31/01/2024

**Development of an optimal cruise control system**

**Assel Ussenova, Albert Jiménez Blanco**

**ABSTRACT** Cruise control is a feature, present in many modern cars, which allows you to set a desired speed, which will be attained and maintained automatically by the car. To develop a system that simulates this behaviour, two steps are needed. First, one needs, from data of input throttle and output velocity, to create a model for the car’s movement. Second, use this model to create the optimal system itself. This and an extra layer of Multi-Objective optimization is what is done in the current project.

**INDEX TERMS** Cruise control, Optimization, Multi-Objective, Quadratic Optimization

**I. INTRODUCTION**

The main objective of this project is to create an optimal cruise control system, which corresponds to a system capable of automatically reaching and keeping a target velocity. This kind of system is very common in new cars, as it allows the driver, mainly on highways, to only have to take care of the direction of the car and not of its velocity.

To create a system of this kind, two steps must be followed: one needs a model of the movement of the car and, using it, a control system which takes care of reaching the desired velocity. In this regard, in the first step, system identification, the idea is that to be able to output the necessary throttles to get to a certain speed, one must be aware of the relation that exists between input throttles and changes in speed. That is, one needs a model of the movement of the car to be able to later make accurate predictions, which is what the optimal (Model Predictive) Control does.

**II. SYSTEM IDENTIFICATION**

The system identification problem can be treated as a Quadratic Optimization Problem, where the value that wants to be minimized is the sum of the square error between the real and the predicted outputs. In this situation, given certain input (*x(t)*) and output (*y(t)*) data and the order of the model that wants to be used, the idea will be to find the coefficients that minimize the objective function.

The kind of model used here is based on the idea of a **Discrete Time Transfer Function** and its **difference equation**. A difference equation refers here to a linear equation involving the last *N* inputs and *M* outputs, which would model the evolution of the system on time, as can be seen in **Equation 1**. This method has the clear advantage that it takes past information both from the input and the output to get a new prediction.



**Equation 1:** Difference equation

The idea, then, is to determine the parameters that allow for the best modeling of the system. In **Equation 1**, for instance, this would mean to find the best values for the coefficients *ai* and *bj*. And what we mean by the best here, are the coefficients that minimize the sum of squared errors between the real outputs and the predicted values, as can be seen in **Equation 2**.

**Equation 2:** Objective function

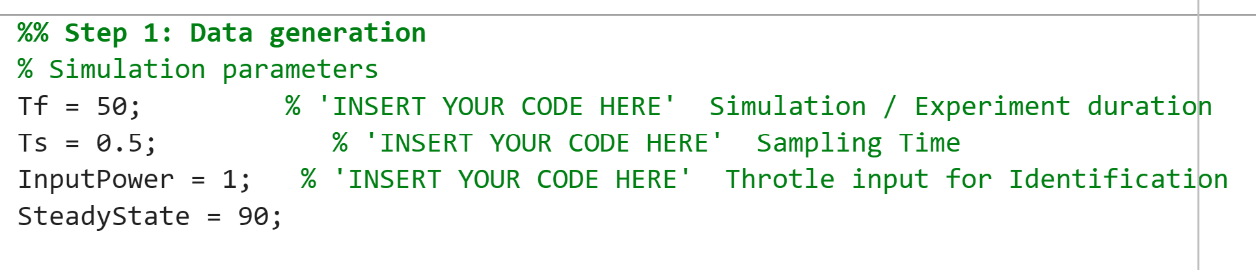
This minimization problem can easily be converted to a quadratic problem with the variables being the parameters that want to be determined. In this regard and considering the column vector *Y* to be the real outputs,  to be the approximated values (with *x* representing the parameters to determine) and *E* being the column vector containing the errors, one gets **Equation 3**, which clearly represents a quadratic optimization problem. It is important to note, though, that the independent term has been dropped because it doesn’t change the optimal parameters and that the expressions for *H* and *c* can easily be computed by developing the product of the two parentheses.



**Equation 3:** System Identification as a Quadratic Problem

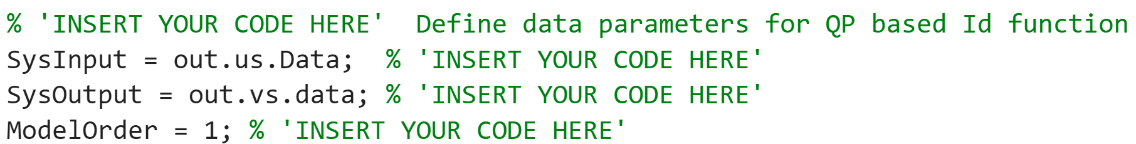
***A. METHODOLOGICAL DECISIONS TAKEN***

In this section, the part of the code that was added to effectively implement and solve this optimization problem will be explained. The first thing that had to be determined were the parameters for the simulation, as can be seen in **Figure 1**. The duration of the simulation was set to 50, the sampling time to 0.5 and the input throttle to 1, as we wanted to have enough samples to get a good model of the system and we wanted to work with a high value for the input, not just small accelerations. Clearly, this is a very important step, as these parameters will have a great influence on the creation of the training data that the model will use.



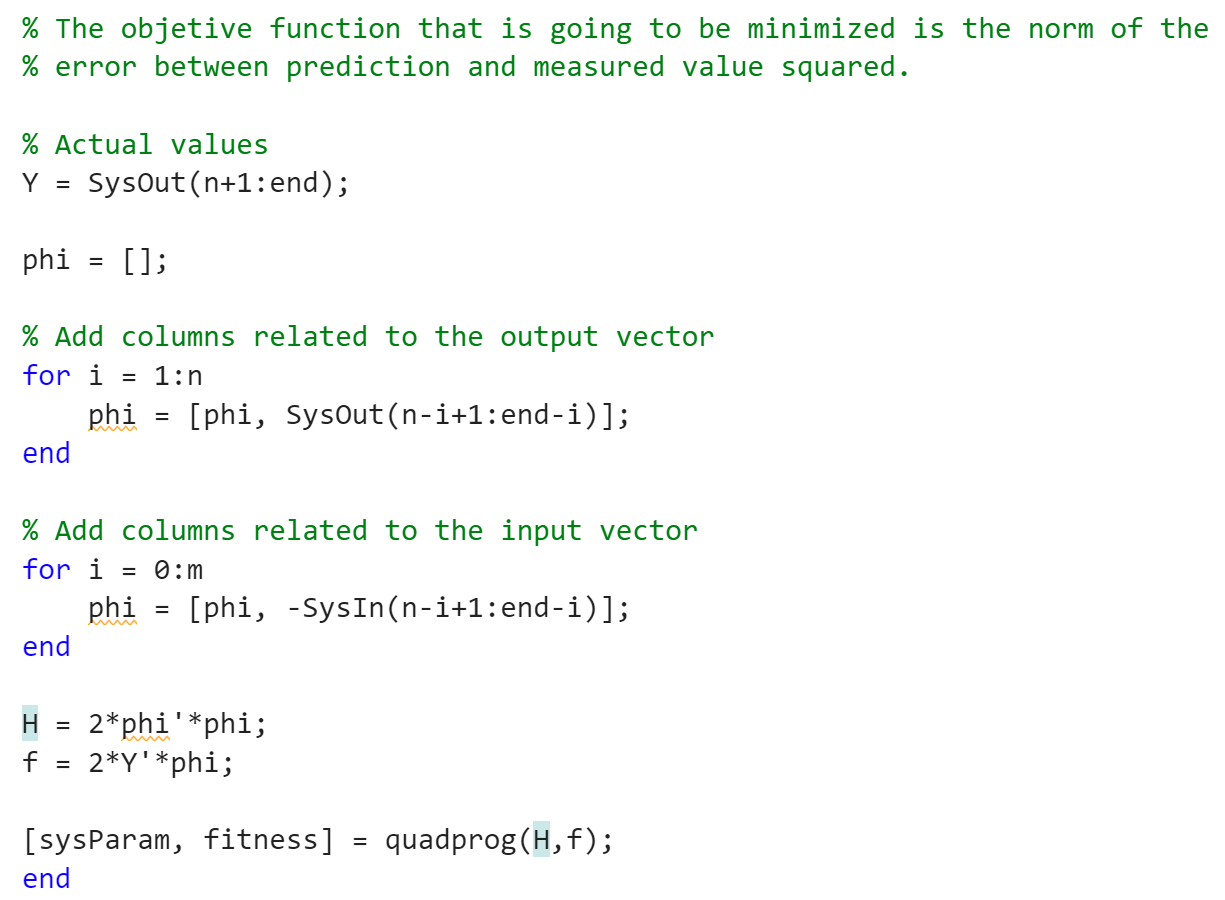
**Figure 1:** Simulation parameters

After this, the data and the order of the model that would be used had to be determined as well. Regarding the data, the input and output created by the simulation were properly identified and the order of the model was set to 1, the easiest possible. As this order was enough to get good results, it was not necessary to change it. This can be seen in **Figure 2**.



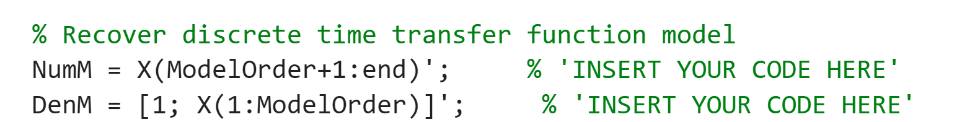
**Figure 2:** Data and model parameters

Then, in the *ID\_QPProblem* the Quadratic Optimization Problem was created and solved. To do so, the vectors and matrices mentioned in the previous section (*H, c, Y...*)were created and the Matlab function *quadprog* was called to solve the corresponding optimization problem, as can be seen in **Figure 3**.



**Figure 3:** Creation and solution of the QP problem

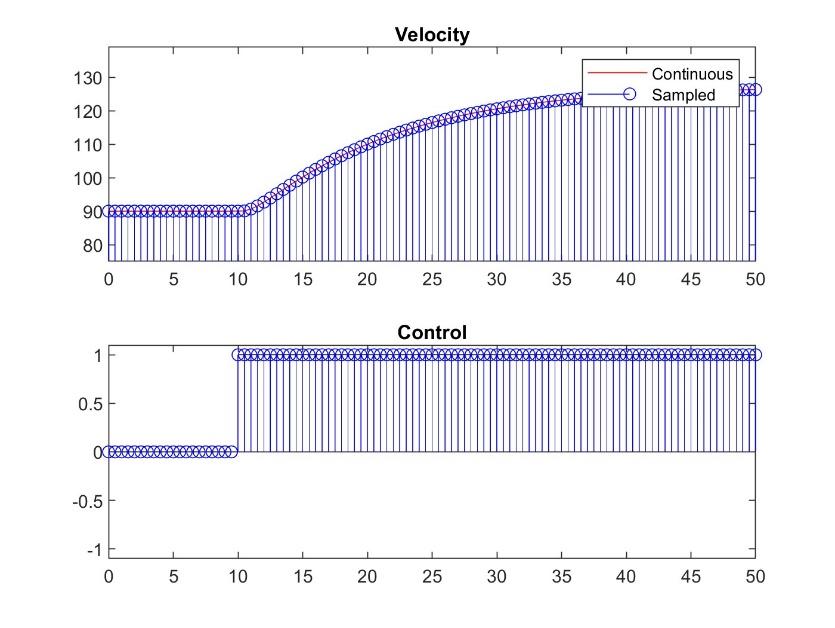
Finally, as can be seen on **Figure 4**, the Discrete Time Transfer Function was recovered from these results.



**Figure 4:** Discrete Time Transfer Function

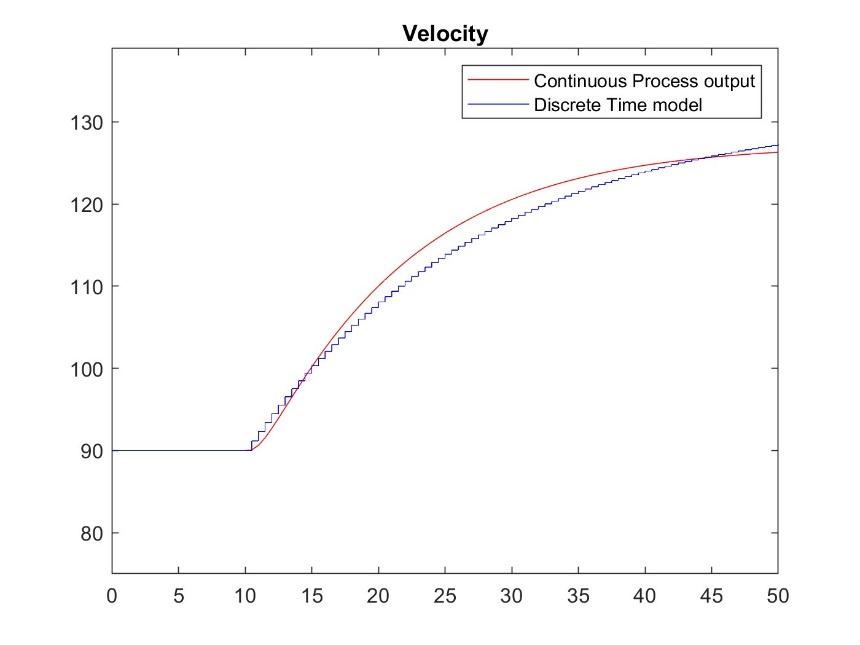
***B. RESULTS***

The inputs and outputs of the simulation performed, which depended on the simulation parameters *Tf, Ts...* mentioned before, can be seen in **Figure 5**. Clearly, there are enough samples to properly see the evolution of the velocity on time, as we expected due to the values chosen for *Tf* and *Ts.*

******

**Figure 5:** Simulation inputs and outputs

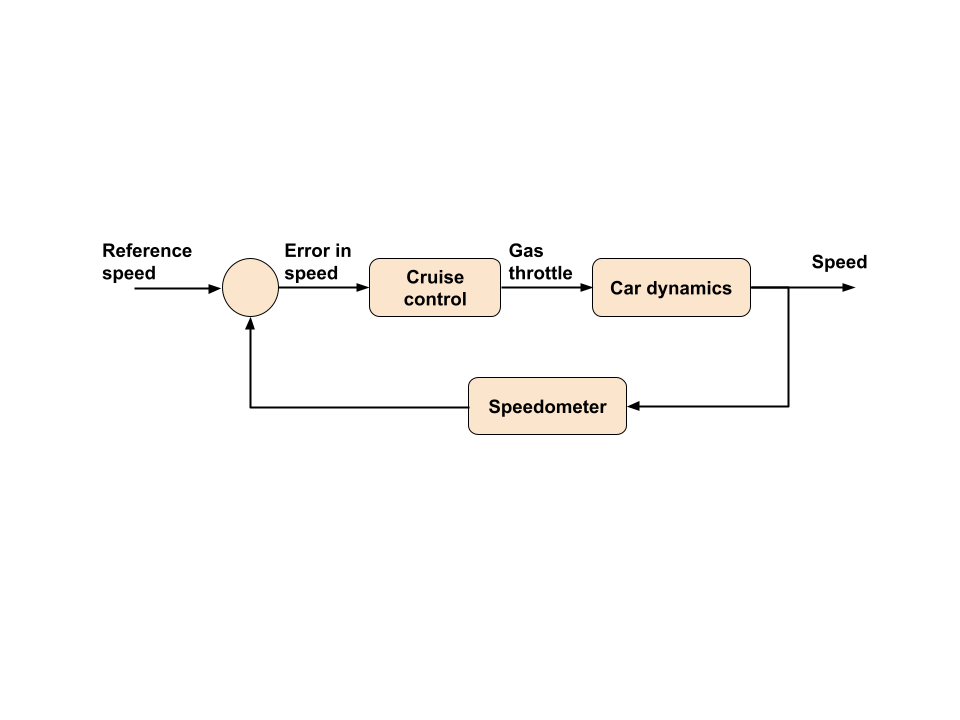
Using these simulated values, the approximation performed by our order 1 model can be observed in **Figure 6**, which compares the simulated velocity and the approximation done by the model. In any case, it seems that the order of the model is enough to properly capture the relation between input and output, representing in this case input throttle and output velocity, respectively. Of course, trying a higher order might yield to a more precise approximation. However, these results seem to indicate that an order 1 model is a good tradeoff between simplicity and performance.

******

**Figure 6:** Comparison of simulated and approximated outputs

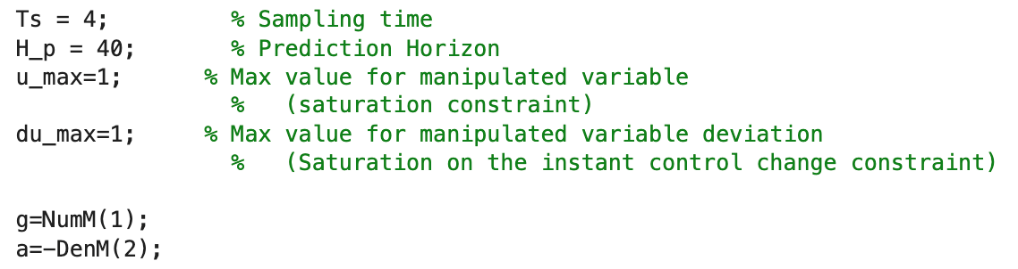
**III. OPTIMAL (MODEL PREDICTIVE) CONTROL**

Optimal (Model Predictive) cruise Control aims to efficiently regulate car speed to achieve a desired reference speed, subject to system constraints. A Feedback Control Loop in cruise control that implements optimal MPC is shown in **Figure 7**. The driver sets a desired speed (reference speed), which serves as the target for the cruise control system. The controller continuously evaluates the difference between the actual speed and the reference speed. To estimate this difference, the Integral Squared Error (ISE) is calculated, which we want to minimize. Based on this error and predictions of future behavior, the controller adjusts the throttle input, influencing the car's acceleration or deceleration. The actual speed is measured and sent back to the controller, closing the loop. This feedback mechanism allows the controller to continually improve its settings, ensuring that the car maintains the desired speed.



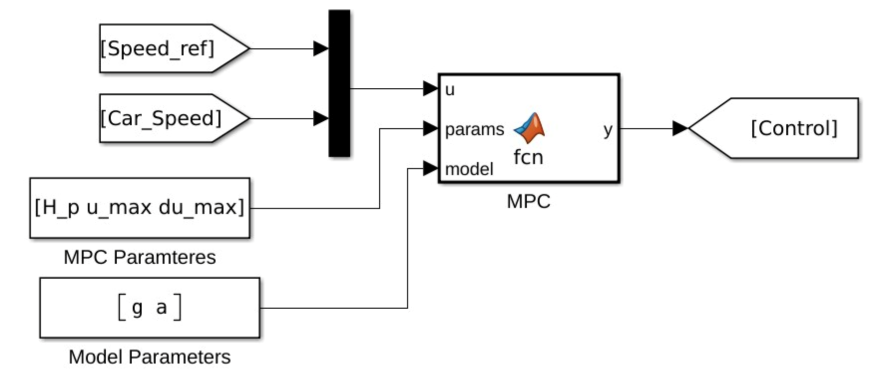
**Figure 7**. Feedback Control Loop

To perform this task, we used the provided MATLAB code that evaluates speed tracking performance and throttle utilization based on Simulink simulation results. Optimal control problem parameters are shown in **Figure 8**. We set the sampling time (Ts) to 4, and the experiment duration (Tf) to 1000 so that we could see the change in speed and throttle use over a longer time. The prediction horizon is set to 40, which allows the control system to more effectively anticipate future states and better manage transient processes such as acceleration or deceleration. The maximum throttle value representing the saturation constraint is set to 1, the maximum throttle deviation imposing saturation constraint on instantaneous control change is set to 1.



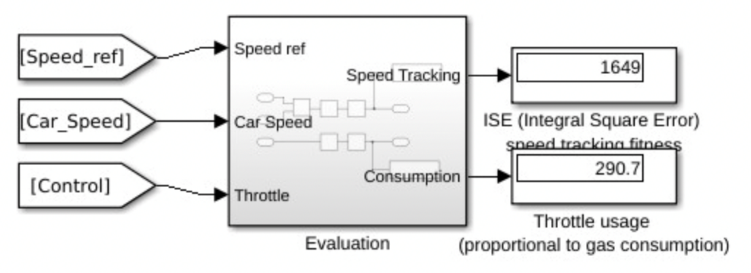
**Figure 8.** Optimal control problem parameters

**Figure 9** shows a Simulink controller model that implements model predictive control. The "MPC" block takes the inputs mentioned earlier (reference speed, car speed, MPC parameters, and Model parameters) to calculate the output (optimal control input (throttle)) based on the given prediction horizon and constraints. It does so by solving a Quadratic Optimization Problem with restrictions, that tries to minimize the ISE in the prediction horizon given certain conditions, as the maximum value for the throttle or the fact that the predicted output velocities are based on the mathematical model found in the previous section (the identified system).



**Figure 9.** Simulink model of the controller

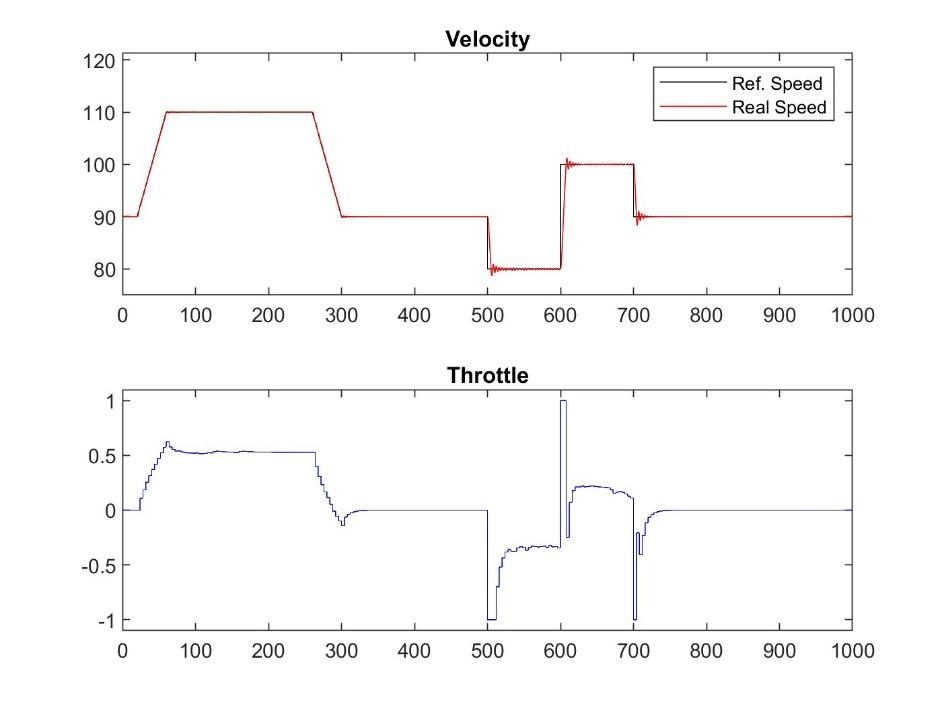
A Simulink Evaluation model designed to evaluate the performance of a Model Predictive Control (MPC) system for cruise control is shown in Figure 10. The "Evaluation" block takes input data such as reference speed, car speed, and control (throttle). The output calculates "Integral Squared Error (ISE)", which is a performance measure that shows how well the system tracks the desired speed, and "Throttle usage", which shows how much throttle is used by the control system to achieve the desired speed. As a result, we see Integral Square Error (ISE) = 1649 and Throttle Usage = 290.7.



**Figure 10.** Simulink Model of MPC system Performance Evaluation

***A. RESULTS***

The results that we get from using the Optimal (Model Predictive) Control and the previously identified System are very good, as can be seen in **Figure 11**. This figure contains two different plots: a first one showing how the real speed converges to the given reference value and a second displaying the throttle values used at each step.



**Figure 11:** Results of the optimal MPC

What can be seen in the first of the plots is that the reference value is properly followed, and that this is done in a fast way, even when abrupt changes in the reference speed happen, such as in timesteps 500, 600 and 700. In the second, however, what we can see is that the input throttle is not always smooth, which can be quite uncomfortable. This already points to the main reason why multi-objective optimization can be very useful: usually one has more than one objective to optimize.

**IV. MULTI-OBJECTIVE OPTIMIZATION**

The problem that we have been working with so far is, to a certain extent, simplistic. This is because we considered that there was only one objective to optimize. However, it is not difficult to think of many situations that require more than one objective to be taken into account. The solution to this kind of problem, as it has different requirements, requires as well different techniques to be solved. This is where multi-objective optimization techniques come into play.

The main idea behind all these methods is to consider more than one objective when searching for the optimal value. There are, broadly speaking, two main approaches to this problem: **Preference-based** and **Ideal based** techniques. The first are based on using some higher-level information at the beginning (the preferences) so that single optimization techniques can be used and only one optimal solution is found. The second, which will be used in this project, are based on using techniques to find many good solutions and then using higher-level information to choose one amongst them.

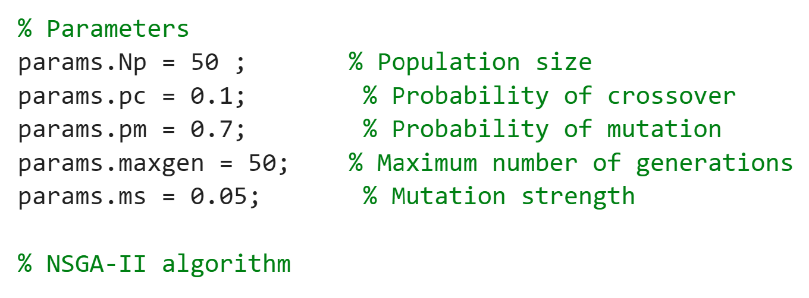
The **Ideal based** techniques, as just mentioned, try to find many good solutions, which basically means to find **non-dominated solutions**. That is, solutions that cannot be bested, in the sense that there isn't any solution that improves at least one of the values of the objective functions whilst being at least as good in the rest. Another way to describe this is to say that these techniques try to get as close as possible to the **Pareto Front** of the problem, corresponding to the non-dominated solutions, and then choose, using higher-level information, one solution among all those. The main way to do so is to use genetic algorithms that start with initial random guesses and get closer to the **Pareto Front** at each generation. In this project, the specific algorithm that will be used is NSGA-II.

The two functions that want to be optimized here are the IAE, the **Integral Absolute Error**, and the TV, the **Total Variation**. The first one measures the total amount of error between the desired and actual output over time and the second indicates the rate of change of the control signal. In both cases, the objective is to minimize them, as a high IAE indicates that the desired output is not properly followed, and a high TV indicates excessive and too abrupt changes in the control signal.

***A. METHODOLOGICAL DECISIONS TAKEN***

It is important to note that the controller used in this section will be a PI (Proportional Integral controller), consisting of two parameters *Ti* and *Kp,* indicating the integral and proportional gain, respectively. These parameters are related, respectively, to the accumulated error and to the error respect the desired value. For simplicity, the value of *Ti* is fixed to 10 and the only parameter to optimize is *Kp*.

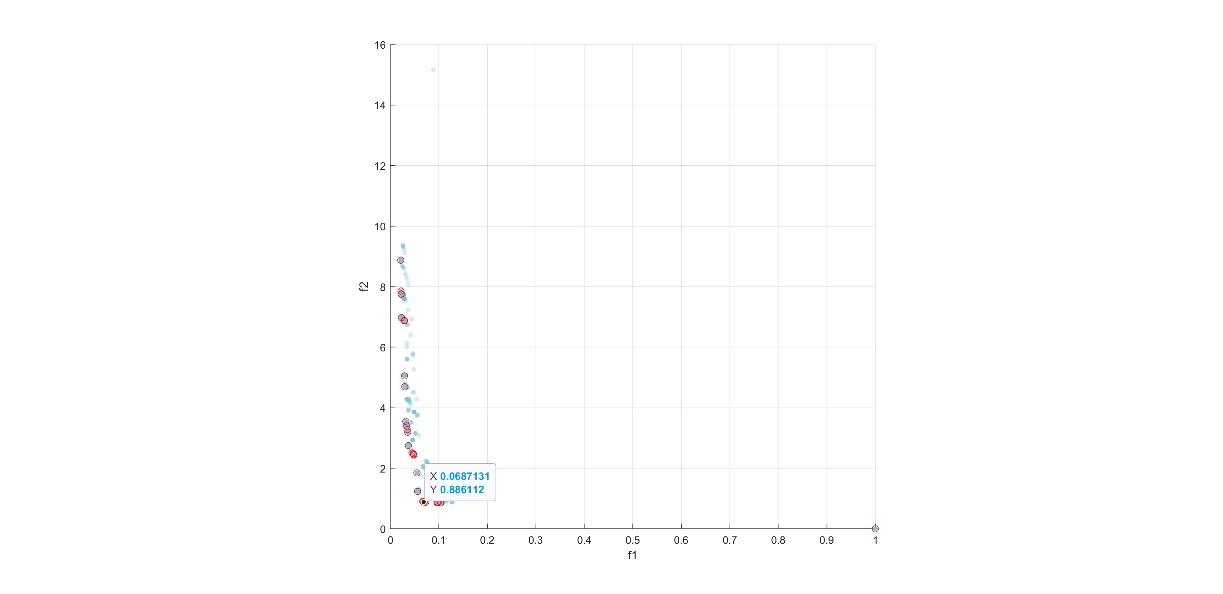
The selection of the value for certain parameters related to how the evolution of the solutions happens upon time, that is, how many generations it will perform, the size of the population... can be seen in **Figure 12**. Basically, the values changed were the population size and the maximum number of generations, both set to 50, as we wanted to be able to depict the Pareto Front more precisely.



**Figure 12:** Selection of evolution parameters

***B. RESULTS***

The results of applying NSGA-II to the optimization functions IAE and TV, can be seen in **Figure 13**, where *f1* corresponds to the **Integral Absolute Error** and *f2* to the **Total Variation**. The plot that we get shows how the solutions have evolved over the different generations. In the figure, the finally found non-dominated solutions are surrounded by a red circle. These points would correspond to our approximate Pareto Front.



**Figure 13:** Results of the NSGA-II algorithm

Clearly, one can see that those solutions having too big of a value either for *f1* or *f2* are not acceptable, as they completely forget about one of the objective functions, contradicting the idea of using multi-objective optimization. In this situation, it seems reasonable that the solution presenting the best **tradeoff** between *f1* and *f2* should be chosen. It seems like any option with a value of *f1* around 0.1 and a value of *f2* around 1 would be good options to consider.

The value for *Kp* finally chosen is the one corresponding to the point with *f1* = 0.0687 and *f2* = 0.8861, which leads to the parameter ***Kp*  = 0.7901**.

**V. CONCLUSION**

In conclusion, in this project we have successfully implemented an optimal cruise control system using techniques from the quadratic and the multi-objective optimization domains. The first part of the project, which only dealt with the creation of an optimal control system that tried to reduce the error between the desired and the actual speed, consisted of two main steps: the identification of the system and creation of the control itself.

It seems that the simplest of models, an order 1 model suffices to properly model the relation between input throttle and output velocity. This can be seen not only in the approximation of the velocity but also in the results of the (Model Predictive) Control, which showed a fast and accurate response even after abrupt changes in the reference speed.

However, this approach seemed to be quite limited, as only one objective function was considered. To get a glimpse on a somewhat more complex configuration of the problem, a final part regarding multi-objective optimization was developed. This allowed us to consider two optimization metrics (IAE and TV) and required some final decision to choose among one of the found solutions. Here a solution was chosen that presented a good tradeoff between the two metrics.

Clearly, we have been working with a simplified version of the problem and developing a functional cruise control system would require more work.