C18

ENTORNO DE OPTIMIZACIÓN EN TIEMPO REAL CON ECOSIMPRO

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

Resumen

Este artículo presenta una visión general de cómo el lenguaje de simulación EcosimPro podría servir como entorno para resolver problemas de optimización dinámica en un controlador predictivo.
Se ha desarrollado una versión de un algoritmo de CPBM que usa internamente modelos no lineales del proceso y por lo tanto calcula la acción del controlador de forma no lineal.

Palabras Clave: EcosimPro, Control predictivo, Optimización no lineal, Tiempo real, Simulación

1 INTRODUCCIÓN

Los controladores predictivos basados en modelos usan un modelo interno dinámico para predecir el comportamiento futuro del proceso en respuesta a las variables manipuladas. Las acciones futuras que deben aplicarse al proceso se calculan usando procedimientos de optimización en conjunción con el modelo respecto a las variables de control, tomando en consideración las restricciones físicas del proceso.
El uso de modelos no lineales conlleva la posibilidad de mejorar el control mejorando la calidad de las predicciones.
Para procesos continuos que trabajan en condiciones de operación nominales y sujetos a pequeñas perturbaciones, el uso de una descripción no lineal del proceso no aporta mucha mejoría pero para procesos que operan en regiones amplias, en puntos de trabajo diferentes donde el modelo lineal deja de ser válido y el controlador no se comporta correctamente, las ventajas de considerar un modelo no lineal son mayores.

2 PROBLEMA DE CONTROL

2.1 USO DEL MODELO

El conocimiento de la dinámica no lineal del proceso nos permite utilizar el modelo descrito por las ecuaciones diferenciales/algebraicas y aplicar un algoritmo de control basado en optimización no lineal que, si interactúa con un proceso real, se transforma en una optimización en tiempo real. En este caso no basta que el funcionamiento del sistema de control sea correcto desde un punto de vista matemático sino que además sea correcto desde un punto de vista temporal.
Supongamos que el proceso está descrito por las siguientes ecuaciones diferenciales / algebraicas:

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

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

donde **y** es el vector de las variables controladas, **u** es el vector de las variables manipuladas cuyas variaciones en el tiempo serán las variables de decisión para el problema de optimización, **x** es el vector de los estados y **p** incluye las perturbaciones no medibles.
Los algoritmos de CPBM utilizan el modelo del proceso para calcular las predicciones de sus salidas a lo largo de un horizonte deseado y evaluar las acciones de control minimizando, cada periodo de muestreo, una cierta función de costo:

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

que típicamente suele ser de forma

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

considerándola como la integral del cuadrado de los errores futuros entre las salidas predichas del proceso y las referencias deseadas y del cuadrado de las acciones de control futuras.

2.2 OPTIMIZACIÓN

La resolución de este problema puede llevarse a cabo discretizando el modelo del proceso y resolver un problema de programación no lineal con o sin restricciones en las variables.
Otra alternativa sería mantener la formulación continua y emplear un lenguaje de simulación para integrar las ecuaciones que describen la dinámica del proceso que se quiere controlar. De este modo, dejando que la simulación del sistema evolucione un tiempo igual al tiempo de predicción (*tp*) deseado ($tp = t * N_2$, donde t es el

periodo de muestreo y N_2 el horizonte de predicción) tomando como condiciones iniciales el estado actual del proceso y considerando como entradas los valores futuros de las variables manipuladas, se generan las predicciones de las salidas y se evalúa el índice de coste J al final de la integración.

Para la optimización de J se necesita hacer una parametrización de las variables de decisión **u**, que se consideran constantes a trozos y también se define el horizonte de control Nu, de modo que

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

tal como se ha representado en la figura 1.

El diagrama de bloques de la estructura del controlador predictivo basado en un modelo no lineal se muestra en la figura 2.

Resolviendo este problema NLP se obtiene la secuencia óptima de las acciones de control futuras

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

pero se aplica al proceso sólo el primer movimiento $\Delta u(t)$. El procedimiento vuelve a repetirse cada periodo de muestreo.

El controlador está definido por dos módulos:

- el que hace referencia a un componente de EcosimPro donde se definen las ecuaciones diferenciales / algebraicas del proceso que se quiere controlar y también la función objetivo que se va a minimizar (J)
- el módulo del optimizador que se encarga de calcular el control óptimo $u(t)$ para aplicarlo al proceso.

El proceso de cálculo del nuevo control $u(t)$ supone un intercambio permanente entre los dos módulos.

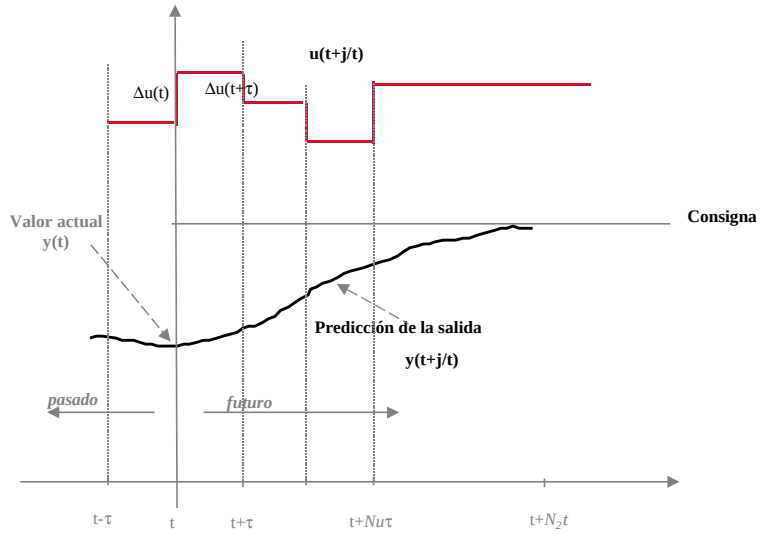

Figura 1 *Estrategia de control predictivo*

Minimizar el índice J impone el uso de técnicas de optimización no lineal sujeto a restricciones:

- sobre los cambios de las variables manipuladas

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

- en las variables manipuladas

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

- en las salidas

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

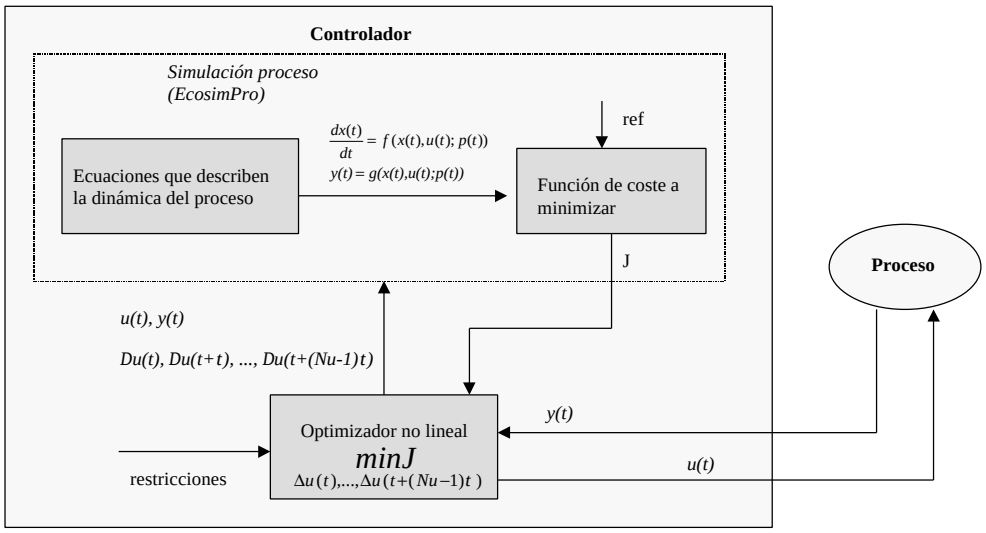

Figura 2 – *Estructura del controlador predictivo no lineal*

2.3 EL ALGORITMO DE CÁLCULO

El optimizador, desarrollado en C++, que por ejemplo puede hacer uso de una rutina comercial de optimización no lineal con restricciones, mandará al componente del simulador (vamos a llamarlo *objetivo.el*) tanto las condiciones iniciales necesarias para empezar la integración como los *Nu* valores de las variables de decisión - los valores futuros de las acciones de control $\Delta u(t), \Delta u(t+t), ..., \Delta u(t+(Nu-1)t)$ y recibirá el valor del índice de costo *J* al finalizar la integración hasta el tiempo final $tp = N_2 * t$. Con este fin, el **optimizador** tiene definida una función objetivo desde la cual se llama al componente de Ecosim que evalúa *J*. Un código típico se muestra a continuación:

```
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;
}
```

donde la variable *exper* se define como

objetivo_simul *exper*;

siendo *simul* la partición creada del componente objetivo. El vector **x**[] contiene las variables de decisión y *obj* es igual al valor de la función objetivo para el punto actual **x**.

El acceso desde una aplicación de C++ a cualquier variable de un componente Ecosim y su partición, definida de tipo DATA o BOUNDARY, se realiza utilizando las funciones dedicadas para modificar o leer una variable, cuya forma general *es exp.setValueType* o *exp.getValueType* respectivamente, donde *Type* contiene el tipo de la variable (Real, Int, Bool, String, Enum) y *exp* es un experimento generado. En nuestro caso, para acceder desde el optimizador a ciertas variables del modulo de simulación se ha hecho uso de las funciones exper.setValueReal, exper.getValueReal y exper.setValueInt dependiendo del tipo de la variable implicada.

También se han utilizado las funciones exper.TIME, exper.TSTOP y exper.CINT para indicar el tiempo inicial y final de integración y el intervalo de comunicación respectivamente. Para arrancar la integración del modelo desde TIME hasta TSTOP se ha usado exper.INTEG().

Por otra parte el componente del simulador tendría la estructura:

```
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
```

En la parte discreta la señal de control continua *u* se ha definido de modo que tenga la estructura que se presenta en la figura 1.

2.4 PRUEBAS EN SIMULACIÓN

Para poder realizar pruebas en simulación con el objetivo de ver el comportamiento del controlador implementado, por ejemplo ante cambios en la referencia, en sus parámetros o en las perturbaciones del proceso, en el diagrama de la figura 2, el bloque 'Proceso' se sustituye por otro componente de Ecosim que servirá como planta real. La parte inicial deberá incluir la llamada a una función de inicialización del controlador (*inicio_con*) que realiza el paso de ciertos parámetros necesarios en la ejecución del regulador (como los horizontes de predicción – *N1*, *N2* y de control - *Nu*, los pesos de la función de coste – *gama*, *beta*, los límites de las variables – *Linf*, *Lsup*, *Uinf*, *Usup*, *Dinf*, *Dsup*, etc.) y también asegura la inicialización del experimento asociado a la partición del componente objetivo: *initEcosim(&exper)*.
Dentro de la zona discreta se hace la llamada a la función *controlador* cada periodo de muestreo (*tsamp*) para conseguir el nuevo valor de la variable manipulada que debe aplicarse al proceso.
Las ecuaciones dinámicas, presentes en la parte continua del componente, no tienen porque ser necesariamente las mismas del modelo interno (definidas en 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
```

La función controlador incluye la llamada a la rutina de optimización utilizada, la e04jbc (n, objfun,...) de la librería NAG, donde *objfun* es la que se ha definido en la sección 2.3. De este modo desde un componente de Ecosim que sirve de proceso se hace referencia a otro componente de Ecosim que se está utilizando como entorno de optimización no lineal.

2.5 EJEMPLO

Para poner un ejemplo sencillo, se ha elegido un proceso SISO descrito por la ecuación diferencial

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

donde *y* es la variable controlada y *u* representa la variable manipulada. La constante de tiempo *t* es un dato fijado por el usuario.
Con el fin de ver el comportamiento del controlador en simulación actuando sobre esta planta, se ha construido el siguiente componente:

```
"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
```

Las funciones C++ inicio_con y controlador hacen referencia a otro componente que representa el simulador del modelo necesario para realizar la optimización:

```
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
```

En este caso, para mantener la salida dentro de los límites *Linf* y *Lsup*, en la expresión de *J* se han introducido funciones de penalización, de modo que en el momento en que *y* estaría fuera de la región permitida, los pesos *Rinf* y *Rsup* tomarían valores muy grandes.

El código C++ del controlador (en particular las funciones *inicio_con* y *controlador*), para poder ser utilizado desde el entorno Ecosim, se ha incluido en una librería estática (lib) que en el momento en que se crea una partición asociada al componente *ejemplo_simul* debe añadirse como objeto externo junto con la otra librería utilizada (la Nag) para indicar el sitio de donde se puedan acceder las funciones utilizadas.

La figura 3 visualiza la respuesta de la salida *y* frente a cambios en la referencia *refy* aplicando el controlador no lineal con restricciones.
Las variaciones en el valor de la consigna se han generado utilizando la zona BOUNDS del experimento asociado:

```
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
```

Algunos cambios en la referencia se han hecho intencionadamente fuera de los límites de la salida para comprobar la eficacia de las funciones de penalización presentes en el índice de optimización.

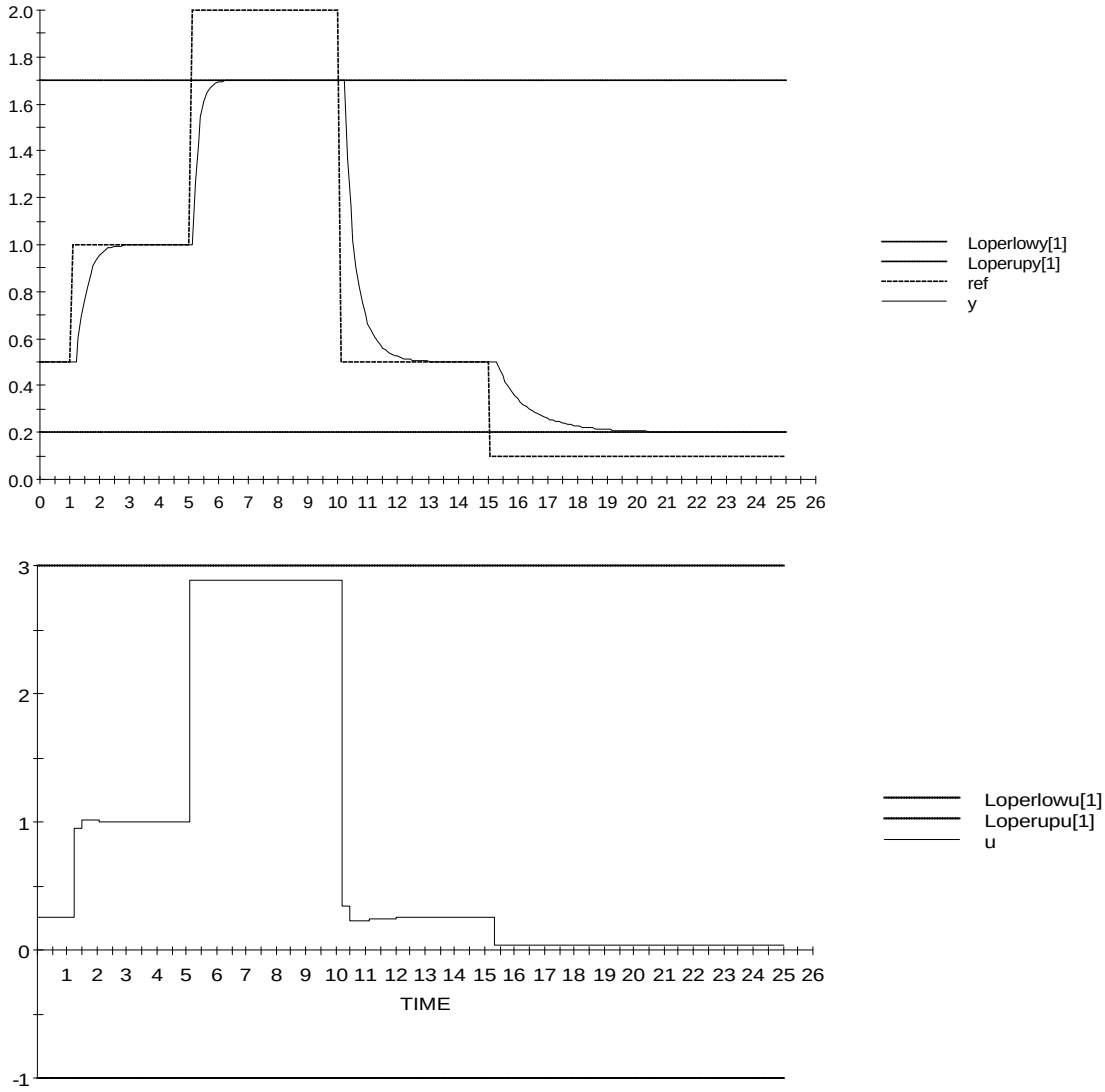

Figura 3. La salida del proceso y la acción del control

3 CONCLUSIONES

Se ha descrito el modo en que se puede utilizar el modelo de un proceso a través del lenguaje de simulación EcosimPro para predecir su comportamiento y también se ha presentado el procedimiento de optimización.

Se ha elegido como simulador la herramienta EcosimPro por ser aplicable a cualquier problema que pueda ser formulado con ecuaciones diferenciales y/o algebraicas y eventos discretos y también por la facilidad que ofrece de integrar el modelo con otros módulos de software, en concreto otras aplicaciones escritas en C++.

Esta implementación del controlador predictivo, debido a la simulación del modelo, tendría el inconveniente de requerir un tiempo significativo de cálculo, dependiendo de la complejidad de las ecuaciones y la estructura del componente, para generar la solución del problema de optimización cada periodo de muestreo. Esto podría llevar a la situación en que el tiempo de cálculo es mayor que el periodo de muestreo, caso en que no se puede hablar de una aplicación real.

Agradecimientos

A Pedro Cobas y a Ramón Pérez de Empresarios Agrupados por sus sugerencias que han ayudado en el desarrollo de este trabajo y por el interés mostrado en aclarar dudas que han ido surgiendo.