

A Comparative Study of Online Implementations of Powered Descent Guidance

Alexandra BOTEZ^{*,1}, Stefan Dragos DRAGAN¹, Tudorel-Petronel AFILIPOAE²,
Ana-Maria NECULAESCU¹

*Corresponding author

¹Agile Systems Engineering,

10 Ing. Zablovski, Bucharest, Romania,

alexandra.botez@agile-systems.ro*, stefan.dragan@agile-systems.ro

²INCAS – National Institute for Aerospace Research “Elie Carafoli”,

B-dul Iuliu Maniu 220, Bucharest, Romania,

afilipoae.tudorel@incas.ro

DOI: 10.13111/2066-8201.2026.18.3.4

Received: 15 July 2026/ Accepted: 31 July 2026/ Published: September 2026

Copyright © 2026. Published by INCAS. This is an “open access” article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>)

Abstract: *This paper presents a comparative study of three implementations of a Powered Descent Guidance (PDG) problem using the autogenerated C code capabilities of the following solvers: CVXGEN, OSQP and ACADOS. The objective of the fixed final time optimization problem is the minimization of the fuel consumption, within a convex framework for the first two solvers and using a nonlinear formulation for the last. The numerical constraints of the problem are tailored to model the capabilities of the Demonstrator for Technologies Validation (DTV) reusable platform developed and operated by INCAS.*

Key Words: *Powered Descent Guidance, Trajectory Optimization, Convex Programming, Nonlinear Programming, Vertical Landing, Rocket Reusability, Fuel Optimization, CVXGEN, OSQP, ACADOS, DTV*

1. INTRODUCTION

The reusability of launch vehicles remains a key driver in the evolution of modern space transportation, substantially lowering the cost of reaching orbit. Due to the strong coupling between translational and rotational channels and the uncertain nature of the process, a successful landing depends on the capability of the onboard computer to generate a new trajectory based on the current state estimations of the vehicle. Generating real-time guidance requires algorithms that can efficiently and robustly solve the underlying non-convex optimization problem, carefully considering a trade-off between computational resources and execution time.

This paper investigates three implementations for solving a 3-DoF PDG problem starting from core mathematical formulations presented in [1] and [2], taking into account the solvers specific limitations. Considering that all three solvers are capable of generating C code, the results analysis will be performed in the Simulink environment using S-Functions encapsulating the autogenerated C code. The objective of this study is to preliminarily assess the potential of each solver to be embedded on real-time applications hardware.

2. CVXGEN IMPLEMENTATION

CVXGEN [3] is a well-known powerful optimization tool used to solve an optimization problem family using Interior Point Methods (IPM), assuming it is defined according to the Disciplined Convex Programming (DCP) framework [4]. The DCP comprises two fundamental principles: an atom library – a collection of functions and sets ('atoms') with explicitly declared properties regarding curvature (convex, concave, affine) and a convexity ruleset – defining what combination of atoms, parameters and online variables guarantee convexity. Therefore, the mathematical model defined must fall within one of the following categories:

- A minimization problem: *convex objective + convex constraints*
- A maximization problem: *concave objective + convex constraints*
- A feasibility problem: *no objective + convex constraints*

While the constraints must conform to the one of the forms below:

$$\begin{aligned} \text{affine} &= \text{affine} \\ \text{convex} &< (=) \text{concave} \\ (\text{affine}, \dots, \text{affine}) &\text{ is convex set} \end{aligned}$$

To formulate a problem in the CVXGEN framework, the dimensions of each parameter and variable must be defined, but not necessarily the numerical values for parameters as the solver instantiates placeholders that be updated with numerical values after the code generation step. Optionally, the user can provide some predefined attributes for the variables and parameters such as nonpositive or PSD (positive semi-definite), but a complete description can be found in [3]. Due to the strict DCP framework, only the 3-DoF model of the vehicle can be implemented within CVXGEN. Moreover, essential constraints relevant for the landing of a launcher vehicle need to be reformulated, either through approximations or convexification techniques. Therefore, the mathematical model presented in [1] that relies on lossless convexification (LCvx) techniques serves as an appropriate basis for the convex implementations considered in this paper.

$$\min_{\sigma(\cdot)} \int_0^{t_f} \sigma(t) dt \quad (1)$$

$$\text{subject to } \ddot{\mathbf{r}}(t) = \mathbf{g} + \mathbf{u}(t) \quad (2)$$

$$\dot{z}(t) = -\alpha\sigma(t) \quad (3)$$

$$\|\mathbf{u}(t)\|_2 \leq \sigma(t) \quad (4)$$

$$\begin{aligned} \rho_1 e^{-z_0(t)} \left[1 - (z(t) - z_0(t)) + \frac{(z(t) - z_0(t))^2}{2} \right] &\leq \sigma(t) \\ &\leq \rho_2 e^{-z_0(t)} [1 - (z(t) - z_0(t))] \end{aligned} \quad (5)$$

$$z(0) = \ln(m_{\text{total}}), r(0) = r_0, \dot{r}(0) = \dot{r}_0, r(t_f) = \dot{r}(t_f) = 0 \quad (6)$$

$$m(t_{\text{final}}) \geq m_{\text{dry}} \quad (7)$$

$$\mathbf{T}_x(t) \geq \cos(\theta_{\text{max}}) \mathbf{T}(t) \quad (8)$$

$$\mathbf{T}_y(t_{\text{final}}) + \mathbf{T}_z(t_{\text{final}}) \rightarrow 0 \quad (9)$$

$$\|\mathbf{v}(t)\| \leq v_{\text{max}} \quad (10)$$

The optimization problem formulated above consist in a minimization of fuel through the LCVx-specific optimization variable $\sigma(t)$ objective function and constraints placed on dynamics, mass evolution, thrust bounds, maximum velocity, gimbal angle and vertical landing conditions. To be compatible with the CVXGEN, the problem formulated above needs to be discretized, for which the first-order-hold discretization method was utilized. The number of discretization points N was set to 20 due to the limitations imposed by CVXGEN in the overall dimension of the problem. Therefore, the constraints corresponding to the vertical landing phase were imposed on the last 4 discretization points.

To ensure a smooth trajectory generation, additional constraints have been placed on the acceleration through a first-order Euler discretization scheme, more conservative limits for the last part of the landing phase, and relaxed for the entire mission:

$$|v_{t+1,1} - v_{t,1}| \leq 0.3 \cdot \Delta t \quad (11)$$

$$|v_{t+1,2} - v_{t,2}| \leq 0.05 \cdot \Delta t, \text{ for } t = \text{numbersteps} - 4, \dots, \text{numberSteps} - 1 \quad (12)$$

$$|v_{t+1,3} - v_{t,3}| \leq 0.05 \cdot \Delta t \quad (13)$$

$$|v_{t+1,1} - v_{t,1}| \leq 0.5 \cdot \Delta t, \text{ for } t = 1, \dots, \text{numberSteps} - 1 \quad (14)$$

The numerical values of the constraints implemented in this formulation are:

$$112 [N] \leq \sigma_k \leq 771 [N] \quad (15)$$

$$0 [deg] \leq \theta_{\text{max}} \leq 5 [deg] \quad (16)$$

$$-1.5 [m/s] \leq v_x \leq 1.5 [m/s] \quad (17)$$

$$-0.8 [m/s] \leq v_y \leq 0.8 [m/s] \quad (18)$$

$$-0.8 [m/s] \leq v_z \leq 0.8 [m/s] \quad (19)$$

$$0 [m] \leq x \leq 1.5x_0 [m] \quad (20)$$

$$56.12 [kg] \leq m \leq 59.72 [kg] \quad (21)$$

A downloadable zip is created on the web interface, containing the following C scripts: `cvxgen_solver_sfunction_ADAMP.c`, `solver.c`, `matrix_support.c`, `ldl.c`, `util.c` and `testsolver.c`. Through legacy tool, they are used to create an S-function called `cvxgen_solver_sfunction_legacy` that is placed in a corresponding Simulink block as in Figure 1.

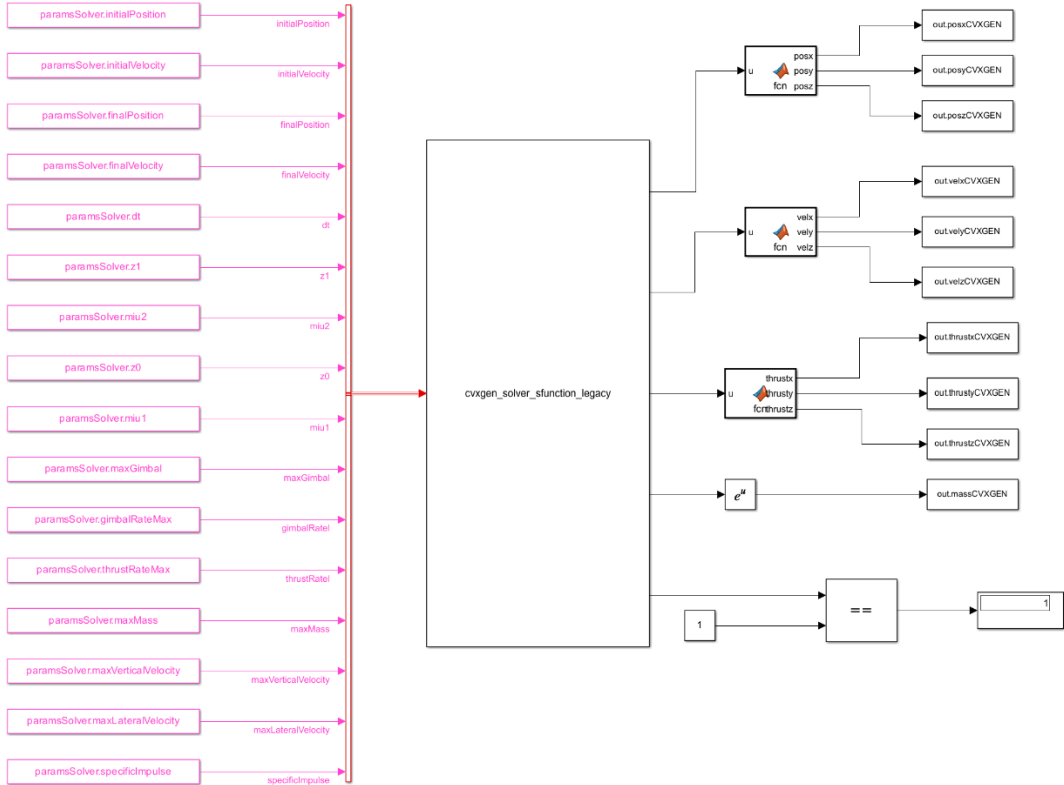


Figure 1. CVXGEN implementation in Simulink

3. OSQP IMPLEMENTATION

OSQP [5] (Operator Splitting Quadratic Program) is another library-free convex optimization tool that uses the Alternating Direction Method of Multipliers (ADMM) method to search for an optimal solution. The problem must be formulated in the Quadratic Program form:

$$\min \frac{1}{2} \mathbf{x}^T \mathbf{P} \mathbf{x} + \mathbf{q}^T \mathbf{x} \quad (22)$$

$$\text{subject to } \mathbf{l} \leq \mathbf{A} \mathbf{x} \leq \mathbf{u} \quad (23)$$

Where $\mathbf{x}$ is the optimization variable containing both states and controls (3 thrusts and the auxiliary variable σ) and $\mathbf{P}$ and $\mathbf{q}$ are a sparse matrix and vector representing weights defined by the user to model a suitable objective function. To draw a fair comparison, $\mathbf{P}$ is a zero-matrix populated with values corresponding to the σ terms in $\mathbf{x}$. The linear equality constraints

are expressed by considering $\mathbf{l}_i = \mathbf{u}_i$. From there, the same discretized core formulation as for CVXGEN can be formulated within the OSQP framework.

The same numerical values as for CVXGEN have been used for OSQP, except for the accelerations, where a different set of constraints have been placed to ensure a reasonably low number of iterations, an essential aspect for further real-time implementations:

$$|v_{t+1,1} - v_{t,1}| \leq 0.5 \cdot \Delta t \quad (24)$$

$$|v_{t+1,2} - v_{t,2}| \leq 0.4 \cdot \Delta t, \text{ for } t = 1, \dots, \text{numberSteps} - 1 \quad (25)$$

$$|v_{t+1,3} - v_{t,3}| \leq 0.4 \cdot \Delta t \quad (26)$$

$$|v_{t+1,1} - v_{t,1}| \leq 0.2 \cdot \Delta t, \text{ for } t = \text{numberSteps} - 7, \dots, \text{numberSteps} - 1 \quad (27)$$

After defining the problem in the Matlab interface, an additional command realizes the generation of the C code - `problem.codegen('code', 'parameters', 'matrices')` and 22 .c and .h files are generated representing the problem. A disadvantage of OSQP is that all parameters are numerically predefined, so to update any parameter a wrapper overwriting the existing one must be defined. This was the solution found to allow a changeable initial states and final states for the vehicle, as shown in Figure 2.

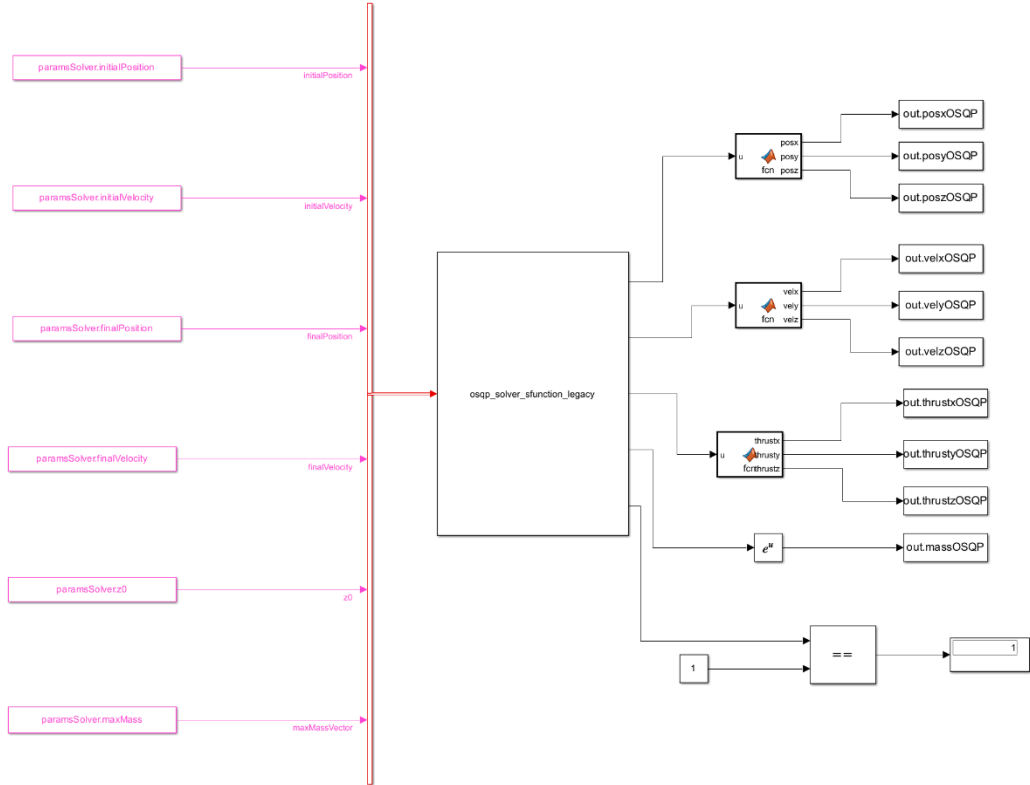


Figure 2. OSQP implementation in Simulink

4. ACADOS IMPLEMENTATION

ACADOS [6] is a free, open-source software package containing a multitude of solvers meant to be used by applications requiring fast embedded optimization algorithms. This package can be used along with many higher-level languages including MATLAB, Python and CasADI, using its interfaces for the design phase of the algorithms. ACADOS features efficient optimal control algorithms for embedded devices implemented in C and it is built upon the performant BLASFEO library for handling linear algebra.

For instance, the high-performance linear algebra library, BLASFEO, demonstrates superior performance compared to code-generated triple-loop linear algebra routines as well as established implementations like BLAS and LAPACK. ACADOS is designed to be a modular tool to be used for a variety of optimization problems and the user is free to choose which combination of modules best suits the problem at hand.

The main particularity that ACADOS has and the main differentiation factor from the other solvers is represented by the convexification procedure implemented in it, allowing the use of the exact Hessian based SQP (Sequential Quadratic Programming) algorithm and the Hessian approximation based SCQP (Successive Convex Quadratic Programming) method for real time optimization.

As previously mentioned, ACADOS follows a structured approach for model formulation within its framework. However, it also offers the user considerable flexibility in defining the system dynamics. Specifically, the user can directly formulate the problem dynamics in continuous time, so the optimization problem can be formulated as general as a continuous time nonlinear optimization problem:

$$\min_{\mathbf{x}, \mathbf{z}, \mathbf{u}} \int_0^T l(\mathbf{x}(t), \mathbf{z}(t), \mathbf{u}(t)) dt + M(\mathbf{x}(t)) \quad (28)$$

$$s. t. \mathbf{x}(0) = \mathbf{x}_0, \quad (29)$$

$$0 = f(\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{z}(t), \mathbf{u}(t)), \quad t \in [0, T], \quad (30)$$

$$0 \geq g(\mathbf{x}(t), \mathbf{z}(t), \mathbf{u}(t)), t \in [0, T]. \quad (31)$$

where $\mathbf{x}: \mathbb{R} \rightarrow \mathbb{R}^{n_x}$ represents the state vector, $\mathbf{u}: \mathbb{R} \rightarrow \mathbb{R}^{n_u}$ is the control input vector and $\mathbf{z}: \mathbb{R} \rightarrow \mathbb{R}^{n_z}$ denotes the algebraic variables vector. The cost function is expressed using the Boltza form, where the running cost is represented by $l: \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_u} \rightarrow \mathbb{R}$ (or Langrange term) and the terminal cost is $M: \mathbb{R}^{n_x} \rightarrow \mathbb{R}$ (or the Mayer term). The dynamics of the system are modeled with $f: \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_u} \rightarrow \mathbb{R}^{n_x} \times \mathbb{R}^{n_z}$ representing a set of implicit differential-algebraic equations. The initial value of the states is marked by $\mathbf{x}(0) = \mathbf{x}_0$ and the path constraints by $g: \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_u} \rightarrow \mathbb{R}^{n_g}$.

Since ACADOS automatically handles the discretization process—based on the scheme specified by the user—the continuous-time dynamics and constraints can be defined directly and efficiently in continuous time.

Because no convexification process has to be performed, the mathematical formulation of the problem can be adopted directly from [2], with all the mentioned constraints on dynamics,

mass evolution, maximum velocity, thrust bounds, gimbal angle and vertical landing conditions. An important mention to be made is that, the only two constraint that were not imposed in the ACADOS formulation but they were used in the Successive Convexification (SCvx) algorithm from [2] are the vertical thrust rate and the glideslope constraints. Also, the numerical constraint imposed in [2] on the velocity magnitude was divided on the vertical component and the lateral components of the velocity as:

$$|v_x| \leq v_{vert}^{max} \quad (32)$$

$$|v_y| \leq v_{lat}^{max} \quad (33)$$

$$|v_z| \leq v_{lat}^{max} \quad (34)$$

The numerical values of the path constraints implemented in this formulation are:

$$112 [N] \leq \sqrt{T_x^2 + T_y^2 + T_z^2} \leq 771 [N] \quad (35)$$

$$0 [deg] \leq \frac{T_x}{\sqrt{T_x^2 + T_y^2 + T_z^2}} \leq 5 [deg] \quad (36)$$

$$-1.5 [m/s] \leq v_x \leq 1.5 [m/s] \quad (37)$$

$$-0.8 [m/s] \leq v_y \leq 0.8 [m/s] \quad (38)$$

$$-0.8 [m/s] \leq v_z \leq 0.8 [m/s] \quad (39)$$

$$0 [m] \leq x \leq 1.5x_0 [m] \quad (40)$$

$$-50 [N] \leq T_y \leq 50 [N] \quad (41)$$

$$-50 [N] \leq T_z \leq 50 [N] \quad (42)$$

$$56.12 [kg] \leq m \leq 59.72 [kg] \quad (43)$$

For this scenario, the vertical landing constraints for ACADOS are defined to be activated for the last 4 discretization nodes of the trajectory. In those nodes, the constraints regarding the allowed lateral thrust and velocity change to:

$$-v_{vert}^{max}(i) \leq v_x \leq v_{vert}^{max}(i) \quad (44)$$

$$-v_{lat}^{max}(i) \leq v_y \leq v_{lat}^{max}(i) \quad (45)$$

$$-v_{lat}^{max}(i) \leq v_z \leq v_{lat}^{max}(i) \quad (46)$$

$$-0.1 \leq T_y \leq 0.1 \quad (47)$$

$$-0.1 \leq T_z \leq 0.1 \quad (48)$$

where $v_{vert}^{max}(i)$ and $v_{lat}^{max}(i)$ are values for the allowed vertical and lateral velocity, with $i \in [N - 3, N]$, N being the number of temporal nodes. They are set by following a process of trial and error in order to achieve a smooth, vertical landing.

After defining the problem in the Matlab interface of ACADOS, the framework allows the user to set which signals to be inputs and outputs for the S-Function [7]. After the problem is solved in the Matlab interface, the script capable of generating the S-Function is created and ready to be run. After running it, the framework generates .mex file representing the S-Function ready to be implemented in Simulink:

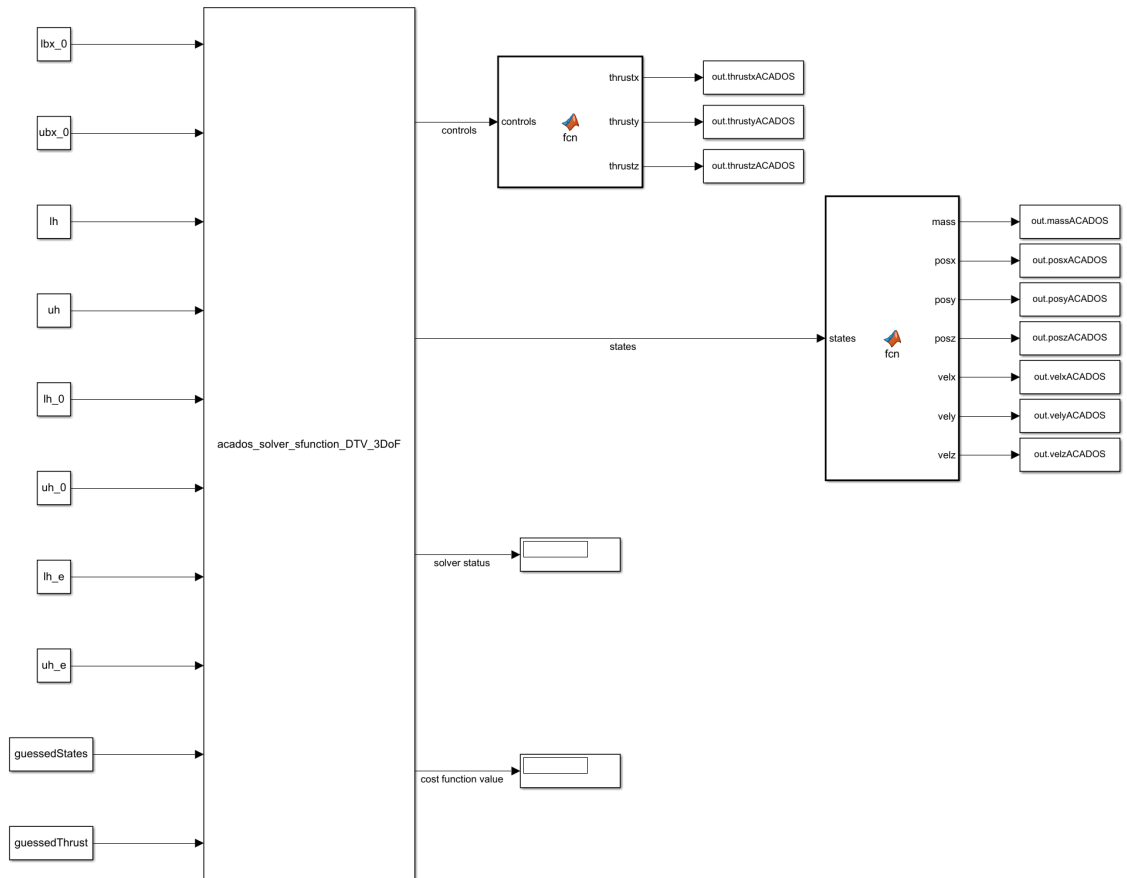


Figure 3. ACADOS implementation in Simulink

Here, the name of the variables lbx_0 and ubx_0 represent the vectors defining the initial states of the system; lh and uh are the vectors containing the path constraints; lh_0 and uh_0 are the vectors defining the initial feasible domain of control-related parameters; lh_e and uh_e are the vectors containing the final state of the system; $guessedStates$ and $guessedThrust$ represent the initial guesses on the states of the system and its control actions. The initial guess is computed as described in [2].

The outputs of the S-Function are the controls (containing the thrust force on each axis), the states across the trajectory (mass, 3 positions and 3 velocities), the solver status and the cost function value.

5. SIMULATION ANALYSIS

An important observation to be made is that the solvers do not accept any parameter as an online parameter. We define a parameter to be an online parameter if its numerical value can be modified after the generation of the S-Function containing the C code. In the next table is shown every acceptable online parameter of each S-function:

Table 1. Solver capabilities comparison

Initial State	ACADOS	CVXGEN	OSQP
m_0	✓	✓	✓
r_x	✓	✓	✓
r_y	✓	✓	✓
r_z	✓	✓	✓
v_x	✓	✓	✓
v_y	✓	✓	✓
v_z	✓	✓	✓
Final State	ACADOS	CVXGEN	OSQP
r_x	✓	✓	✓
r_y	✓	✓	✓
r_z	✓	✓	✓
v_x	✓	✓	✓
v_y	✓	✓	✓
v_z	✓	✓	✓
Path Constraints	ACADOS	CVXGEN	OSQP
$v_{x\max}$	✓	-	✗
$v_{y\max}$	✓	-	✗
$v_{z\max}$	✓	-	✗
$T_{\max}$	✓	✓	✗
$T_{\min}$	✓	✓	✗
$T_{\max lateral}$	✓	-	-
$\delta_{\max}$	✓	✓	✗
$\dot{T}_{\max}$	-	✓	✗
$\dot{\delta}_{\max}$	-	✓	✗
Other parameters	ACADOS	CVXGEN	OSQP
dt	✗	✓	✗
I_{sp}	✗	✓	✗

Before presenting the results of the simulations, the limit conditions are as follows: the vehicle starts from a height of 10 [m] above the ground level and it is considered to have 0 velocity on each axis:

$$initialPosition = [10; 0; 0][m] \quad (49)$$

$$initialVelocity = [0; 0; 0][m/s] \quad (50)$$

$$initialMass = 59.72[kg] \quad (51)$$

The final state of the system is desired to be:

$$finalPosition = [CoG; 5; 4][m] \quad (52)$$

$$initialVelocity = [0; 0; 0][m/s] \quad (53)$$

where $CoG = 0.7$ [m] represents the position for the center of gravity of the vehicle above the ground level.

A comparison of the trajectories generated with each solver is presented in the figures below. Firstly, the 3D trajectory is represented in Figure 4, highlighting the particularities of each solver. For example, CVXGEN is first increasing the altitude then decreasing it, favoring a bang-bang control strategy.

This corresponds to what can be observed in Figure 6 where the velocity increases toward 0.5 [m/s] before decreasing and Figure 7 where the initial thrust for CVXGEN solver is higher than for the other solver.

Observing all figures, the behaviors of ACADOS and OSQP are similar, with the only notable differences in the thrust plot (Figure 7), where ACADOS keeps a more linear evolution for lateral thrust compared to OSQP. As DTV is a fuel-efficient vehicle, no significant differences can be observed between the implementations considered (Figure 8), only slightly more fuel consumed for ACADOS.

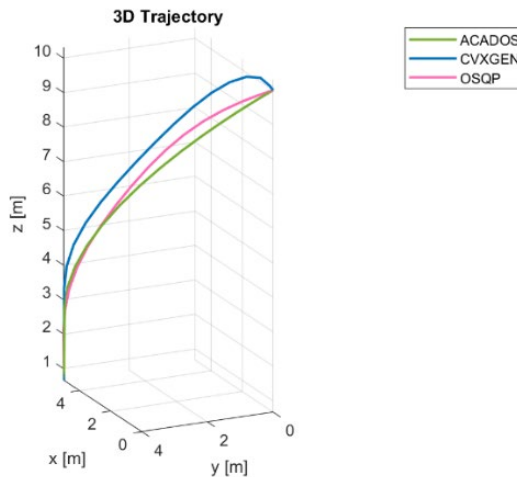


Figure 4. 3D trajectories

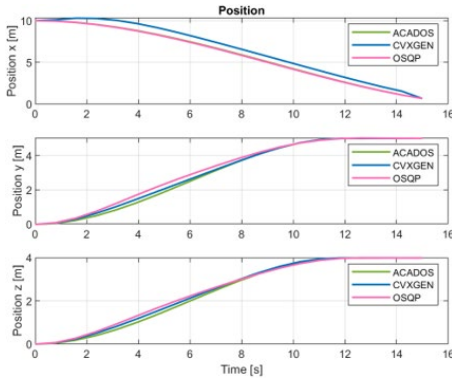


Figure 5. Positions

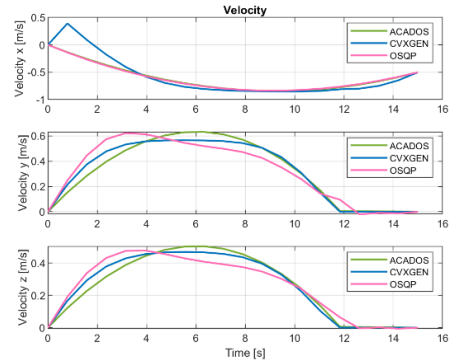


Figure 6. Velocities

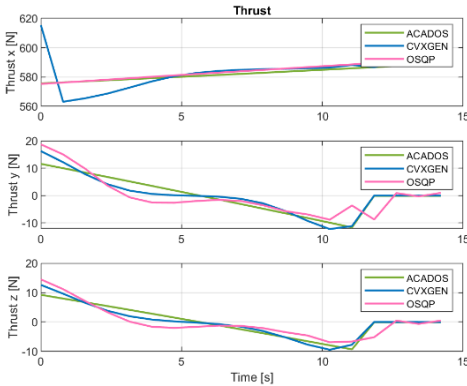


Figure 7. Thrusts

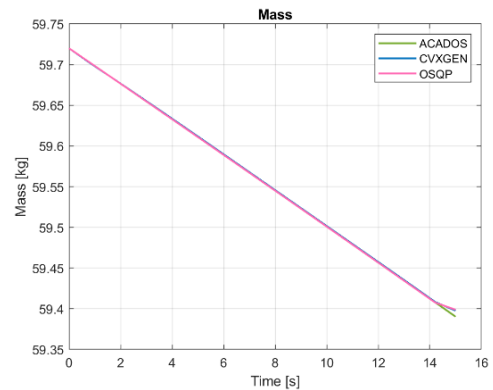


Figure 8. Mass evolution

6. CONCLUSIONS

This paper presented a comparative analysis of three optimization frameworks (CVXGEN, OSQP and ACADOS) for the implementation of a fixed final time PDG problem intended for autonomous vertical landing applications. The study focuses on their suitability for real-time onboard implementation by exploiting their capability to generate standalone C code that can be embedded into onboard computers.

The three implementations were developed using mathematical formulations adapted to the specific capabilities and limitations of each solver. For the convex optimization solvers, CVXGEN and OSQP, the problem formulation relied on LCvx techniques and the mathematical model had to be discretized by the user.

The ACADOS implementation has the advantage of directly formulating the problem in continuous time using nonlinear dynamics and constraints, eliminating the need for prior convexification and discretization of the initial problem.

Simulation results showed that all three solvers were capable of generating feasible landing trajectories satisfying the imposed mission constraints.

Another important aspect highlighted during the implementation process is the ease with which S-Functions can be generated and integrated using each framework. In the case of CVXGEN and OSQP, the generation of the S-Function architecture involves a more complex workflow, as neither framework provides an automatic Simulink wrapper generation

capability. Consequently, additional effort is required from the user to manually interface the generated C code with the Simulink environment. In contrast, ACADOS provides built-in support for the automatic generation of S-Functions through scripts included within the framework itself. Furthermore, the S-Function architecture, particularly regarding its inputs and outputs, can be easily configured and modified according to the requirements of the application.

Overall, the obtained results indicate that all three solvers represent viable candidates for embedded powered descent guidance applications. However, a fair comparison between their execution times on a space-relevant processor cannot be performed at this level of analysis. For this reason, the most important future direction of this research is to evaluate the performance of each implementation through Software-in-the-Loop (SIL) and Hardware-in-the-Loop (HIL) testing campaigns in order to determine their computational efficiency and real-time suitability for onboard deployment.

REFERENCES

- [1] A. Botez and S. D. Dragan, Solver comparison for LCVx-based powered descent guidance problems, *INCAS BULLETIN*, (print) ISSN 2066–8201, (online) ISSN 2247–4528, ISSN–L 2066–8201/2025, vol 17, issue 3, pp. 3–11, <https://doi.org/10.13111/2066-8201.2025.17.3.1>, 2025.
- [2] S. D. Dragan and A. Botez, Successive Convexification for Powered Descent Guidance: A Comparative Study of Fuel-Optimal Cost Functions, *INCAS BULLETIN*, (print) ISSN 2066–8201, (online) ISSN 2247–4528, ISSN–L 2066–8201/2025, vol 17, issue 3, pp. 13–25, <https://doi.org/10.13111/2066-8201.2025.17.3.2>, 2025.
- [3] J. Mattingley and S. Boyd, CVXGEN: a code generator for embedded convex optimization, *Optimization and Engineering*, **13**(1):1–27, 2012.
- [4] M. Grant, S. Boyd and Y. Ye, *Disciplined Convex Programming*, DOI:10.1007/0-387-30528-9_7, In book: Global Optimization (pp.155–210), 2006.
- [5] B. Stellato, G. Banjac, P. Goulart, A. Bemporad and S. Boyd, OSQP: An operator splitting solver for quadratic programs, *Mathematical Programming Computation*, 12:637–672, <https://doi.org/10.1007/s12532-020-00179-2>, 2020.
- [6] J. E. Mattingley, *Code generation for embedded convex optimization*, Thesis (Ph.D.)--Stanford University, Submitted to the Department of Electrical Engineering, 2011.
- [7] * * * ACADOS documentation, [Online], Available: https://docs.acados.org/matlab_octave_interface/index.html.