C18

REAL-TIME OPTIMISATION ENVIRONMENT WITH ECOSIMPRO

Smaranda Cristea, César de Prada
Universidad de Valladolid
Prado de la Magdalena s/n, 47005 Valladolid - Spain
smaranda@autom.uva.es; prada@autom.uva.es

Abstract

This paper gives a general overview of how the EcosimPro simulation tool can serve as an environment for resolving problems of dynamic optimisation in a predictive controller.
A version of a CPBM algorithm has been developed which internally uses non-linear processing models and therefore calculates the action of the controller in a non-linear manner.

Key words: EcosimPro, predictive control, non-linear optimisation, real-time, simulation

1 INTRODUCTION

Predictive controllers based on models utilise an internal dynamic model to predict the future behaviour of the process in response to the manipulated variables. The future actions that must be applied to the process are calculated using optimisation procedures in conjunction with the model with respect to the control variables, taking into consideration the physical restrictions of the process.
The use of non-linear models heightens the possibility of improving control and, in turn, improving the quality of the predictions.
For continuous processes that work under nominal operating conditions and which are subject to small disturbances, the use of a non-linear model of the process does not offer much improvement. However, for processes that operate on a wider scale, at different work points where the linear model is no longer valid and the controller does not behave properly, the advantages of a non-linear model are greater.

2 PROBLEM OF CONTROL

2.1 USE OF THE MODEL

If we know the non-linear dynamics of the process we can use the model described by differential/algebraic equations and add a control algorithm based on non-linear optimisation which, if it interacts with a real process, becomes transformed into real-time optimisation. In this case it will not suffice that control system operation is correct from the mathematical point of view; it must also be correct from the temporary point of view.
Let us assume that the process is described by the following differential/algebraic equations:

$$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), \mathbf{u}(t); \mathbf{p}(t))$$
$$\mathbf{y}(t) = g(\mathbf{x}(t), \mathbf{u}(t); \mathbf{p}(t))$$

where **y** is the vector of the controlled variables, **u** is the vector of the manipulated variables whose variations in time will be the variables of decision for the optimisation problem, **x** is the vector of the states and **p** includes the unmeasurable disturbances.
The CPBM algorithms use the process model to calculate the predictions of its outputs to the desired limit and to evaluate the control actions by minimising a certain cost function during each sampling time:

$$\min_{\mathbf{u}} J(\mathbf{y}, \mathbf{u})$$

which is typically expressed as follows

$$J = \int_0^{tp} \left[g\,(y(t) - r(t))^2 + b\,(\Delta u(t))^2 \right] dt$$

considering it as the integral of the square of future errors between the predicted outputs of the process and the desired references and the square of future control actions.

2.2 OPTIMISATION

This problem can be solved by discretising the process model and by solving a non-linear programming problem with or without restrictions in the variables.
Another alternative would be to maintain continuous formulation and use a simulation language to integrate the equations which describe the dynamics of the process to be controlled. By allowing system simulation to evolve over a time equal to the desired prediction time (*tp*) (*tp*=*t***N*$_2$, where *t* is the sampling time and *N*$_2$ the horizon of prediction) using the actual process state as the initial conditions and considering the future

values of the manipulated variables as inputs, predictions are generated from the outputs and the cost index *J* is evaluated at the end of integration.
For the optimisation of *J*, the decision variables **u** which are considered to be constants at intervals must be parametrised and the control horizon *Nu* must also be defined so that

$$u(t+kt) = u(t+(k+1)t), \qquad k = Nu,..., N_2$$

as shown in Figure 1 below.

Figure 2 shows a block diagram of the structure of the predictive controller based on a non-linear model.
By resolving this NLP problem we obtain the optimum sequence of future control actions

$$\Delta\mathbf{u}_{\mathbf{opt}} = [\Delta u(t) \quad \Delta u(t+t) \quad ... \quad \Delta u(t+(Nu-1)t)]$$

but only the first movement $\Delta u(t)$ is applied to the process. The procedure is repeated during each sampling time.

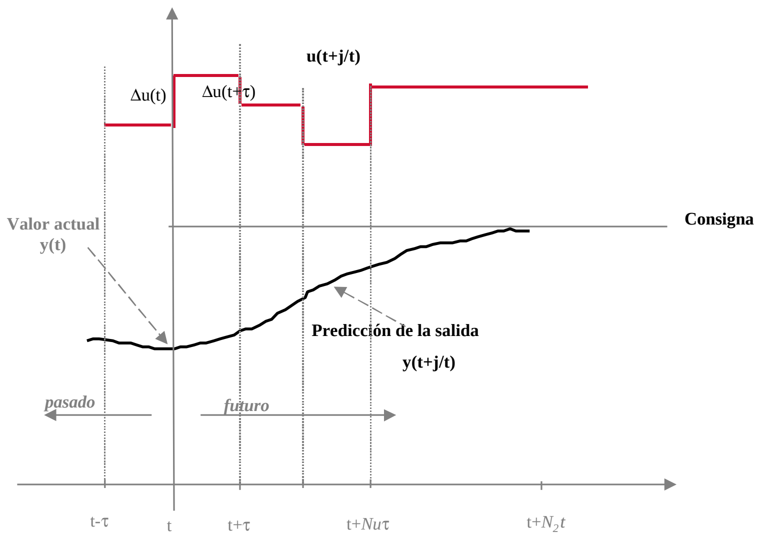

Figure 1 *Predictive Control Strategy*

Minimising the index *J* imposes the use of non-linear optimisation techniques subject to restrictions:

- on changes to the manipulated variables

$$|u(t+kt) - u(t+(k+1)t)| \le \Delta u_{max}, \quad k = 0,...Nu-1$$

- on the manipulated variables

$$U_{\inf} \le u(t+kt) \le U_{\sup}, \qquad k = 0,..., N_2$$

- on the outputs

$$Y_{\inf} \le y(t+kt) \le Y_{\sup}, \qquad k = 1,..., N_2$$

The controller is defined by two modules:

- one that makes reference to an EcosimPro component where the differential/algebraic equations which are to be controlled are defined as well as the objective function which is going to be minimised (*J*)
- the optimiser module responsible for calculating optimum control *u*(*t*) so that it can be applied to the process.

The calculation process of the new control *u*(*t*)

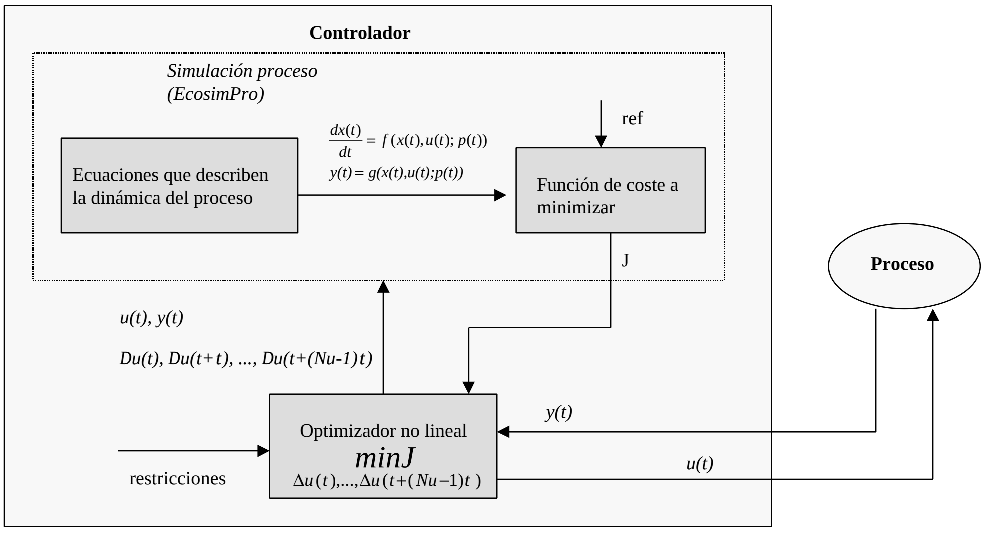

Figure 2 – *Structure of the non-linear predictive controller*

implies a permanent exchange between the two modules.

2.3 CALCULATION ALGORITHM

The optimiser, developed in C++ which, for example, can make use of a commercial non-linear optimisation routine with restrictions, will send to the simulator component (let's call it *objetivo.el*) both the initial conditions necessary to begin integration and the *Nu* values of the decision variables – the future values of the control actions $\Delta u(t), \Delta u(t+t), ..., \Delta u(t+(Nu-1)t)$ and when the integration finishes it will receive the value of the cost index *J* up to the final time $tp = N_2 * t$. To this end, an objective function is defined for the **optimiser**, from which the EcosimPro component which evaluates *J* is called. A typical code is shown below:

```
objfun (int n, double x[], double *obj, ...)
{
double indice;
char name[20];
int i;

...
// Pasar algunos parámetros
exper.setValueReal("beta", beta);
exper.setValueReal("gama", gama);
exper.setValueInt("N2", N2);
exper.setValueInt("NU", Nu);

// Pasar el valor actual de la salida
exper.setValueReal(("y", y);

// Pasar el valor actual del control
exper.setValueReal("u_proceso",
u_proceso);

// Pasar a Ecosim los incrementos futuros

for (i=0; i<Nu; i++)
{
sprintf(name,"delta_uf[%d]", i+1);
exper.setValueReal(name, x[i]);
}

// Pasar los valores iniciales para
// las variables de estado

...

// dynamic

exper.TIME = 0;
exper.TSTOP = N2*per_muestreo;
exper.CINT = 1.0;
exper.INTEG();

// Se recoge el valor del índice de coste
indice = exper.getValueReal("J");

*objf = indice;
}
```

where the variable *exper* is defined as

objetivo_simul *exper*;

where *simul* is the partition created from the objective component. The vector **x**[] contains the decision variables and *obj* is equal to the value of the objective function for the current point **x**.
Access from a C++ application to any variable of an EcosimPro component and its partition, defined as DATA or BOUNDARY type, is gained using the dedicated functions to modify or read a variable, for which the general form is *exp.setValueType* or *exp.getValueType*, respectively, where *Type* contains the type of variable (Real, Int, Bool, String, Enum) and *exp* is a generated experiment. In our case, to access certain variables of the simulation module from the optimiser, we have made use of the functions exper.setValueReal, exper.getValueReal and exper.setValueInt, depending on the type of variable involved.
We have also used the functions exper.TIME, exper.TSTOP and exper.CINT to indicate the initial and final integration time and the communication interval, respectively. To start the integration of the model from TIME up to TSTOP we have used exper.INTEG().

On the other hand, the simulator component would have the following structure:

```
COMPONENT objetivo (INTEGER maxnu = 10)
DATA
        ...
        REAL tsamp = 60

        INTEGER NU = 3
        REAL u_proceso = 25.8

        REAL ref = 1.97366
        REAL beta = 0.1
        REAL gama = 1.0

        -- Horizonte de predicción
        INTEGER N2 = 18
DECLS
        ...
        REAL J, u, uant
        REAL delta_uf[maxnu]
        INTEGER i = 0
        BOOLEAN Sample = TRUE
INIT
        ...
        u = u_proceso
        ...
DISCRETE
        WHEN (Sample) THEN
                IF(i < NU) THEN
                        i = i+1
                END IF

                uant = u
                u = uant + delta_uf[i]

                Sample = FALSE
                Sample = TRUE AFTER tsamp

        END WHEN
        ...
CONTINUOUS
```

```
-- Las ecuaciones dinámicas:
-- x' = f(x,u,p)
-- y = g(x,u,p)

...

-- El índice de costo
       J' = gama*(y-ref)*(y-ref) +
beta*(u-uant)*(u-uant)

END COMPONENT
```

In the discrete part, the continuous control signal *u* has been defined so that it has the structure shown in Figure 1.

2.4 TESTS DURING SIMULATION

To be able to carry out tests during simulation with the aim of observing the behaviour of the controller that has been implemented -for example in the light of changes in the reference, in its parameters or in the process disturbances- the 'Process' block in Figure 2 is substituted with another EcosimPro component which will serve as real plant. The initial part must include the call to a controller initialisation function (*inicio_con*) which passes certain parameters necessary in the execution of the regulator (such as the horizons of prediction – *N1*, *N2* and of control - *Nu*, the weights of the cost function – *gama*, *beta*, the limits of the variables – *Linf*, *Lsup*, *Uinf*, *Usup*, *Dinf*, *Dsup*, etc) and also ensures the initialisation of the experiment associated with the partition of the objective component: *initEcosim(&exper).*
Within the discrete zone, a call is made to the *controlador* function in each sample time (*tsamp*) to obtain the new value of the manipulated variable that must be applied to the process.
The dynamic equations, present in the continuous part of the component, do not have to be the same as those in the internal model (defined in *objetivo.el*).

```
"C++" FUNCTION NO_TYPE inicio_con(IN
INTEGER N1, IN INTEGER N2, IN INTEGER Nu,
IN REAL Linf, IN REAL Lsup, IN REAL Uinf,
IN REAL Usup, IN REAL Dinf, IN REAL Dsup,
IN REAL gama, IN REAL beta, IN REAL
per_muestreo)

"C++" FUNCTION REAL controlador(IN REAL
y_medida, IN REAL u_planta, IN REAL ref)

COMPONENT proceso
DATA
...
       REAL tsamp = 0.8
       INTEGER N1=1
       INTEGER N2=20
       INTEGER NU=2
       REAL beta=0.01
       REAL gama=1.0
       REAL Loperlowy=1.5
       REAL Loperupy=2.7
       REAL Loperlowu=-1.0
       REAL Loperupu=5.0
       REAL Llowincu=-0.5
       REAL Lupincu=0.5
DECLS
...
       REAL ym, u, refy
       REAL u_new
       BOOLEAN sample = TRUE
INIT
       inicio_con(N1, N2, NU,
              Loperlowy, Loperupy,
              Loperlowu, Loperupu,
              Llowincu, Lupincu,
              gama, beta, tsamp)
...
DISCRETE
...
-- Llamada al controlador predictivo
       WHEN (sample == TRUE) THEN
       u_new = controlador (y, u, refy)
       u = u_new
       sample = FALSE
       sample = TRUE AFTER tsamp
       END WHEN
...
CONTINUOUS
-- Las ecuaciones dinámicas:
-- x' = f(x,u,p)
-- y = g(x,u,p)
...
END COMPONENT
```

The controller function includes the call to the optimisation routine used, e04jbc (n, objfun,...) from the NAG library, where *objfun* is that which was defined in section 2.3. In this way, an EcosimPro component which serves as a process makes reference to another EcosimPro component which is being used as a non-linear optimisation environment.

2.5 EXAMPLE

As a simple example, we have chosen an SISO process described by the differential equation

$$t\,\frac{dy}{dt} + y^2 = u$$

where *y* is the controlled variable and *u* represents the manipulated variable. The time constant *t* is a datum fixed by the user.
In order to observe the behaviour of the simulated controller acting on the plant, the following component was built:

```
"C++" FUNCTION NO_TYPE inicio_con(IN
INTEGER N1, IN INTEGER N2, IN INTEGER Nu,
IN REAL Linf, IN REAL Lsup, IN REAL Uinf,
IN REAL Usup, IN REAL Dinf, IN REAL Dsup,
IN REAL gama, IN REAL beta, IN REAL
per_muestreo)

"C++" FUNCTION REAL controlador(IN REAL
y_medida, IN REAL u_planta, IN REAL ref)

COMPONENT ejemplo_simul

DATA
       REAL tau = 0.6
       REAL tsamp = 0.3 "sampling time"
-- Los horizontes
       INTEGER N1=1
       INTEGER N2=15
       INTEGER NU=1
-- Los pesos
       REAL beta=0.01
       REAL gama=1.0
-- Los límites
       -- en la salida
       REAL Linf = 0.2
       REAL Lsup = 1.7
       -- en la entrada
       REAL Uinf = -1.0
       REAL Usup = 3.0
       -- en las variaciones del control
       REAL Dinf = -2.0
       REAL Dsup = 2.0
DECLS
       REAL u, y, refy
       REAL u_new

       BOOLEAN sample = TRUE
INIT
       u = 0.25
       y = 0.5
       refy = 0.5

       inicio_con(N1, N2, NU, Linf, Lsup,
Uinf, Usup, Dinf, Dsup, gamma, beta,
tsamp)

DISCRETE
-- Llamada al controlador predictivo
       WHEN (sample == TRUE) THEN
       u_new = controlador (y, u, refy)
       u = u_new
       sample = FALSE
       sample = TRUE AFTER tsamp
       END WHEN
CONTINUOUS
       y' = (u-y*y)/tau
END COMPONENT
```

The C++ inicio_con and controller functions make reference to another component which represents the simulator of the model necessary for the optimisation:

```
COMPONENT ejemplo_coste (INTEGER maxnu=10)

DATA
       REAL tau = 0.6
       REAL tsamp = 0.3
       INTEGER NU = 3
       REAL u_proceso = 0.0
       REAL ref = 1.0
       REAL beta = 0.01
       REAL Lsup = 5.0
       REAL Linf = -5.0
DECLS
       REAL u, uant, J, Rinf, Rsup
       REAL delta_uf[maxnu]

       INTEGER i=0
       BOOLEAN Sample = TRUE
INIT
       u = u_proceso
DISCRETE
       WHEN (Sample) THEN
              IF(i < NU) THEN
                     i = i+1
              END IF
              uant = u
              u = uant + delta_uf[i]

              Sample = FALSE
              Sample = TRUE AFTER tsamp

       END WHEN
 CONTINUOUS

       y' = (u-y*y)/tau

       J' = (y-ref)**2 + beta*(u-uant)**2
              + Rsup*(Lsup-y)**2
              + Rinf*(Linf-y)**2

       Rsup = ZONE (y > Lsup) 1000.0
              OTHERS 0.0
       Rinf = ZONE (y < Linf) 1000.0
              OTHERS 0.0
END COMPONENT
```

In this case, to keep the output within the limits *Linf* and *Lsup*, penalty functions have been introduced in the *J* expression so that the moment that *y* goes beyond the permitted zone, the weights *Rinf* and *Rsup* would take on very large values.

In order to be able to use the C++ code of the controller (and especially the *inicio_con* and *controlador* functions) from the EcosimPro environment, it has been included in a static library (lib) which, the moment a partition associated with the component *ejemplo_simul* is created, must be added as an external object together with the other library being used (la Nag) to indicate the place from which the functions used can be accessed.

Figure 3 shows the response of the output *y* in the light of changes in the reference *refy* applying the non-linear controller with restrictions.
The variations in the value of the instruction have been generated using the BOUNDS zone of the associated experiment:

```
EXPERIMENT exp1 ON ejemplo_simul.cpnl
DECLS

INIT
   -- Dynamic variables
   y = 0.5

BOUNDS
       refy = 0.5
       + 0.5*step (TIME, 1.0, newInt())
       + step (TIME, 5.0, newInt())
       -1.5*step (TIME, 10.0, newInt())
       -0.4*step (TIME, 15.0, newInt())
BODY
       TIME = 0
       TSTOP = 25
       CINT = 0.1
       INTEG()
END EXPERIMENT
```

Some changes in the reference have been intentionally made outside the limits of the output to verify the efficiency of the penalty functions present in the optimisation index.

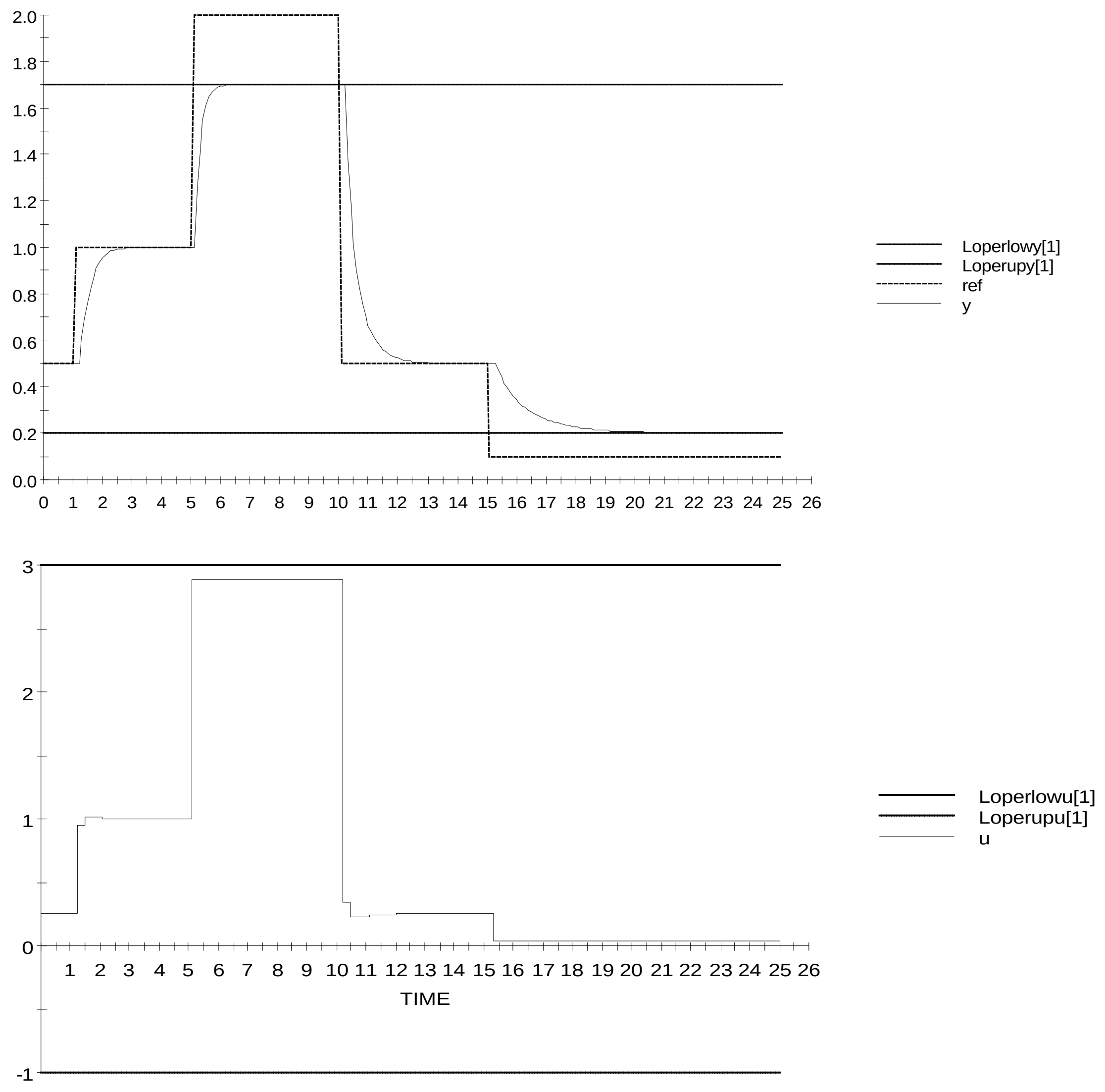

Figure 3. Process output and control action

3 CONCLUSIONS

We have seen how the behaviour of a process model can be predicted with the EcosimPro simulation tool and we have also addressed the optimisation procedure.

EcosimPro was selected for 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 of the ease with which the model can be integrated with other software modules, specifically other applications written in C++.
The results of the simulation show that, depending on the complexity of the equations and the structure of the component, the implementation of this predictive controller will require significant calculation time to generate the solution to the optimisation problem during each sampling time. This could lead to a situation in which the calculation time is greater than the sampling time, in which case we cannot talk about a real application.

Acknowledgements

The authors thank Pedro Cobas and Ramón Pérez of Empresarios Agrupados for the suggestions they have made which have helped in the development of this work, and for the interest they have shown in clarifying the doubts we have had.