Successive Convexification for Fuel-Optimal Rocket Landing

Addy (Jin Hyun) Park

February 24, 2025

1 Introduction

Motivated by SpaceX's recent mid-air "catch" of the Falcon 9 booster, I implemented a successive convexification algorithm based on the paper *Successive Convexification for Fuel-Optimal Powered Landing with Aerodynamic Drag and Non-Convex Constraints*. This paper seeks to solve the problem of powered descent guidance by formulating the nonlinear programming problem as a series of second-order cone programs (SOCP). This technique is generally referred to as successive convex programming (SCP). In the problem that we consider, there are two sources of nonconvexities in the original problem: (1) nonlinear dynamics and (2) a lower bound on the thrust magnitude. The advantage of approximating the nonlinear program as a series of SOCP's is that it allows us to leverage existing efficient Interior Point Method (IPM) solvers, making real-time trajectory generation possible.

This approach can broadly be divided into six steps: (1) convexify any nonconvex constraints, (2) add a trust region to avoid artificial unboundedness, (3) add virtual control terms to avoid artificial infeasibility, (4) add penalty terms that penalize usage of virtual control, (5) apply a discretization scheme of the implementer's choice to obtain a finite-dimensional convex optimization problem, and (5) choose an "update rule " for the trust region radius.

2 Problem Formulation

2.1 Dynamics

The rocket is modeled as a point mass subject to 3-DOF translational motion. This is a reasonable approximation when the rotational and translation states are weakly coupled. That is, the rocket's attitude can be changed rapidly without significantly perturbing the vehicle's translational position. This approximation is frequently used for quadrotor dynamics and is also reasonable for rockets with high attitude control authority. Moreover, the paper includes an aerodynamic drag term, which adds nonlinearity to the dynamics. The 3-DOF dynamics is as follows:

$$\frac{d\mathbf{x}}{dt} = f(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \dot{m}(t) \\ \dot{\mathbf{r}}(t) \\ \dot{\mathbf{v}}(t) \end{bmatrix} = \begin{bmatrix} -\alpha \left\| \mathbf{T}(t) \right\|_2 \\ \mathbf{v}(t) - \dot{m}_{\text{bp}} \\ \frac{1}{m(t)}[\mathbf{T}(t) + \mathbf{D}(t)] + \mathbf{g} \end{bmatrix} \tag{1}$$

where we have included the mass-depletion dynamics $\dot{m}(t)$, $\dot{m}_{\text{bp}}$ is a constant back pressure loss, $\mathbf{D}(t)$ is the aerodynamic drag, and the control input $\mathbf{u}(t)$ is equal to the thrust vector $\mathbf{T}(t)$.

The drag force is modeled as quadratic drag:

$$\mathbf{D}(t) = -\frac{1}{2}\rho C_D S_D \left\|\mathbf{v}(t)\right\|_2 \mathbf{v}(t)$$

where ρ is the air density, C_D is the coefficient of drag, and S_D is the surface area of the rocket. These parameters are assumed to stay constant during the duration of the flight. Furthermore, the model assumes that the rocket's thrust vector is always aligned with the rocket's longitudinal axis – there is no thrust vectoring. Therefore, the thrust vector can serve as a surrogate for the rocket's attitude.

2.2 Control and State Constraints

2.2.1 Thrust Magnitude Constraint

The thrust magnitude of a rocket engine typically has both an upper bound and a lower bound. This renders the feasible region nonconvex as shown in the Figure 1 below:

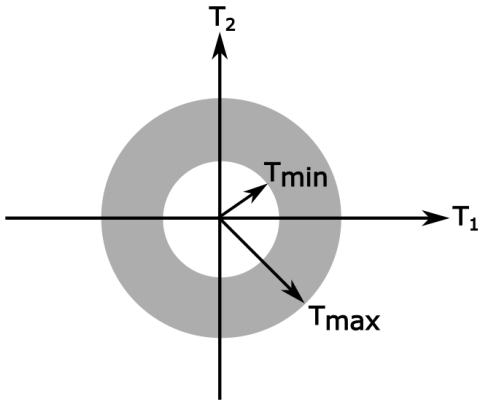

Figure 1: Thrust magnitude constraint.

This constraint can be expressed as the following:

$$T_{\min} \leq \left\|\mathbf{T}(t)\right\|_2 \leq T_{\max} \tag{2}$$

2.2.2 Thrust Pointing Constraint

Rockets also often have a constraint on their tilt angle away from an upright orientation. That is to say, the rocket's thrust vector is restricted to point within an angle of $\theta_{\max}$ from the vertical direction (represented by the unit vector $\hat{n}_u$). As you can see in Figure 2, this constraint is convex for $-90^\circ \leq \theta_{\max} \leq 90^\circ$ and is nonconvex otherwise. This constraint can be expressed as the following:

$$\hat{\mathbf{n}}_u^T \mathbf{T}(t) \geq \left\|\mathbf{T}(t)\right\|_2 \cos(\theta_{\max}) \tag{3}$$

The paper assumes $-90^\circ \leq \theta_{\max} \leq 90^\circ$ so this constraint is not a source of nonconvexity.

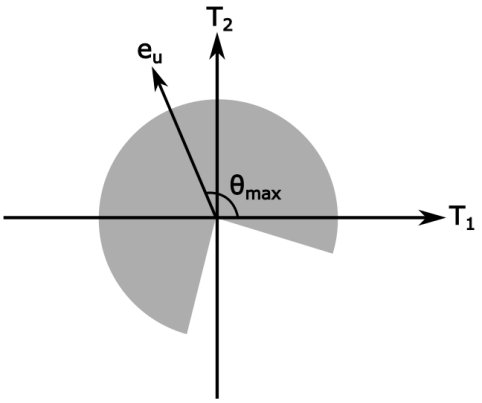

Figure 2: Thrust pointing constraint.

2.2.3 Glideslope Constraint

A glideslope constraint limits the rocket's trajectory to lie within a cone of angle γ_{gs} relative to the unit vertical vector at the landing site, $\hat{e}_u$, as depicted in Figure 3. This ensures sufficient elevation during approach in order to prevent collision with obstacles on the surface or terrains.

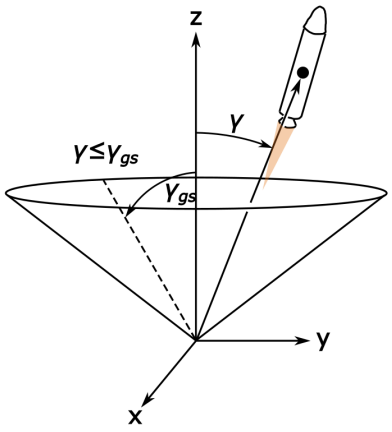

Figure 3: Glideslope constraint.

This constraint can be expressed as the following:

$$\hat{\mathbf{e}}_u^T \mathbf{r}(t) \geq \|\mathbf{r}(t)\|_2 \cos(\gamma_{\text{gs}}) \tag{4}$$

2.3 Boundary Conditions

The initial condition may be anything as long as it satisfies the glideslope constraint. As for the final conditions, the paper defines the landing site to be the origin, $O = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$. Therefore, we want $\mathbf{r}(t_f) = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$. We also want the velocity vector $\mathbf{v}(t)$ to be zero at the final time t_f. And lastly, we want the rocket to be oriented upwards at the final time (but realistically, we

just need it to be within some small angle θ_f from the vertical). The boundary conditions are summarized below:

$$\mathbf{r}(0) = \mathbf{r}_0,\ \mathbf{v}(0) = \mathbf{v}_0 \tag{5}$$

$$\mathbf{r}(t_f) = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T,\ \mathbf{v}(t_f) = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T,\ \|\mathbf{T}(t_f)\|_2 \cos(\theta_f) \leq \hat{\mathbf{e}}_u \mathbf{T}(t_f) \tag{6}$$

2.4 Nonconvex Problem Formulation

With all of the constraints in place, we can formulate the optimization problem as the following:

Problem 1

$$\min_{\sigma, \mathbf{u}} \ -m(t_f)$$

$$\text{subject to } (1) - (6)$$

We seek to minimize fuel consumption by maximizing the mass at the final time. This is a nonconvex optimization due to (1) and (2) being nonconvex. (1) is not a convex constraint because all equality constraints must be affine in a convex problem. (2) is not a convex constraint because $g(\mathbf{u}) = -\|\mathbf{T}\|_2 + T_{\min}$ is not a convex function and inequality constraints of the form $g(\mathbf{u}) \leq 0$ require g to be a convex function (or equivalently, requires g to be concave for inequality constraints of the form $g(\mathbf{u}) \geq 0$).

2.5 Convexification

To convert Problem 1 into a convex problem, (1) and (2) must be convexified. The dynamics constraint (1) can be convexified via linearization into a linear equality constraint. The thrust magnitude constraint (2) can be convexified using a technique known as "Lossless Convexification."

2.5.1 Lossless Convexification

This is a well-known technique in the field of trajectory optimization and was first introduced in the paper *Lossless Convexification of a Class of Optimal Control Problems with Non-Convex Control Constraints* by Behçet Açıkmeşe and Lars Blackmore in 2011. It effectively takes the nonconvex constraint in (2) and "lifts" it to a higher dimension by adding an additional input dimension, Γ. By constraining Γ to the following constraints, the lifted geometry of the feasible set becomes convex:

$$\|\mathbf{T}\|_2 \leq \Gamma \tag{7}$$

$$T_{\min} \leq \Gamma \leq T_{\max} \tag{8}$$

The lifted geometry can be visualized in Figure 4, which is clearly convex. By writing (8) slightly differently as $\|\mathbf{T}\|_2 \leq \min(\Gamma, T_{\max})$, the geometry in Figure 4 becomes more intuitive. That is, the new lifted feasible region is the intersection of a second-order cone given by (7), a half-space given by $T_{\min} \leq \Gamma$, and a cylinder given by $\|\mathbf{T}\|_2 \leq T_{\max}$.

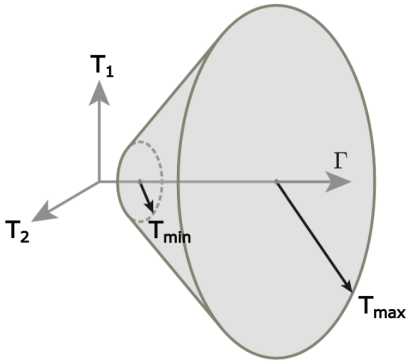

Figure 4: Relaxed feasible region $\mathcal{V}$.

This feasible region is relaxed in that it is larger and includes the original nonconvex set in (2). But notice that it also includes (T_1, T_2) that is not feasible with respect to the original constraint (2). This is where lossless convexification is useful. The key result of lossless convexification is that, at the optimum, the solution (u_1^*, u_2^*, Γ^*) will lie on the boundary of $\mathcal{V}$ such that $\Gamma = \|\mathbf{T}\|_2$ at the optimum. That is to say, it not only satisfies the relaxed constraints (7) and (8) but also satisfies the more strict original constraint (2). Knowing this, we can likewise substitute in Γ for $\|\mathbf{T}\|_2$ in any of the other constraints as well (just as we did with (8)). In this case, the only constraint that changes is the mass depletion dynamics:

$$\dot{m}(t) = -\alpha\Gamma(t) - \dot{m}_{\text{bp}} \tag{9}$$

2.5.2 Linearization

I chose to take a different approach in the linearization and discretization steps than the paper. First, I did a change of variable to map the original time coordinate to a normalized time coordinate $\tau \in [0, 1]$, via a mapping $\tau(t) = t/t_f$. Then using the chain rule, the dynamics in (1) can be rewritten in the normalized time:

$$\frac{d\mathbf{x}}{d\tau} = \left(\frac{d\tau}{dt}\right)^{-1} \frac{d\mathbf{x}}{dt}$$

Hereafter, let us denote $\sigma := (d\tau/dt)^{-1}$. We can now linearize the dynamics in normalized time about some reference trajectory $(\hat{\mathbf{x}}(\tau), \hat{\mathbf{u}}(\tau), \hat{\Gamma}(\tau), \hat{\sigma})$. If we write $d\mathbf{x}/d\tau$ as $d\mathbf{x}/d\tau = \mathbf{g}(\mathbf{x}, \mathbf{u}, \Gamma, \sigma)$, the linearization can be obtained by a first-order Taylor approximation of the function $\mathbf{g}(\mathbf{x}, \mathbf{u}, \Gamma, \sigma)$ about $(\hat{\mathbf{x}}(\tau), \hat{\mathbf{u}}(\tau), \hat{\Gamma}, \hat{\sigma})$. The result is a linear time-varying (LTV) system:

$$\begin{aligned}
\frac{d\mathbf{x}}{d\tau} &= \mathbf{g}(\mathbf{x}, \mathbf{u}, \Gamma, \sigma) \approx \mathbf{g}(\hat{\mathbf{x}}, \hat{\mathbf{u}}, \hat{\Gamma}, \hat{\sigma}) + \left.\frac{\partial \mathbf{g}}{\partial \mathbf{x}}\right|_{(\hat{\mathbf{x}}(\tau), \hat{\mathbf{u}}(\tau), \hat{\Gamma}(\tau), \hat{\sigma})} (\mathbf{x}(\tau) - \hat{\mathbf{x}}(\tau)) + \left.\frac{\partial \mathbf{g}}{\partial \mathbf{u}}\right|_{(\hat{\mathbf{x}}(\tau), \hat{\mathbf{u}}(\tau), \hat{\Gamma}(\tau), \hat{\sigma})} (\mathbf{u}(\tau) - \hat{\mathbf{u}}(\tau)) \\
&\quad + \left.\frac{\partial \mathbf{g}}{\partial \Gamma}\right|_{(\hat{\mathbf{x}}(\tau), \hat{\mathbf{u}}(\tau), \hat{\Gamma}(\tau), \hat{\sigma})} (\Gamma(\tau) - \hat{\Gamma}(\tau)) + \left.\frac{\partial \mathbf{g}}{\partial \sigma}\right|_{(\hat{\mathbf{x}}(\tau), \hat{\mathbf{u}}(\tau), \hat{\Gamma}(\tau), \hat{\sigma})} (\sigma - \hat{\sigma}) \\
&= \mathbf{g}(\hat{\mathbf{x}}, \hat{\mathbf{u}}, \hat{\Gamma}, \hat{\sigma}) + \hat{\sigma} \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{(\hat{\mathbf{x}}(\tau), \hat{\mathbf{u}}(\tau), \hat{\Gamma}(\tau))} (\mathbf{x}(\tau) - \hat{\mathbf{x}}(\tau)) + \hat{\sigma} \left.\frac{\partial \mathbf{f}}{\partial \mathbf{u}}\right|_{(\hat{\mathbf{x}}(\tau), \hat{\mathbf{u}}(\tau), \hat{\Gamma}(\tau))} (\mathbf{u}(\tau) - \hat{\mathbf{u}}(\tau)) \\
&\quad + \hat{\sigma} \left.\frac{\partial \mathbf{f}}{\partial \Gamma}\right|_{(\hat{\mathbf{x}}(\tau), \hat{\mathbf{u}}(\tau), \hat{\Gamma}(\tau))} (\Gamma(\tau) - \hat{\Gamma}(\tau)) + \hat{\sigma} \left.\frac{\partial \mathbf{f}}{\partial \sigma}\right|_{(\hat{\mathbf{x}}(\tau), \hat{\mathbf{u}}(\tau), \hat{\Gamma}(\tau))} (\sigma - \hat{\sigma})
\end{aligned}$$

Let us define the following matrices:

$$A(\tau) := \hat{\sigma}\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\bigg|_{(\hat{\mathbf{x}}(\tau),\hat{\mathbf{u}}(\tau),\hat{\Gamma}(\tau))} \tag{10}$$

$$B(\tau) := \hat{\sigma}\frac{\partial \mathbf{f}}{\partial \mathbf{u}}\bigg|_{(\hat{\mathbf{x}}(\tau),\hat{\mathbf{u}}(\tau),\hat{\Gamma}(\tau))} \tag{11}$$

$$G(\tau) := \hat{\sigma}\frac{\partial \mathbf{f}}{\partial \Gamma}\bigg|_{(\hat{\mathbf{x}}(\tau),\hat{\mathbf{u}}(\tau),\hat{\Gamma}(\tau))} \tag{12}$$

$$\Sigma(\tau) := f(\hat{\mathbf{x}}(\tau), \hat{\mathbf{u}}(\tau), \hat{\Gamma}(\tau)) \tag{13}$$

$$\mathbf{z}(\tau) := -A(\tau)\hat{\mathbf{x}}(\tau) - B(\tau)\hat{\mathbf{u}} - G(\tau)\hat{\Gamma}(\tau) \tag{14}$$

Then we can write the linearization compactly as:

$$\frac{d\mathbf{x}}{d\tau} = A(\tau)\mathbf{x}(\tau) + B(\tau)\mathbf{u}(\tau) + G(\tau)\Gamma(\tau) + \Sigma(\tau)\sigma + \mathbf{z}(\tau) \tag{15}$$

2.6 Convex Problem Formulation

With the dynamics linearized and constraint (2) convexified, we can write down the convexified problem, which the paper refers to as the "convex sub-problem":

Problem 2

$$\min_{\Gamma,\sigma,\mathbf{u}} \; -m(t_f)$$

$$\text{subject to } (5) - (8), (15)$$

Note that the convex sub-problem is a SOCP due to the second-order cone constraint (7).

2.7 Discretization

Thus far, the problem formulation has been in continuous time. An optimization problem in continuous time is infinite-dimensional and such a problem can neither be solved analytically nor is it computationally tractable to solve numerically. Therefore, discretization of the problem is needed to convert it to a finite-dimensional optimization problem. While the paper discretizes first and then linearizes, I decided to linearize first and then discretize as is done in this paper written by the same authors. Also, conveniently, an implementation for the related paper is provided by Sven Niederberger, whose implementation I modified for this problem.

The trajectory is discretized into K evenly spaced discretization points. We denote the k^{th} temporal node as τ_k and is defined by:

$$\tau_k := \left(\frac{k}{K-1}\right)$$

Moreover, we use a first-order hold on the control input $\mathbf{u}$ in between time steps. That is, over the interval $\tau \in [\tau_k, \tau_{k+1}]$, $\mathbf{u}(\tau)$ can be written as:

$$\mathbf{u}(\tau) = \alpha_k \mathbf{u}(\tau_k) + \beta_k \mathbf{u}(\tau_{k+1}) \tag{16}$$

where $\alpha_k := \left(\frac{\tau_{k+1}-\tau}{\tau_{k+1}-\tau_k}\right)$ and $\beta_k := \left(\frac{\tau-\tau_{k+1}}{\tau_{k+1}-\tau_k}\right)$. Notice that $\mathbf{u}(\tau)$ on the interval $\tau \in [\tau_k, \tau_{k+1}]$ is fully specified by $\mathbf{u}(\tau_k)$ and $\mathbf{u}(\tau_{k+1})$. Only two parameters are needed to describe $\mathbf{u}(\tau)$ over a continuous interval $\tau \in [\tau_k, \tau_{k+1}]$. This demonstrates how discretization parameterizes a continuous function using just a finite number of parameters. We apply the same first-order hold on $\Gamma(\tau)$:

$$\Gamma(\tau) = \alpha_k \Gamma(\tau_k) + \beta_k \Gamma(\tau_{k+1}) \tag{17}$$

Recall the solution to an LTV system, $\mathbf{x}(t) = A(t)\mathbf{x}(t) + B(t)\mathbf{u}(t)$:

$$\mathbf{x}(t) = \Phi(t, t_0)\mathbf{x}_0 + \int_{t_0}^{t} \Phi(t, \tau) B(\tau)\mathbf{u}(\tau) d\tau \tag{18}$$

where $\Phi(t, t_0)$ is a state-transition matrix given by the solution to:

$$\frac{d}{dt}\Phi(t, t_0) = A(t)\Phi(t, t_0),\ \Phi(t_0, t_0) = I$$

To obtain $\mathbf{x}(\tau_{k+1})$ from $\mathbf{x}(\tau_k)$ and $\mathbf{u}(\tau_k)$ from the previous time step, we make the substitution $t_0 = t_k$ and $t = t_{k+1}$ in (17). Then by inspection, the solution to (15) is the following:

$$\mathbf{x}_{k+1} = \bar{A}_k \mathbf{x}_k + \bar{B}_k \mathbf{u}_k + \bar{C}_k \mathbf{u}_{k+1} + \bar{G}_k \Gamma_k + \bar{H}_k \Gamma_{k+1} + \bar{\Sigma}_k \sigma + \bar{\mathbf{z}}_k \tag{19}$$

where

$$\bar{A}_k := \Phi_A(\tau_{k+1}, \tau_k) \tag{20}$$

$$\bar{B}_k := \int_{\tau_k}^{\tau_{k+1}} \Phi_A(\tau_{k+1}, \xi) B(\xi) \alpha_k(\xi) d\xi \tag{21}$$

$$\bar{C}_k := \int_{\tau_k}^{\tau_{k+1}} \Phi_A(\tau_{k+1}, \xi) B(\xi) \beta_k(\xi) d\xi \tag{22}$$

$$\bar{G}_k := \int_{\tau_k}^{\tau_{k+1}} \Phi_A(\tau_{k+1}, \xi) G(\xi) \alpha_k(\xi) d\xi \tag{23}$$

$$\bar{H}_k := \int_{\tau_k}^{\tau_{k+1}} \Phi_A(\tau_{k+1}, \xi) G(\xi) \beta_k(\xi) d\xi \tag{24}$$

$$\bar{\Sigma}_k := \int_{\tau_k}^{\tau_{k+1}} \Phi_A(\tau_{k+1}, \xi) \Sigma(\xi) d\xi \tag{25}$$

$$\bar{\mathbf{z}}_k := \int_{\tau_k}^{\tau_{k+1}} \Phi_A(\tau_{k+1}, \xi) \mathbf{z}(\xi) d\xi \tag{26}$$

*ξ was used as the dummy variable in the integration and not τ to avoid confusion with the normalized time coordinate τ.
Equation (19) is the linearized discrete-time dynamics of the system.

2.8 Relaxation via Virtual Control

One ill consequence of linearization is that it may render the convex sub-problem infeasible when the original nonconvex problem in fact does permit a solution. This is detrimental to the SCP algorithm as the SCP algorithm cannot continue iterating, resulting in a termination of the algorithm without having converged. To prevent this, the paper introduces an additive relaxation term

$\boldsymbol{\nu}$ that acts as a synthetic control input (called *virtual control*) that acts on the rocket when there exists no feasible $\mathbf{u}$ that is sufficient to satisfy all constraints. With the virtual control included, (19) becomes:

$$\mathbf{x}_{k+1} = \bar{A}_k\mathbf{x}_k + \bar{B}_k\mathbf{u}_k + \bar{C}_k\mathbf{u}_{k+1} + \bar{G}_k\Gamma_k + \bar{H}_k\Gamma_{k+1} + \bar{\Sigma}_k\sigma + \bar{\mathbf{z}}_k + \boldsymbol{\nu}_k \tag{27}$$

Because the virtual control is a synthetic control input that cannot actually be physically applied to the rocket (it only exists as a buffer against infeasibility), we want the virtual control to be zero at the converged solution. Therefore, we need to penalize the usage of virtual control as to discourage its use as much as possible. The new objective function includes this penalty term:

$$\min_{\Gamma,\sigma,\mathbf{u}}(-m(t_f) + w_\nu \left\|\bar{\boldsymbol{\nu}}\right\|_1) \tag{28}$$

where $\bar{\boldsymbol{\nu}} = \begin{bmatrix}\boldsymbol{\nu}_0 & \cdots & \boldsymbol{\nu}_{K-1}\end{bmatrix}$ and w_ν is a strictly positive weight.

2.9 Trust Region

Because the linearized dynamics are only accurate near the reference trajectory (that you linearized about), the solution from each iteration of the SCP algorithm must be kept "sufficiently close" to the reference trajectory. Additionally, another side effect of linearization is that it can cause the problem to be unbounded below (i.e. cost can be driven to negative infinity) when the original nonconvex has a solution with bounded cost. For these two reasons, the paper defines a trust region:

$$\begin{aligned}\delta\mathbf{x}(\tau_k) &= \mathbf{x}(\tau_k) - \hat{\mathbf{x}}(\tau_k)\\ \delta\mathbf{u}(\tau_k) &= \mathbf{u}(\tau_k) - \hat{\mathbf{u}}(\tau_k)\\ \delta\sigma &= \sigma - \hat{\sigma}\end{aligned}$$

To prevent the solution from deviating too far from the reference trajectory, we introduce the following trust region constraint:

$$\left\|\delta\mathbf{x}(\tau_k)\right\|_1 + \left\|\delta\mathbf{u}(\tau_k)\right\|_1 + \left\|\delta\sigma\right\|_1 \leq \eta \quad \forall\, k \in \{0, 1, ..., K-1\} \tag{29}$$

where η is the trust region radius.

To prevent the SCP algorithm from being overly conservative or overly optimistic, the paper implements an "update rule" for updating the trust region radius with each iteration of the SCP algorithm. "Overly conservative" in this context means that the solution is staying too close to the reference trajectory to a point where it could've found a better trajectory by straying farther away from the reference trajectory. "Overly optimistic" means that the solution explored too far away from the reference trajectory in hopes that it would discover a better trajectory, but it didn't. To define what exactly constitutes being too conservative or optimistic, we define an improvement ratio:

$$\rho := \frac{\mathcal{J}(\hat{\mathbf{x}}, \hat{\mathbf{u}}, \hat{\Gamma}, \hat{\sigma}) - \mathcal{J}(\mathbf{x}^*, \mathbf{u}^*, \Gamma^*, \sigma^*)}{\mathcal{J}(\hat{\mathbf{x}}, \hat{\mathbf{u}}, \hat{\Gamma}, \hat{\sigma}) - \mathcal{L}(\mathbf{x}^*, \mathbf{u}^*, \Gamma^*, \sigma^*)} = \frac{\text{actual improvement}}{\text{predicted improvement}} \tag{30}$$

where $\mathcal{J}$ is the nonlinear augmented cost and $\mathcal{L}$ is the linear augmented cost. Typically, in a general SCP problem, you will have nonconvex state and control constraints. To convexify them,

you may choose to either linearize the constraints into an affine one (i.e. half-space constraint) and add slack variables or use lossless convexification. If you use choose to linearize the constraint, there then needs to be a distinction between violation of the linearized constraint versus violation of the original nonconvex constraint.

At a high level, $\mathcal{J}$ looks at how much you've violated the actual original nonconvex constraints whereas $\mathcal{L}$ looks at how much you've violated the lineraized constraints. For example, for constraint (2), we can choose to linearize it rather than using lossless convexification. Recall the part of (2) that makes it nonconvex:

$$g(\mathbf{u}) = T_{\min} - \|\mathbf{u}\|_2 \leq 0$$

A first-order Taylor approximation of the LHS about a reference $\hat{\mathbf{u}}$ yields the following:

$$g(\mathbf{u}) \approx g(\hat{\mathbf{u}}) + \left.\frac{\partial g}{\partial \mathbf{u}}\right|_{\hat{\mathbf{u}}} = (T_{\min} - \|\hat{\mathbf{u}}\|_2) - \frac{\hat{\mathbf{u}}^T}{\|\hat{\mathbf{u}}\|_2}(\mathbf{u} - \hat{\mathbf{u}}) \leq s \tag{31}$$

where we have added a slack variable s to avoid artificial infeasibility due to linearization, just as we did for linearizing the dynamics. Again, we would then need to add a penalty term to the objective function to ensure that the slack variable is "used" only when it is necessary to avoid problem infeasibility. The resulting cost function that includes this penalty term on s is the linear augmented cost $\mathcal{L}$. Whereas, the nonlinear augmented cost $\mathcal{J}$ would penalize how much the optimum $\mathbf{u}^*$ violates the original nonconvex cost $\|\mathbf{u}\|_2 \leq T_{\min}$ (i.e. you would add a term $(\|\mathbf{u}^*\|_2 - T_{\min})$ to the cost function).

In (30), $\mathcal{J}(\hat{\mathbf{x}}, \hat{\mathbf{u}}, \hat{\Gamma}, \hat{\sigma})$ represents the true nonlinear cost of the reference trajectory (i.e. solution from previous iteration). $\mathcal{J}(\mathbf{x}^*, \mathbf{u}^*, \Gamma^*, \sigma^*)$ represents the true nonlinear cost of the new solution from the most recent iteration. These nonlinear augmented costs not only penalize actual violation of the nonconvex constraints (rather than penalizing the slack) but also the *defect*. Defect is the difference between the trajectory that the linearized dynamics predicts and the trajectory that the true nonlinear dynamics predicts. Penalizing the defect discourages solutions that are dynamically infeasible with respect to the true nonlinear dynamics. In contrast, the linear augmented cost $\mathcal{L}$ penalizes the virtual control $\boldsymbol{\nu}$ rather than penalizing the defect in the dynamics. Also, it penalizes violation of the linearized control and state constraints (by penalizing the slack variable introduced after linearizing) rather than penalizing violation of the original nonconvex control and state constraints. Therefore, $\mathcal{L}(\mathbf{x}^*, \mathbf{u}^*, \Gamma^*, \sigma^*)$ represents how much the linearized dynamics and linearized control and state constraints are violated in the convex sub-problem.

The numerator of (30) represents the true nonlinear cost of the reference trajectory (i.e. solution from previous iteration) minus the true nonlinear cost of the new solution from the most recent iteration – it is the improvement in the true nonlinear cost. The denominator represents the true nonlinear cost of the reference trajectory minus the linear cost of the new solution (i.e. how much the linearized constraints are violated) – it is the improvement in the cost that the solution to the convex sub-problem predicts. The update rule for the trust region radius used is as follows:

- $\rho < \rho_0$: Actual cost improvement was much smaller than the linearized sub-problem predicted
  $\rightarrow$ Reject solution $(\mathbf{x}^*, \mathbf{u}^*, \Gamma^*, \sigma^*)$ and shrink the trust region by a shrink factor β_{sh}.

- $\rho \in [\rho_0, \rho_1)$: Actual cost improvement is reasonable but still below the threshold ρ_1
  $\rightarrow$ Accept solution $(\mathbf{x}^*, \mathbf{u}^*, \Gamma^*, \sigma^*)$ as the new reference and shrink the trust region by a shrink factor β_{sh}.

- $\rho \in [\rho_1, \rho_2)$: Actual cost is close to predicted cost, meaning linear approximation was accurate
  $\rightarrow$ Accept the solution $(\mathbf{x}^*, \mathbf{u}^*, \Gamma^*, \sigma^*)$ as the new reference and keep radius η the same.

- $\rho > \rho_2$: Actual cost improvement was larger than linear prediction, meaning that the linearized sub-problem was too conservative
  $\rightarrow$ Accept $(\mathbf{x}^*, \mathbf{u}^*, \Gamma^*, \sigma^*)$ as the new reference and increase radius η by a growth factor β_{gr}.

The update rule is a crucial component of the SCP algorithm in that it helps with convergence of the algorithm.

2.10 Convex Problem Formulation

Now we can state the convex problem in a discrete-time setting with virtual control (to avoid artificial infeasibility) and a trust region (to avoid artificial unboundedness):

Problem 3

$$\min_{\Gamma, \sigma, \mathbf{u}} \left(-m(t_f) + w_v \left\|\bar{\boldsymbol{\nu}}\right\|_1\right)$$

$$\text{subject to } (5) - (8), (27), (29)$$

Problem 3 is the convex sub-problem that the SCP algorithm needs to solve iteratively until convergence. Note that the sub-problem is a SOCP due to the second-order cone constraint (7). Since we used lossless convexification, we do not have any penalty terms for slack variables from linearization of control and state constraints.

2.11 Successive Convexification Algorithm

The successive convexification algorithm is as follows:

Algorithm 1 Successive Convexification for 3-DOF Rocket Landing

Require: Maximum number of iterations `max_iter`, number of discretization points K, trust region parameters $(\rho_0, \rho_1, \rho_2, \beta_{\text{sh}}, \beta_{\text{gr}})$, initial trajectory $(X^{\text{init}}, U^{\text{init}}, \Gamma^{\text{init}})$, initial final-time guess σ, and model object `m` (containing dynamics, constraints, etc.).

Ensure: A converged trajectory $(X^*, U^*, \Gamma^*, \sigma^*)$.

`m.nondimensionalize()` ▷ Scale problem for improved numerical accuracy
`(`X`,`U`,`Γ`) = m.initialize_trajectory(`X`,`U`,`Γ`)` ▷ Initial guess for X, U, Γ
$\sigma \leftarrow$ `m.t_f_guess` ▷ Initial guess for σ
`integrator = FirstOrderHold(m,`K`)` ▷ Contains integration methods for discretization
`problem = SCProblem(m,`K`)` ▷ Formulate sub-problem
`last_nonlinear_cost` $\leftarrow$ `None` ▷ Initialize $\mathcal{J}$
`converged` $\leftarrow$ `False` ▷ Flag for convergence

for `it` $= 1$ **to** `max_iter` **do**
  `(`$\bar{A}_k$`,`$\bar{B}_k$`,`$\bar{C}_k$`,`$\bar{G}_k$`,`$\bar{H}_k$`,`$\bar{S}_k$`,`$\mathbf{z}_k$`)` $\leftarrow$ `integrator.calculate_discretization(`X`,`U`,`σ`,`Γ`)`
    ▷ Compute linearized system matrices
  `problem.set_parameters(`$\bar{A}_k$`,`$\bar{B}_k$`,`$\bar{C}_k$`,`$\bar{G}_k$`,`$\bar{H}_k$`,`$\bar{S}_k$`,`$\mathbf{z}_k$`,`X_{last}`,`σ_{last}`,`X_{last}`,`U_{last}`,`Γ_{last}`,`w_ν`,`η`)`
    ▷ Pass in system matrices, reference trajectory, weights, and trust region radius to sub-problem
  while `True` **do**:

```
        problem.solve(...)                                                    ▷ Solve sub-problem
        X_new ← problem.get_variable(X)                                       ▷ Extract solution
        U_new ← problem.get_variable(U)
        σ_new ← problem.get_variable(σ)
        Γ_new ← problem.get_variable(Γ)
        X_nonlinear ← integrator.integrate_nonlinear_piecewise(X_new, U_new, σ_new, Γ_new)
        L_dynamics ← ‖ν‖_1 from problem.get_variable(ν)   ▷ Cost associated with violating linearized dynamics (i.e. cost of using virtual control)
        J_dynamics ← ‖X_new − X_nonlinear‖_1                                  ▷ Penalizing dynamic infeasibility
        L_constraints ← m.get_linear_cost()
        J_constraints ← m.get_nonlinear_cost(X_new, U_new)
        L_current ← L_dynamics + L_constraints
        J_current ← J_dynamics + J_constraints

        if J_last = None then
            J_last ← J_current
            (X, U, σ, Γ) ← (X_new, U_new, σ_new, Γ_new)
            break while
        end if
        actual_change ← J_last − J_current
        predicted_change ← J_last − L_current
        if |predicted_change| < 10^−4 then
            converged ← True
            break while
        else
            ρ ← actual_change / predicted_change
            if ρ < ρ_0 then
                η ← η/β_sh                          ▷ Reject solution if ρ below threshold and shrink η
            else
                (X, U, σ, Γ) ← (X_new, U_new, σ_new, Γ_new)   ▷ Accept solution as new reference trajectory
                J_last ← J_current
                if ρ < ρ_1 then
                    η ← η/β_sh                                                ▷ Shrink η
                else if ρ ≥ ρ_2 then
                    η ← ηβ_gr                                                 ▷ Grow η
                end if
                break while
            end if
        end if
        problem.set_parameters(tr_radius = η)                                 ▷ Update η in sub-problem
    end while
    if converged = True then
        break for
    end if
end for
```

3 Simulation Results

The algorithm was simulated using the same parameters used in the paper and converged after 12 iterations. Figure 5 shows the converged trajectories for the state $\mathbf{x}$ and Figure 6 shows the converged trajectory for the control input $\mathbf{T}$.

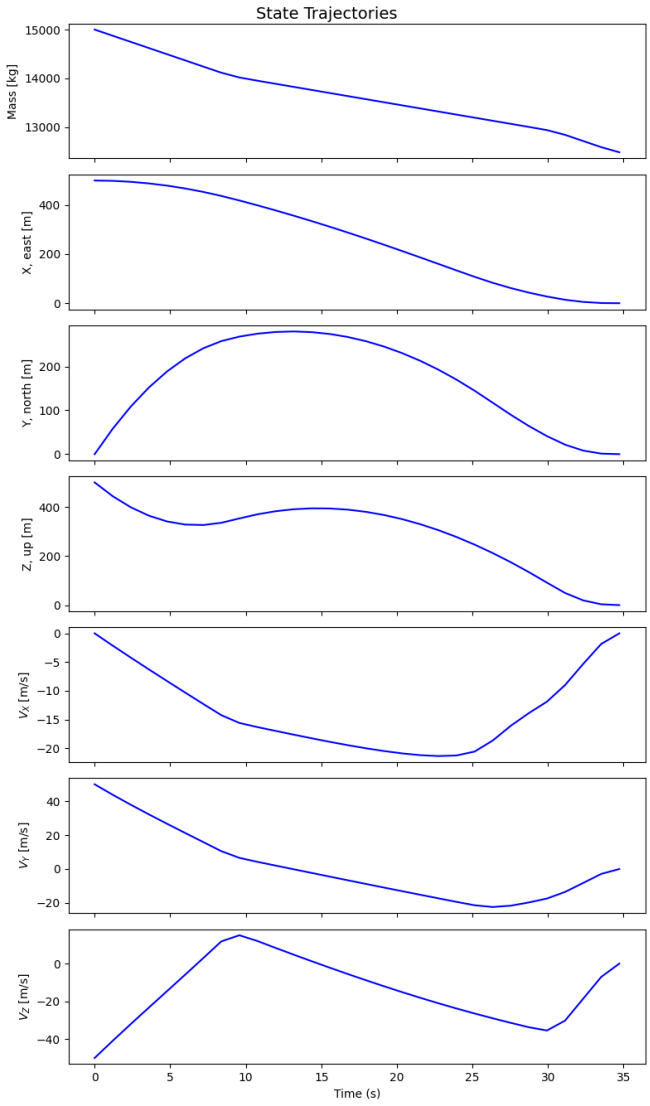

Figure 5: Plots of state trajectories for the converged solution.

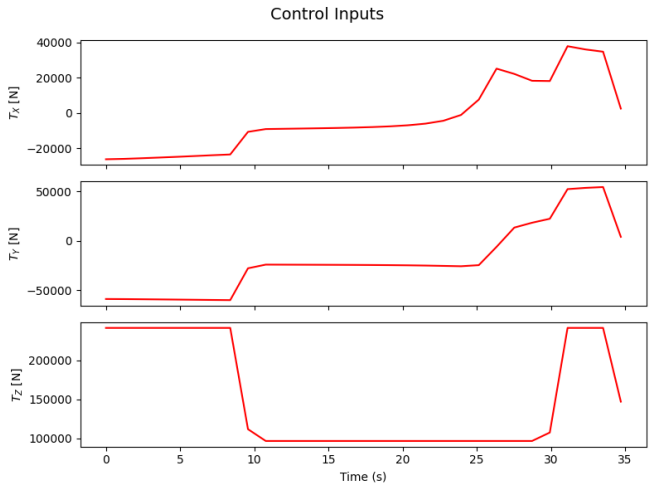

Figure 6: Plots of control trajectories for the converged solution.

To ensure that virtual control is zero at the converged solution, I plotted the 2-norm of the virtual control over iteration in Figure 7. I was able to confirm that the virtual control is indeed zero at the converged solution.

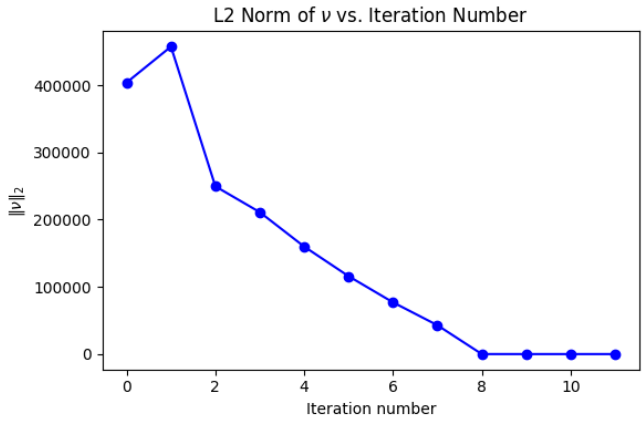

Figure 7: Plot of 2-norm of virtual control for each iteration of the algorithm.

Figure 8 on the next page shows the 3D plot of the converged optimal trajectory.

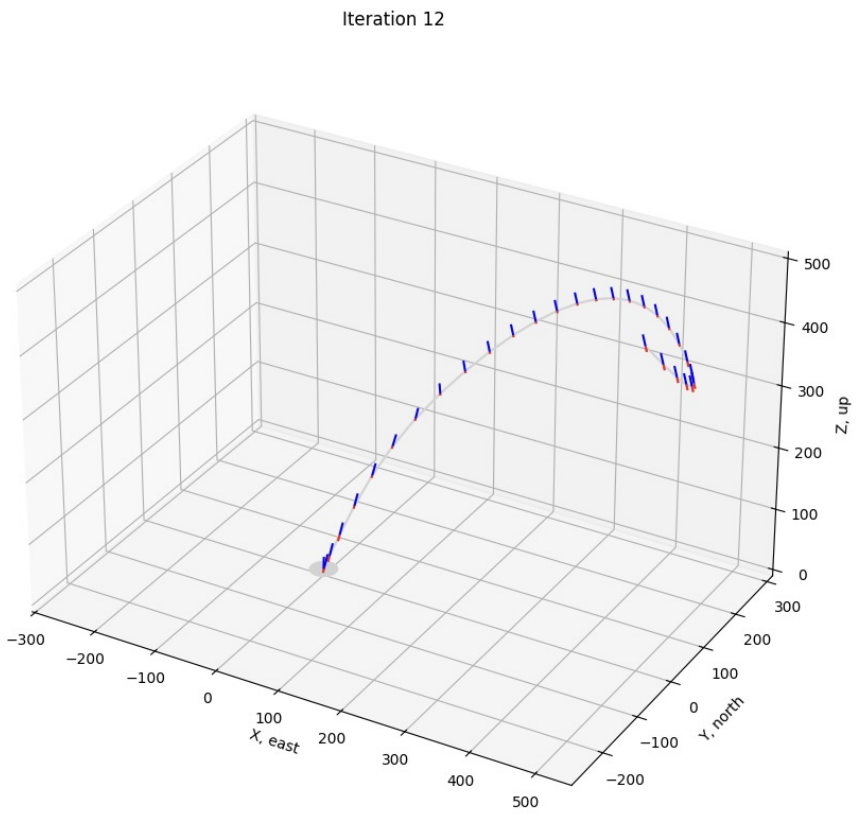

Figure 8: 3D Plot of the converged optimal trajectory of the rocket. The red line represents the thrust vector and its length indicates the thrust magnitude.

References

[1] Szmuk, M., Açikmeşe, B., Berning A. W., and Huntington G. *Successive Convexification for Fuel-Optimal Powered Landing with Aerodynamic Drag and Non-Convex Constraints*. Available at: https://www.researchgate.net/publication/306356846_Successive_Convexification_for_Fuel-Optimal_Powered_Landing_with_Aerodynamic_Drag_and_Non-Convex_Constraints, 2017.

[2] Açikmeşe, B. and Blackmore, L. *Lossless Convexification of a Class of Optimal Control Problems with Non-Convex Control Constraints*. Automatica, 47(2):341–347, 2011. http://larsblackmore.com/AcikmeseBlackmoreAutomatica10.pdf

[3] Szmuk, M. and Açikmeşe, B. *Successive Convexification for 6-DoF Mars Rocket Powered Landing with Free-FinalTime*. arXiv:1802.03827, 2018. https://arxiv.org/pdf/1802.03827