Demonstration of Convex Programming for Powered Descent Guidance

Krishna Soni*
*"The" University of Texas at Austin, Austin, TX**

***Abstract*—This report explores implementations of convex programming approaches of fuel-optimal powered descent algorithms, which in the last decade have become increasingly important in enabling scientific discovery and exploration. These types of problems are traditionally non-trivial due to the non-convex control constraints but due to results from "lossless convex convexification" approaches shown in [1], the associated problem can be formulated as a finite-dimensional, second-order cone (SOCP) program with guarantees on a globally optimal solution. With recent advancements in embedded convex solvers, problems of this classification can be solved in real-time on embedded platforms, which further lays the groundwork for sophisticated planetary missions requiring strict pointing constraints and performance guarantees. This paper will first give a brief overview of the fuel-optimal landing problem and review prior work up until this point. Different formulations of the soft-landing problem are posed and solved with a convex solver CVX. A numerically solvable version of the problem is presented, with a time-of-flight (TOF) search algorithm that allows solving the problem in a successive manner. Finally, a 3-degree-of-freedom dynamics model with an accompanying trajectory-following algorithm is developed, which enables the simulation of a closed-loop landing trajectory with noise to test the fidelity of the approach presented in this report.**

***Index Terms*—Convex optimization, optimal control, planetary soft landing, lossless convexification**

I. Introduction

Autonomous spacecraft are critical enablers of exploring other planetary bodies within our solar system, in the case of planetary missions with atmospheres (using the Mars Science Laboratory missions as an example). First, the entry phase (usually involving an ablative heat shield) aims to null most of the surface relative velocity. Once the landing platform is slowed to supersonic speeds, a parachute is deployed. At a prescribed point in the landing profile, the parachute is released, and the powered descent phase of the landing profile commences. The powered descent phase aims to minimize any landing error accumulated throughout the flight's descent phase. Re-entry through the atmosphere and parachute descent can yield high uncertainty to the final landing site of an autonomous vehicle due to limited control authority and high uncertainty due to disturbances in the atmosphere.

Propulsive landing capabilities enable the delivery of scientific payloads and, eventually, humans to other celestial bodies in the solar system. In the past, most planetary missions had relaxed landing constraints, with tolerances on the order of tens of kilometers. However, for modern scientific missions and, eventually, human-rated missions, the uncertainty of that degree is not tolerable.

For example, planning a robot (or human) mission to travel significant distances to reach a target science objective adds unnecessary complexity and cost. Moreover, it may make specific mission profiles completely unfeasible. As a result, propulsive landing algorithms that minimize landing error and enable large divert maneuvers (as in the case of the Mars Science Laboratory, MSL) are necessary to achieve high-value science objectives. Figure 1 shows a high-level overview of different phases of entry descent and landing, as well as comparisons of capabilities of legacy algorithms compared to more modern algorithms such as G-FOLD [1].

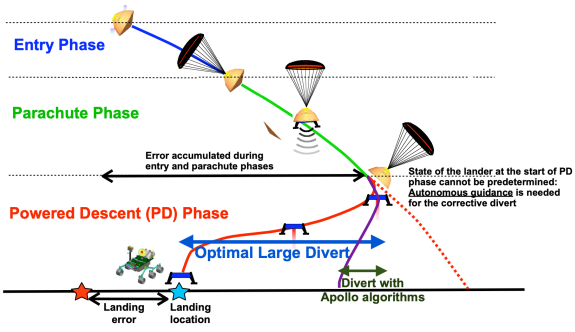

Fig. 1. Diagram highlighting different phases of entry, descent, and landing for the Mars Science Laboratory, with comparisons of Apollo-era divert algorithms and G-FOLD [1]

A desire to provide a pin-point landing capability has led to the developing of algorithms such as G-FOLD (Guidance for Fuel Optimal Large Diverts) in 2004.

G-FOLD was developed to achieve optimal powered descent performance while minimizing fuel usage and landing error. The algorithm aims to generate feasible trajectories incorporating thrust, pointing, and dynamics constraints. These constraints may also entail safety factors for surface avoidance and maximum velocity on approach.

The constrained optimization problem for soft-landing is an inherently non-convex problem, mainly due to the thrust control constraints on the vehicle. G-FOLD relaxes these constraints by relaxing the problem and performing a change of variables technique to reformulate the problem as a second-order cone program (SOCP). This resultant problem is convex and guarantees a globally optimal solution. Furthermore, recent innovations have automated code generation for interior point method solvers, aligned with the needs of the pin-point landing problem, allowing the algorithm to be run on real-time embedded platforms.

Examples of this algorithm running on autonomous vehicles have already been proven. G-FOLD was tested on Masten's "Xombie" vertical take-off, and vertical landing (VTVL) test bed in 2012 [1], showcasing lateral divert maneuvers of 550m, 650m, and 750m (generated offline resulting in successful execution with 1m landing accuracy and 1.24 m/s velocity error. In 2013, the algorithm was tested again but run online. Diverts of 570m and 800m resulted in successful landing trajectories with 3m of landing error and 1 m/s error of velocity error throughout the flown trajectory [2]. Commercial implementation of this algorithm has already come to fruition, as SpaceX readily leverages the G-FOLD framework to perform pin-point landing for their Falcon 9 and Starship launch vehicles.

While this paper does not encompass a full-scale solution of G-FOLD, it does demonstrate an implementation of the algorithm using a commonly used solver CVX in a MATLAB framework. In the following sections, a general overview of the soft-landing problem is given and demonstrated to be solved via CVX. Then, a discretized version of the problem, amenable to real-time frameworks, is presented and integrated into a 3-DOF simulation encompassing sensor noise. Finally, results from a Monte Carlo simulation from the 3-DOF simulation are presented.

II. Problem Formulation/Background

This section will summarize the problem formulation for an autonomous vehicle performing a propulsive pin-point landing on a planet with a uniform gravity field. [3] and [4] formulate this problem as a set of sub-problems. First, the constraints and objectives of the original soft-landing (non-convex) problem are defined. Second, the problem is relaxed by introducing a slack variable and a change of variables technique. This relaxation allows the problem to be written in SOCP form. Finally, discretization of the problem objective function and constraints are applied, making it amenable for solving in real-time and setting the stage for developing numerical simulations described in later sections.

For this trajectory optimization problem, a Mars descent vehicle is modeled. In general, the powered descent phase of a mars landing vehicle occurs at a relatively low altitude to the planet's surface. Here, aerodynamic forces are treated as disturbance forces versus being accounted for directly in the problem formulation since, compared to the entry interface or parachute descent, the velocities are small, so aerodynamic forces are not dominant. Similarly, the problem formulation in this report only considers the 3-DOF case of translational dynamics and does not account for attitude dynamics for simplicity. Further, it is assumed that all dynamics are represented in the surface-fixed frame (with the target landing site oriented at the origin), as depicted in Figure 2.

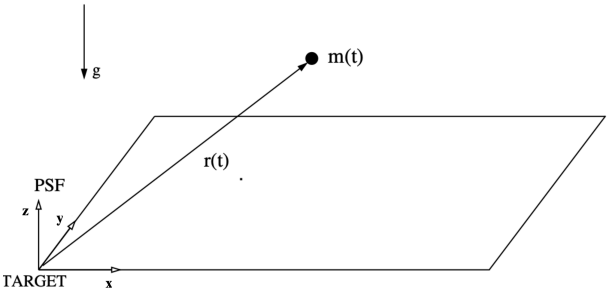

Fig. 2. Surface-fixed frame [3]

Here, the transnational dynamics are represented as:

$$\begin{aligned} \ddot{\boldsymbol{r}} &= \boldsymbol{g} + \frac{\boldsymbol{T}_c(t)}{m(t)} \\ \dot{m}(t) &= -\alpha \left\| \boldsymbol{T}_c(t) \right\| \\ \boldsymbol{r} &\in \mathbb{R}^3, \boldsymbol{T}_c(t) \in \mathbb{R}^3, m(t) \in \mathbb{R} \end{aligned} \tag{1}$$

Where $\boldsymbol{r}$ represents the position of the vehicle relative to the target (landing site), $\boldsymbol{T}_c$ is the net thrust imposed on the spacecraft. Where, given n thrusters, nominal thrust T, and engine cant angle ϕ, we can denote the net thrust as:

$$\boldsymbol{T}_c = (nT\cos\phi)\boldsymbol{e} \tag{2}$$

Similarly, $\boldsymbol{g}$ is the constant gravitation acceleration vector of the planet. m is the spacecraft mass (which

includes the dry structure, propellant, and oxidizer), and α is a positive scalar that describes the change in mass (fuel consumption rate) denoted by:

$$\alpha = \frac{1}{I_{sp} g_0 \cos\phi} \tag{3}$$

Where g_0 is the Earth's gravitational constant, and I_{sp} is the engines' specific impulse.

For any propulsive vehicle, thrust is bounded by throttling constraints on the engine. Thrust for each engine is bounded by:

$$\begin{aligned} &T_2 \leq T(t) \leq T_2 \\ &\rho_1 \leq \|T_c(t)\| \leq \rho_2 \\ &0 < \rho_1 < \rho_2 \end{aligned} \tag{4}$$

Where ρ_1 and ρ_2 are positive constants given by $\rho_1 = nT_1 \cos\phi$ and $\rho_2 = nT_2 \cos\phi$, corresponding to the deepest throttle (lowest thrust) and highest thrust for the engines. Note that (4) denotes a non-convex constraint, which will be relaxed by a change of variables technique in later formulations of the optimization problem (shown in later sections). For clarity, the geometry of the vehicle in question is depicted in Figure 3.

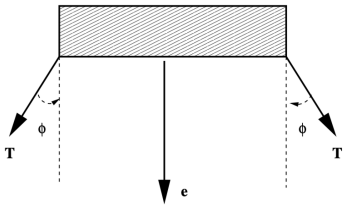

Fig. 3. Spacecraft geometry for minimum fuel problem [3]

Building on this, further constraints can be imposed to formulate the trajectory optimization problem. Here, constraints on the initial and terminal conditions are imposed:

$$\begin{aligned} m(0) &= m_{\text{wet}} \\ \boldsymbol{r}(0) &= \boldsymbol{r}_0 \\ \dot{\boldsymbol{r}}(0) &= \dot{\boldsymbol{r}}_0 \\ \boldsymbol{r}(t_f) &= [0,0,0]^\intercal \\ \dot{\boldsymbol{r}}(t_f) &= [0,0,0]^\intercal \end{aligned} \tag{5}$$

Where t_f denotes the final time-of-flight (TOF) and m_{wet} is the initial wet mass of the vehicle (including dry structure mass).

Additional constraints based on safety considerations are imposed, where $r_z > 0, \forall t \in [0, t_f]$ meaning that the trajectory problem is constrained such that the generated vehicle trajectory does not travel through the ground. [3] also imposes constraints on the vehicle angle relative to the ground (i.e., a glide-slope constraint) to accommodate obstacle avoidance or usage of terrain-relative navigation for state estimation. Here, the glide constraint angle is defined by

$$\theta_{\text{alt}} = \arctan\left\{ \frac{\sqrt{r_x^2(t) + r_y^2(t)}}{r_z(t)} \right\} \tag{6}$$

Here additional state constraints (such as the glide-slope angle shown above) may be described in a general form (as a second-order cone inequality) shown in (7), where the state is defined as $\boldsymbol{x}(t) = [\boldsymbol{r}, \dot{\boldsymbol{r}}]^\intercal$, $S_j \in \mathbb{R}^{n_j \times 6}$, $c_j \in \mathbb{R}^6$, $n_j \leq 6, v_j \in \mathbb{R}^{n_j}$ and $a_j \in \mathbb{R}$.

$$\begin{aligned} \|S_j \boldsymbol{x}(t) - \boldsymbol{v}_j\| + \boldsymbol{c}_j^T \boldsymbol{x}(t) + \boldsymbol{a}_j \leq 0, \\ j = 1, \ldots, n_s, \forall t \in [0, t_f] \end{aligned} \tag{7}$$

The glide-slope constraint can therefore be expressed as:

$$\|S\boldsymbol{x}\| + \boldsymbol{c}_j^\intercal \boldsymbol{x} \leq 0 \tag{8}$$

where,

$$\begin{aligned} S &= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \\ \boldsymbol{c} &= \begin{bmatrix} -\tan\theta_{\text{alt}} & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^\intercal \end{aligned} \tag{9}$$

At this point, based on the defined parameters and relationships described above, the soft-landing problem may be formulated as follows:

Problem 1

$$\begin{aligned} &\underset{t_f, \boldsymbol{T}_c(\cdot)}{\text{maximize}} \quad m(t_f) = \min_{\boldsymbol{T}_c(\cdot), t_f} \int_0^{t_f} \|\boldsymbol{T}_c(t)\| \, dt \\ &\text{subject to} \\ &\ddot{\boldsymbol{r}}(t) = \boldsymbol{g} + \boldsymbol{T}_c/m(t), \quad \dot{m}(t) = -\alpha \|\boldsymbol{T}_c(t)\|, \\ &\rho_1 \leq \|T_c(t)\| \leq \rho_2, \quad r_z(t) \geq 0, \\ &\|S_j \boldsymbol{x}(t) - \boldsymbol{v}_j\| + \boldsymbol{c}_j^T \boldsymbol{x}(t) + \boldsymbol{a}_j \leq 0, j = 1, \ldots, n_s, \\ &m(0) = m_{\text{wet}}, \ \boldsymbol{r}(0) = \boldsymbol{r}_0, \ \dot{\boldsymbol{r}}(0) = \dot{\boldsymbol{r}}_0, \\ &\boldsymbol{r}(t_f) = \dot{\boldsymbol{r}}(t_f) = \boldsymbol{0} \end{aligned}$$

The problem above encompasses a non-linear constraint on thrust control, $\rho_1 \leq \|T_c(t)\| \leq \rho_2$, as well as a non-linear dynamics of the form $\ddot{\boldsymbol{r}}(t) = \boldsymbol{g} + \boldsymbol{T}_c/m(t)$. In successive problems below, both constraints are relaxed based on results from optimal control theory and lossless convexification, and a change of variables technique introduced in [3].

A. Relaxation of Thrust Constraints

[3] introduces a modification to **Problem 1** above as a separate sub-problem:

Problem 2

$$\underset{t_f, \boldsymbol{T}_c(\cdot), \Gamma(t)}{\text{minimize}} \quad \int_0^{t_f} \Gamma(t) dt$$

subject to

$$\ddot{\boldsymbol{r}}(t) = \boldsymbol{g} + \boldsymbol{T}_c/m(t), \quad \dot{m}(t) = -\alpha\Gamma(t),$$
$$\|\boldsymbol{T}_c(t)\| \leq \Gamma(t),$$
$$\rho_1 \leq \Gamma(t) \leq \rho_2, \quad r_z(t) \geq 0,$$
$$\|S_j \boldsymbol{x}(t) - \boldsymbol{v}_j\| + \boldsymbol{c}_j^T \boldsymbol{x}(t) + \boldsymbol{a}_j \leq 0, j = 1, \ldots, n_s,$$
$$m(0) = m_{\text{wet}}, \ \boldsymbol{r}(0) = \boldsymbol{r}_0, \ \dot{\boldsymbol{r}}(0) = \dot{\boldsymbol{r}}_0,$$
$$\boldsymbol{r}(t_f) = \dot{\boldsymbol{r}}(t_f) = \boldsymbol{0}$$

Here the slack variable $\Gamma \in \mathbb{R}$ is introduced, which replaces $\|\boldsymbol{T}_c(t)\|$ in **Problem 1** and adds an extra constraint $\|\boldsymbol{T}_c(t)\| \leq \Gamma(t)$. Note that since **Problem 2** is a relaxation of **Problem 1**, feasible solutions for **Problem 1** admit solutions for **Problem 2**, but the converse is not necessarily true. [3] introduces a lemma in their work which aims to prove that the optimal solution to **Problem 2** admits a feasible solution to **Problem 1**:

Lemma 1: Consider a solution of Problem 2 given by $[t_f^*, T_c^*(\cdot), \Gamma^*(\cdot)]$. Then, $[t_f^*, T_c^*(\cdot)]$ is also a solution of Problem 1 and $\|T_c^*(t)\| = \rho_1$ or $\|T_c^*(t)\| = \rho_2$ for $t \in [0, t_f^*]$.

Leveraging the Hamiltonian of **Problem 2**, the necessary conditions of optimality are given in the general form, using Pontryagin's maximum principle. Similarly, the stationarity condition, transversality condition, and associated co-state equations are proven to be satisfied. It is found that the lemma declared in [3] implies that the thrust magnitude constraint may be put into the form of a convex constraint based on the introduction of a slack variable. Further, it is proven that if a exists an optimal solution for **Problem 2**, there is consequently a solution for **Problem 1**, meaning that **Problem 2** may be solved as a convex program in order to obtain a solution to **Problem 1**, with non-convex control constraints. Figure 4 shows a visual depiction of this transformation, where the non-convex thrust constraints form an annulus (clearly non-convex) which is mapped to a convex set, represented as a 3D cone with the introduction of a slack variable Γ.

B. Change of Variables

[3] further introduces a change of variables which aids in constructing **Problem 2** as a continuous-time optimal control problem, with convex cost and convex constraints of linear or second-order cone form. Such modification is necessary in allowing **Problem 2** to be solved numerically.

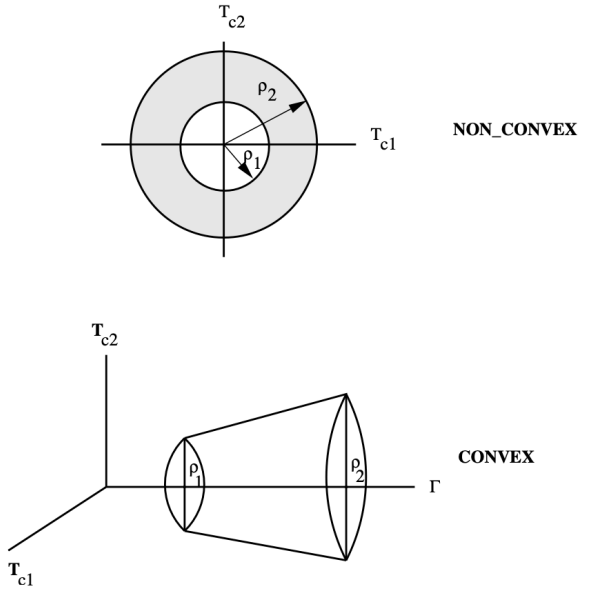

Fig. 4. Convex transformation of thrust control constraints with slack variable Γ [3]

Here, the following change of variables is made:

$$\sigma \triangleq \frac{\Gamma}{m} \quad \text{and} \quad \boldsymbol{u} \triangleq \frac{\boldsymbol{T}_c}{m} \tag{10}$$

As a result, the dynamics constraints may be rewritten as:

$$\ddot{\boldsymbol{r}}(t) = \boldsymbol{u}(t) + \boldsymbol{g} \tag{11}$$
$$\frac{\dot{m}(t)}{m(t)} = -\alpha\sigma(t) \tag{12}$$

By some manipulation,

$$m(t) = m_0 \exp\left\{-\alpha \int_0^{t_f} \sigma(\tau) d\tau\right\} \tag{13}$$

Note that because α is given and positive, minimizing fuel consumption equivocates to minimizing in the integral $\int_0^{t_f} \sigma(\tau) d\tau$ shown above. Similarly, the control constraints may be expressed as follows:

$$\|\boldsymbol{u}(t)\| \leq \sigma(t), \quad \forall t \in [0, t_f] \tag{14}$$
$$\frac{\rho_1}{m(t)} \leq \sigma(t) \leq \frac{\rho_2}{m(t)}, \quad \forall t \in [0, t_f] \tag{15}$$

It is of note that the inequality above doesn't necessarily define a convex set, as if m is considered the decision variable for the problem, the constraints above yield a

bi-linear, which is non-convex. As a result, an additional variable is introduced where:

$$z \triangleq \ln m \tag{16}$$

This yields updated equations for control and mass depletion constraints where:

$$\dot{z}(t) = -\alpha\sigma(t) \tag{17}$$

$$\rho_1 e^{-z(t)} \leq \sigma(t) \leq \rho_2 e^{-z(t)}, \quad \forall t \in [0, t_f] \tag{18}$$

The left-hand side of the inequality in (18) admits a convex feasible region, however, the right-hand side does not. Here, a Taylor expansion of e^{-z} is used. The first three terms of the Taylor expansion are used to approximate a second-order cone form on the left-hand size of the inequality, while the right-hand side uses only the first two terms to bring it to a linear form.

$$\mu_1(t)\left[1 - [z(t) - z_0(t)] + \frac{[z(t) - z_0(t)]^2}{2}\right] \leq \sigma(t) \tag{19}$$

$$\leq \mu_2(t)\left\{1 - [z(t) - z_0(t)]\right\}, \quad \forall t \in [0, t_f] \tag{20}$$

where,

$$\mu_1 \triangleq \rho_1 e^{-z_0}, \quad \mu_2 \triangleq \rho_2 e^{-z_0} \tag{21}$$

$z_0 = \ln[m_{\text{wet}} - \alpha\rho_2]$ is the lower bound on the $z(t)$ at a given time t. Since $z(t)$ corresponds to the mass depletion at a given time t, an additional constraint is imposed (based on the total mass of the vehicle and max/min throttle limits):

$$\ln(m_{\text{wet}} - \alpha\rho_2) \leq z(t) \leq \ln(m_{\text{wet}} - \alpha\rho_1) \tag{22}$$

From manipulation and change of variables described above, **Problem 3** is posed.

Problem 3

$$\underset{t_f,\, u(\cdot),\, \sigma(\cdot)}{\text{minimize}} \quad \int_0^{t_f} \sigma(t)dt$$

subject to

$$\ddot{\boldsymbol{r}}(t) = \boldsymbol{g} + \boldsymbol{u}(t), \quad \dot{z}(t) = -\alpha\sigma(t),$$

$$\|\boldsymbol{u}(t)\| \leq \sigma(t),$$

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

$$\leq \mu_2(t)\left\{1 - [z(t) - z_0(t)]\right\}, \quad \forall t \in [0, t_f],$$

$$\ln(m_{\text{wet}} - \alpha\rho_2) \leq z(t) \leq \ln(m_{\text{wet}} - \alpha\rho_1),$$

$$\|\boldsymbol{r}_{x,y}(t)\| \leq \tan\theta_{\text{alt}} r_z(t),$$

$$m(0) = m_{\text{wet}}, \ \boldsymbol{r}(0) = \boldsymbol{r}_0, \ \dot{\boldsymbol{r}}(0) = \dot{\boldsymbol{r}}_0,$$

$$\boldsymbol{r}(t_f) = \dot{\boldsymbol{r}}(t_f) = \boldsymbol{0}$$

Above, the glide-slope constraint is simplified based on a modification posed in [4], $\boldsymbol{r}_{x,y}$ are the longitudinal components of $\boldsymbol{r}$ and r_z denotes altitude. Where It is shown that **Problem 3** has all equality and inequality constraints defining convex feasible regions (intersections) of state and control spaces with a convex objective function. As a result, **Problem 3** is amenable to being formulated as a finite-dimensional SOCP problem so that it can be solved numerically (for a given TOF) after discretization.

C. Discretization

[3] [4] introduces discretization in order for **Problem 3** to be solved numerically. Discretization is based on piece-wise linear control input, given N number of time steps of equal length. For $\Delta t > 0$, $t_k = k\Delta t$, $\forall k \in 0, \ldots, N$, the control and state evolution take the form:

$$\boldsymbol{u}(t) = \boldsymbol{u}_k + (\boldsymbol{u}_{k+1} - \boldsymbol{u}_{k+1})\tau \tag{23}$$

$$\sigma(t) = \sigma_k + (\sigma_{k+1} - \sigma_{k+1})\tau \tag{24}$$

$$\tau = \frac{t - t_k}{\Delta t}, \quad \text{for } t \in [t_k, t_{k+1}), \quad k = 0, \ldots, N-1 \tag{25}$$

Using this formulation, **Problem 3** may be formed as a finite-dimensional SOCP, as stated in **Problem 4** below, which can be numerically solved:

Problem 4

$$\underset{\boldsymbol{u}_0 \ldots \boldsymbol{u}_N,\, \sigma_0 \ldots \sigma_N}{\text{minimize}} \quad -z_N$$

subject to

$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k + \frac{\Delta t}{2}(\dot{\boldsymbol{r}}_k + \dot{\boldsymbol{r}}_{k+1}) + \frac{\Delta t^2}{12}(\boldsymbol{u}_{k+1} - \boldsymbol{u}_k),$$

$$\dot{\boldsymbol{r}}_{k+1} = \dot{\boldsymbol{r}}_k + \frac{\Delta t}{2}(\boldsymbol{u}_k + \boldsymbol{u}_{k+1}) - \boldsymbol{g}\Delta t,$$

$$z_{k+1} = z_k - \frac{\alpha\Delta t}{2}(\sigma_k + \sigma_{k+1}),$$

$$\|\boldsymbol{u}_k\| \leq \sigma_k,$$

$$\mu_{1,k}\left[1 - (z_k - z_{0,k}) + \frac{(z_k - z_{0,k})^2}{2}\right] \leq \sigma(t),$$

$$\leq \mu_{2,k}\left[1 - (z_k - z_{0,k})\right],$$

$$\ln(m_{\text{wet}} - \alpha\rho_2) \leq z_k \leq \ln(m_{\text{wet}} - \alpha\rho_1),$$

$$\|[r_{x,k}, r_{y,k}]^\intercal\| \leq \tan\theta_{\text{alt}} r_{z,k},$$

$$m(0) = m_{\text{wet}}, \ \boldsymbol{r}_0 = \boldsymbol{r}_0, \ \dot{\boldsymbol{r}}_0 = \dot{\boldsymbol{r}}_0,$$

$$\boldsymbol{r}_N = \dot{\boldsymbol{r}}_N = \boldsymbol{0},$$

$$N\Delta t = t_f,$$

$$r_{z,k} \geq 0$$

D. Time of Flight Determination

At this time, a **Problem 4** can be solvable numerically, but a necessary input is a given time-of-flight for a given

set of initial conditions. [4] achieves this by finding the t_f, which allows for the minimum fuel usage via the Golden Search Algorithm [5]. In this report, the bounds of t_f are determined as follows:

$$\frac{m_{\text{wet}} \left\| \boldsymbol{v}_f - \boldsymbol{v}_0 \right\|}{\rho_2} \leq t_f^* \leq \frac{m_{\text{wet}} - m_{\text{dry}}}{\alpha \rho_1} \tag{26}$$

[4] notes that minimum fuel configuration is a uni-modal function will a global minimum. The G-FOLD algorithm is enveloped in an outer search to find the optimal final time t_f^*, and then numerically solving the fuel-optimal guidance problem (**Problem 4**) with t_f^*. Results show in the following section this portion of the algorithm is the most time-intensive and is the largest detractor for on-board implementation methods. However, further reduction in solution time is not explored in this report.

III. Implementation/Simulation Results

The following section builds on the problem formulation described in previous sections in developing a 3-DOF simulation implementing G-FOLD. First, the implementation of the fixed-time fuel-optimal guidance problem is discussed. A search method for finding the optimal final time given a set of initial conditions is also illustrated and tested in simulation (open-loop). Finally, a closed-loop 3-DOF simulation, with non-linear dynamics and noise models, is developed and tested in Monte Carlo simulation.

For the purposes of this report, all simulations shown below are encompassed by the configuration and initial conditions shown in Table I (matching testing done in [3]).

TABLE I
Initial conditions for numerical and closed-loop simulations

Parameter	Value
$\boldsymbol{r}_0$	$[1500, 100, 2000]^\intercal$ m
$\boldsymbol{r}_f$	$[0, 0, 0]^\intercal$ m
$\boldsymbol{v}_0$	$[100, 0.01, -75]^\intercal$ m/s
$\boldsymbol{v}_f$	$[0, 0, 0]^\intercal$ m/s
$T_{\max}$	18600 N
T_2	14880 N
T_1	5580 N
I_{sp}	225 s
m_{dry}	1505 kg
m_{wet}	1905 kg
$\boldsymbol{g}$	$[0, 0 - 3.7114]^\intercal$ m/s^2
g_0	9.80665 m/s^2
ϕ	27^o

A. Numerical Simulation of Fixed-Time Trajectory Optimization Problem

Problem 4 was solved by constructing the optimization problem in CVX (using the SEDUMI solver) in a MATLAB environment. As part of the development process, a simplified trajectory optimization problem was formed (shown below), which aims to minimize control input and is only subject to dynamics constraints. Here, we define **Problem 5** as:

Problem 5

$$\begin{aligned}
&\underset{U_{0:N-1}}{\text{minimize}} \quad \left\| U_{0:N-1} \right\|^2 \\
&\text{subject to} \\
&\boldsymbol{r}_{k+1} = \boldsymbol{r}_k + \frac{\Delta t}{2} \left(\dot{\boldsymbol{r}}_k + \dot{\boldsymbol{r}}_{k+1} \right) + \frac{\Delta t^2}{12} \left(\boldsymbol{u}_{k+1} - \boldsymbol{u}_k \right), \\
&\dot{\boldsymbol{r}}_{k+1} = \dot{\boldsymbol{r}}_k + \frac{\Delta t}{2} \left(\boldsymbol{u}_k + \boldsymbol{u}_{k+1} \right) - \boldsymbol{g} \Delta t, \\
&\left\| [r_{x,k}, r_{y,k}]^\intercal \right\| \leq \tan \theta_{\text{alt}} r_{z,k}, \\
&m(0) = m_{\text{wet}}, \ \boldsymbol{r}(0) = \boldsymbol{r}_0, \ \dot{\boldsymbol{r}}(0) = \dot{\boldsymbol{r}}_0, \\
&\boldsymbol{r}_N = \dot{\boldsymbol{r}}_N = \boldsymbol{0}, \\
&N \Delta t = t_f, \\
&r_{z,k} \geq 0
\end{aligned}$$

Where $U_{0:N-1}$ represents a block matrix of all control inputs from $\boldsymbol{u}(0) \ldots \boldsymbol{u}(N-1)$. The resultant trajectories from each problem are shown in Figure 13. Figures 5-10 show the resultant position, velocity, and thrust for **Problem 4** and **Problem 5**. As expected, both problems admit feasible trajectories which drive the vehicle to the origin with zero velocity at the terminal step.

Similarly, Figure 11 compares fuel usage from each optimization problem. Here, it is seen that **Problem 4** clearly minimizes fuel usage, compared to the relaxed **Problem 5** (as expected). The terminal mass of **Problem 4** vs. **Problem 5** is 1535.32kg vs. 1149.54kg, respectively. **Problem 5** clearly violates physical limits, as the vehicle's dry mass is 1505kg meaning the vehicle ran out of propellant with the trajectory generated. While not a physically relevant comparison (since **Problem 5** does not consider thrust constraints), this test did show that the algorithm prescribed by [4] [3] satisfies the constraints highlighted in previous sections while minimizing fuel consumption.

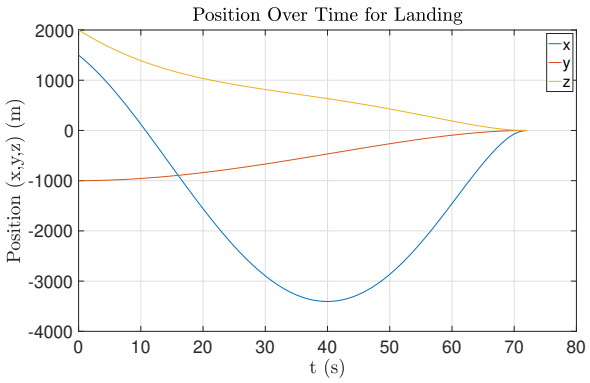

Fig. 5. Position over time for **Problem 5**

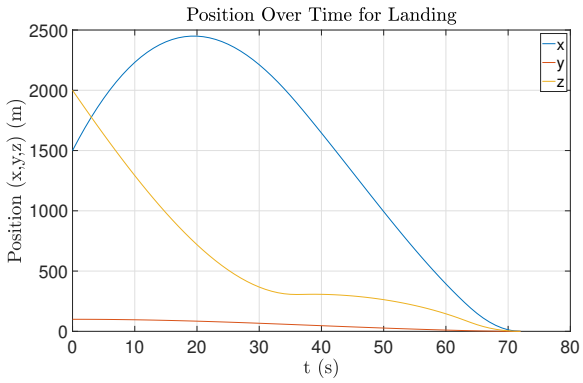

Fig. 8. Position over time for **Problem 4**

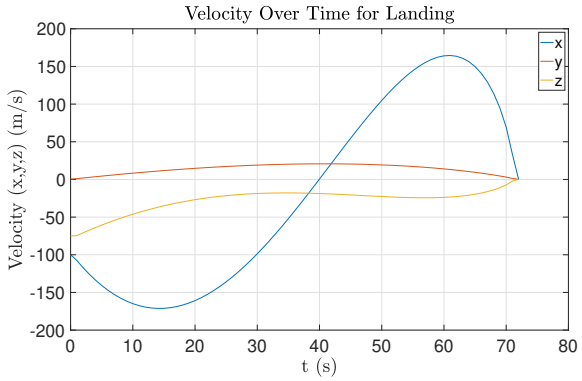

Fig. 6. Velocity over time for **Problem 5**

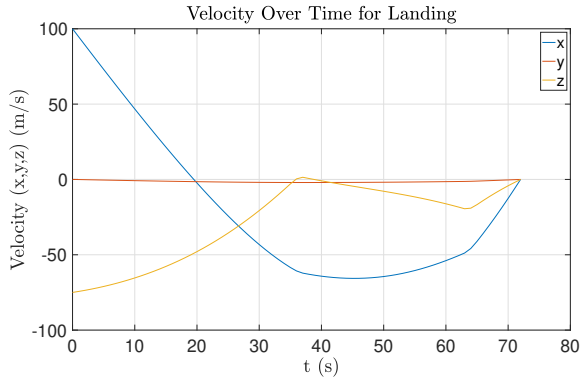

Fig. 9. Velocity over time for **Problem 4**

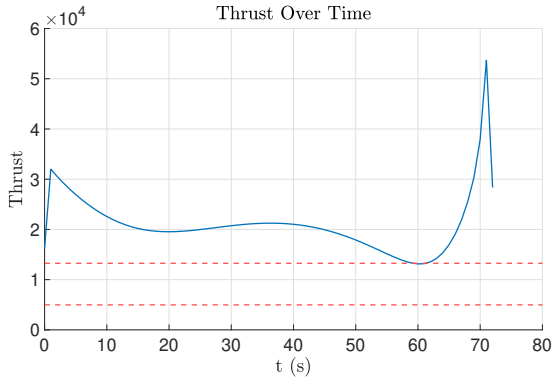

Fig. 7. Plot of vehicle thrust over time for **Problem 5**, with max/min throttle limits denoted in red. Note that since there is no constraint on thrust in the simplified problem, the thrust limits are violated.

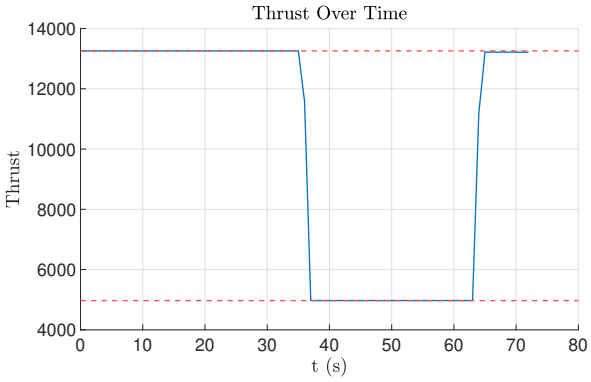

Fig. 10. Plot of vehicle thrust (N) over time for **Problem 4**, here the thrust limits are shown to be satisfied, as expected.

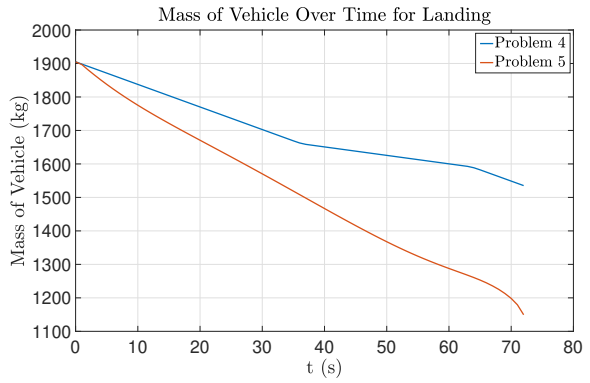

Fig. 11. Plot showing a comparison of mass consumption of each vehicle. While **Problem 5** admits a feasible trajectory, it is shown that the mass consumption exceeds physical limits, $m_{\text{dry}} = 1505$kg. **Problem 4**, however allows for margin (with a final wet mass of 1537.9kg)

B. Time of Flight Search

Previous sections described that the golden-search algorithm was used to find an optimal t_f^*. Here, MATLAB's fminbnd function was used to run the line-search algorithm specified by [3], bounded min/max final times as determined by (26). Figure 14 shows testing that shows different objective function values (fuel consumption) for varying times of t_f based on the initial conditions in Table I. fminbnd is able to find a globally optimal solution. It is of note, running on a desktop computer, that this search takes an average of 120 seconds to complete, which is a significant detractor for running this algorithm online on an embedded platform.

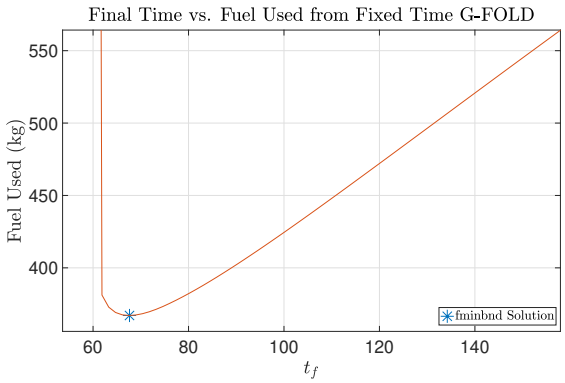

Fig. 14. Plot showing fuel used from varying t_f and the resultant solution from fminbnd at the global minimum, where $t_f^* = 67.6342$s

C. Generation of Fuel Optimal Trajectory

At this point, G-FOLD can be used to numerically solve a fuel-optimal trajectory problem, given a set of initial conditions where the final time is unknown. The following procedure highlights how G-FOLD is used to generate a fuel-optimal trajectory to be used in a closed-loop simulation:

1) Determine $t_{f,min}$, $t_{f,max}$ from initial conditions, and (26)
2) Using fminbnd, determine the optimal t_f^* which minimizes the objective function in **Problem 4** over $t_f \in [t_{f,min}, t_{f,max}]$
3) Re-solve **Problem 4** using t_f^*, to produce a optimal trajectory, and save $\boldsymbol{r}^*$, $\boldsymbol{v}^*$, $\boldsymbol{u}^*$, $\boldsymbol{m}^*$
4) Based on the update rate of the simulation, interpolate the trajectory using the MATLAB spline interpolation function spapi

Note that Step 4 is necessary above since the number of waypoints (discrete steps) used by CVX is much smaller than the update rate of what may be used by a simulation. For this report, $N = 50$ waypoints were used.

D. 3-DOF Simulation Overview

The 3-DOF simulation developed utilizes a solved fuel-optimal trajectory and closed-loop integration with trajectory following algorithm (trajectory controller) subject to non-linear dynamics, with injected noise on state inputs running at 100Hz. Figure 15 shows an overview of the 3-DOF simulation used to test G-FOLD's performance.

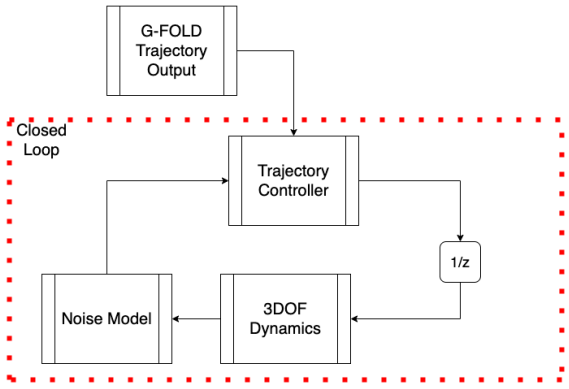

Fig. 15. Simplified block diagram of 3DOF simulation

The components of the simulation are described below:

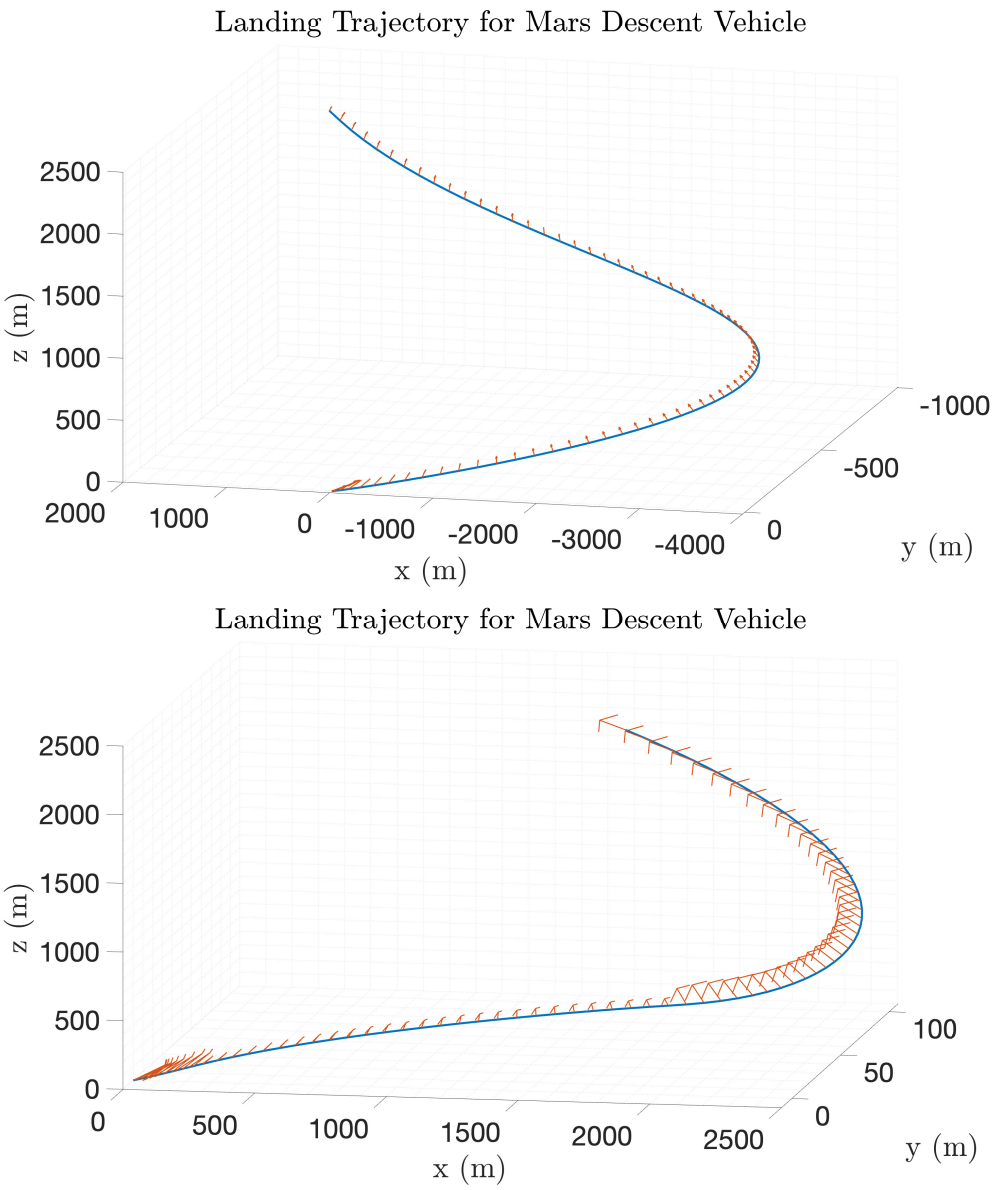

Fig. 13. Comparison of trajectories generated from **Problem 5** (top) and **Problem 4** (bottom). Red arrows show the thrust direction of the vehicle as it descends to the target landing site.

1) 3-DOF Dynamics Model: A simplified 3-DOF dynamics model is developed and integrated using MATLAB's ode45 solver. The state vector is defined as:

$$\boldsymbol{X}(t) = \begin{bmatrix} \boldsymbol{r}(t) \\ \dot{\boldsymbol{r}}(t) \\ m(t) \end{bmatrix} \tag{27}$$

where $\boldsymbol{X}$ is initialized to $\boldsymbol{X}(0) = [\boldsymbol{r}_0, \boldsymbol{v}_0, m_{\text{wet}}]^\intercal$ as defined in Table I. The dynamics equations are defined as:

$$\dot{\boldsymbol{X}}(t) = \begin{bmatrix} \dot{\boldsymbol{r}}(t) \\ \ddot{\boldsymbol{r}}(t) \\ \dot{m} \end{bmatrix} \tag{28}$$

where,

$$\ddot{\boldsymbol{r}}(t) = \boldsymbol{a}_{\text{thrust}} + \boldsymbol{g} \tag{29}$$

$$\dot{m}(t) = -\alpha \left\| \boldsymbol{u}(t) \right\| \tag{30}$$

$$\boldsymbol{a}_{\text{thrust}} = \frac{\boldsymbol{u}(t)}{m(t)} \tag{31}$$

Here, $\boldsymbol{u}(t)$ is the output thrust command generated at a given time t from the trajectory controller.

2) Noise Model: A simplified noise model was generated, adding Gaussian noise to the current state $\boldsymbol{X}$. The configuration used for simulation results is shown in Table II.

TABLE II
NOISE MODEL CONFIGURATION FOR 3-DOF SIMULATION

Parameter	1-σ Noise
$\boldsymbol{r}$	$[0.01, 0.01, 0.01]^\intercal$ m
$\dot{\boldsymbol{r}}$	$[0.002, 0.002, 0.002]^\intercal$ m/s
m	0.01 kg

3) Trajectory Controller: A PD control law was developed which leverages the solved optimal trajectory from G-FOLD as a feed-forward control channel. Here the control law is defined as follows:

$$\boldsymbol{u}(t) = m(t)(K_p \boldsymbol{e}_r + K_d \boldsymbol{e}_v) + \boldsymbol{u}_{ff}(t) \tag{32}$$

where the error terms $\boldsymbol{e}_r$ and $\boldsymbol{e}_v$ are the difference between the reference trajectory and the current vehicle state.

$$\boldsymbol{e}_r = \boldsymbol{r}^*(t) - \boldsymbol{r}(t) \tag{33}$$

$$\boldsymbol{e}_v = \boldsymbol{v}^*(t) - \boldsymbol{v}(t) \tag{34}$$

The feed-forward term $\boldsymbol{u}_{ff}$ is defined from the reference trajectory at time t where:

$$\boldsymbol{u}_{ff}(t) = m^*(t)\boldsymbol{a}^*(t) \tag{35}$$

Thrust limits were enforced by the addition of saturation logic which limits control output based on the vehicle configuration. In addition, a relaxation of 1% was used to help with tracking errors near the terminal stage of the simulation. From testing, gains of $K_p = 0.10$ and $K_d = 0.05$ were chosen.

E. Single Shot 3-DOF Simulation Results

The 3-DOF simulation yielded results that showed good correspondence with the open-loop trajectory shown in previous sections. Figures 16 and 17 shows the tracking error of the trajectory controller over the duration of the simulation. The tracking error was held to within ± 1 m and ± 0.5 m/s for position and velocity channels, respectively. Figure 18 shows the thrust profile of the vehicle, generally staying within vehicle thrust constraints (with the relaxation of about 1% as prescribed).

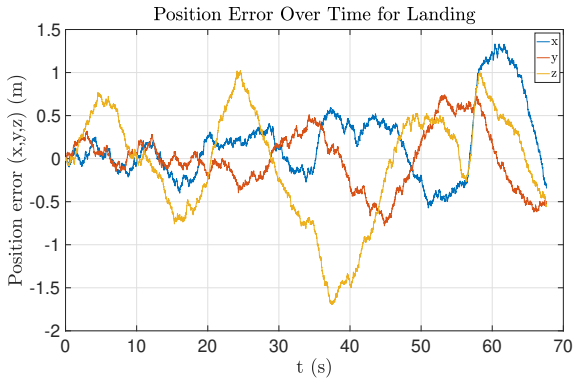

Fig. 16. Position tracking error from 3-DOF simulation

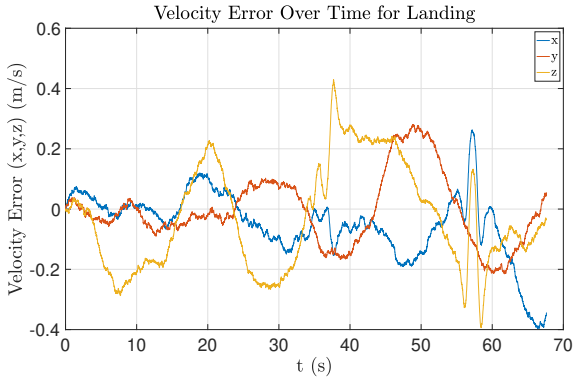

Fig. 17. Velocity tracking error from 3-DOF simulation

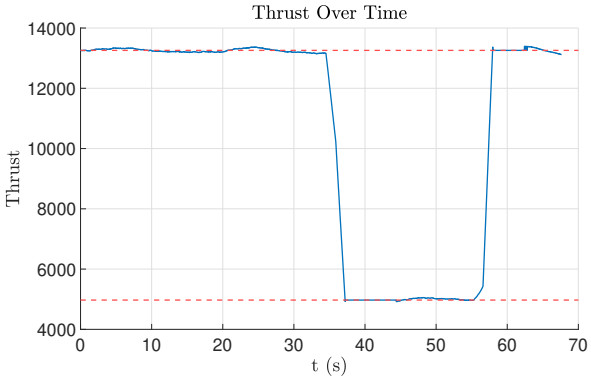

Fig. 18. Vehicle thrust over time from 3-DOF simulation

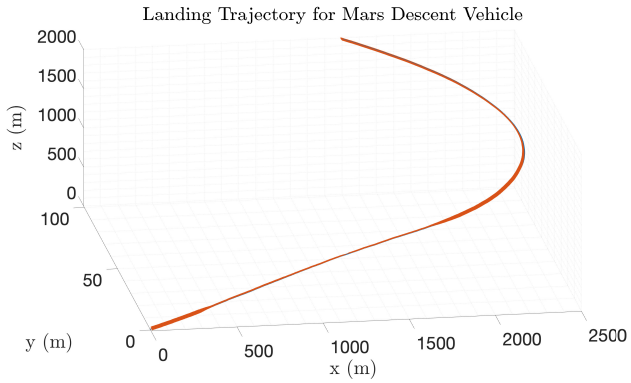

Fig. 19. Trajectory of a mars descent vehicle from 3-DOF simulation data

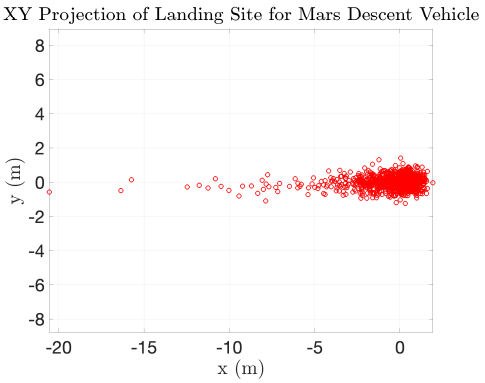

Fig. 20. XY projection of landing dispersions of Mars Descent Vehicle

F. Monte Carlo Results

A Monte Carlo analysis was performed to test the fidelity of the simulation. Parameters were varied over $N = 1000$ runs to understand better the trajectory controller's performance and the fidelity of G-FOLD's solution algorithm. Figure 20 shows that the landing error is generally centered on the origin (to with $\pm 1.7m$), but in some cases, the landing error was observed to be up to 20m. Figures 21 and 22 show the position tracking error and velocity tracking error are bounded by up to $\pm$20m and ± 6 m/s respectively. Table III shows that the dispersions from Monte Carlo analysis show good correspondence to the reference trajectory. However, there are still a non-negligible amount of cases where significant position/velocity errors may be intolerable. Note also that the noise configuration for this simulation denotes that the vehicle has fairly good knowledge of its state vector, which may not be the case in reality. More conservative configurations tended to yield high instances of "crashing" (i.e., missing the landing objective and violating the terminal velocity constraints) at the termination of the simulation, so smaller noise bounds were selected. This is mainly a reflection on the low-fidelity control scheme chosen for this simulation and the inability of the current implementation to re-plan in the event of larger disturbances.

TABLE III
COMPARISON OF TERMINAL STATE FROM MONTE CARLO DATA AND REFERENCE TRAJECTORY

Parameter	Reference Trajectory	Monte Carlo Results
$\left\|\boldsymbol{r}_f\right\|$	0 m	1.2666 ± 1.7036m
$\left\|\boldsymbol{v}_f\right\|$	0 m/s	0.3044 ± 0.4191m/s
m_f	1537.9 kg	1538.2 ± 0.9205kg

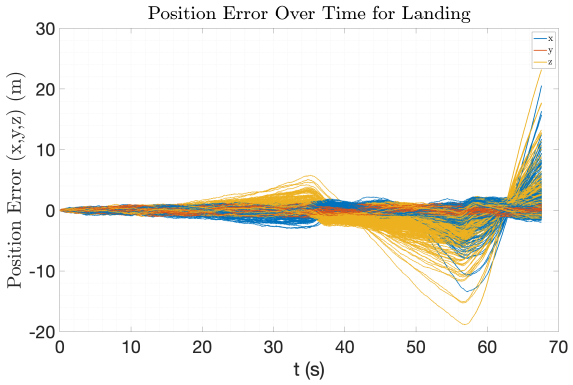

Fig. 21. Position tracking error from Monte Carlo runs

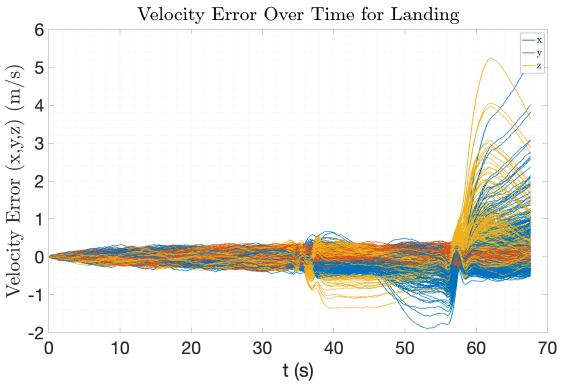

Fig. 22. Velocity tracking error from Monte Carlo runs

IV. CONCLUSION/FUTURE WORK

In this report, methods described by [3] [4] for pin-point landing were successfully implemented and tested in CVX using the MATLAB framework. An approach to generate a fuel-optimal trajectory for soft landing was developed and leveraged as an input for an integrated

3-DOF simulation. Furthermore, an accompanying trajectory following algorithm was successfully developed, illustrating a step towards a high-fidelity implementation of G-FOLD, which can be run on an embedded platform. This implementation was further tested in Monte Carlo analysis, and results showed good correspondence to the ideal solution.

In future iterations, the 3-DOF simulation may be extended to include accurate sensor models and a navigation filter to improve on the simplified noise model highlighted in previous sections. A 6-DOF simulation can later be developed, providing a better emulation of an autonomous vehicle. Furthermore, the control and planning scheme can significantly be improved. Recent works have improved the TOF search by either using lookup tables to cut down on search time or improving solvers optimized for this problem domain. While simple, the PD controller scheme is myopic because it cannot re-plan or react to significant disturbances at run-time. PD (or PID) control does not handle hard constraints necessary for this problem domain. Improved controller schemes, such as Model Predictive Control (MPC), have been shown in [6] to mitigate the shortcomings of simpler proportional control laws illustrated in this work. Finally, this work can be extended to leverage modern interior point solvers and code generation tools, allowing this algorithm to be run on embedded platforms, extending the work done in [1].

REFERENCES

[1] A. Behçet, "Flight testing of trajectories computed by g-fold: Fuel optimal large divert guidance algorithms for planetary landing," *AIAA Journal of Guidance, Control and Dynamics*, 2012.

[2] D. Dueri, "Automated custom code generation for embedded, real-time second order cone programming," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 1605–1612, 2014.

[3] A. Behçet, "Convex programming approach to powered descent guidance for mars landing," *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 5, 2007.

[4] A. Behçet, "Enhancements on the convex programming based powered descent guidance algorithm for mars landing," in *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, AIAA, 18 August 2008 - 21 August 2008.

[5] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, 2000.

[6] J. Wang, "Optimal rocket landing guidance using convex optimization and model predictive control," *Journal of Guidance, Control, and Dynamics*, vol. 42, no. 5, 2019.

V. Appendix

The following sections show the MATLAB code used in this report:

A. *runMarsLandingSimulation.m*

```
%% Simulate Mars Lander
%  Top level simulation function for Mars Landing Simulation using G-FOLD

%% Set initial conditions
cfg               = struct();

nx                = 3;
% earth gravity
cfg.g0            = 9.80665;
% vehicle dry mass
cfg.dry_mass      = 1505;
% vehicle ISP
cfg.ISP           = 225;
% vehicle max throttle
cfg.max_throttle  = 0.8;
% vehicle min throttle
cfg.min_throttle  = 0.3;
% mars gravity vector [m/s^2]
cfg.g             = [0 ;0; -3.7114];
% Max total thrust force at 1.0 throttle [N]
cfg.T_max         = 6 * 3100;
% The cant angle of thrusters [deg]
cfg.phi            = 27*deg2rad(1);
% n thrusters
cfg.nThruster      = 1;
% glide slope constraint
cfg.gamma          = 10*deg2rad(1);

% Vehicle mass with fuel [kg]
cfg.wet_mass       = 1905;
theta              = 180*deg2rad(1);

% thrust limits
rho1    = (cfg.min_throttle * cfg.T_max) *cos(cfg.phi);
rho2    = (cfg.max_throttle * cfg.T_max) *cos(cfg.phi);

cfg.rho1 = rho1;
cfg.rho2 = rho2;

%% Set initial conditions
r0 = [1.5,  0.1, 2]' .* 1e3;
v0 = [100 ,0.01, -75]';
%% Terminal conditions (make this the origin)
rf = [0, 0, 0]';
vf = [0, 0, 0]';

% number of waypoints
N  = 50;
% controller parameters (PD controller)
cfg.Kp = 0.1;
cfg.Kd = 0.05;
% safety factor for violating controller constraints
cfg.controllerPctMargin = 0.01;
% simulation frequency (100Hz)
cfg.dtSim               = 0.01;

%% Sensor Noise
cfg.posSigma  = 0.01;
cfg.velSigma  = 0.002;
cfg.massSigma = 0.01;

```

```
%% MC Settings
cfg.runMCSim = true;
cfg.nRuns    = 1000;


%% Run Simulation
[tfStar,fuelUsed,r,v,u,m] = runGFOLD(N,r0,v0,rf,vf,cfg.wet_mass,theta,cfg);
%
% % interpolate trajectory based on waypoints, and get an object with
% % necessary spline functions
TObj = interpolateTrajectory(N,tfStar,fuelUsed,r,v,u,m,cfg);

%% run MSL landing simulation with trajectory following
[dataOut] = simulateMSLDynamics(r0,v0,cfg.dtSim,tfStar,TObj,cfg);
% Plot results
plotResults(dataOut,cfg);

dataOuts = {};

cfgNom = cfg;
if cfg.runMCSim

    cfg = cfgNom;

    parfor_progress(cfg.nRuns);

    dtSim = cfg.dtSim;

     posSigma  = cfg.posSigma;
     velSigma  = cfg.velSigma;

    parfor ii=1:cfg.nRuns
        % perturb initial conditions
        r0 = [1.5,  0.1, 2]' .* 1e3 +  normrnd(0, posSigma, 3, 1);
        v0 = [100 ,0.01, -75]'  + normrnd(0, velSigma, 3, 1);
        %% Run Simulation
        [tfStar,fuelUsed,r,v,u,m] = runGFOLD(N,r0,v0,rf,vf,cfg.wet_mass,theta,cfg);

        % % interpolate trajectory based on waypoints, and get an object with
        % % necessary spline functions
        TObj = interpolateTrajectory(N,tfStar,fuelUsed,r,v,u,m,cfg);

        [dataOuts{ii}] = simulateMSLDynamics(r0,v0,dtSim,tfStar,TObj,cfg);
        parfor_progress;
    end
    plotMCResults(dataOuts,cfg);

    parfor_progress(0);

end
```

B. *runGFOLD.m*

```
    function [tfStar,fuelUsed,r,v,u,m] = runGFOLD(N, r0, v0, rf, vf, wet_mass, theta, cfg)
%% runGFOLD Runs the G-FOLD algorithm
%  runs TOF search and then solves fixed time problem

% extract constant parameters
g0           = cfg.g0;
ISP          = cfg.ISP;
dry_mass     = cfg.dry_mass;

min_throttle = cfg.min_throttle;
max_throttle = cfg.max_throttle;
phi          = cfg.phi;
T_max        = cfg.T_max;
```

```
14
15 % calculate derived parameters
16 alpha   = 1/(ISP * g0 * cos(phi));
17 rho1    = (min_throttle * T_max) *cos(phi);
18 rho2    = (max_throttle * T_max) *cos(phi);
19
20 cfg.rho1 = rho1;
21 cfg.rho2 = rho2;
22
23
24 % create bounding cases for finding optimal time
25 % min final time
26 % min case for GFOLD (use max throttle)
27 min_tf = wet_mass * norm(vf - v0) / rho2;
28 % max final time
29 % worst case for time for GFOLD
30 max_tf = (wet_mass - dry_mass) / (alpha * rho1);
31
32 % first find the optimal time for G-FOLD to commit to a landing
33 % based on the input parameters
34
35 % use FMINBND to find the optimal time for G-FOLD, and then solve
36
37 % first declare objective function (which is fixed time problem 4 from
38 % paper)
39 objFcn      = @(t) fixedTimeGFOLD(t,N,r0,v0,rf,vf,wet_mass,theta,cfg);
40
41 fminOptions = optimset('TolX',0.5, 'Disp','iter');
42
43 % find time which minimizes objective
44 tfStar = fminbnd(objFcn, min_tf, max_tf, fminOptions);
45
46 % now that we have the optimal final time, run GFOLD to get the output
47 % trajectory
48 [fuelUsed,r,v,u,m] = fixedTimeGFOLD(tfStar, N, r0, v0, rf, vf, wet_mass, theta, cfg);
49
50 end
```

C. *fixedTimeGFOLD.m*

```
1  function [fuel_used,r,v,u,mass] = fixedTimeGFOLD(t,N,r0,v0,rf,vf,wet_mass,theta,cfg)
2  %FIXEDTIMEGFOLD Solves fixed-time G-FOLD problem
3  %               first solves for optimal tfStar, then re-solves fixed
4  %               time G-FOLD problem to get fuel-optimal trajectory based on IC
5
6  % number of dimensions
7  nx = 3;
8  % get dt from N and final time
9  dt = t/(N-1);
10
11 % extract constant parameters
12 g0           = cfg.g0;
13 ISP          = cfg.ISP;
14 g            = cfg.g;
15 gamma        = cfg.gamma;
16
17 min_throttle = cfg.min_throttle;
18 max_throttle = cfg.max_throttle;
19 phi          = cfg.phi;
20 T_max        = cfg.T_max;
21
22 % calculate derived parameters
23 alpha   = 1/(ISP * g0 * cos(phi));
24 rho1    = (min_throttle * T_max) * cos(phi);
25 rho2    = (max_throttle * T_max) * cos(phi);
26
```

```
% always use SEDUMI since it yields the best results

% solve problem 4 from paper (discretized fuel minimization problem)
cvx_begin QUIET

    % define variables from problem
    variable r(nx,N)
    variable v(nx,N)
    variable u(nx,N)
    variable z(1,N)
    variable sig(1,N)

    % for this problem we'd like to minimize the fuel usage
    % i.e. maximize terminal fuel value
    minimize( -z(N) )

    subject to

    % initial condition constraints
    r(:,1) == r0;
    v(:,1) == v0;
    z(:,1) == log(wet_mass);
    % terminal constraints
    r(:,N) == rf;
    v(:,N) == vf;

    % now enforce constraints at each timestep
    for k = 1:N
        % enforce thrust constraint
        norm(u(:,k)) <= sig(:,k);

        % use inner terms due to simplification of log when evaluating mu
        z0_innerTerm   = wet_mass - alpha*rho2*(k-1)*dt;
        zupper_comp    = wet_mass - alpha*rho1*(k-1)*dt;

        z0             = log(z0_innerTerm);
        zupper         = log(zupper_comp);

        % calculate mu
        % note exp(-log(term) ) -> 1/term
        mu1            = rho1/z0_innerTerm;
        mu2            = rho2/z0_innerTerm;

        % enforce sigma
        sig(:,k)      >= mu1*(1 - (z(:,k) - z0) + (1/2)*(z(:,k) - z0)^2 );
        sig(:,k)      <= mu2*(1 - (z(:,k) - z0) );

        % enforce zdt
        z(:,k)        >= z0;
        z(:,k)        <= zupper;

        % glide slope
        norm([r(1,k) ; r(2,k)]) <= tan(gamma) * r(3,k);

    end

    % at each timestep, enforce dynamics constraints
    for k = 1:(N-1)
        % use update equations for position and velocity based on paper

        % update velocity
        v(:,k+1) == v(:,k) + (dt/2)*(u(:,k) + u(:,k+1)) + g*dt;

        % update position
        r(:,k+1) == r(:,k) + (dt/2)*(v(:,k) + v(:,k+1)) ...
                        + (dt^2/12)*(u(:,k+1) - u(:,k));

        % update z = ln(wet_mass)
        z(:,k+1) == z(:,k) - (alpha*dt/2)*(sig(k+1) + sig(k));
```

```
    end

    % enforce that we don't have feasible trajectories under the surface
    r(3,1:N-1) ≥ -1e-3;

cvx_end

% init to wet mass
fuel_used = wet_mass;
mass      = wet_mass;

% calculate mass if we converged:
if strcmp(cvx_status, 'Solved')
    % use short hand to get mass property
    mass = exp(z);
    % fuel used is initial minus final
    fuel_used = mass(1) - mass(N);

% if we have an infeasibile trajectory, just output the same fuel used
elseif strcmp(cvx_status, 'Infeasible')
    fuel_used = wet_mass;
else
    fprintf('ERROR: WAS NOT ABLE TO SOLVE: %s', cvx_status);
end


end
```

D. *interpolateTrajectory.m*

```
function [S] = interpolateTrajectory(N,tfStar,fuelUsed,r,v,u,m,cfg)

% based on optimal t, get the timeline for trajectory
ts = linspace(0, tfStar, N);
% now interpolate trajectory based on the number of waypoints to allow for
% a 100Hz update
S.r = spapi(4, ts, r);
S.v = spapi(3, ts, v);
S.a = spapi(2, ts, u);
S.m = spapi(2, ts, m);

end
```

E. *simulateMSLDynamics.m*

```
function [dataOut] = simulateMSLDynamics( r0, v0, dt, tfStar, S, cfg)
%% simulateMSLDynamics
%  simulates MSL dynamics and trajectory following

% get number of iterations
N    = ceil(tfStar/dt);
tVec = 0:dt:tfStar;

% get initial state
X0 = [r0; v0; cfg.wet_mass];

% get initial control input
[u0,errorPosk,errorVelk] = mslTrajectoryController(X0,0,S,cfg);

% initialize Xk and uk
Xk = X0;
uk = u0;
```

```
18
19 Xs(1,:) = Xk';
20 Us(1,:) = uk';
21 errorPos(1,:) = errorPosk';
22 errorVel(1,:) = errorVelk';
23
24 for kk=2:N
25     tk    = tVec(kk);
26
27     tspan = [0 dt];
28
29     % run ODE integration
30     [¬,XMatk] = ode45(@(t,X) integrateMSLDynamics(t,X,uk,cfg),tspan,Xk);
31
32     % collect ouputs
33     Xkp1 = XMatk(end,:)';
34
35     % set for next cycle
36     Xk = Xkp1;
37
38     % add noise (TODO add sensor models)
39     Xk = addMeasurementNoise(Xk,cfg);
40
41     % run controller
42     [uk, errPosk, errVelk] = mslTrajectoryController(Xk,tk,S,cfg);
43
44     % save history
45     Xs(kk,:) = Xk';
46     Us(kk,:) = uk';
47     errorPos(kk,:) = errPosk';
48     errorVel(kk,:) = errVelk';
49
50 end
51
52 % package data
53 dataOut          = struct();
54 dataOut.Xs       = Xs;
55 dataOut.Us       = Us;
56 dataOut.errorPos = errorPos;
57 dataOut.errorVel = errorVel;
58 dataOut.tVec     = tVec;
59
60
61 end
```

F. integrateMSLDynamics.m

```
1  function [Xdot] = integrateMSLDynamics(t,X,u,cfg)
2  %% integrateMSLDynamics Integration function of 3DOF dynamics
3  %  @param t:    Required time parameter for ODE function
4  %  @param X:    7x1 vector representing the position and velocity of the MSL
5  %               vehicle, with the current mass of the vehicle, where X =[x,v,m]'
6  %  @param u:    control input
7  %  @param cfg:  configuration for the vehicle
8
9
10 % extract state
11 x = X(1:3);
12 v = X(4:6);
13 m = X(7);
14
15 % calculate mdot
16 % extract params
17 g0    = cfg.g0;
18 ISP   = cfg.ISP;
19 phi   = cfg.phi;
```

```
alpha = 1/(ISP * g0 * cos(phi));
% mass always negative
mDot  = -alpha * norm(u);

% calculate acceleration
aThrust  = u./m;

a = aThrust + cfg.g;

Xdot = [v; a; mDot];

end
```

G. addMeasurementNoise.m

```
function [Xkp] = addMeasurementNoise(Xk,cfg)
%addMeasurementNoise Adds measurement noise to current state
posSigma  = cfg.posSigma;
velSigma  = cfg.velSigma;
massSigma = cfg.massSigma;

Xkp       = zeros(7,1);
Xkp(1:3)  = Xk(1:3) + normrnd(0, posSigma, 3, 1);
Xkp(4:6)  = Xk(4:6) + normrnd(0, velSigma, 3, 1);
Xkp(7)    = Xk(7) + normrnd(0, massSigma, 1, 1);

end
```

H. problem4.m

```
%% The following shows the solutions using numeric solvers to
%  the modifications made to the G-FOLD algorithm, using the reference:

% Convex programming approach to powered descent guidance for mars landing."
% Acikmese, Behcet, and Scott R. Ploen.

clear all
close all

%% Problem 4 from Paper:

%% Set initial conditions

nx             = 3;

% earth gravity
g0             = 9.80665;
% vehicle dry mass
dry_mass       = 1505;
% vehicle ISP
Isp            = 225;
% vehicle max throttle
max_throttle   = 0.8;
% vehicle min throttle
min_throttle   = 0.3;
% mars gravity vector [m/s^2]
g              = [0 ;0; -3.7114];
% Max total thrust force at 1.0 throttle [N]
T_max          = 6 * 3100;
% The cant angle of thrusters [deg]
phi            = 27*deg2rad(1);
% n thrusters
```

```
nThruster      = 1;
% glide slope constraint
gamma          = 4*deg2rad(1);

% Vehicle mass with fuel [kg]
wet_mass       = 1905;

%% Initial conditions
% note for now this is in x,z (TODO make this x,y,z)
r0 = [1.5,  0.1, 2]' .* 1e3;
v0 = [100 ,0.01, -75]';
%% Terminal conditions (make this the origin)
rf = [0, 0, 0]';
vf = [0, 0, 0]';

% Determine TOF
% Time that was used in the paper
tf = 72;
dt = 1;

N  = ceil(tf / dt)+1;
ts = 0:dt:tf;

%% Solve problem 4 from paper

% Define intermediate variables
alpha   = 1/(Isp * g0 * cos(phi));
rho1    = (min_throttle*T_max) *cos(phi);
rho2    = (max_throttle*T_max) *cos(phi);


%% Perform Solve with CVX

cvx_begin

    % define variables from problem
    variable r(nx,N)
    variable v(nx,N)
    variable u(nx,N)
    variable z(1,N)
    variable sig(1,N)

    % for this problem we'd like to minimize the fuel usage
    % i.e. maximize terminal fuel value
    minimize( -z(N) )

    subject to

    % initial condition constraints
    r(:,1) == r0;
    v(:,1) == v0;
    z(:,1) == log(wet_mass);
    % terminal constraints
    r(:,N) == rf;
    v(:,N) == vf;

    % now enforce constraints at each timestep
    for k = 1:N
        % enforce thrust constraint
        norm(u(:,k)) <= sig(:,k);

        % use inner terms due to simplification of log when evaluating mu
        z0_innerTerm   = wet_mass - alpha*rho2*(k-1)*dt;
        zupper_comp    = wet_mass - alpha*rho1*(k-1)*dt;

        z0             = log(z0_innerTerm);
        zupper         = log(zupper_comp);

        % calculate mu
```

```
        % note exp(-log(term) ) -> 1/term
        mu1             = rho1/z0_innerTerm;
        mu2             = rho2/z0_innerTerm;

        % enforce sigma
        sig(:,k)      >= mu1*(1 - (z(:,k) - z0) + (1/2)*(z(:,k) - z0)^2 );
        sig(:,k)      <= mu2*(1 - (z(:,k) - z0) );

        % enforce z
        z(:,k)        >= z0;
        z(:,k)        <= zupper;

        % glide slope
        norm([r(1,k) ; r(2,k)]) <= tan(gamma) * r(3,k);

    end

    % at each timestep, enforce dynamics constraints
    for k = 1:(N-1)
        % use update equations for position and velocity based on paper

        % update velocity
        v(:,k+1) == v(:,k) + (dt/2)*(u(:,k) + u(:,k+1)) + g*dt;

        % update position
        r(:,k+1) == r(:,k) + (dt/2)*(v(:,k) + v(:,k+1)) ...
                        + (dt^2/12)*(u(:,k+1) - u(:,k));

        % update z = ln(wet_mass)
        z(:,k+1) == z(:,k) - (alpha*dt/2)*(sig(k+1) + sig(k));
    end

    % enforce that we don't have feasible trajectories under the surface
    r(3,:) >= -0.001;

    slope = 10;


cvx_end

m      = exp(z);

norm_u = vecnorm(m.*u,2,1);

figure()
plot(ts, r(1,:),'LineWidth',2);
hold on;
plot(ts, r(2,:),'LineWidth',2);
hold on;
plot(ts, r(3,:),'LineWidth',2);
xlabel('t (s)','FontSize',20, 'Interpreter','latex')
ylabel('Position (x,y,z) (m)','FontSize',20, 'Interpreter','latex')
legend({'x','y','z'})
title('Position Over Time for Landing','FontSize',20, 'Interpreter','latex')
grid on;
set(gca,'fontsize',40)

figure()
yline(T_max*max_throttle*cos(phi),'r--','LineWidth',3)
hold on;
yline(T_max*min_throttle*cos(phi),'r--','LineWidth',3)
hold on;
plot(ts, norm_u,'LineWidth',3);
xlabel('t (s)','FontSize',20, 'Interpreter','latex')
ylabel('Thrust','FontSize',20, 'Interpreter','latex')
title('Thrust Over Time','FontSize',22, 'Interpreter','latex')
grid on;
set(gca,'fontsize',40)
```

```
figure()
plot(ts, v(1,:),'LineWidth',2);
hold on;
plot(ts, v(2,:),'LineWidth',2);
hold on;
plot(ts, v(3,:),'LineWidth',2);
xlabel('t (s)','FontSize',20, 'Interpreter','latex')
ylabel('Velocity (x,y,z) (m/s)','FontSize',20, 'Interpreter','latex')
legend({'x','y','z'})
title('Velocity Over Time for Landing','FontSize',20, 'Interpreter','latex')
grid on;
set(gca,'fontsize',40)


figure();
plot3(r(1,:), r(2,:),r(3,:),'LineWidth',3);
hold on;
quiver3(r(1,:), r(2,:),r(3,:),u(1,:), u(2,:),u(3,:), 0.4,'LineWidth',1.5)
xlabel('x (m)','FontSize',20, 'Interpreter','latex')
ylabel('y (m)','FontSize',20, 'Interpreter','latex')
zlabel('z (m)','FontSize',20, 'Interpreter','latex')
title('Landing Trajectory for Mars Descent Vehicle','FontSize',20, 'Interpreter','latex')
grid minor;
set(gca,'fontsize',40)

figure()
plot(ts, m,'LineWidth',3)
xlabel('t (s)','FontSize',20, 'Interpreter','latex')
ylabel('Mass of Vehicle (kg)','FontSize',20, 'Interpreter','latex')
titleStr = ['Mass of Vehicle Over Time for Landing'];
title(titleStr,'FontSize',22, 'Interpreter','latex')
grid on;
set(gca,'fontsize',40)
```

I. problem5.m

```
%% The following shows the solutions using numeric solvers to
%  the modifications made to the G-FOLD algorithm, using the reference:

% Convex programming approach to powered descent guidance for mars landing."
% Acikmese, Behcet, and Scott R. Ploen.

clear all
close all

%% Problem 5 (simplified problem 4) from paper:

%% Set initial conditions
nx       = 3;

% earth gravity
g0       = 9.80665;
% vehicle dry mass
dry_mass = 1505;
% vehicle ISP
Isp      = 225;
% vehicle max throttle
max_throttle = 0.8;
% vehicle min throttle
min_throttle = 0.3;
% mars gravity vector [m/s^2]
g             = [0 ;0; -9.80665];
% Max total thrust force at 1.0 throttle [N]
T_max         = 6 * 3100;
% The cant angle of thrusters [deg]
phi           = 27*deg2rad(1);
```

```
% n thrusters
nThruster      = 1;
% glide slope constraint
gamma          = 10*deg2rad(1);

% Vehicle mass with fuel [kg]
wet_mass       = 1905;

%% Initial conditions
% note for now this is in x,z (TODO make this x,y,z)
r0 = [1.5,  -1, 2]' .* 1e3;
v0 = [-100 ,0.4, -75]';
%% Terminal conditions (make this the origin)
rf = [0, 0, 0]';
vf = [0, 0, 0]';

% Determine TOF
% Time that was used in the paper
tf = 72;
dt = 1;

N  = ceil(tf / dt)+1;
ts = 0:dt:tf;

%% Solve problem 1 from paper

% Define intermediate variables
alpha   = 1/(Isp * g0 * cos(phi));
rho1    = (min_throttle*T_max) *cos(phi);
rho2    = (max_throttle*T_max) *cos(phi);

cvx_begin

    % define variables from problem
    variable r(nx,N)
    variable v(nx,N)
    variable u(nx,N)

    % for this problem we'd like to minimize the control effort for
    % landing, subject to dynamics constraints
    minimize( square_pos( norm(u) ) )

    subject to

    % initial condition constraints
    r(:,1) == r0;
    v(:,1) == v0;
    % terminal constraints
    r(:,N) == rf;
    v(:,N) == vf;

    % at each timestep, enforce dynamics constraints
    for k = 1:(N-1)
        % use update equations for position and velocity based on paper

        % update velocity
        v(:,k+1) == v(:,k) + (dt/2)*(u(:,k) + u(:,k+1)) + g*dt;

        % update position
        r(:,k+1) == r(:,k) + (dt/2)*(v(:,k) + v(:,k+1)) ...
                        + (dt^2/12)*(u(:,k+1) - u(:,k));

    end

    % enforce that we don't have feasible trajectories under the surface
    r(3,:) ≥ 0;

cvx_end

```

```
100
101 u_norms = norms(u);
102 u_dirs = atan2(u(2,:), u(1,:));
103
104 % Compute approximate vehicle mass over time
105 m = zeros(1, N);
106 m(1) = wet_mass;
107 for k=1:N-1
108     T_k = u_norms(k) * m(k);
109     m_dot = -T_k / (Isp * g0 * cos(phi));
110     m(k+1) = m(k) + m_dot * dt;
111 end
112
113
114
115 norm_u = vecnorm(m.*u,2,1);
116
117 figure()
118 plot(ts, r(1,:),'LineWidth',2);
119 hold on;
120 plot(ts, r(2,:),'LineWidth',2);
121 hold on;
122 plot(ts, r(3,:),'LineWidth',2);
123 xlabel('t (s)','FontSize',20, 'Interpreter','latex')
124 ylabel('Position (x,y,z) (m)','FontSize',20, 'Interpreter','latex')
125 legend({'x','y','z'})
126 title('Position Over Time for Landing','FontSize',20, 'Interpreter','latex')
127 grid on;
128 set(gca,'fontsize',40)
129
130 figure()
131 yline(T_max*max_throttle*cos(phi),'r--','LineWidth',3)
132 hold on;
133 yline(T_max*min_throttle*cos(phi),'r--','LineWidth',3)
134 hold on;
135 plot(ts, norm_u,'LineWidth',3);
136 xlabel('t (s)','FontSize',20, 'Interpreter','latex')
137 ylabel('Thrust','FontSize',20, 'Interpreter','latex')
138 title('Thrust Over Time','FontSize',22, 'Interpreter','latex')
139 grid on;
140 set(gca,'fontsize',40)
141
142
143 figure()
144 plot(ts, v(1,:),'LineWidth',2);
145 hold on;
146 plot(ts, v(2,:),'LineWidth',2);
147 hold on;
148 plot(ts, v(3,:),'LineWidth',2);
149 xlabel('t (s)','FontSize',20, 'Interpreter','latex')
150 ylabel('Velocity (x,y,z) (m/s)','FontSize',20, 'Interpreter','latex')
151 legend({'x','y','z'})
152 title('Velocity Over Time for Landing','FontSize',20, 'Interpreter','latex')
153 grid on;
154 set(gca,'fontsize',40)
155
156
157 figure();
158 plot3(r(1,:), r(2,:),r(3,:),'LineWidth',3);
159 hold on;
160 quiver3(r(1,:), r(2,:),r(3,:),u(1,:), u(2,:),u(3,:), 0.4,'LineWidth',1.5)
161 xlabel('x (m)','FontSize',20, 'Interpreter','latex')
162 ylabel('y (m)','FontSize',20, 'Interpreter','latex')
163 zlabel('z (m)','FontSize',20, 'Interpreter','latex')
164 title('Landing Trajectory for Mars Descent Vehicle','FontSize',20, 'Interpreter','latex')
165 grid minor;
166 set(gca,'fontsize',40)
167
168 figure()
```

```
plot(ts, m,'LineWidth',3)
xlabel('t (s)','FontSize',20, 'Interpreter','latex')
ylabel('Mass of Vehicle (kg)','FontSize',20, 'Interpreter','latex')
titleStr = ['Mass of Vehicle Over Time for Landing'];
title(titleStr,'FontSize',22, 'Interpreter','latex')
grid on;
set(gca,'fontsize',40)
```

J. *tofSearchPlot.m*

```
%% simple script to generate plot for time of flight data

%% Set initial conditions
cfg               = struct();

nx                = 3;
% earth gravity
cfg.g0            = 9.80665;
% vehicle dry mass
cfg.dry_mass      = 1505;
% vehicle ISP
cfg.ISP           = 225;
% vehicle max throttle
cfg.max_throttle  = 0.8;
% vehicle min throttle
cfg.min_throttle  = 0.3;
% mars gravity vector [m/s^2]
cfg.g             = [0 ;0; -3.7114];
% Max total thrust force at 1.0 throttle [N]
cfg.T_max         = 6 * 3100;
% The cant angle of thrusters [deg]
cfg.phi            = 27*deg2rad(1);
% n thrusters
cfg.nThruster      = 1;
% glide slope constraint
cfg.gamma          = 10*deg2rad(1);

% Vehicle mass with fuel [kg]
cfg.wet_mass       = 1905;
theta              = 180*deg2rad(1);

% thrust limits
rho1    = (cfg.min_throttle * cfg.T_max) *cos(cfg.phi);
rho2    = (cfg.max_throttle * cfg.T_max) *cos(cfg.phi);

cfg.rho1 = rho1;
cfg.rho2 = rho2;

%% Set initial conditions
r0 = [1.5,  0.1, 2]' .* 1e3;
v0 = [100 ,0.01, -75]';
%% Terminal conditions (make this the origin)
rf = [0, 0, 0]';
vf = [0, 0, 0]';

% number of waypoints
N  = 50;
% controller parameters (PD controller)
cfg.Kp = 0.1;
cfg.Kd = 0.05;
% safety factor for violating controller constraints
cfg.controllerPctMargin = 0.01;
% simulation frequency (100Hz)
cfg.dtSim               = 0.01;

%% Sensor Noise
```

```
cfg.posSigma  = 0.01;
cfg.velSigma  = 0.002;
cfg.massSigma = 0.01;

%% MC Settings
cfg.runMCSim = false;
cfg.nRuns    = 100;


% extract constant parameters
g0           = cfg.g0;
ISP          = cfg.ISP;
dry_mass     = cfg.dry_mass;

min_throttle = cfg.min_throttle;
max_throttle = cfg.max_throttle;
phi          = cfg.phi;
T_max        = cfg.T_max;

% calculate derived parameters
alpha   = 1/(ISP * g0 * cos(phi));
rho1    = (min_throttle * T_max) *cos(phi);
rho2    = (max_throttle * T_max) *cos(phi);

cfg.rho1 = rho1;
cfg.rho2 = rho2;


% create bounding cases for finding optimal time
% min final time
% min case for GFOLD (use max throttle)
min_tf = wet_mass * norm(vf - v0) / rho2;
% max final time
% worst case for time for GFOLD
max_tf = (wet_mass - dry_mass) / (alpha * rho1);

dt = 1;
ts = linspace(min_tf,max_tf,100);

ms = [];

for tfk = ts
[fuelUsed,r,v,u,m] = fixedTimeGFOLD(tfk, N, r0, v0, rf, vf, wet_mass, theta, cfg);
ms = [ms;fuelUsed];
end


objFcn      = @(t) fixedTimeGFOLD(t,N,r0,v0,rf,vf,wet_mass,theta,cfg);

fminOptions = optimset('TolX',0.5, 'Disp','iter');

% find time which minimizes objective
tfStar = fminbnd(objFcn, min_tf, max_tf, fminOptions);

% now that we have the optimal final time, run GFOLD to get the output
% trajectory
[fuelStar,r,v,u,m] = fixedTimeGFOLD(tfStar, N, r0, v0, rf, vf, wet_mass, theta, cfg);


figure()
plot(tfStar, fuelStar,'*','MarkerSize',30,'LineWidth',2);
hold on;
plot(ts, ms,'LineWidth',2);
legend({'fminbnd Solution'},'FontSize',30, 'Interpreter','latex')
xlabel('$t_f$','FontSize',24, 'Interpreter','latex')
ylabel('Fuel Used (kg)','FontSize',24, 'Interpreter','latex')
title('Final Time vs. Fuel Used from Fixed Time G-FOLD','FontSize',24, 'Interpreter','latex')
grid on;
set(gca,'fontsize',40)
```