Università degli Studi di Padova

Dipartimento di Ingegneria Industriale

Corso di Laurea Magistrale in Ingegneria Aerospaziale

Preliminary design of a cold gas aerospike engine experiment

Internal Supervisor

Prof. **Daniele Pavarin**

Università degli Studi di Padova

External Supervisor

Dr.-Ing. **Christian Bach**

Technische Universität Dresden

Candidate

Giorgio Tesser

Matricola : 1157041

A.A. 2018-2019

Abstract

This paper shows a preliminary design of a cold gas engine, equipped with an aerospike nozzle, for space applications. Aerospike nozzles, toroidal or linear ones, have altitude compensation properties that recall the possibility of creating a single stage to orbit (SSTO) aircraft. Aerospike nozzles, in fact, have higher performance than a common bell nozzle when operating with lower expansion ratios than the optimal one.

Although they are best known for their altitude compensating capabilities, the advantages of using an aerospike nozzle for purely space applications are numerous, including the possibility of designing the nozzle for a higher expansion ratio, achieving better vacuum performances without significantly increasing mass and size. This work is part of the MACARONIS project, which aims to develop, manufacture and qualify a ceramic aerospike nozzle for future space missions. An analysis on state of the art cold gas thruster is developed and then the critical components of the system are analysed individually.

In preparation for MACARONIS project, a first definition of objectives, requirements and constraints is presented. Then, two designs are developed, one consisting of a single fluid line and one featuring redundant components. A genetic algorithm was developed to optimize the two configurations and to provide a decision-making, flexible tool for the team. Finally, the results obtained are presented and some possible ideas for the development of this work are listed.

Keyword: aerospike, genetic algorithm, cold gas thruster

Abstract in italiano

In questo elaborato viene mostrato un design preliminare di un motore cold gas, dotato di ugello aerospike, per applicazioni spaziali.

Un ugello aerospike, toroidale o lineare, presenta delle proprietà di compensazione dell'altitudine che richiamano alla possibilità di realizzazione di un velivolo single stage to orbit (SSTO). Gli ugelli aerospike infatti, per rapporti di espansione inferiori rispetto a quello ottimale, hanno performance più elevate rispetto ad un comune ugello a campana. Benchè siano maggiormente noti per la loro capacità di compensazione dell'altitudine, i vantaggi connessi all'impiego un ugello aerospike per applicazioni puramente spaziali sono numerosi, tra cui la possibilità di progettare l'ugello per un rapporto di espansione maggiore, ottenendo prestazioni migliori senza aumentare considerevolmente peso e ingombro e un migliore sfruttamento dello spazio disponibile. Questo lavoro si sviluppa nell'ambito del progetto MACARONIS, che ha l'obiettivo di sviluppare, produrre e qualificare un ugello aerospike in ceramica per future missioni spaziali.

Un'analisi dello stato dell'arte riguardante i propulsori a gas freddo viene redatta. Questa tecnologia, data la sua semplicità, viene considerata la più matura e adatta ai piccoli spacecraft. Successivamente, i componenti critici del sistema vengono individuati e analizzati singolarmente.

In preparazione del progetto MACARONIS, viene presentata una prima definizione di obiettivi, requisiti e vincoli riguardante il sistema propulsivo.

Successivamente, sono sviluppati due progetti, uno costituito da una singola linea fluida e uno con componenti ridondanti. Mentre il primo progetto consta dei componenti minimi per la realizzazione di un propulsore a gas freddo, il secondo progetto, prendendo spunto dai satelliti GRACE, segue il concetto di *redundancy* e *reliability*, mantenendo la propria funzionalità anche in caso di malfunzionamenti.

Viene poi stilata una lista di design drivers, basati sull'analisi dello stato dell'arte e sull'esperienza pregressa maturata alla TU Dresden nell'ambito del ceramic 3D printing. Per ottimizzare il sistema, un algoritmo genetico è stato sviluppato. La caratteristica peculiare di questi algoritmi, non comprendendo processi di derivazione, sta nel poter lavorare in spazi non convessi. Gli input possono essere sia valori continui, come una temperatura all'interno di un range definito, sia non continui, come un numero identificativo di una caratteristica. Nello spazio così definito, l'algoritmo compara numerose stringhe di design per trovare la configurazione ottimale. La versatilità di un algoritmo genetico è stata verificata in questo elaborato producendo anche una versione alternativa dell'algoritmo che consente all'utente di ottenere risultati specificando un target obiettivo, in questo caso di spinta. É stato scelto un algoritmo genetico poichè si voleva fornire un tool decisionale facilmente modificabile e adattabile alle esigenze future del team. I risultati ottenuti dalle due versioni di algoritmo vengono presentati, commentati e successivamente vengono stilati dei possibili spunti per lo sviluppo di questo lavoro.

Index

List of Figures

Acronyms

ADCS Attitude Determination and Control System. 42

ASME American Society of Mechanical Engineers. 32, 33

ATH attitude control thruster. 29

CGMT Cold Gas Micro Thruster. 19

CGPS Cold Gas Propulsion System. 27, 30

CNAPS Canadian Nanosatellite Advanced Propulsion System. 22

COTS Components of the shelf. 50, 71

CPOD CubeSat Proximity Operations Demonstration. 26

DLC Double line configuration. 48, 57, 72

FF Fitness Function. 52, 54, 56, 63–65, 69–71, 73, 74, 76

GA Genetic algorithm. 51–56, 60, 63, 65, 69–71, 73, 75–77

GRACE Gravity Recovery and Climate Experiment. 28, 37, 45

HPLV high pressure latching valve. 28

HPT high pressure transducer. 28

Isp Specific impulse. 14, 34, 62

LPF low pressure filter. 29

LPT low pressure transducer. 28

OLC One line configuration. 45, 48, 57, 70, 72, 75

OTH orbit control thruster. 29

PR pressure regulator. 28

RV relief valve. 28

SSTL Surrey Satellite Technology Ltd. 19, 36

SSTO Single Stage to Orbit. 1, 3, 6, 13, 14

Chapter 1

Introduction

In this chapter, a brief introduction on space propulsion concepts and Single Stage to Orbit (SSTO) is given. The two main typologies of aerospike nozzles are presented and then the objective of this work is stated.

1.1 Space Propulsion

The common characteristic of all space vehicle launched from Earth is that they are actuated by a rocket motor, that uses the chemical energy of solid or liquid propellants to steadily and rapidly produce a large quantity of hot high-pressure gas, which is then expanded and accelerated through a nozzle. This large mass of combustion products flowing out of the nozzle at supersonic speed possesses a lot of momentum and, leaving the vehicle behind, causes the vehicle itself to acquire a momentum in the opposite direction. This is represented as the action of the force we know as thrust.

Each space launch vehicle has a specific flight objective, such as a specific orbit or a moon landing. It uses between two and six stages [2], each with its own propulsion system and each fired sequentially after the lower stage is expended. Multistage rocket vehicles permit higher vehicle velocities and more payload for space vehicles.

As represented in figure 1.1, rockets have multiple stages in order to reach maximum efficiency.

Once the propellant of a particular stage is finished, the mass of that stage is no longer useful and therefore, by releasing this mass, it is possible to accelerate the other stages with its useful payload to a higher terminal velocity than would be attained if multiple staging were not used.

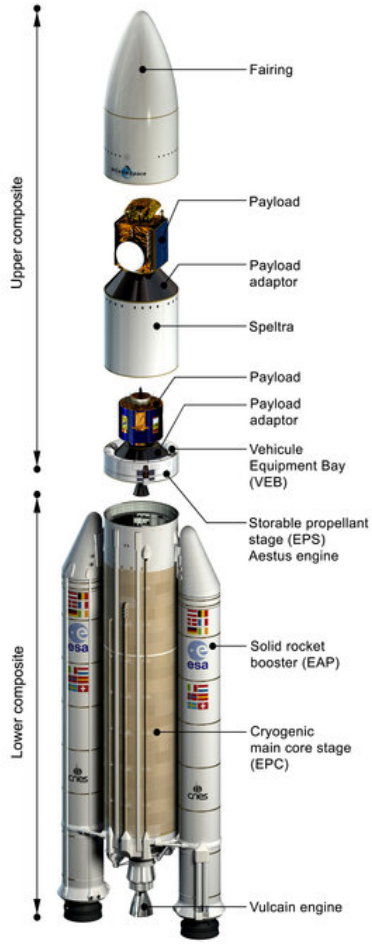


Figure 1.1: Schematic rappresentation of the Ariane 5G [3]

In order to avoid the usage of propellant to accelerate a useless mass, the empty mass of an expended stage is separated from the vehicle. Increasing the number of stages leads to a decrease of the takeoff mass, until the number of the physical mechanisms get so numerous and complex, that the vehicle becomes heavy and unreliable. As already said, usually the number of stages is between 2 and 6.

Moreover, the payload mass is essentially proportional to the takeoff mass. If a payload of 50 kg requires a 6000 kg multistage rocket, a 500 kg payload would require a 60000 kg rocket unit with an identical number of stages, and a similar configuration with the same payload fraction.

The total ideal velocity of a multistage vehicle (in tandem or series-staging) consists of the sum of the individual stage velocity increments.

For n stages, the final velocity increment Δv_f is:

$$\Delta v_f = \sum_{i=1}^{n} \Delta v_i = \Delta v_1 + ... + \Delta v_n \tag{1.1}$$

For a simplified case of a vacuum flight in a gravity-free field the total Δv_f required is presented in Eqn.1.2 [2], where $c_1, ..., c_n$ are the effective exhaust velocities of the respective stage.

$$\Delta v_f = c_1 \ln\left(\frac{m_{01}}{m_{f1}}\right) + ... + c_n \ln\left(\frac{m_{0n}}{m_{fn}}\right) \tag{1.2}$$

An attractive concept is the SSTO vehicle, because it avoids the cost and complexity of staging and is expected to have improved reliability, having fewer components and a simpler structure.

The reduction of Earth-to-orbit launch costs in conjunction with an increase in launcher reliability and operational efficiency are the key demands on future space transportation systems, like SSTO [4].

At the moment, no Earth-launched SSTO launch vehicle has been built and tested. An explanation of that resides in the Tsiolkovsky rocket equation.

Neglecting drag and gravitational attraction, the Tsiolkovsky equation can be written as:

$$\Delta v = I_{sp} g_0 \ln\left(\frac{m_0}{m_f}\right) \tag{1.3}$$

We define m_0 as the initial mass of the rocket. It consists of the payload mass m_{PL}, the empty mass m_E and the propellent mass m_p.

$$m_0 = m_{PL} + m_E + m_p \tag{1.4}$$

The empty mass is the also called structural mass, because it consists of the mass of the fuel tanks, the engines, the structures, etc.

It is convenient to define two coefficients, the payload ratio

$$\lambda = \frac{m_{PL}}{m_E + m_p} = \frac{m_{PL}}{m_0 - m_{PL}} \tag{1.5}$$

and the structural ratio

$$\varepsilon = \frac{m_E}{m_E + m_p} = \frac{m_E}{m_0 - m_{PL}} \tag{1.6}$$

The mass ratio n, assuming all the propellant is consumed, can be written as

$$n = \frac{m_{PL} + m_E + m_p}{m_{PL} + m_E} \tag{1.7}$$

λ, ε and n are correlated by the equation

$$m_E = \frac{\varepsilon}{1-\varepsilon} m_p \tag{1.8}$$

whereas Eqn 1.5 gives

$$m_{PL} = \lambda(m_E + m_P) = \lambda\left(\frac{\varepsilon}{1-\varepsilon} m_p + m_p\right) = \frac{\lambda}{1-\varepsilon} m_p \tag{1.9}$$

Substituting Eqns 1.8 and 1.9 into Eqn 1.7 leads to

$$n = \frac{1+\lambda}{\epsilon+\lambda} \tag{1.10}$$

Setting the Δv equal to the burnout speed v_{bo}, when the propellants have been used up, and using the relation in Eqn 1.10, yields

$$v_{bo} = I_{sp} g_0 \ln n = I_{sp} g_0 \ln \frac{1+\lambda}{\varepsilon+\lambda} \tag{1.11}$$

This equation is plotted in figure 1.2.

The objective is to increase the payload mass while while keeping the structural weight to a minimum. Current materials technology places a lower limit on ε of about 0.1. For $\varepsilon = 0.05$, Eqn 1.10 yields

$$v_{bo} = 1.94 I_{sp} g_0 = 0.019 I_{sp} \tag{1.12}$$

A classical rocket is characterized by a specific impulse of about 300s, providing in this case $\Delta v = 5.7$ km/s, a value inferior than the circular orbital velocity at earth's surface, that is 7.905 km/s. Therefore, a standalone booster with these characteristics could not orbit the payload.

In order to reach orbit with one stage, a minimum specific impulse of 416s would be required, and only most advanced liquid hydrogen/liquid oxygen engines have this kind of performance but they are extremely complex and expensive.

Moreover, performance data for rocket engines are practically always lower than the theoretical ones. This difference is related to imperfections in the mixing process, combustion and expansion of the propellant. Different loss sources are shown in figure 1.3.

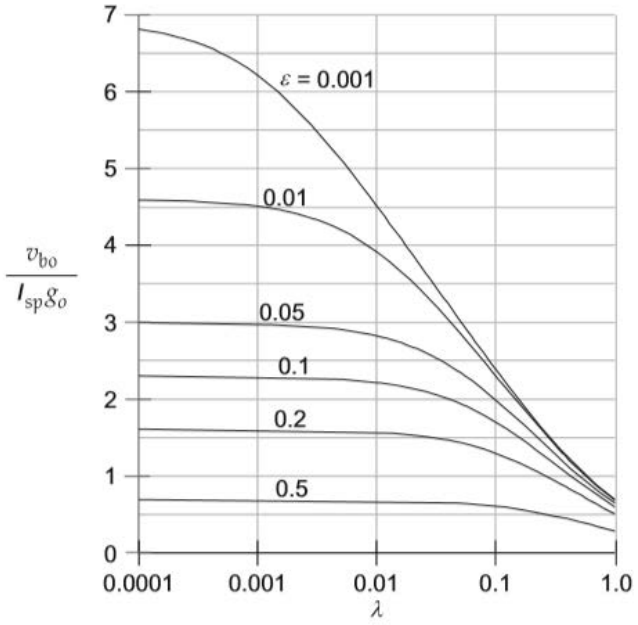


Figure 1.2: Dimensionaless bornout speed vs payload ratio [5]

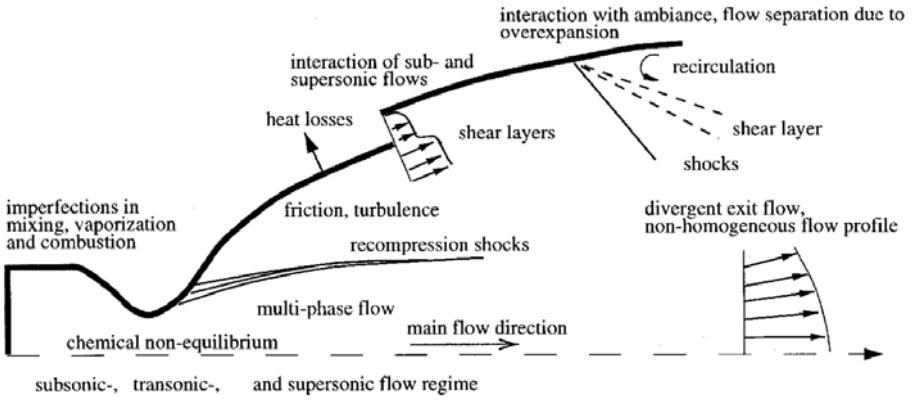


Figure 1.3: Flow phenomena and loss sources in rocket nozzle [4]

Table 1.1 summarize the performance losses in the thrust chambers and nozzles of typical high-performance rocket engines: The SSME- and Vulcain 1 engine [4]. It is the Space Shuttle main engine, Rocketdyne hydrogen – oxygen engine and hydrogen – oxygen core engine of European Ariane-5 launcher.

As it is possible to see from table 1.1, the nonadaptation of the exhaust flow, due to the different ambient pressures at different altitudes, induces a significant negative thrust contribution. Moreover, ambient pressures that are higher than nozzle wall exit pressures increase the risk of flow separation inside the nozzle. The flow separation may cause the generation of side loads.

Losses	Vulcain 1 [%]	SSME [%]
Chemical nonequilibrium	0.2	0.1
Friction	1.1	0.6
Divergence, nonuniformity of exit flow	1.2	1.0
Imperfection in mixing and combustion	1.0	0.5
Nonadapted nozzle flow	$0 - 15$	$0 - 15$

Table 1.1: Losses in high-performance rocket nozzles

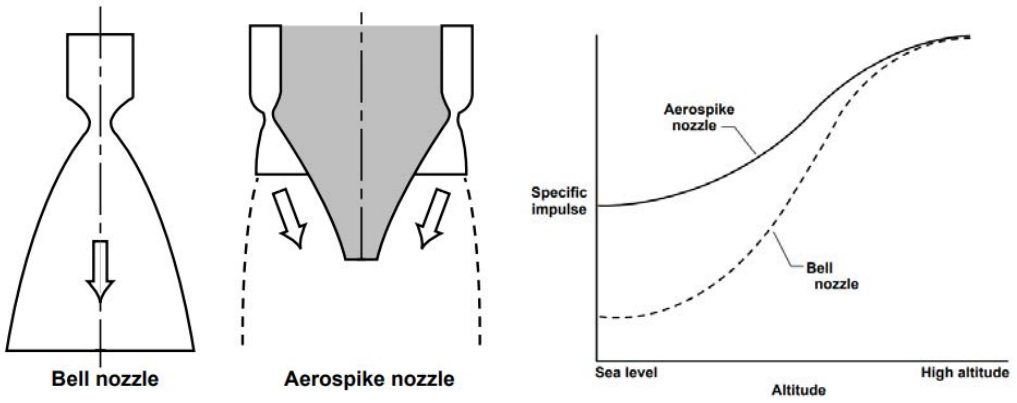
Increasing the efficiency of the thrust chamber is crucial in order to make a SSTO vehicle feasible.

1.2 The aerospike nozzle

Conventional bell-type nozzles, which are used in practically every rocket, induce significant performance losses when used in off-design operation. This means that both overexpanded, during low altitude operations with ambient pressures higher than the nozzle exit pressure, and underexpanded, during high altitude operations with ambient pressures lower than the nozzle exit pressure, situations induce performance losses. These losses rise up to 15%[4].

Moreover, it is ideally possible to design the nozzle with a much higher area ratio in order to achieve better vacuum performance, but the flow would then separate inside the nozzle in overexpanded conditions, with the generation of side-loads that would modify the rocket attitude.

Unlike conventional bell-shaped nozzles, which show optimal performances at one particular pressure (thus altitude), aerospike nozzles are well known for their altitude compensating capabilities. Aerospike nozzles can be described as inverted bell nozzles where the flow expands on the outside of the nozzle instead of being completely constrained by the nozzle walls. As shown in figure 1.4a, in the aerospike nozzle, the flow is not bound by an outer contour, but an inner contour.



(a) (b)

Figure 1.4: Comparison between a conventional bell nozzle and an aerospike nozzle[6]

Aerospike nozzles therefore adapt to the outer pressure, operating at an optimum nozzle expansion ratio for a range of altitudes [7]. This is particularly critical for Single-Stage-to-Orbit (SSTO) vehicles, which operate both in the atmosphere and in vacuum, but it is important to notice that this improvement over conventional bell-shaped nozzles only occurs at altitudes lower than the design pressure ratio. At altitudes higher than the design altitude (or pressure ratio), plug nozzles essentially operate similarly to bell nozzles [8]. This is shown in figure 1.4b.

Figure 1.5 shows different behaviour of the aerospike nozzle at high and low pressure levels. As it is possible to see, the diameter of the exhaust flow plume increases with altitude.

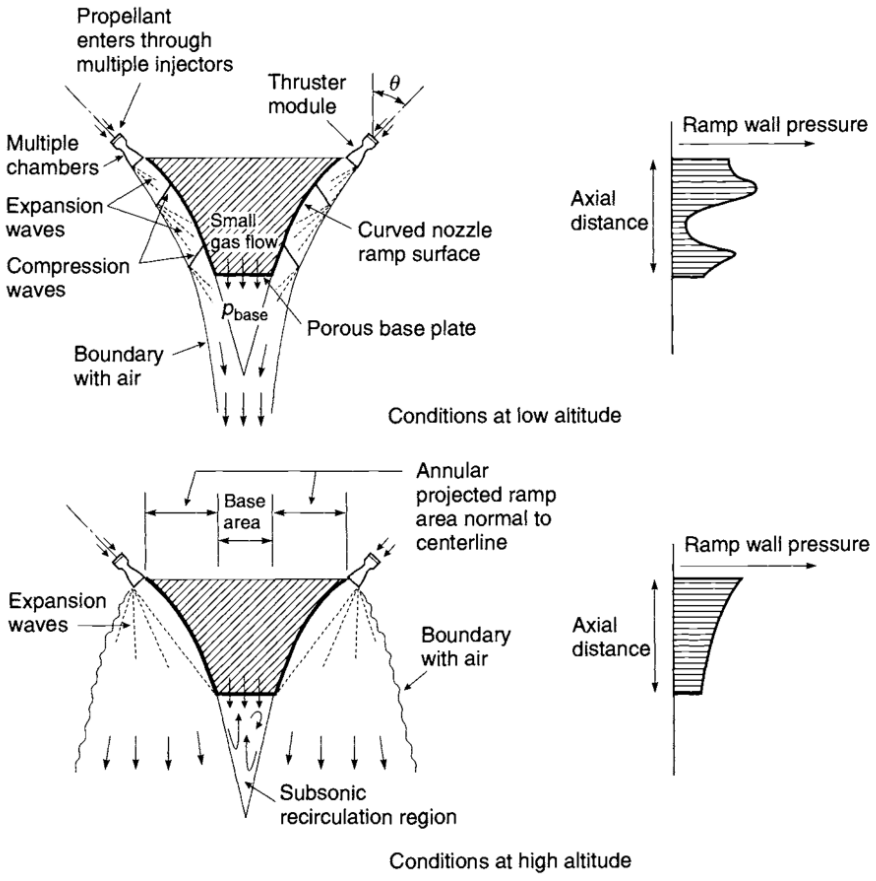


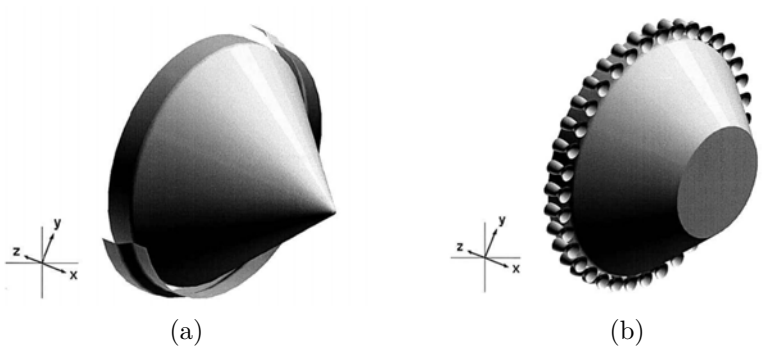
Figure 1.5: Pressure profile and flow pattern along the ramp of an aerospike nozzle [2]

It is possible to divide the aerospike nozzles into two different categories: the toroidal aerospike and the linear one.

The toroidal aerospike

Circular plug nozzles, called also toroidal nozzles, have been mainly designed in two configurations(Fig.1.6). The first one, illustrated in figure 1.6a, is characterised by a toroidal chamber and throat while the second one (Fig.1.6b) uses a cluster of circular (or quasirectangular) nozzle modules.

There are multiple benefits using the cluster configuration over the toroidal one. Firstly, the cluster design does not require to control a constant gap throat during manufacturing and thermal expansion during operations. Variation of the throat area can cause a variation in performance, side-loads and thrust vectoring deviations. Moreover, the cooling of a toroidal throat and the control of the combustion instabilities are challenging problems. The further losses introduced by a cluster design are induced by the gaps between the nozzle modules and the flowfield interactions downstream of the module exits, but they can be minimized.



(a) (b)

Figure 1.6: Classical toroidal plug (a) and toroidal plug with clustered chambers (b)

Figure 1.7 shows the main characteristics of the flow along a full length spike nozzle (left side) and a truncated spike nozzle (right side). While the central images represent the behaviour of the systems under design pressure ratio, the top and the bottom ones show two off-design configurations, respectively overexpanded and underexpanded ones.

Firstly, the full length spike behaviour is characterized. When the pressure ratio is lower than the design one, the full length spike assures that the flow expands near the central plug body without separation.

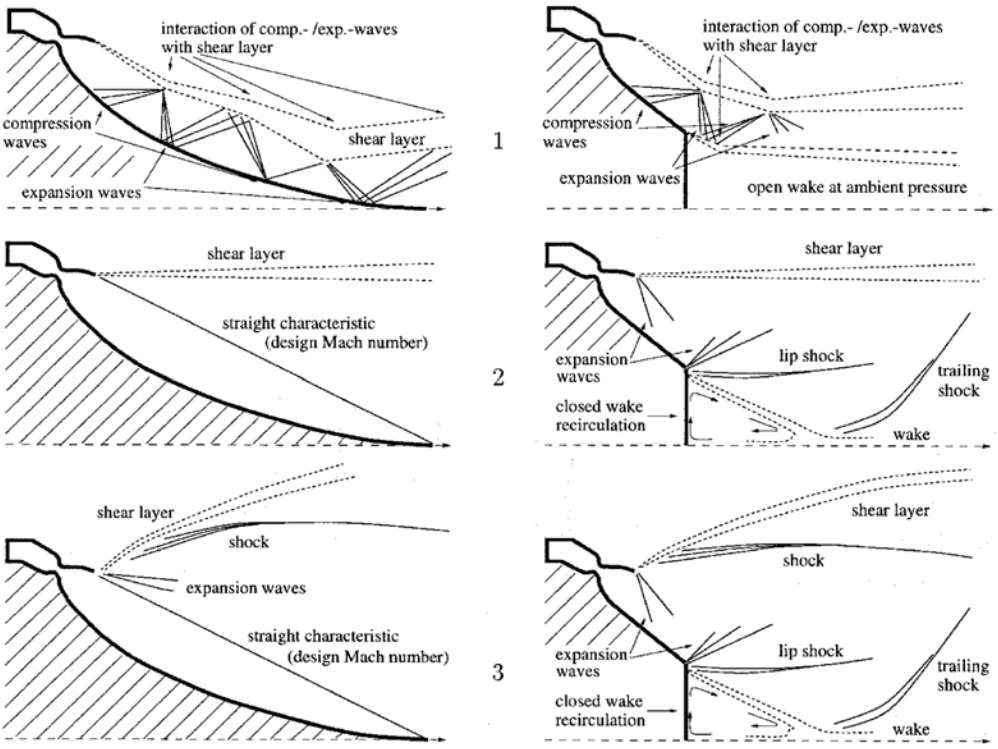


Figure 1.7: Flow phenomena, at off-design pressure ratios (top, bottom) and design pressure ratio (center), of a plug nozzle with full length (left) and truncated central body (right) [4]

The flow adapts to the ambient pressure through a system of recompression shocks and expansion waves, which interact with the shear level forming the characteristic barrel-like shape of the exhaust. Expansion and recompression phenomena are connected to the up- and down-variations of spike wall pressure profiles, causing several inflections of the shear-layer, which is enlarged by the turbolent diffusion along the spike.

The design pressure ratio is shown in Fig.1.7 (left column, center). The characteristic with the design Mach number should be a straight line, starting from the tip of the central plug body, and the shear layer is parallel to the centreline. The exit flow profile is influenced by the nonhomogeneous flow in the throat region. It has to be said that usually this aspect is not considered while designing the spike contour, especially in early-phases of the design. For pressure ratios higher than the design pressure ration, the wall pressure distribution remains constant and the plug nozzle behave like a conventional nozzle, losing its altitude adaptation capabilities (this aspect is also shown in figure 1.8).

In some cases the spike can be extremely long, causing an increase in the overall mass and, more important, poor mechanical properties of the spike itself (the longer, the thinner). The truncation of the central plug body is an advantage because it can reduce the mass of the system, increase the machinability and the reliability of the nozzle. Moreover, if not truncated, it is hard to properly cool the central end part of the central body. On the other hand, the flow changes its behaviour because of the missing of a part of the plug. Firstly, where before there was the full spike, an open wake flow appears at lowe pressure ratios, characterized by a pressure near the ambient one (Fig.1.7, right column, top).

When the pressure ratio gets close to the design pressure ratio, a close wake forms, characterized by a constant base pressure that is independent from the ambient pressure (Fig.1.7, right column, center).

The pressure in the wake is below ambient pressure and the base area induces a negative thrust, proportionally to the percentage of truncation and the total size of the base area. Decreasing the length of the plug body increases the negative thrust because the total base area increases.

Decreasing the ambient pressure, the pressure within the closed wake remains constant but becomes higher than the ambient pressure, resulting in a positive thrust contribution of the total base area.

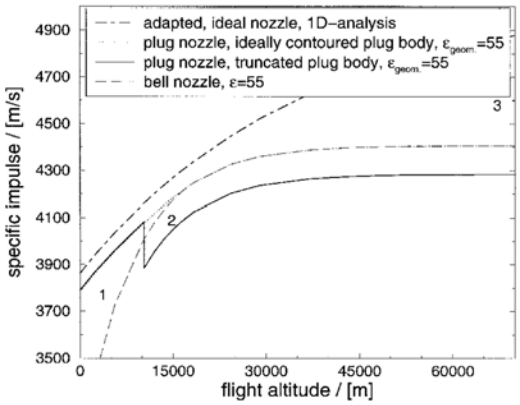


Figure 1.8: Performance of numerically simulated plug nozzle with truncated central body [4]

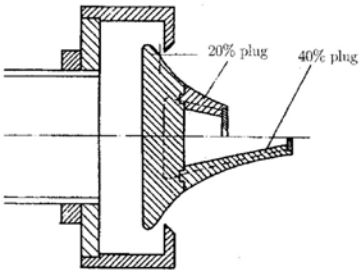
In figure 1.8, performance of a numerically simulated plug nozzle are compared to the same plug nozzle with full-length central body and a conventional bell nozzle. It is possible to appreciate the altitude compensating capabilities of the aerospike nozzle when the pressure ratio is lower than the design pressure ratio and the performance decrease induced by the truncation.

Linear plug nozzles

Flow field characteristics and performance of a linear plug nozzles are similar to those of circular plug nozzles and their behaviour, as a function of the ambient pressure, is similar to the one depicted in Fig.1.8.

However, the surrounding flow disturbs the exhaust flow at both side of the linear plug. These disturbances results in an undesirable expansion of the flow normal to the main direction (i.e. the axial direction) and, consequently, in an effective performance loss. For truncated nozzles, the wake flow behaviour could be strongly influenced by the penetration of ambient pressure. The implementation of end plates at both side of the spike can avoid the penetration of ambient pressure. A possible implementation is shown in figure 1.9a, representing a prototype realized by Arca. Figure 1.9b shows a baseline model of an aerospike nozzle, developed within the framework of the Esa FESTIP Technology Program, charactirezed by a linear throat.

(a) Arca linear aerospike prototype [9]



(b) Design of linear subscale plug nozzle of the ESA FESTIP Technology Program (plug size: 192 mm high, 214 mm wide)[4]

Figure 1.9

In summary, the performance of circular and linear plug nozzle is adversely affected by external airflow. The altitude compensation capability is indisputable and the design pressure should be chosen as high as possible, because for pressure ratio above the design pressure ratio the nozzle lose this capability. In addition, plug nozzle have ease in vehicle and engine integration [4].

1.3 The project

The objective of this work is to complete a preliminary study of a cold gas aerospike engine experiment for orbital and space applications. The intensive scientific investigation of aerospike nozzles and the possibilities of their thrust vector control form an essential part of the research field Space Transportation at the Chair of Space Systems at the Technische Universität Dresden. In previous projects, the functionality of an aerodynamic thrust vector control for small and medium thrust classes of aerospike nozzles could be confirmed. The aim of the follow-up project MACARONIS is to develop and produce a qualified cold gas satellite engine with an additively manufactured ceramic nozzle for future space missions. Within the scope of this master thesis, a research on state of the art cold gas thruster and a first sketch of such a cold gas engine are made in preparation for the project. Computer-aided iterative design adaptations are carried out in order to determine an optimized configuration for a first experimental engine setup with regard to defined boundary conditions.

Although aerospike nozzles altitude compensating capabilities have been historically associated with SSTO, recent studies show that they could have also a significant impact on deep space missions.

According to ESA investigation [10], aerospike nozzles represent an overall better package prospect than comparable bell nozzle. This aspect is related to their high mass efficiency and the form factor ensured via truncation. These reasons mark aerospike nozzles as particularly attractive for the emerging small satellite market.

Moreover, previous studies have shown the need for higher thrust engines for small planet missions to mitigate the significant gravity losses of insertion manoeuvres. In those cases, classical bell nozzles would have large form factors, causing a difficult integration in the spacecraft itself and in the interface area with the launcher. In addition, these large form factors often cause the adoption of deployable shields for these engines, because they are a target for micro meteorites with very high strike probabilities.

Quoting ESA, “aerospikes offer an alternative route to high thrust engines for inter-planetary missions and compact and high thrust options for small satellite missions” and more “this compact packaging also makes aerospike configurations interesting for lander vehicles”.

To better understand the possibilities of aerospike nozzles, a brief list of advantages, not related with SSTO, is given below ([11] and [12]):

- A conventional bell nozzle exit radius is usually limited by the footprint of the vehicle. On the other hand, this consideration doesn't apply to an aerospike nozzle, since the flow is not bound by an outer geometrical surface. It can be said that an aerospike nozzle allows a better vehicle base area utilization
- For the same reason, an aerospike nozzle allows higher expansion ratios compared to bell nozzles, given a fixed vehicle base area
- Consequently, the higher expansion ratio leads to better vacuum performance (i.e. higher thrust) compared to conventional bell nozzles
- Some application have shown that the high expansion ratio produces a vacuum Specific impulse (Isp) increase of more than 10%, resulting in 10–18% propellant mass and 8–12% system mass reduction [13], whereas others[14] shows a 8-9% mass reduction
- The absence of the external surface and the implementation of truncation lead to a very compact form factor
- Exploration and interplanetary missions would make use of the altitude compensating capabilities of aerospike engines

According to ESA, the diffusion of aerospike nozzles has been drastically limited in the past by two major disadvantages: the cooling of the spike and the manufacturing of the engine layout.

The central spike is subject to a much higher thermal load than a conventional bell nozzle. This could be solved by cooling the spike using propellant circulation, heat exchangers or transpiration cooling, or directly truncating the spike to limit the exposed area (or a combination of them). MACARONIS project solves the thermal issue through the implementation of a cold gas engine.

In the past, the manufacture of an aerospike engine was more difficult and extremely more expensive than a classical rocket nozzle. According to ESA investigation, the advent of additive manufacturing is supposed to ease the manufacturing of such nozzles and, more importantly, the advance in ceramic materials could resolve or at least mitigate the thermal issue.

Ceramic 3D-printing could make possible the realization of a ceramic nozzle, with integrated features such as coolant channels inside the spike, without the need of drilling channels with CNC machines. Regarding this topic, MACARONIS team has a partnership with state of the art ceramic 3D-printing companies.

Chapter 2

Research

The purpose of this chapter is to identify and analyse the current developmental status of cold gas propulsion technologies for small spacecraft and to present an overview of the available systems. Critical components of a cold gas thruster are then identified and analysed.

2.1 State of the art of cold gas thrusters

Cold gas thruster, thus providing limited space propulsion, are one of the most mature technologies for small spacecraft due to their simplicity. These system have been used intensively in small satellites, from 60's to the present day. Thrust is produced by the expulsion of an inert, non-toxic propellant which can be stored in high pressure gas or saturated liquid forms. As opposed to traditional rocket engines, in cold gas thrusters no combustion takes place and so the amount of thrust produced is lower compared to conventional rocket engines. Cold gases are used in small buses due to their simplicity, low costs and reliability, but more importantly, they are used when a small total input is required, for example for attitude control applications and the association of small volume and low weight.

Anis [15] says that cold gas propulsion " has proven to be the most suitable and successful low thrust space propulsion for LEO maneuvers, due to its low complexity, efficient use of propellant which presents no contamination and thermal emission besides its low cost and power consumed " and that " the major benefits obtained from this system are low budget, mass, and volume".

Table 2.1 shows the current state-of-the-art for cold and warm gas propulsion systems for small spacecraft, selected by NASA Ames' Mission Design Division [16].

Product	**Manufacturer**	**Thrust**	**Specific Impulse [s]**	**Propellant**	**TRL status**
MicroThruster	Marotta	0.05-2.36 N	65	Nitrogen	9
Butane Propulsion System	SSTL	0.5 N	80	Butane	9
Nanoprop 3U	Gomspace/ NanoSpace	0.01-1 mN	60*-110	Butane	9
Nanoprop 6U	Gomspace/ NanoSpace	4-40 mN	60*-110	Butane	9
MiPS Cold Gas	VACCO	53 mN	40	Butane	7
MarCO-A and B MiPS	VACCO	25 mN	40	R236FA	9
CPOD	VACCO	10 mN	40	R236FA	7
POPSAT-HIP1	Micro Space	0.083-1.1mN	32-43	Argon	8
CNAPS	UTIAS/SFL	12.5 – 40 mN	40	Sulfur Hexafluoride	9
CPOD	VACCO	25 mN	40	R134A/ R236FA	6

Table 2.1: Cold gas propulsion system.
*Information was taken from brochure and may need to be updated by vendor

2.1.1 Cold Gas Micro Thruster

Marotta [17] built and qualified the Cold Gas Micro Thruster (CGMT) that flew on the NASA ST-5 mission for fine attitude control. The CGMT is shown in figure 2.1. The ST-5 micro-satellite (launch mass 55kg) successfully launched on March 22, 2006. When coupled with its control electronics, this Thruster achieves < 1 Watt Peak Power, has a response time of 5ms and it uses gaseous nitrogen as propellant. The CGMT system consists of a high-pressure tank, a pressure sensor and a thruster in blow-down configuration, producing ~ 2.36 N thrust at 154 bar that becomes ~ 105 mN at 7 bar, near exhaustion. Including tubing, its maximum length is 65.8 mm.

Figure 2.1: Illustration of the CGMT device [17]

2.1.2 SSTL: DMC and SNAP-1

Surrey Satellite Technology Ltd (SSTL) used a butane propulsion system in several small spacecraft missions. It has been used in a wide range of applications in Low Earth Orbit (LEO) and Medium Earth Orbit (MEO). Traditionally, nitrogen is a very common propellant for cold gas thruster but, as Gibbon says [18], from a small satellite point of view there are more advantages on using liquefied gases. Nitrogen stores at a relatively low density, even at high pressure, and consequently the tank has to be heavy and large. Butane is the propellant used by SSTL because it can be store as a liquid, its theoretical Isp is just 10% inferior than nitrogen one but its density Isp is much higher, 362 Ns per litre compare to 165 Ns per liter for nitrogen. One of the SSTL propulsion system is shown in Fig.2.2, the DMC propulsion system [19]. This propellant system has a dry mass of < 6 kg and the propellant mass is 2.35 kg. Each propulsion system has two butane propellant tank with a storage pressure of 200 bar.

Figure 2.2: DMC butane propulsion system [19]

The butane is stored in liquid phase. As it is possible to see in figure 2.3, the outlet of one of these tanks is connected to a series of solenoid valves, isolating the single resistojet thruster. Moreover, a heater is used to vaporize the propellant and improve the specific impulse performance with respect to the cold gas mode. In fact, a drawback of using butane is that it must be expelled in gaseous form, vice versa the Isp is reduced dramatically. Another advantage of butane is that at 20 °C, its vapour pressure is 2.1 bar absolute. This has two significant advantages: firstly, a traditional thin wall spacecraft tank (storage pressure 20 bar) will be suitable, with high safety factors. Secondly, it is possible to use 2.1 bar as a chamber pressure, avoiding the use of complex pressure regulators.

This system has been in design for more than five years and uses a RS-422 electrical interface. The propellant flow under its own vapor pressure, approximately 2 bar absolute, and it is ejected by a conventional convergent/divergent nozzle with a throat diameter of 0.42 mm. Every machined component is made of stainless steel.

Another low cost butane propulsion system (Fig.2.4) has been built by SSTL and flown on its first nano-satellite: SNAP-1. This spacecraft was equipped with a small cold gas propulsion system utilising 32.6 grams of butane propellant, which, since launch, has been used to raise the spacecraft's semi-major axis by over 3 kilometres.

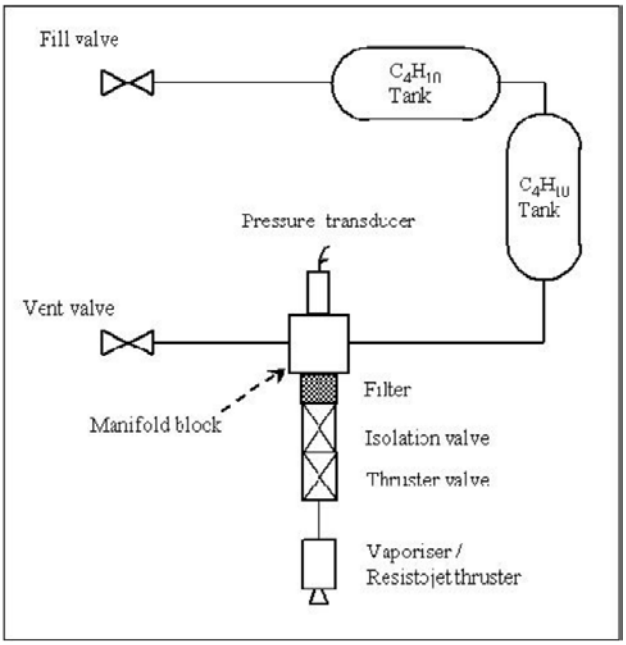


Figure 2.3: Schematic of the DMC butane propulsion system [19]

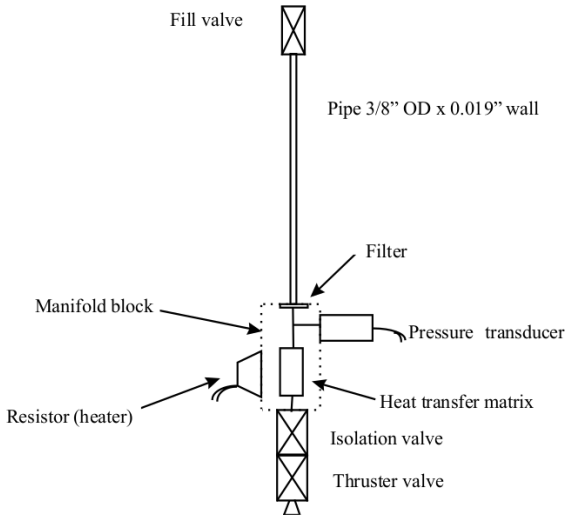


Figure 2.4: Schematic of the SNAP-1 propulsion system [18]

2.1.3 CNAPS

In June 2014, two 15 kg small spacecraft were lauched by Space Flight Laboratory at University of Toronto Institute to demonstrate formation flying. The Canadian Nanosatellite Advanced Propulsion System (CNAPS) [20] is shown in Figure 2.5. It was used for orbital control maneuvers, as orbit acquisition and phasing, station keeping and formation control. CNAPS used liquid sulfur hexafluoride, a non-toxic propellant. Its high vapor pressure and density were crucial for making CNAPS a self-pressurizing system [21].

Two filters were included in the system to remove contaminants that would damage the solenoid valves. A pressure relief valve on the storage tank was used in order to assure safety on the ground or the launch vehicle, in case of an overpressure event. CNAPS was designed with for thrusters, which nozzles are located on a single face of the spacecraft bus. Attitude control was possible because the nozzle were located at a certain distance from the center-of-mass, allowing the system to be used for momentum management.

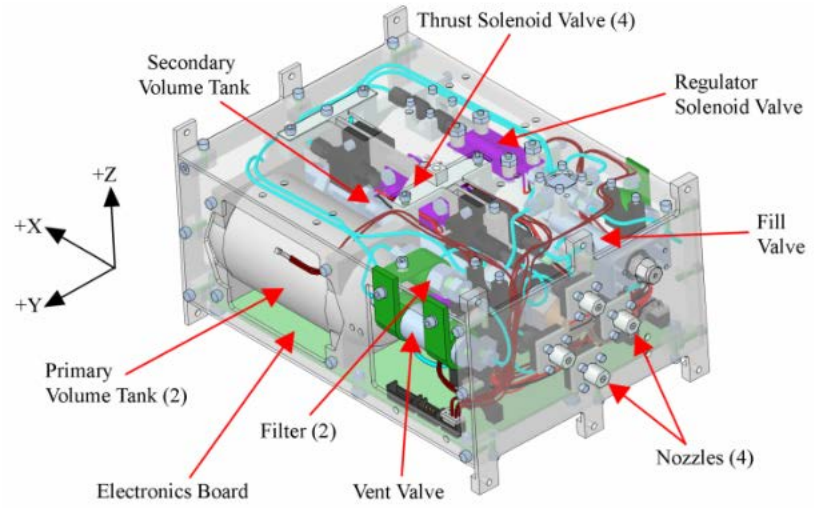


Figure 2.5: Interior view of CNAPS [20]

2.1.4 POPSAT

POPSAT-HIP1 Cubesat mission was launched in June 2014, featuring a micropropulsion system designed by Microspace Rapid Pte Ltd in Singapore. Microspace micropopulsion system is composed by 8 micro-nozzles, 6 of them placed on the same face and the other 2 along one of the long side of the satellite, as shown in figure 2.6. Each nozzle provides 1mN nominal thrust at 5 bar of Argon propellant. An electromagnetic microvalve operates each thruster, assuring an opening time of 1 ms and unlimited longest opening time. The total delta-V has been estimated from laboratory data to be between 2.25 and 3.05 m/s [22]. The block diagram of the propulsion system is shown in Fig.2.7.

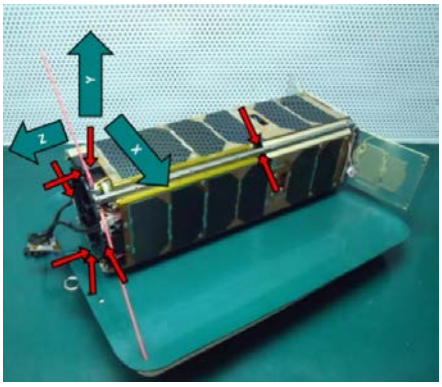


Figure 2.6: Position of the thruster in POPSAT [22]

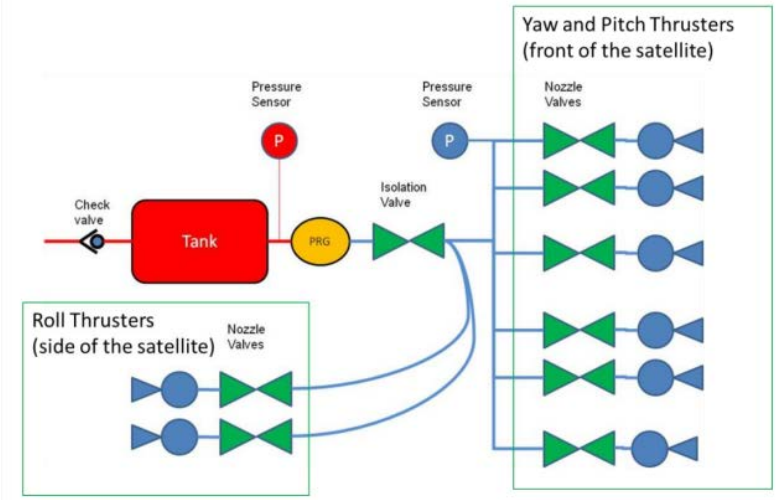


Figure 2.7: Block Diagram of the POPSAT-HIP1 propulsion system [22]

2.1.5 MEMS

Nanospace developed a micro-electromechanical systems (MEMS) featuring four butane propellant thruster, suitable for cubesat platform. Closed loop thrust control is implemented in the system, which means that the thruster is capable of proving a thrust magnitude from zero to full capacity $1mN$ with 5 μN resolution, with a feedback of the delivered thrust in real time.

The propulsion module dimensions are 10x10x3 cm and the dry mass of the system is 0.220 kg, with an average power consumption of 2 W during operation. Figures 2.8 and 2.9a show the block schematics and the design of the propulsion module. The module contains four complete microthrusters, a propellant tank and a fill/drain valve. A single microthruster, as shown in figure 2.9b includes the isolation valve, the proportional flow control valve, that assures continuous thrust regulation, the thrust chamber including the gas heaters, the mass flow sensor, providing the real-time feedback, and the nozzle [23].

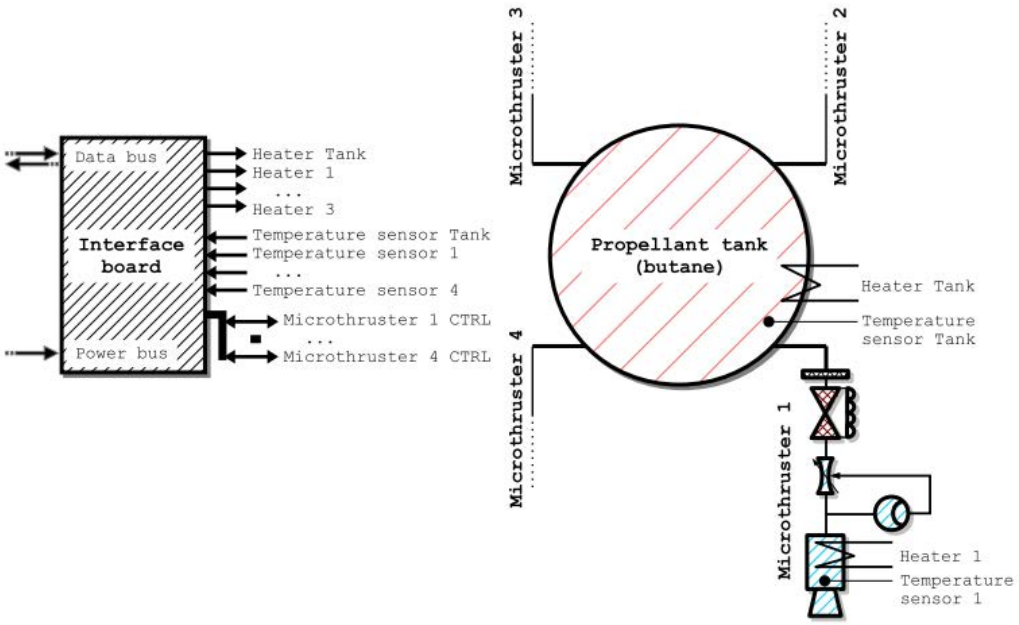
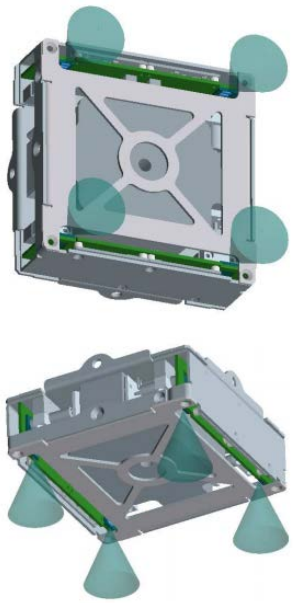
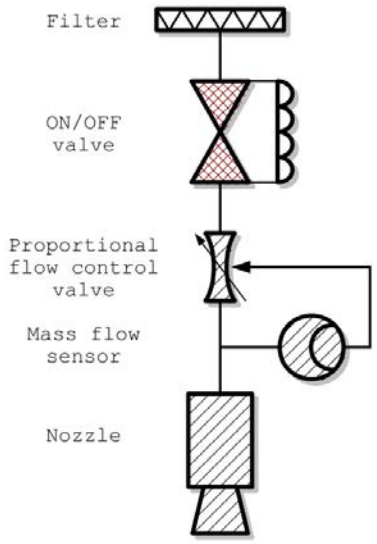


Figure 2.8: Complete cold gas propulsion system based on MEMS technology and the interface electronics board, which connects to the spacecraft power bus and data bus [23]



(a) Mechanical design of the cold gas propulsion module



(b) A single microthruster with all components

Figure 2.9: Details of the MEMS propulsion system [23]

2.1.6 CPOD

Tyvak Nano-Satellite Systems developed the CubeSat Proximity Operations Demonstration (CPOD) mission, which exploits two cubesat for maturing rendezvous, proximity operations and docking maneuvers. The cubesats utilizes a reliable cold gas propulsion system developed and build by Vacco Industries. It uses self pressuring R134a propellant, that provides high volume efficiency. The system provides up to 186 N-s of total impulse, with a specific impulse of 40 s and a nominal thrust of 10mN. Eight thrusters are distributed in pairs at the four corners of the module. The system has been tested extensively in vacuum environment, up to 70000 firings. Figure 2.10 shows the propulsion system in the central unit [24].

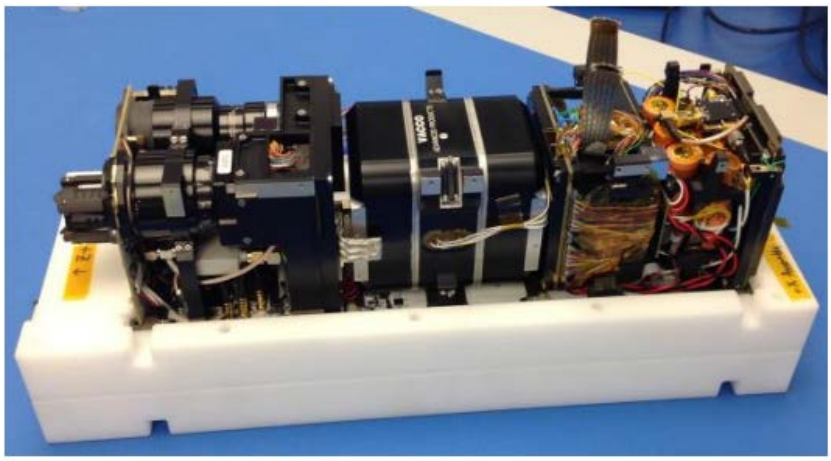


Figure 2.10: Internals of CPOD showing 1U of RPOD Payload (Left), 1U of Propulsion (Middle) and 1U of Avionics (Right) [24]

2.1.7 VACCO: CGPS

Vacco [25] built another interesting cold gas propulsion system, based on the principles of simplicity and cost-effectiveness. The Cold Gas Propulsion System (CGPS) uses a simple blowdown system, without need of the traditional pressure regulator. Thus, the propulsion system operates at a pressure up to the full storage pressure. This required the development of thrusters capable of functioning with inlet pressures to 207 bar. The thruster utilizes latching valves that require a single electrical pulse to open or close, minimizing the power consumption because between pulses the thruster is magnetically latched in either the open or closed position. In figure 2.11 the flight system schematic is shown.

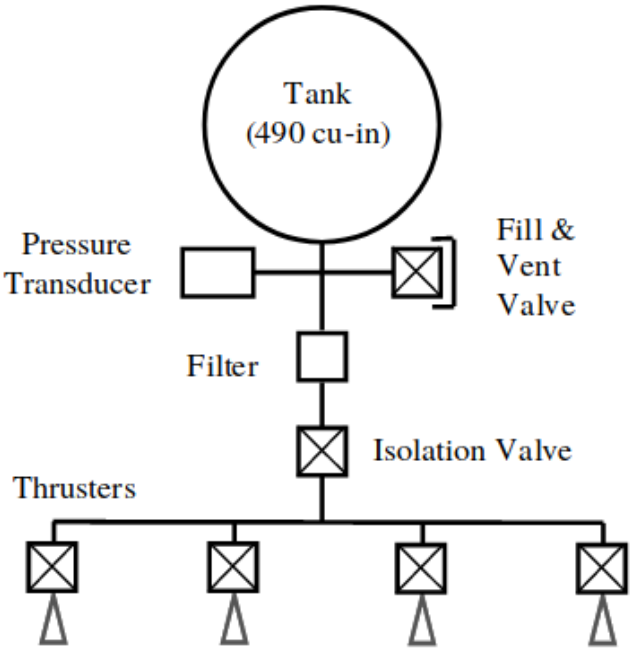


Figure 2.11: Flight system schematic [25]

The 8 liter tank stores gaseous nitrogen at a pressure of 207 bar. The system then features a Fill & Drain Valve, facilitating filling and venting operations. A filter is inserted to protext the 4 thruster from particle contamination.

2.1.8 GRACE

Although not on NASA's list, Gravity Recovery and Climate Experiment (GRACE) cold gas attitude control, flown on Grace Mission, is particularly interesting. The primarily driving requirements for the cold gas propulsion system were [26]:

- No single point of failures are allowed with the exception of fracture of tank and pipe, cloaking of filters and fill and drain valve opening
- For a mass balanced spacecraft the tanks shall be located symmetrically about the center of mass (a design with only one tank is not possible because it is required that accelerometer must be located in the spacecraft's center of mass)
- The variation of the spacecraft's center of mass shall be less than $\pm$ 2 μm over one orbit ($\sim$ 90 minutes) and less than $\pm$0.1 mm during the time the two satellites are in scientific operational mode ($\sim$6 months)
- The budget of a small satellite mission requires the cost to be minimized

The resulting design, as shown in Fig.2.12, is composed by two independent branches, each one characterized by the following components:

- one tank, that stores 16 kg of gaseous nitrogen (GN2), operating at a pressure of 320 bar and a temperature of 30°C and equipped with three thermistores for temperature monitoring
- one high pressure transducer (HPT), which records of the pressure outside the tank are used for failure detection and monitoring of the nitrogen mass
- one high pressure latching valve (HPLV), which purpose is to isolate the branch in case of malfunctioning of some components
- one pressure regulator (PR) that links the 320 bar storage pressure to the thruster's feed operating pressure of 1.5 bar
- one relief valve (RV) protects the lower pressure part form over-pressure in case of a pressure regulator leakage, opening at 3.5 bar
- one low pressure transducer (LPT) evaluates the thruster's feed pressure and it's used for failure detection and for determination of the thruster performance

- one low pressure filter (LPF) protects the thruster from contaminants
- one orbit control thruster (OTH) is used for delta-v maneuvers and itìs capable of providing 40 mN thrust, pointing in opposite flight direction
- six attitude control thruster (ATH) are used for attitude control, providing 10mN each

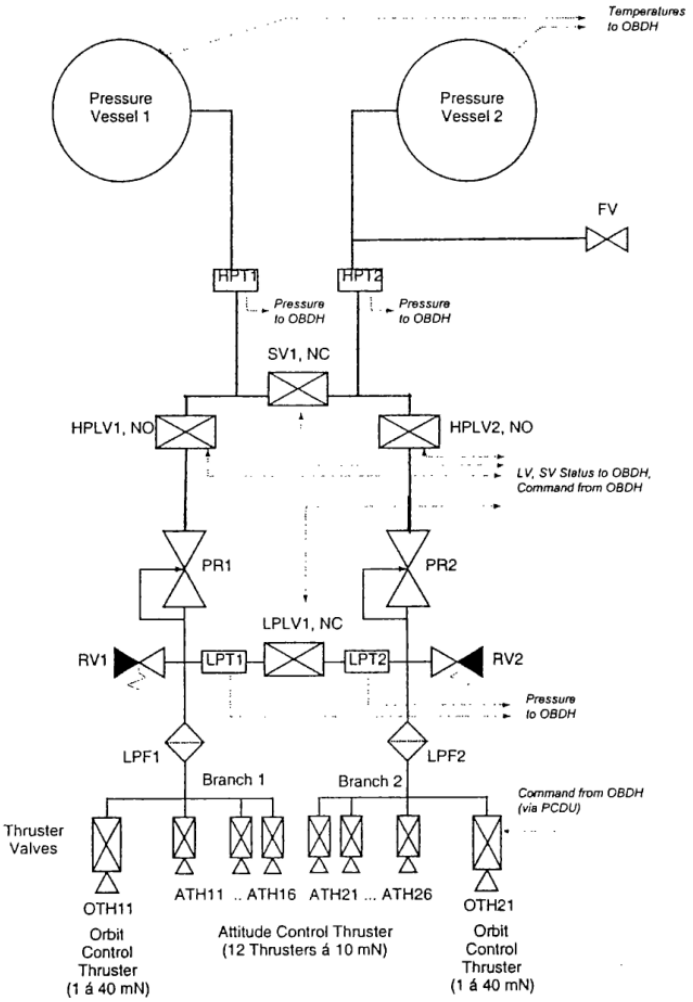


Figure 2.12: Schematic of the GRACE cold gas subsystem [26]

2.2 Main components

In this section the main components of a cold gas thruster are presented and described. Moreover, for each component some possible design options are analysed and discussed.

2.2.1 Feed system

Two categories of pressurized feed systems with stored inert gas have been developed and flown, a pressure regulated one and the so called blowdown system. The first one uses a pressure regulator to reduce the pressure from the high-pressure gas storage tank to the operating pressure. This system allows constant chamber pressure and nearly constant thrust, but needs a pressure regulator that sometimes can be complex to design or expensive to buy. The second category is called blowdown system, in which, without a pressure regulator, the gas expands at storage pressure. Vacco's CGPS is part of this category. The gas is stored under pressure inside the propellant tank and during operation, the gas expands and the tank pressure decrease steadily down to 25 to 45% of the initial values [27]. This configuration is simpler and has less components, but the major drawbacks are that the average specific impulse is lower, the tank bigger and all the components of the thruster has to be designed to withstand the storage pressure. Regarding cold gas thruster, both configurations have been used, with both monopropellant and bipropellant systems. In this work pressure regulated, monopropellant systems will be analyzed, due to the higher performance of a pressure regulated system coupled with the simplicity of a monopropellant thruster. Blowdown configuration could be implemented if the development of a pressure regulated thruster proves to be too complex or expensive. Only gas storage is studied in this work due to its simplicity and reliability, thus liquid (butane,R124a) propellant, as said in the state of the art analysis, proved to be a good choice sometimes.

The main components of pressure regualated cold gas propulsion system are shown in Tab.2.2.

Tank, including Fill/Vent valve	Propellant	Heaters, if needed
Filters	Pressure sensors	Pressure regulator
Control valves/latching valves	Relief valve	Nozzle

Table 2.2: Main component of a cold gas pressure regulated system

2.2.2 Tank

The type of propellant has to be considered when designing the tank, but the storage of gases is relatively simple. In general, spherical tanks are used, which are designed for high pressures up to 300 bar. In the past, all-titanium tanks were produced form the heat-treated alloy Ti6AI4V. In some cases, aluminum tanks are used, for their manufacturing simplicity and affordability [15]. Nowadays, composite tanks have spread in many fields, from aerospace to industrial and firefighting industry , which are thin-walled titanium tanks with Kevlar or carbon fiber reinforcements. The tanks have usually only one port for loading the gas and its withdrawal during the mission.

The space, zero-g environment has a significant influence on the orientation of liquid propellant, thus the storage of liquid propellant is more complex. A short list of possible options is given below [28].

- In spin-stabilized satellites, the centrifugal force could be enough to orient the propellant.
- For nonspinning satellites, the tank has to be designed to orient the propellant towards the feeding port
  - It is possible to use diaphragms inside the propellant tank, in order to separate the liquid propellant and the pressuring gas. This method is reliable and have several advantages. They can be qualified for up to 100 loading and unloading cycles, so they can be tested on component and subsystem level. They avoid sloshing and force the propellant towards the outlet. Disadvantages occurs for rubber diaphragms, like pressurizing gas penetration and permeability. On the other hand, metallic diaphragms have been developed by the US company ARDE Inc. They have near zero permeation rates, they can store propellant that are incompatible with rubber and they are compatible with severe requirements concerning sloshing and center of gravity stability.
  - It is possible to use blader tank. The propellant is stored in a cylindrical bladder tank and ejected through a perforated tube in the center of the tank.
  - Surface tension tanks have been developed as a competitive alternative involving monopropellant and bipropellant propulsion systems. Surface tensions are rather low, but in low gravity conditions they can become a predominant component of the liquid dynamics. They exploit narrow gaps, vanes and sponge-like structures to move the propellant towards the tank outlet.

From a structural point of view, satellite propellant tanks used in cold gas propulsion systems are either spherical or cylindrical in shape. For a spherical tank, the force F tending to separate the tank tending to separate the tank is:

$$F = P\pi r^2 \tag{2.1}$$

The stress σ can be compute as,

$$\sigma = \frac{P\pi r^2}{2\pi rt} = \frac{Pr}{2t} \tag{2.2}$$

The thickness of the tank is accurately calculated by including joint efficiency in Eqn.2.3 and is given as follows,

$$t = \frac{Pr}{2\sigma e} \tag{2.3}$$

where σ is the allowable tension, t is the thickness, r is the internal radius, P is the design pressure and e is the joint efficiency.

Eqn.2.4, developed by American Society of Mechanical Engineers (ASME), introduce an additional term of $0.2P$ in the denominator [29]. This term was added to take into consideration the non linearity in stress that develops in thick spherical shells.

$$t = \frac{Pr}{2\sigma e - 0.2P} \tag{2.4}$$

Eqn.2.4 is valid for $t < 0.365r$ and $P < 0.0665\sigma e$. It has to be said that Eqn.2.4 estimate the thickess based on pressure only. Another equation given by ASME, for the design of spherical shells due to internal pressure is the following.

$$t = \frac{D}{2}[e^{0.5P/\sigma e} - 1] \tag{2.5}$$

where D is the internal diameter. Eqn.2.5 is derived from plastic analysis and is applicable to all thickness and pressures.

In case of cylindrical pressure vessel, the hoop stress is twice that in spherical one. The longitudinal stresses in cylindrical tank the same as in spherical one. Theoretically, the hoop stress σ_h is computed from the equilibrium of forces on a cut along the longitudinal axis as,

$$2\sigma_h t dx = 2prdx \tag{2.6}$$

Thus,

$$\sigma_h = \frac{Pr}{t} \tag{2.7}$$

Eqn.2.8, introduced by ASME, include the joint efficiency and calculate the required thickness in the circumferential direction due to internal pressure,

$$t = \frac{Pr}{\sigma e - 0.6P} \tag{2.8}$$

Eqn.2.8 is valid for $t < 0.5r$ and $P < 0.385\sigma e$. In this case, the additional term of $0.6P$ was added to take in consideration the nonlinearity in stress that develops when the thickness of a cylinder exceeds $0.1r$. Similarly, the equation for the required thickness in the longitudinal direction, due to internal pressure is given as,

$$t = \frac{Pr}{2\sigma e + 0.4P} \tag{2.9}$$

Eqn.2.9 is valid for $t < 0.5r$ and $P < 1.25\sigma e$. Another equation given by ASME, for the design of cylindrical shells due to internal pressure is the following.

$$t = \frac{D}{2}[e^{P/\sigma e} - 1] \tag{2.10}$$

Eqn.2.10 is derived from plastic analysis and is applicable to all thickness and pressures.

Equations 2.2 through 2.9 are applicable to solid wall as well as layered wall constructions. For layered vessel, the Joint Efficiency Factor is calculated as,

$$e = \frac{\sum e_i t_i}{t} \tag{2.11}$$

Cilindrical shells need to be close by two head. Head configurations include hemispherical, torispherical and ellipsoidal shapes. As a reference, hemisferical and torispherical head formulas are reported. The design of an hemisferical head follow the same equations of the sferical shells, so Eqn.2.4 or Eqn.2.5 can be applied.

A commonly used ellipsoidal head has a ratio of 2:1 (base radius to depth). The required thickness is given by the following equation:

$$t = \frac{PD}{2\sigma e - 0.2P} \tag{2.12}$$

Ellipsoidal heads with a radius-to-depth ratio other than 2:1 follow the following equation:

$$t = \frac{PDK}{2\sigma e - 0.2P} \tag{2.13}$$

where

$$K = \frac{1}{6}\left[2 + (\frac{D}{2h})^2\right] \tag{2.14}$$

and $D/2h$ varies between 1.0 and 3.0.

Torispherical heads can be designed using the following equation:

$$t = \frac{PLM}{2\sigma e - 0.2P} \tag{2.15}$$

where

$$M = \frac{1}{4}\left[3 + \left(\frac{L}{r}\right)^{1/2}\right] \tag{2.16}$$

and L/r varies between 1.0 and 16.67

2.2.3 Propellant

The choice of the propellant determines the performance of a propulsion system, as well as the design and optimization of the thruster. A cold gas thruster is characterized by two conditions: the propellant has to be in gas phase when ejected from the thruster and no combustion should occur. The choice of the propellant is guided by some characteristics, such as an higher atom weight and moderate low boiling and melting temperature. It has to be said that mass efficient storage of the gas is a major concern. Other considerations, regarding for example price and toxicity, may be important in most cases. Table 2.2.3 shows the performance of the cold gas propellant analysed [30]. Xenon, an heavy and inert gas, is a potential cold gas propellant, due to its high density and molecolar weight. However, the viscosity increases with temperature within a certain range, creating some challenges when using this gas and decreasing the overall Isp, that becomes very low if heating Xenon. On the other hand, helium and hydrogen have much higher specific impulse but they are lighter than Xenon. Their low density leds to bigger tanks, and their temperature for liquid or solid storage are technically demanding. For example, hydrogen would be an ideal propellant with an extremely high Isp, but it requires cryogenic storage or extremely large tanks. Generally, nitrogen is the first choice in choosing gases. This is because its storage is simpler than helium and hydrogen and its higher molecular size allows a minor leakage percentage. Gaseous nitrogen, GN_2 is inert, colorless, odorless, nontoxic, noncorrosive, and nonflammable. Since gaseous nitrogen is inert, special materials of construction are not normally required. The use of gaseous nitrogen instead of liquid cryogenic nitrogen, involves lower costs for hardware development and production as will be the costs associated with maintaining a less complex storage system.

Propellant	**Isp**[a]	**Isp**[a]	ρ	M_w	T_m	T_b
	s	s	g/cm^3	kg/kmol	$^\circ C$	$^\circ C$
H_2, Hydrogen	296	272	0.02	2.0	-259	-253
He, Helium	179	165	0.04	4.0	-272	-269
Ne, Neon	82	75	0.19	20.4	-249	-246
N_2, Nitrogen	80	73	0.28	28	-210	-196
Ar, Argon	57	52	0.44	39.9	-189	-186
Xe, Xenon	31	28	2.74[b]	131.3	-112	-108
CH_4, Metane	114	105	0.19	16	-182.5	-161.5
C_3H_8, Propane	...	...	Liquid	41.1	-187.7	-42.1
C_4H_{10}, Butane	...	...	Liquid	58.1	-138.3	-0.5
CO_2, Carbon Dioxide,	67	61	Liquid	44	...	-78 (S)
SF_6, Sulfur hexafluoride	...	...	...	146.1	...	-64 (S)
R134A	71.61	65.1	1.225	102	...	-26.1
R236FA	58.7	53.4	1.373	152	...	-1.44

Table 2.3: Cold gas propulsion system.
[a] At 25°C. Assume expansion to zero pressure in the case of the theoretical value
[b] Likely stored at lower pressure value (138 bar) to maximize propellant-to-tank weight ratio (S) stands for sublimation

Although gaseous nitrogen storage is simpler, it requires larger and thicker pressure vessel designs. On the other hand, cryogenic tanks are lighter and smaller. VACCO [25] used a nitrogen propulsion system, storing nitrogen at 207 bar in the 8.0 liters tank. Ghasemi and Rezaeiha [31] designed a gaseous nitrogen propulsion system, storing 6 kg at 330 bar in a 16.5 liters empty tank volume. Grace mission [26] used 16 kg of gaseous nitrogen stored at a pressure of 320 bar and a temperature of 30°C.

Propane, C_3H_8, was used as cold gas propellant in microsatellites in the past.

Butane, C_4H_{10}, has been used in many mission recently due to the possibility to store it in liquid form and its self-pressurizing capabilities. However, these gases are organic compounds and hazard classified, so it is more complex to handle, rising the costs. SSTL [18] used butane in several small spacecraft mission.

Carbon dioxide, CO_2, theoretically has the advantage to sublimate directly from -78°C at a pressure of 1 bar. Moreover it provides a fairly high specific impulse. Thus, up to now there are no small spacecraft that used a low temperature or cryogenic CO_2 thruster. On the other hand, Lev and Herscovitz [32] used 500g of CO_2, stored in liquid phase at a pressure of 150 bar and a temperature of 80°C when heated. Each time before operating the propulsion system, the entire propellant tank is heated until the propellant is fully vaporized. The use of the heated gas propulsion system holds a few key advantages over the two previously described conventional systems:

- Most gases, near their critical temperature, have a compressibility factor below one (Z< 1), reducing tank pressure, when the propellant is fully vaporized
- The propellant is liquid during the majority of the mission and it is vaporised only during operations. When the propellant is liquid, it is self-pressurized at the vapour pressure related to the cold tank temperature
- Controlling the propellant temperature makes possible steady and continuous firing since all the propellant is in gaseous form

The drawbacks related to the heating are:

- A meticulous thermal design is required to limit the power losses induced by the heating of the entire tank that inevitably transfer some heat to the rest of the system, reducing the power efficiency of the overall propulsion system.
- This design reduces the firing readiness level because, prior each operation, the tank has to be heated and the propellant vaporized. This action may take tens of minutes
- Due to repetitive thermal cycles, the requirement on the heating system are severe and this makes the heating system more complex and less reliable than a common cold gas propulsion system

Lastly, sulfur hexafluoride, SF_6, has a molecular weight higher than Xenon, sublimates at -64°C at a pressure of 1 bar. Moreover, it is non-flammable and not toxic. These characteristics makes it an interesting candidate for new cold gas thruster technologies.

Gas storage

Ley [28] suggests a simple method to compute the tank volume using the general gas equation:

$$P \cdot V_T = m_G \cdot R \cdot T \cdot Z \tag{2.17}$$

where P is the maximum allowed tank pressure $[N/m^2]$, V_T is the tank volume $[m^3]$, R is the gas constant $[m^2/(s^2K)]$, T is the gas temperature $[K]$ and Z is the compressibility factor. The compressibility factor adjusts the ideal gas law to account for the real gas behaviour. It is important to not forget the influence of pressure on the compressibility factor, since for nytrogen it amounts to 0.9954 at 20 bar and 293 K, while it amounts to 1.1182 at 280 bar and 293 K, corresponding to a volume increase of about 11 %. The compressibility factor for specific gases can be read from generalized compressibility charts that plot Z as a function of pressure at constant temperature.

Liquid storage

The propellant is stored in liquid phase while the pressure is the vapor pressure at the storage temperature. Propellant is let out from the tank into a plenum, where it is vaporized through expansion or external heating. In some cases, like [32], liquid propelland is vaporized inside the propellant tank. For simplicity, only gas storage will be evaluated in this project.

2.2.4 Filters

Thus they are usually included in the pressure regulator, a filter in the high pressure section protects the pressure regulator from particle contamination. Most of the low pressure filters has a 2 μm mesh, thus high pressure ones reach 20 μm mesh. As a reference, table 2.2.4 shows some of the properties of a filter produced by Omnidea-RTG [33].

2.2.5 Pressure sensor

A pressure sensor, or pressure transducer, is a device that generates a signal as a function of the pressure imposed. More complex systems, like GRACE [26], have an high pressure trasducer and a low pressure trasducer.

Operating Media	Inert gas
Maximum expected operating pressure	350 bar
Leakage Rate	0 ssc/s (all welded design)
Maximum Flow Rate	17 l/min at ΔP of 1 bar
Filter Mesh Size	2 μm
Mass	0,076 Kg
Dimensions	50 x 30 x 30 mm
Materials	Stainless Steel

Table 2.4: Filter produced by Omnidea-RTG

ETM/ETL-422(X)-375 (M) SERIES, made by Kulite Sensors Ltd., it is a miniature threaded pressure transducer that can be configured in both high and low pressure mode. It weighs maximum 80 grams. The hexagonal head and o-ring seal make it easy to mount and simple to apply. It's operating temperature range is -55°C to +135°C. A solid state piezoresistive sensing element is located behind a metal diaphragm that behaves like a force collector. A film of non-compressible silicone oil transfer the force to the sensing element. This design makes the device highly stable, reliable and rugged, maintaining the advantages of microcircuit, such as miniaturization, repeatability, low power consumption and high natural frequencies, which makes the device suitable for shock pressure measurements. A version of this has been used in SSTL nano-satellite SNAP-1 (2.1.2).

2.2.6 Pressure regulator

A pressure regulator is a control valve that reduces the input pressure of a fluid to a desired value at its output. Regulators are used for gases and liquids, and can be an integral device with an output pressure setting, a restrictor and a sensor all in the one body, or consist of a separate pressure sensor, controller and flow valve. VACCO Aerospace Products offers a compact, pre-integrated Pressure Regulator and Relief Valve that is ideally suited to cold gas propulsion systems. It weighs about 360 grams.

2.2.7 Latching valve

A latching valve takes a momentary electrical signal and delivers a constant pneumatic output, until a second signal is delivered. For this reason, it is usually called bi-stable valve. In other words, latching valves open or close after receiving an electrical input. Vacco produces both high and low pressure titanium latching valves maintaining the weight low(340 grams for both configurations). Most latching valves have a preferred state and they are called “normally open” or “normally closed”.

2.2.8 Relief valve

Most components are designed to operate within a precise pressure range. A relief valve is crucial to protect these components from overpressure and avoid possible damages to the entire system. They are the safeguards which limit maximum pressure in a system and they are implemented in most reliable designs. Vacco Aerospace produces space qualified relief valves, with over 10 years of successful flight heritage. Low leakage, over a wide range of temperatures, is guarantee thanks to the implementation of a teflon set design.

Chapter 3

Design

In this chapter, a preliminary definition of objectives, requirements and constraints is presented. Then, two raw configurations are shown and the design drivers for the optimization algorithm tabled.

3.1 Objectives

Within the framework of the MACARONIS project, the cold gas thruster will be launched in orbit and will operated in space environment. The aim of this work is not to design the entire experiment, but is to develop a first iteration of the thruster including all its components. Usual subsystems like the power distribution and the thermal control system are not taken into account in the evaluation but they are included in the mass estimation. Thus, the requirements and constraints presented in section 3.2 do not include structural, thermal, electronic or communication considerations.

The objectives of the MACARONIS project are to:

- design a cold gas aerospike engine with an additively manufactured ceramic nozzle
- measure its performance in space environment
- qualify the additively manufactured ceramic aerospike engine

3.2 Requirements and Constraints

3.2.1 Functional Requirements

F.1: The experiment shall operate the thruster.
F.2: The experiment shall measure the acceleration of the satellite*.
F.3: The experiment shall measure the pressure of the thrust chamber.
F.4: The experiment shall measure the temperature of the thrust chamber.
F.5: The experiment shall measure the pressure of the gas tank.
F.6: The experiment shall measure the temperature of the gas tank.
F.7: The experiment shall keep the tank at a constant temperature.

*:at the moment, it is not clear if the experiment is allowed to change the attitude of satellite. In order to measure the force produced by the thruster, the experiment requires the measurement of the acceleration of the satellite or in alternative, of the counter force produce by the Attitude Determination and Control System (ADCS) of the satellite itself.

3.2.2 Performance Requirements

Relate to F.1

P.1.1: The thruster shall produce a minimum acceleration of 0.01 m/s^2*
P.1.2: The thruster shall operate for at least 30 s.
P.1.3: The thruster should operate for 60 s.
P.1.4: The response time should be maximum 100 ms.

*: at the moment, it is not clear the mass of the satellite in which the experiment is going to be mounted. This requirement has been written considering a small satellite of mass less than 500 kg.

Relate to F.2

P.2.1: The acceleration measurement shall be made with a resolution at least 0.001 m/s^2.
P.2.2: Sample the acceleration with a frequency of 1 KHz.

Relate to F.3

P.3.1: The experiment shall measure the pressure inside the thrust chamber with a minimum resolution of 10 mbar

P.3.2: The experiment shall measure the pressure inside the thrust chamber with a frequency of 1 KHz, when the thruster is switched on.

P.3.3: The experiment shall measure the pressure inside the thrust chamber with a frequency of 10 Hz, when the thruster is switched off.

Relate to F.4

P.4.1: The experiment shall measure the temperature inside the thrust chamber with a minimum resolution of 0.1 K.

P.4.2: The experiment shall measure the temperature inside the thrust chamber with a frequency of 1 KHz, when the thruster is switched on.

P.4.3: The experiment shall measure the temperature inside the thrust chamber with a frequency of 10 Hz, when the thruster is switched off.

Relate to F.5

P.5.1: The experiment shall measure the pressure inside the gas tank with a minimum resolution of 10 mbar

P.5.2: The experiment shall measure the pressure inside the gas tank with a frequency of 1 KHz.

Relate to F.6

P.6.1: The experiment shall measure the temperature inside the tank with a minimum resolution of 0.1 K.

P.6.2: The experiment shall measure the temperature inside the tank with a frequency of 1 KHz.

Relate to F.7

P.7.1: The tank temperature variation shall not exceed $\pm$ 1 K.

3.2.3 Design Requirements

D.1: The experiment shall have a maximum mass of 12Kg.
D.2: The thruster shall be able to operate in vacuum environment.
D.3: The thruster shall produce a measurable force.
D.4: The thruster should be able to perform multiple shots.
D.5: The nozzle shall be designed for ceramic additive manufacturing.

3.2.4 Operational Requirements

O.1: The experiment shall be able to activate the thruster autonomously.
O.2: The experiment shall be able to acquire and store data autonomously.

3.2.5 Constraints

C.1: The experiment cost shall not exceed the defined cost budget of 25'000 €, including a 10 % safety margin.

3.3 Raw design of the system

Two schematic configuration were developed, based on the research on the state of the art and following the requirements and constraints, in order to complete the objectives. The design of both configuration is based on the cold gas subsystem used on the two satellites of the GRACE mission.

3.3.1 OLC

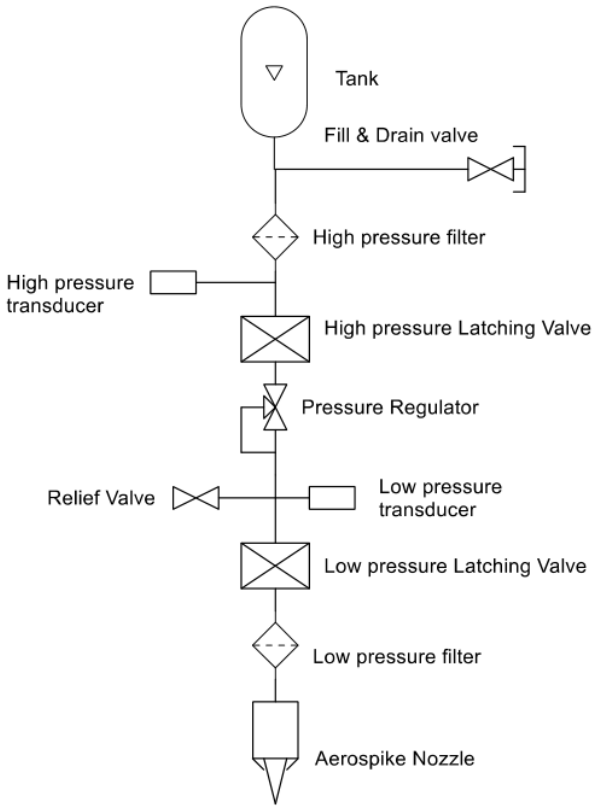


Figure 3.1: Schematic of the OLC

The first configuration, as shown in Fig.3.1, is composed by a single fluidline, called One line configuration (OLC), from the tank to the nozzle and consists of:

- A tank, equipped with a Fill/Drain valve.
- An high pressure filter is inserted to protect the pressure transducer from contamination. It has to be said that most pressure transducer are equipped with filters, so this component may be deleted in future analysis.
- An high pressure transducer for failure detection. If the high pressure transducer indicates that the high pressure decreases by a certain value, it means an high leakage in the low pressure part of the systems.
- An high pressure latching valve, normally opened and used for isolation in case of failure. For example, in case of leakage of the low pressure part, it is possible to close the line with the high pressure latching valve, avoiding the diffusion of the gas inside of the satellite.
- A pressure regulator, used to reduce the pressure from the storage pressure to the operating pressure of the thrust chamber.
- A low pressure transducer, coupled with a relief valve. The low pressure transducer measures the pressure at the nozzle's inlet and it is used for performance evaluation. Moreover, one relief valve is introduced and opens at a certain pressure above the operating pressure, protecting the low pressure part from overpressure in case of a pressure regulator leakage.
- A low pressure latching valve, normally closed and opened during operations.
- A low pressure filer, to protect the thruster from contamination
- A toroidal aerospike nozzle was chosen because of its simplicity. It doesn't suffer from surrounding flow side disturbs (like a linear one) and the drawbacks connected to the thermal expansion of the toroidal thrust chamber doesn't apply to cold gas thrusters. It's easier to design, to simulate and to manufacture.

Table 3.1 shows a mass budget for this configuration. As already said, the total mass of the experiment is up to 12 Kg. The optimization program calculates the best combination between the gas mass, tank mass and nozzle characteristics in the given maximum mass, so it is important to give an estimation of the other components mass.

The program uses the Maximum Estimated mass, that is calculated from the Current Best Estimated Mass, with a 30 % increase due to the fact that it is a preliminary estimation. The ceramic nozzle mass is included in the 30%. The pipes mass has been estimated as a proportion with other state of the art cold gas thruster.

Component	**Mass [kg]**
Fill/Vent Valve[1]	0,113
Pressure regulator[1]	0,360
High pressure transducer[2]	0,080
High pressure Latching Valve[1]	0,340
2xFilter[3]	0,152
Low pressure transducer[2]	0,080
Low pressure Latching Valve[1]	0,340
Relief Valve[1]	0,280
Piping[4]	0.25
Other Sensors[4]	0.5
Electronics[4]	0.5
Current Best Estimated Mass	3.245
Maximum Estimated Mass	3.8935
Available Mass	8.1065

Table 3.1: Mass estimation of the principal components
[1] : data based on Vacco Aerospace products
[2] : data based on Kulite Sensors Ltd. products
[3] : data based on Omnidea-RTG products
[4] : mass budget for non designed elements

3.3.2 DLC

A second configuration was developed, based on the principle of redundancy, and it is formed by two fluid lines composed by the same components of the OLC, hence the name Double line configuration (DLC). In this design, presented in Fig.3.2, no single points of failure are allowed with the exception of the fracture of the tank or pipe, cloaking of filters and Fill/Drain valve opening.

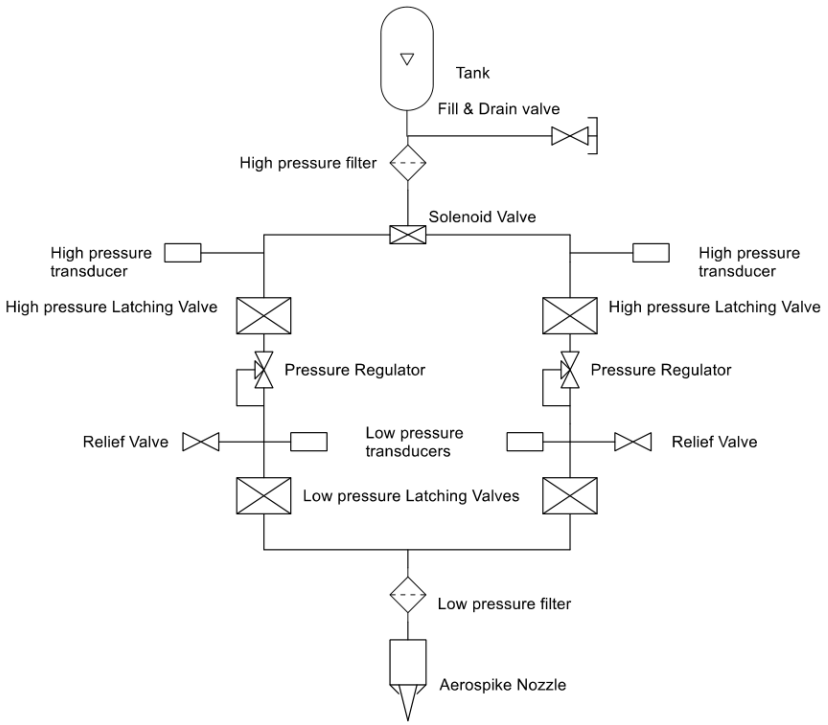


Figure 3.2: Schematic of the DLC

One fluid line is set as the primary one. The other is set as the secondary one and it is meant to be used in case of malfunctioning of the primary one. If the high pressure transducer indicates that the high pressure decreases by a certain value, it means an high leakage in the low pressure part of the systems. In this case, it is possible to close the high pressure latching valve and use the redundant fluid line. This configuration is far more safe then the first one, but the increased number of component increases the overall costs and decreases the available gas mass. The new mass budget is shown in Tab.3.2

Component	**Mass [kg]**
Fill/Vent Valve[1]	0,113
Pressure regulator[1]	0,720
High pressure transducer[2]	0,160
High pressure Latching Valve[1]	0,68
2xFilter[3]	0,152
Low pressure transducer[2]	0,160
Low pressure Latching Valve[1]	0,680
Relief Valve[1]	0,560
Solenoid Valve[1]	0,36
Piping	0.5[4]
Other Sensors	0.5[4]
Electronics	0.7[4]
Current Best Estimated Mass	5.285
Maximum Estimated Mass	6.8705
Available Mass	5.1295

Table 3.2: Mass estimation of the principal components
[1] : data based on Vacco Aerospace products
[2] : data based on Kulite Sensors Ltd. products
[3] : data based on Omnidea-RTG products
[4] : mass budget for non designed elements

3.3.3 Design Drivers

The design drivers obtained from the analysis of the state of the art, the requirements and previous knowledge developed at TU Dresden are shown in Tab.3.3. These values are used in the optimization program illustrated in Chapter 4.

Design Driver	**Values**
Tank shape	Spherical | Cylindrical
Tank Material	Aluminium | Stainless Steel
Maximum total mass*	OLC:8.1065 Kg | DLC:5.1295 Kg
Gas Mass	0-6 Kg
Propellant	Nitrogen|Helium|Argon|Xenon|Methane
Tank Temperature	283-313 K
Tank Pressure	100-300 Bar
Trust chamber pressure	2-6 Bar
Expansion Ratio	2-8
Nozzle exit radius	10-20 mm
Throat width	>1 mm

Table 3.3: Design drivers of the optimization program
*: maximum total mass available to the program, where OLC refers to the one line configuration and DLC refers to the double line configuration

According to paragraph 2.2.2, most popular vessels are cylindrical or spherical. The tank material can be chosen between aluminium and stainless steel. Components of the shelf (COTS) industrial vessels are mostly composed by an aluminium or stainless steel part wrapped by a composite reinforcement. On the other hand, the evaluation of this complex structure is difficult and time consuming, especially since commercial vessel composition is not precisely known. Thus, in the framework of this work, the results obtained from the tank mass are for reference and the 30% added margin for the Maximum Estimated Mass takes in account this uncertainty. The analysed propellant was decided according to paragraph 2.2.3 evaluations and considering a gas storage, pressure regulated configuration.

Chapter 4

Genetic Algorithm

In this chapter, a brief introduction about Genetic algorithm (GA) is presented. Then, the logic behind the implemented optimization algorithm is discussed and every characteristic detailed.

4.1 The algorithm

A GA is part of a family of algorithms called *evolutionary computation*. Evolutionary computation is inspired by biological evolution, and is a branch of artificial intelligence. In other words, they are a family of trial and error problem solvers, a population-based metaheuristic optimization algorithms. There are many types of evolutionary algorithms, including:

- Ant colony optimization
- Artifical immune system
- Cultural algorithm
- Genetic algorithm
- Memetic algorithm
- Neuroevolution
- Particle swarm optimization
- Swarm intelligence

The listed methods represent a small fraction of all evolutionary algorithms.

The optimization algorithm used to evaluate the characteristics of the cold gas thruster is based on the genetic algorithm theory.

GAs are an efficient tool for solving problems in which both continuous and discrete variables, but also discontinuous and non-convex design spaces are involved. For this type of problems, standard nonlinear programming techniques are inefficient and computationally expensive[34]. This is possible because GAs are a zeroth order method and so derivatives are not involved in the calculation of the optimized solution. GAs evaluate a lot combinations in the design space by simulating the processes of evolution and natural selection, such as reproduction, mutation, recombination, and selection. A design string of variables is selected, and a population of design strings is rated on how well it performs in an Fitness Function (FF).

The objective of the FF is to test the solutions, such as the environment selects the organisms based on their ability to live within it. The better the solution, the highest is the probability to reproduce and so, to pass its genes to the following generation. Poor designs score bad results, so the probability to survive and reproduce is low. In this way, bad genes, i.e. useless or bad characteristics, are not transmitted to the following generation.

For this application, a custom designed GA has been implemented, combining traditional GA theory and some *Killer Queen Method* principles. The algorithm involves two cycles, an inner cycle and an outer one. The inner cycle of the GA is shown in Fig.4.1.

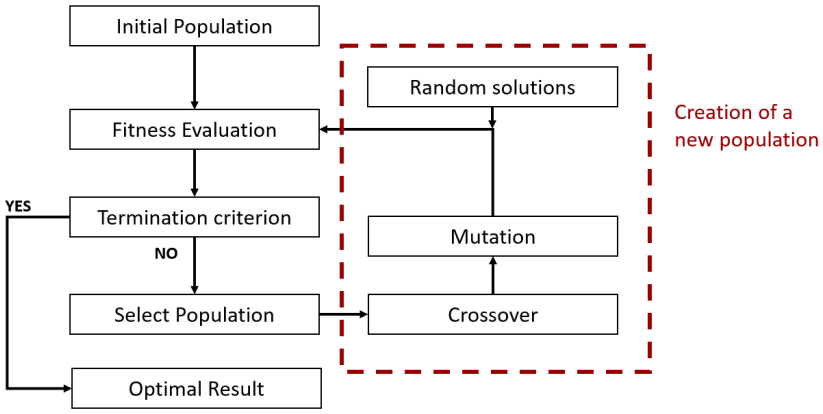


Figure 4.1: Schematic of the GA

Each step of this cycle will be explained further on. The result of the inner cycle is called *Queen*, i.e. the best best solution of the cycle. The concept of the Queen is taken from the *Killer Queen Method*, which generic schematic is shown in Fig.4.2.

In the Killer Queen method, a Queen is selected for each generation and the other solutions are deleted. The next generation is formed by the queen itself and the offspring, generated by mutation, i.e random variation, of the queen design string. It is possible to understand that this method rely uniquely on large quantities of mutations to generate the new population. This process mimics some insect colonies where a single queen generates the next population by itself.

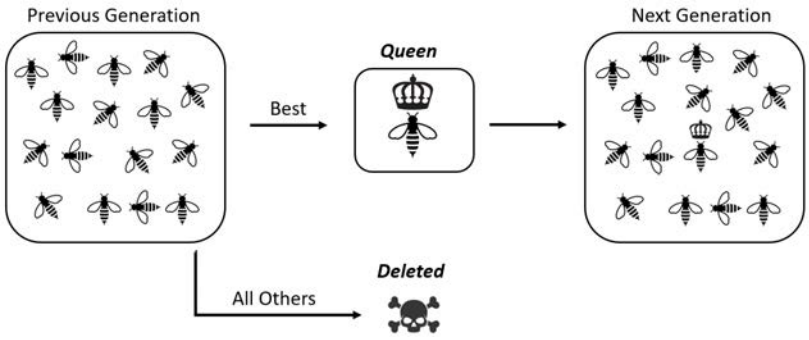


Figure 4.2: Schematic of the Killer Queen Method

In the implemented GA, the inner cycle last population is deleted, except for the Queen, that is the only information passed from the inner to the outer cycle.

Fig.4.3 shows the outer cycle. It consists on the repetition of n inner cycles (in this case 10), in order to produce n queens that are the input for the last cycle.

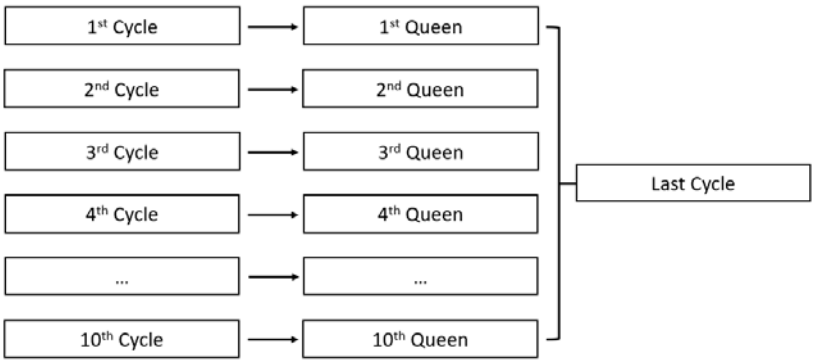


Figure 4.3: Schematic of the outer cycle

Inner Cycle

As shown in Fig.4.1 the inner cycle is composed by:

- Creation of the initial population
- Evaluation of the population using a FF
- Termination criterion, that leads to the identification of the queen (or optimal result) or to the continuing of the cycle
- Selection of the best individuals of the generation, or in other words the creation of a breeding pool
- Crossover of the design strings of the selected breeding pool individuals
- Mutation, a random variation of some element of the design strings of some individual
- Creation of the new generation

4.2 Initial Population

The design string is the core of a GA and its purpose is to represent every possible combination of attributes in a mixed design field, composed by both discrete and continuous variables. The peculiar aspect of a GA is that the initial population is composed by n individuals, each one featuring m genes. Each gene represents one design driver presented in Tab.3.3, by means of a discrete or continuous variable.

Genes are chosen from the design drivers table (Tab.3.3). Some of the design drivers are selected as genes (Tab.4.1), so that each individual represents a combination of values extracted randomly from the ranges, and others are used as bonus-malus conditions (see 4.3.3). The initial parameters influence both the convergence and the rate of convergence of the algorithm.

These initial ranges can be refined in order to reduce the simulation time, after multiple trial simulations. Since a GA evaluates a large number of combination, it is crucial to provide meaningful design ranges to avoid local maxima and not to waste time and resources.

Gene	**Values**
Tank shape	Spherical | Cylindrical
Tank Material	Aluminium | Stainless Steel
Gas Mass	0-6 Kg
Propellant	Nitrogen|Helium|Argon|Xenon|Methane
Tank Temperature	283-313 K
Tank Pressure	100-300 Bar
Trust chamber pressure	2-6 Bar
Expansion Ratio	2-8
Nozzle exit radius	10-20 mm

Table 4.1: Genes of the GA

Some guidelines are listed here:

- In some cases, for example the tank temperature, only integers are eligible as a gene. This means that each design string has an integer, randomly generated inside the range, as a value for the tank temperature
- The tank pressure range has been restricted to 250-300 Bar because every trial simulation showed no solution in the 100-250 Bar range
- It is possible to fix some parameters to get the desired solution. For example, it is possible to fix Helium as the propellant and the solution will shape accordingly.

Tab.4.1 defines the design space in which the GA has to work.

4.3 Fitness Evaluation

The FF is a subroutine of the GA that, for each individual of the population, has to estimate the performance of that particular design string and has to rank the individuals based on their score. This subroutine is divided into three sections, called *Tank Properties*, *Thuster Performance* and *Fitness Level*.

The first part calculates the properties of the tank, including its overall mass, dimensions and the mass of the propellant. The second part evaluates the thruster performance, in particular focussing on the aerospike nozzle properties and geometry. The last part include the evaluation of the fitness score, namely the fitness level of each design string. This needs the definition of function, and the related coefficient, that calculates the fitness level.

4.3.1 Tank Properties

The *Tank Properties* section takes as input some of the design drivers, such as the tank shape, the tank material, the gas mass, the propellant, the tank temperature and the tank pressure, marked in red in Fig.4.4. The tank volume has been calculated using the ideal gas law (see 2.2.3), with the compressibility factor Z fixed at 1. The geometrical properties of the tank derive from its volume and its shape. A fixed ratio of was used between the height and the radius of the cylindrical tank, since most commercial tanks have similar ratios ($h = 5 \div 7 \cdot r$).

The calculation of the thickness of the vessel is based on the model shown in paragraph 2.2.2.

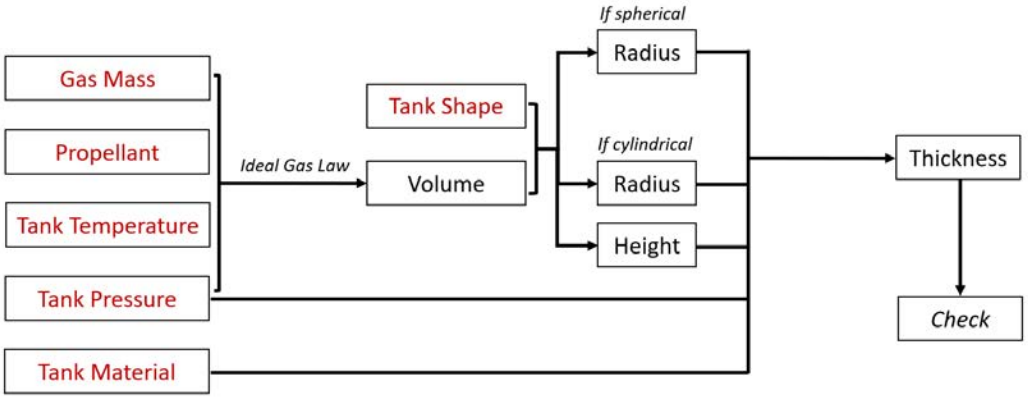


Figure 4.4: Schematic of the *Tank Properties* section

A mass check has been implemented at the end of the subroutine. Since the input parameters, marked in red in Fig.4.4, are random values in the design space, unrealistic solutions are very frequent. In this case, an unrealistic solution is stated as a solution in which the mass of the tank, calculated from the thickness and the shape parameters (namely radius and height), plus the propellant mass is higher than the available mass. The available mass is what was called maximum total mass in Tab.3.3, and is different for the two configurations (i.e.OLC and DLC). A design string that produces an unrealistic design gets an high malus that greatly influences its fitness level, as it is described in paragraph 4.3.3.

4.3.2 Thruster Performance

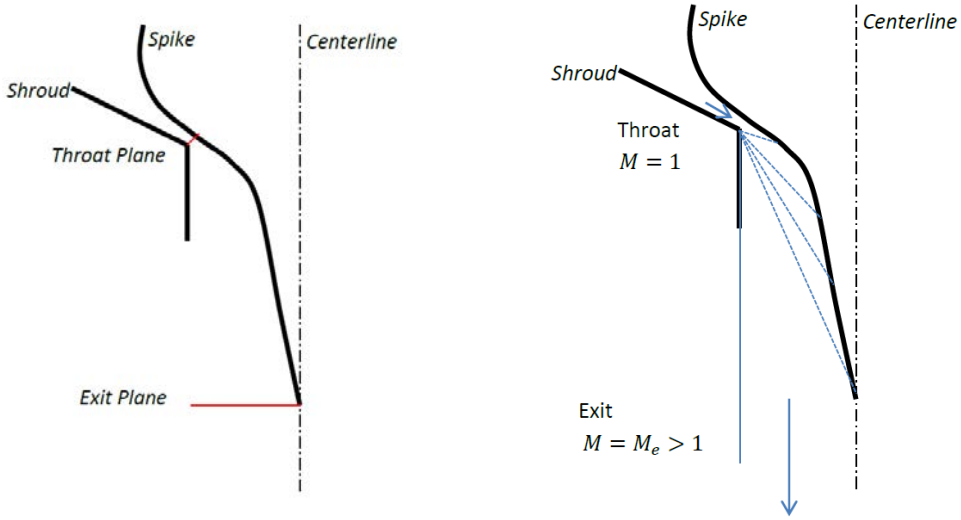
The evaluation of the thruster performance is based on MIT RocketTeam's Spike Contour Algorithm[35]. This algorithm is used for designing the contour of the spike of an aerospike nozzle. The algorithm also evaluates the thermodynamic properties of the working fluid along the spike, and the performance of the nozzle.

Model summary

The algorithm analyses the flow through the supersonic section of the aerospike nozzle. The flow, along the spike, accelerates and expands through the expansion fan and the spike contour is found by posing the constancy of the mass flow. The underlying assumptions of the thermodynamic model are that the expansion is adiabatic and isentropic, and the ratio of specific heat γ and the molar mass of the working fluid are constant through the aerospike nozzle. The generated spike contour is cylindrically symmetric and this algorithm doesn't include the concept of truncation. Naturally, the errors introduced by these simplification are considered adequate to the early state of the design. The approach of the model is based on a paper written by C.C. Lee for NASA's Marshal Space Flight Center [36].

Algorithm details

Though the inner contour consists of the spike, the outer solid contour is formed by the shroud. As it possible to see in Fig.4.5a, the throat is formed by the corner of the shroud, namely the lip and the spike. The exit plane is perpendicular to the centreline and includes the tip of the spike.



(a) (b)

Figure 4.5: Model characteristics and expansion fan [36]

As depicted in Fig.4.5b, the flow is expanded in the supersonic part of the nozzle by a Prandtl-Meyer expansion fan, represented in blue dashed line, that emanates form the corner of the nozzle shroud. The input of the algorithm and the related value are listed in Tab.4.2.

Input	**Description**	**Value**
p_c	Combustion chamber pressure	Design string gene
T_c*	Combustion chamber temperature	Design string gene
p_a	Ambient pressure	Vacuum
$A_e/A_t = \varepsilon$	Expansion ratio	Design string gene
N	Number of points of the mesh along the spike	100
γ	Ratio of specific heats	1.4
m_{molar}	Molar mass of the working fluid	Design string gene

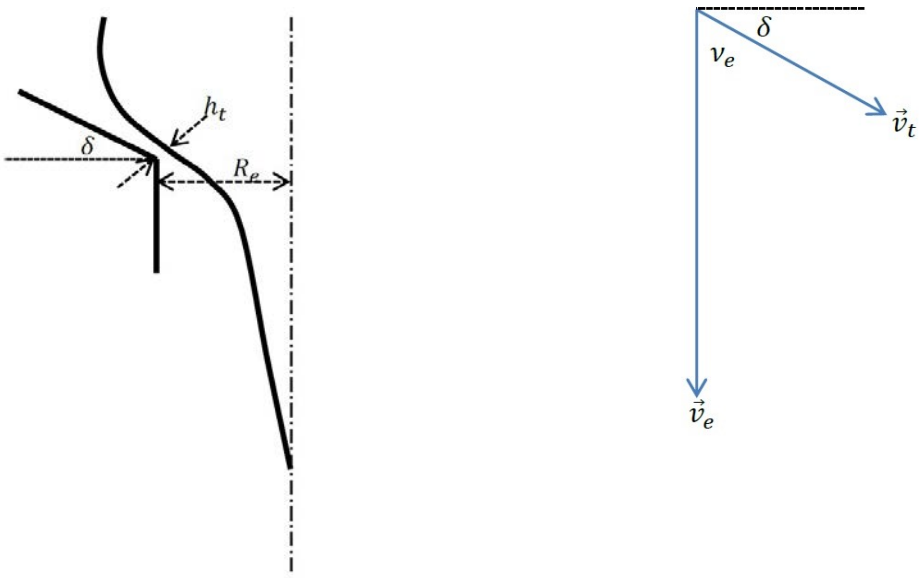
Table 4.2: Required Input for the algorithm
*:T_c is not directly known. For simplicity, has been imposed $T_s \approx T_c$, where T_s is the tank temperature

The exit Mach number M_e can be found using the inversion of Stodola's Aera-Mach Equation, described in [37]. This method takes as an input the expansion ratio ε, which is a gene. The exit pressure can be found using the isentropic expansion equation (4.1), knowing ε and M_e.

$$p_e = p_c \cdot \left(1 + \frac{\gamma - 1}{2} \cdot M_e^2\right)^{\frac{-\gamma}{\gamma - 1}} \tag{4.1}$$

Before the throat the velocity of the flow $\vec{v_t}$ is parallel to the shroud surface, inclined by δ in respect to the horizontal. The expansion fan accelerates the flow from $M = 1$, at the throat, to the exit Mach M_e and rotates the velocity vector by an angle equal to $\theta = 90° - \delta$, so that the exit velocity $\vec{v_e}$ is parallel to the centerline of the nozzle.

Fig.4.6a represents the nozzle geometry, highlighting the angle δ, the throat width h_t and the exit radius R_e. Fig.4.6b shows the rotation of the velocity vector, from $\vec{v_t}$ to $\vec{v_e}$, through the expansion fan.



(a) Nozzle geometry [36] (b) Velocity vector rotation [36]

Figure 4.6

The turning angle ν_e can be found via the Prandtl-Meyer relation:

$$\nu_e = \sqrt{\frac{\gamma+1}{\gamma-1}} \cdot atan\left(\sqrt{\frac{\gamma-1}{\gamma+1} \cdot (M_e^2-1)}\right) - atan\left(\sqrt{M_e^2-1}\right) \tag{4.2}$$

and according to what was stated,

$$\nu_e + \delta = \frac{\pi}{2} \tag{4.3}$$

and so,

$$\delta = \frac{\pi}{2} - \nu_e \tag{4.4}$$

The ratio between the throat width and the exit radius, which is a design string element, is given by Eqn.4.5

$$\frac{h_t}{R_e} = \frac{\varepsilon - \sqrt{\varepsilon \cdot (\varepsilon - sin(\delta))}}{\varepsilon \cdot sin(\delta)} \tag{4.5}$$

As explained in paragraph 4.3.3, a limitation on the throat width was implemented in the GA. Due to expertise on the realization on small ceramic objects, the minimum possible nozzle throat width h_t is limited to 1 mm.

The flow, passing through the expansion fan, expands through an infinite number of Mach lines. Every Mach line starts from the corner of the shroud and intersects the spike contour. N points are considered in the algorithm, which consist of N intersections on the spike contour and so, N Mach lines.

Every Mach line is correlated to a Mach number M_x, and is inclined of a particular angle $\phi_x = \nu_e - \nu_x + \mu_x$ (see Fig.4.7), where ν_e is the Prandtl-Meyer angle at the exit Mach number (Eqn.4.2), ν_x is the Prandtl-meyer angle at the current Mach number

$$\nu_x = \sqrt{\frac{\gamma+1}{\gamma-1}} \cdot atan\left(\sqrt{\frac{\gamma-1}{\gamma+1} \cdot (M_x^2-1)}\right) - atan\left(\sqrt{M_x^2-1}\right) \tag{4.6}$$

and μ_x is the Mach angle at the current Mach number M_x (Eqn.4.7).

$$\mu_x = asin\frac{1}{M_x} \tag{4.7}$$

Since the expansion fan is cylindrically symmetrical about the centreline of the nozzle, each Mach line can be revolved about the centreline forming a frustum.

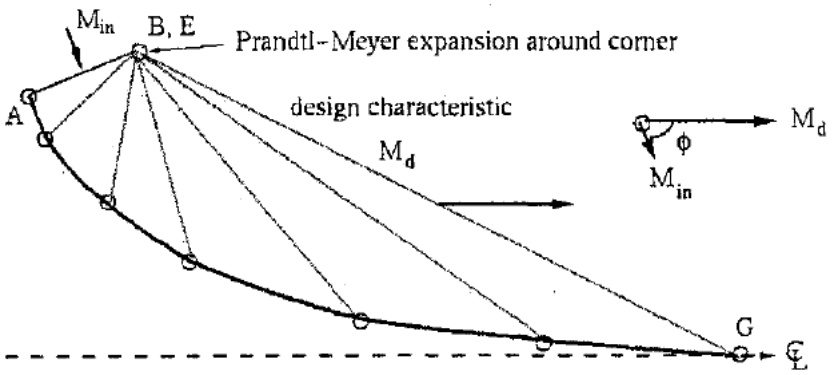


Figure 4.7: Plug contour nodes and Mach lines [38]

The circular inferior base of the frustum has a radius R_e, and on the other hand the top base is where each Mach line intersects the plug contour, at a radius R_x. The side of the frustum makes an angle ϕ_x with the centreline and the flow passes through the frustum at a Mach number M_x.

From the continuity, the mass flow through the throat $\dot{m}_t$ equals the mass flow through each frustum, $\dot{m}_x$. So,

$$\dot{m}_t = \dot{m}_x \tag{4.8}$$

$$\rho_t v_t A_t = \rho_x v_x A_x \tag{4.9}$$

and

$$A_x = \frac{\frac{\rho_t}{\rho_c} \cdot A_t}{\frac{\rho_x}{\rho_c}\frac{\nu_x}{\nu_t} \cdot sin(\mu_x)} \tag{4.10}$$

where ρ_c refers to the density of the fluid in the combustion chamber.

Eqn. 4.11 is used to calculate the side area of the frustum.

$$A_x = \frac{\pi \cdot (R_e^2 - R_x^2)}{sin(\phi_x)} \tag{4.11}$$

By equating Eqn.4.10 and Eqn.4.11, the ratio $\frac{R_x}{R_e}$ can be found as a function of M_x, as shown in Eqn.4.12. The complete derivation can be found in [37].

$$\frac{R_x}{R_e} = \sqrt{1 - \frac{\left[(\frac{2}{\gamma+1} \cdot (1 + \frac{\gamma-1}{2}) \cdot M_x^2)\right]^{\frac{\gamma+1}{2(\gamma-1)}} sin(\nu_e - \nu_x + \mu_x)}{\varepsilon}} \tag{4.12}$$

The height of the frustum X_x, defined as the axial distance from the shroud lip to the intersection between the Mach line and the spike contour, can be found as:

$$X_x = \frac{R_e - R_x}{tan(\phi_x)} \tag{4.13}$$

By computing N Mach lines in the range $[1, M_e)$, N points are found, each one defined by R_x and X_x, and so the spike contour can be drawn.

This model can also calculate the specific impulse. The thrust force produced by the engine, integrating the spike pressure over the spike area, is:

$$F_x = \dot{m}_t v_t sin(\delta) + (p_t - p_a) A_t sin(\delta) + \int_0^{A_x} (p_x - p_a)\, dA \tag{4.14}$$

Eqn.4.14 can also be used to estimate the Isp loss in case of truncation at a certain axial distance X_x. Dividing by the mass flow, it is possible to explicit the Isp,

$$Isp_x = v_t sin(\delta) + \frac{(p_t - p_a) A_t sin(\delta)}{\rho_t v_t A_t} + \int_0^{A_x} \frac{(p_x - p_a)}{\rho_t v_t A_t} dA \tag{4.15}$$

and after some steps[37],

$$Isp_x = v_t sin(\delta) \left[1 + \frac{1}{\gamma}\left(1 - \left(\frac{\gamma+1}{2}^{\frac{\gamma}{\gamma-1}} \left(\frac{p_a}{p_c}\right)\right)\right)\right] + \frac{v_t}{\gamma}\left(\frac{\gamma+1}{2}\right)^{\frac{\gamma}{\gamma-1}} \left(\frac{\varepsilon}{2}\right) \cdot \sum_{x=1}^{N} \left[\left(\frac{p_{x-1} - p_a}{p_c}\right) + \left(\frac{p_x - p_a}{p_c}\right)\right] \left[\left(\frac{R_{x-1}}{R_e}\right)^2 - \left(\frac{R_x}{R_e}\right)^2\right] \tag{4.16}$$

This model allows the calculation of the most important parameters of an aerospike nozzle, including its pressure profile, geometry, thrust and Isp.

4.3.3 Fitness Level

The calculation of the Fitness Level is the last part of the FF subroutine. The purpose of this section is to evaluate and rank the individuals of the population based on the results obtained by *Tank Properties* and *Thruster Performance* sections.

Usually, for a GA, the principles that rule the ranking system are the creator's footprint. The ranking system is the core of the algorithm, it is the section that decides what solution will be rewarded and what solution will be deleted.

The underlying idea of this ranking system is based on the objective of the experiment, that is, summing up, testing a cold gas aerospike engine (see Chapter 3). Based on that, the design principles of the Fitness Level subroutine are:

- The produced force should be measurable. Given that, it's not important the entity of the produced thrust, since a particular target is not required
- In order to collect a meaningful amount of data, the thruster should operate for the longest possible time
- The subroutine shall follow the design driver table (Tab.3.3). Each design string that features a value that exceeds the ranges, shall be penalized
- The subroutine shall that in account the cost diversity between different design drivers. Since it is mainly related to the choice of the propellant, from an this algorithm point of view, the subroutine shall take into account the cost of the propellant

These guidelines led to following FF (Eqn.4.17), where FL refers to the Fitness Level (i.e. the total score of the design string), α_i are the coefficients that multiply the FL_i variables.

Tab.4.3 sum up the meaning of the subscripts.

$$FL = \alpha_m * FL_m + \alpha_{TI} * FL_{TI} + \alpha_{NT} * FL_{NT} + \alpha_{cost} * FL_{cost} + \alpha_{ER} * FL_{ER} + \alpha_{\dot{m}} * FL_{\dot{m}} \quad (4.17)$$

The flat negative variables involving mass and nozzle throat (i.e. FL_m^- and FL_{NT}) represent constraints. Usually, in GA, constraints are included as high negative values that limit the reproduction possibility to the solutions that don't respect those constraints.

Flat Variables		
Mass	FL_m^-	If the system mass, obtained by the combination of the design string elements, is higher than the available mass, this design string scores a negative FL_m. The exact amount is calculated in order to delete that solution, even in case every other FL score is high
Nozzle Throat	FL_{NT}	Design strings that lead to a nozzle throat width lower than 1 mm get a penalty. The exact amount is calculated in order to delete that solution, even in case every other FL score is high
Cost	FL_{cost}	The higher the cost, the higher the penalty, but two design strings with the same propellant get the same penalty
Proportional Variables		
Mass	FL_m^+	If the system mass, obtained by the combination of the design string elements, is lower than the available mass, this design string scores a positive FL_m. In this case, the lower is the difference, i.e. the closer are the two values, the higher is the score
Total Impulse	FL_{TI}	The higher the total impulse I_{tot}, the higher the bonus
Expansion Ratio	FL_{ER}	The higher the expansion ratio, the higher the bonus
Mass Flow	$FL_{\dot{m}}$	The lower the mass flow, the higher the bonus

Table 4.3: Description of the FF variables

The total impulse is the only parameter requested by the client of the research and it represents an important parameter for various reasons. The total impulse I_t is the thrust force F integrated over the burning time t.

$$I_{tot} = \int_0^t F dt \tag{4.18}$$

Equation 4.18 shows that I_{tot} can be easily correlated with experimental data. During a static bench test, it is possible to measure the thrust and the burning time, plot the data and the total impulse corresponds to the area under the curve.

Since the total impulse is not dependant on the combustion parameters, the amount of thrust produced nor the type of propellant burned, it is used as a useful indicator to compare different rocket motors [39]. The GA compares drastically different solutions, so the total impulse has been found as a good comparing parameter, taking into account the early stage of the research. Moreover, according to the guidelines laid down at the beginning of this subsection, I_{tot} gives information about the thrust, the burning time and their relationship.

The condition on the I_{tot} doesn't specify the ratio between F and Δt, so multiple solutions with the same I_{tot} are possible. In order to have only one optimal solution, two conditions were implemented, one on the mass flow and one on the expansion ratio. The condition on the mass flow leads to the fact that solutions with low mass flow are rewarded with high bonuses. This means that those solutions, considering the same mass stowed, are characterized by a greater "burning" time. Moreover, configurations with high expansion ratio, are rewarded with high bonuses. As conventional bell nozzles, in vacuum environment the gains obtained from increasing the expansion ratio are limited. Moreover, for aerospike nozzles (with truncation) the increase of the expansion ratio leads to a slower increase of the volume compared to bell nozzle.

As it is possible to notice, the conditions imposed with this FF leads to a precise solution, the one with the highest allowed expansion ratio and lowest allowed throat area.

Instead of building a simple algorithm involving fixed expansion ratio and throat area (and so throat width and exit radius), it was decided to build a GA for future improvements and definition of the requirements. The early state of the research doesn't allow the definition of strict requirements, but the purpose of this work is also to build a decision-making tool. As shown in paragraph 5.2, the FF can be adjusted easily to generate a design string that matches a particular objective. In that case, the use of a genetic algorithm allowed the user to change the algorithm with few steps, instead of building another algorithm from scratch.

4.4 Termination Criterion, Crossover and Mutation

As an output of the Fitness Evaluation, the algorithm has built an ordered matrix of design strings. On top there are the best design strings, i.e. the design strings that produce an higher fitness level, and at the bottom the worst design strings.

The termination criterion is implemented here. The algorithm stores the best solution of each generation and after a minimum number of generations, if the best solution of the last X generations is the same, the cycle stops. In other words, set as Queen the best solution of a generation, if that design string remains Queen for X generations, the cycle breaks, that solution is marked as the optimal result of that inner cycle and that solution is passed to the outer cycle. In the algorithm, X is set as 15.

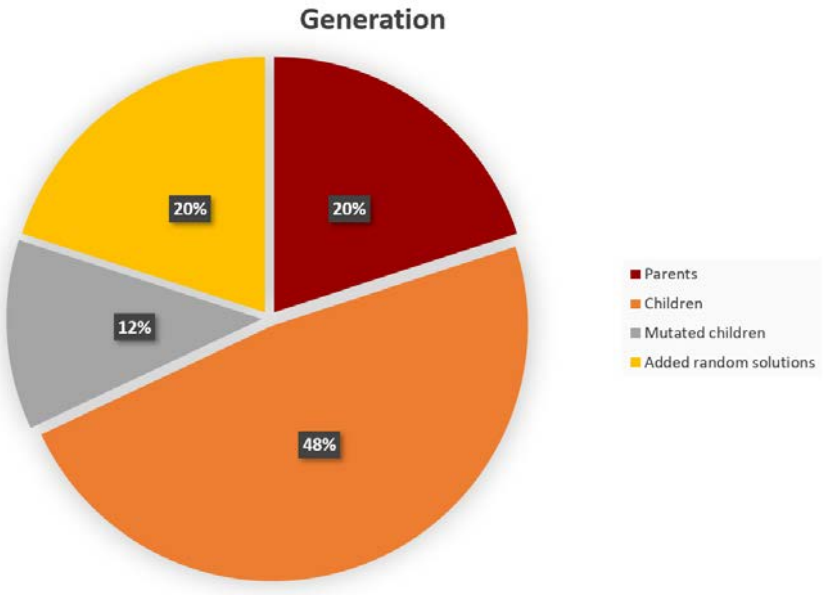


Figure 4.8: Composition of the following generation

If the termination criterion is not satisfied, the ordered matrix of design strings is used to create the mating pool, that consists of the top 20% of each generation. This group is named as *parents* and, via crossover and mutation, generates the subsequent generation. The parents matrix is randomly shuffled and then the design strings are mixed in pairs. A *child* gets half of the genes from parent1 and the other half from parent2. This process is called **crossover** and generates 60 % of the following generation.

Mutation takes place during the mating process. 20% of the children are characterized by a mutation that involves some of the core parameters of the design string, such as expansion ratio and tank temperature. Finally, the last 20% is composed by new design strings, randomly generated in the exact same process as the first generation.

These new random solutions and the mutation principle are two ways to avoid the algorithm to converge to local maxima. Fig.4.8 sum up the composition of the subsequent generation.

4.5 Outer cycle

It has been said that, when the termination criterion is satisfied, that solution, called *Queen* is passed to the outer cycle. Figure 4.3 shows the outer cycle, that consists of n inner cycles and after each inner cycle, the algorithm stores the corresponding Queen. At the moment, n is set as 10. Then, after collecting n different Queens, the algorithm compares them and prepares a new population for the last cycle.

Table 4.4 shows some elements of a typical collection of 10 Queens. As it is possible to see, some elements repeat themselves throughout the design strings, some elements are similar but few of them varies.

The algorithm prepares a new population following these principles:

- For identification numbers, the algorithm takes the most frequent number and sets that value as fixed for the new population
- In case of repeated or similar values, the algorithm sets that value (or the mean value) as fixed for the new population
- If there is a variability in a particular gene of the design strings, the algorithm will limit the design range using the minimum and maximum number found in the Queens collection. The restricted range is used in the last cycle to randomly produce elements of that design string.

Referring to Tab.4.4, the identification number that characterizes the propellant is always the same, so the algorithm sets this value as fixed. Every design string of the new population is going use nitrogen (0 means nitrogen) as its propellant.

Looking to the propellant mass, it is possible to see that there's a slight variability, negligible for the purposes of this work, so the algorithm uses the mean value.

The exit radius column has a more evident variability. The original design strings have been built from a wide range [10-20]mm, the subroutine looks at

the maximum and minimum value and set a new,limited range [15-20]mm, from which the new population will be derived.

The last cycle subroutine is identical to the one of the inner cycles, but the starting population has fixed parameters and shrunk ranges, so that it focuses the variable parameters.

Propellant	...	**Propellant Mass [kg]**	...	**Exit Radius[mm]**
0	...	2.99486	...	20
0	...	2.99599	...	17
0	...	2.92212	...	19
0	...	2.99599	...	17
0	...	2.98162	...	18
0	...	2.97228	...	15
...	...	...	...	...

Table 4.4: Example of some genes of the Queens collection

Chapter 5

Results and Conclusions

In this chapter some solutions of the algorithm are shown and discussed. Two solutions for each configuration (i.e. OL e DL configuration) are presented, the algorithm optimized solution and a target solution.

The algorithm optimized solution is the solution obtained using the FF explained in Chapter 4. This solution is used as a reference and a validation of the method, since, as is explained in paragraph 5.1, the results could be predicted analysing the FF. This solution also matches the research client requests.

Then, a target solution is presented in order to show the possibilities of the algorithm and what future improvements can lead to.

It is easily understandable that the solutions of a GA are in direct relationship with the FF and the initial design ranges. To modify one of those subject leads to a drastically different solution. The definitive FF should be implemented once the objective and the requirements of the experiment are fixed, but the value of a flexible algorithm (such as a GA) is that it can be used as a decision-making tool for finding the requirement and the initial parameters of the experiment. This argument is expanded in paragraph 5.2, where the algorithm is used to understand what the definition of a requirement means for the other parameters.

5.1 Optimized solution

The following solutions have been derived using the FF presented in paragraph 4.3.3 and, as already said, the imposed conditions lead uniquely to a solution characterized by the maximum expansion ratio and the minimum allowed throat area (i.e width). If the solution given from the GA matches the expected results, it can be said that the algorithm has proven its reliability.

OLC results

Table 5.1 shows the output of the algorithm for the OLC.

Propellant	Nitrogen
Tank Material	Aluminum
Tank Shape	Spherical
Thickness	5 mm
Radius	148 mm
Storage pressure	295 bar
Storage temperature	306 K
Gas mass	4.4 Kg
Full tank mass	8.1 kg
Chamber pressure	2 Bar
Exit radius	17 mm
Nozzle Throat	1 mm
Expansion Ratio	8
Mass flow	0.046 kg/s
Force	$\sim$ 35 N
Isp	$\sim$ 78 s
Exit mach number	3.3

Table 5.1: Optimized results of the OLC

As it possible to understand , the results agree with the expected output.

The optimisation based on the highest possible total impulse I_{tot} and the minimum mass flow leads to the lowest possible throat width and to the highest possible expansion ratio.

These considerations are correlated with the chamber temperature. As an initial simplification, the chamber temperature is equal to the storage temperature. From Eqn.5.1, representing the mass flow of a chocked bell nozzle, it is possible to understand that an increase of the chamber temperature T_c leads to a decrease of the mass flow. On the other hand, an increase of the storage temperature leads to an increase in the volume of the tank (i.e. mass) or a decrease in the storable mass (keeping the storage pressure constant). The algorithm has to find a balance between these considerations.

$$\frac{\dot{m}}{A_t} = \sqrt{\frac{k}{R}} \cdot \frac{p_0}{T_0} \cdot \left(\frac{2}{k+1}\right)^{\frac{k+1}{2(k-1)}} \tag{5.1}$$

Nitrogen as a propellant was requested by the research client. Thus a GA is capable of taking constant parameters as an input, it was not necessary, because the FF selected nitrogen over the other available propellants. In this case, the optimized solution matches the requested output. The first rows of Tab.5.1 give an estimation of the tank properties, as mass, volume and geometry. As stated in chapter 3.3.3 these results can be used as a reference to find suitable COTS, since it was not possible to simulate commercial composite tanks, that represent the state of the art for pressure vessels.

The implemented Bonus-Malus system works and no solution possesses a mass higher than the available one, matching the mass requirement D.1 (3.2.3), or a throat width lower than imposed one. As already said, the chamber pressure was fixed to a common value and other analyses are needed to estimate the influence of the pressure on a ceramic, 3D printed structure. The produced force, as requirement D.3 (3.2.3) states, is measurable and with the assumption of a satellite mass lower than 500 kg, matches the acceleration requirement P.1.1 (3.2.2). The algorithm also calculates the exit properties of the gas, such as exit pressure, exit temperature and exit Mach number. As a first estimation, considering the mass flow constant, the thruster should operate for $\sim$ 100 s, matching the requirement P.1.3 (3.2.2).

DLC results

Table 5.2 shows the results for the DLC. As expected, the tank properties are different, due to the minor available mass. The performance outputs can be considered equivalent and this solution matches the requirements too. The small differences between some OLC and DLC outputs, such as the 1 N difference in thrust produced, are considered negligible.

Propellant	Nitrogen
Tank Material	Aluminum
Tank Shape	Spherical
Thickness	4.2 mm
Radius	127 mm
Storage pressure	295 bar
Storage temperature	305 K
Gas mass	2.8 Kg
Full tank mass	5.1 kg
Chamber pressure	2 Bar
Exit radius	17 mm
Nozzle Throat	1 mm
Expansion Ratio	~ 8
Mass flow	0.044 kg/s
Force	~ 34 N
Isp	~ 78 s
Exit mach number	3.3

Table 5.2: Optimized resultsof the DLC

5.2 Target solution

The next solution is presented as an example of the GA capabilities. As already said, the final version of the FF should be implemented once the requirements and constraints of the experiment are fully defined. This solution shows how easy is to implement new requirements in a GA.

The target solution has to fulfil the following requests:

- DL configuration
- Cylindrical vessel
- Xenon as propellant
- A thrust target, $18N \leq F_{obj} \leq 19N$

The first condition can be implemented by commenting out the proper line:

```
#maximum_tank_mass = 8.1065 # OL configuration
maximum_tank_mass = 5.1295 # DL configuration
```

The second and the third conditions can be imposed with the following lines:

```
# ID of the tank shape, spherical=0, cylindrical=1
tank_min=1
tank_max=2
# ID of the propellant
# 0=Gaseos nitrogen, 1=Helium, 2=Argon,
# 3=Xenon, 4=Methane
prop_min = 3
prop_max = 4
# Propellant
population[0:,0] = numpy.random.randint(low=prop_min,
high=prop_max, size=size)
# Tank shape
population[0:,1] = numpy.random.randint(low=tank_min,
high=tank_max, size=size)
```

In Python, the Numpy function numpy.random.randint(a,b,n) select n random integer in range [a,b). Conditions can be imposed by appropriately changing the initial intervals.

For example, an initial interval [`prop_min`,`prop_max`) means that every solution will use `prop_min` (in this case `prop_min` =3,i.e. Xenon). The same consideration can be applied for the selection of the tank shape.

The thrust target represents a completely new condition. This can be implemented by adding few lines to the FF. First of all, the chamber pressure value was unfixed and an initial range has been set as [1,3] bar. Then a new condition, penalizing all solutions producing a force out of the requested range, was added to the FF as follows, where `F[i]` represents the thrust produced by the i-th solution and `FL_F[i]` the respective (negative) fitness level entry.

```
for i in range (0,sol_per_pop):
       if F[i] < 18 or F[i] > 19:
           FL_F[i]= -6e10
```

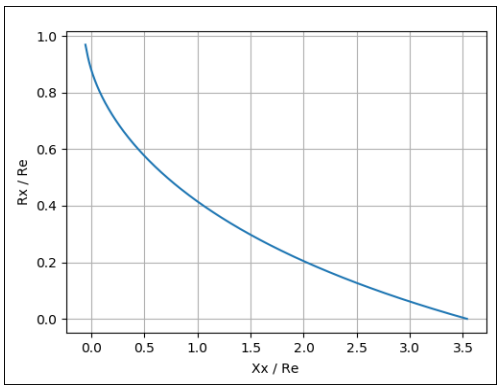
This condition is sufficient to exclude every solution that doesn't match the request and to make the algorithm adjust the initial parameters in order to fulfil the request. The target solution is presented above(Table 5.3).

Propellant	Xenon	**Tank Material**	Stainless steel
Tank Shape	Cylindrical	**Radius**	40 mm
Height	201 mm	**Radial thickness**	2.5 mm
Axial thickness	2 mm	**Storage pressure**	274 bar
Storage temperature	303 K	**Gas mass**	3.9 Kg
Full tank mass	5 kg	**Chamber pressure**	1.09 Bar
Exit radius	17 mm	**Nozzle Throat**	1 mm
Expansion Ratio	~ 8	**Mass flow**	0.055 kg/s
Thrust	~ 18.9 N	**Isp**	~ 36 s
Exit Mach number	3.3		

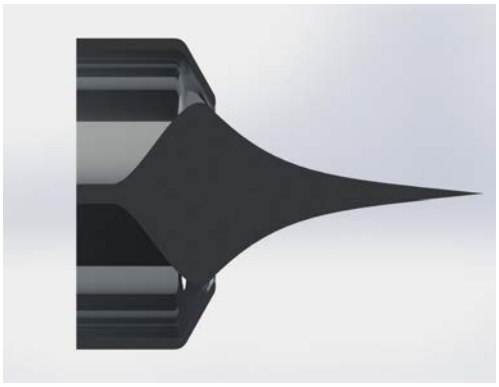
Table 5.3: Target solution

5.3 Spike contour

The MIT RocketTeam's Spike Contour Algorithm[35] integrated into the GA is capable of plotting the spike contour. In Fig.5.1a, the spike contour of the OLC, target solution is presented, where the x-axis represents the distance from the shroud tip and the y-axis represents the relative radius. The axis are normalized by the external radius Re. Negative values on the x-axis are possible because the spike contour is plotted from the throat area which, as can be seen from Fig.4.5a, is located before the vertical line passing through the shroud tip, which represents the zero of the x-axis. These information can be easily imported into a CAD program, like Solidworks®, in order to design a 3D model of the nozzle(Fig.5.1b). Note that, while the spike contour has been designed using the tool, the inner part of the nozzle is just representative of a possible shape of the nozzle and it is inspired by existing aerospike nozzles. Fig.5.1b clearly shows that the final portion of the spike would be too complex to manufacture and the result would be very fragile. In other words, truncation is always needed in order to have a feasible and robust design.



(a) Spike contour of the OL target solution



(b) Spike contour render of the OL target solution

Figure 5.1

5.4 Conclusions

In the framework of the project MACARONIS, a preliminary study of a cold gas aerospike engine was performed. A research was conducted on the current state of the art cold gas thrusters, critical components were discussed and two raw systems were designed. An optimization tool was developed, its functionality as a decision-making tool validated and three solutions presented. This work represents a good starting point for several improvements. The objectives, requirements and constraints listed in Chapter 3 are just a first proposal and several discussions between future team members are needed in order to complete the chapter. Once the requirements are fully defined, the final version of the FF should be implemented (as stated in 4.3.3 and 5.2) in order to obtain the desired solution. In section 5.2, the GA proved to be a versatile tool, easy to adjust for the MACARONIS team. The implementation of some structural calculation evaluating the influence of the operating pressure on the thrust chamber clearly represents a possible improvement of the algorithm, as well as the introduction of more complex tanks such as composite or multi-layer vessels.

The author takes this opportunity to thank Professor Daniele Pavarin for the help and patience shown during the period abroad. A special thanks goes to the supervisor of this work, Dr.-Ing. Christian Bach, and to his colleagues Dipl.-Ing. Martin Propst and Dipl.-Ing. Jan Sieder-Katzmann, for the support, professionality and friendliness.

A thought goes to my family, who supported me during these years of study, and to all friends.

Appendix A

Code

In this appendix the GA code is presented. To avoid repetition, the following code concerns the optimized solution, while section 5.2 shows how to modify the code to get the target solution.

A.1 Main

```
"""
@author: Giorgio Tesser
mail: giorgio.tesser@studenti.unipd.it
"""
# GENETIC ALGORITHM
#------------------------------------------------------

# Run Main.py

#------------------------------------------------------

# List of libraries
import numpy
import math
from scipy.optimize import fsolve
from collections import namedtuple
from collections import Counter
import timeit

# Import other scripts
import Data
```

```
import Fitness
import GA
import aero_solver
import re_run

# Timer; calculates the simulation time
start = timeit.default_timer()
# --------------
# INITIALIZATION
# -------------

"""Number of variables
Propellant, tank shape, pressure of the tank, mass of the gas
   , temperature of the
tank, chamber pressure, exit radius, expansion ratio,
   material of the tank"""

# The maximum mass of the experiment consists of the
   available mass for the
# tank and the propellant
# There are two possibilities, based on the mass budget (
   Chapter 3)
# One line configuration -----> maximum_tank_mass = 8.1065 Kg
# Double line configuration -----> maximum_tank_mass = 5.1295
    Kg

# Select the desidered configuration by commenting the other
   one
# maximum_tank_mass = 8.1065

maximum_tank_mass = 5.1295
num_genes=9 # The design string consist in 9 genes
sol_per_pop= 50 # Number of random solution for each
   generation

# Initial parameters
# It is possible to change the following parameters in order
   to change the
# design space of the algorithm

chamber_pressure_min=2 # Operating pressure
```

```
chamber_pressure_max=3
Re_min=15 # Minimum Exit Radius
Re_max=20 # Maximum Exit Radius
er_min=6 # Minimum Expansion Ratio
er_max=8 # Maximum Expansio Ratio
ps_min=295 # Minimum Storage Pressure
ps_max=296 # Maximum Storage Pressure
T_min=283
T_max=310

# ID of the tank shape, spherical=0, cylindrical=1

tank_min=0
tank_max=2

# ID of the propellant
# 0=Gaseos nitrogen, 1=Helium, 2=Argon, 3=Xenon, 4=Methane
prop_min = 0
prop_max = 5
# Minimum and maximum gass mass. To improve velocity of the
   algorihm put
# feasible numbers, in order to avoid the calculation of
   irrealistic solutions.
# (i.e. gas_mass_max = 6 when maximum_tank_mass = 5.1295)

# Check the desidered configuration (OL or DL), read the
   maximum_tank_mass and
# then check these two values accordingly
# My experience: 2.5-5 for OL, 1.5-3 for DL

gas_mass_min=1
gas_mass_max=4

# Coeffcient for the fitness function. Change the values to
   adjust ratio
# between the coefficient of the fitness function
# See chapter 4 for the explanation of the Bonus/Malus system


coeff_FL_epsilon = 2 # Mass Ratio coefficient
coeff_FL_it = 5 # Total impulse coefficient
```

```
coeff_FL_ht = 1 # Nozzle Throat coefficient
coeff_FL_cost = 100 # Cost coefficient
coeff_FL_er = 1 # Expansion Ratio coefficient
coeff_FL_m_dot = 2 # Mass flow coefficient

N = 10 # Number of Queens
pop_size = (sol_per_pop,num_genes) # Defining the population
    size
sol_size = (1,num_genes) # Defining the solution size
sol = numpy.zeros(sol_size) # Initializing the solution vector
# Initializing an empty vector. It is necessary at the end of
     the loop to store
# N Queens (i.e N solution of the inner cycle, see Chapter 4)
    , in one matrix
sol_add = numpy.zeros(sol_size)

#List of propellant: 0=Gaseos nitrogen, 1=Helium, 2=Argon, 3=
    Xenon, 4=Methane

for it in range (0,N): # CREATION OF THE INITIAL POPULATION
    new_population = Data.population(pop_size, prop_min,
        prop_max,tank_min,tank_max,ps_min,
              ps_max,gas_mass_min,gas_mass_max,T_min,T_max,
                 chamber_pressure_min,
              chamber_pressure_max, Re_min, Re_max, er_min,
                 er_max)

    # The following lines are the initialization of the
        termination criterion, when the last n best solution
        are the same, toll = 0
    toll = 2
    generation=0
    condition1=True
    Best_solution_size = (1,num_genes)
    Queens = numpy.zeros(Best_solution_size)
    Queens_add = numpy.zeros(Best_solution_size)
    while 1:
        if toll==0:
            condition1=False
        if condition1==True:
```

```
zn = Data.parameters (new_population,sol_per_pop) #
    Matrix contaiing gas parameters R,M,Z
V = Fitness.tank (new_population,zn) # Volume of the
     tank
t = Fitness.tank_design(V,sol_per_pop,new_population
    ,zn) # Include both width and radii
empty_tank_mass = Fitness.empty_tank_mass (t,
    sol_per_pop,new_population,zn) # Empty tank mass
gas_size = (sol_per_pop,1)
gas_mass=numpy.zeros(gas_size)
gas_mass [0:,0] = new_population[0:,3] # Gas mass
full_tank_mass = empty_tank_mass + gas_mass # Full
    tank mass
m_diff= maximum_tank_mass - full_tank_mass

# Initialization of the vectors for : performance
    evaluation and creation of the new generation

parents_num = int(0.2*sol_per_pop)
random_solutions = int(0.2*sol_per_pop) # 20% random
     solutions
random_sol_shape = (random_solutions,num_genes)
performance_size=[sol_per_pop,16]
performance_matrix = numpy.zeros(shape=
    performance_size)
EngineParameters = namedtuple('EngineParameters', '
    Tc␣Pc␣molar_m␣gamma␣Re␣er␣Ps')
for j in range (0,sol_per_pop):
    T = new_population[j,4]
    p_chamber = new_population[j,5]
    p_storage = new_population[j,2]
    Re = new_population[j,6]
    er = new_population[j,7]
    molar_mass = gas_mass[j]*1000/zn[j,1]
    params = EngineParameters(Tc=T, Pc=p_chamber,
        molar_m=molar_mass, gamma=1.4, Re=Re, er=er,
        Ps= p_storage)
    performance = aero_solver.aerosolver(params)
    # The namedtuple values will be inserted in a
        Performance Matrix.
```

```
        # Note that performance returns 'delta Re ht At
            er Pc Tc m_dot F Isp Pe Me Cf'"
        performance_matrix[j,0] = performance.delta
        performance_matrix[j,1] = performance.Re
        performance_matrix[j,2] = performance.ht
        performance_matrix[j,3] = performance.At
        performance_matrix[j,4] = performance.er
        performance_matrix[j,5] = performance.Pc
        performance_matrix[j,6] = performance.Tc
        performance_matrix[j,7] = performance.m_dot
        performance_matrix[j,8] = performance.F
        performance_matrix[j,9] = performance.Isp[-1]
        performance_matrix[j,10] = performance.Pe
        performance_matrix[j,11] = performance.Te
        performance_matrix[j,12] = performance.Me
        performance_matrix[j,13] = performance.Cf
        performance_matrix[j,14] = performance.Ps
        performance_matrix[j,15] = full_tank_mass[j]
    # Calculation of the fitness level of the current
        generation
    FL= Fitness.FL_calc(new_population,m_diff,
        sol_per_pop,parents_num,performance_matrix,
        coeff_FL_it,coeff_FL_cost,coeff_FL_er,
        coeff_FL_m_dot,coeff_FL_epsilon,coeff_FL_ht)
    # Sum of each entries of the FL
    fl= FL[:,0] + FL[:,1] + FL[:,2] + FL[:,3] + FL[:,4]
        + FL[:,5]
    old_pop=numpy.copy(new_population)
    # Finding the maximum fitness level of the current
        generation
    fl_max = numpy.where(fl == numpy.max(fl))
    fl_max_idx = fl_max[0][0]
    # Store the best solution of the current generation
         in the vector Queen
    Queens[generation,:] = old_pop[fl_max_idx,:]
    Queens = numpy.r_[Queens,Queens_add]
    # Selecting parents
    parents = GA.select_mating_pool(fl,sol_per_pop,
        parents_num,new_population,num_genes)
    numpy.random.shuffle(parents)
```

```
        offspring_size=[pop_size[0]-parents.shape[0]-
            random_solutions,num_genes]
        # Crossover
        offspring_crossover = GA.crossover(parents,
            offspring_size)
        # Mutation
        offspring_mutation = GA.mutation(
            chamber_pressure_min,chamber_pressure_max,
            offspring_crossover,offspring_size)
        # Creation of the new population: 20% parents, 20%
            random, 60% crossover (20% of these mutated)
        new_population[0:parents.shape[0], :] = parents
        new_population[parents.shape[0]:sol_per_pop-
            random_solutions, :] = offspring_mutation
        new_population[sol_per_pop-random_solutions:,:] =
            Data.population (random_sol_shape,prop_min,
            prop_max,tank_min,tank_max,ps_min,
           ps_max,gas_mass_min,gas_mass_max,T_min,T_max,
               chamber_pressure_min,
           chamber_pressure_max, Re_min, Re_max, er_min,
               er_max)
        numpy.random.shuffle(new_population)
        generation=generation+1
        #print(generation)

        # Termination criterion
        if generation>10:
            toll1=0
            for i in range (2,20):
                    toll_v = (Queens[Queens.shape[0]-i,:]-
                        Queens[Queens.shape[0]-(i+1),:])
                    toll1=toll1+numpy.sum(toll_v)
            toll=toll1
    else:
        break
# Results of the inner cycle

# Finding the highest fitness level
fl_max = numpy.where(fl == numpy.max(fl))
fl_max_idx = fl_max[0][0]
# and the best solution of the inner cycle
```

```
    Best_solution = old_pop[fl_max_idx]
    Best_performance = performance_matrix[fl_max_idx,:]
    Results = namedtuple('Results', 'Idx␣fl␣FL␣Propellant_Type␣
       Tank_Type␣Tank_material␣Tank_Dimensions␣Storage_Pressure
       ␣gass_mass␣full_tank_mass␣Tank_temperature␣
       Chamber_Pressure␣delta␣Re␣ht␣At␣er␣Pc␣Tc␣m_dot␣F␣Isp␣Pe␣
       Te␣Me␣Cf␣Ps')
    Final_results = Results(Idx=fl_max_idx, fl=fl[fl_max_idx],
       FL=FL[fl_max_idx], Propellant_Type=Best_solution[0],
       Tank_Type = Best_solution [1],Tank_material =
       Best_solution[8], Tank_Dimensions = t[fl_max_idx,:],
       Storage_Pressure = Best_solution[2],
                          gass_mass = Best_solution[3],
                             full_tank_mass = Best_performance
                             [15], Tank_temperature =
                             Best_solution[4], Chamber_Pressure
                              = Best_solution[5],
                          delta = Best_performance[0], Re =
                             Best_performance[1], ht =
                             Best_performance[2], At =
                             Best_performance[3], er =
                             Best_performance[4],
                          Pc = Best_performance[5],Tc =
                             Best_performance[6], m_dot =
                             Best_performance[7], F =
                             Best_performance[8], Isp =
                             Best_performance[9],
                          Pe = Best_performance[10], Te =
                             Best_performance[11], Me =
                             Best_performance[12], Cf =
                             Best_performance[13], Ps =
                             Best_performance[14])
    # storing the solution of the inner cycle in a vector
       called "sol"
    sol[it,:] = old_pop[fl_max_idx,:]
    sol = numpy.r_[sol,sol_add]



sol = numpy.delete(sol, (sol.shape[0]-1), axis=0)
```

```
# Evaluation of the variance of the term sol, use it to
   calibre the fitness function or the fitness function
   parameters
#for i in range (0,sol.shape[1]):
# mean_value = numpy.mean(sol[:,i])
# xi = sol[:,i] - mean_value
# variance[0,i] = (numpy.sum (numpy.power(xi,2)))/(sol.shape
   [0]-1)
# variance[1,i] = numpy.power(variance[0,i],0.5)

# Inizializing the last population
new_population = numpy.zeros (shape=pop_size)
# As stated in chapter 4, here the algorithm uses:
# - most frequent value for identification numbers
# - mean value for values with small variance
# - shrink the design range for values that vary a lot
most_freq_prop = Counter(sol[:,0])
propellant_type = most_freq_prop.most_common(1)[0][0]
most_freq_tank_material= Counter(sol[:,8])
tank_material = most_freq_tank_material.most_common(1)[0][0]
most_freq_er = Counter(sol[:,7])
err = most_freq_er.most_common(1)[0][0]
new_population[0:,0] = propellant_type # Propellant type
new_population[0:,1] = numpy.mean(sol[:,1]) # Tank type
new_population[0:,2] = numpy.mean(sol[:,2]) # Storage pressure
new_population[0:,3] = numpy.mean(sol[:,3]) # Gas_Mass
new_population[0:,4] = numpy.mean(sol[:,4]) # Tank temperature
new_population[0:,5] = numpy.mean(sol[:,5]) # Chamber pressure
Re_min_r = numpy.min (sol[:,6]) * 1000
Re_max_r = numpy.max (sol[:,6]) * 1000
new_population[0:,6] = numpy.random.randint(low=Re_min_r,high=
   Re_max_r, size=pop_size[0]) # Exit radius in m ( so from 10
    to 20 mm)
new_population[0:,6] = new_population[0:,6]/1000
er_min = numpy.min (sol[:,7])
er_max = numpy.max (sol[:,7])
#new_population[0:,7] = numpy.random.randint(low=er_min*10,
   high=er_max*10, size=pop_size[0]) # Expansion ratio
#new_population[0:,7] = new_population[0:,7]/10

new_population[0:,7] = err # Propellant type
```

```
new_population[0:,8] = tank_material # Tank material
# Final inner cycle using the new defined population
Optimized_results = re_run.rerun (new_population,sol_per_pop,
   maximum_tank_mass,coeff_FL_it,coeff_FL_cost,coeff_FL_er,
   coeff_FL_m_dot,num_genes,coeff_FL_epsilon,coeff_FL_ht)
stop = timeit.default_timer() # timer
print('Time: ', stop - start)
print(Optimized_results)
```

A.2 Data

```
"""
@author: Giorgio Tesser
mail: giorgio.tesser@studenti.unipd.it
"""
# List of libraries
import numpy

 # Calculation of some properties : propellants and materials
def parameters (new_population,sol_per_pop):
    zn_size = (sol_per_pop,4)

    zn = numpy.zeros (zn_size)
    gas_mass = new_population[0:,3]
    # Calculation of the property of the gasses
    # Compressibility factor fixed as 1, as an improvement it
       is possible to
    # implement Z = f (pressure, temperature)
    for i in range(0,sol_per_pop):
        if (new_population[i,0] == 0):
            # Gaseos Nitrogen
            zn[i,0]=1 # Z = compressibility
            molar_mass=28 ## g/mol
            zn[i,1]= gas_mass[i]*1000/molar_mass # n mol
        if (new_population[i,0] == 1):
            # Helium
            zn[i,0]=1 # Z = compressibility
            molar_mass= 4 # g/mol
            zn[i,1]= gas_mass[i]*1000/molar_mass # n mol
        if (new_population[i,0] == 2):
            # Argon
            zn[i,0]=1 # Z = compressibility
            molar_mass=39.9 # g/mol
            zn[i,1]= gas_mass[i]*1000/molar_mass # n mol
        if (new_population[i,0] == 3):
            # Xenon
            zn[i,0]=1 # Z = compressibility
            molar_mass=131.3 # g/mol
            zn[i,1]= gas_mass[i]*1000/molar_mass # n mol
```

```
        if (new_population[i,0] == 4):
            # Methane
            zn[i,0]=1 # Z = compressibility
            molar_mass=16 # g/mol
            zn[i,1]= gas_mass[i]*1000/molar_mass # n mol

    # Tank material properties
    for i in range(0,sol_per_pop):
        if (new_population[i,8] == 0):
            # aluminium tank
            zn[i,2] = 503e6 * 0.00014504 # conversion from Pa
                to KSI
            zn[i,3] = 2810 # density
        if (new_population[i,8] == 1):
            # stainless steel
            zn[i,2] = 650e6 * 0.00014504 # conversion from Pa
                to KSI
            zn[i,3] = 7900 # density
# if (new_population[i,8] == 2):
# # composite, carbon fiber
# zn[i,2] = e6 * 0.00014504 # conversion from Pa to KSI
# zn[i,3] = 2699 # density
    return zn

def population(pop_size, prop_min, prop_max,tank_min,tank_max,
    ps_min,
               ps_max,gas_mass_min,gas_mass_max,T_min,T_max,
                   chamber_pressure_min,
               chamber_pressure_max, Re_min, Re_max, er_min,
                   er_max):
        population = numpy.zeros (shape=pop_size)
        size = pop_size[0] # Initializing the matrix
        population[0:,0] = numpy.random.randint(low=prop_min,
            high=prop_max, size=size) # Propellant
        population[0:,1] = numpy.random.randint(low=tank_min,
            high=tank_max, size=size) # Tank type
        population[0:,2] = numpy.random.randint(low=ps_min,high
            =ps_max, size=size) # Tank pressure [bar]
        population[0:,2] = population[0:,2]*1e5 # Converted to
            [Pa]
```

```
        population[0:,3] = numpy.random.uniform(low=
            gas_mass_min,high=gas_mass_max, size=size) #
            Propellant Mass
        population[0:,4] = numpy.random.randint(low=T_min,high=
            T_max, size=size) # Tank temperature; initial range
            283-313 K
        population[0:,5] = numpy.random.randint(
            chamber_pressure_min,chamber_pressure_max, size=size
            ) # Operating Pressure
        population[0:,5] = population[0:,5]*1e5
        population[0:,6] = numpy.random.randint(low=Re_min,high
            =Re_max, size=size) # Exit radius in [mm]
        population[0:,6] = population[0:,6]/1000 # Converted in
             [m]
# new_population[0:,7] = numpy.random.randint(low=er_min*10,
    high=er_max*10, size=colomn_size)
# new_population[0:,7] = new_population[0:,7]/10
        population[0:,7] = numpy.random.randint(low=er_min,high
            =er_max+1, size=size) # Expansion Ratio
        population[0:,8] = numpy.random.randint(low=0,high=2,
            size=size)
        return population
```

A.3 Solver

```
# Aerospike Nozzle Design Solver
# Matthew Vernacchia - MIT Rocket Team
# 2014 Jan 15

# Modified by Giorgio Tesser, mail: giorgio.tesser@studenti.
   unipd.it

from math import atan, sin, tan, pi, asin
import numpy as np
from matplotlib import pyplot as plt
from collections import namedtuple

### Warning logging ###
import logging
formatter = logging.Formatter('%(levelname)s␣{%(pathname)s:%(
   lineno)d}:␣%(message)s')
ch = logging.StreamHandler()
ch.setFormatter(formatter)
logger = logging.getLogger('simple_example')
logger.addHandler(ch)

### Physical Constants ###
# Molar Gas Constant [J mol^-1 K^-1]
R_mol = 8.314
# Acceleration due to gravity at surface [m s^-2]
g = 9.81


### Set up the engine parameters tuple ###
EngineParameters = namedtuple('EngineParameters', 'Tc␣Pc␣
   molar_m␣gamma␣Re␣er␣Ps')
params = EngineParameters(Tc=1900, Pc=4e5, molar_m=20.18, gamma
   =1.4, Re=0.015, er=12, Ps=200e5)
zero_params = EngineParameters(Tc=0, Pc=0, molar_m=0, gamma=0,
   Re=0, er=0, Ps=0)
def aerosolver(params):
    # Number of points along spike to solve for
    N = 100
```

```
    # the mach numbers to examine
    M = np.zeros((N,))
    # the radius of the plug at some point x, normalized by
       the outer radius
    RxRe = np.zeros((N,))
    # The axial distance from the lip to point x, normalized
       by the outer
    # radius
    X = np.zeros((N,))
    # The flow velocity to mach wave angle [rad]
    mu = np.zeros((N,))
    # the turning angle [rad]
    nu = np.zeros((N,))
    # the pressure at point x [Pa]
    P = np.zeros((N,))
    # the temperature at point x [K]
    T = np.zeros((N,))
    # the cumulative Isp up to point x [s]
    Isp = np.zeros((N,))
    # Thrust force [N]
    F = 0
    # Throat mass flow [kg s^-1]
    m_dot = 0
    # Throat Area [m^2]
    At = 0
    # Throat gap width [m]
    ht = 0
    # Angle between shroud surface and a plane normal to the
       nozzle axis [rad]
    delta = 0
    # Ambient pressure [Pa]
    Pa=1
    # Nozzle Thrust Coefficent [-]
    Cf = 0
    ''' Solve for the spike contour and nozzle conditions
       given the engine parameters.params
    Engine Parameters namedtuple Pa Ambient pressure [Pa]
        '''
    #last_params = params
    #gamma
    y=params.gamma
```

```
    #massa molare
    molar_m = params.molar_m
    # Find the Specific Gas Constant
    R = R_mol / molar_m * 1000

#### Compute the Exit Mach Number ####
    Me = get_Mach( params )
    # use this to find the exit (tip-plane) pressure (assumung
        isetropic expansion)
    Pe = params.Pc * (1 + (y-1)/2*Me**2)**(-y/(y-1))

    #### Find the total flow turning angle [rad] ####
    nu_e = ((y+1)/(y-1))**0.5 * atan( ((y-1)/(y+1)*(Me**2-1))
       **0.5 ) \
           - atan( (Me**2-1)**0.5 )
    # use this to find the angle of the shroud edge with the
       normal to
    # the axial direction [rad]
    delta = pi/2 - nu_e

    #### Find the throat gap width / outer radius ratio []
       ####
    htRe = (params.er - (params.er*(params.er-sin(delta)))
       **0.5) \
    / (params.er*sin(delta))
    #### Find the velocity and temperature at the throat (
       assuming M=1) ####
    # throat thermo temperature [K]
    Tt = params.Tc / (1 + (y-1)/2)
    # throat pressure [Pa]
    Pt = params.Pc * (1 + (y-1)/2) ** (-y/(y-1))
    # throat fluid velocity [m s^-1]
    vt = np.sqrt( y*R*Tt )
    #### Examine a range of Mach numbers to determine the
       shape of the nozzle
    # the Mach numbers to examine
    M = np.linspace(1, Me, N)
    # the Isp due to momentum flux and pressure at the throat
       [sec]
    Isp[0] = vt*sin(delta)*( 1 + 1/y*(1-((y+1)/2)**(y/(y-1))*Pa
       /params.Pc) ) / g
```

```
    for x in range(N):
        mu[x] = asin( 1/M[x] ) # See Lozano's Fall2012 Lec17
            Notes
        # use the Prandtl-Meyer equation to find nu[x]
        nu[x] = ((y+1)/(y-1))**0.5 * atan( ((y-1)/(y+1)*(M[x
            ]**2-1))**0.5 ) \
        - atan( (M[x]**2-1)**0.5 )
        # use CC Lee Eqn (26) to find Rx/Re for the point
        RxRe2 = ( 1 - ( 2/(y+1)*(1+(y-1)/2*M[x]**2))**((y+1)
            /(2*y-2)) \
            * sin( nu_e - nu[x] + mu[x] ) / params.er )
        if RxRe2 > 0:
            RxRe[x] = RxRe2**0.5
        else: RxRe[x] = 0
        # find the X (axial) coordinate of the point. CC Lee
            Eqn (19)
        X[x] = (1 - RxRe[x]) / tan( nu_e - nu[x] + mu[x] )
        # find the pressure
        P[x] = params.Pc * (1 + (y-1)/2*M[x]**2)**(-y/(y-1))
        if x > 0:
            Isp[x] = Isp[x-1] + ( vt/y*((y+1)/2)**(y/(y-1)) * \
                params.er/2 * ( (P[x-1]-Pa)/params.Pc + (P[x]-Pa
                    )/params.Pc ) * \
                (RxRe[x-1]**2 - RxRe[x]**2) ) / g
        #find the temperature
        T[x] = params.Tc / (1+ (y-1)/2 * M[x]**2)
    Te = T[N-1]
    # throat width [meter]
    ht = htRe * params.Re
    # I added this function to avoid the presence of
        unreasonable numbers like 1.05
    ht = ht * 1000
    if ht > 1 :
        ht = np.floor(ht)
    ht = ht / 1000
    # throat area [meter^2]
    At = pi*ht*(2*params.Re - ht*sin(delta))
    # fluid density at the throat [kg m^-3]
    rho_t = Pt / (R*Tt)
    # throat mass flow [kg sec^-1]
    m_dot = rho_t*At*vt
```

```
    # Find the thrust [N]
    F = Isp[-1] * m_dot * g
    # Find the thrust coefficient [-]
    Cf = F / (params.Pc*At)
    #-----------------------------------------------------
    Performance_results = namedtuple('Performance_results', '
       delta␣Re␣ht␣At␣er␣Pc␣Tc␣m_dot␣F␣Isp␣Te␣Pe␣Me␣Cf␣Ps')
    delta = delta*180/pi
    Re = params.Re * 1000
    ht = ht * 1000
    Pc = params.Pc/1e5
    Ps = params.Ps/1e5

    performance = Performance_results(delta=delta, Re=Re, ht=ht
        , At=At, er=params.er, Pc=Pc, Tc=params.Tc, m_dot=m_dot,
         F=F, Isp=Isp, Te=Te, Pe=Pe, Me=Me, Cf=Cf, Ps=Ps)
# results_string = \
# 'Engine Geometry:' \
# + '\n' + '\tShroud angle, delta = %.1f degrees'%(delta*180/
    pi) \
# + '\n' + '\tShroud lip radius, Re = %.1f mm'%(params.Re
    *1000) \
# + '\n' + '\tThroat width, ht = %.2f mm'%(ht*1000) \
# + '\n' + '\tThroat area, At = %.8f m^2'%(At) \
# + '\n' + '\tExpansion ratio, er = %.2f'%(params.er) \
# + '\n' + 'Chamber Conditions:' \
# + '\n' + '\tChamber pressure, Pc = %.3f MPa'%(params.Pc/1.0
    e6) \
# + '\n' + '\tChamber temperature, Tc = %.0f K'%(params.Tc) \
# + '\n' + '\tExhaust Avg Molar Mass = %.1f g mol^-1'%(params
    .molar_m) \
# + '\n' + 'Engine Performance:' \
# + '\n' + '\tMass flow rate, m_dot = %.3f kg sec^-1'%(m_dot)
     \
# + '\n' + '\tThrust force, F = %.1f N'%(F) \
# + '\n' + '\tSpecific impulse, Isp = %.1f sec'%(Isp[-1]) \
# + '\n' + '\tExit pressure, Pe = %.2f Pa'%(Pe) \
# + '\n' + '\tExit Mach number, Me = %.2f'%(Me) \
# + '\n' + '\tThrust coefficient, Cf = %.2f'%(Cf)
    return performance
""" Giorgio's comment:
```

```
    Uncomment the following part and add properly the plot
       function to the main.py
    to plot the spike contour
    Alternatively it is possible to launch the Aerospike
       Nozzle Design Solver
    from terminal, fill the UI with the requested parameters
       and then copy the
    spike contour"""


# def plot(self):
# '''Plot the current solution results for the nozzle
   conditions.'''
# plt.suptitle('Simulated Nozzle Conditions vs Axial Distance
    from Throat Normalized by Exit Radius')
# plt.subplot(2,3,1)
# plt.cla()
# plt.plot( X, RxRe )
# plt.ylabel('Rx / Re')
# plt.grid(True)
#
# plt.subplot(2,3,4)
# plt.cla()
# plt.plot( X, M )
# plt.ylabel( 'Mach Number')
# plt.xlabel('Xx / Re')
# plt.grid(True)
#
# plt.subplot(2,3,2)
# plt.cla()
# plt.plot( X, P/1.0e6)
# plt.plot( X, np.ones((N,))*Pa/1.0e6, 'r' )
# plt.ylabel('Pressure [MPa]')
# plt.grid(True)
#
# plt.subplot(2,3,5)
# plt.cla()
# plt.plot( X, T )
# plt.ylabel('Temperature [K]')
# plt.xlabel('Xx / Re')
# plt.grid(True)
```

```
#
# plt.subplot(2,3,3)
# plt.cla()
# plt.plot( X, Isp )
# plt.ylabel(r'Cumulative $I_{sp}$')
# plt.xlabel('Xx / Re')
# plt.grid(True)
#
# plt.subplot(2,3,6)
# plt.cla()
# plt.text(0.1,0.8, r'$A_e/A_t$ = %.1f'%(last_params.er))
# plt.text(0.1,0.7, r'$P_c =$ %.2f MPa'%(last_params.Pc/1e6))
# plt.text(0.1,0.6, r'$T_c =$ %.0f K'%(last_params.Tc))
# plt.text(0.1,0.5, r'$M_{molar} =$ %.1f g mol^-1'%(
   last_params.molar_m))
# plt.text(0.1,0.4, r'$\gamma =$ %.2f'%(last_params.gamma))
#
# plt.show(block=False)

### Utility Thermofluids Functions ###

def get_Mach( params ):
    '''
    Find the exit Mach number given the engine parameters.
    This function uses the params.er (expansion ratio) and
       params.gamma (gas specific heat ratio)

    Explicit Inversion of Stodola's Area-Mach Equation
    Source: J. Majdalani and B. A. Maickie
    http://maji.utsi.edu/publications/pdf/HT02_11.pdf
    '''
    n = 5 # order of the aproximation
    X = np.zeros((n,))
    M = np.zeros((n,))

    e = 1/float(params.er) # expansion ratio
    y = params.gamma # ratio of specific heats
    B = (y+1)/(y-1)
    k = np.sqrt( 0.5*(y-1) )
    u = e**(1/B) / np.sqrt( 1+k**2 )
    X[0] = (u*k)**(B/(1-B))
```

```
    M[0] = X[0]

    for i in range(1,n):
        lamb = 1/( 2*M[i-1]**(2/B)*(B-2) + M[i-1]**2 *B**2*k
            **2*u**2 )
        X[i] = lamb*M[i-1]*B*( M[i-1]**(2/B) - M[i-1]**2*B*k
            **2*u**2 \
            + ( M[i-1]**(2+2/B)*k**2*u**2*(B**2-4*B+4) \
            - M[i-1]**2*B**2*k**2*u**4 + M[i-1]**(4/B)*(2*B-3) \
            + 2*M[i-1]**(2/B)*u**2*(2-B) )**0.5 )
        M[i] = M[i-1] + X[i]
    if abs( np.imag( M[n-1] ) ) > 1e-5:
        logger.warning('Exit␣Mach␣Number␣has␣nonzero␣imaginary␣
            part!')
    Me = np.real( M[n-1] )
    return Me

#------------------------------------------------------

""" Giorgio's comment:
    Unused part of the Aerospike Nozzle Design Solver, but it
        may be useful in future """

#def get_Pe( params ):
# ''' Find the nozzle exit static pressure, given the engine
    parameters.
# returns nozzle exit pressure [Pa]
# '''
# # Find the exit Mach number
# Me = get_Mach(params)
# y = params.gamma
# # Use this to find the exit (tip-plane) pressure (assumung
    isetropic expansion)
# Pe = params.Pc * (1 + (y-1)/2*Me**2)**(-y/(y-1))
# return Pe

#def get_Pa_from_alt( alt ):
# ''' Find the ambient atmospheric pressure, given the
    altitude above mean sea level in Earth's atmosphere.
# This functionnis only valid for input altitudes up to 44 km
    .
```

```
# alt Altitiude above mean sea level [m]
# returns Atmospheric pressure [Pa]
# '''
# if alt < 44331:
# # Pressure (http://psas.pdx.edu/RocketScience/
   PressureAltitude_Derived.pdf, eqn 9)
# Pa = 100 * ((44331.514 - alt)/(11880.516))**(1/0.1902632)
# return Pa
# else:
# logger.warning('Atmospheric pressure model does not extend
   to the altitude requested. returning 0 Pa pressure')
# return 0

#def get_alt_from_Pa( Pa ):
# ''' Given the ambient atmospheric pressure, find the
   altitude above mean sea level in Earth's atmosphere.
# This functionnis only valid for input altitudes up to 44 km
   .
# alt Altitiude above mean sea level [m]
# returns Atmospheric pressure [Pa]
# '''
# if Pa > 0:
# # Altitude (http://psas.pdx.edu/RocketScience/
   PressureAltitude_Derived.pdf, eqn 10)
# alt = 44331.5 - 4946.62*Pa**0.190263
# return alt
# else:
# logger.warning('Negative pressure input. returning zero
   altitude')
# return 0

#def get_Re_from_thrust( params, F ):
# ''' Find the shroud radius, given the thrust force and the
   other parameters.
# F Desired pressure-matched thrust force [N]
# returns Needed shroud radius [m]
# '''
# y = params.gamma
# Pe = get_Pe(params)
# # Find the pressure-matched thrust coefficient, using
# # Rocket Propulsion Elements 7th Ed, Equation 3-30
```

```
# C_F = ( (2*y**2)/(y-1) * (2/(y+1))**((y+1)/(y-1)) \
# * (1-(Pe/params.Pc)**((y-1)/y)) )**0.5
# # Use the thrust coefficient to find the throat area
   required
# At = F / (C_F * params.Pc)
# Ae = At * params.er
# # The Exit Area is a circle of radius Re
# Re = (Ae/pi)**0.5
# return Re

#def get_thrust( params ):
# ''' Find the pressure-matched thrust of the engine, given
   the engine parameters.
# returns Thrust force at matched pressure (Pa=Pe) [N]
# '''
# y = params.gamma
# Pe = get_Pe(params)
# # Find the pressure-matched thrust coefficient, using
# # Rocket Propulsion Elements 7th Ed, Equation 3-30
# C_F = ( (2*y**2)/(y-1) * (2/(y+1))**((y+1)/(y-1)) \
# * (1-(Pe/params.Pc)**((y-1)/y)) )**0.5
# # Find the throat area
# Ae = pi*params.Re**2
# At = Ae/params.er
# F = C_F*At*params.Pc
# return F
#
#def get_Re_from_mass_flow( params, m_dot ):
# ''' Find the shroud radius, given the mass flow and the
   other parameters.
# m_dot Desired mass flow [kg / s]
# returns Needed shroud radius [m]
# '''
# y = params.gamma
# # Find the Specific Gas Constant
# R = R_mol / params.molar_m * 1000
# # Find the Throat Area require for the specified mass flow,
    using
# # Eocket Propul;sion Equations 7th Ed, Equation 3-24
# At = m_dot / ( params.Pc * y * (2/(y+1))**((y+1)/(2*y-2)) \
# / (y*R*params.Tc)**0.5)
```

```
# # Use the Throat Area and the Expansion Ratio to find the
   Exit Area
# Ae = At * params.er
# # The Exit Area is a circle of radius Re
# Re = (Ae/pi)**0.5
# return Re
#
#def get_er_from_Pe( params, Pe ):
# ''' Find the needed expansion ratio, given a desired exit
   pressure and the other engine parameters.
# Pe Desired exit pressure [Pa]
# returns Needed expansion ratio [-]
# '''
# y = params.gamma
# # Rocket Propulsion Elements 7th Ed, Equation 3-25
# AtAe = ((y+1)/2)**(1/(y-1)) * (Pe/params.Pc)**(1/y) \
# * ( (y+1)/(y-1)*( 1 - (Pe/params.Pc)**((y-1)/y) ) )**0.5
# er = 1/AtAe
# return er
```

A.4 Fitness Function

```
"""
@author: Giorgio Tesser
mail: giorgio.tesser@studenti.unipd.it
"""
import math
import numpy
from scipy.optimize import fsolve

# Calculation of the volume of the tank
def tank (new_population,zn):
# Z = numpy.zeros (shape=colomn_size)
# Mm = numpy.zeros (shape=colomn_size)
# Rp = numpy.zeros (shape=colomn_size)
    tank_pressure = new_population[0:,2]
    T = new_population[0:,4]
    Z = zn[0:,0]
    n = zn [0:,1]
    R=8.314472
    Vt = (n*R*T*Z)/tank_pressure
    return Vt

# Geometry of the tank. Section based on
# Ames R. Farr and Maan H. Jawad.
# "Guidebook for the design of asme section VIII pressure
   vessels"
# Imperial units used

def tank_design (V,sol_per_pop,new_population,zn):
    t_size = (sol_per_pop,4)
    t = numpy.zeros (t_size)
    for i in range(0,sol_per_pop):
          if (new_population[i,1] == 0): # sphere
              # Depending on the solution (i), can be aluminum
                   or stainless steel
              sigma = zn[i,2] # evoking zn from "data.py"
              r=((3/4.0)*V[i]/math.pi) # radius of the sphere
              r=math.pow(r,(1/3.0))
              r=r*39.37008
```

```
            P=new_population[i,2]*1.3*0.00014504 # Pressure
               in PSI
            e= (P/sigma)/0.0665
            t[i,0]=(P*r)/((2*sigma*e)-(0.2*P))
            t[i,1]=numpy.copy(t[i,0])
            t[i,2]=r
            t[i,3]=numpy.copy(t[i,2])
            # 0 and 1 are the thickness of the sphere (that
               are equal)
            # 2 and 3 are the radii of the sphere ( that are
                equal)
        else: # cylinder
            # Depending on the solution (i), can be aluminum
                or stainless steel
            sigma = zn[i,2] # evoking zn from "data.py"
            # Most common ratio between h and r is in range
               [6.5-7]
            r=math.pow((V[i]/6.75*math.pi),(1/2.0)) # in
               meters
            h=6.75*r
            h=h*39.37008 # in inches
            r=r*39.37008 # in inches
            P=new_population[i,2]*1.3*0.00014504 # Pressure
               in PSI
            e_circ= (P/sigma)/0.385
            t_circ=(P*r)/((sigma*e_circ)-(0.6*P))
            t[i,0]=t_circ
            e_long= (P/sigma)/1.25
            t_long=(P*h)/((2*sigma*e_long)+(0.4*P))
            t[i,1]=t_long
            t[i,2]=r
            t[i,3]=6.75*(t[i,2])
            # 0 and 1 are the thickness of the cylinder
            # 2 is the radius of the cylinder
            # 3 is the height of the cylinder
    t=t*0.0254 # in meters
    return t

# Calculation of the empty tank mass, based on density of teh
    material,
# thickness and shape
```

```
def empty_tank_mass (t,sol_per_pop,new_population,zn):
    etm_size = (sol_per_pop,1)
    empty_tank_mass = numpy.zeros (etm_size)
    for i in range(0,sol_per_pop):
          if (new_population[i,1] == 0):
              density = zn[i,3]
              solid_volume1=((4/3.0)*math.pi*math.pow(t[i
                 ,2],(3.0)))
              solid_volume2=((4/3.0)*math.pi*math.pow((t[i,2]-t
                 [i,0]),3))
              empty_tank_mass[i] = density*(solid_volume1-
                 solid_volume2)
              # 0 and 1 are the thickness of the sphere (that
                 are equal)
              # 2 and 3 are the radii of the sphere ( that are
                  equal)
          else:
              density = zn[i,3]
              solid_volume1=math.pi*math.pow(t[i,2],2)*t[i,3]
              solid_volume2=math.pi*math.pow((t[i,2]-t[i,0]),2)
                 *(t[i,3]-t[i,1])
              empty_tank_mass[i] = density*(solid_volume1-
                 solid_volume2)
              # 0 and 1 are the thickness of the cylinder
              # 2 is the radius of the cylinder
              # 3 is the height of the cylinder
    return empty_tank_mass

# Evaluation of the fitness level

def FL_calc(new_population,m_diff,sol_per_pop,parents_num,
    performance_matrix,coeff_FL_it,coeff_FL_cost,coeff_FL_er,
    coeff_FL_m_dot,coeff_FL_epsilon,coeff_FL_ht):

    # Constant of the fitness calculation

    x1=11
    x1_i=11
    x2=12
    x3=11
    x4=11
```

```
x4_i=11
x5=11
x5_i=10
g0 = 9.81

# Initialization of the vectors

size1= (sol_per_pop)
FL_epsilon = numpy.zeros(size1)
FL_er = numpy.zeros(size1)
FL_cost = numpy.zeros(size1)
It = numpy.zeros (size1)
FL_it = numpy.zeros (size1)
FL_ht = numpy.zeros(size1)
FL_m_dot = numpy.zeros(size1)
m_diff_abs = abs(m_diff)

# Difference between the available mass and the calculated
    mass

for i in range (0,sol_per_pop):

    # Finding the lowest difference in mass, i.e. the
       solution that
    # exploits all the available mass

    best_m = numpy.where(m_diff_abs == numpy.min(m_diff_abs
       ))
    best_m_idx = best_m[0][0]
    m_diff_abs[best_m_idx] = math.pow (10,10)

    # Bonus for the solution that respect the condition.
    # Proportional bonus, higher for the lowest mass
       difference

    if m_diff[best_m_idx] > 0 and m_diff[best_m_idx] <5 :
        FL_epsilon[best_m_idx] = math.pow (math.e,x1)
        x1=x1-x1_i/(sol_per_pop+1)

    # Flat malus for solutions that don't respect the
       condition
```

```
        else:
            FL_epsilon[best_m_idx] = -6e10

    # Total Impulse
    for i in range (0,sol_per_pop):
        Isp = performance_matrix[i,9]
        propellant_mass = new_population[i,3] #propellant_mass
            = gas_mass
        It[i] = Isp * propellant_mass * g0
        It2 = numpy.copy(It)# I use a fake It to calculate the
            maximum at each iteration
        # Proportional bonus, higher for higher It
    for i in range (0,sol_per_pop):
        best_it = numpy.where ( It2 == numpy.max(It2))
        best_it_idx = best_it[0][0]
        It2[best_it_idx] = 1
        FL_it[best_it_idx] = math.pow(math.e,x3)
        x3 = x3 - 0.01

    # Flat penalizations for solution with ht<1mm and small
        bonus for the
    # solutions that respect the condition

    for i in range (0, sol_per_pop):
        ht_ev = performance_matrix[i,2]
        if ht_ev < 1:
            FL_ht[i] = -6e10
        else:
            FL_ht[i] = +1e4

    # Flat malus for propellant cost

    for i in range (0, sol_per_pop):
        propellant_cost = [0,0,0,0,0,0]
        propellant_cost[0]=275 #nitrogen
        propellant_cost[1]=275 #helium
        propellant_cost[2]=275 #argon
        propellant_cost[3]=1585 #xenon
        propellant_cost[4]=295 #methane
        gas_type = new_population[i,0]
        if gas_type==0:
```

```
            FL_cost[i] = -propellant_cost[0]
        if gas_type==1:
            FL_cost[i] = -propellant_cost[1]
        if gas_type==2:
            FL_cost[i] = -propellant_cost[2]
        if gas_type==3:
            FL_cost[i] = -propellant_cost[3]
        if gas_type==4:
            FL_cost[i] = -propellant_cost[4]

    # Bonus for expansion ratio
    # Proportional bonus, higher for the higher expansion
       ratios
    exp_ratio =numpy.copy(new_population[:,7])
    for i in range (0, sol_per_pop):
        best_er = numpy.where(exp_ratio == numpy.max(exp_ratio)
            )
        best_er_idx = best_er[0][0]
        exp_ratio[best_er_idx] = 0
        FL_er[best_er_idx] = math.pow (math.e,x4)
        x4=x4-x4_i/(sol_per_pop+1)

    # Malus for mass flow
    # Proportional bonus, higher for the higher mass flows
    m_dot1 =numpy.copy(performance_matrix[:,7])
    for i in range (0, sol_per_pop):
        best_m_dot1= numpy.where(m_dot1 == numpy.max(m_dot1))
        best_m_dot1_idx = best_m_dot1[0][0]
        m_dot1[best_m_dot1_idx] = 0
        FL_m_dot[best_m_dot1_idx] = -math.pow (math.e,x5)
        x5=x5-x5_i/(sol_per_pop+1)

    # Initialization of a global FL vector

    FL_dim = [sol_per_pop,6]
    FL = numpy.zeros(FL_dim)

    # Definition of a FL vector, using coefficient stated at
       the beginning of
    # Main.py and FL defined here
    FL[:,0] = coeff_FL_epsilon*FL_epsilon
```

```
    FL[:,1] = coeff_FL_it*FL_it
    FL[:,2] = coeff_FL_ht*FL_ht
    FL[:,3] = coeff_FL_cost*FL_cost
    FL[:,4] = coeff_FL_er*FL_er
    FL[:,5] = coeff_FL_m_dot*FL_m_dot
    return FL

# Calculation of the fitness level for the rerun
# It's similar but it takes into account less parameters
def FL_calc_rerun(new_population,m_diff,sol_per_pop,parents_num
    ,performance_matrix,coeff_FL_it,coeff_FL_cost,coeff_FL_er,
    coeff_FL_m_dot,coeff_FL_epsilon):

    # Constant of the fitness calculation

    x1=10.0
    x1_i=10.0
    x2=11

    # Initialization of the parameters

    size1= (sol_per_pop)
    FL_m_dot = numpy.zeros(size1)
    FL_ht = numpy.zeros(size1)
    m_diff_abs = abs(m_diff)
    FL_epsilon = numpy.zeros(size1)

    # Flat penalizations for solution with ht<1mm and small
       bonus for the
    # solutions that respect the condition

    for i in range (0, sol_per_pop):
        ht_ev = performance_matrix[i,2]
        if ht_ev < 1:
            FL_ht[i] = -6e10
        else:
            FL_ht[i] = +1e4

    # Malus for mass flow
    # Proportional bonus, higher for the higher mass flows
```

```
m_dot1 =numpy.copy(performance_matrix[:,7])
for i in range (0, sol_per_pop):
    best_m_dot1= numpy.where(m_dot1 == numpy.max(m_dot1))
    best_m_dot1_idx = best_m_dot1[0][0]
    m_dot1[best_m_dot1_idx] = 0
    FL_m_dot[best_m_dot1_idx] = -math.pow (math.e,x1)
    x1 = x1-x1_i/(sol_per_pop+1)


for i in range (0,sol_per_pop):

    # Finding the lowest difference in mass, i.e. the
       solution that
    # exploits all the available mass

    best_m = numpy.where(m_diff_abs == numpy.min(m_diff_abs
       ))
    best_m_idx = best_m[0][0]
    m_diff_abs[best_m_idx] = math.pow (10,10)

    # Bonus for the solution that respect the condition.
    # Proportional bonus, higher for the lowest mass
       difference

    if m_diff[best_m_idx] > 0 and m_diff[best_m_idx] <5 :
        FL_epsilon[best_m_idx] = math.pow (math.e,x1)
        x1=x1-x1_i/(sol_per_pop+1)

    # Flat malus for solutions that don't respect the
       condition
    else:
        FL_epsilon[best_m_idx] = - math.pow (math.e,x2)


# Initialization of a global FL vector

FL_dim = [sol_per_pop,3]
FL = numpy.zeros(FL_dim)

# Definition of a FL vector, using coefficient stated at
   the beginning of
```

```
    # Main.py and FL defined here

    FL[:,0] = FL_ht
    FL[:,1] = coeff_FL_m_dot*FL_m_dot
    FL[:,2] = coeff_FL_epsilon*FL_epsilon
    return FL
```

A.5 GA

```
"""
@author: Giorgio Tesser
mail: giorgio.tesser@studenti.unipd.it
"""
# List of libraries
import math
import numpy

# Selection of the parents based on the 20% best FL
def select_mating_pool(fl,sol_per_pop,parents_num,
   new_population,num_genes):
    FL_parents = numpy.copy(fl)
    parents = numpy.empty((parents_num, num_genes))
    for i in range(0,parents_num):
        max_FL = numpy.where (FL_parents == numpy.max(
           FL_parents))
        max_FL_idx = max_FL[0][0]
        parents[i,:] = new_population [max_FL_idx,:]
        FL_parents[max_FL] = -1e5
    return parents

# Crossover
def crossover(parents,offspring_size):
    offspring = numpy.empty (offspring_size)
    for k in range (offspring_size [0]):
        parent1_idx = k%parents.shape[0] #index of the first
           parent to mate
        parent2_idx = (k+1)%parents.shape[0] #index of the
           second parent to mate
        #The new offspring will have half of its genes from
           parent 1 and the other half from parent 2
        offspring [k,0:offspring_size[0]:2] = parents[
           parent1_idx,0:offspring_size[0]:2]
        offspring [k,1:offspring_size[0]:2] = parents[
           parent2_idx,1:offspring_size[0]:2]
    return offspring

# Mutation, uncomment lines with ## to add mutation
```

```
def mutation(chamber_pressure_min,chamber_pressure_max,
    offspring_crossover,offspring_size):
    mutation=numpy.int(0.2*offspring_size[0])
    random_mutation5=numpy.random.randint(0,offspring_size[0],
        mutation)
    random_mutation7=numpy.random.randint(0,offspring_size[0],
        mutation)
    for i in range(mutation):
        # Mutation of the chamber pressure
        ##idx5 = random_mutation5[i]
        ##if (offspring_crossover[idx5,5]<(pcmax-5e4) and
            offspring_crossover[idx5,5]<(pcmin-5e4) ):
            ##random_value5 = numpy.random.randint(-5e4, 5e4,
                1)
            ##offspring_crossover[idx5, 5] =
                offspring_crossover[idx5, 5] +random_value5

        # Mutation of the expansion ratio
        idx7 = random_mutation7[i]
        random_value7 = numpy.random.randint(-0.5, 1.5, 1)
        offspring_crossover[idx7, 7] = offspring_crossover[idx7
            , 7] +random_value7
    return offspring_crossover

# Generation of random solution in the design ranges
def random_gen(random_sol_shape,chamber_pressure,ps_min,ps_max,
    tank_min,tank_max,gas_mass_min,gas_mass_max,Re_min,Re_max,
    er_min,er_max,prop_min,prop_max):
    random_solutions= random_sol_shape[0]
    random_gen=numpy.zeros(random_sol_shape)
    # Propellant type
    random_gen[0:random_solutions,0] = numpy.random.randint(low
        =prop_min,high=prop_max, size=random_solutions)
    # Tank type
    random_gen[0:random_solutions,1] = numpy.random.randint(low
        =tank_min,high=tank_max, size=random_solutions)
    # Tank pressure in bar
    random_gen[0:random_solutions,2] = numpy.random.randint(low
        =ps_min,high=ps_max, size=random_solutions)
    # converted to Pa
```

```
    random_gen[0:random_solutions,2] = random_gen[0:
        random_solutions,2]*1e5
    # Gas_Mass
    random_gen[0:random_solutions,3] = numpy.random.uniform(low
        =gas_mass_min,high=gas_mass_max, size=random_solutions)
    # Tank temperature
    random_gen[0:random_solutions,4] = numpy.random.randint(low
        =283,high=313, size=random_solutions)
    # Thrust chamber pressure
    random_gen[0:random_solutions,5] = chamber_pressure
    # Exit radius in mm
    random_gen[0:random_solutions,6] = numpy.random.randint(low
        =Re_min,high=Re_max, size=random_solutions)
    # converted in m
    random_gen[0:random_solutions,6] = random_gen[0:
        random_solutions,6]/1000
    # Expansion ratio
    random_gen[0:random_solutions,7] = numpy.random.randint(low
        =er_min,high=er_max+1, size=random_solutions)
    # Tank material:0=aluminum,1=stainless steel
    random_gen[0:random_solutions,8] = numpy.random.randint(low
        =0,high=2, size=random_solutions)
    return random_gen
```

A.6 Rerun

```
"""
@author: Giorgio Tesser
mail: giorgio.tesser@studenti.unipd.it
"""
# List of libraries
import numpy
import math
from scipy.optimize import fsolve
import Data
import Fitness
import GA
import aero_solver
from collections import namedtuple

# Definition of the re_run cycle.

# It's not commented becuase it's the exact same copy of the
   main.py, see that
# for explanation of the functions

def rerun(new_population,sol_per_pop,maximum_tank_mass,
   coeff_FL_it,coeff_FL_cost,coeff_FL_er,coeff_FL_m_dot,
   num_genes, coeff_FL_epsilon,coeff_FL_ht):

    # Defining the population size
    pop_size = (sol_per_pop,num_genes)
    num_parents_mating = 4
    toll = 2
    generation=0
    condition1=True
    Best_solution_size = (1,num_genes)
    Queens = numpy.zeros(Best_solution_size)
    Queens_add = numpy.zeros(Best_solution_size)
    colomn_size= (sol_per_pop)

    while 1:
        if toll==0:
            condition1=False
```

```
        if condition1==True:
            zn = Data.parameters (new_population,sol_per_pop)
            V = Fitness.tank (new_population,zn)
            t = Fitness.tank_design(V,sol_per_pop,new_population
                ,zn)
            empty_tank_mass = Fitness.empty_tank_mass (t,
                sol_per_pop,new_population,zn)
            gas_size = (sol_per_pop,1)
            gas_mass=numpy.zeros(gas_size)
            gas_mass [0:,0] = new_population[0:,3]
            full_tank_mass = empty_tank_mass + gas_mass
            m_diff= maximum_tank_mass - full_tank_mass
            parents_num = int(0.2*sol_per_pop)
            performance_size=[sol_per_pop,16]
            performance_matrix = numpy.zeros(shape=
                performance_size)
            EngineParameters = namedtuple('EngineParameters', '
                Tc␣Pc␣molar_m␣gamma␣Re␣er␣Ps')
            for j in range (0,sol_per_pop):
                T = new_population[j,4]
                p_chamber = new_population[j,5]
                p_storage = new_population[j,2]
                Re = new_population[j,6]
                er = new_population[j,7]
                molar_mass = gas_mass[j]*1000/zn[j,1]
                params = EngineParameters(Tc=T, Pc=p_chamber,
                    molar_m=molar_mass, gamma=1.27, Re=Re, er=er,
                     Ps= p_storage)
                performance = aero_solver.aerosolver(params)
                performance_matrix[j,0] = performance.delta
                performance_matrix[j,1] = performance.Re
                performance_matrix[j,2] = performance.ht
                performance_matrix[j,3] = performance.At
                performance_matrix[j,4] = performance.er
                performance_matrix[j,5] = performance.Pc
                performance_matrix[j,6] = performance.Tc
                performance_matrix[j,7] = performance.m_dot
                performance_matrix[j,8] = performance.F
                performance_matrix[j,9] = performance.Isp[-1]
                performance_matrix[j,10] = performance.Pe
                performance_matrix[j,11] = performance.Te
```

```
            performance_matrix[j,12] = performance.Me
            performance_matrix[j,13] = performance.Cf
            performance_matrix[j,14] = performance.Ps
            performance_matrix[j,15] = full_tank_mass[j]
        FL= Fitness.FL_calc(new_population,m_diff,
            sol_per_pop,parents_num,performance_matrix,
            coeff_FL_it,coeff_FL_cost,coeff_FL_er,
            coeff_FL_m_dot,coeff_FL_epsilon,coeff_FL_ht)
        # Sum of each entries of the FL
        fl= FL[:,0] + FL[:,1] + FL[:,2] + FL[:,3] + FL[:,4]
            + FL[:,5]
        old_pop=numpy.copy(new_population)
        fl_max = numpy.where(fl == numpy.max(fl))
        fl_max_idx = fl_max[0][0]
        Queens[generation,:] = old_pop[fl_max_idx,:]
        Queens = numpy.r_[Queens,Queens_add]
        parents = GA.select_mating_pool(fl,sol_per_pop,
            parents_num,new_population,num_genes)
        offspring_size=[pop_size[0]-parents.shape[0],
            num_genes]
        offspring_crossover_rerun = GA.crossover(parents,
            offspring_size)
        new_population[0:parents.shape[0], :] = parents
        new_population[parents.shape[0]:sol_per_pop, :] =
            offspring_crossover_rerun
        numpy.random.shuffle(new_population)
        generation=generation+1

        if generation>10:
            toll1=0
            for i in range (2,20):
                    toll_v = (Queens[Queens.shape[0]-i,:]-
                        Queens[Queens.shape[0]-(i+1),:])
                    toll1=toll1+numpy.sum(toll_v)
            toll=toll1
    else:
        break

fl_max = numpy.where(fl == numpy.max(fl))
fl_max_idx = fl_max[0][0]
Best_solution = old_pop[fl_max_idx]
```

```
    Best_performance = performance_matrix[fl_max_idx,:]

    Results = namedtuple('Results', 'Idx␣fl␣FL␣Propellant_Type␣
        Tank_Type␣Tank_material␣Tank_Dimensions␣Storage_Pressure
        ␣gass_mass␣full_tank_mass␣Tank_temperature␣
        Chamber_Pressure␣delta␣Re␣ht␣At␣er␣Pc␣Tc␣m_dot␣F␣Isp␣Pe␣
        Te␣Me␣Cf␣Ps')
    Optimized_results = Results(Idx=fl_max_idx, fl=fl[
        fl_max_idx], FL=FL[fl_max_idx], Propellant_Type=
        Best_solution[0],Tank_Type = Best_solution [1],
        Tank_material = Best_solution[8], Tank_Dimensions = t[
        fl_max_idx,:], Storage_Pressure = Best_solution[2],
                            gass_mass = Best_solution[3],
                                full_tank_mass = Best_performance
                                [15], Tank_temperature =
                                Best_solution[4], Chamber_Pressure
                                 = Best_solution[5],
                            delta = Best_performance[0], Re =
                                Best_performance[1], ht =
                                Best_performance[2], At =
                                Best_performance[3], er =
                                Best_performance[4],
                            Pc = Best_performance[5],Tc =
                                Best_performance[6], m_dot =
                                Best_performance[7], F =
                                Best_performance[8], Isp =
                                Best_performance[9],
                            Pe = Best_performance[10], Te =
                                Best_performance[11], Me =
                                Best_performance[12], Cf =
                                Best_performance[13], Ps =
                                Best_performance[14])
    return Optimized_results
```

Bibliography

[1] T. D. Thompson. Trw space log. *Vol. 32 to 34, TRW Space and Electronics Group*, 1996 and 1997-1998.

[2] George P. Sutton. *Rocket Propulsion Elements*. John Wiley & Sons, Inc., New York, 2001. (Cited at pages 1, 3 e 8)

[3] ESA-European Space Agency. Components of an ariane 5g launcher. `https://www.esa.int/spaceinimages/Images/2004/04/Components_of_an_Ariane_5G_launcher`, 2004. Online; accessed 29 March 2019. (Cited at page 2)

[4] Gerald Hagemann, Hans Immich, Thong Van Nguyen, and Gennady E. Dumnov. Advanced rocket nozzles. *JOURNAL OF PROPULSION AND POWER*, September-October 1998. (Cited at pages 3, 5, 7, 10, 11 e 12)

[5] Howard D. Curtis. *Orbital Mechanics for Engineering students*. Elsevier Ltd., Oxford, 2014. (Cited at page 5)

[6] Stephen Corda, Bradford A. Neal, Timothy R. Moes, Timothy Cox, Richard J. Monaghan, Leonard S. Voelker, Griffin P. Corpening, and Richard R. Larson. Flight Testing the Linear Aerospike SR-71 Experiment (LASRE). 1998. (Cited at page 7)

[7] M. Propst, V. Liebmann, J. Sieder-Katzmann, C. Bach, and M. Tajmar. Maximizing side force generation in aerospike nozzles for attitude and trajectory control. *69th International Astronautical Congress*, 2018. (Cited at page 8)

[8] Eric Besnard and John Garvey. Aerospike engines for nanosat and small launch vehicles. *Space 2004 Conference and Exhibit*, 28 - 30 September 2004. (Cited at page 8)

[9] Arca. `http://www.arcaspace.com/`. Accessed: 24-4-2019. (Cited at page 12)

[10] ESA. Investigation on the key issues for the realisation of an aerospike rocket engine. *ESA-GSTP-TECMPC-SOW-015959*, 8 October 2019. (Cited at page 13)

[11] J. Sieder-Katzmann, C. Bach, M. Propst, and M. Tajmar. Evaluation of the performance potential of aerodynamically thrust vectored aerospike nozzles. *67th International Astronautical Congress*, 2016. (Cited at page 14)

[12] C. Bach, S. Schöngarth, B. Bust, M. Propst, J. Sieder-Katzmann, and M. Tajmar. How to steer an aerospike. *69th International Astronautical Congress*, 2018. (Cited at page 14)

[13] N. M. Erni, S. A. Whitemore, and D. J. Baker. Closed-loop attitude control using fluid dynamic vectoring on an aerospike nozzle. *IREASE*, 2012. (Cited at page 14)

[14] T. et al. Bui. Flight research of an aerospike nozzle using high power solid rockets. *41st AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit*, 2005. (Cited at page 14)

[15] Assad Anis. Cold gas propulsion system – an ideal choice for remote sensing small satellites. *NED University of Engineering and Technology Pakistan.* (Cited at pages 17 e 31)

[16] Ames' Mission Design Division. *Small Spacecraft Technology State of the Art.* NASA, Moffet Field, California, 2015. (Cited at page 17)

[17] D. Tate Schappell, Pete Smith, and Nick Solway. Advances in marotta electric and small satellite propulsion fluid control activities. *41st AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit*, 2005. (Cited at page 19)

[18] D. Gibbon and Dr C. Underwood. Low cost butane propulsion systems for small spacecraft. *15th AIAA / USU Conference on Small Satellites*, 2001. (Cited at pages 19, 21 e 36)

[19] Dmc-1g (disaster monitoring constellation- first generation) missions. `https://directory.eoportal.org/web/eoportal/satellite-missions/d/dmc`. Accessed: 12-4-2019. (Cited at pages 19, 20 e 21)

[20] Grant Bonin, Niels Roth, Josh Armitage, and E. Robert Risi, Ben ad Zee. Canx-4 and CanX-5 precision formation flight: Mission accomplished! *29th Annual AIAA/USU, Conference on Small Satellites*, 2015. (Cited at page 22)

[21] R. S. Pauliukonis. Fuel system comprising sulfur hexafluoride and lithium containing fuel. *https://patents.google.com/patent/US3325318*. (Cited at page 22)

[22] G. Manzoni and Y. L. Brama. Cubesat micropropulsion characterization in leo. *29th Annual AIAA/USU Conference on Small Satellites*, 2015. (Cited at page 23)

[23] Urmas Kvell, Marit Puusepp, Franz Kaminski, Jaan-Eerik Past, Kristoffer Palmer, Tor-Arne Grönland, and Mart Noorma. Nanosatellite orbit control using mems cold gas thrusters. *Proceedings of the Estonian Academy of Sciences*, 2014. (Cited at pages 24 e 25)

[24] John Bown, Marco Vila, and Austin Williams. Cubesat based rendezvous, proximity operations, and docking in the cpod mission. *Tyvak Nano-Satellite Systems nc.*, 2015. (Cited at page 26)

[25] Joseph M. Cardin and J. Acosta. Design and test of an economical cold gas propulsion system. *14 th Annual AIAA/USU Conference on Small Satellites*. (Cited at pages 27 e 35)

[26] Markus Schelkle. The grace cold gas attitude and orbit control system. *3rd International Conference on Spacecraft Propulsion, Cannes*, 2000. (Cited at pages 28, 29, 35 e 37)

[27] George P. Sutton. *History of Liquid Propellant Rocket Engines*. American Institute of Aeronautics and Astronautics, Inc., Reston, Virginia, 2006. (Cited at page 30)

[28] Wilfried Ley, Klaus Wittmann, and Willy Hallmann. *Handbook of Space Technology*. John Wiley & Sons, Inc., 2009. (Cited at pages 31 e 37)

[29] Ames R. Farr and Maan H. Jawad. *Guidebook for the design of asme section VIII pressure vessels*. Asme, 2010. (Cited at page 32)

[30] Hugo Nguyen, Johan Köhler, and Lars Stenmark. The merits of cold gas micropropulsion in state-of-the-art space missions. *IAC-02-S.2.07*, 2002. (Cited at page 34)

[31] Mojtaba Ghasemi and Abdolrahim Rezaeiha. Flight qualification tests of a cold gas propulsion system for a small satellite. *Conference on Propulsion and Power*, 2012. (Cited at page 35)

[32] Dan R. Lev and Jacob Herscovitz. Carbon dioxide heated gas propulsion system for nano-satellites. *31th AIAA / USU Conference on Small Satellites*, 2017. (Cited at pages 36 e 37)

[33] Inert gas filter. `http://omnidea-rtg.de/site/index.php`. Accessed: 24-4-2019. (Cited at page 37)

[34] Mohammad Fatehi, Mehran Nosratollahi, Amirhossein Adami, and S.M.Hadi Taherzadeh. Designing space cold gas propulsion system using three methods: Genetic algorithms, simulated annealing and particle swarm. *International Journal of Computer Applications*, 2015. (Cited at page 52)

[35] Matt Vernacchia. Spike contour algorithm. *MIT RocketTeam - Pyralis Rocket Engine*, 2013. (Cited at pages 57 e 75)

[36] C.C. Lee. Fortran programs for plug nozzle design. *Prepared for the Advanced Propulsion Section, Propulsion and Mechanics Branch, P&VE Division of the George C Marshall Space Flight Center by Brown Engineering Company*, 1963. (Cited at pages 57, 58 e 59)

[37] J. Majdalani and B. A. Maickie. Explicit inversion of stodola's area-mach number equation. *Transactions of the ASME*, 2011. (Cited at pages 59, 61 e 62)

[38] Khalid Khan. Theoretical and numerical simulation analyses of the inviscid and the viscous flowfields of an aerospike nozzle. 2018. (Cited at page 61)

[39] Timothy S. Van Milligan. *Model Rocket Propulsion*. Apogee. (Cited at page 65)