Nonlinear Optimal Control Final Project: Minimum-Landing-Error Powered-Descent Guidance for Mars Landing Using Convex Optimization

University of Washington, Seattle, WA

Nick Nuxoll & Pranav Ramasahayam

26 November 2025

1 Motivation and System Description

Achieving precise landing on Mars has been one of the most important challenges in modern space exploration. Earlier missions (*Viking*, *Pathfinder*, and the *Mars Exploration Rovers*) were forced to deal with large landing-error ellipses that stretched for tens, even hundreds, of kilometers [1]. The large uncertainties thus limited where a spacecraft could safely land. This kept missions away from scientifically valuable, but hazardous terrain. As more precarious landing sites were considered, such as the Mars Science Laboratory's *Curiosity* mission, the necessity for a more performant landing algorithm became apparent. To meet this demand, the development of faster, more reliable onboard guidance algorithms that could work in real time, whilst using fuel efficiently and obeying strict safety requirements, was needed.

The powered-descent phase begins right after the vehicle completes an atmospheric entry and parachuted descent. Once the parachute is released, the vehicle is still moving at a high entry speed only a few kilometers above the Mars surface. The spacecraft then relies entirely on its thrusters to decelerate, change directions, and guide itself to its desired target. In this phase, aerodynamic effects are near-negligible, and the vehicle is modeled as a point mass under the influence of gravity with a controllable net thrust vector and magnitude. The thrust produced by the engines is limited by both minimum and maximum bounds. This is because the engines cannot be completely shut off or exceed their physical capabilities. As fuel is used, the mass of the craft decreases and must also be accounted for in the modeling.

The main objective of the powered-descent guidance algorithm is to bring the spacecraft from the initial position and velocity of entry, to the final state where it reaches the desired location on the surface and a zero velocity. During the descent, several safety constraints are enforced. The spacecraft must remain above the surface at all times and must stay within a safe glideslope. Traditionally, nonlinear optimization methods were not deemed reliable enough for a real-time, onboard implementation [1]. In order to solve this, the authors transform the problem into a convex optimization problem. This guarantees a globally optimal solution in a predictable and efficient way. This makes the approach practical, and usable for real missions and greatly improves landing accuracy and mission safety. Real-world testing of the larger G-FOLD algorithm [2] proved its viability and reliability, leading to its use in the *Curiosity* and *Perseverance* rovers [4], extending even to the booster landing algorithms used by SpaceX's *Falcon 9* and Blue Origin's *New Glenn*. This report presents a reimplementation of just one part of the G-FOLD algorithm: the convex optimization problems used to generate optimal powered-descent trajectories for landing guidance.

2 Formulation of the Optimal Control Problem

The precursor work [1] to this paper formulated a similar problem: minimum-fuel powered-descent guidance. It assumed that reaching the target was possible, and sought to minimize the fuel used by applying the optimal commanded thrust magnitude and pointing. The resulting continuous-time optimal control problem is as follows:

$$\max_{t_f, T_c(\cdot)} m(t_f) = \min_{t_f, T_c(\cdot)} \int_0^{t_f} \|T_c(t)\| \, \mathrm{d}t \tag{1}$$

subject to

$$\ddot{r}(t) = g + T_c(t)/m(t), \quad \dot{m}(t) = -\alpha \|T_c(t)\| \tag{2}$$

$$0 < \rho_1 \leq \|T_c(t)\| \leq \rho_2, \quad r_1(t) \geq 0 \tag{3}$$

$$\|S_j x(t) - v_j\| + c_j^T x(t) + a_j \leq 0 \tag{4}$$

$$j = 1, \dots, n_s$$

$$m(0) = m_{\text{wet}}, \quad r(0) = \mathbf{r}_0, \quad \dot{r}(0) = \dot{\mathbf{r}}_0 \tag{5}$$

$$r(t_f) = \dot{r}(t_f) = 0 \tag{6}$$

Starting with Eq. (1), the fuel-maximization problem is the dual problem to the minimization of applied input. $T_c(t)$ is the net thrust vector acting on the spacecraft, and $\ddot{r}(t)$ depends on this thrust, gravity and the mass, $m(t)$, as seen in Eq. (2). α is the fuel-burn rate per unit of thrust, used to find the rate of change of mass as a function of the commanded thrust. Then, inequality (3) defines the bounds on the thrust. Once started up, the thruster has a minimum thrust. Inequality (4) defines the path constraints of the trajectory, which can be used to avoid obstacles in the landing zone or to maintain a minimum glideslope. Lastly, Eq. (5) and (6) define initial and terminal conditions. The final fuel consumption also depends on the terminal time t_f, but this will have to be addressed outside the optimal control formulation and instead as part of the larger guidance algorithm. Unfortunately, inequalities (2) and (3) break the convexity of the problem, requiring a solution. The solution to the nonconvexity is to apply a relaxation of the thrust constraint, termed "lossless convexification" by the authors. A slack variable $\Gamma(t)$ is introduced, satisfying the following constraint:

$$\|T_c(t)\| < \Gamma(t) \tag{7}$$

Replacing all uses of $\|T_c(t)\|$ with $\Gamma(t)$ creates a new relaxed problem. It's clear that a solution to the original problem is a solution to the relaxed problem, but the reverse is not immediately obvious, nor that an optimal solution of the relaxed problem is an optimal or even feasible solution to the original. This relaxation is lossless, as shown by the authors. *Lemma 1* [1] shows that an optimal solution to the original problem can be found by solving the relaxed problem. The obtained result is incredibly valuable in practical implementation, as convex problems are computationally easy compared to nonlinear programs.

Next, further modifications are made to the initial problem to allow the implementation of the numerical solution. A change of variables is introduced, and the preceding constraints are reformulated:

$$\sigma \triangleq \frac{\Gamma}{m} \text{ and } u \triangleq \frac{T_c}{m} \tag{8}$$

$$\ddot{r}(t) = u(t) + g \tag{9}$$

$$\frac{\dot{m}(t)}{m(t)} = -\alpha\sigma(t) \tag{10}$$

$$\|u(t)\| \leqq \sigma(t) \ \forall \ t \in [0, t_f] \tag{11}$$

$$\frac{\rho_1}{m(t)} \leq \sigma(t) \leq \frac{\rho_2}{m(t)} \ \forall \ t \in [0, t_f] \tag{12}$$

Inequality (12) introduces nonconvexity due to $m(t)$ itself being a variable of the problem, causing the combined inequalities to become bilinear. To resolve this issue, another change of variable is introduced.

$$z \triangleq \ln m \tag{13}$$

This allows Eq. (10) to be rewritten, along with inequality (12).

$$\dot{z}(t) = -\alpha\sigma(t) \tag{14}$$

$$\rho_1 e^{-z(t)} \leq \sigma(t) \leq \rho_2 e^{-z(t)} \ \forall \ t \in [0, t_f] \tag{15}$$

However, inequality (15) is still nonconvex due to the right-hand side. The authors then use a Taylor series expansion to linearize this inequality. The left-hand side uses the first three terms while the right-hand side uses only the first two terms.

$$\rho_1 e^{-z_0}\left[1 - (z - z_0) + \frac{(z - z_0)^2}{2}\right] \leq \sigma \leq \rho_2 e^{-z_0}\left[1 - (z - z_0)\right] \tag{16}$$

Another change of variables is made for simplification, and then inequality (16) is rewritten, and $z_0(t)$ is defined.

$$\mu_1 \triangleq \rho_1 e^{-z_0} \text{ and } \mu_2 \triangleq \rho_2 e^{-z_0} \tag{17}$$

$$\mu_1 \left[1 - (z - z_0) + \frac{(z - z_0)^2}{2}\right] \leq \sigma \leq \mu_2 \left[1 - (z - z_0)\right] \tag{18}$$

$$z_0(t) = \ln\left(m_{\text{wet}} - \alpha \rho_2 t\right) \tag{19}$$

m_{wet} is the initial fully fueled mass of the vehicle, and $z_0(t)$ represents a lower bound on $z(t)$. An upper bound can also be established.

$$\ln\left(m_{\text{wet}} - \alpha \rho_2 t\right) \leq z(t) \leq \ln\left(m_{\text{wet}} - \alpha \rho_1 t\right) \tag{20}$$

The Taylor series approximation of the fuel constraints was found to be acceptably accurate, roughly a 2% error after about 70 seconds [1]. This is in the range of the expected optimal terminal time, so the approximation is not expected to create a large mismatch between the simplified problem and the true solution. The final step is to discretize the continuous-time problem. The authors choose a zero-order-hold discretization, essentially meaning that the input is held constant for the length of the time-step Δt. The state equation takes the resulting form:

$$x_{k+1} = A x_k + B u_k \tag{21}$$

$$x_{k+1} = A^k x_0 + A^{k-1} B u_1 + A^{k-2} B u_2 + \cdots + A B u_{k-2} + B u_{k-1} \tag{22}$$

$$x_{k+1} = A^K x_0 + \begin{bmatrix} A^{k-1}B & A^{k-2}B & \dots & AB & B \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{k-1} \end{bmatrix} \tag{23}$$

The A and B matrices have been discretized using the standard method. The first term, A^k, is termed Φ_K. Then, the input is considered as a combination of the accumulated effect of gravity and the actual control input. Considering only the "unperturbed" or "free" response of the system by ignoring the control input:

$$\Lambda_k = B + AB + \cdots + A^{k-1} B \tag{24}$$

$$\xi_k = \Phi_k \mathbf{y}_0 + \Lambda_k \begin{bmatrix} g \\ 0 \end{bmatrix} \tag{25}$$

$$\mathbf{y}_0 = \begin{bmatrix} \mathbf{r}_0 \\ \dot{\mathbf{r}}_0 \\ \ln m_{\text{wet}} \end{bmatrix} \tag{26}$$

Then, incorporating the effect of the applied input:

$$\mathbf{y}_k = \xi_k + \Psi_k \eta \tag{27}$$

Each row of Ψ_k is given by the form shown in Eq. (23), padded by zeros on the first and last row to account for the first and last time step.

$$\Psi_k = \begin{bmatrix} 0 & 0 & \dots & 0 & 0 \\ B & 0 & \dots & 0 & 0 \\ AB & B & \dots & 0 & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ A^{k-2}B & A^{k-3}B & \dots & B & 0 \\ A^{k-1}B & A^{k-2}B & \dots & AB & B \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix} \tag{28}$$

η is a stacked vector of the input vectors at each time step, until the final control input at time step $k = N-1$, where N is the discrete number of time steps considered.

$$u_k = \begin{bmatrix} u_{x,k} \\ u_{y,k} \\ u_{z,k} \\ \sigma_k \end{bmatrix} \quad \rightarrow \quad \eta = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix} \tag{29}$$

The discrete cost of an input η is given by $\omega^T \eta$, where ω is an $N \times 1$ vector of the form:

$$\omega = \begin{bmatrix} \Delta t e_\sigma^T & \dots & \Delta t e_\sigma^T \end{bmatrix}^T \tag{30}$$

$$e_\sigma = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T \tag{31}$$

Put more simply, ω multiplies each σ_k by Δt, corresponding to the zero-order-hold discretization. A few other auxiliary matrices are defined for mathematical completeness:

$$\Upsilon_k = \begin{bmatrix} 0_{4\times4} & \dots & 0_{4\times4} & I_{4\times4} & 0_{4\times4} & \dots & 0_{4\times4} \end{bmatrix} \tag{32}$$

$$E = \begin{bmatrix} I_{6\times6} & 0 \end{bmatrix} \tag{33}$$

$$E_r = \begin{bmatrix} I_{3\times3} & 0_{3\times4} \end{bmatrix} \tag{34}$$

$$E_v = \begin{bmatrix} 0_{3\times3} & I_{3\times3} & 0_{3\times1} \end{bmatrix} \tag{35}$$

$$E_u = \begin{bmatrix} I_{3\times3} & 0_{3\times1} \end{bmatrix} \tag{36}$$

$$F = \begin{bmatrix} 0_{1x6} & 1 \end{bmatrix} \tag{37}$$

$$\tag{38}$$

In Eq. (32), the I is in the k-th position. These are used to select from the state or control vectors, but are ultimately just an artifact of the mathematical statement, as for the programmatic solution, array/matrix indexing is used. This leads to the final formulation of the discrete-time optimal control problem to minimize the fuel usage.

$$\min_{N,\eta} \omega^T \eta \tag{39}$$

subject to

$$\|E_u \Upsilon_k \eta\| \leq e_\sigma^T \Upsilon_k \eta, \quad k = 0, \dots, N \tag{40}$$

$$\begin{aligned} &\mu_1(t_k)\left[1 - \{F\mathbf{y}_k - z_0(t_k)\} + \frac{\{F\mathbf{y}_k - z_0(t_k)\}^2}{2}\right] \\ &\quad \leq e_\sigma^T \Upsilon_k \eta \leq \mu_2(t_k)(1 - \{F\mathbf{y}_k - z_0(t_k)\}) \\ &\quad k = 1, \dots, N \end{aligned} \tag{41}$$

$$\begin{aligned} &\ln(m_{\text{wet}} - \alpha\rho_2 t_k) \leq F\mathbf{y}_k \leq \ln(m_{\text{wet}} - \alpha\rho_1 t_k) \\ &\quad k = 1, \dots, N \end{aligned} \tag{42}$$

$$\begin{aligned} &\|S_j E\mathbf{y}_k - v_j\| + c_j^T E\mathbf{y}_k + a_j \leq 0 \\ &\quad k = 1, \dots, N, \quad j = 1, \dots, n_s \end{aligned} \tag{43}$$

The authors' later work [3], and the focus of this project, considers the case where reaching the intended landing point may not be feasible. Then, for real-world applicability, a variation of this optimal control problem must be solved to instead minimize the landing error. Generally, the constraints are quite similar as can be seen below.

$$\min_{N,\eta} \|E_r y_N\|^2 \tag{44}$$

subject to

$$\|E_u \Upsilon_k \eta\| \leq \mathbf{e}_4^T \Upsilon_k \eta, \quad k = 0, \dots, N \tag{45}$$

$$\begin{aligned} &\rho_1 e^{-z_0(t_k)}\left[1 - (F\mathbf{y}_k - z_0(t_k)) + \frac{(F\mathbf{y}_k - z_0(t_k))^2}{2}\right] \\ &\quad \leq \mathbf{e}_4^T \Upsilon_k \eta \leq \rho_2 e^{-z_0(t_k)}[1 - (F\mathbf{y}_k - z_0(t_k))] \\ &\quad k = 1, \dots, N \end{aligned} \tag{46}$$

$$E_r \mathbf{y}_k \in \mathbf{X}, \quad k = 1, \dots, N \tag{47}$$

$$F y_N \geq \ln m_{\text{dry}} \tag{48}$$

$$y_N^T \mathbf{e}_1 = 0, \quad E_v y_N^T = \mathbf{0} \tag{49}$$

$$\mathbf{y}_k = \Phi_k \begin{bmatrix} \mathbf{r}_0 \\ \dot{\mathbf{r}}_0 \\ \ln m_{\text{wet}} \end{bmatrix} + \Lambda_k \begin{bmatrix} \mathbf{g} \\ 0 \end{bmatrix} + \Psi_k \eta \qquad k = 1, \dots, N \tag{50}$$

There are a few small differences, however. Previously, inequality (4) was used for path trajectories. Here, inequality (47) is used, where **X** defines the feasible set free of obstacles and within the glideslope. Next, inequality (48) explicitly constrains the minimum fuel at the terminal time. Lastly, equation (49) constrains only the final altitude and velocity to 0, allowing for feasible solutions where the vehicle lands away from the intended target.

As mentioned previously, finding the optimal trajectory also requires searching the space of terminal times. This is not part of the optimal control formulation itself, instead addressed algorithmically by using the results of the optimization in a golden-section search. The authors show in the original paper that the fuel use is a unimodal function of time [1]. Thus, the golden-section search converges relatively quickly, but its main benefit is computational efficiency. With conservative lower and upper bounds, the realistic search space for the optimal terminal time is reduced in just a few iterations. The implementation of the search is detailed in the **Appendix**, as part of the MATLAB code implementation. The second-order cone programs were solved using CVX, a widely used convex optimization toolbox with an interface in MATLAB. This implementation takes around 20 seconds from problem initialization to solution, which could reasonably provide a solution before the start of the powered-descent phase. Actual flight software using this algorithm would likely be written in C or C++, greatly decreasing runtime. There are of course also many computational and algorithmic optimizations to be made, and this is just one part of the larger G-FOLD algorithm.

3 Results & Analysis

The first considered case is the minimum-fuel landing scenario. Using the problem parameters given in section **V** of [1], the initial state is then:

$$\mathbf{y}_0 = \begin{bmatrix} 1500 \\ 0 \\ 2000 \\ -75 \\ 0 \\ 100 \\ \ln 1905 \end{bmatrix} \tag{51}$$

This is given as `Case 1` in the provided code, found in the **Appendix**. The optimal 3D trajectory is shown in Fig. 1 below.

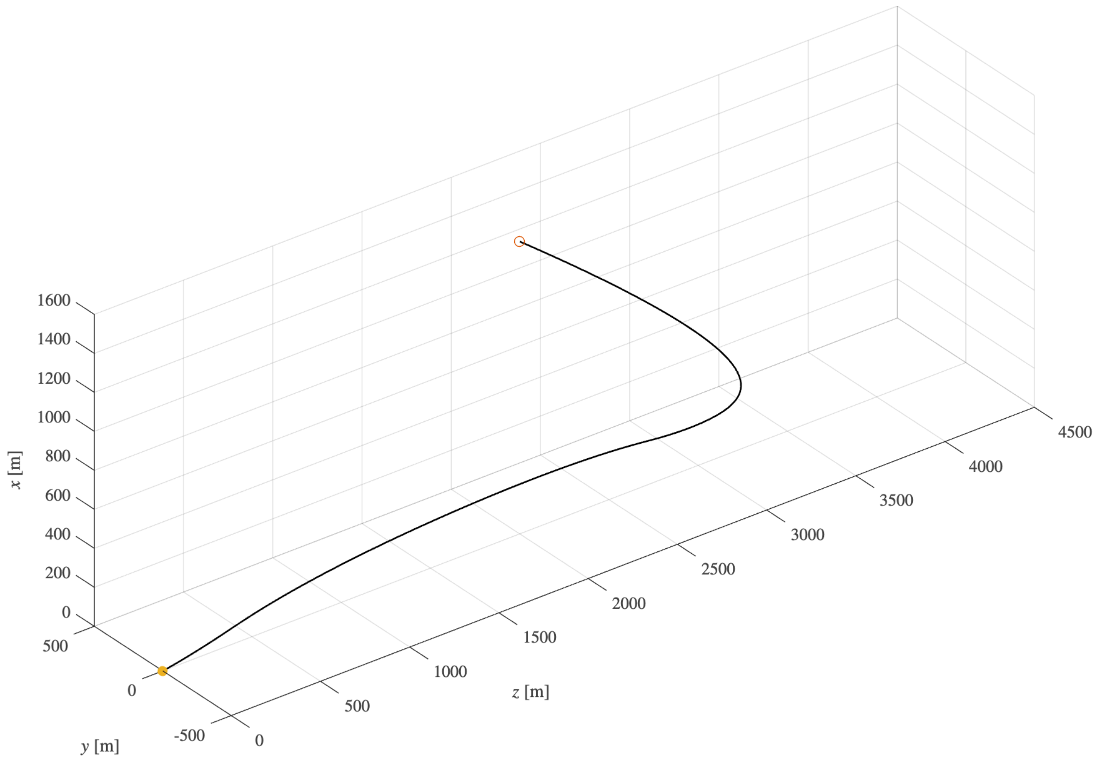


Figure 1: 3D view of optimal trajectory for minimum-fuel case.

It is important to note that the authors define height above the surface along the x-axis, leaving the y-and-z-axes as the horizontal plane. The vehicle arcs forward along the positive z-axis before reversing its horizontal velocity to reach the origin. It stays along the y-axis for the entire trajectory, as there was no initial offset from it or velocity pushing it away from the y-axis. This intuitively makes sense, and lends credence to the problem formulation. Next, in Fig. 2, the solution trajectory can be analyzed in more detail.

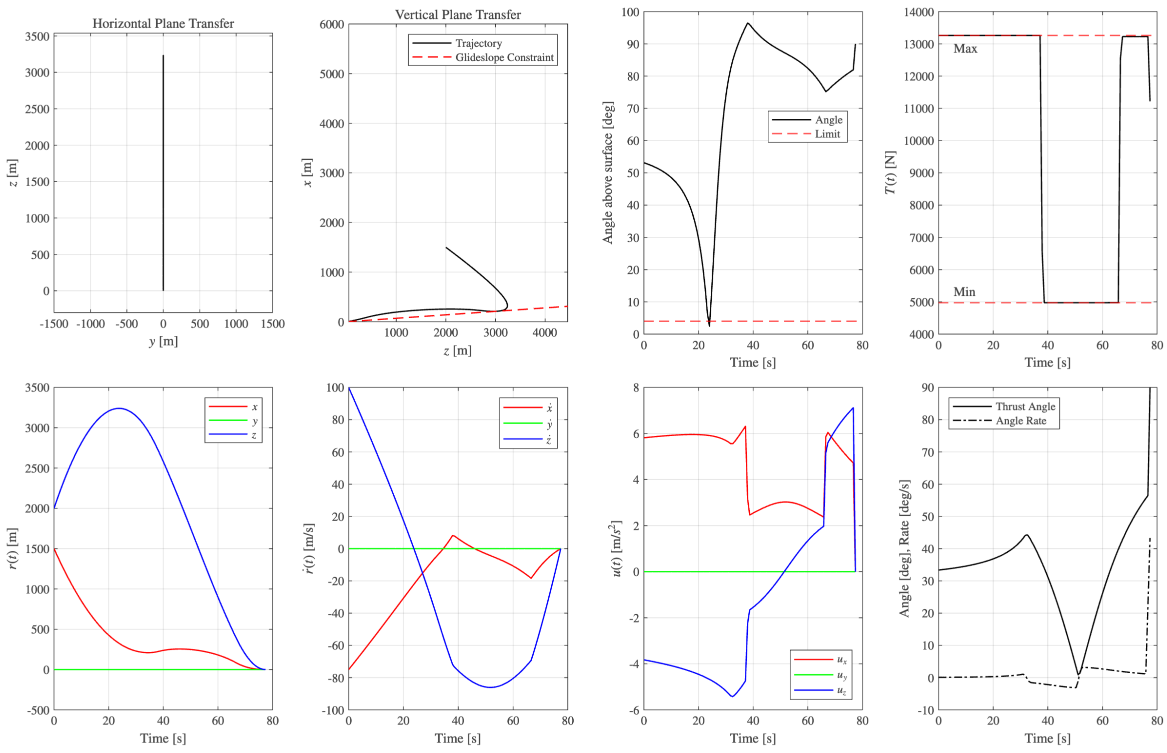


Figure 2: Overview of flight profile for minimum-fuel case.

Looking at the horizontal slice, the vehicle stays along the y-axis. Then, the vertical slice gives further insight into how the constraints affect the optimization. By requiring a minimum glideslope angle of 4°, the optimal trajectory is one that quickly drops to the minimum altitude and glides along it. The authors had the same finding, and forgoing the glideslope leads to a subsurface flight, obviously infeasible. The "angle above surface" plot shows the same result, where the vehicle touches the constraint briefly. The most telling sign of an optimal solution is the *max-min-max* structure of the commanded thrust in the next plot, as previous work in the area has shown this to be the optimal thrust profile for powered-descent. The position and velocity plots show a gradual trajectory to reach the origin, without sudden sharp turns or drops. The acceleration plot, shows a maximum acceleration of roughly 0.8g, which would be easily survivable for the vehicle. Lastly, the thrust angle and angle rate plot shows how the vehicle negates its original horizontal velocity before maneuvering to the target. The results found from this implementation are very close to those of the original paper, see Fig. 7 and 8 in [1] for comparison.

Next, moving to the minimum-landing-error case. The main paper [3] tests a modification to the initial condition, given below.

$$\mathbf{y}_0 = \begin{bmatrix} 1500 \\ 500 \\ 2000 \\ -75 \\ 40 \\ 100 \\ \ln 1905 \end{bmatrix} \tag{52}$$

With this initial condition, reaching the origin is not feasible, and thus the optimization objective is minimizing the landing-error. However, in this implementation, it was found that the vehicle was able to reach the origin. The cause of this discrepancy was not found, even after rewriting parts of the algorithm and checking for numerical issues. One potential cause is the choice of solver used with CVX, the convex optimization library used in the MATLAB implementation. The original paper used the SeDuMi solver, while this implementation used the default SDPT3 solver. It was found during implementation that the SeDuMi solver was unable to converge due to issues with numerical precision, causing early termination. SDPT3 is the out-of-the-box choice of solver due to its robustness and ease of integration into an algorithm, but SeDuMi is said to provide more accurate results if the problem allows its use. To allow for testing the minimum-landing-error portion of the algorithm, a modified initial condition was tested.

$$\mathbf{y}_0 = \begin{bmatrix} 1500 \\ 2500 \\ 2500 \\ -75 \\ 40 \\ 100 \\ \ln 1905 \end{bmatrix} \tag{53}$$

This test is denoted as `Case 2` in the code. By situating the vehicle further from the origin, reaching the origin has been made infeasible despite the implementation issues. First, the 3D view of the calculated trajectory is below in Fig. 3.

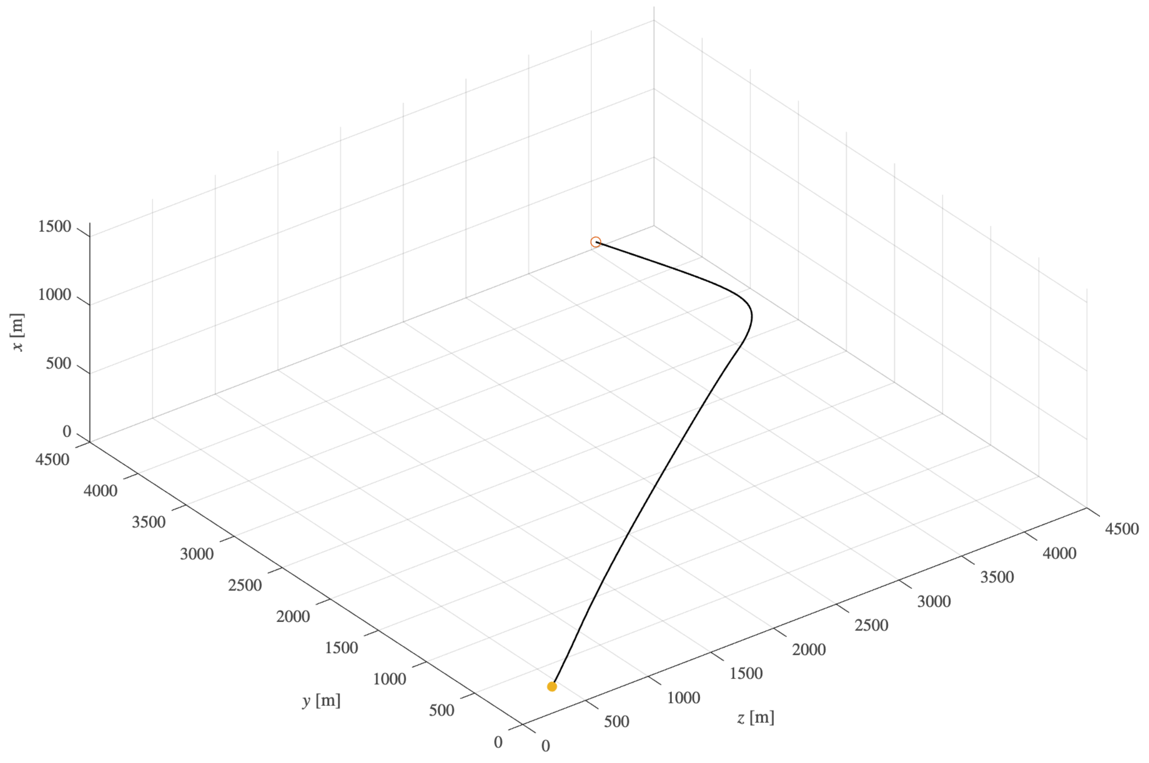


Figure 3: 3D view of optimal trajectory for minimum-landing-error case.

Visibly, the vehicle's final touchdown point is not at the origin, instead a distance of about 550 meters away. This shows that the optimal control formulation is successful in guiding the vehicle to a safe landing nearby despite the infeasibility of the commanded target. With the much further initial condition (adding 2000 meters along the y-axis and 500 along the z-axis), the vehicle shows great cross-range capability. Moving to the overview of the trajectory in Fig. 4 allows further analysis.

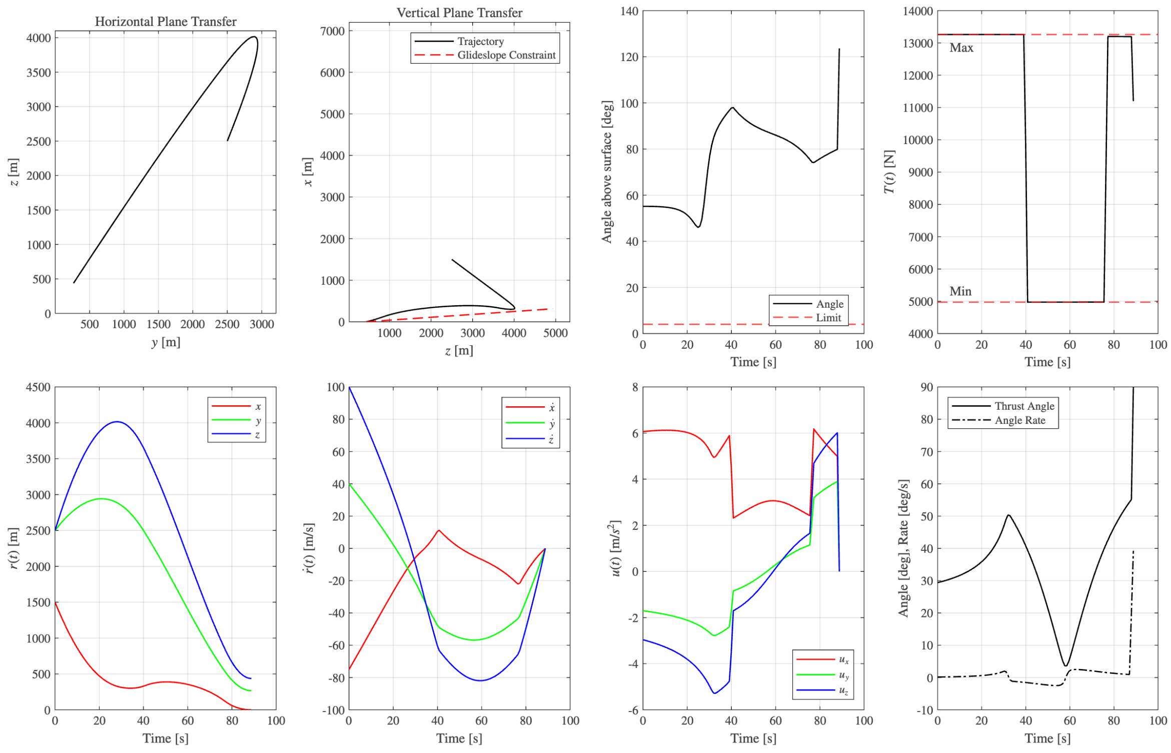


Figure 4: Overview of flight profile for minimum-landing-error case.

The view of the horizontal plane shows that the vehicle travels a maximum distance of about 4500 meters, even with the limited altitude to maneuver. The trajectory in the vertical plane again skims along the glideslope, showing this to likely be the most efficient path. The angle above surface does not hit the limit as before, instead staying further up in the range. This indicates that the vehicle is using more thrust to stay above the glideslope, as it maintains a minimum separation from the surface. The thrust profile again shows the *max-min-max* structure. Then, the position, velocity, commanded acceleration, and thrust angle/rate plots paint a similar picture. While the results for this case cannot be directly compared to the original paper, the similar characteristics of the trajectory show that the implementation is largely accurate, despite the presented discrepancy.

Appendix

```
clear; close all; clc;
```

The min and max thrust are defined for various uses.

```
rho1 = 4972; % [N]
rho2 = 13260; % [nN]

% Case 1
r0 = [1500 0 2000]';
rdot0 = [-75 0 100]';
rf = [0 0 0]';

% Case 2
% r0 = [1500 2500 2500]';
% rdot0 = [-75 40 100]';
% rf = [0 0 0]';
```

Next, the bounds for the golden section search are calculated, based on the initial altitude, velocity, and fuel-flow rate at minimum thrust. These are somewhat arbitrary and were found by experimentation, leading to quick convergence with a sufficiently large search space.

```
lb = 3 * (r0(1) / abs(rdot0(1))); % lower bound for optimal time search
ub = 2/3 * 400 / (4.53*10^-4 * rho1); % upper bound
tol = 5e-1; % max difference between x1 and x2 to iterate until

 % Case 1
[opt_t_f, optval, states, thrusts] = gss(lb, ub, tol, r0, rdot0, rf, 'Fuel');
```

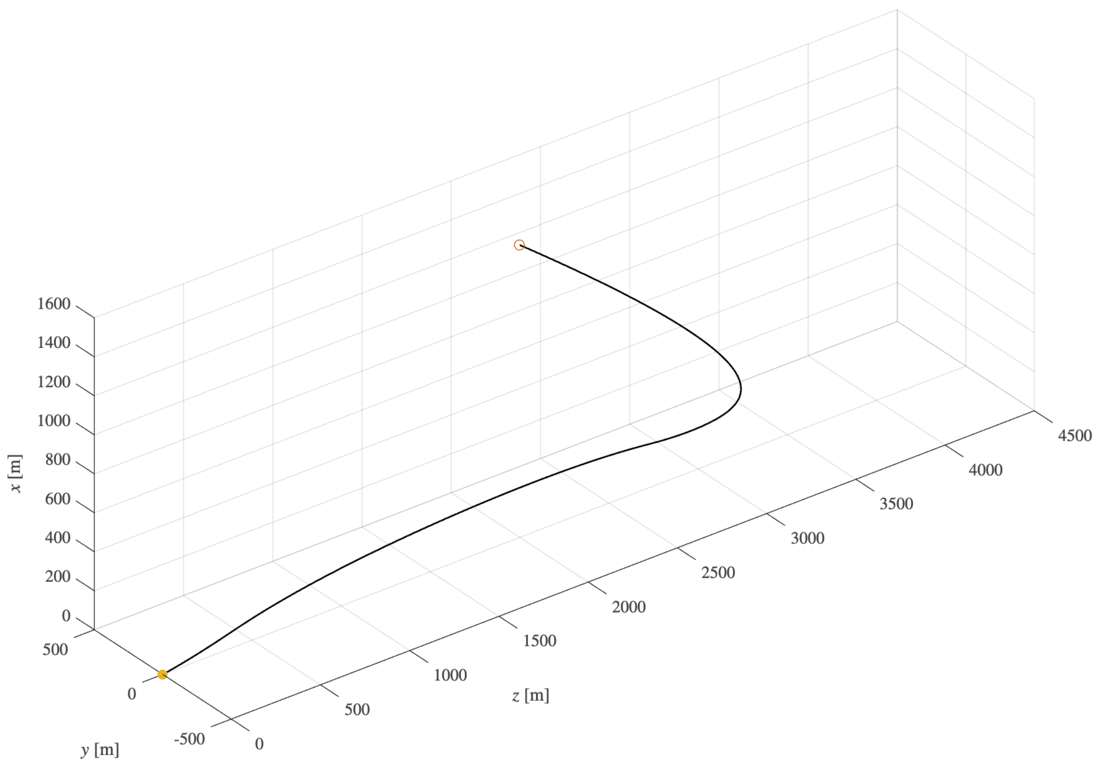
```
lb = 60.00, x1 = 96.09 (492.5347), x2 = 82.31 (465.5764), ub = 118.40
lb = 60.00, x1 = 82.31 (465.5764), x2 = 73.79 (468.8726), ub = 96.09
lb = 73.79, x1 = 87.57 (473.3540), x2 = 82.31 (465.5764), ub = 96.09
lb = 73.79, x1 = 82.31 (465.5764), x2 = 79.05 (463.5383), ub = 87.57
lb = 73.79, x1 = 79.05 (463.5383), x2 = 77.04 (463.9040), ub = 82.31
lb = 77.04, x1 = 80.29 (463.9944), x2 = 79.05 (463.5383), ub = 82.31
lb = 77.04, x1 = 79.05 (463.5383), x2 = 78.28 (463.4870), ub = 80.29
lb = 77.04, x1 = 78.28 (463.4870), x2 = 77.81 (463.5842), ub = 79.05
lb = 77.81, x1 = 78.58 (463.4894), x2 = 78.28 (463.4870), ub = 79.05
----------------------------------------------------------------------
Final Values
lb = 77.81, x1 = 78.28, x2 = 78.10, ub = 78.58
Optimal t_f: 78.19 [s], Optimal Value: 463.4933
```

```
% Case 2
% [opt_t_f, optval, states, thrusts] = gss(lb, ub, tol, r0, rdot0, rf, 'Landing Error');

delta_t = opt_t_f/size(states, 2);
N = size(states, 2) - 1;
tspan = 0:delta_t:N*delta_t;

x = states(1, :);
```

```
y = states(2, :);
z = states(3, :);
vx = states(4, :);
vy = states(5, :);
vz = states(6, :);
ux = thrusts(1, :);
uy = thrusts(2, :);
uz = thrusts(3, :);

gs_range = 0:1500:4500;

angle_above_surface = 90 + atan2d(vx, sqrt(vy.^2 + vz.^2));

thrust_mag = thrusts(4, :) .* exp(states(7, :));

thrust_angle = 90 - atan2d(ux, sqrt(uy.^2 + uz.^2));
thrust_angle_rate = gradient(thrust_angle, delta_t);
```

Below is a plot of the calculated optimal trajectory in 3D.

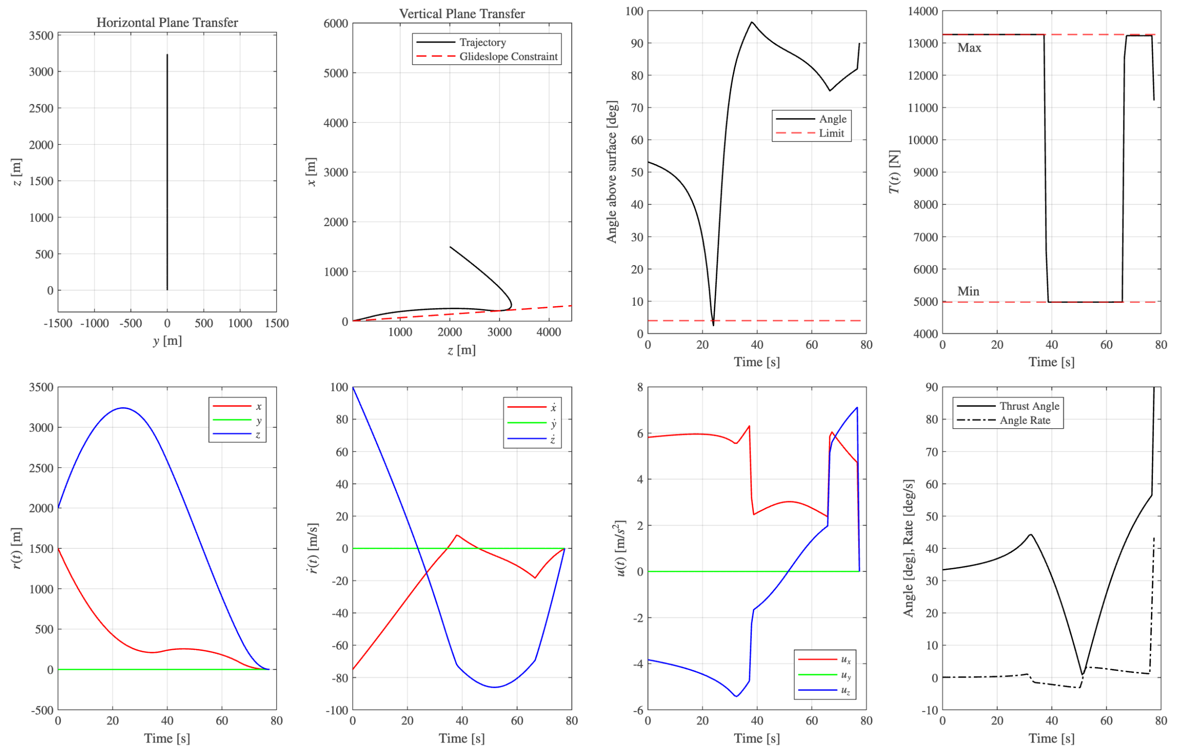
```
figure(1)
trajectory = tiledlayout(1, 1, "TileSpacing", "tight", "Padding", "tight");
set(gcf, 'Position', [100 100 1000 600])

nexttile
plot3(states(3,:), states(2,:), states(1,:), 'k', 'LineWidth', 1)
hold on; grid on; axis equal;
plot3(r0(3), r0(2), r0(1), 'o', 'MarkerSize', 6)
plot3(states(3, end), states(2, end), states(1, end), '.', 'MarkerSize', 18)
xlabel('$z$ [m]'); ylabel('$y$ [m]'); zlabel('$x$ [m]');
xlim([0 4500]); ylim([-500 500]); zlim([0 1600]);
```



The following plots are created to match the presented results from the paper.

```
figure(2)
group_plot = tiledlayout(2, 4, "TileSpacing", "compact", "Padding", "compact");
set(gcf, "Position", [100 100 1300 800]);

nexttile
plot(y, z, 'k', 'LineWidth', 1);
grid on; axis equal; box on;
xlabel('$y$ [m]'); ylabel('$z$ [m]');
% ylim([0 4100]);
xlim([-1500 1500]);
title('Horizontal Plane Transfer');

nexttile
plot(z, x, 'k', 'LineWidth', 1);
hold on; grid on; axis equal; box on;
plot(gs_range + z(end), sin(deg2rad(4)) * gs_range, 'r--', 'LineWidth', 1);
xlabel('$z$ [m]'); ylabel('$x$ [m]');
ylim([0 6000]);
legend('Trajectory', 'Glideslope Constraint', 'Location', 'northeast');
title('Vertical Plane Transfer');

nexttile
hold on; grid on; box on;
plot(tspan, angle_above_surface, 'k', 'LineWidth', 1);
yline(4, 'r--', 'LineWidth', 1);
ylabel('Angle above surface [deg]'); xlabel('Time [s]');
legend('Angle', 'Limit', 'Location', 'best');
```

```
nexttile
plot(tspan, thrust_mag, 'black', 'LineWidth', 1);
hold on; grid on; box on;
yline(rho1, 'r--', 'LineWidth', 1); yline(rho2, 'r--', 'LineWidth', 1);
text(5, 1.07 * rho1, 'Min'); text(5, 0.97 * rho2, 'Max');
xlabel('Time [s]'); ylabel('$T(t)$ [N]');

nexttile
hold on; grid on; box on;
plot(tspan, x, 'r', 'LineWidth', 1);
plot(tspan, y, 'g', 'LineWidth', 1);
plot(tspan, z, 'b', 'LineWidth', 1);
xlabel('Time [s]'); ylabel('$r(t)$ [m]');
legend('$x$', '$y$', '$z$', 'Location', 'northeast');

nexttile
hold on; grid on; box on;
plot(tspan, vx, 'r', 'LineWidth', 1);
plot(tspan, vy, 'g', 'LineWidth', 1);
plot(tspan, vz, 'b', 'LineWidth', 1);
xlabel('Time [s]'); ylabel('$\dot{r}(t)$ [m/s]');
legend('$\dot{x}$', '$\dot{y}$', '$\dot{z}$', 'Location', 'northeast');

nexttile
hold on; grid on; box on;
plot(tspan, ux, 'r', 'LineWidth', 1);
plot(tspan, uy, 'g', 'LineWidth', 1);
plot(tspan, uz, 'b',  'LineWidth', 1);
xlabel('Time [s]'); ylabel('$u(t)$ [m/$s^2$]');
legend('$u_x$', '$u_y$', '$u_z$', 'Location', 'southeast');

nexttile
hold on; grid on; box on;
plot(tspan, thrust_angle, 'k', 'LineWidth', 1);
plot(tspan, thrust_angle_rate, 'k-.', 'LineWidth', 1);
xlabel('Time [s]'); ylabel('Angle [deg], Rate [deg/s]');
legend('Thrust Angle', 'Angle Rate', 'Location', 'northwest');
```



This function implements the golden section search used to solve for the optimal value. The lower/upper bounds, tolerance, and initial/final conditions are passed as input, along with a flag specifying whether to minimize fuel use or landing error. The golden section search uses the golden ratio (~0.618) to pick evaluation points, and then compares these values to set the new lower/upper bound for the next iteration. If CVX returns a 'Failed' or 'Infeasible' result, the optimal value is set much higher (around three orders of magnitude higher than a real expected value) to close off points higher/lower than the infeasible point. 'Inaccurate' solutions are allowed, and in testing, the optimal time never returned an inaccurate solution.

```
function [opt_t_f, optval, states, thrusts] = gss(lb, ub, tol, r0, rdot0, rf, flag)
    gr = (sqrt(5) - 1)/2;
    d = gr * (ub - lb);

    x1 = lb + d;
    x2 = ub - d;

    while d > tol
        [x1_status, x1_optval, ~] = powered_descent(r0, rdot0, rf, x1, flag);
        [x2_status, x2_optval, ~] = powered_descent(r0, rdot0, rf, x2, flag);

        if ~strcmp(x1_status, 'Solved')
            if strcmp(x1_status, 'Failed') || strcmp(x1_status, 'Infeasible')
                x1_optval = 10^6;
            end
        end

        if ~strcmp(x2_status, 'Solved')
            if strcmp(x2_status, 'Failed') || strcmp(x2_status, 'Infeasible')
                x2_optval = 10^6;
            end
        end

        formatSpec = 'lb = %.2f, x1 = %.2f (%.4f), x2 = %.2f (%.4f), ub = %.2f\n';
        fprintf(formatSpec, lb, x1, x1_optval, x2, x2_optval, ub);

        if x1_optval < x2_optval
            lb = x2;
            d = gr * (ub - lb);
            x2 = x1;
            x1 = lb + d;
        elseif x1_optval > x2_optval
            ub = x1;
            d = gr * (ub - lb);
            x1 = x2;
            x2 = ub - d;
        end

        opt_t_f = (x1 + x2)/2;
    end

    fprintf('----------------------------------------------------------------------\n')
    fprintf('Final Values\n');
    formatSpec = 'lb = %.2f, x1 = %.2f, x2 = %.2f, ub = %.2f\n';
    fprintf(formatSpec, lb, x1, x2, ub);

    [~, optval, states, thrusts] = powered_descent(r0, rdot0, rf, opt_t_f, flag);
    fprintf('Optimal t_f: %.2f [s], Optimal Value: %.4f', opt_t_f, optval);
end
```

This function implements the optimal control problem formulated in the two papers. Inputs are initial conditions, final position and time, and the solver flag. Basic problem parameters are initialized, then the discrete A and B matrices are found. The ξ and Ψ matrices are created in a batched form, as required by CVX, and finally the S and c matrices as well. Instead of the matrices used for indexing in the original problem formulation, simple MATLAB array indexing is used.

```
function [status, optval, states, thrusts] = powered_descent(r0, rdot0, rf, t_f, flag)
    g = [-3.7114 0 0]'; % [m/s^]

    m_wet = 1905; % [kg]
    m_dry = 1505;

    burn_rate = 4.53*10^-4; % [s/m]

    rho1 = 4972; % [N]
    rho2 = 13260; % [N]

    y0 = [r0; rdot0; log(m_wet)];

    glideslope_angle_bound = deg2rad(86);

    N = 100; % [steps]
    delta_t = t_f/N; % [s]
```

```
A_c = [zeros(3,3) eye(3) zeros(3,1); zeros(4,7)];

B_c = [zeros(3,4); eye(3), zeros(3,1); zeros(1,3) -burn_rate];

[A_d, B_d] = c2d(A_c, B_c, delta_t);

[xi_batched, Psi_batched] = build_batch_matrices(A_d, B_d, N, y0, g);

S = [zeros(2, 1), eye(2)];
c = [-tan(glideslope_angle_bound); zeros(2, 1)];

omega = zeros(1, 4*N);
for M = 4:4:4*N
    omega(1, M) = delta_t;
end

k = (0:N)';

%z0, mu_1, and mu_2 are found ahead of time
z0 = log(m_wet - rho2 * burn_rate * delta_t * k);
mu_1 = rho1 * exp(-z0);
mu_2 = rho2 * exp(-z0);

cvx_precision best % ill-conditioned otherwise
cvx_begin quiet
    variable eta(4*(N+1), 1)

    u_matrix = reshape(eta, 4, N+1);
    u_k = u_matrix(1:3, :);
    sigma_k = u_matrix(4, :)';

    y_k = [y0 reshape(xi_batched(:) + Psi_batched * eta, 7, N)];

    y_N = y_k(:, end);

    z_k = y_k(7, :)';

    % choice of objectives/final conditions
    if strcmp(flag, 'Fuel')
        minimize omega*eta(1:4*N, 1)
        subject to
            y_N(1:6) == [rf; zeros(3, 1)];

        pos_error = y_k(1:3, :) - repmat(rf, 1, N+1);

    elseif strcmp(flag, 'Landing Error')
        minimize norm(y_N(1:3))
        subject to
            y_N(1) == 0;
            y_N(4:6) == 0;
            pos_error = y_k(1:3, :) - repmat(y_N(1:3), 1, N+1);
        end

        % constraints present in both problems
        subject to
            y_N(7) >= log(m_dry);

            norms(u_k, 2, 1)' <= sigma_k;

            mu_1 .* (1 - (z_k - z0) + 0.5 * (z_k - z0).^2) ...
                <= sigma_k <= mu_2 .* (1 - (z_k - z0));

            log(m_wet - rho2 * burn_rate * delta_t * k) ...
                <= z_k <= log(m_wet - rho1 * burn_rate * delta_t * k);

            norms(S * pos_error, 2, 1) + c' * pos_error <= 0;
    cvx_end

    status = cvx_status;
    optval = cvx_optval;
    thrusts = reshape(eta, 4, N+1);
    states = [y0 reshape(xi_batched(:) + Psi_batched * eta, 7, N)];
end
```

This function creates the batched ξ and Ψ matrices used to propagate the dynamics, their definition comes from the zero-order hold discretization the paper uses.

```
function [xi_batched, Psi_batched] = build_batch_matrices(A, B, N, y0, g)
    xi_batched = zeros(7, N);
    Psi_batched = zeros(7*N, 4*(N+1));

    current_Lambda = B;

    for k = 1:N
        xi_batched(:, k) = A^k * y0 + current_Lambda * [g; 0];

        if k < N
            current_Lambda = A * current_Lambda + B;
        end

        % Psi is calculated recursively for efficiency
        if k == 1
            Psi_batched(1:7, 1:4) = B;
        else
            Psi_batched(7*k-6:7*k, :) = A * Psi_batched(7*(k-1)-6:7*(k-1), :);
            Psi_batched(7*k-6:7*k, 4*k-3:4*k) = B;
        end
    end
end
```

References

[1] Behcet Acikmese and Scott R. Ploen. "Convex Programming Approach to Powered Descent Guidance for Mars Landing". In: *Journal of Guidance, Control, and Dynamics* 30.5 (Sept. 2007), pp. 1353–1366. ISSN: 0731-5090, 1533-3884. DOI: 10.2514/1.27553. (Visited on 11/26/2025).

[2] Behcet Acıkmese et al. "FLIGHT TESTING OF TRAJECTORIES COMPUTED BY G-FOLD: FUEL OPTIMAL LARGE DIVERT GUIDANCE ALGORITHM FOR PLANETARY LANDING". In: ().

[3] Lars Blackmore, Behçet Açikmeşe, and Daniel P. Scharf. "Minimum-Landing-Error Powered-Descent Guidance for Mars Landing Using Convex Optimization". In: *Journal of Guidance, Control, and Dynamics* 33.4 (July 2010), pp. 1161–1171. ISSN: 0731-5090, 1533-3884. DOI: 10.2514/1.47202. (Visited on 11/26/2025).

[4] Michael Szmuk, Behcet Acikmese, and Andrew W. Berning. "Successive Convexification for Fuel-Optimal Powered Landing with Aerodynamic Drag and Non-Convex Constraints". In: *AIAA Guidance, Navigation, and Control Conference*. San Diego, California, USA: American Institute of Aeronautics and Astronautics, Jan. 2016. ISBN: 978-1-62410-389-6. DOI: 10.2514/6.2016-0378. (Visited on 11/26/2025).