ \omega(t) < \ \omega_{max} \end{aligned} \tag{3.1}$$

Where the states θ and ω are the angle and angular rate of the satellite rotating about a fixed axis, u_{max} and ω_{max} are the maximum angular acceleration and angular rate about the axis of rotation, and $t \in [0, 1]$. Note that the control is assumed to be equal to acceleration,and the reorientation is assumed to take just one second so that the problem is as simple as possible. The following equations can be used to calculate θ_f, u_{max}, v_{max}, and the axis of rotation $\boldsymbol{v}$.

$$q_{error} = q_f \otimes q_0^* = \begin{bmatrix} q0 \\ \boldsymbol{q}_v \end{bmatrix}_{error} \tag{3.2}$$

$$\theta_f = 2cos^{-1}(q_0) \tag{3.3}$$

$$\boldsymbol{v} = \frac{\boldsymbol{q_v}}{\|\boldsymbol{q_v}\|_2} \tag{3.4}$$

$$u_{max} = \frac{\tau_{max}}{\|\boldsymbol{J}\boldsymbol{v}\|_\infty} \tag{3.5}$$

$$\omega_{max} = \frac{\|\boldsymbol{v}\|_2}{\|\boldsymbol{v}\|_\infty}\omega_{max} \tag{3.6}$$

Where $\otimes$ represents quaternion multiplication. Note that the torque and rate constraints must be adjusted because the constraints in the 3-dof problem are ∞-norm constraints, and are thus different depending on the axis of rotation.

This 1DOF problem can be solved analytically using the maximum principle. The maximum principle equations are written below:

The Hamiltonian:

$$H = u^2 + \lambda_1 x_2 + \lambda_2 u + \mu_1(\omega^2 - \omega_{max}^2) + \mu_2(u^2 - u_{max}^2) \tag{3.7}$$

Where μ_1 and μ_2 are used to enforce the constraints on u and ω, and are positive respectively if $(u^2 - \omega_{max}^2)$ or $(\omega^2 - \omega_{max})$ are positive (i.e. the constraint would violated), and zero otherwise [35]. λ_1 and λ_2 enforce the dynamics.

The Costate Equation:

$$\begin{bmatrix} \dot{\lambda_1} \\ \dot{\lambda_2} \end{bmatrix} = - \begin{bmatrix} 0 \\ \lambda_1 \end{bmatrix} - \begin{bmatrix} 0 \\ 2\omega \end{bmatrix} \mu_1 \tag{3.8}$$

The Stationarity Condition:

$$0 = 2u + \lambda_2 + \mu_2(2u) \tag{3.9}$$

These equations give enough information to determine an analytical solution to the 1DOF problem. The first step in this derivation is to notice that there are four different cases depending on whether or not the rate constraint and the torque constraint is active. There will be a separate control function for each of these cases.

3.1.1 Case 1: No Active Constraints

In this case, μ_1 and μ_2 are zero. Therefore:

$$\begin{bmatrix} \dot{\lambda_1} \\ \dot{\lambda_2} \end{bmatrix} = - \begin{bmatrix} 0 \\ \lambda_1 \end{bmatrix} \rightarrow \lambda_2 = -\lambda_1 t + \lambda_{2,0} \quad (\lambda_1, \lambda_{2,0} \text{ are constants}) \tag{3.10}$$

$$0 = 2u + \lambda_2 \rightarrow u = \frac{\lambda_1}{2} t - \frac{\lambda_{2,0}}{2} \tag{3.11}$$

$$u = c_1 t + c_2 \tag{3.12}$$

The solution to this problem is thus a linear control function like the function shown in figure 3.1, and can be found by calculating the constants c_1 and c_2 that result in a rest to rest reorientation of the appropriate angle in $t_{f,max}$ seconds.

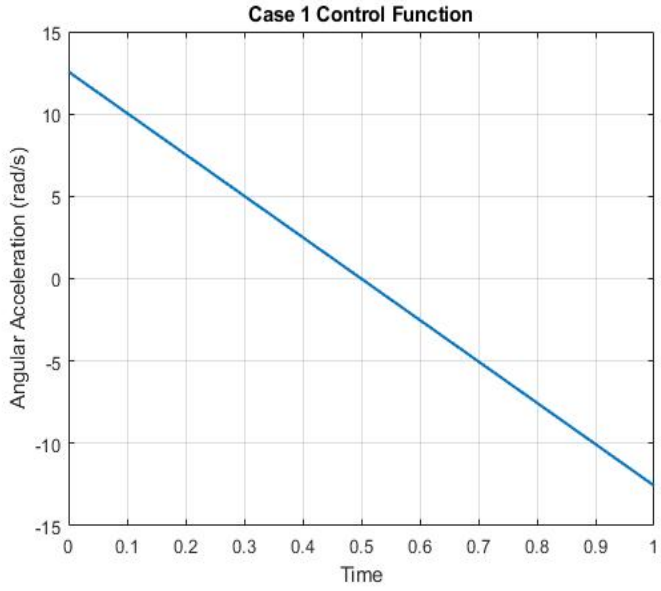


Figure 3.1: Initialization Control Function for No Active Constraints

Integrating the control yields equations for θ and ω (note that $t_f = 1$):

$$\omega(t) = \tfrac{1}{2}c_1 t^2 + c_2 t \tag{3.13}$$

$$\omega(t_f) = \tfrac{1}{2}c_1 + c_2 = 0 \tag{3.14}$$

$$\theta(t) = \tfrac{1}{6}c_1 t^3 + \tfrac{1}{2}c_2 t^2 \tag{3.15}$$

$$\theta(t_f) = \tfrac{1}{6}c_1 + \tfrac{1}{2}c_2 = \theta_f \tag{3.16}$$

Solving this system of equations for c_1 and c_2 yields an expression for the control:

$$u(t) = -12\theta_f t + 6\theta_f \tag{3.17}$$

3.1.2 Case 2: Rate Constraint Active

The next step in the derivation is to notice that whenever μ_1 is active, the satellite is rotating at its maximum rate, which means the control must neither increase the rotation rate (because that would violate a constraint), nor decrease the rotation rate, (because then μ_1 would be zero by definition and the optimal control will be linear again). In addition, because of the symmetry in the problem, the torque before and after the midway point of the maneuver should be equal and opposite. Therefore, the optimal control can be defined as a piecewise function:

$$u(t) = \begin{cases} u_0 - u_0 \frac{t}{t_1} & t < t_1 \\ 0 & t_1 \leq t < t_f - t_1 \\ \frac{(t - t_f + t_1)}{t_1} u_0 & t_f - t_1 \leq t \leq t_f \end{cases}$$

An example of this control function is shown in figure 3.2. Integrating yields a system of equations:

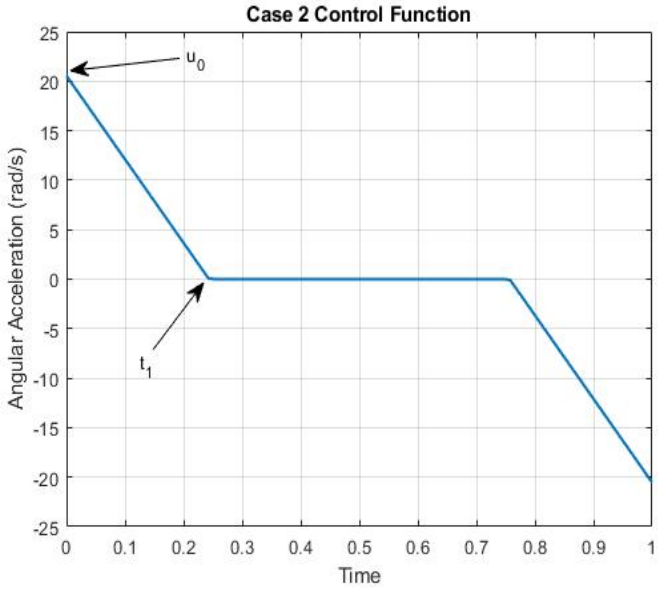


Figure 3.2: Initialization Control Function for Active Rate Constraint

$$\omega_{max} = \frac{1}{2} u_0 t_1 \tag{3.18}$$

$$\frac{\theta_f}{2} = u_0 \frac{t_1^2}{3} + \omega_{max}(\tfrac{1}{2} - t_1) \tag{3.19}$$

Solving these equations yields an equation for t_1 and u_0, and thus an analytical solution for the optimal control:

$$t_1 = \frac{3\theta_f - \omega_{max}}{2\omega_{max}} \tag{3.20}$$

$$u_0 = \frac{2\omega_{max}}{t_1} \tag{3.21}$$

3.1.3 Case 3: Torque Constraint Active

Similar to the case where the rate constraint is active, whenever μ_2 is active, the max torque is being applied and the torque can neither increase (because that would violate a constraint), nor decrease (because then μ_2 would be zero by definition). Similarly, the optimal control can be defined as a piecewise function.

$$u(t) = \begin{cases} u_{max} & t < t_1 \\ u_{max} - \frac{2(t-t_1)}{1-2t_1} u_{max} & t_1 \leq t < t_f - t_1 \\ u_{max} & t_f - t_1 \leq t \leq t_f \end{cases}$$

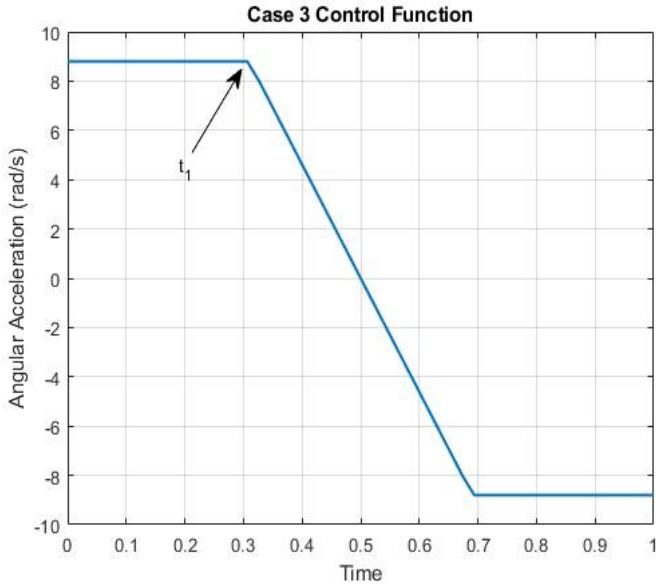


Figure 3.3: Initialization Control Function for Active Torque Constraint

An example of this control function is shown in figure 3.3. Again, integrating yields a system of equations:

$$\omega(t) = u_{max}t_1 + u_{max}(t - t_1) - u_{max}\frac{(t-t_1)^2}{1-2t_1}, \;\; t \in [t_1, \tfrac{1}{2}] \tag{3.22}$$

$$\theta_f = 2\left[\frac{1}{2}u_{max}t_1^2 + u_{max}t_1(\tfrac{1}{2} - t_1) + \tfrac{1}{3}u_{max}(\tfrac{1}{2} - t_1)^2\right] \tag{3.23}$$

Solving equation 3.23 yields an expression for t_1 and thus an analytical solution for the optimal control:

$$t_1 = \frac{1}{2} - \frac{3}{2}\sqrt{\frac{(u_{max} - 4\theta_f)}{3u_{max}}} \tag{3.24}$$

3.1.4 Case 4: Torque and Rate Constraint Active

This case combines combines the previous two cases. Therefore, the control is defined as the following piecewise function:

$$u(t) = \begin{cases} u_{max} & t < t_1 \\ u_{max} - \frac{(t-t_1)}{t_2-t_1}u_{max} & t_1 \leq t < t_2 \\ 0 & t_2 \leq t < t_f - t_2 \\ \frac{(t-t_f+t_2)}{t_2-t_1}u_{max} & t_f - t_2 \leq t < t_f - t_1 \\ u_{max} & t_f - t_1 \leq t \leq t_f \end{cases}$$

An example of this control is shown in figure 3.4. Once again, integrating yields equations:

$$\omega_{max} = \frac{(t_1 + t_2)u_{max}}{2} \tag{3.25}$$

$$\theta_f = 2\left[\tfrac{1}{2}u_{max}t_1^2 + u_{max}t_1(t_2 - t_1) + \tfrac{1}{3}u_{max}(t_2 - t_1)^2 + \omega_{max}(\tfrac{1}{2} - t_2)\right] \tag{3.26}$$

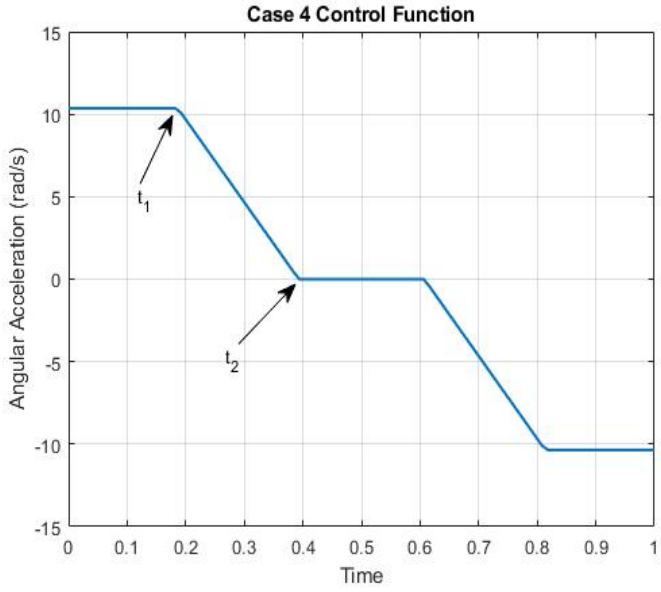


Figure 3.4: Initialization Control Function for Active Torque and Rate Constraints

Solving these equations yields an analytical solution for t_1 and t_2:

$$t_1 = \frac{\omega_{max} - 3\sqrt{\frac{1}{3}(-\omega_{max}^2 + u_{max}\omega_{max} - \theta_f u_{max})}}{u_{max}} \tag{3.27}$$

$$t_2 = \frac{\omega_{max} + 3\sqrt{\frac{1}{3}(-\omega_{max}^2 + u_{max}\omega_{max} - \theta_f u_{max})}}{u_{max}} \tag{3.28}$$

The final step is to convert the solution of the 1DOF problem to an initial trajectory for the original problem. The equations below show how this can be done:

$$\boldsymbol{u}(t) = -\frac{u(t)\boldsymbol{J}\boldsymbol{v}}{t_{f,max}^2} \tag{3.29}$$

$$\boldsymbol{q}_{error}(t) = \begin{bmatrix} \cos(\frac{\theta(t)}{2}) \\ \sin(\frac{\theta(t)}{2})\boldsymbol{v} \end{bmatrix}_{error} \tag{3.30}$$

$$\boldsymbol{q}(t) = \boldsymbol{q}_0 \otimes \boldsymbol{q}_{error}(t)^* \tag{3.31}$$

$$\boldsymbol{h}_b(t) = \frac{\omega(t)J\boldsymbol{v}}{t_{f,max}} \tag{3.32}$$

$$\boldsymbol{h}_w(t) = -\boldsymbol{h}_b(t) \tag{3.33}$$

Where $\theta(t)$ and $\omega(t)$ can be calculated by integrating the optimal control functions of the 1DOF problem. Also note that in equations 3.29 and 3.32, the functions are scaled based on the total reorientation time $t_{f,max}$, since the 1DOF problem assumes that the reorientation time t_f is 1.

3.1.5 Case 5: Exclusion Constraints Active

If the exclusion constraint is active, the 1DOF solution is not that close to actual optimal solution. In addition, there are some rare configurations of constraints and initial conditions where initialization using the 1DOF solution will cause SOAR to never find a feasible solution, which can be the case when the initial trajectory takes the body frame boresight vector through a region where two exclusion constraints overlap slightly. Therefore, the 1DOF method is not necessarily better than other methods whenever an exclusion constraint is active. There are other methods that can be used to find an initial feasible trajectory (see 1.2). However, these other initialization methods provide diminishing returns with added complexity, and are beyond the scope of this thesis. The key takeaway of this section is simply that SOAR works best if its initial reference trajectory is both close to the optimal solution and feasible. If faster convergence is desired for SOAR (or potentially for any other trajectory optimization problem), then finding a feasible and near optimal initial reference trajectory is likely to help.

3.2 Comparison of 1DOF Initialization with Interpolation Initialization

Another common method for initializing a trajectory optimization problem is to simply interpolate from the initial state to the final state. To demonstrate the effectiveness of the 1DOF initialization method, this section compares the performance of the 1DOF initial trajectory and the performance of an interpolated initial trajectory, in which the angular momentum $\boldsymbol{h}_b$ and the wheel momentum $\boldsymbol{h}_w$ are linearly interpolated and the attitude $\boldsymbol{q}$

is spherically interpolated from the initial to the final state, and the control is zero. A 120 degree reorientation with $N = 30$ collocation points was used to test each initialization method, and the torque constraint T_{max} and rate constraint ω_{max} were varied to ensure that each case was tested. In addition, an exclusion constraint was added to also test the case where an exclusion constraint is active. Table 3.1 shows a comparison of the average solve times for 10 trials and the number of subproblem iterations between these two methods, and figures 3.5-3.10 compare the states and control of both the converged solution and the two initial reference trajectories.

Table 3.1: Initialization Method Performance

Method	Case	$\omega_{max}(rad/s)$	$T_{max}(Nm)$	Iterations	Mean Solve Time (s)
1DOF	1	0.15	2.8E-3	1	0.7820
Interp	1	0.15	2.8E-3	2	1.0792
1DOF	2	0.15	2.8E-3	1	0.6312
Interp	2	0.1	2.8E-3	2	1.135
1DOF	3	0.15	8.5E-5	1	0.6115
Interp	3	0.15	8.5E-5	4	2.0880
1DOF	4	0.12	1E-4	1	0.6945
Interp	4	0.12	1E-4	4	2.0820
1DOF	5	0.15	2.8E-3	4	2.0470
Interp	5	0.15	2.8E-3	4	2.0034

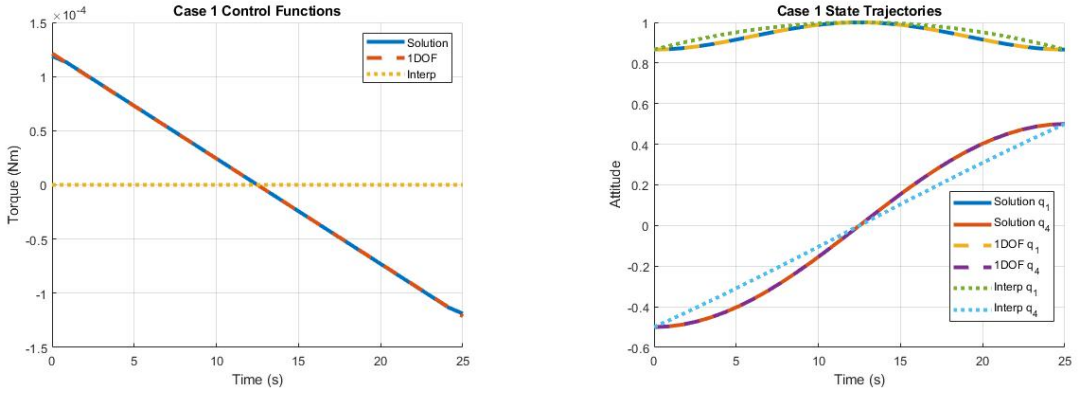


Figure 3.5: Comparison of State and Control Trajectories for No Active Constraints

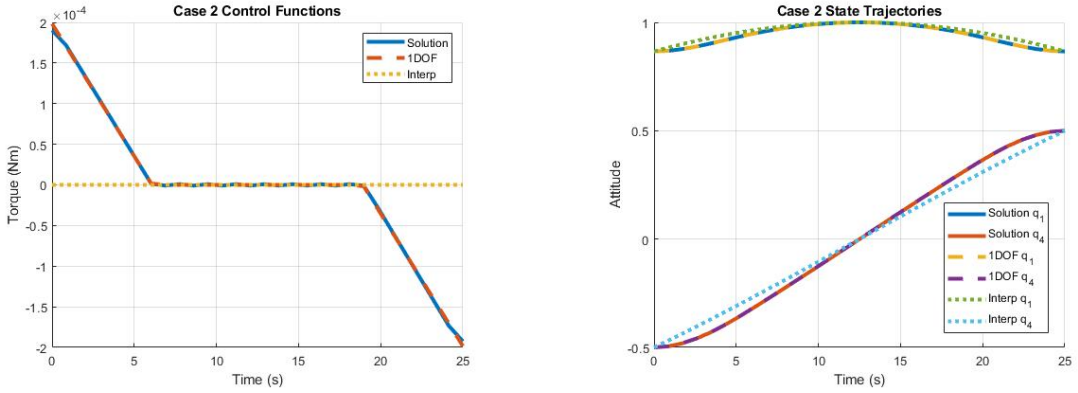


Figure 3.6: Comparison of State and Control Trajectories for Active Rate Constraint

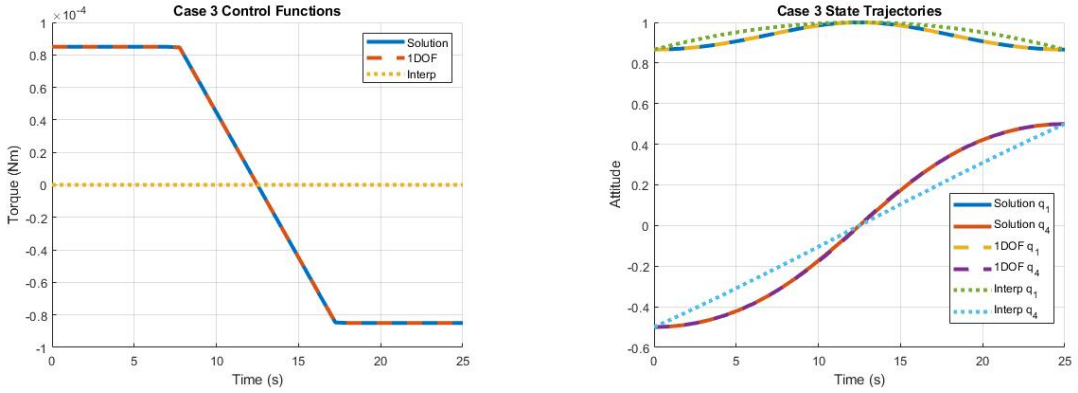


Figure 3.7: Comparison of State and Control Trajectories for Active Torque Constraint

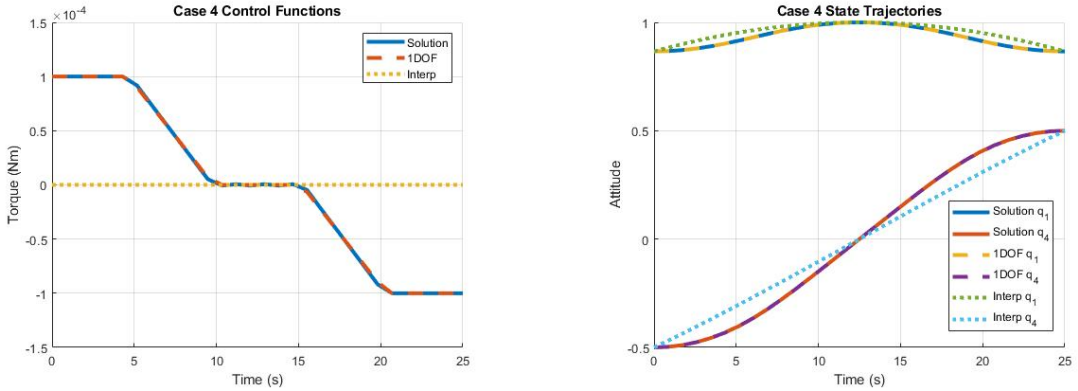


Figure 3.8: Comparison of State and Control Trajectories for Active Rate and Torque Constraints

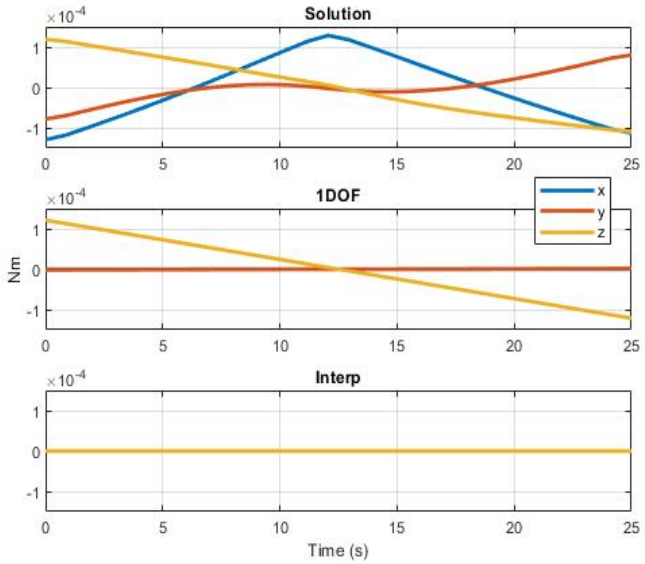


Figure 3.9: Comparison of Control Functions for Active Exclusion Constraint

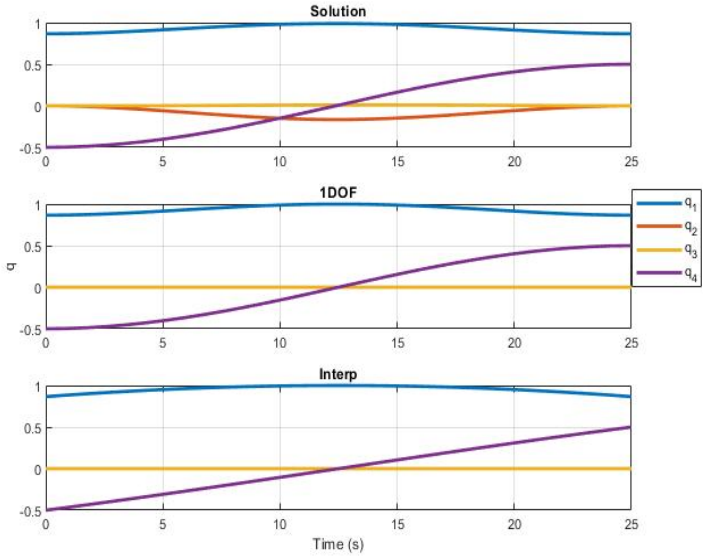


Figure 3.10: Comparison of Attitude Trajectories for Active Exclusion Constraint

Note that in plots 3.5-3.8, only the control in the z-axis and the first and fourth values of $\boldsymbol{q}$ are shown to avoid cluttering and since the other values are near zero. These plots show how closely the 1DOF solution approximates the actual solution in the first four cases, which explains why the 1DOF method performs better than the interpolation method in these cases. Figures 3.9 and 3.10 shows that when an exclusion constraint is active, the 1DOF method does not closely approximate the solution. Finding a better method for initialization in a case where an exclusion constraint is active remains an area for further investigation. Other than cases where an exclusion constraint is active, the 1DOF method consistently enables SOAR to converge in the minimum number of subproblem iterations and in a much shorter amount of time.

Another way to improve solve time would be to reduce the number of parameters that are being optimized. The next chapter shows how the required number of collocation points can be reduced by efficiently distributing the collocation points, which further reduces the solve time.

Chapter 4

REDISTRIBUTING COLLOCATION POINTS TO AVOID CONSTRAINT VIOLATION

SOAR utilizes the direct collocation method, which means that the constraints are only enforced at the collocation points [36]. This limitation is problematic for attitude exclusion constraints because it is possible that an exclusion constraint will be violated between collocation points, even if the constraint is met at every collocation point. This chapter introduces a new method for preventing constraint violations between the collocation points by increasing the density of collocation points near the constraint, without adding any new constraints or increasing the number of collocation points.

4.1 Introduction to Exclusion Constraint Violation

Figures 4.1 and 4.2 illustrate trajectories computed by SOAR which have exclusion cone constraint violations. Note that these figures show a 2-dimensional representation of the problem, where the blue line is the x and z component of the boresight vector's trajectory, the red circle encompasses the x and z components of the exclusion cone, and the asterisks represent the x and y components of the boresight vector's trajectory at the collocation points. A constraint violation happens if the blue line crosses over the red circle.

Experimentation has shown that constraint violation between collocation points occurs for almost any trajectory where the exclusion constraint is active, no matter the size of the exclusion constraint. There are a handful of ways to mitigate the problem. Adding more collocation points tends to reduce the constraint violation, but this is not always an option since more points increases the solve time. Another solution is to solve for the points in the state trajectory between the collocation points where the exclusion constraints are

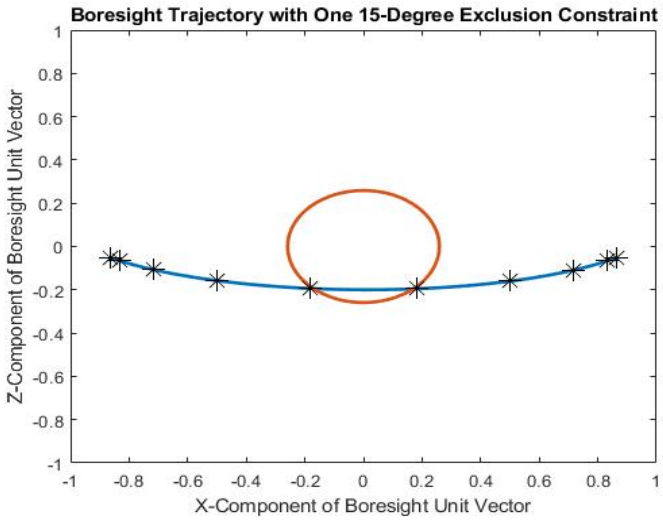


Figure 4.1: Constraint Violation Between Collocation Points For a 15 Degree Constraint

the closest to being violated, and to add an exclusion constraint there. These points are like the original collocation points, except that only the exclusion constraint is enforced at these new points, so it is more efficient computationally than just adding more collocation points [37]. A third solution to this problem is to simply increase the size of the exclusion constraint. However, this prevents the discrete solution determined by SOAR from closely approximating the optimal solution, and can also cause artificial infeasibility. All of these methods either increase the number of variables (and consequently, the solve time), or change the constraints. Since SOAR already requires an integration step to set up the discrete dynamics of the problem, the constraint violation can be detected and accounted for by changing the distribution of the collocation points to increase the density near the constraint. This new method captures the constraint violation avoidance benefit of having a large number of collocation points, without drastically increasing the solve time.

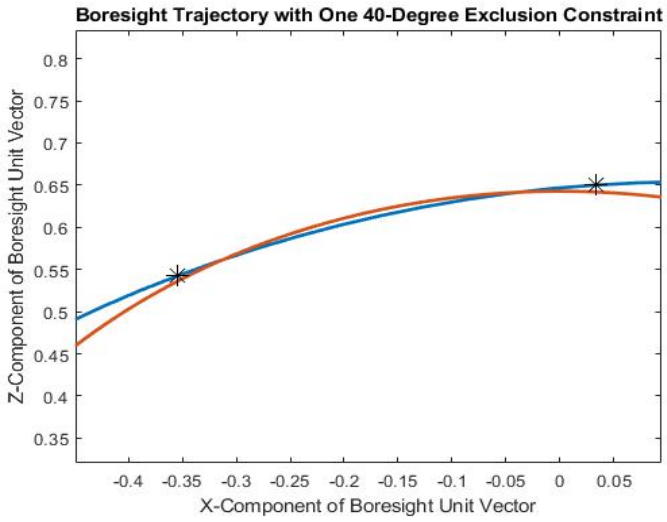


Figure 4.2: Constraint Violation Between Collocation Points For a 40 Degree Constraint

4.2 Determining Where the Constraint Violation is Likely

The first step in redistributing the collocation points, is to determine where (where in terms of the times $t \in (t_0, t_f)$), the constraint violation is occurring between the collocation points. Therefore, during SOAR's integration step, the constraint violation in degrees at each point in the trajectory is calculated using the equations below:

$$v_i(t) = \phi_i - cos^{-1}(\boldsymbol{q}\boldsymbol{v}_{body}\bar{\boldsymbol{q}} \cdot \boldsymbol{v}_{inertial}),\ i = 1, ..., K \tag{4.1}$$

$$V(t) = max(0, v_i) \tag{4.2}$$

Where ϕ is the angle in degrees of the exclusion constraint, $\bar{\boldsymbol{q}}$ is the attitude quaternion conjugate, and K is the number of exclusion constraints. The result is a new parameter, $V(t)$, which keeps track of maximum constraint violation at every point in the trajectory. Also note that a fast approximation for cos^{-1} should be used, since perfect accuracy is not necessary. Figure 4.3 shows an example of what $V(t)$ looks like for the same trajectory in

figure 4.1. This example uses 10 integration steps between 10 collocation points, for a total of 91 points.

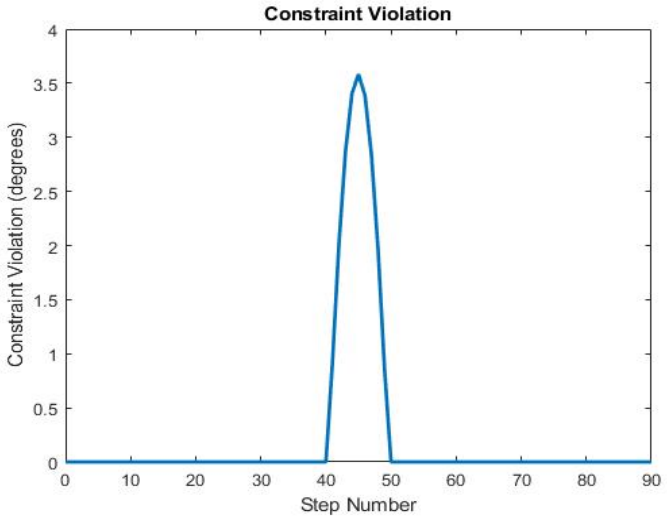


Figure 4.3: Constraint Violation V(t)

This is not a perfect estimate of where the constraint violation is likely to happen, since points close to collocation points will have a lower constraint violation. Therefore, the following equation is used to calculate the constraint violation divided by the amount of time to the nearest collocation point.

$$\bar{V}(t) = max\left(\frac{V(t)}{t_{sub}D_a(t)}, \frac{V(t)}{t_{sub}D_p(t)}\right) \tag{4.3}$$

Where $D_a(t)$ and $D_p(t)$ are the number of time steps to the previous and next collocation point, respectively, and t_{sub} is the length of time of each time step. Figure 4.4 shows a plot of $\bar{V}(t)$ from the trajectory in figure 4.1.

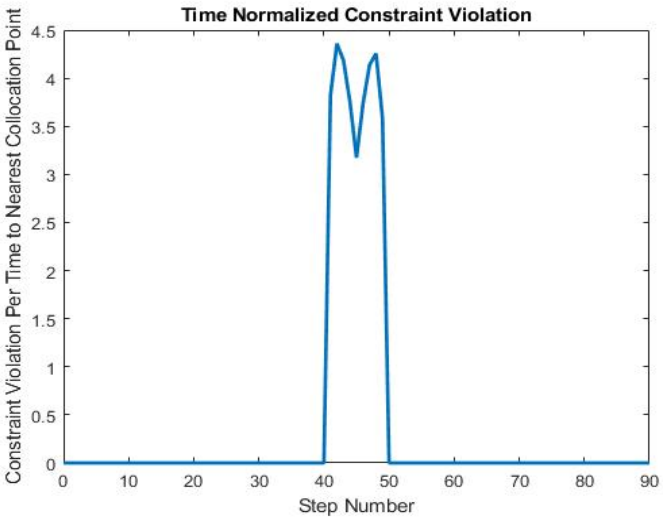


Figure 4.4: Constraint Violation Per Time To/From Nearest Collocation Point $\bar{V}(t)$

$\bar{V}(t)$ gives a rough estimate of where in the trajectory the constraint violation is happening, regardless of the proximity of a collocation point.

4.3 Redistribution of Collocation Points

The goal is to redistribute the collocation points to minimize the maximum constraint violation. It is assumed that the amount of constraint violation at any point is a decreasing function of the time to the nearest collocation point. In other words, moving a collation point towards a point will reduce the amount of constraint violation at that point. Now consider a distribution of collocation points τ^* such that amount of constraints violation between every collocation point is equal. Given the assumption that moving a nearby collocation point towards any point in the continuous trajectory will reduce the constraint violation at that point, any change in τ^* will cause the constraint violation to increase between at least one pair of collocation points. This would result in an increase of the maximum constraint violation. Therefore, the distribution τ^* is optimal.

To achieve this optimal distribution, the collocation points must be placed in such a way that the max constraint violation between each collocation point is equal. A first order

approximation of the constraint violation as a function of distance to the nearest collocation point can be used to estimate where the collocation points should be. Also, this first order approximation is given by the $\bar{V}(t)$ (figure 3.4). Therefore, integrating $\bar{V}(t)$ from t_1 to t_2 yields an estimate for the constraint violation between at t_2 if the nearest collocation point is at t_1. Therefore a first order estimate of τ^* is the distribution that evenly divides the integral of $\bar{V}(t)$. This is accomplished with the following equations:

$$\bar{V}_{int}(t) = \int_{t_0}^{t} \bar{V}(t)\, dt \tag{4.4}$$

$$\bar{V}_{total} = \bar{V}_{int}(t_f) \tag{4.5}$$

$$\bar{V}_{sub} = \bar{V}_{total}/(N-1) \tag{4.6}$$

$$\tau_0 = 0,\ \tau_N = s \tag{4.7}$$

$$\tau_i = \{t \mid \bar{V}_{int}(t) = i\bar{V}_{sub},\ i = 1, ..., N-1\} \tag{4.8}$$

This integration is accomplished numerically. Although these equation only approximate the optimal distribution of collocation points, since SOAR already relies on iterations, this distribution eventually converges to the optimal distribution.

There is one last refinement that must be made. This redistribution algorithm concentrates all the collocation points near the constraint. Ideally, some of the collocation points should be left evenly distributed so that the algorithm balances constraint violation avoidance, and evenly approximating the optimal control. In addition, experimentation has shown that there is a tendency for the collocation points to overshoot the optimal distribution. The following updates need to be made:

$$\sigma(t) = \bar{V}_{sub} \tag{4.9}$$

$$\bar{V}_{new}(t) = (1 - w_c)\bar{V}(t) + w_c\sigma(t) \tag{4.10}$$

Where w_c is a parameter between 0 and 1 which dampens the redistribution of the collocation points. In practice, a weight of 0.75 is used, but this value can be increased to slow down the redistribution, or decreased to speed up the redistribution. Equations 4.4-4.8 are recalculated with this new $\bar{V}_{new}$. The effect of adding σ, which is just a uniform distribution, is that during the integration, the distance from a collocation point now adds to the integration, encouraging the points to be evenly distributed. Figures 4.5 and 4.6 illustrate these new variables for the trajectory in 4.1.

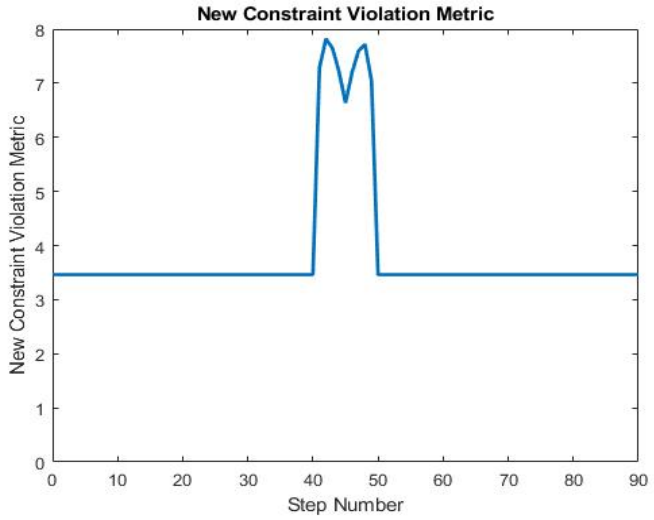


Figure 4.5: New Constraint Violation Metric V_{new}

Note that the asterisks represent the points which evenly divide V_{int}.

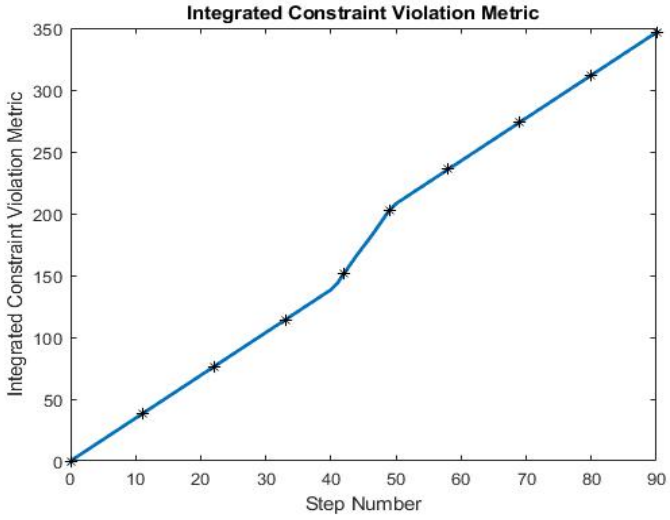


Figure 4.6: New Integrated Constraint Violation Metric V_{int}

Figure 4.7 compares the trajectory from figure 4.1 with a new trajectory which uses the new collocation redistribution algorithm, uses eight collocation points instead of ten, and results in zero constraint violation.

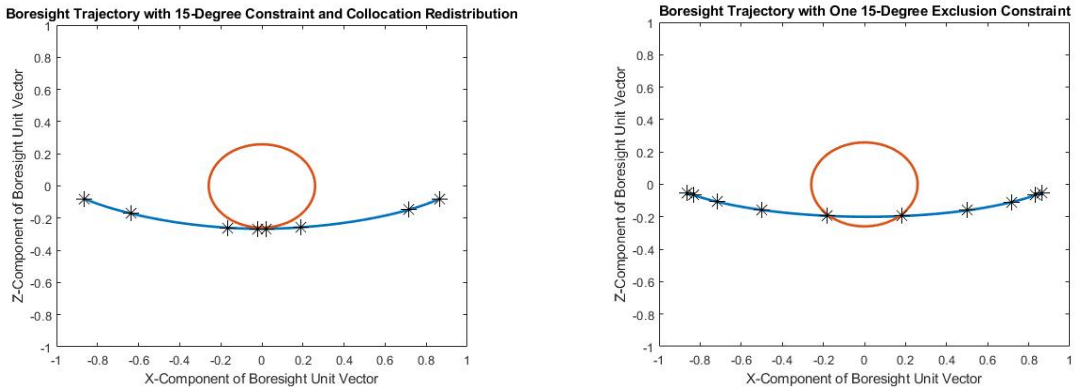
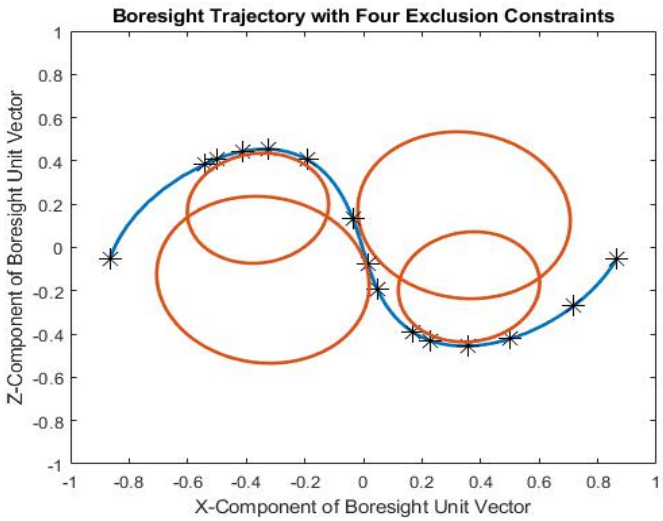


Figure 4.7: Comparison of Solution with and without Collocation Redistribution

In addition, the algorithm also works with multiple pointing constraints:

Figure 4.8: Trajectory of Boresight with Four Exclusion Constraints



While this algorithm does usually result in SOAR using more subproblem iterations and adds an extra step for each iteration, it also enables SOAR to use fewer collocation points, which usually saves time overall and prevents any constraint violation. A secondary benefit of this algorithm is that it increases the density of collocation points near constraints, meaning it also enables the piecewise control function to better approximate the continuous optimal control function near the exclusion constraints. In addition to just the CAC problem, this algorithm could be improve efficiency for many other trajectory generation algorithms which use SCP. The benefits of this algorithm will be further demonstrated in the next chapter.

Chapter 5
SIMULATION TESTING

The intent of this chapter is to demonstrate that SOAR works well for a wide variety of initial conditions, even when actuator dynamics and sensor uncertainties are considered. Four Monte Carlo simulations with a total of 1400 attitude reorientations are used to do this. The results show that SOAR always calculates a feasible trajectory, typically in less than a second and efficiently avoids constraint violation.

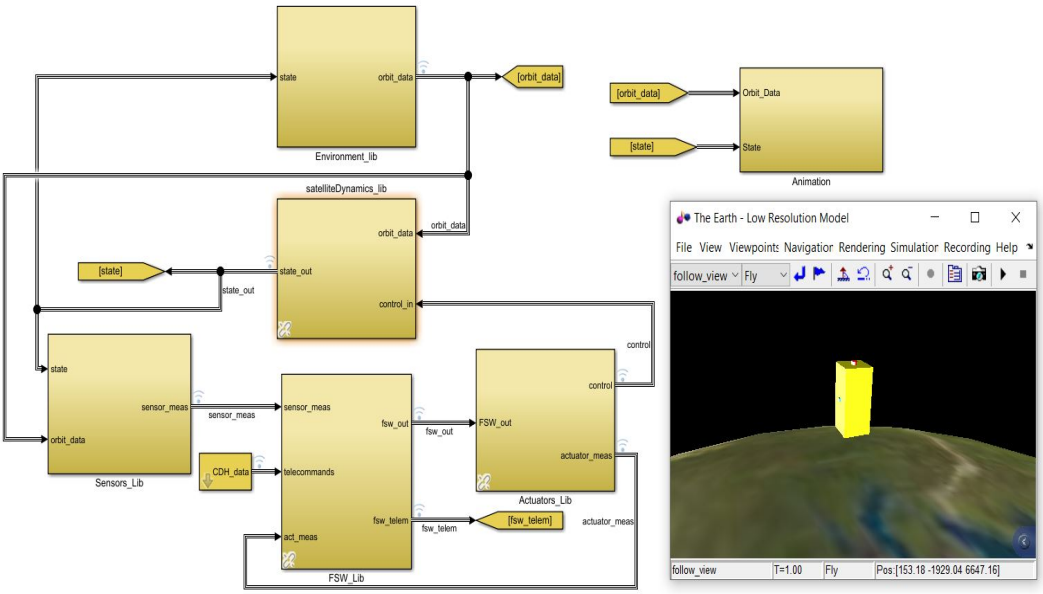
5.1 The Simulation

These Monte Carlo tests were performed using a Simulink-based cubesat simulator developed by the UW AACT, and can be downloaded with instructions from Github [1]. The simulator was developed for SOC-i, and as a result includes models specific to a 2U cubesat with reaction wheels, magnetorquers, a sun sensor, a magnetometer, a gyroscope, and photodiodes. Obviously, not all satellites will use this same configuration. However, there are too many possible configurations for a satellite for the scope of this thesis, so only this one configuration is tested, with the expectation that if SOAR works for a low complexity cubesat, then it will probably work for most other satellites. This section provides a brief description of how the simulator works to accurately model a real cubesat in low earth orbit. More information about how the simulator works can be found in [38] and [1].

5.1.1 Satellite Dynamics Library

The satellite dynamics library uses numerical integration to determine the cubesat's state given the cubesat's initial state, control torques, disturbance torques, and forces. The cubesat's state is:

Figure 5.1: UW AACT's Cubesat Simulator



$$\boldsymbol{x} = [\boldsymbol{r}_{eci}\, \boldsymbol{q}_{eci}\, \boldsymbol{h}_b\, \boldsymbol{h}_{rwa}]^\top \tag{5.1}$$

Where $\boldsymbol{r}_{eci}$ is the cubesat's position in kilometers in the earth-centered-inertial (ECI) frame, $\boldsymbol{q}_{eci}$ is the quaternion which rotates the ECI frame to the cubesat's body frame, $\boldsymbol{h}_b$ is the cubesat's angular velocity momentum in NMs, and $\boldsymbol{h}_{rwa}$ is the angular momentum stored in the cubesat's reaction wheels in Nms. Given an initial state, the state can be determined at every time step by calculating the rates of change of the state and integrating. The rates are calculated with these equations [39]:

$$\ddot{\boldsymbol{r}}_{eci} = \boldsymbol{a}_{total} = \boldsymbol{a}_g + \boldsymbol{a}_{srp} + \boldsymbol{a}_{drag} \tag{5.2}$$

$$\dot{\boldsymbol{h}}_b = \boldsymbol{T}_{ext} - \boldsymbol{T}_{rwa} - \boldsymbol{J}^{-1}\boldsymbol{h}_b \times (\boldsymbol{h}_b + \boldsymbol{h}_{rwa}) \tag{5.3}$$

$$\boldsymbol{T}_{ext} = \boldsymbol{\mu}_b \times \boldsymbol{B}_b + \boldsymbol{T}_{dist} \tag{5.4}$$

$$\dot{\boldsymbol{q}}_{eci} = \frac{1}{2}\boldsymbol{q}_{eci} \otimes \boldsymbol{J}^{-1}\boldsymbol{h}_b \tag{5.5}$$

$$\dot{\boldsymbol{h}}_{rwa} = \boldsymbol{T}_{rwa} \tag{5.6}$$

Where $\boldsymbol{a}_g$ is the acceleration due to gravity and is calculated using spherical harmonic gravity model (such as EGM2008), $\boldsymbol{a}_{srp}$ is the acceleration due to solar radiation pressure, $\boldsymbol{a}_{drag}$ is the acceleration due to drag, J is the cubesat's inertia matrix, $\boldsymbol{T}_{ext}$ is the external torque, $\boldsymbol{\mu}_b$ is the magnetic dipole created by the cubesat's magnetorquers, $\boldsymbol{B}_b$ is the magnetic field vector in the body frame, $\boldsymbol{T}_{dist}$ is the total disturbance torque (from solar radiation pressure, drag, and an estimated residual dipole), and $\boldsymbol{T}_{rwa}$ is the torque acting on the reaction wheels.

5.1.2 Environment Library

The environment library calculates all relevant environment information given the cubesat's state, including $\boldsymbol{a_g}$, $\boldsymbol{a}_{srp}$, $\boldsymbol{a}_{drag}$, and $\boldsymbol{T}_{dist}$. It also calculates an inertial sun vector, an inertial magnetic field vector, and whether or not the cubesat is in eclipse. The input to this library is the simulation time, and the state vector (5.1). The sun vector is determined using Algorithm 29 in Vallado's Fundamentals of Astronautics and Applications [40], and the magnetic field vector is determined using the World Magnetic Model (WMM) [41].

5.1.3 Sensor Library

The sensor library takes the environment information and the state of the cubesat and outputs the sensor measurements for the sun sensor, gyroscopes, magnetometers, and photodiodes. Generally speaking, the ideal sensor output is calculated (two angles for the sun sensor, and a three dimensional vector for the magnetometer and gyroscope, and a five dimensional vector for the photodiodes), then Gaussian noise is added to accurately simulate what the actual sensors will output. A bias is also added for both the gyroscopes and magnetometer. The bias and variance of the Gaussian noise are picked based on actual measurements from

the sensors which will fly on SOC-i. Lastly, the sensor outputs are rotated if the sensor frames are not aligned with cubesat's body frame.

5.1.4 Actuator Library

The actuator library takes in actuator commands (reaction wheel rpm's and magnetorquer dipoles), and calculates the torques and dipoles that result from the actuators. Just like in the sensor library, Gaussian noise is added to the commanded rpms and dipoles to simulate the actual noisy behavior of the actuators. Next, a state space model is used to simulate the relationship between commands and outputs. The magnetorquers respond almost instantly, so the output is assumed to be equal to the input. However, the reaction wheels do have a consequential amount of delay. As a result, a model for the dynamics of the reaction wheels is needed. Using input and output data gathered from the reaction wheels, Matlab's System Identification Toolbox was used to create a 3rd order state space model for the reaction wheels. The torque produced by the reaction wheels can then be modeled with the equations below:

$$\boldsymbol{x}_{n+1} = \boldsymbol{A}\boldsymbol{x}_n + \boldsymbol{B}\boldsymbol{u} \tag{5.7}$$

$$\boldsymbol{y}_n = \boldsymbol{C}\boldsymbol{x}_n \tag{5.8}$$

$$\boldsymbol{Y} = \frac{2\pi}{60}[\boldsymbol{y}_1\ \boldsymbol{y}_2\ \boldsymbol{y}_3\ \boldsymbol{y}_4]^\top \tag{5.9}$$

$$\boldsymbol{T}_{rwa} = \boldsymbol{A}_{w2b} J_w (\frac{d}{dt}\boldsymbol{Y}) \tag{5.10}$$

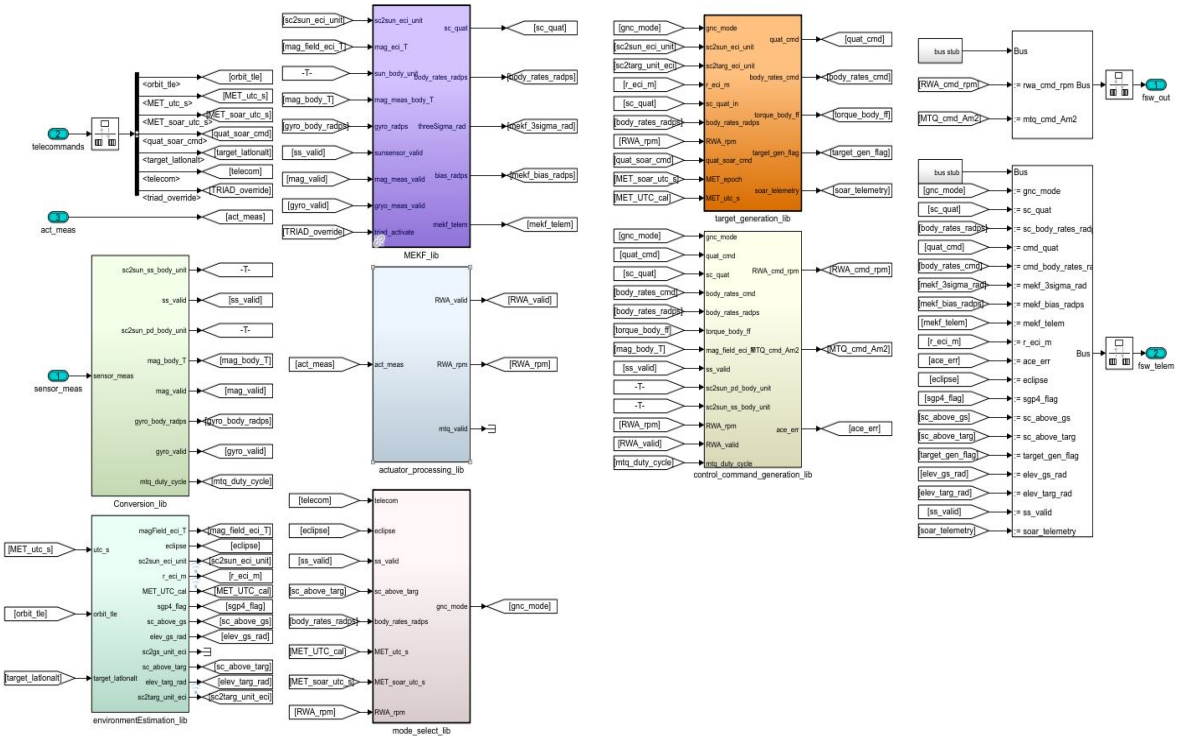
Where $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ are the matrices determined by Matlab's System Identification Toolbox, $\boldsymbol{u}$ is the commanded rpm, $\boldsymbol{y}$ is the actual rpm, $\boldsymbol{Y}$ is the vector which stores the rotation rates of all the reaction wheels in radians per second, J_w is the inertia of a reaction wheel, and $\boldsymbol{A}_{w2b}$ is the matrix which maps the torque from each reaction wheel to the net torque in the body frame.

The actuator library also outputs actuator measurements, which only include the measured RPM's since the magnetorquer dipoles are not measured. Just like in the sensor library, Gaussian noise is added to better approximate the actual reaction wheel rotation rate sensor.

5.1.5 Flight Software Library

The only remaining library (other than the optional animation library) is the flight software library (figure 5.2), which calculates the appropriate actuator commands given the sensor measurements, actuator measurements, and commands from the command and data handling system if there are any. The following libraries are used to achieve this:

Figure 5.2: Flight Software Model



Sensor Processing Library

This library rotates the sensor measurements back into the body frame, undoes the bias that is known to exist in some of the sensors, and averages together values from redundant sensors. The result is body frame vectors of the sun, magnetic field, and a rotation rate.

Environment Estimation Library

This library is similar in a lot of ways to the environment library (Section 5.1.2). It calculates the satellite's position using an SGP4 propogator [42], and calculates the inertial sun and magnetic field vectors using Vallado's Algorithm 29, and the World Magnetic Model [41].

Multiplicative Extended Kalman Filter (MEKF) Library

This library compares the body frame vectors from the sensor processing library with the inertial vectors from the environment estimation library to determine the cubesat's attitude quaternion, $\boldsymbol{q}_{eci}$ [6, 7, 43]. In addition, bias in the gyroscopes is accounted for. Figure 5.3 illustrates how the MEKF works:

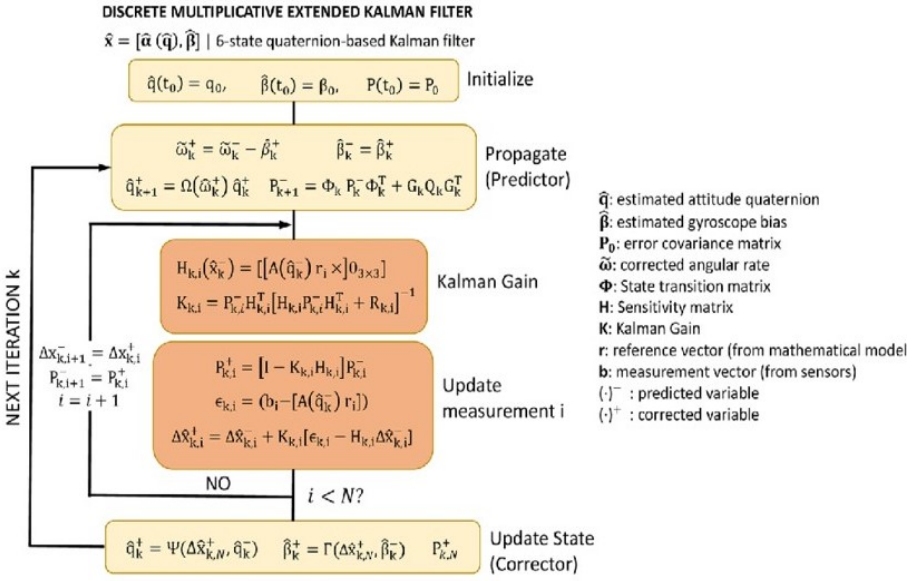


Figure 5.3: Multiplicative Extended Kalman Filter [6, 7]

This library outputs the spacecrafts estimated attitude, rotation rate, and an estimated state covariance matrix $\boldsymbol{P}$, which will be used later to estimate the satellite's attitude estimation error.

Actuator Processing

This library is similar to the sensor processing library, except it is used for signals from the actuators (in this case, just the reaction wheels).

Mode Select Library

This library determines the GNC mode. For this test, the only GNC modes were Attitude Stabilization Mode (ASM) and SOAR mode. In ASM, the cubesat holds a specific attitude with zero angular rate. In SOAR mode, the cubesat reorients using the SOAR algorithm.

Guidance Library

The guidance library contains SOAR, and outputs a commanded attitude and rotation rate. If the GNC mode is Attitude Hold, the commanded attitude is constant, and the commanded rate is zero. In SOAR mode, the commanded attitude and rate is determined by SOAR.

Control Library

This library compares the estimated attitude and rate with the commanded attitude and rate, and calculates control torques using a PD quaternion regulator [33]. In other words, this is where the closed loop feedback control torques are added. In addition, a commanded dipole is determined with a proportional controller to so that the magnetorquers can be used to reduce the total angular momentum of the system: $(\boldsymbol{h}_b + \boldsymbol{h}_w)$. The output to this block is the actuator commands.

The actuator commands from the flight software become the inputs to the actuator library, and the loop is closed. The result is a software based test bed for SOAR.

5.2 Monte Carlo Simulation Setup

In order to confirm that the SOAR algorithm can accurately reorient a satellite without violating constraints, the cubesat simulator was used to simulate 1400 SOAR maneuvers for a variety of exclusion constraints. The initial conditions were picked to simulate a 2U cubesat in low earth orbit. In addition, constraints and the initial attitude and initial wheel momentum were randomly generated to cover almost every possible reorientation and every possible set of constraints. The commanded attitude was always $[1\ 0\ 0\ 0]^\top$ to avoid the unwinding phenomenon [44], which SOAR is susceptible to (because of this, SOAR should always use a commanded an attitude of $[1\ 0\ 0\ 0]^\top$, which can be done by rotating all vectors and matrices into a new coordinate frame such that the commanded attitude is $[1\ 0\ 0\ 0]^\top$, computing a trajectory with SOAR, and then converting back to the original frame). Initializations which happened to be infeasible (for example, if the commanded

attitude caused one of the pointing constraints to be violated) were discarded.

The total simulation time for each maneuver was 120 seconds: the first 90 seconds allowed the system (particularly the MEKF) to reach a steady state before a 30 second SOAR maneuver. At 90 seconds, the state determined by the MEKF library was used as the initial state for SOAR. In addition, a constraint buffer was calculated and added to the exclusion angles to ensure that the satellite does not violate the constraint despite the uncertainty in state estimation and control, using the method described in the next section.

5.3 Constraint Buffer Calculation

The constraint buffer is simply equal to the upper bound on the satellite's estimated pointing error There are many sources of uncertainty which can increase pointing error. These include sensor mounting, sensor bias and noise, environment model inaccuracy, navigation errors, actuator delays, etc. These all fall into three categories: estimation errors, navigation errors, and control errors. The total error is:

$$e_{total} = e_{estimation} + e_{navigation} + e_{control} \tag{5.11}$$

To use SOAR as efficiently as possible, it is important to calculate the total pointing error as accurately as possible. Assuming the MEKF filter's state and measurement matrices are calculated appropriately based on the actual sensor uncertainties, the 3-σ estimation error can be calculated using the covariance matrix from the kalman filter:

$$e_{roll} = 3|\boldsymbol{P}_{1,1}|^{1/2} \tag{5.12}$$

$$e_{pitch} = 3|\boldsymbol{P}_{2,2}|^{1/2} \tag{5.13}$$

$$e_{yaw} = 3|\boldsymbol{P}_{3,3}|^{1/2} \tag{5.14}$$

$$e_{estimation} = e_{est,0} + (e_{roll}^2 + e_{pitch}^2 + e_{yaw}^2)^{1/2} \tag{5.15}$$

Where equations 5.12 - 5.14 define the error in the roll pitch and yaw axes, and $e_{est,0}$ is the base estimation error, due to errors such as sensor mounting inaccuracies and environment

model inaccuracies. In a real-world implementation, this will be nonzero. In this simulation, $e_{est,0}$ is zero.

The navigation error is the constraint violation which results from SOAR commanding a state which violates a constraint. Due to the collocation reallocation algorithm in Chapter 4, this error should be nearly zero. A conservative value of 0.1 degrees was used.

The control error can be estimated by finding the difference between the commanded and the actual RPMs of the reaction wheels using the reaction wheel state space model (5.7-10) for the largest expected maneuver in terms of commanded RPMs. In addition, the delay in commanding the reaction wheels due to the GNC subsystem's step size also contributes to the control error. Overall, the easiest way to estimate the control error is to just simulate a few maneuvers, and find an upper bound on the control errors. In particular, only the control error which might contribute to a constraint violation (i.e. control errors which are not in the direction of the trajectory) should be considered for calculating the size of the constraint buffer. A conservative value of 1 degree was used.

Therefore, the total size of the constraint buffer for these Monte Carlo simulations is:

$$\theta_{buffer} = 1.1 + (e_{roll}^2 + e_{pitch}^2 + e_{yaw}^2)^{1/2} \tag{5.16}$$

5.4 Test 1: One Hard Exclusion Constraint

The first Monte Carlo test simulated 500 SOAR maneuvers each with one randomly generated exclusion constraint and 8 collocation points. The size of each exclusion constraint had a uniform distribution between 10 and 60 degrees. The table below summarizes SOAR's performance for this test:

Table 5.1: Monte Carlo Test 1 Performance Summary				
Value	Constraint Violation (deg)	Pointing Error (deg)	Subproblem Iterations	Solve Time (s)
Max	3.1426	2.4551	9	1.0141
Mean	52.0928	0.9355	1.51	0.3011

Note that only negative values for constraint violation indicate that a constraint violation occurred. Figures 5.4-5.7 show the results of these maneuvers.

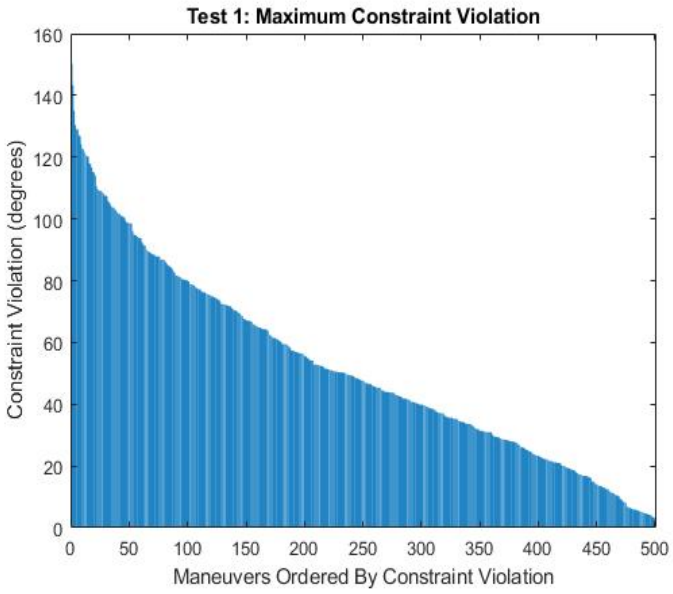


Figure 5.4: One Exclusion Constraint - Overall Constraint Violation

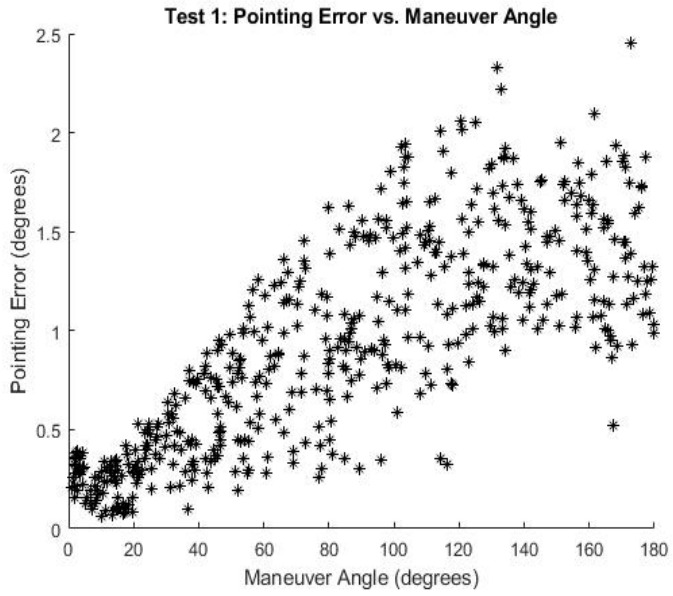


Figure 5.5: One Exclusion Constrain - Pointing Error

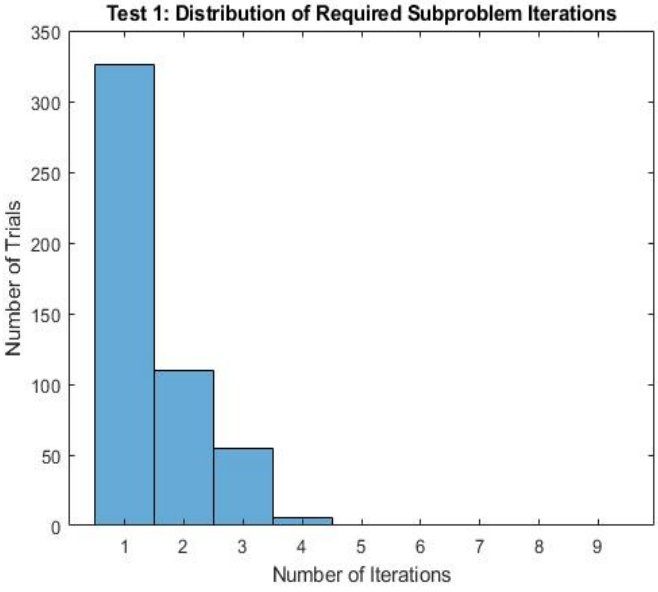


Figure 5.6: One Exclusion Constraint - Subproblem Iterations

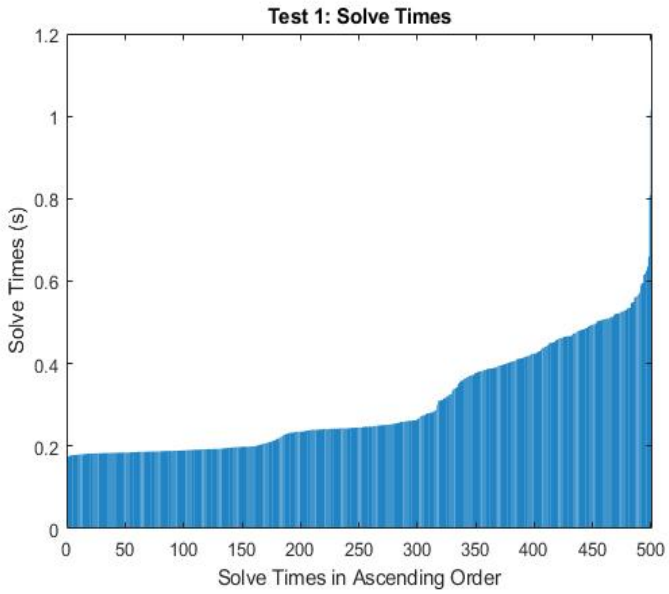


Figure 5.7: One Exclusion Constraint - Solve Time

5.5 Test 2: Three Hard Exclusion Constraints

The second Monte Carlo test simulated 300 SOAR maneuvers with three exclusion constraints (same size as previous test) each and 12 collocation points. The table below shows SOAR's overall performance for this test.

Table 5.2: Monte Carlo Test 2 Performance Summary

Value	Constraint Violation (deg)	Pointing Error (deg)	Subproblem Iterations	Solve Time (s)
Max	2.6060	2.2209	11	2.3477
Mean	26.9929	0.8372	1.4467	0.5186

Figures 5.8-5.11 show the results of these maneuvers.

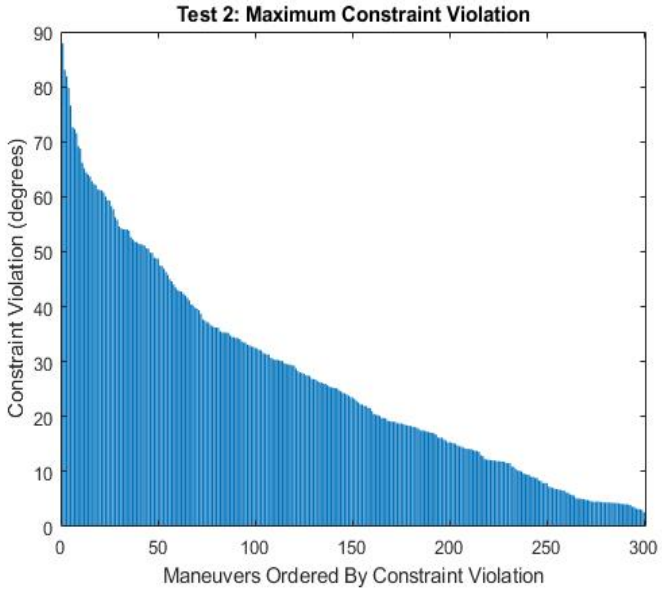


Figure 5.8: Three Exclusion Constraints - Overall Constraint Violation

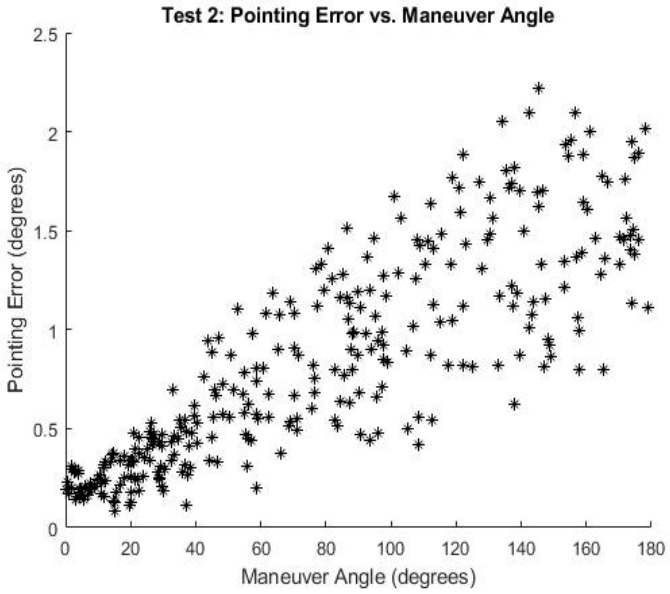


Figure 5.9: Three Exclusion Constraints - Pointing Error

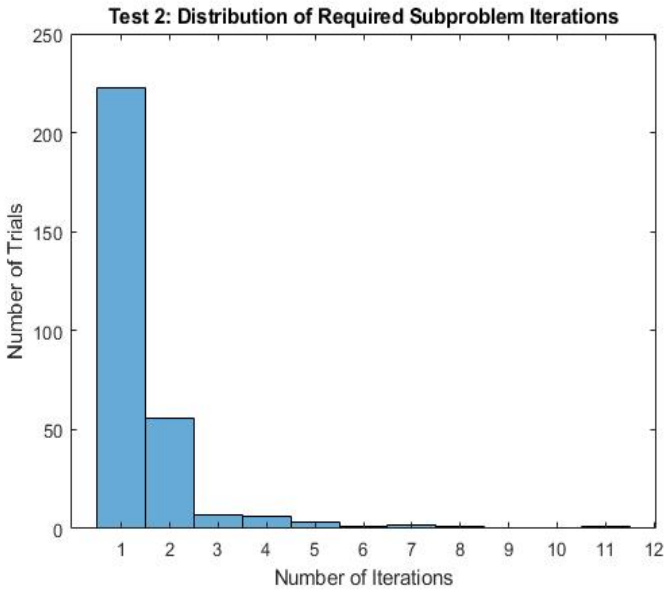


Figure 5.10: Three Exclusion Constraints - Subproblem Iterations

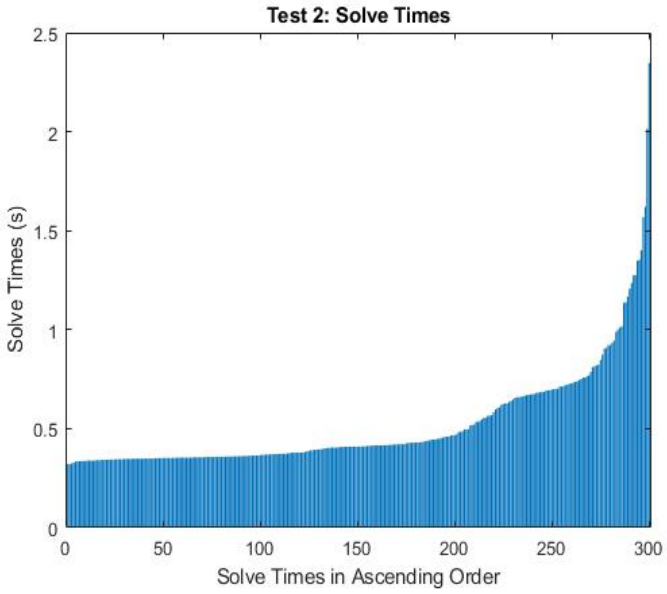


Figure 5.11: Three Exclusion Constraints - Solve Time

5.6 Test 3: One Exclusion Constraint, One Inclusion Constraint

The second Monte Carlo test simulated 300 SOAR maneuvers with one exclusion constraint and one inclusion constraint using ten collocation points. The size of the inclusion constraint was fixed at 180 degrees to avoid overly constraining the set of feasibly initial conditions. The table below shows SOAR's overall performance for this test.

Table 5.3: Monte Carlo Test 3 Performance Summary

Value	Constraint Violation (deg)	Pointing Error (deg)	Subproblem Iterations	Solve Time (s)
Max	3.0903	2.2778	8	0.6772
Mean	53.6580	0.7914	1.2500	0.1712

Figures 5.12-5.15 show the results of these maneuvers.

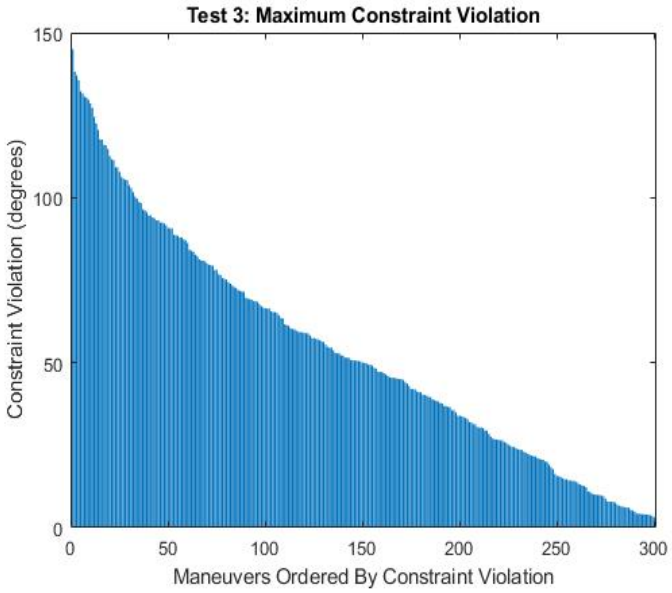


Figure 5.12: Mixed Hard Constraints - Overall Constraint Violation

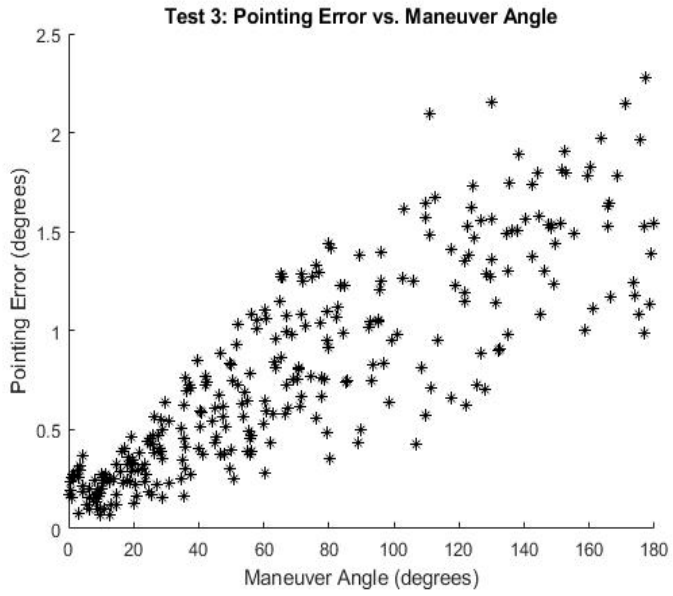


Figure 5.13: Mixed Hard Constraints - Pointing Error

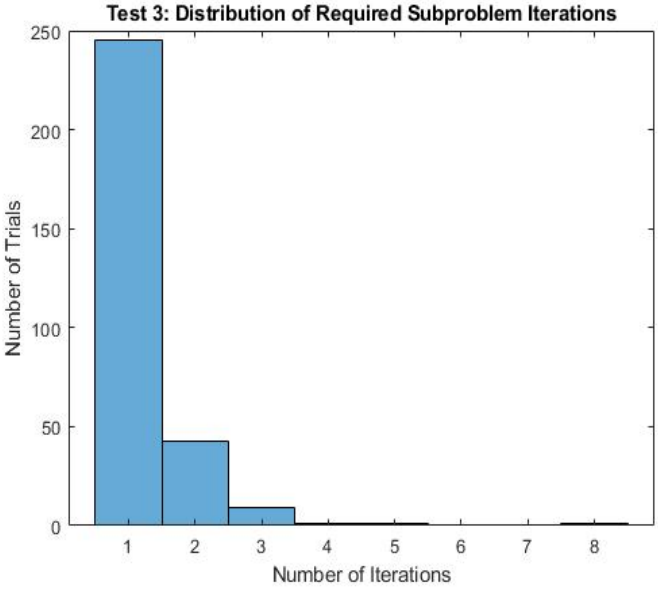


Figure 5.14: Mixed Hard Constraints - Subproblem Iterations

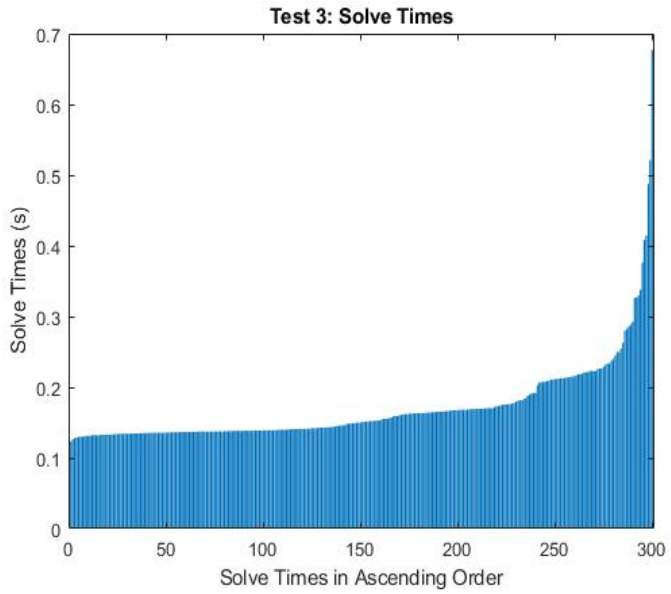


Figure 5.15: Mixed Hard Constraints - Solve Time

5.7 Test 4: One Exclusion Constraint, One Inclusion Constraint, One Soft Constraint

The fourth Monte Carlo test simulated 300 SOAR maneuvers with one exclusion constraint, one inclusion constraint, and one soft constraint using 10 collocation points. This test used the exact same initial conditions as the previous test, with the only difference being the inclusion of a soft constraint. In addition, the soft constraint was fixed at 180 degrees, both to increase the chance that it would be an active constraint and so that the effect of the soft constraint could be measured. The table below shows SOAR's overall performance for this test.

Table 5.4: Monte Carlo Test 4 Performance Summary

Value	Constraint Violation (deg)	Pointing Error (deg)	Subproblem Iterations	Solve Time (s)
Max	2.6850	7.1527	18	2.1188
Mean	51.1350	1.1666	2.6400	0.2877

Figures 5.16-5.20 show the results of these maneuvers.

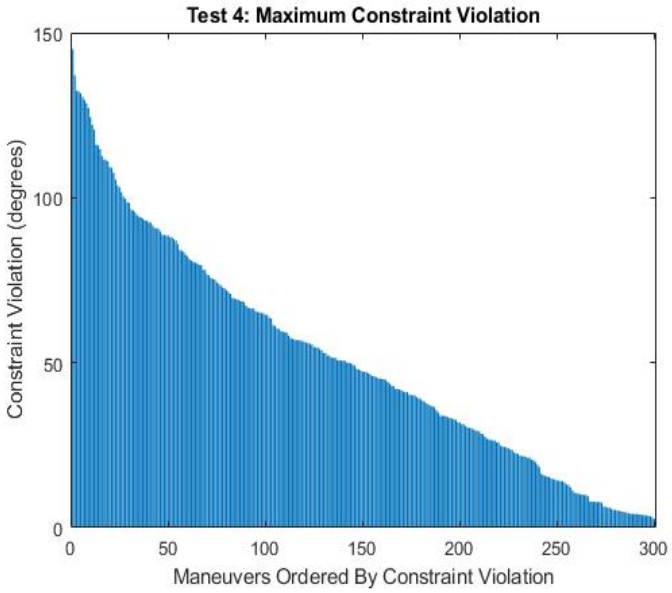


Figure 5.16: Mixed Hard and Soft Constraints - Overall Constraint Violation

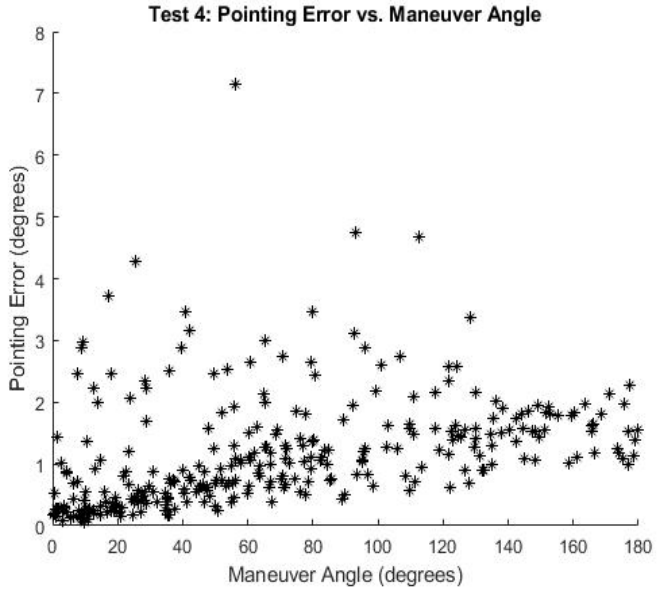


Figure 5.17: Mixed Hard and Soft Constraints - Pointing Error

This test resulted in a higher than normal amount of pointing error (figure 5.17). This is likely due to a rare case where the initial and final attitude result in the soft body and inertial vectors lining up. In this case, the satellite must reorient very quickly to avoid a constraint violation, and a quick reorientation exacerbates the control pointing error.

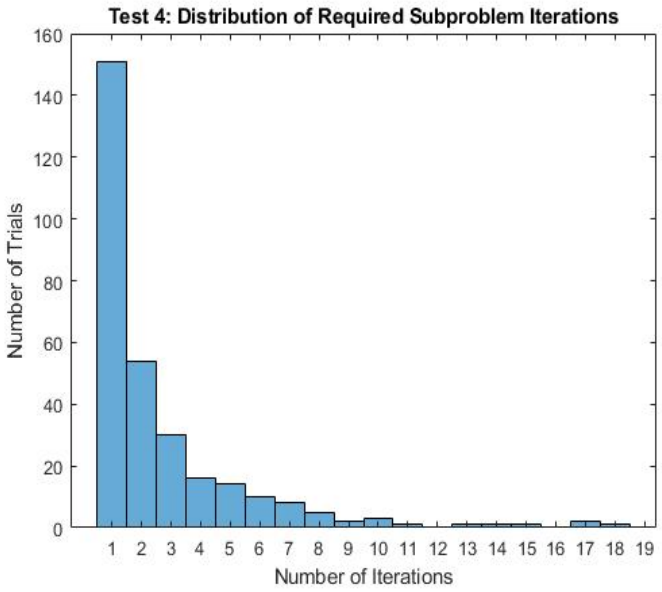


Figure 5.18: Mixed Hard and Soft Constraints - Subproblem Iterations

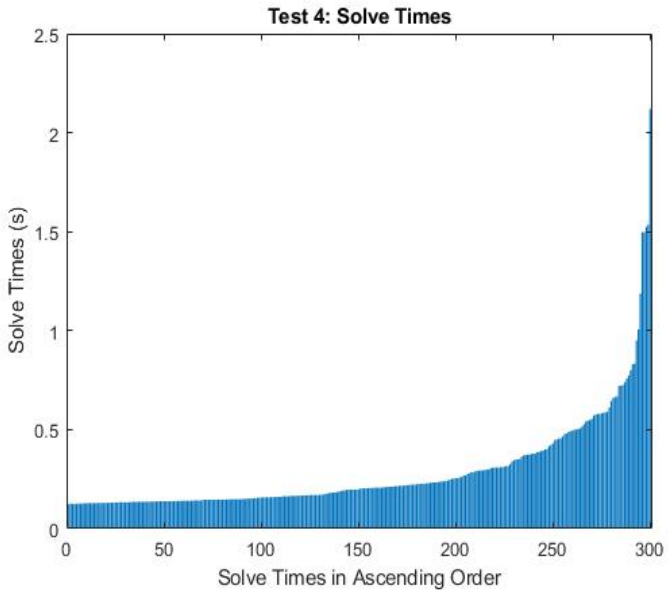


Figure 5.19: Mixed Hard and Soft Constraints - Solve Time

In addition to measuring constraint violation, pointing error, subproblem iterations, and solve times, the following integral was also recorded for the previous two Monte Carlo tests so that the effect of the soft constraint could be measured:

$$\int_{t_0}^{t_f} max(\boldsymbol{v}_b^\top \boldsymbol{v}_i, 0)dt \tag{5.17}$$

This integral was intentionally chosen because it is analogous to the amount of solar energy hitting the side of the satellite during the maneuver, if $\boldsymbol{v}_b$ is normal to the side of the satellite and $\boldsymbol{v}_i$ is the sun vector. Figure 5.20 compares this integral for tests three and four.

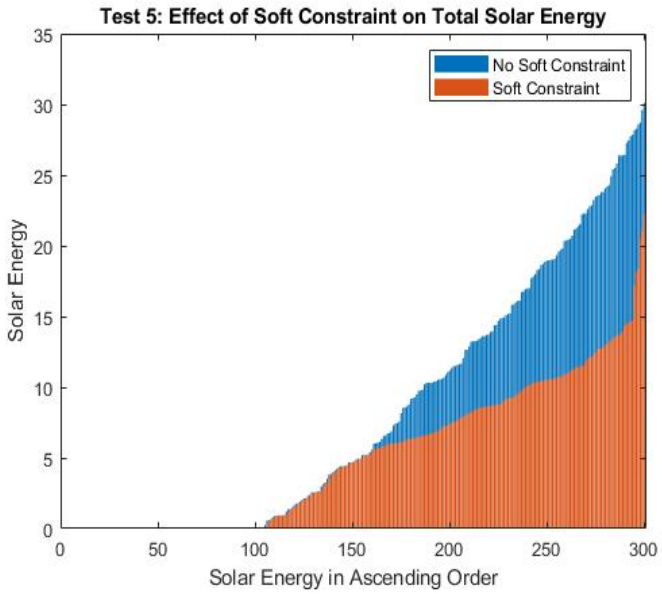


Figure 5.20: Mixed Constraints - Solar Energy Comparison

Since the same initial conditions and constraints with the exception of the soft constraint were used for tests four and five, this figure shows that the soft constraint is effective at enforcing some integral constraints. It is likely that SOAR could be used to manage the temperature of a satellite in addition to attitude control, by adjusting the bound on the soft integral constrain until the continuous integral constraint, such as 5.17 is below a certain requirement. However this connection is beyond the scope of this thesis and is an area of future work.

To summarize the results of these simulations, SOAR was able to control a satellite 1,400 times without ever violating a hard constraint. In addition, it appears that the pointing error was biased away from constraints, and was actually always lower than the largest constraint buffer which was 1.8 degrees, so the constraint buffer was actually not needed. The pointing error at the end of the maneuver was typically around one degree, and never above 2.5 degrees for hard constraints, and rarely above 5 degrees for soft constraints except for an outlier at 7 degrees. The maximum amount of solve time was 2.1188 seconds and the average

solve time was below one second. All these results suggest that SOAR, or some other convex optimization based attitude control system, have the potential to enable a higher level of autonomy in satellites.

Chapter 6
CONCLUSION

The primary conclusion of this thesis is that SOAR, a convex optimization based satellite guidance and control algorithm, can quickly calculate trajectories which safely and reliable reorient a satellite even with a wide variety of constraints. The use of the maximum principle for initialization as well as a collocation point redistribution algorithm all enable SOAR to find trajectories even faster without risk of a constraint violation. In addition, a suite of Monte Carlo simulations showed that SOAR works consistently even with pointing inaccuracies. As a result, a convex optimization approach to spacecraft attitude control is closer to reaching the high standards of space flight software.

6.1 Future Work

Furthermore, the full benefits of convex optimization based attitude control have yet to be fully explored. It very likely that SOAR could be used to enhance a satellite's thermal regulation system through the use of soft constraints, or to control a fleet of satellites all with state dependent pointing constraints. In addition, performance improvements are still possible. A better method for initialization for cases where exclusion constraints are active would greatly reduce SOAR's solve time. Also, only a single cost function and dynamics equation were used in this thesis. Alternative cost functions, such as ones that minimize reorientation time, or alternative dynamics equations for satellites with actuators other than reaction wheels are worth exploring.

BIBLIOGRAPHY

[1] AACT. University of Washington Aeronautics and Astronautics CubeSat Team Website: https://aact.space, 2022.

[2] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge university press, 2004.

[3] Matthew Kelly. An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation. *SIAM Review*, 59:849–904, 01 2017.

[4] Danylo Malyuta, Taylor P. Reynolds, Michael Szmuk, Thomas Lew, Riccardo Bonalli, Marco Pavone, and Behcet Acikmese. Convex Optimization for Trajectory Generation, 2021.

[5] Taylor P. Reynolds. SOAR Interface Control Document. *AACT Google Drive*, 2019.

[6] Alexandre Cortiella, David Vidal, Jaume Jane, Enric Juan, Roger Olivé, Adrià Amézaga, Joan Munoz-Martin, Hugo Carreno-Luengo, and Adriano Camps. 3Cat-2: Attitude Determination and Control System for a GNSS-R Earth Observation 6U CubeSat Mission. *European Journal of Remote Sensing*, 49:759–776, 02 2017.

[7] J.L. Junkins J. L. Crassidis. *Optimal Estimations of Dynamic Systems*. Chapman and Hall/CRC Applied Mathematics and Nonlinear Science Series, 2012.

[8] Yue Yu. Efficient Algorithms for Convex Optimization based Control. 2021.

[9] D. Dueri. Real-time Optimization in Aerospace Systems. Ph.D. Dissertation, University of Washington. 2018.

[10] Yuanqi Mao, Michael Szmuk, and Behçet Açıkmeşe. Successive Convexification: A Superlinearly Convergent Algorithm for Non-convex Optimal Control Problems. 04 2018.

[11] Riccardo Bonalli, Abhishek Cauligi, Andrew Bylard, and Marco Pavone. GuSTO: Guaranteed Sequential Trajectory Optimization via Sequential Convex Programming. 02 2019.

[12] Panagiotis Tsiotras and Mehran Mesbahi. Toward an Algorithmic Control Theory. *Journal of Guidance, Control, and Dynamics*, 40:1–3, 02 2017.

[13] Ping Lu. Introducing Computational Guidance and Control. *Journal of Guidance, Control, and Dynamics*, 40(2):193–193, 2017.

[14] Taylor P. Reynolds. Computational Guidance and Control for Aerospace Systems. Ph.D. Dissertation, University of Washington. 2018.

[15] Gurkirpal Singh, Glenn Macala, Edward Wong, Robert Rasmussen, Gurkirpal Singh, Glenn Macala, Edward Wong, and Robert Rasmussen. *A Constraint Monitor Algorithm for the Cassini Spacecraft*.

[16] Yoonsoo Kim, Mehran Mesbahi, Gurkirpal Singh, and Fred Hadaegh. On the Convex Parameterization of Constrained Spacecraft Reorientation. *Aerospace and Electronic Systems, IEEE Transactions on*, 46:1097 – 1109, 08 2010.

[17] Glenn Lightsey and Henri Kjellberg. Discretized Constrained Attitude Pathfinding and Control for Satellites. *Journal of Guidance, Control, and Dynamics*, 36:1301–1309, 09 2013.

[18] Unsik Lee and Mehran Mesbahi. Feedback control for spacecraft reorientation under attitude constraints via convex potentials. *IEEE Transactions on Aerospace and Electronic Systems*, 50(4):2578–2592, 2014.

[19] Yoonsoo Kim and Mehran Mesbahi. Quadratically Constrained Attitude Control via Semidefinite Programming. *Automatic Control, IEEE Transactions on*, 49:731 – 735, 06 2004.

[20] H.J. Sussmann and J.C. Willems. 300 Years of Optimal Control: From the Brachystochrone to the Maximum Principle. *IEEE Control Systems Magazine*, 17(3):32–44, June 1997.

[21] Behcet Acikmese and Scott R. Ploen. Convex Programming Approach to Powered Descent Guidance for Mars Landing. *Journal of Guidance, Control, and Dynamics*, 30(5):1353–1366, 2007.

[22] Danylo Malyuta, Yue Yu, Purnanand Elango, and Behcet Acikmese. Advances in Trajectory Optimization for Space Vehicle Control, 2021.

[23] Taylor Reynolds, Michael Szmuk, Danylo Malyuta, Mehran Mesbahi, Behçet Açıkmeşe, and John Carson. Dual Quaternion Based Powered Descent Guidance with State-Triggered Constraints, 04 2019.

[24] Michael Szmuk, Carlo Alberto Pascucci, and Behçet Açıkmeşe. Real-Time Quad-Rotor Path Planning for Mobile Obstacle Avoidance Using Convex Optimization. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9, 2018.

[25] Michael Szmuk, Taylor P. Reynolds, and Behçet Açıkmeşe. Successive Convexification for Real-Time Six-Degree-of-Freedom Powered Descent Guidance with State-Triggered Constraints. *Journal of Guidance, Control, and Dynamics*, 43(8):1399–1413, Aug 2020.

[26] Danylo Malyuta, Taylor Reynolds, Michael Szmuk, Behcet Acikmese, and Mehran Mesbahi. *Fast Trajectory Optimization via Successive Convexification for Spacecraft Rendezvous with Integer Constraints*.

[27] L.S. Pontriagin, V.G. Boltânskij, Karreman Mathematics Research Collection, K.N. Trirogoff, L.W. Neustadt, R.V. Gamkrelidze, and E.F. Mi\^enko. *The Mathematical Theory of Optimal Processes*. Interscience publishers. Interscience Publishers, 1962.

[28] F.L. Lewis, V.L. Syrmos, and V.L. Syrmos. *Optimal Control*. A Wiley-interscience publication. Wiley, 1995.

[29] John T. Betts. Survey of Numerical Methods for Trajectory Optimization. *Journal of Guidance, Control, and Dynamics*, 21(2):193–207, 1998.

[30] Taylor P. Reynolds, Charles L. Kelly, Cole Morgan, Arnela Grebovic, Jerrold Erickson, Henry Brown, William C. Pope, Jonathan Casamayor, Kyle Kearsley, Gorkem Caylak, Kyle E. Fisher, Cameron Wutzke, Kylle Ashton, James Rosenthal, Devan Tormey, Ellory Freneau, Garrett Giddings, Hasan Emin Horata, Anika Dighde, Saharsh Parakh, Jiaping Zhen, John C. Purpura, Daniel B. Pratt, and Anders Hunt. SOC-i: A CubeSat Demonstration of Optimization-Based Real-Time Constrained Attitude Control. In *2021 IEEE Aerospace Conference (50100)*, pages 1–18, 2021.

[31] A. Domahidi, E. Chu, and S. Boyd. ECOS: An SOCP Solver for Embedded Systems. In *European Control Conference (ECC)*, pages 3071–3076, 2013.

[32] Mathworks. Second-Order Cone Programming Solver. 2021.

[33] Bong Wie, Haim Weiss, and Ari Arapostathis. Quaternion feedback regulator for spacecraft eigenaxis rotation. *Journal of Guidance Control and Dynamics*, 12:375–380, 05 1989.

[34] Taewan Kim, Purnanand Elango, Danylo Malyuta, and Behçet Açıkmeşe. Guided Policy Search using Sequential Convex Programming for Initialization of Trajectory Optimization Algorithms. 10 2021.

[35] Richard F. Hartl, Suresh P. Sethi, and Raymond G. Vickson. A Survey of the Maximum Principles for Optimal Control Problems with State Constraints. *SIAM Review*, 37(2):181–218, 1995.

[36] Behcet Acikmese, Daniel Scharf, Lars Blackmore, and Aron Wolf. *Enhancements on the Convex Programming Based Powered Descent Guidance Algorithm for Mars Landing.*

[37] Daniel Dueri, Yuanqi Mao, Zohaib Mian, Jerry Ding, and Behçet Açikmeşe. Trajectory Optimization with Inter-Sample Obstacle Avoidance via Successive Convexification. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 1150–1156, 2017.

[38] Devan Tormey. Guidance Navigation Control System Architecture for the SOCi CubeSat Satellite for Optimal Control and imaging. *UW ResearchWorks Archive*, 00 2020.

[39] Marcel J. Sidi. *Spacecraft Dynamics and Control: A Practical Engineering Approach.* Cambridge Aerospace Series. Cambridge University Press, 1997.

[40] David A Vallado. Fundamentals of Astrodynamics and Applications, 2001.

[41] P. Alken C. Beggan M. Nair G. Cox A. Woods S. Macmillan B. Meyer M. Paniccia A. Chulliat, W. Brown. The US/UK World Magnetic Model for 2020-2025. *National Centers for Environmental Information, NOAA, Tech. Rep.*, 2020.

[42] R. Roehrich T. Kelso, F. Hoots. Spacetrack Report no.3 - Models for Propagation of NORAD Element Sets. *NASA, Tech. Rep.*, 1988.

[43] F. Landis Markley. Attitude Determination Using Two Vector Measurements. 1998.

[44] Christopher G. Mayhew, Ricardo G. Sanfelice, and Andrew R. Teel. On Quaternion-Based Attitude Control and the Unwinding Phenomenon. In *Proceedings of the 2011 American Control Conference*, pages 299–304, 2011.