MILQR Trajectory Optimisation & Tracking for an Under-Actuated Satellite

Author: Nick Goodson

August 25, 2021

1 Introduction

Background

This document is intended as a reference for recreating the optimal attitude control and trajectory optimization system implemented on the Stanford University Satellite PyQubed1. The satellite was built as a student project supervised by Professor Zac Manchester, and had a planned launch date in November 2020. Unfortunately, progress was abruptly halted part way through HITL testing due to Covid-19. The MILQR algorithm described below was developed in Professor Manchester's Robotic Exploration Lab in the Stanford University Department of Aeronautics & Astronautics. I did not invent this algorithm, but was responsible for adapting and implementing it within a flight software system for satellite attitude control.

Objective

The objective is to control the attitude of a satellite in low-earth orbit using only magnetic torque coils (magnetorquers). The cube shaped satellite PyQubed1 has 6 magnetorquers, one on each of it's faces. These actuators allow the satellite to generate a magnetic moment vector in any direction, of a limited magnitude. This magnetic moment interacts with the Earth's magnetic field, creating torques which change the satellite's angular momentum (and hence change the satellite's attitude).
A system is needed to control these magnetorquers such that the desired attitude is reached within a bounded time and with minimal expenditure of electrical power. This control system must also deal with the under-actuated nature of the satellite (discussed in a later section).

2 Satellite Dynamics

We are considering the attitude of the satellite with respect to an Earth-centred Inertial (ECI) reference frame which will be denoted as {N}. The satellite's body frame will be denoted as {B}. Fixed cartesian co-ordinate systems within these reference frames will be assumed. The state, X, consists of the satellite's attitude and angular velocity. The choice of attitude representation is unit quaternions with an active rotation (body to inertial); hence, we have ${}^N q^B \in \mathbb{S}^3$ and $\omega_{B/N} \in \mathbb{R}^3$. A superscript {b} will denote derivatives taken in the body frame. The state vector and it's derivative are

$$X = \begin{bmatrix} {}^N q^B \\ \omega_{B/N} \end{bmatrix}_{7\times 1} \quad \dot{X} = \begin{bmatrix} {}^N \dot{q}^B \\ {}^b\dot{\omega} \end{bmatrix}_{7\times 1} = f(X, U)$$

Rigid Body Dynamics

The satellite is treated as a rigid body and it's motion is modelled using Euler's equations. $J \in \mathbb{R}^{3\times3}$ is the inertia matrix in the satellite's body frame and τ is the net torque vector. We will assume the effects of external torques from gravity gradients, atmospheric drag and solar radiation pressure are negligible in this derivation and treat τ as purely a control input. For brevity, the frame subscripts are dropped from $\omega_{B/N}$.

$$J^b\dot{\omega} + [\omega]_\times J\omega = \tau$$

Where $[\,\cdot\,]_\times$ is denotes the skew symmetric cross product matrix. Rearranging for ${}^b\dot{\omega}$ gives

$${}^b\dot{\omega} = J^{-1}(\tau - [\omega]_\times J\omega)$$

Quaternion Kinematics

The time rate of change of a quaternion is related to the angular velocity by

$$\dot{q} = \frac{1}{2} q \otimes \begin{bmatrix} 0 \\ \omega \end{bmatrix}$$

Noting that a quaternion consists of a scalar component and a 3-dimensional vector component, we can rewrite the above expression as follows

$$q = \begin{bmatrix} s \\ v \end{bmatrix} \quad \rightarrow \quad \dot{q} = \frac{1}{2}\begin{bmatrix} s \\ v \end{bmatrix} \otimes \begin{bmatrix} 0 \\ \omega \end{bmatrix} \quad = \quad \frac{1}{2}\begin{bmatrix} s\cdot 0 - v^T\omega \\ s\omega + v\cdot 0 + v\times\omega \end{bmatrix}$$

$$\rightarrow \quad \dot{q} = \frac{1}{2}\begin{bmatrix} -v^T \\ sI_3 + [v]_\times \end{bmatrix}\omega$$

Linearization

For the iLQR algorithm, we need a first order expansion of the dynamics. When running the optimization, this expansion will be taken at each time-step about a point on a nominal trajectory, $(\bar{X}, \bar{U})$. Note that for now we are setting $U = \tau$. This will be re-evaluated in the section on under-actuated control with magnetorquers.

$$\begin{aligned} \dot{X} = f(X,U) \quad \rightarrow \quad \dot{\bar{X}} + \delta\dot{X} &= f(\bar{X} + \delta X, \bar{U} + \delta U) \\ &\approx f(\bar{X},\bar{U}) + \frac{\partial f}{\partial X}\Big|_{\bar{X},\bar{U}}\delta X + \frac{\partial f}{\partial U}\Big|_{\bar{X},\bar{U}}\delta U \end{aligned}$$

The state and control Jacobians $\frac{\partial f}{\partial X}$ and $\frac{\partial f}{\partial U}$ are found as follows

$$\frac{\partial f}{\partial X} = \begin{bmatrix} \frac{\partial \dot{q}}{\partial q} & \frac{\partial \dot{q}}{\partial \omega} \\ \frac{\partial \dot{\omega}}{\partial q} & \frac{\partial \dot{w}}{\partial w} \end{bmatrix}_{7\times7}$$

$$\frac{\partial \dot{q}}{\partial q} = \frac{1}{2}\begin{bmatrix} 0 & -\omega^T \\ \omega & -[\omega]_\times \end{bmatrix}_{4\times4}; \quad \frac{\partial \dot{q}}{\partial \omega} = \begin{bmatrix} -v^T \\ sI_3 + [v]_\times \end{bmatrix}_{4\times3}; \quad \frac{\partial \dot{\omega}}{\partial q} = \underline{0}_{3\times4}; \quad \frac{\partial \dot{w}}{\partial w} = J^{-1}([J\omega]_\times - [\omega]_\times J)_{3\times3}$$

The partial derivatives above can be checked easily by post-multiplying them with $q = [s\ v]^T$ or ω. The last one is derived from the product rule due to ω appearing twice in Euler's equations. The control Jacobian is

$$\frac{\partial f}{\partial U} = \begin{bmatrix} \frac{\partial \dot{q}}{\partial U} \\ \frac{\partial \dot{\omega}}{\partial U} \end{bmatrix} = \begin{bmatrix} \underline{0}_{4\times3} \\ J^{-1} \end{bmatrix}_{7\times3}$$

Runge-Kutta Step

RK methods are an effective way to forward propagate a dynamical system. The following is an example of how a second order RK-step can be performed for the satellite. The same general process will work for an RK-step of any order. The only trick here is that the quaternion has to be re-normalized after the step to ensure it maintains its unit norm length. This normalization can be applied to the Linearized system as well. In practice that is usually not necessary but it is shown here for completeness.

The RK integration involves a half step, followed by a full step, followed by normalization of the quaternion. Mathematically this is

$$\begin{aligned} x_{k+1/2} &= x_k + \frac{h}{2} f(x_k, u_k) \\ \tilde{x}_{k+1} &= x_k + h f(x_{k+1/2}, u_k) \\ x_{k+1} &= \begin{bmatrix} \tilde{q}_{k+1} \ / \ ||\tilde{q}_{k+1}|| \\ \tilde{\omega}_{k+1} \end{bmatrix} \quad \text{where} \quad ||\tilde{q}|| = \sqrt{\tilde{q}^T \tilde{q}} \end{aligned}$$

If we perform this same RK-step on the linearized system, it is equivalent to finding the discrete time Jacobians $A_k = \frac{\partial x_{k+1}}{\partial x_k}$ and $B_k = \frac{\partial x_{k+1}}{\partial u_k}$. Considering again the half step, followed by the full step, followed by normalization, we can apply the chain rule as follows

$$\begin{aligned} A_k &= \frac{\partial x_{k+1}}{\partial x_k} = \frac{\partial x_{k+1}}{\partial \tilde{x}_{k+1}} \cdot \Big(\frac{\partial \tilde{x}_{k+1}}{\partial x_k} + \frac{\partial \tilde{x}_{k+1}}{\partial x_{k+1/2}} \cdot \frac{\partial x_{k+1/2}}{\partial x_k} \Big) \\ B_k &= \frac{\partial x_{k+1}}{\partial u_k} = \frac{\partial x_{k+1}}{\partial \tilde{x}_{k+1}} \cdot \Big(\frac{\partial \tilde{x}_{k+1}}{\partial u_k} + \frac{\partial \tilde{x}_{k+1}}{\partial x_{k+1/2}} \cdot \frac{\partial x_{k+1/2}}{\partial u_k} \Big) \end{aligned}$$

We derived expressions for $\frac{\partial f}{\partial X}$ and $\frac{\partial f}{\partial U}$ in the previous section. These continuous time Jacobians are used when evaluating the partial derivatives in the expressions above for the discrete time Jacobians. To simplify notation we will use the following aliases for the continuous time Jacobians evaluated at particular values of X and U.

$$\begin{aligned} \text{Let} \quad A_1 &= \frac{\partial f(x_k, u_k)}{\partial x_k} \quad \text{and} \quad A_2 = \frac{\partial f(x_{k+1/2}, u_k)}{\partial x_{k+1/2}} \\ \text{Also, let} \quad B_1 &= \frac{\partial f(x_k, u_k)}{\partial u_k} \quad \text{and} \quad B_2 = \frac{\partial f(x_{k+1/2}, u_k)}{\partial u_k} \end{aligned}$$

Derivations of the partial derivatives used in the expression for A_k are shown below in full. Similar steps apply for B_k. but are omitted here for brevity.

$$\begin{aligned} &\text{We know} \quad x_{k+1/2} = x_k + \frac{h}{2} f(x_k, u_k) \quad \text{therefore,} \quad \frac{\partial x_{k+1/2}}{\partial x_k} = I_7 + \frac{h}{2} A_1 \\ &\text{Similarly} \quad \tilde{x}_{k+1} = x_k + h f(x_{k+1/2}, u_k) \quad \text{so} \quad \frac{\partial \tilde{x}_{k+1}}{\partial x_k} = I_4 \quad \text{and} \quad \frac{\partial \tilde{x}_{k+1}}{\partial x_{k+1/2}} = hA_2 \end{aligned}$$

Lastly we need to consider the gradient of the normalization step, denoted as $\frac{\partial x_{k+1}}{\partial \tilde{x}_{k+1}}$. The time step subscript is omitted in the following derivation. Normalization of the quaternion is given by

$$q = \frac{\tilde{q}}{\sqrt{\tilde{q}^T \tilde{q}}} \quad \text{hence, applying the product rule} \quad \frac{\partial q}{\partial \tilde{q}} = \frac{I_4}{\sqrt{\tilde{q}^T \tilde{q}}} - \frac{\tilde{q}\tilde{q}^T}{(\tilde{q}^T \tilde{q})^{(3/2)}}$$

The angular velocity is unchanged by the normalization step so the full normalization Jacobian is

$$N = \frac{\partial x_{k+1}}{\partial \tilde{x}_{k+1}} = \begin{bmatrix} \frac{I_4}{\sqrt{\tilde{q}^T \tilde{q}}} - \frac{\tilde{q}\tilde{q}^T}{(\tilde{q}^T \tilde{q})^{(3/2)}} & \underline{0}_{4\times 3} \\ \underline{0}_{3\times 4} & I_3 \end{bmatrix}_{7\times 7}$$

Combining all of the above gives an expression for the discrete time state Jacobian. Similar steps are used to find the discrete time control Jacobian.

$$A_k = \frac{\partial x_{k+1}}{\partial x_k} = N\Big(I_7 + hA_2 \cdot (I_7 + \frac{h}{2}A_1)\Big)$$
$$B_k = \frac{\partial x_{k+1}}{\partial u_k} = N\Big(hB_2 + hA_2 \cdot \frac{h}{2}B_1\Big)$$

3 The iLQR algorithm

The iLQR algorithm is an extension of the finite horizon Linear Quadratic Regulator to control of non-linear systems. The algorithm solves for the sequence of controls which minimize a cost functional, J. Typically J is designed to drive the system from an initial state to a goal state. The solution yields an optimal trajectory $\{\mathbf{X}, \mathbf{U}\}$ which is a sequence of states and controls satisfying the system dynamics. Additionally, a sequence of feedback control gains $\{\mathbf{K}\}$ are returned which enable accurate tracking of the optimal trajectory.

The algorithm is finite horizon, meaning it is solved for a fixed number of time steps N with a specified step size, h. The total cost when starting at state x_0 and applying the control sequence $\mathbf{U}$ is given by

$$J(x_0, \mathbf{U}) = \sum_{k=0}^{N-1} \ell(x_k, u_k) + \ell_f(x_N)$$

where ℓ is the cumulative cost and ℓ_f is the terminal cost which is evaluated only at the final state. The objective of the iLQR algorithm can then be expressed mathematically as solving

$$\mathbf{U}^* = \underset{u_0,\ldots,u_{N-1}}{\arg\min}\; J(x_0, \mathbf{U})$$

The basis of the algorithm is as follows:

1. **Obtaining a baseline trajectory**: We roll-out an initial trajectory as a starting point for our optimisation. This is done using the RK-step method described above for N time steps with a random control sequence $\mathbf{U}$. We also calculate the cost J of this baseline trajectory and the discrete time Jacobians A_k and B_k at each time step.

2. **The Value function**: At each time step, we want the control input u_k that minimizes the cost we will accumulate from that point on-ward. We denote the optimal (minimum) cost at all future steps as the "cost to go". We capture this idea mathematically by defining a quantity called the *Value* function, V_k

$$V_k(x) = \min_{u_k}[\ \ell(x_k, u_k) + V_{k+1}(x_{k+1})\ ]$$

where the *Value* function at the next time-step, V_{k+1}, is the optimal cost to go at time step k. Equivalently, we can express the $Value$ function using the (discrete time) non-linear dynamics as

$$V_k(x) = \min_{u_k}[\ \ell(x_k, u_k) + V_{k+1}(f(x_k, u_k))\ ]$$

The cost to go at the final time step is zero and the cost to go at the second to last time step is simply $V_N(x) = \ell_f(x_N)$. Because we have this explicit expression for the cost to go at time step $N-1$, we start our optimisation here and work backwards in time along the trajectory. To make that completely clear, our first step would be to solve

$$V_{N-1}(x) = \min_{u_{N-1}}[\ \ell(x_{N-1}, u_{N-1}) + \ell_f(x_N)\ ]$$

Aside: From the principle of optimality, we know that the optimal control sequence from time step k on-wards is the same as the optimal control for time step k followed by the optimal control sequence from time step $k+1$ on-wards. Hence, we can independently solve for the optimal control at each time step.

3. **Quadratic Value function approximation**: We approximate the *Value* function $V_k(x_k)$ locally around point (x_k, u_k) on the trajectory as $\Delta_2 V_k(\delta x_k)$ where Δ_2 denotes a quadratic Taylor expansion with respect to both x and u. This convex approximation is desirable as its minimum can be found trivially. To find the 2^{nd} order Taylor expansion of the *Value* function we first rename the argument of the minimisation as $Q(x, u)$. The 2^{nd} order expansion of Q for small perturbations around a nominal point on the trajectory is found as follows

$$\begin{aligned} Q_k(x_k + \delta x_k, u_k + \delta u_k) &= \ell(x_k + \delta x_k, u_k + \delta u_k) + V_{k+1}(f(x_k + \delta x_k, u_k + \delta u_k)) \\ &\approx Q_k(x_k, u_k) + \Delta_2 \ell(\delta x_k, \delta u_k) + \Delta_2 V_{k+1}(\delta x_{k+1}) \end{aligned}$$

$$\rightarrow \quad \Delta_2 Q_k(\delta x_k, \delta u_k) = \Delta_2 \ell(\delta x_k, \delta u_k) + \Delta_2 V_{k+1}(A_k \delta x_k + B_k \delta u_k)$$

We see that a quadratic approximation to the *Value* function consists of 2^{nd} order expansions of both the cost function and the cost to go. This approximation is truncated to first order for the dynamics δx_{k+1} allowing us to substitute our local linear approximation $A_k \delta x_k + B_k \delta u_k$. The use of a quadratic *Value* function with linear dynamics is where the linear quadratic regulator (LQR) derives its name. Writing out the expansions explicitly gives the following expressions.

$$\begin{aligned} \Delta_2 \ell(\delta x_k, \delta u_k) &= \ell_{\mathrm{u}}^T \delta u_k + \frac{1}{2} \delta u_k^T \ell_{\mathrm{uu}} \delta u_k + \ell_{\mathrm{x}}^T \delta x_k + \frac{1}{2} \delta x_k^T \ell_{\mathrm{xx}} \delta x_k + \delta u_k^T \ell_{\mathrm{ux}} \delta x_k \\ \Delta_2 V_{k+1}(\delta x_{k+1}) &= V_{\mathrm{x}}^T (A_k \delta x_k + B_k \delta u_k) + \frac{1}{2} (A_k \delta x_k + B_k \delta u_k)^T V_{\mathrm{xx}} (A_k \delta x_k + B_k \delta u_k) \end{aligned}$$

The subscripts of ℓ and V represent differentiation with respect to the variable in the subscript. The time step subscript has also been omitted from V_{k+1}. Grouping terms into Jacobians and Hessians multiplied by δx_k, δu_k or both gives a simplified expression for the local quadratic approximation.

$$\Delta_2 Q(\delta x_k, \delta u_k) = Q_{\mathrm{x}}^T \delta x_k + Q_{\mathrm{u}}^T \delta u_k + \frac{1}{2} \delta x_k^T Q_{\mathrm{xx}} \delta x_k + \frac{1}{2} \delta u_k^T Q_{\mathrm{uu}} \delta u_k + \delta u_k^T Q_{\mathrm{ux}} \delta x_k$$

The terms of this quadratic expansion are given below.

$$\begin{aligned} Q_{\mathrm{x}}^T &= \ell_{\mathrm{x}}^T + V_{\mathrm{x}}^T A \\ Q_{\mathrm{u}}^T &= \ell_{\mathrm{u}}^T + V_{\mathrm{x}}^T B \\ Q_{\mathrm{xx}} &= \ell_{\mathrm{xx}} + A^T V_{\mathrm{xx}} A \\ Q_{\mathrm{uu}} &= \ell_{\mathrm{uu}} + B^T V_{\mathrm{xx}} B \\ Q_{\mathrm{ux}} &= \ell_{\mathrm{ux}} + B^T V_{\mathrm{xx}} A \end{aligned}$$

4. **Solving for the optimal control**: The expressions above are evaluated using the known cost function, known cost to go (initially $\ell_f(x_N)$) and the discrete time Jacobians. We can now solve for the control δu_k^* which minimises $\Delta_2 Q(\delta x_k, \delta u_k)$. Because this approximate *Value* function is quadratic, we can simply find its gradient with respect to the control and set it equal to zero.

$$\begin{aligned} \Delta_2 V_k(\delta x_k) &= \min_{\delta u_k} \Delta_2 Q_k(\delta x_k, \delta u_k) \\ \rightarrow \quad \delta u_k^*(\delta x_k) &= \arg\min_{\delta u_k} \Delta_2 Q_k(\delta x_k, \delta u_k) \\ \frac{\partial \Delta_2 Q_k(\delta x_k, \delta u_k)}{\partial \delta u_k} &= Q_{\mathrm{u}} + Q_{\mathrm{uu}} \delta u_k + Q_{\mathrm{ux}} \delta x_k = 0 \end{aligned}$$

$$\begin{aligned} \rightarrow \quad \delta u_k^*(\delta x_k) &= -Q_{\mathrm{uu}}^{-1} (Q_{\mathrm{u}} + Q_{\mathrm{ux}} \delta x_k) \\ &= -l_k - K_k \delta x_k \end{aligned}$$

This optimal control perturbation consists of a feed-forward correction l_k and feedback gain matrix K_k. These terms are combined with the nominal control u_k in a forward line search.

5. **Finding the new cost to go**: After solving for the optimal control at the second to last time step $N-1$, we want to do the same thing at the the previous time step $N-2$. This requires an expression for the cost to go from $N-2$. From the definition of the *Value* function, the cost to go for the previous time step is simply the optimal *Value* at the current time step. Hence, we sub the optimal control into the expression for the approximate *Value* function at time $N-1$.

$$\Delta_2 V_{N-1}(\delta x_{N-1}) = \Delta_2 Q_{N-1}(\delta x_{N-1}, \delta u^*_{N-1})$$

   This method for obtaining the cost to go for time step $N-2$ works for a general time step $k-1$. Expanding gives

$$\Delta_2 V_k(\delta x_k) =$$
$$Q_{\text{x}}^T \delta x_k - Q_{\text{u}}^T(l_k + K_k\delta x_k) + \frac{1}{2}\delta x_k^T Q_{\text{xx}}\delta x_k + \frac{1}{2}(l_k + K_k\delta x_k)^T Q_{\text{uu}}(l_k + K_k\delta x_k) - (l_k + K_k\delta x_k)^T Q_{\text{ux}}\delta x_k$$

   grouping terms we get

$$\Delta_2 V_k(\delta x_k) =$$
$$(Q_{\text{x}}^T - Q_{\text{u}}^T K_k + l_k^T Q_{\text{uu}} K_k - l_k^T Q_{\text{ux}})\delta x_k + \frac{1}{2}\delta x_k^T(Q_{\text{xx}} + K_k^T Q_{\text{uu}} K_k - 2K_k^T Q_{\text{ux}})\delta x_k + (\frac{1}{2} l_k^T Q_{\text{uu}} l_k - Q_{\text{u}}^T l_k)$$
$$= \mathbf{V}_{\text{x}}{}^T \delta x_k + \frac{1}{2}\delta x_k^T \mathbf{V}_{\text{xx}}\delta x_k + d\mathbf{V}$$

   This expression for the approximate cost to go at time step $k-1$ is already in a quadratic form. Also note the constant term $d\mathbf{V}$ which comes from the cost of the feed-forward control correction, l_k.

6. **The backward pass**: After solving the optimization at the second to last time step $N-1$, using $l_f(x_N)$ as our cost to go, we now know how to find the cost to go for the previous time step. At that time step we solve for the optimal control again. This process (called the backward pass) is repeated until reaching the first time step. The result is a set of $N-1$ feed-forward controls l and feedback gain matrices K.

7. **Control correction line-search**: The optimal controls l and K found in the backward pass are corrections to the nominal control trajectory $\{\mathbf{U}\}$. By approximating the *Value* function around the nominal trajectory $\{\mathbf{X}, \mathbf{U}\}$, we solved for the optimal **change** in control δu^* at each time step. Hence, the new control at each time step should be $u_k + \delta u_k^*$. To apply these corrections we roll-out a new trajectory from our initial state x_0, updating the nominal control at each time step.

   However there is an issue here. A quadratic approximation of the *Value* function was used when solving for δu_k^*. This control is not truly optimal unless the true *Value* function is also a quadratic. In reality directly applying this correction could potentially increase rather than decrease the *Value.* This problem is illustrated by *Figure 1.* Note that the marker at the lowest point of the orange curve corresponds to the approximate optimal control δu_k^*. The true desired optimum is shown by the circular blue marker.

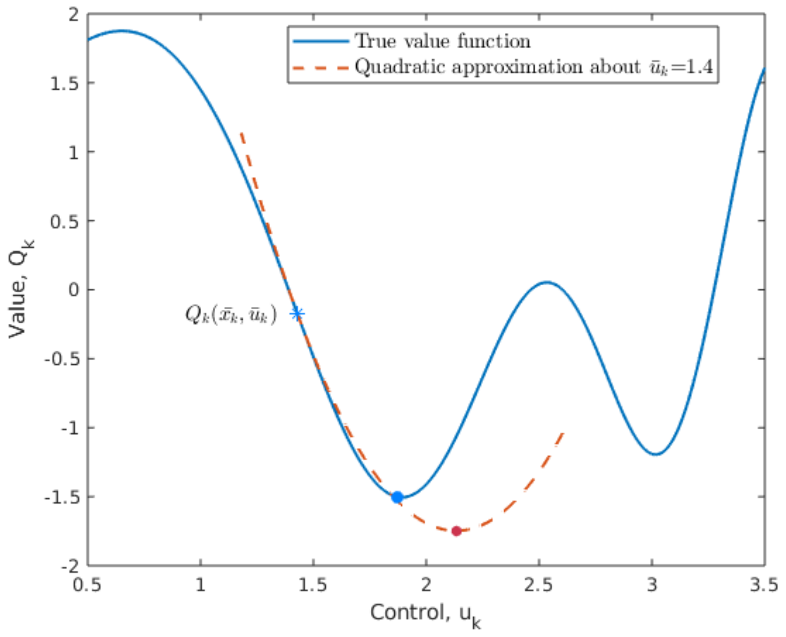

Figure 1: Illustration of why minimizing with respect to a quadratic approximation of the *Value* function is sub-optimal. The real value function is a high-dimensional surface, and the 2D plot shown is for illustration purposes only

To overcome this issue we employ a line-search during the new forward-rollout. The basis of the line search is multiplying the feed-forward control correction l_k by a scaling factor, $\alpha \leq 1$. We initially start with $\alpha = 1$, then decrease it until the cost reduction of the new trajectory is sufficiently large. The forward roll-out line search algorithm is included here for clarity.

Algorithm 1: iLQR line-search

```
Result: The new nominal trajectory $\{\mathbf{X}, \mathbf{U}\}$
$x_0^{new} \leftarrow x_0$;
$\alpha \leftarrow 1$;
while $cost - cost^{new} < tol$ do
    for $k=0$ to $N-1$ do
        $u_k^{new} \leftarrow u_k - \alpha l_k - K_k(x_k^{new} - x_k)$
        $x_{k+1}^{new} \leftarrow f(x_k^{new}, u_k^{new})$
        $cost^{new} \leftarrow cost^{new} + \ell(x_k^{new}, u_k^{new})$
    end
    $cost^{new} \leftarrow cost^{new} + \ell_f(x_N^{new})$;
    $\alpha \leftarrow \alpha/2$
end
```

8. **Repeat**: Everything from step 2 to this point was one iteration of the iLQR optimisation. The whole process is repeated many times until convergence. Convergence is achieved when either the feed-forward control corrections l_k, or the change in cost $(cost - cost^{new})$ become sufficiently small. Convergence is discussed in more detail under **Implementation Details**.

4 Control Limits

Box Constraints

The vanilla iLQR algorithm assumes that a control input u_k of any magnitude can be applied at each time step. Real systems use actuators with limited capabilities. For the satellite, the control input is the torque τ from the magnetorquers. The simplest way to capture control limits is with box constraints

$$\underline{\mathbf{b}} \leq \mathbf{u} \leq \overline{\mathbf{b}}$$

where $\underline{\mathbf{b}}, \overline{\mathbf{b}}$ are vectors containing the lower and upper control bounds. These box constraints can naively be enforced by clamping the controls. Element wise clamping is denoted by $[\![\cdot]\!]_b$ where

$$[\![u_k]\!]_b = \min\Big(\overline{\mathbf{b}}, \max(u_k, \underline{\mathbf{b}})\Big)$$

The issue with this approach is that if the optimal control is outside of the control bounds, the clamped controls will no longer be optimal. In fact, after clamping the search direction may no longer correspond to a descent direction, harming convergence.

Box Quadratic-Program (BoxQP) Solver

A better method which involves solving a constrained optimisation for the control is described here. Rather than solving for δu_k^* by setting the gradient of the quadratic *Value* function approximation equal to zero, we solve the following quadratic program at each time step:

$$\begin{aligned} \underset{\delta u}{\text{minimize}} \quad & \Delta_2 Q(\delta x, \delta u) \\ \text{subject to} \quad & \underline{\mathbf{b}} \leq u + \delta u \leq \overline{\mathbf{b}} \end{aligned}$$

The solver selected for this problem is a Projected-Newton method for the following reasons: The iLQR algorithm takes advantage of the principle of optimality such that at every time step a small QP is solved rather than solving one large QP for the entire trajectory. This limits the size of the hessian. Additionally, each QP along the backward pass is similar to the previous one, so warm starting the algorithm is possible. In this case, warm starting means setting the initial guess for the optimal feed-forward control, $l_k = l_{k+1}$. Lastly, the Projected-Newton algorithms are a subclass of active set methods which are specifically designed for problems with simple constraints such as box-clamping.

The objective is to solve for the feed-forward corrections, l_k and the feedback gains K_k; however, additional control input from feedback $K_k \delta x_k$ can only be applied if a control isn't already saturated. This means rows of the feedback gain matrix corresponding to clamped controls should be zeroed out. The optimal feed-forward gains are obtained from the result of

$$\begin{aligned} l_k = \underset{\delta u}{\arg\min} \quad & Q_{\mathrm{u}}^T \delta u + \frac{1}{2} \delta u^T Q_{\mathrm{uu}} \delta u \\ \text{subject to} \quad & (\underline{\mathbf{b}} - u) \leq \delta u \leq (\overline{\mathbf{b}} - u) \end{aligned}$$

where the objective function is simply the relevant part of $\Delta_2 Q(\delta x, \delta u)$ (the other terms containing δx are essentially constants when optimising δu). Note this objective function does not contain $\delta u^T Q_{\mathrm{ux}} \delta x$. as we are solving exclusively for l_k. The feedback term is calculated afterwards using Q_{ux}.

Projected-Newton Optimization

Consider the optimization problem above for l_k rewritten in a more general form in terms of a variable x.

$$\underset{x}{\text{minimize}} \quad f(x) = q^T x + x^T Q x$$
$$\text{subject to} \quad \underline{\mathbf{b}} \leq x \leq \overline{\mathbf{b}}$$

At the start of each iteration, we determine which entries of x are constrained (clamped). The first iteration begins with a guess of x, which we clamp appropriately $x = [\![x]\!]_b$. The gradient of the objective function is $\nabla_{\mathrm{x}} f = q + Qx$, the Hessian is $\nabla_{\mathrm{xx}} f = Q$. Newtons method applied to optimization [2] has the standard update:

$$\Delta x = -\frac{f'(x)}{f''(x)} \quad \text{or for multiple dimensions} \quad \Delta x = -\nabla_{\mathrm{xx}} f^{-1} \nabla_{\mathrm{x}} f$$

However, we only want to perform the Newton step in the free subspace (the non-clamped entries of x). The complementary clamped and free indices of $x \in \mathbb{R}^n$ are

$$\mathrm{c} = j \in 1, ..., n \;\; \text{where} \begin{cases} x_j = \underline{\mathbf{b}}_j \text{ and } \;\nabla_{\mathrm{x}} f_j > 0 \\ \textit{or} \\ x_j = \overline{\mathbf{b}}_j \text{ and } \;\nabla_{\mathrm{x}} f_j < 0 \end{cases}$$

$$\mathrm{f} = j \in 1, ..., n \;\; \text{where} \;\; \{ \; j \neq c$$

Entries of x at the boundaries are only clamped if the gradient direction is such that the Newton step would move the entry x_j outside the constraints. To improve readability, we sort into the index partitions $\{\mathrm{f}, \mathrm{c}\}$:

$$x \leftarrow \begin{bmatrix} x_{\mathrm{f}} \\ x_{\mathrm{c}} \end{bmatrix} \quad q \leftarrow \begin{bmatrix} q_{\mathrm{f}} \\ q_{\mathrm{c}} \end{bmatrix} \quad Q \leftarrow \begin{bmatrix} Q_{\mathrm{ff}} & Q_{\mathrm{fc}} \\ Q_{\mathrm{cf}} & Q_{\mathrm{cc}} \end{bmatrix}$$

We can now take our Newton step in the free subspace. The gradient in the free subspace is.

$$\nabla_{\mathrm{x_f}} f = q_{\mathrm{f}} + Q_{\mathrm{ff}} x_{\mathrm{f}} + Q_{\mathrm{fc}} x_{\mathrm{c}}$$

Hence, the step in the free subspace is given by

$$\Delta x_{\mathrm{f}} = -Q_{\mathrm{ff}}^{-1} \nabla_{\mathrm{x_f}} f = -Q_{\mathrm{ff}}^{-1}(q_{\mathrm{f}} + Q_{\mathrm{fc}} x_{\mathrm{c}}) - x_{\mathrm{f}}$$

The full Newton step is then

$$\Delta x = \begin{bmatrix} \Delta \mathrm{x_f} \\ 0 \end{bmatrix}$$

In a similar manner to the control update of the iLQR forward pass, we use a line search to update x. Every iteration of the line search can change which indices are clamped, as the step size α is reduced.

$$x_{new} = [\![x + \alpha \Delta x]\!]_b$$

The parameter α is initially set to 1 then reduced until the Armijo condition is satisfied:

$$\frac{f(x) - f(x_{new})}{\nabla_{\mathrm{x}} f^T \alpha \Delta x} > \gamma$$

where $0 < \gamma < \frac{1}{2}$ is the minimum acceptable cost reduction ratio. Note that the denominator of the Armijo condition is the expected cost reduction for the step.

The entire process above is repeated for more Newton steps until either the change in cost becomes sufficiently small, the gradient becomes sufficiently small or an iteration ends with all the indices clamped.

When applied to our control problem, the boxQP solver returns l_k as well as the indices that are clamped. The free dimensions of Q_{uu}, denoted as Q_{ff} are used to find the corresponding free rows of the feedback gain matrix. These are calculated as $K_{\mathrm{f}} = Q_{\mathrm{ff}}^{-1} Q_{\mathrm{ux,f}}$. The rows of K corresponding to clamped controls K_{c} are zeroed out.

5 Multiplicative Error Representations

This section is based on the work done in [3]. Satellites are capable of arbitrarily large three-dimensional rotations meaning the choice of attitude representation is important. Even though rotations are three-dimensional, there are no singularity free three parameter attitude representations. A common solution to this problem is to use a higher dimensional attitude representation such as quaternions, $q \in \mathbb{R}^4$. However, applying vector calculus methods to functions involving quaternions is non-trivial and requires consideration of the space in which differential rotations live.

Unit Quaternions

For this application we have chosen to work with unit quaternions $q \in \mathbb{S}^3$, $||q|| = 1$. Some utilities for working with unit quaternions are defined in this section. Multiplication is defined as

$$q_2 \otimes q_1 = L(q_2)q_1 = R(q_1)q_2$$

where $L(q)$ and $R(q)$ are orthonormal matrices,

$$L(q) = \begin{bmatrix} s & -v^T \\ v & sI_3 + [v]_\times \end{bmatrix}$$
$$R(q) = \begin{bmatrix} s & -v^T \\ v & sI_3 - [v]_\times \end{bmatrix}$$

The inverse of a quaternion is equivalent to a rotation in the opposite direction. For a unit quaternion, its inverse q^{-1} is equal to its conjugate q^*. Some useful definitions derived from this are

$$q^{-1} = q^* = \begin{bmatrix} s \\ -v \end{bmatrix}$$
$$L(q^{-1}) = L(q)^{-1} = L(q)^T$$
$$R(q^{-1}) = R(q)^{-1} = R(q)^T$$

To rotate a vector $r \in \mathbb{R}^3$ by a quaternion, we transform the vector into a quaternion with a zero scalar part. This operation can be written as

$$\tilde{r} = Hr := \begin{bmatrix} 0 \\ I_3 \end{bmatrix} r$$

Quaternion Differential Calculus

Derivatives describe the relationship between an infinitesimal perturbation input to a function and the corresponding infinitesimal output. For vector spaces such as $\mathbb{R}^3$, the infinitesimal input lives in the same space as the nominal input and they are added together using standard vector addition. Quaternions describing rotations are compounded by quaternion multiplication rather than vector addition and infinitesimal unit quaternions live on the tangent plane to $\mathbb{S}^3$, as illustrated by Figure 2. This means infinitesimal rotations live in $\mathbb{R}^3$ which is logical given that they share this space with angular velocity. Extra information on this topic is given in Appendix 1. This fundamental difference between rotations and other vectors mean the standard methods for taking derivatives of vectors cannot be applied without modification.

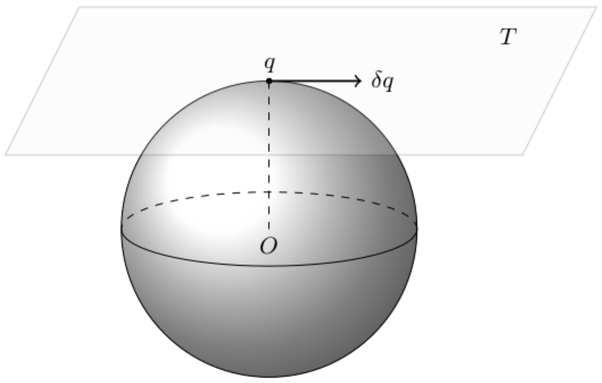

Figure 2: Infinitesimal unit quaternions are confined to a plane in $\mathbb{R}^3$ tangent to $\mathbb{S}^3$, illustrated here with a plane in $\mathbb{R}^2$ tangent to a sphere in $\mathbb{S}^2$. Credit [3]

Given that infinitesimal rotations live in $\mathbb{R}^3$, we would like a three parameter representation for these perturbations. The representation chosen here is Rodrigues parameters. The map from Rodrigues parameters $\phi \in \mathbb{R}^3$ to unit quaternions is called the Cayley Map, $\varphi : \mathbb{R}^3 \to \mathbb{S}^3$.

$$q = \varphi(\phi) = \frac{1}{\sqrt{1 + ||\phi||_2}} \begin{bmatrix} 1 \\ \phi \end{bmatrix}$$

The inverse Cayley Map is

$$\phi = \varphi^{-1}(q) = \frac{v}{s} = \frac{\mathbf{r} sin(\theta/2)}{cos(\theta/2)} = \mathbf{r} tan(\theta/2)$$

From the above definition of φ^{-1} using $tan(\cdot)$ we can see that singularities will occur at $\theta = \pi$. This should not be a concern given that we are using the Rodrigues parameters to represent small rotations. If this singularity were an issue, we could use Modified Rodrigues parameters which are defined in exactly the same way except using the "square-root" of the quaternion. This corresponds to halving the rotation so that singularities occur at 2π.

Jacobians and Hessians of Functions with Quaternion Inputs

Consider finding the Jacobian of a vector valued function with a vector input $z = g(x) : \mathbb{R}^n \to \mathbb{R}^n$. We perform a first order expansion by considering a perturbation to the input and output.

$$\begin{aligned} z + \delta z &= g(x + \delta x) \\ &\approx g(x) + \frac{\partial g(x)}{\partial x}\Big|_x \delta x \end{aligned}$$

Note the Jacobian $\frac{\partial g}{\partial x}$ is evaluated at the nominal input x. We want the equivalent of this for a vector valued function with quaternion input $y = f(q) : \mathbb{S}^3 \to \mathbb{R}^n$. We will apply a differential rotation $\phi \in \mathbb{R}^3$ to this function .

$$\begin{aligned} y + \delta y &= f(q \otimes \delta q) = f(L(q)\varphi(\phi)) \\ &\approx f(q) + \frac{\partial f(q)}{\partial \phi}\Big|_q \phi \end{aligned}$$

We calculate the Jacobian $\frac{\partial f}{\partial \phi}$ by differentiating $f(L(q)\varphi(\phi))$ with respect to ϕ and evaluating it at $\phi = 0$. This is equivalent to evaluating the Jacobian at the nominal input q, just as we would for any other Jacobian.

$$\begin{aligned}\frac{\partial f(q)}{\partial \phi}\Big|_q &= \frac{\partial}{\partial \phi}\Big(f(L(q)\varphi(\phi))\Big) = \frac{\partial f}{\partial q}\cdot\frac{\partial q}{\partial \phi} \;=\; \frac{\partial f}{\partial q}L(q)\frac{\partial \varphi(\phi)}{\partial \phi}\Big|_{\phi=0} \\ &= \frac{\partial f}{\partial q}L(q)H \;=\; \frac{\partial f}{\partial q}G(q) \;=\; \frac{\partial f}{\partial q}\begin{bmatrix} -v^T \\ sI_3 + [v]_\times \end{bmatrix}\end{aligned}$$

Where we have defined the attitude Jacobian $G(q) = L(q)H$. Note that $L(q)$ is essentially a constant when differentiating with respect to ϕ. Additionally, differentiating the Cayley map gives a Jacobian, that when evaluated at $\phi = 0$, returns the matrix H defined earlier.

If $y = f(q)$ returns a scalar, we can define the Hessian of $f(q)$ by taking the Jacobian of its Jacobian $\frac{\partial f}{\partial \phi}$ with respect to ϕ. Again we evaluate this at $\phi = 0$.

$$\nabla_2 f(q) = \frac{\partial}{\partial \phi}\Big(\frac{\partial f}{\partial q}L(q)\frac{\partial \varphi(\phi)}{\partial \phi}\Big)$$

applying the product rule

$$= \frac{\partial}{\partial \phi}\Big(\frac{\partial f}{\partial q}\Big)\cdot L(q)\frac{\partial \varphi(\phi)}{\partial \phi}\Big|_{\phi=0} \;+\; \frac{\partial f}{\partial q}\cdot\frac{\partial}{\partial \phi}\Big(L(q)\frac{\partial \varphi(\phi)}{\partial \phi}\Big)$$

Note that

$$L(q)\frac{\partial \varphi(\phi)}{\partial \phi}\Big|_{\phi=0} = G(q)$$

We also know that L(q) is essentially a constant when differentiated with respect to ϕ. Therefore we get

$$\nabla_2 f(q) = \frac{\partial}{\partial \phi}\Big(\frac{\partial f}{\partial q}\Big)G(q) \;+\; \frac{\partial f}{\partial q}L(q)\frac{\partial^2 \varphi(\phi)}{\partial \phi^2}\Big|_{\phi=0}$$

When differentiating $\frac{\partial f}{\partial q}$, we can apply the chain rule in the same way as we did for the Jacobian. Finding the second derivative of the Cayley map is a verbose exercise not shown here because it is a vector valued function (The result is a vector of Hessians). However, it is simplified dramatically when evaluated at $\phi = 0$ giving the following result

$$\nabla_2 f(q) = G(q)^T\frac{\partial^2 f}{\partial q^2}G(q) \;+\; I_3\frac{\partial f}{\partial q}q$$

Jacobians of Quaternion Valued Functions

So far we have looked at functions taking in unit quaternions and returning a vector or a scalar. We now consider functions which map quaternions to quaternions, $q' = f(q) : \mathbb{S}^3 \to \mathbb{S}^3$. When considering a differential rotational perturbation at the input ϕ, there is a differential rotation at the output ϕ'.

$$L(q')\varphi(\phi') = f(L(q)\varphi(\phi))$$

The Jacobian relates ϕ' to ϕ; hence, we rearrange the previous expression for ϕ'.

$$\phi' = \varphi^{-1}\Big(L(q')^T f(L(q)\varphi(\phi))\Big)$$

The Jacobian is a local linearization of this relationship, about $\phi = 0$. Hence we differentiate with respect to ϕ. Applying the chain rule gives

$$\begin{aligned}\frac{\partial \phi'}{\partial \phi}\Big|_{\phi=0} &= \frac{\partial \varphi^{-1}}{\partial q}\cdot\frac{\partial}{\partial \phi}\Big(L(q')^T f(L(q)\varphi(\phi))\Big) \\ &= H^T L(q')^T\frac{\partial f}{\partial q}L(q)H \\ &= G(q')^T\frac{\partial f}{\partial q}G(q)\end{aligned}$$

The leading H^T appears because as $\phi \to 0$ (and therefore $\phi' \to 0$) $L(q')^T f(q) \to I_q$ where Iq is the identity quaternion. When differentiating the inverse Cayley Map evaluated at the Identity quaternion we see $\frac{\partial \varphi}{\partial q} \to H^T$.

6 MILQR

Some modifications are required to incorporate multiplicative error representations into the iLQR algorithm. The resulting algorithm first proposed in [3] has been named "MILQR".

Linearized Dynamics

The previously derived linearization of the satellite's dynamics does not properly account for differential rotations. The current linearized dynamics about a nominal state $[\overline{q}_k, \overline{\omega}_k]^T$ are

$$\begin{aligned} \delta x_{k+1} &= A_k \delta x_k + B_k \delta u_k \\ \text{where} \quad \delta x_k &= \begin{bmatrix} \delta q \\ \delta \omega \end{bmatrix} = \begin{bmatrix} q_k - \overline{q}_k \\ \omega_k - \overline{\omega}_k \end{bmatrix} \in \mathbb{R}^7 \end{aligned}$$

This perturbation vector δx_k is not ideal because the differential rotation (currently δq) should live in $\mathbb{R}^3$ and should be determined from the nominal attitude $\overline{q}_k$ using quaternion multiplication rather than subtraction. Making these changes, our new error vector is

$$\delta x_k = \begin{bmatrix} \phi \\ \delta \omega \end{bmatrix} = \begin{bmatrix} \varphi^{-1}(\overline{q}_k^{-1} \otimes q_k) \\ \omega_k - \overline{\omega}_k \end{bmatrix} \in \mathbb{R}^6$$

We also need to update our Jacobians, using the expressions derived in the previous section with the attitude Jacobian $G(q)$. Noting that only the quaternion part needs correcting, the Jacobians are updated using

$$\begin{aligned} E(x) &= \begin{bmatrix} G(q) & \underline{0}_{4\times 3} \\ \underline{0}_{3\times 3} & I_3 \end{bmatrix}_{7\times 6} \\ \underline{A}_k &= E(x_{k+1})^T A_k E(x_k) \\ \underline{B}_k &= E(x_{k+1})^T B_k \end{aligned}$$

Note that now the feedback gain matrices are $K \in \mathbb{R}^{3\times 6}$ (they map a 6×1 error state vector to a 3×1 control vector). This means that during the forward pass, the error between the satellite's new state and the nominal trajectory needs to be mapped to a 6×1 state error vector using the inverse Cayley map on the quaternion part. The same mapping also needs to be used when running the controller on the real satellite. In that instance the error is the difference between the measured satellite attitude and the nominal trajectory output from MILQR.

Cost function

A cost function is required which will drive the system from an initial state to a desired final state. In the iLQR algorithm, the cost when starting at state x_0 and applying the control sequence $\mathbf{U}$ was defined as

$$J(x_0, \mathbf{U}) = \sum_{k=0}^{N-1} \ell(x_k, u_k) + \ell_f(x_N)$$

where ℓ is the cumulative cost and ℓ_f is the terminal cost which is evaluated only at the final state. We know that the iLQR algorithm requires a second order expansion of the cost function, and that we want a function that is easy to optimise. One function for which the second order expansion can be found trivially and which is easy to optimise is a quadratic. Hence, a possible cost function is

$$\ell(x,u) = \frac{1}{2}(x-x_g)^T Q(x-x_g) + \frac{1}{2}u^T R u$$
$$\ell_f(x_N) = \frac{1}{2}(x_N-x_g)^T Q_f(x_N-x_g)$$

where x_g is the desired final state (or a point on a desired trajectory). Without these error terms between the current and the goal state, the controller would only work as a regulator driving the state towards the origin. Using subtraction to find the error is not a suitable method for quaternions. A simple alternative cost function for quaternions is

$$\ell(q) = 1 - |q_g^T q|$$

This cost function is actually the geodesic distance on the unit sphere $\mathbb{S}^3$ between the current attitude and the desired attitude. At the goal, the term $q_g^T q = \pm 1$. The Jacobian and Hessian of this cost function can be found using the expressions derived earlier for vector-valued functions taking quaternion inputs. Specifically

$$\ell_{\mathrm{q}}^T = \pm q_g^T G(q)$$
$$\ell_{\mathrm{qq}} = \pm I_3(q_g^T q)$$

The Jacobian takes on the sign which minimizes $\ell(q)$ and the Hessian takes on the sign of $q_g^T q$.
Using a quadratic cost for the angular velocity and the geodesic cost for the attitude, our cost function becomes

$$\ell(x,u) = w_c(1-|q_g^T q|) \;+\; \frac{1}{2}(\omega-\omega_g)^T Q(\omega-\omega_g) + \frac{1}{2}u^T R u$$
$$\ell_f(x_N) = w_f(1-|q_g^T q_N|) \;+\; \frac{1}{2}(\omega_N-\omega_g)^T Q_f(\omega_N-\omega_g)$$

where w_c and w_f are weighting factors and the matrices $Q, Q_f \in \mathbb{R}^3$ and $R \in \mathbb{R}^3$ are diagonal and positive definite. We need a quadratic expansion of this cost function to plug into iLQR. For the cumulative cost this can be derived as follows

$$\begin{aligned}\Delta_2\ell(\delta x,\delta u) &= \ell_{\mathrm{q}}^T\phi + \ell_{\omega}^T\delta\omega + \ell_{\mathrm{u}}^T\delta u + \frac{1}{2}\phi^T\ell_{\mathrm{qq}}\phi + \frac{1}{2}\delta\omega^T\ell_{\omega\omega}\delta\omega + \frac{1}{2}\delta u^T\ell_{\mathrm{uu}}\delta u \\ &= \pm w_c q_g^T G(q)\phi + (\omega-\omega_g)^T Q\delta\omega + u^T R\delta u \;\pm \frac{1}{2}\phi^T w_c I_3(q_g^T q)\phi + \frac{1}{2}\delta\omega^T Q\delta\omega + \frac{1}{2}\delta u^T R\delta u\end{aligned}$$

We can ignore ℓ_{ux} as it is zero for the cost function we have selected. The standard form for iLQR is found by merging the q and ω terms. To do this, we set

$$\delta x = \begin{bmatrix}\phi \\ \delta\omega\end{bmatrix}_{6\times 1}$$

The terms of this expansion can then be written as

$$\ell_{\mathrm{x}} = \begin{bmatrix}\pm w_c G(q)^T q_g \\ Q(\omega-\omega_g)\end{bmatrix}_{6\times 1}$$
$$\ell_{\mathrm{u}} = [Ru]_{3\times 1}$$
$$\ell_{\mathrm{xx}} = \begin{bmatrix}\pm w_c I_3(q_g^T q) & \underline{0}_3 \\ \underline{0}_3 & Q\end{bmatrix}_{6\times 6}$$
$$\ell_{\mathrm{uu}} = R_{3\times 3}$$

Using a similar method, the terms for the terminal cost are

$$\ell_{f\mathrm{x}} = \begin{bmatrix} \pm w_f G(q)^T q_g \\ Q_f(\omega - \omega_g) \end{bmatrix}_{6\times 1}$$
$$\ell_{f\mathrm{xx}} = \begin{bmatrix} \pm w_f I_3(q_g^T q) & \underline{0}_3 \\ \underline{0}_3 & Q_f \end{bmatrix}_{6\times 6}$$

7 Under-actuated Control with Magnetorquers

The torque applied to the satellite by the interaction of its magnetoruqers with the Earth's magnetic field is given by

$$\begin{aligned}\tau &= u \times B \\ &= -[B]_\times u\end{aligned}$$

where u is the magnetic moment created by the torque coils and B is the local Earth magnetic field vector. The cross product means that the satellite can only create torques whose resultant vector lies in a plane perpendicular to the local Earth magnetic field lines. This prevents the satellite from being able to rotate itself around an axis parallel with the Earth's magnetic field lines. The system is therefore under-actuated as not all degrees of freedom are directly controllable. Fortunately it is still possible to control all 3DOF of the satellite's attitude due to the changes in the local Earth magnetic field vector as the satellite moves along its orbit. A control trajectory can take advantage of these changes to generate torques in all three axes of the inertial frame.

Controllability Grammian

For the satellite attitude to be controllable, there must be a sufficient amount of change in the Earth's magnetic field during the actuation time of the control trajectory. Since MILQR is a finite horizon optimization algorithm, we need to know how long the control trajectory should be in order to achieve controllability. The standard method for checking the controllabiltiy of a time varying linear system is the controllability Grammian.

$$C = \int_{t_0}^{t_f} e^{A\tau} B B^T e^{A^T \tau} d\tau$$

where $e^{A\tau}$ is the state transition matrix. If the matrix $C \in \mathbb{R}^{n\times n}$ is full rank, the system is controllable on the interval $[t0, tf]$. We will use the controllability grammian of the dynamical system describing the satellite's angular momentum. This will provide a useful heurisitc for determining how long the control trajectory will need to be. In the inertial frame, the angular momentum of the satellite can be described using the first order differential equation

$$\dot{L} = -[B]_\times u$$

This gives the grammian

$$C = \int_{t_0}^{t_f} [B]_\times [B]_\times^T d\tau$$

The skew symmetric matrix $[B]_\times \in \mathbb{R}^3$ is always rank 2; however, it is time varying due to the changing Earth magnetic field along the orbit. For full 3-axis controllability we need C to be rank 3. The condition number of C can be used as an indication of its invertability (and hence rank), so we numerically integrate the grammian forward in time until the condition number reaches a tolerance. This gives the final time for the control maneuver. (Note: the condition number is defined as the ratio of the maximum and minimum singular values of a matrix).

In order to find the Earth's magnetic field at given times on an orbit, we use a high fidelity magnetic field model. In this case the International Geomagnetic Reference Field (IGRF). The orbit itself is propagated using a model such as SGP4.

Adjustments to the Algorithm

Some small adjustments are made to the algorithm to incorporate magnetorquer control.
Euler's equations become

$$J^b\dot{\omega} + [\omega]_\times J\omega = -B_\times u$$

so the continuous time control Jacobian becomes

$$\frac{\partial f}{\partial U} = \begin{bmatrix} \underline{0}_{4\times 3} \\ -J^{-1}[B]_\times \end{bmatrix}$$

The sequence of magnetic field vectors $\mathbf{B}$ along the orbit are passed into the algorithm in the inertial frame (ECI). At each time step, the local magnetic field vector B has to be rotated into the body frame using the current satellite attitude q_k.

8 Implementation Details

Regularization

In the iLQR algorithm (and the BoxQP solver) we invert Q_{uu} to solve for the optimal controls l_k and K_k. In general terms we are solving problems of the form $Ax = B$ for x. However, there are more efficient ways of solving this equation which don't involve a matrix inversion. Specifically, we know that Q_{uu} should be a positive definite matrix because

$$Q_{\mathrm{uu}} = \ell_{\mathrm{uu}} + B^T V_{\mathrm{xx}} B$$

and we selected $\ell_{\mathrm{uu}} = R$ to be a positive definite matrix. Additionally, the value function is always positive so its hessian V_{xx} must be positive definite. A stable and efficient solution to a matrix equation involving a positive definite matrix is to use a Cholesky factorisation. This involves finding a matrix L such that $LL^T = Q_{\mathrm{uu}}$. The solution to the system $Ax = b$ can then be found by back substitution of $LY = B$ followed by $L^T x = Y$.
An issue with implementing Cholesky factorisations is that floating point errors can make the matrix Q_{uu} non positive definite. A simple solution to overcome this is regularization. We regularize by setting $Q_{\mathrm{uu}} = Q_{\mathrm{uu}} + \lambda I$ where I is the identity matrix and λ is a regularization parameter. This idea comes from the Levenberg-Marquardt algorithm. We initially start with a small $\lambda \leq 1$. If the Cholesky factorisation fails we increase λ then try again, and if it succeeds we decrease λ and move to a new iteration. Ideally λ should be close to zero before the algorithm can be said to have converged.

Appendix

Appendix 1 - Quaternion Derivatives

The fact that differential rotations live in the same space as angular velocity is illustrated further by rearranging the quaternion kinematic equation for ω.

$$\dot{q} = \frac{1}{2} q \begin{bmatrix} 0 \\ \omega \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ \omega \end{bmatrix} = 2q^* \dot{q} \rightarrow \omega = 2H^T q^* \dot{q}$$

Here we see that ω is found by undoing the quaternion rotation q such that ω lies in a plane tangent to the identity quaternion. This idea is illustrated in the Figure below.

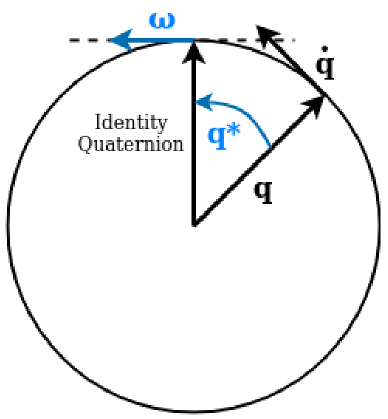

Figure 3: Illustration of the relationship between angular velocity and differential quaternions

References

[1] Yuval Tassa, Nicolas Mansard, Emo Todorov *Control-Limited Differential Dynamic Programming*. International Conference on Robotics and Automation 2014.

[2] Newton's Method in Optimization https://en.wikipedia.org/wiki/Newton%27s_method_in_optimization

[3] Brian Jackson, Zachary Manchester *Planning with Attitude* 2020