Universidade Federal do Ceará

Centro de Tecnologia

Pós-Graduação em Engenharia Elétrica

CONTROLE DE CONDUTIVIDADE UTILIZANDO ALGORITMOS PREDITIVOS PARA UTILIZAÇÃO EM IMPRESSÃO “OFF-SET

Eber de Castro Diniz

Fortaleza

Março 2006

Eber de Castro Diniz

CONTROLE DE CONDUTIVIDADE UTILIZANDO ALGORITMOS PREDITIVOS PARA UTILIZAÇÃO EM IMPRESSÃO “OFF-SET

Dissertação submetida à Universidade Federal do Ceará como parte dos requisitos para a obtenção do grau de Mestre em Engenharia Elétrica.

Prof. Luiz Henrique S. C. Barreto, Dr

Prof. Otacílio da M. Almeida, Dr

Fortaleza

Março 2006

RESUMO

Resumo da dissertação apresentada à Universidade Federal do Ceará como parte dos requisitos para obtenção do grau de Mestre em Engenharia Elétrica.

CONTROLE DE CONDUTIVIDADE UTILIZANDO ALGORITMOS PREDITIVOS PARA UTILIZAÇÃO EM IMPRESSÃO "OFF-SET"

O presente trabalho propõe o controle da condutividade do sistema de molha utilizando-se um algoritmo preditivo em conjunto com um identificador de sistema (algoritmo dos Mínimos Quadrados Estendido). Foi realizada uma comparação entre três algoritmos preditivos: GPC (Generalized Predictive Control), SPGPC (Smith Preditor Generalized Predictive Control) e SGPC (Stable Generalised Predictive Control), colocados em iguais condições de parâmetros numéricos, condições climáticas e utilizando o mesmo algoritmo de identificação, de modo a averiguar o de melhor desempenho para o processo de controle de condutividade.

Palavras-chave: Controle preditivo, GPC, SGPC, Preditor Smith, Identificação de Sistemas.

ABSTRACT

Abstract of dissertation presented at Universidade Federal do Ceará as partial fulfillment of the requirements for the Master degree of in Electrical Engineering.

CONDUCTIVITY CONTROL USING PREDICTIVE ALGORITHMS APPLIED TO OFF-SET PRINTING

The present work proposes the conductivity control of moisten system applying an predictive algorithm together with the Square Minima Extended. A comparison was accomplished among three predictive algorithms: GPC (Generalized Predictive Control), SPGPC (Smith Preditor Generalized Predictive Control) and SGPC (Stable Generalised Predictive Control), put in the same conditions of numeric parameters, climatic conditions, and using the same identification algorithm, to determine which it is the most suitable for the process of conductivity control.

Key words: Predictive Control, GPC, SGPC, Smith predictor, System identification.

SUMÁRIO

LISTA DE FIGURAS

SIMBOLOGIA

Símbolo	Significado
$y(t + k\|t)$	Horizonte de predição
$u(t + k\|t)$	Horizonte de controle
$w(t + k)$	Horizonte de referência
Δ	Fator diferencial
$\delta(j)$	Fator de ponderação do horizonte de predição
$\lambda(j)$	Fator de ponderação do horizonte de controle
J	Função de custo do algoritmo GPC
G	Matriz de peso da resposta forçada
F	Matriz de peso da resposta livre
I	Matriz identidade
N_1	Início do horizonte de predição
N_2	Fim do horizonte de predição
N_u	Tamanho do horizonte de controle
$u_{\rightarrow}$	Vetor do horizonte de controle
H_A	Hankel da matriz A
Γ_A	Submatriz do Hankel de A até o horizonte de controle
$c_{\rightarrow}$	Referência a ser seguida até o horizonte de controle
M_A	Submatriz do Hankel de A do horizonte de controle ao horizonte de predição
c_{∞}	Referência a ser seguida até o horizonte de predição
σ	Fator de esquecimento do algoritmo SGPC
x	Vetor de variáveis controladas
θ	Vetor de parâmetros
ε_m	Erro de modelagem
ε_0	Erro de observação
S	Função de custo do algoritmo dos mínimos

	quadrados
Θ	Matriz de parâmetros
X	Matriz de variáveis controladas
E	Matriz de erros
$\psi(k-1)$	Vetor de variáveis regressoras
ε*	Ruído branco

ACRÔNIMOS E ABREVIATURAS

Símbolo	**Significado**
CARIMA	Controlador Auto-Regressivo de Média Móvel(Controller Auto-Regressive Moving-Average)
ARMAX	Modelo Auto-Regressivo de Média Móvel com entrada exógena
SISO	Sistemas com uma entrada e uma saída(Single - Input Single-Output)
MISO	Sistemas com múltiplas entradas e uma saída(Multiple-Input Single-Output).
BIBO	Entrada Limitada Saída Limitada (Bounded-Input Bounded-Output)
MPC	Modelos de Controle de Predição(Model Predictive Control)
GPC	Controle Preditivo Generalizado(Generalized Predictive Control)
SPGPC	Controle Preditivo Generalizado utilizando Preditor Smith(Generalized Predictive Control with Smith Predictor)
SGPC	Controle Preditivo Generalizado Estável(Stable Generalized Predictive Control)

INTRODUÇÃO GERAL

O sistema de impressão "*off-set"* convencional utiliza o sistema de molha para diferenciar as áreas de grafismo e contragrafismo, isto é, regiões de impressão e não-impressão presentes no papel. Nas áreas da matriz que recebem água não será depositada tinta, possibilitando a tintagem somente nas áreas de grafismo. Sendo assim, pode-se pensar na importância de um bom e bem regulado sistema de molha, assim como o sistema de tintagem. A boa conservação e regulação dos sistemas mencionados acima são indispensáveis para se obter e se manter uma boa qualidade de impressão.

Nota-se que os principais problemas de impressão estão na maioria das vezes relacionados com a falta de regulação e manutenção dos sistemas de tintagem e molha. A manutenção do sistema regulado e em boas condições de uso não são os únicos fatores, deve-se considerar também os materiais e matérias-primas utilizadas, como a chapa, a tinta, a solução de fonte, a água utilizada, o álcool (ou outro agente tensoativo substituto), entre outros.

Portanto é pertinente dizer que um bom andamento do processo de impressão dependerá basicamente dos seguintes fatores: as condições mecânicas da impressora, a regulação dos componentes da máquina de impressão, os materiais e matérias-primas utilizadas, bons impressores e, especialmente, o controle de condutividade da água utilizado no sistema de molha.

Variações de condutividade superiores a mais ou a menos 200 µS dificultam bastante a manutenção do controle da alimentação de tinta e de solução de molhagem; condutividade baixa ocasiona problemas de velatura, emulsionamento excessivo, entupimento de pontos e manchas no produto impresso; condutividade elevada provoca problemas de cegueira de chapa, estrias de rolos e variações de cor durante a impressão. A manutenção da solução de

molha em um nível de condutividade desejado é de vital importância, tanto econômica quanto qualitativa, para as empresas gráficas.

Os modelos de controle preditivo tiveram um desenvolvimento considerável nos últimos anos, tanto no meio acadêmico quanto na indústria. A razão deste sucesso é atribuída ao fato de este ser o método mais abrangente quando se trata de problemas de controle no domínio do tempo[10]. O fato deste tipo de controle já ter sido utilizado em processos químicos[24] motivou a utilização desta família de algoritmos, uma vez que o processo tratado neste trabalho possui mesma natureza. Uma comparação entre três dos mais conhecidos algoritmos preditivos(GPC, SPGPC e SGPC) foi feita, de modo a analisar qual destes melhor se adaptaria ao processo em questão.

Este trabalho de dissertação se propõe a fazer uma análise de algoritmos preditivos no controle de condutividade, tendo sua robustez implementada pela adição de um modelador matemático da planta a ser analisada. No Capítulo 1 são descritas aplicações do controle de condutividade, bem como seu princípio. Inclui-se nestas aplicações o que se propõe no presente trabalho, o controle de condutividade em impressões "*off-set*". No Capítulo 2 faz-se uma explanação acerca dos algoritmos preditivos, tanto sua concepção quanto a sua descrição matemática. Os três algoritmos de controle utilizados no presente trabalho são descritos. O Capítulo 3 descreve os procedimentos de modelagem matemática de processos reais, e uma descrição mais detalhada do algoritmo dos Mínimos Quadrados Estendido, algoritmo utilizado para a modelagem matemática do processo do sistema de molha. No Capítulo 4, tanto o sistema como as ferramentas matemáticas utilizadas são descritos, tendo no Capítulo 5, a análise dos resultados experimentais obtidos.

CAPÍTULO 1

CONDUTIVIDADE EM LÍQUIDOS

1.1 INTRODUÇÃO

Condução elétrica é o movimento de partículas eletricamente carregadas através de um meio para sua transmissão. A condutividade elétrica é a medição deste fenômeno. Na água, geralmente é utilizada para medir concentração de mineral ou íons. A condutividade mede a pureza da água ou a concentração de outros materiais químicos possuidores de íons.

A água que utilizada no dia a dia, seja encanada, seja advinda dos rios ou chuvas, contém várias substâncias. Por isso ela não pode ser considerada água pura (H_2O). Devido a este fato, a água encanada pode ser utilizada no diariamente. Porém, com relação às pesquisas científicas ou industriais é necessário,esporadicamente, a utilização da água pura.

A técnica utilizada para diminuir a condutividade, afim de conseguir uma água mais pura, baseia-se na remoção gradual de eletrólitos. No entanto, se todos os eletrólitos são removidos, a condutividade não decai até zero. Isto ocorre pelo fato de uma mínima parte das moléculas de água, cerca de 500 p.p.m, possui íons de hidrogênio (H^+) e hidroxilas (OH^-). Teoricamente, neste ponto, a condutividade torna-se 0.0548 µS/cm at 25 °C. A condutividade da água é afetada pela temperatura, por conta que a mesma se torna menos viscosa e os íons podem mover-se mais livremente a maiores temperaturas. Por convenção, as medidas de condutividade possuem referências a uma temperatura de 25 °C, embora ocasionalmente a referência de 20 °C também seja utilizada.

1.2 MEDIÇÃO DA CONDUTIVIDADE DA ÁGUA SEM NECESSIDADE DE ELETRÔDOS.

A ausência de eletrôdos faz com que a medida de condutividade não requera o contato direto com o líquido na qual se deseja medir a condutividade. Ao invés disto, um par de fios circundando dois núcleos toroidais é encapsulado em torno de uma caixa protetora, exercendo a função do sensor, como mostrando na Figura 1.1. Um toróide é colocado em tensão constante, o qual gera um campo magnético no líquido. Com isso, a corrente induzida pelo campo magnético no líquido acopla magneticamente o segundo toróide, produzindo corrente. Esta corrente induzida pelo enrolamento do segundo toróide é diretamente proporcional ao valor do acoplamento magnetico, variando este valor proporcionalmente com a condutividade da solução. Esta corrente, uma vez amplificada, fornece o valor da condutividade na solução a ser medida.

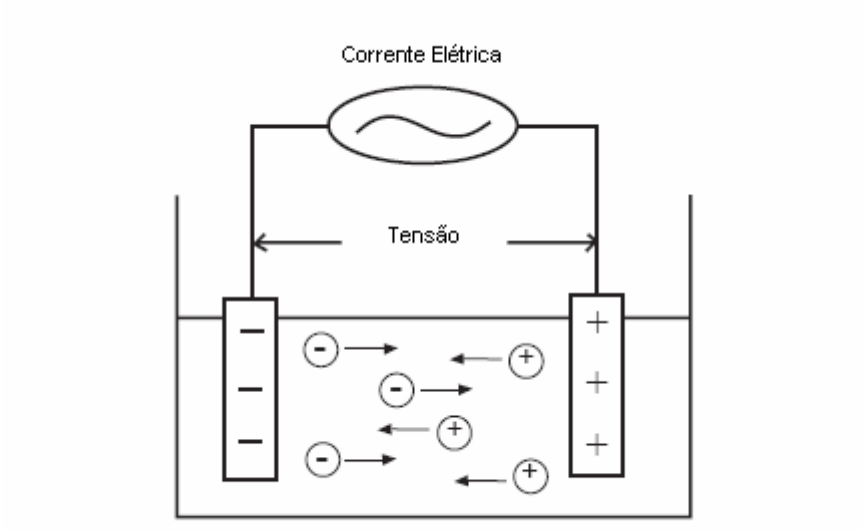

Figura 1.1 – Eletrodos toroidais induzem uma corrente elétrica no líquido com a finalidade de obter a condutividade do mesmo.

1.3 COMPENSAÇÃO DE TEMPERATURA

Nos líquidos, quando a temperatura aumenta, ocorre o mesmo com a condutividade. A condutividade medida em uma certa temperatura, portanto deve ser compensada, para que

diferentes leituras possam ser comparadas. Este é um fato relevante por conta da precisão em que a maioria dos processos necessita.

1.3.1 POLARIZAÇÃO

A aplicação de corrente elétrica na solução pode causar o acúmulo de espécies iônicas nas proximidades da superfície do eletrodo, ou no caso do presente trabalho no invólucro toroidal, como mostrado na Figura 1.2. Portanto, a resistência de polarização cresce na superfície do eletrôdo, resultando em medições incorretas devido a presença de componentes parasitas na condutividade da solução. Este problema pode ser resolvido colocando o eletrôdo em uma região onde o fluxo de líquido seja suficiente de forma a “limpar” estes depósitos de espécies iônicas.

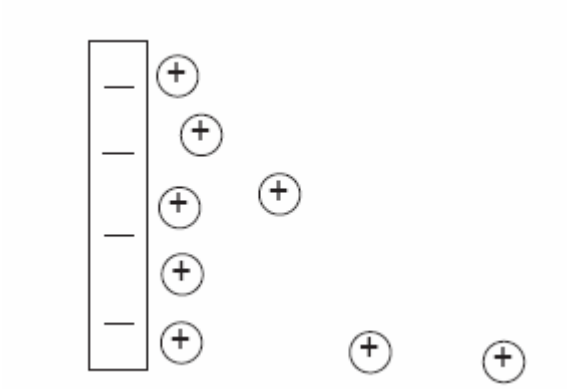

Figura 1.2 – Íons presentes nas proximidades do eletrodo causam erros de medição de condutividade nos líquidos.

1.3.2 EFEITOS DE CAMPO

Erros causados por efeito de campo, isto é, a parte da medição do campo que fica fora do espaço geométrico, mostrado na Figura 1.3. Quando a medição é realizada próximo as extremidades do sensor os valores obtidos não representam a real condutividade do sistema, uma vez que esta região apresenta não-linearidades. O uso do sistema com eletrôdos encapsulados diminui em grande parte o problema causado.

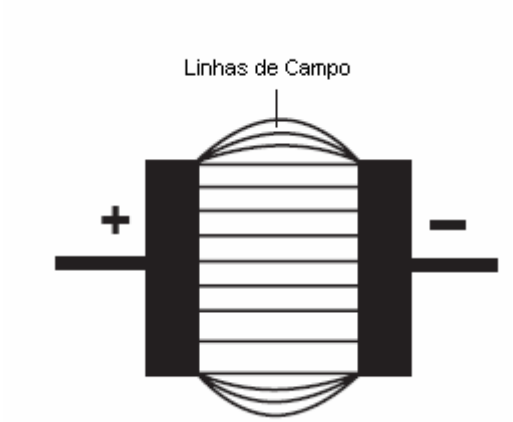

Figura 1.3 – Erros causados pelo efeito de campo. Observe-se que a leitura feita nas extremidades dos eletrodos torna-se não-line

1.3.3 MUDANÇAS NA FREQÜÊNCIA.

Baixas freqüências são aplicadas em medições de baixa condutividade, onde a resistência de polarização é mínima comparada a resistência da solução. Neste caso também a contribuição para a redução no efeito capacitivo no cabo de transmissão do sensor, que torna-se inconveniente quando a condutividade é baixa. Altas freqüências são aplicadas em medições de condutividade alta, onde a condutividade da solução é baixa. Na maioria das medições, a freqüência cresce proporcionalmente ao crescimento da condutividade da amostra da solução, evitando erros de polarização em altos valores de condutividade.

1.4 IMPORTÂNCIA DA CONDUTIVIDADE DA ÁGUA EM IMPRESSÕES OFFSET.

O controle de condutividade da água em impressões off-set (em sua maioria jornais diários) é feita, na maioria dos casos, por um processo manual que utiliza equipamentos de medida, eletrônicos ou analógicos, para verificação. Isto afeta tanto a qualidade do jornal quanto o custo de manutenção do maquinário de impressão[22].

A manutenção de uma condutividade em um nível desejado é um fator importante para a secagem da tinta no papel, e também para o brilho do material impresso. A não manutenção desta condutividade acarreta a não limpeza na área de não-impressão, o que ocasiona em

manchas no papel. Também afeta no deslocamento de fibras do papel para tinta via rolo de impressão, acarretando em sujeira da primeira e conseqüente troca, sendo que o preço desta tinta é geralmente muito elevado. Além disso a fixação da tinta no papel não se dá de modo uniforme, de modo que ao manusear a publicação o usuário tem partes da tinta aderida nas partes de contato.

Vários fatores influem na condutividade da água, que vai desde os agentes da mistura quanto fatores externos, como por exemplo a chuva. Por isso o controle manual não possui requisitos necessários para adquirir uma boa impressão.

1.5 CONSIDERAÇÕES FINAIS

A aplicação do controle de condutividade da água está ligado a diversas atividades, o que torna este trabalho aplicável a diversos ramos da atividade industrial. No presente trabalho nos atemos a condutividade aplicada em impressões *off-set*, utilizado com grande freqüência em parques gráficos de jornais e revistas. A redução nos custos e a melhora da qualidade de impressão foram os principais objetivos da motivação desta dissertação.

CAPÍTULO 2

ALGORITMOS PREDITIVOS

2.1 INTRODUÇÃO

Controles Baseados em Predição ou Model Predictive Control (MPC)[10] remontam sua origem da década de 70[9], desde então tiveram um considerável desenvolvimento. O termo MPC não se refere a um algoritmo ou a uma estratégia de controle em particular, mas sim a todas as estratégias que utilizam um modelo de processo o qual tem por finalidade obter um sinal de controle responsável por minimizar a função de custo. Este tipo de método deriva controladores lineares e não-lineares com estruturas muito similares. As idéias principais da família de algoritmos preditivos são[10] :

- Uso explícito de um modelo para predição da saída do processo em instantes futuro (aqui chamado de horizonte).
- Cálculo da sequência de controle de modo a minimizar uma função de custo.
- A cada instante o sinal de controle ao longo do horizonte de predição é calculado, sendo que somente o primeiro sinal de controle (que realmente irá atuar no sistema) é aplicado.

Os algoritmos MPC diferem em relação ao modelo no qual a planta a ser controlada é baseada, no tipo de ruído a ser modelado e na função de custo a ser reduzida. Este tipo de estratégia possui muitos trabalhos publicados com sua aplicação, além de ser amplamente utilizado no meio acadêmico e na indústria. Dos trabalhos baseados em MPCs, podemos citar o controle de robôs ([1]) e de anestesia clínica[2]. O bom desempenho deste algoritmo nestas aplicações mostra a capacidade dos algoritmos MPC de alcançar um controle eficiente de

sistemas, além de operar durante longos períodos de tempo sem intervenção humana(por serem adaptativos, ou self-tunning).

As estratégias MPC apresentam uma série de vantagens se comparados com outros métodos, entre as quais citamos:

- Possuem um conceito muito intuitivo. (o que os torna muito atrativo para pessoas com conhecimento de controle limitado).
- Podem ser utilizados em uma grande variedade de processos, desde os com dinâmicas simples até mais complexas, incluindo processos com grande atraso de transporte, de fase não-mínima ou instável.
- A sua implementação em sistemas MISO ou MIMO é relativamente fácil, desde que se conheça o algoritmo para sistemas SISO e seja utilizado em sua forma mais básica, isto é, sem restrição.
- Possuem uma metodologia de domínio público com princípios básicos, o que permite aprimoramentos, como no caso do SGPC [3].

2.1.1 ESTRATÉGIA MPC

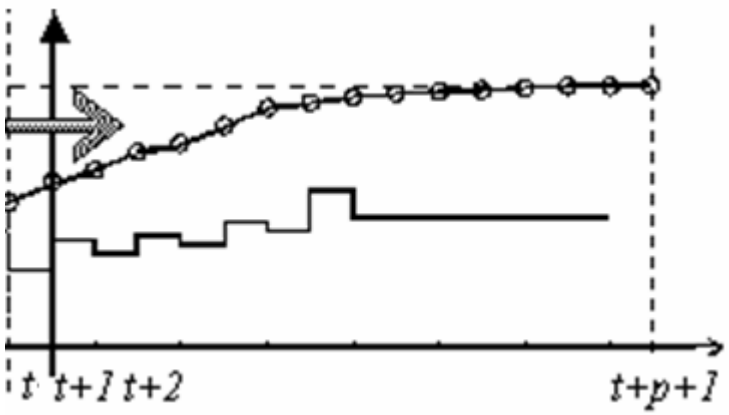

Figura 2.1 - Horizontes em um modelo preditivo. A linha cheia com bolinhas representa o horizonte de saída livre (“free-respons”) do sinal a partir do instante t+1. A linha logo abaixo representa as variáveis de controle calculadas a serem aplicados nos atuadores, em um futuro de predição definido.

A metodologia de todos os controladores da família MPC é caracterizada pela seguinte estratégia, representada também na figura 2.1:

1. As saídas futuras para um determinado horizonte N, chamado horizonte de predição, são previstas em cada instante t usando o modelo do processo. Cada saída prevista $y(t + k|t)$ (onde esta notação indica o valor da variável y no instante $t+k$ calculada no instante t) para $k=1,2,3...N$ depende dos valores conhecidos da saída (ou também da entrada) até o instante t e dos valores de controle futuros $u(t + k|t)$, $k=0,1,2...N-1$, que são os sinais os quais serão enviados ao sistema e serão calculados.
2. A escolha dos futuros sinais de controle é calculada otimizando-se um determinado critério com o intuito de manter o processo o mais próximo possível da trajetória de referência $w(t + k)$. Este critério geralmente toma a forma de uma função de erros nos quais estes são tomados como a diferença entre o sinal de saída e o sinal predito, elevado ao quadrado. O esforço de controle está incluído nesta função, na maioria dos casos.
3. O sinal de controle $u(t)$ é enviado ao processo enquanto os sinais de controle calculados, referentes a instantes no futuro($u(t+1), u(t+2)...u(t+N-1)$) são descartados, pois no período de amostragem seguinte o valor da saída $y(t+1)$ já é conhecido, então o passo 1 é repetido com este novo valor e todas as seqüências são atualizadas. Esta estratégia é conhecida como horizonte deslizante.

O modelo do processo também é de fundamental importância na estratégia de controle. O modelo escolhido deve ser capaz de detectar as dinâmicas do processo de modo a realizar previsões mais precisas das saídas futuras. Além disso, deve ser de fácil compreensão. Sendo

o MPC não é um modelo único e sim uma estratégia de controle, muitos modelos podem ser utilizados para tal intuito.

O modelo baseado em função de transferência é, talvez, o mais difundido na comunidade acadêmica e utilizado na maioria dos métodos de modelagem de processos, pois sua representação emprega poucos parâmetros e é valida para todos os tipos de processos [1][9][13][18]. A formulação em espaço de estados é também utilizada em algumas formulações, pois descreve facilmente sistemas multi-variáveis (MISO, SIMO ou MIMO).

A utilização do controle preditivo tem grande vantagem com relação aos métodos de controle clássicos, como o PID, as variáveis dos atuadores são controladas baseadas em erros passados, enquanto o controle preditivo trabalha tanto com erros passados como futuros, estes últimos previstos[11].

A seguir os três algoritmos de controle utilizados neste projeto serão descritos.

2.2 CONTROLADOR PREDITIVO GENERALIZADO (GENERALIZED PREDICTIVE CONTROL (GPC))

O algoritmo GPC foi proposto por Clarke *et al.*[4] e se tornou um dos mais populares algoritmos preditivos utilizados tanto na indústria quanto no meio acadêmico. Sua eficácia tem sido comprovada em muitas aplicações industriais, mostrando boa desempenho e certo grau de robustez.

A idéia básica do algoritmo GPC consiste em calcular uma seqüência futura de sinais de controle de modo a minimizar uma função de custo definida sob um horizonte de predição utilizando modelos de equações a diferenças polinomiais. O índice a ser otimizado é a esperança matemática de uma função quadrática medindo a distância entre a resposta do sistema previsto e uma referência, também predita, ambas sob um horizonte de predição, somado a uma função quadrática que mede o esforço de controle.

Este algoritmo, além de fornecer uma solução analítica para os problemas de controle, pode ser utilizado em plantas instáveis e de fase não-mínima, incorporando o conceito de horizonte de controle e de fator de esquecimento em sua função de custo.

Primeiramente, introduziremos o modelo geral da planta a ser analisada, mostrada na equação a seguir:

$$A(z^{-1})y(t) = B(z^{-1})z^{-d}u(t-1) + C(z^{-1})\frac{e(t)}{\Delta} \tag{2.1}$$

Sendo *u(t)* e *y(t)* sinais de controle e de saída da planta, respectivamente, e *e(t)* é um ruído branco. Este modelo é conhecido como Controlador Auto-Regressivo de Média Móvel (CARIMA), o qual tem-se utilizado em muitas aplicações industriais nos quais se faz a modelagem de distúrbios não-estacionários.

Os polinômios *A*, *B* e *C* são os seguintes polinômios, onde z^{-1} é o operador de atraso:

$$\begin{aligned} A(z^{-1}) &= 1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_{na} z^{-na} \\ B(z^{-1}) &= b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_{nb} z^{-nb} \\ C(z^{-1}) &= c_0 + c_1 z^{-1} + c_2 z^{-2} + \ldots + c_{nc} z^{-nc} \end{aligned} \tag{2.2}$$

A função de custo do GPC, a ser minimizada, é mostrada a seguir:

$$J(N_1, N_2, N_U) = \sum_{j=N_1}^{N_2} \delta(j)\left[y(t+j\,|\,t) - w(t+j)\right]^2 + \sum_{j=1}^{N_u} \lambda(j)\left[\Delta u(t+j-1)\right]^2 \tag{2.3}$$

Sendo que *y(t + j|t)* é uma predição ótima *j* passos a frente da saída do sistema no instante *t*, N_1 e N_2 são os horizontes de custo mínimo e máximo, respectivamente, N_u é o horizonte de controle,$\delta(j)$ e $\lambda(j)$ são os fatores de ponderação e *w(t + j)* é a trajetória de referência a partir do instante de predição,isto é, a partir do instante *t*.

O objetivo do controle é calcular a seqüência futura dos sinais *u(t),u(t+1),u(t+2)* de modo que a saída futura da planta *y(t + j)* aproxime-se, o máximo possível, do sinal de referência *w(t + j)*. Este objetivo é alcançado minimizando a equação (2.3).

A otimização da função de custo com predição ótima de $y(t + j)$ para $j >= N_1$ e $j <= N_2$ pode ser obtida através do uso da equação de Diophatine:

$$1 = E_j(z^{-1})\tilde{A}(z^{-1}) + z^{-j}F_j(z^{-1}) \tag{2.4}$$

sendo:

$\tilde{A}(z^{-1}) = \Delta A(z^{-1})$

Os polinômios F_j e E_j são únicos, sendo o primeiro de grau *na*, que representa o grau do polinômio $A(z^{-1})$, e o segundo de grau $j-1$. Estes podem ser obtidos dividindo-se 1 por $\tilde{A}(z^{-1})$ de modo que o resto desta divisão seja fatorado como $z^{-j}F_j(z^{-1})$. O quociente desta divisão fornece o polinômio $E_j(z^{-1})$.

Se a equação (2.1) for multiplicada por $\Delta E_j(z^{-1})\, z^{j}$:

$$\tilde{A}(z^{-1})E_j(z^{-1})y(t+j) = E_j(z^{-1})B(z^{-1})\Delta u(t+j-d-1) + E_j(z^{-1})e(t+j) \tag{2.5}$$

Considerando (2.4), a equação (2.5) pode ser escrita como:

$$(1 - z^{-j}F_j(z^{-1}))y(t+j) = E_j(z^{-1})B(z^{-1})\Delta u(t+j-d-1) + E_j(z^{-1})e(t+j)$$

ou ainda:

$$y(t+j) = F_j(z^{-1})y(t) + E_j(z^{-1})B(z^{-1})\Delta u(t+j-d-1) + E_j(z^{-1})e(t+j) \tag{2.6}$$

Como o grau do polinômio $E_j(z^{-1})$ é igual a $j-1$ os termos com relação ao ruído da equação (2.6) estão todos no futuro. Portanto define-se que a predição ótima $y(t + j)$ é dada por:

$$y(t+j \mid t) = G_j(z^{-1})\Delta u(t+j-d-1) + F_j(z^{-1})y(t) \tag{2.7}$$

sendo:

$G_j(z^{-1}) = E_j(z^{-1})B(z^{-1})$

Com o intuito de minimizar a função de custo proposta pelo GPC (equação (2.3)) os sinais de controle a serem calculados $u(t),\ u(t+1),...,u(t+N)$ para atingir-se este objetivo.

Como o sistema considerado possui um tempo morto de *d* períodos de amostragem, a saída do sistema será influenciada pelo sinal *u(t)* após um período de amostragem *d+1*.

Considerando as predições ótimas *j* passos à frente:

$$\begin{aligned} &y(t+j+1\,|\,t) = G_{d+1}(z^{-1})\Delta u(t) + F_{d+1}y(t) \\ &y(t+j+2\,|\,t) = G_{d+2}(z^{-1})\Delta u(t) + F_{d+2}y(t) \\ &... \\ &y(t+j+N\,|\,t) = G_{d+N}(z^{-1})\Delta u(t) + F_{d+N}y(t) \end{aligned} \tag{2.8}$$

Que pode ser reescrito como:

$$y \;=\; Gu \;+\; F(z^{-1})y(t) \;+\; G'(z^{-1}) \;\Delta u(t\text{-}1) \tag{2.9}$$

Onde define-se o termo $\mathbf{F}(z^{-1})y(t) + \mathbf{G}'(z^{-1})\,\Delta u(t\text{-}1)$ como a resposta livre do sistema. Sendo:

$$\boldsymbol{y} = \begin{bmatrix} y(t+d+1\,|\,t) \\ y(t+d+1\,|\,t) \\ ... \\ y(t+d+N\,|\,t) \end{bmatrix},\; \boldsymbol{u} = \begin{bmatrix} \Delta u(t) \\ \Delta u(t+1) \\ ... \\ \Delta u(t+N-1) \end{bmatrix},\; \boldsymbol{G} = \begin{bmatrix} g_0 & 0 & ... & 0 \\ g_1 & g_0 & ... & 0 \\ ... & ... & ... & ... \\ g_{N-1} & g_{N-1} & ... & g_0 \end{bmatrix},$$

$$\mathbf{G}'(z^{-1}) = \begin{bmatrix} (G_{d+1}(z^{-1}) - g_0)z \\ (G_{d+2}(z^{-1}) - g_0 - g_1 z^{-1})z^2 \\ ... \\ (G_{d+N}(z^{-1}) - g_0 - g_1 z^{-1} - ... - g_{N-1} z^{-(N-1)})z^N \end{bmatrix},\; \mathbf{F}(z^{-1}) = \begin{bmatrix} F_{d+1}(z^{-1}) \\ F_{d+2}(z^{-1}) \\ ... \\ F_{d+N}(z^{-1}) \end{bmatrix}.$$

Agrupando os dois últimos termos da equação (2.9), os quais dependem apenas de amostras passadas:

$$y = Gu + f \tag{2.10}$$

Assim, a equação (2.3) pode ser reescrita como:

$$J = (Gu + f - w)^T (Gu + f - w) + \lambda u^T u \tag{2.11}$$

Sendo:

$$w = [\ w(t+d+1)\ w(t+d+2)\ ...\ w(t+d+N)\ ]^T$$

A equação 2.11 pode também ser reescrita como

$$J = \frac{1}{2}u^T Hu + b^T u + f_0 \tag{2.12}$$

sendo:

$$H = 2(G^T G + \lambda I)$$
$$b^T = 2(f - w)^T G$$
$$f_0 = (f - w)^T (f - w)$$

Assim, o valor mínimo da função de custo *J*, admitindo que não há restrição na variável de controle, pode ser calculada tornando o gradiente de *J* seja igual a zero, resultando em:

$$u = -H^{-1}b = (G^T G + \lambda I)^{-1} G^T (w - f) \tag{2.13}$$

Tendo em conta que apenas o primeiro elemento do vetor *u* é utilizado para o cálculo do sinal dos atuadores, tem-se que o sinal de controle é dado por:

$$\Delta u(t) = K(w - f) \tag{2.14}$$

sendo **K** é a primeira linha da matriz $(\mathbf{G^T G + \lambda I})^{-1}\mathbf{G}^T$.

Uma demonstração de como o GPC pode ser aplicado a um processo em malha fechada é mostrado na figura 2.2:

As linhas duplas na figura 2.2 indicam que o bloco envia um vetor ao sistema, enquanto as linhas simples indicam que o bloco envia uma única variável.

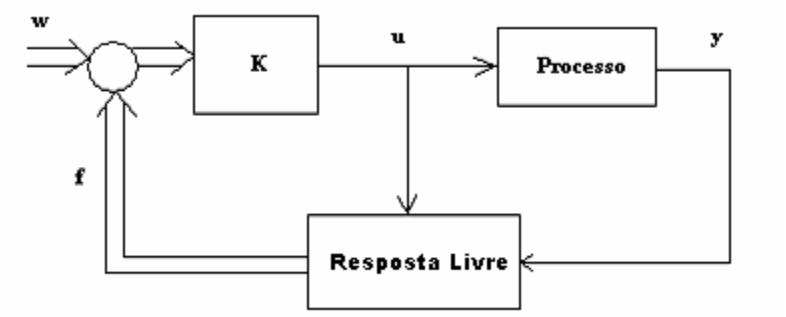

Figura 2.2 - Diagrama do Controlador GPC. A saída do bloco K é o sinal a ser aplicado no atuador.

2.3 CONTROLADOR PREDITIVO GENERALIZADO UTILIZANDO UM PREDITOR SMITH (SPGPC[12]).

Este algoritmo é muito similar ao GPC, no qual efetua-se a troca do preditor ótimo pelo preditor Smith na malha de controle. Além do diagrama mostrado na figura 2.2, o controlador GPC também pode ser descrito como uma estrutura formada por um preditor ótimo e um controlador clássico de dois graus de liberdade (2-dof, two degrees of freedom) [10]. Para demonstrar esta afirmação, utiliza-se o modelo CARIMA da planta, com tempo morto d, $N_1=d+1, N_2=d+N, N_u=N$ e fatores de esquecimento $\delta(j) = 1$ e $\lambda(j) = \lambda$. Assim, o modelo CARIMA, pode ser escrito como:

$$\begin{aligned} &y(t+d+j\,|\,t) = (1-a_1)y(t+d+j-1\,|\,t) + (a_1-a_2)y(t+d+j-2\,|\,t) + \ldots \\ &+(a_{na})y(t+d+j-n_a-1\,|\,t) + b_0\Delta u(t+j-1) + \ldots + b_{nb}\Delta u(t+j-1-n_b) \end{aligned} \tag{2.15}$$

Se esta equação for aplicada recursivamente para j = 1,2,...N tem-se:

$$\begin{bmatrix} y(t+d+1\,|\,t) \\ y(t+d+2\,|\,t) \\ y(t+d+3\,|\,t) \end{bmatrix} = G \begin{bmatrix} \Delta u(t) \\ \Delta u(t+1) \\ \ldots \\ \Delta u(t+N-1) \end{bmatrix} + H \begin{bmatrix} \Delta u(t-1) \\ \Delta u(t-2) \\ \ldots \\ \Delta u(t-nb) \end{bmatrix} + S \begin{bmatrix} y(t+d\,|\,t) \\ y(t+d-1\,|\,t) \\ y(t+d-na\,|\,t) \end{bmatrix} \tag{2.16}$$

Na qual $\boldsymbol{G}$, $\boldsymbol{H}$ e $\boldsymbol{S}$ são matrizes constantes de dimensão N x N, N x n_b e N x n_a + 1, respectivamente. Esta equação pode ser escrita na forma vetorial:

$$y = Gu + Hu' + Sy' = Gu + f \tag{2.17}$$

sendo $\mathbf{f} = \mathbf{Hu'} + \mathbf{Sy'}$ possui apenas termos no passado, correspondendo a resposta livre do sistema.

Se $\mathbf{y}$ for introduzido na função de custo, $J(N)$ será uma função de $\mathbf{y'}$, $\mathbf{u}$, $\mathbf{u'}$ e da seqüência de referência. Minimizando $J(N)$ em função de u:

$$M\begin{bmatrix}\Delta u(t)\\ \Delta u(t+1)\\ \dots\\ \Delta u(t+N-1)\end{bmatrix} = P_0\begin{bmatrix}y(t+d\,|\,t)\\ y(t+d-1\,|\,t)\\ y(t+d-na\,|\,t)\end{bmatrix} + P_2\begin{bmatrix}y(t+d+1\,|\,t)\\ y(t+d+2\,|\,t)\\ y(t+d+3\,|\,t)\end{bmatrix} + P_3\begin{bmatrix}\Delta u(t-1)\\ \Delta u(t-2)\\ \dots\\ \Delta u(t-nb)\end{bmatrix} \quad (2.18)$$

sendo que $\mathbf{M} = \mathbf{G}^T\mathbf{G}+\lambda\mathbf{I}$ e $\mathbf{R} = \mathbf{G}^T$ possuem dimensões N x N, $\mathbf{P_0} = -\mathbf{G}^T\mathbf{S}$ possui dimensão N x n_a + 1, e $\mathbf{P_1} = -\mathbf{G}^T\mathbf{H}$ possui dimensões N x n_b. Como em algoritmos com horizonte deslizante apenas o valor de $\Delta u(t)$ é calculado, e se **q** for a primeira linha da matriz $\mathbf{M}^{-1}$, $\Delta u(t)$ é dado por:

$$\Delta u(t) = P_0\begin{bmatrix}y(t+d+1\,|\,t)\\ y(t+d+2\,|\,t)\\ y(t+d+3\,|\,t)\end{bmatrix} + P_1\begin{bmatrix}\Delta u(t-1)\\ \Delta u(t-2)\\ \dots\\ \Delta u(t-nb)\end{bmatrix} + P_2\begin{bmatrix}\Delta u(t-1)\\ \Delta u(t-2)\\ \dots\\ \Delta u(t-nb)\end{bmatrix} \quad (2.19)$$

Portanto, o incremento de controle pode ser escrito como:

$$\Delta u(t) = qP0y' + qP1u' + qP2w$$

A equação acima pode ser mostrada no diagrama mostrado na figura 2.3:

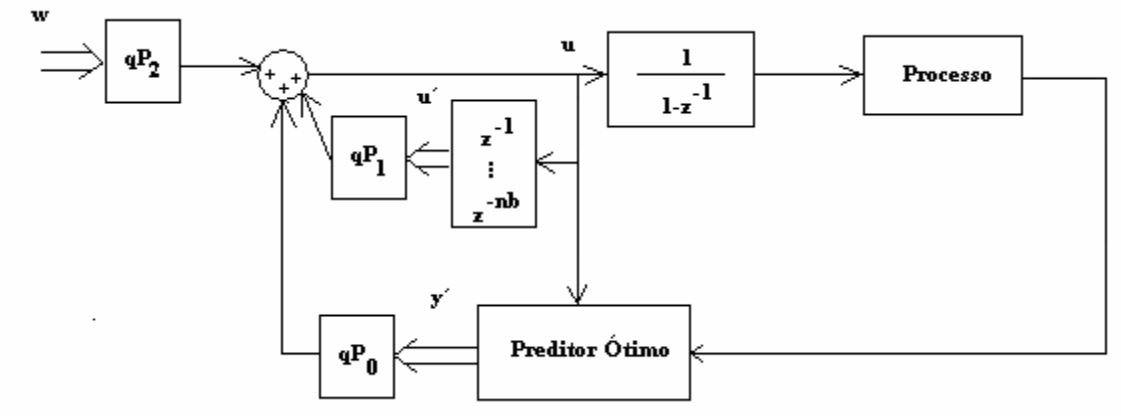

Figura 2.3 - Estrutura GPC equivalente a Fig. 2.2

Este diagrama pode ser facilmente simplificado, e adicionando o ruído de medida e o distúrbio planta inerente a modelagem de um controlador clássico 2-dof, tem-se que o modelo do GPC pode ser também descrito como o modelo mostrado na figura 2.4:

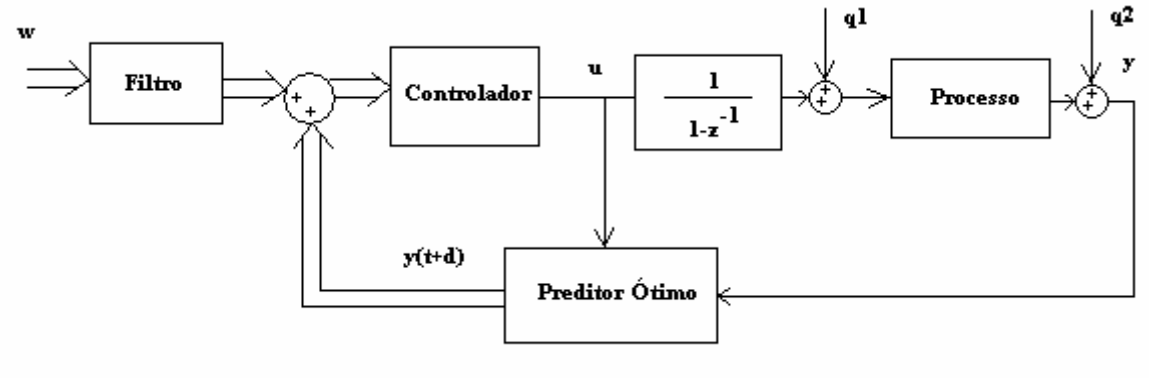

Figura 2.4 - GPC equivalente a um controlador 2-dof

Identificando o preditor ótimo separadamente neste novo modelo, pode-se melhorar a robustez do algoritmo GPC modificando seu preditor. Utilizando os polinômios *A* e *B* e o atraso d mostrados na equação (2.18) temos, para j=1:

$$\hat{y}(t+1|t) = (1-\tilde{A}(z^{-1}))zy(t) + \tilde{B}(z^{-1})u(t-d) \tag{2.20}$$

Utilizando o mesmo procedimento para $j = 2..., \hat{y}(t+1|t)$ têm-se:

$$\hat{y}(t+j|t) = (1-\tilde{A}(z^{-1}))z)^j\, y(t) + \sum_{i=1}^{j}(1-\tilde{A}(z^{-1}))^{i-1}\,\tilde{B}(z^{-1})u(t-d+j-1) \tag{2.21}$$

A saída prevista no instante *t+d* será:

$$\hat{y}(t+d|t) = (1-\tilde{A}(z^{-1}))z)^d\, y(t) + (1-(1-\tilde{A}(z^{-1}))^d)z^d P(z)u(t) \tag{2.22}$$

sendo $P(z) = \dfrac{B(z^{-1})z^{-1}}{A(z^{-1})}z^{-d}$ o modelo da planta.

Definindo $R(z) = (1-(1-\tilde{A}(z^{-1}))^d)z^d$ a predição pode ser escrita como:

$$\hat{y}(t+d|t) = R(z)y(t) + (z^d - R(z))P(z)u(t) \tag{2.23}$$

que resulta no diagrama mostrado na figura 2.5. Fazendo *R(z)=1* em (2.23), têm-se a equação do preditor Smith:

$$y(t+d|t) = y(t) + (1-z^{-d})\frac{B(z^{-1})z^{-1}}{A(z^{-1})}u(t) \tag{2.24}$$

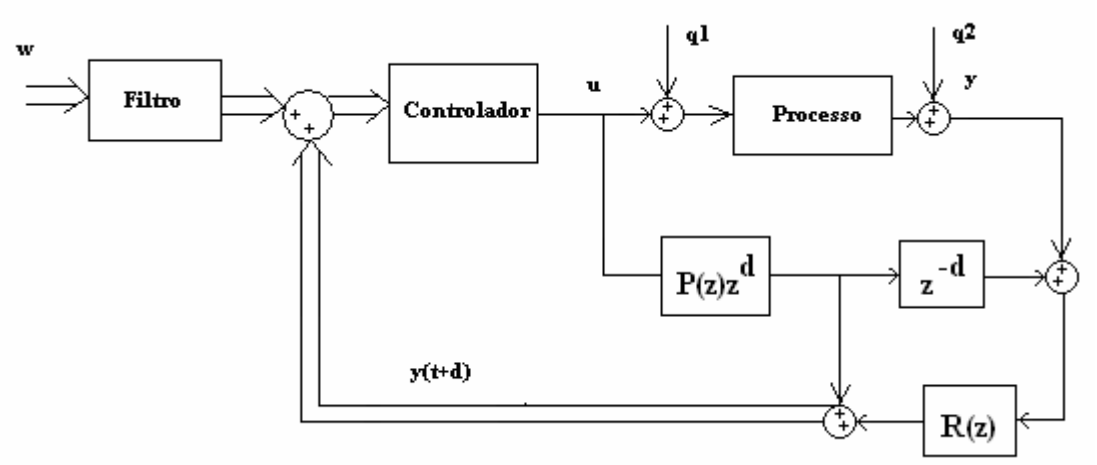

Figura 2.5 - Estrutura equivalente do GPC utilizando o Preditor Ótimo

Resultados comparativos entre GPC e GPC com preditor Smith mostram que este último possui melhor robustez e performance, se aplicados a plantas com estabilidade garantida [5] e se $|R(z)| > 1$ para o preditor ótimo. O polinômio *R(z)* ainda pode ser utilizado como parâmetro de medição de robustez comparativa entre o preditor Ótimo e o Smith[25]. O preditor Smith ainda pode ser utilizado na concepção de resposta livre e resposta forçada do sistema, bastando usar tal preditor para o cálculo da resposta livre até o tempo morto do sistema, a partir daí utilizando-se o preditor ótimo.

2.4 CONTROLADOR PREDITIVO GENERALIZADO COM ESTABILIDADE GARANTIDA(STABLE GENERALIZED PREDICTIVE CONTROL (SGPC))

A estratégia de controle do algoritmo GPC é de fácil entendimento e análise, como foi mostrado anteriormente. No entanto, em oposição a popularidade que este algoritmo adquiriu no decorrer dos anos, GPC possui a deficiência por não oferecer um resultado estável comprovado. Esta estabilidade é garantida somente em alguns casos especiais [6].

O algoritmo SGPC se baseia na estabilização da planta em malha fechada antes da aplicação da estratégia de controle, o que garante a estabilidade nominal do sistema. Além disso, o SGPC mostrou-se mais robusto no que tange estabilidade, além de ser mais eficiente computacionalmente, se comparado a outros algoritmos baseados no GPC baseados em

estabilidade garantida[7].O objetivo desta estratégia é substituir a sequência de ponderação infinita utilizada no GPC que relaciona a saída e as entradas de controle pelo sinal de referência simplesmente utilizando uma sequência de ponderação infinita. Esta simplificação advém da própria característica de estabilização do SGPC, a qual também nos fornece explicitamente o polinômio da função de transferência em malha fechada, além de uma quantificação dos erros através de resultados assintóticos. Tem-se, portanto, que o SGPC possui características que o torna extremamente robusto para o uso em processos adaptativos, pois todas as equações podem ser atualizadas recursivamente de maneira a se adaptar às mudanças nos parâmetros da planta. A simplicidade do SGPC nos fornece uma certa confiabilidade no que diz respeito às análises das margens de estabilidade robusta, bem como sua otimização.

Como mencionado anteriormente, o SGPC é uma variante do algoritmo GPC, em que este último somente inicia sua estratégia de controle após a planta em malha fechada ter sido estabilizada, aplicando a seguinte lei de controle:

$$Y(z)\Delta u_t = c_t - X(z)y_t \tag{2.25}$$

sendo os polinômios *X(z)* e *Y(z)* são definidos como:

$$a(z)\Delta(z)Y(z) + b(z)X(z) = 1 \tag{2.26}$$

Satisfazendo a identidade de Bezout. *a(z)* e *b(z)* são os coeficientes do denominador e do numerador, respectivamente, e c_t é o sinal de referência para o sistema de malha fechada.

O cálculo dos valores de saída futuros e os valores dos atuadores, também futuros, é dado por:

$$y_{\rightarrow} = \Gamma_b c_{\rightarrow} + H_b c_{\rightarrow} + M_b c_{\infty} \tag{2.27}$$

$$\Delta u_{\rightarrow} = \Gamma_A c_{\rightarrow} + H_A c_{\rightarrow} + M_A c_{\infty} \tag{2.28}$$

Sendo todos os elementos das equações acima definidos em [1].O vetor c_∞ é definido de tal modo que o valor da saída *y* em regime seja igual ao da referência *r*, sendo que para o sinal de referência r_{t+i}, onde i é maior que o horizonte de predição, o sinal de referência é constante e igual ao ganho do polinômio *b(z)*.

A estratégia de controle é realizada aplicando-se um pre-filtro à entrada de referência, e calculando-se polinômios de controle que se situam no atuador e na realimentação. O algoritmo utilizado para este cálculo é descrito abaixo [6]:

1. Foi resolvida a seguinte identidade de Bezout:

$$B(z)N(z) + A(z)M(z) = 1 \tag{2.29}$$

Em que : $A(z) = \Delta a(z), B(z) = \Delta b(z)$

2. Foram calculados os coeficientes do pré-filtro:

$$p_r^T = e^T[\Gamma_b^T\Gamma_b + \sigma\Gamma_A^T\Gamma_A]^{-1}[\Gamma_b^T - \Gamma_b^T M_b C_{\overline{X}} - \sigma\Gamma_A M_A C_{\overline{X}}] \tag{2.30}$$

Sendo σ o fator de ponderação e $C_{\overline{X}}$ definido em [6].

3. Foram calculados os coeficientes do polinômio SGPC em malha fechada através do vetor:

$$p_c = e^T[\Gamma_b^T\Gamma_b + \sigma\Gamma_A^T\Gamma_A]^{-1}[\Gamma_b^T H_B - \sigma\Gamma_A^T H_A] \tag{2.31}$$

4. Foram calculados os coeficientes de N´´(z) e M´´(z) resolvendo a seguinte identidade de Bezout:

$$b(z)[z^{-1}N''(z)] + A(z)M''(z) = 1 \tag{2.32}$$

5. Foi implementado o algoritmo seguindo o esquema abaixo:

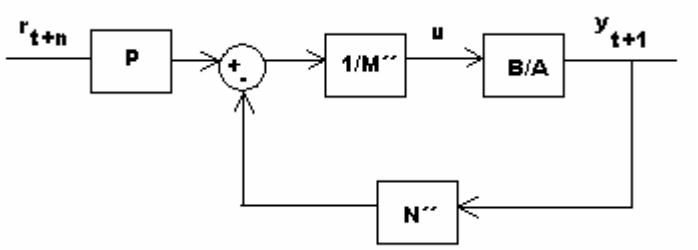

Figura 2.6 - Configuração do SGPC

2.5 CONSIDERAÇÕES FINAIS

Os algoritmos apresentados neste capítulo possuem vantagens e desvantagens condicionadas à aplicação. O GPC é o que apresenta menor esforço computacional dos três mencionados, porém não tem estabilidade garantida para processos com tempo morto elevado[8], necessitando assim de uma aplicação do SPGPC. O SGPC é o mais estável, possuindo na malha de referência um pré-filtro destinado a estabilizar a planta antes da aplicação do algoritmo de controle. Porém apresenta o esforço computacional mais elevado dos algoritmos aqui apresentados, mesmo sendo, entre os algoritmos com estabilidade garantida, o que possui menor esforço computacional [7].

A definição do algoritmo a ser utilizado dependerá da modelagem matemática do processo, descrita no capítulo seguinte, no qual será possível verificar se a planta possui pólos instáveis e se o tempo morto do processo é considerável.

CAPÍTULO 3

MODELAGEM MATEMÁTICA

3.1 INTRODUÇÃO

Um modelo matemático de um sistema real é um análogo matemático que representa algumas das características observadas em tal sistema. As características do sistema devidamente reproduzidas no modelo depende do objetivo para o qual o este está sendo desenvolvido. Apesar da possibilidade de desenvolver modelos com propriedades bastante diferentes, a característica mais importante a ser representada é a do modelo dinâmico, ou a evolução temporal do sistema.

Dois fatores são imprescindíveis ao modelador matemático. Entenda-se como modelador matemático o processo transcrição de um processo real em equações matemáticas. Em primeiro lugar, o modelo desenvolvido para um determinado sistema consiste apenas em uma representação aproximada. Conseqüentemente, não existe o modelo específico para o sistema, mas sim uma família de modelos com características e desempenhos variados. Em segundo lugar, o modelo é uma aproximação de apenas algumas características do sistema real. Portanto, devemos fazer considerações simplificadoras, de modo a obter apenas características mais importantes do sistema em determinada faixa de operação.

Em suma, não se pretende que um modelo, independentemente da área em estudo, seja uma cópia exata do mundo real, mas sim uma simplificação capaz de revelar os processos chave do fenômeno em causa, de forma a ser possível perceber e prever novas situações dentro do universo em estudo.

Um bom modelo induz a um melhor entendimento de uma determinada situação, mas deve-se atentar ao fato de que este deve ser robusto, isto é, um modelo que não contenha

situações fora dos limites da sua definição ou assentando em pressupostos que possam levar a resultados inconsistentes. O modelo deve ainda ser capaz de antecipar resultados que podem subseqüentemente ser verificados através de observações experimentais.

Uma consideração pertinente é a de se supor que o sistema sendo modelado comporta-se de forma aproximadamente linear. Tal suposição é normalmente verificada observando-se o comportamento de um sistema numa faixa relativamente estreita de operação. Formalmente diz-se que um sistema é linear se ele satisfaz o princípio da superposição. Este princípio baseia-se em, se dado um sistema excitado por uma entrada $u_1(t)$ produz uma saída $y_1(t)$ e quando excitado por uma saída $u_2(t)$ produzir uma saída $y_2(t)$, se o sistema for excitado por uma entrada $au_1(t) + bu_2(t)$ ele irá produzir uma saída $ay_1(t) + by_2(t)$, sendo a e b constantes reais. A consideração de linearidade normalmente simplifica bastante o modelo a ser desenvolvido.

3.1.1 MODELAGEM BASEADA NA FÍSICA DO PROCESSO

A modelagem física foi mais utilizada pelos controladores de processo até meados dos anos 70, pois era praticamente a única alternativa viável para se obter soluções para inúmeros problemas práticos. Este tipo de modelagem baseia-se no conhecimento prévio tanto das equações que regem os componentes do sistema quanto as equações características do sistema em si. Por exemplo, para saber qual o comportamento de um circuito eletrônico analógico, além do conhecimento das equações inerentes a cada componente deste(transistores, resistores, capacitores, indutores, amplificadores operacionais, etc) deve-se saber das equações que descrevem um circuito elétrico(lei das malhas, lei dos nós, etc). De posse de todas estas equações pode-se saber o comportamento deste circuito para uma determinada excitação.

Há também o problema da não-linearidade, característica da maioria dos sistemas reais. Como o trabalho com sistemas lineares possui uma teoria elaborada, com métodos de análise eficazes e simples [15], deve-se trabalhar em um sistema real de modo que a resposta do sistema, para determinados valores de entrada, resulte em valores de saída fazendo com que o sistema comporte-se linearmente. Este tipo de limitação não é incomum, atentando ao fato que geralmente se trabalha dentro de faixas de atuação do sistema, as quais sofrem um processo de linearização para que se enquadrem dentro do escopo de análise de sistemas lineares [15].

3.1.2 MODELAGEM BASEADA NA IDENTIFICAÇÃO DE SISTEMAS.

Em poucas situações práticas haverá tempo e conhecimento suficientes para desenvolver um modelo a partir das equações que descrevem a física do processo. Nesses casos, seria totalmente inviável desenvolver um modelo baseado nas leis que regem os fenômenos do sistema real.

A Identificação de Sistemas é um procedimento alternativo. Este procedimento se propõe a obter um modelo matemático que explique, pelo menos em parte e de forma aproximada, a relação causa e efeito presente nos dados. Ou seja, encontrar um modelo que, excitado por uma entrada *u(k)*, possua uma saída *y(k)*. Esta modelagem denomina-se modelagem caixa preta, uma vez que os componentes internos do sistema possuem dinâmica desconhecida.

Devido a complexidade do sistema proposto nesta pesquisa, que trabalha com plantas elétricas, mecânicas e seu acoplamento, a modelagem matemática a ser adotada será a da identificação de sistemas.

3.2 TIPOS DE RUÍDO

Quando se trata de sinais, pode-se definir ruído como flutuações advindas de fatores externos que afetam o a seqüência de informações transmitidas(sinal), de tal modo que esta pode ser descaracterizada de sua forma original e por conseguinte não interpretada corretamente pelo receptor. Em geral o ruído tem natureza aleatória, como, por exemplo, o ruído branco ou os ruídos coloridos, que são os tipos de ruídos tratados no presente trabalho.

3.2.1 RUÍDO BRANCO

Sinal ou processo puramente aleatório com uma certa densidade de potência espectral. O termo “ruído branco” é geralmente aplicado a um sinal ruidoso no domínio do tempo caracterizado por uma função de autocorrelação igual a zero em todo o espectro de freqüência. Deve ser notado que o adjetivo branco diz respeito as propriedades estocástcas do sinal. Em outras palavras, o fato do ruído ser branco indica que todas as freqüências do sinal são igualmente importantes. Por outro lado, a qualificação “branco” não informa às propriedades estatísticas do ruído, ou seja, a distribuição da amplitude. Portanto, para o ruído branco, a amplitude em um dado instante pode ter diversas distribuições como a Gaussiana ou a distribuição uniforme.

3.2.2 RUÍDO COLORIDO

É um sinal com parte aleatória (branca) e parte determinística. Esse tipo de ruído pode ser modelado como um processo auto-regressivo ou de média móvel excitado por ruído branco. O espectro do ruído colorido não tem potência em todas as freqüências, sendo que a maior parte da potência espectral está concentrada em faixas relativamente estreitas de freqüência, daí o adjetivo “colorido”.

3.3 IDENTIFICAÇÃO DE SISTEMAS UTILIZANDO ALGORITMO DOS MÍNIMOS QUADRADOS ESTENDIDO

Será apresentada a seguir uma explanação de dois algoritmos de identificação de sistemas: Algoritmo dos Mínimos Quadrados e Algoritmo dos Mínimos Quadrados Estendido.

3.3.1 ALGORITMO DOS MÍNIMOS QUADRADOS

A origem da idéia básica deste algoritmo pode ser encontrada nos trabalhos de Gauss sobre estudos astronômicos. Este trabalho menciona que se as observações astronômicas e outras grandezas, nas quais se baseiam o cálculo de órbitas, fossem absolutamente corretas, o resultado, ainda que obtido a partir de apenas três ou quatro observações, também seria estritamente correto e, portanto, o uso de outras medições seria útil apenas para confirmação, mas desnecessárias para correção ou ajuste. Novamente, todas as medições e observações nada mais são que aproximação da realidade, o mesmo é verdade para os resultados que nela se baseiam, e o alvo mais elevado de todos os cálculos feitos relacionados a fenômenos concretos deve ser o de se aproximar, tanto quanto possível, da verdade. Isso, entretanto, não pode ser alcançado a não ser pela adequada combinação de observações adicionais àquelas estritamente necessárias à determinação das grandezas desconhecidas. Esse problema só pode ser resolvido apropriadamente quando um conhecimento apropriado da órbita já foi obtido, conhecimento esse que posteriormente deve ser ajustado de forma a satisfazer todas as observações da forma mais exata possível.[13].

No texto acima fica claro que, por causa dos erros de medição, mais observações do que o número mínimo são necessárias. Por isso, o uso dessas observações, supostamente redundantes, resultará na redução dos efeitos dos erros. Então, faz-se necessário um número

de observações estritamente necessário para a determinação das grandezas desconhecidas, geralmente igual ao número de parâmetros do modelo a ser identificado.

Supondo-se que uma determinada planta seja descrita pelo seguinte modelo:

$$y = \sum_{j=1}^{p} x_j \theta_j = x^T \theta \tag{3.1}$$

Em que $\boldsymbol{x^T}$ = [x_1 x_2 ... x_P] é um vetor de variáveis controladas e $\boldsymbol{\theta}$ = [θ_1 θ_2...θp]^T um vetor de parâmetros.

Se θ for desconhecido, pode ser obtido a partir da observação de y para p valores distintos de x e da resolução do sistema de equações resultante. O valor de θ obtido por este método seria estritamente preciso se, por um lado, as observações de y e os valores de **x** utilizados fossem absolutamente corretos, e se, por outro lado, o fenômeno pudesse ser exatamente descrito pela equação (3.1). Sabe-se, como mencionado anteriormente, que no mundo real, tanto as observações, quanto os modelos matemáticos não são mais o que aproximações da verdade e, por isso, o método descrito nunca poderia conduzir a valores exatos para os parâmetros.

Um modelo matemático mais realista para o fenômeno considerado será então:

$$y = x^T \theta + \varepsilon_m \tag{3.2}$$

Em que ε_m é uma variável aleatória que representa o erro de modelagem.

Qualquer observação y poderia ser descrita da seguinte forma:

$$y = x^T \theta + \varepsilon_m + \varepsilon_0 \tag{3.3}$$

Sendo ε_0, uma variável aleatória que representa os erros de observação (de y e **x**).

Será definido um novo vetor de variáveis aleatórias:

$$\varepsilon = \varepsilon_m + \varepsilon_0 \tag{3.4}$$

Representando os erros de modelagem e de observação, pode-se rescrever (3.3) como:

$$y = x^T\theta + \varepsilon \tag{3.5}$$

A natureza aleatória de ε não permite um cálculo exato de θ baseado em observações de y.

No entanto, pode-se supor que o conjunto de N observações $y_1, \ldots y_i, \ldots, y_N$ tenta dar uma informação sobre θ, e que os erros $\varepsilon_1, \ldots, \varepsilon_i, \ldots, \varepsilon_N$ possuem tendência a não ser significativamente grandes. Neste contexto, os parâmetros podem ser calculados, ou estimados minimizando a seguinte função:

$$S = \sum_{i=1}^{N} y_i - x_i^T\theta \tag{3.6}$$

Isto é, procurando minimizar a soma dos quadrados dos erros.

Em notação matricial ter-se-á:

$$S = (Y - X\Theta)^T (Y - X\Theta) \tag{3.7}$$

Sendo:

$$Y = [\, y_1, \, \&, y_N \,]^T$$

$$\mathrm{X}=\begin{bmatrix} x_{11} & x_{12} & \ldots & & x_{1P} \\ x_{21} & x_{22} & \ldots & & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ x_{N1} & \ldots & \ldots & \ldots & \ldots x_{NP} \end{bmatrix} = \begin{bmatrix} x_1^T \\ x_2^T \\ \ldots \\ x_N^T \end{bmatrix}$$

Tendo assim:

$$Y = X\Theta + E \tag{3.8}$$

Sendo:

$\mathbf{E} = [\, \varepsilon_1, \ldots, \varepsilon_N \,]^{\mathrm{T}}$

Derivando S em relação a θ:

$$\frac{dS}{d\Theta} = -2X^T Y + 2X^T X\Theta \tag{3.9}$$

Fazendo $\Theta = 0$:

$$\frac{dS}{d\Theta} = 0 \Rightarrow X^T X \hat{\Theta} = X^T Y \tag{3.10}$$

Se $X^T X$ puder ser invertida, obtém-se uma estimativa única:

$$\hat{\Theta} = (X^T X)^{-1} X^T Y \tag{3.11}$$

Note-se que $N >= P$ é a condição necessária para que $\mathbf{X^T X}$ deve poder ser invertida.

Estes resultados apresentados foram desenvolvidos para um modelo geral do tipo y = f(x,teta). Os resultados obtidos têm caráter geral e aplicam-se também ao caso de modelos algébricos. Será então apresentada uma formulação dos Mínimos Quadrados para modelos ARMAX, modelo no qual será utilizado neste trabalho.

Na maioria dos problemas de identificação de sistemas, as restrições que vão gerar a equação matricial a ser resolvida por mínimos quadrados serão tomadas a partir de sinais temporais, ou seja, de seqüências de números. Nesse caso pode-se representar o modelo como:

$$y(k \mid \hat{\Theta}) = \psi^T (k\text{-}1)\hat{\Theta} + \varepsilon(k) \tag{3.12}$$

Sendo que k indica o instante considerado $n_\Theta = \dim[\hat{\Theta}]$ um vetor de variáveis regressoras, a saber:

$$\psi(k-1) = [\psi_1, \psi_2, ..., \psi_{n_\Theta}]^T \tag{3.13}$$

Tomadas até o instante $k - 1$. A única diferença entre a equação (3.12) e a (3.5) é que, na primeira, faz-se alusão a um sistema dinâmico.

Portanto, o modelo dinâmico do tipo (3.12) tomado sobre uma massa de dados, gera restrições que podem ser representadas por uma equação matricial:

$$y = \Psi \hat{\Theta} + \varepsilon \tag{3.14}$$

Assim, os resultados obtidos no estudo anterior com modelos gerais podem ser aplicados a equação matricial (3.14). Com isso, a função de custo para modelos ARX pode ser expressa como:

$$J_{MQ} = \sum_{k=1}^{N} \varepsilon(k \mid k-1, \hat{\Theta})^2 = \varepsilon^T \varepsilon = \| \varepsilon \|^2 \tag{3.15}$$

Sendo, que é o erro de predição (ou resíduo) cometido no instante k, ao fazer a predição com $\varepsilon(k \mid k-1, \hat{\Theta})$ informação apenas até o instante k – 1, usando o vetor estimado. Assim, o estimador dos mínimos quadrados pode ser representado como:

$$\hat{\Theta}_{MQ} = [\Psi^T \Psi]^{-1} \Psi^T y \tag{3.16}$$

ou ainda:

$$\theta_{MQ} = \left[\frac{1}{N} \sum_{k=1}^{N} \psi(k-1) \psi^T(k-1) \right]^{-1} \left[\frac{1}{N} \sum_{k=1}^{N} \psi(k-1) y(k) \right] \tag{3.17}$$

3.3.2 ALGORITMO DOS MÍNIMOS QUADRADOS ESTENDIDO

Há situações em que o estimador dos mínimos quadrados é polarizado. Entende-se por polarização o desvio ou erro do valor do parâmetro calculado com relação ao parâmetro real. Isso ocorre em algumas situações nas quais o ruído ou erro na equação de regressores (parâmetros medidos da planta, tanto de entrada como saída) for autocorrelacionado [14].

Basicamente existem três condições a serem satisfeitas para evitar polarização [14]:

1. $[\Psi^T \Psi]^{-1} \Psi^T \Psi = I$
2. $[\Psi^T \Psi]^{-1} \Psi^T \Psi$ e $\boldsymbol{\varepsilon}$ devem ser estatisticamente independentes, ou não-correlacionados.
3. A média de $\boldsymbol{\varepsilon}$ deve ser nula, ou seja, $\boldsymbol{\varepsilon}$ deve ser um ruído branco.

As três condições devem ser satisfeitas simultaneamente, o que não ocorrerá caso **ε** seja auto-correlacionado, ou seja, colorido. Consequentemente, uma forma de evitar a polarização é transformar a equação matricial (3.14) em:

$$y^* = \Psi^* \hat{\Theta}^* + \varepsilon^* \tag{3.18}$$

Sendo **ε*** um ruído branco de forma que as condições 1 e 2 supracitadas sejam satisfeitas para uma matriz de regressores Ψ^*. Este resultado é obtido adicionando os parâmetros do ruído colorido no vetor de regressores, de tal modo que a matriz de regressores inclua, além dos parâmetros de entrada e saída os parâmetros do ruído autocorrelacionado.

3.4 CONSIDERAÇÕES FINAIS

Como a grande maioria dos modelos matemáticos depende de certos parâmetros, não totalmente determinados pelas características físicas teóricas dos processos, utiliza-se a técnica dos mínimos quadrados como estratégia de estimação desses parâmetros, pela sua facilidade de implementação e intuitividade do algoritmo. Na estimação de parâmetros em modelos lineares, será utilizada a técnica dos mínimos quadrados que define uma regressão normal, e em existindo erros de observação ou experimentação nos dados do problema ou na variável resposta, que é fato na maioria dos processos reais, faz-se uma pequena modificação deste algoritmo, de modo a se modelar o ruído dentro dos parâmetros calculados pelos mínimos quadrados.

CAPÍTULO 4

VISÃO GERAL DO SISTEMA

4.1 INTRODUÇÃO

Após a escolha dos algoritmos a serem utilizados no sistema, tanto de controle como de identificação, necessita-se da definição e parâmetros dos componentes reais no qual esse sistema irá utilizar. Neste capítulo são descritos: os tipos de atuadores e seus circuitos acionadores, de modo a atender a especificação de vazão de líquido necessária para efetuar-se o controle do processo e atender os requisitos elétricos necessários para o acionamento destes atuadores; o tipo de microcontrolador a ser utilizado, de modo a encontrar uma relação custo-benefício satisfatória, onde não haja oneração do sistema mas também possua performance satisfatória para o processo; métodos de aquisição e análise de dados, onde se possa fazer uma análise qualitativa do processo e uma comparação entre os algoritmos. Nesta fase ocorre a definição do software a ser utilizado, que atenda os pré-requisitos de possibilidade de implementação dos algoritmos, com funções matemáticas que facilitem o desenvolvimento das equações inerentes a estes, ferramentas gráficas em que se possa analisar o comportamento do processo "*on-line*" e simulado e análise da circuitaria necessária para o sistema sem a necessidade de análise demorada em bancada. Estas são apenas algumas das muitas especificações para ter-se uma análise clara de um sistema real.

Neste capítulo será definida toda a estrutura de hardware e software utilizada neste projeto.

4.2 ESTRUTURA GERAL DO SISTEMA

O diagrama de blocos do sistema é mostrado na Figura 4.1:

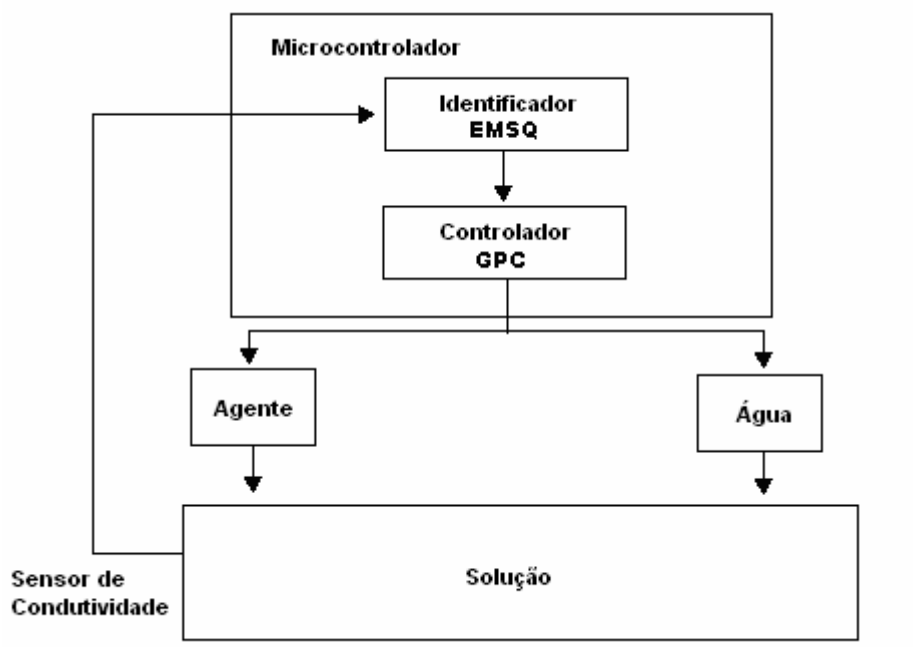

Figura 4.1 – Diagrama de blocos do sistema de controle de condutividade de água.

Como apresentado na Figura 4.1, percebe-se que esta é uma planta MISO, onde as duas entradas são os atuadores da água e do agente e a saída é a condutividade fornecida pelo sensor. Ambos os algoritmos, de identificação e de controle, encontram-se na memória do microcontrolador, sendo executados sequencialmente, primeiro o identificador, e logo após o controlador. Os dados de saída do microcontrolador, que se referem aos valores calculados pelo algoritmo de controle, são enviados ao canal PWM(sinais modulados por largura de pulso, ou pulse width modulation), que por sua vez são remetidos a placa de acionamento dos motores de corrente contínua(i.e. os atuadores), onde ocorre o bombeamento da água e do agente de molha para o reservatório com o objetivo de controlar a condutividade desejada. Fechando a malha de controle, um sensor de condutividade com sensibilidade variando de 0 à 5000μS(microSiemens) que possui uma saída padrão de 4 a 20mA diretamente proporcional ao valor da condutividade. Esta saída é ligada a um resistor de 10kOhms , como mostrado na figura 4.4, no qual a queda de tensão é utilizada pelo canal analógico-digital do microcontrolador(A/D) para que este valor seja lido como uma variável, utilizado para o

cálculo dos parâmetros em ambos os algoritmos. Esse valor é diretamente proporcional à queda de tensão no resistor acoplado a saída do sensor.

4.3 EQUIPAMENTOS UTILIZADOS NO PROJETO

Nesta seção são descritos os equipamentos que fizeram parte do projeto, a saber: microcontrolador, fonte de alimentação, sensor de condutividade, tipo de atuadores e circuito de acionamento dos atuadores.

4.3.1 MICROCONTROLADOR PIC 18F452

Os microcontroladores PIC da Microchip®[16] são bem aceitos no mercado principalmente por suas características de arquitetura, alta desempenho e por seu reduzido conjunto de instruções (RISC).

O microcontrolador 18F452 foi escolhido por possuir internamente à sua estrutura conversores analógico-digitais, permitindo a leitura do valor de condutividade a ser monitorado.

Algumas das principais características do PIC 18F452 são:

- Módulo serial compatível com RS-232(USART).
- Dois canais PWM de 10 bits.
- Conversor A/D de 10 bits.
- “Watchdog timer” e detector de tensão de “Brown-out”.
- 10 milhões de instruções por segundo(MIPS) com um cristal de 40MHz.
- Memória “Flash” interna de 32Kbytes, RAM de 1536 bytes.

A função do 18F452 será a de armazenar e processar os algoritmos de controle e identificação, fato que justifica a escolha deste microcontrolador por sua grande memória de armazenamento (32Kbytes). O tamanho das variáveis também é grande, por se utilizar

matrizes de ponto flutuante(float), justificando a grande capacidade da memória RAM residente(1536 bytes). O mapa da memória utilizada neste trabalho é mostrado na Figura 4.2. O conversor A/D de 10 bits fornece o grau de precisão necessário para a leitura da condutividade pelo microcontrolador, valor este a ser utilizado por ambos algoritmos. Os canais PWM de 10 bits fornecem a resolução necessária para se fornecer a precisão necessária ao circuito dos atuadores, que foram utilizados para acionar os motores de corrente contínua. O fato de este microcontrolador possuir duas saídas PWM foi condição *sine qua non* para sua escolha, uma vez que utilizamos dois atuadores no processo, cada um utilizando um canal PWM.

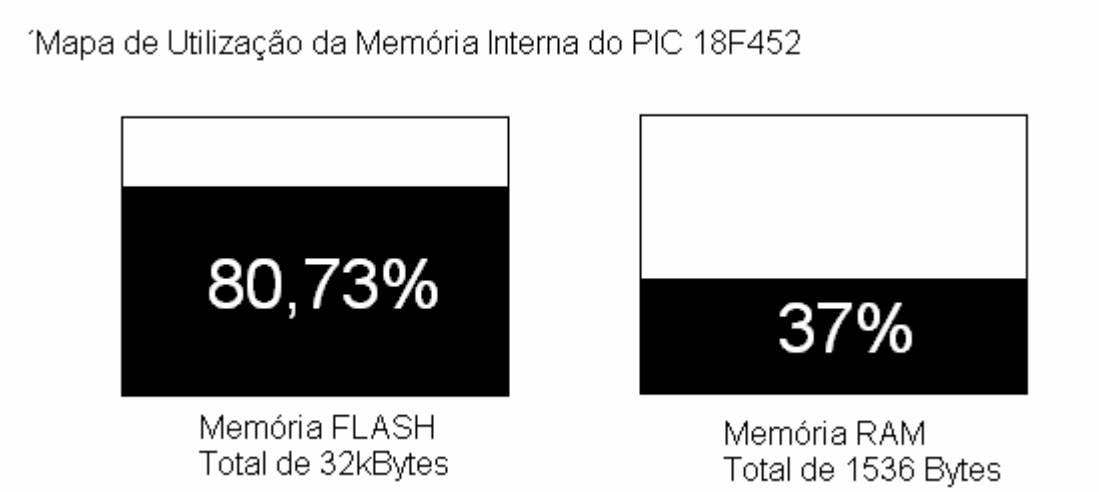

Figura 4.2 – Mapa da Memória Utilizada pelo Microcontrolador PIC 18F452 no presente trabalho.

O módulo serial, configurado a uma taxa de transmissão de 9600 Bauds, efetuou a transmissão dos dados do sensor para o PC, do mesmo modo que enviava os valores calculados a serem aplicados a saídas PWM.

4.3.2 SENSOR DE CONDUTIVIDADE RT-10

O sensor de condutividade RT-10 da Moranlord Inc.® é mostrado na figura 4.3:

Figura 4.3 – Sensor de condutividade utilizado no projeto.

Este sensor fornece uma saída que varia de 4-20mA, padrão muito utilizado tanto nos EUA, Europa e Brasil, corrente esta que varia proporcionalmente dentro dos limites de condutividade do aparelho, variando de 0 a 5000μS. Este equipamento utiliza uma fonte de alimentação AC de 24V.

A saída do sensor foi diretamente acoplada a um resistor de 220 Ohms. O circuito de acoplamento e a transformação linear dos dados obtidos são mostrados conforme Figura 4.4:

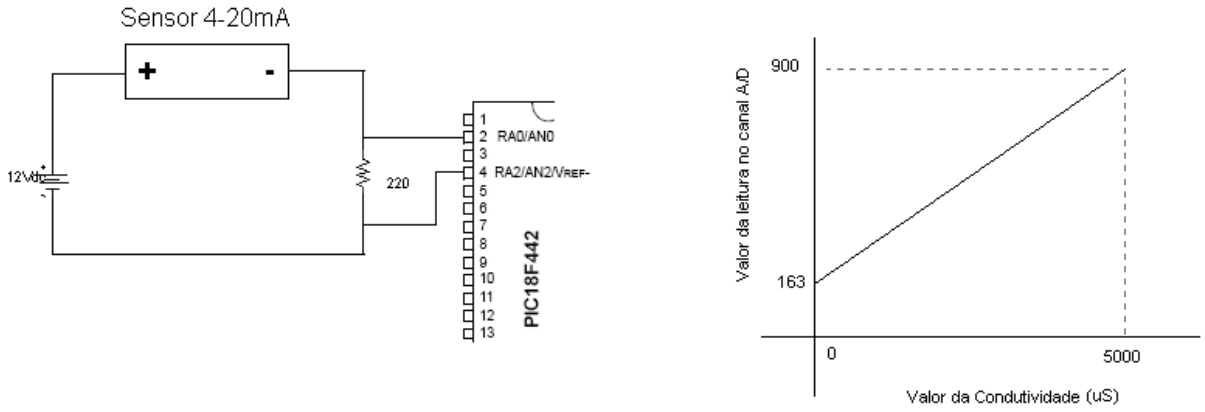

Figura 4.4 – Circuito de acoplamento do sensor com o microcontrolador e a função de transformação de dados.

Quando o sensor faz a leitura do valor de condutividade mínima(0 uS), sua saída fornece 4mA de corrente. Isso resulta em uma queda de tensão no resistor de 220Ohms de 0,88V. Esse valor é lido dentro do microcontrolador, para uma resolução de 10 bits, como 163. Quando a condutividade máxima é atingida(5000 uS), o sensor fornece uma corrente de 20mA, resultando uma queda de tensão de 4,4V, valor este que é lido pelo microcontrolador

como 900. Todos os valores entre o mínimo e o máximo podem ser encontrados através de uma função linear, uma vez que a condutividade neste processo possui esta característica.

4.3.3 ATUADORES

Para o bombeamento dos líquidos foram utilizadas bombas de sucção de água, com vazão de 4,5 litros/minuto a plena carga. Estes são motores de corrente contínua, alimentados à uma tensão de 12V, que quando em pleno acionamento utilizam uma corrente de 1,6A.

4.3.4 CIRCUITO DE ACIONAMENTO DOS ATUADORES

Utilizou-se um circuito bastante conhecido para acionamento de MOSFETs[17], os quais são utilizados para acionar diretamente os atuadores. O circuito é mostrado na Figura 4.5.

O funcionamento do circuito ocorre da seguinte maneira: o transistor Q1 é fechado e o diodo D1 se polariza diretamente e mantém Q2 bloqueado, chaveando assim o MOSFET de canal p. A resistência R7 serve para descarregar o capacitor Ciss intríseca do MOSFET, e diminuir o tempo de chaveamento.

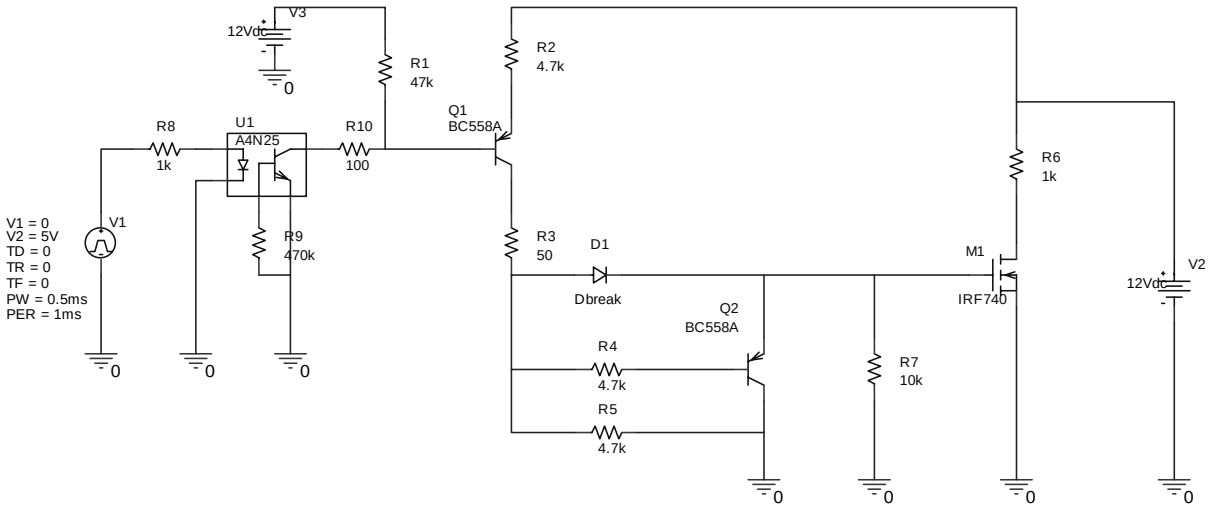

Figura 4.5 – Circuito de acionamento dos Motores de Corrente Contínua utilizados para bombeamento dos líquidos

Como o atuador trabalha com tensão de 12V, antes da implementação fizemos a simulação no software Orcad®, obtendo o resultado mostrado na Figura 4.6. O gráfico superior representa a forma de onda PWM enviada pelo microcontrolador, e a parte inferior representa a tensão na porta(gate) do MOSFET, que será utilizada para acionar os atuadores. A freqüência do PWM utilizado neste projeto foi de 1,2 kHz , valor este configurado internamente no microcontrolador.

4.3.5 FONTE ESTABILIZADA

Utilizou-se uma fonte estabilizada HY1300HOBBY de 13,8Vdc da Hayama®, com saída máxima de 10A, para alimentação do circuito de acionamento dos atuadores. Uma vez que o consumo total do circuito gira em torno de 3,5A (3,2A dos atuadores e 300mA do circuito lógico), essa fonte supriu as necessidades requeridas para este projeto.

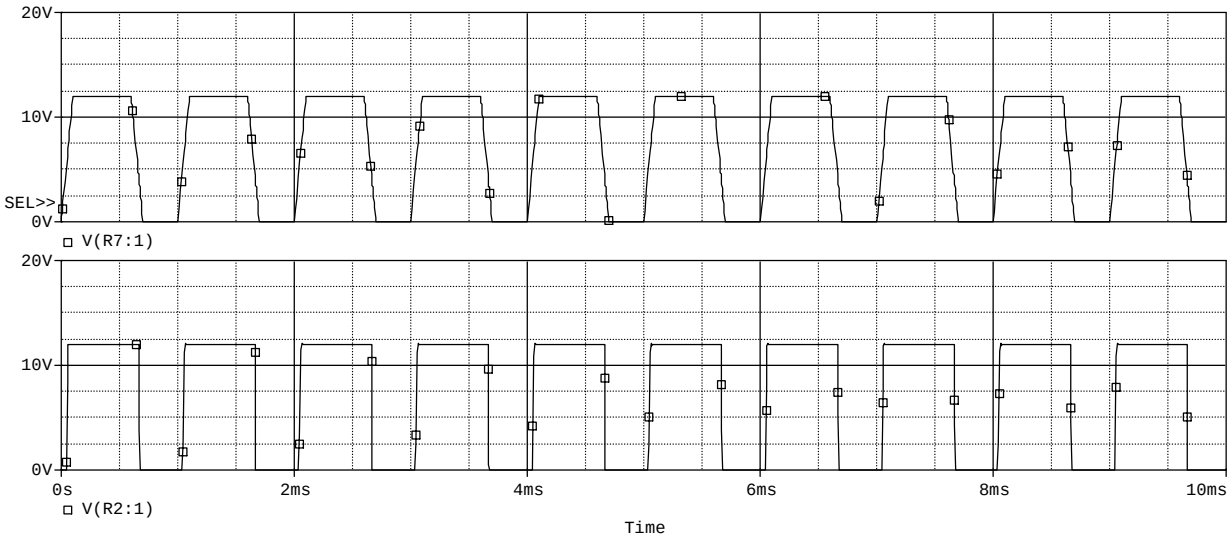

Figura 4.6 – Simulação do circuito de acionamento de MOSFETs mostrado na Fig 4.5.

4.4 SOFTWARE UTILIZADO NO SISTEMA

Utilizando-se a ferramenta matemática Matlab® e o Simulink® foi desenvolvido um sistema capaz de fornecer uma análise dos dados referentes aos parâmetros do processo em tempo real, tanto identificação como controle. Isso foi conseguido utilizando-se *s-functions*,

blocos especiais do Simulink os quais tem a capacidade de armazenar códigos de programa utilizando a linguagem do próprio Matlab.

Foi desenvolvida uma *s-function* com a função de enviar dados ao microcontrolador, o qual fazia o tratamento destes para verificação de qual canal PWM deveria ser acionado com o valor captado. Esta mesma *s-function* efetuava o recebimento dados captados pelo canal A/D do integrado, contudo somente quando havia requisição deste tipo de dado pelo software contido na *s-function*. Deste modo, foi desenvolvida a interface com a porta serial a qual fazia a comunicação entre o microcontrolador e o Simulink. O firmware gravado no microcontrolador tinha o algoritmo semelhante ao da s-function, de modo a emular o sistema real.

Para cada controlador preditivo também foi desenvolvida uma *s-function*, bem como para o algoritmo dos Mínimos Quadrados Estendido. Estes blocos foram conectados de modo a trabalhar na planta, em que a *interface* entre o processo real e o Simulink foi realizada pela *s-function* descrita no parágrafo anterior. Assim conseguiu-se avaliar os algoritmos de identificação e controle com dados reais em um processo "*online*", com as facilidades oferecidas pelas funções matemáticas do Matlab. O controle portanto foi feito *a priori* em computador, tendo o microcontrolador a única função de terminal, fazendo a interface entre os algoritmos e o sistema real. A utilização do Simulink deveu-se ao fato de este possuir uma interface de fácil manutenção e uso, além da facilidade em obter-se gráficos e em desenvolver-se algoritmos utilizando esta ferramenta. *A posteriori*, ambos os algoritmos foram alocados no microcontrolador, eliminando a necessidade da ferramenta matemática, e por conseqüência do computador, para o controle do sistema.

4.4.1 IDENTIFICAÇÃO EM TEMPO REAL DO SISTEMA UTILIZANDO O ALGORITMO DOS MÍNIMOS QUADRADOS ESTENDIDO VIA S-FUNCTIONS.

Para aumentar a robustez do sistema, os parâmetros da planta foram identificados em uma simulação em separado, uma vez que o algoritmo de identificação não possui os parâmetros reais da planta antes do início do processo, podendo levar o sistema a instabilidade[18]. Utilizando-se do sistema descrito na seção 4.4, construímos o diagrama de bloco utilizando o Simulink mostrado na Figura 4.7.

O bloco identificado como “AD e DA” é uma *s-function* que faz a comunicação serial com o microcontrolador. As duas entradas deste bloco se referem diretamente aos dois canais PWM que o microcontrolador possui, fazendo com que o valor recebido em cada entrada seja enviada para o seu canal respectivo. O bloco “State Generator” gera espaços de estados utilizados pelo algoritmo de identificação. São gerados variáveis de estados para as duas entradas e para a saída. O bloco “RLS Estimate” é o algoritmo Estendido dos Mínimos Quadrados em si. Este bloco é uma *s-function* que faz a identificação dos parâmetros. A *s-function* “Param” faz o tratamento dos dados calculados por “RLS Estimate” e efetua a impressão dos parâmetros do sistema nos blocos “modelagemagente” e “modelagemagua”. Estes blocos fazem referência direta a modelagem do processo, separada como dito anteriormente em duas plantas SISO, ambas perfazendo então o caso MISO descrito no item 4.2. O bloco “Relógio de Tempo Real” é uma *s-function* que faz referência a uma biblioteca dinâmica do Windows® de mesmo nome, ajustando o Simulink para que o mesmo trabalhe em tempo real.

4.4.2 CONTROLE E IDENTIFICAÇÃO DO SISTEMA EM TEMPO REAL UTILIZANDO S-FUNCTIONS.

Com os parâmetros identificados pelo primeiro processo, foi implementado o algoritmo de controle e o de identificação trabalhando ao mesmo tempo, numa alusão ao diagrama da Figura 4.1, a qual a única modificação foi a mudança do local do processamento, passando do microcontrolador para o PC. Primeiramente, a partir dos parâmetros calculados, efetuou-se a simulação do sistema utilizando os controladores preditivos dentro do Simulink, numa alusão ao processo real que seria controlado.

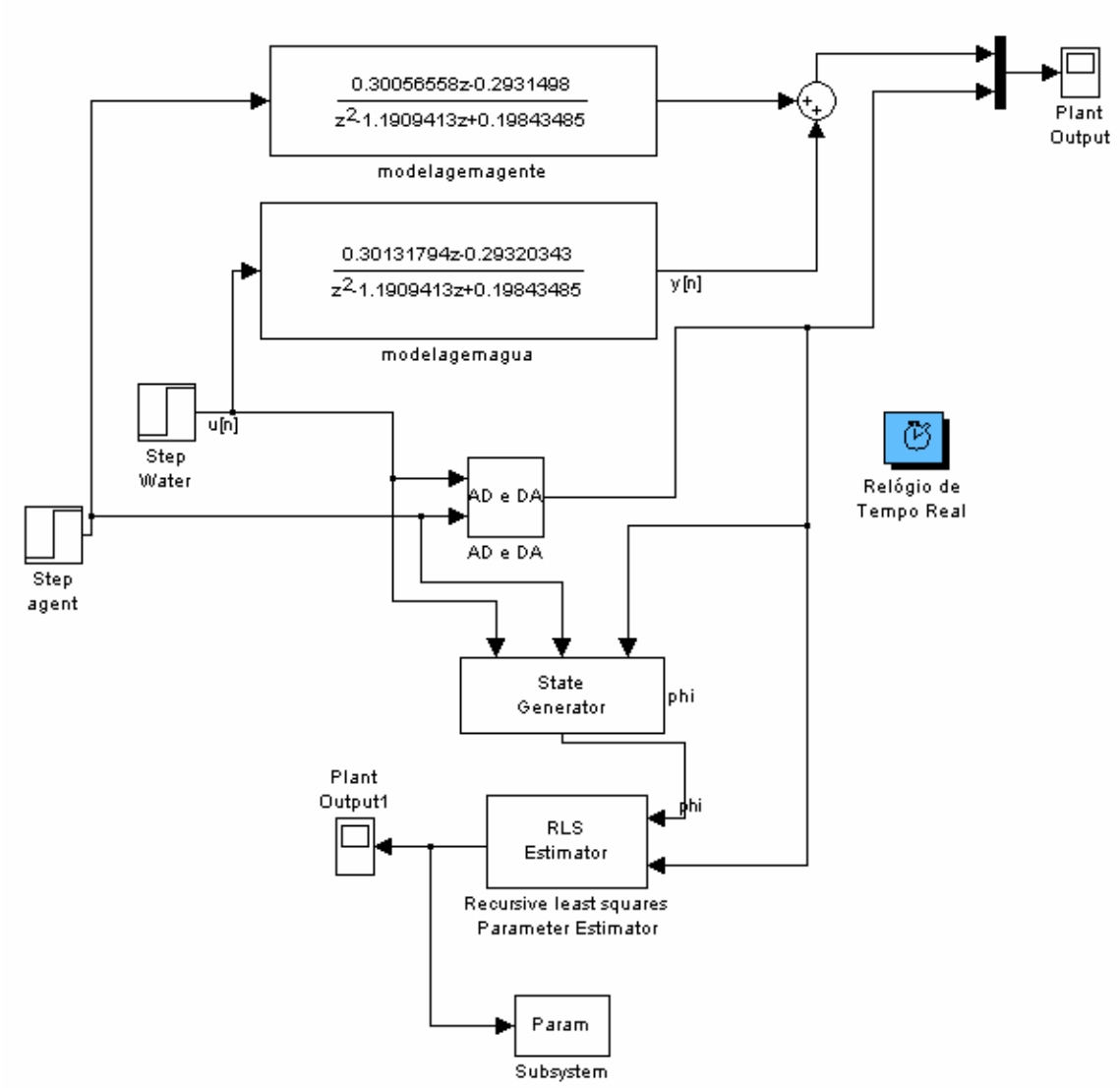

Figura 4.7 – Diagrama de blocos no Simulink do sistema de identificação dos parâmetros do processo

Os diagramas de blocos deste sistema é mostrado nas figuras 4.8, 4.12 e 4.16 para o GPC, SPGPC e SGPC, respectivamente. Em seqüência aplicou-se à planta real os mesmos algoritmos, utilizando a interface desenvolvida em *s-function* para enviar e receber dados do microcontrolador. Os diagramas deste sistema são mostrados nas figuras 4.9, 4.13 e 4.17 para o GPC, SPGPC e SGPC, respectivamente.

Os blocos “Planta Água” e “Planta Agente” são as funções de transferência do sistema, com os parâmetros calculados do mesmo modo descrito no ítem 4.4.1. O bloco “White Noise1”, um ruído branco, é filtrado por uma função de transferência “Plant 2”, gerando um ruído colorido utilizado na simulação para representar os possíveis distúrbios que a planta poderia sofrer no processo real. Os blocos “State Generator”, “RLS Estimate”, “Param”, “modelagemagente” e “modelagemagua”, são os mesmos que foram mencionados no ítem 4.1.1. Como se trata de um sistema MISO dividido em dois sistemas SISO, foi necessário um controlador para cada sistema. Os blocos “predictive Controller agua” e “free Response agua” correspondem ao preditor ótimo e a resposta livre aplicados ao processo correspondente a água, respectivamente, enquanto os blocos “predictive Controller agente” e “free Response agente” correspondem ao preditor ótimo e a resposta livre aplicados ao processo correspodente ao agente, de modo similar ao processo correspondente a água. Os blocos “Agua Gain” e “Agente Gain” parametrizam a saída de modo a dividir o esforço dos atuadores em ambas as plantas. Se, por exemplo, o bloco “Água Gain” possuir o valor associado 0,3, e o “Agente Gain” possuir o mesmo valor em 0,7, significa dizer que a da água deverá realizar um esforço tal a elevar a condutividade para 30% do valor final requerido, enquanto a planta do agente terá um esforço maior, de 70%. Em nosso caso viu-se que colocando o esforço de 50% para cada processo a planta estabilizava, na simulação. Outros valores de divisão do esforço computacional desestabilizavam o processo.

O algoritmo SPGPC possui estrutura similar ao GPC. A única mudança ocorre no algoritmo de resposta livre, o qual utiliza-se o preditor Smith em conjunto com o preditor ótimo como mencionado no Capítulo III, mostrado no diagrama da figura 4.12. Apesar da estrutura similar, o uso de um estimador tem importância crucial para este algoritmo. Diferente da utilização deste para o GPC e SGPC, visando adicionar apenas robustez ao sistema, o SPGPC utiliza-se da modelagem on-line para antecipar os valores de saída para o bloco “predictive Controller agua” e “predictive Controller agente”. A inferência com relação aos parâmetros da planta ou polarização dos mesmos poderia, neste caso, levar o processo a instabilidade. Porém o critério utilizado para o algoritmo utilizar a estimação *on-line* foi o de os parâmetros estarem dentro da faixa de 10%, para mais ou para menos, dos valores previamente identificados. Fora destes valores utilizou-se os parâmetros da planta identificada *a priori*. Isto foi feito com o intuito de adicionar robustez ao sistema, uma vez que foi visto que a convergência na identificação entre a planta real e modelada somente ocorreu após 100 segundos do início do processo.

De modo análogo, o algoritmo de identificação foi colocado em conjunto com o algoritmo SGPC, como mostrado na Figura 4.16. Os blocos “Prefilter agua”, “denominador agua” e “numerador agua” são *s-functions* que possuem o código o pré-filtro, o polinômio do denominador e o polinômio do numerador, respectivamente relacionados ao processo da água. De modo semelhante têm-se as *s-functions* “Prefilter agente”, “denominador agente” e “numerador agente” relacionado a planta identificada relacionado ao atuador do agente.

Os três algoritmos foram simulados no Simulink com um horizonte de predição de 3 passos a frente, um horizonte de referência de 3 passos a frente e com o fator de ponderação de 20%. A referência empregada nos blocos “Step” foi de valor 1, de modo a padronizar e facilitar a análise comparativa entre os algoritmos. A escolha destes parâmetros atendeu ao

requisito de um baixo consumo de memória por parte do microcontrolador, uma vez que esta era limitada.

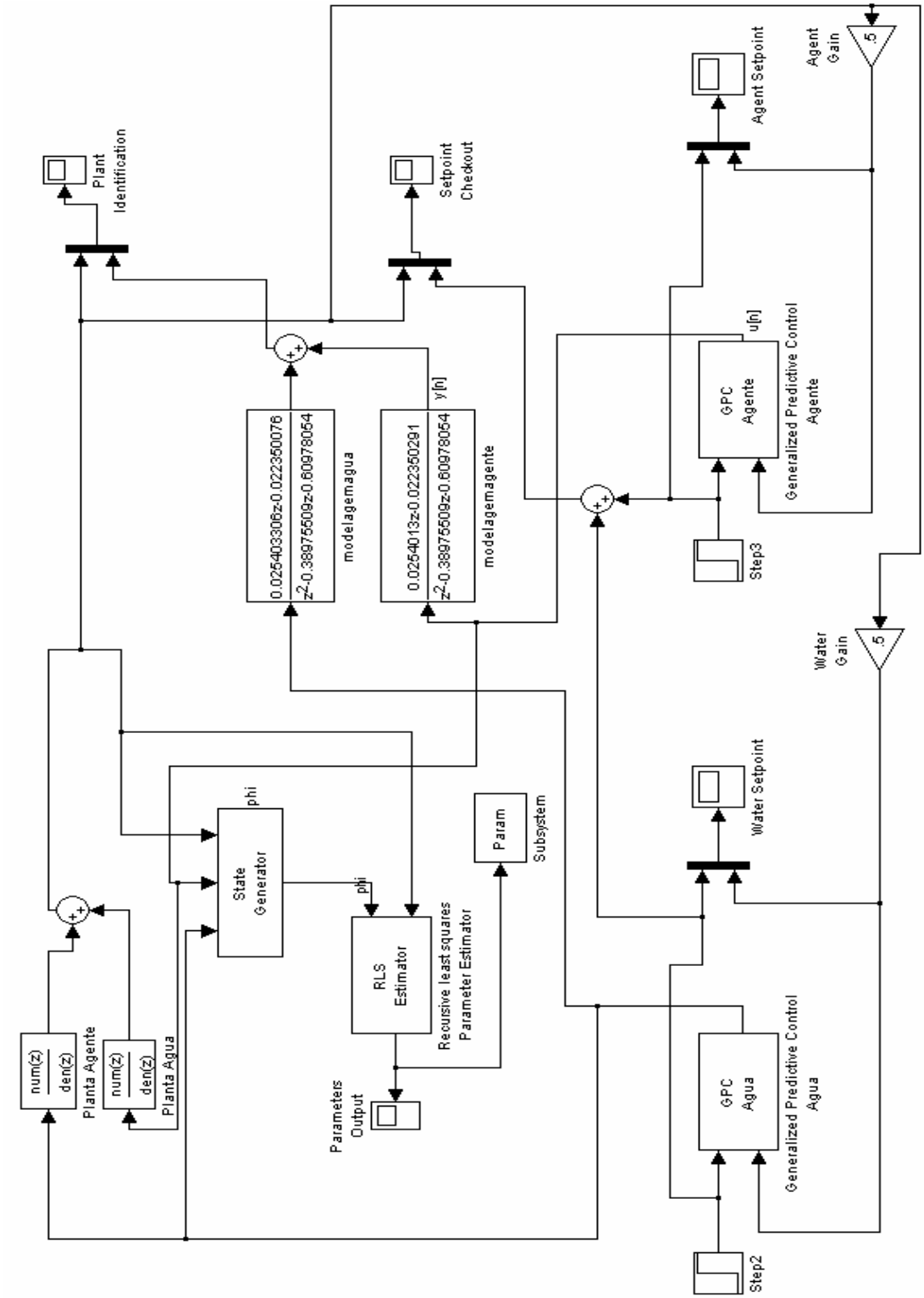

Figura 4.8 – Diagrama de blocos no Simulink do sistema de identificação dos parâmetros do processo em conjunto com o algoritmo de controle GPC aplicado a planta identificada.

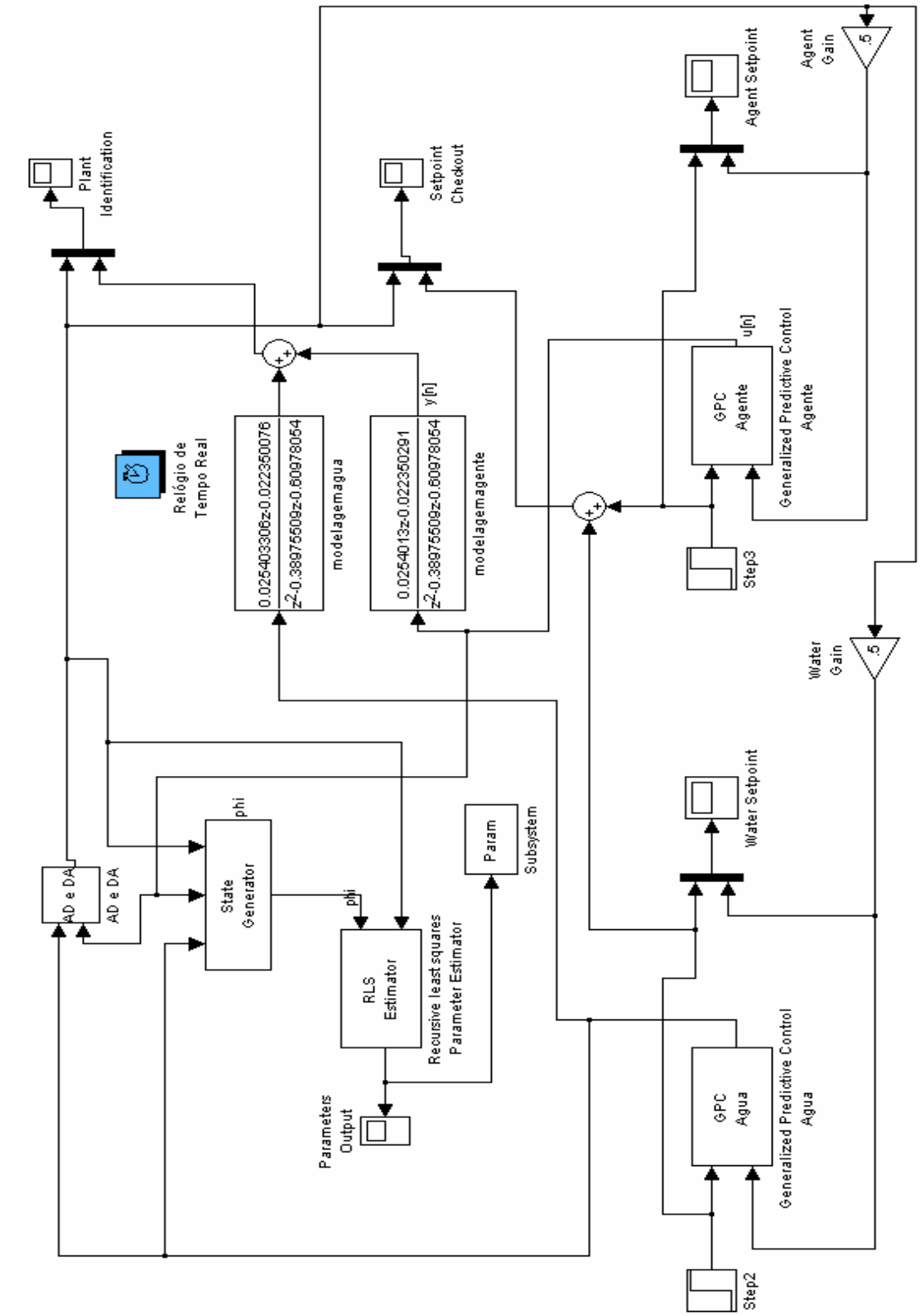

Figura 4.9 – Diagrama de blocos no Simulink do sistema de identificação dos parâmetros do processo em conjunto com o algoritmo de controle GPC aplicado a planta real

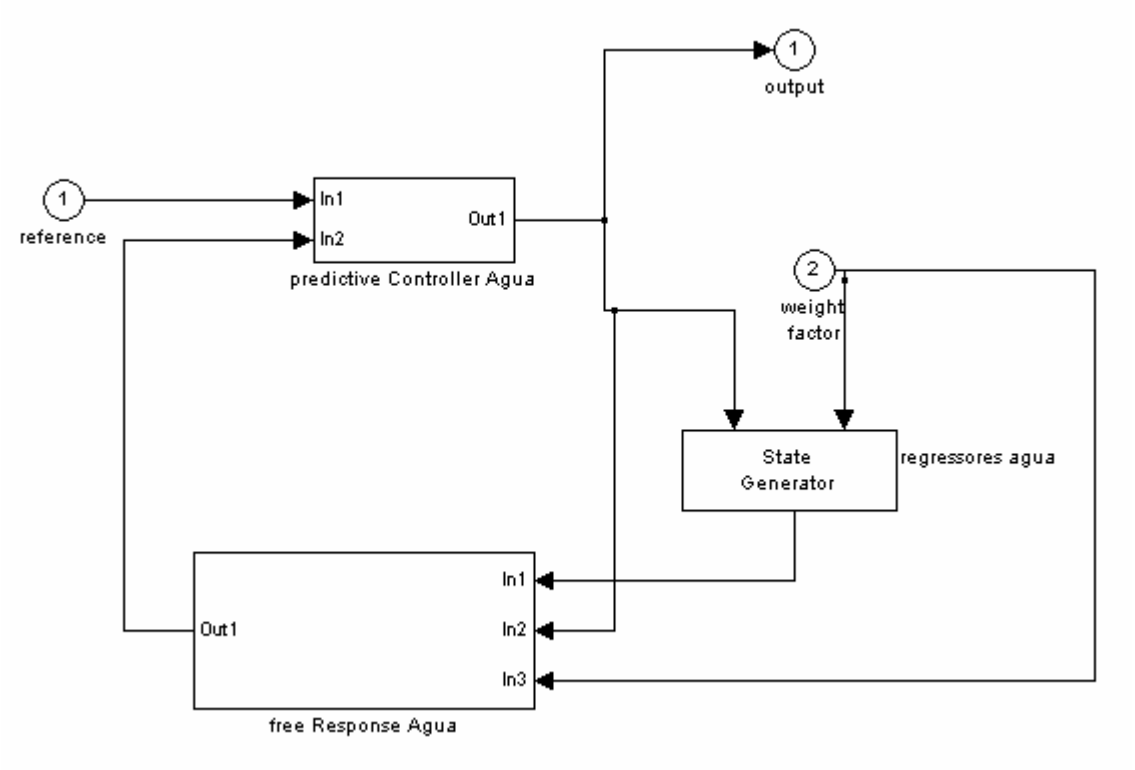

Figura 4.10 – Blocos sob a mascara "GPC Água" das figuras 4.8 e 4.9

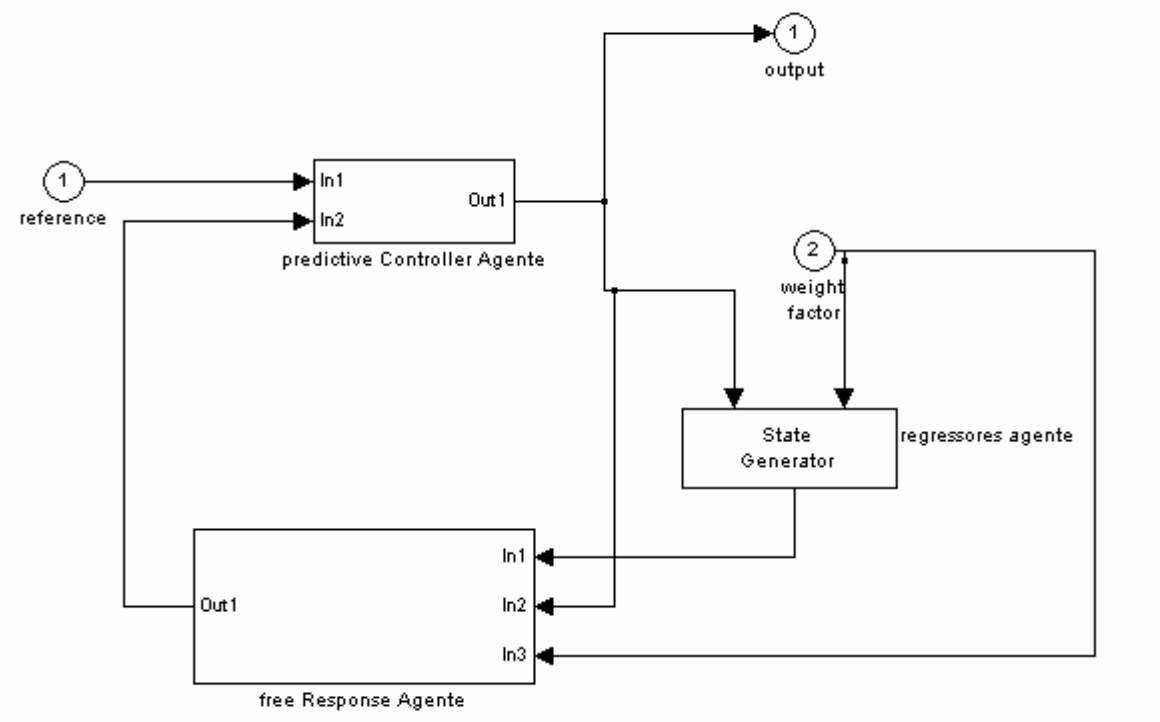

Figura 4.11 – Blocos sob a mascara "GPC Agente" das figuras 4.8 e 4.9

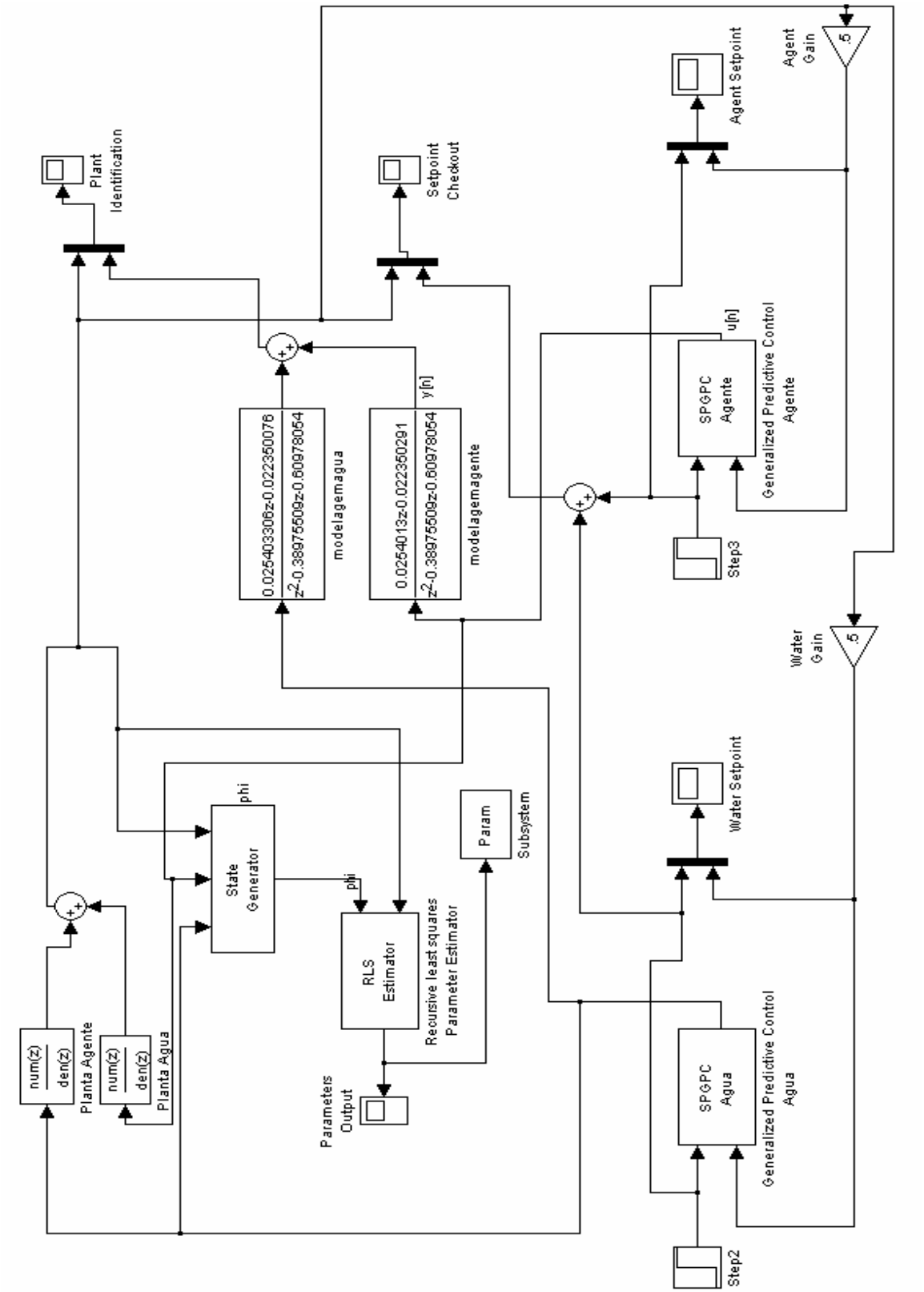

Figura 4.12 – Diagrama de blocos no Simulink do sistema de identificação dos parâmetros do processo em conjunto com o algoritmo de controle SPGPC aplicado a planta identificada

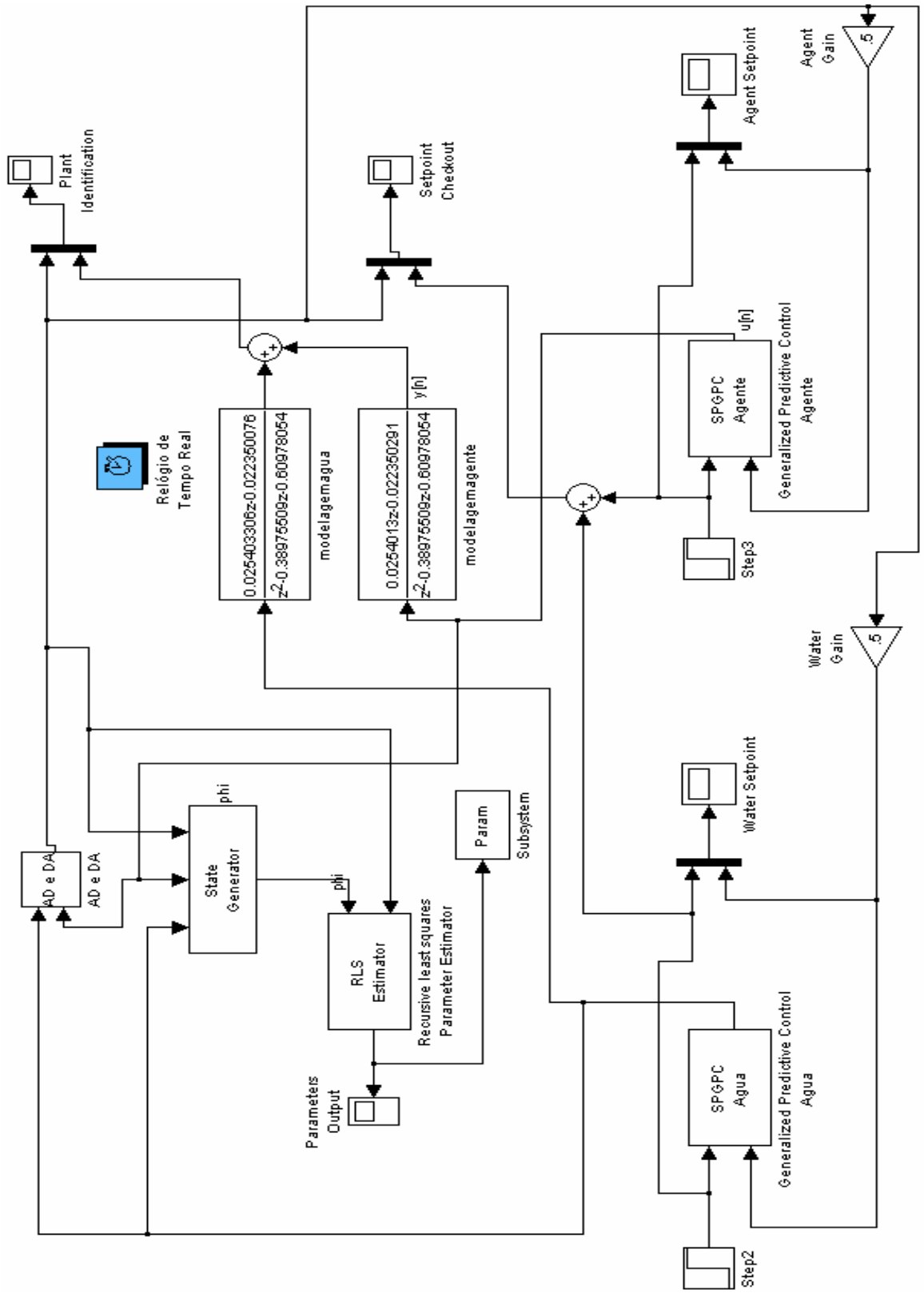

Figura 4.13 – Diagrama de blocos no Simulink do sistema de identificação dos parâmetros do processo em conjunto com o algoritmo de controle SPGPC aplicado a planta real.

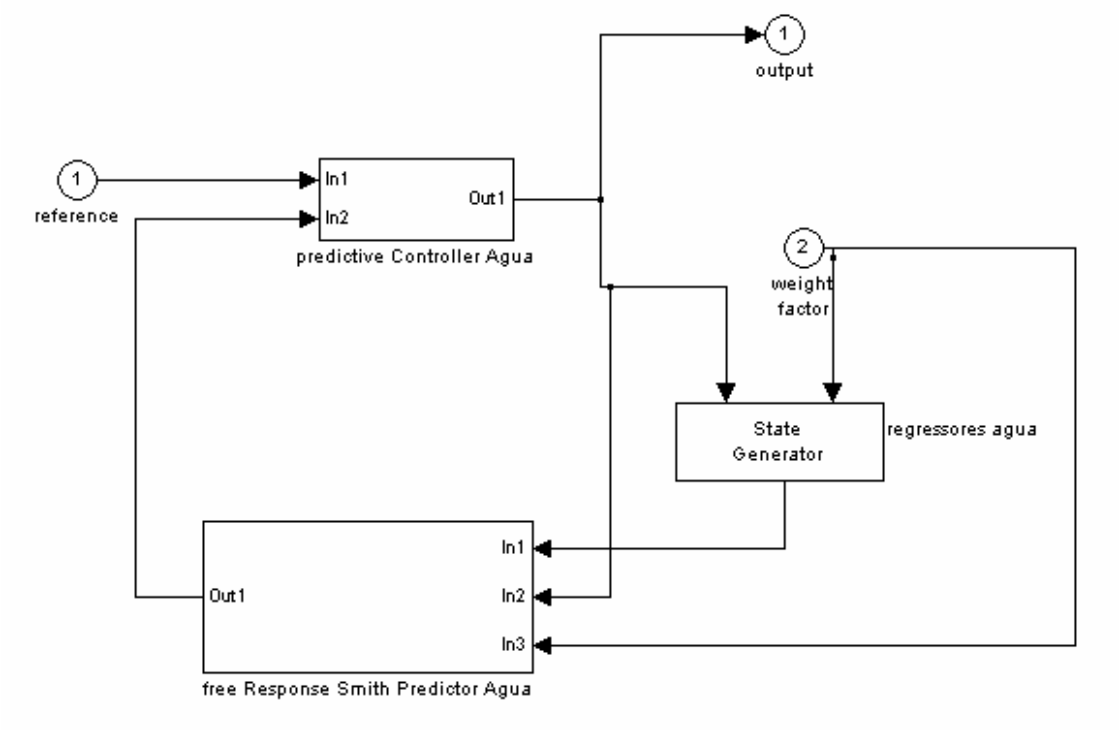

Figura 4.14 – Blocos sob a mascara “SPGPC Agua” das figuras 4.12 e 4.13

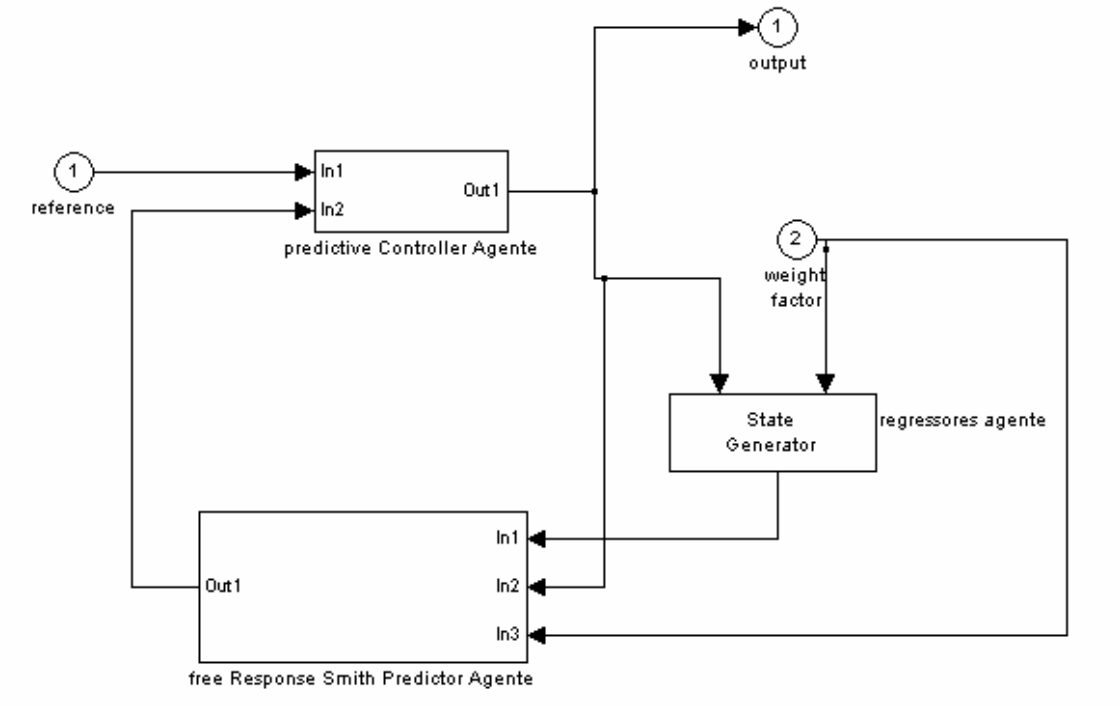

Figura 4.15 – Blocos sob a mascara “SPGPC Agente” das figuras 4.12 e 4.13

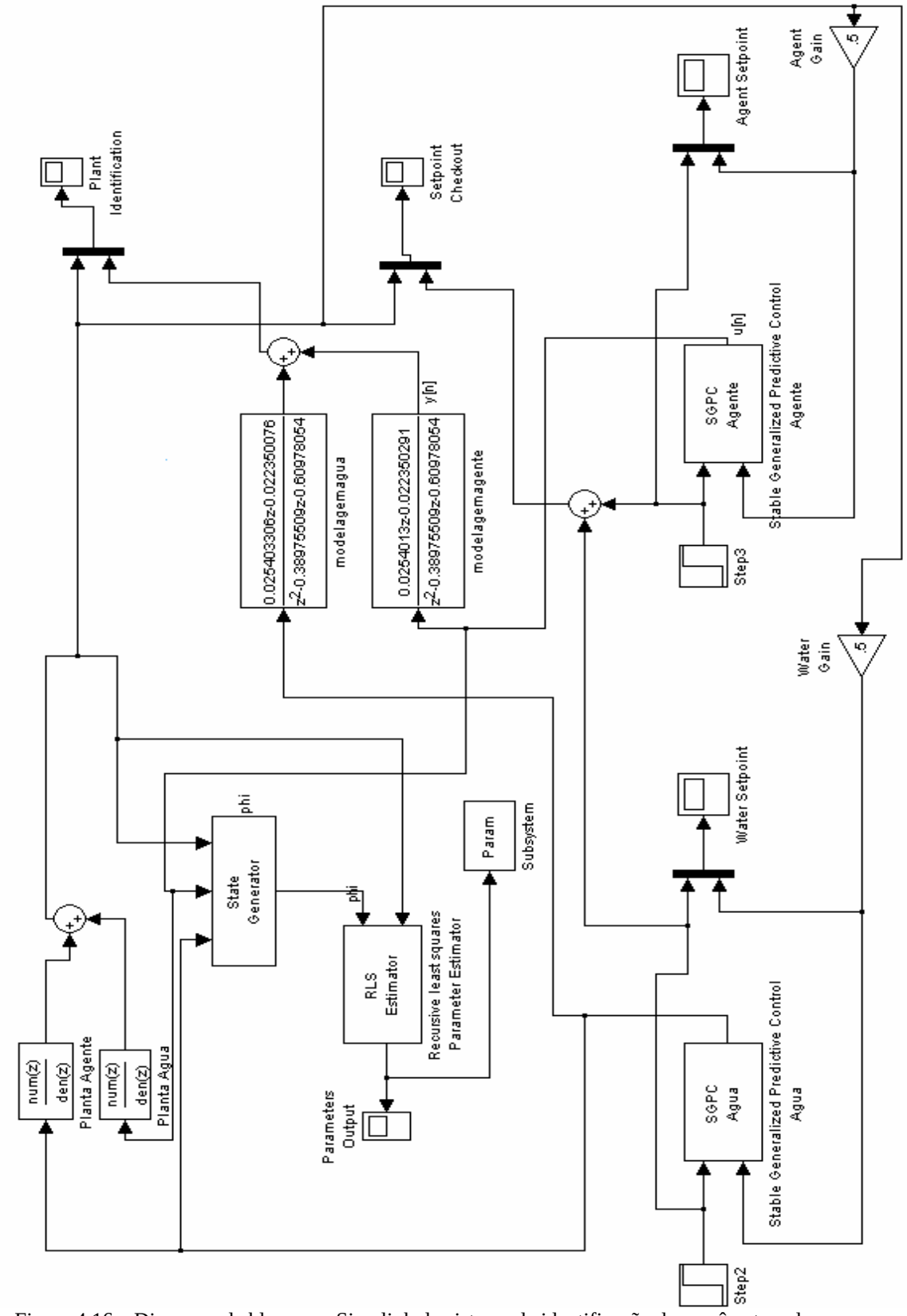

Figura 4.16 – Diagrama de blocos no Simulink do sistema de identificação dos parâmetros do processo em conjunto com o algoritmo de controle SGPC,aplicado a planta identificada.

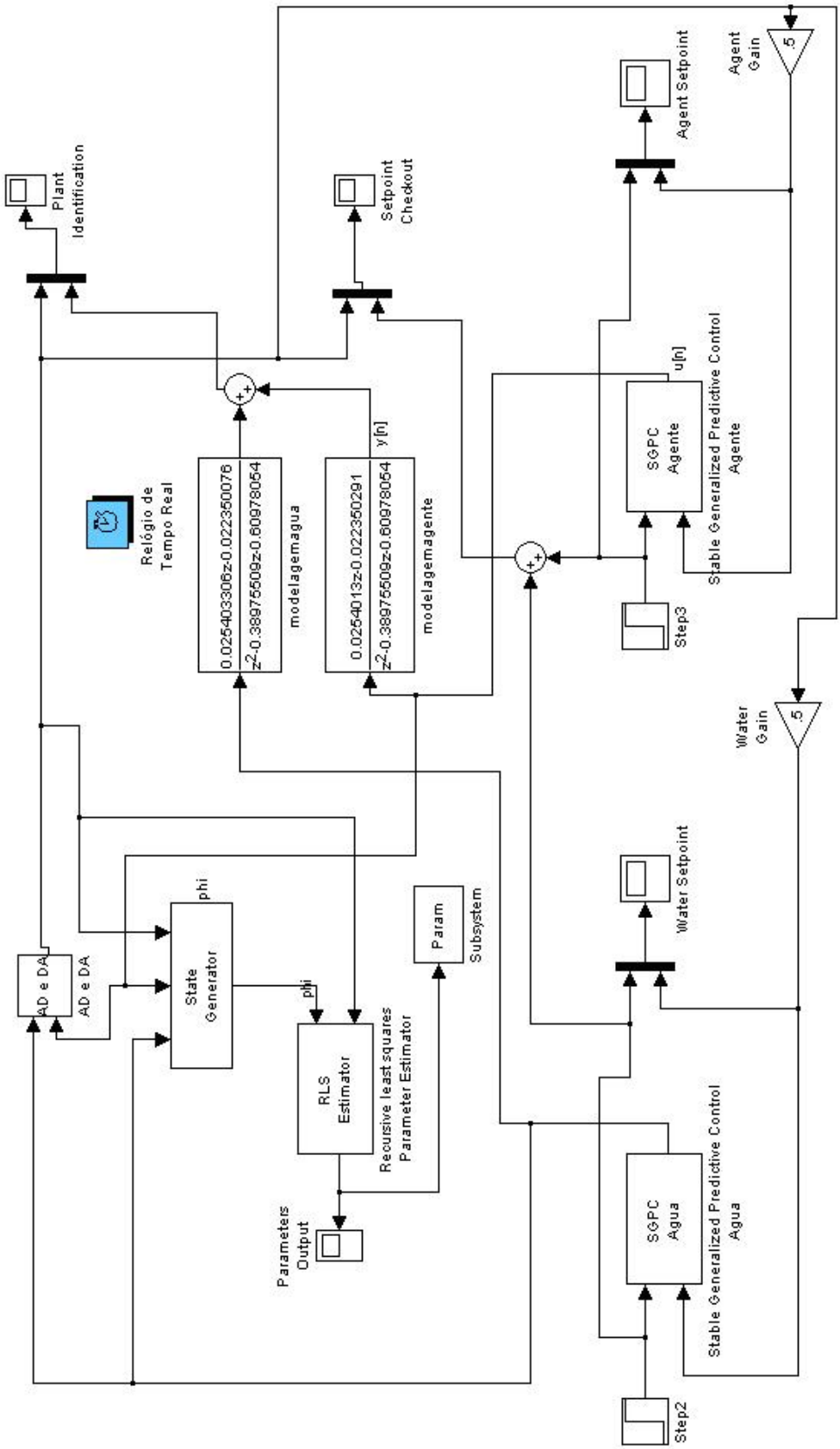

Figura 4.17 – Diagrama de blocos no Simulink do sistema de identificação dos parâmetros do processo em conjunto com o algoritmo de controle SGPC,aplicado a planta real.

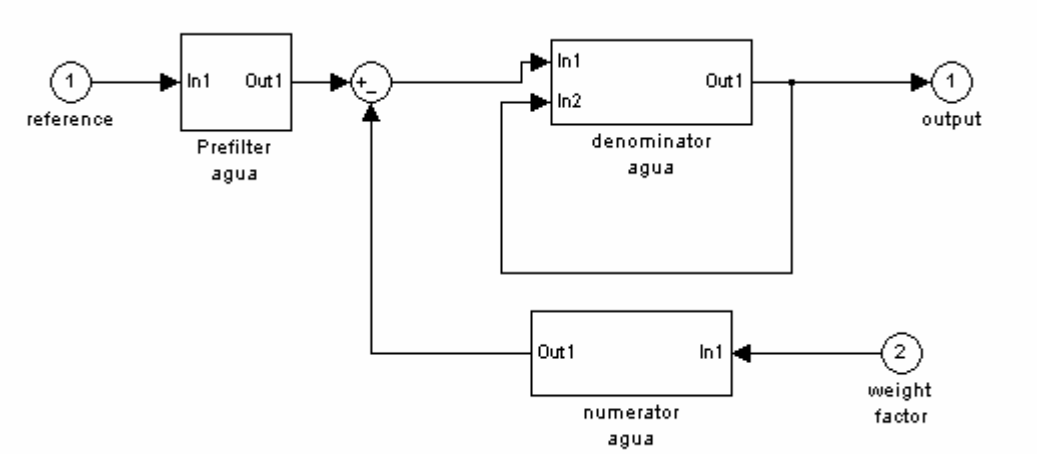

Figura 4.18 – Blocos sob a mascara “SGPC Agua” das figura 4.16 e 4.17

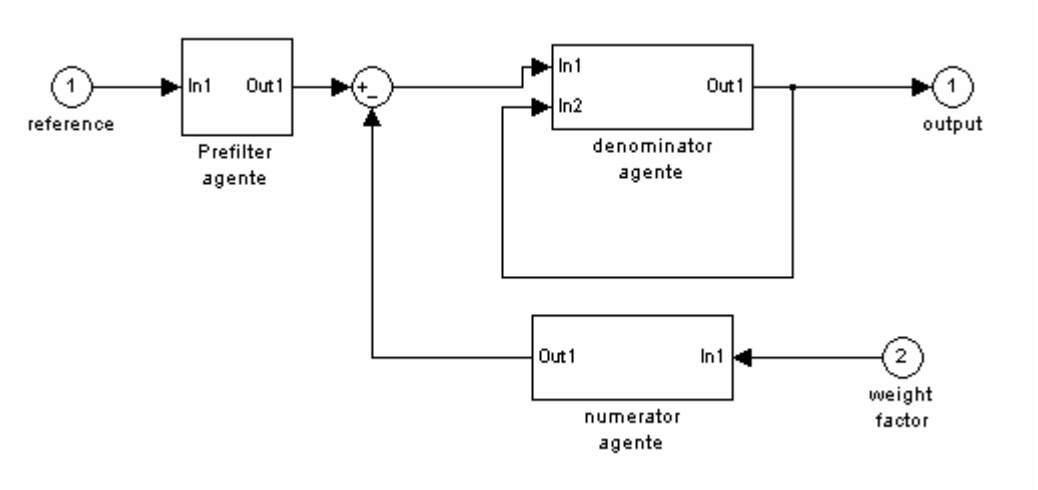

Figura 4.19 – Blocos sob a mascara “SGPC Agente” figura 4.16 e 4.17

4.5 CONSIDERAÇÕES FINAIS

A definição dos equipamentos que foram utilizados antes da execução do projeto foi um passo importante, uma vez que erros inerentes a uma modelagem mal feita, tanto de hardware como software, poderiam acarretar transtornos e tempo dispensado em uma possível análise dos problemas ocorridos. O uso de *s-functions* contribui para a análise gráfica do sistema, permitindo-se verificar o funcionamento de todos os algoritmos inerentes ao processo, além de facilitar os cálculos dos mesmos, uma vez que utiliza o Matlab para a resolução das equações. Assim se pode analisar as características dos três algoritmos preditivos sem a necessidade de implementação *a priori* na planta do processo. A identificação do sistema

utilizando *s-functions* também facilitou o trabalho de análise, uma vez que os gráficos puderam ser analisados “on-line”, observando-se as características do processo e a presença de ruídos em tempo real.

CAPÍTULO 5

MODELAGEM E RESULTADOS EXPERIMENTAIS

5.1 INTRODUÇÃO

A modelagem do sistema tem importância fundamental na realização do projeto de controle proposto neste trabalho, uma vez que se pode fazer a análise minuciosa da estrutura e implementar diversos algoritmos eliminando a necessidade de um procedimento experimental a cada vez que se necessitar de dados para o projeto, pois o comportamento matemático da estrutura reflete de maneira muito aproximada o comportamento da planta real, para uma determinada região em que se deseja fazer a análise. Os resultados experimentais tornam-se então uma confirmação do que já foi avaliado matematicamente, servindo para corroborar os dados obtidos na modelagem.

Neste capítulo, faz-se primeiramente a modelagem matemática da planta, isto é, o cálculo dos parâmetros das equações que regem o sistema, escolhendo o modelo ARMAX para descrever o processo. Após, aplica-se os algoritmos de controle e identificação juntos na planta modelada para análise comparativa, utilizando a resposta do sistema ao controle e o esforço dos atuadores para análise qualitativa dos algoritmos. Em seguida faz-se a aplicação destes algoritmos no sistema real, para ratificar os dados e análise efetuadas através das simulações.

5.2 IDENTIFICAÇÃO EXPERIMENTAL DO SISTEMA

O Estimador dos Mínimos Quadrados Estendidos é uma modificação no Estimador dos Mínimos Quadrados que visando remover a polarização dos parâmetros do sistema, através da modelagem do ruído como parâmetros da planta, devido a mudanças climáticas ou alteração dos agentes do sistema. Esta polarização é alocada dentro de um ruído colorido modelado,

definindo um modelo ARMAX. Portanto, no nosso caso, a planta modelada obedece a seguinte equação:

$$y[k] = \sum_{n=1}^{3} a_n y[k-n] + \sum_{n=1}^{2} b_n u_1[k-n] + \sum_{n=1}^{2} c_n u_2[k-n] + d_1 v[k] + d_2 v[k-1] \quad (5.1)$$

Sendo y[n] a saída do sistema, $u_1[n]$ é a entrada da água, $u_2[n]$ é a entrada do agente e v[k] é um ruído branco. Como pode ser visto no modelo acima, o ruído colorido modelado é de segunda ordem, que implementa um alto grau de robustez para que a presença de possíveis distúrbios não tenham influência significativa na modelagem. A planta modelada também foi de segunda ordem. Esta escolha se deveu ao fato que posteriormente este algoritmo seria implementado em um microcontrolador, que diferentemente de um microcomputador não possui recursos de memória que permita uma modelagem com maior nível de detalhamento.

Utilizou-se a equação abaixo para a identificação de sistemas MISO, que é o mesmo caso do modelo SISO com o vetor de regressores estendidos para as duas entradas[14]:

$$\hat{\theta}_k = \hat{\theta}_{k-1} + K_k[y(k) - \psi_k^T \hat{\theta}_{k-1}] \quad (5.2)$$

$$K_k = P_{k-1}\psi_k[\psi_k^T P_{k-1}\psi_k + 1]^{-1} \quad (5.3)$$

$$\xi(k) = y(k) - \psi_k^T \hat{\theta}_k \quad (5.4)$$

$$P_k = P_{k-1} - K_{k-1}\psi_k^T P_{k-1} \quad (5.5)$$

Sendo o vetor de regressores definido como:

$$\psi(k-1) = \begin{bmatrix} y[k-1], y[k-2], y[k-3], u_1[k-1], u_1[k-2], u_2[k-1], \\ u_2[k-2], v[k], v[k-1] \end{bmatrix} \quad (5.6)$$

Ao efetuar a primeira identificação da planta utilizando-se das ferramentas descritas no Capítulo IV, obtive-se graficamente a seguinte identificação da planta:

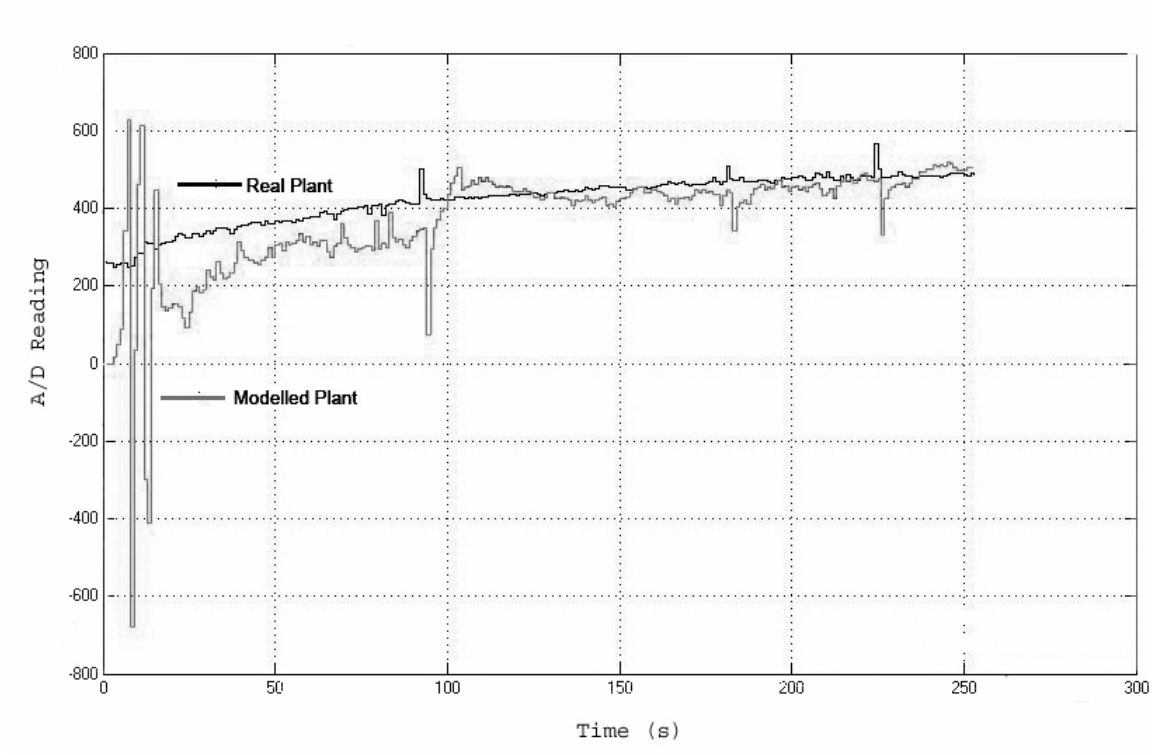

Figura 5.1 – Primeira Identificação On-line do Processo

Como se nota, após 100 segundos a saída da planta modelada estava próxima à saída da planta real, para uma mesma entrada. Os parâmetros da planta modelada, mostrados na equação (5.2), calculados com o algoritmo dos Mínimos Quadrados Estendido foram:

$$\begin{aligned} &a_1 = 1, a_2 = -1.3240372, a_3 = 0.34165083, b_1 = 0.86538162, \\ &b_2 = \text{-}0.82439612,\ c_1 = 1.0350718, c_2 = -0.97811792 \end{aligned} \tag{5.7}$$

Para validar esta identificação, efetuou-se o mesmo processo mudando-se a fonte da água e com diferentes condições climáticas(alteração da umidade do ar), as quais afetam os valores de condutividade dos reagentes. Estas condições são consideradas ruídos, logo, teoricamente, não afetam a identificação do sistema, uma vez que são modelados em separado e não fazem parte dos parâmetros da planta identificada. Para esta identificação foi conseguido o gráfico da Figura 5.2, mostrado abaixo:

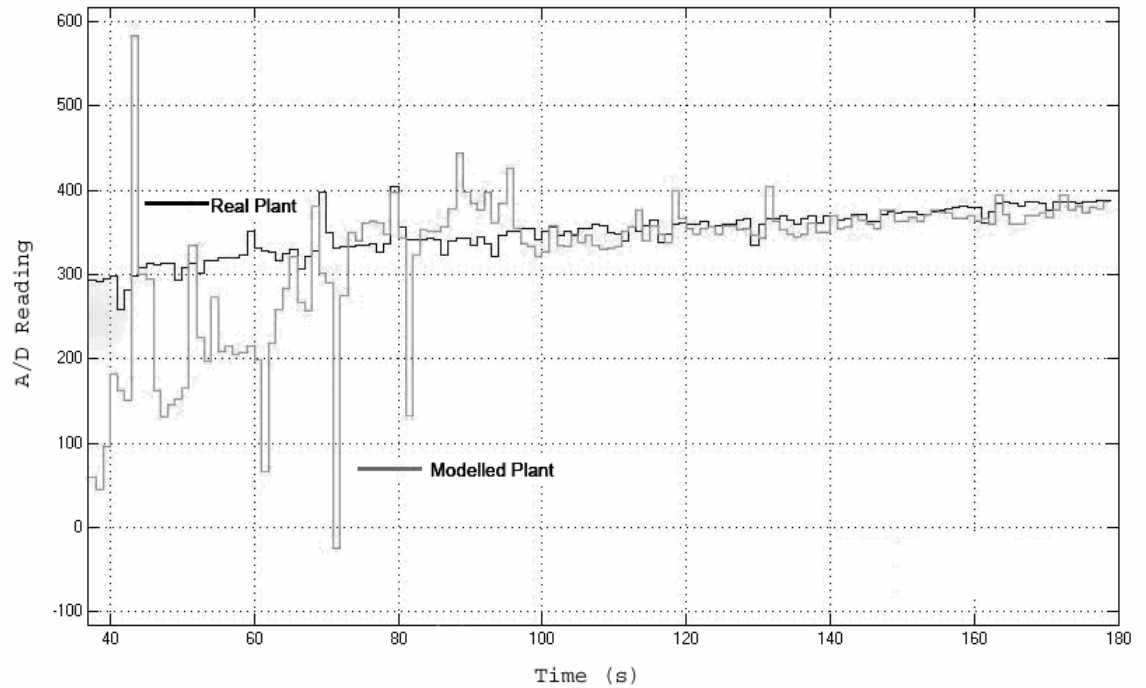

Figura 5.2 – Segunda Identificação On-line do Processo

Como não houve diferença significativa no valor dos parâmetros com relação a primeira identificação (em torno de 0,6% de erro), foi utilizado os parâmetros da primeira planta identificada.

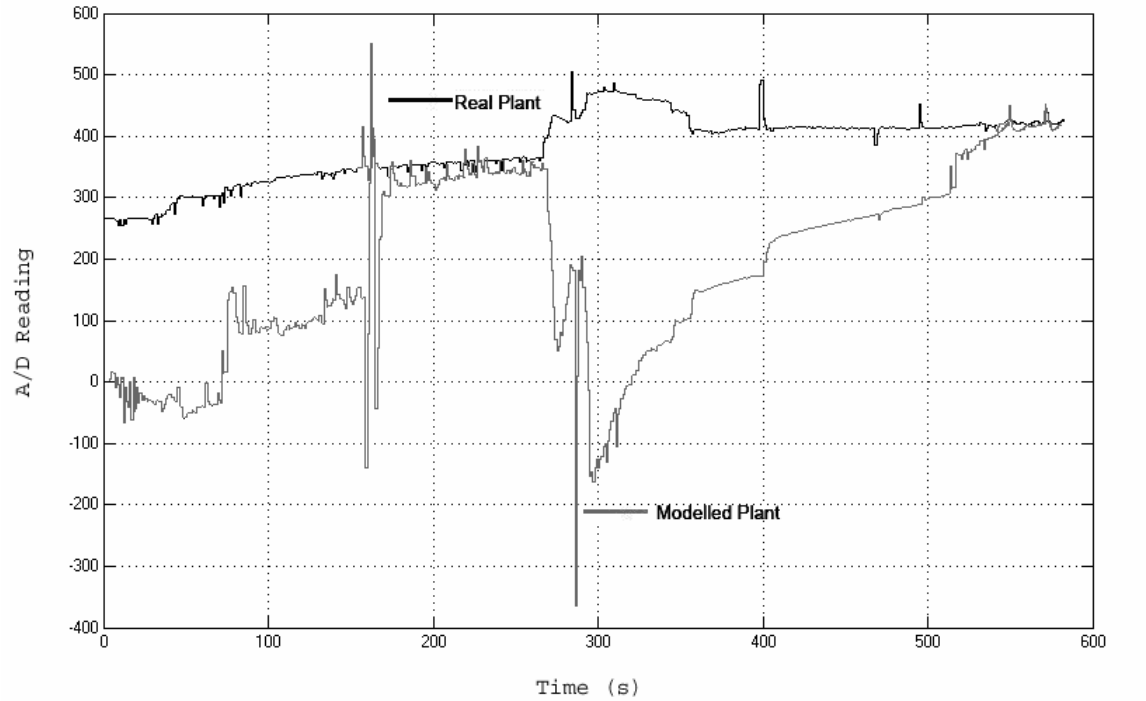

Figura 5.3 – Identificação da Planta com a Presença de Distúrbio

Como teste de robustez introduzimos um distúrbio considerável ao sistema, efetuando a adição de uma grande quantidade de agente de molha na mistura, adicionando água logo após, de modo a alterar a condutividade. O resultado é visto na Figura 5.3.

Com os parâmetros das plantas em mãos, foi feita uma análise de cada sistema SISO no qual o processo era composto. O gráfico do lugar das raízes para a planta relacionada com a água é mostrada na Figura 5.4. Fato a se notar é que a planta possui um pólo próximo ao círculo de raio unitário(z=0.9729), o que denota que a mesma está no limiar da instabilidade[20]. Além disso, a proximidade deste pólo com o zero(z=0.9526) não garante que este sistema seja controlável tão pouco observável[20], como pode ser verificado na Figura 5.4 e na Figura 5.5. Adiciona-se a isso o fato de os dados identificados da planta serem apenas uma aproximação do modelo real da mesma. A garantia de estabilidade nestas condições é somente garantida pelo algoritmo SGPC[6].

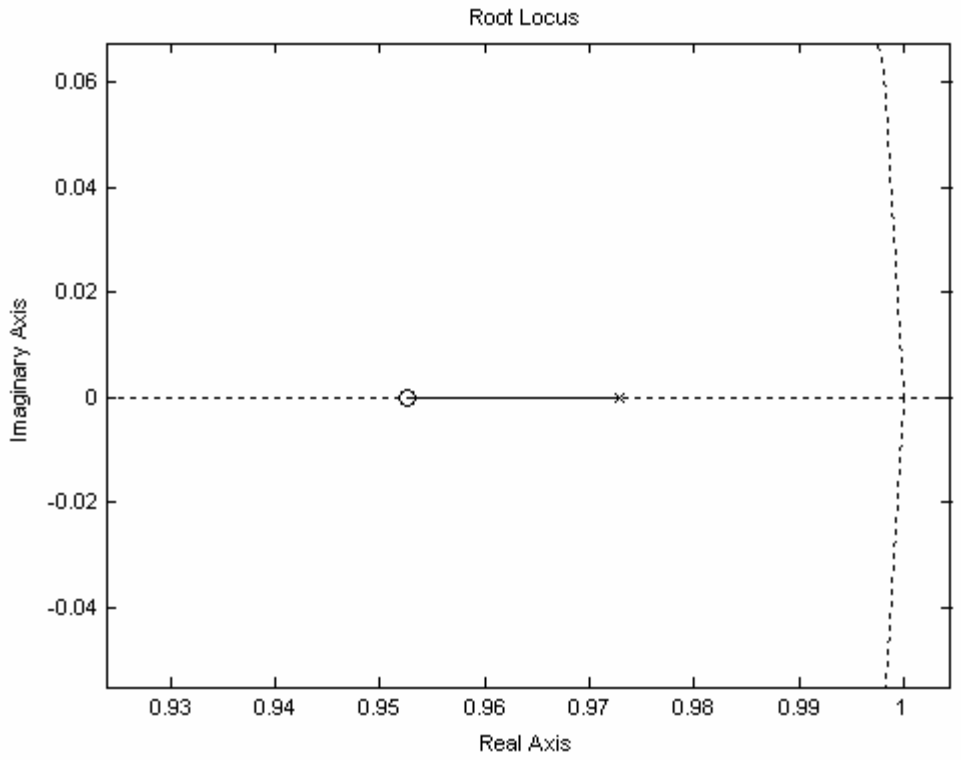

Figura 5.4 – Lugar das raízes para a planta SISO relacionada com a água, focalizando o zero e o pólo mais próximos.

A partir destes dados, utilizou-se o critério de Routh-Hurwitz modificado através de transformação *r* para se fazer a análise de estabilidade do sistema[20], indicando se o sistema é BIBO estável ou não. Este sistema utiliza a seguinte transformação bi-linear para mapeamento do plano z no plano r:

$$z = \frac{r+1}{r-1} \tag{5.8}$$

A equação característica do sistema é dada por:

$$F(z) = z^2 - 1,3240372z + 0,34165083 = 0 \tag{5.9}$$

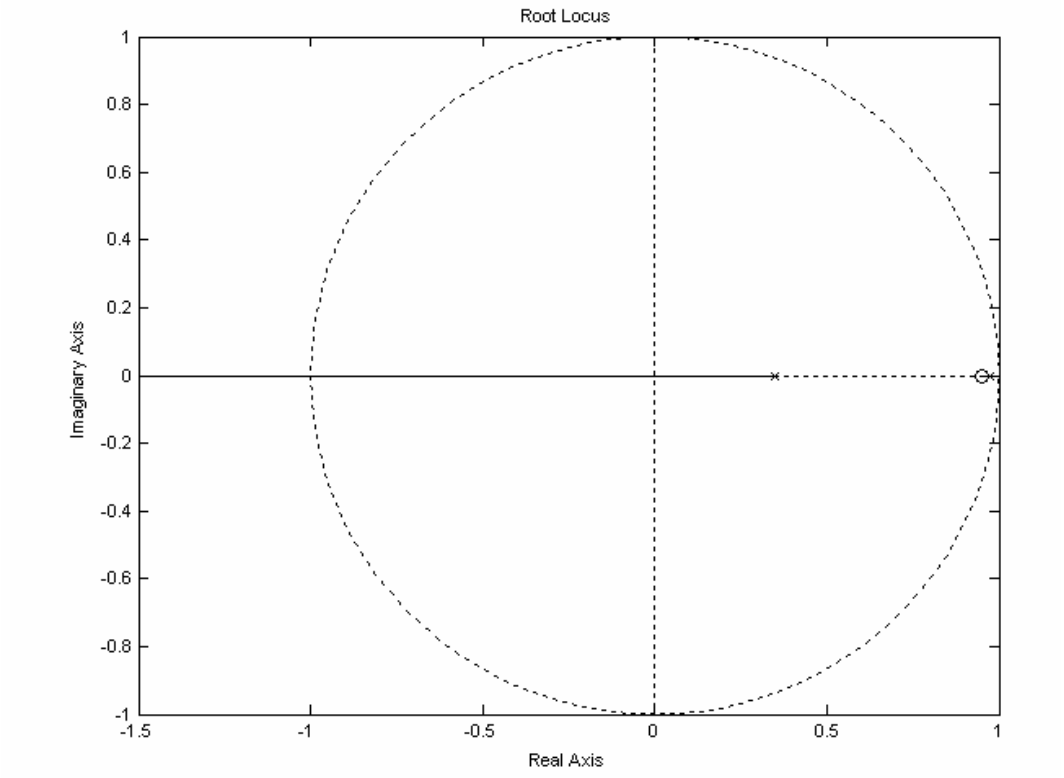

Figura 5.5 – Lugar das raízes para a planta SISO relacionada com a água

Utilizando-se a equação (5.5), a equação (5.6), no domínio *r*, é dada por:

$$0,01761363r^2 + 1,316698r + 0,6759620 = 0 \tag{5.10}$$

Têm-se então a partir do critério de Routh-Hurwitz:

$$\begin{bmatrix} r^2 & 0,01761363 & 0,6759620 \\ r^1 & 1,316698 & 0 \\ r^0 & 0,8724378 & \end{bmatrix} \tag{5.11}$$

Como não houve mudança de sinal, podemos concluir que este sistema é BIBO estável [20].

Fazendo a mesma análise para a planta SISO relacionada ao agente, temos o gráfico do lugar das raízes mostrado na Figura 5.6.

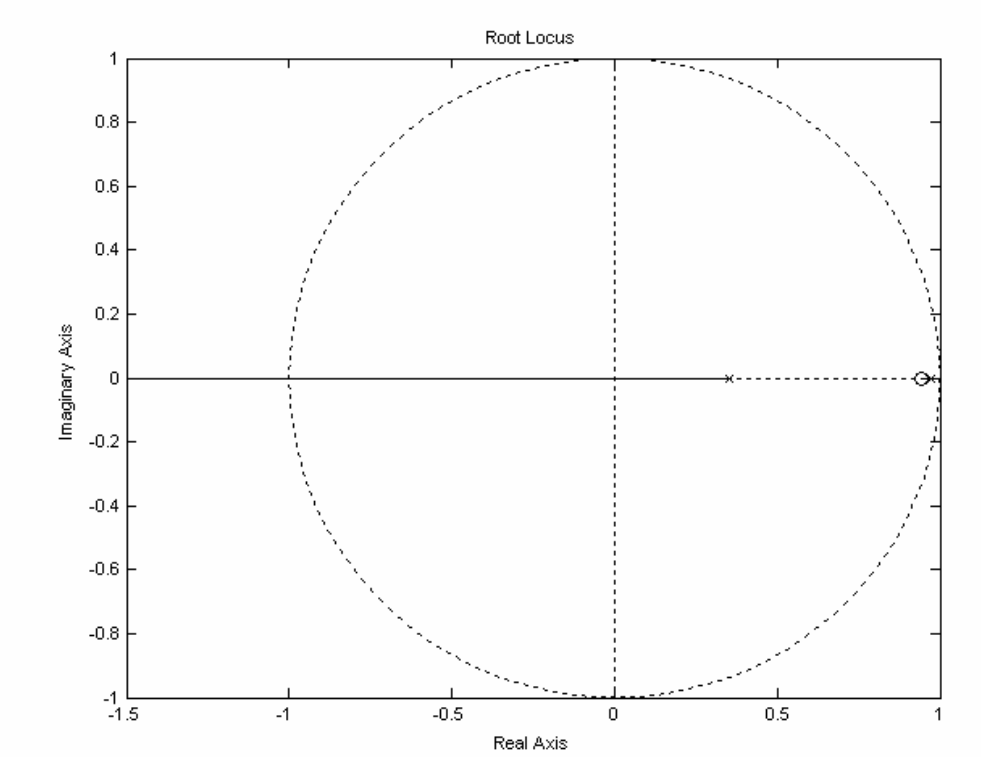

Figura 5.6 – Lugar das raízes para a planta SISO relacionada com a agente

Novamente recai-se no problema de um pólo (z=0,9729) muito próximo a um zero (z=0,9450). Mas como a equação característica é a mesma para a modelagem da planta relacionada à água, pode-se concluir que este processo também é BIBO estável, logo controlável e observável.

5.3 ANÁLISE COMPARATIVA DOS ALGORITMOS DE CONTROLE UTILIZANDO A PLANTA IDENTIFICADA.

Tendo por base os parâmetros calculados na identificação da planta feito no ítem anterior, efetuamos o controle do sistema utilizando os algoritmos GPC, SPGPC e SGPC. No caso específico do SPGPC, utilizou-se um tempo morto de três constantes de tempo. Para efeito comparativo e devido aos baixos recursos de memória do microcontrolador, utilizamos um horizonte de predição de três passos à frente, um horizonte de controle de um passo a frente e um fator de ponderação de 0,8 para os três algoritmos. Com isso, utilizamos o modelo CARIMA para cada uma das entradas de controle(água e agente). O modelo CARIMA é dado por:

$$A(z^{-1})y(t) = B(z^{-1})z^{-d}u(t-1) + C(z^{-1})e(t)/\Delta \qquad (5.12)$$

O resultado obtido para o GPC utilizando o modelo da planta é mostrado na Figura 5.7. O SPGPC obteve melhor resultado no que tange a robustez ao ruído, rapidez do processo para alcançar o "*setpoint*" desejado, como mostrado na Figura 5.9, com um esforço de controle muito próximo ao do GPC, como pode ser visto ao se comparar as figuras 5.8 e 5.10. A resposta da planta a aplicação do algoritmo SGPC é mostrada na Figura 5.11, enquanto o esforço de controle requisitado pelo mesmo algoritmo é mostrado na Figura 5.12. Tomando-se como um erro em regime de 5%, a planta que utiliza o GPC somente se adeqüa a esta especificação 120 constantes de tempo após o início do processo, número que diminui para 15 constantes de tempo no caso do SPGPC. Enquanto a planta que utiliza SGPC chega a este valor em 14 constantes de tempo, nas mesmas condições. Se for tomado como parâmetro

especificado para o processo um erro de regime de 1%, o GPC consegue êxito apenas 143 constantes de tempo após o início do processo, contra 19 constantes de tempo no SPGPC, ao passo que o SGPC atende a mesma especificação após 18 constantes de tempo. O erro de regime absoluto da primeira planta, quando não há mais variação neste parâmetro, foi de 0,001% para ambos GPC e SPGPC, enquanto o SGPC conseguiu erro de 0% no mesmo caso. Porém o processo com o SGPC teve um sobresinal de 480%, enquanto não houve sobresinal nas outras duas plantas. Esta última característica possui uma relevância com relação a saturação dos atuadores no processo experimental, uma vez que na simulação este sinal não possuia limitações. Nos três algoritmos foram adicionados ruídos coloridos de segunda ordem com 1% do tamanho do sinal de referência, simulando assim o ruído do processo experimental. Todos mostraram robustez a este tipo de ruído.

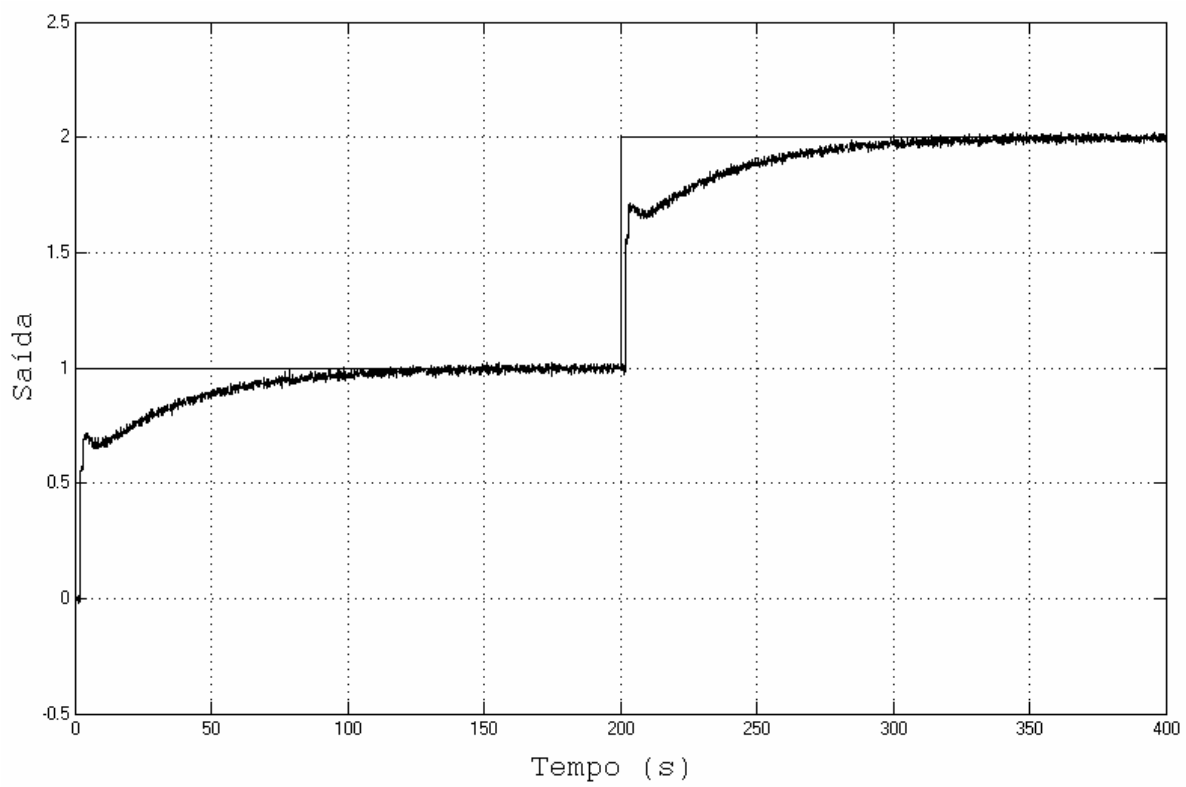

Figura 5.7 – Controle da planta modelada utilizando-se GPC

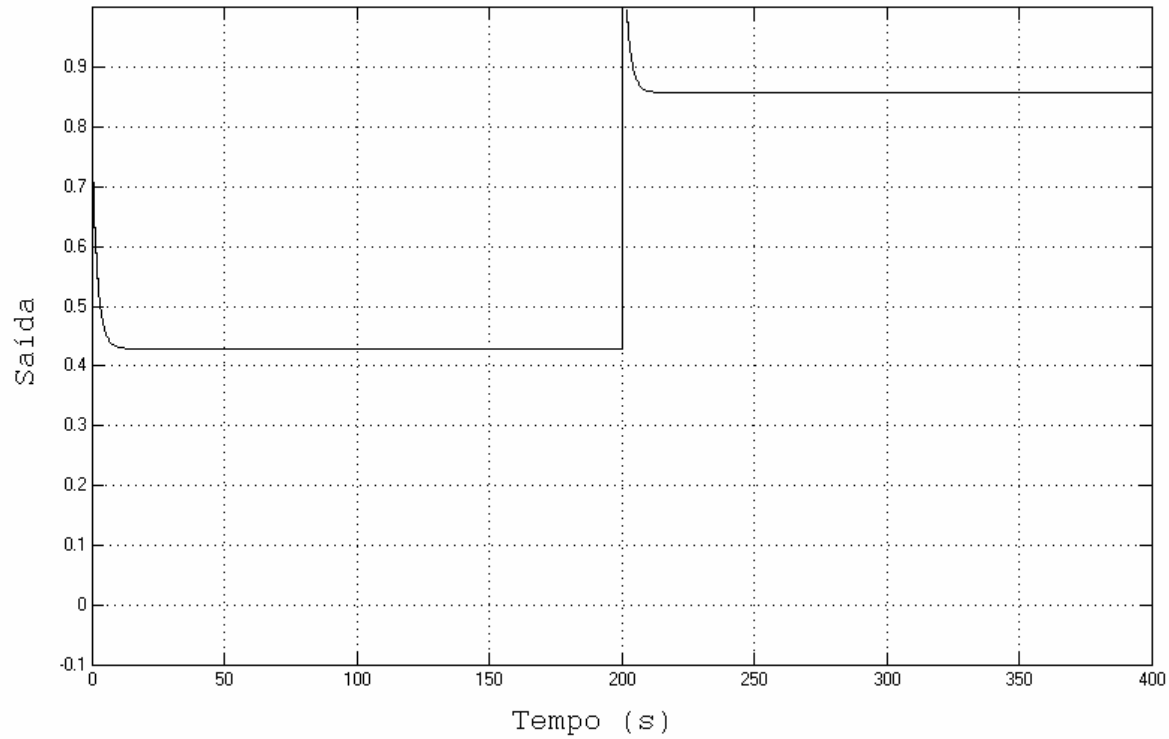

Figura 5.8 – Esforço de controle no atuador na planta modelada utilizando-se GPC

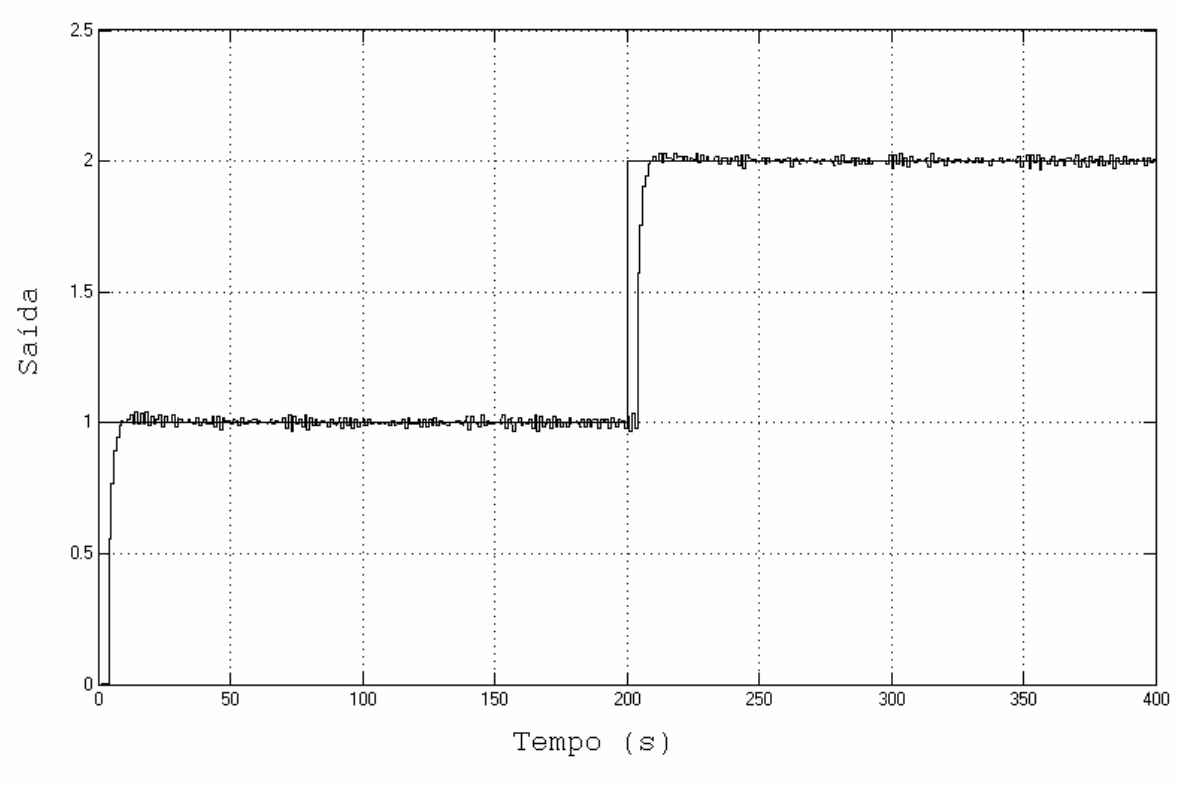

Figura 5.9 – Controle da planta modelada utilizando-se SPGPC

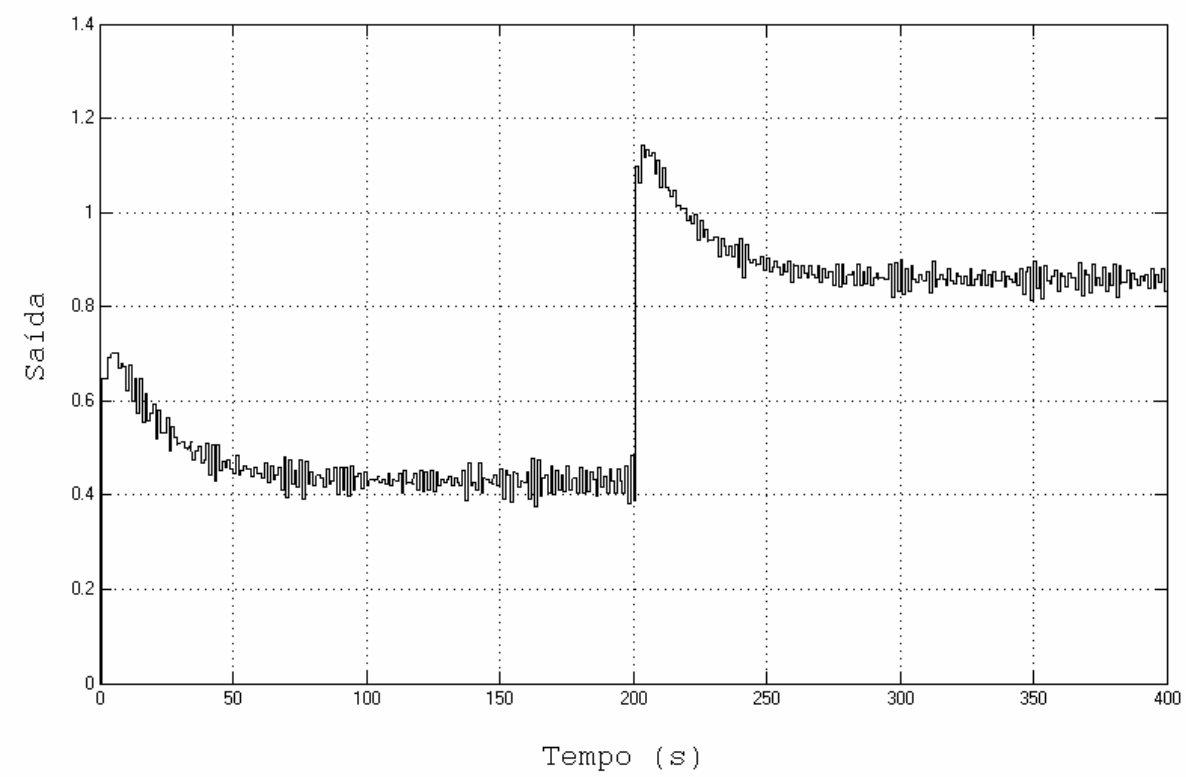

Figura 5.10 – Esforço de controle no atuador na planta modelada utilizando-se SPGPC

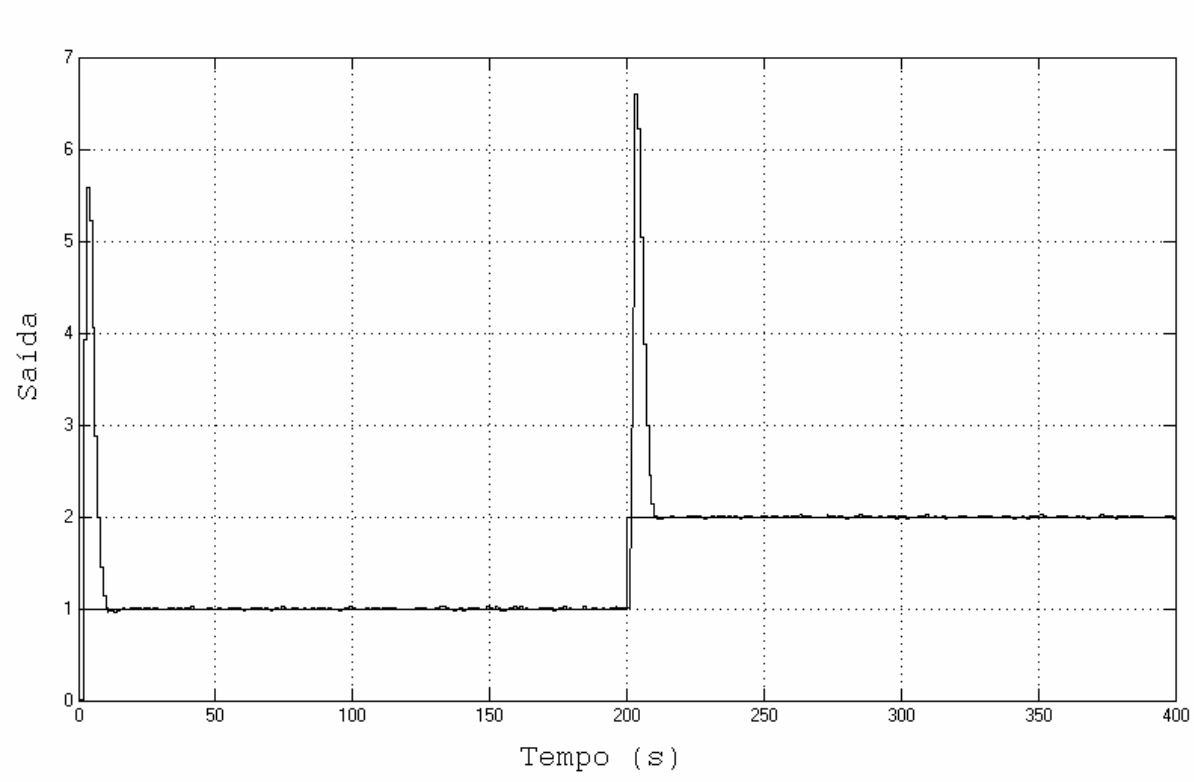

Figura 5.11 – Controle da planta modelada utilizando-se SGPC

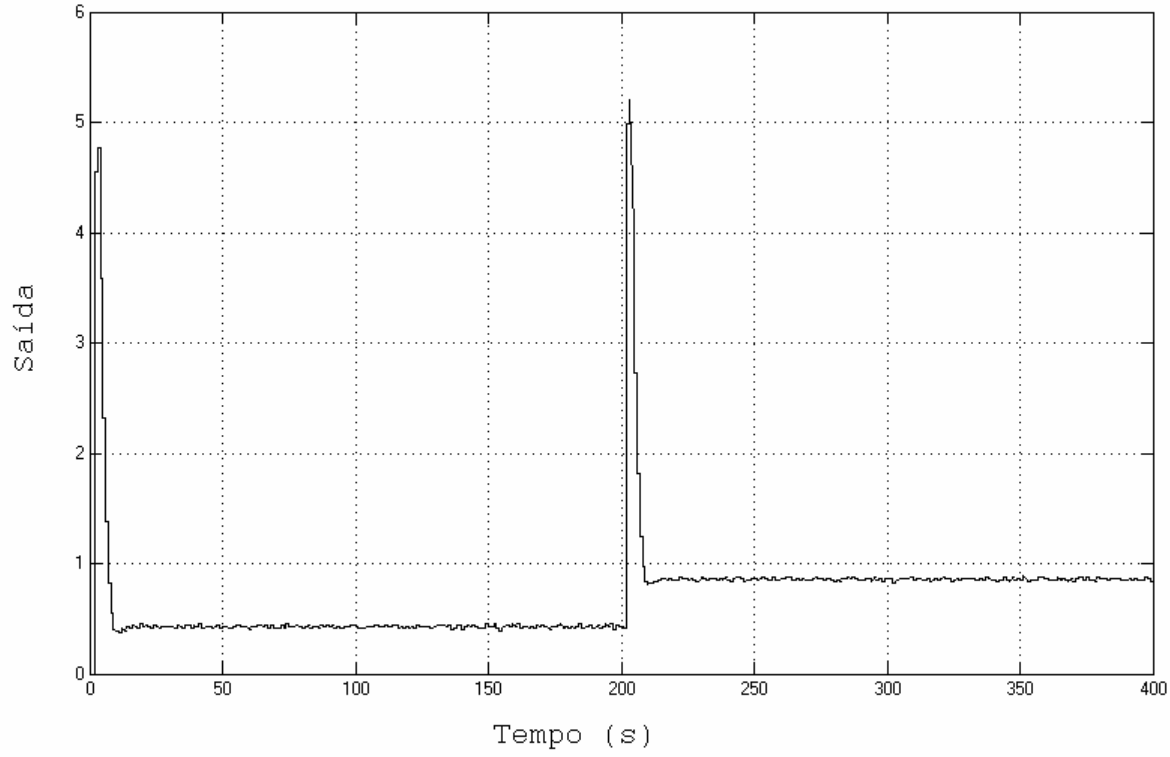

Figura 5.12 – Esforço de controle no atuador na planta modelada utilizando-se SGPC

5.4 ANÁLISE COMPARATIVA DOS ALGORITMOS DE CONTROLE NO PROCESSO EXPERIMENTAL.

Analisando-se os gráficos percebe-se que o SGPC não consegue seguir a referência com o êxito, como mostrado na Figura 5.13. Na análise experimental foi notado que a aplicação do algoritmo SGPC à planta saturava os atuadores rapidamente. Pode-se ver isso claramente na simulação, quando a resposta da planta a atuação deste algoritmo apresentou um grande sobresinal. Vê-se também que este algoritmo apresentou um erro, relacionado à referência, de cerca de 15%, e atingiu o estado de regime com 200 segundos. Sendo este o primeiro projeto a utilizar-se de tal algoritmo em uma análise experimental[19], não se pode avaliar sua performance com relação a outros processos.

A performance do GPC é mostrada na Figura 5.14. Não houve saturação nos atuadores, como no SGPC, mas a atividade destes foi significativa, pois notou-se a presença de ruídos no sinal, como pode-se notar na Figura 5.14, apesar da adição de um filtro capacitivo na entrada analógico/digital do microcontrolador. O erro de regime foi de 3%. O sobresinal foi praticamente ausente, salientando o fato de que este foi igual ao erro de regime após a referência do sinal ter sido alcançada, 100 segundos após o início do processo.

Uma análise do SPGPC pode ser conseguida através da Figura 5.15. Este algoritmo apresentou menor nível de ruído e utilização dos atuadores para estabilização do processo[23]. O erro de regime ficou em 1%. Este algoritmo porém foi o mais lento dos três analisados, com relação ao alcance do regime, que ocorreu apenas a 155 segundos após o início do processo.

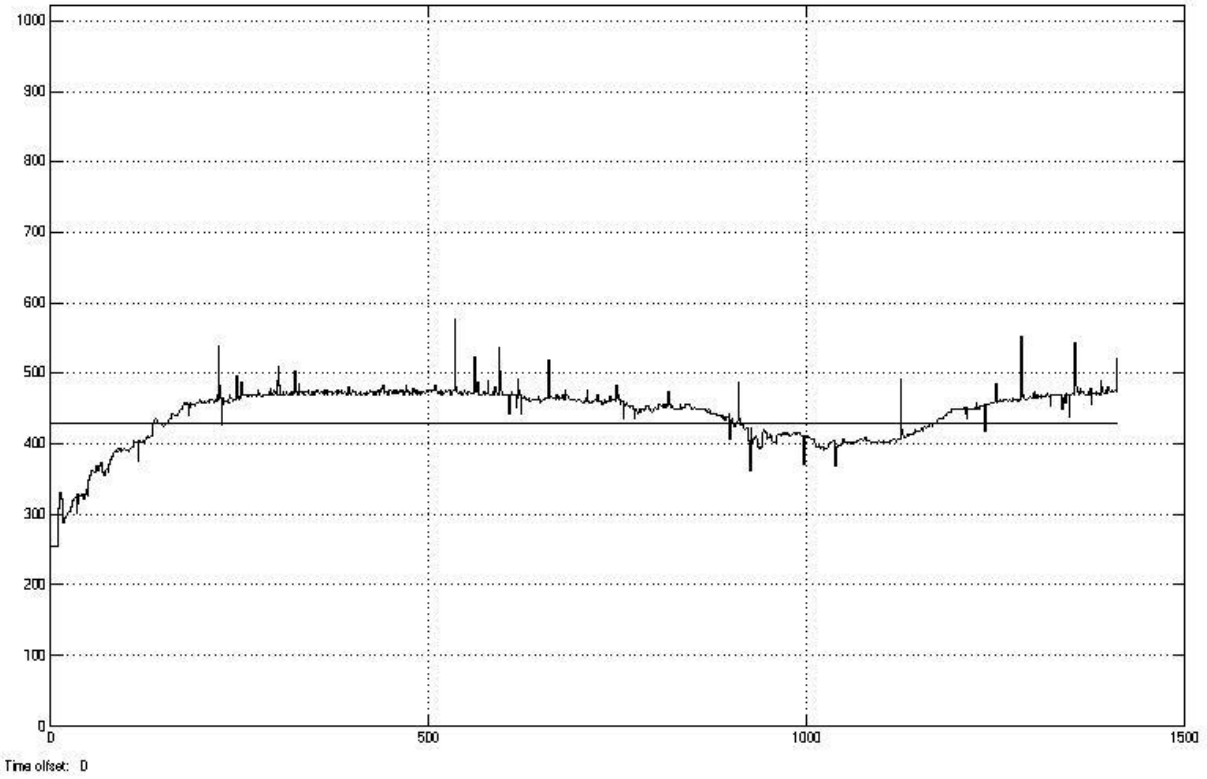

Figura 5.13 – Controle da planta real utilizando-se SGPC

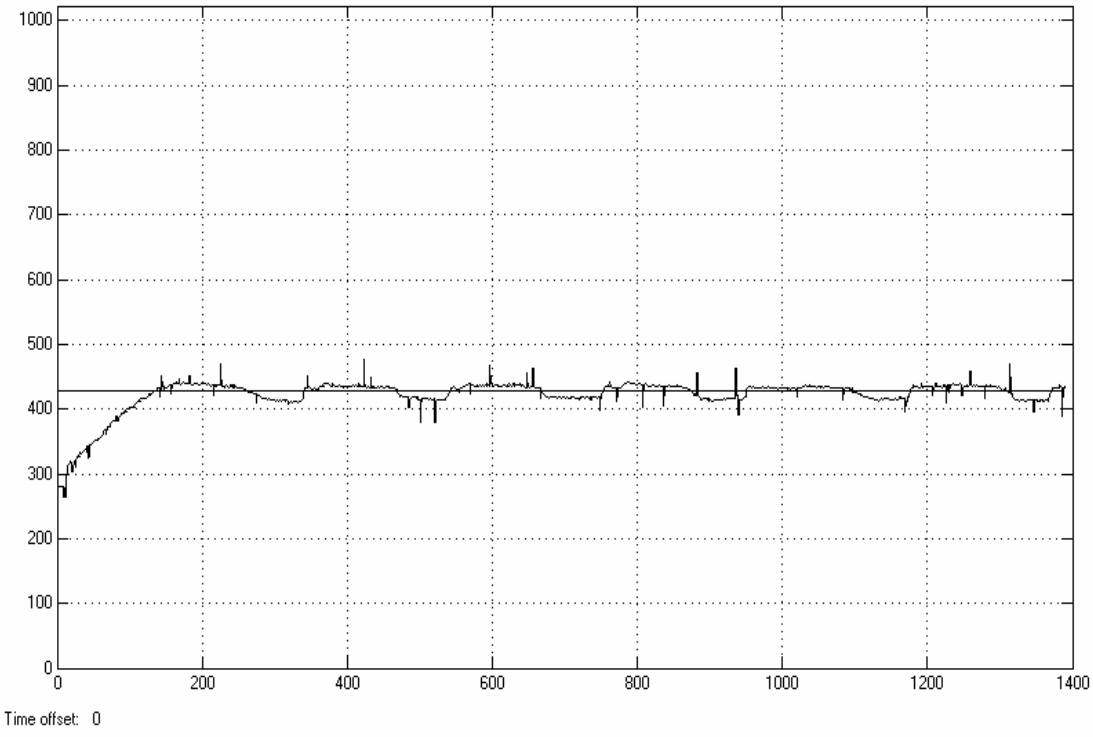

Figura 5.14 – Controle da planta real utilizando-se SPGPC

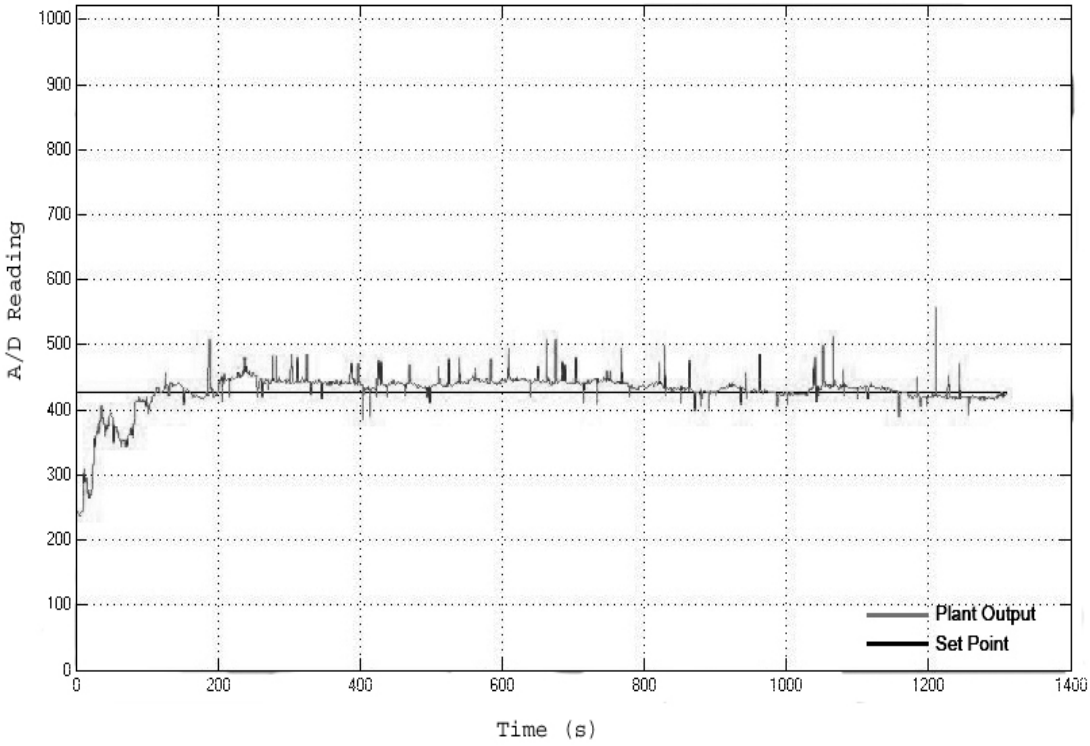

Figura 5.15 – Controle da planta real utilizando-se GPC

5.5 CONSIDERAÇÕES FINAIS.

A desempenho de três algoritmos preditivos foi comparada neste capítulo para utilização em um controle de condutividade. O SGPC não possui publicações com aplicação em planta real exceto o mostrado no presente trabalho[19], mostrando um desempenho não satisfatório para tal aplicação. O melhor algoritmo mostrado, com relação aos parâmetros a serem atingidos, foi o SPGPC. Porém o GPC apresentou desempenho similar, além de possuir esforço computacional inferior ao primeiro. Este fato é de grande valia quando se implementar o processo em um microcontrolador, o qual possui recursos limitados. Além disso, o GPC pode trabalhar sem grandes mudanças de performance em uma planta identificada *off-line*, enquanto o SPGPC necessita de um identificador *on-line* para sua execução. Esse fato se torna importante no caso da escolha de um microcontrolador mais barato e com menos recursos de memória.

CONCLUSÃO GERAL

Este trabalho apresentou a aplicação de três algoritmos preditivos para controle de condutividade em impressões *off-set*. Este tipo de controle tem grande importância em impressões de jornais e periódicos, uma vez que afeta diretamente na qualidade da impressão e na troca de tintas, este último fator determinante na redução de custos. A ausência de controladores aplicados a condutividade em líquidos e o alto custo de controladores de vazão de “agente de molha” (líquido que altera a condutividade da água) foram a principal motivação deste trabalho.

O sistema foi composto de um sensor de condutividade, dois motores de corrente contínua utilizados em sucção de líquido, um sistema de aquisição e processamento baseado em um microcontrolador PIC18F452 da Microchip e um circuito de acionamento para os motores supracitados. Os algoritmos de controle e identificação foram implementados dentro do microcontrolador foi colocado um algoritmo de identificação de sistemas baseado no método dos mínimos quadrados recursivo, para que as mudanças nos parâmetros da planta pudessem ser detectadas no caso de desgaste dos atuadores, problemas de precisão do sensor ou mudanças climáticas. Neste mesmo microcontrolador foram colocados os algoritmos preditivos, a saber, o “Generalised Predictive Control” (GPC), o “Smith Predictor Generalised Predictive Control” (SPGPC) e o “Stable Generalised Preditive Control” (SGPC).

Para uma análise mais apurada, utilizou-se a ferramenta matemática Matlab® para a identificação da planta e análise dos algoritmos na planta modelada. Para isso foi desenvolvido uma *interface* entre o Matlab e a planta real, de modo que o processamento matemático fosse efetuado dentro da ferramenta matemática. Um programa, funcionamento como terminal burro, foi alocado dentro do microcontrolador, de modo que este recebia os dados do Matlab e efetuava o acionamento dos atuadores via PWM, sendo que logo em

seguida recebia os dados da leitura do canal analógico-digital provenientes do sensor de condutividade.

Utilizando-se da ferramenta supracitada, os parâmetros da planta foram calculados por uma identificação "*on-line"*. De posse destes dados, aplicou-se os três controladores em uma simulação para efeito de comparação. O SGPC apresentou a melhor performance em regime permanente, porém seu transitório apresentou um sobressinal muito maior que os outros dois algoritmos. Apesar de a simulação ter sido feita aplicando-se ruído colorido de segunda ordem, o SPGPC e o GPC não apresentaram resultados muito diferentes, o que era de se esperar no caso de ausência de ruídos ou distúrbios. No caso do controle "*off-line"*, o algoritmo de identificação foi colocado em conjunto com o algoritmo de controle, de modo a simular uma situação real.

Fazendo uso do mesmo processo utilizado para a identificação dos parâmetros da planta, os algoritmos de controle foram colocados na planta real. Atentando para o fato de o SGPC ter obtido um grande sobressinal na simulação, este não conseguiu obter um controle satisfatório da condutividade, obtendo um erro com relação a referência de 15%. Isto se deveu ao fato deste algoritmo saturar rapidamente os atuadores. O GPC apresentou erro de 3%, porém mostrou-se um processo bastante ruidoso devido ao fato do acionamento contínuo nos atuadores. A melhor performance foi obtida pelo SPGPC, conseguindo um erro de regime de 1% e sem a presença de ruídos de amplitude alta, por fazer baixa utilização dos atuadores.

A aplicação comercial deste sistema é de grande valia, uma vez que jornais de médio e pequeno porte não possuem recursos suficientes para a aquisição de um controlador de vazão, geralmente importados. O controlador proposto possui baixo custo e eficiência comprovada, além de instalação e manuseio simples.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] M.A.Lelic and P.E. Wellstead. Generalized Pole Placement Self Tuning Controller. Part 2. Application to Robot Manipulator Control. Int. J. Control, 46(2):569-601, 1987.

[2] D.A. Linkers and M. Mahfonf. Advances in Model-Based Predictive Control, chapter Generalized Predictive Control in Clinical Anaesthesia. Oxford University Press, 1994.

[3] B. Kouvaritakis, J.A. Rossiter and A.O.T. Chang. Stable Generalized Predictive Control: An Algorithm with Guaranteed Stability. Proceedings IEE, Part D, 139(4):349-332, july 1991.

[4] D.W. Clarke, C. Mohtadi, and P.S. Tuffs. Generalized Preditive Control. Part I. The Basic Algorithm. Automatica, 23(2):149-160, 1987.

[5] J.E. Normey-Rico and E.F. Camacho. A Smith Predictor Based Generalized Predictive Controller. Internal Report GAR 1996/02. Universidad de Sevilla, 1996.

[6] Kouvaritakis, B., Rossiter, J.A. and Chang, A.O.T. Stable Generalised Predictive Control:An algorithm with guaranteed stability. Proceedings IEE, Part D, 139(4):349-362, 1992.

[7] Rossiter, J.A., Kouvaritakis, B. Numerical robustness and efficiency of generalised predictive control algorithms with guaranteed stability. Proceedings IEE, Control Theory Appl., Vol. 141, No. 3, 154-162, 1994.

[8] J.E. Normey-Rico, C. Bordons, and E.F. Camacho. Improving the Robustness of Dead-time Compensating PI Controllers. Control Engineering Practice, 5(6):801-810, 1997.

[9] Richalet, J.A., A. Rault, J.D. Testud and J. Papon, "Model Predictive Heuristic Control: Applications to Industrial Processes," Automatica, 14, 413-428 (1978).

[10] Camacho,E. F. and Bordons, Carlos “Model Predictive Control”, 1999, Springer-Verlag Berlin and Heidelberg GmbH & Co.k.

[11] J.M Martin-Sanchez and J. Rodellar. Adaptive Predictive Control. From the concepts to plant optimization. Prentice-Hall International(UK), 1996.

[12] O.J.M. Smith. Close Control of Loops with Deadtime. Chem. Eng. Prog., 53(5):217, 1957.

[13] Gauss, K.G. Theory of Motion of the Heavenly Bodies. New York: Dover. 1963

[14] Aguirre, L.A.; “Introdução à Identificação de Sistemas – Técnicas Lineares e Não-Lineares Apliacadas a Sistemas Reais”, 2002, UFMG publishing.

[15] Chen, Chi-Tsong. Linear System Theory and Design, 3rd Edition, 1999, Oxford University Press.

[16] Microchip; “PIC18FXX2 28/40-pin High Performance, Enhanced FLASH Microcontrollers with 10-Bit A/D”; Data Sheet – 2004

[17] Barbi, Ivo. “Eletrônica de Potência: Projetos de Fontes Chaveadas”, 2001, Edição do Autor.

[18] Diniz, Eber de Castro, Barreto L. H. S. C. ;. A new predictive control of water conductivity using a microcontroller applied to a off-set printing.. Congresso Brasileiro de Eletrônica de Potencia - COBEP 2005, 2005. v. 1. p. 429-432.

[19] Diniz, Eber de Castro, Almeida, O. M., Barreto, L. H. S. C.;. Utilização de um Controlador Preditivo com Estabilidade Garantida para Controle de Condutividade em Impressões Offset – VII SBAI, 2005. São Luís.

[20] Kuo, BenjaMin C., Digital Control Systems, 2nd edition. Oxford University Press.

[21] North Carolina Department of Environment and Natural Resources. Division of Pollution Prevention and Environmental Assistance. Publication

[22] Klaus Walther, “Tudo Sobre Molhagem em Impressão Offset”, Material publicitário.

[23] Diniz, Eber de Castro, Almeida, O. M., Barreto, L. H. S. C.;. A new predictive control of water conductivity with dead-time using a microcontroller applied to a off-set printing. Induscon 2006,2006. Recife

[24] Quin, S.J., Badgwell, T.A.. An Overview of Industrial Model Predictive Control Technology. In Chemical Process Control: Assessment and New Directions for Research. AIChE Symposium Series 316, 93. Jeffrey C. Kantor, Carlos E. Garciam Brice Carnahan. Fds. 232-256, 1997.

[25] Normey-Rico, Julio E., Camacho, Eduardo F., Gómez-Ortega, Juan. Robustez e Predição em Conotroladores Preditivos Generalizados. Em Proceedings of XII Brazilian Automatic Control Conference – XII CBA. Vol. I, pp. 157-162. Setembro 1998.

APÊNDICE I

S-FUNCTIONS

I.1 O QUE SÃO S-FUNCTIONS

S-functions podem ser utilizadas para adicionar blocos personalizados em um modelo do Simulink. Uma *S-function* é a descrição, em linguagem de programação, de um bloco do Simulink escrito em Matlab, C, C++, Fortran ou Ada. A forma de uma S-function é bem geral e pode ser utilizada.em sistemas contínuos, discretos ou híbridos.

I.2 COMO UMA S-FUNCTION FUNCIONA

A execução de um modelo do Simulink é processado em estágios. Uma S-function compreende a um número de métodos chamados recursivamente que fazem uma determinada tarefa em cada estágio da simulação. Durante a simulação de um modelo, em cada estágio, o Simulink faz a chamada de um determinado método para cada bloco que possua uma S-function. A tarefa executada por cada método inclui:

- Iniciação:: Neste estágio, o Simulink inicia a S-function. Isto inclui a iniciação da estrutura de simulação, determinação do número de entrada e saídas do sistema, o tempo de amostragem do bloco, alocação das áreas de armazenamento da memória e tamanho das estruturas(matrizes).
- Cálculo da variável de tempo do passo seguinte: Neste estágio ocorre o cálculo do tempo do passo subsqüente se um bloco de tempo de amostragem tenha sido criado.
- Cálculo das saídas: Depois que a chamada deste método tenha sido completada, todas os portos de saída dos blocos são validados para o tempo de amostragem atual.

- Atualização das variáveis de estado discretas: Nesta chamada o bloco deve executar os procedimentos passo-a-passo pertecentes a cada algoritmo alocado em seu respectivo bloco.

I.3 IMPLEMENTANDO S-FUNCTIONS.

Uma S-function pode ser implementada ou como um arquivo do Matlab(.m) ou um arquivo MEX. Como o corrente trabalho utilizou-se apenas de arquivos do Matlab, é dele que vamos nos ater nesta explanação. Um arquivo .m de uma S-function consiste em uma função do Matlab da seguinte forma:

[*sys,x0,str,ts]=f(t,x,u,flag,p1,p2,...)*

na qual *f* é o nome da S-function, t é o tempo atual, x é o vetor de espaço de estados correspondente ao bloco da S-function, *u* representa as entradas deste mesmo bloco, *flag* indica qual das funções da S-function será executada e *p1, p2,...* são os parâmetros adicionais definidos pelo usuário. Durante a simulação o Simulink invoca repetidamente a S-function f, utilizando a flag para indicar qual função dentro da S-function será executada. A cada tempo de amostragem a S-function retorna os parâmetros no seguinte formato:

[*sys,x0,str,ts]=f(t,x,u,flag,p1,p2,...)*

Um modelo de implementação de um arquivo .m dentro de uma S-function está localizada no diretório *toolbox>simulink>blocks* dentro da pasta de instalação do Matlab. Este modelo consiste em uma função principal e a estrutura das subfunções, cada uma destas fazendo referência a variável flag. Estas subfunções, as quais são chamadas de métodos recursivos, executam uma determinada tarefa dentro da S-function durante a simulação.

I.4 ESTRUTURA DAS S-FUNCTIONS EM UM ARQUIVO DO MATLAB.

O exemplo a seguir cria um bloco S-function que recebe uma entrada e mostra a saída multiplicada por três. O código do arquivo .m é mostrado a seguir:

```
    function [sys,x0,str,ts] = vezestres(t,x,u,flag)
    % Seleção do flag. Seleciona a função a ser executada a
%cada estágio da simulação.
    switch flag,
    case 0
    [sys,x0,str,ts] = mdlInitializeSizes; % Iniciação
    case 3
    sys = mdlOutputs(t,x,u); % Cálculo das saídas
    case { 1, 2, 4, 9 }
    sys = []; % Flags não utilizadas
    otherwise
    % Tratamento de erros
    error(['Unhandled flag = ',num2str(flag)]);
    end;
    % Fim da função vezestres
    % Função mdlInitializeSizes coloca os valores iniciais dos
% estados, tempos de amostragem e o tamanho das
    % estruturas(variáveis) a serem utilizadas

    function [sys,x0,str,ts] = mdlInitializeSizes
    % Chama a função simsizes para que retorna o tamanho das
    % estruturas
    sizes = simsizes;
    % Carrega os tamanhos das estruturas nas variáveis de
```

```
% inciação
sizes.NumContStates= 0;
sizes.NumDiscStates= 0;
sizes.NumOutputs= 1;
sizes.NumInputs= 1;
sizes.DirFeedthrough=1;
sizes.NumSampleTimes=1;
% Carrega o vetor sys acerca do tamanho das variáveis da
% s-function
sys = simsizes(sizes);
%
x0 = []; % Não possui espaço de estados contínuos
%
str = []; % Sem ordem de estados
%
ts = [-1 0]; % Herda o tempo de amostragem da simulação.
% Fim da função mdlInitializeSizes.

% Função mdlOutputs faz os cálculos.
function sys = mdlOutputs(t,x,u)
sys = 2*u;
% Fim da função mdlOutputs.
```

Descrevemos agora o arquivo S-function para melhor entendimento:

1. A primeira linha especifica os argumentos de entrada e saída.

```
function [sys,x0,str,ts] = timesthree(t,x,u,flag)
```

2. Depois da primeira linha, o arquivo S-function é dividido em diferentes casos(ou subfunções) determinados pela variável flag. Como mostrado no código, apenas considerou-se no exemplo os casos 1 e 3. Para os casos 1,2 4 e 9 simplesmente fizemos sys=[]. As últimas duas linhas da função vezestres fazem o tratamento de erro da mesma durante a simulação.

3. Para o caso 0 (iniciação), a função *mdlInitializeSizes* faz a chamada do comando *simsizes*.Sem argumentos, *simsizes* cria uma estrutura de variáveis de pode ser preenchida com os valores iniciais.

```
sizes = simsizes;
sizes.NumContStates= 0;
sizes.NumDiscStates= 0;
sizes.NumOutputs= 1;
sizes.NumInputs= 1;
sizes.DirFeedthrough=1;
sizes.NumSampleTimes=1;
```

Utilizando-se o commando *simsizes* novamente transforma a variável sizes em um vetor linha que será retornado para o Simulink via *sys*(saída do sistema):

```
sys = simsizes(sizes);
```

Então as varáveis de estados (x0), as strings de ordem de estado (str) e os tempos de amostragem (ts), são iniciados.

```
x0 = []; % No continuous states
str = []; % No state ordering
```

```
ts = [-1 0]; % Inherited sample time
```

4. Para o caso 3 (que retorna os parâmetro calculados para o sistema), a função mdlOutputs executa os cálculos.

```
sys = 3*u;
```

APÊNDICE II

ALGORITMOS

Os algoritmos utilizados pelas s-functions descritas nos Capítulos 4 e 5 são mostradas aqui.

predictiveControllerGPCRLSMisoAgua.m

```
function  [sys, x0, str, ts]  = redictiveControllerGPCRLSMISOagua(t,x,u,flag,prediction,num,den,ffactor)
global outputagua;
global gagua;
global fpcagua;
global wagua;
global unAntpcagua;
global numagua;
global numagente;
global denboth;
if abs(flag) == 2
    wagua=u(1)*diag(eye(prediction));
elseif flag == 3
   %fpc=u(2:prediction+1)
   %untAnt=u(prediction+2);
%    b=[num zeros(1,prediction)];%[4.86 -4.62 zeros(1,prediction)];
%    Size=size(den);
%       a=[den(2:Size(2)) zeros(1,prediction)];%[-.819 -.11 zeros(1,prediction)]; %1 is supressed in the denominator
    b=numagua;
    Size=size(denboth);
    a=[denboth(2:Size(2)) zeros(1,prediction)];
   %Qdo o identificador estiver online
   if ((a(2) > 1.5) || (a(2) < .5));
     b=[num 0 0 0];
     Size=size(den);
     a=[den(2:Size(2)) 0 0 0];
   end;
   %fim Qdo o identificador estiver online
    gagua(1)=b(1);
    for J=2:prediction; %prediction steps
       gagua(J)=0;
       for I=1:J-1;
          gagua(J) = gagua(J) - a(I)*gagua(J-I);
       end
       for I=1:J;
          gagua(J) = gagua(J) + b(I);
       end
    end
    for J=1:prediction; %prediction steps
       G(:,J)=[zeros(1,J-1)';gagua(1:prediction-J+1)'];
    end
    G=inv(G'*G+eye(prediction)*ffactor)*G';
    G=G(1,:);
    outputagua=[G*(wagua-fpcagua)]+unAntpcagua;
    sys=outputagua;
```

```
elseif flag == 4
  sys = [];
elseif flag == 0
  % Return sizes of parameters and initial conditions
  sys(1) = 0;                % 0 continuous states
  sys(2) = 0;                % 2*order discrete states (num and den)
  sys(3) = 1;                % 2*order outputs (num and den)
  sys(4) = prediction+2;          % 2*order inputs (num and den)
  sys(5) = 0;                % no roots
  sys(6) = 0;                % no direct feedthrough
  sys(7) = 1;                % 1 sample time
  outputagua = 0;
  gagua=zeros(1,prediction);
  fpcagua=zeros(1,prediction)';
  wagua=zeros(1,prediction)';
  unAntpcagua=0;
  numagente=[0 0 0 0 0 0 0];
  numagua=[0 0 0 0 0 0 0];
  denboth=[0 0 0 0 0 0 0];
  x0 = [];
  ts = [-1, 0];
else
  % all other flags, return an empty matrix
  sys = [];
end
```

predictiveControllerGPCRLSMisoAgente.m

```
function [sys, x0, str, ts] = predictiveControllerGPCRLSMISOagente(t,x,u,flag,prediction,num,den,ffactor)
global outputagente;
global gagente;
global fpcagente;
global wagente;
global unAntpcagente;
global wagente;
global numagua;
global numagente;
global denboth;
if abs(flag) == 2
    wagente=u(1)*diag(eye(prediction));
elseif flag == 3
   %fpc=u(2:prediction+1)
   %untAnt=u(prediction+2);
%   b=[num zeros(1,prediction)];%[4.86 -4.62 zeros(1,prediction)];
%   Size=size(den);
%      a=[den(2:Size(2)) zeros(1,prediction)];%[-.819 -.11 zeros(1,prediction)]; %1 is supressed in the denominator
    b=numagente;
    Size=size(denboth);
    a=[denboth(2:Size(2)) zeros(1,prediction)]/2;
   %Qdo o identificador estiver online
   if ((a(2) > 1.5) || (a(2) < .5));
     b=[num 0 0 0];
     Size=size(den);
     a=[den(2:Size(2)) 0 0 0];
   end;
```

```
   %fim Qdo o identificador estiver online
    gagente(1)=b(1);
    for J=2:prediction; %prediction steps
       gagente(J)=0;
       for I=1:J-1;
          gagente(J) = gagente(J) - a(I)*gagente(J-I);
       end
       for I=1:J;
          gagente(J) = gagente(J) + b(I);
       end
    end
    for J=1:prediction; %prediction steps
       G(:,J)=[zeros(1,J-1)';gagente(1:prediction-J+1)'];
    end
    G=inv(G'*G+eye(prediction)*ffactor)*G';
    G=G(1,:);
    outputagente=[G*(wagente-fpcagente)]+unAntpcagente;
    sys=outputagente;
elseif flag == 4
  sys = [];
elseif flag == 0
  % Return sizes of parameters and initial conditions
  sys(1) = 0;                   % 0 continuous states
  sys(2) = 0;                   % 2*order discrete states (num and den)
  sys(3) = 1;                   % 2*order outputs (num and den)
  sys(4) = prediction+2;            % 2*order inputs (num and den)
  sys(5) = 0;                   % no roots
  sys(6) = 0;                   % no direct feedthrough
  sys(7) = 1;                   % 1 sample time
  outputagente = 0;
  gagente=zeros(1,prediction);
  fpcagente=zeros(1,prediction)';
  wagente=zeros(1,prediction)';
  unAntpcagente=0;
  %Qdo o identificador estiver online
  %b=[num 0 0 0];
  %Size=size(den);
  %a=[den(2:Size(2)) 0 0 0];
  %fim Qdo o identificador estiver online
  numagente=[0 0 0 0 0 0 0];
  numagua=[0 0 0 0 0 0 0];
  denboth=[0 0 0 0 0 0 0];
  x0 = [];
  ts = [-1, 0];
else
  % all other flags, return an empty matrix
  sys = [];
end
```

freeResponseGPCRLSMisoAgua.m

```
function  [sys, x0, str, ts]  = freeResponseGPCRLSMISOagua(t,x,u,flag,prediction,num,den)
global ynAntagua;
global unAtualagua;
global unAtualagente;
global ynagua;
global fagua;
```

```
        global unAntagua;
        global unAntagente;
        global fpcagua;
        global unAntpcagua;
        global numagua;
        global numagente;
        global denboth;
        if abs(flag) == 2
        % sample hit, return the next discrete states
          unAntagua=u(1:3);
          ynAntagua=u(4:6);
          unAtualagua=u(7);
        %   yn=u(8);
          fagua(1)=u(8);
        elseif flag == 3
        %    b=[num zeros(1,prediction)];%[4.86 -4.62 zeros(1,prediction)];
        %    Size=size(den);
        %       a=[den(2:Size(2)) zeros(1,prediction)];%[-.819 -.11 zeros(1,prediction)]; %1 is supressed in the
denominator
          b=numagua;
          Size=size(denboth);
          a=[denboth(2:Size(2)) zeros(1,prediction)];
          %Qdo o identificador estiver online
          if ((a(2) > 1.5) || (a(2) < .5));
            b=[num 0 0 0];
            Size=size(den);
            a=[den(2:Size(2)) 0 0 0];
          end;
          %fim Qdo o identificador estiver online
           ynagente=fagua(1)
           ynagua= 2*(-a(1)*ynAntagua(1) - a(2)*ynAntagua(2))  +  b(2)*unAtualagua + b(3)*unAntagua(1) +
b(2)*unAtualagente + b(3)*unAntagente(1);
           ynagua=ynagua*0.5;
           fagua(1)=ynagua;
        %     fagua(2)= (1-a(1))*ynagua + (a(1)-a(2))*ynAntagua(1) + a(2)*ynAntagua(2) + b(3)*unAntagua(1) -
b(3)*unAntagua(2);
          fagua(2)=(1-a(1))*ynagua  +  (a(1)-a(2))*ynAntagua(1)  +  a(2)*ynAntagua(2)  +  b(3)*unAtualagua  -
b(3)*unAntagua(1);
          fagua(3)=(1-a(1))*fagua(2) + (a(1)-a(2))*fagua(1) +  a(2)*ynAntagua(1);
          for I=4:prediction+1;
            fagua(I)=(1-a(1))*fagua(I-1) + (a(1)-a(2))*fagua(I-2) +  a(2)*fagua(I-3);
          end
          I=I+1;
          fagua(I)=unAntagua(1)
          fpcagua=[fagua(2:prediction+1)]';
        %   unAntpc=unAnt(1);
          unAntpcagua=unAtualagua;
          sys=[fagua(2:prediction+2)];
        elseif flag == 4
          sys = [];
        elseif flag == 0
          % Return sizes of parameters and initial conditions
          sys(1) = 0;                   % 0 continuous states
          sys(2) = 0;                   % 2*order discrete states (num and den)
          sys(3) = prediction+1;                % 2*order outputs (num and den)
          sys(4) = 8;                   % 2*order inputs (num and den)
          sys(5) = 0;                   % no roots
          sys(6) = 0;                   % no direct feedthrough
```

```
  sys(7) = 1;                    % 1 sample time
  ynAntagua=zeros(1,3);
  unAntagua=zeros(1,3);
  fagua=zeros(1,prediction+2);
  numagente=[0 0 0 0 0 0 0];
  numagua=[0 0 0 0 0 0 0];
  denboth=[0 0 0 0 0 0 0];
  unAtualagua=0;
  ynagua=0;
  x0 = [];
  ts = [-1, 0];
else
  % all other flags, return an empty matrix
  sys = [];
end
```

freeResponseGPCRLSMisoAgente.m

```
function  [sys, x0, str, ts]  = freeResponseGPCRLSMISOagente(t,x,u,flag,prediction,num,den)
global ynAntagente;
global unAtualagente;
global unAtualagua;
global ynagente;
global fagente;
global unAntagente;
global unAntagua;
global fpcagente;
global unAntpcagente;
global numagua;
global numagente;
global denboth;
if abs(flag) == 2
% sample hit, return the next discrete states
  unAntagente=u(1:3);
  ynAntagente=u(4:6);
  unAtualagente=u(7);
%   yn=u(8);
  fagente(1)=u(8);
elseif flag == 3
%   b=[num zeros(1,prediction)];%[4.86 -4.62 zeros(1,prediction)];
%   Size=size(den);
%     a=[den(2:Size(2)) zeros(1,prediction)];%[-.819 -.11 zeros(1,prediction)]; %1 is supressed in the
denominator
   b=numagente;
   Size=size(denboth);
   a=[denboth(2:Size(2)) zeros(1,prediction)];
  %Qdo o identificador estiver online
  if ((a(2) > 1.5) || (a(2) < .5));
    b=[num 0 0 0];
    Size=size(den);
    a=[den(2:Size(2)) 0 0 0];
  end;
  %fim Qdo o identificador estiver online
  ynagente=fagente(1)
```

```
        ynagente= 2*(-a(1)*ynAntagente(1) - a(2)*ynAntagente(2)) + b(2)*unAtualagente + b(3)*unAntagente(1) + b(2)*unAtualagua + b(3)*unAntagua(1);
        ynagente=ynagente * 0.5;
        fagente(1)=ynagente;
      % fagente(2)= (1-a(1))*ynagente + (a(1)-a(2))*ynAntagente(1) + a(2)*ynAntagente(2) + b(3)*unAntagente(1) - b(3)*unAntagente(2);
        fagente(2)= (1-a(1))*ynagente + (a(1)-a(2))*ynAntagente(1) + a(2)*ynAntagente(2) + b(3)*unAtualagente - b(3)*unAntagente(1);
        fagente(3)=(1-a(1))*fagente(2) + (a(1)-a(2))*fagente(1) + a(2)*ynAntagente(1);
        for I=4:prediction+1;
          fagente(I)=(1-a(1))*fagente(I-1) + (a(1)-a(2))*fagente(I-2) + a(2)*fagente(I-3);
        end
        I=I+1;
        fagente(I)=unAntagente(1)
        fpcagente=[fagente(2:prediction+1)]';
      % unAntpc=unAnt(1);
        unAntpcagente=unAtualagente;
        sys=[fagente(2:prediction+2)];
      elseif flag == 4
       sys = [];
      elseif flag == 0
       % Return sizes of parameters and initial conditions
       sys(1) = 0;                % 0 continuous states
       sys(2) = 0;                % 2*order discrete states (num and den)
       sys(3) = prediction+1;             % 2*order outputs (num and den)
       sys(4) = 8;                % 2*order inputs (num and den)
       sys(5) = 0;                % no roots
       sys(6) = 0;                % no direct feedthrough
       sys(7) = 1;                % 1 sample time
       ynAntagente=zeros(1,3);
       unAntagente=zeros(1,3);
       fagente=zeros(1,prediction+2);
       numagente=[0 0 0 0 0 0 0];
       numagua=[0 0 0 0 0 0 0];
       denboth=[0 0 0 0 0 0 0];
       unAtualagente=0;
       ynagente=0;
       x0 = [];
       ts = [-1, 0];
      else
       % all other flags, return an empty matrix
       sys = [];
      end
```

freeResponseGPCRLSSmithMisoAgua.m

```
      function [sys, x0, str, ts] = freeResponseGPCRLSSmithMisoAgua (t,x,u,flag,prediction,num,den,deadtime)
      global a;
      global b;
      global ynAnt;
      global unAtual;
      global yn;
      global f;
      global unAnt;
```

```
global fpc;
global unAntpc;
global ynNoDelay;
global ynAntModel;
if abs(flag) == 2
% sample hit, return the next discrete states
  unAnt=u(1:4);
  ynAnt=u(6:8);
  unAtual=u(9);
  yn=u(10);
elseif flag == 3
%   b=[num zeros(1,prediction)];%[4.86 -4.62 zeros(1,prediction)];
%   Size=size(den);
%     a=[den(2:Size(2)) zeros(1,prediction)];%[-.819 -.11 zeros(1,prediction)]; %1 is supressed in the denominator
%Qdo o identificador estiver online
   a(2)
   if ((a(2) > -0.01) || (a(2) < -1));
    b=[num 0 0 0];
    Size=size(den);
    a=[den(2:Size(2)) 0 0 0];
   end;
  %fim Qdo o identificador estiver online
 ynModel = -a(1)*ynAnt(1) - a(2)*ynAnt(2)  +  b(2)*unAnt(3) + b(3)*unAnt(4)
  f(1)
  ynNoDelay(3)=ynNoDelay(2);
  ynNoDelay(2)=ynNoDelay(1);
  ynNoDelay(1) = -a(1)*ynNoDelay(2) - a(2)*ynNoDelay(3)  +  b(2)*unAtual + b(3)*unAnt(1)
  %Preditor Smith
  %predicoes
  yp(1) = yn + ynNoDelay(1) - ynModel;
  yp(2) = ynAnt(1) + ynNoDelay(2) - ynAntModel(1);
  yp(3) =  ynAnt(2) + ynNoDelay(3) -  ynAntModel(2);
  f(1)=yp(1);
  f(2)= (1-a(1))*yp(1) + (a(1)-a(2))*yp(2) + a(2)*yp(3) + b(3)*unAtual - b(3)*unAnt(1);
  f(3)=(1-a(1))*f(2) + (a(1)-a(2))*f(1) +  a(2)*yp(1);
  %Preditor Otimo
  for I=4:prediction+4;
   f(I)=(1-a(1))*f(I-1) + (a(1)-a(2))*f(I-2) +  a(2)*f(I-3);
  end
  I=I+1;
  f(I)=unAnt(1);
  fpc=[f(1:prediction+deadtime)]';
%   unAntpc=unAnt(1);
  unAntpc=unAtual
  ynAntModel(4)=ynAntModel(3);
  ynAntModel(3)=ynAntModel(2);
  ynAntModel(2)=ynAntModel(1);
  ynAntModel(1)=ynModel;
  sys=[f(2:prediction+deadtime+2)];
elseif flag == 4
 sys = [];
elseif flag == 0
 % Return sizes of parameters and initial conditions
 sys(1) = 0;                    % 0 continuous states
 sys(2) = 0;                    % 2*order discrete states (num and den)
 sys(3) = prediction+deadtime+1;                % 2*order outputs (num and den)
 sys(4) = 10;                    % 2*order inputs (num and den)
```

```
  sys(5) = 0;                 % no roots
  sys(6) = 0;                 % no direct feedthrough
  sys(7) = 1;                 % 1 sample time
  ynAnt=zeros(1,5);
  unAnt=zeros(1,4);
  f=zeros(1,prediction+2);
  ynNoDelay=[0 0 0];
  ynAntModel=[0 0 0 0 0];
  unAtual=0;
  yn=0;
  x0 = [];
  ts = [-1, 0];
else
  % all other flags, return an empty matrix
  sys = [];
end
```

freeResponseGPCRLSSmithMisoAgente.m

```
function [sys, x0, str, ts] = freeResponseGPCRLSSmithMisoAgente
(t,x,u,flag,prediction,num,den,deadtime)
global a;
global b;
global ynAnt;
global unAtual;
global yn;
global f;
global unAnt;
global fpc;
global unAntpc;
global ynNoDelay;
global ynAntModel;
if abs(flag) == 2
% sample hit, return the next discrete states
  unAnt=u(1:4);
  ynAnt=u(6:8);
  unAtual=u(9);
  yn=u(10);
elseif flag == 3
%   b=[num zeros(1,prediction)];%[4.86 -4.62 zeros(1,prediction)];
%   Size=size(den);
%     a=[den(2:Size(2)) zeros(1,prediction)];%[-.819 -.11 zeros(1,prediction)]; %1 is supressed in the
denominator
%Qdo o identificador estiver online
   a(2)
   if ((a(2) > -0.01) || (a(2) < -1));
     b=[num 0 0 0];
     Size=size(den);
     a=[den(2:Size(2)) 0 0 0];
   end;
  %fim Qdo o identificador estiver online
 ynModel = -a(1)*ynAnt(1) - a(2)*ynAnt(2)  +  b(2)*unAnt(3) + b(3)*unAnt(4)
  f(1)
  ynNoDelay(3)=ynNoDelay(2);
  ynNoDelay(2)=ynNoDelay(1);
```

```
    ynNoDelay(1) = -a(1)*ynNoDelay(2) - a(2)*ynNoDelay(3)  +  b(2)*unAtual + b(3)*unAnt(1)
    %Preditor Smith
    %predicoes
    yp(1) = yn + ynNoDelay(1) - ynModel;
    yp(2) = ynAnt(1) + ynNoDelay(2) - ynAntModel(1);
    yp(3) =  ynAnt(2) + ynNoDelay(3) -  ynAntModel(2);
    f(1)=yp(1);
    f(2)= (1-a(1))*yp(1) + (a(1)-a(2))*yp(2) + a(2)*yp(3) + b(3)*unAtual - b(3)*unAnt(1);
    f(3)=(1-a(1))*f(2) + (a(1)-a(2))*f(1) +  a(2)*yp(1);
    %Preditor Otimo
    for I=4:prediction+4;
      f(I)=(1-a(1))*f(I-1) + (a(1)-a(2))*f(I-2) +  a(2)*f(I-3);
    end
    I=I+1;
    f(I)=unAnt(1);
    fpc=[f(1:prediction+deadtime)]';
  %   unAntpc=unAnt(1);
    unAntpc=unAtual
    ynAntModel(4)=ynAntModel(3);
    ynAntModel(3)=ynAntModel(2);
    ynAntModel(2)=ynAntModel(1);
    ynAntModel(1)=ynModel;
    sys=[f(2:prediction+deadtime+2)];
  elseif flag == 4
    sys = [];
  elseif flag == 0
    % Return sizes of parameters and initial conditions
    sys(1) = 0;                 % 0 continuous states
    sys(2) = 0;                 % 2*order discrete states (num and den)
    sys(3) = prediction+deadtime+1;                 % 2*order outputs (num and den)
    sys(4) = 10;                  % 2*order inputs (num and den)
    sys(5) = 0;                 % no roots
    sys(6) = 0;                 % no direct feedthrough
    sys(7) = 1;                 % 1 sample time
    ynAnt=zeros(1,5);
    unAnt=zeros(1,4);
    f=zeros(1,prediction+2);
    ynNoDelay=[0 0 0];
    ynAntModel=[0 0 0 0 0];
    unAtual=0;
    yn=0;
    x0 = [];
    ts = [-1, 0];
  else
    % all other flags, return an empty matrix
    sys = [];
  end
```

predictiveControllerGPCRLSSmithMisoAgua.m

```
function [sys, x0, str, ts] = predictiveControllerGPCRLSSmithMisoAgua
(t,x,u,flag,prediction,num,den,ffactor,deadtime)
global output;
global g;
global fpc;
```

```
global w;
global unAntpc;
global a;
global b;
if abs(flag) == 2
    w=u(1)*diag(eye(prediction+deadtime));
elseif flag == 3
   %fpc=u(2:prediction+1)
   %untAnt=u(prediction+2);
%    b=[num zeros(1,prediction)];%[4.86 -4.62 zeros(1,prediction)];
%    Size=size(den);
%       a=[den(2:Size(2)) zeros(1,prediction)];%[-.819 -.11 zeros(1,prediction)]; %1 is supressed in the
denominator
   prediction=prediction+deadtime;
   %Qdo o identificador estiver online
   if ((a(2) > -0.01) || (a(2) < -1));
     b=[num 0 0 0];
     Size=size(den);
     a=[den(2:Size(2)) 0 0 0];
   end;
  %fim Qdo o identificador estiver online
   %fpc=u(2:prediction+1)
   %untAnt=u(prediction+2);
   g(1)=b(1);
   for J=2:(prediction); %prediction steps
      g(J)=0;
      for I=1:J-1;
        g(J) = g(J) - a(I)*g(J-I);
      end
      for I=1:J;
        g(J) = g(J) + b(I);
      end
   end
   for J=1:prediction; %prediction steps
      G(:,J)=[zeros(1,J-1)';g(1:prediction-J+1)'];
   end
   G=inv(G'*G+eye(prediction)*ffactor)*G';
   G=G(1,:);
   output=[G*(w-fpc)]+unAntpc;
   if output < 0;
      output = 0;
   end
   if output > 600;
      output = 600;
   end
   sys=output;
elseif flag == 4
  sys = [];
elseif flag == 0
% Return sizes of parameters and initial conditions
  prediction=prediction+deadtime;
  sys(1) = 0;                  % 0 continuous states
  sys(2) = 0;                  % 2*order discrete states (num and den)
  sys(3) = 1;                  % 2*order outputs (num and den)
  sys(4) = prediction+2;            % 2*order inputs (num and den)
  sys(5) = 0;                  % no roots
  sys(6) = 0;                  % no direct feedthrough
  sys(7) = 1;                  % 1 sample time
```

```
  output = 0;
  g=zeros(1,prediction);
  fpc=zeros(1,prediction+deadtime)';
  w=zeros(1,prediction)';
  unAntpc=0;
  x0 = [];
  ts = [-1, 0];
else
  % all other flags, return an empty matrix
  sys = [];
end
```

predictiveControllerGPCRLSSmithMisoAgente.m

```
function [sys, x0, str, ts] = predictiveControllerGPCRLSSmithMisoAgente
(t,x,u,flag,prediction,num,den,ffactor,deadtime)
global output;
global g;
global fpc;
global w;
global unAntpc;
global a;
global b;
if abs(flag) == 2
    w=u(1)*diag(eye(prediction+deadtime));
elseif flag == 3
   %fpc=u(2:prediction+1)
   %untAnt=u(prediction+2);
%   b=[num zeros(1,prediction)];%[4.86 -4.62 zeros(1,prediction)];
%   Size=size(den);
%      a=[den(2:Size(2)) zeros(1,prediction)];%[-.819 -.11 zeros(1,prediction)]; %1 is supressed in the
denominator
   prediction=prediction+deadtime;
   %Qdo o identificador estiver online
   if ((a(2) > -0.01) || (a(2) < -1));
     b=[num 0 0 0];
     Size=size(den);
     a=[den(2:Size(2)) 0 0 0];
   end;
  %fim Qdo o identificador estiver online

   %fpc=u(2:prediction+1)
   %untAnt=u(prediction+2);
   g(1)=b(1);
   for J=2:(prediction); %prediction steps
      g(J)=0;
      for I=1:J-1;
        g(J) = g(J) - a(I)*g(J-I);
      end
      for I=1:J;
        g(J) = g(J) + b(I);
      end
   end
   for J=1:prediction; %prediction steps
      G(:,J)=[zeros(1,J-1)';g(1:prediction-J+1)'];
```

```
    end
    G=inv(G'*G+eye(prediction)*ffactor)*G';
    G=G(1,:);
    output=[G*(w-fpc)]+unAntpc;
    if output < 0;
       output = 0;
    end
    if output > 600;
       output = 600;
    end
    sys=output;
elseif flag == 4
  sys = [];
elseif flag == 0
% Return sizes of parameters and initial conditions
  prediction=prediction+deadtime;
  sys(1) = 0;                   % 0 continuous states
  sys(2) = 0;                   % 2*order discrete states (num and den)

  sys(3) = 1;                   % 2*order outputs (num and den)
  sys(4) = prediction+2;            % 2*order inputs (num and den)
  sys(5) = 0;                   % no roots
  sys(6) = 0;                   % no direct feedthrough
  sys(7) = 1;                   % 1 sample time
  output = 0;
  g=zeros(1,prediction);
  fpc=zeros(1,prediction+deadtime)';
  w=zeros(1,prediction)';
  unAntpc=0;
  x0 = [];
  ts = [-1, 0];
else
  % all other flags, return an empty matrix
  sys = [];
```

denominadorSGPCMisoAgua.m

```
function  [sys, x0, str, ts]  = denominadorSGPCMisoAgua (t,x,u,flag,n,sigma,mi,num,den)
global saidaagua;
global saidanumagua;
global saidadenagua;
global ynagua;
if abs(flag) == 2
    a=zeros(n);
    A=zeros(n);
    b=zeros(n);
    B=zeros(n);
    coefa=den;
    coefA=[coefa 0]-[0 coefa];
    coefb=num;
    coefB=[0 coefb];
sizecoefa=size(coefa);
sizecoefA=size(coefA);
sizecoefb=size(coefb);
sizecoefB=size(coefB);
```

```
degreeA=sizecoefa(2);
degreeB=sizecoefb(2);
coefa=coefa*antidiag(sizecoefa(2));
coefA=coefA*antidiag(sizecoefA(2));
coefB=coefB*antidiag(sizecoefb(2)+1);
coefb=coefb*antidiag(sizecoefb(2));
for I=1:n;
   if I<=sizecoefa(2);
      a(I,1:n)=[coefa((sizecoefa(2)+1-I):sizecoefa(2)) zeros(1,n-I)];
  else
      a(I,1:n)=[zeros(1,I-sizecoefa(2))  coefa(1:sizecoefa(2)) zeros(1,n-I)];
  end
end
for I=1:n;
   if I<=sizecoefA(2);
      A(I,1:n)=[coefA((sizecoefA(2)+1-I):sizecoefA(2)) zeros(1,n-I)];
  else
      A(I,1:n)=[zeros(1,I-sizecoefA(2))  coefA(1:sizecoefA(2)) zeros(1,n-I)];
  end
end
for I=1:n;
   if I<=sizecoefb(2);
      b(I,1:n)=[coefb((sizecoefb(2)+1-I):sizecoefb(2)) zeros(1,n-I)];
  else
      b(I,1:n)=[zeros(1,I-sizecoefb(2))  coefb(1:sizecoefb(2)) zeros(1,n-I)];
  end
end
for I=1:(n-1);
   if I<=sizecoefb(2);
      B(I+1,1:n)=[coefb((sizecoefb(2)+1-I):sizecoefb(2)) zeros(1,n-I)];
  else
      B(I+1,1:n)=[zeros(1,I-sizecoefb(2))  coefb(1:sizecoefb(2)) zeros(1,n-I)];
  end
end
sizeComps=degreeA+degreeB+1;
for I=1:(sizeComps);
   if I<=sizecoefa(2);
      LAcomp(I,1:sizeComps)=[coefA((sizecoefA(2)+1-I):sizecoefA(2)) zeros(1,sizeComps-I)];
  else
      LAcomp(I,1:sizeComps)=[zeros(1,I-sizecoefA(2))  coefA(1:sizecoefA(2)) zeros(1,sizeComps-I)];
  end
end
%for I=1:(sizeComps);
%    if I<=sizecoefb(2);
%       Lbcomp(I,1:sizeComps)=[coefb((sizecoefb(2)+1-I):sizecoefb(2)) zeros(1,sizeComps-I)];
%   else
%       Lbcomp(I,1:sizeComps)=[zeros(1,I-sizecoefb(2))  coefb(1:sizecoefb(2)) zeros(1,sizeComps-I)];
%   end
%end
for I=1:(sizeComps);
   if I<=sizecoefB(2);
      Lbcomp(I,1:sizeComps)=[coefB((sizecoefB(2)+1-I):sizecoefB(2)) zeros(1,sizeComps-I)];
  else
      Lbcomp(I,1:sizeComps)=[zeros(1,I-sizecoefB(2))  coefB(1:sizecoefB(2)) zeros(1,sizeComps-I)];
  end
end
CAcomp=LAcomp(1:sizeComps,1:degreeB+1);
Cbcomp=Lbcomp(1:sizeComps,1:degreeA);
```

```
coefa=coefa*antidiag(sizecoefa(2));
coefA=coefA*antidiag(sizecoefA(2));
coefB=coefB*antidiag(sizecoefb(2)+1);
coefb=coefb*antidiag(sizecoefb(2));
coefa=[coefa zeros(1,n-sizecoefa(2))];
coefA=[coefA zeros(1,n-sizecoefA(2))];
coefB=[coefB zeros(1,n-sizecoefb(2)-1)];
coefb=[coefb zeros(1,n-sizecoefb(2))];
La=a(1:n,1:mi);
LA=A(1:n,1:mi);
Lb=b(1:n,1:mi);
LB=B(1:n,1:mi);
Mb=b(1:n,mi+1:n);
MA=A(1:n,mi+1:n);
YX=inv([LA LB])*[1 zeros(1,n-1)]';
coefY=[YX(1:(n/2))]';
coefX=[YX((n/2+1):n)]';
sizecoefX=size(coefX);
coefX=coefX*antidiag(sizecoefX(2));
for I=1:(n);
   if I<=sizecoefX(2);
      X(I,1:n)=[coefX((sizecoefX(2)+1-I):sizecoefX(2)) zeros(1,n-I)];
  else
      X(I,1:n)=[zeros(1,I-sizecoefX(2))  coefX(1:sizecoefX(2)) zeros(1,n-I)];
  end
end
CXcomp=X(mi+1:n,1:n);
invertme=Lb'*Lb + sigma*LA'*LA; %right way
pr=[1 zeros(1,mi-1)]*inv(invertme)*(Lb' - Lb'*Mb*CXcomp - sigma*LA'*MA*CXcomp);
pc=[1 zeros(1,mi-1)]*inv(invertme)*(LB'*hankel(coefB) + sigma*LA'*hankel(coefA));
%pc=[1 zeros(1,mi-1)]*inv(invertme)*(Lb'*hankel(coefB) + sigma*LA'*hankel(coefA));
MN=inv([CAcomp Cbcomp])*[pc(1:(degreeA+1)) zeros(1,degreeB)]';
%MN=inv([CAcomp Cbcomp])*[pc(1:(sizecoefa(2)+1)) zeros(1,sizecoefb(2))]';
coefM=[MN(1:degreeA)]';
coefM=[coefM 0]-[0 coefM];
coefM=coefM(1:degreeA);
coefN=[MN((sizecoefa(2)+1):(degreeA+degreeB+1))]';
   saidadenagua=-((coefM(2:mi)*u(2:n/2))/coefM(1))+u(1)/coefM(1);
   a=saidadenagua
%    yn(1)=-[yn(2:5)]*coefa(2:5)'+[saidaden u(2:n/2-1)']*coefb(1:4)';
elseif flag == 3
   saidadenagua=saidadenagua
   sys=saidadenagua;
elseif flag == 4
 sys = [];
elseif flag == 0
 % Return sizes of parameters and initial conditions
 sys(1) = 0;                  % 0 continuous states
 sys(2) = 0;                  % 2*order discrete states (num and den)
 sys(3) = 1;                  % 2*order outputs (num and den)
 sys(4) = n+1;                   % 2*order inputs (num and den)
 sys(5) = 0;                  % no roots
 sys(6) = 0;                  % no direct feedthrough
 sys(7) = 1;                  % 1 sample time
 saidadenagua=0;
 outputagua = 0;
 ynagua=[0 0 0 0 0 0];
 x0 = [];
```

```
  ts = [-1, 0];
else
  % all other flags, return an empty matrix
  sys = [];
end
```

denominadorSGPCMisoAgente.m

```
function  [sys, x0, str, ts]  = denominadorSGPCMisoAgente (t,x,u,flag,n,sigma,mi,num,den)
global saidaagente;
global saidanumagente;
global saidadenagente;
global ynagente;
if abs(flag) == 2
    a=zeros(n);
    A=zeros(n);
    b=zeros(n);
    B=zeros(n);
    coefa=den;
    coefA=[coefa 0]-[0 coefa];
    coefb=num;
    coefB=[0 coefb];
sizecoefa=size(coefa);
sizecoefA=size(coefA);
sizecoefb=size(coefb);
sizecoefB=size(coefB);
degreeA=sizecoefa(2);
degreeB=sizecoefb(2);
coefa=coefa*antidiag(sizecoefa(2));
coefA=coefA*antidiag(sizecoefA(2));
coefB=coefB*antidiag(sizecoefb(2)+1);
coefb=coefb*antidiag(sizecoefb(2));
for I=1:n;
    if I<=sizecoefa(2);
       a(I,1:n)=[coefa((sizecoefa(2)+1-I):sizecoefa(2)) zeros(1,n-I)];
   else
       a(I,1:n)=[zeros(1,I-sizecoefa(2))  coefa(1:sizecoefa(2)) zeros(1,n-I)];
   end
end
for I=1:n;
    if I<=sizecoefA(2);
       A(I,1:n)=[coefA((sizecoefA(2)+1-I):sizecoefA(2)) zeros(1,n-I)];
   else
       A(I,1:n)=[zeros(1,I-sizecoefA(2))  coefA(1:sizecoefA(2)) zeros(1,n-I)];
   end
end
for I=1:n;
    if I<=sizecoefb(2);
       b(I,1:n)=[coefb((sizecoefb(2)+1-I):sizecoefb(2)) zeros(1,n-I)];
   else
       b(I,1:n)=[zeros(1,I-sizecoefb(2))  coefb(1:sizecoefb(2)) zeros(1,n-I)];
   end
end
for I=1:(n-1);
    if I<=sizecoefb(2);
```

```
      B(I+1,1:n)=[coefb((sizecoefb(2)+1-I):sizecoefb(2)) zeros(1,n-I)];
   else
      B(I+1,1:n)=[zeros(1,I-sizecoefb(2))  coefb(1:sizecoefb(2)) zeros(1,n-I)];
   end
end
sizeComps=degreeA+degreeB+1;
for I=1:(sizeComps);
    if I<=sizecoefa(2);
      LAcomp(I,1:sizeComps)=[coefA((sizecoefA(2)+1-I):sizecoefA(2)) zeros(1,sizeComps-I)];
   else
      LAcomp(I,1:sizeComps)=[zeros(1,I-sizecoefA(2))  coefA(1:sizecoefA(2)) zeros(1,sizeComps-I)];
   end
end
%for I=1:(sizeComps);
%    if I<=sizecoefb(2);
%      Lbcomp(I,1:sizeComps)=[coefb((sizecoefb(2)+1-I):sizecoefb(2)) zeros(1,sizeComps-I)];
%   else
%      Lbcomp(I,1:sizeComps)=[zeros(1,I-sizecoefb(2))  coefb(1:sizecoefb(2)) zeros(1,sizeComps-I)];
%   end
%end
for I=1:(sizeComps);
    if I<=sizecoefB(2);
      Lbcomp(I,1:sizeComps)=[coefB((sizecoefB(2)+1-I):sizecoefB(2)) zeros(1,sizeComps-I)];
   else
      Lbcomp(I,1:sizeComps)=[zeros(1,I-sizecoefB(2))  coefB(1:sizecoefB(2)) zeros(1,sizeComps-I)];
   end
end
CAcomp=LAcomp(1:sizeComps,1:degreeB+1);
Cbcomp=Lbcomp(1:sizeComps,1:degreeA);
coefa=coefa*antidiag(sizecoefa(2));
coefA=coefA*antidiag(sizecoefA(2));
coefB=coefB*antidiag(sizecoefb(2)+1);
coefb=coefb*antidiag(sizecoefb(2));
coefa=[coefa zeros(1,n-sizecoefa(2))];
coefA=[coefA zeros(1,n-sizecoefA(2))];
coefB=[coefB zeros(1,n-sizecoefb(2)-1)];
coefb=[coefb zeros(1,n-sizecoefb(2))];
La=a(1:n,1:mi);
LA=A(1:n,1:mi);
Lb=b(1:n,1:mi);
LB=B(1:n,1:mi);
Mb=b(1:n,mi+1:n);
MA=A(1:n,mi+1:n);
YX=inv([LA LB])*[1 zeros(1,n-1)]';
coefY=[YX(1:(n/2))]';
coefX=[YX((n/2+1):n)]';
sizecoefX=size(coefX);
coefX=coefX*antidiag(sizecoefX(2));
for I=1:(n);
    if I<=sizecoefX(2);
      X(I,1:n)=[coefX((sizecoefX(2)+1-I):sizecoefX(2)) zeros(1,n-I)];
   else
      X(I,1:n)=[zeros(1,I-sizecoefX(2))  coefX(1:sizecoefX(2)) zeros(1,n-I)];
   end
end
CXcomp=X(mi+1:n,1:n);
invertme=Lb'*Lb + sigma*LA'*LA; %right way
pr=[1 zeros(1,mi-1)]*inv(invertme)*(Lb' - Lb'*Mb*CXcomp - sigma*LA'*MA*CXcomp);
```

```
pc=[1 zeros(1,mi-1)]*inv(invertme)*(LB'*hankel(coefB) + sigma*LA'*hankel(coefA));
%pc=[1 zeros(1,mi-1)]*inv(invertme)*(Lb'*hankel(coefB) + sigma*LA'*hankel(coefA));
MN=inv([CAcomp Cbcomp])*[pc(1:(degreeA+1)) zeros(1,degreeB)]';
%MN=inv([CAcomp Cbcomp])*[pc(1:(sizecoefa(2)+1)) zeros(1,sizecoefb(2))]';
coefM=[MN(1:degreeA)]';
coefM=[coefM 0]-[0 coefM];
coefM=coefM(1:degreeA);
coefN=[MN((sizecoefa(2)+1):(degreeA+degreeB+1))]';
   saidadenagente=-((coefM(2:mi)*u(2:n/2))/coefM(1))+u(1)/coefM(1);
   a=saidadenagente
%    yn(1)=-[yn(2:5)]*coefa(2:5)'+[saidaden u(2:n/2-1)']*coefb(1:4)';
elseif flag == 3
   saidadenagente=saidadenagente
   sys=saidadenagente;
elseif flag == 4
  sys = [];
elseif flag == 0
  % Return sizes of parameters and initial conditions
  sys(1) = 0;                  % 0 continuous states
  sys(2) = 0;                  % 2*order discrete states (num and den)
  sys(3) = 1;                  % 2*order outputs (num and den)
  sys(4) = n+1;                   % 2*order inputs (num and den)
  sys(5) = 0;                  % no roots
  sys(6) = 0;                  % no direct feedthrough
  sys(7) = 1;                  % 1 sample time
  saidadenagente=0;
  outputagente = 0;
  ynagente=[0 0 0 0 0 0];
  x0 = [];
  ts = [-1, 0];
else
  % all other flags, return an empty matrix
  sys = [];
end
```

numeradorSGPCMisoAgua.m

```
function  [sys, x0, str, ts]  = numeradorSGPCMisoAgua (t,x,u,flag,n,sigma,mi,num,den)
global saidaagua;
global saidanumagua;
global saidadenagua;
global entradanumagua;
global coefN;
global ynagua;
if abs(flag) == 2
   a=zeros(n);
   A=zeros(n);
   b=zeros(n);
   B=zeros(n);
   coefa=den;
   coefA=[coefa 0]-[0 coefa];
   coefb=num;
   coefB=[0 coefb];
sizecoefa=size(coefa);
sizecoefA=size(coefA);
```

```
sizecoefb=size(coefb);
sizecoefB=size(coefB);
degreeA=sizecoefa(2);
degreeB=sizecoefb(2);
coefa=coefa*antidiag(sizecoefa(2));
coefA=coefA*antidiag(sizecoefA(2));
coefB=coefB*antidiag(sizecoefb(2)+1);
coefb=coefb*antidiag(sizecoefb(2));
for I=1:n;
   if I<=sizecoefa(2);
      a(I,1:n)=[coefa((sizecoefa(2)+1-I):sizecoefa(2)) zeros(1,n-I)];
  else
      a(I,1:n)=[zeros(1,I-sizecoefa(2))  coefa(1:sizecoefa(2)) zeros(1,n-I)];
  end
end
for I=1:n;
   if I<=sizecoefA(2);
      A(I,1:n)=[coefA((sizecoefA(2)+1-I):sizecoefA(2)) zeros(1,n-I)];
  else
      A(I,1:n)=[zeros(1,I-sizecoefA(2))  coefA(1:sizecoefA(2)) zeros(1,n-I)];
  end
end
for I=1:n;
   if I<=sizecoefb(2);
      b(I,1:n)=[coefb((sizecoefb(2)+1-I):sizecoefb(2)) zeros(1,n-I)];
  else
      b(I,1:n)=[zeros(1,I-sizecoefb(2))  coefb(1:sizecoefb(2)) zeros(1,n-I)];
  end
end
for I=1:(n-1);
   if I<=sizecoefb(2);
      B(I+1,1:n)=[coefb((sizecoefb(2)+1-I):sizecoefb(2)) zeros(1,n-I)];
  else
      B(I+1,1:n)=[zeros(1,I-sizecoefb(2))  coefb(1:sizecoefb(2)) zeros(1,n-I)];
  end
end
sizeComps=degreeA+degreeB+1;
for I=1:(sizeComps);
   if I<=sizecoefa(2);
      LAcomp(I,1:sizeComps)=[coefA((sizecoefA(2)+1-I):sizecoefA(2)) zeros(1,sizeComps-I)];
  else
      LAcomp(I,1:sizeComps)=[zeros(1,I-sizecoefA(2))  coefA(1:sizecoefA(2)) zeros(1,sizeComps-I)];
  end
end
%for I=1:(sizeComps);
%    if I<=sizecoefb(2);
%       Lbcomp(I,1:sizeComps)=[coefb((sizecoefb(2)+1-I):sizecoefb(2)) zeros(1,sizeComps-I)];
%   else
%       Lbcomp(I,1:sizeComps)=[zeros(1,I-sizecoefb(2))  coefb(1:sizecoefb(2)) zeros(1,sizeComps-I)];
%   end
%end
for I=1:(sizeComps);
   if I<=sizecoefB(2);
      Lbcomp(I,1:sizeComps)=[coefB((sizecoefB(2)+1-I):sizecoefB(2)) zeros(1,sizeComps-I)];
  else
      Lbcomp(I,1:sizeComps)=[zeros(1,I-sizecoefB(2))  coefB(1:sizecoefB(2)) zeros(1,sizeComps-I)];
  end
end
```

```
CAcomp=LAcomp(1:sizeComps,1:degreeB+1);
Cbcomp=Lbcomp(1:sizeComps,1:degreeA);
coefa=coefa*antidiag(sizecoefa(2));
coefA=coefA*antidiag(sizecoefA(2));
coefB=coefB*antidiag(sizecoefb(2)+1);
coefb=coefb*antidiag(sizecoefb(2));
coefa=[coefa zeros(1,n-sizecoefa(2))];
coefA=[coefA zeros(1,n-sizecoefA(2))];
coefB=[coefB zeros(1,n-sizecoefb(2)-1)];
coefb=[coefb zeros(1,n-sizecoefb(2))];
La=a(1:n,1:mi);
LA=A(1:n,1:mi);
Lb=b(1:n,1:mi);
LB=B(1:n,1:mi);
Mb=b(1:n,mi+1:n);
MA=A(1:n,mi+1:n);
YX=inv([LA LB])*[1 zeros(1,n-1)]';
coefY=[YX(1:(n/2))]';
coefX=[YX((n/2+1):n)]';
sizecoefX=size(coefX);
coefX=coefX*antidiag(sizecoefX(2));
for I=1:(n);
   if I<=sizecoefX(2);
      X(I,1:n)=[coefX((sizecoefX(2)+1-I):sizecoefX(2)) zeros(1,n-I)];
  else
      X(I,1:n)=[zeros(1,I-sizecoefX(2))  coefX(1:sizecoefX(2)) zeros(1,n-I)];
  end
end
CXcomp=X(mi+1:n,1:n);
invertme=Lb'*Lb + sigma*LA'*LA; %right way
pr=[1 zeros(1,mi-1)]*inv(invertme)*(Lb' - Lb'*Mb*CXcomp - sigma*LA'*MA*CXcomp);
pc=[1 zeros(1,mi-1)]*inv(invertme)*(LB'*hankel(coefB) + sigma*LA'*hankel(coefA));
%pc=[1 zeros(1,mi-1)]*inv(invertme)*(Lb'*hankel(coefB) + sigma*LA'*hankel(coefA));
MN=inv([CAcomp Cbcomp])*[pc(1:(degreeA+1)) zeros(1,degreeB)]';
%MN=inv([CAcomp Cbcomp])*[pc(1:(sizecoefa(2)+1)) zeros(1,sizecoefb(2))]';
coefM=[MN(1:degreeA)]';
coefM=[coefM 0]-[0 coefM];
coefM=coefM(1:degreeA);
coefN=[MN((sizecoefa(2)+1):(degreeA+degreeB+1))]';
   entradanumagua=u;
elseif flag == 3
   coefN
   [saidaagua entradanumagua(1:n/2-1)']'
   saidanumagua=coefN*[saidaagua entradanumagua(1:n/2-1)'];
   saidanumagua=saidanumagua
   sys=saidanumagua;
elseif flag == 4
  sys = [];
elseif flag == 0
  % Return sizes of parameters and initial conditions
  sys(1) = 0;                  % 0 continuous states
  sys(2) = 0;                  % 2*order discrete states (num and den)
  sys(3) = 1;                  % 2*order outputs (num and den)
  sys(4) = n+1;                   % 2*order inputs (num and den)
  entradanumagua=zeros(1,n+1)';
  coefN=zeros(1,n/2);
  sys(5) = 0;                  % no roots
  sys(6) = 0;                  % no direct feedthrough
```

```
  sys(7) = 1;                    % 1 sample time
  yn=[0 0 0 0 0 0];
  saidanumagua=0;
  outputagua = 0;
  x0 = [];
  ts = [-1, 0];
else
  % all other flags, return an empty matrix
  sys = [];
end
```

numeradorSGPCMisoAgente.m

```
function  [sys, x0, str, ts]  = numeradorSGPCMisoAgente (t,x,u,flag,n,sigma,mi,num,den)
global entradanumagente;
global saidaagente;
global saidanumagente;
global saidadenagente;
global ynagente;
global coefN;
if abs(flag) == 2
   a=zeros(n);
   A=zeros(n);
   b=zeros(n);
   B=zeros(n);
   coefa=den;
   coefA=[coefa 0]-[0 coefa];
   coefb=num;
   coefB=[0 coefb];
sizecoefa=size(coefa);
sizecoefA=size(coefA);
sizecoefb=size(coefb);
sizecoefB=size(coefB);
degreeA=sizecoefa(2);
degreeB=sizecoefb(2);
coefa=coefa*antidiag(sizecoefa(2));
coefA=coefA*antidiag(sizecoefA(2));
coefB=coefB*antidiag(sizecoefb(2)+1);
coefb=coefb*antidiag(sizecoefb(2));
for I=1:n;
   if I<=sizecoefa(2);
      a(I,1:n)=[coefa((sizecoefa(2)+1-I):sizecoefa(2)) zeros(1,n-I)];
  else
      a(I,1:n)=[zeros(1,I-sizecoefa(2))  coefa(1:sizecoefa(2)) zeros(1,n-I)];
  end
end
for I=1:n;
   if I<=sizecoefA(2);
      A(I,1:n)=[coefA((sizecoefA(2)+1-I):sizecoefA(2)) zeros(1,n-I)];
  else
      A(I,1:n)=[zeros(1,I-sizecoefA(2))  coefA(1:sizecoefA(2)) zeros(1,n-I)];
  end
end
for I=1:n;
   if I<=sizecoefb(2);
```

```
      b(I,1:n)=[coefb((sizecoefb(2)+1-I):sizecoefb(2)) zeros(1,n-I)];
   else
      b(I,1:n)=[zeros(1,I-sizecoefb(2))  coefb(1:sizecoefb(2)) zeros(1,n-I)];
   end
end
for I=1:(n-1);
    if I<=sizecoefb(2);
      B(I+1,1:n)=[coefb((sizecoefb(2)+1-I):sizecoefb(2)) zeros(1,n-I)];
   else
      B(I+1,1:n)=[zeros(1,I-sizecoefb(2))  coefb(1:sizecoefb(2)) zeros(1,n-I)];
   end
end
sizeComps=degreeA+degreeB+1;
for I=1:(sizeComps);
    if I<=sizecoefa(2);
      LAcomp(I,1:sizeComps)=[coefA((sizecoefA(2)+1-I):sizecoefA(2)) zeros(1,sizeComps-I)];
   else
      LAcomp(I,1:sizeComps)=[zeros(1,I-sizecoefA(2))  coefA(1:sizecoefA(2)) zeros(1,sizeComps-I)];
   end
end
%for I=1:(sizeComps);
%    if I<=sizecoefb(2);
%      Lbcomp(I,1:sizeComps)=[coefb((sizecoefb(2)+1-I):sizecoefb(2)) zeros(1,sizeComps-I)];
%   else
%      Lbcomp(I,1:sizeComps)=[zeros(1,I-sizecoefb(2))  coefb(1:sizecoefb(2)) zeros(1,sizeComps-I)];
%   end
%end
for I=1:(sizeComps);
    if I<=sizecoefB(2);
      Lbcomp(I,1:sizeComps)=[coefB((sizecoefB(2)+1-I):sizecoefB(2)) zeros(1,sizeComps-I)];
   else
      Lbcomp(I,1:sizeComps)=[zeros(1,I-sizecoefB(2))  coefB(1:sizecoefB(2)) zeros(1,sizeComps-I)];
   end
end
CAcomp=LAcomp(1:sizeComps,1:degreeB+1);
Cbcomp=Lbcomp(1:sizeComps,1:degreeA);
coefa=coefa*antidiag(sizecoefa(2));
coefA=coefA*antidiag(sizecoefA(2));
coefB=coefB*antidiag(sizecoefb(2)+1);
coefb=coefb*antidiag(sizecoefb(2));
coefa=[coefa zeros(1,n-sizecoefa(2))];
coefA=[coefA zeros(1,n-sizecoefA(2))];
coefB=[coefB zeros(1,n-sizecoefb(2)-1)];
coefb=[coefb zeros(1,n-sizecoefb(2))];
La=a(1:n,1:mi);
LA=A(1:n,1:mi);
Lb=b(1:n,1:mi);
LB=B(1:n,1:mi);
Mb=b(1:n,mi+1:n);
MA=A(1:n,mi+1:n);
YX=inv([LA LB])*[1 zeros(1,n-1)]';
coefY=[YX(1:(n/2))]';
coefX=[YX((n/2+1):n)]';
sizecoefX=size(coefX);
coefX=coefX*antidiag(sizecoefX(2));
for I=1:(n);
    if I<=sizecoefX(2);
      X(I,1:n)=[coefX((sizecoefX(2)+1-I):sizecoefX(2)) zeros(1,n-I)];
```

```
  else
     X(I,1:n)=[zeros(1,I-sizecoefX(2))  coefX(1:sizecoefX(2)) zeros(1,n-I)];
  end
end
CXcomp=X(mi+1:n,1:n);
invertme=Lb'*Lb + sigma*LA'*LA; %right way
pr=[1 zeros(1,mi-1)]*inv(invertme)*(Lb' - Lb'*Mb*CXcomp - sigma*LA'*MA*CXcomp);
pc=[1 zeros(1,mi-1)]*inv(invertme)*(LB'*hankel(coefB) + sigma*LA'*hankel(coefA));
%pc=[1 zeros(1,mi-1)]*inv(invertme)*(Lb'*hankel(coefB) + sigma*LA'*hankel(coefA));
MN=inv([CAcomp Cbcomp])*[pc(1:(degreeA+1)) zeros(1,degreeB)]';
%MN=inv([CAcomp Cbcomp])*[pc(1:(sizecoefa(2)+1)) zeros(1,sizecoefb(2))]';
coefM=[MN(1:degreeA)]';
coefM=[coefM 0]-[0 coefM];
coefM=coefM(1:degreeA);
coefN=[MN((sizecoefa(2)+1):(degreeA+degreeB+1))]';
   entradanumagente=u;
elseif flag == 3
   coefN
   [saidaagente entradanumagente(1:n/2-1)']'
   saidanumagente=coefN*[saidaagente entradanumagente(1:n/2-1)']';
   saidanumagente=saidanumagente
   sys=saidanumagente;
elseif flag == 4
 sys = [];
elseif flag == 0
 % Return sizes of parameters and initial conditions
 sys(1) = 0;                  % 0 continuous states
 sys(2) = 0;                  % 2*order discrete states (num and den)
 sys(3) = 1;                  % 2*order outputs (num and den)
 sys(4) = n+1;                   % 2*order inputs (num and den)
 entradanumagente=zeros(1,n+1)';
 coefN=zeros(1,n/2);
 sys(5) = 0;                  % no roots
 sys(6) = 0;                  % no direct feedthrough
 sys(7) = 1;                  % 1 sample time
 ynagente=[0 0 0 0 0 0];
 saidanumagente=0;
 outputagente = 0;
 x0 = [];
 ts = [-1, 0];
else
 % all other flags, return an empty matrix
 sys = [];
end
```

prefilterSGPC.m

```
function  [sys, x0, str, ts]  = prefilterSGPC(t,x,u,flag,n,sigma,mi,num,den)
global saidapre;
if abs(flag) == 2
   a=zeros(n);
   A=zeros(n);
   b=zeros(n);
   B=zeros(n);
   coefa=den;
```

```
    coefA=[coefa 0]-[0 coefa];
    coefb=num;
    coefB=[0 coefb];
sizecoefa=size(coefa);
sizecoefA=size(coefA);
sizecoefb=size(coefb);
sizecoefB=size(coefB);
degreeA=sizecoefa(2);
degreeB=sizecoefb(2);
coefa=coefa*antidiag(sizecoefa(2));
coefA=coefA*antidiag(sizecoefA(2));
coefB=coefB*antidiag(sizecoefb(2)+1);
coefb=coefb*antidiag(sizecoefb(2));
for I=1:n;
   if I<=sizecoefa(2);
      a(I,1:n)=[coefa((sizecoefa(2)+1-I):sizecoefa(2)) zeros(1,n-I)];
  else
      a(I,1:n)=[zeros(1,I-sizecoefa(2))  coefa(1:sizecoefa(2)) zeros(1,n-I)];
  end
end
for I=1:n;
   if I<=sizecoefA(2);
      A(I,1:n)=[coefA((sizecoefA(2)+1-I):sizecoefA(2)) zeros(1,n-I)];
  else
      A(I,1:n)=[zeros(1,I-sizecoefA(2))  coefA(1:sizecoefA(2)) zeros(1,n-I)];
  end
end
for I=1:n;
   if I<=sizecoefb(2);
      b(I,1:n)=[coefb((sizecoefb(2)+1-I):sizecoefb(2)) zeros(1,n-I)];
  else
      b(I,1:n)=[zeros(1,I-sizecoefb(2))  coefb(1:sizecoefb(2)) zeros(1,n-I)];
  end
end
for I=1:(n-1);
   if I<=sizecoefb(2);
      B(I+1,1:n)=[coefb((sizecoefb(2)+1-I):sizecoefb(2)) zeros(1,n-I)];
  else
      B(I+1,1:n)=[zeros(1,I-sizecoefb(2))  coefb(1:sizecoefb(2)) zeros(1,n-I)];
  end
end
sizeComps=degreeA+degreeB+1;
for I=1:(sizeComps);
   if I<=sizecoefa(2);
      LAcomp(I,1:sizeComps)=[coefA((sizecoefA(2)+1-I):sizecoefA(2)) zeros(1,sizeComps-I)];
  else
      LAcomp(I,1:sizeComps)=[zeros(1,I-sizecoefA(2))  coefA(1:sizecoefA(2)) zeros(1,sizeComps-I)];
  end
end
%for I=1:(sizeComps);
%   if I<=sizecoefb(2);
%      Lbcomp(I,1:sizeComps)=[coefb((sizecoefb(2)+1-I):sizecoefb(2)) zeros(1,sizeComps-I)];
%  else
%      Lbcomp(I,1:sizeComps)=[zeros(1,I-sizecoefb(2))  coefb(1:sizecoefb(2)) zeros(1,sizeComps-I)];
%  end
%end
for I=1:(sizeComps);
   if I<=sizecoefB(2);
```

```
      Lbcomp(I,1:sizeComps)=[coefB((sizecoefB(2)+1-I):sizecoefB(2)) zeros(1,sizeComps-I)];
   else
      Lbcomp(I,1:sizeComps)=[zeros(1,I-sizecoefB(2))  coefB(1:sizecoefB(2)) zeros(1,sizeComps-I)];
   end
end
CAcomp=LAcomp(1:sizeComps,1:degreeB+1);
Cbcomp=Lbcomp(1:sizeComps,1:degreeA);
coefa=coefa*antidiag(sizecoefa(2));
coefA=coefA*antidiag(sizecoefA(2));
coefB=coefB*antidiag(sizecoefb(2)+1);
coefb=coefb*antidiag(sizecoefb(2));
coefa=[coefa zeros(1,n-sizecoefa(2))];
coefA=[coefA zeros(1,n-sizecoefA(2))];
coefB=[coefB zeros(1,n-sizecoefb(2)-1)];
coefb=[coefb zeros(1,n-sizecoefb(2))];
La=a(1:n,1:mi);
LA=A(1:n,1:mi);
Lb=b(1:n,1:mi);
LB=B(1:n,1:mi);
Mb=b(1:n,mi+1:n);
MA=A(1:n,mi+1:n);
YX=inv([LA LB])*[1 zeros(1,n-1)]';
coefY=[YX(1:(n/2))]';
coefX=[YX((n/2+1):n)]';
sizecoefX=size(coefX);
coefX=coefX*antidiag(sizecoefX(2));
for I=1:(n);
    if I<=sizecoefX(2);
      X(I,1:n)=[coefX((sizecoefX(2)+1-I):sizecoefX(2)) zeros(1,n-I)];
   else
      X(I,1:n)=[zeros(1,I-sizecoefX(2))  coefX(1:sizecoefX(2)) zeros(1,n-I)];
   end
end
CXcomp=X(mi+1:n,1:n);
invertme=Lb'*Lb + sigma*LA'*LA; %right way
pr=[1 zeros(1,mi-1)]*inv(invertme)*(Lb' - Lb'*Mb*CXcomp - sigma*LA'*MA*CXcomp);
    saidapre=sum(pr)*u(1);
elseif flag == 3
    sys=saidapre;
elseif flag == 4
  sys = [];
elseif flag == 0
  % Return sizes of parameters and initial conditions
  sys(1) = 0;                  % 0 continuous states
  sys(2) = 0;                  % 2*order discrete states (num and den)
  sys(3) = 1;                  % 2*order outputs (num and den)
  sys(4) = n+1;                   % 2*order inputs (num and den)
  sys(5) = 0;                  % no roots
  sys(6) = 0;                  % no direct feedthrough
  sys(7) = 1;                  % 1 sample time
  saidapre=0;
  saida = 0;
  x0 = [];
  ts = [-1, 0];
else
  % all other flags, return an empty matrix
  sys = [];
```

```
end
%   This M-file performs an online update of the plant model in the file rls.m
%
%The input arguments is the block name to be updated
%
%   Copyright (c) 2000 by Otacílio da Mota Almeida.
%   Changed by Eber de Castro Diniz for MISO case
```

PARAMRLSMISO.m

```
function [sys, x0, str, ts] = var_par(t,x,u,flag,name)
global a;
global b;
global c;
if abs(flag) == 2
% sample hit, return the next discrete states
  num = [u(1:2)'];
  den = [1 ,u(3:4)'];
  %Qdo o identificador estiver online
  %b=[u(1:2)' 0 0 0 0 0];
  %a=[u(3:4)' 0 0 0 0 0 0];
  %fim qdo o identificador estiver online
  l=num;
  p=den;
    sys =[l, p];
    num=['[' sprintf('%.8g ',l) ']'];
    den=['[' sprintf('%.8g ',p) ']'];
    set_param(name,'Numerator',num,'Denominator',den);
elseif flag == 3
  % return the block outputs
  sys = x;
  sys = sys(:);
elseif flag == 4
  sys = [];
elseif flag == 0
  % Return sizes of parameters and initial conditions
  sys(1) = 0;                   % 0 continuous states
  sys(2) = 5;                   % 2*order discrete states (num and den)
  sys(3) = 5;                   % 2*order outputs (num and den)
  %  sys(4) = 4;                    % 2*order inputs (num and den)
  sys(4) = 6;                   % 2*order inputs (num and den)
  sys(5) = 0;                   % no roots
  sys(6) = 0;                   % no direct feedthrough
  sys(7) = 1;                   % 1 sample time
  x0 = zeros(5,1);
  ts = [-1, 0];
else
  % all other flags, return an empty matrix
  sys = [];
end
```

RLSESTSMISO.m

```
function  [sys, x0, str, ts]  = rlsestsMISO(t,x,u,flag,nstates,lambda,dt)
global newerr;
global ykanterior;
global phianterior;
%RLSESTS S-function to perform system identification.
%
%   This M-file is designed to be used in a Simulink S-function block.
%   It performs parameter estimation using the Recursive Least Squares
%   Parameter Estimation Algorithm with Exponential Data Weighting
%
%   The input arguments are
%
%    nstates:       the number of states in the states vector
%    lambda:        the exponential data weighting factor
%    dt:            how often to sample points (secs)
%
%   The RLS estimator is defined by the following equations:
%
%                                1      P(k-2) * phi(k-1) * [y(k) - phi(k-1)'theta(k-1)]
%    theta[k] = theta[k-1] + ------ * ----------------------------------------------------
%                            lambda        lambda + phi(k-1)' * P(k-2) * phi(k-1)
%
%                   1       P(k-2) * phi(k-1) * phi(k-1)' * P(k-2)
%    P(k-1) = ------ * ---------------------------------------
%                lambda    lambda + phi(k-1)' * P(k-2) * phi(k-1)
%
%   where:
%
%    theta:                    the parameter estimates
%    phi:              the state vector
%    P:                the covariance matrix
%    lambda:                   the exponential data weighting factor
%
%   See also SFUNTMPL., "Adaptive Filtering, Prediction, and Control",
%   G. C. Goodwin & K. S. Sin.
%   Copyright (c) 2001 by Otacílio da Mota Almeida.
%   Changed by Eber de Castro Diniz for MISO case
if abs(flag) == 2     % flag = 2 --> real time hit
  % sample hit, return the next discrete states, which are the
  % next parameter estimates
  theta = x(1:nstates+1);                                          % parameter estimates(this is the anterior
theta
  P = zeros(nstates+1,nstates+1);                                  % get covariance matrix
  P(:) = x(nstates+2:(nstates+1+(nstates+1)*(nstates+1)));
  yk = u(nstates - 1);                                             % system output
  phi = [u(1:nstates-2)' newerr]';                                         % state vector
  est_err = yk - phi' * theta;                                % estimation error
  den = 1 + phi' * P * phi;                             % lambda + phi' * P * phi
  theta_new = theta + P * phi * (est_err / den);% new parameter estimates
  Pnew = (P - P * phi * phi' * P / den) / 1;       % new covariance
  newerr(3)=newerr(2);
  newerr(2)=newerr(1);
  newerr(1) = yk - phi' * theta_new;
  ykanterior = yk;
  phianterior = phi;
  sys = [theta_new', Pnew(:)'];                                % return them
elseif flag == 4      % flag = 4 --> Return next sample hit
```

```
  sys = [];
elseif flag == 0      % flag = 0 --> Return sizes of parameters and initial conditions
  sys(1) = 0;                                    % 0 continuous states
  sys(2) = nstates+1+(nstates+1)*(nstates+1); % enough discrete states to hold the estimates
                                                 % and the covariance matrix
  sys(3) = nstates;                        % nstate estimate outputs
  sys(4) = nstates-1;                      % nstate+1 (regression vector + system output)
                                                 % inputs
  sys(5) = 0;                                    % 0 roots
  sys(6) = 0;                                    % no direct feedthrough
  sys(7) = 1;                                    % 1 sample time
  % initialize the covariance matrix and initial estimates
  P = eye(nstates+1, nstates+1) * 1e6;
  %  theta_ini=[3.9999999 2.0000001   -1.6 0.8 0];
  theta_ini=[zeros(1,nstates+1)];
  newerr=[0 0 0];
  phianterior = zeros(nstates+1,1);
  x0 = [theta_ini, P(:)'];
  ts = [dt, 0];
elseif flag == 3      % flag = 3 --> Return outputs, only at sample hits
    sys = x(1:nstates);
    sys = sys(:);
else
  sys = [];
end
```

antidiag.m

```
function [out] = antidiag(n)
out=zeros(n);
out(1,1:n)=[zeros(1,n-1) 1];
for I=2:n-1;
    out(I,1:n)=[zeros(1,n-I) 1 zeros(1,I-1)];
end
out(n,1:n)=[1 zeros(1,n-1)];
```

enviapwm.m

```
function  [sys, x0, str, ts]  = enviapwm(t,x,u,flag,dt)
global fid;
global saidapwm1;
global saidapwm2;
if abs(flag) == 2
    saidapwm1=u(1);
    saidapwm2=u(2);
elseif flag == 3
% return the block outputs
  if saidapwm1 > 800,
      saidapwm1=800;
  end
  if saidapwm1 < 0,
     saidapwm1=0;
```

```
 end
 if saidapwm2 < 0;
   saidapwm2=0;
end
 if saidapwm2 > 800;
    saidapwm2=800;
 end
 cportwrite(fid,'a');
 cportwrite(fid,int2str(saidapwm1));
 cportwrite(fid,'f');
 a=0;
 advalue=0;
 value=zeros(1,3);
 counter=1;
 cportwrite(fid,'c');
 cportwrite(fid,'f');
 while 1,
   [A,err]=cportgetchar(fid,1);
   if A == 'f';
     break;
   end
   value(counter)=A;
   counter=counter+1;
 end
 if counter == 5,
  advalue=(value(1)-48)*1000+(value(2)-48)*100+(value(3)-48)*10+(value(4)-48);
 end
 if counter == 4,
  advalue=(value(1)-48)*100+(value(2)-48)*10+value(3)-48;
 end
 if counter == 3,
   advalue=(value(1)-48)*10+(value(2)-48)*1;
 end
 cportwrite(fid,'b');
 cportwrite(fid,int2str(saidapwm2));
 cportwrite(fid,'f');
 a=1023-advalue;
 sys = [a];
%  fclose(fid);
%  sys = sys(:);
elseif flag == 4
%  fclose(fid);
 sys = [];
elseif flag == 0
 % Return sizes of parameters and initial conditions
 sys(1) = 0;                % 0 continuous states
 sys(2) = 0;                % 2*order discrete states (num and den)
 sys(3) = 1;                % 2*order outputs (num and den)
 sys(4) = 2;                % 2*order inputs (num and den)
 sys(5) = 0;                % no roots
 sys(6) = 0;                % no direct feedthrough
 sys(7) = 1;                % 1 sample time
 saidapwm1=0;
 saidapwm2=0;
 a=0;
 fid=cportopen('com1');
 cportconfig(fid,'BaudRate',9600);
```

```
  x0 = [];
  ts = [dt, 0];
elseif flag == 9  %Termination
    cportwrite(fid,'a');
    cportwrite(fid,'0');
    cportwrite(fid,'f');
    cportwrite(fid,'b');
    cportwrite(fid,'0');
    cportwrite(fid,'f');
    cportclose(fid);
else
  % all other flags, return an empty matrix
  sys = [];
end
```

O algoritmo utilizado pelo microcontrolador, servindo apenas como interfaceamento entre o Matlab e a planta real é mostrado abaixo.

sendreceive.c

```
#include <18F452.h>
#device ADC=10
#fuses hs,noprotect,nowdt,nobrownout
#use delay(clock=20000000)
#use rs232(baud=9600,xmit=PIN_C6,rcv=PIN_C7,BRGH1OK)
void main() {
   char option;
   int16 charRcvd;
   int16 Value;
   int16 a[5]={0x30,0x30,0x30,0x30,0x30};
   char index;
   setup_timer_2(T2_DIV_BY_16, 255, 1);//Atuador a 1.2kHz  1/20M * 4 * 16 * 256
   setup_ccp1(CCP_PWM);   // Configure CCP1 as a PWM
   setup_ccp2(CCP_PWM);   // Configure CCP2 as a PWM
   setup_adc_ports( ALL_ANALOG);//Todas as portas sao analogicas(PORTA A)
   setup_adc(ADC_CLOCK_INTERNAL);//Usa o clock interno. 50kHz de frequencia de amostragem
   set_adc_channel(0);//Canal 0 para leitura do AD(AN0)
   Value=0;
   index=0;
   set_pwm1_duty(Value);
   set_pwm2_duty(Value);
   while(true) {
      if(kbhit()) {
        charRcvd=getc()&0x00ff;
        if((charRcvd == 'a') //pwm1
         ||(charRcvd == 'b') //pwm2
         ||(charRcvd == 'c')
         ||(charRcvd == 'd')
         ||(charRcvd == 'e')) {
           index=0;
```

```
            option = charRcvd;
          } else {
            if(charRcvd == 'f') {
               if(index==1) {
                 Value = a[0]-0x30;
               }
               if(index==2) {
                 Value = (a[0]-0x30)*10+a[1]-0x30;
               }
               if(index==3) {
                 Value = (a[0]-0x30)*100+(a[1]-0x30)*10+(a[2]-0x30);
               }
               if(index==4) {
                 Value = (a[0]-0x30)*1000+(a[1]-0x30)*100+(a[2]-0x30)*10+a[3]-0x30;
               }
               if(option == 'a') {
                 set_pwm1_duty(Value);
               }
               if(option == 'b') {
                 set_pwm2_duty(Value);
               }
               if(option == 'c') {//AD
                 Value=read_adc();
                 printf("%lu",Value);
                 putc('f');
               }
            } else {
               a[index]=0x00ff&charRcvd;
               index+=1;
            }
          }
        }
      }
    }
```