Sequential Convex Programming for a 3DOF Rocket Landing Problem

Kutay Demiralay, Department of Aeronautics and Astronautics

Project Codes!

Introduction

-Why Autonomy is Important?: Autonomous technologies represent the future, driving precision, efficiency, environmental sustainability, and safety.

-What is SCP ?: Sequential Convex Programming (SCP) is a, capable optimization technique that enhances autonomy by iteratively solving convex subproblems to find an optimal solution for complex, non-convex problems.

-Why do we Care About Landing a Rocket Vertically Using Thrusters? : It allows us to safely land a rocket on its legs at any desired location, enabling reuse of the rocket, lowering the cost of space flight.

-In this project I implemented the SCP algorithm from reference [1] in a Python coding environment to understand its application for trajectory optimization for autonomous Fixed Final time 3DOF rocket landing.

-For comparison I also solved the same problem using a nonlinear solver package.

-I aim to learn the SCP in depth and identify potential improvements and compare it with another general and widely used solving technique. My goal is to understand the algorithm's limits and analyze its behavior in different scenarios.

-Challengers are implementing the code from the paper [1]; adapting the 6 DoF algorithm, dynamics to 3 DoF problem; writing a code that is properly tuned for reasonable performance; and solving the same problem using a general nonlinear solving package available in Python programming language.

Problem Description

-Problem involves trajectory optimization for landing a fixed final time 3 DoF rocket, given initial, final constraints and dynamics. The dynamics are nonlinear and thus non-convex. In SCP algorithm we handle non-convexities through convexification, achieved via linearization, discretization, and integration. We are trying to minimize fuel and/or control input usage in landing.

$$\frac{d}{dt}\begin{bmatrix} m(t) \\ r_x \\ r_y \\ v_x \\ v_y \\ \theta \\ \omega \\ T_B(t) \\ \dot{T}_B(t) \end{bmatrix} = \begin{bmatrix} -\alpha \|T_B(t)\|_2 \\ v_x \\ v_y \\ \frac{1}{m}\sin(\theta + u[0]) T_B(t) \\ \frac{1}{m}\left(\cos(\theta + u[0]) T_B(t) - mg\right) \\ \omega \\ \frac{1}{J_B}\left(-\sin(u[0]) T_B(t) r_T\right) \\ \dot{T}_B(t) \\ u(t) \end{bmatrix}$$

Equation 1: Dynamics

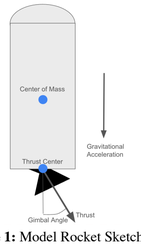

Figure 1: Model Rocket Sketch

-We compute the linearized system's state and control matrices and compare them with the nonlinear ones to ensure that, upon convergence, the nonlinear dynamics are accurately satisfied.

-With these techniques, our algorithm converts the original problem into a sequence of second-order cone programming (SOCP), which can be reliably and quickly solved using interior point method.

-Convex subproblems are solved using 'ECOS' solver, a default solver coming within CVXPY model.

-In General Nonlinear Optimizer algorithm we use scipy.optimize modules general solving capabilities for non-convex optimization problems, under the hood scipy.optimize uses sequential least squares method for this problem.

Parameter/Constraint	Value	Units	Description
m	2	kg	Wet (initial) mass of the rocket
J_B	1e-2	kg*m^2	Moment of inertia of the rocket
r_T	1e-2	m	Distance from thrust center to the center of mass
g	1	m/s^2	Gravitational acceleration
n_x	6	-	Number of state variables: [rx, ry, vx, vy, theta, omega]
n_u	2	-	Number of control inputs: [gimbal angle, thrust]
α (alpha)	0.10	-	Proportionality constant for mass depletion rate
Total Time of Flight	4	seconds	Total duration for the rocket landing maneuver

Table 1: Rocket Model Parameters

Constraint Type	Description	Value	Units
Boundary Conditions	Initial Position (X, Y)	(4, 4)	meters
	Initial Velocity (Vx, Vy)	(-2, -1)	meters/second
	Initial Angle	0	radians
	Initial Angular Velocity	0	radians/second
	Final Position (X, Y)	(0, 0)	meters
	Final Velocity (Vx, Vy)	(0, 0)	meters/second
	Final Angle	0	radians
	Final Angular Velocity	0	radians/second
State Constraints	Maximum Angle	±1.047	radians
	Maximum Angular Velocity	±1.047	radians/second
	Non-Negative Vertical Position	≥ 0	meters
Control Constraints	Maximum Gimbal Angle	±0.122	radians
	Minimum Thrust	2	newtons
	Maximum Thrust	5	newtons

Table 2: Constraints Defined for Rocket Landing Problem

Assumption
Aerodynamic forces are negligible in the dynamics of the vehicle.
The vehicle is subject to constant gravitational acceleration.
The center of mass (r_CM_B) and the moment of inertia (J_B) remain constant.
A first-order hold on the control, u(t), over each time step.
The vehicle is modeled as a rigid body.

Table 3: Assumptions

Background

Technical approach for the SCP algorithm, derived from Reference [1] (6DOF), is applied similarly to the 3DOF problem. The 6DOF problem uses the same principles as the 3DOF problem but it doesn't involve vectors in the z direction, quaternions, or direction cosine matrices. In my code, I also used the 4th Order Runge-Kutta Method for integration, incorporated the Penalized Trust Region Method, and implemented an adaptive step size method, virtual control, as different from Reference [1].

1. Compute the actual and predicted changes:

$$\text{Actual change} = J_{\text{nonlinear},k} - J_{\text{nonlinear},k+1}$$

$$\text{Predicted change} = J_{\text{nonlinear},k} - J_{\text{linear},k+1}$$

2. Calculate the ratio ρ:

$$\rho = \frac{\text{Actual change}}{\text{Predicted change}}$$

3. Adjust the trust region radius Δ based on ρ:

$$\Delta_{k+1} = \begin{cases} \Delta_k/\alpha & \text{if } \rho < \rho_0 \\ \Delta_k/\alpha & \text{if } \rho_0 \le \rho < \rho_1 \\ \Delta_k \times \beta & \text{if } \rho \ge \rho_2 \end{cases}$$

$$\text{Minimize:} \quad f^T x$$

$$\text{Subject to:} \quad \|A_i x + b_i\|_2 \le c_i^T x + d_i$$

$$i = 1, \ldots, m$$

Linearize around the current solution x_k:

$$f(x) \approx f(x_k) + \nabla f(x_k)^T (x - x_k)$$

$$g(x) \approx g(x_k) + \nabla g(x_k)^T (x - x_k)$$

Solve convex subproblem:

$$\min_p \quad \frac{1}{2} p^T H_k p + \nabla f(x_k)^T p$$

subject to:

$$g(x_k) + \nabla g(x_k)^T p \le 0$$

Update solution:

$$x_{k+1} = x_k + \alpha_k p_k$$

Equation 2: Adjusting Trust Region **Equation 3:** Second-order cone Programming **Equation 4:** Sequential Least Squares

The scipy.optimize.minimize command, which applies the sequential least squares method for nonlinear problems, minimizes the cost. This technique requires predefined ordinary differential equations and numerical integration, for which I used the 4th-order Runge-Kutta method again here.

Proposed Approach

-After defining constraints and dynamics, SCP follows four main steps: initialization, linearization, discretization, and solving convex subproblems to minimize the cost function.

-General Nonlinear Optimizer Algorithm is simpler and doesn't require linearization. Its code is similar to the LQR algorithms or Kalman filter codes we've used in our AA548 course. You just need to plug in the appropriate objective function and constraints into the scipy.optimize.minimize command, and the module handles the rest.

Initialization

1. **For** $k \in [0, K]$:

$$\bar{m}_k = \left(\frac{K-k}{K}\right) m_{\text{wet}} + \left(\frac{k}{K}\right) m_{\text{dry}}$$

$$\bar{r}_{I,k} = \left(\frac{K-k}{K}\right) r_{I,i} + \left(\frac{k}{K}\right) r_{I,f}$$

$$\bar{v}_{I,k} = \left(\frac{K-k}{K}\right) v_{I,i} + \left(\frac{k}{K}\right) v_{I,f}$$

$$\bar{T}_{B,k} = \bar{m}_k g e_{u,I}$$

$$\bar{x}_k^{(0)} = \begin{bmatrix} \bar{m}_k & \bar{r}_{I,k}^T & \bar{v}_{I,k}^T & 0 & \bar{T}_{B,k} & 0 \end{bmatrix}^T$$

$$\bar{u}_k^{(0)} = \begin{bmatrix} 0 & \frac{T_{\max} - T_{\min}}{2} \end{bmatrix}^T$$

2. **End Initialization Loop**
3. **For** $k \in [0, K]$:
   - Compute $\tilde{A}_k^{(0)}$, $\tilde{B}_k^{-,(0)}$, $\tilde{B}_k^{+,(0)}$, and $\tilde{z}_k^{(0)}$.
4. **End Initialization Loop**

Successive Convexification Loop

1. **For** $i \in [1, \text{imax}]$:
   1. Using $\bar{x}_k^{(i-1)}$, $\bar{u}_k^{(i-1)}$, $\tilde{A}_k^{(i-1)}$, $\tilde{B}_k^{-,(i-1)}$, $\tilde{B}_k^{+,(i-1)}$, and $\tilde{z}_k^{(i-1)}$, solve to obtain $\bar{x}_k^{(i)}$ and $\bar{u}_k^{(i)}$ that minimize $\|\bar{T}_{B,k}\|_2$.
   2. Set:

$$x_k^{(i)} \to \bar{x}_k^{(i)}, \quad u_k^{(i)} \to \bar{u}_k^{(i)}$$

   3. Compute $\tilde{A}_k^{(i)}$, $\tilde{B}_k^{-,(i)}$, $\tilde{B}_k^{+,(i)}$, and $\tilde{z}_k^{(i)}$.
   4. If $\|\eta^{(i)}\|_2 \le \text{tol}$, exit the loop.
2. **End Successive Convexification Loop**

Algorithm 1: Sequential Convex Programming Algorithm

1. **Initialization:** Initialize the initial state and final state, the number of state and control variables. Set physical constants, and define control limits.
2. **Continuous-Time Dynamics (ODE):** Create a differential equation that describes how the rocket's state changes over time based on its current state and control inputs:

$$\dot{r}_x = v_x$$
$$\dot{r}_y = v_y$$
$$\dot{v}_x = \frac{\sin(\theta + \gamma) \cdot T}{m}$$
$$\dot{v}_y = \frac{\cos(\theta + \gamma) \cdot T - m \cdot g}{m}$$
$$\dot{\theta} = \omega$$
$$\dot{\omega} = \frac{-\sin(\gamma) \cdot T \cdot r_T}{I}$$

3. **Integration:** Use the Runge-Kutta method to approximate the rocket's state at discrete time steps.
4. **Optimization:** Plug in the nonlinear problem into a non-convex problem solver.

Algorithm 2: General Nonlinear Optimizer Algorithm

$$\dot{x}(t) = A(t)x(t) + B(t)u(t) + z(t)$$

$$A(t) \triangleq \left.\frac{\partial f(x,u)}{\partial x}\right|_{\substack{\bar{x}(t) \\ \bar{u}(t)}}$$

$$B(t) \triangleq \left.\frac{\partial f(x,u)}{\partial x}\right|_{\substack{\bar{x}(t) \\ \bar{u}(t)}}$$

$$z(t) \triangleq f(\bar{x}(t), \bar{u}(t)) - A(t)\bar{x}(t) - B(t)\bar{u}(t)$$

Equation 5: Linearization Process

$$t_k \triangleq \left(\frac{k}{K-1}\right) t_f, \quad k \in [0, K)$$

$$u(t) = \lambda_k^-(t) u_k + \lambda_k^+(t) u_{k+1}$$

$$\lambda_k^-(t) \triangleq \left(\frac{t_{k+1} - t}{t_{k+1} - t_k}\right)$$
$$\lambda_k^+(t) \triangleq \left(\frac{t - t_k}{t_{k+1} - t_k}\right)$$

$$\frac{d}{dt}\Phi_A(t, t_k) = A(t)\Phi_A(t, t_k), \quad \Phi_A(t_k, t_k) = I$$

$$\tilde{A}_k \triangleq \Phi_A(t_{k+1}, t_k)$$
$$\tilde{B}_k^- \triangleq \tilde{A}_k \int_{t_k}^{t_{k+1}} \lambda_k^-(\tau)\Phi_A(t_k, \tau) B d\tau$$
$$\tilde{B}_k^+ \triangleq \tilde{A}_k \int_{t_k}^{t_{k+1}} \lambda_k^+(\tau)\Phi_A(t_k, \tau) B d\tau$$
$$\tilde{z}_k \triangleq \tilde{A}_k \int_{t_k}^{t_{k+1}} \Phi_A(t_k, \tau) z(\tau) d\tau$$

$$x_{k+1} = \tilde{A}_k x_k + \tilde{B}_k^- u_k + \tilde{B}_k^+ u_{k+1} + \tilde{z}_k + \nu_k$$

Equation 6: Discretization and Integration

Experimental results

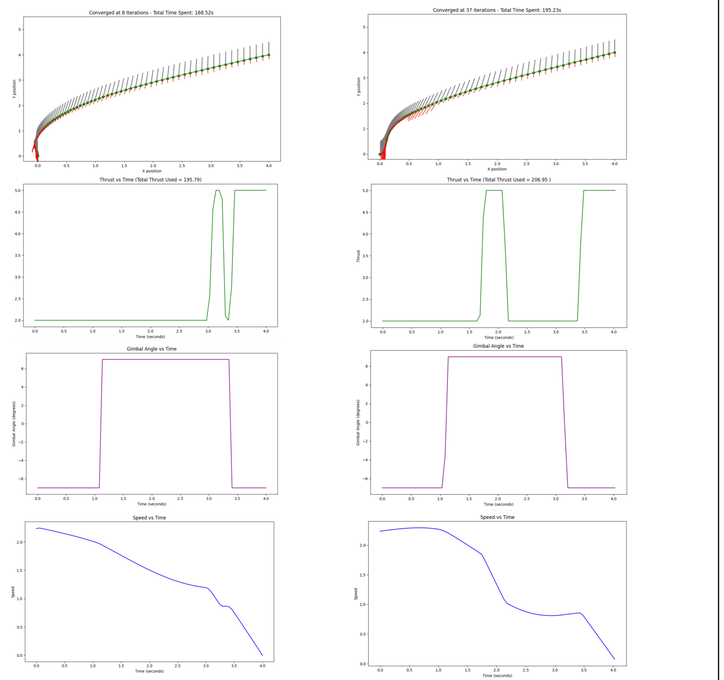

Figure 2: Results from Sequential Convexification Algorithm **Figure 3**: Results from General Nonlinear Optimizer Algorithm

- Both algorithms satisfy all the constraints, are feasible
- Two algorithms produced different optimal trajectories.
- The SCP algorithm uses less thrust (195.79 vs. 206.65 total thrust usage score).
- The SCP algorithm works faster on the same computer, runner, with the same number of trajectory points, maximum iterations, and the same problem, dynamics, and constraints (158.02 sec vs. 180 sec total solving time)
- At higher initial state values or longer desired flight times, trajectory optimization may not work as intended. I found that SCP performs better over a wider range of conditions compared to the general nonlinear optimizer.

Conclusions

-Even though developing and tuning SCP code is challenging and time-consuming, it's worth it. SCP offers better performance for this specific problems compared to general open-source codes designed for a variety of problems.

-Additionally, I can employ more tricks to make the SCP algorithm run even faster and with higher precision, which wouldn't be possible with a general nonlinear optimizer module.

-In the future, I want to develop a robust SCP algorithm that can handle external disturbances like wind or measurement noise. This project didn't consider wind, aerodynamic forces, or measurement errors, which isn't realistic.

References
[1] Szmuk, M., Eren, U., & Acikmese, B. (2017). Successive Convexification for Mars 6-DoF Powered Descent Landing Guidance. AIAA Guidance, Navigation, and Control Conference. doi:10.2514/6.2017-1500
[2]Samet Uzun, D-GMSR, MIT License, (2024), University of Washington, GitHub repository. https://github.com/sametuzun781/D-GMSR
[3] Sven Niederberger, SCvx, MIT License, (2018), GitHub repository. https://github.com/EmbersArc/SCvx
[4] Uzun, S., Elango, P., Garoche, P.-L., & Acikmese, B. (2024). Optimization with Temporal and Logical Specifications via Generalized Mean-based Smooth Robustness Measures. arXiv preprint arXiv:2405.10996.