EDL Trajectory Optimization via Convex Optimization Implementation

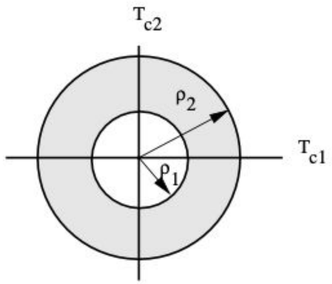
Gagandeep Thapar

Convex Optimization for Trajectory Generation: A Tutorial on Generating Dynamically Feasible Trajectories Reliably and Efficiently - D. Malyuta, T. Reynolds, M. Szmuk, T. Lew, R. Bonalli, M. Pavone, B. Acikmese [1]

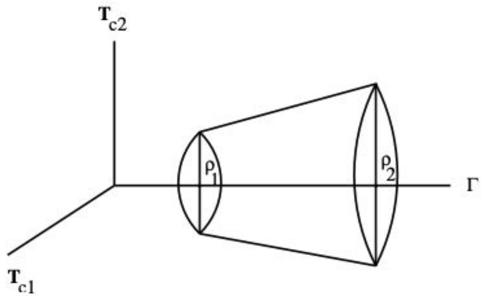
F22 AERO 560 Final; Mehiel

Background on Convex Optimization

- Used to help model propulsive landings, where not all thrust profiles are available
- Involves extrapolating a concave space into a convex space where we can add constraints based on the vehicle and landing parameters
  - The ability to add constraints allows for this model to be applicable for different situations
- Provides a globally optimal solution with a polynomial runtime



NON_CONVEX



CONVEX

Fig. 3 Convexification of thrust magnitude constraint.

Implementation: Landing a Rocket on Mars

- First we created a rocket structure to house all relevant rocket properties
  - Acceleration due to gravity: 3.711 [m/s^2]
  - Dry mass: 1505.0 [kg]
  - Wet mass: 1905.0 [kg]
  - Specific impulse: 225.0 [s]
  - Engine gimbal angle: 27 [degrees]
  - Minimum thrust: 30% of 3100N [N]
  - Maximum thrust: 80% of 3100 [N]
  - Initial position: [2000, 0, 1500] [m]
  - Initial velocity: [80, 30, -75] [m/s]
  - Discrete time step size: 1.0 [s]
  - Continuous time dynamic matrices
  - Number of states: 7 (3D Position + 3D Velocity + Mass)
  - Number of inputs: 4 (Thrust direction + Thrust Magnitude)

Implementation: Powered descent guidance fixed flight time

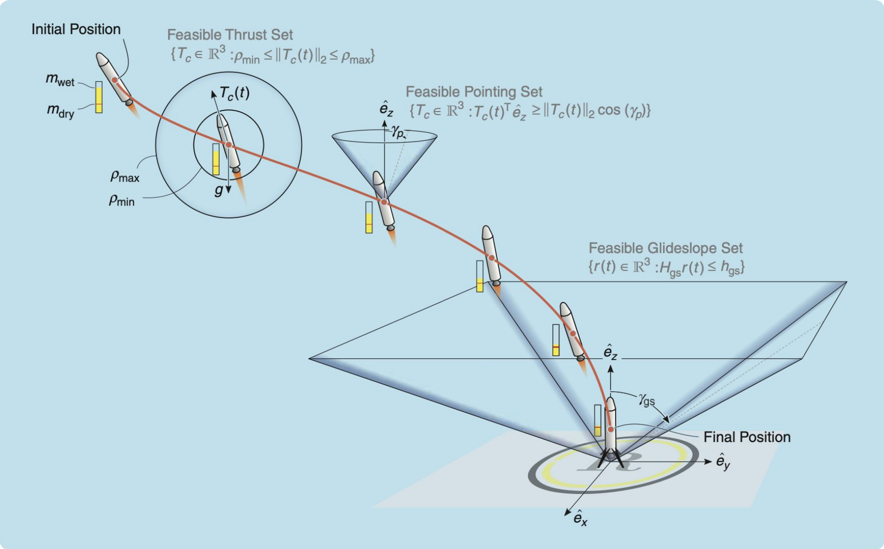
- Powered descent guidance fixed flight time (pdg_fft) function contains most of the Convex Optimization
- Position
  - Initial: [2000, 0, 1500] m
  - Final: [0; 0; 0] m → Want to land on the ground
- Velocity
  - Initial: [80, 30, -75] m/s
  - Final: [0; 0; 0] m/s → Want zero velocity, otherwise boom
- Mass
  - Initial: log(Wet mass)
  - Final: >= log(Dry Mass)
    - Don’t care what the mass is as long as we’re within bounds

Aside: Dissection of MATLAB Optimization Toolbox

- **Optimprob** → Optimization Problem
    - Setup the optimization problem object
    - Can instantiate the objective function
        - e.g., *I want to minimize thrust*
- **Optimvar** → Optimization Variable
    - Setup the different variables that need to be optimized as part of the problem
    - e.g., *Trajectory to minimize thrust*
- **Optimconstr** → Optimization Constraint
    - Different constraints that the optimization needs to follow
    - Can set up several constraints to follow
    - e.g., *Mass should never be below Dry Mass*

Implementation: Constraints

- **Cost function** → Sum of all the thrust
- **Dynamics** → Dynamics must be simulated and taken into account
- **Thrust bounds** → Minimum and maximum thrust capabilities
- **Mass physical bounds** → Changing mass and limited propellant
- **Attitude pointing** → Angle of descent must be 0 at time of landing; shouldn't over-rotate
- Implementation using MATLAB Optimization Toolbox
  - Optimprob
  - Optimvar
  - Optimconstr



Implementation: Constraints

- **Cost function**
  - *sum(zeta)*
- **Dynamics**
  - $x_{k+1} = Ax_k + Bu_k + p$
- **Thrust bounds**
  - $Z_0 = log(m_{wet} - alpha * thrust_{max} * t)$
  - $\boldsymbol{\sigma}_Z = Z - Z_0$
  - $zeta_{min} = u_{min} * (1-\boldsymbol{\sigma}_Z + 0.5*\boldsymbol{\sigma}_Z^2)$
  - $zeta_{max} = u_{max} * (1-\boldsymbol{\sigma}_Z)$
- **Mass physical bounds**
  - $Mass_{min} = Z_0$
  - $Mass_{max} = log(mass_{wet} - alpha * thrust_{min} * t)$
- **Attitude Constraint**
  - $dot(u, [0;0;1]) >= zeta * cos(\gamma_{Point})$

- **X_k: State Vector$_{(7x1)}$**
  - Position$_{(3x1)}$
  - Velocity$_{(3x1)}$
  - Mass$_{(1x1)}$
- **u_k: Input Vector$_{(4x1)}$**
  - Thrust Direction$_{(3x1)}$
  - Thrust Magnitude$_{(1x1)}$
- **p: Constant Gravity$_{(7x1)}$**
  - Zeros$_{(3x1)}$
  - Gravity$_{(3x1)}$
  - Zeros$_{(1x1)}$

Implementation: Constraints

- **Cost function**
  - *sum(zeta)*
- **Dynamics**
  - $X_{k+1} = Ax_k + Bu_k + p$
- **Thrust bounds**
  - $Z_0 = log(m_{wet} - alpha * thrust_{max} * t)$
  - $\boldsymbol{\sigma}_Z = Z - Z_0$
  - $zeta_{min} = u_{min} * (1-\boldsymbol{\sigma}_Z + 0.5*\boldsymbol{\sigma}_Z^2)$
  - $zeta_{max} = u_{max} * (1-\boldsymbol{\sigma}_Z)$
- **Mass physical bounds**
  - $Mass_{min} = Z_0$
  - $Mass_{max} = log(mass_{wet} - alpha * thrust_{min} * t)$
- **Attitude Constraint**
  - $dot(u, [0;0;1]) >= zeta * cos(\gamma_{Point})$

- Want to minimize the thrust required to land the rocket
- *Zeta* is the vector containing information of thrust magnitude

Implementation: Constraints

- **Cost function**
  - $sum(zeta)$
- **Dynamics**
  - $X_{k+1} = Ax_k + Bu_k + p$
- **Thrust bounds**
  - $Z_0 = log(m_{wet} - alpha * thrust_{max} * t)$
  - $\boldsymbol{\sigma}_Z = Z - Z_0$
  - $zeta_{min} = u_{min} * (1 - \boldsymbol{\sigma}_Z + 0.5 * \boldsymbol{\sigma}_Z^2)$
  - $zeta_{max} = u_{max} * (1 - \boldsymbol{\sigma}_Z)$
- **Mass physical bounds**
  - $Mass_{min} = Z_0$
  - $Mass_{max} = log(mass_{wet} - alpha * thrust_{min} * t)$
- **Attitude Constraint**
  - $dot(u, [0;0;1]) >= zeta * cos(\gamma_{Point})$

- Simple Kinematics of the System
- $r_{k+1} = r_k + v_k * \boldsymbol{\Delta} t$
- $v_{k+1} = v_k + (zeta_k + g) * \boldsymbol{\Delta} t$
- $z_{k+1} = z_k + (alpha * zeta) * \boldsymbol{\Delta} t$

Implementation: Constraints

- **Cost function**
  - *sum(zeta)*
- **Dynamics**
  - $X_{k+1} = Ax_k + Bu_k + p$
- **Thrust bounds**
  - $Z_0 = log(m_{wet} - alpha * thrust_{max} * t)$
  - $\boldsymbol{\sigma}_Z = Z - Z_0$
  - $zeta_{min} = u_{min} * (1-\boldsymbol{\sigma}_Z + 0.5*\boldsymbol{\sigma}_Z^2)$
  - $zeta_{max} = u_{max} * (1-\boldsymbol{\sigma}_Z)$
- **Mass physical bounds**
  - $Mass_{min} = Z_0$
  - $Mass_{max} = log(mass_{wet} - alpha * thrust_{min} * t)$
- **Attitude Constraint**
  - $dot(u, [0;0;1]) >= zeta * cos(\gamma_{Point})$

- Need to stay within allowable thrust boundaries
- Z_0 => *Current Minimum Mass; assumes max thrust for full trajectory until now*
- $\boldsymbol{\sigma}_Z$ => *Maximum mass difference*

Implementation: Constraints

- **Cost function**
  - *sum(zeta)*
- **Dynamics**
  - $X_{k+1} = Ax_k + Bu_k + p$
- **Thrust bounds**
  - $Z_0 = \log(m_{wet} - alpha * thrust_{max} * t)$
  - $\boldsymbol{\sigma}_Z = Z - Z_0$
  - $zeta_{min} = u_{min} * (1-\boldsymbol{\sigma}_Z + 0.5*\boldsymbol{\sigma}_Z^2)$
  - $zeta_{max} = u_{max} * (1-\boldsymbol{\sigma}_Z)$
- **Mass physical bounds**
  - $Mass_{min} = Z_0$
  - $Mass_{max} = \log(mass_{wet} - alpha * thrust_{min} * t)$
- **Attitude Constraint**
  - $dot(u, [0;0;1]) >= zeta * \cos(\gamma_{Point})$

- Need to stay within allowable thrust boundaries
- Z_0 => *Current Minimum Mass*
- Can repeat the same process to determine Current Maximum Mass

Implementation: Constraints

- **Cost function**
  - *sum(zeta)*
- **Dynamics**
  - $X_{k+1} = Ax_k + Bu_k + p$
- **Thrust bounds**
  - $Z_0 = log(m_{wet} - alpha * thrust_{max} * t)$
  - $\boldsymbol{\sigma}_Z = Z - Z_0$
  - $zeta_{min} = u_{min} * (1-\boldsymbol{\sigma}_Z + 0.5*\boldsymbol{\sigma}_Z^2)$
  - $zeta_{max} = u_{max} * (1-\boldsymbol{\sigma}_Z)$
- **Mass physical bounds**
  - $Mass_{min} = Z_0$
  - $Mass_{max} = log(mass_{wet} - alpha * thrust_{min} * t)$
- **Attitude Constraint**
  - $dot(u, [0;0;1]) >= zeta * cos(\gamma_{Point})$

- Want to ensure our current angle is less than maximum gimbal angle
- *dot(u, [0;0;1])* provides cosine of the pointing angle relative to the local vertical
- Can compare the actual pointing angle with the maximum pointing angle allowed

MATLAB Implementation

```
% time-based constraints
for i = 1:iters-1

    % dynamic constraints
    dynamicConstraint(i,1:3) = r(:,i+1)     == r(:,i) + (v(:,i))*delt;
    dynamicConstraint(i,4:6) = v(:,i+1)     == v(:,i) + (zeta(i)*u(:,i) + rocket.g')*delt;
    dynamicConstraint(i,7)   = z(i+1)       == log(rocket.m_wet - rocket.alpha * zeta(i)*delt);

    % thrust
    zNOT = log(rocket.m_wet - rocket.alpha*rocket.thrust_max*t(i));
    uMin = rocket.thrust_min*exp(-zNOT);
    uMax = rocket.thrust_max*exp(-zNOT);
    sigZ = z(i) - zNOT;

    thrustMinConstraint(i) = zeta(i) >= uMin*(1-sigZ + 0.5*sigZ^2);
    thrustMaxConstraint(i) = zeta(i) <= uMax*(1-sigZ);

    % mass
    massMinConstraint(i) = zNOT <= z(i);
    massMaxConstraint(i) = z(i) <= log(rocket.m_wet - rocket.alpha*rocket.thrust_min*t(i));

    % attitude
    attitudeConstraint(i,:) = dot(u(:,i), [0;0;1]) >= zeta(i)*cosd(rocket.pointing_max);

end
```

Dynamic and Pointing Constraints

```
r0 = r(:,1) == rocket.pos_initial;
rF = r(:,end) == zeros(3,1);
v0 = v(:,1) == rocket.vel_initial;
vF = v(:,end) == zeros(3,1);
z0 = z(1) == log(rocket.m_wet);
zF = z(end) >= log(rocket.m_dry);
```

Initial and Final Conditions

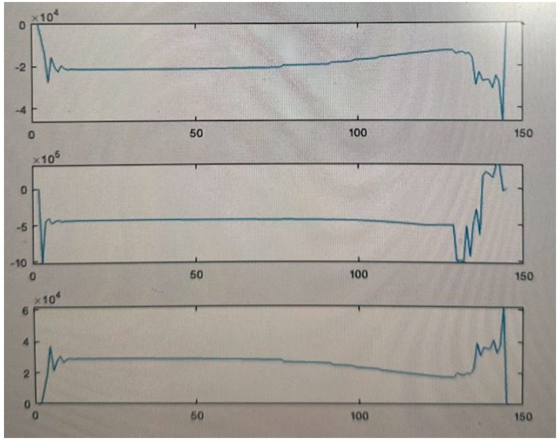
Implementation: Convex Solver

- Set constraints and optimization variables using a for loop to iterate through time steps
  - This is where we started to run into problems…
    - Initial Conditions
    - Second Order Constraints
    - Lack of space?
    - Wonky dynamics
    - Large computation time
      - ~30-45min

```
Solving problem using fmincon.
Error using optim.problemdef.OptimizationProblem/solve
Requested 58354x58354 (30.8GB) array exceeds maximum array size preference
(16.0GB). This might cause MATLAB to become unresponsive.

Error in AERO560_Final_ThaparSanghvi (line 216)
x = solve(pdg, x0);

Related documentation

fx >>
```

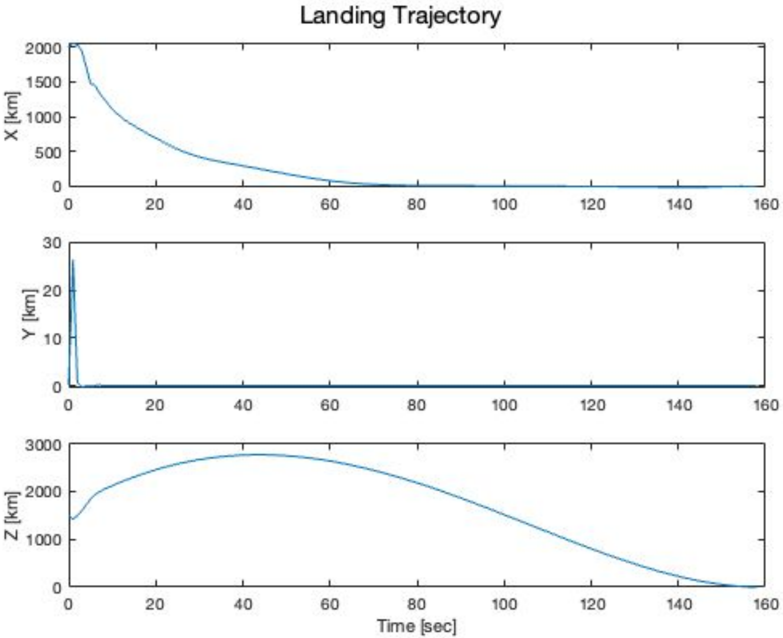
Implementation: Convex Solver

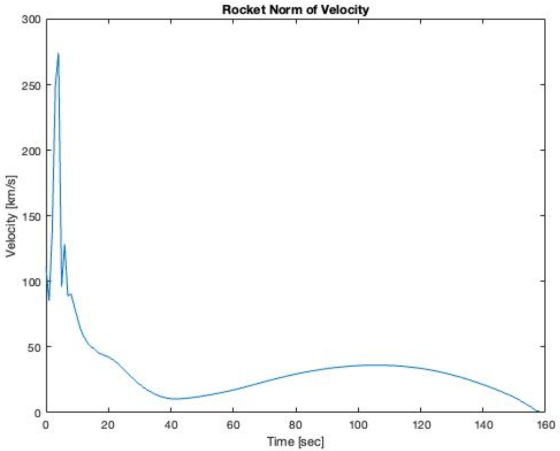
- Poor solutions
  - Likely due to the optimization method
  - Lack of Cone constraints
  - Dynamics were misrepresented (right)
    - Position ~10,000s of km
    - Velocity ~1,000s of km/s
- Altered state dynamics (as seen in previous slides)
  - Able to solve for trajectory; not identical to results in literature
    - Likely due to lack of cone constraints, different dynamics

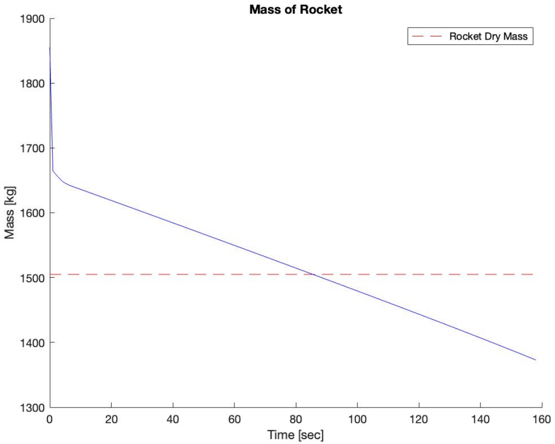


Position of Rocket [km]

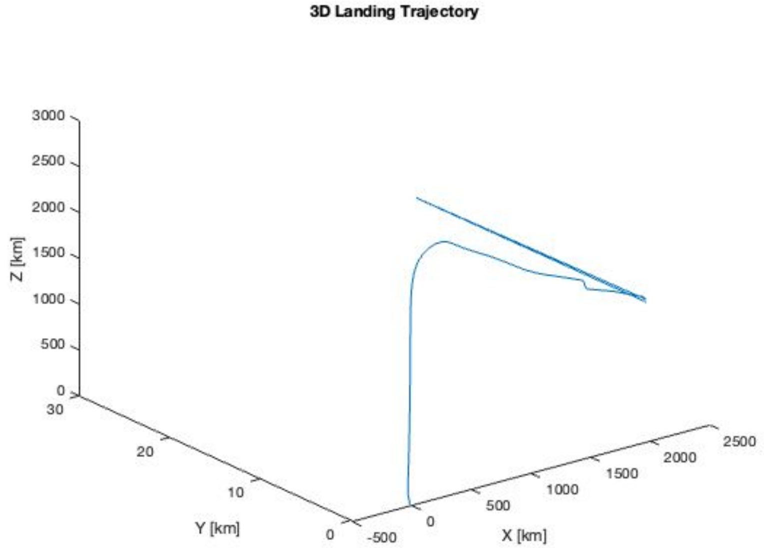
Solution

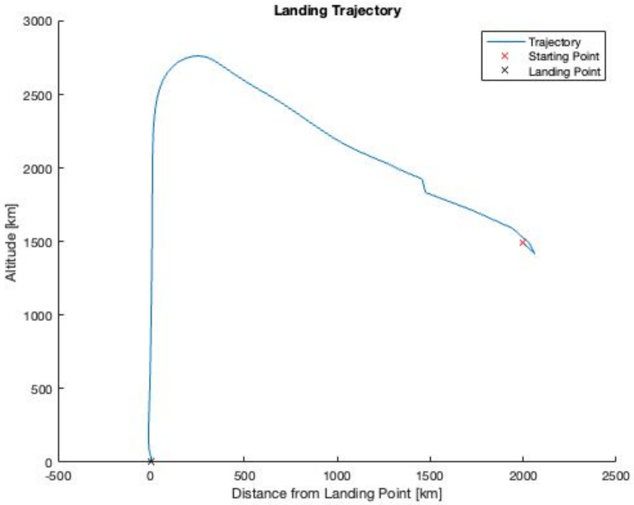

Landing Trajectory
X [km]
Y [km]
Z [km]
Time [sec]


Rocket Norm of Velocity
Velocity [km/s]
Time [sec]


Mass of Rocket
Rocket Dry Mass
Mass [kg]
Time [sec]

Solution


3D Landing Trajectory
Z [km]
3000
2500
2000
1500
1000
500
0
30
20
10
0
Y [km]
-500
0
500
1000
1500
2000
2500
X [km]


Landing Trajectory
Trajectory
Starting Point
Landing Point
Altitude [km]
3000
2500
2000
1500
1000
500
0
-500
0
500
1000
1500
2000
2500
Distance from Landing Point [km]

Future work

- Add conical constraints
- Implement glideslope constraint
- Discretize and use control law to make it more applicable
- Adjust mass differential equation

Takeaways

- Able to land in specified landing spot using exact constraints
- Sharp trajectory doesn’t pass intuition checks; likely a consequence of optimization technique/constraints
- Unable to meet mass requirements
  - Likely due to misrepresentation of how mass is consumed over time
- Optimization is difficult and computationally expensive
  - Lot of work to be done for flight implementation