Replication of Convexified Soft Landing Optimal Control by Açıkmeşe et al. and Blackmore et al.

Jeff Cui

To my friend Chao, reaching for the neutron stars.

December 20, 2018

1 Introduction

This final project extends Chao Ju's final project last year [3], an implementation of a reduced form of the convexified soft landing optimal control by Açıkmeşe et al. [1] and Blackmore et al. [2] Building on his project, here I implement the original paper.

The problem is to soft-land a chemically propelled vehicle, abstracted as a point mass with a mass flow $\dot{m}$ dependent on the throttle, on a planetary surface with given landing target coordinates $\boldsymbol{q}$. The problem is to first minimize the landing error, and then minimize fuel consumption, under nonconvex constraints that are convexified to allow for convex optimization. [1] [2] further prove that the convexification is lossless in terms of optimality.

1.1 Extending Chao's Final Project

Chao's final project implemented a reduced form of the problem. First, the mass flow $\dot{m}$, representing fuel consumption, is not considered. Second, nonconvex thrust constraints $\rho_1 \leq \|\boldsymbol{T}_c(t)\| \leq \rho_2$ $(0 \leq \rho_1 < \rho_2 \leq T_{\max})$, representing engine throttle constraints, are not considered. In this final project, I build on Chao's work and implement these constraints in the original papers as well.

2 Problem Formulation by Açıkmeşe et al. and Blackmore et al.

The problem is formulated in [1] [2], explained below to show understanding.

2.1 Variables

The vehicle is treated as a point mass to avoid dealing with attitude dynamics.

$\boldsymbol{x} = [\boldsymbol{r}(t), \dot{\boldsymbol{r}}(t)]$ is the state vector of the vehicle, where $\boldsymbol{r}(t)$ is the position.
$m(t)$ is the mass of the vehicle.
θ is the maximum allowed angle between the thrust and the skyward direction.
γ is the maximum glideslope constraint during flight.
$\boldsymbol{\omega}$ is the planet's angular velocity. $\boldsymbol{g}$ is the gravitational acceleration.
ρ_1 is the throttled thrust lower bound, ρ_2 the upper bound, and $T_{\max}$ the maximum thrust at full throttle; $\boldsymbol{T}_c(t)$ is the thrust. $0 \leq \rho_1 \leq \boldsymbol{T}_c(t) \leq \rho_2 \leq T_{\max}$.
$\alpha > 0$ is the specific fuel consumption (kg/(N*s)).
t_f is the total time of flight.
$E = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ such that $E\boldsymbol{r}(t)$ is the ground track at time t.

2.2 Dynamics

We need to model the translational dynamics and the mass flow.

For the state vector $\boldsymbol{x}$, we have

$$\dot{x}(t) = A(\boldsymbol{\omega})\boldsymbol{x}(t) + B(\boldsymbol{g} + \frac{\boldsymbol{T}_c(t)}{m(t)}) \tag{1}$$

$$A = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ -S(\boldsymbol{\omega})^2 & -2S(\boldsymbol{\omega}) \end{bmatrix}, S = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}, B = \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix} \tag{2}$$

$A(\boldsymbol{\omega})$ models the state change due to Coriolis effects. Chao has given a very thorough explanation in his final project [3], which I will refrain from belabouring the point. B maps to the $\dot{\boldsymbol{r}}(t)$ part in $\boldsymbol{x}$. Combined, they model the change in vehicle position and velocity on a rotating planetary surface reference frame.

The mass flow is modeled proportional to the thrust, as

$$\dot{m}(t) = -\alpha\|\boldsymbol{T}_c(t)\| \tag{3}$$

The dynamics are nonlinear due to mass flow and leads to nonconvex constraints. [1] linearizes it by taking $\ln m$ and changing the variables accordingly. This will be described later as well.

2.3 Nonconvex Formulation

2.3.1 Nonconvex Minimal Landing Error Problem

$$\min_{t_f, \boldsymbol{T}_c} \|E\boldsymbol{r}(t_f) - \boldsymbol{q}\| \tag{4}$$

(t_f is the time of flight, $E = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ is the ground track matrix, $\boldsymbol{q} \in \mathbb{R}^2$ the landing target.)

subject to: (1) (3) $\forall t \in [0, t_f]$ (the dynamics equations above), and

$$\boldsymbol{x}(t) \in \boldsymbol{X} \quad \forall t \in [0, t_f] \quad \text{(Flight envelope)} \tag{5}$$

$$0 < \rho_1 \leq \|\boldsymbol{T}_c(t)\| < \rho_2, \quad \hat{\boldsymbol{n}}^T\boldsymbol{T}_c(t) \geq \|\boldsymbol{T}_c(t)\|\cos\theta \quad \text{(Thrust constraints)} \tag{6}$$

($\hat{\boldsymbol{n}}$ is the skyward unit vector $\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$.)

$$m(0) = m_0, \quad m(t_f) \geq m_0 - m_f > 0 \quad \text{(Initial mass; fuel constraint)} \tag{7}$$

(m_0 is the initial wet mass; m_f is the fuel mass. Fuel must not be depleted at termination.)

$$\boldsymbol{r}(0) = \boldsymbol{r}_0, \dot{\boldsymbol{r}}(0) = \dot{\boldsymbol{r}}_0 \quad \text{(Initial state)} \tag{8}$$

$$\hat{\boldsymbol{n}}^T \boldsymbol{r}(t_f) = 0, \dot{\boldsymbol{r}}(t_f) = \boldsymbol{0} \quad \text{(Terminal state: soft landing)} \tag{9}$$

In [1], the optimal final position on the ground is denoted $\boldsymbol{d}^*_{P1}$. ($\boldsymbol{d}^*_{P1} \in \mathbb{R}^2$) This is used later in the minimum fuel problem.

2.3.2 Nonconvex Minimal Fuel Problem

With $\boldsymbol{d}^*_{P1}$ above,

$$\min_{t_f, \boldsymbol{T}_c} \int_0^{t_f} \alpha \|\boldsymbol{T}_c(t)\| dt \tag{10}$$

subject to (1) (3) (5) (6) (7) (8) (9) above, and:

$$\|E\boldsymbol{r}(t_f) - \boldsymbol{q}\| \leq \|\boldsymbol{d}^*_{P1} - \boldsymbol{q}\| \quad \text{(Optimal landing error)} \tag{11}$$

Note that this constraint itself is convex already, but the other constraints are not. [1] and [2] has convexified the landing error constraint: instead of constraining for an exact position or an exact error, it is formulated such that the vehicle can land anywhere within the optimal error radius, forming a closed disk. This does not alter optimality as all the optimal solutions will still be on the boundary, but convexifies the constraint.

2.4 Convexification of Thrust Constraints

[1] and [2] convexified the thrust constraints by introducing a slack variable $\Gamma(t)$ such that $\rho_1 \leq \Gamma(t) \leq \rho_2$. This is a segment and, therefore, convex. Then, the convex thrust upper bound $\|\boldsymbol{T}_c(t)\| \leq \Gamma(t)$ is straightforward. Next, [1] comes up with a beautiful(!) constraint, $\hat{\boldsymbol{n}}^T \boldsymbol{T}_c(t) \geq \cos\theta\ \Gamma(t)$, convexifying the thrust pointing constraint and the thrust lower bound.

2.4.1 Convex Relaxed Minimum Landing Error Problem

$$\min_{t_f, \boldsymbol{T}_c, \Gamma} \|E\boldsymbol{r}(t_f) - \boldsymbol{q}\| \tag{12}$$

subject to (1) (5) (7) (8) (9), and:

$$\dot{m}(t) = -\alpha\Gamma(t) \quad \forall t \in [0, t_f] \quad \text{(Mass flow)} \tag{13}$$

$$\|\boldsymbol{T}_c(t)\| \leq \Gamma(t),\ 0 < \rho_1 \leq \Gamma(t) \leq \rho_2,\ \hat{\boldsymbol{n}}^T \boldsymbol{T}_c(t) \geq \cos\theta\ \Gamma(t) \quad \text{(Thrust)} \tag{14}$$

In [1], the optimal final position on the ground is denoted $\boldsymbol{d}^*_{P3}$. ($\boldsymbol{d}^*_{P3} \in \mathbb{R}^2$) This is used later in the convexified minimum fuel problem.

2.4.2 Convex Relaxed Minimum Fuel Problem

$$\min_{t_f, \boldsymbol{T}_c, \Gamma} \int \Gamma(t) dt \tag{15}$$

subject to (1) (5) (7) (8) (9) (13) (14), and:

$$\|E\boldsymbol{r}(t_f) - \boldsymbol{q}\| \leq \|\boldsymbol{d}^*_{P3} - \boldsymbol{q}\| \quad \text{(Optimal landing error)} \tag{16}$$

2.5 Convexification of Nonlinear Mass Flow

As discussed above, the dynamics are nonlinear due to mass flow and leads to nonconvex constraints. Açıkmeşe et al. [1] linearizes it in Appendix A as follows:

$$\sigma := \frac{\Gamma}{m}, \boldsymbol{u} := \frac{\boldsymbol{T}_c}{m}, z := \ln m. \tag{17}$$

2.5.1 Convex Relaxed Minimum Landing Error Problem, Changed Variables

$$\min_{t_f, \boldsymbol{u}, \sigma} \|E\boldsymbol{r}(t_f) - \boldsymbol{q}\| \tag{18}$$

subject to (5) (8) (9), and:

$$\dot{\boldsymbol{x}}(t) = A(\boldsymbol{\omega})\boldsymbol{x}(t) + B(\boldsymbol{g} + \boldsymbol{u})\ \forall t \in [0, t_f] \tag{19}$$

$$\dot{z}(t) = -\alpha\sigma(t)\ \forall t \in [0, t_f] \tag{20}$$

And changed variable mass (z) boundary conditions:

$$z_0 = \ln m_0, z_f = \ln m_f, z(t_f) = z_0 + \int_0^{t_f} -\alpha\sigma(t)dt \geq \ln(m_0 - m_f) > 0 \tag{21}$$

And changed thrust (u, σ) constraints:

$$\|\boldsymbol{u}(t)\| \leq \sigma(t),\ \rho_1 e^{-z(t)} \leq \sigma(t) \leq \rho_2 e^{-z(t)},\ \hat{\boldsymbol{n}}^T \boldsymbol{u}(t) \geq \cos\theta\ \sigma(t) \tag{22}$$

Write $\rho_1 e^{-z(t)} \leq \sigma(t) \leq \rho_2 e^{-z(t)}$ as a second-order cone for optimization:

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

Where $z_0(t) = \ln(m_0 - \alpha\rho_2 t)$. [1]

The cone is centered around $t = 0$. It is a good enough approximation for model predictive control. At each frame, we treat the current frame as $t = 0$, optimize, and only use the first (or the first few) control outputs; therefore, the error on the σ will be bounded and the constraints satisfied.

Here we denote the optimal final position on the ground $\boldsymbol{d}^*_{P5}$. ($\boldsymbol{d}^*_{P5} \in \mathbb{R}^2$) This is used later in the minimum fuel problem.

2.5.2 Convex Relaxed Minimum Fuel Problem, Changed Variables

$$\min_{t_f, \boldsymbol{u}, \sigma} \int_0^{t_f} \sigma(t) dt \tag{23}$$

Subject to (5) (8) (9) (19) (20) (21) (22), and

$$\|E\boldsymbol{r}(t_f) - \boldsymbol{q}\| \leq \|\boldsymbol{d}^*_{P5} - \boldsymbol{q}\| \tag{24}$$

3 Implementation

My implementation uses a straightforward discretization with $dt = 1$. Flight parameters are provided by [1] for a simulated Mars soft landing. See 5.1.

A model predictive controller is implemented. At each frame, the optimal control and the current frame is solved and saved. Then, a new optimization is started on the next frame predicted by the dynamics. For the optimizer code, see 5.2. It is also available at `https://jeffcui.com/assignments/cs164/final/optimizer.py`.

3.1 Possible Extensions

The original paper by Blackmore et al. used a radial basis discretization [2]. This was not implemented.

I also did not implement a line search of optimal flight time, detailed in [2]. However, I did replicate its experimental conclusion that the minimal fuel consumption is monotonically dependent on flight time in 4.1.

Additionally, the flight envelope $\boldsymbol{X}$ is only defined by the glideslope constraint in my implementation. I have omitted speed constraints from my implementation.

4 Results

The vehicle soft landed and remained in constraints. See Figure 1. Please watch the animation at `https://jeffcui.com/assignments/cs164/final/landing.mp4`. Snapshots can be found in 5.3.

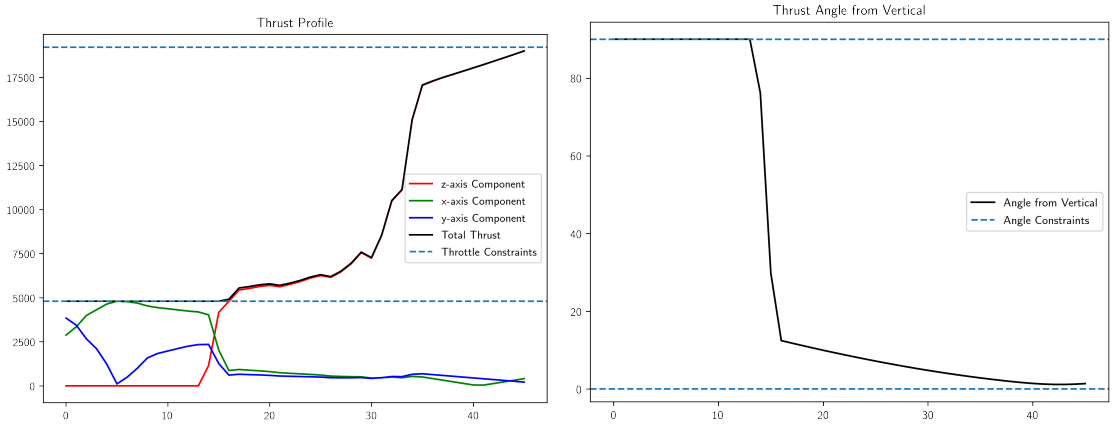

Figure 1: Thrust profile and angle from vertical.

4.1 Effect of Flight Time on Fuel Usage

Blackmore et al. discovered that experimentally, minimal fuel consumption is monotonically dependent on flight time, justifying line search as a practical approach for finding the optimal flight time. [2] Here, I replicate this experiment with $t_f \in [45, 59]$. $t = 45$ is infeasible given the set of constraints and flight parameters. See Figure 2.

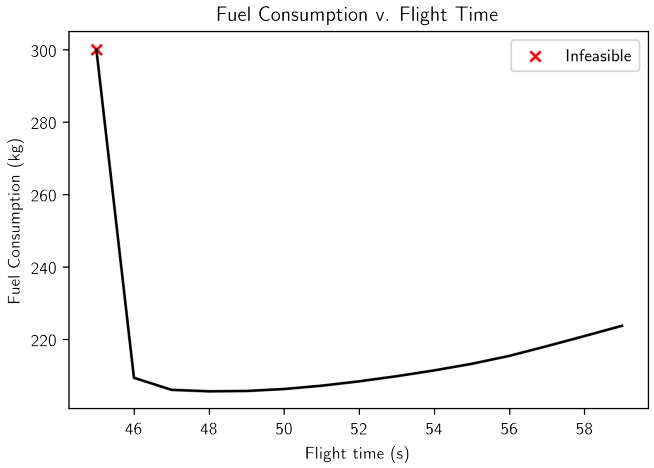

Figure 2: Fuel v. Flight Time.

References

[1] Behçet Açıkmeşe, John M Carson, and Lars Blackmore. Lossless convexification of nonconvex control bound and pointing constraints of the soft landing optimal control problem. *IEEE Transactions on Control Systems Technology*, 21(6):2104–2113, 2013.

[2] Lars Blackmore, Behcet Acikmese, and Daniel P Scharf. Minimum-landing-error powered-descent guidance for mars landing using convex optimization. *Journal of guidance, control, and dynamics*, 33(4):1161–1171, 2010.

[3] Chao Ju. Cs164 final project: Soft landing on mars, a 3d convex problem. *CS164 Final Project*, 2017.

5 Appendix

5.1 Flight Parameters

Parameter	Value	Unit
$\boldsymbol{g}$	$(-3.71, 0, 0)^T$	m/s^2
$\boldsymbol{\omega}$	$10^{-5} \times (2.53, 0, 6.62)^T$	rad/s
$\boldsymbol{x}_0$	$(2400, 450, -330, -10, -40, 10)^T$	m, m, m, m/s, m/s, m/s
m_0	2000	kg
m_f	300	kg
$T_{\max}$	24000	N
ρ_1	$0.2\ T_{\max} = 4800$	N
ρ_2	$0.8\ T_{\max} = 19200$	N
θ	90	deg
γ	75	deg
N	47	s

5.2 Optimizer Code

Raw: `https://jeffcui.com/assignments/cs164/final/optimizer.py`

```
import cvxpy as cp
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import rc
rc('text.latex', preamble=r'\usepackage{sfmath}')
rc('text', usetex=True)
```

```
def formulate_MLEP(x_0, m_0, m_f, q, t, dt):
    # number of discrete steps
    N = int(t / dt)

    # reparametrizations
    z_0 = np.log(m_0)
    log_empty_mass = np.log(m_0 - m_f)

    # precalculate e^{-z_0} for constraints
    e_neg_z_0 = np.exp(-z_0)

    # precalculate mass under max thrust for constraints
    # NOTE: z0(t) is logarithmic mass w.r.t. time
    #       under constant maximum thrust,
    #       whereas z_0 is log of initial mass.
    z0 = np.zeros((1, N))
    for i in range(N):
        z0[0, i] = np.log(m_0 - alpha * rho_2 * dt * i)

    # minimum landing error problem
    x = cp.Variable((6, N))
    z = cp.Variable((1, N))
    u = cp.Variable((3, N))
    sigma = cp.Variable((1, N))

    MLEP_objective = cp.Minimize(cp.norm(E*x[:3, N-1] - q))
    MLEP_constraints = [
        z[0, 0] == z_0,
        z[0, N-1] >= log_empty_mass,
        x[:,0] == x_0,
        e_1.T * x[:3,N-1] == 0,
        cp.norm(x[3:,N-1]) <= eps_final_velocity,
    ]

    for i in range(N-1):
        MLEP_constraints.append(
            x[:,i+1] == x[:,i] + (A*x[:,i] + B*(g+u[:,i])) * dt
        )
        MLEP_constraints.append(
            z[:,i+1] == z[:,i] - alpha * sigma[:,i]
        )
    MLEP_constraints += [
```

```
        cp.norm(u, axis=0) <= sigma[0,:],

        rho_1 * e_neg_z_0 * (
            1 - (z[0,:] - z0[0,:]) + (z[0,:] - z0[0,:])**2/2
        ) <= sigma[0,:],

        rho_2 * e_neg_z_0 * (
            1 - (z[0,:] - z0[0,:])
        ) >= sigma[0,:],

        x[0,:] >= cp.norm(x[:3,:], axis=0) * cos_glideslope,

        n_hat * u >= cos_theta * sigma[0,:],
    ]

    return (
        cp.Problem(MLEP_objective, MLEP_constraints),
        x, z, u, sigma
    )


def formulate_MFP(x_0, m_0, m_f, q, t, dt, optimal_pos):
    N = int(t / dt)

    # reparametrizations
    z_0 = np.log(m_0)
    log_empty_mass = np.log(m_0 - m_f)

    # precalculate e^{-z_0} for constraints
    e_neg_z_0 = np.exp(-z_0)

    # precalculate mass under max thrust for constraints
    # NOTE: z0(t) is logarithmic mass w.r.t. time
    #       under constant maximum thrust,
    #       whereas z_0 is log of initial mass.
    z0 = np.zeros((1, N))
    for i in range(N):
        z0[0, i] = np.log(m_0 - alpha * rho_2 * dt * i)

    # precalculate the minimum landing error norm
    optimal_pos_error = np.linalg.norm(optimal_pos - q)

    # minimum fuel problem
```

```
x = cp.Variable((6, N))
z = cp.Variable((1, N))
u = cp.Variable((3, N))
sigma = cp.Variable((1, N))

MFP_objective = cp.Minimize(cp.sum(sigma) * dt)

# with the same constraints as MLEP above
MFP_constraints = [
    z[0, 0] == z_0,
    z[0, N-1] >= log_empty_mass,
    x[:,0] == x_0,
    e_1.T * x[:3,N-1] == 0,
    cp.norm(x[3:,N-1]) <= eps_final_velocity,
]

for i in range(N-1):
    MFP_constraints.append(
        x[:,i+1] == x[:,i] + (A*x[:,i] + B*(g+u[:,i])) * dt
    )
    MFP_constraints.append(
        z[:,i+1] == z[:,i] - alpha * sigma[:,i]
    )

MFP_constraints += [
    cp.norm(u, axis=0) <= sigma[0,:],

    rho_1 * e_neg_z_0 * (
        1 - (z[0,:] - z0[0,:]) + (z[0,:] - z0[0,:])**2/2
    ) <= sigma[0,:],

    rho_2 * e_neg_z_0 * (
        1 - (z[0,:] - z0[0,:])
    ) >= sigma[0,:],

    x[0,:] >= cp.norm(x[:3,:], axis=0) * cos_glideslope,

    n_hat * u >= cos_theta * sigma[0,:],
]

# and the landing error constraints
MFP_constraints.append(
    cp.norm(E*x[:3, N-1] - q) <= optimal_pos_error
```

```
    )

    return (
        cp.Problem(MFP_objective, MFP_constraints),
        x, z, u, sigma
    )


def solve(x_0, m_0, m_f, q, t, dt):
    N = int(t / dt)

    MLEP_problem, x, z, u, sigma = formulate_MLEP(
        x_0, m_0, m_f, q, t, dt
    )
    MLEP_problem.solve()

    # denoted d_{p_3}^* in the paper
    # the minimum error position
    optimal_pos = x.value[1:3, N-1]

    MFP_problem, x, z, u, sigma = formulate_MFP(
        x_0, m_0, m_f, q, t, dt, optimal_pos
    )
    MFP_problem.solve()
    return x, z, u, sigma, MLEP_problem, MFP_problem


# --- PLANETARY CONSTANTS ---
# rotation angular velocity
omega = np.array([2.53e-5, 0, 6.62e-5]).T
# gravitational acceleration
g = np.array([-3.71, 0, 0]).T


# --- INIT ---
# angular velocity matrix
S = np.array([
    [       0, -omega[2],   omega[1]],
    [omega[2],         0,  -omega[0]],
    [omega[1], omega[0],           0],
])

# matrix A, fictitious force
```

```
A = np.block([
    [np.zeros((3, 3)), np.eye(3)],
    [           -S**2,       -2*S],
])

# matrix B, velocity getter
B = np.block([
    [np.zeros((3, 3))],
    [np.eye(3)],
])

# unit vectors
e_1 = np.array([1,0,0]).T
e_2 = np.array([0,1,0]).T
e_3 = np.array([0,0,1]).T

# skyward
n_hat = e_1

# matrix E, ground projection
E = np.array([
    e_2.T,
    e_3.T,
])


# --- VEHICLE ---
# max thrust
T_max = 24000
# thrust upper bound
rho_2 = 0.8 * T_max
# thrust lower bound
rho_1 = 0.2 * T_max
# fuel consumption rate
alpha = 5e-4
# initial mass
m_0 = 2000
# fuel mass
m_f = 300
# initial state
x_0 = np.array([2400, 450, -330, -10, -40, 10]).T
```

```
# --- OPTIMIZATION PARAMS ---
dt = 1

# thrust envelope
theta = np.deg2rad(90)

# glideslope envelope
glideslope = np.deg2rad(75)

# precalculate cos values
cos_theta = np.cos(theta)
cos_glideslope = np.cos(glideslope)

eps_final_velocity = 1e-2

# flight time
t = 47

# landing target coordinates
q = np.array([0, 0])

# ------- OPTIMIZATION -------

m_1 = np.nan
delta_m = np.nan

# i: time elapsed since start of simulation
history = []
elapsed_time = np.linspace(0, t-2*dt, int((t-1)/dt))
for i in elapsed_time:
    x, z, u, sigma, MLEP_problem, MFP_problem = solve(
        x_0, m_0, m_f, q, t - i, dt
    )

    history.append({
        'x': x.value,
        'z': z.value,
        'u': u.value,
        'sigma': sigma.value,
    })

    # proceed to next time point
    x_0 = x.value[:, 1]
```

```
        m_1 = np.exp(z.value[0, 1])
        delta_m = m_1 - m_0
        m_0 += delta_m
        m_f += delta_m

        print('######## t = %.2f ########' % (i+1))
        print('r\t%.2f\t%.2f\t%.2f' % tuple(x_0[:3]))
        print('v\t%.2f\t%.2f\t%.2f' % tuple(x_0[3:]))
        print('a\t%.2f\t%.2f\t%.2f' % tuple(u.value[:,0]))
        print('m\t%.2f' % m_0)
        print('dm\t%.2f' % delta_m)

# ------- PLOTTING -------
history_clean = {}
for key in history[0].keys():
    history_clean[key] = []
    for i in range(len(history)):
        history_clean[key].append(history[i][key][:,0])
    history_clean[key] = np.stack(history_clean[key])

trajectory = history_clean['x'][:,:3]
velocity = history_clean['x'][:,3:]
zs, xs, ys = trajectory.T

m = np.exp(history_clean['z'])
T_c = history_clean['u'] * m
T_c_norm = np.linalg.norm(T_c, axis=1)
T_c_angle_from_vertical = np.arccos(T_c[:,0] / T_c_norm)

T_c_zs, T_c_xs, T_c_ys = T_c.T
T_c_rearranged = np.array([
    T_c_xs,
    T_c_ys,
    T_c_zs,
]).T

# thrust length per newton
T_c_plot_length_per_newton = 0.04
T_c_rearranged *= T_c_plot_length_per_newton
```

```
# ------- PLOT TRAIL -------
fig = plt.figure(dpi=300, figsize=(12, 4))
ax = fig.add_subplot(121, projection='3d')
ax.set_aspect('equal')

ax.plot([450-1200,450+1200],[0,0],[0,0], color='r', alpha=0)
ax.plot([0,0],[-330-1200,-330+1200],[0,0], color='b', alpha=0)
ax.plot([0,0],[0,0],[0,2400], color='g', alpha=0)

ax.view_init(elev=15., azim=120)
ax.plot(xs, ys, zs)
for i in range(len(xs)):
    ax.plot(
        [xs[i], xs[i]],
        [ys[i], ys[i]],
        [0, zs[i]],
        linewidth=0.2,
        color='b',
    )
for i in range(len(xs)):
    ax.plot(
        [xs[i], xs[i] - T_c_rearranged[i][0]],
        [ys[i], ys[i] - T_c_rearranged[i][1]],
        [zs[i], zs[i] - T_c_rearranged[i][2]],
        linewidth=0.5,
        color='r',
    )
ax.scatter(xs, ys, np.zeros(len(xs)), s=1)

ax = fig.add_subplot(122, projection='3d')
ax.set_aspect('equal')

ax.plot([450-1200,450+1200],[0,0],[0,0], color='r', alpha=0)
ax.plot([0,0],[-330-1200,-330+1200],[0,0], color='b', alpha=0)
ax.plot([0,0],[0,0],[0,2400], color='g', alpha=0)

ax.view_init(elev=15., azim=240)
ax.plot(xs, ys, zs)
for i in range(len(xs)):
    ax.plot(
        [xs[i], xs[i]],
        [ys[i], ys[i]],
        [0, zs[i]],
```

```
        linewidth=0.2,
        color='b',
    )
for i in range(len(xs)):
    ax.plot(
        [xs[i], xs[i] - T_c_rearranged[i][0]],
        [ys[i], ys[i] - T_c_rearranged[i][1]],
        [zs[i], zs[i] - T_c_rearranged[i][2]],
        linewidth=0.5,
        color='r',
    )
ax.scatter(xs, ys, np.zeros(len(xs)), s=1)
plt.show()


# ------- PLOT THRUST PROFILE -------
plt.figure(figsize=(8,6), dpi=300)
plt.title('Thrust Profile')
plt.plot(elapsed_time, np.abs(T_c_zs),
        color='r', label='z-axis Component')
plt.plot(elapsed_time, np.abs(T_c_xs),
        color='g', label='x-axis Component')
plt.plot(elapsed_time, np.abs(T_c_ys),
        color='b', label='y-axis Component')
plt.plot(elapsed_time, T_c_norm,
        color='k', label='Total Thrust')
plt.axhline(rho_1, linestyle='--',
        label='Throttle Constraints')
plt.axhline(rho_2, linestyle='--')
plt.legend()
plt.show()


# ------- PlOT THRUST ANGLE FROM VERTICAL -------
plt.figure(figsize=(8,6), dpi=300)
plt.title('Thrust Angle from Vertical')
plt.plot(elapsed_time, np.rad2deg(T_c_angle_from_vertical),
        color='k', label='Angle from Vertical')
plt.axhline(np.rad2deg(theta),
        linestyle='--', label='Angle Constraints')
plt.axhline(0, linestyle='--')
plt.legend()
plt.show()
```

```
# ------- PLOT ANIMATION -------
# trail fade out
TRAIL_DISSIPATE_ALPHA = 0.5
TRAIL_THRESH = 0.05

for i in tqdm(range(len(xs))):

    fig = plt.figure(dpi=300)
    ax = fig.add_subplot(111, projection='3d')
    ax.set_aspect('equal')

    # hack: fix aspect ratio
    ax.plot([450-1200,450+1200],[0,0],[0,0], color='r', alpha=0)
    ax.plot([0,0],[-330-1200,-330+1200],[0,0], color='b', alpha=0)
    ax.plot([0,0],[0,0],[0,2400], color='g', alpha=0)

    # target
    ax.plot([-500,500],[0,0],[0,0], color='r', alpha=0.2)
    ax.plot([0,0],[-500,500],[0,0], color='r', alpha=0.2)
    ax.scatter(0, 0, 0, s=5, color='r', alpha=0.2)

    ax.view_init(elev=15., azim=120)

    # trail
    for j in range(i):
        trail_alpha = TRAIL_DISSIPATE_ALPHA ** (i-j+1)
        if trail_alpha < TRAIL_THRESH:
            trail_alpha = TRAIL_THRESH
        ax.plot(
            [xs[j], xs[j]],
            [ys[j], ys[j]],
            [0, zs[j]],
            linewidth=0.2,
            color='b',
            alpha=trail_alpha,
        )
        ax.plot(
            [xs[j], xs[j] - T_c_rearranged[j][0]],
            [ys[j], ys[j] - T_c_rearranged[j][1]],
            [zs[j], zs[j] - T_c_rearranged[j][2]],
            linewidth=1,
```

```
            color='r',
            alpha=trail_alpha,
        )
        ax.scatter(
            xs[j], ys[j], 0,
            s=1,
            color='b',
            alpha=trail_alpha,
        )
        ax.scatter(
            xs[j], ys[j], zs[j],
            s=2,
            color='orange',
            alpha=trail_alpha,
        )

    # current state
    ax.plot(
        [xs[i], xs[i]],
        [ys[i], ys[i]],
        [0, zs[i]],
        linewidth=0.2,
        color='b',
    )
    ax.plot(
        [xs[i], xs[i] - T_c_rearranged[i][0]],
        [ys[i], ys[i] - T_c_rearranged[i][1]],
        [zs[i], zs[i] - T_c_rearranged[i][2]],
        linewidth=1,
        color='r',
    )
    ax.scatter(xs[i], ys[i], np.zeros(len(xs))[i], s=1, color='b')
    ax.scatter(xs[i], ys[i], zs[i], s=2, color='orange')

    plt.savefig('./figs/%02d.png' % i)
    plt.close(fig)
```

5.3 Flight Trail Snapshots

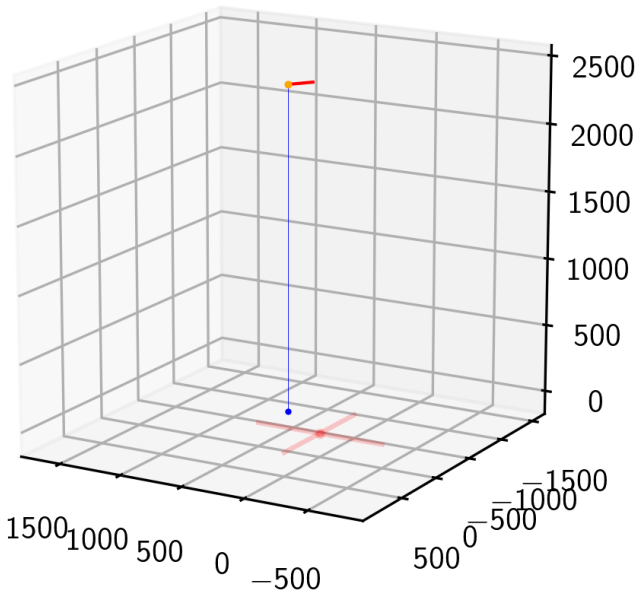

Figure 3: t=0

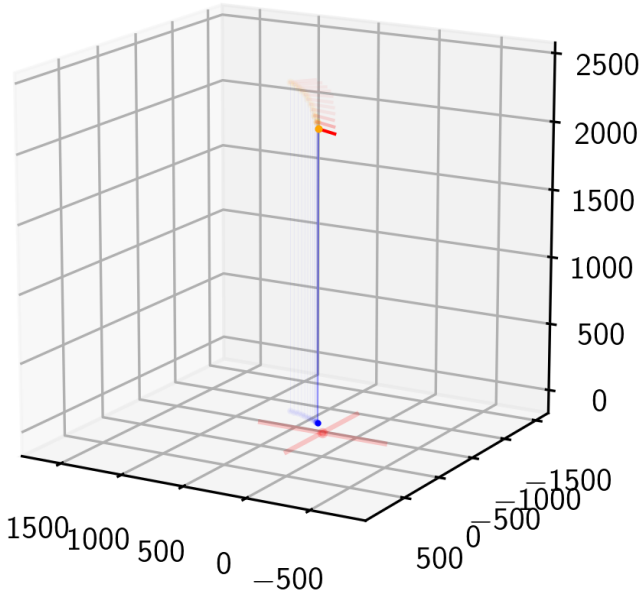

Figure 4: t=10

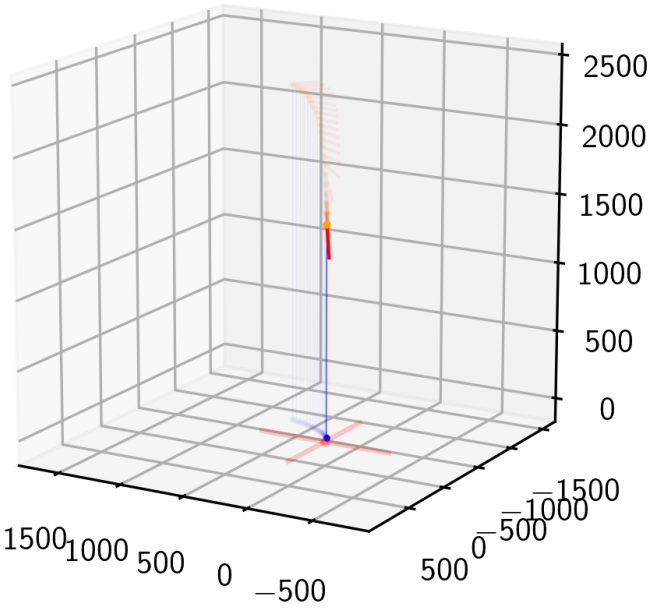

Figure 5: t=20

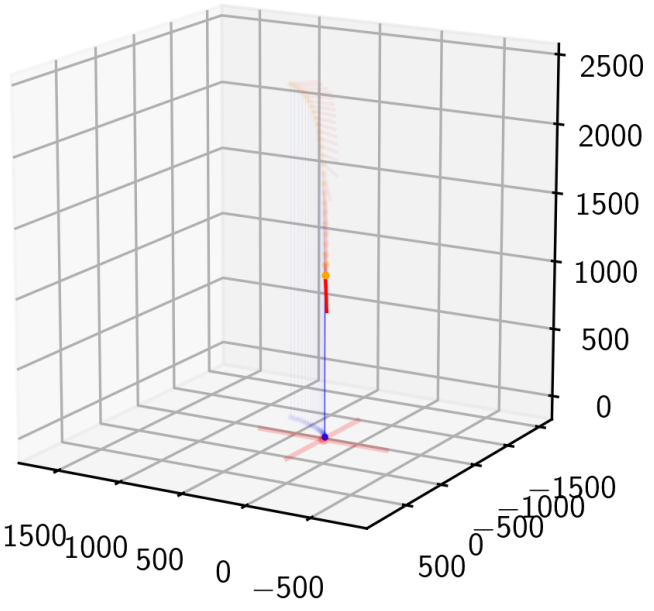

Figure 6: t=25

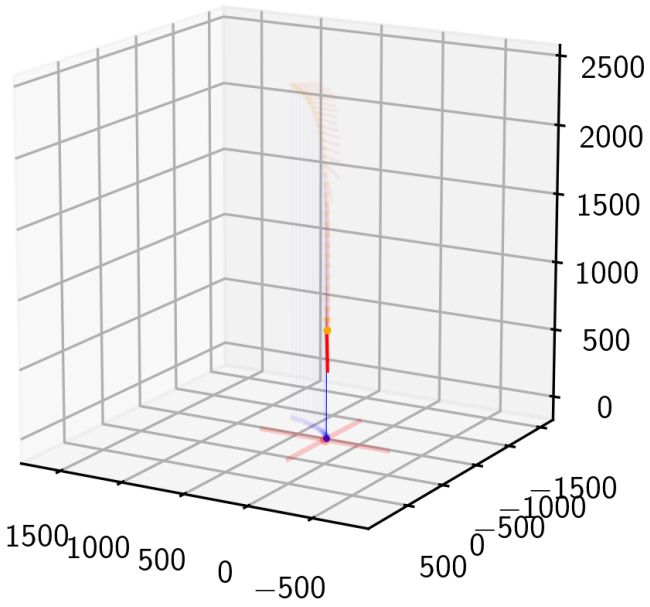

Figure 7: t=30

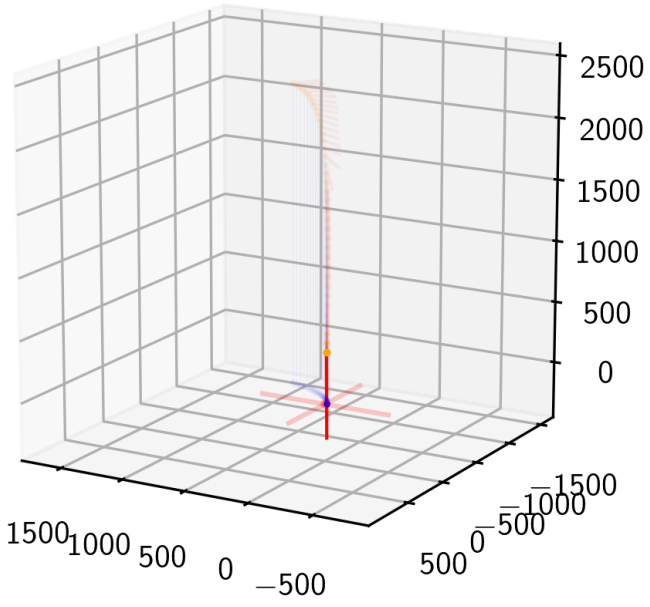

Figure 8: t=35

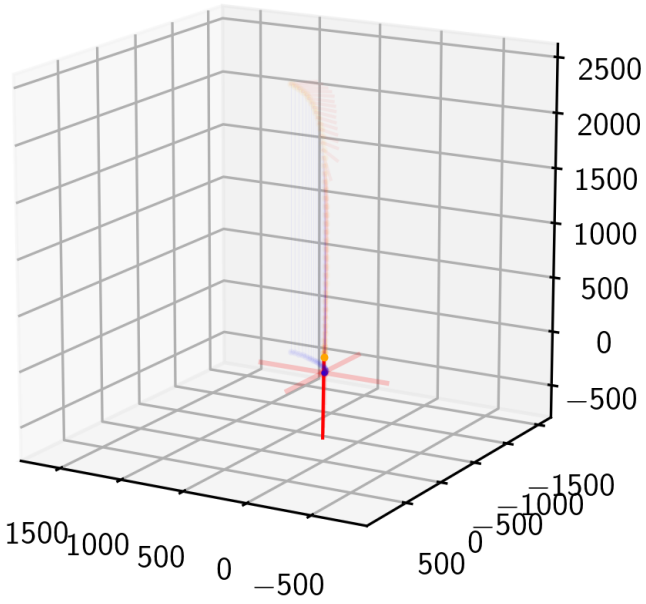

Figure 9: t=40

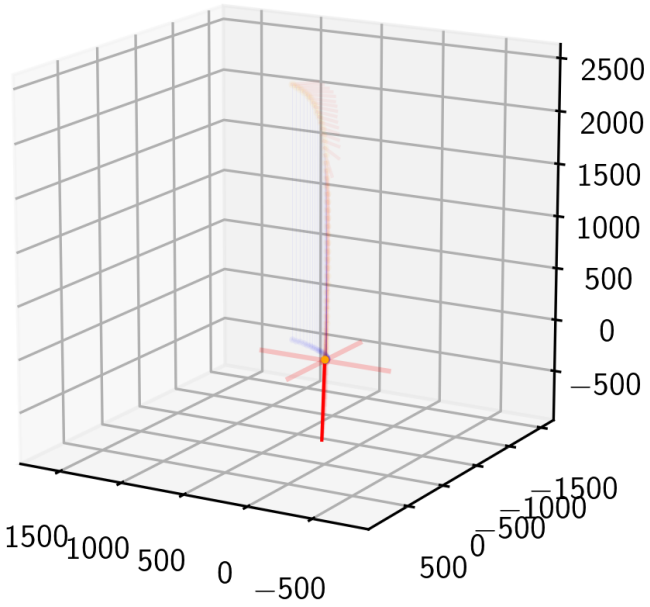

Figure 10: End state