POLITECNICO DI TORINO

Master of Science in Aerospace Engineering

Master Degree Thesis

Development of an intelligent controller using Genetic Programming

Supervisors
Prof. Giorgio Guglieri
Prof. Edmondo Minisci

Candidate
Nicola Menga

OCTOBER 2019

This thesis has been redacted in partial fulfilment of the requirements for the degree of Master of Science in Aerospace Engineering, in collaboration with the Intelligent Computational Engineering Laboratory (ICE Lab), Department of Mechanical and Aerospace Engineering, University of Strathclyde.

Development of an intelligent controller using Genetic Programming

Internal Supervisor
Prof. Giorgio Guglieri

External Supervisor
Prof. Edmondo Minisci

Candidate
Nicola Menga
S243871

To my parents, the best individuals
of each generation...

Abstract

Very complex systems require sophisticated controllers in order to achieve high performance. The demand for ever higher performance is putting the traditional controllers in crisis especially when there is a lack of information on the environment in which the system is operating or when the goals are not completely clear.
These needs can be met by Intelligent Control (IC), a class of control techniques whose development is recently increased thanks to modern processing capacity. Flexibility is the key for intelligent controllers in order to adapt to different scenarios. Therefore in this master thesis an intelligent control system was designed by using Genetic Programming (GP) algorithm in Python.
The capabilities of genetic programming to generate a proper controller structure are shown in off-line process, where disturbances are not taken into account and the control structure is not updated over time. In addition the behaviours of several controllers are compared in presence of disturbances in order to understand the robustness.
Then an intelligent adaptive architecture has been proposed. In this instance the update of the control structure is done by using genetic programming in on-line adaptation process when a disturbance occurs in the plant.
The idea of the suggested method has been tested on mass-spring-damper system and on a modified version of the standard Goddard problem. In these problems the genetic programming receives as inputs the state of the plant so that it generates the right values of control variables required in order to minimize the discrepancies over time between the current and desired values of the state variables.
The obtained results show the potentialities of genetic programming to adapt the structure of the controllers for several conditions in which plant parameters have been changed.

Contents

Acronyms

AC Automatic Control. 17, 18

AI Artificial Intelligence. 1, 17, 18

ARC NASA Ames Research Center. 21

ASE Autonomous Science-craft Experiment. 21

CASPER Continuous Activity Scheduling Planning Execution and Replanning. 21, 22

CGP Cartesian Genetic Programming. 6

DEAP Distributed Evolutionary Algorithms in Python. 33

EA Evolutionary Algorithm. 3, 25, 45

EO-1 Earth Observing One. 21, 22

GA Genetic Algorithm. 5, 23

GEP Gene Expression Programming. 5

GP Genetic Programming. 3, 4, 7, 8, 15, 20, 21, 25, 27, 31, 41, 43, 45, 49, 58, 62, 64, 66

HPC High Performance Computer. 67

IAE Integral Absolute Error. 29

IC Intelligent Control. 1, 17, 18

ISE Integral Squared Error. 29

ITAE Integral Time-weighted Absolute Error. 29

JPL Jet Propulsion Laboratory. 21

MAPE Mean Absolute Percentage Error. 55

MRAC Model Reference Adaptive Control. 22

MSD Mass-Spring-Damper. 25–27, 29, 30, 33, 45, 55, 56

OR Operations Research. 17, 18

SCL Space Command Language. 22

Chapter 1

Introduction

Many of the technologies in different fields use complex algorithms developed with Artificial Intelligence (AI). Artificial Intelligence means the discipline that deals with to provide computer models for the development of systems, both hardware and software, whose operating principle is based on human reason.
Recently the interest of the scientific community has been captured and for that reason a strong development of the field is started. Although the first work in artificial intelligence dates back in the early forties, the fame of Artificial Intelligence (AI) can only be noticed in 2010. This can be easily explained considering the different possibilities offered by new innovative technologies now on the market and they prove to be more and more efficient and cutting-edge, constituting the foundations of this discipline.
The fields of applications of artificial intelligence are considerable and range from medicine to agriculture up to engineering. Thus from the development of the theory of automatic controls a class of control techniques was born and it is called Intelligent Control (IC). It arises from the need to have systems capable of responding efficiently and at the same time reliable in cases where there is a lack of information on the control variables or the operating environment or when some characteristics of the system components vary according to non-deterministic mathematical models.
In the aerospace sector, controllers are partially responsible of increased aerospace system performance. And not only that, control systems together to other new technologies in the aerospace sector, will represent the key elements in defining not only the performances but also the reliability and the real costs of future aerospace systems.
In the space sector many agencies and private companies have already recently expressed real interest on it, not only for satellite, space stations and interplanetary probe controllers but also for the management and use of various tools used in them.
Regarding the control engineering, the substantial advantage of the use of IC during the design phase is that the design process would no longer be analytical but computational. For that reason these algorithms could be adapted and applied without particular efforts for the design of other controllers that have different requirements.
Therefore a sort of conflict and challenge between man and machine is born. Suffice it to say that currently in many fields the AI manages to carry out com-

plex problems in a much more accurate way and sometimes even in very short time compared to what men are capable of doing.

Chapter 2

Genetic programming

The Genetic Programming (GP) belongs to the category of Evolutionary Algorithm (EA), a branch of machine learning. It was born to satisfy one of the very first needs required by artificial intelligence to machines: to solve problems independently without the need for them to be explicitly programmed for a specific problem.
As can be guessed from the name, genetic programming is inspired by Darwinism, i.e. the theory of biological evolution.
A population, made up of different computer programs called individuals, evolves from generation to generation through genetic operators up to the determination of a program, often represented by the best individual, able of solving tasks more or less complex [16].
Just like nature, genetic programming is a completely random process [25]. If on one hand sometimes satisfactory results cannot be guaranteed through its use, on the other it turns out to be a very effective technique that allows us to overcome the limitations present in deterministic methodologies.

2.1 Preparatory steps

There are some main sets to be defined and initially required by genetic programming:

- terminal set
- function set
- fitness measure
- genetic programming parameters
- stopping criteria

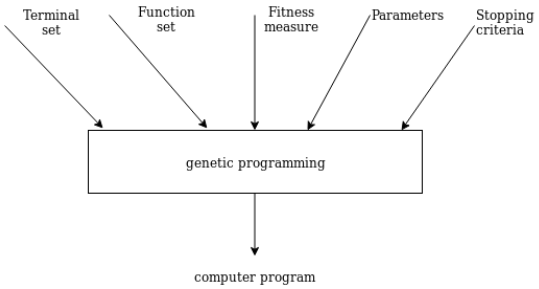


Figure 2.1: Preparatory steps as inputs of genetic programming [25]

As shown in figure 2.1, the main steps can be considered the inputs of genetic programming. Their definition is strictly dependent on the type of problem that one intends to solve and not on the type of evolutionary strategy that one wants to use.
The identification of the first two sets, respectively the terminal and the function set, automatically defines the size of the search space. The direction of the search is indicated by the third requisite required as input, the fitness measure. The last two, the GP parameters and the stopping criteria, regulate the evolution process.

2.1.1 Representation

Before analyzing each of previous steps, it is good to describe how a program created through Genetic Programming (GP) is represented.

2.1.1.1 Tree representation

There are different ways to represent a program and among these the tree syntax is one of the most used.
For example consider the following tree:

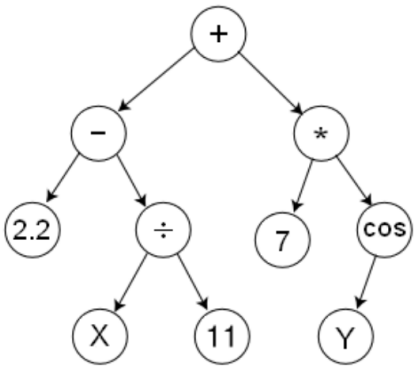


Figure 2.2: Syntax tree representing a program [11]

In the figure 2.2 the tree program is represented and it can be traced back

to the following mathematical function:

$$(2.2 - \frac{X}{11}) + (7 * cos(Y))$$

Notice that the structure of these trees is composed of nodes and leaves, in this example $(2.2, 11, 7, X, Y)$ are the leaves whilst $(+, -, \div, cos)$ are the nodes. Each program will be built according to this structure which has a node as its base, called the main root which acts as a glue between the various branches.

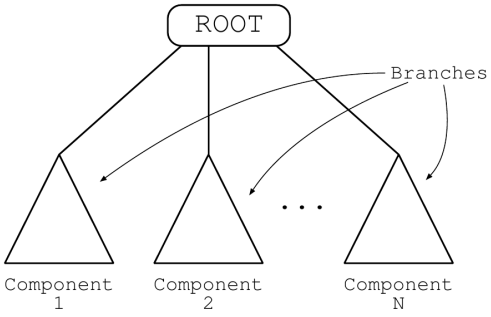


Figure 2.3: General structure of a parse tree [25]

A tree can develop both in height and in width. However, there are several limitations on these dimensions as one of the most well-known problematic phenomenon in genetic programming is bloat: the uncontrolled growth of the size of the trees.

2.1.1.2 Linear chromosome representation

In Gene Expression Programming (GEP) the programs share a particular aspect by GA, the programs are encoded in linear chromosomes of fixed length. Despite the fixed length of the string, the programs can be translated in trees with different sizes and shapes. A genotype/phenotype system is so created in GEP. For example, the linear string, representing the genotype:

$$\boxed{01234567}$$

can be transcoded in a visual form representing the phenotype as:

$$\boxed{Q * - + abcd}$$

where Q represents the square root function and a, b, c, d are the variables and the constants used in it. These linear strings are called k-expressions and this structural organization allows to the genetic operations to create always valid individuals.
A more detailed description of genetic expression programming can be found in [5].

2.1.1.3 Cartesian representation

Another form of genetic programming, the Cartesian Genetic Programming (CGP) uses a graph representation. Such representation is inspired by a method used in digital circuits. The genotype in this case is represented by a list of integers and the programs are represented in a grid of nodes, hence its name Cartesian.
For example a typical cartesian individual is defined as a set $\{G,n_i,\ n_o,n_n,\ F,\ n_f,n_r,n_c,l\}$, where:

- G is a set of integers representing the genotype
- n_i are the program inputs
- n_n are the node input connections and functions
- n_o are the program output connections
- F is a set in which the n_f functions are stored
- n_r and n_c are respectively the numbers of nodes in a row and column
- l is the level back parameter which defines the program interconnectivity and determines how many previous column of cells may have the outputs connected to a node in the current column

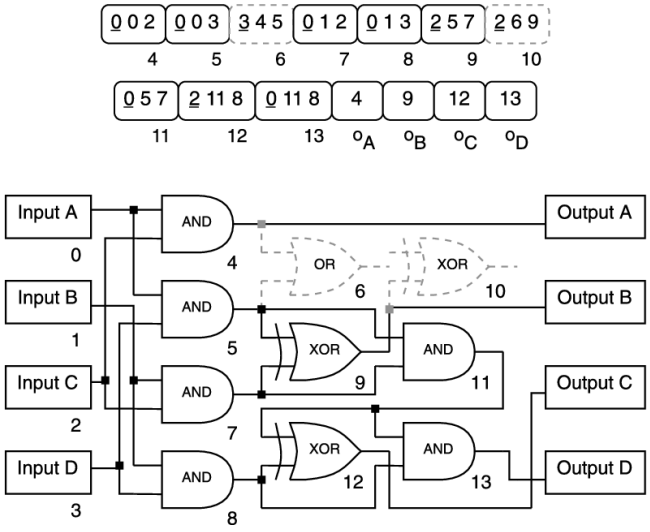


Figure 2.4: Genotype and phenotype of a cartesian genetic program [30]

The figure 2.4 shows the form of genotype and the corresponding phenotype of a program created by CGP.
More details on cartesian genetic programming can be found in [13].

2.1.2 Terminal set

The terminal set consists of:

- arguments
- constants

In the tree representation these terms constitute the leaves of the tree.
The arguments are the variables, i.e. the external inputs. The constants can be predefined or randomly generated.
A particular type of constants called ephemeral random constant have been created to solve the problem of constant creation according to [17]. Randomly created constants that are inserted and fixed in the initial individuals, without any change in value for the rest of the simulation.
In the example given in figure 2.2 the variables are (X, Y) while the constants used are $(2.2, 7, 11)$.
It is necessary to specify that unfortunately the definition of the constants is not always simple, in fact it is one of the main open issue og GP. The identification of the constants is difficult, especially in cases where there is no clear idea of the form of the final solution.

2.1.3 Function set

The function set is the second main ingredient for the creation of initial individuals. It consists of a set of arithmetic functions.
In the graphical representation the functions are the nodes of the tree. Like the terminal set, the choice of functions must primarily respect two requirements: closure and sufficiency.
The first one ensures that all functions give as output results all of the same type. It would indeed be impossible to mix both numbers and booleans in the same arithmetic function or even without considering an appropriate conversion system.
It should also be noted that during the simulation individuals interact with each other through genetic operators, so it is necessary to prevent future individuals from containing illegal genetic information.
The second requirement is that the definition of both the Functional set and the Terminal set is made taking into consideration that a possible solution to the problem can be expressed only through the elements contained in them.
It becomes appropriate to have a theoretical knowledge or at least an idea based on the experience of the problem, perhaps using other methods at disposal.
To satisfy the sufficiency requirement one could think of adding more and more terms, this does not slow down the search process but could lead to the achievement of a final result that is not completely satisfactory as expected.
The sum of the two sets is called Primitive set. Its good definition can be well acceptable if among the combinations of the elements that it contains there is a satisfactory solution to solve the problem.
Returning to the example tree in figure 2.2, the functions used are $(+, -, *, /, cos)$.

2.1.4 Fitness function

After having seen how to correctly define the Primitive set, it is now appropriate to show how the method is used to evaluate each individual present in a population.
So once we have defined the search space in which the solution is contained we do not know the position of the solution in this space. In order to point the search in the exact direction to that region, it is necessary to introduce a new concept linked to the numerical evaluation of an individual. The definition of the fitness function is of fundamental importance for the search for an optimal solution.
There are many ways to evaluate a program, just think that the same program could also be evaluated on more than one aspect, i.e. the Multi-objective optimization.
An individual who has a better fitness function than another is more likely to pass on his genes to later generations.

2.1.5 Other genetic programming parameters

The user must also define the control parameters inherent to genetic programming. They define the behavior of a simulation.
The most important parameters of the GP are the size of the generation and the population and the numerical values of probability that the user must assign to genetic operators.
There are no precise rules for optimizing these parameters but they are selected only after several simulations. The choice of parameters obviously must be performed taking into consideration the time that this technique requires. For example, a very big population size may require consequently a large evaluation time.

2.1.6 Stopping criteria

The last step is the definition of a criterion to terminate the search process.
In general the stop condition could be determined by choosing a tolerance value on the fitness function values which, once reached from generation to generation, is equivalent to having an acceptable solution for the problem. Otherwise another stopping criteria could be represented by the maximum number of generations that one wants to run, once you reach the last generation, the search is automatically completed. The solution is again represented by the individual with the best value of fitness function seen among all the generations.

2.2 Executional steps

Having thus defined the main ingredients in order to be able to perform a simulation, a series of steps are now introduced that are different for each problem. These steps are performed autonomously in fact generally there is no interaction of the human intervention during a simulation.

The executional steps of genetic programming are as follows:

- initialisation;
- iterative processes on a population by using genetic operations;
- end simulation with best element that meets the termination criteria.

The operations that are performed during the genetic programming algorithm are shown in figure 2.5:

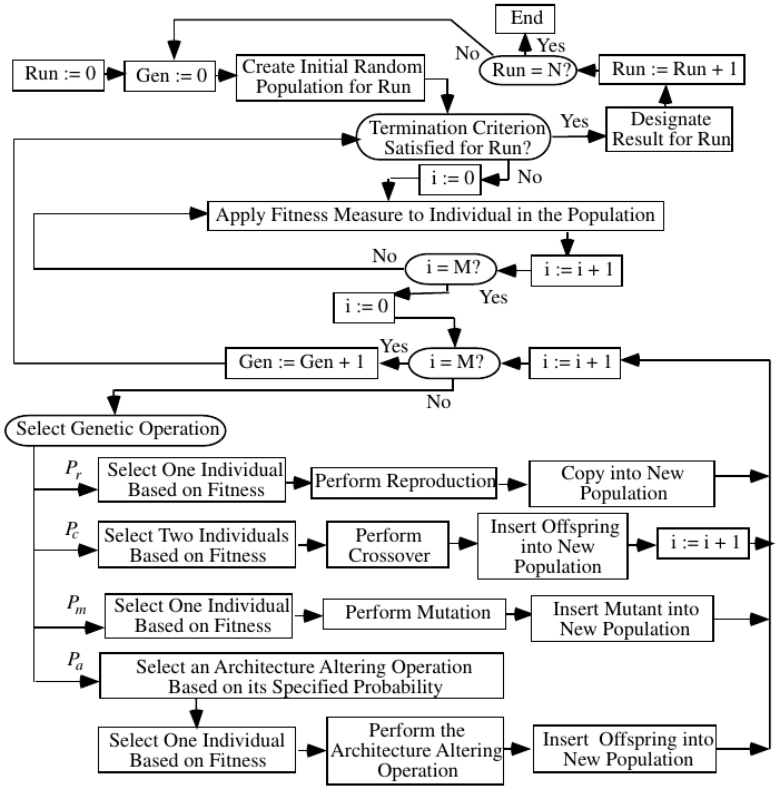


Figure 2.5: Genetic programming flowchart [15]

2.2.1 Initialisation

The first step is the initialization of the population. A series of individuals at generation 0 is randomly created by the algorithm.
Individuals are generated using the functions, constants and variables previously defined in the primitive set by the user.
As already mentioned, the creation of these initial individuals takes place in a completely random manner, so that the definition of the initial region, from which the search is carried out, will also be random [16].
The individuals present in generation 0 are individuals of fixed size through the parameters present in the fourth step of the preparatory step, they can all have the same size or have a variable size.
There are several methods for initializing individuals:

1. full: create trees all the same size

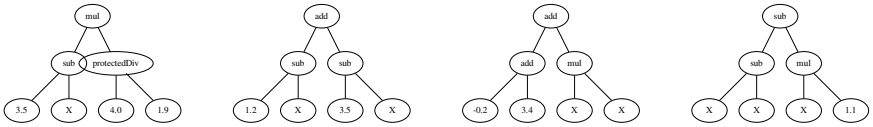


Figure 2.6: Full initialisation

2. grow: creates trees of varying size

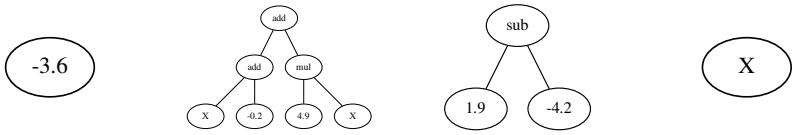


Figure 2.7: Grow initialisation

3. half and half: it creates one half of the trees all the same size and the other half a variable size

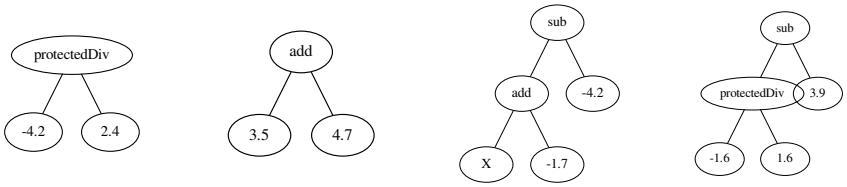


Figure 2.8: Half and half initialisation

All individuals, created in this phase, are characterized by not optimal fitness values and are therefore unable to represent a solution [25]. They are in fact primordial individuals, still very simple programs. Nevertheless, each individual has a different fitness value than another, so there will be better programs and at the same time worse than others. It is precisely on these differences that genetic programming bases its entire evolutionary process.
Thus the process of initializing individuals is of crucial importance for the rest of the search. Future generations will converge around the local minimum found during this very first phase.

2.2.2 Selection

As was already anticipated, the individuals are programs who are still too far away to be a solution. They need to undergo a process of evolution. The questions thus arise spontaneously: which individuals must be selected to pass on their genetic material?
The basis of this technique is the diversity among individuals which suggests that the selection method should not be very rigorous. Therefore, even if the selection is based on fitness function values, this does not mean that the worst individual should be completely discarded. In this way genetic information could be lost without being exploited, which could perhaps be important in populations of successive generations.
Just as deduced by Charles Darwin, the selection process is one of the key mechanisms of evolution. It is therefore partly thanks to it that there is a progressive increase in individuals with optimal characteristics, capable of solving tasks in the environment in which they live best.
According to [25], there are two main strategies developed for the process of selection of individuals within the population in mono-objective optimization such as:

- roulette wheel
- tournament selection

2.2.2.1 Roulette wheel

The selection process is based on the fitness value of individuals so an individual with better fitness is more likely to be chosen, according to [19].
The Roulette wheel method owes its name to the famous casino game.

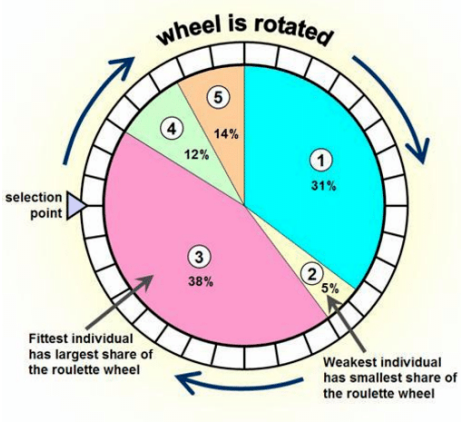


Figure 2.9: Roulette wheel selection method [2]

Imagine having a roulette wheel as shown in the figure 2.9. Notice its subdivision into parts, each characterized by a different angular opening proportional to the value that belongs to it.
Similarly, in genetic programming the roulette is divided into as many parts as individuals in a population and the breadth of the angular sector is always proportional to the fitness value attributed to each individual.
The area of each angular sector, which represents the probability of selection, can be expressed in mathematical terms such as:

$$p_i = \frac{ff_i}{\sum_{j=1}^{N} ff_j} \tag{2.1}$$

where N is the total number of individuals in a population and ff is the fitness evaluation. It is therefore seen that with this type of selection an element that has a worse fitness value can also survive in the next generation than a better one, since the probability of selection is never zero.

2.2.2.2 Tournament selection

Another method always based on the fitness value is called tournament selection and consists in selecting a group of individuals present within a population, selecting the best one. Consequently the most important parameter of this method is the tournament size that is the number of individuals selected to participate in the tournament.
Remember that the choice of individuals also in this method is handled randomly.
Very large tournament size values correspond to a decrease in diversity in later generations [3], as the probability to select the same individuals over and over again in a population grows proportionately with the increase in tournament size.

2.2.3 Operators

An essential aspect for evolution is certainly the variation of genetic information within individuals generation after generation.

After the evaluation of the fitness and the selection process, each individual selected is subject to one of the three main genetic operators:

- crossover
- mutation
- replication

2.2.3.1 Crossover

The crossover operator consists of combining genetic information from two different individuals, called parents, present in a population, to create new offspring.

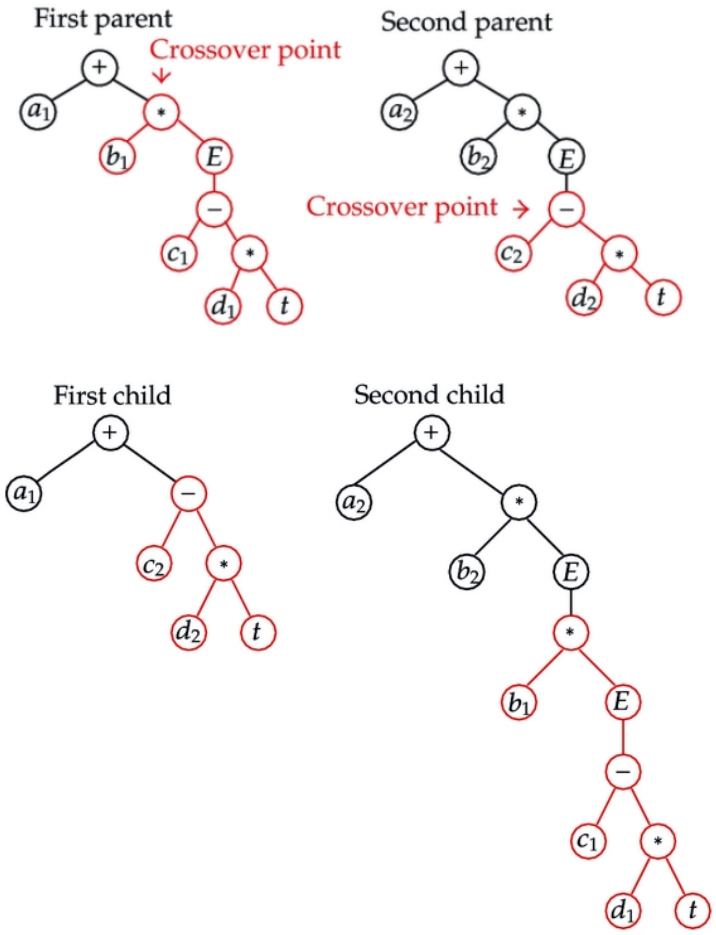


Figure 2.10: Parents and offspring [27]

In crossover operation two individuals are selected from the population and play the role of parents, as shown in figure 2.10. It is also important to notice that the length of their trees does not necessarily have to coincide, for that

reason it is generally very probable that the two parents have more often than not a different length. The first step that the crossover makes is the random selection on both individuals of a point, called a crossover point.

Subsequently the creation of the two individuals is automatically completed. The linking of the branch above the crossover point from the first generator and the branch below to the crossover point from the second parent forms a new individual offspring. The second is also created symmetrically, as shown in the figure 2.10.
It is an operation that tends to preserve the position of genetic material, operators who act in this way are called homologous as stated in [23].
So the basic idea of the crossover is precisely to create new and increasingly complex individuals through the exchange of mathematical blocks (genetic material) between them. Obviously the implementation requires special attention in order to create structurally valid individuals.

2.2.3.2 Mutation

Another genetic operator used in genetic programming is the mutation operator. Just as in the crossover a random point is identified within the structure of an individual. Subsequently, the sub-branch is completely replaced by another that is automatically generated randomly.
This operator allows to expand the search field in which to find the optimal solution.

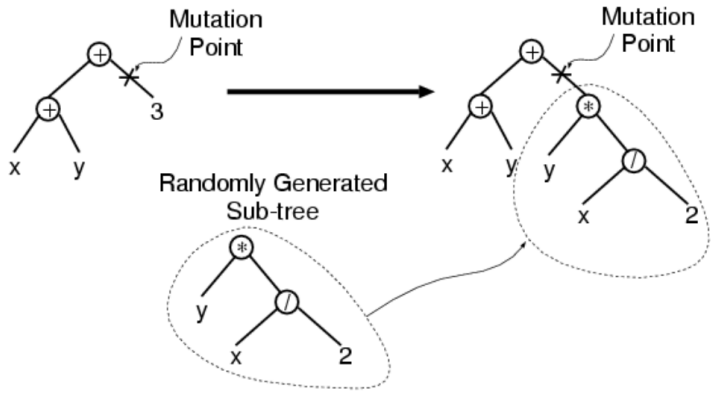


Figure 2.11: Mutation [24]

The generation of new tree structures is controlled by limits that regulate the maximum reachable depth.
Even though its useful the mutation operator is most often omitted in some problems.

2.2.3.3 Replication

Replication is another very important operator for the evolutionary process. It ensures stability in the successful of the search process and consists in the exact copy of the best individual in the next generation without it suffering any alteration.

It also called elitism and is attributed to the ability of the algorithm to memorize the best solution found on which part of the new individuals are created.

$$p_R = 1 - p_C - p_M \tag{2.2}$$

From the formula (2.2) the probability of replication of an individual in future generations is definitively obtained once that the probability parameters have been defined for the crossover and mutation operators.

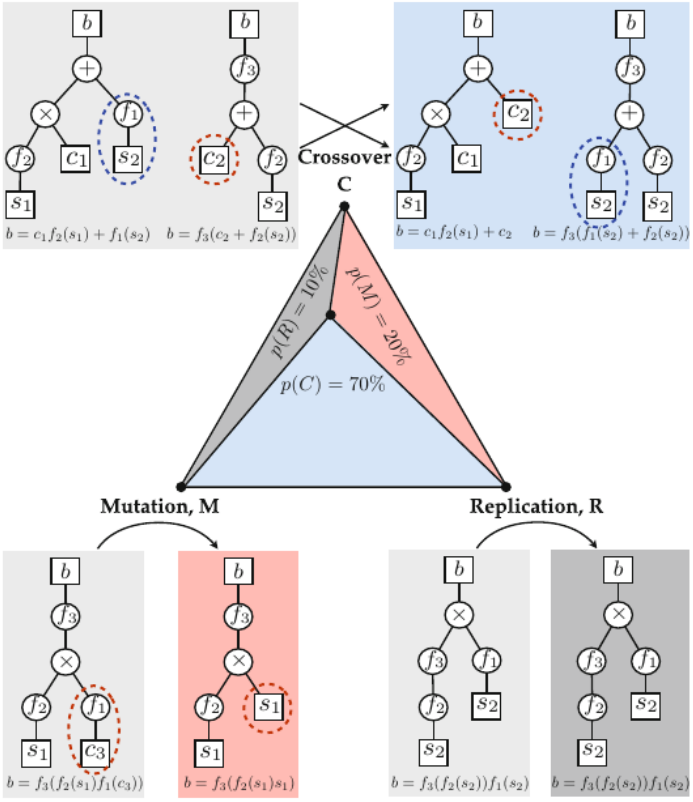


Figure 2.12: Common percentages of genetic operations in genetic programming [29]

2.3 Introns and bloat

Sometimes the performance of GP may be drastically reduced. The main cause is that it fails because of the formation of blocks that do not modify the fitness values. These blocks give the name of introns [22]. Particular attention should be paid to individuals who behave differently, even if they are different.
There is a close relationship between introns and bloat, as specified in [25]. The term bloat refers to the phenomenon whereby the size of the trees, representing the individuals, is rapidly growing without any improvement in terms of fitness.

Suffice it to say that a structural growth of individuals corresponds to a proportional growth of computational complexity, which represents a real problem for computers. Especially in controllers where the speed of response assumes a certain importance in time-dependent systems. In addition to the waste of computer resources, bloat is the cause of a double slowing down of the search process, as it is very difficult to modify the individuals affected by this phenomenon. The computation times to find the best solution thus increase exponentially when bloat occurs. Because of the derived problems, bloat has been a studied phenomenon over the past decades. Many techniques have been developed. The simplest technique consists in imposing a maximum limit on the height and depth reachable by the trees. Obviously the addition of these constraints must be done in such a way that the programs created are able to take a suitable form to solve the problem. It will then be essential to initially have an estimate of the minimum size attributable to the solution.

2.4 Convergence

The convergence is another phenomenon in evolutionary algorithms, as reported in [1] "The fundamental problem which many meta-heuristic optimization methods aim to counteract with various algorithmic tricks is the stagnation in a locally, but not globally optimal solution". Remembering that the key of evolution is the diversity to create a best individual who survives in the environment, the convergence problem consists on creating a generation in which all individuals are the same. It happens especially when the probability parameter of mutation is very low, so that the crossover tends to converge in a single solution.
There are some techniques to avoid the converge and to encourage the diversity as suggested in [25]:

- by using a small tournament size;
- by not using the reproduction operator;
- by splitting large populations into demes (sub-populations).

Chapter 3

Intelligent control

Today highly sophisticated controllers are required for very complex systems that require high performance. The demand for ever higher performance is putting the traditional controllers in crisis. So most of these needs can be met by Intelligent Control, the process of autonomous decision based on the interaction of the disciplines of Artificial Intelligence (AI), Operations Research (OR) and Automatic Control (AC).

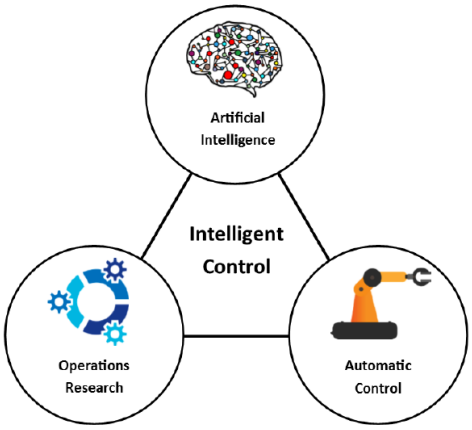


Figure 3.1: Multidisciplinary of intelligent control [26]

In 1993 the IEEE Control Systems Society gave a proper definition of an intelligent control system as described in the report [9]: "An intelligent control system is designed so that it can autonomously achieve a high level goal, while its components, control goals, plant models and control laws are not completely defined, either because they were not known at the design time or because they changed unexpectedly."
So, innovative control techniques, such as intelligent control result more appropriate for systems whose information is unknown or incompletely known.

3.1 Adaptive Intelligent Control

In order to define a model of a plant usually one has to define before two major steps which include the structural identification and the parameter identification. Therefore the interpretation of the physical laws and the acquisition of the values of the system's functional parameters represent the fundamentals steps to create a plant model.
As the main characteristics of an intelligent control systems is its ability to control a plant whose model are unknown, in adaptive control real time measurements of plant inputs and outputs are taken in order to develop a new plant model on which a new controller will be designed.
Obviously not all adaptive controllers are intelligent. One can be grouped in IC only when the control system is obtained by using methods of AI,OR and AC. However it is necessary to clarify that a controller designed for example through AI techniques is not necessarily intelligent because it must be able to adapt or learn online as well.

3.2 Learning Control

A system is called learning if the information that comes from environment or from other unknown resources is reused for new action commands. The learning is a key characteristic of the autonomous systems, thank to which the performance of the system will be improved.
The learning process can be applied in many control areas to enhance the performances, for example learning about the plant how the changes occur to derive new model plant or learning about the controller in order to tune control parameters.
It is important for learning process to have a memory where to store the information to be learnt, otherwise every time that a change in the plant occurs the system must adapt itself to the new operating conditions starting from scratch. Moreover the quality and the quantity of information to store depend by the designer.

3.3 On-line and Off-line Control

Three types of controls can be defined depending on the combination of use of adaptive and learning capabilities and it is possible distinguish if a system is characterized by on-line or off-line adaptation/learning and intelligence.

1. Off-line adaptation/learning without update:

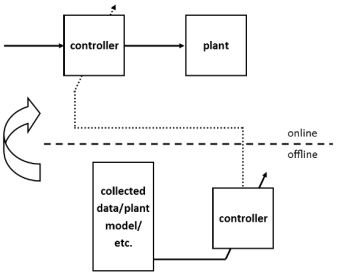


Figure 3.2: Off-line not intelligent control [26]

- the controls are computed, before the system is operated, by means of an artificial intelligent strategy
- the controls are not updated during the operation of the plant
- the system is neither adaptive nor learning

2. Off-line adaptation/learning with update:

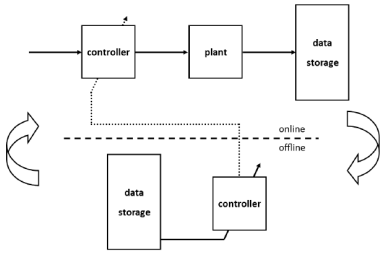


Figure 3.3: Off-line intelligent control [26]

- the controls are computed by means of an artificial intelligent strategy
- the controls are updated regularly off-line and uploaded to the plant for operation
- the system is either adaptive or learning

3. Online adaptation/learning:

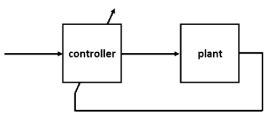


Figure 3.4: Online intelligent control [26]

- the controls are computed by means of an artificial intelligent strategy
- the controls are updated online during operations
- Adaptive or learning system

3.4 Intelligent control by using Genetic Programming

Real systems are often constrained, there are many restrictions so that a system can safely operate. These constraints can be soft or hard. Generally if a soft constraint is violated, it can lead to a decrease in general performance of the system. Instead the violation of hard constraints is different and sometimes it can involve some serious damage. The constraints usually are defined during the design phases. Sometimes can be very difficult to respect the constraints especially when is difficult to take in consideration all the external and internal factors that can interact with the system. Furthermore such difficulties can come for example from a lack of information on the control variable or on the environment in which the system will operate or even when it is not possible to predict the variation of some characteristics of the system over time.
A particular method to design an intelligent control by using GP is proposed. The main intention is to reconstruct the structure of the controller when a change occurs in plant parameters.
Both the off-line and on-line design approaches use genetic programming. Firstly the design for the controller is computed by using specific design plant parameters, then if a variation in the plant occurs, new controller structures are redesigned and the old laws are replaced.
The architecture is presented in the following figure:

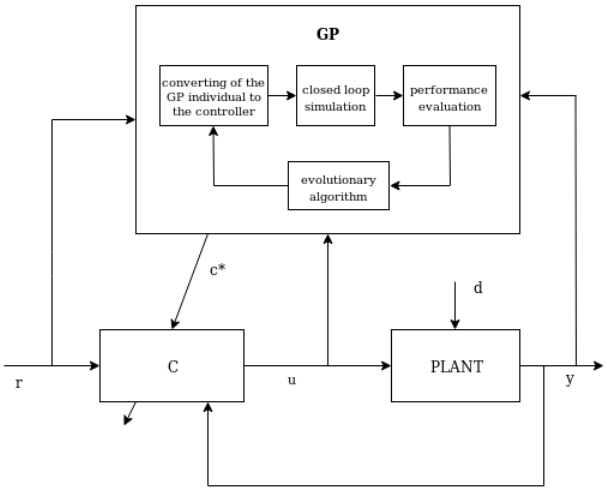


Figure 3.5: Intelligent adaptive control architecture with genetic programming technique

As already mentioned a disturbance in plant d starts the genetic programming algorithm. Receiving as inputs, the reference model r and the outputs

from the plant y, it generates the best controller $c*$ that will be more suitable to work with the new plant parameters. Then the old controller C, designed also by genetic programming, will be replaced by the new more efficient controller. Currently the main drawback of this architecture is the high computational cost. For that reason it can be more easily applied in processes whose characteristic times are long. The implementation of this architecture in processes with low characteristic times can be a hazard because while the evolution algorithm is operating, the system continues to work with the changed plant parameters and if a valid controller is not given in time, the entire system could be severely damaged. Therefore in order to accelerate the evolution process, the GP search of optimal controller starts with an initial population whose individuals belong to the off-line controller design.

3.5 Applications

3.5.1 Aerospace application

One of the most intelligent control application in aerospace field is the autonomous flight software used in Earth Observing One (EO-1) as classified in this taxonomy paper [4].
The Autonomous Science-craft Experiment (ASE) consists in a package of autonomy software developed by NASA Ames Research Center (ARC) and Jet Propulsion Laboratory (JPL). The main goal of such intelligent control system is to construct a series of hierarchical instructions to be executed by the spacecraft, as shown in figure 3.6:

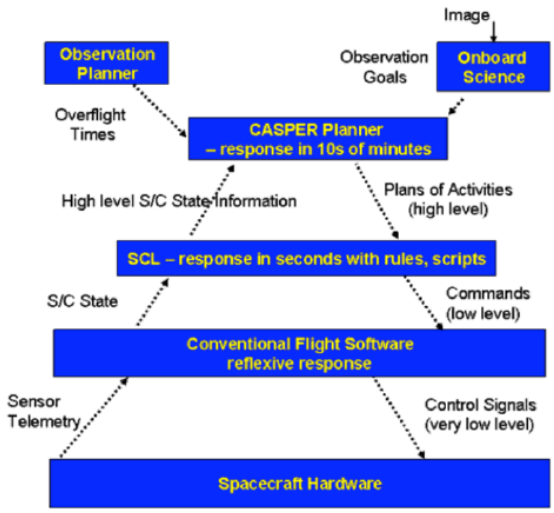


Figure 3.6: Software layered architecture [6]

Interesting events are detected by the images taken by the scientific instrumentation on board the spacecraft. The Continuous Activity Scheduling Planning Execution and Replanning (CASPER) planner represents the high level in the control and creates operation plans in order to enhance the efficiency of acquiring scientific data, as demonstrated in figure 3.7:

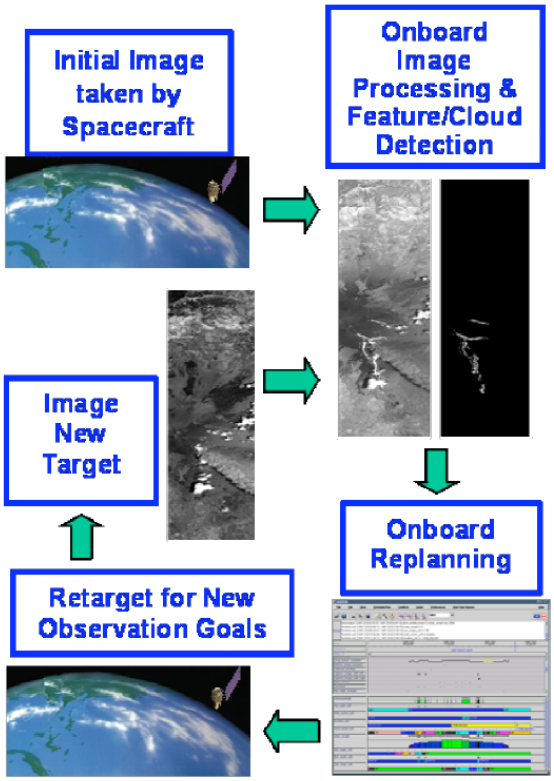


Figure 3.7: Autonomous science applications [6]

Moreover CASPER receives as inputs not only scientific data but even engineering information in order to reach the correct spacecraft attitude.
Then the high level scheduled activities are passed to Space Command Language (SCL) executive system. It consists in scripts and rules for execution of the high level commands previously created by using the low-level actions.
These commands created by SCL can be adapted to environmental changes and can alter the parameters of the controller on-line by taking autonomous decisions.
The EO-1 spacecraft demonstrated for the first time the feasibility of an autonomous science mission in which the spacecraft was able to plan and schedule processing autonomously.

3.5.2 Non-Aerospace application

A very interesting medical application comes from the design of a bio cybernetic system by Ivan Sekaj from Slovak University of Technology [14].
In particular an optimisation method was proposed for parameter setting of a MRAC of blood glucose in the human body.
There are several theories and mathematical model for artificial pancreas, but each patient needs a different parameter setting. "Conventional identification and controller design methods based on analytical approaches are often not able

to ensure good quality of the model and of the controller" [14], for that reason an approach based on GA has been argued in this work.
Diabetes mellitus type 1 is one of the most frequent contemporary civilization diseases with significant contribution to human mortality and its occurrence is still growing.

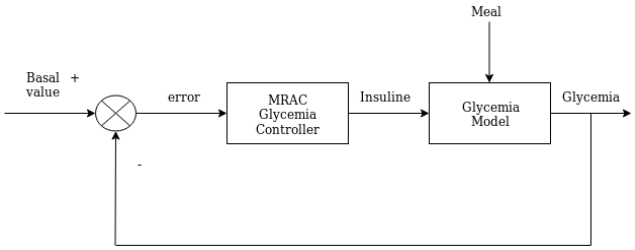


Figure 3.8: Feedback closed loop for glycemia control [14]

As shown in figure 3.8 the feedback closed loop is presented in order to take the level of sugar in blood constant within in precise range. Obviously the main disturbance of plant, representing by the glycemia, is the taking meal.
The goal of this method is reduce the error between the value of glycemia and basal value. The basal value is individual in each patient and depends by the age and other many individual factors.

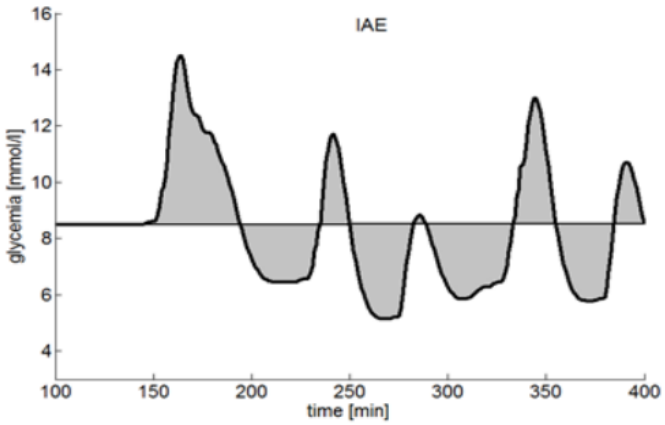


Figure 3.9: Glycemia control after disturbances [14]

The graph 3.9 shows the trend of glycemia during the time. It can be observed that at certain time there are some disturbances that lead to an increasing of glycemia from reference level. The grey areas show the error that must be reduced.
The role of the controller is to administer insulin to control the glycemia level just when the disturbance occurs and the error increases. To do this genetic algorithm is applied. Simplify the argument supposing that the output controller is:

$$u(t) = f(parameters) \tag{3.1}$$

So during the evolutionary search process, chromosomes are defined as vectors which contain a certain number of coefficient parameters inside:

$$chromosome = \{\Gamma_1, \Gamma_2, \Gamma_3, \gamma_1, \gamma_2\} \tag{3.2}$$

In this instance the goal of evolutionary algorithm is to find the optimal values for these coefficients and so to seek the best vector of values which minimize the error during the time.
A four-days test showed that the adaptive control improved the control of the glycemia and resulted more accurate to respond to a disturbance compared to manual setting.
The presented method described an intelligent system whose controller can adapt its output when a variation of the plant conditions is caused by disturbances. The glycemia trend varies depending on the type of food assumed by the patient. So, the goal of the adaptive controller is that to find the right parameters to inject the correct quantity of insulin whenever a meal is assumed in order to achieve and stabilize the glycemia in a certain range of values.

Chapter 4

Test case: mass-spring-damper system

This section deals with two major contents: the implementation of genetic programming to design an off-line controller structure for a Mass-Spring-Damper (MSD) system first and then an intelligent method will be proposed regarding how genetic programming can be also used to re-calculate the control structure when the operating conditions change at a certain time point.
As first approach to evolutionary algorithms and particularly to the genetic programming, there are some reasons because MSD was chosen as test case for a first approach to EA and particularly GP:

- it is a very common problem to solve simply made up of few parameters (three);
- the linear dynamic equation of equilibrium, neglecting non linear phenomena like friction, is well known and easily mathematically derivable;
- it is a model that is well suited for modelling other objects with complex properties.

4.1 Dynamic model

Mass-Spring-Damper (MSD) system is a linear problem with second order dynamic ordinary differential equation. Such system is composed by a mass moved by an external force, a spring force and a damping force.

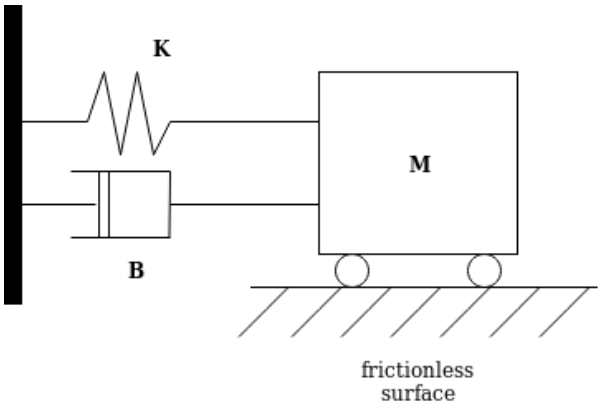


Figure 4.1: Mass-spring-damper system

To simplify the model, let's assume that the mass can move only toward a direction on the plane without taking in consideration the friction between the wheels and the surface.
Furthermore the spring is ideal and its stiffness is assumed as constant.
The governing equation of the system is derived by using the Newton's second law of motion:

$$M \cdot \ddot{x}(t) + B \cdot \dot{x}(t) + K \cdot x(t) = u(t) \tag{4.1}$$

where M is the mass, B the damping coefficient, K the stiffness coefficient and $u(t)$ is the external force acting on the mass.

4.2 Controller design

As already mentioned, the proposed design of controller is applied on a MSD system.
The goal is achieved if the mass position will follow the set point command for a time interval. The set point command is the desired value for the mass position.
The block scheme of the closed loop is shown in figure 4.2:

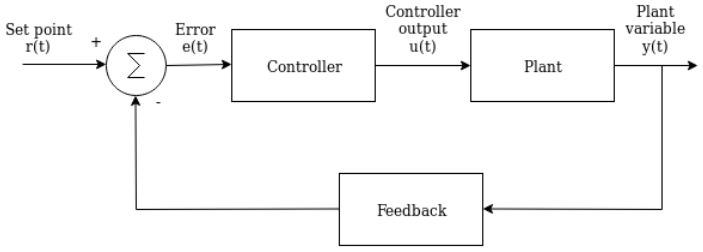


Figure 4.2: Block scheme of the closed loop

Starting with initial conditions of the plant, the controller must be able to reduce the error in each time step. The controller input is the error computed

as the difference between the set point value and the real system value:

$$e(t) = r(t) - y(t) \tag{4.2}$$

Then the controller is in charge to produce as output a value in terms of force inside the plant. Therefore the controller output $u(t)$ is seen by the MSD system as an external force acting on the mass.
According to [29] generally if the structure of a controller is already defined and if the task is only to optimize the parameters, genetic algorithm can be used but if the form of the controller is unknown, a possible approach to define its structure is the genetic programming.

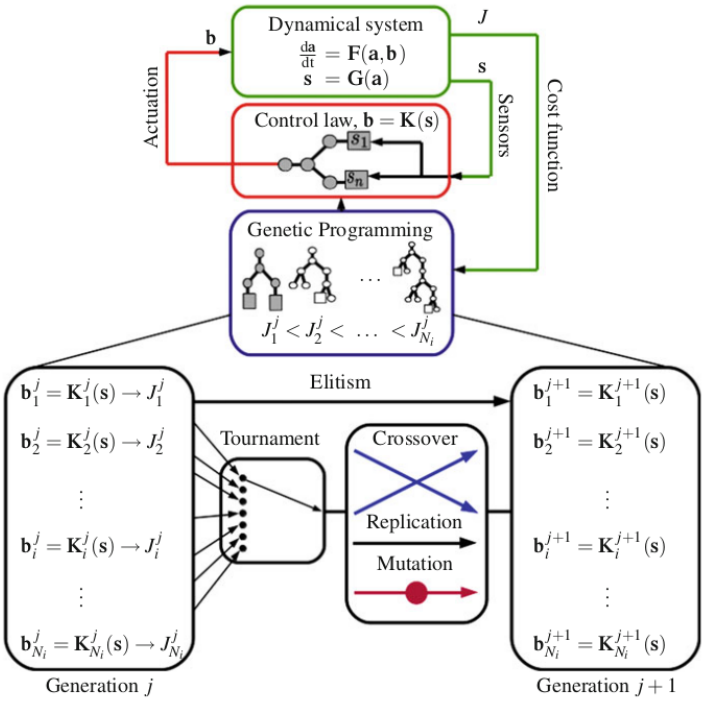


Figure 4.3: Control design using GP [29]

The figure 4.3 shows the concept of control design by using GP.
During the process every individual, representing a possible candidate for being the controller law, is evaluated by the dynamical system.
The evolutionary strategy is based on the phases described in the following pseudo-code:

Algorithm 1 Genetic programming

Input: POPsize, Pcrossover, Pmutation, nodes and leaves
Output: Sbest

```
1: procedure GENETIC PROGRAMMING
2:   Initialize the population
3:   Evaluate the population
4:   while stop condition() do
5:     Selection of individual
6:     Individual modification by genetic operators
7:     Build next generation
8:   return Sbest
```

At the end of the process, the individual with the minimum value of fitness is selected as the best individual and it can be used as controller for the system.

4.3 Genetic programming settings

In this section the main genetic programming configuration, that has been used in the simulations for the mass-spring-damper system, is described.

4.3.1 Primitive set

The variables in genetic programming are leaves in the tree representation of a classic program. The following set of variables has been used:

$$variables = \{err, d_{err}, i_{err}\}$$

where they respectively are the error, the derivative and the integral term of the error.
Concerning the function, they are the nodes in three representation. The functions are basically the arithmetic operations used to create the program. The set of functions that have been used is:

$$functions = \{Add, Sub, Mul, Abs, Log, Tanh, Sqrt\}$$

where they respectively are addition, subtraction, multiplication, absolute value, logarithm, hyperbolic tangent and square root operators. It is noted that these operators receive a different number as argument which in logic and computer science it is called the arity of a function. The specification of this number is very important to create a valid syntax for the trees both during the initial creation of the individuals and during the genetic operations.
The set of numeric constants used is:

$$constants = \{pi, e, rand1, rand2, rand3, rand4\}$$

where the first one is the pi constant, the second term is the Napier constant and the remaining terms are random values called ephemeral random constants as defined:

- rand101, rand102 in the range from -100 to 100 (decimals)

- rand103, rand104 in the range from 0 to 9 (integers)

The value of an ephemeral constant is a random value picked within a specific range and, once it is inserted inside a three, this value never changes unless it is replaced by another ephemeral constant.
In the genetic programming the evolution of constants is still an opened issue. The main issue is the difficulty to generate the right constants for a specific problem whose the structure of the solution is not completely clear and at the same time the process of optimization for the constants seems to be very tricky during the evolution process, Koza argues in [18] that "the finding of numeric constants is a skeleton in the GP closet".
The definition of the elements inside each set defines the search space and it is fundamental for the creation of programs and for the general attitude and performance of genetic programming optimization.

4.3.2 Fitness function

As already discussed before, the fitness function indicates how an individual is appropriate to solve a specific problem. It represents the high level statement whereby the programs are evaluated.
It is necessary to evaluate the closed loop performance in MSD case. In literature there are several performance index that can be used as fitness function.
The three commonly used measures are defined in [28] as:

$$ISE = \int err^2 dt \tag{4.3}$$

$$IAE = \int |err| dt \tag{4.4}$$

$$ITAE = \int t \cdot |err| dt \tag{4.5}$$

respectively the Integral Squared Error (ISE), the Integral Absolute Error (IAE) and the Integral Time-weighted Absolute Error (ITAE).
In this case the IAE performance index has been chosen because with respect to the others it doesn't use additional weights to the error and its answer tends to manifest a less number of oscillations, as written in [20].
The discrete time form is:

$$IAE = \sum_{n=1}^{N} T_{N,n} |err_n| \tag{4.6}$$

where $T_{N,n}$ is the simulation step size, n is the simulation step ans N is the number of simulation steps.
Since it is also required to damp the oscillations, the derivative term was added inside the performance index.
The final form is:

$$J_{IAE} = \sum_{n=1}^{N} T_{N,n} (|err_n| + \alpha |d_{err_n}|) \tag{4.7}$$

the α term is a constant that can be adjust experimentally. According to [14], to increase α term in equation 4.7 equals to increase the oscillation dumping.

Chapter 5

Aerospace case: Goddard problem

After the methodology has been well tested on MSD test case, a well-known aerospace case has been adopted to implement a similar control approach on it. The Goddard problem has been proposed for the first time by Robert Goddard in 1919. It concerns the ascent of a rocket in atmosphere and its main goal is to reach the highest altitude as possible, considering the imposition of some constraints like on the maximum fuel consumption.

5.1 Dynamic model

A varied version of the standard Goddard problem has been adopted whose dynamical model of the rocket can be expressed by the following ordinary differential equation:

$$\begin{cases} \dot{r} = v_r \\ \omega = \dot{\theta} = \frac{v_t}{r} \\ \dot{v_r} = \frac{(T_r - D_r)}{m} - g + \frac{v_t^2}{r} \\ \dot{v_t} = \frac{(T_t - D_t)}{m} - \frac{(v_r \cdot v_t)}{r} \\ \dot{m} = -\frac{\sqrt{T_r^2 + T_t^2}}{c} \end{cases} \tag{5.1}$$

where r is the altitude from the center of the Earth, v_r is the radial velocity and v_t is the tangential velocity measured respect to the center of Earth, ω is the angular velocity, θ is the angular displacement, m is the mass of the rocket and c is the exhaust speed along the axis of the engine that is possible to define as:

$$c = g_0 I_{sp}$$

where g_0 is the standard acceleration due to gravity and I_{sp} is the specific impulse.
Notice that the gravitation acceleration g which appears in equation 5.1 is defined as:

$$g = g_0 (\frac{R_e}{r})^2$$

where R_e is the Earth radius and r is the current altitude.
The terms D_r and D_t represent the drag forces respectively for the radial and tangential directions and that assume the following formulas:

$$\begin{aligned} D_r &= \frac{1}{2}\rho C_d v_r \sqrt{v_r^2 + v_t^2} S \\ D_t &= \frac{1}{2}\rho C_d v_t \sqrt{v_r^2 + v_t^2} S \end{aligned} \tag{5.2}$$

in which ρ is the air density, C_d is the aerodynamic drag coefficient and S is the reference area.
The T_r and T_t terms are the inputs of the system, i.e. the throttles.
Finally it is possible to define a vector containing the five states defined as:

$$\{x\} = \{r, \theta, v_r, v_t, m\}$$

5.2 Controller design

The design approach follows that shown previously in figure 4.3 but a different application in GP methodology is proposed in this instance.
The main intention is to create two controller laws at the same time both for the radial and tangential throttles.
The tree syntax representation is used as before but now as already mentioned the outputs of the evolution process is represented by two sub-trees that describe the two control laws needed for this aerospace problem.

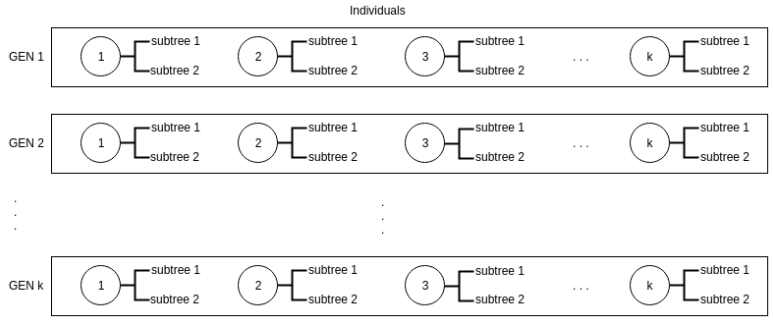


Figure 5.1: Multi-output genetic programming

As shown in the above graph 5.1, a multiple output program tree structure, as presented in [32], has been adopted to minimize the errors between the states of the real system and the references. The reference terms have been found through a previous optimization work in order to achieve the optimal trajectory. As before the reference or the set-point signal represents the optimal trend of the states throughout the time.

5.3 Genetic programming settings

In this section the main genetic programming configuration, that has been used in the simulations for the Goddard problem, is described.

5.3.1 Primitive set

The set of variables to take in consideration is slightly larger then the previous one due to the complexity introduced by this aerospace case:

$$variables = \{errR, errTheta, errVr, errVt, errM\}$$

these errors are computed as the discrepancies of states from the reference values (R, Θ, V_r, V_t, M) during the time:

$$e_R(t) = R(t) - r(t)$$

$$e_\Theta(t) = \Theta(t) - \theta(t)$$

$$e_{Vr}(t) = V_r(t) - v_r(t)$$

$$e_{Vt}(t) = V_t(t) - v_t(t)$$

$$e_M(t) = M(t) - m(t)$$

The functions set is composed:

$$functions = \{Add, Sub, Mul, TriAdd, Abs, Tanh, Sqrt\}$$

where the $TriAdd$ function returns the sum of three addends.
Notice that the *Log* function has been deleted from this set because this time inside the terminal set there are constants with high values and overflow phenomenon could occur during the evolution process.
So the set of numeric constants has been defined with ephemeral constants:

$$constants = \{rand1, rand2, rand3, rand4\}$$

where:

- rand1,rand2,rand3,rand4 in the range from -500 to 500 (decimals)

5.3.2 Fitness function

Basically the definition of the fitness function is the same of that used previously but since the number of control variables is more than one, the multi-objective optimization has been adopted. Therefore the selection of the individuals in a population must considerate now more than one objective functions at the same time, as proposed in [10]. In our case five fitness functions are minimized simultaneously:

$$min(ff_r, ff_\theta, ff_{v_r}, ff_{v_t}, ff_m) \tag{5.3}$$

The selection of individuals is more complex this time because more objectives have to be considered. For that reason the selection of individuals has done through Pareto set. It is a method frequently used in multi-objective optimization problems.
Nowadays there are no methods to improve at the same time all objective functions at the same time for every single step in the optimization process. Therefore the Pareto efficiency comes from the impossibility to enhance an object function, harming seriously at the same time an another one (more information in Appendix A).

Chapter 6

Results

In this chapter the results to design the structure of the controller will be shown in a first moment both for the MSD system and Goddard problem. All simulations have been developed by using Distributed Evolutionary Algorithms in Python (DEAP) library (see Appendix B).
The results in the first part of this chapter refer only to off-line design and therefore no change in parameters of plant has been taken into account. The control law is fixed during the total time of the simulation and there will not be the necessity to be updated.
Instead in the second part of this chapter some changes of the plant parameters will be considered during the time simulation. In this case some constraints have been added into the system and if the error of the control variable to be controlled is too high from the desired value in a such way that it could violate the constrains, the update of the controller law will be necessarily required.

6.1 Off-line design control for mass-spring-damper system

6.1.1 Genetic programming design results by varying set point command

Different set point commands have been used as input signals to the system to show the capacity by which genetic programming develops the programs for the controller on its own.
Classic set point signals in control engineering have been used:

- Sine wave
- Square wave
- Multi-step

For these first initial test, the genetic programming parameters have been set up in the same way for a fair comparison.

Objective:	Find a program whose solution matches signal input command in time interval [0,20] seconds
Terminal set:	$[err, d_{err}, i_{err}, R, pi, e]$, with R four ephemeral constants
Function set:	$[+, -, *, Log, Sqrt, Tanh, Abs]$
Fitness measure:	IAE
Selection:	Tournament (size=3)
Initialization pop:	Full
Parameters:	pop size 200, gen size 100, crossover 60% , mutation 30%
Termination:	generation size

Table 6.1: Genetic programming settings for MSD test cases

The values of plant parameters, previously described in figure 4.1 and now taken as initial parameters to design the control laws are:

- M=1 kg
- B=10 Ns/m
- K=20 N/m

6.1.1.1 Sine wave signal

The sine wave command is a continuous wave and its form is a function of time:

$$y(t) = r(t) = A \cdot sin(\omega t) \tag{6.1}$$

where A is the amplitude and ω the angular frequency, in this case $A = 1$ and $\omega = 1$ rad/s.

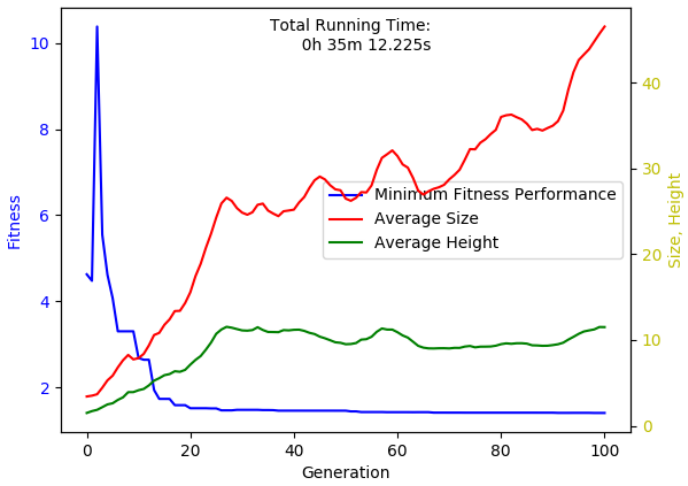


Figure 6.1: Genetic programming statistics during the generation evolution for sine wave signal

Figure 6.1 shows the trend of fitness function and the development on size and shape of the trees throughout the generations.
It can be observed that for the initial generations, as already said previously, the values of fitness is very high and then it tends to decrease until it stabilizes to a value of 1.4096 in final generation. The lower the fitness value, the lower the error is and the performance of system will be more accurate.
Obviously the size and the height of the trees increase during the optimization process, so the best individual has the following mathematical form:

$$\begin{aligned} &Mul(err, Add(Sub(Sub(Sqrt(-35.5586), Tanh(Sub(Sqrt(i_err), Sqrt(i_err))\\ &)), Sub(Log(96.0902), Abs(Sub(Sub(Add(Log(Mul(Mul(Add\\ &(pi, 41.9094), Mul(9, i_err)), -49.8909)), Abs(-45.0665)), Log(d_err)\\ &), Sub(Abs(d_err), Abs(Mul(96.0902, Abs(Log(Mul(err, err))))))\\ &))))), Mul(96.0902, Abs(Log(Mul(err, err)))))) \end{aligned} \tag{6.2}$$

It is possible to turn this long equation in the tree representation:

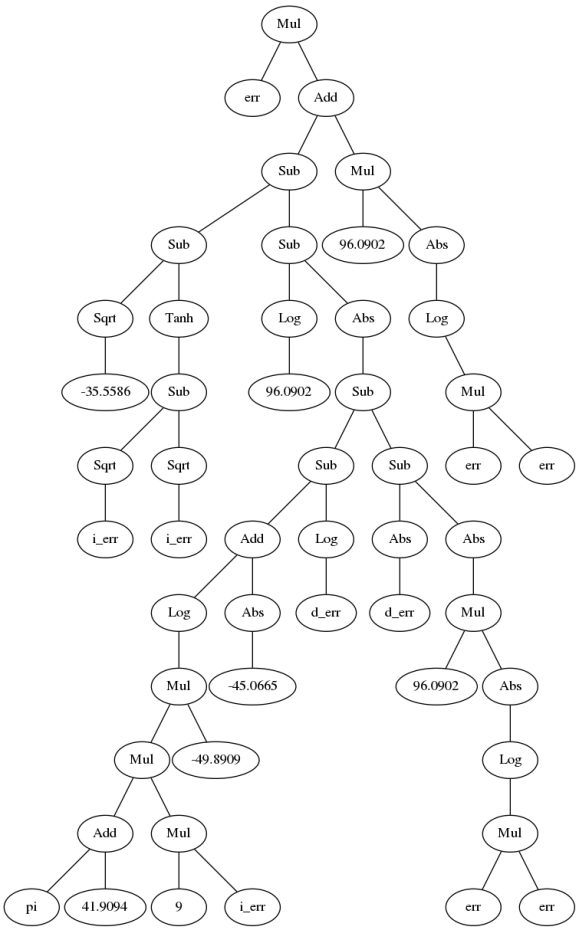


Figure 6.2: Best individual for sine wave signal

From the figure 6.1 it can be noted the total time taken by optimization process with genetic programming, it is approximately 35 minutes with the parameters shown in table 6.1. The time taken by design the structure of the controller isn't a problem because it is done before that the system is running. In the following figure 6.3 the mass position has been compared with the sine wave command:

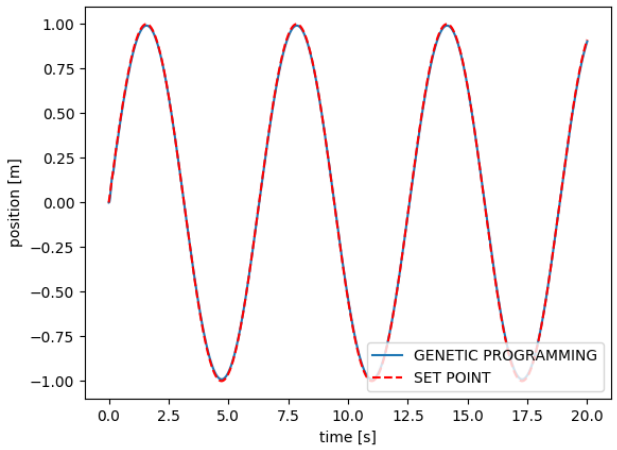


Figure 6.3: System position and set point (sine wave) command during the time

The position of the mass seems to follow perfectly the set point but obviously the error between the real position system and the reference will never be equal to zero. It is possible to observe that in figure 6.4:

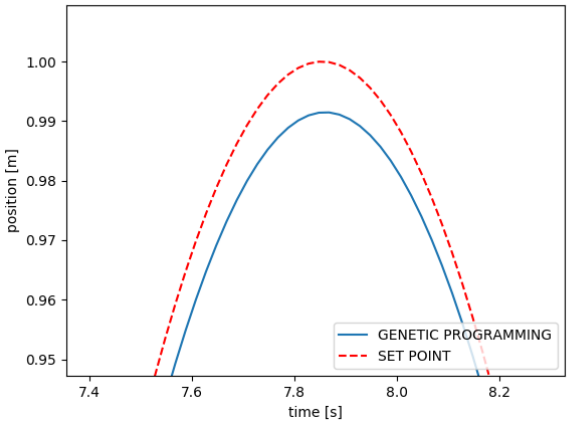


Figure 6.4: Error between the current position system and the commanded position

6.1.1.2 Square wave signal

There are more than one possibility to describe a square wave. It can be described as the sign function of a sinusoid:

$$y(t) = r(t) = sgn(sin(\omega t)) \tag{6.3}$$

where $\omega = 2\pi f$, f is the frequency of the square wave, in this case $\omega = 1$ rad/s. The maximum and the minimum values in the square wave signal are respectively 1, -1 and 0 in the discontinuities.

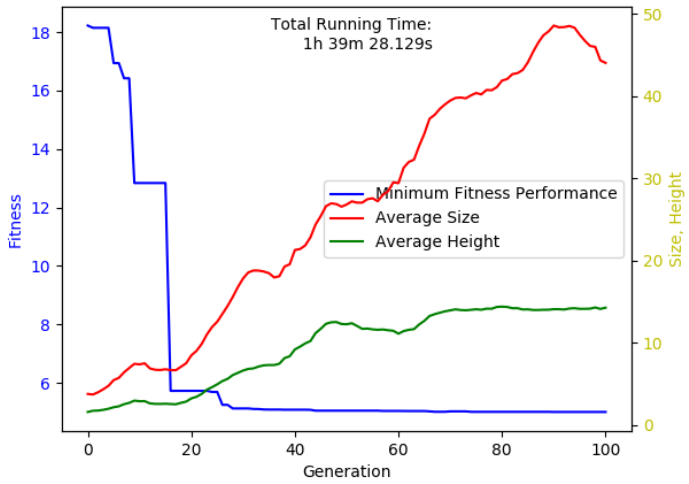


Figure 6.5: Genetic programming statistics during the generation evolution for square wave signal

The trend of the fitness function and the shape of threes are equal to the previous case but the time spent by the optimization process is almost tripled. The mathematical expression of the best individual is bigger according to the figure 6.5 :

$$\begin{aligned} &Mul(Tanh(Mul(Sqrt(Sqrt(Sub(7, Mul(8, Mul(8, Mul(9, err)))))), Tanh(Tanh \\ &\quad (Mul(Sqrt(95.5874), Tanh(Tanh(Mul(Add(9, 7), Mul(9, Mul(Sqrt(Add \\ &\quad (Mul(8, Mul(9, err)), Add(e, e))), Tanh(Tanh(err)))))))))), Mul(8, 5)) \end{aligned} \tag{6.4}$$

It is less complex if compared to the previous controller law expression found for the sinusoidal signal, consequently the relative tree is slightly smaller:

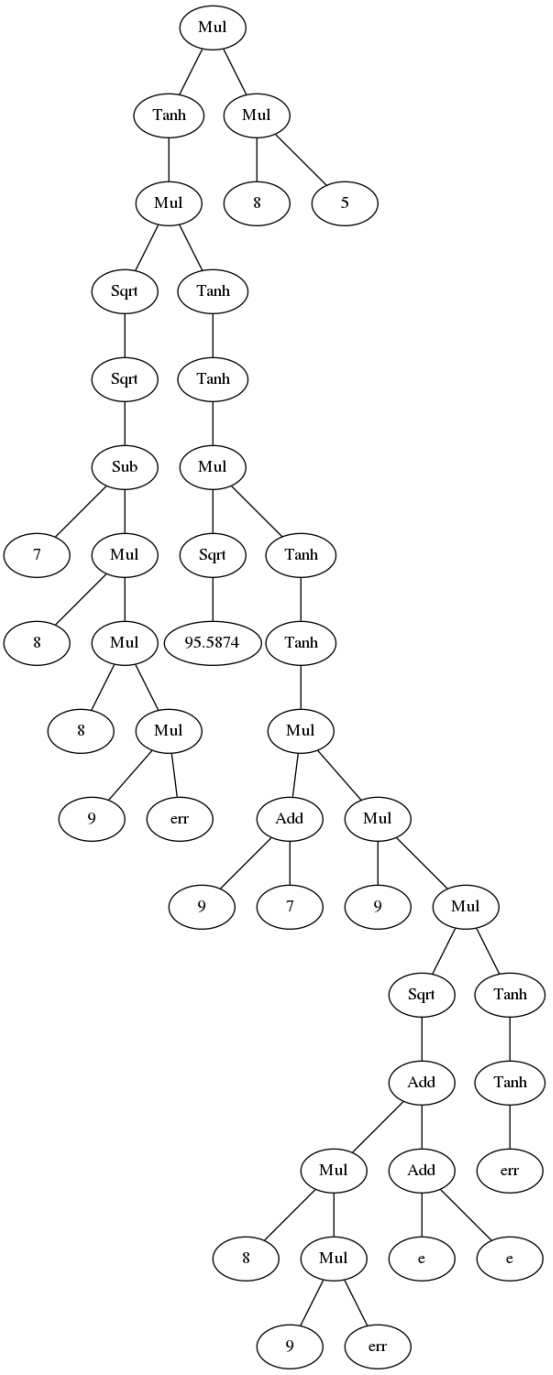


Figure 6.6: Best individual for square wave signal

With respect to the sinusoidal command where the error was very small for the total time of the simulation, the error with a square wave signal is not always small, especially when the variation of desired position occurs instantaneously as can be observed in figure 6.7:

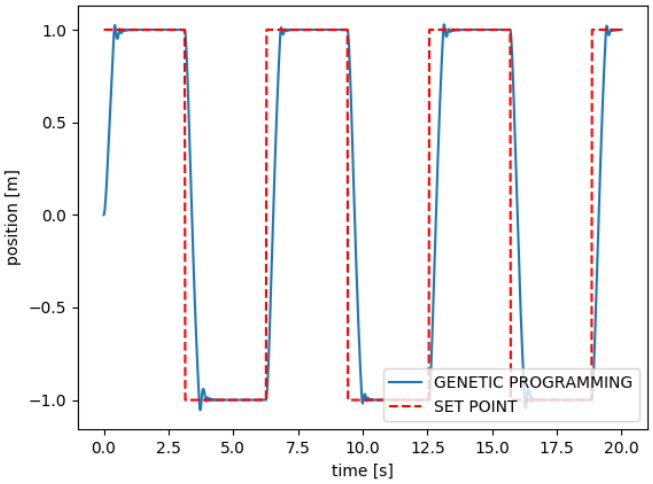


Figure 6.7: System position and set point (square wave) command during the time

An individual found as output of genetic programming is only one among all potential optimal solutions.
Moreover the results obtained with the evolutionary algorithms come from dynamic processes and sometimes the form of the predicted solution can be complex and difficult to handle. So, it is difficult to model or to attempt to change the solution found if it does not solve the problem as expected.
For that reason particular importance is attributed to the effective use of the results in the reality. Remarking that the results are the output of a stochastic search algorithm, it is normally recommended to do more than just a simulation to get solutions which are also statistically reliable.

6.1.1.3 Multi-step signal

The multi-step command has been defined as steps with different values on the position commanded during the time and in this case it is defined as following:

- to 1 m from 0 to 4 s
- to 5 m from 4 to 10 s
- to 2 m from 10 to 14 s
- to 8 m from 14 to 20 s

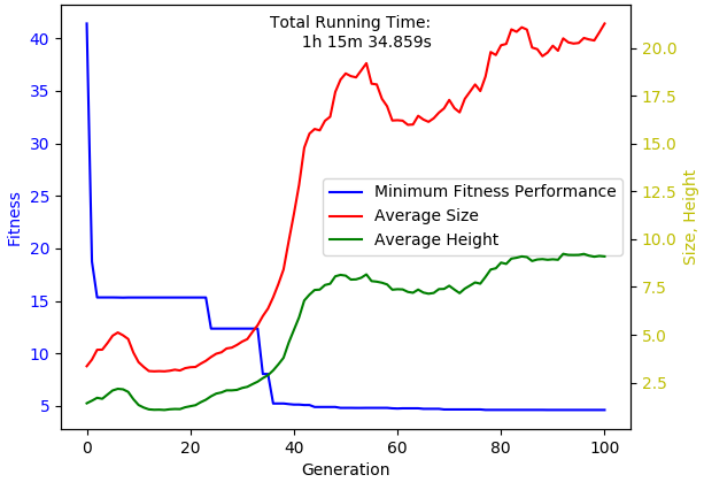


Figure 6.8: Genetic programming statistics during the generation evolution for multi-step signal case

As shown in figure 6.8 the fitness stabilizes around a value of 4.601 after the sixtieth generation. It has a value bigger than the two cases shown before cause of the variation on the position values. In this way a bigger amount of error is accumulated during the time so the fitness presents at the same time a big value.
The best individual discovered is:

$$\begin{aligned} &Mul(Mul(Tanh(Mul(Tanh(Mul(Tanh(Mul(Tanh(err), 81.6497)), Abs(8))), \\ &Abs(Mul(Sqrt(Abs(pi)), Log(6))))), Abs(Abs(pi))), 81.6497) \end{aligned} \tag{6.5}$$

and its tree representation is:

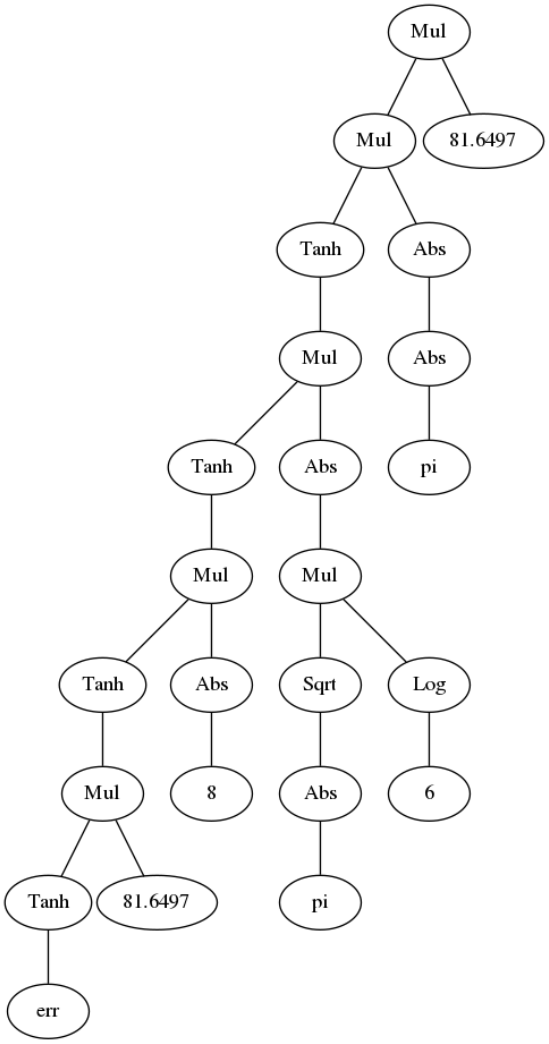


Figure 6.9: Best individual for multi-step signal

Figure 6.10 shows an evident overshoot at 10 s.
It can be minimized or sometimes deleted by varying the genetic programming parameters, for example by increasing the population and generation size.
In stochastic search algorithms as GP, even a little change in parameters as probability of mutation or crossover, sometimes takes a big change in the evolution process, but it is also true that sometimes big changes in GP parameters can lead to small variations of the results.

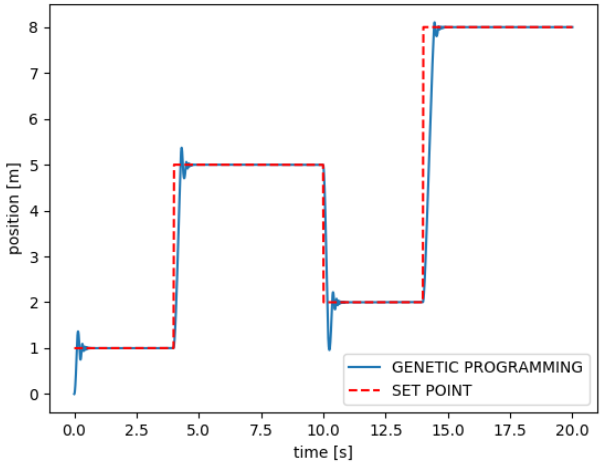


Figure 6.10: System position and set point (multi-step) command during the time

Magnifying the figure 6.10, a steady state error, however minimal, is always present. To reduce it, a new control law is required. A new control law, i.e. a new individual with a better value of fitness can help to minimize the overshoot, the steady state error and the oscillations, shown below in figure 6.11:

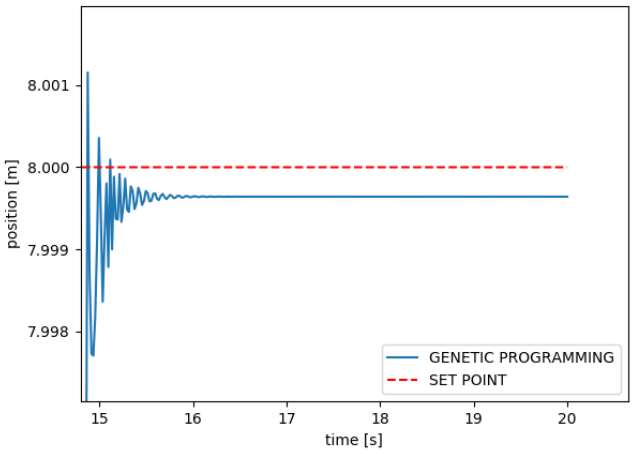


Figure 6.11: Oscillations and steady state error between the real position system and multi-step signal

6.1.2 Behaviour of the controllers by varying system parameters

In real systems many components can be stressed or damaged during the operative life. This section deals with the changes of the plant parameters without taking into account an update of the control law.
The goal is to show how the solutions found with design parameters work by using different plant parameters. It is also interesting to notice in this case how the trend of position error evolve during the time with the variation of these plant parameters.
Therefore multiple controller laws have been tested with several parameters that are different from those parameters on which the design of main control laws are based. The values of the design parameters (M, B and K) are the same introduced in the other previous cases.
The choice to use multiple controller laws, specifically three, comes from the need to have more reliability of the results, remembering the fact that the process of optimization is a stochastic search in evolutionary algorithm.
The multi-step signal, previously introduced, has been used as input signal for these tests. One of the three controller laws has been shown before in equation (6.5). The others two laws have been obtained with the same settings in table 6.1. Their mathematical forms (see Appendix C.1.1 for the tree representations and GP statistics) are presented below:

- second controller law:

$$\begin{aligned} &Mul(Abs(Add(Add(Add(Sub(err,-82.3398),Sub(6,Sub(Mul(Sub(Sub(Mul\\ &(Sqrt(e),Sqrt(err)),err),Sub(Sub(err,Sub(Tanh(Mul(pi,89.0697)),err)),\\ &err)),Sqrt(err)),err))),Sub(Tanh(d_e rr),Sub(Sub(Mul(Sub(Sub(Mul\\ &(Sqrt(e),Sqrt(err)),err),Sub(Sub(err,Sub\\ &(Sqrt(i_e rr),err)),Sub(Sqrt(i_e rr),err))),Sqrt(err)),err),err))),Sub\\ &(err,Sub(Mul(Sqrt(Sqrt(Abs(err))),Tanh(d_e rr)),err)))),Sqrt(i_e rr)) \end{aligned} \tag{6.6}$$

- third controller law:

$$Mul(i_e rr, Sub(Add(err, Add(err, Add(err, 37.7908))), Abs(Sqrt(Abs(6))))) \tag{6.7}$$

Considering that the simulations were run with the same GP settings, the three solution can be compared in terms of fitness values:

$$\text{third controller} > \text{second controller} > \text{first controller}$$

so the first is the best among the three controllers.
In the following sections only the graphs including the errors will be shown, so for the position graphs see the Appendix C.1.1.

6.1.2.1 Mass variation

The values of mass used for the tests have been chosen relatively close to the design value as shown in the graphs below:

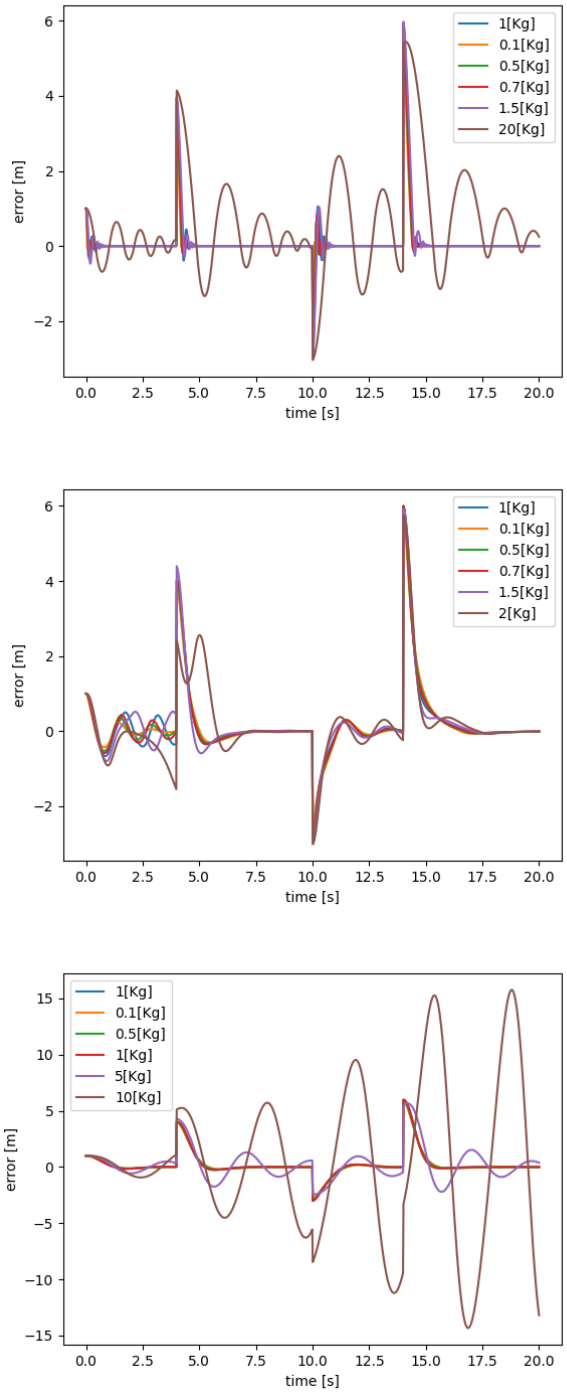


Figure 6.12: Position errors by varying mass values for the first, second and third controllers

Remembering that the forms of the GP solutions are difficult to manage and comprehend, therefore different values of mass can be found in the three comparisons of the controllers. The main problem is that the controller law loses its effectiveness for certain values and for that reason the error tends to increase very quickly so that a comparison couldn't be done with the same parameters. An example of this phenomenon will be provided after. Generally solutions found by EA are adequate only for the problem to solve, so a little change on plant parameters can lead to a big variation on the behavior of the system.
Magnifying the graph of the first control and considering that the behaviour doesn't change for the other controllers:

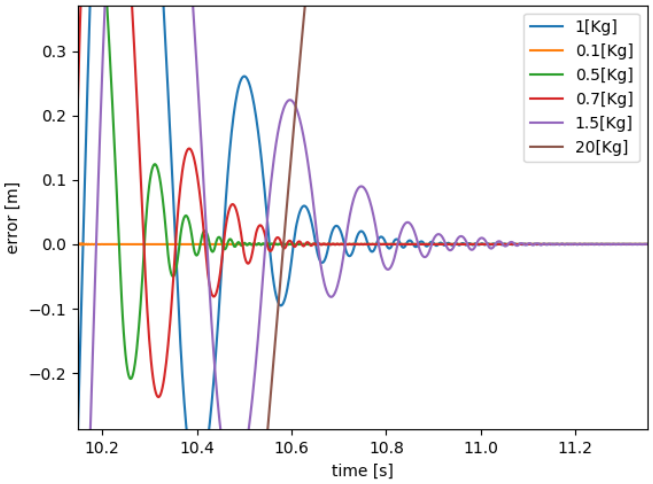


Figure 6.13: Details of the first controller graph

an important observation from figure 6.15 is that the error trend decreases during the time if the mass values are lower than the design mass value $M = 1$ kg. So in these cases the controller works better if compered to the case with the design mass value. Considering now the MSD as a real case, the plant parameters often tend to reduce their values and not the opposite. For higher mass values the error tends instead to increase, as for example the third controller case with a mass equal to 10 kg cannot be still defined as a controlled system.

6.1.2.2 Spring stiffness variation

A similar analysis of the error propagation described for the mass variation case section was used for the spring stiffness variation case.

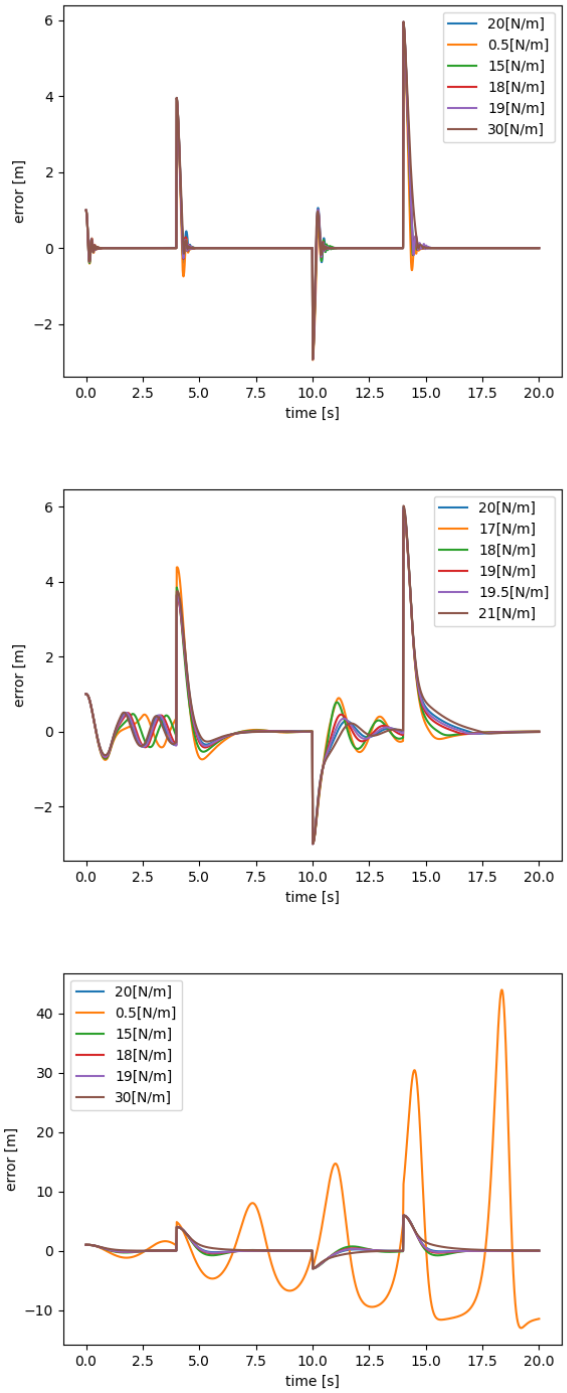


Figure 6.14: Position errors by varying spring values for the first, second and third controllers)

In this case for lower values of stiffness coefficient the error increase and this behaviour appears even in the other controllers. The opposite behaviour appears in error values if the stiffness coefficient increases.
Furthermore for the first controller, it is possible to notice that the curves even for high value of stiffness get close, i.e. the error between them doesn't vary significantly.
As it is possible to notice from the second graph relative to the second controller in figure 6.14, the variation of the stiffness values is very close to the design one $K = 20$ N/m. Indeed returning to what was mentioned before, for some solutions a small change sometimes can lead an unacceptable growth of the error, as shown in the following figure:

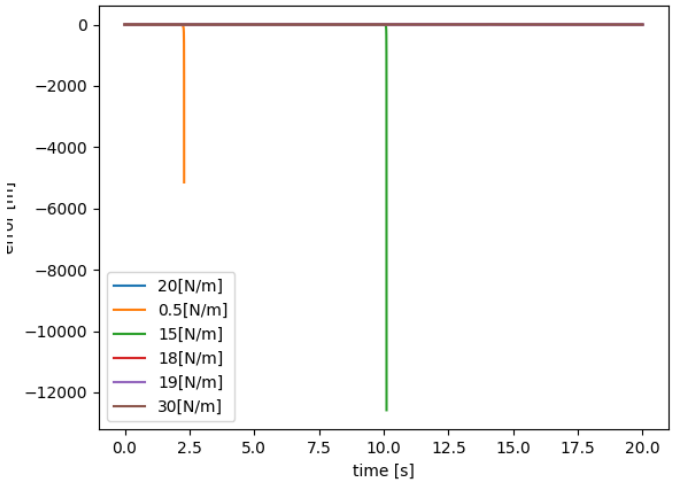


Figure 6.15: Error trend in second controller with different values of stiffness coefficient

A stiffness value of 0.5 N/m causes an error that becomes too high during time for the third controller.
For that reason in a real system an update of the controller before the error of the control variable achieves a certain value should be taken into account. The update of the control form is really necessary especially in systems in which there are constraints to prevent problems to the normal system operation or in some cases to avoid catastrophic situations.

6.1.2.3 Damper variation

The variation of damping coefficient has been taken in consideration. The approach is still the same used in the previous variation cases, described in the beginning of this section.

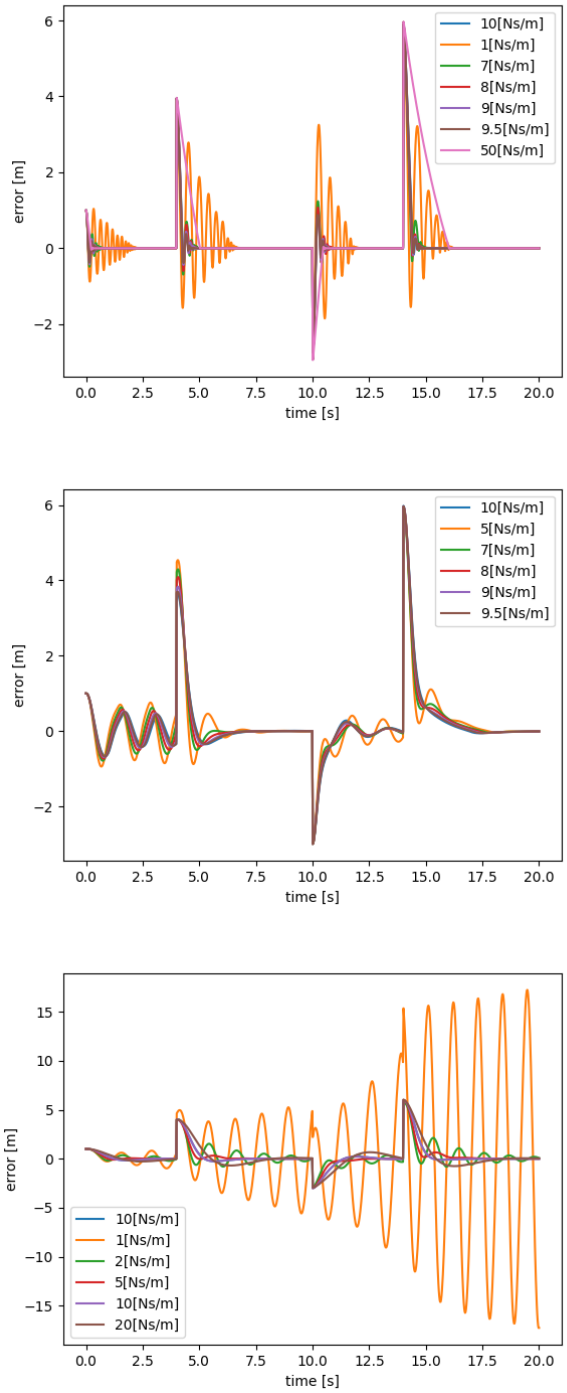


Figure 6.16: Position errors by varying damping parameters for the first, second and third controllers)

As shown in figure 6.16, even in this test case it is possible to notice that the controllers continue to manifest an acceptable behaviour if the variations from the design damping value $D = 10$ Ns/m are moderate.
However the responds of the controllers show some differences for big variations. With a damping value equal to 1 Ns/m the error trend tends to increase showing an oscillatory behavior that is damped after a transitional period in the first controller, whilst the same value of damping coefficient causes considerable oscillations in the third controller.

6.2 Off-line design control for Goddard problem

The results obtained without considering disturbances over all time simulation show possible structure form of the controller for Goddard problem.

Objective:	Find programs whose solutions match the optimal references in time interval [0,800] seconds
Terminal set:	$[errR, errTheta, errVr, errVt, errM, R]$, with R four ephemeral constants
Function set:	$[+, -, *, TriAdd, Sqrt, Tanh, Abs]$
Fitness measure:	IAE
Selection:	NSGA II
Initialization pop:	Full
Parameters:	pop size 500, gen size 500, crossover 70% , mutation 10%
Termination:	generation size

Table 6.2: Genetic programming settings for Goddard problem

The table 6.2 shows the main GP settings used, notice that the population and generation sizes are increased because the complexity of the problem, to be solved, needs a more big search space.

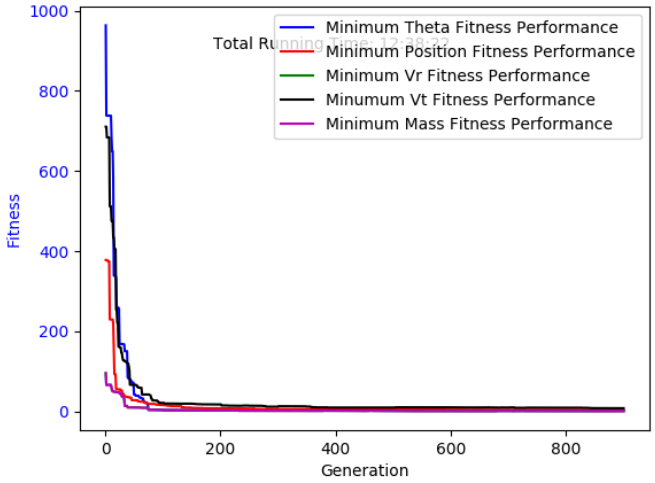


Figure 6.17: Genetic programming statistics during the generation evolution for Goddard offline design control

The simulation took 12 hours and 38 minutes, the time is considerably increased but it can be disregarded in an off-line design.
Remember that the solution is represented by individual that contains two subtrees which are respectively the radial and the tangential throttle control laws (see the Appendix C.2.1 for mathematical forms).
The controller form are very complex, the size of individuals becomes bigger and bigger from generation to generation. In real applications a designer must respect some constraints on the complexity of the controller given by hardware specifications. In some applications the time responses must be supplied within precise times and if the controller structure is more complex, this need could not be met.
In those cases the designer may define an additional objective in which the evaluation process deals with the program complexity.

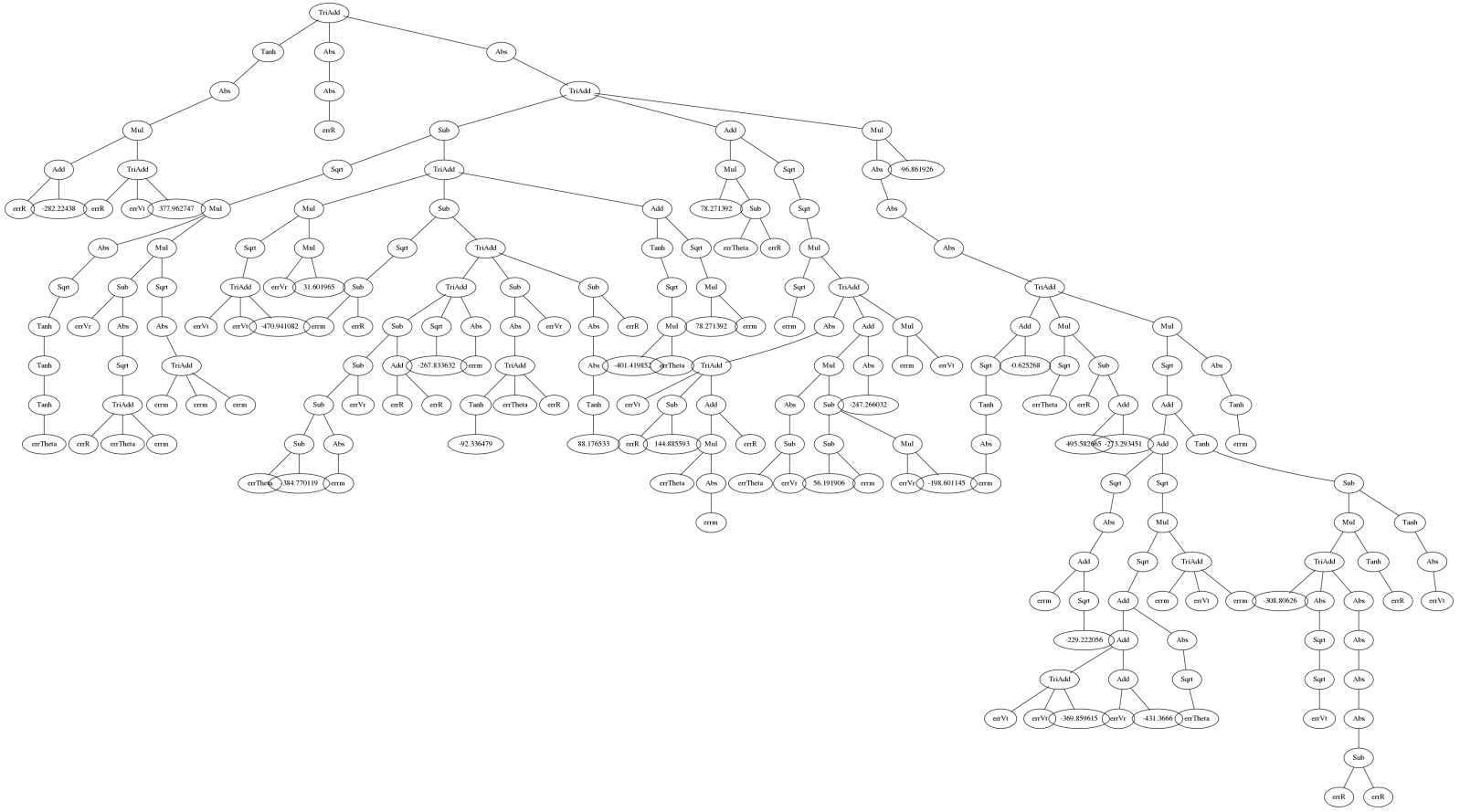


Figure 6.18: Tree program representation for radial throttle

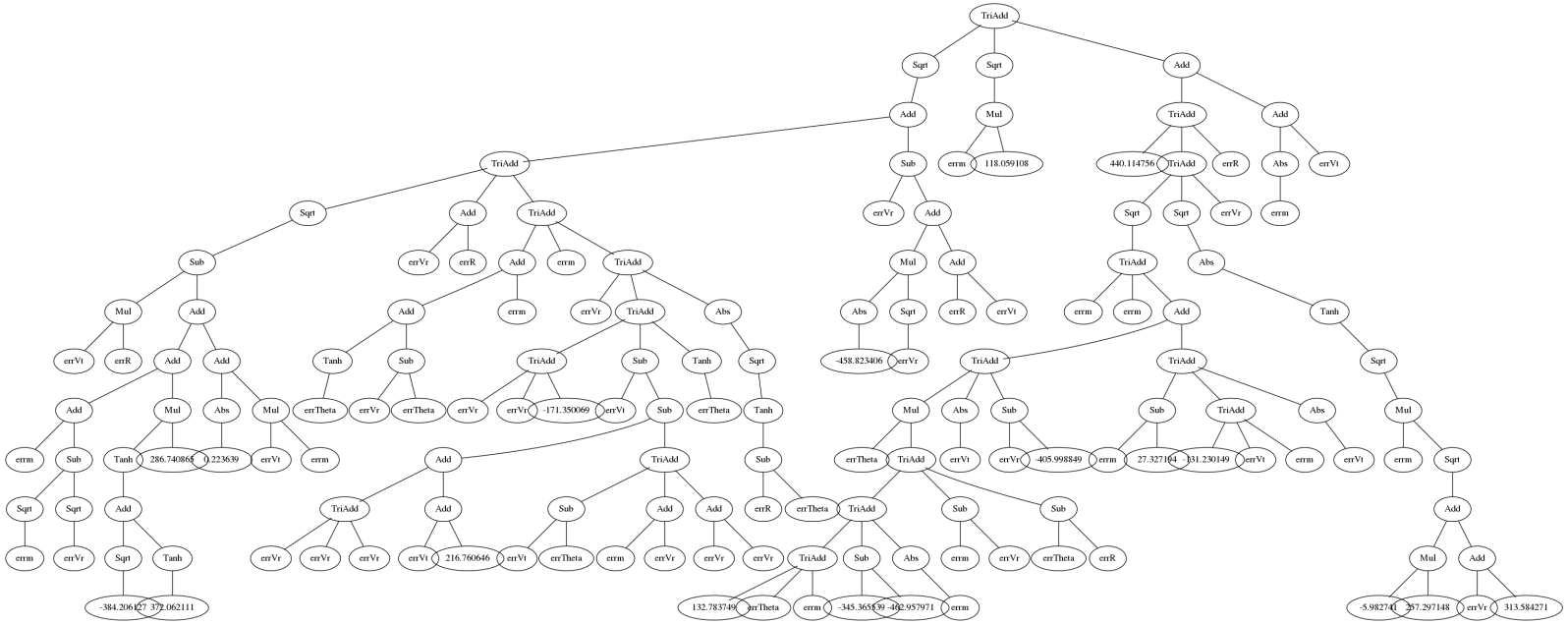
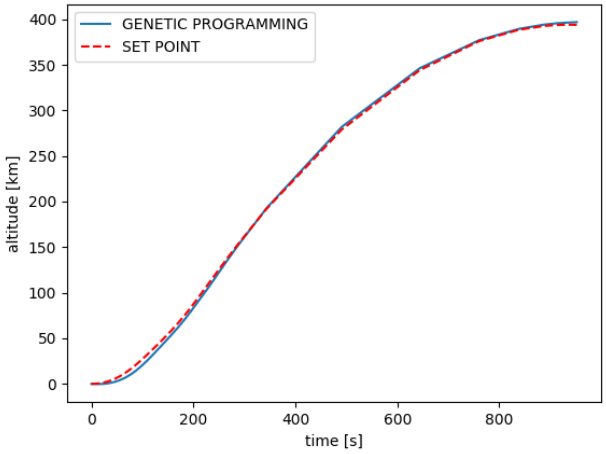
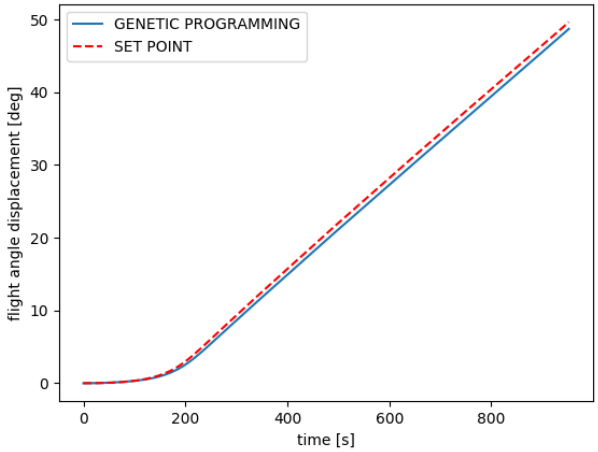


Figure 6.19: Tree program representation for tangential throttle

Finally the graphs of the states over time by using this controller are shown below:





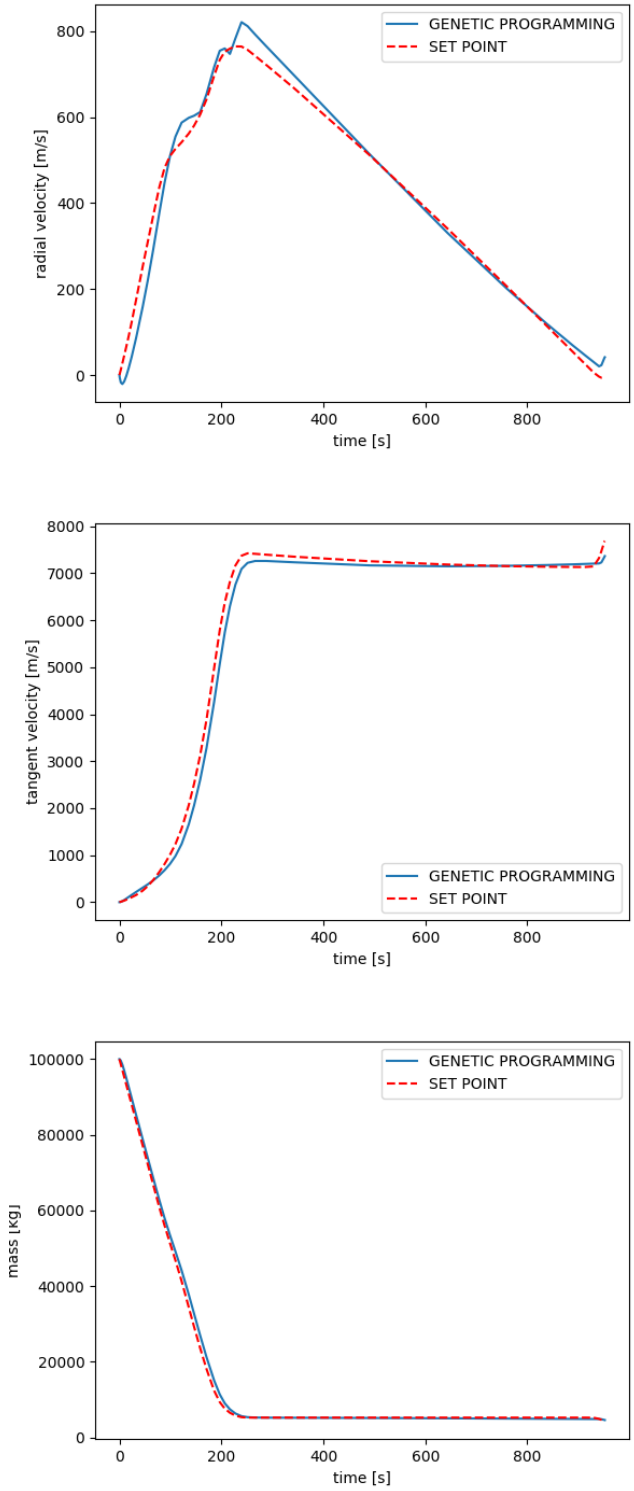


Figure 6.20: Graphs of altitude, flight angle displacement, radial velocity, tangential velocity and mass over time

Also, figure 6.20 clearly shows the capabilities of genetic programming to create the structure of the controller.
In fact it is possible see how genetic programming controller tends to minimize the errors of all states between the current values and the desired values over time.
It can be observed considering the Mean Absolute Percentage Error (MAPE) defined as:

$$\text{MAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \frac{|y_i(t) - r_i(t)|}{r_i(t)} \tag{6.8}$$

where n is the number of the computed errors over time, $y(t)$ is the current state and $r(t)$ is the commanded state.

State	Mean absolute percentage error (%)
Altitude	0.04
Flight angle displacement	7.58
Radial velocity	21.92
Tangential velocity	5.49
Mass	4.60

Table 6.3: Mean absolute percentage error in state variables

The table 6.3 expresses the controller accuracy in percentage per each state variables.
Moreover the Goddard rocket plant is more complex if compared to MSD because some parameters inside the plant are constrained in a specific interval of values.
Genetic programming, as already stated, is a random search algorithm and it is unconstrained by its nature.
The introduction of constraints shall be done carefully in genetic programming. Some individuals may violate the constrains during the evolution therefore such individuals must be necessarily penalized.
Constraint handling with penalty strategies have been proposed in works like [31] and [7].

6.3 On-line design control for mass-spring-damper system

In this section, the results for the design of the on-line intelligent controller are presented. Firstly it is significant to introduce the presence of some constraints that have been imposed on the position of the mass.
Position limit-switches have been used like constraints so that the mass cannot exceed the position limit and remained within a precise position range. They have been defined as:

- limit-switch on the maximum position reachable by the mass : 10 m
- limit-switch on the minimum position reachable by the mass: -5 m

Now it is possible to explain in more details the concrete approach used in the intelligent design of the controller.

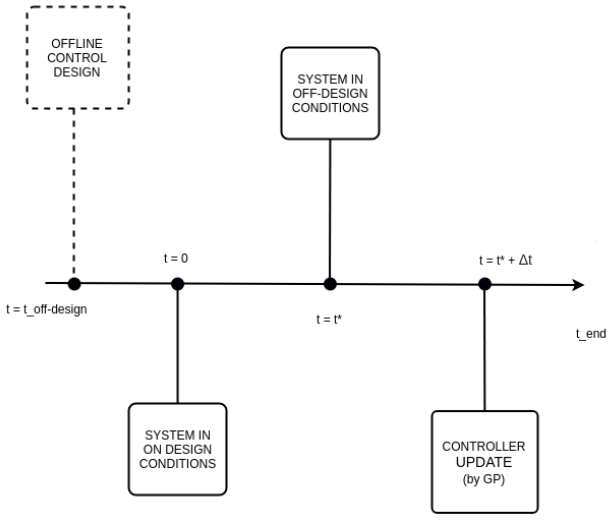


Figure 6.21: Time line of the simulation process

As shown in the above figure 6.21 the simulation starts at $t = 0$, the plant is controlled by a law designed in off-line trough genetic programming algorithm. After the error occurs in the plant at $t = t^*$, the genetic programming seeks a new individual whose behaviour is more suited for the plant that presents the variation of the parameters. The length of the time interval Δt, i.e. the time taken by evolution algorithm is very important because it represents the time in which the system works in altered conditions. Therefore it is very important that the MSD system does not violate the constraints during this time interval ranging from $t = t^*$ to $t = t^* + \Delta t$. Once a new controller is found, it is updated and the simulation ends when the time $t = t_{end}$ is reached.
Consider now the following example in which the position mass must follow the multi-step setpoint signal defined as:

- to 1 m from 0 to 60 s
- to 2 m from 60 to 150 s
- to 4 m from 150 to 200 s
- to 5 m from 200 to 300 s
- to 6 m from 300 to 450 s
- to 9 m from 450 to 600 s

The plant has the following parameters:

$$M = 1\ kg \quad B = 10\ Ns/m \quad K = 20\ N/m$$

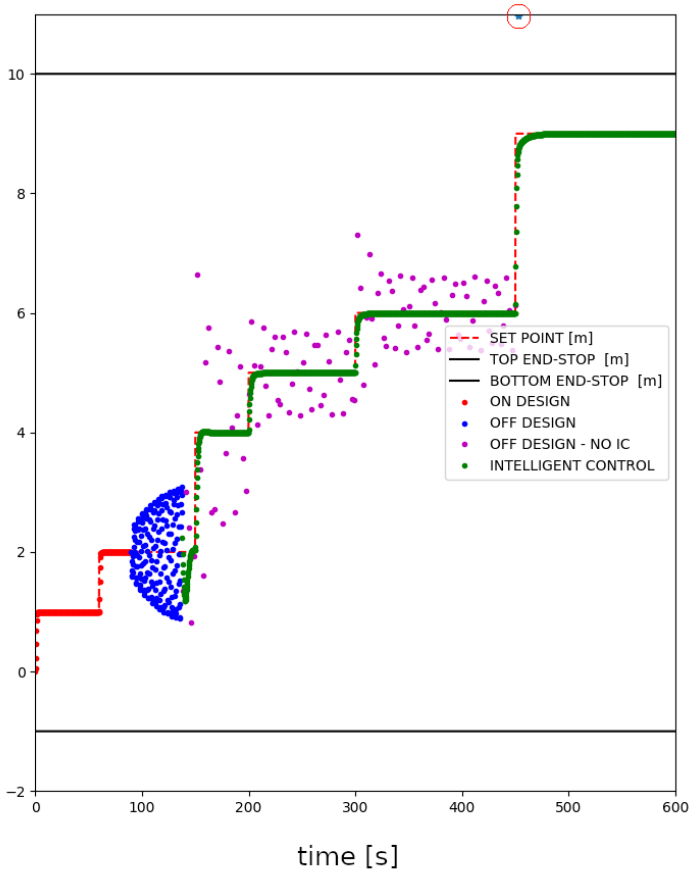


Figure 6.22: Online design control simulation

Figure 6.22 shows a complete run for 600 s.
The red dots represent the position values of the mass over time by using the controller designed off-line. At the beginning of the simulation the general behaviour of the system is acceptable as the mass position reaches the desired position.
Hereunder the control equation and its tree representation found during the off-line design are shown:

$$\boxed{add(multiply(i_{err}, add(add(29.256, err), err)), Log(err))} \tag{6.9}$$

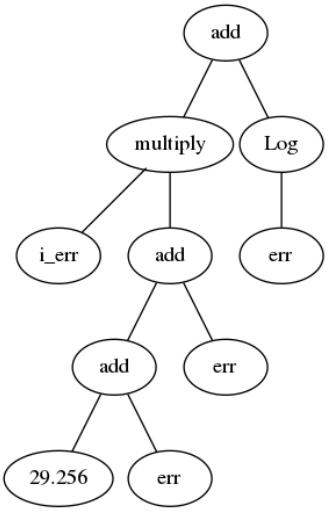


Figure 6.23: Offline-control law

As already mentioned, the plant components may manifest degradation in real application. Therefore in order to simulate a such situation, the introduction of a variation on the damping coefficient is considered at $t^* = 90$ s. The value of damping coefficient changes from the design value 10 Ns/m into 3 Ns/m. The blue dots represent the system working in off design conditions. So, the system is now uncontrolled and manifests oscillations. These oscillations increase in amplitude and such behaviour may be very dangerous for the integrity of the system. In fact the purple dots shows the trend of mass position until to the end of the time if no corrective action was taken. As can be observed from the figure 6.22, the mass position at 453 s violets the maximum constraint on position values, exceeding 10 m as represented by the blue asterisk in red circle. The update of the control law is necessary to avoid such situation.
The green dots describe the position mass, the plant is controlled through the update of the controller designed on-line by GP.
The new structure of the controller is:

$$\boxed{add(multiply(i_err, absolute(add(Tanh(e), err))), add(i_err, err))} \tag{6.10}$$

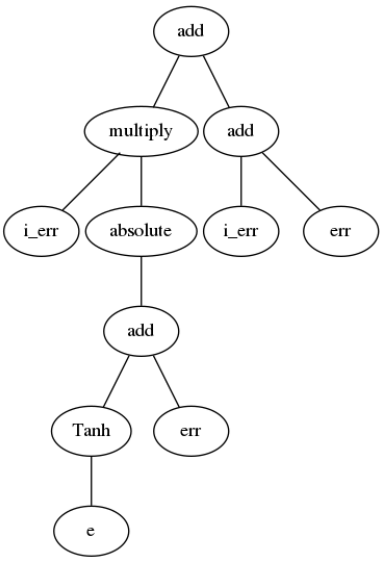


Figure 6.24: Online-control law

Genetic programming took $\Delta t = 48$ s in order to update the controller law in this case. The update is introduced at $t^* + \Delta t = 119$ s, where t^* is the time instant when the variation on damping coefficient occurs and Δt is the time interval in which the system works in off design conditions.

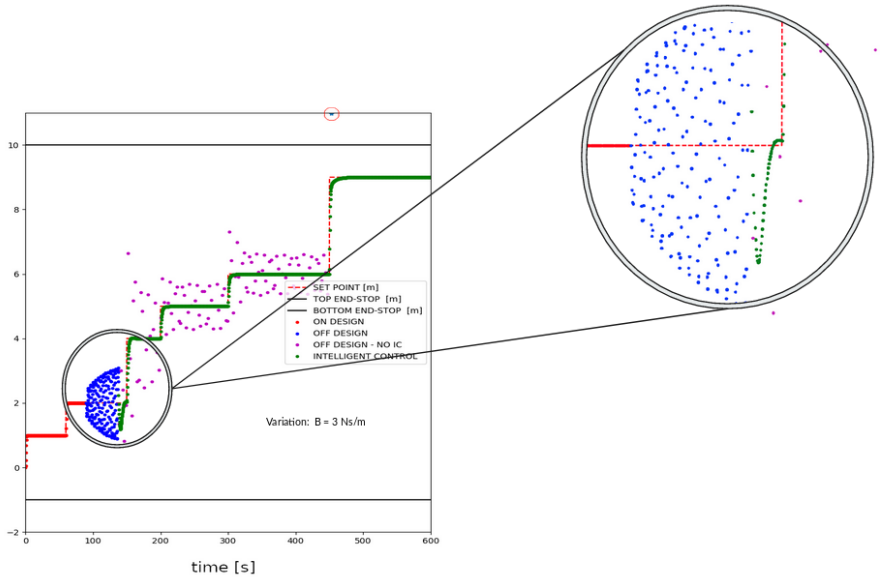


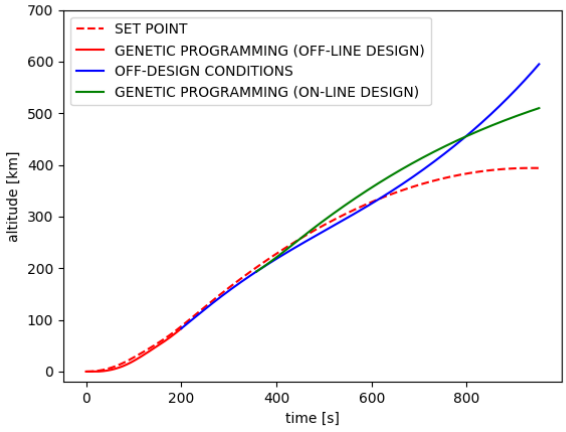
Figure 6.25: Update of the controller law

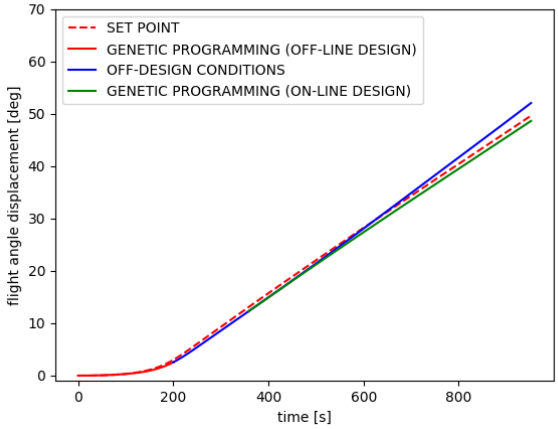
Figure 6.25 shows in details the update of the controller. The oscillatory

behaviour of the system is damped by the new control law. It is evident that the update enhances the system performances so that the system may operate more accurately with the new plant parameters and the violation of the constraint does not occur any more.

6.4 On-line design control for Goddard problem

The same approach described before, see the figure 6.21, has been adopted in Goddard problem in order to obtain the results for the on-line design of the controller.
The previous result in off-line case, figure 6.18 and 6.19, has been used as initial control structure during the period in which the system is operating in on-design conditions where no disturbance is taken into account.
In this instance a variation of the drag coefficient is considered. The C_D value instantaneously varies from 0.3 to 12 at $t^* = 200$ s. Such variation has been set randomly in order to simulate the incoming of a change in the plant.





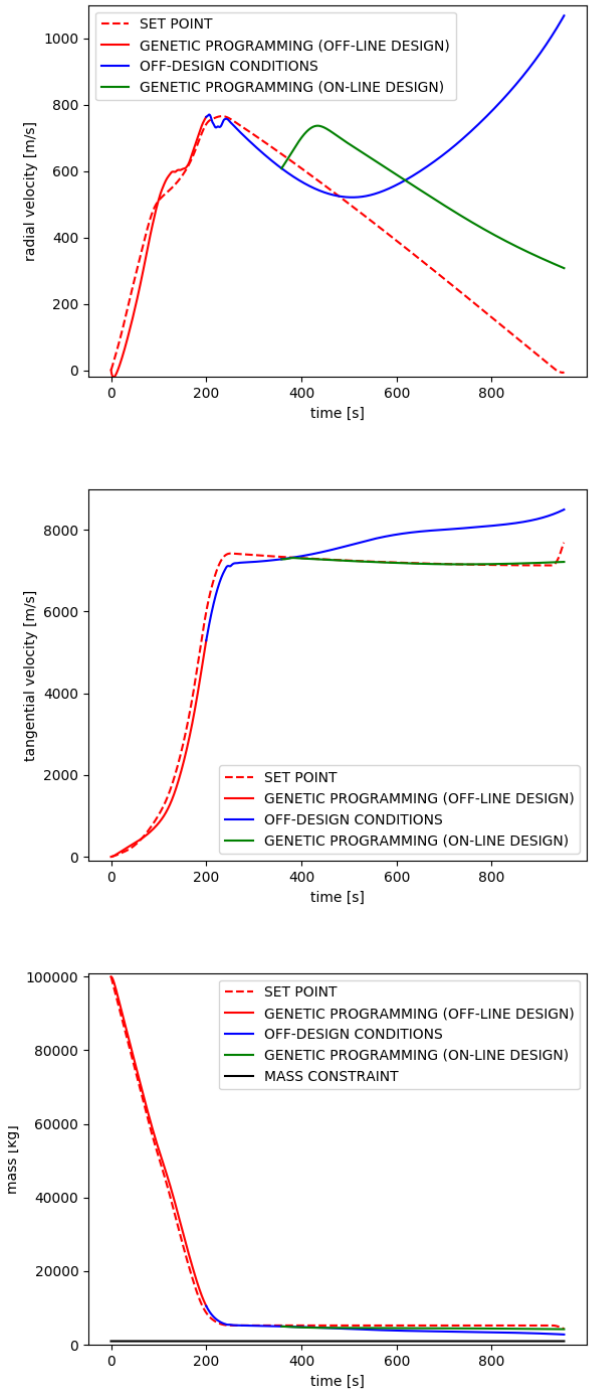


Figure 6.26: States with on-line update of the controller (12 cores)

The graphs in figure 6.26 show an attempt to re-establish the control on each states after the incoming of the disturbance updating the controller through GP.
The red lines represent the system controlled by a law designed in off-line, in fact one may notice that the state values follow the commanded values in red dot lines.
The blue lines represent the response of the system in presence of the disturbance.
The behaviour of the system characterised by the new control structure is represented by the green lines.
Remember that the individuals found in off-line design case for Goddard problem have been used to initialize the starting population in genetic programming evolution for the update.
The new control structures found are:

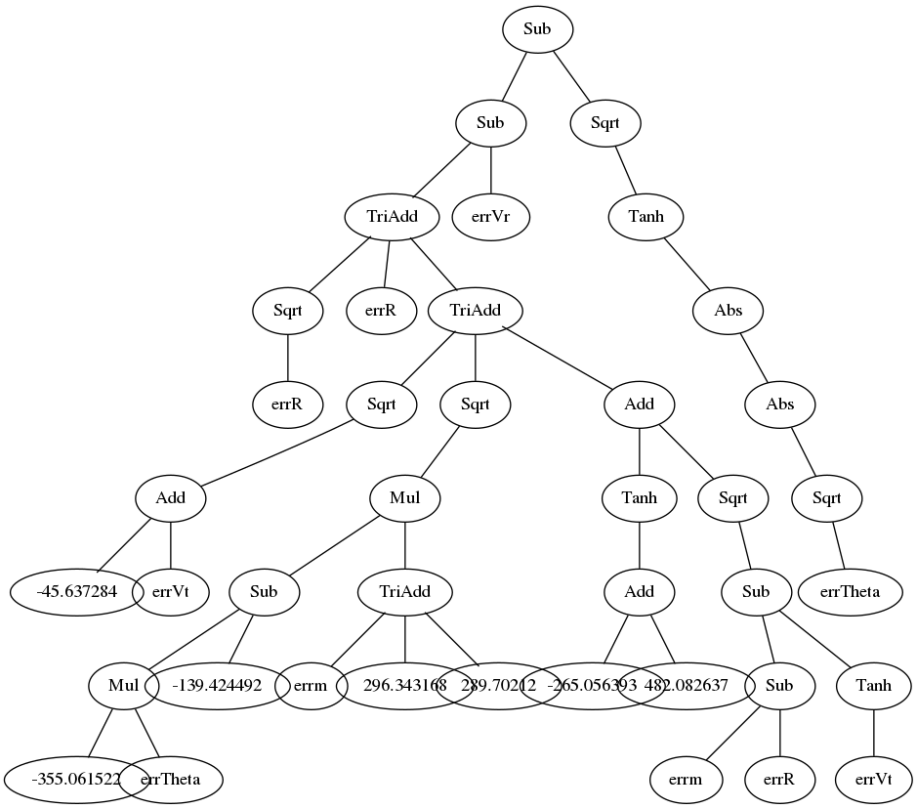


Figure 6.27: Update of the radial throttle control law

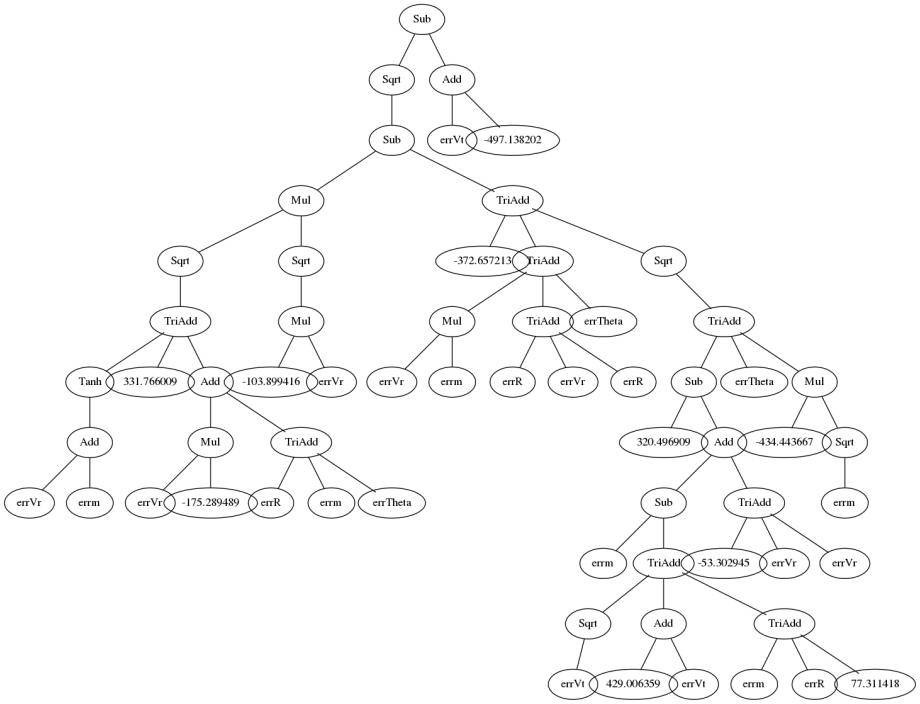


Figure 6.28: Update of the tangential throttle control law

The action of the update controller using genetic programming is not satisfactory even though the solution seems to be suitable to operate with the new conditions. In fact the errors for some states, like position and radial velocity, cannot be negligible.
It can be explained considering the time taken by genetic programming to find the new controller. The results shown in figure 6.26 have been obtained using 12 cores (Intel® Core^{TM} i7-8750H CPU @ 2.20GHz) and the time needed by genetic programming for updating is $\Delta t \simeq 158$ s.
Such amount of time is important because the initial conditions of the plant are substantially changed from the conditions that genetic programming has received as inputs at 200 s when the disturbance occurred in the plant.
Table below shows the computational time Δt for different numbers of employed cores (see the equations in Appendix C.2.2):

Core numbers	Δt (s)
1	746.146
3	450.855
6	336.408
9	191.109
12	158.383

Table 6.4: Computational time in on-line process depending by cores

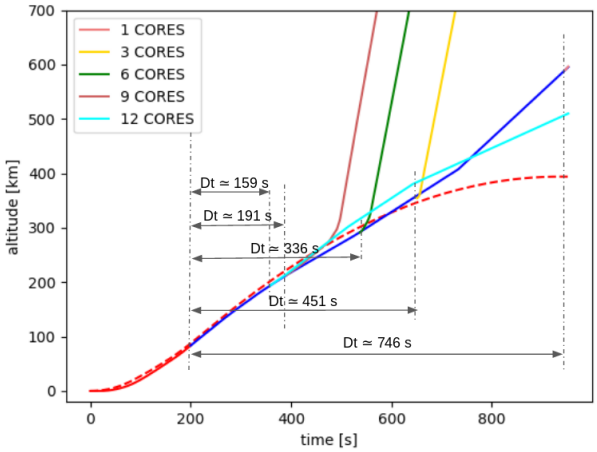
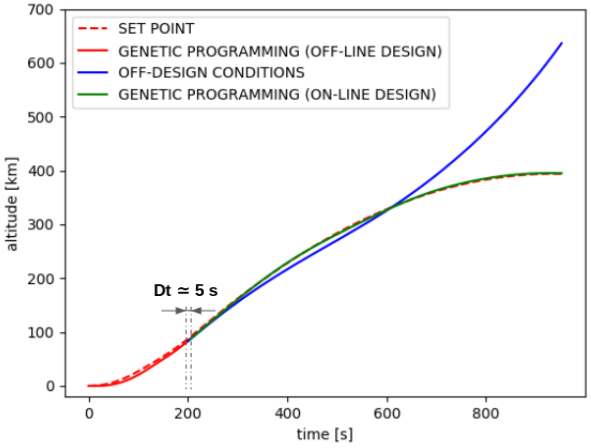
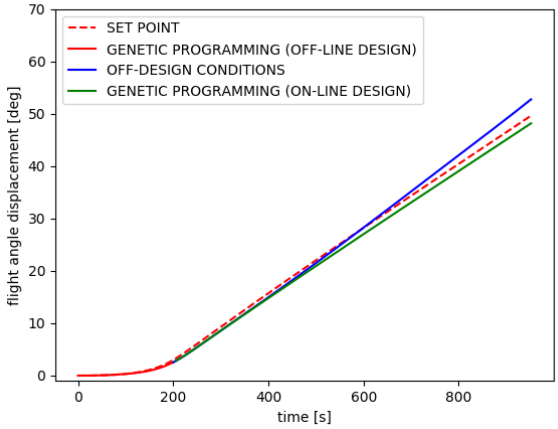
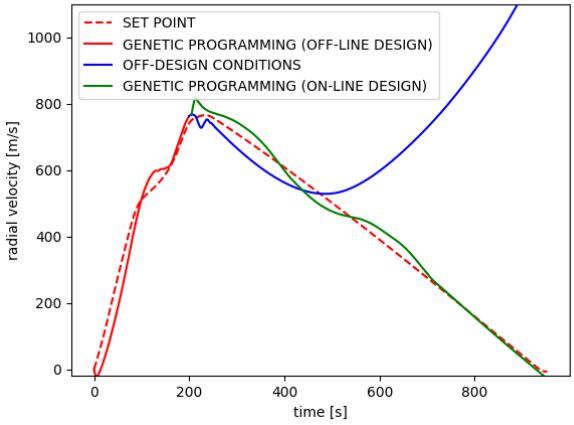


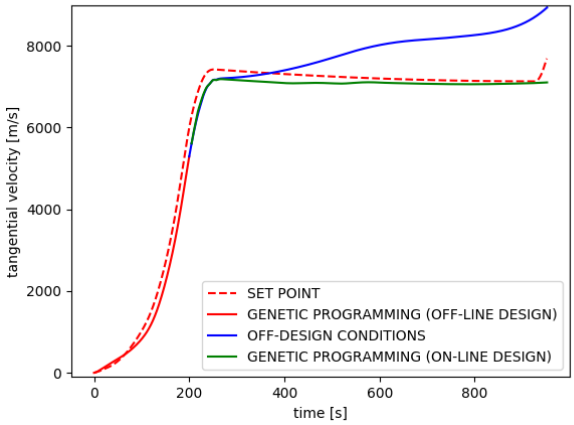
Figure 6.29: Time spent by GP for updating varying the processing cores

It is clear from figure 6.29, representing only the altitude state, that the time taken by GP considerably decreases employing more computing resources.
Suppose to freeze this time interval to 5 s, time in which genetic programming seeks an individual able to operate better with the disturbance in the plant, i.e. about 32 times less then the results found using 12 cores.
Considering this hypothesis, the results using as on-line structure of the controller those in figures 6.27 and 6.28, are shown hereunder:




SET POINT
GENETIC PROGRAMMING (OFF-LINE DESIGN)
OFF-DESIGN CONDITIONS
GENETIC PROGRAMMING (ON-LINE DESIGN)
flight angle displacement [deg]
time [s]


SET POINT
GENETIC PROGRAMMING (OFF-LINE DESIGN)
OFF-DESIGN CONDITIONS
GENETIC PROGRAMMING (ON-LINE DESIGN)
radial velocity [m/s]
time [s]


SET POINT
GENETIC PROGRAMMING (OFF-LINE DESIGN)
OFF-DESIGN CONDITIONS
GENETIC PROGRAMMING (ON-LINE DESIGN)
tangential velocity [m/s]
time [s]

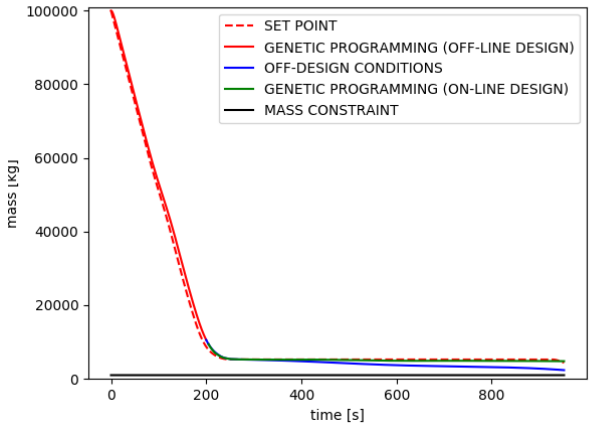


Figure 6.30: States with on-line update assuming $\Delta t = 5$ s

The graphs clearly show the capabilities of genetic programming in on-line application. In fact after the disturbance, GP updates the controller in order to restore the control of the plant afflicted by the disturbance as represented by the green lines. Notice that if such update process does not occur, the state variables would tend to diverge as described by the blue lines and the errors would be large. In this instance, assuming short time $\Delta t = 5$ s, the control law is applied when the initial conditions are not very different from those with which the genetic programming individuals have been evaluated. Therefore the updated control laws are suitable to re-establish the desired plant behaviour even in the presence of the disturbance.

6.4.1 High-performance computing

Since, as seen, the computational time is a key aspect for the update of the controller in on-line process for Goddard problem, the simultaneous execution of blocks code has been considered. Therefore a large population is divided in a such way that each individual is evaluated at the same time by a different processing core. Such technique is called parallel computing. All the results, shown before, have been obtained using this technique but with a limited number of processing cores (12).
In this instance higher computational capabilities are required so computational resources were provided by HPC@POLITO, a project of Academic Computing within the Department of Control and Computer Engineering at the Politecnico di Torino in order to achieve the following results.
Legion was the employed computer cluster with the characteristics presented in table below (more specifications can be found in [12]):

LEGION	
Architecture	Linux Infiniband-EDR MIMD Distributed Shared-Memory Cluster
Node Interconnect	Infiniband EDR 100 Gb/s
Service Network	Gigabit Ethernet 1 Gb/s
CPU Model	Intel Xeon Scalable Processors Gold 6130 2.10 GHz 16 cores (2x)
Sustained performance (Rmax)	21.094 TFLOPS
Peak performance (Rpeak)	27.238 TFLOPS
Computing Cores	448
Number of Nodes	14
Total RAM Memory	5.376 TB DDR4 REGISTERED ECC
OS	Centos 7.6.1810 - OpenHPC 1.3.8.1
Scheduler	SLURM 18.08.8

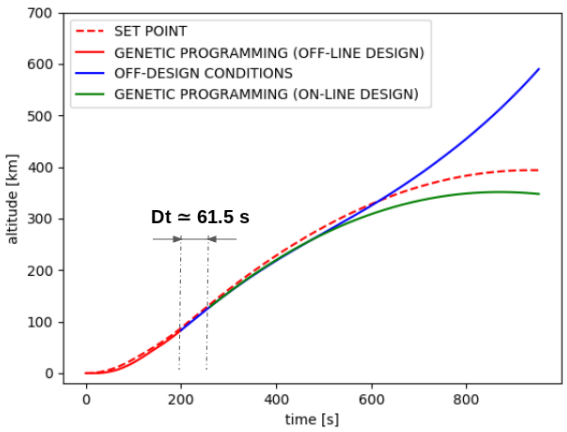
Table 6.5: Legion computer cluster specifications

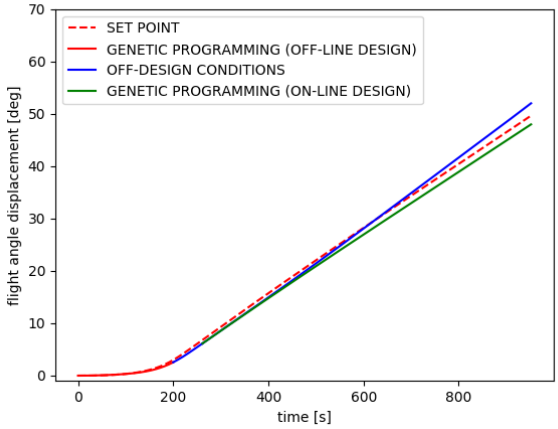
The current state of deap library allows to run the code only in local machine, for that reason the computing capacity were limited in a single node. A node has 32 cores whose 30 cores have been effectively employed in this work.
The results found are:

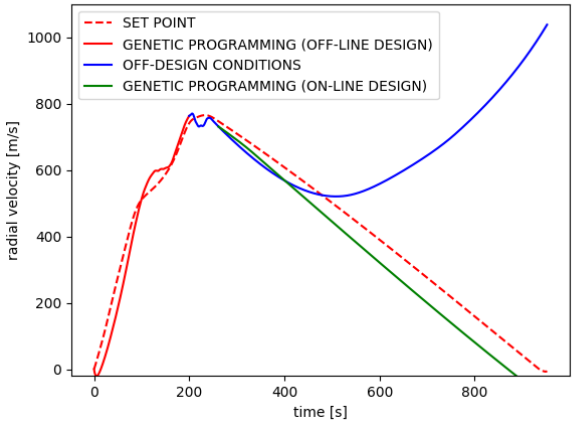
Core numbers	Δt (s)
30	61.532

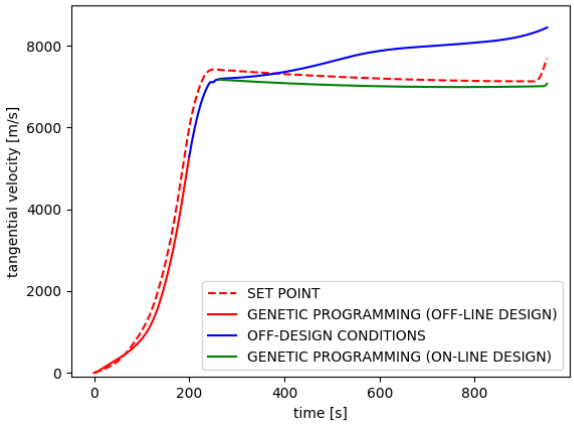
Table 6.6: Computational time in on-line process on HPC single node

The parallel computing shows how the increased number of employed multi-processors helps to speed up the code. In fact the computational time is reduced and the Δt is changed from roughly 158 s with 12 cores to 61.532 s with 30 cores. The equations of the solution found using HPC can be found in Appendix C.2.2.1.




SET POINT
GENETIC PROGRAMMING (OFF-LINE DESIGN)
OFF-DESIGN CONDITIONS
GENETIC PROGRAMMING (ON-LINE DESIGN)
flight angle displacement [deg]
time [s]
0
10
20
30
40
50
60
70
0
200
400
600
800


SET POINT
GENETIC PROGRAMMING (OFF-LINE DESIGN)
OFF-DESIGN CONDITIONS
GENETIC PROGRAMMING (ON-LINE DESIGN)
radial velocity [m/s]
time [s]
0
200
400
600
800
1000
0
200
400
600
800


SET POINT
GENETIC PROGRAMMING (OFF-LINE DESIGN)
OFF-DESIGN CONDITIONS
GENETIC PROGRAMMING (ON-LINE DESIGN)
tangential velocity [m/s]
time [s]
0
2000
4000
6000
8000
0
200
400
600
800

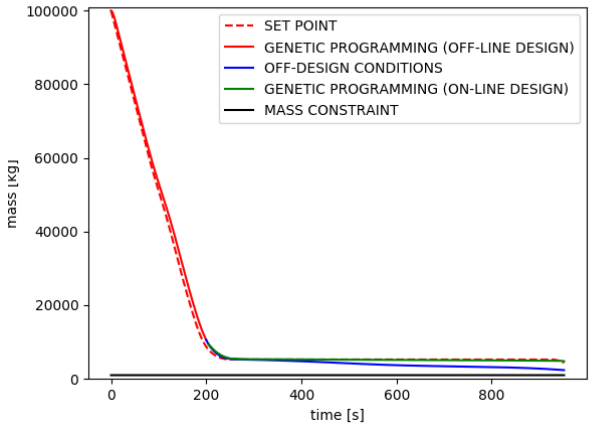


Figure 6.31: States with on-line update of the controller (30 cores)

In figure 6.31, the graphs prove in practice that in future genetic programming could be implemented as a possible approach to update the structure of the controller even in complex systems presenting uncertainties when the computer technology will permit to reduce the computational times.

Chapter 7

Conclusions and future work

Genetic programming is a powerful technique able to solve complex problems. Its developing abilities are very suitable for designing control systems avoiding the complexity of the control theory. Moreover, genetic programming method is very effective in presence of uncertainties, for example when there is a lack information on the plant or the operating environment. Although powerful tools are provided to people who are not necessarily experts in that specific field, the results obtained shall be carefully tested before a possible implementation.
In control engineering, the main advantage of this methodology is to find the appropriate structure of the controller starting from a high-level problem statement for a given problem without prior knowledge about the system is required. Starting from basic control laws, more complex and suitable laws emerge automatically during the evolution.
Space missions are characterized by large uncertainties due to external environment and other many factors not well known therefore to afford them the adaptive capabilities of genetic programming may be a possible key approach to create an intelligent controller.
In this thesis, the demonstration of genetic programming capabilities has been shown in the on-line reconstruction of the controller structure. Soon GP may find applications thanks to innovative technologies which will decrease the computational times, its current drawback.

For future work, a possible direction may be to create an evolutionary algorithm to further reduce the computational times improving the parallel computing and to develop a more efficient method to handle the constraints.
Moreover it would be very interesting to combine genetic programming with other methodologies such as Artificial Neural Network (ANN) in intelligent control. There are two main possibilities to be explored:

- neural networks can be trained with disturbance pattern in order to allocate the optimal constant terms in the trees, previously created by GP, when a disturbance occurs in the plant;
- the architecture of the neural networks can be designed by GP considering that such architectures require expertise knowledge and a lot of attempts.

GP/ANN applications could be very fascinating in intelligent control area because such combination method would incorporate the adaptive ability of the

genetic programming to the learning capacity of the artificial neural network, fundamental aspects of intelligence.

Appendix A

Pareto optimization

The concept of this method is named after Vilfredo Pareto, an Italian economist and sociologist who used this concept in his studies of economic efficiency and income distribution in 1906.
The Pareto optimization offers a good answer in terms of set of solutions that define the best trade-off in problems where more than one objective function must minimized or maximized [8]. In multi-objective the goodness of a solution is determined by the dominance.

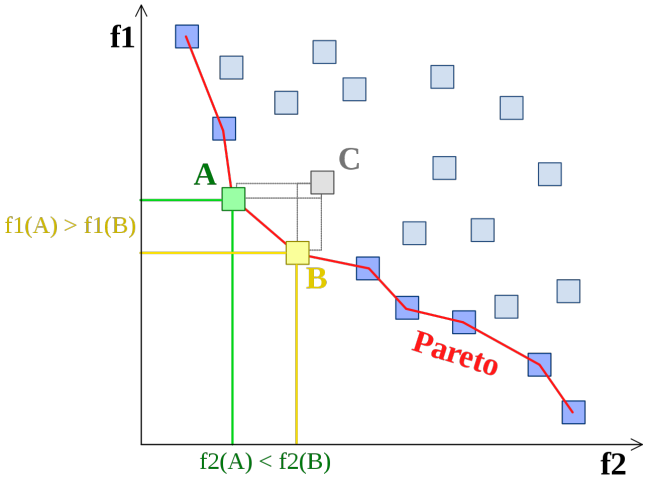


Figure A.1: Dominated and non-dominated solutions [21]

Consider as example the problem shown in the figure above A.1 that represents solutions to solve a problem in which both fitness functions f_1 and f_2 must be maximized.
A solution x dominates another solution y if:

- x is no worse in all objectives then y
- x is equal to y in some objectives but is at the same time strictly better in one other at least

The point C is dominated by both point A and B. But as far as the point B or point A are concerned, neither solution dominates. Such solutions are called non-dominated solutions and all of them form a set of solutions that are not dominated by others also present in this set, called Pareto front.

Appendix B

DEAP

Distributed Evolutionary Algorithms in Python (DEAP) is developed at the Computer Vision and Systems Laboratory (CVSL) at Université Laval, in Quebec city, Canada.
It is a novel evolutionary computation framework for rapid prototyping and testing of ideas. It seeks to make algorithms explicit and data structures transparent. It works in perfect harmony with parallelisation mechanism such as multiprocessing.
The DEAP framework is built over the Python programming language that provides the essential glue for assembling sophisticated evolutionary computation systems. Its aim is to provide practical tools for rapid prototyping of custom evolutionary algorithms, where every step of the process is as explicit and easy to read and understand as possible. It also places a high value on both code compactness and code clarity.
More details can be found in [10].

Appendix C

Results: additional figures

C.1 Test case: mass-spring-damper system

C.1.1 Off-line design

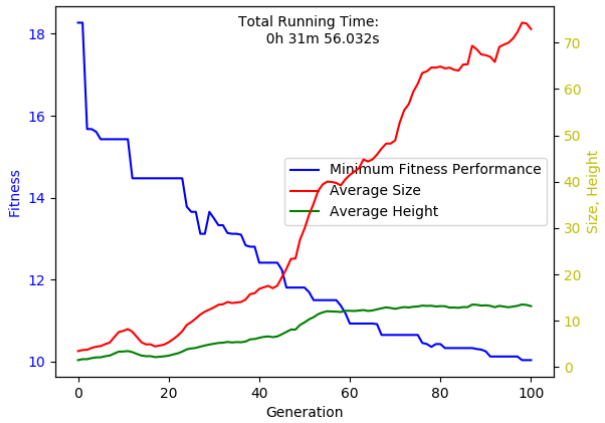


Figure C.1: Genetic programming statistics during the generation evolution for multi-step signal (second run)

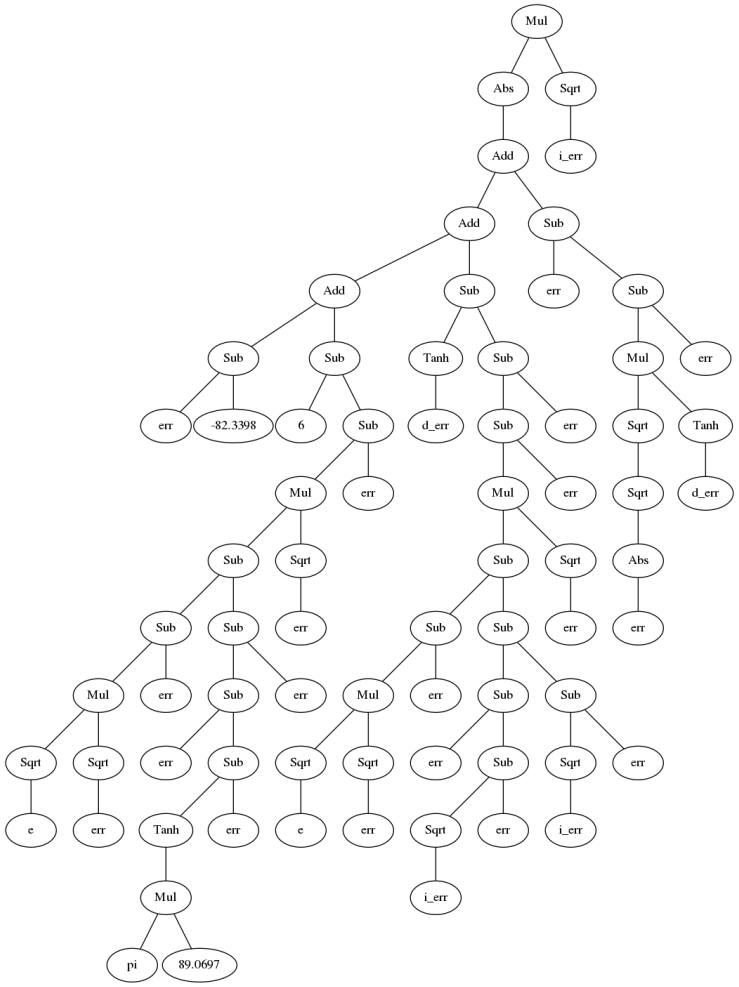


Figure C.2: Best individual for multi-step signal (second controller)

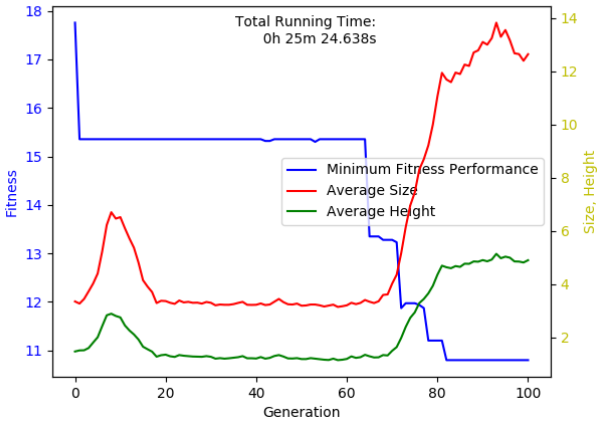


Figure C.3: Genetic programming statistics during the generation evolution for multi-step signal (third run)

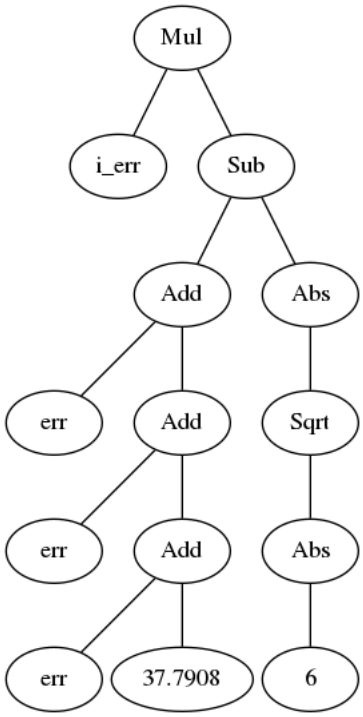


Figure C.4: Best individual for multi-step signal (third controller)

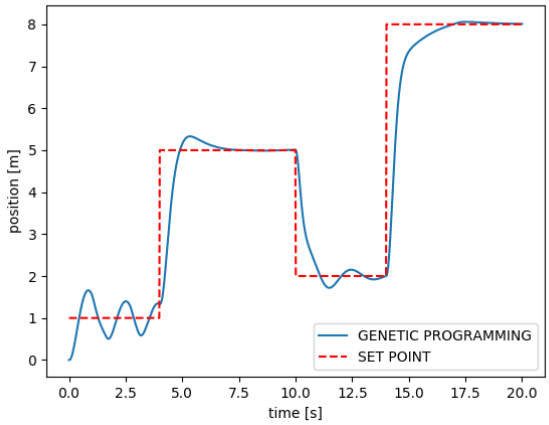


Figure C.5: System position and set point (multi-step) command during the time (second controller)

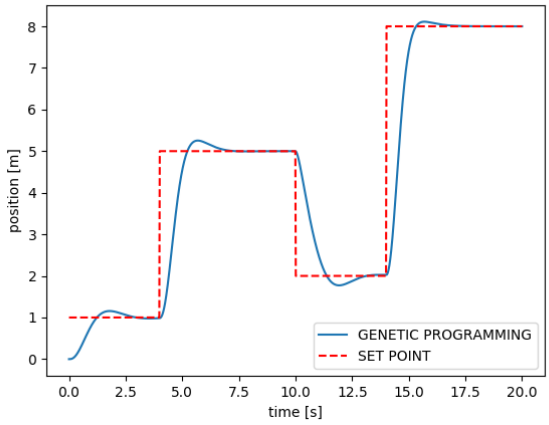


Figure C.6: System position and set point (multi-step) command during the time (third controller)

C.2 Aerospace case: Goddard problem

C.2.1 Off-line design

- mathematical expression of the radial throttle control law:

$$Sqrt(TriAdd(Abs(Sub(Add(TriAdd(Sub(Sub(Sqrt(errR), Add(errVr, -336.252482)), Sub(errVt, errVt)), errTheta, Abs(Abs(errVr))), TriAdd(Add(Sub(Mul(-70.85579, 25.369957), TriAdd(errR, errR, -192.45208)), Sqrt(Mul(errVr, errm))), TriAdd(Sqrt(Abs(TriAdd(Sqrt(Sqrt(Sqrt(TriAdd(errTheta, errVt, errVr)))), Add(Tanh(errm), Add(417.541074, errR)), Mul(Add(272.761683, errVt), Abs(errTheta))))), Tanh(Tanh(189.147772)), Add(Mul(281.174641, errTheta), Add(-463.187985, -44.566462))), errVt)), Add(Abs(Sub(Sub(-294.912831, errm), Mul(errm, errVr))), Add(Tanh(Sqrt(Add(356.090788, Sub(errTheta, errVt)))), Sqrt(Mul(errVt, -163.151633))))))), Mul(Add(Sub(errm, Abs(Sqrt(Add(450.227029, errVr)))), Mul(Sqrt(Sqrt(Tanh(Abs(Abs(Sub(Add(errTheta, Tanh(Sub(errm, errVr))), Add(Abs(Mul(errTheta, errTheta)), Abs(Mul(Tanh(errVt), Mul(TriAdd(Mul(errVr, 165.979145), Sqrt(errR), Abs(Sub(Sub(errm, -383.76704), Sqrt(errm)))), Tanh(Add(errR, Mul(Sub(Tanh(Add(errTheta, errm)), Abs(errm)), Abs(Sqrt(errVt))))))))))))), Tanh(TriAdd(Mul(Mul(errm, errVr), Abs(31.587732)), Sqrt(Abs(errR)), Sqrt(TriAdd(errVr, errR, Sqrt(Mul(Tanh(115.522294), TriAdd(errTheta, 19.959912, 78.486749))))))))), Sqrt(Add(TriAdd(Sub(Tanh(Abs(Sqrt(Add(errTheta, Abs(Sqrt(errR)))))), Add(errVr, -186.266856)), -393.259084, errVt), Sub(errTheta, 304.859982)))), Mul(errm, 114.654252)))$$ (C.1)

- mathematical expression of the tangential throttle control law:

$$TriAdd(Sqrt(Mul(errm, 56.715711)), TriAdd(TriAdd(Sub(Tanh(TriAdd(errTheta, -435.431012, -288.399033)), Sqrt(Sub(-269.090055, -216.348377))), Sqrt(errm), Add(Mul(Abs(TriAdd(Add(Tanh(errVr), Tanh(Mul(Mul(errVr, errVt), TriAdd(errVt, Add(Abs(Abs(errVt)), Sqrt(Add(errm, 189.687451))), Mul(Sqrt(Sub(Mul(Abs(errVt), errTheta), Tanh(errTheta))), Sqrt(Sqrt(Abs(TriAdd(Sqrt(Mul(Tanh(Sub(-72.106792, errTheta)), Add(TriAdd(errVr, errm, errVt), Mul(-214.408013, -424.368337)))), Tanh(-484.034856), Abs(errVt))))))))), Sqrt(errm), Tanh(Abs(TriAdd(errR, Abs(-407.653047), errVr))))), 82.141667), TriAdd(Abs(errVr), TriAdd(Sqrt(Sub(errR, -358.30903)), errm, Sub(errR, Tanh(Sqrt(Mul(181.443512, errVr))))), Tanh(Tanh(errVt))))), Sqrt(Mul(265.132498, errm)), Sqrt(TriAdd(Abs(errm), Sqrt(-217.067638), TriAdd(Sqrt(-95.90338), errm, Mul(Sub(Tanh(Add(Abs(errm), TriAdd(errTheta, 151.21694, errVt))), Sub(Sub(Mul(errTheta, Add(errR, errR)), Add(Sub(Add(Add(errVr, errR), Add(132.515184, errm)), Add(TriAdd(errVt, Sub(Add(Sqrt(Sub(-493.510467, errVr)), Sqrt(188.460917)), Add(Tanh(Mul(errVr, errR)), Sub(-22.727781, errVr))), errm), TriAdd(errR, 421.834855, 228.627005))), Sqrt(Tanh(330.127231)))), Tanh(errm))), Sqrt(Abs(Tanh(Sub(134.690301, errTheta))))))))), Tanh(Add(errVt, Mul(errR, errm))))$$

(C.2)

C.2.2 On-line design

- mathematical expression of the radial throttle control law (1 core):

$$Sub(TriAdd(Abs(Sub(errVr, 7.900593)), Tanh(-354.893854), Abs(Add(errVt, Mul(Sqrt(499.496265), Sqrt(errm))))), -47.247372)$$

(C.3)

- mathematical expression of the tangential throttle control law (1 core):

$$Sub(Sqrt(Sub(Sub(Abs(errR), Abs(errm)), TriAdd(Mul(411.628734, errVt), TriAdd(errVt, 343.34771, errVr), TriAdd(Mul(Mul(Sqrt(Tanh(Add(errR, 226.82443))), Mul(TriAdd(-388.762661, 242.483295, -195.313505), Tanh(115.641917))), Sqrt(-69.938948)), Tanh(Mul(TriAdd(errR, errR, errVr), Add(errVt, errVr))), errVr)))), Tanh(-1.144805))$$

(C.4)

- mathematical expression of the radial throttle control law (3 cores):

$$Sub(Abs(Add(Sqrt(Abs(Mul(Mul(Abs(Mul(errm, -51.765722)), errR), Mul(Tanh(Tanh(Sqrt(errVt))), -174.463585)))), Mul(errVt, errVr))), -47.247372)$$

(C.5)

- mathematical expression of the tangential throttle control law (3 cores):

$$TriAdd(errVt, TriAdd(Add(-328.487666, errVt), Sub(Abs(TriAdd(errm, errVt, errTheta)), errVr), TriAdd(Sub(Sub(errTheta, errm), Sqrt(errTheta)), errTheta, 97.024843)), Abs(TriAdd(errm, errTheta, errR)))$$

(C.6)

- mathematical expression of the radial throttle control law (6 cores):

$$TriAdd(errVt, Abs(TriAdd(Mul(Sqrt(errm), errVr), Sqrt(Sqrt(-483.694546)), TriAdd(TriAdd(Sqrt(Mul(-498.911321, errVr)), TriAdd(Abs(TriAdd(Tanh(Tanh(Add(errVr, errVr))), errTheta, errVt)), TriAdd(Tanh(Add(Mul(TriAdd(errm, -165.266, errm), Add(errVt, errVt)), Mul(Add(errVr, errTheta), Abs(errTheta)))), errR, errR), TriAdd(Add(errR, errVr), TriAdd(Sub(Sub(errR, errVt), Mul(-50.59781, errR)), errR, errVt), Mul(errTheta, TriAdd(Sqrt(-466.870687), Tanh(Sqrt(Add(Sub(errm, errm), TriAdd(errTheta, errR, errVr)))), Sub(TriAdd(Sqrt(95.63591), Add(errVt, 268.723319), TriAdd(errTheta, errR, -197.353347)), Abs(Sub(errR, 56.058507)))))))), Abs(Sub(Sub(Sub(Add(errm, errVt), TriAdd(-272.084422, errm, errm)), errVr), errTheta))), Sqrt(errm), Add(Add(Tanh(Sqrt(errm)), errVt), Sqrt(Sub(errVt, errm))))))), errVt)$$

(C.7)

- mathematical expression of the tangential throttle control law (6 cores):

$$Add(Sqrt(Sub(Tanh(Sub(Abs(Sub(85.290271, errR)), Tanh(Abs(Sub(errm, errVt))))), Add(-47.247372, Mul(Tanh(errTheta), Add(Sub(errTheta, Tanh(Sub(Sqrt(errVt), Abs(Sub(errVr, errm))))), Abs(Add(Sub(Mul(Sub(455.88907, errm), Abs(errm)), TriAdd(TriAdd(errR, -331.028936, errVr), Add(errVr, 463.268523), TriAdd(Abs(Sub(errR, 168.295835)), -81.538744, TriAdd(errTheta, errVt, errR)))), errR)))))), Sqrt(TriAdd(TriAdd(errm, errR, errVt), errR, Tanh(TriAdd(-85.772341, -254.968912, errR)))))$$

(C.8)

- mathematical expression of the radial throttle control law (9 cores):

$$Sub(Abs(Sub(TriAdd(Add(Add(Tanh(errR), TriAdd(errVr, errTheta, errVr)), TriAdd(Tanh(Sqrt(Sub(Tanh(TriAdd(Sub(Add(errVr, errTheta), Mul(errR, errVt)), TriAdd(Tanh(errR), Sub(13.106664, errVr), Tanh(errVt)), Add(Tanh(261.88947), Tanh(-188.276585)))), errVt)), Add(Add(Tanh(Sub(errm, errVt)), Sqrt(Add(errVr, -380.439634)), Add(TriAdd(Abs(errVr), Add(errR, TriAdd(errR, errR, errm)), Abs(178.600458)), errm)), TriAdd(Sub(errm, errVt), Add(-492.07265, errVr), Mul(245.276733, errVr)))), TriAdd(errm, Add(Abs(Sub(Tanh(Abs(374.297369)), TriAdd(Sqrt(errTheta), Sqrt(-414.255018), Abs(errR)))), Mul(errTheta, errVr)), errm), Tanh(Mul(errVt, errTheta))), -84.899681)), Tanh(errTheta))$$

(C.9)

- mathematical expression of the tangential throttle control law (9 cores):

$$Sub(Sqrt(Sub(403.641953, TriAdd(Tanh(Add(Sqrt(-166.112045), Tanh(TriAdd(Tanh(Sub(errVr, Abs(errVt))), Sub(Add(errTheta, 468.125334), Sqrt(errVt)), Add(Tanh(errm), Add(Sqrt(Sub(errR, errVt)), Tanh(Abs(errR))))))))), Mul(errm, Sub(Add(TriAdd(errVt, Sub(errTheta, -99.420895), errR), Sub(Add(TriAdd(errVt, errVr, errR), Tanh(errVt)), Abs(TriAdd(errVt, errR, 268.869042)))), Abs(TriAdd(errVt, errR, 268.869042)))), TriAdd(Add(Abs(Sqrt(Tanh(errR))), TriAdd(Mul(errVt, errVr), errVt, errTheta)), Tanh(Abs(Tanh(errTheta))), Tanh(Sqrt(errVt))))))), Tanh(-1.144805))$$

(C.10)

C.2.2.1 High-performance computer

- mathematical expression of the radial throttle control law:

$$Abs(Add(TriAdd(Mul(Sqrt(Sub(errVt, Tanh(97.249645))), Sqrt(Add(errTheta, errm))), Add(377.308903, 291.6056), Sub(Add(Abs(Sub(Add(errTheta, Mul(TriAdd(Add(Add(Abs(errVt), TriAdd(errm, 160.462015, errm)), Sqrt(Sqrt(359.755819))), errTheta, 375.888504), Tanh(errm))), Add(errm, errTheta))), Abs(errTheta)), Mul(Add(errVt, errVr), Tanh(errm)))), Abs(Add(errVt, errVr))))$$

(C.11)

- mathematical expression of the tangential throttle control law:

$$Sub(Sqrt(Sub(403.641953, TriAdd(Sqrt(Sqrt(errm)),\\ Sqrt(-370.815091), TriAdd(Add(Mul(-246.639438, errm),\\ Sub(Sqrt(-82.537675), TriAdd(errTheta, Tanh(TriAdd(Mul(errVt, errVr),\\ Sub(errR, 23.525674), Add(errVr, errVt))), -30.844872))), errVt,\\ Sub(Abs(Sqrt(errm)), Mul(Add(errm, errVt), TriAdd(Tanh(errR),\\ Sqrt(errR), Mul(141.473099, errTheta)))))))), -246.639438) \tag{C.12}$$

Appendix D

Python Scripts

D.1 MSD _ OFFLINE.py

The primit

```
/* This Source Code Form is subject to the terms of the Mozilla
                                    Public
 * License, v. 2.0. If a copy of the MPL was not distributed with
                                     this
 * file, You can obtain one at http://mozilla.org/MPL/2.0/. */
/*
----- Copyright (C) 2019 University of Strathclyde and Authors ----
                                    --
------------------- Author: Francesco Marchetti ------------------
-------------- e-mail: francesco.marchetti@strath.ac.uk -----------
--------------------- Author: Nicola Menga --------------------------
-------------- e-mail: nicola.menga@strath.ac.uk -------------------
-----------------------nicola.menga@studenti.polito.it-------------
*/

import tkinter as tk
import _pickle as cPickle
import pickle
from scipy.integrate import solve_ivp
from scipy.interpolate import PchipInterpolator
import numpy as np
from numpy import *
import networkx as nx
from networkx.drawing.nx_agraph import graphviz_layout
import pygraphviz as pgv
import random
from deap import gp
import matplotlib.pyplot as plt
import sys
import timeit
import time
import pandas as pd
from functools import reduce
from operator import *
from deap import algorithms
from deap import base
from deap import creator
from deap import tools
import multiprocessing
```

```
from interruptingcow import timeout
import matplotlib.animation as animation
from matplotlib import style
import itertools
from functools import wraps
from time import time

def Sqrt(a):
        if a > 0:
            return np.sqrt(a)
        else:
            return np.abs(a)


def Log(a):
        if a > 0:
            return np.log(a)
        else:
            return np.abs(a)


def Tanh(a):
        return np.tanh(a)


def setpoint(t,type_set):
    if type_set=="Multistep":
        if (t > 0 and t < 4):
            r = 1
        elif (t > 4 and t < 10):
            r = 5
        elif (t > 10 and t < 14):
            r = 2
        elif (t > 14 and t < 21):
            r = 8
        else:
            r = 0
    elif  type_set=="Sinusoidal":
        A = 1
        w = 1
        r = A*np.sin(w*t)
    elif type_set=="Constant":
        r=7
    elif type_set=="Square":
            r=np.sign(np.sin(t))
    return r

start = timeit.default_timer()


a0 = 1                              #MASS M [kg]
a1 = 10                             #DAMPING C [Ns/m]
a2 = 20                             #STIFNESS K [N/m]

top_end_stop = 10
bottom_end_stop = -1

rise_time = 1                       #rising time [s]

total_time_simulation = 20

tempo = np.linspace(1e-05, total_time_simulation,
```

```
                                    total_time_simulation * 50)

type_set= "Square"  #Multistep, Sinusoidal, Constant, Square signal

set_point = []
for items in tempo:
        set_point.append(setpoint(items,type_set))

set_interp = PchipInterpolator(tempo, set_point)
r_max = np.amax(set_point)
r_min = np.amin(set_point)


ar = 2 * r_max / (rise_time ** 2)
vr = ar * rise_time
yr = 0.9 * r_max
force_constraint = a1 * vr + a2 * yr + a0 * ar

old = 0

size_pop = 300                  #Population size
size_gen = 50                   #Generation size
Mu = int(size_pop)
Lambda = int(size_pop * 1.5)
pb_cross = 0.6                  #crossover probability
pb_mut = 0.3                    #mutation probability
limit_height = 17               #Max height
limit_size = 400                #Max size

def main():
"""The main function is defined as a container in order to execute
                                  the evolutionary process. Here
                                  the type of algorithm to be used
                                  is chosen like the probability of
                                   crossover and mutation. Moreover
                                   the post processing of the
                                  results as graph of position mass
                                   and the forms of the trees are
                                  inserted in this main function.
                                  """
    pool = multiprocessing.Pool(12) parallel computing

    toolbox.register("map", pool.map)

    global size_gen, size_pop, Mu, Lambda, pb_mut, pb_cross,
                                        random_individual

    random.seed()
    print("INITIAL POP SIZE: %d" % size_pop)

    print("GEN SIZE: %d" % size_gen)

    print("\n")

    pop = toolbox.population(n=size_pop)
    hof = tools.HallOfFame(size_gen)
    Mu = int(size_pop)
    Lambda = int(size_pop * 1.5)

    stats_fit = tools.Statistics(lambda ind: ind.fitness.values)
    stats_size = tools.Statistics(len)
    stats_height = tools.Statistics(attrgetter("height"))
```

```
mstats = tools.MultiStatistics(fitness=stats_fit, height=
                                stats_height, size=stats_size
                                )
mstats.register("avg", np.mean, axis=0)
mstats.register("min", np.min, axis=0)
mstats.register("max", np.max, axis=0)

pop, log = algorithms.eaMuPlusLambda(pop, toolbox, Mu, Lambda,
                                pb_cross, pb_mut, size_gen,
                                stats=mstats,halloffame=hof,
                                verbose=True)

stop = timeit.default_timer()
total_time = stop - start
mins, secs = divmod(total_time, 60)
hours, mins = divmod(mins, 60)

gen = log.select("gen")
fit_min = log.chapters["fitness"].select('min')

perform = []
p = 0
for items in fit_min:
    perform.append(fit_min[p][0])
    p = p + 1

height_avgs= log.chapters["height"].select("avg")
size_avgs = log.chapters["size"].select("avg")
fig, ax1 = plt.subplots()
line1 = ax1.plot(gen, perform, "b-", label="Minimum Fitness
                                Performance")
ax1.set_xlabel("Generation")
ax1.set_ylabel("Fitness", color="b")
for tl in ax1.get_yticklabels():
    tl.set_color("b")

ax2 = ax1.twinx()
line2 = ax2.plot(gen, size_avgs, "r-", label="Average Size")
line3 = ax2.plot(gen, height_avgs, "g-", label="Average Height"
                                )
ax2.set_ylabel("Size, Height", color="y")
for tl in ax2.get_yticklabels():
    tl.set_color("y")

lns = line1 + line2 + line3
labs = [l.get_label() for l in lns]
ax1.legend(lns, labs, loc="center right")
textstr = ('Total Running Time:\n  %dh %dm %.3fs' % (hours,
                                mins, secs))
ax1.text(0.65, 0.9, textstr, transform=ax1.transAxes, fontsize=
                                10,
         horizontalalignment='right')

plt.savefig('Stats_OFFLINE')
plt.show()

print("\n")
print("THE BEST VALUE IS:")
print(hof[0])
print("\n")
print("THE HEIGHT OF THE BEST INDIVIDUAL IS:")
print(hof[0].height)
```

```
print("\n")
print("THE SIZE OF THE BEST INDIVIDUAL IS:")
print(len(hof[0]))

with open("Control Law.txt", "w") as file:
    file.write(str(hof[0]))

with open("Best individuals.txt", "w") as bestever:
    for i in range(size_gen):
        bestever.write(str(hof[i]))
        bestever.write("\n \n")

print("\n")

value = toolbox.evaluate(hof[0])
print("THE EVALUATION OF THE BEST INDIVIDUAL IS:")
print(value)
print("\n")

expr = hof[0]
nodes, edges, labels = gp.graph(expr)
g = pgv.AGraph()
g.add_nodes_from(nodes)
g.add_edges_from(edges)
g.layout(prog="dot")
for i in nodes:
    n = g.get_node(i)
    n.attr["label"] = labels[i]
g.draw("tree.png")

image = plt.imread('tree.png')
fig, ax = plt.subplots()
im = ax.imshow(image)
ax.axis('off')
plt.show()
sys.stdout.write("TOTAL RUNNING TIME:\n %d (h):%d (m):%.3f (s)
                                    \n" % (hours, mins, secs))



x_ini = [1e-05, 1e-05, 1e-05]

def sys2GP(t, x):
    global a0, a1, a2
    M, C, K = a0, a1, a2

    y = x[0]
    dydt = x[1]
    ei = x[2]
    dxdt = [1e-05, 1e-05, 1e-05]

    r = set_interp(t)
    e = r - y

    ctrl = toolbox.compile(hof[0])
    u = ctrl(e, -dydt, ei)

    dxdt[1] = (- C * dydt - K * y + u) / M
    dxdt[0] = dydt
    dxdt[2] = e

    return [dxdt[0], dxdt[1], dxdt[2]]
```

```
tevals = np.linspace(1e-05, total_time_simulation, 100000)

solgp = solve_ivp(sys2GP, [1e-05, total_time_simulation], x_ini
                                    , first_step=0.0001, t_eval=
                                    tevals)
ygp = solgp.y[0, :]
dyy = solgp.y[1, :]
ttgp = solgp.t

controller_value = np.array(controller_value)

set_point_off = []
for items in ttgp:
    set_point_off.append(set_interp(items))
r = set_point_off

errgp = r - ygp


fig, ax2 = plt.subplots()
ax2.set_xlabel("time [s]")
ax2.set_ylabel("position [m]")
plt.plot(ttgp, ygp, label="GENETIC PROGRAMMING")
plt.plot(ttgp, set_point_off, 'r--', label="SET POINT")
plt.legend(loc="lower right")
plt.savefig('Position_plot_OFFLINE.png')
plt.show()

fig1, ax3 = plt.subplots()
ax3.set_xlabel("time [s]")
plt.plot(ttgp, ygp, label="POSITION [m]")
plt.plot(ttgp, errgp, label="ERROR [m]")
plt.plot(ttgp, dyy, label="VELOCITY [m/s]")
plt.plot(ttgp, r, 'r--', label="SET POINT [m]")
plt.legend(loc="lower right")
plt.savefig('Motion_graphs_OFFLINE.png')
plt.show()


chapter_keys = log.chapters.keys()
sub_chaper_keys = [c[0].keys() for c in log.chapters.values()]

data = [list(map(itemgetter(*skey), chapter)) for skey, chapter
        in zip(sub_chaper_keys, log.chapters.values())]
data = np.array([[*a, *b, *d] for a, b, d in zip(*data)])

columns = reduce(add, [["_".join([x, y]) for y in s] for x, s
                                    in zip(chapter_keys,
                                    sub_chaper_keys)])
df = pd.DataFrame(data, columns=columns)

keys = log[0].keys()
data = [[d[k] for d in log] for k in keys]
for d, k in zip(data, keys):
    df[k] = d
df.to_csv('Evolution process.csv', encoding='utf-8', index=
                                    False)

# plt.figure(figsize=(18,18),dpi=500)
# graph = nx.DiGraph(history.genealogy_tree)
# graph = graph.reverse()
```

```
# colors = [toolbox.evaluate(history.genealogy_history[i])[0]
                                    for i in graph]
# positions = graphviz_layout(graph, prog="dot")
# nx.draw_networkx_labels(graph, positions)
# nx.draw_networkx_nodes(graph, positions, node_color=colors)
# nx.draw_networkx_edges(graph, positions)
# plt.savefig('HISTORY.png',dpi=500)
# plt.show()

master = tk.Tk()
master.title("GP Mass-Spring-Damper")

w_width = 1000
w_height = 500

init_pos = [500, 250]
w = tk.Canvas(master, width=w_width, height=w_height)
w.pack()

x0 = init_pos[0]
y0 = init_pos[1]
block = w.create_rectangle(x0 - 50, y0 - 50, x0 + 50, y0 + 50,
                                    fill="black")
floor = 350

ceiling = 50
w.create_line(50, ceiling, 50, floor, width=3, fill="brown")
w.create_line(50, floor, w_width - 50, floor, width=3)

w.create_line(50, y0 + 20, 70, y0 + 20, width=2, fill="red")
damper2 = w.create_line(70, y0 + 20, 100, y0 + 20, width=20,
                                    fill="red")
damper1 = w.create_line(100, y0 + 20, x0, y0 + 20, width=4)
pos_old = 0

base_spring = w.create_line(50, y0 - 20, 70, y0 - 20, width=2,
                                    fill="green")
final_spring = w.create_line(150, y0 - 20, x0, y0 - 20, width=3
                                    )
spring_1 = w.create_line(70, y0 - 20, 250, y0 + 5, width=3)
spring_2 = w.create_line(250, y0 + 5, 300, y0 - 5, width=3)
spring_3 = w.create_line(200, y0 - 20, 199, 100, width=3)

wheel_1 = w.create_oval(0, 0, 0, 0)
wheel_2 = w.create_oval(0, 0, 0, 0)

for j in dyy:
    w.move(block, j / 100, 0)
    pos = w.coords(block)

    if pos[2] > pos_old or pos[2] == pos_old:
        w.delete(damper1)
        w.delete(final_spring)
        w.delete(spring_1)
        w.delete(spring_2)
        w.delete(spring_3)
        w.delete(wheel_1)
        w.delete(wheel_2)
        damper1 = w.create_line(100, y0 + 20, pos[0], y0 + 20,
                                    width=5, fill="red")
        final_spring = w.create_line(abs(pos[0] - 300), y0 - 20
                                    , pos[0], y0 - 20,
```

```
                                        width=2, fill="green"
                                        )
            pos_final_spring = w.coords(final_spring)
            wheel_1 = w.create_oval(pos[0], floor - 50, pos[0] + 25
                                        , floor, fill="blue")
            wheel_2 = w.create_oval(pos[2] - 25, floor - 50, pos[2]
                                        , floor, fill="blue")
            spring_1 = w.create_line(pos_final_spring[0] - 10,
                                        pos_final_spring[1] +
                                         20, pos_final_spring
                                        [0],
            pos_final_spring[1], width=3, fill="green")
            pos_s1 = w.coords(spring_1)
            spring_3 = w.create_line(70, y0 - 20, 90, y0 - 40,
                                        width=3, fill="green"
                                        )
            pos_s3 = w.coords(spring_3)
            spring_2 = w.create_line(pos_s3[2], pos_s3[3], pos_s1[0
                                        ], pos_s1[1], width=3
                                        , fill="green")
            pos_s2 = w.coords(spring_2)
            pos_old = pos[2]
        else:
            w.delete(damper1)
            w.delete(final_spring)
            w.delete(spring_1)
            w.delete(spring_2)
            w.delete(spring_3)
            w.delete(wheel_1)
            w.delete(wheel_2)
            final_spring = w.create_line(abs(pos[0] - 300), y0 - 20
                                        , pos[0], y0 - 20,
                                        width=2, fill="green"
                                        )
            pos_final_spring = w.coords(final_spring)
            damper1 = w.create_line(100, y0 + 20, pos[0], y0 + 20,
                                        width=5, fill="red")
            wheel_1 = w.create_oval(pos[0], floor - 50, pos[0] + 25
                                        , floor, fill="blue")
            wheel_2 = w.create_oval(pos[2] - 25, floor - 50, pos[2]
                                        , floor, fill="blue")
            spring_1 = w.create_line(pos_final_spring[0] - 10,
                                        pos_final_spring[1] +
                                         20, pos_final_spring
                                        [0],
            pos_final_spring[1], width=3, fill="green")
            pos_s1 = w.coords(spring_1)
            spring_3 = w.create_line(70, y0 - 20, 90, y0 - 40,
                                        width=3, fill="green"
                                        )
            pos_s3 = w.coords(spring_3)
            spring_2 = w.create_line(pos_s3[2], pos_s3[3], pos_s1[0
                                        ], pos_s1[1], width=3
                                        , fill="green")
            pos_s2 = w.coords(spring_2)

        master.update()


    pool.close()
    return (pop, log, hof)
```

```
flag = False
flag2 = False
flag3 = False
pas = False
fitness_old = 10e10
penalty=0

def evaluate(individual):
"""The evaluate function is the fundamental function in which
                                    individuals are evaluated in
                                    order to define how an individual
                                     is able to solve the problem. In
                                     this problem, the fitness
                                    function is defined to reduce the
                                     error between the current states
                                     and the desired states through
                                    the adoption of the Integral of
                                    Absolute Error method."""
    global flag, flag2, flag3, pas, old, fitness_old, rise_time,
                                        total_time_simulation,
                                        force_constraint,penalty

    old = 0

    flag = False
    flag2 = False
    flag3 = False
    pas = False
    penalty=0

    func = toolbox.compile(expr=individual)

    def sys(t, x):
        global a0, a1, a2, old, flag, flag2, flag3,penalty

        y = x[0]
        dydt = x[1]
        dyi = x[2]

        dxdt = [1e-05, 1e-05, 1e-05]

        fix = old

        r = set_interp(t)

        e = r - y

        value = func(e, -dydt, dyi)

        dxdt[0] = dydt
        dxdt[2] = e

        if  value > -force_constraint and value < force_constraint:
            dxdt[1] = (- a1 * dydt - a2 * y + value) / a0
            old = value
            flag3 = True
        else:
            dxdt[1] = (- a1 * dydt - a2 * y + fix) / a0
            flag2 = True
            penalty += abs(force_constraint-value)
```

```
        return [dxdt[0], dxdt[1], dxdt[2]]

    passs = (total_time_simulation) * 50
    tev = np.linspace(1e-05, total_time_simulation, int(passs))
    x_ini = [1e-05, 1e-05, 1e-05]  # initial conditions

    try:
        with timeout(10, exception=RuntimeError):
            sol = solve_ivp(sys, [1e-05, total_time_simulation],
                                                x_ini, t_eval=tev)
    except RuntimeError:
        return np.random.uniform(fitness_old * 1.2, fitness_old * 1
                                            .6),


    yy = sol.y[0, :]
    dyy = sol.y[1, :]
    tt = sol.t

    set_point_off = []
    for items in tt:
        set_point_off.append(set_interp(items))
    r = set_point_off

    err1 = r - yy

    i = 0
    step = []
    step.append(tt[1] - tt[0])
    while i < len(tt) - 1:
        step.append(tt[i + 1] - tt[i])
        i = i + 1

    #INTEGRAL OF ABSOLUTE ERROR
    IAE = []
    alpha = 0.1
    for p, m, n in zip(err1, dyy, step):
        IAE.append(n * (abs(p) + alpha * abs(m)))


    if flag2 is True:
        pas = True
        x = sum(IAE) + penalty

    if flag3 is True and flag2 is False:
        fitness = sum(IAE)
        if fitness < fitness_old:
            fitness_old = fitness

    return (x,) if pas is True else (fitness,)

"""The Primitive set is determined in the following code lines. The
                                 mathematical operations,
                                constants and variables are
                                defined. Such step is very
                                important for the algorithm
                                because with such definition the
                                size of search space is
                                automatically defined."""
pset = gp.PrimitiveSet("MAIN", 3)
pset.addPrimitive(np.add, 2, name="Add")
pset.addPrimitive(np.subtract, 2,name="Sub")
```

```
pset.addPrimitive(np.multiply, 2,name="Mul")
pset.addPrimitive(np.abs, 1,name="Abs")
pset.addPrimitive(Log, 1)
pset.addPrimitive(Tanh, 1)
pset.addPrimitive(Sqrt, 1)
pset.addTerminal(np.pi, name="pi")
pset.addTerminal(np.e, name="e")
pset.addEphemeralConstant("rand101", lambda: round(random.uniform(-
                                    100, 100), 4))
pset.addEphemeralConstant("rand102", lambda: round(random.uniform(-
                                    100, 100), 4))
pset.addEphemeralConstant("rand103", lambda: random.randint(0, 9))
pset.addEphemeralConstant("rand104", lambda: random.randint(0, 9))
pset.renameArguments(ARG0='err')
pset.renameArguments(ARG1='d_err')
pset.renameArguments(ARG2='i_err')

creator.create("Fitness", base.Fitness, weights=(-1.0,))
creator.create("Individual", gp.PrimitiveTree, fitness=creator.
                                    Fitness)
toolbox = base.Toolbox()
toolbox.register("expr", gp.genFull, pset=pset, min_=1, max_=2)
toolbox.register("individual", tools.initIterate, creator.
                                    Individual, toolbox.expr)
toolbox.register("population", tools.initRepeat, list, toolbox.
                                    individual)
toolbox.register("compile", gp.compile, pset=pset)
toolbox.register("evaluate", evaluate)
toolbox.register("select", tools.selTournament,tournsize=3)
toolbox.register("mate", gp.cxOnePoint)
toolbox.register("mutate", gp.mutUniform, expr=toolbox.expr, pset=
                                    pset)
toolbox.decorate("mate", gp.staticLimit(key=attrgetter("height"),
                                    max_value=limit_height))
toolbox.decorate("mutate", gp.staticLimit(key=attrgetter("height"),
                                     max_value=limit_height))
toolbox.decorate("mate", gp.staticLimit(key=len, max_value=
                                    limit_size))
toolbox.decorate("mutate", gp.staticLimit(key=len, max_value=
                                    limit_size))
# history = tools.History()
# toolbox.decorate("mate", history.decorator)
# toolbox.decorate("mutate", history.decorator)

if __name__ == "__main__":
    pop, log, hof = main()
```

D.2 MSD _ ONLINE.py

```
/* This Source Code Form is subject to the terms of the Mozilla
                                    Public
 * License, v. 2.0. If a copy of the MPL was not distributed with
                                     this
 * file, You can obtain one at http://mozilla.org/MPL/2.0/. */
/*
------------ Copyright (C) 2019 University of Strathclyde----------
--------------------- Author: Nicola Menga ------------------------
-------------- e-mail: nicola.menga@strath.ac.uk-------------------
------------------------nicola.menga@studenti.polito.it-------------
```

```
*/

import _pickle as cPickle
import pickle
from scipy.integrate import solve_ivp
from scipy.interpolate import PchipInterpolator
import numpy as np
from numpy import *
import networkx as nx
from networkx.drawing.nx_agraph import graphviz_layout
import pygraphviz as pgv
import random
from deap import gp
import matplotlib.pyplot as plt
import sys
import timeit
import time
import pandas as pd
from functools import reduce
from operator import *
from deap import algorithms
from deap import base
from deap import creator
from deap import tools
import multiprocessing
from interruptingcow import timeout
import matplotlib.animation as animation
from matplotlib import style
import itertools
from functools import wraps
from time import time

def Sqrt(a):
        if a > 0:
            return np.sqrt(a)
        else:
            return np.abs(a)


def Log(a):
        if a > 0:
            return np.log(a)
        else:
            return np.abs(a)


def Tanh(a):
        return np.tanh(a)

def top_endstop(t, top_end_stop):
    return top_end_stop

def bot_endstop(t, bottom_end_stop):
    return bottom_end_stop

def setpoint(t):
    if (t > 0 and t < 60):
        r = 1
    elif (t > 60 and t < 150):
        r = 2
    elif (t > 150 and t < 200):
        r = 4
```

```
    elif (t > 200 and t < 300):
        r = 5
    elif (t > 300 and t < 450):
        r = 6
    elif (t > 450 and t < 601):
        r = 9
    else:
        r = 0

    return r

a0 = 1
a1 = 10
a2 = 20

top_end_stop = 10
bottom_end_stop = -1

rise_time = 1

total_time_simulation = 600

tempo = np.linspace(1e-05, total_time_simulation,
                                    total_time_simulation * 50)

set_point = []
for items in tempo:
    set_point.append(setpoint(items))

set_interp = PchipInterpolator(tempo, set_point)
r_max = np.amax(set_point)
r_min = np.amin(set_point)

if r_max > top_end_stop:
    print("MAXIMUM POSITION CONSENTED: %d [m]" % top_end_stop)
    print("COMMAND NOT GRANTED by the presence of the end stop")
if r_min < bottom_end_stop:
    print("MINIMUM POSITION CONSENTED: %d [m]" % bottom_end_stop)
    print("COMMAND NOT GRANTED by the presence of the end stop")
    sys.exit(0)

ar = 2 * r_max / (rise_time ** 2)
vr = ar * rise_time
yr = 0.9 * r_max
force_constraint = a1 * vr + a2 * yr + a0 * ar

old = 0

size_pop = 150
size_gen = 100
Mu = int(size_pop)
Lambda = int(size_pop * 1.5)
pb_cross = 0.6
pb_mut = 0.1
limit_height = 17
limit_size = 400
flag_offdesign = False



def main():
```

```
"""The main function is defined as a container in order to execute
                                    the evolutionary process. Here
                                    the type of algorithm to be used
                                    is chosen like the probability of
                                     crossover and mutation. Moreover
                                     the post processing of the
                                    results as graph of position mass
                                     and the forms of the trees are
                                    inserted in this main function.
                                    """
    pool = multiprocessing.Pool(12)
    global flag_offdesign
    if flag_offdesign is True:
        toolx.register("map", pool.map)
    else:
        toolbox.register("map", pool.map)

    global size_gen, size_pop, Mu, Lambda, pb_mut, pb_cross,
                                        random_individual

    random.seed()
    print("INITIAL POP SIZE: %d" % size_pop)

    print("GEN SIZE: %d" % size_gen)

    print("\n")

    if flag_offdesign is True:
        pop2 = toolx.population()
        pop3 = toolx.popx(random_individual)
        final_pop= pop2 + pop3
        individualss = [ind for ind in final_pop]
        fitnesses = toolx.map(toolx.evaluate, individualss)
        for ind, fit in zip(individualss, fitnesses):
            ind.fitness.values = fit
        hofx = tools.HallOfFame(size_gen)
        Mux = int(len(final_pop))
        Lambdax = int(len(final_pop)*1.5)
    else:
        pop = toolbox.population(n=size_pop)
        hof = tools.HallOfFame(size_gen)
        Mu = int(size_pop)
        Lambda = int(size_pop * 1.5)

    stats_fit = tools.Statistics(lambda ind: ind.fitness.values)
    stats_size = tools.Statistics(len)
    stats_height = tools.Statistics(attrgetter("height"))
    mstats = tools.MultiStatistics(fitness=stats_fit, height=
                                        stats_height, size=stats_size
                                        )
    mstats.register("avg", np.mean, axis=0)
    mstats.register("min", np.min, axis=0)
    mstats.register("max", np.max, axis=0)

    if flag_offdesign is True:
        popx, logx = algorithms.eaMuPlusLambdaTol(individualss,
                                            toolx, Mux, Lambdax,
                                            pb_cross, pb_mut,
                                            size_gen, 15 , stats=
                                            mstats, halloffame=hofx,
                                            verbose=True)
    else:
```

```
        pop, log = algorithms.eaMuPlusLambda(pop, toolbox, Mu,
                                             Lambda, pb_cross, pb_mut,
                                              size_gen, stats=mstats,
                                             halloffame=hof, verbose=
                                             True)
    if flag_offdesign is True:
        expr = hofx[0]
        nodes, edges, labels = gp.graph(expr)
        g = pgv.AGraph()
        g.add_nodes_from(nodes)
        g.add_edges_from(edges)
        g.layout(prog="dot")
        for i in nodes:
            n = g.get_node(i)
            n.attr["label"] = labels[i]
        g.draw("tree_NEW_IC.png")

        gen = logx.select("gen")
        fit_min = logx.chapters["fitness"].select('min')

        perform = []
        p = 0
        for items in fit_min:
            perform.append(fit_min[p][0])
            p = p + 1

        height_avgs = logx.chapters["height"].select("avg")
        size_avgs = logx.chapters["size"].select("avg")
        fig, ax1 = plt.subplots()
        line1 = ax1.plot(gen, perform, "b-", label="Minimum Fitness
                                              Performance")
        ax1.set_xlabel("Generation")
        ax1.set_ylabel("Fitness", color="b")
        for tl in ax1.get_yticklabels():
            tl.set_color("b")

        ax2 = ax1.twinx()
        line2 = ax2.plot(gen, size_avgs, "r-", label="Average Size"
                                             )
        line3 = ax2.plot(gen, height_avgs, "g-", label="Average
                                             Height")
        ax2.set_ylabel("Size, Height", color="y")
        for tl in ax2.get_yticklabels():
            tl.set_color("y")

        lns = line1 + line2 + line3
        labs = [l.get_label() for l in lns]
        ax1.legend(lns, labs, loc="center right")

        plt.savefig('Stats_ONLINE')
        plt.close()
    else:
        expr = hof[0]
        nodes, edges, labels = gp.graph(expr)
        g = pgv.AGraph()
        g.add_nodes_from(nodes)
        g.add_edges_from(edges)
        g.layout(prog="dot")
        for i in nodes:
            n = g.get_node(i)
            n.attr["label"] = labels[i]
        g.draw("tree_old.png")
```

```
        gen = log.select("gen")

        fit_min = log.chapters["fitness"].select('min')

        perform = []
        p = 0
        for items in fit_min:
            perform.append(fit_min[p][0])
            p = p + 1

        height_avgs = log.chapters["height"].select("avg")
        size_avgs = log.chapters["size"].select("avg")
        fig, ax1 = plt.subplots()
        line1 = ax1.plot(gen, perform, "b-", label="Minimum Fitness
                                            Performance")
        ax1.set_xlabel("Generation")
        ax1.set_ylabel("Fitness", color="b")
        for tl in ax1.get_yticklabels():
            tl.set_color("b")

        ax2 = ax1.twinx()
        line2 = ax2.plot(gen, size_avgs, "r-", label="Average Size"
                                            )
        line3 = ax2.plot(gen, height_avgs, "g-", label="Average
                                           Height")
        ax2.set_ylabel("Size, Height", color="y")
        for tl in ax2.get_yticklabels():
            tl.set_color("y")

        lns = line1 + line2 + line3
        labs = [l.get_label() for l in lns]
        ax1.legend(lns, labs, loc="center right")

        plt.savefig('Stats_OFFLINE')
        plt.close()

    pool.close()
    return (popx, logx, hofx) if flag_offdesign is True else (pop,
                                        log, hof)

flag = False
flag2 = False
flag3 = False
pas = False
fitness_old = 10e10
penalty=0

def evaluate(individual):
"""The evaluate function is the fundamental function in which
                                    individuals are evaluated in
                                    order to define how an individual
                                     is able to solve the problem. In
                                     this problem, the fitness
                                    function is defined to reduce the
                                     error between the current states
                                     and the desired states through
                                    the adoption of the Integral of
                                    Absolute Error method."""
    global flag, flag2, flag3, pas, old, fitness_old, rise_time,
                                        total_time_simulation,
                                        force_constraint,penalty ,
```

```
                                        flag_offdesign

old = 0

flag = False
flag2 = False
flag3 = False
pas = False
penalty=0

if flag_offdesign is True:
    func = toolx.compile(expr=individual)
else:
    func = toolbox.compile(expr=individual)

def sys(t, x):

    global a0, a1, a2, old, flag, flag2, flag3,penalty,
                                            force_constraint

    y = x[0]
    dydt = x[1]
    dyi = x[2]

    dxdt = [1e-05, 1e-05, 1e-05]

    fix = old

    r = set_interp(t)
    e = r - y

    value = func(e, -dydt, dyi)

    dxdt[0] = dydt
    dxdt[2] = e

    if value > -force_constraint and value < force_constraint:
        dxdt[1] = (- a1 * dydt - a2 * y + value) / a0
        old = value
        flag3 = True
    else:
        dxdt[1] = (- a1 * dydt - a2 * y + fix) / a0
        flag2 = True
        penalty += abs(force_constraint-value)

    return [dxdt[0], dxdt[1], dxdt[2]]

if flag_offdesign is True:
    passintt = (total_time_simulation - (change_time))*50
    tev = np.linspace(change_time, total_time_simulation, int(
                                            passintt))
    try:
        with timeout(5, exception=RuntimeError):
            sol = solve_ivp(sys, [change_time,
                                                  total_time_simulation
                                                  ], xnew_ini,
                                                  t_eval=tev)
    except RuntimeError:
        return 100000,
else:
    passs = (total_time_simulation) * 50
    tev = np.linspace(1e-05, total_time_simulation, int(passs))
```

```
        x_ini = [1e-05, 1e-05, 1e-05]
        try:
            with timeout(7, exception=RuntimeError):
                sol = solve_ivp(sys, [1e-05, total_time_simulation]
                                                    , x_ini,t_eval=
                                                    tev)
        except RuntimeError:
            flag=True
            return 100000,

    yy = sol.y[0, :]
    dyy = sol.y[1, :]
    tt = sol.t

    set_point_off = []
    for items in tt:
        set_point_off.append(set_interp(items))
    r = set_point_off

    err1 = r - yy

    i = 0
    step = []
    step.append(tt[1] - tt[0])
    while i < len(tt) - 1:
        step.append(tt[i + 1] - tt[i])
        i = i + 1

    IAE = []
    alpha = 0.1
    for p, m, n in zip(err1, dyy, step):
        IAE.append(n * (abs(p) + alpha * abs(m)))

    if flag2 is True:
        pas = True
        x = sum(IAE) + penalty


    if flag3 is True and flag2 is False:
        fitness = sum(IAE)
        if fitness < fitness_old:
            fitness_old = fitness

    return (x,) if pas is True else (fitness,)

"""The Primitive set is determined in the following code lines. The
                                    mathematical operations,
                                constants and variables are
                                defined. Such step is very
                                important for the algorithm
                                because with such definition the
                                size of search space is
                                automatically defined."""
pset = gp.PrimitiveSet("MAIN", 3)
pset.addPrimitive(np.add, 2)
pset.addPrimitive(np.subtract, 2)
pset.addPrimitive(np.multiply, 2)
pset.addPrimitive(np.abs, 1)
pset.addPrimitive(Log, 1)
pset.addPrimitive(Tanh, 1)
pset.addPrimitive(Sqrt, 1)
pset.addTerminal(np.pi, name="pi")
```

```
pset.addTerminal(np.e, name="e")
pset.addEphemeralConstant("rand101", lambda: round(random.uniform(-
                                    100, 100), 4))
pset.addEphemeralConstant("rand102", lambda: round(random.uniform(-
                                    100, 100), 4))
pset.addEphemeralConstant("rand103", lambda: random.randint(0, 9))
pset.addEphemeralConstant("rand104", lambda: random.randint(0, 9))
pset.renameArguments(ARG0='err')
pset.renameArguments(ARG1='d_err')
pset.renameArguments(ARG2='i_err')

creator.create("Fitness", base.Fitness, weights=(-1.0,))
creator.create("Individual", gp.PrimitiveTree, fitness=creator.
                                    Fitness)
toolbox = base.Toolbox()
toolbox.register("expr", gp.genFull, pset=pset, min_=1, max_=2)
toolbox.register("individual", tools.initIterate, creator.
                                    Individual, toolbox.expr)
toolbox.register("population", tools.initRepeat, list, toolbox.
                                    individual)
toolbox.register("compile", gp.compile, pset=pset)
toolbox.register("evaluate", evaluate)
toolbox.register("select", tools.selTournament,tournsize=3)
toolbox.register("mate", gp.cxOnePoint)
toolbox.register("mutate", gp.mutUniform, expr=toolbox.expr, pset=
                                    pset)
toolbox.decorate("mate", gp.staticLimit(key=attrgetter("height"),
                                    max_value=limit_height))
toolbox.decorate("mutate", gp.staticLimit(key=attrgetter("height"),
                                     max_value=limit_height))
toolbox.decorate("mate", gp.staticLimit(key=len, max_value=
                                    limit_size))
toolbox.decorate("mutate", gp.staticLimit(key=len, max_value=
                                    limit_size))

if __name__ == "__main__":
    pop, log, hof = main()

best_oldfit=toolbox.evaluate(hof[0])
print(best_oldfit)

a0_old = a0
a1_old = a1
a2_old = a2

change_time = 90  # [s]
print("\n ADD SOME ALTERATIONS TO PHYSICAL COMPONENTS OF THE PLANT
                                    AT %.2f [s]" % change_time)
print("WRITE THE NUMBER FOR THE PARAMETER THAT YOU WANT CHANGE:
                                    MASS ( 1 ), SPRING ( 2 ), DAMPER
                                    ( 3 )")
flag = int(input())
if flag == 1:
    a0 = float(input("CHANGE VALUE OF THE MASS in [Kg] : "))
elif flag == 2:
    a1 = float(input("CHANGE VALUE OF THE SPRING in [N/m] : "))
elif flag == 3:
    a2 = float(input("CHANGE VALUE OF THE DAMPER in [Ns/m] : "))
else:
    print("ERROR, please digit 1 for mass, 2 for spring or 3 for
                                        damper")
    sys.exit[0]
```

```
ar = 2 * r_max / (rise_time ** 2)
vr = ar * rise_time
yr = 0.9 * r_max
force_constraint = a1 * vr + a2 * yr + a0 * ar

x_ini = [1e-05, 1e-05, 1e-05]


def sys2GP(t, x):
    global a0_old, a1_old, a2_old
    M, C, K = a0_old, a1_old, a2_old

    y = x[0]
    dydt = x[1]
    ei = x[2]
    dxdt = [1e-05, 1e-05, 1e-05]

    r = set_interp(t)
    e = r - y

    ctrl = toolbox.compile(hof[0])
    u = ctrl(e, -dydt, ei)

    dxdt[1] = (- C * dydt - K * y + u) / M
    dxdt[0] = dydt
    dxdt[2] = e

    return [dxdt[0], dxdt[1], dxdt[2]]

passint = (total_time_simulation) * 4
tevals = np.linspace(1e-05, total_time_simulation, int(passint))

solgp = solve_ivp(sys2GP, [1e-05, total_time_simulation], x_ini,
                                    t_eval=tevals)
ygp = solgp.y[0, :]
dyy = solgp.y[1, :]
ttgp = solgp.t

rrr = []
for i in ttgp:
    rrr.append(set_interp(i))

tes = np.zeros(len(ttgp), dtype='float')

ii = 0
for i in ttgp:
    tes[ii] = top_endstop(i, top_end_stop)
    ii = ii + 1
errgp = rrr - ygp

bes = np.zeros(len(ttgp), dtype='float')

ii = 0
for i in ttgp:
    bes[ii] = bot_endstop(i, bottom_end_stop)
    ii = ii + 1

figManager = plt.get_current_fig_manager()
figManager.resize(*figManager.window.maxsize())
plt.ion()
plt.plot(ttgp, rrr, 'r--', label="SET POINT [m]")
```

```
plt.plot(ttgp, tes, 'k', label="TOP END-STOP  [m]")
plt.plot(ttgp, bes, 'k', label="BOTTOM END-STOP  [m]")
animated_plot = plt.plot(ttgp, ygp, 'r.', label="ON DESIGN")[0]

i = 0
x=0
for items in ttgp:
    plt.figure(1)
    plt.ylim(bottom_end_stop - 1, top_end_stop + 1)
    plt.xlim(0, total_time_simulation)

    if items > change_time:
        index, = np.where(ttgp == items)
        break
    animated_plot.set_xdata(ttgp[0:i])
    animated_plot.set_ydata(ygp[0:i])
    plt.draw()
    if (i%5)==0:
        plt.savefig('video/'+"img" + str(x) + ".png")
        x=x+1
    plt.pause(0.00000001)
    i = i + 1

xnew_ini = [float(ygp[index]), float(dyy[index]), float(errgp[index
                                    ])]
u_design = hof[0]
init_height=hof[0].height

print("Initial heigh:")
print(init_height)


print("\n OFF DESIGN CONTROLLER")
print(u_design)
print("\n \n")

output = open("hof_MSD.pkl", "wb")
cPickle.dump(hof, output, -1)
output.close()

objects = []
with (open("hof_MSD.pkl", "rb")) as openfile:
    while True:
        try:
            objects.append(pickle.load(openfile))
        except EOFError:
            break

start = time()
flag_gpnew = True

if __name__ == "__main__":
    random_individual=int(size_gen)
    flag_seed_populations = True
    flag_seed_populations1 = True
    flag_offdesign = True
    size_pop, size_gen, pb_mut, pb_cross = size_gen, 5, 0.5, 0.1


    def initPOP1():
        global objects
        return objects[0]
```

```
    toolx = base.Toolbox()

    toolx.register("expr", gp.genHalfAndHalf, pset=pset, min_=
                                    init_height, max_=init_height
                                    +3)
    toolx.register("individual", tools.initIterate, creator.
                                    Individual, toolx.expr)
    toolx.register("popx", tools.initRepeat, list, toolx.individual
                                    )
    toolx.register("population", tools.initIterate, list, initPOP1)
    toolx.register("compile", gp.compile, pset=pset)
    toolx.register("evaluate", evaluate)
    toolx.register("select", tools.selTournament, tournsize=5)
    toolx.register("mate", gp.cxOnePoint)
    toolx.register("mutate", gp.mutUniform, expr=toolbox.expr, pset
                                    =pset)
    toolx.decorate("mate", gp.staticLimit(key=attrgetter("height"),
                                     max_value=limit_height))
    toolx.decorate("mutate", gp.staticLimit(key=attrgetter("height"
                                    ), max_value=limit_height))
    toolx.decorate("mate", gp.staticLimit(key=len, max_value=
                                    limit_size))
    toolx.decorate("mutate", gp.staticLimit(key=len, max_value=
                                    limit_size))

    popx, logx, hofx = main()
end = time()
t_offdesign = end - start
print("RUNTIME GENETIC PROGRAMMINF FOR IC:")
print(t_offdesign)

print("\n NEW INTELLIGENT ON DESIGN CONTROL LAW:")
print(hofx[0])
print("\n")


def sys2GP_c(t, x):
    global a0, a1, a2, u_design
    M, C, K = a0, a1, a2

    y = x[0]
    dydt = x[1]
    ei = x[2]
    dxdt = [1e-05, 1e-05, 1e-05]

    r = set_interp(t)
    e = r - y

    ctrl = toolbox.compile(u_design)
    u = ctrl(e, -dydt, ei)

    dxdt[1] = (- C * dydt - K * y + u) / M
    dxdt[0] = dydt
    dxdt[2] = e

    return [dxdt[0], dxdt[1], dxdt[2]]


passint_c = (change_time + t_offdesign - (change_time)) * 4
tevals_c = np.linspace(change_time, change_time + t_offdesign, int(
                                    passint_c))
```

```
passint_wwc = (total_time_simulation- (change_time)) * 8
tevals_wwc = np.linspace(change_time, total_time_simulation, int(
                                    passint_c))

solgp_c = solve_ivp(sys2GP_c, [change_time, change_time +
                                    t_offdesign], xnew_ini, t_eval=
                                    tevals_c)
solgp_wwc = solve_ivp(sys2GP_c, [change_time, total_time_simulation
                                    ], xnew_ini, t_eval=tevals_wwc)

ygp_c = solgp_c.y[0, :]
dyy_c = solgp_c.y[1, :]
ttgp_c = solgp_c.t



ygp_wwc = solgp_wwc.y[0, :]
dyy_wwc = solgp_wwc.y[1, :]
ttgp_wwc = solgp_wwc.t


rrr_c = []
for i in ttgp_c:
    rrr_c.append(set_interp(i))

errgp_c = rrr_c - ygp_c

plt.ion()
animated_plot_c = plt.plot(ttgp_c, ygp_c, 'b.', label="OFF DESIGN")
                                    [0]


i = 0
for items in ttgp_c:
    plt.figure(1)
    if ygp_c[i] >= top_end_stop:
        print("WARNING: position of system exceed the top end stop:
                                        TRY WITH A SMALLER MASS"
                                    )
    if ygp_c[i] <= bottom_end_stop:
        print("WARNING: position of system exceed the top end stop:
                                        TRY WITH A SMALLER MASS"
                                    )
    animated_plot_c.set_xdata(ttgp_c[0:i])
    animated_plot_c.set_ydata(ygp_c[0:i])
    plt.draw()
    if (i%5)==0:
        plt.savefig('video/'+"img" + str(x) + ".png")
        x=x+1
    plt.pause(0.00000001)
    i = i + 1

ttgp_wwcc=[]
ygp_wwcc=[]

for t,y in zip(ttgp_wwc,ygp_wwc):
    if t > t_offdesign+change_time:
        ttgp_wwcc.append(t)
        ygp_wwcc.append(y)
```

```
plt.ion()
animated_plot_wc = plt.plot(ttgp_wwcc, ygp_wwcc, 'm.', label="OFF
                                    DESIGN - NO IC")[0]
i = 0
for itmemms in ttgp_wwcc:
    plt.figure(1)
    animated_plot_wc.set_xdata(ttgp_wwcc[0:i])
    animated_plot_wc.set_ydata(ygp_wwcc[0:i])
    if ygp_wwcc[i] >= top_end_stop:
        print("WARNING: position of system exceed the top end stop"
                                            )
        plt.draw()
        plt.plot(itmemms,ygp_wwcc[i],"*")
        break
    if ygp_wwcc[i] <= bottom_end_stop:
        print("WARNING: position of system exceed the top end stop"
                                            )
        plt.draw()
        plt.plot(itmemms, ygp_wwcc[i], "*")

        break
    plt.draw()
    if i%5==0:
        plt.savefig('video/'+"img" + str(x) + ".png")
        x=x+1
    plt.pause(0.001)
    i = i + 1

passint_IC = (total_time_simulation - (change_time + t_offdesign))
                                    * 4
tevals_IC = np.linspace(change_time + t_offdesign,
                                    total_time_simulation, int(
                                    passint_IC))

xnew_ini_gp = [float(ygp_c[-1]), float(dyy_c[-1]), float(errgp_c[-1
                                    ])]


def sys2GP_IC(t, x):
    global a0, a1, a2
    M, C, K = a0, a1, a2

    y = x[0]
    dydt = x[1]
    ei = x[2]
    dxdt = [1e-05, 1e-05, 1e-05]

    r = set_interp(t)
    e = r - y

    ctrl = toolx.compile(hofx[0])
    u = ctrl(e, -dydt, ei)

    dxdt[1] = (- C * dydt - K * y + u) / M
    dxdt[0] = dydt
    dxdt[2] = e

    return [dxdt[0], dxdt[1], dxdt[2]]


solgp_IC = solve_ivp(sys2GP_IC, [change_time + t_offdesign,
                                    total_time_simulation],
```

```
                                    xnew_ini_gp, t_eval=tevals_IC)

ygp_IC = solgp_IC.y[0, :]
dyy_IC = solgp_IC.y[1, :]
ttgp_IC = solgp_IC.t

plt.ion()
animated_plot_gp = plt.plot(ttgp_IC, ygp_IC, 'g.', label="
                                    INTELLIGENT CONTROL")[0]



i = 0
for items in ttgp_IC:
    plt.figure(1)
    if items == change_time + t_offdesign:
        plt.legend(loc='best')
    animated_plot_gp.set_xdata(ttgp_IC[0:i])
    animated_plot_gp.set_ydata(ygp_IC[0:i])
    plt.draw()
    plt.pause(0.00000001)
    if (i%5)==0:
        plt.savefig('video/'+"img" + str(x) + ".png")
        x=x+1
    i = i + 1
plt.show(block=True)
```

D.3 GODDARD _ OFFLINE.py

```
/* This Source Code Form is subject to the terms of the Mozilla
                                    Public
 * License, v. 2.0. If a copy of the MPL was not distributed with
                                     this
 * file, You can obtain one at http://mozilla.org/MPL/2.0/. */
/*
----- Copyright (C) 2019 University of Strathclyde and Authors ----
------------------- Author: Francesco Marchetti ------------------
-------------- e-mail: francesco.marchetti@strath.ac.uk -----------
--------------------- Author: Nicola Menga --------------------------
-------------- e-mail: nicola.menga@strath.ac.uk -------------------
------------------------nicola.menga@studenti.polito.it-------------
*/

from scipy.integrate import solve_ivp
import numpy as np
import operator
import pygraphviz as pgv
import random
from deap import gp
import matplotlib.pyplot as plt
import sys
import timeit
from deap import algorithms
from deap import base
from deap import creator
from deap import tools
from deap.algorithms import varOr
import multiprocessing
from scipy.interpolate import PchipInterpolator
```

```
import matplotlib.animation as animation
from matplotlib import style
import datetime
from time import time
from functools import partial
import _pickle as cPickle
import pickle
from interruptingcow import timeout

def TriAdd(x, y, z):
    return x + y + z
def Abs(x):
    return abs(x)
def Tanh(a):
    return np.tanh(a)
def Mul(left, right):
    try:
        return left * right
    except (RuntimeError, RuntimeWarning, TypeError,
                                       ArithmeticError, BufferError,
                                        BaseException, NameError,
                                       ValueError,
            FloatingPointError, OverflowError):
        return left
def Sqrt(x):
    try:
        if x > 0:
            return np.sqrt(x)
        else:
            return abs(x)
    except (RuntimeError, RuntimeWarning, TypeError,
                                       ArithmeticError, BufferError,
                                        BaseException, NameError,
                                       ValueError):
        return 0

def xmate(ind1, ind2):
    i1 = random.randrange(len(ind1))
    i2 = random.randrange(len(ind2))
    ind1[i1], ind2[i2] = gp.cxOnePoint(ind1[i1], ind2[i2])
    return ind1, ind2
def xmut(ind, expr):

    i1 = random.randrange(len(ind))
    i2 = random.randrange(len(ind[i1]))
    choice = random.random()
    indx = gp.mutUniform(ind[i1], expr, pset=pset)
    ind[i1] = indx[0]
    return ind,

start = timeit.default_timer()

class Rocket:

    def __init__(self):
        self.GMe = 3.986004418 * 10 ** 14 #Earth gravitational
                                            constant [m^3/s^2]
        self.Re = 6371.0 * 1000       #Earth Radius [m]
        self.Vr = np.sqrt(self.GMe / self.Re)            #[m/s]
        self.H0 = 10.0                                   #[m]
        self.V0 = 0.0
        self.M0 = 100000.0                               #[kg]
```

```
        self.Mp = self.M0 * 0.99
        self.Cd = 0.6
        self.A = 4.0                                        #[m2]
        self.Isp = 300.0                                    #[s]
        self.g0 = 9.80665                                   #[m/s2]
        self.Tmax = self.M0 * self.g0 * 1.5
        self.MaxQ = 14000.0                                 #[Pa]
        self.MaxG = 8.0
        self.Htarget = 400.0 * 1000                         #[m]
        self.Rtarget = self.Re + self.Htarget               #[m/s]
        self.Vtarget = np.sqrt(self.GMe / self.Rtarget)     #[m/s]

    @staticmethod
    def air_density(h):
        global flag
        beta = 1 / 8500.0
        rho0 = 1.225
        try:
            return rho0 * np.exp(-beta * h)
        except RuntimeWarning:
            flag = True
            return rho0 * np.exp(-beta * obj.Rtarget)

Nstates = 5
Ncontrols = 2
obj = Rocket()
nEph = 2
mutpb = 0.25
cxpb = 0.55
change_time = 200

size_pop = 700            #Population size
size_gen = 900            #Generation size
Mu = int(size_pop)
Lambda = int(size_pop * 1.4)

limit_height = 17         #Max height
limit_size = 400          #Max size

nbCPU = multiprocessing.cpu_count()

tref = np.load("time.npy")
total_time_simulation = tref[-1]
del tref

flag=False
flag_time=False
flag_notgood=False
feasible=True

def main():
"""The main function is defined as a container in order to execute
                                    the evolutionary process. Here
                                    the type of algorithm to be used
                                    is chosen like the probability of
                                     crossover and mutation. Moreover
                                     the post processing of the
                                    results as for example the graphs
                                     of position mass, the statistics
                                     of each run and the forms of the
                                     trees are inserted in this main
                                    function."""
```

```
global flag_seed_populations, flag_seed_populations1
global tfin, flag
global size_gen, size_pop, Mu, Lambda, mutpb, cxpb
global Rfun, Thetafun, Vrfun, Vtfun, mfun

flag = False
fit_old = [1e5, 1e5, 1e5, 1e5, 1e5]
Rref = np.load("R.npy")
Thetaref = np.load("Theta.npy")
Vrref = np.load("Vr.npy")
Vtref = np.load("Vt.npy")
mref = np.load("m.npy")
tref = np.load("time.npy")
tfin = tref[-1]

Rfun = PchipInterpolator(tref, Rref)
Thetafun = PchipInterpolator(tref, Thetaref)
Vrfun = PchipInterpolator(tref, Vrref)
Vtfun = PchipInterpolator(tref, Vtref)
mfun = PchipInterpolator(tref, mref)

del Rref, Thetaref, Vrref, Vtref, mref, tref

pool = multiprocessing.Pool(nbCPU)

toolbox.register("map", pool.map)
pop = toolbox.population(n=size_pop)

print("INITIAL POP SIZE: %d" % size_pop)

print("GEN SIZE: %d" % size_gen)

print("\n")

random.seed()
history.update(pop)

hof= tools.ParetoFront()

stats_fit = tools.Statistics(lambda ind: ind.fitness.values)
stats_size = tools.Statistics(len)
stats_height = tools.Statistics(operator.attrgetter("height"))
mstats = tools.MultiStatistics(fitness=stats_fit, height=
                                stats_height, size=stats_size
                                )

mstats.register("avg", np.mean, axis=0)
mstats.register("min", np.min, axis=0)
mstats.register("max", np.max, axis=0)

pop, log = algorithms.eaMuPlusLambda(pop, toolbox, Mu, Lambda,
                                mutpb, cxpb, size_gen, stats=
                                mstats, halloffame=hof,
                                verbose=True)

stop = timeit.default_timer()
total_time = stop - start
tformat = str(datetime.timedelta(seconds=int(total_time)))

res = open("HallOfFame_2Cont", "w")
for i in range(len(hof)):
```

```
        res.write("{}: 1".format(i) + str(hof[i][0]) + "\n" + "{}:
                                            2".format(i) + str(hof[i]
                                            [1]) + "\n")
res.close()

gen = log.select("gen")
fit_avg = log.chapters["fitness"].select('min')

perform = []
perform2 = []
perform3 = []
perform4 = []
perform5 = []
p = 0
for items in fit_avg:
    perform.append(fit_avg[p][0])
    perform2.append(fit_avg[p][1])
    perform3.append(fit_avg[p][2])
    perform4.append(fit_avg[p][3])
    perform5.append(fit_avg[p][2])
    p = p + 1

fig, ax1 = plt.subplots()
ax1.plot(gen[1:], perform[1:], "b-", label="Minimum Theta
                                    Fitness Performance")
ax1.plot(gen[1:], perform2[1:], "r-", label="Minimum Position
                                    Fitness Performance")
ax1.plot(gen[1:], perform3[1:], "g-", label="Minimum Vr Fitness
                                     Performance")
ax1.plot(gen[1:], perform4[1:], "k-", label="Minumum Vt Fitness
                                     Performance")
ax1.plot(gen[1:], perform5[1:], "m-", label="Minimum Mass
                                    Fitness Performance")
ax1.set_xlabel("Generation")
ax1.set_ylabel("Fitness", color="b")
ax1.legend(loc="best")
for tl in ax1.get_yticklabels():
    tl.set_color("b")
textstr = ('Total Running Time: {}'.format(tformat))
ax1.text(0.65, 0.9, textstr, transform=ax1.transAxes, fontsize=
                                    10,
         horizontalalignment='right')

plt.savefig('Stats_OFFLINE')
plt.show()

expr1 = hof[0][0]
expr2 = hof[0][1]
nodes1, edges1, labels1 = gp.graph(expr1)
nodes2, edges2, labels2 = gp.graph(expr2)
g1 = pgv.AGraph()
g1.add_nodes_from(nodes1)
g1.add_edges_from(edges1)
g1.layout(prog="dot")
for i in nodes1:
    n = g1.get_node(i)
    n.attr["label"] = labels1[i]
g1.draw("tree1.png")

g2 = pgv.AGraph()
g2.add_nodes_from(nodes2)
g2.add_edges_from(edges2)
```

```
g2.layout(prog="dot")
for i in nodes2:
    n = g2.get_node(i)
    n.attr["label"] = labels2[i]
g2.draw("tree2.png")

image1 = plt.imread('tree1.png')
fig1, ax1 = plt.subplots()
im1 = ax1.imshow(image1)
ax1.axis('off')
image2 = plt.imread('tree2.png')
fig2, ax2 = plt.subplots()
im2 = ax2.imshow(image2)
ax2.axis('off')
plt.show()

sys.stdout.write("TOTAL RUNNING TIME: {} \n".format(tformat))

fTr = toolbox.compile(expr=expr1)
fTt = toolbox.compile(expr=expr2)
x_ini = [obj.Re, 0.0, 0.0, 0.0, obj.M0]

def sys2GP(t, x):
    R = x[0]
    theta = x[1]
    Vr = x[2]
    Vt = x[3]
    m = x[4]

    r = Rfun(t)
    th = Thetafun(t)
    vr = Vrfun(t)
    vt = Vtfun(t)
    mf = mfun(t)

    er = r - R
    et = th - theta
    evr = vr - Vr
    evt = vt - Vt
    em = mf - m

    dxdt = np.zeros(Nstates)

    rho = obj.air_density(R - obj.Re)
    Dr = 0.5 * rho * Vr * np.sqrt(Vr ** 2 + Vt ** 2) * obj.Cd *
                                        obj.A
    Dt = 0.5 * rho * Vt * np.sqrt(Vr ** 2 + Vt ** 2) * obj.Cd *
                                        obj.A
    g = obj.g0 * (obj.Re / R) ** 2
    g0 = obj.g0
    Isp = obj.Isp

    dxdt[0] = Vr
    dxdt[1] = Vt / R
    dxdt[2] = fTr(er, et, evr, evt, em) / m - Dr / m - g + Vt
                                        ** 2 / R
    dxdt[3] = fTt(er, et, evr, evt, em) / m - Dt / m - (Vr * Vt
                                        ) / R
    dxdt[4] = - np.sqrt(fTr(er, et, evr, evt, em) ** 2 + fTt(er
                                        , et, evr, evt, em) ** 2)
                                        / g0 / Isp
```

```
    return dxdt

solgp = solve_ivp(sys2GP, [0.0, tfin], x_ini)
rout = solgp.y[0, :]
thetaout = solgp.y[1, :]
vrout = solgp.y[2, :]
vtout = solgp.y[3, :]
mout = solgp.y[4, :]
tgp = solgp.t

if tgp[-1] < tfin:
    print("integration stopped prematurely")
rR = np.zeros(len(tgp), dtype='float')
tR = np.zeros(len(tgp), dtype='float')
vrR = np.zeros(len(tgp), dtype='float')
vtR = np.zeros(len(tgp), dtype='float')
mR = np.zeros(len(tgp), dtype='float')

ii = 0
for i in tgp:
    rR[ii] = Rfun(i)
    tR[ii] = Thetafun(i)
    vrR[ii] = Vrfun(i)
    vtR[ii] = Vtfun(i)
    mR[ii] = mfun(i)
    ii = ii + 1

fig2, ax2 = plt.subplots()
ax2.set_xlabel("time [s]")
ax2.set_ylabel("altitude [km]")
plt.plot(tgp, (rout-obj.Re)/1e3, label="GENETIC PROGRAMMING")
plt.plot(tgp, (rR-obj.Re)/1e3, 'r--', label="SET POINT")
plt.legend(loc="best")
plt.savefig('Altitude_OFFLINE.png')

fig3, ax3 = plt.subplots()
ax3.set_xlabel("time [s]")
ax3.set_ylabel("flight angle displacement [deg]")
plt.plot(tgp, np.rad2deg(thetaout), label="GENETIC PROGRAMMING"
                                    )
plt.plot(tgp, np.rad2deg(tR), 'r--', label="SET POINT")
plt.legend(loc="best")
plt.savefig('Flight_angle_displacement_OFFLINE.png')

fig4, ax4 = plt.subplots()
ax4.set_xlabel("time [s]")
ax4.set_ylabel("radial velocity [m/s]")
plt.plot(tgp, vrout, label="GENETIC PROGRAMMING")
plt.plot(tgp, vrR, 'r--', label="SET POINT")
plt.legend(loc="best")
plt.savefig('Radial_velocity_OFFLINE.png')

fig5, ax5 = plt.subplots()
ax5.set_xlabel("time [s]")
ax5.set_ylabel("tangential velocity [m/s]")
plt.plot(tgp, vtout, label="GENETIC PROGRAMMING")
plt.plot(tgp, vtR, 'r--', label="SET POINT")
plt.legend(loc="best")
plt.savefig('Tangential_velocity_OFFLINE.png')

fig6, ax6 = plt.subplots()
ax6.set_xlabel("time [s]")
```

```
    ax6.set_ylabel("mass [kg]")
    plt.plot(tgp, mout, label="GENETIC PROGRAMMING")
    plt.plot(tgp, mR, 'r--', label="SET POINT")
    plt.legend(loc="best")
    plt.savefig('Mass_OFFLINE.png')
    plt.show()

    pool.close()
    return pop, log, hof

def evaluate(individual):
"""The evaluate function is the fundamental function in which
                                    individuals are evaluated in
                                    order to define how an individual
                                     is able to solve the problem. In
                                     this problem, the fitness
                                    function is defined to reduce the
                                     error between the current states
                                     and the desired states through
                                    the adoption of the Integral of
                                    Absolute Error method."""
    global flag, flag_notgood
    global Rfun, Thetafun, Vrfun, Vtfun, mfun, Trfun
    global tfin, t_eval2, penalty, fit_old

    penalty = np.zeros((13))
    flag = False
    flag_notgood=False

    fTr = toolbox.compile(expr=individual[0])
    fTt = toolbox.compile(expr=individual[1])

    x_ini = [obj.Re, 0.0, 0.0, 0.0, obj.M0]

    def sys(t, x):
        global penalty, flag, flag_notgood, cd_var
        R = x[0]
        theta = x[1]
        Vr = x[2]
        Vt = x[3]
        m = x[4]

        r = Rfun(t)
        th = Thetafun(t)
        vr = Vrfun(t)
        vt = Vtfun(t)
        mf = mfun(t)

        if R < obj.Re:
            penalty[0] = penalty[0] + abs(R - obj.Re) / obj.Htarget
            R = obj.Re
            flag = True
            if t < 100:
                penalty[0] = penalty[0] + abs(R - obj.Re) / obj.
                                                    Htarget
        elif R > obj.Rtarget:
            penalty[1] = penalty[1] + abs(R - obj.Rtarget) / obj.
                                                Htarget
            R = obj.Rtarget
            flag = True
        if m < obj.M0 - obj.Mp:
```

```
        penalty[2] = penalty[2] + abs(m - (obj.M0 - obj.Mp)) /
                                            obj.M0
        m = obj.M0 - obj.Mp
        flag = True
    elif m > obj.M0:
        penalty[3] = penalty[3] + abs(m - obj.M0) / obj.M0
        m = obj.M0
        flag = True

    if Vr > 1000:
        penalty[4] = penalty[4] + abs(Vr - 1000) / 765.
                                            0826978731632
        Vr = 1000
        flag = True
    elif Vr < -30:
        penalty[4] = penalty[4] + abs(Vr - 30) / 765.
                                            0826978731632
        Vr = -30
        flag = True
    elif Vr < 0 and t < 100:
        penalty[4] = penalty[4] + abs(Vr) / 765.0826978731632
        flag = True
        Vr = 0

    if Vt > 10000:
        penalty[5] = penalty[5] + abs(Vt - 10000) / 7684.
                                            936222117861
        flag = True
        Vt = 10000
    elif Vt < 0:
        penalty[5] = penalty[5] + abs(Vt - 0) / 7684.
                                            936222117861
        Vt = 0
        flag = True

    if theta > 1.0472:
        penalty[6] = penalty[6] + abs(theta - 1.0472) / 0.
                                            8666158523063785
        theta = 1.0472
        flag = True
    elif theta < 0:
        penalty[6] = penalty[6] + abs(theta - 0) / 0.
                                            8666158523063785
        theta = 0
        flag = True

    er = r - R
    et = th - theta
    evr = vr - Vr
    evt = vt - Vt
    em = mf - m

    rho = obj.air_density(R - obj.Re)
    Dr = 0.5 * rho * Vr * np.sqrt(Vr ** 2 + Vt ** 2) * cd_var *
                                            obj.A
    Dt = 0.5 * rho * Vt * np.sqrt(Vr ** 2 + Vt ** 2) * cd_var *
                                            obj.A
    g = obj.g0 * (obj.Re / R) ** 2
    g0 = obj.g0
    Isp = obj.Isp

    Tr = fTr(er, et, evr, evt, em)
```

```
        Tt = fTt(er, et, evr, evt, em)

        if (fTr(er, et, evr, evt, em) > 0 and fTr(er, et, evr, evt,
                                                  em) < obj.Tmax) and (
                fTt(er, et, evr, evt, em) > 0 and fTt(er, et, evr,
                                                      evt, em) < obj.
                                                      Tmax):
            # if (fTr(er, et, em) > 0 and fTr(er, et, em) < obj.
                                                    Tmax) and (fTt(er, et
                                                    , em) > 0 and fTt(er,
                                                     et, em) < obj.Tmax):
            dxdt = np.array((Vr,
                             Vt / R,
                             Tr / m - Dr / m - g + Vt ** 2 / R,
                             Tt / m - Dt / m - (Vr * Vt) / R,
                             -np.sqrt(Tt ** 2 + Tr ** 2) / (g0 *
                                                            Isp)
                                                            ))
        else:
            dxdt = np.array((Vr,
                             Vt / R,
                             0 / m - Dr / m - g + Vt ** 2 / R,
                             0 / m - Dt / m - (Vr * Vt) / R,
                             -np.sqrt(0 ** 2 + 0 ** 2) / (g0 * Isp)
                                                            ))

            flag_notgood = True
        return dxdt

    tin = 0.0
    teval = np.linspace(0.0, tfin, int(tfin*3))
    sol = solve_ivp(sys, [tin, tfin], x_ini, t_eval=teval)
    y1 = sol.y[0, :]
    y2 = sol.y[1, :]
    y3 = sol.y[2, :]
    y4 = sol.y[3, :]
    y5 = sol.y[4, :]
    tt = sol.t

    r = Rfun(tt)
    theta = Thetafun(tt)
    vr = Vrfun(tt)
    vt = Vtfun(tt)
    m = mfun(tt)

    err1 = (r - y1) / obj.Htarget
    err2 = (theta - y2) / 0.8666158523063785
    err3 = (vr - y3)/obj.Vtarget
    err4 = (vt - y4)/obj.Vtarget
    err5 = (m - y5) / obj.M0

    i = 0
    pp = 1
    step = np.zeros(len(y1), dtype='float')
    step[0] = tt[1] - tt[0]
    while i < len(tt) - 1:
        step[pp] = tt[i + 1] - tt[i]
        i = i + 1
        pp = pp + 1

    IAE = np.zeros((5, len(err1)))
    j = 0
```

```
    for a, b, c, d, e, n in zip(err1, err2, err3, err4, err5, step)
                                        :
        IAE[0][j] = n * abs(a)
        IAE[1][j] = n * abs(b)
        IAE[2][j] = n * abs(c)
        IAE[3][j] = n * abs(d)
        IAE[4][j] = n * abs(e)
        j = j + 1

    fitness = [sum(IAE[0]), sum(IAE[1]), sum(IAE[2]), sum(IAE[3]),
                                        sum(IAE[4])]
    if flag_notgood is True:
        return [10000,10000,10000,10000,10000]

    if flag is True:
        x = np.array([fitness[0] + (penalty[0] + penalty[1]) /
                                            fitness[0],
                      fitness[1] + (penalty[6]) / fitness[1],
                      fitness[2] + (penalty[4]) / fitness[2],
                      fitness[3] + (penalty[5]) / fitness[3],
                      fitness[4] + (penalty[2] + penalty[3]) /
                                                  fitness[4]]
                                                  )
    return x if flag is True else fitness

"""The Primitive set is determined in the following code lines. The
                                    mathematical operations,
                                   constants and variables are
                                   defined. Such step is very
                                   important for the algorithm
                                   because with such definition the
                                   size of search space is
                                   automatically defined."""
pset = gp.PrimitiveSet("MAIN", 5)
pset.addPrimitive(operator.add, 2, name="Add")
pset.addPrimitive(operator.sub, 2, name="Sub")
pset.addPrimitive(operator.mul, 2, name="Mul")
pset.addPrimitive(TriAdd, 3)
pset.addPrimitive(Abs, 1)
pset.addPrimitive(Tanh, 1)
pset.addPrimitive(Sqrt, 1)
for i in range(nEph):
    pset.addEphemeralConstant("rand{}".format(i), lambda: round(
                                        random.uniform(-500, 500), 6)
                                        )

pset.renameArguments(ARG0='errR')
pset.renameArguments(ARG1='errTheta')
pset.renameArguments(ARG2='errVr')
pset.renameArguments(ARG3='errVt')
pset.renameArguments(ARG4='errm')

creator.create("Fitness", base.Fitness, weights=(-0.2, -0.4, -4.0,
                                    -0.1, -2.0))
creator.create("Individual", list, fitness=creator.Fitness, height=
                                    1)
creator.create("SubIndividual", gp.PrimitiveTree, fitness=creator.
                                    Fitness)
toolbox = base.Toolbox()
toolbox.register("expr", gp.genFull, pset=pset, type_=pset.ret,
                                    min_=1, max_=2)
```

```
toolbox.register("leg", tools.initIterate, creator.SubIndividual,
                                    toolbox.expr)
toolbox.register("legs", tools.initRepeat, list, toolbox.leg, n=
                                    Ncontrols)
toolbox.register("individual", tools.initIterate, creator.
                                    Individual, toolbox.legs)
toolbox.register("population", tools.initRepeat, list, toolbox.
                                    individual)
toolbox.register("compile", gp.compile, pset=pset)
toolbox.register("evaluate", evaluate)
toolbox.register("select", tools.selNSGA2)
toolbox.register("mate", xmate)
toolbox.register("expr_mut", gp.genFull, min_=1, max_=3)
toolbox.register("mutate", xmut, expr=toolbox.expr_mut)
toolbox.decorate("mate", gp.staticLimit(key=operator.attrgetter("
                                    height"), max_value=limit_height)
                                    )
toolbox.decorate("mate", gp.staticLimit(key=len , max_value=
                                    limit_size))

if __name__ == "__main__":
    obj = Rocket()
    pop, log, hof = main()
```

D.4 GODDARD _ ONLINE.py

```
/* This Source Code Form is subject to the terms of the Mozilla
                                    Public
 * License, v. 2.0. If a copy of the MPL was not distributed with
                                     this
 * file, You can obtain one at http://mozilla.org/MPL/2.0/. */
/*
----- Copyright (C) 2019 University of Strathclyde and Authors ----
------------------- Author: Francesco Marchetti -------------------
-------------- e-mail: francesco.marchetti@strath.ac.uk ------------
--------------------- Author: Nicola Menga -------------------------
-------------- e-mail: nicola.menga@strath.ac.uk -------------------
------------------------nicola.menga@studenti.polito.it-------------
*/

from scipy.integrate import solve_ivp
import numpy as np
import operator
import pygraphviz as pgv
import random
from deap import gp
import matplotlib.pyplot as plt
import sys
import timeit
from deap import algorithms
from deap import base
from deap import creator
from deap import tools
import multiprocessing
from scipy.interpolate import PchipInterpolator
from time import time
from interruptingcow import timeout
import pickle
```

```
def TriAdd(x, y, z):
    return x + y + z

def Sqrt(a):
    if a > 0:
        return np.sqrt(a)
    else:
        return np.abs(a)


def Log(a):
    if a > 0:
        return np.log(a)
    else:
        return np.abs(a)


def Tanh(a):
    return np.tanh(a)


def Mul(a, b):
    return a * b


def Sub(a, b):
    return a - b


def Add(a, b):
    return a + b


def Abs(a):
    return np.abs(a)

def xmate(ind1, ind2):
    i1 = random.randrange(len(ind1))
    i2 = random.randrange(len(ind2))
    ind1[i1], ind2[i2] = gp.cxOnePoint(ind1[i1], ind2[i2])
    return ind1, ind2
def xmut(ind, expr):

    i1 = random.randrange(len(ind))
    i2 = random.randrange(len(ind[i1]))
    choice = random.random()
    indx = gp.mutUniform(ind[i1], expr, pset=pset)
    ind[i1] = indx[0]
    return ind,

class Rocket:

    def __init__(self):
        self.GMe = 3.986004418 * 10 ** 14                #Earth
                                                    gravitational constant [m
                                                    ^3/s^2]
        self.Re = 6371.0 * 1000                          #Earth
                                                    Radius [m]
        self.Vr = np.sqrt(self.GMe / self.Re)            #[m/s]
        self.H0 = 10.0                                   #[m]
        self.V0 = 0.0
        self.M0 = 100000.0                               #[kg]
```

```
        self.Mp = self.M0 * 0.99
        self.Cd = 0.6
        self.A = 4.0                                        #[m2]
        self.Isp = 300.0                                    #[s]
        self.g0 = 9.80665                                   #[m/s2]
        self.Tmax = self.M0 * self.g0 * 1.5
        self.MaxQ = 14000.0                                 #[Pa]
        self.MaxG = 8.0
        self.Htarget = 400.0 * 1000                         #[m]
        self.Rtarget = self.Re + self.Htarget               #[m/s]
        self.Vtarget = np.sqrt(self.GMe / self.Rtarget)     #[m/s]

    @staticmethod
    def air_density(h):
        global flag
        beta = 1 / 8500.0
        rho0 = 1.225
        try:
            return rho0 * np.exp(-beta * h)
        except RuntimeWarning:
            flag = True
            return rho0 * np.exp(-beta * obj.Rtarget)

Nstates = 5
Ncontrols = 2
obj = Rocket()
nEph = 2
mutpb = 0.1
cxpb = 0.7
change_time = 200

size_pop = 30            #Population size
size_gen = 30            #Generation size
Mu = int(size_pop)
Lambda = int(size_pop * 1.4)

limit_height = 17         #Max height
limit_size = 400          #Max size

cd_var=12


tref = np.load("time.npy")
total_time_simulation = tref[-1]
del tref

flag=False
flag_time=False
flag_notgood=False
feasible=True
flag_offdesign= False
flag = False

Rref = np.load("R.npy")
Thetaref = np.load("Theta.npy")
Vrref = np.load("Vr.npy")
Vtref = np.load("Vt.npy")
mref = np.load("m.npy")
tref = np.load("time.npy")
tfin = tref[-1]

Rfun = PchipInterpolator(tref, Rref)
```

```
Thetafun = PchipInterpolator(tref, Thetaref)
Vrfun = PchipInterpolator(tref, Vrref)
Vtfun = PchipInterpolator(tref, Vtref)
mfun = PchipInterpolator(tref, mref)

def main():
"""The main function is defined as a container in order to execute
                                    the evolutionary process. Here
                                    the type of algorithm to be used
                                    is chosen like the probability of
                                     crossover and mutation. Moreover
                                     the post processing of the
                                    results as graph of position mass
                                     and the forms of the trees are
                                    inserted in this main function.
                                    """
    global flag_offdesign
    global tfin, flag
    global size_gen, size_pop, Mu, Lambda, mutpb, cxpb
    global Rfun, Thetafun, Vrfun, Vtfun, mfun

    pool = multiprocessing.Pool()
    toolx.register("map", pool.map)
    pop2 = toolx.population()
    pop3 = toolx.popx(random_individual)
    final_pop= pop2 + pop3

    individualss = [ind for ind in final_pop]
    fitnesses = toolx.map(toolx.evaluate, individualss)
    for ind, fit in zip(individualss, fitnesses):
        ind.fitness.values = fit

    hofx = tools.ParetoFront()
    Mux = int(len(final_pop))
    Lambdax = int(len(final_pop)*1.5)

    print("INITIAL POP SIZE: %d" % size_pop)

    print("GEN SIZE: %d" % size_gen)

    print("\n")

    random.seed()

    if flag_offdesign is True:
        popx, logx = algorithms.eaMuPlusLambda(individualss, toolx,
                                               Mux, Lambdax, mutpb,
                                              cxpb, size_gen,
                                              halloffame=hofx, verbose=
                                              True)

    reson = open("HallOfFame_2Cont_ON", "w")
    for i in range(len(hofx)):
        reson.write("{}: 1".format(i) + str(hofx[i][0]) + "\n" + "{
                                              }: 2".format(i) + str(
                                              hofx[i][1]) + "\n")
    reson.close()

    expr1 = hofx[0][0]
    expr2 = hofx[0][1]

    nodes1, edges1, labels1 = gp.graph(expr1)
```

```
    nodes2, edges2, labels2 = gp.graph(expr2)
    g1 = pgv.AGraph()
    g1.add_nodes_from(nodes1)
    g1.add_edges_from(edges1)
    g1.layout(prog="dot")
    for i in nodes1:
        n = g1.get_node(i)
        n.attr["label"] = labels1[i]
    g1.draw("tree1.png")

    g2 = pgv.AGraph()
    g2.add_nodes_from(nodes2)
    g2.add_edges_from(edges2)
    g2.layout(prog="dot")
    for i in nodes2:
        n = g2.get_node(i)
        n.attr["label"] = labels2[i]
    g2.draw("tree2.png")

    plt.show()
    plt.close()

    image1 = plt.imread('tree1.png')
    fig1, ax1 = plt.subplots()
    im1 = ax1.imshow(image1)
    ax1.axis('off')
    image2 = plt.imread('tree2.png')
    fig2, ax2 = plt.subplots()
    im2 = ax2.imshow(image2)
    ax2.axis('off')
    plt.show()
    plt.close()


    sys.stdout.write("TOTAL RUNNING TIME: {} \n".format(tformat))

    pool.close()
    pool.join()
    return (popx, logx, hofx) if flag_offdesign is True else (pop,
                                        log, hof)

def evaluate(individual):
"""The evaluate function is the fundamental function in which
                                    individuals are evaluated in
                                    order to define how an individual
                                     is able to solve the problem. In
                                     this problem, the fitness
                                    function is defined to reduce the
                                     error between the current states
                                     and the desired states through
                                    the adoption of the Integral of
                                    Absolute Error method."""
    global flag, flag_offdesign,flag_notgood
    global Rfun, Thetafun, Vrfun, Vtfun, mfun, Trfun
    global tfin, t_eval2, penalty, cd_var

    penalty = np.zeros((7))
    flag = False
    flag_notgood=False

    fTr = toolx.compile(expr=individual[0])
    fTt = toolx.compile(expr=individual[1])
```

```
def sys(t, x):
    global penalty, flag,flag_notgood, cd_var
    R = x[0]
    theta = x[1]
    Vr = x[2]
    Vt = x[3]
    m = x[4]

    r = Rfun(t)
    th = Thetafun(t)
    vr = Vrfun(t)
    vt = Vtfun(t)
    mf = mfun(t)

    if R < obj.Re:
        penalty[0] = penalty[0] + abs(R - obj.Re)/obj.Htarget
        R = obj.Re
        flag = True
        if t<100:
            penalty[0] = penalty[0] + abs(R - obj.Re) / obj.
                                                  Htarget
    elif R > obj.Rtarget:
        penalty[1] = penalty[1] + abs(R - obj.Rtarget)/obj.
                                                 Htarget
        R = obj.Rtarget
        flag = True
    if m < obj.M0 - obj.Mp:
        penalty[2] = penalty[2] + abs(m - (obj.M0 - obj.Mp))/
                                                 obj.M0
        m = obj.M0 - obj.Mp
        flag = True
    elif m > obj.M0:
        penalty[3] = penalty[3] + abs(m - obj.M0)/obj.M0
        m = obj.M0
        flag = True


    if Vr > 1000:
        penalty[4] = penalty[4] + abs(Vr - 1000)/765.
                                                 0826978731632
        Vr = 1000
        flag = True
    elif Vr < -30 :
        penalty[4] = penalty[4] + abs(Vr - 30)/ 765.
                                                 0826978731632
        Vr = -30
        flag = True
    elif Vr < 0 and t < 100:
        penalty[4] = penalty[4] + abs(Vr)/ 765.0826978731632
        flag = True
        Vr = 0


    if  Vt > 10000:
        penalty[5] = penalty[5] + abs(Vt - 10000)/7684.
                                                 936222117861
        flag = True
        Vt = 10000
    elif Vt < 0 :
        penalty[5] = penalty[5] + abs(Vt - 0)/ 7684.
                                                 936222117861
```

```
            Vt = 0
            flag = True

        if theta > 1.0472:
            penalty[6]=penalty[6] + abs(theta - 1.0472)/0.
                                                8666158523063785
            theta=1.0472
            flag=True
        elif theta < 0:
            penalty[6] = penalty[6] + abs(theta - 0)/0.
                                                8666158523063785
            theta=0
            flag=True


        er = r - R
        et = th - theta
        evr = vr - Vr
        evt = vt - Vt
        em = mf - m

        rho = obj.air_density(R - obj.Re)
        Dr = 0.5 * rho * Vr * np.sqrt(Vr ** 2 + Vt ** 2) * cd_var *
                                              obj.A
        Dt = 0.5 * rho * Vt * np.sqrt(Vr ** 2 + Vt ** 2) * cd_var *
                                              obj.A
        g = obj.g0 * (obj.Re / R) ** 2
        g0 = obj.g0
        Isp = obj.Isp

        Tr = fTr(er, et,evr, evt,  em)
        Tt = fTt(er, et,evr, evt,  em)

        if (fTr(er, et, evr, evt, em)>0 and fTr(er, et, evr, evt,
                                          em)<obj.Tmax) and (fTt(er
                                          , et, evr, evt, em)>0 and
                                           fTt(er, et, evr, evt, em
                                          )<obj.Tmax):
            dxdt = np. array((Vr,
                              Vt / R,
                               Tr / m - Dr / m - g + Vt ** 2 / R,
                               Tt / m - Dt / m - (Vr * Vt) / R,
                               -np.sqrt(Tt ** 2 + Tr ** 2) / (g0 *
                                                              Isp)
                                                              ))
        else:
            dxdt = np. array((Vr,
                              Vt / R,
                               0 / m - Dr / m - g + Vt ** 2 / R,
                               0 / m - Dt / m - (Vr * Vt) / R,
                               -np.sqrt(0 ** 2 + 0 ** 2) / (g0 * Isp)
                                                              ))

            flag_notgood=True
        return dxdt

    tin = 0.0

    if flag_offdesign is True:
        x_ini = xnew_ini
        tin = change_time
        teval = t_eval2
```

```
try:
    with timeout(1.2, exception=RuntimeError):
        sol = solve_ivp(sys, [tin, tfin], x_ini, t_eval=teval)
except RuntimeError:
    return [1e20,1e20,1e20,1e20,1e20]

y1 = sol.y[0, :]
y2 = sol.y[1, :]
y3 = sol.y[2, :]
y4 = sol.y[3, :]
y5 = sol.y[4, :]
tt = sol.t

r = Rfun(tt)
theta = Thetafun(tt)
vr = Vrfun(tt)
vt = Vtfun(tt)
m = mfun(tt)

err1 = (r - y1) / obj.Htarget
err2 = (theta - y2) / 0.8666158523063785
err3 = (vr - y3)/765.0826978731632
err4 = (vt - y4)/7684.936222117861
err5 = (m - y5) / obj.M0

i = 0
pp = 1
step = np.zeros(len(y1), dtype='float')
step[0] = tt[1] - tt[0]
while i < len(tt) - 1:
    step[pp] = tt[i + 1] - tt[i]
    i = i + 1
    pp = pp + 1

IAE = np.zeros((5, len(err1)))
j = 0
for a, b, c, d, e, n in zip(err1, err2, err3, err4, err5, step)
                                        :
    IAE[0][j] = n * abs(a)
    IAE[1][j] = n * abs(b)
    IAE[2][j] = n * abs(c)
    IAE[3][j] = n * abs(d)
    IAE[4][j] = n * abs(e)
    j = j + 1

fitness = [sum(IAE[0]), sum(IAE[1]), sum(IAE[2]), sum(IAE[3]),
                                      sum(IAE[4])]
if flag_notgood is True:
    return [1e20,1e20,1e20,1e20,1e20]

if flag is True:
    x = np.array([fitness[0] + (penalty[0] + penalty[1])/
                                          fitness[0],
          fitness[1] + (penalty[6])/fitness[1],
          fitness[2] + (penalty[4])/fitness[2],
          fitness[3]  + (penalty[5])/fitness[3],
          fitness[4] + (penalty[2]+ penalty[3])/fitness[4]])

return x if flag is True else fitness
```

```
"""The Primitive set is determined in the following code lines. The
                                    mathematical operations,
                                   constants and variables are
                                   defined. Such step is very
                                   important for the algorithm
                                   because with such definition the
                                   size of search space is
                                   automatically defined."""
pset = gp.PrimitiveSet("MAIN", 5)
pset.addPrimitive(operator.add, 2, name="Add")
pset.addPrimitive(operator.sub, 2, name="Sub")
pset.addPrimitive(operator.mul, 2, name="Mul")
pset.addPrimitive(TriAdd, 3)
pset.addPrimitive(Abs, 1)
pset.addPrimitive(Tanh, 1)
pset.addPrimitive(Sqrt, 1)
for i in range(nEph):
    pset.addEphemeralConstant("rand{}".format(i), lambda: round(
                                        random.uniform(-500, 500), 6)
                                        )

pset.renameArguments(ARG0='errR')
pset.renameArguments(ARG1='errTheta')
pset.renameArguments(ARG2='errVr')
pset.renameArguments(ARG3='errVt')
pset.renameArguments(ARG4='errm')

creator.create("Fitness", base.Fitness, weights=(-0.5, -0.2, -0.5,
                                    -0.1, -1.0))
creator.create("Individual", list, fitness=creator.Fitness, height=
                                    1)
creator.create("SubIndividual", gp.PrimitiveTree, fitness=creator.
                                    Fitness)

obj = Rocket()

x_ini = [obj.Re, 0.0, 0.0, 0.0, obj.M0]

def sys2GP(t, x):
    R = x[0]
    theta = x[1]
    Vr = x[2]
    Vt = x[3]
    m = x[4]

    r = Rfun(t)
    th = Thetafun(t)
    vr = Vrfun(t)
    vt = Vtfun(t)
    mf = mfun(t)


    errR = r - R
    errTheta = th - theta
    errVr = vr - Vr
    errVt = vt - Vt
    errm = mf - m

    Tr = Sqrt(TriAdd(Abs(Sub(Add(TriAdd(Sub(Sub(Sqrt(errR), Add(
                                        errVr, -336.252482)), Sub(
                                        errVt, errVt)), errTheta, Abs
                                        (Abs(errVr))),TriAdd(Add(Sub(
```

```
                                        Mul(-70.85579, 25.369957),
                                        TriAdd(errR, errR, -192.45208
                                        )), Sqrt(Mul(errVr, errm))),
                                        TriAdd(Sqrt(Abs(TriAdd(Sqrt(
                                        Sqrt(Sqrt(TriAdd(errTheta,
                                        errVt, errVr)))), Add(Tanh(
                                        errm), Add(417.541074, errR))
                                        ,Mul(Add(272.761683, errVt),
                                        Abs(errTheta))))), Tanh(Tanh(
                                        189.147772)),Add(Mul(281.
                                        174641, errTheta), Add(-463.
                                        187985, -44.566462))), errVt)
                                        ),Add(Abs(Sub(Sub(-294.912831
                                        , errm), Mul(errm, errVr))),
                                        Add(Tanh(Sqrt(Add(356.090788,
                                         Sub(errTheta, errVt)))),
                                        Sqrt(Mul(errVt, -163.151633))
                                        ))))), Mul(Add(Sub(errm, Abs(
                                        Sqrt(Add(450.227029, errVr)))
                                        ), Mul(Sqrt(Sqrt(Tanh(Abs(Abs
                                        (Sub(Add(errTheta, Tanh(Sub(
                                        errm, errVr))), Add(Abs(Mul(
                                        errTheta, errTheta)), Abs(Mul
                                        (Tanh(errVt), Mul(TriAdd(Mul(
                                        errVr, 165.979145), Sqrt(errR
                                        ), Abs(Sub(Sub(errm, -383.
                                        76704), Sqrt(errm)))),Tanh(
                                        Add(errR, Mul(Sub(Tanh(Add(
                                        errTheta, errm)), Abs(errm)),
                                         Abs(Sqrt(errVt))))))))))))
                                        ), Tanh(TriAdd(Mul(Mul(errm,
                                        errVr), Abs(31.587732)), Sqrt
                                        (Abs(errR)),Sqrt(TriAdd(errVr
                                        , errR, Sqrt(Mul(Tanh(115.
                                        522294), TriAdd(errTheta, 19.
                                        959912, 78.486749)))))))),
                                        Sqrt(Add(TriAdd(Sub(Tanh(Abs(
                                        Sqrt(Add(errTheta, Abs(Sqrt(
                                        errR))))))), Add(errVr, -186.
                                        266856)), -393.259084, errVt)
                                        ,Sub(errTheta, 304.859982))))
                                        , Mul(errm, 114.654252)))
    Tt = TriAdd(Sqrt(Mul(errm, 56.715711)), TriAdd(TriAdd(Sub(Tanh(
                                        TriAdd(errTheta, -435.431012,
                                         -288.399033)), Sqrt(Sub(-269
                                        .090055, -216.348377))), Sqrt
                                        (errm),Add(Mul(Abs(TriAdd(Add
                                        (Tanh(errVr), Tanh(Mul(Mul(
                                        errVr, errVt), TriAdd(errVt,
                                        Add(Abs(Abs(errVt)), Sqrt(Add
                                        (errm, 189.687451))), Mul(
                                        Sqrt(Sub(Mul(Abs(errVt),
                                        errTheta), Tanh(errTheta))),
                                        Sqrt(Sqrt(Abs(TriAdd(Sqrt(Mul
                                        (Tanh(Sub(-72.106792,
                                        errTheta)),Add(TriAdd(errVr,
                                        errm, errVt), Mul(-214.408013
                                        , -424.368337)))), Tanh(-484.
                                        034856),Abs(errVt)))))))))),
                                        Sqrt(errm), Tanh(Abs(TriAdd(
                                        errR, Abs(-407.653047), errVr
                                        ))))),82.141667), TriAdd(Abs(
```

```
                                        errVr), TriAdd(Sqrt(Sub(errR,
                                         -358.30903)), errm,Sub(errR,
                                         Tanh(Sqrt(Mul(181.443512,
                                        errVr))))),Tanh(Tanh(errVt)))
                                        )), Sqrt(Mul(265.132498, errm
                                        )), Sqrt(TriAdd(Abs(errm),
                                        Sqrt(-217.067638), TriAdd(
                                        Sqrt(-95.90338), errm, Mul(
                                        Sub(Tanh(Add(Abs(errm),
                                        TriAdd(errTheta, 151.21694,
                                        errVt))), Sub(Sub(Mul(
                                        errTheta, Add(errR, errR)),
                                        Add(Sub(Add(Add(errVr, errR),
                                        Add(132.515184,errm)), Add(
                                        TriAdd(errVt, Sub(Add(Sqrt(
                                        Sub(-493.510467, errVr)),Sqrt
                                        (188.460917)),Add(Tanh(Mul(
                                        errVr,errR)),Sub(-22.727781,
                                        errVr))),errm),TriAdd(errR,
                                        421.834855,228.627005))),Sqrt
                                        (Tanh(330.127231)))),Tanh(
                                        errm))),Sqrt(Abs(Tanh(Sub(134
                                        .690301, errTheta)))))))),
                                        Tanh(Add(errVt, Mul(errR,
                                        errm))))

    rho = obj.air_density(R - obj.Re)
    Dr = 0.5 * rho * Vr * np.sqrt(Vr ** 2 + Vt ** 2) * obj.Cd * obj
                                        .A
    Dt = 0.5 * rho * Vt * np.sqrt(Vr ** 2 + Vt ** 2) * obj.Cd * obj
                                        .A
    g = obj.g0 * (obj.Re / R) ** 2
    g0 = obj.g0
    Isp = obj.Isp

    dxdt = np.array((Vr,
                     Vt / R,
                     Tr / m - Dr / m - g + Vt ** 2 / R,
                     Tt / m - Dt / m - (Vr * Vt) / R,
                     -np.sqrt(Tt ** 2 + Tr ** 2) / g0 / Isp))

    return dxdt


tevals = np.linspace(0.0, change_time, int(change_time*3))
tevalsref = np.linspace(0.0, total_time_simulation, int(
                                    total_time_simulation*3))

solgp = solve_ivp(sys2GP, [0, change_time], x_ini, t_eval=tevals)
rout = solgp.y[0, :]
thetaout = solgp.y[1, :]
vrout = solgp.y[2, :]
vtout = solgp.y[3, :]
mout = solgp.y[4, :]
ttgp = solgp.t

solgppp = solve_ivp(sys2GP, [0, total_time_simulation], x_ini,
                                    t_eval=tevalsref)
ttgppp = solgppp.t

rR = Rfun(tevalsref)
```

```
tR = Thetafun(tevalsref)
vrR = Vrfun(tevalsref)
vtR = Vtfun(tevalsref)
mR = mfun(tevalsref)

i = 0
for i in range(len(ttgp)):
    if ttgp[i] >= change_time:
        index = i
        break

objects = []
with (open("hof_GODDARD.pkl", "rb")) as openfile:
    while True:
        try:
            objects.append(pickle.load(openfile))
        except EOFError:
            break

init_height1=int(objects[0][0].height)
init_height2=(objects[0][1].height)

start = time()
if __name__ == "__main__":
    random_individual=int(size_gen)

    def initPOP1():
        global objects
        return objects[0]

    toolx = base.Toolbox()
    toolx.register("expr", gp.genFull, pset=pset, type_=pset.ret,
                                        min_=init_height1, max_=
                                        init_height1)
    toolx.register("leg", tools.initIterate, creator.SubIndividual,
                                         toolx.expr)
    toolx.register("legs", tools.initRepeat, list, toolx.leg, n=
                                        Ncontrols)
    toolx.register("individual", tools.initIterate, creator.
                                        Individual, toolx.legs)
    toolx.register("popx", tools.initRepeat, list, toolx.individual
                                        )
    toolx.register("population", tools.initIterate, list, initPOP1)
    toolx.register("compile", gp.compile, pset=pset)
    toolx.register("evaluate", evaluate)
    toolx.register("select", tools.selNSGA2)
    toolx.register("mate", xmate)
    toolx.register("expr_mut", gp.genFull, min_=2, max_=5)
    toolx.register("mutate", xmut, expr=toolx.expr_mut)
    toolx.decorate("mate", gp.staticLimit(key=operator.attrgetter("
                                        height"), max_value=
                                        limit_height))
    toolx.decorate("mate", gp.staticLimit(key=len, max_value=
                                        limit_size))

    obj = Rocket()
    xnew_ini = [float(rout[index]), float(thetaout[index]), float(
                                        vrout[index]), float(vtout[
                                        index]), float(mout[index])]
    t_eval2 = np.linspace(change_time, tfin, int(tfin*3))
    flag_seed_populations = True
    flag_offdesign = True
```

```
    flag_prop = False
    size_pop, size_gen, cxpb, mutpb = size_gen, 100, 0.8, 0.1
    popx, logx, hofx = main()
end = time()
t_offdesign = end - start

def sys2GP_c(t, x):
    global cd_var

    R = x[0]
    theta = x[1]
    Vr = x[2]
    Vt = x[3]
    m = x[4]

    r = Rfun(t)
    th = Thetafun(t)
    vr = Vrfun(t)
    vt = Vtfun(t)
    mf = mfun(t)

    errR = r - R
    errTheta = th - theta
    errVr = vr - Vr
    errVt = vt - Vt
    errm = mf - m

    Tr = Sqrt(TriAdd(Abs(Sub(Add(TriAdd(Sub(Sub(Sqrt(errR), Add(
                                        errVr, -336.252482)), Sub(
                                        errVt, errVt)), errTheta, Abs
                                        (Abs(errVr))), TriAdd(Add(Sub
                                        (Mul(-70.85579, 25.369957),
                                        TriAdd(errR, errR, -192.45208
                                        )), Sqrt(Mul(errVr, errm))),
                                        TriAdd(Sqrt(Abs(TriAdd(Sqrt(
                                        Sqrt(Sqrt(TriAdd(errTheta,
                                        errVt, errVr)))), Add(Tanh(
                                        errm), Add(417.541074, errR))
                                        , Mul(Add(272.761683, errVt),
                                         Abs(errTheta))))), Tanh(Tanh
                                        (189.147772)), Add(Mul(281.
                                        174641, errTheta), Add(-463.
                                        187985, -44.566462))), errVt)
                                        ), Add(Abs(Sub(Sub(-294.
                                        912831, errm), Mul(errm,
                                        errVr))), Add(Tanh(Sqrt(Add(
                                        356.090788, Sub(errTheta,
                                        errVt)))), Sqrt(Mul(errVt, -
                                        163.151633)))))), Mul(Add(Sub
                                        (errm, Abs(Sqrt(Add(450.
                                        227029, errVr)))), Mul(Sqrt(
                                        Sqrt(Tanh(Abs(Abs(Sub(Add(
                                        errTheta, Tanh(Sub(errm,
                                        errVr))), Add(Abs(Mul(
                                        errTheta, errTheta)), Abs(Mul
                                        (Tanh(errVt), Mul(TriAdd(Mul(
                                        errVr, 165.979145), Sqrt(errR
                                        ), Abs(Sub(Sub(errm, -383.
                                        76704), Sqrt(errm)))), Tanh(
                                        Add(errR, Mul(Sub(Tanh(Add(
                                        errTheta, errm)), Abs(errm)),
                                         Abs(Sqrt(errVt))))))))))))))
```

```
                                    ), Tanh(TriAdd(Mul(Mul(errm,
                                    errVr), Abs(31.587732)), Sqrt
                                    (Abs(errR)), Sqrt(TriAdd(
                                    errVr, errR, Sqrt(Mul(Tanh(
                                    115.522294), TriAdd(errTheta,
                                     19.959912, 78.486749)))))))
                                    ), Sqrt(Add(TriAdd(Sub(Tanh(
                                    Abs(Sqrt(Add(errTheta, Abs(
                                    Sqrt(errR)))))), Add(errVr, -
                                    186.266856)), -393.259084,
                                    errVt), Sub(errTheta, 304.
                                    859982)))), Mul(errm, 114.
                                    654252)))
    Tt = TriAdd(Sqrt(Mul(errm, 56.715711)), TriAdd(TriAdd(Sub(Tanh(
                                    TriAdd(errTheta, -435.431012,
                                     -288.399033)), Sqrt(Sub(-269
                                    .090055, -216.348377))), Sqrt
                                    (errm), Add(Mul(Abs(TriAdd(
                                    Add(Tanh(errVr), Tanh(Mul(Mul
                                    (errVr, errVt), TriAdd(errVt,
                                     Add(Abs(Abs(errVt)), Sqrt(
                                    Add(errm, 189.687451))), Mul(
                                    Sqrt(Sub(Mul(Abs(errVt),
                                    errTheta), Tanh(errTheta))),
                                    Sqrt(Sqrt(Abs(TriAdd(Sqrt(Mul
                                    (Tanh(Sub(-72.106792,
                                    errTheta)), Add(TriAdd(errVr,
                                     errm, errVt), Mul(-214.
                                    408013, -424.368337)))), Tanh
                                    (-484.034856), Abs(errVt)))))
                                    ))))), Sqrt(errm), Tanh(Abs(
                                    TriAdd(errR, Abs(-407.653047)
                                    , errVr))))), 82.141667),
                                    TriAdd(Abs(errVr), TriAdd(
                                    Sqrt(Sub(errR, -358.30903)),
                                    errm, Sub(errR, Tanh(Sqrt(Mul
                                    (181.443512, errVr))))), Tanh
                                    (Tanh(errVt))))), Sqrt(Mul(
                                    265.132498, errm)), Sqrt(
                                    TriAdd(Abs(errm), Sqrt(-217.
                                    067638), TriAdd(Sqrt(-95.
                                    90338), errm, Mul(Sub(Tanh(
                                    Add(Abs(errm), TriAdd(
                                    errTheta, 151.21694, errVt)))
                                    , Sub(Sub(Mul(errTheta, Add(
                                    errR, errR)), Add(Sub(Add(Add
                                    (errVr, errR), Add(132.515184
                                    , errm)), Add(TriAdd(errVt,
                                    Sub(Add(Sqrt(Sub(-493.510467,
                                     errVr)), Sqrt(188.460917)),
                                    Add(Tanh(Mul(errVr, errR)),
                                    Sub(-22.727781, errVr))),
                                    errm), TriAdd(errR, 421.
                                    834855, 228.627005))), Sqrt(
                                    Tanh(330.127231)))), Tanh(
                                    errm))), Sqrt(Abs(Tanh(Sub(
                                    134.690301, errTheta)))))))),
                                    Tanh(Add(errVt, Mul(errR,
                                    errm))))

    if m <= obj.M0-obj.Mp:
        Tr = 0.0
```

```
        Tt = 0.0
        m = obj.M0 - obj.Mp

    rho = obj.air_density(R - obj.Re)
    Dr = 0.5 * rho * Vr * np.sqrt(Vr ** 2 + Vt ** 2) * cd_var * obj
                                        .A
    Dt = 0.5 * rho * Vt * np.sqrt(Vr ** 2 + Vt ** 2) * cd_var * obj
                                        .A
    g = obj.g0 * (obj.Re / R) ** 2
    g0 = obj.g0
    Isp = obj.Isp

    dxdt = np.array((Vr,
                     Vt / R,
                     Tr / m - Dr / m - g + Vt ** 2 / R,
                     Tt / m - Dt / m - (Vr * Vt) / R,
                     -np.sqrt(Tt ** 2 + Tr ** 2) / g0 / Isp))

    return dxdt

passint_c = (change_time + t_offdesign - (change_time)) * 4
tevals_c = np.linspace(change_time, change_time + t_offdesign, int(
                                    passint_c))
xnew_ini = [float(rout[index]), float(thetaout[index]), float(vrout
                                    [index]), float(vtout[index]),
                                    float(mout[index])]

solgp_c = solve_ivp(sys2GP_c, [change_time, change_time +
                                    t_offdesign], xnew_ini, t_eval=
                                    tevals_c)

rout_c = solgp_c.y[0, :]
thetaout_c = solgp_c.y[1, :]
vrout_c = solgp_c.y[2, :]
vtout_c = solgp_c.y[3, :]
mout_c = solgp_c.y[4, :]
for i in range(len(mout_c)):
    if mout_c[i] < obj.M0-obj.Mp:
        mout_c[i] = obj.M0-obj.Mp
ttgp_c = solgp_c.t

for i in range(len(ttgp_c)):
    if ttgp_c[i] >= t_offdesign+change_time:
        index_c = i
        break

passint_gp = (tfin - (change_time + t_offdesign)) * 4
tevals_gp = np.linspace(change_time + t_offdesign, tfin, int(
                                    passint_gp))
xnew_ini_gp = [float(rout_c[index_c]), float(thetaout_c[index_c]),
                                    float(vrout_c[index_c]), float(
                                    vtout_c[index_c]), float(mout_c[
                                    index_c])]

def sys2GP_gp(t, x):
    global flag_prop,cd_var
    fTr = toolx.compile(hofx[0][0])
    fTt = toolx.compile(hofx[0][1])
    R = x[0]
    theta = x[1]
    Vr = x[2]
    Vt = x[3]
```

```
    m = x[4]

    r = Rfun(t)
    th = Thetafun(t)
    vr = Vrfun(t)
    vt = Vtfun(t)
    mf = mfun(t)

    er = r - R
    et = th - theta
    evr = vr - Vr
    evt = vt - Vt
    em = mf - m

    Tr = fTr(er, et, evr, evt, em)
    Tt = fTt(er, et, evr, evt, em)


    if m <= obj.M0 - obj.Mp:
        Tr = 0.0
        Tt = 0.0
        m = obj.M0 - obj.Mp

    rho = obj.air_density(R - obj.Re)
    Dr = 0.5 * rho * Vr * np.sqrt(Vr ** 2 + Vt ** 2) * cd_var * obj
                                        .A
    Dt = 0.5 * rho * Vt * np.sqrt(Vr ** 2 + Vt ** 2) * cd_var * obj
                                        .A
    g = obj.g0 * (obj.Re / R) ** 2
    g0 = obj.g0
    Isp = obj.Isp

    dxdt = np.array((Vr,
                     Vt / R,
                     Tr / m - Dr / m - g + Vt ** 2 / R,
                     Tt / m - Dt / m - (Vr * Vt) / R,
                     -np.sqrt(Tt ** 2 + Tr ** 2) / g0 / Isp))

    return dxdt

flag_prop = False
solgp_gp = solve_ivp(sys2GP_gp, [change_time + t_offdesign, tfin],
                                    xnew_ini_gp, t_eval=tevals_gp)

rout_gp = solgp_gp.y[0, :]
thetaout_gp = solgp_gp.y[1, :]
vrout_gp = solgp_gp.y[2, :]
vtout_gp = solgp_gp.y[3, :]
mout_gp = solgp_gp.y[4, :]

for i in range(len(mout_gp)):
    if mout_gp[i] < obj.M0 - obj.Mp:
        mout_gp[i] = obj.M0 - obj.Mp
ttgp_gp = solgp_gp.t

passint_cp = (tfin - change_time + t_offdesign) * 10
tevals_p = np.linspace(change_time + t_offdesign, tfin, int(
                                    passint_cp))
xnew_inip = [float(rout_c[index_c]), float(thetaout_c[index_c]),
                                    float(vrout_c[index_c]), float(
                                    vtout_c[index_c]), float(mout_c[
                                    index_c])]
```

```
solgpp = solve_ivp(sys2GP_c, [change_time + t_offdesign, tfin],
                                    xnew_inip, t_eval=tevals_p)


rp = solgpp.y[0, :]
thetap = solgpp.y[1, :]
vrp = solgpp.y[2, :]
vtp = solgpp.y[3, :]
mp = solgpp.y[4, :]
tp = solgpp.t

cc=np.zeros(len(ttgppp), dtype='float')
for i in range(len(ttgppp)):
    cc[i]=obj.M0 - obj.Mp

plt.figure(3)
plt.ylim(bottom=-20,top=700)
plt.ylabel("altitude [km]")
plt.xlabel("time [s]")
plt.plot(ttgppp, (rR-obj.Re)/1e3, 'r--', label="SET POINT")
plt.plot(ttgp, (rout-obj.Re)/1e3, 'r', label="GENETIC PROGRAMMING (
                                    OFF-LINE DESIGN)")
plt.plot(ttgp_c, (rout_c-obj.Re)/1e3, color='b', label="OFF-DESIGN
                                    CONDITIONS")
plt.plot(tp, (rp-obj.Re)/1e3, color='b')
plt.plot(ttgp_gp, (rout_gp-obj.Re)/1e3,  color='g', label="GENETIC
                                    PROGRAMMING (ON-LINE DESIGN)")
plt.legend(loc='best')
plt.savefig('Altitude_ONLINE.png')
plt.show()

plt.figure(4)
plt.ylim(bottom=-1,top=70)
plt.ylabel("flight angle displacement [deg]")
plt.xlabel("time [s]")
plt.plot(ttgppp, np.rad2deg(tR), 'r--', label="SET POINT")
plt.plot(ttgp, np.rad2deg(thetaout),'r', label="GENETIC PROGRAMMING
                                     (OFF-LINE DESIGN)")
plt.plot(ttgp_c, np.rad2deg(thetaout_c), color='b', label="OFF-
                                    DESIGN CONDITIONS")
plt.plot(tp, np.rad2deg(thetap),  color='b')
plt.plot(ttgp_gp, np.rad2deg(thetaout_gp),  color='g', label="
                                    GENETIC PROGRAMMING (ON-LINE
                                    DESIGN)")
plt.legend(loc='best')
plt.savefig('Flight_angle_displacement_ONLINE.png')
plt.show()

plt.figure(5)
plt.ylim(bottom=-20,top=1100)
plt.ylabel("radial velocity [m/s]")
plt.xlabel("time [s]")
plt.plot(ttgppp, vrR, 'r--', label="SET POINT")
plt.plot(ttgp, vrout,'r', label="GENETIC PROGRAMMING (OFF-LINE
                                    DESIGN)")
plt.plot(ttgp_c, vrout_c,  color='b', label="OFF-DESIGN CONDITIONS"
                                    )
plt.plot(tp, vrp,  color='b')
plt.plot(ttgp_gp, vrout_gp,  color='g', label="GENETIC PROGRAMMING
                                    (ON-LINE DESIGN)")
plt.legend(loc='best')
plt.savefig('Radial_velocity_ONLINE.png')
```

```
plt.show()

plt.figure(6)
plt.ylim(bottom=-100,top=9000)
plt.ylabel("tangential velocity [m/s]")
plt.xlabel("time [s]")
plt.plot(ttgppp, vtR, 'r--', label="SET POINT")
plt.plot(ttgp, vtout,'r', label="GENETIC PROGRAMMING (OFF-LINE
                                DESIGN)")
plt.plot(ttgp_c, vtout_c, color='b', label="OFF-DESIGN CONDITIONS")
plt.plot(tp, vtp, color='b')
plt.plot(ttgp_gp, vtout_gp,  color='g', label="GENETIC PROGRAMMING
                                (ON-LINE DESIGN)")
plt.legend(loc='best')
plt.savefig('Tangential_velocity_ONLINE.png')
plt.show()

plt.figure(7)
plt.ylim(bottom=-10,top=101000)
plt.ylabel("mass [kg]")
plt.xlabel("time [s]")
plt.plot(ttgppp, mR, 'r--', label="SET POINT")
plt.plot(ttgp, mout,'r', label="GENETIC PROGRAMMING (OFF-LINE
                                DESIGN)")
plt.plot(ttgp_c, mout_c,  color='b', label="OFF-DESIGN CONDITIONS")
plt.plot(tp, mp,  color='b')
plt.plot(ttgp_gp, mout_gp,  color='g', label="GENETIC PROGRAMMING (
                                ON-LINE DESIGN)")
plt.plot(ttgppp, cc,  color='k', label="MASS CONSTRAINT")
plt.savefig('Mass_ONLINE.png')
plt.legend(loc='best')
plt.show()

print("\n")
print("Tr new: ", hofx[0][0])
print("Tt new: ", hofx[0][1])

print(hofx[0].fitness)

print(t_offdesign)

with open("Control Laws.txt", "w") as file:
    file.write(str(hofx[0][0]))
    file.write("\n \n")
    file.write(str(hofx[0][1]))

with open("RUNTIME.txt", "w") as file:
    file.write(str(t_offdesign))
```

List of Figures

List of Tables

References

[1] Michael Affenzeller and Winkler Stephan. *Genetic Algorithms and Genetic Programming: Modern Concepts and Pratical Applications*. Chapman and Hall/CRC, 2017.

[2] J.E. Baker. "Reducing bias and inefficiency in the selection algorithm, proceedings of the second international conference on genetic algorithms." In: (1987).

[3] Miller Brad L. and Goldberg David E. "Genetic Algorithms,Tournament Selection, and the Effects of Noise". In: *Complex Systems* (1995).

[4] Wilson Callum et al. *Intelligent Control: A Taxonomy **. Tech. rep. IceLab, University of Strathclyde, 2019.

[5] Ferreira Cândida. "Gene Expression Programming: A New Adaptive Algorithm for Solving Problems". In: *Complex Systems* (2002).

[6] Steve Chien, Sherwood Rob, and Tran Daniel. "Using Autonomy Flight Software to Improve Science Return on Earth Observing One". In: *Journal of Aerospace Computing, Information, and Communication* (2005). DOI: 10.2514/1.12923.

[7] Coello Coelle. "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art". In: *Computer Methods in Applied Mechanics and Engineering* (2002). DOI: 10.1016/s0045-7825(01)00323-1.

[8] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. 2001.

[9] *Defining Intelligent Control. Report to the Task Force on Intelligent Control*. Tech. rep. IEEE Control Systems Society, 1993, pp. 1–31.

[10] Félix-Antoine Fortin et al. "DEAP: Evolutionary Algorithms Made Easy". In: *Journal of Machine Learning Research* (July 2012).

[11] *Genetic Programming: bio-inspired machine learning*. URL: http://geneticprogramming.com/about-gp/tree-based-gp/.

[12] *HPC@POLITO*. URL: http://www.hpc.polito.it.

[13] Miller Julian F. *Introduction to Evolutionary Computation and Genetic Programming*. 2011.

[14] Branislav Kadlic, Ivan Sekaj, and Daniel Pernecký. "Design of Continuous-time Controllers using Cartesian Genetic Programming". In: *IFAC Proceedings Volumes* (2014). DOI: 10.3182/20140824-6-za-1003.00915.

[15] John R. Koza. *Genetic programming II: automatic discovery of reusable programs*. MIT, 1998.

[16] John R. Koza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Springer, 2005.

[17] John R. Koza. *Genetic programming: on the programming of computers by means of natural selection*. Mit press, 1992.

[18] John R. Koza. "Tutorial on advanced genetic programming, at genetic programming". In: (1997).

[19] Adam Lipowski and Dorota Lipowska. "Roulette-wheel selection via stochastic acceptance". In: *Physica A: Statistical Mechanics and its Applications* (2012). DOI: 10.1016/j.physa.2011.12.004.

[20] *Measures of controlled system performance*. URL: http://www.online-courses.vissim.us/Strathclyde/measures_of_controlled_system_pe.htm.

[21] *Pareto front*. URL: https://upload.wikimedia.org/wikipedia/%20commons/thumb/b/b7/Front%20pareto.svg/640px-Front%20pareto.svg.png.

[22] Smith Peter W.H. *Controlling Code Growth in Genetic Programming*. Tech. rep. 2001.

[23] Michael Defoin Platel, Manuel Clergue, and Philippe Collard. "Maximum Homologous Crossover for Linear Genetic Programming". In: *Lecture Notes in Computer Science Genetic Programming* (2003). DOI: 10.1007/3-540-36599-0_18.

[24] Riccardo Poli. *Genetic Programming Practice and Theory*. IEEE Congress on Evolutionary Computation, 2007.

[25] Riccardo Poli et al. *A field guide to genetic programming*. Lulu Press, 2008.

[26] Annalisa Riccardi et al. *Assesment of intelligent control techniques for space applications*. Tech. rep. IceLab, University of Strathclyde, 2019.

[27] Konstantinos Salpasaranis and Vasilios Stylianakis. "A Hybrid Genetic Programming Method in Optimization and Forecasting: A Case Study of the Broadband Penetration in OECD Countries." In: *Advances in Operations Research* (2012). DOI: 10.1155/2012/904797.

[28] Ivan Sekaj, Marian Tarnik, and Rudolf Goga. "Parameter optimisation of Artificial Pancreas adaptive controller using genetic algorithm". In: *2015 IEEE 19th International Conference on Intelligent Engineering Systems (INES)* (2015). DOI: 10.1109/ines.2015.7329706.

[29] Bernd R. Noack. Thomas Duriez Steven L. Brunton. *Machine learning control: taming nonlinear dynamics and turbulence*. Springer, 2016.

[30] James Alfred Walker, Julian Francis Miller, and Rachel Cavill. "A multi-chromosome approach to standard and embedded cartesian genetic programming". In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation - GECCO 06* (2006). DOI: 10.1145/1143997.1144153.

[31] Özgür Yeniay. "Penalty Function Methods for Constrained Optimization with Genetic Algorithms". In: *Mathematical and Computational Applications* (2005). DOI: 10.3390/mca10010045.

[32] Yun Zhang and Mengjie Zhang. "A Multiple-Output Program Tree Structure in Genetic Programming". In: *CiteSeerX* (2004). DOI: 10.1.1.130.8373.

Acknowledgements

I would like to really thank my external advisor Prof. Edmondo Minisci and the PhD student Francesco Marchetti for the precious support, for the patience and for the constant help during my stay in Strathclyde university.

My sincere thanks to all the people who gave me useful tips during the weekly meetings in ICE Lab.

I would like to thank Prof. Giorgio Guglieri for having accepted to be my internal advisor for Politecnico di Torino.

I would like to dedicate this thesis to my sister Eva and my parents who changed my life for the better.