SUPERVISION OF MULTIVARIABLE PREDICTIVE CONTROLLERS

Rachid Aref Ghraizi, Cesar, de Prada Moraga and Ernesto Martínez
Dept of Systems Engineering and Robotisation
Faculty of Scient, University of Valladolid
c/ Real de Burgos s/n, 47011, Valladolid, Spain
Telephone: 983423162. Fax: 983423161
e_mail: rachid@automo.uva.es
prada@autom.uva.es

Abstract

This work analyses the supervision of the performance of Model-based Predictive Controllers (MBPC). Different methodologies are analysed to assess the performance of the controller, determining a benchmark and supervising it online. The study involves the simulation of a chemical reactor, illustrating the results and demonstrating the limitations and advantages of these methods.

Key Words: Predictive control, supervision, EcosimPro, benchmark, cost function, simulation, optimisation

1. Introduction

Multivariable Model-based Predictive Control (MBPC) is gradually being applied in industry to handle complex processes. Its widespread success is due to the fact that it can be used in the majority of industrial processes as well as the advantages it offers, such as: its multivariable nature, handling of restrictions, handling of disturbances, prediction of controlled variables and calculation of the manipulated variables, etc.

Thanks to these factors, over the past decade Model-based Predictive Control has become the most prominent control strategy in advanced applications in the chemical industries [Scokaert & Rawlings, (1996)].

There are two key elements involved when it comes to designing a strategy for Model-based Predictive Control, hereinafter referred to as MBPC. They are the identification of the process model to be controlled and the selection of the controller tuning parameters. The incorrect use of these elements together with the effect of other internal or external factors can cause deflection of the optimal performance point for which the controller was designed. To guarantee good performance therefore, a method of supervision must be devised which at each moment informs of the efficiency of the results obtained, indicating how the controller is behaving during online performance.

This assessment is particularly important. If supervision reveals poor performance, then the causes must be determined and actions suggested to improve it and maintain optimal performance conditions so as to ensure adequate control of the process.

Analysis of the performance of a controller is divided into three stages, as shown in Figure 1:

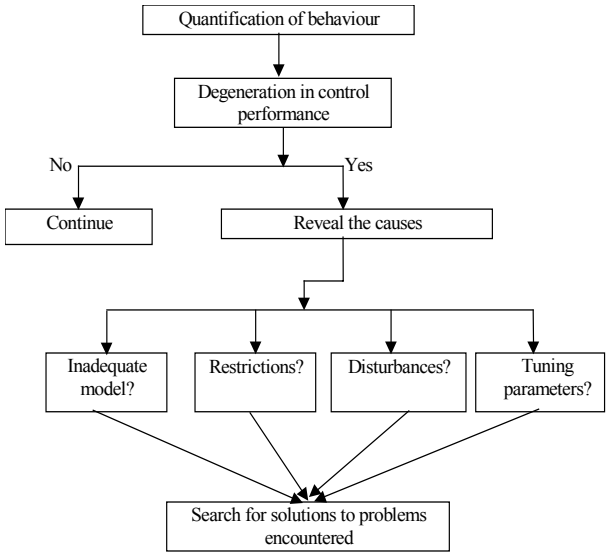


Figure 1: Sketch Illustrating the Stages of Analysis of Controller Performance

1. **First stage:** Performance assessment. This stage quantifies the performance and attempts to define and evaluate different methods to decide whether or not the control system shows satisfactory performance. In other words, it determines whether the problem is due to changes or disturbances in the process, or whether it really lies in the controller and its tuning.

2. **Second stage:** Diagnosis of poor performance. If the previous stage shows poor performance in the controller, the logical solution is to determine the causes and decide whether they are due to:
   - ✓ Inadequate model
   - ✓ Restrictions
   - ✓ Tuning parameters

- ✓ Disturbances

3. **Third stage:** Establishment of methods and guidelines to correct poor performance and restore control on the basis of the causes:

- ✓ If the model is the cause of degeneration, then online or offline identification must be established and a new model determined which fulfils the necessary requirements
- ✓ If the causes are linked to tuning, then guidelines to change horizons, methods or weights must be proposed to achieve better control. Selection of these parameters must constitute an efficient compromise between the performance of the path and the aggressiveness of control
- ✓ If restrictions are the cause, then methods to make them flexible must be proposed
- ✓ If disturbances are the cause, then the most efficient way must be found to offset them and find a model which describes the disturbances more adequately

Since numerous factors can cause abrupt or gradual deterioration in the performance of controllers it has to be supervised and assessed. It is often difficult to analyse diagnosis problems and supervise performance with unprocessed data taken from the plant [Kozub, 1997]. In such cases, the stochastic and statistical analysis tools must be suitably formulated to detect statistically significant changes.

Although this is a recent subject and there is not much information in the literature available, some methods of supervising MBPC have been proposed and they include the following:

Comparison of the real performance with the estimated optimal performance, resolving the LQG problem [Huang and Shah,1999]. Comparison of the real control performance with the historical performance employed by the expected cost function value of the MBPC, using a time frame within which it is known that the controller performs well [Patwardhan et al, 1998]. These authors also propose assessing controller performance using the objective cost function of the controller and of the process. Other works were carried out by Zhang and Henson who compare the objective function values from the output of the process linear model with the output of the real plant [Zhang and Henson, 1999]. A new approach to the matter has recently been taken in which the work is not limited only to supervising the performance; it now also monitors it and determines the causes of poor performance [Jochen Schaferand, Ali Cinar 2002].

Our study focuses on assessing the performance of MBPC using some of the aforementioned methods. Diagnosis and supervision is limited to assessing controller performance in different situations; for the moment we won't go into the steps taken to distinguish the causes and propose solutions to the problems which bring about degeneration of the controller. To illustrate the proposed methodology we use case studies based on a model of the chemical reactor. A modification to the benchmark is proposed and the simulation is run with EcosimPro.

2. Methods for Assessing MBPC Performance

MBPC methodology is based on real time optimisation of the following cost function presented in equation 2.1:

$$J^*_{(t)} = \sum_{sal=1}^{NSAL} \gamma_{(sal)} \sum_{j=1}^{N2} \left(\hat{y}_{(sal)(k)} - \hat{w}_{(sal)(k)} \right) + \sum_{ent=1}^{NENT} \beta_{(ent)} \sum_{j=1}^{Nu} \Delta u_{(ent)(k)} \qquad (2.1)$$

where:

k = t+j, y $\hat{W}$, $\hat{Y}$, Δu^* are the vectors of the internal reference of the controller, the process output prediction and the optimal vector of the control signal over time t

NENT, ***NSAL*** are the number of inputs and outputs

Nu, N2 are the control and prediction horizons, respectively

γ *and* β are the weight matrices of the controlled and manipulated variables

sal is the number of outputs

This equation includes information on the prediction and control horizons. The controller calculates the optimal control movements, minimising the objective function over the possible control movements [Patwardhan R. et al, 1998].

$J^*(t)$ is the design value aimed at during control, whereas the real cost function is a random variable affected by the disturbances and noises which affect the process (Jochen Schaferand, Ali Cinar, 2002].

If we describe J(t) as the desired cost function $J^*_{des}(t)$ then we could call the process cost function the real cost function $J_{real}(t)$, and the benchmark for assessment controller performance ρ, which is calculated by equation 2.2 as follows:

$$\rho_{(t)} = \frac{J^*_{des(t-N2)}}{J_{real(t)}} \qquad (2.2)$$

The expected value of this benchmark is: $E(\rho_{(t)}) = 1$

where E is the expectation.

2.1 Method of the Design Case and the Process

R Patwardhan et al propose comparing the desired cost function with the real cost function to obtain a benchmark ρ which shows the performance of the controller [Patwardhan R. et al, 1998]. This method is illustrated in Figure 2.

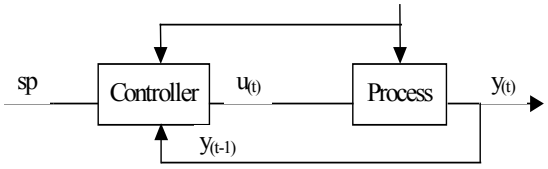


Figure 2: Graphic Illustration of the Method of the Design Case and the Process

The desired cost function J_{des} has the same form as equation 2.1. It is calculated with data from the controller obtained as a result of the predictions of the process variables and the control variables. The result of J_{des} must be saved so that it can be used when N_2 sampling periods have elapsed, together with the real cost function which is calculated in the time $t+N_2$ using process data obtained in the last $t-N_2$ sampling periods. The calculation of $J_{real}(t)$ is described in equation 2.3:

$$J_{real(t)} = \sum_{sal=1}^{NSAL} \gamma_{(sal)} \sum_{j=1}^{N2} \left(y_{(sal)(k)} - \hat{w}_{(sal)(k)} \right) + \sum_{ent=1}^{NENT} \beta_{(ent)} \sum_{j=1}^{Nu} \Delta u_{(ent)(k)} \qquad (2.3)$$

where $k = t + j - N_2$ and y is the system output, and the remaining parameters are the same as those explained above in equation 2.1. The real cost function J_{real} was calculated using data on the process variables which were obtained during the last $t-N_2$ sampling periods and the variables and parameters of controller tuning during the same period. Figure 3 graphically illustrates this explanation and how the analysis is performed once N_2 sampling steps have passed, since it is the minimum horizon required to be able to obtain information on the real process.

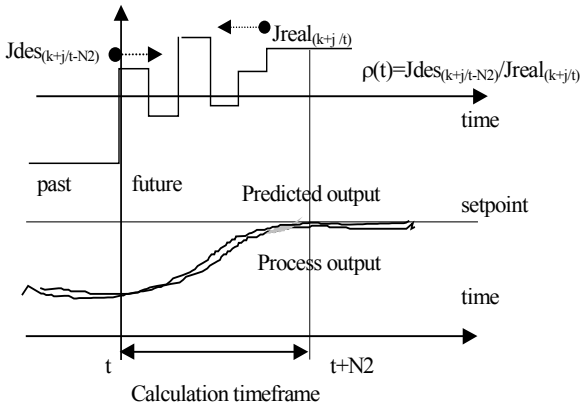


Figure 3: Graphic Illustration of the Calculation of Objective Functions over Time

Once these two objective functions which use information from the real process and information from the controller prediction results have been obtained, the benchmark can be determined by comparing the optimal design and the real objective function denoted by ρ and it is calculated as in equation 2.2.

2.2 Method of the Linear Model Case and the Process

Zhang and Henson in their work propose calculating the benchmark ρ by comparing the real process and the linear model to determine controller performance [Zhang and Henson, 1999]. Figure 4 graphically summarises this method giving an outline of its basic structures.

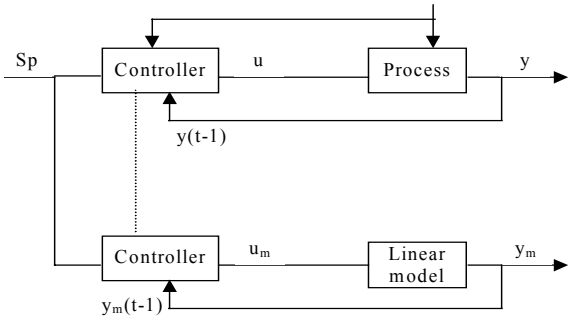


Figure 4: Graphic Illustration of the Method of the Linear Model Case and the Process

Unlike the previous method, the benchmark ρ is obtained with the desired cost function (J_{des}) when control is applied to the linear model of the process instead of the plant and the disturbances in the linear model are not taken into consideration, in this case taking into account that the controller tuning parameters and setpoints must be the same for both at every moment.

The cost functions will be calculated in the same way as equation 2.3 in the previous paragraph; that is, the necessary data is saved both for the process and for the linear model, and once N_2 sampling periods have elapsed the cost functions can be calculated.

J_{real} is obtained as shown in equation 2.4. The method of calculation is similar to that of equation 2.3, except that in this analysis the effect at the inlets is not taken into account.

$$J_{real(t)} = \sum_{sal=1}^{NSAL} \gamma_{(sal)} \sum_{j=1}^{N2} \left(y_{(sal)(k)} - \hat{y}_{(sal)(k)} \right) \quad (2.4)$$

where k = t+j-N_2, $\hat{y}$ is the vector of the real plant output prediction, and the remaining parameters are the same as those described previously.

However, as explained before, the desired cost function or J_{des} is obtained when control is applied to the linear model of the process instead of the plant. In other words, the data for calculating J_{des} are from the linear model and the calculation is similar to J_{real} in equation 2.4. J_{des} is calculated as shown below in equation 2.5.

$$J_{des(t)} = \sum_{sal=1}^{NSAL} \gamma_{(sal)} \sum_{j=1}^{N2} \left(y^{m}_{(sal)(k)} - \hat{y}^{m}_{(sal)(k)} \right) (2.5)$$

where k = t+j-N_2. y^m y $\hat{y}^m$ are the vectors of the outputs from the linear model and the prediction, respectively. Once these two objective functions of the process and the linear model have been obtained, the benchmark denoted by ρ can be determined by comparing the two and it is calculated as in equation 2.2. Figure 5 illustrates this calculation method. According to the authors, the method does not provide information on the effect of the parameters of MBPC tuning. They also state that the expected value of this function must be one that is calculated when the model is perfect and with no disturbances and noises. There is, however, no significant information if this is not the case, and so they turn to a statistical analysis of it.

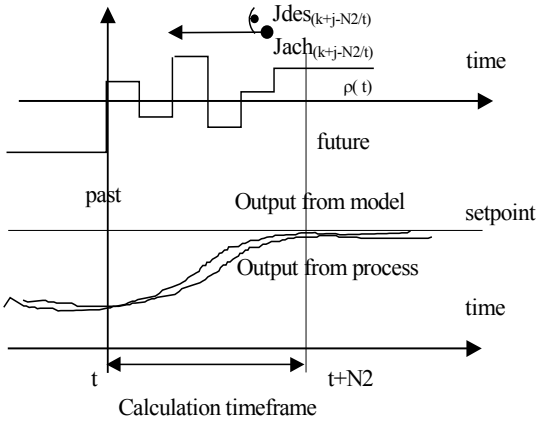


Figure 5: Graphic Illustration of the Calculation of Objective Functions over Time

Zhang and Henson identify ρ as a stochastic variable and propose performing a statistical assessment to detect statistically significant changes in controller performance. According to the authors, $\rho_{(t)}$ is a highly correlated sequence, so they propose using an ARMA model to identify them, as shown in equation 2.6.

$$A(q^{-1})\rho_{(t)} = C(q^{-1})\xi_{(t)} \quad (2.6)$$

where:

ξ is a random, zero average, uncorrelated Gaussian variable, $\mathbf{A(q^{-1})}$ and $\mathbf{C(q^{-1})}$ are the estimated polynomials using standard identification techniques and $\mathbf{q^{-1}}$ delay operand. These coefficients with variance can be identified using standard methods. But because $\rho_{(t)}$ is a highly correlated sequence, they propose using an AR instead of an ARMA model $(1 - a(q^{-1}))r(t) = x(t)$ and, lastly, they define $\mathrm{D}r(t)$ as in equation 2.7 and analyse its variance.

$$\mathrm{D}r_{(t)} = \frac{\hat{A}(q^{-1})}{\hat{C}(q^{-1})} r_{(t)} \quad (0.1)$$

The estimated noise variance is used to determine 95% of the confidence limit of $\Delta\rho$.

3. Modified Method of Controller Performance Analysis

The design case method probably does not apply to many real cases because the authors do not take into account in their analyses the unmeasurable disturbances that can affect the process, which exist in real cases and their consideration is of great importance. During the simulation of this method, is was observed that in the absence of unmeasurable disturbances controller performance is good, whereas

when they do exist controller performance deteriorates significantly even though it is operating correctly, thus limiting this method. Further on we will see a simulation of these two cases.

The design case method is limited to a special study. To overcome this difficulty we suggest that in the calculation, the original benchmark be changed for the benchmark shown in equation 3.1:

$$\rho_{(t)} = J_{real(t)} - J_{des(t-n)} \qquad (3.1)$$

where n is equal to N_2 in the method of the design case and the process, and zero in method of the linear model case and the process.

The cost functions J_{des}, J_{real} are calculated in the same way as in the design case method. The expected value of this new benchmark should be zero in optimal cases, since it is the difference between the expected cost function and the real cost function. But since this function is a random function due to the disturbances, its real value will be close to zero, and a value not far from zero is considered to be an appropriate value. The results obtained are satisfactory and can be seen in section 5.

In the method of the linear model case and the process, the same change of ρ was made in the calculation. We did this because the same occurred as in the previous case, and we consider that the information provided by this method was not sufficient for the analysis. As we will see in the illustrations further on, Δρ is almost zero after few simulations and it does not change even when the disturbances are increased or the tuning parameters are changed, and it is logical that disturbances are not considered in the linear model and J_{des} becomes very small.

4. Description and Simulation of the Process

The methods for assessing MBPC behaviour were applied to a chemical reactor into which a flow is introduced at the inlet, and due to the chemical reactions a new component is generated which will be returned to the outlet together with part of the original component [que no fue reaccionada???] which did not react. Due to the exothermic nature of the reaction, a coolant flow is introduced into the liner to keep the process temperature at the operating point, generating a change in its final temperature. The mass flows are kept the same since there is no loss during the reaction. Figure 6 gives a graphic illustration of the reactor which has the following characteristics:

- ✓ Three controlled variables: cb Concentration of product B at the outlet, tl Temperature inside the reactor at the outlet, ca Concentration of product A at the outlet
- ✓ Two manipulated variables: fl Inlet flow of product A, fr Coolant flow
- ✓ Three measurements of measurable disturbances: ca0 Concentration of inlet flow of product A, tl0 Temperature of the inlet flow, tr0 Temperature of the coolant inlet flow

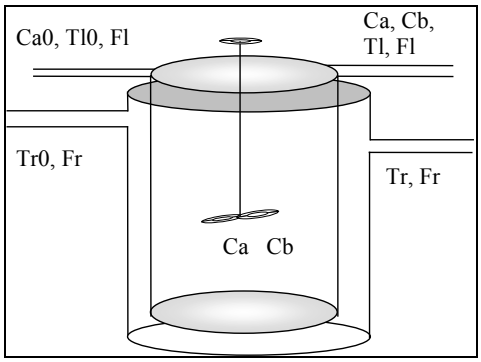


Figure 6: Graphic Illustration of the Chemical Reactor

4.1. Implementation of the Model in EcosimPro

To be able to simulate and assess MBPC performance, we have implemented in EcosimPro the mathematical model of the process in each of the cases -linear and non-linear- necessary to be able to run the simulations and study the performance in accordance with the above-mentioned methods. To this effect, each of the necessary functions programmed in C++ which intervene in the control process or in the supervision of the controller was called from EcosimPro, creating an independent library in each case.

To be able to perform process simulation tests in order to see how the controller performed –for example, when faced with changes in the benchmark, in its parameters or in the process disturbances- the differential equations which will serve as real plant were included in the continuous block of the component *react_proceso* in the *react_proceso.el* library. These are summarised below:

```
COMPONENT react_proceso(INTEGER nsal=3, INTEGER nent=2, INTEGER nper=3, INTEGER futuro=40)

DATA
…
DISCRETE
WHEN ( sample == TRUE  ) THEN
....
CONTINUOUS
        Ca' = f(Fl, Ca0, Vl, k)
```

```
        Cb' = f(Fl, Ca, Vl, k, Cb)
        Tl'  = f(Fl, Ca, Vl, Tl0,Tl,  k, H)
        Tr'  = f(Fr, Tr0,Tr, T, Vr)
….
END COMPONENT
```

The linear model of the process has been programmed in the discrete block of the component *react_modelo* in the *react_modelo.el* library. The code is as follows:

```
COMPONENT react_modelo (INTEGER nsal=3, INTEGER nent=2, INTEGER nper=3, INTEGER futuro=40)
DATA
....
DISCRETE
        WHEN ( sample == TRUE  ) THEN
….
Cb_mod      =      3.0534*Cbmod_old[2]      -
3.369994*Cbmod_old[3] \…. etc.
Tl_mod      =      3.5111*Tlmod_old[2]      -
4.6254905*Tlmod_old[3] \… etc.
Ca_mod      =      3.6213*Camod_old[2]      -
5.1413589*Camod_old[3] \… etc
….
END COMPONENT
```

To be able to run these simulations, the necessary functions of the controller were programmed in C++ using the source code of the HITO program which facilitated the application of the predictive controller and the calculation of the cost functions.

At the beginning, in each of the libraries, declarations have been made of the C++ functions which will subsequently be called in the discrete part. Some of these functions are shown below:

```
"C" FUNCTION REAL  regulador (IN REAL y[], IN REAL u[], IN REAL v[], IN REAL ref[], IN REAL y_predic[], IN REAL res_free[], IN INTEGER predic_ok[], IN REAL err_y_predic[], IN REAL ref_int[], IN REAL u_futura[], OUT REAL ro)
```

```
"C" FUNCTION NO_TYPE ini_hito (IN INTEGER NSAL, IN INTEGER NENT, IN INTEGER NPER, IN REAL y00[], IN REAL u0[], IN REAL v0[], IN INTEGER tipo_modelo, ...etc)
```

```
"C" FUNCTION NO_TYPE reini_hito (IN INTEGER cambio, IN INTEGER cambio_N1, IN INTEGER cambio_N2, IN INTEGER cambio_N3, IN INTEGER cambio_N4,  IN INTEGER cambio_NU, IN INTEGER cambio_res, IN INTEGER cambio_modelo, ...etc)
```

In the two cases, the initial part must include the call to an initialisation function of the controller *(ini_hito*) which passes certain necessary parameters in the execution of the controller (such as the horizons of prediction – *N*1, *N2* and control - *N*u, the weights of the cost function – *gam*a, *bet*a, the limits of the variables – *Lin*f, *Lsu*p, *Uin*f, *Usu*p, *Din*f, *Dsu*p, etc) and also ensures initialisation of the experiment associated with the partition of each component.

In the discrete zone, a call is made to the function *regulador* in each sampling period *(tsam*p) to obtain the new value of the manipulated variable that has to be applied to the process and the calculation of the benchmark ρ.

```
DISCRETE
….
WHEN ( sample == TRUE  ) THEN
….
regulador (y, u, v, refy, y_predic, y_libre, predic_ok, err_y_predic, ref_int, u_futura, ro)
….
sample = FALSE
sample = TRUE AFTER tsamp

END WHEN
```

Similarly, calls are made to other necessary functions in the discrete zone.

In order that the C++ code of the controller can be used from the EcosimPro environment, it is included as an external object the moment the partitions associated with the components are created, together with the other library used (la Nag) to indicate the place from where the functions used can be accessed.

4.2. Application of the Simulation to Specific Cases

With the method of the design case and the process, the simulation and the call to the controller for the calculation of the benchmark ρ did not give rise to complications. All that was required were data from the process and the controller whereby calls were made to the necessary functions and the data were updated during each sampling period.

However, calculation of the benchmark presented in the method of the linear model case and the process referred to in paragraph 2.2 required data from the linear model and the process. As indicated previously, the latter are presented in the EcosimPro libraries as *rector_modelo* and *reactor_proceso*. Analysis of controller performance requires that the controller tuning parameters be the same, as well as the benchmarks of the process and the linear model, and that any change be made in the two components.

To ensure these conditions, data exchanges and transfers had to be allowed between the two components, making one of them take the necessary data from or pass them to the other. The policy to be followed was that during each sampling period, each component was to first individually check if it was its turn. If it was, it would get permission and disable the permission of the other component until it had completed its functions and the data had been updated, at which point it would cede permission to the other component which remained on standby. If one of the components checks and finds it does not have permission, it has to wait until it is enabled by the other component. Once again with the help of C++ and within the source code of the HITO program, we programmed the necessary functions which would be called in the discrete zones to execute the operation.

This operation required that each component carry out three basic functions: save the data and parameters, update the data and parameters, and check its turn. The latter consists in checking the existence of a file which is created when a component takes its turn and which is deleted when its work is completed so that the other component can continue. When a component finds this file it waits until the other component deletes it, and once it has been deleted it has its turn.

As occurred previously, the functions of C++ which would be called during the simulation were declared global in each library. A summary of how these functions were declared and used is given below.

The following are the declarations of functions which gave a component its turn or took it away.

```
"C" FUNCTION NO_TYPE  borrar_arch ( )
"C" FUNCTION NO_TYPE  crea_arch  ( )
"C"  FUNCTION  NO_TYPE  chekea_arch  (OUT
INTEGER stado_arch)
```

The functions which gave a component its turn or took it away were written as shown below:

```
COMPONENT     react_X     (INTEGER     nsal=3,
INTEGER nent=2, INTEGER nper=3, INTEGER
futuro=40)
DATA
....
DISCRETE
        WHEN ( sample == TRUE  ) THEN
                chekea_arch (stado_arch)
        WHILE (stado_arch == 0)
                chekea_arch (stado_arch)
        END WHILE

-- Llamada al controlador predictivo
…..
borrar_arch()
END COMPONENT
```

Parameters were updated by each component as it had its turn. In other words, after the other component had saved the parameters and finished its turn:

guarda_param (ref_Cb, ref_Tl, ref_Ca, cambio, cambio_N1, cambio_N2,...etc)

actua_param (ref_Cb, ref_Tl, ref_Ca, cambio, cambio_N1, cambio_N2,...etc).

Concluding this section, we would add that the variables and the necessary data were obviously declared as DECLS or DATA, depending on the type of datum required.

5. Simulation Results

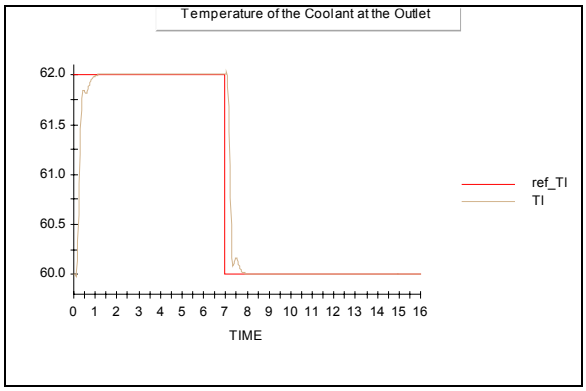
This section contains the results of the benchmark using the methods described previously and the modified method.

We will first present the results of the benchmark obtained using the method of the design case and the process. Afterwards, with the same data and without making changes in the controller tuning parameters, we will present the results using the modified method. We will then compare the two methods and comment on the results. After this analysis, we will go on to analyse the method of the linear model case and the process.

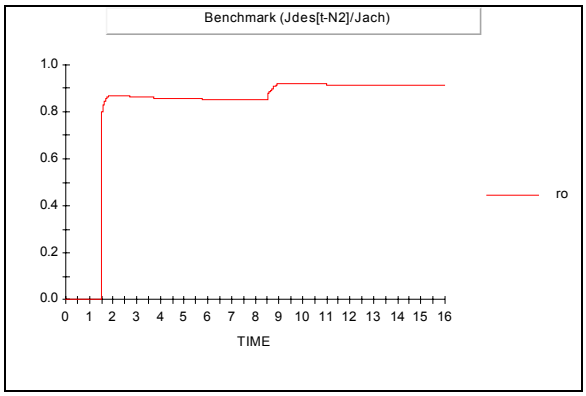
To carry out this simulation, the following data were considered:

- ✓ GPC controller
- ✓ Prediction horizons N2 = { 25, 25, 25 }
- ✓ Control horizon NU = { 2, 2 }
- ✓ Lowest physical limits Linf = { 6.7, 57.0, 0.0 }
- ✓ Highest physical limits Lsup = { 8.3, 72.0, 1.0 }
- ✓ Weight factors relative to the fulfilment of each setpoint - gamma = {3.0, 3.0, 0.1}
- ✓ Weight factors which penalise changes to the manipulated variables - beta = {0.5, 0.5}
- ✓ Setpoints
- ✓ Cb = 7.2, Tl = 62/60, Ca = 0.8

In this case, changes will only be made to the setpoint during simulation. From the start of simulation up to 7 hours, the coolant temperature setpoint at the outlet is fixed at 62, and from then on up to the end of simulation it is changed to 60 as shown in Graph 1.

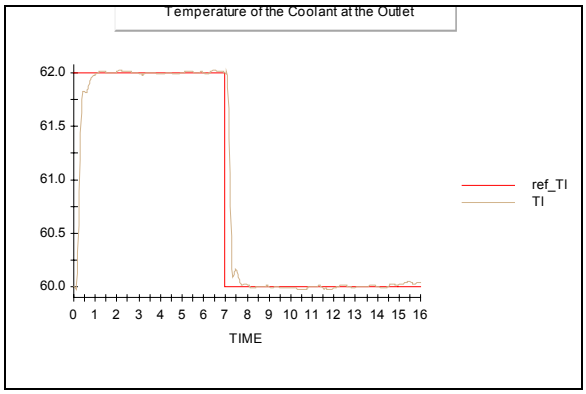


Graph 1: Coolant Temperature at the Outlet of the Process



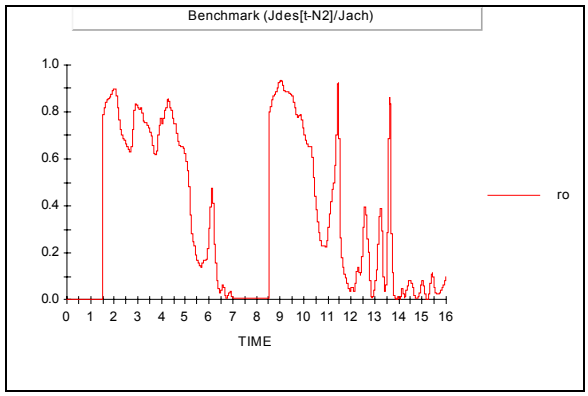
Graph 2: Benchmark ρ without Disturbances in the Process

En Graph 1 we can see the coolant temperature at the outlet of the process without consideration of disturbances, and in Graph 2 we can see the performance of the controller using the method of the design case and the process. With this method ρ had to be one so that performance was optimal; we can see that without disturbances the performance was very close to one, indicating an adequate result.



Graph 3

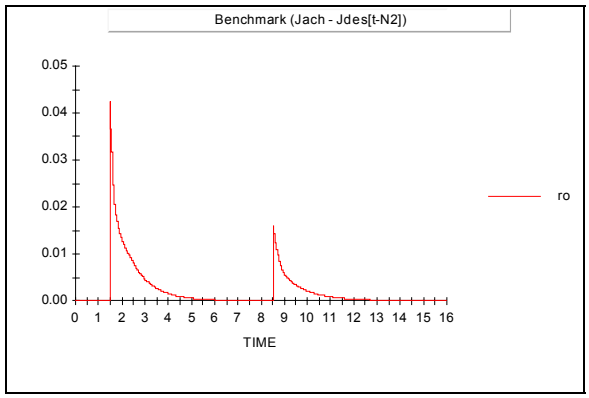
In Graph 3 we can see the temperature of the coolant at the outlet of the process, but with disturbances which were applied at the outlet of the process.



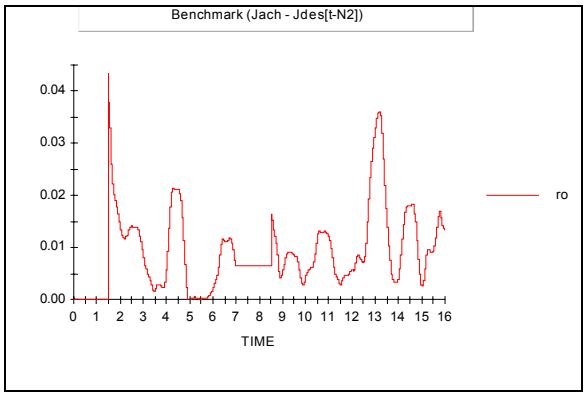
Graph 4 Benchmark ρ with Disturbances in the Process

In Graph 4 we can see the performance of the controller using the same method as before, but with disturbances. Unlike the result of Graph 2, the existence of disturbances has caused degeneration of the performance, while in reality the controller is working properly and the controlled variables are reaching the benchmark correctly. Therefore, this method of calculating the benchmark does not provide apt information to get an idea of controller performance with noise in the process.

Graphs 5 and 6 show the result of controller performance without and with disturbances, respectively, but applying the modified method of the design case and the process. In Graph 5 we can see the performance of the controller without the presence of noise, and as required the benchmark goes to zero. Graph 6 illustrates the presence of noise because the benchmark value is not exactly zero, but it does maintain a small value close to zero which shows that this way of implementing the benchmark is more suitable than the previous way.

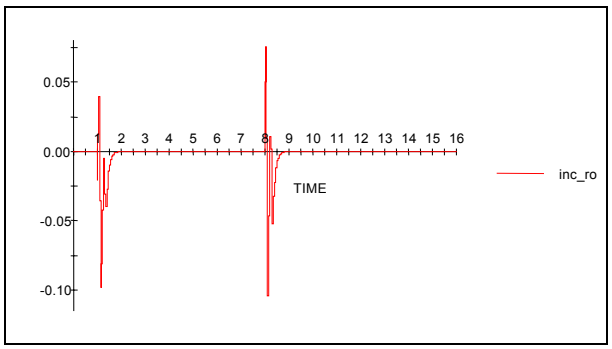


Graph 5: Modified Benchmark without Disturbances ρ

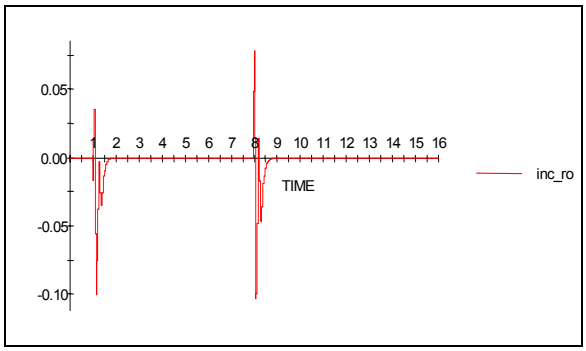


Graph 6: Benchmark ρ Modified with Disturbances

We are now going to present the benchmark results obtained using the method of the linear model case and the process. Afterwards, as we did at the beginning, using the same data and without making changes to the controller tuning parameters, we will present the results obtained using the modified method and we will then compare the two and comment on the results.

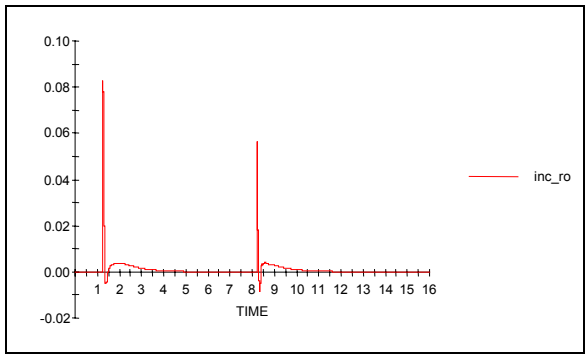


Graph 7: Increase in the Benchmark ρ without Disturbances

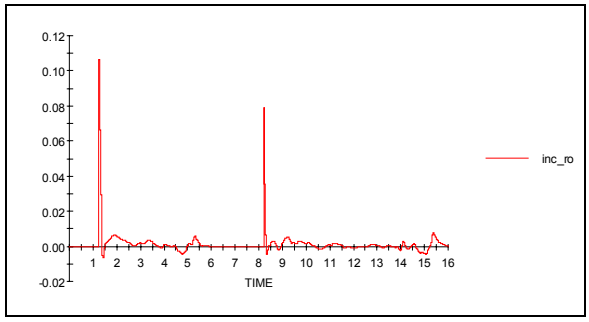


Graph 8: Increase in the Benchmark ρ with Disturbances

Graphs 7 and 8 illustrate the results of the simulation of Δρ without and with disturbances, respectively. As we can see, there are no significant changes between the two graphs even with disturbances which affect the process in the case of the second graph.



Graph 9: Results of Modified Δρ (Case without Disturbances)



Graph 10: Results of Modified Δρ (Case with Disturbances)

The last two Graphs, 9 and 10, present the results of the modified Δρ. Here it can be seen that the results of the two graphs are different, and this is obvious because the results of Graph 10 show that there were disturbances which affected the process.

Lastly, our presentation of the modified methods is at the moment limited to studying the performance of the predictive controller which is reflected in the results of the graphs presented throughout this work. The effects on controller performance caused by changes to its tuning parameters have not been addressed in this work, although tests have been performed and the results have been satisfactory.

6. Conclusions

We have presented two methods for supervising the performance of Model-based Predictive Controllers and a modification has been made to them. The two methods described have been simulated and their versions have been modified to assess MBPC performance.

EcosimPro was chosen to run the simulation because it can be applied to any problem which can be formulated with differential and/or algebraic equations and discrete events, and also because it facilitates integration of the model with other software modules; specifically, other applications written in C++.

A GPC controller was applied for control, using the source code of the HITO program [para ellos???] for the two methods.

The results of the simulation of supervision of the Predictive Controller have been presented and we have commented on them.

References

[1] Huang Biao, Kadali Ramesh, Zhao Xia, Tamayo, Edgar C., Hana Ahmed., (2000) An Investigation into the Poor Performance of a Model Predictive Control System on an Industrial CGO Cooker, Control Engineering Practice, 8, p 619-631

[2] Huang, B. and S.L. Shah. (1999) Performance Assessment of Control Loop. Springer-verlag, London

[3] Huang L., Shah S. L., Kwok E., (1997).Good, Bad or Optimal? Performance Assessment of Multivariable Process. Automatica, 33, p 1175-1183

[4] Huang L., Shah S. L., Fujii H., (1997) The Unitary Interactor Matrix and its Estimation using Closed-loop Data. Journal of Process Control, 7, p 195-207

[5] Jochen Schaferand, Ali Cinar. 15th Triennial Congress, Barcelona, Spain

[6] Kozub, D. J. (1999) Controller Performance Monitoring and Challenges. CPC-V Proceeding

[7] Patwardhan R. S., Shah S. L., GenichiEmoto, (1998) Performance Analysis of Model-based Predictive Controller: An Industrial Case Study. AIChE Annual meeting

[8] Scokaert P. O. M., Rawlings J. B., (1996) On Infeasibilities in Model Predictive Control. *Proceedings of CPC*-V. Tahoe City, CA, p 331-334

[9] Zhang Y., Henson M. A., (1999). A Performance Measure for Constrained Model Predictive Controllers. European Control Conference