Time-Optimal Attitude Maneuvers

Enrique Babio

The goal of this class project is to investigate the application of optimal control theory in attitude dynamics. In particular, we will focus on looking up time-optimal trajectories. Since there is no interest in any particular system, we use the dynamics for simple symmetric rigid-body. The problem of interest is to study the optimal control when the input consists of three controls acting on three orthogonal axis that where some norm of the control is bounded. The expected solution should be of bang-bang type.

I. Introduction

Attitude maneuvers are a core problem to the control of aerospace vehicles. As such, understanding and optimizing the problem is a key aspect to the design of better performing vehicles. The performance index to be optimized will depend on the specific application and vehicle considered. For high performance aircraft and satellites, a possible performance index is the time used for a rest-to-rest maneuver. This has direct applications to satellite re-targeting or high-acceleration maneuvers for aircraft allowing for faster redirecting of the aircraft path.

A survey on time-optimal maneuvers is provided by Scrivener [1]. Optimal maneuvers for different configurations: number of angular velocity controls, rigid and flexible bodies, and rotating around the eigenaxis is provided.

Here, since we aim to achieve insight on maneuvers for both spacecraft and aircraft we will focus on rigid-body maneuvers under bounded controls. For large angle reorientations introducing quaternions will simplify the problem of computing the difference between attitudes, besides avoiding the singularities of euler angles, as proposed by Wie and Weiss [2]. In this case they introduce a regulator that follows the eigenaxis rotation between initial and final state which is the minimum action path. Chowdhry [3] solves the time-optimization of the attitude stabilization (i.e. drive angular velocity to 0) for different configurations where one angular rate control is not available for its application to super-maneuverable aircraft.

The problem is interesting from a theoretical point of view since it involves a number of different tools. The attitude equations of motion are highly nonlinear, and without any assumptions result in complex motion. Attitude itself calls for the use of quaternions or similar formulations since Euler angle strategies may become badly-defined and little intuitive. Finally, optimal control theory is necessary in order to achieve the necessary conditions for the time optimality of the control as stated by Longuski [4]. Besides, the final control is not readily intuitive: eigenaxis maneuvers provide the minimum length path, however aircraft tend to combine rolling and pitching motion.

In particular, we will follow the approach used by Bilimoria [5] and Fleming [6] but using a slightly different notation.

An axis-symmetric rigid-body will be considered where all three angular velocities can be controlled independently and where the control is norm by some norm. It would also be interesting to follow some of the approaches used by Byers [7] to look for real-time solving capabilities for the control so that it could be flyable.

II. Analysis

A. Problem formulation

As covered by Bilimoria the equations of motion for the rotation around principal axes of inertia are:

$$I_x\dot{\Omega}_x + (I_z - I_y)\Omega_z\Omega_y = \tau_x \tag{1}$$

$$I_y\dot{\Omega}_y + (I_x - I_z)\Omega_x\Omega_z = \tau_y \tag{2}$$

$$I_z\dot{\Omega}_z + (I_y - I_x)\Omega_y\Omega_x = \tau_z \tag{3}$$

For simplicity, we will assume that the vehicle is symmetric and therefore all three moments of inertia are identical and equal to I. Under those conditions the equations of motion decouple into:

$$I\dot{\Omega}_x = \tau_x \tag{4}$$

$$I\dot{\Omega}_y = \tau_y \tag{5}$$

$$I\dot{\Omega}_z = \tau_z \tag{6}$$

Or expressed in vector notation:

$$I\vec{\Omega} = \vec{\tau} \tag{7}$$

And the control is bounded by:

$$|\tau_x|^q + |\tau_y|^q + |\tau_z|^q \leq \tau_{max}^q \tag{8}$$

Where q is the norm of interest; this is a general way of bounding the control. The most interesting norms will be the 2-norm and the infinity-norm which correspond to a the maximum applied being independent of the direction and to three indepedently bounded controls by τ_{max}.

We can consider all three controls to be equally bounded by τ_{max} we can non-dimensionalize the equations of motion using $u_i = \tau_i/\tau_{max}$ and a characteristic time $t = \sqrt{I/\tau_{max}}$ so the characteristic angular velocity $\omega_i = \Omega_i\sqrt{I/\tau_{max}}$. The vector equation of motion now simplifies to its final form:

$$\vec{\omega} = \vec{u} \tag{9}$$

Beside the equations relating velocity to the control torques we need the equations of motion for the attitude quaternions. A description on how quaternions relate to euler angles and angular velocities can be found in Parwana and Kothari[8] and useful relations for quaternion derivatives are provided by Graf [9]. For more details on the quaternion convention or the derivation and partial differentiation we refer to the Appendix A. Our attitude quaternion is q and we introduce a angular velocity quaternion $\omega = [0\ \vec{\omega}]^T$

$$\dot{q} = 1/2\ q \circ \omega \tag{10}$$

Now we are ready to state our optimization problem. Our performance index is minimum final time, in other words:

$$\min J = t_f \tag{11}$$

Subject to the vector process equations9 and 10. Our control by the definition of the non-dimensionalization is subject to $|u_i| \leq 1$. And the initial and final states are fixed and defined by the maneuver to be optimized:

$$q(0) = [1 \quad 0 \quad 0 \quad 0]^T \tag{12}$$

$$q(t_f) = q_f \tag{13}$$

$$\vec{\omega}(0) = \vec{\omega}(t_f) = 0 \tag{14}$$

So the only free boundary is the final time t_f which is the performance index to be minimized.

B. Hamiltonian and costate equations

With the formulation for the equations of motion we define the Hamiltonian for the motion:

$$H(x, u, \lambda) = \frac{1}{2}\lambda_q^T(q \circ \omega) + \lambda_\omega^T \vec{u} \tag{15}$$

Where λ_q and λ_ω are the co-state vectors with a size of 4 elements and 3 elements respectively. We realize that since there is no explicit time dependence the Hamiltonian is constant throughout the motion.

In order to get the costate equations we get apply the Euler-Lagrange equations and find the appropiate derivatives. In order to simplify the resulting expressions Matrix calculus is used to come up with more compact expressions, see Appendix A for details on the derivation. The expression for the attitude costates is:

$$\dot{\lambda}_q = -\frac{\partial H}{\partial q} = \frac{1}{2}\lambda_q \circ \omega \tag{16}$$

The partial derivative for $\dot{\lambda}_\omega$ depends on omega which appears only in the quaternion product, but it cannot be obtained straight forward using the expressions for quaternion derivatives since $\dot{\lambda}_\omega$ is not a 4 element vector. However, if we extend it to a 4 element vector where the first element is forced to be 0 then we can find the partial derivative in terms of the imaginary part of the quaternion derivative.

$$\dot{\lambda}_\omega = -\frac{\partial H}{\partial \omega} = -\frac{1}{2}\mathrm{Im}(q' \circ \lambda_q) \tag{17}$$

The control logic is that of bang-bang controller which can be expected for a minimum-time type strategy. Because of this we have the possibility of having singular sub-arcs. Later we will have to study the possible consequences on the control.

We now look at the boundary conditions. In the problem statement we have stated that the final and initial states are fixed, also the initial time is fixed. Since the only free state is the final time t_f we look for the associated condition in the transversality condition:

$$H_f dt_f - \lambda^T dx_f + dg = H_f dt_f - \lambda^T dx_f + dt_f = (H_f + 1)dt_f \implies H_f + 1 = 0 \tag{18}$$

Since dt_f is the only free differential. Also, since the Hamiltonian is constant we have that:

$$H = -1 \tag{19}$$

C. Derivation of the optimal control

In order to find the optimal control we recognize that we have an inequality constraint in the control of the following form:

$$1 = |u_x|^q + |u_y|^q + |u_z|^q \tag{20}$$

This means that $H_u^* = 0$ cannot be directly satisfied. This function is not a switching function either so we have to apply Pontryagin's Minimum Principle since in its most basic form. We have to look for the optimal control such that:

$$H(x^*, u^*, \lambda^*) \leq H(X^*, u, \lambda^*) \tag{21}$$

Or equivalently for our Hamiltonian:

$$\lambda_\omega^{*T} \vec{u}^* \leq \lambda_\omega^{*T} \vec{u} \tag{22}$$

Since the expression above is linear in terms of $\vec{u}$ the minimum will necessarily lie on the inequality constraint of the norm of $\vec{u}$. We introduce the constraint in the expression above by solving for u_z in terms of u_x and u_y and the sign of

u_z which we don't know yet:

$$u_z = \text{sgn}(u_z)\,(1 - |u_x|^q - |u_y|^q)^{\frac{1}{q}} \tag{23}$$

Now we have to look for a minimum of:

$$\mathcal{L} = \lambda_{\omega_x} u_x + \lambda_{\omega_y} u_y + \text{sgn}(u_z)\lambda_{\omega_3}(1 - |u_x|^q - |u_y|^q)^{\frac{1}{q}} \tag{24}$$

In order to find the minimum we take partial derivatives with respect u_x and u_y. In the derivatives we identify the expression for $|u_z|$ so we substitute it back:

$$\frac{\partial \mathcal{L}}{\partial u_x} = 0 = \lambda_{\omega_x} - \text{sgn}(u_x u_z)\,\lambda_{\omega_z}\,(1 - |u_x|^q - |u_y|^q)^{\frac{1-q}{q}}\,|u_x|^{q-1} = \lambda_{\omega_x} - \text{sgn}(u_x u_z)\,\lambda_{\omega_z}\,|u_z|^{1-q}|u_x|^{q-1} \tag{25}$$

$$\frac{\partial \mathcal{L}}{\partial u_y} = 0 = \lambda_{\omega_y} - \text{sgn}(u_y u_z)\,\lambda_{\omega_z}\,(1 - |u_x|^q - |u_y|^q)^{\frac{1-q}{q}}\,|u_y|^{q-1} = \lambda_{\omega_y} - \text{sgn}(u_y u_z)\,\lambda_{\omega_z}\,|u_z|^{1-q}|u_y|^{q-1} \tag{26}$$

And on the expressions above we solve for u_x or u_y:

$$\lambda_{\omega_{x,y}} - \text{sgn}(u_{x,y} u_z)\,\lambda_{\omega_z}\,|u_z|^{1-q}|u_{x,y}|^{q-1} = 0 \tag{27}$$

$$\implies \text{sgn}(u_{x,y})|u_{x,y}|^{q-1} = \frac{\lambda_{\omega_{x,y}}}{\lambda_{\omega_z}}\text{sgn}(u_z)\,|u_z|^{q-1} \tag{28}$$

$$\implies \text{sgn}(u_{x,y})|u_{x,y}| = \text{sgn}\left(\frac{\lambda_{\omega_{x,y}}}{\lambda_{\omega_z}}\right)\left|\frac{\lambda_{\omega_{x,y}}}{\lambda_{\omega_z}}\right|^{\frac{1}{q-1}}\text{sgn}(u_z)\,|u_z| \tag{29}$$

$$= u_{x,y} = \text{sgn}\left(\frac{\lambda_{\omega_{x,y}}}{\lambda_{\omega_z}}\right)\left|\frac{\lambda_{\omega_{x,y}}}{\lambda_{\omega_z}}\right|^{\frac{1}{q-1}} u_z \tag{30}$$

And substituting these expressions in the expression we obtained in 23 we obtain:

$$|u_z| = \left(\frac{|u_z|^q}{|u_z|^q}\left|\frac{\lambda_{\omega_z}}{\lambda_{\omega_z}}\right|^{\frac{q}{q-1}}|u_z|^q - \left|\frac{\lambda_{\omega_x}}{\lambda_{\omega_z}}\right|^{\frac{q}{q-1}}|u_z|^q - \left|\frac{\lambda_{\omega_x}}{\lambda_{\omega_z}}\right|^{\frac{q}{q-1}}|u_z|^q\right)^{\frac{1}{q}} \tag{31}$$

$$\implies |\lambda_{\omega_z}|^{\frac{q}{q-1}} = \frac{1}{|u_z|^q}|\lambda_{\omega_z}|^{\frac{q}{q-1}} - |\lambda_{\omega_x}|^{\frac{q}{q-1}} - |\lambda_{\omega_x}|^{\frac{q}{q-1}} \tag{32}$$

$$\implies \frac{1}{|\lambda_{\omega_z}|^{\frac{q}{q-1}}}|\lambda_{\omega_z}|^{\frac{q}{q-1}} + |\lambda_{\omega_x}|^{\frac{q}{q-1}} + |\lambda_{\omega_x}|^{\frac{q}{q-1}} = \frac{1}{|u_z|^q} \tag{33}$$

$$\implies |u_z| = \frac{|\lambda_{\omega_z}|^{\frac{1}{q-1}}}{\left[|\lambda_{\omega_x}|^{\frac{q}{q-1}} + |\lambda_{\omega_y}|^{\frac{q}{q-1}} + |\lambda_{\omega_z}|^{\frac{q}{q-1}}\right]^{\frac{1}{q}}} \tag{34}$$

This defines the magnitude of each control input as function of the magnitude of the costates, but we still don't know how to find the sign of u_z. However to pick the right sign we just need to recall that our goal is to minimize the Hamiltonian, therefore we have to pick the sign with the opposite sign to λ_{ω_z}.

Once we have found the right u_z we can solve for $u_{x,y}$, using 30. The expression for all three controls became similar as it could be expected:

$$u_i = \frac{-\text{sgn}(\lambda_{\omega_i})|\lambda_{\omega_i}|^{\frac{1}{q-1}}}{\left[|\lambda_{\omega_x}|^{\frac{q}{q-1}} + |\lambda_{\omega_y}|^{\frac{q}{q-1}} + |\lambda_{\omega_z}|^{\frac{q}{q-1}}\right]^{\frac{1}{q}}} \tag{35}$$

Finally, we have to consider the possibility of singular control. In order to rule it out we recall the $H = -1$ condition obtained from the transversality condition. This allows us to guarantee that there will be non singular sub-arcs. Singular subarcs will only happen if $\lambda_\omega = 0$ for a finite interval, this implies that $\dot{\lambda}_\omega = 0$, but since q is an atittude quaternion $\dot{\lambda}_\omega = 0$ will only happen if $\lambda_q = 0$. This cannot happen at the same time that $\lambda_\omega = 0$ since that would make the Hamiltonian vanish, contradicting the transversality condition. Since there are no singular sub-arcs the controller above satisfies the necessary conditions for an optimal control.

The controller structure is fundamentally of a bang-bang nature. There is a component controlling the allocation of the control to the different actuators, but on top of this amplitude allocation the sign is controlled in the same fashion as a usual bang-bang control. Choosing a norm will determine the behavior of the control amplitude allocation, allowing for more independence between the controls that will lead to more and more complex control profiles and possible faster maneuver since the set of possible controls increases with the norm.

In the limit, where the norm goes to infinity the controls become completely independent with the form of switching functions. Each control component must satisfy $|u_i| \leq 1$. Then from the limit of the control in 35, or following the usual reasoning for switching functions minimizing the Hamiltonian we get:

$$u_i = -\text{sgn}(\lambda_{\omega_i}) \tag{36}$$

III. Numerical Solution

Once the TPBVP has been defined, the next step is to solve it. The discussion of the numerical solution will be split in three different topics, the first one will discuss the general strategy used for the problem and the statement of the boundary conditions, the second one will cover the initial guesses used for the solver, the third one considers the possible methods used for the solution and the advantages and disadvantages of each one them.

A. Statement of the constraints

The problem to be solved right now can be thought of a feasibility problem where we satisfy the constraints of the TPBVP. As such we will resort to numerical methods for its solution, but we should try to reduce the dimensionality of the problem and improve its conditioning as much as possible.

The first guess for the numerical solver will be obtained from the numerical integration of some initial conditions. Since we know the initial values for the states these will be fixed and will not be allowed to change, this helps to reduce

the dimensionality of the problem by 7 states. In the most basic form of the solution, we have reduced the problem to finding the initial values for the 7 co-states.

Also, we know that the Hamiltonian is constant throughout the motion and $H = -1$. The information that this constraint provides is merely in the terms of scaling of the costates. However, for our particular problem since the scaling of the co-states is not critical since we use them in two ways: for modulating amplitude (where the relative values is the used information) and for a switching function (where only the sign matters). This means that in order to avoid ill-conditioned numerical problems we can reformulate the constant Hamiltonian as:

$$|\lambda_{q_f}| = 1 \tag{37}$$

But recalling the costate equation for $\dot{\lambda}_q$ we can observe that it is just the rotation in time of λ_q under the angular velocity $\vec{\omega}$. This means that the magnitude of λ_q stays constant in time, and therefore we check the magnitude at the initial conditions where it is more convenient for our initial guess. Therefore the transversality condition is substituted by:

$$|\lambda_{q_0}| = 1 \tag{38}$$

Other than this condition, we will also check for state constraints at the final time. Since the final time is variable it is also another variable to be found in our constraints feasibility problem solution. We have a total of 8 constraints, 7 states and the just found condition on the module of λ_q, and 8 free variables, the initial states and the final time, so the problem is well defined.

B. Initial guesses and continuation strategy

After defining the constraints to be satisfied, the next step is how to come up with initial guesses for the solver. All solvers will work by solving an approximated problem, therefore we must come up with an initial guess close enough to the final solution in order for the solver to converge.

A possible strategy is to use a continuation on the norm. If the solution for a specific norm is known, its initial co-states can be used as a first guess for the problem of a similar norm. However, in any case a initial solution for a particular norm must be known. In the followin paragraph we come up with an initial guess for the 2-norm controller, so for solving higher norm controllers we will continuate the norm up to our desired final norm.

In the case of the 2-norm control the initial solution can be found analytically with the following thought process. This solution is not meant to be a formal solution since it is only used as an initial guess. The 2-norm control has no preferred direction of control since all controls are embedded in a sphere, therefore the optimal control will be that uses the shortest path. This shortest path will correspond to the rotation about the eigenaxis that connects the initial and final states. Since the initial quaternion is fixed to $q_0 = [1 \quad 0 \quad 0 \quad 0]^T$, the final quaternion provides the angle θ to be

rotated and the eigenaxis $\vec{n}$ information in the angle-axis expression of the quaternion:

$$q(t) = \left(\cos\frac{\theta}{2}\,,\ \sin\frac{\theta}{2}\,\vec{n}\right) \tag{39}$$

In order to consider the time evolution of the rotation, we can consider the problem as a 1-d rotation where the rest-to-rest maneuver will follow a triangular angular velocity profile with one switch of the control exactly at the half of the path. The time to the switch since the beginning of the maneuver can be found as:

$$\frac{1}{2}t_{sw} = \theta_f \qquad \Longrightarrow \qquad t_f = 2t_{sw} = 2\sqrt{\theta_f} \tag{40}$$

The angular velocity must be parallel to the eigenaxis and therefore so will be the $\vec{u}$, the switching must happen in all three components at the same time and then they must be equal to zero at $t_f/2$. We previously said that λ_q rotates in time with ω so it must follow also follow an eigenaxis rotation. We can express it as the composition of the rotation of q and the initial offset:

$$\lambda_q = q(t)\lambda_{q_0} \tag{41}$$

And then for λ_ω we get:

$$\dot{\lambda}_\omega = -\frac{1}{2}\mathrm{Im}(q(t)' \circ q(t) \circ \lambda_{q_0}) = -\frac{1}{2}\mathrm{Im}(\lambda_{q_0}) \tag{42}$$

So from the expression above states that $\lambda_\omega(t)$ is linear with time and knowing that λ_ω must be parallel to the rotation axis we can write the expression below. For that we use the fact that $\lambda_\omega(0)$ just oppose the rotation axis for the input to be aligned with it.

$$\lambda_\omega(0) = \frac{1}{2}\mathrm{Im}(\lambda_{q_0})(t_f/2) = -k\vec{n}$$

So from this expression we get that:

$$\lambda_{q_0} = -[0 \quad \vec{n}]^T \tag{43}$$

$$\lambda_{\omega_0} = -\frac{t_f}{4}\vec{n} \tag{44}$$

C. Solving methods

Different solvers for the feasibility problem were considered with their advantages and disadvantages. The options are summarized below with a discussion on their applicability to our problem of interest

1. Collocation methods

They are based on discretization of the motion into several intervals where the motion is integrated usually in an implicit way. These methods are characterized by their efficiency but lack the accuracy of other methods. MATLAB's bvp4c was the first option to be tested since it is meant for this kind of problems.

Its first problem comes from the collocation of swicthing and signum functions which cause the Jacobian to be singular. This was solved by approximating the signum function by a hyperbolic tangent so the continuation strategy would increase the norm and also the accuracy of the hyperbolic tangent approximation.

They are able to converge to valid solutions for low order norms but they were not able to converge with the desired accuracy as the continuation increased.

2. Multiple shooting method with Newton Iterations

A shooting method consists on running solving the feasibility problem by obtaining a Jacobian of how changes in the initial state about some guess change the final conditions. This is leads to a Newton iteration scheme where the errors and their Jacobian are used to compute the new initial conditions until the error is minimized.

The Jacobian approximation for changes in the initial conditions is only valid for an interval around the initial guess depending on the sensitivity of the problem. In order to minimize this sensitivity multiple shooting is used. It decreases the sensitivity by splitting the time interval into several intervals. On each of these intervals a shooting is performed. Since the number of states has increased, you have a initial state guess at the beginning of each shooting, extra constraints have to be added. These constraints ensure the continuity between shootings so that all different shootings are consistent. This decreases the sensitivity and increases the robustness of the method at the expense of more design constraints and increased computational cost. The preferred number of shooting intervals was set to 3.

In order to apply this method an integration scheme has to be used. Here MATLAB's ode 45 RungeKutta integrator was used with tight integration constraints to prevent numerical error from interfering in the computation of the Jacobian matrix. Specifically the relative tolerance was set to 10^{-11} and the absolute tolerance to 10^{-14}.

This method works as long as the Jacobian approximation works. It seemed to have convergence problems as the norm increased. It may have been solved by decreasing the step in the continuation.

3. Multiple shooting method with fmincon

This method is based on the previous method. The integration and constraint formulation to ensure continuity between different shootings is formulated in the same way. The difference lies in the solver used to find the new initial guess after each iteration.

We used MATLAB's fmincon, a convex optimizer that allows for constraints directly. Therefore instead of trying to minimize the constraint functions in the way of an augmented Lagrangian objective function we let fmincon handle the

constraints. This solver introduces methods for satisfying the constraint that go beyond the Newton's method we were using before. This leads to a greater radius of convergence.

The objective function to be minimized was time and the constraints were the ones already stated. However, we are not particularly interested in minimizing time, we already know that the solution will be time optimal if the constraints are satisfied. Optimization paramters are chosen according to this idea. The tolerance for the constraints is set to 10^{-4} and for the objective function optimility the tolerance is set to 1 since we are not really concerned about it.

This approach was the one that allowed for successful computation of the optimal control. The parameters for the integration are the same as those above. Results discussed below were found using this method.

IV. Results

Results were obtained for an arbitrary maneuver. The initial state is neutral attitude: roll, pitch and yaw all equal to zero. The final state are the following roll, pitch and yaw angles (in that order):

$$\phi = -20^\circ \quad \theta = 10^\circ \quad \psi = 90^\circ \tag{45}$$

We are particularly interested in two of the norms. The 2-norm and ∞-norm. The 2-norm shows the maneuver for the shortest path, can be computed in real time using the approximation provided in section IIIc, and works as a reference solution. The ∞-norm uses three independent bounds, resembles more closely the limits of an attitude control system, and offers the possibility for unexplored more optimal solutions than those found by the 2-norm.

We start by discussing in detail the solution for the 2-norm solution. Figure 1 shows the main characteristics of the maneuver. Time-history is provided for the main parameters of interest: the states, attitude (in Euler angles) and angular velocity, the applied torques and the co-states that the determine the input. The costates of the attitude are not represented since they do not offer any insight. The initial co-states for the maneuver are:

$$\lambda_{q_0} = [-0.0006 \quad 0.2474 \quad 0.0833 \quad -0.9532]^T \qquad \lambda_{\omega_0} = [0.1583 \quad 0.0533 \quad -.6093] \tag{46}$$

The total used for the maneuver is 2.5596 and the amplitude of total rotated angle is 1.6378 radians. A closer look to the graphics reveal that our initial guess for the solution is indeed correct. The λ_ω behave linearly with time and being equal to 0 at exactly the half of the motion where the inputs reverse. The amplitude of these is set so that they are aligned with the eigenaxis of the rotation.

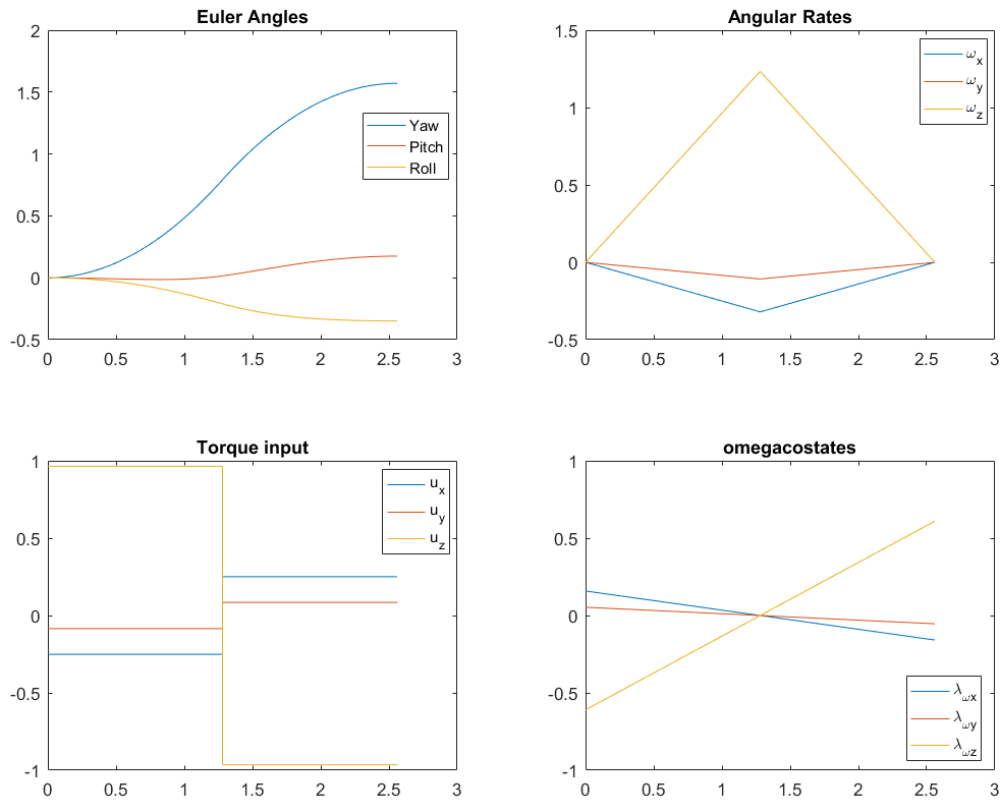

Fig. 1 Solution, inputs and co-states for the switching function for the 2-norm optimal maneuver.

We now proceed to discuss the convergence process to the infinity norm. This was done by increasing the norm of the controller a about a 10 percent each iteration up to a norm of 20. Figure 2 shows some selected norms in the continuation process that show the evolution of the controller. As the norm of the controls increase the coupling between axes becomes more and more relaxed, for the ∞-norm they should be three independent controls bounds so the all three controls should lie on the ± 1 bound.

Once the 20-norm controller is found, it is used as the starting solution for the ∞-norm controller implemented as in equation 36. The three controls become completely indepdent, the characteristics for the new control are plotted in figure 3 in a similar fashion to those of the 2-norm control. The initial value for the co-states is:

$$\lambda_{q_0} = [-0.0405 \quad 0.0809 \quad -0.3818 \quad -0.9179]^T \qquad \lambda_{\omega_0} = [0.0594 \quad -0.0751 \quad -.5617] \tag{47}$$

The maneuver time has decreased to 2.3540. A decrease above 8 percent in the maneuver time. The controls have the form of independent switching functions as it could be expected. This means that the angular velocity is also composed of linear intervals. These results are coherent with those found by Bilimoria[5]. The shape for the co-states

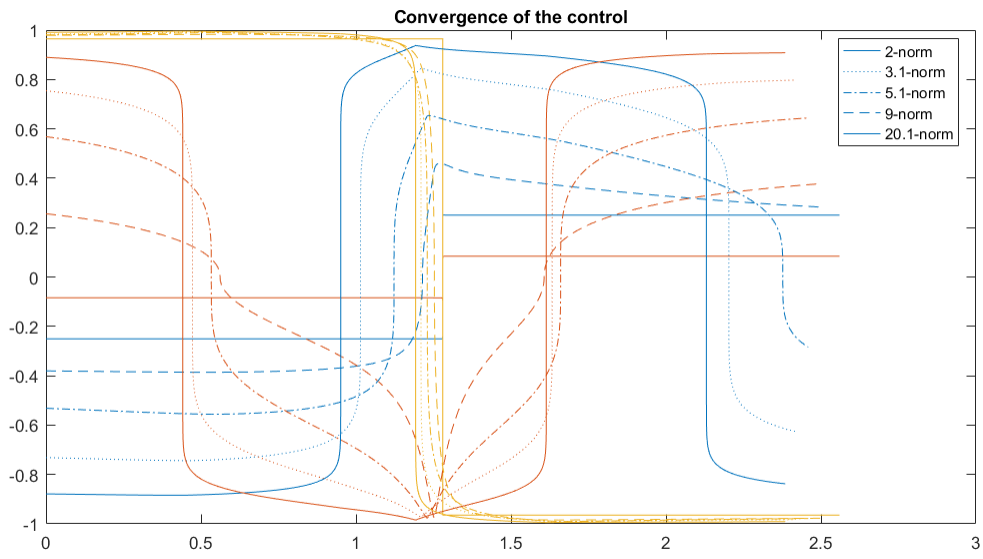

Fig. 2 Three axes controls for different norms ranging from to 2 to 20. As the norm increases the three controls become more and more decoupled. Reproduced from Bilimoria [5].

of the switching function has become more complex.

It is interesting to note that the maneuver time has decreased with respect to the eigenaxis maneuver even though we are following a longer path. This is because the controller is reorienting the body so that a torque greater than any individual limit is applied in the desired rotation of motion from an inertial perspective as point out by Bilimoria[5].

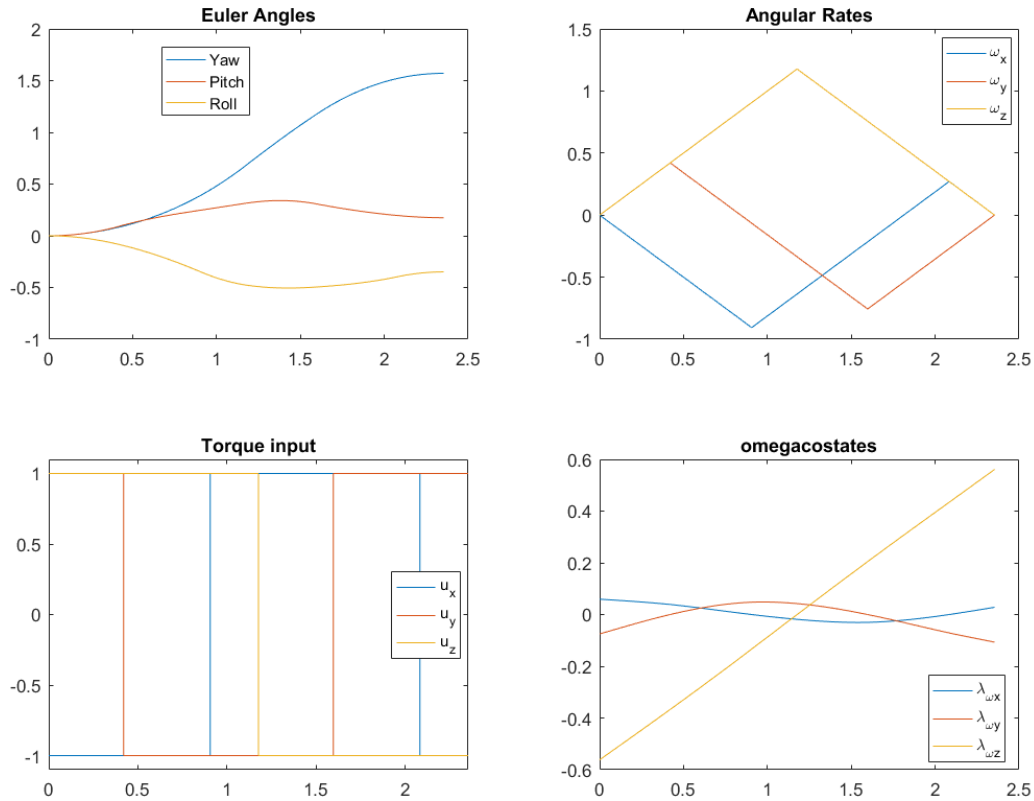

Fig. 3 Solution, inputs and costates for the switching function for the ∞-norm optimal maneuver.

V. Conclusions

The time-optimal maneuver has been studied for the two most relevant cases the 2-norm and the ∞-norm bounded controls. The conclusions for this work can be grouped in three main categories: optimality of the control, those regarding the solution, and finally the applicability and insight obtained from the results.

About the control we can state that:

1) First order necessary conditions for an arbitrary norm control have been found using Pontryagin's Minimum Principle. The resulting control is has the form of a modified switching function.
2) For the ∞-norm controller, the resulting controls are independent switching functions.
3) The existence of singular sub-arcs for the optimal solution has been ruled out.

When it comes to solution of the system, the following can be concluded:

1) The condition arising from the transversality condition can be rewritten to improve numerical condition in this particular problem.
2) An analytical solution was found for the 2-norm controller or eigenaxis maneuver.

3) A continuation strategy based on the norm can be used to find any specific norm. The infinity norm was found following this procedure starting in the 2-norm.
4) Multiple shooting methods can be used for increased robustness and accuracy in the solution at the expense of increased computation cost.

Finally, about the method used and the results obtained we can state

1) Solutions of a norm higher than 2 improve the time used for a rest-to-rest attitude maneuver. In our particular example an improvement of 8% was found.
2) An arbitrary maneuver was tested. It has no special characteristics so this method should be applicable to any maneuver.
3) The computation expense of the ∞-norm maneuver prevents it from its application to attitude control. The computation time used to find the initial co-states is not reasonable to be performed in real time. This could be done for 2-norm control since an analytical solution is known.

This last finding is of critical importance to the applicability of this kind of maneuvers in real systems. Suboptimal approaches could be used to allow fast attitude maneuvers such as the 2-norm control. Other authors such as Byers[7] have worked on this paper with intention of leveraging the known structure of the control to simplify the computations. These would allow the implementation of this kind of optimal maneuvers in the AOCS/FCS systems of spacecraft and aircraft. If the topic was to be researched more deeply the next steps would be in this direction.

References

[1] Scrivener, S. L., and Thompson, R. C., "Survey of time-optimal attitude maneuvers," *Journal of Guidance, Control, and Dynamics*, Vol. 17, No. 2, 1994, pp. 225–233.

[2] Wie, B., Weiss, H., and Arapostathis, A., "A quaternion feedback regulator for spacecraft eigenaxis rotations," *Guidance, Navigation and Control Conference*, 1989, p. 4128.

[3] Chowdhry, R., and Cliff, E., "Optimal rigid body motions, part 2: Minimum time solutions," *Journal of optimization theory and applications*, Vol. 70, No. 2, 1991, pp. 255–276.

[4] Longuski, J. M., Guzmán, J. J., and Prussing, J. E., *Optimal control with aerospace applications*, Springer, 2014.

[5] Bilimoria, K. D., and Wie, B., "Time-optimal three-axis reorientation of a rigid spacecraft," *Journal of Guidance, Control, and Dynamics*, Vol. 16, No. 3, 1993, pp. 446–452.

[6] Fleming, A., Sekhavat, P., and Ross, I. M., "Minimum-time reorientation of a rigid body," *Journal of guidance, control, and dynamics*, Vol. 33, No. 1, 2010, p. 160.

[7] Byers, R., and Vadali, S., "Quasi-closed-form solution to the time-optimal rigid spacecraft reorientation problem," *Journal of Guidance, Control, and Dynamics*, Vol. 16, No. 3, 1993, pp. 453–461.

[8] Parwana, H., and Kothari, M., “Quaternions and Attitude Representation,” *arXiv preprint arXiv:1708.08680*, 2017.

[9] Graf, B., “Quaternions and dynamics,” *arXiv preprint arXiv:0811.2889*, 2008.

Appendices

A. Mathematical notation

A. Quaternion and Vector notation and derivatives

Here we use the so called Hamilton quaternion convention. A unitary quaternion q with components q_0, q_x, q_y, q_z is defined as:

$$q = \begin{bmatrix} q_0 & q_x & q_y & q_z \end{bmatrix}^T \tag{48}$$

Where q_0 represents the real part of the quaternion associated with the rotation angle and q_x, q_y, q_z represent the imaginary part associated with the rotation axis.

The quaternion product of the quaternions q and p is defined as usual and denoted by $q \circ p$. The quaternion conjugate is denoted by q'. We highlight the quaternion product as a matrix product as shown in Parwana [8] and its relations with the conjugate quaternion:

$$q \circ p = (q' \circ p')' = Q(q)p = \bar{Q}(p)q = \begin{bmatrix} q_0 & -q_x & -q_y & -q_z \\ q_x & q_0 & -q_z & q_y \\ q_y & q_z & q_0 & -q_x \\ q_z & -q_y & q_x & q_0 \end{bmatrix} \begin{bmatrix} p_0 \\ p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} p_0 & -p_x & -p_y & -p_z \\ p_x & p_0 & p_z & -p_y \\ p_y & -p_z & p_0 & p_x \\ p_z & p_y & -p_x & p_0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_x \\ q_y \\ q_z \end{bmatrix} \tag{49}$$

The time derivative of q as a function of the angular velocity in the rotating body frame $\vec{\omega}$ is

$$\dot{q} = 1/2\ q \circ [0 \quad \vec{\omega}]^T \tag{50}$$

In the Hamiltonian we get an expression of the form $\lambda^T(q \circ p)$, where λ is a 4 element column vector. This expression is scalar. Expressing it as a matrix product we can find its derivative with respect to q and p, using matrix calculus to obtain more compact expressions for the costates ODEs, we will use a column gradient convention. We start by finding the derivative of the quaternion product:

$$\frac{\partial}{\partial p}\lambda^T(q \circ p) = \frac{\partial}{\partial p}\lambda^T Q(q)p = (\lambda^T Q(q))^T = Q(q)^T\lambda = Q(q')\lambda = q' \circ \lambda \tag{51}$$

$$\frac{\partial}{\partial q}\lambda^T(q \circ p) = \frac{\partial}{\partial p}\lambda^T \bar{Q}(p)q = (\lambda^T \bar{Q}(p))^T = \bar{Q}(p')\lambda = \lambda \circ p' \tag{52}$$

Where the derivative of the quaternion is obtained directly from Matrix calculus and the second one is obtained by inspection of the resulting matrix. We now obtain the expressions for the partial of the Hamiltonian taylored to the

dynamics obtained for our problem:

$$H(x, u, \lambda) = \frac{1}{2}\lambda_q^T (q \circ \omega) + \lambda_\omega^T \vec{u} \tag{53}$$

$$-\frac{\partial H}{\partial q} = -\frac{1}{2}\frac{\partial}{\partial q}\lambda_q^T (q \circ \omega) = -\frac{1}{2}\lambda_q \circ \omega' = \frac{1}{2}\lambda_q \circ \omega \tag{54}$$

$$-\frac{\partial H}{\partial \omega} = -\frac{1}{2}\frac{\partial}{\partial \omega}\lambda_q^T (q \circ \omega) = -\frac{1}{2} q' \circ \lambda_q \tag{55}$$

B. Derivative of a norm

The norm used in this report is defined in terms of an absolute value for the greatest generality. We prove here the derivative:

$$\frac{d}{dx}|x|^q = \begin{cases} \frac{d}{dx}x^q = q\ x^{q-1} = q\ |x|^{q-1} & \text{for } x > 0 \\ \frac{d}{dx}(-x)^q = q\ (-x)^{q-1} = q\ |x|^{q-1} & \text{for } x < 0 \end{cases} \tag{56}$$

Or in a more compact way using the signum function:

$$\frac{d}{dx}|x|^q = \text{sgn}(x)\ q\ |x|^{q-1} \tag{57}$$

We will also use extensively the following property of the signum function:

$$\text{sgn}(x\ y) = \text{sgn}(x)\ \text{sgn}(y) \tag{58}$$

B. MATLAB code

This appendix includes all relevant code developed in MATLAB. For the sake of simplicity only the code regarding Multiple Shooting with fmincon is included. All other numerical methods can be found on GitHub: https://github.com/ebabio/attitudeTimeOptimalManeuvers.git (including this very same report)

A. attitudeManeuver

Main script, checks if in continuation or not, solves the problem and displays results.

```
%% 3d Attitude Test

%% Script mode

reuse = 1
%% Setup workspace
addpath(genpath(pwd))
if(reuse == 1 && exist('orbitFinal', 'var'))
    clearvars -except orbitFinal reuse continuation parameters fileName
else
    clearvars -except continuation fileName
end
clc

%% Define boundary conditions

% Initial / final conditions
roll = deg2rad(-20);
pitch  = deg2rad(10);
yaw = deg2rad(90);

q0 = [1 0 0 0]';
qf = eul2quat([yaw pitch roll])';

omegaBCs = zeros(3,1);
```

```
x0BCs = [q0; omegaBCs];
xfBCs = [qf; omegaBCs];

% Control parameters
if(exist('continuation', 'var') && exist('parameters', 'var') && continuation)
    continuationCoefficient = 1.08;
    parameters.lNorm = floor(10 *continuationCoefficient * parameters.lNorm) /
        10;  % round to first decimal
    parameters.k = continuationCoefficient * parameters.k;
else
    parameters.lNorm = 2;
    parameters.k = 5e1;
end
display(['Looking for optimal control with norm ' num2str(parameters.lNorm) ':
    '])

%% Setup

% 1. Trajectory Integration routine
control = @(t,x) omegaControl(t, x, parameters);
shootingOde = @(t, x) (attitudeODEs(t, x, control));

orbit = Orbit(zeros(14,1), shootingOde);
orbit.odeOptions = odeset('RelTol',1e-11,'AbsTol',1e-14);

% 2. Boundary Conditions
shootingBCs = @(x0, xf) (attitudeBCs(x0BCs, x0, xfBCs, xf, shootingOde));

% 3. Optimizer setup
fObjective = @(x) x(end);
```

```
% 3.a: fmincon
struct.nIntervals = 3;
struct.setupOrbit = orbit;
gConstraints = @(x) ( nonlconWrapper([x0BCs; x], shootingBCs, struct) );
fminconoptions = optimoptions('fmincon', 'Display', 'iter', '
    FiniteDifferenceStepSize', 1e-4, ...
     'ConstraintTolerance', 1e-4, 'OptimalityTolerance', 1, ...
     'MaxIterations', 1e3, 'MaxFunctionEvaluations', 1e6, ...
     'UseParallel', true);
% 3.b: fminunc
fLagrangian = @(x) (sqrt(gConstraints(x)'*gConstraints(x)));
fminuncoptions = optimoptions('fminunc', 'Display', 'iter', '
    FiniteDifferenceStepSize', 1e-4, 'UseParallel', true, 'OptimalityTolerance
    ',1e-4);

%% Initial solution

if(~exist('orbitFinal', 'var'))
    % First guess
    thetaf = 2*acos(qf(1));
    n = qf(2:4)/norm(qf(2:4));
    tf = 2*sqrt(thetaf); % * (.98 + 0.02*randn(1));

    lambda_q = -[0; n];% + .2*randn(4,1);
    lambda_omega = -tf/4*n; % + .2*randn(3,1);
    lambda0 = [lambda_q; lambda_omega];

    % First actual integration
    orbit.x0 = [ x0BCs; lambda0];
    orbit.integrateX0(1*tf);
else
    % Reuse previous solution as final guess
```

```
    orbit.x0 = orbitFinal.x0; % + 0.01*randn(size(orbit.x0));
    orbit.integrateX0(1*orbitFinal.tf);
end

% Setup independent shootings
dim = numel(orbit.x0);
tfVector = orbit.tf * (1:struct.nIntervals)./struct.nIntervals;
for i=1:struct.nIntervals
    if(i==1)
        x0Opt = orbit.x0(1:dim);
    else
        index0 = (i-1)*dim;
        x0Opt(index0+1:index0+dim) = deval(orbit.odeSol, tfVector(i-1));
    end
end
x0Opt(end+1) = orbit.tf;



%% Solve
% 1. Test for initial guess (initial point BCs should be satisfied)
f0 = shootingBCs(orbit.x0, orbit.xf);
f0(8:end)
orbit.x0(8:14)

% 2. Iterate shooting
tic
[xFinal, tf, exitflaf, output] = fmincon(fObjective, x0Opt(dim/2+1:end), [],
    [], [], [], [], [], gConstraints, fminconoptions);
%[xf, tf, exitflaf, output] = fminunc(fLagrangian, x0Opt(dim/2+1:end),
    fminuncoptions);
toc
orbitFinal = copy(orbit);
```

```
orbitFinal.x0 = [x0BCs; xFinal(1:dim/2)];
orbitFinal.integrateX0(tf);

% 3. Test for final guess (all BCs should be satisfied)
f = shootingBCs(orbitFinal.x0, orbitFinal.xf)
orbitFinal.x0(8:14)

%% Evaluate the initial solution at all times

maneuver0.t = orbit.t;
maneuver0.x = orbit.x;

maneuver0.q = maneuver0.x(1:4,:);
maneuver0.eul = quat2eul(maneuver0.q')';
maneuver0.omega = maneuver0.x(5:7,:);
maneuver0.lambda_q =  maneuver0.x(8:11,:);
maneuver0.lambda_omega =  maneuver0.x(12:14,:);

maneuver0.u = zeros(3, numel(maneuver0.t));
for i=1:numel(maneuver0.t)
    maneuver0.u(:,i) = control(maneuver0.t, maneuver0.x(:,i));
end

% plot Data
figure(1)
clf reset

subplot(2,2,1)
plot(maneuver0.t, maneuver0.eul)
title('Euler Angles')
legend('Yaw', 'Pitch', 'Roll', 'Location', 'best')
subplot(2,2,2)
```

```
plot(maneuver0.t, maneuver0.omega)
title('Angular Rates')
legend('\omega_x', '\omega_y', '\omega_z', 'Location', 'best')
subplot(2,2,3)
plot(maneuver0.t, maneuver0.u)
title('Torque input')
legend('u_x', 'u_y', 'u_z')
subplot(2,2,4)
plot(maneuver0.t, maneuver0.lambda_omega)
title('omegacostates')
legend('\lambda_\omega_x', '\lambda_\omega_y', '\lambda_\omega_z', 'Location',
    'best')


%% Evaluate the final solution at all times
maneuver.t = orbitFinal.t;
maneuver.x = orbitFinal.x;

maneuver.q = maneuver.x(1:4,:);
maneuver.eul = quat2eul(maneuver.q')';
maneuver.omega = maneuver.x(5:7,:);
maneuver.lambda_q =  maneuver.x(8:11,:);
maneuver.lambda_omega =  maneuver.x(12:14,:);

maneuver.u = zeros(3, numel(maneuver.t));
for i=1:numel(maneuver.t)
    maneuver.u(:,i) = control(maneuver.t, maneuver.x(:,i));
end

%plot data
figure(2)
clf reset
```

```
subplot(2,2,1)
plot(maneuver.t, maneuver.eul)
title('Euler Angles')
legend('Yaw', 'Pitch', 'Roll', 'Location', 'best')
subplot(2,2,2)
plot(maneuver.t, maneuver.omega)
title('Angular Rates')
legend('\omega_x', '\omega_y', '\omega_z', 'Location', 'best')
subplot(2,2,3)
plot(maneuver.t, maneuver.u)
title('Torque input')
legend('u_x', 'u_y', 'u_z', 'Location', 'best')
subplot(2,2,4)
plot(maneuver.t, maneuver.lambda_omega)
title('omegacostates')
legend('\lambda_\omega_x', '\lambda_\omega_y', '\lambda_\omega_z', 'Location',
    'best')
```

B. attitudeManeuverLoop

Initializes and runs the continuation scheme up to norm 20 while logging all intermediate information. ∞-norm must be solved manually.

```
%% Run attitudeManeuver until hitting stopping condition

%% Setup workspace
clear all

fileName = ['results/' datestr(datetime, 'mmdd-HHMM' )];
diary ([fileName ' Diary.txt'])



%% Iterate
continuation = 1;
parameters.lNorm = 2;

while(parameters.lNorm < 20)
    attitudeManeuver
    save([fileName 'lNorm' num2str(parameters.lNorm) ' Workspace'])
    pause(.1);
end

display('continuation successfully finished')
diary off
```

C. attitudeODEs

ODEs for the motion other than control

```
function dxdt = attitudeODEs(t, x, control)

% Reassign variables
x = x';          %since quaternion and row vectors in matlab
q = x(1:4);
omega = x(5:7);
lambda_q = x(8:11);
% lambda_omega = x(12:14);

% Synthethize control
u = control(t,x);

% Compute derivatives
% a. Quaternion multiplication way
qDot = 1/2 * quatmultiply(q, [0 omega]); % check derivation
omegaDot = u;
lambdaDot_q = 1/2 * quatmultiply(lambda_q, [0 omega]); % check derivation
lambdaDot_omega = -1/2 * quatmultiply(quatconj(q), lambda_q); % check
    derivation

dxdt = [qDot omegaDot lambdaDot_q lambdaDot_omega(2:4)]';
return
```

D. attitudeBCs

BCs for the motion other than control

```
function [BCs, pathCs] = attitudeBCs(x0BCs, x0, xfBCs, xf, xDotFun, tf)
```

```
BCs = zeros(15,1);
pathCs = zeros(2,1);

if(nargin == 6)
    collocation = 1;
else
    collocation = 0;
end

%% Allow for xDotFun collocation parameters
if(collocation)
    xDotWrapper = @(tau, x) (xDotFun(tau, x, tf));
else
    xDotWrapper = @(t,x) (xDotFun(t, x));
end

%% Initial state constraints
% Reassign variables
q0 = x0(1:4);
omega0 = x0(5:7);
lambda_q0 = x0(8:12);
lambda_omega0 = x0(13:14);

% Hamiltonian
xDot = xDotWrapper([],x0);
H0 = [lambda_q0;lambda_omega0]' * xDot(1:7);

BCs(1:7)  = [q0-x0BCs(1:4); omega0-x0BCs(5:7)];
pathCs(1) = H0+1;
BCs(15) = norm(lambda_q0)-1;
```

```
%% Final State Constraints
qf = xf(1:4);
omegaf = xf(5:7);
lambda_qf = xf(8:12);
lambda_omegaf = xf(13:14);

% Hamiltonian
xDot = xDotWrapper([],xf);
Hf = [lambda_qf;lambda_omegaf]' * xDot(1:7);

BCs(8:14)  = [qf-xfBCs(1:4); omegaf-xfBCs(5:7)];
pathCs(2) = Hf+1;
% BCs(15) = pathCs(2);
return
```

E. omegaControl

Implementation of the control indepedently of the equation of motion. If the required norm is greater it will directly compute the infinity norm controller.

```
function u = omegaControl(t, x, parameters)

lNorm = parameters.lNorm;
k = parameters.k;

% Reassign variables
q = x(1:4);
omega = x(5:7);
lambda_q = x(8:11);
lambda_omega = x(12:14);
```

```
% Modified control
lambda_omega_exp = @(i) abs(lambda_omega(i))^(1/(lNorm-1));
den = (lambda_omega_exp(1)^lNorm + lambda_omega_exp(2)^lNorm +
    lambda_omega_exp(3)^lNorm)^(1/lNorm);

fun = @(x) sign(x);

u(1) = -fun(k*lambda_omega(1)) * lambda_omega_exp(1) / den;
u(2) = -fun(k*lambda_omega(2)) * lambda_omega_exp(2) / den;
u(3) = -fun(k*lambda_omega(3)) * lambda_omega_exp(3) / den;
```

F. Orbit

Class for integration of the motion in a convenient manner.

```
classdef Orbit < matlab.mixin.Copyable
    % Important note on handle classes: handle objects behave as handles/
        pointers
    % To create a different object with the same values use COPY method

    properties
        odefun

        x0

        x
        t

        xf
        tf

        odeSol
        odeOptions
        odeIntegrator
    end

    methods
        function obj = Orbit(x0, odefun)
            if(nargin ==2)
                obj.odefun = odefun;
                obj.x0 = x0;
                obj.x = obj.x0;
                obj.t = 0;
            elseif(nargin ==0)
```

```
        %empty constructor
    else
        warning('number of inputs not recognized');
    end
end

function addDeltaX0(obj, deltaX0)
    if(size(deltaX0,1)>numel(obj.x0))   %update vector may contain
       final time
        obj.x0 = obj.x0 + deltaX0(1:numel(obj.x0));
        obj.tf = obj.tf + deltaX0(numel(obj.x0)+1);
    else
        obj.x0 = obj.x0 + deltaX0;
    end
end

function obj = integrateX0(obj, tEnd)
    if(nargin < 2)
        tEnd = obj.tf;
    end

    obj.odeSol = ode45(obj.odefun, [0 tEnd], obj.x0, obj.odeOptions);
    obj.t = obj.odeSol.x;
    obj.x = obj.odeSol.y;

    obj.tf = obj.t(end);
    obj.xf = obj.x(:,end);
end

function phi = STM(obj, tEnd, dimensions)
    % STM found by variation of the initial conditions
    n = numel(obj.x0);
```

```
if(nargin < 2)
    tEnd = obj.tf;
end
if(nargin < 3)
    activeDimensions = ones(1,n);
    withTime = 0;
else
    % transform from indexes to an indicator vector
    withTime = find(dimensions==n+1);
    activeDimensions(dimensions) = ones(1,numel(dimensions));
    activeDimensions = activeDimensions(1:n);
end

phi = eye(n);
delta = 1e-3;

% Reference solution
obj.odeSol = ode45(obj.odefun, [0 tEnd], obj.x0, obj.odeOptions);
obj.t = obj.odeSol.x;
obj.x = obj.odeSol.y;

obj.tf = obj.t(end);
obj.xf = obj.x(:,end);

% Compute STM
for i=find(activeDimensions==1)
    perturbation = delta * circshift([1; zeros(n-1,1)], i-1);
    xDelta0 = obj.x0 + perturbation;
    [~,xDeltaf] = ode45(obj.odefun, [0 tEnd], xDelta0, obj.
        odeOptions);
    phi(:,i) = (xDeltaf(end,:)' - obj.xf)/delta;
```

```
            end
            for i=find(activeDimensions==0)
                phi(:,i) = zeros(n,1);
            end

            if(withTime)
                phi(:,end+1) = obj.odefun([], obj.xf);
            end
        end
    end

end
```

G. nonlconWrapper

Wrapper function for the nonlinear constraints. It wraps the integration of the motion and the constraints as MATLAB's optimization toolbox expects. It also handles the number of intervals for multiple shooting method and defects between intervals.

```
function [g, h] = nonlconWrapper(x, shootingBCs, parameters)

%% Assign parameters
nIntervals = parameters.nIntervals;
setupOrbit = parameters.setupOrbit;

dim = (numel(x)-1)/nIntervals;
%% Evaluate orbits

tf = x(end);

iterOrbit(nIntervals) = Orbit();
```

```
tfVector = tf * (1:nIntervals)./nIntervals;
for i=1:nIntervals
    iterOrbit(i) = copy(setupOrbit);
    if(i==1)
        iterOrbit(i).x0 = x(1:dim);
        iterOrbit(i).tf = tfVector(i);
    else
        index0 = (i-1)*dim;
        iterOrbit(i).x0 = x(index0+1:index0+dim);
        iterOrbit(i).tf = tfVector(i) - tfVector(i-1);
    end
    iterOrbit(i).integrateX0();
end

%% Evaluate constraints

% Trajectory constraints
constraintErr = shootingBCs(iterOrbit(1).x0, iterOrbit(end).xf);

% Defects
defectErr = zeros(dim * (nIntervals-1),1);
for i  = 2:nIntervals
    index0 = dim*(i-2);
    defectErr(index0+1:index0+dim) = iterOrbit(i).x0 - iterOrbit(i-1).xf;
end

%% Wrap up for returning
g = 0;
h = [constraintErr; defectErr];
```