Chapter 3: Model Predictive Control

Model predictive control (MPC) uses a dynamic model of the plant to predict the future actions of the manipulated variables on the plant output (Garcia *et. al.*, 1989). The future moves of the manipulated variables can then be determined by minimising the difference between the setpoint and the predicted output. These optimisation calculations are performed during each time step using the latest measurements. Only the first calculated move of the manipulated variable is therefore used, since at each time step the manipulated variables are recalculated.

The main advantages of model predictive control are best shown by the following systems (Morari, 1991, Morari & Lee, 1999):

1. A system with a large number of manipulated and controlled variables.
2. A system with constraints on the manipulated and/or controlled variables.
3. Processes whose control objectives are changed.
4. When equipment, for instance a sensor, fails.
5. Processes with a large amount of dead time.

The advantages of using model predictive control are as follows (Morari, 1991):

1. The technique does not require a state-space or transfer function process model. It uses a step or impulse response directly, requiring no explicit process identification procedures.
2. On-line optimisation is used, enabling easy handling of constraints and actuator/sensor failures.
3. The operation of the controller can be shown by plotting the predicted closed loop behaviour, providing the operator with a clear indication of why the manipulated variable is changed.

MPC also has a few disadvantages (Hugo, 2000). The first is that it is more complex to design a MPC controller than for example a PID controller. All the control objectives and constraints have to be incorporated into a single objective function which can then be

optimised. For simple systems it is therefore not worthwhile to design a MPC when nearly the same results can be achieved by using a simple PID controller.

The basic structure of a model predictive controller is shown in Figure 3-1 (Deshpande, 1989, Morari & Zafiriou, 1989).

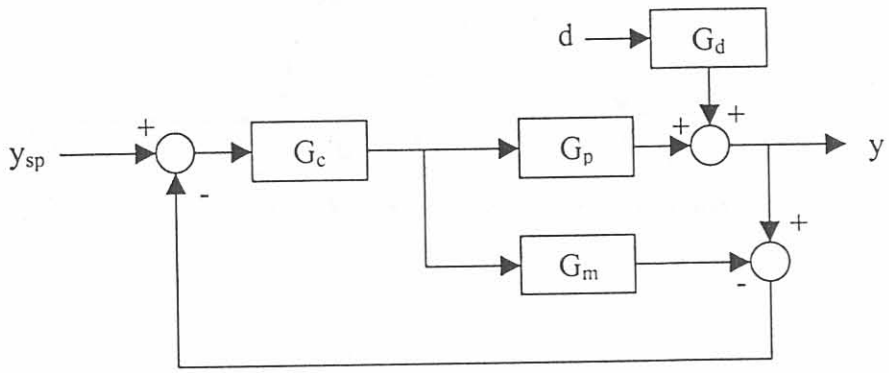

Figure 3-1: Model predictive control

The algorithm executed during each time step n by the MPC is as follows:

1. The present and future control actions (m_n, ...m_{n+k}) are determined by minimising the difference between the predicted process output ($\hat{c}_{n+1}$) and the target (y_{sp}). The quadratic objective function is normally used to minimise this error.
2. Only the first computed control action (m_n) is implemented.
3. At time $n+1$ a new measurement becomes available and the whole procedure is repeated.

3.1 Step Response Model

A typical response of a system to a unit step change in the input (m) is shown in Figure 3-2. The system had an initial value of c_0 and the step change was applied at time $t = 0$.

After each time step (Δt) the value of the response is stored in a step response coefficient (a_i). The response of the system with initial condition c_0 to any step change in the input (Δm) can now be calculated by using these step response coefficients (Seborg *et al*, 1989):

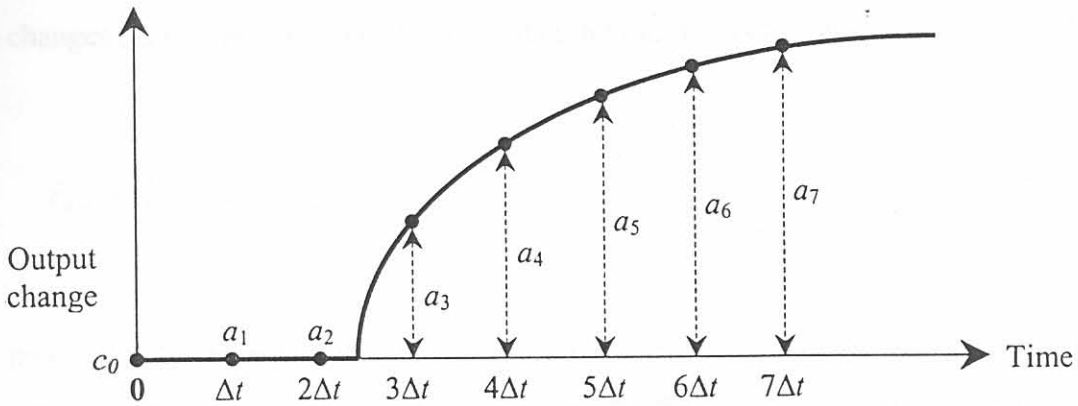

Figure 3-2: Step response coefficients

After one time step, the predicted output ($\hat{c}_1$) is: $\hat{c}_1 = c_0 + a_1\Delta m_0$ .....(3.1)

After two time steps, the predicted output ($\hat{c}_2$) is: $\hat{c}_2 = c_0 + a_2\Delta m_0$ .....(3.2)

After T time steps, the predicted output ($\hat{c}_T$) is: $\hat{c}_T = c_0 + a_T\Delta m_0$ .....(3.3)

When two step changes are made to the system, one at $t = 0$ (Δm_0) and one at $t = \Delta t$ (Δm_1), the response can be predicted as follows:

After one time step, the predicted output ($\hat{c}_1$) is: $\hat{c}_1 = c_0 + a_1\Delta m_0$ .....(3.4)

After two time steps, the predicted output ($\hat{c}_2$) is: $\hat{c}_2 = c_0 + a_2\Delta m_0 + a_1\Delta m_1$ ....(3.5)

where the second term ($a_2\Delta m_0$) represents the contribution of the first step change (Δm_0) to the total change (2 time steps after the disturbance has taken place). The third term ($a_1\Delta m_1$) represents the contribution of the second step change (Δm_0) to the total change (only 1 time step after the disturbance has taken place).

After T time steps, the predicted output ($\hat{c}_T$) is: $\hat{c}_T = c_0 + a_T\Delta m_0 + a_{T-1}\Delta m_1$ ..(3.6)

In general, the output for the process for a certain input can be expressed as a sequence of step changes ($\Delta m_i = m_i - m_{i-1}$) in the input at each time step as follows:

$$\hat{c}_{n+1} = c_0 + \sum_{i=1}^{n} a_i \Delta m_{n+1-i} \quad \text{.....} \quad (3.7)$$

The problem with this equation is that if it is applied for a long time (n large), the number of calculations would become very large, since every change in the past will influence the current value of c. This problem can be overcome by using the fact that the effect of changes in the input will become constant after a certain period of time for most processes (excluding integrators). The effect on the output due to changes in the inputs (Δm) that have occurred longer back than a time equal to the settling time of the process, will therefore remain constant. Equation 3.7 can therefore be rewritten as follows (Marlin, 1995):

$$\hat{c}_{n+1} = c_0 + \sum_{i=T+1}^{n} a_i \Delta m_{n+1-i} + \sum_{i=1}^{T} a_i \Delta m_{n+1-i} \quad \text{.....} \quad (3.8)$$

where T is the model horizon and should be chosen such that $T\Delta t \approx$ settling time. The second term in the equation contains all the changes in inputs (Δm's) that have occurred more than T time steps ago and whose effects on the output will remain the same. The third term in the equation contains those changes in inputs that have occurred less than T time steps ago and whose effects on the output have not yet reached steady state. After each new sampling period, the value of the second term will therefore only change by one value: by one Δm that occurred T time steps ago (n-T). The first term (initial condition c_0) and the second term can therefore be combined in one recursive equation as follows:

$$\bar{c}_n = \bar{c}_{n-1} + a_T \Delta m_{n-T} \quad \text{.....} \quad (3.9)$$

Combining this equation with Equation 3.8 results in the following recursive equation:

$$\hat{c}_{n+1} = \bar{c}_n + \sum_{i=1}^{T} a_i \Delta m_{n+1-i} \quad \text{.....} \quad (3.10)$$

This form of prediction corresponds to open-loop prediction, since it does not correct for model errors. A corrected prediction (c^*_{n+1}) is used to compensate for model errors using the difference between the actual measured value (c_m) and the computed value at time n ($\hat{c}_n$), and then shifting it one time step forward (assuming that the error will remain the same):

$$c^*_{n+1} - \hat{c}_{n+1} = c_m - \hat{c}_n \quad \text{..... (3.11)}$$

This correction compensates for model errors as well as unmeasured load changes. By combining Equation 3.11 and Equation 3.10, it can be reduced to:

$$c^*_{n+1} = (c_m - \hat{c}_n) + \bar{c}_n + \sum_{i=1}^{T} a_i \Delta m_{n+1-i} \quad \text{..... (3.12)}$$

3.2 Basic Model Predictive Control Algorithm

The algorithm will be introduced by looking at the situation encountered each time the model predictive feedback controller is executed. This time step will be called n. It is assumed that the manipulated variable has been changed in the past, influencing the controlled variable. The controlled variables can furthermore be influenced by disturbances.

The aim of the controller is to determine the future adjustments in the manipulated variable (Δm^f) that will result in the predicted controlled variable returning to the setpoint as quickly as possible. The difference between the current controlled variable and the setpoint can easily be measured, but the controlled variable will still change in the future due to the past changes of the manipulated variable (Δm). This behaviour of the controlled variables in the future without changes in the future manipulated variables can be calculated. The future errors should then be calculated and used to determine the future adjustments of the manipulated variable. The predicted value of the controlled variable at each future instance j (c^f_j) as

influenced by only the past changes in the manipulated variable (Δm) can be calculated as follows (see Figure 3-3):

$$c_j^f = \bar{c}_n + \sum_{i=1}^{T} a_{j+i} \Delta m_{n+1-i} \qquad j = 1 \text{ to } V \qquad (3.13)$$

This equation, however, does not incorporate feedback, since the measured value at time n is never used. The feedback signal is an error value (E^n) that is calculated by subtracting the current calculated value of the controlled variable ($\hat{c}_n$) from the current measured value (c_m). This error is assumed to stay constant for the future predictions and is therefore added to each future prediction (c_i^f):

$$c_j^f = (c_m - \hat{c}_n) + \bar{c}_n + \sum_{i=1}^{T} a_{j+i} \Delta m_{n+1-i} \quad j = 1 \text{ to } V \qquad (3.14)$$

This equation will therefore accommodate for differences between the measured and calculated values due to modelling errors and disturbances.

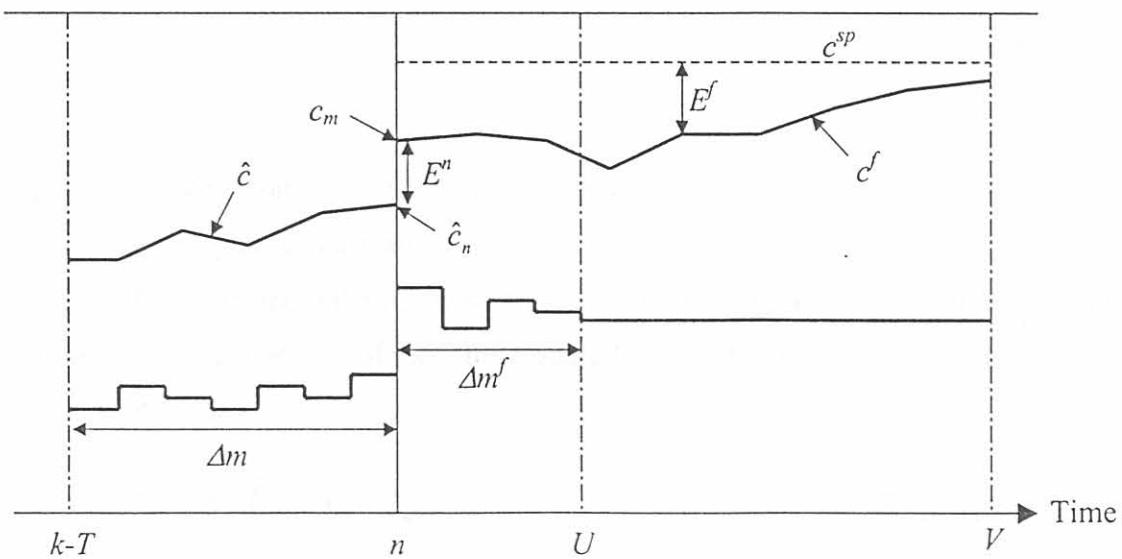

Figure 3-3: Dynamic response of manipulated and controlled variables

The future changes in the controlled variables ($\hat{c}$) due to future moves in the manipulated variables (Δm^f) can also be calculated. In MPC, the number of future control actions or control moves that will be calculated during each time step, is called the control horizon (U). The number of future outputs (controlled variables) that will be calculated using the control horizon is called the prediction horizon (V). At each time step n the next U values of m are calculated as well as the next V output directions ($\hat{c}$) caused by the future calculated changes of m:

$$\hat{c}_j = \sum_{i=1}^{j} a_i \Delta m^f_{j-i} \qquad j = 1 \text{ to } V \qquad \text{.....(3.15)}$$

A general equation for these calculations can be written in matrix form:

$$\begin{bmatrix} \hat{c}_1 \\ \hat{c}_2 \\ \text{M} \\ \hat{c}_V \end{bmatrix} = \begin{bmatrix} a_1 & 0 & \Lambda & 0 \\ a_2 & a_1 & & 0 \\ \text{M} & a_2 & a_1 & \text{M} \\ a_V & a_{V-1} & \Lambda & a_{V-U+1} \end{bmatrix} \begin{bmatrix} \Delta m^f_0 \\ \Delta m^f_1 \\ \text{M} \\ \Delta m^f_{U-1} \end{bmatrix} \qquad \text{.....(3.16)}$$

or

$$\hat{c} = A \cdot \Delta m^f$$

The objective of the controller is now to minimise the difference between the future controlled variables (due to past and future changes in the manipulated variables) and the setpoint (closed loop error). If the future setpoints are not known, it is to be assumed the same as the current setpoint. The sum of the errors squared objective function is used:

$$OBJ = \sum_{j=1}^{V} \left[c^{sp}_j - \left(c^f_j + \hat{c}_j \right) \right]^2 \qquad \text{.....(3.17)}$$

and in matrix form:

$$OBJ = \left[c^{sp} - \left(c^{f} + \hat{c}\right)\right]^{T}\left[c^{sp} - \left(c^{f} + \hat{c}\right)\right] = \left[E^{f} - A\Delta m^{f}\right]^{T}\left[E^{f} - A\Delta m^{f}\right]$$

with: $E^{f} = c^{sp} - c^{f}$

$\hat{c} = A \cdot \Delta m^{f}$

The objective function can now be minimised by changing only these future manipulated variables (Δm^{f}) that will influence the value of $\hat{c}$. For an ideal controller, the minimum will result when there is no difference between the predicted output of the closed loop system and the desired setpoint:

$$E^{f} - A\Delta m^{f} = 0 \quad \text{.....(3.18)}$$

If the number of predicted outputs and the number of control moves are equal ($V = U$), then the future control moves (Δm) for this case can be easily calculated:

$$\Delta m^{f} = (A)^{-1} E^{f} \quad \text{..... (3.19)}$$

This, however, will not result in the best controller. By setting $U = V$, a minimal prototype (deadbeat) controller will result, which will require excessive moves of the manipulated variable. A better controller can be obtained by setting $U < V$. This will result in a system that is overdetermined, but an optimum solution can be obtained using the least-squares method. This method minimises the objective function by differentiating with respect to Δm^{f} and setting it equal to zero, which results in the following solution for Δm^{f}:

$$\Delta m^{f} = (A^{T}A)^{-1} A^{T} E^{f} \quad \text{..... (3.20)}$$

The objective function can be expanded by adding the square of the changes in the manipulated variables. This will penalise the movement of the manipulated variables. The two terms in the objective function can furthermore be multiplied with a weighting matrix. The relative values of these two tuning parameters (weighting matrices) will determine the

importance placed on the controlled variable and the variability of the manipulated variable. The objective function will now look as follows:

$$OBJ = \left[E^f - A\Delta m^f\right]^T \mathrm{W}_1^T \mathrm{W}_1 \left[E^f - A\Delta m^f\right] + \left(\Delta m^f\right)^T W_2^T W_2 \Delta m^f \quad \ldots\ldots\ldots (3.21)$$

When differentiating this function with respect to Δm^f and setting it equal to zero, the following solution for Δm^f is obtained:

$$\Delta m^f = (A^T W_1^T W_1 A + W_2^T W_2)^{-1} A^T W_1^T W_1 E^f \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3.22)$$

During each time step, only the first control action (Δm_1^f) is implemented. The next output (c_{n+1}) is then observed, the predictions are corrected and Equation 3.22 is applied again. A problem that may occur, especially when the $A^T A$ matrix is ill-conditioned, is that excessively large changes in the manipulated variable may result. This problem can be overcome by modifying the weighting matrices.

3.3 MPC for MIMO Systems

The results developed for a SISO system can be extended to MIMO systems by using the principle of superposition (Luyben, 1990):

$$\hat{c}_{1,n+1} = \sum_{i=1}^{T} a_{11i} \Delta m_{1,n+1-i} + \sum_{i=1}^{T} a_{12i} \Delta m_{2,n+1-i} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3.23)$$

$$\hat{c}_{2,n+1} = \sum_{i=1}^{T} a_{21i} \Delta m_{1,n+1-i} + \sum_{i=1}^{T} a_{22i} \Delta m_{2,n+1-i} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3.24)$$

where $\hat{c}_{1,n+1}$ denotes the predicted value of c_1 at the $(n+1)^{\text{th}}$ sampling instant. T is selected in accordance with the largest model horizon of the four models. The error function can also be written in matrix form:

$$\hat{E} = -A\Delta m^f + E^f \quad \text{.....(3.25)}$$

For a two by two system, the vectors $\hat{E}$ and E^f will be of length $2V$ and Δm^f will be of length $2U$. The matrix A will look as follows:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad \text{.....(3.26)}$$

with each partition $A_{i,j}$ a triangular matrix of the same form as described for SISO systems. Each $A_{i,j}$ will have a dimension of $(V \times U)$, resulting in an A matrix with dimensions $(2V \times 2U)$. The predictive control law will then look the same as for a SISO system. The weighting matrix for each individual manipulated/controlled variable pair ($W_{1,ij}$ and $W_{2,ij}$) is also combined into one matrix by putting each individual matrix at the diagonal of the large matrix as follows:

$$W_1 = \begin{bmatrix} W_{1,11} & 0 \\ 0 & W_{1,22} \end{bmatrix}, \qquad W_2 = \begin{bmatrix} W_{2,11} & 0 \\ 0 & W_{2,22} \end{bmatrix} \quad \text{.....(3.27)}$$

3.3.1 Different horizons for each input/output

If the same horizons are used for each input/output pair, problems may arise if the system has large differences in the dynamics or dead time. If the one input/output pair has a slow response or a large dead time, then the model horizon (T) has to be very large to ensure that steady state is reached. If another input/output pair now has quick response, the time step has to be very small so that the dynamic information is not lost. The largest model horizon and the smallest time step therefore have to be used for each input/output pair, resulting in an unnecessarily large controller. This effect is clearly shown in Figure 3-4 where the last four step response coefficients for the first input/output pair (a_{11}) are unnecessary, since this response has already reached its steady state at the third time interval. The long horizon,

however, is necessary for the second input/output pair (a_{22}), which takes much longer to reach steady state.

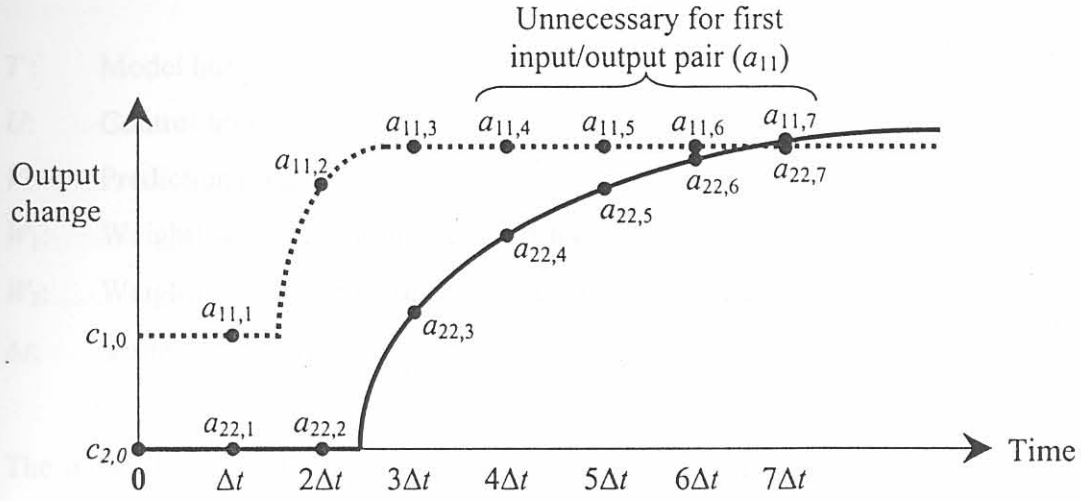

Figure 3-4: Differing dynamics

This problem can be overcome if the model horizon for each output can be specified separately. The model horizon for the first output (c_1) therefore only has to be 3, while the horizon for the second output (c_2) has to be at least seven. The prediction horizon for each output and the control horizon for each input can also be specified separately to ensure that the controller is as small as possible. Each partition (A_{ij}) in the total coefficient matrix (A) can now have different dimensions. The dimension of A_{ij} will now be $V_i \times U_j$ and the dimension of the combined A-matrix will be $\left(\sum_{i}^{Outputs} V_i\right) \times \left(\sum_{j}^{Inputs} U_j\right)$. This is shown in the following example:

$$A = \begin{bmatrix} [a_1]_{1\times 1} & [c_1 \quad c_2 \quad c_3]_{1\times 3} \\ \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}_{2\times 1} & \begin{bmatrix} d_{11} & d_{12} & d_{13} \\ d_{21} & d_{22} & d_{23} \end{bmatrix}_{2\times 3} \end{bmatrix}$$

where the control horizons are: For first input (m_1), $U_1 = 1$

For second input (m_2), $U_2 = 3$

and the prediction horizons are: For first output (c_1), $V_1 = 1$

For second output (c_2), $V_2 = 2$

3.4 Tuning of a Model Predictive Controller

The following design parameters can be adjusted to tune a model predictive controller:

T: Model horizon

U: Control horizon

V: Prediction horizon

W_1: Weighting matrix for predicted errors

W_2: Weighting matrix for control moves (changes in manipulated variable)

Δt: Sampling period

The model horizon ($T\Delta t$) is usually selected such that it is larger than the open loop settling time (time for the open loop step response to reach 99% of its final value). Values between 20 and 70 are the recommended range for T (Prett & Garcia, 1988).

The prediction horizon (V) is the number of predictions that are used in the optimisation calculations. Increasing V will produce a more conservative controller (stabilising effect), but it will also increase the computational effort. V can either be used as a separate tuning factor, or it can be set equal to $T + U$.

The control horizon (U) is the number of future control actions that will be calculated during optimisation. A good first guess for U is to choose U such that $U\Delta t \cong t_{60}$, the time for the open loop response to reach 60% of its steady state value.

The weighting matrices (W_1 and W_2) have many potential tuning parameters, but it is usually sufficient to set W_1 equal to the identity matrix (I) and $W_2 = fI$. Larger values of f will penalise the magnitude of ΔMV more, resulting in less vigorous control.

The sampling period (Δt) should be small enough to prevent losing important dynamic information. If Δt is too small, however, T must be made very large to satisfy the first criterion, which is not desirable.

3.5 Handling Integrators

Integrators will cause problems in MPC due to the following two reasons (McDonald & McAvoy, 1986):

- The assumption that the process will reach a steady state is not applicable for integrators, since they will never reach steady state. This assumption has been made to derive the recursive equation 3.12.
- The second assumption that will not be applicable for integrators is the fact that the measured error will stay constant for the prediction horizon. If the error was caused by an integrator, the future error can be estimated by utilising a ramp function.

Before looking at a solution for these two problems, we must first take a closer look at the nature of integrators. Integrators often show an initial dynamic response similar to that of a first order response (see Figure 3-5). The only difference is that it will not settle at a constant value, but will continue to increase steadily. A pure integrator can therefore be said to be at "steady state" if it changes at a constant rate (if the slope does not change). The model horizon (T) should therefore be as large as the time it takes fore the system to settle to a constant slope.

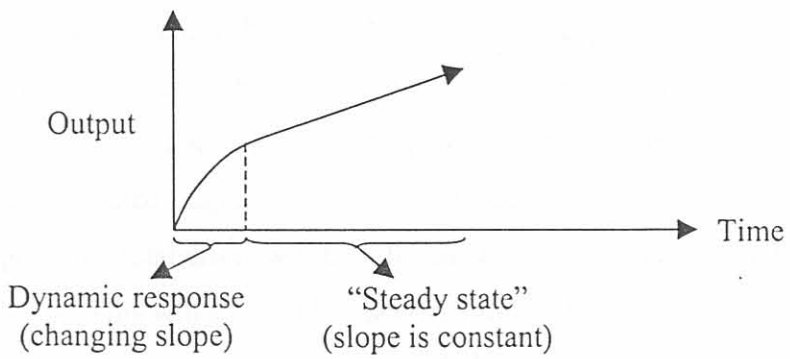

Figure 3-5: Integrator response

When this constant slope region is reached, only the accumulated change of the input (sum of all the input changes that occurred more than T time steps ago) needs to be considered to determine the effect on the output. The amount that the output will change is then simply the slope (for a unit step response) times the time period, times the accumulated change in input. The slope can be determined by dividing the difference of the last two step response

coefficients, which should be in the constant slope region, by the time step. The equation for the change in the output (Δc) due to an accumulated change in the input (Δm^{tot}) can then be written as follows:

$$\Delta c = \frac{a_T - a_{T-1}}{\Delta t} \cdot \Delta t \cdot \Delta m^{tot} = (a_T - a_{T-1})\Delta m^{tot} \quad \text{.....(3.28)}$$

Equation 3.10 can therefore be modified to also incorporate this integral action by simply adding this term:

$$\hat{c}_{n+1} = \bar{c}_n + \sum_{i=1}^{T} a_i \Delta m_{n+1-i} + (a_T - a_{T-1})\Delta m^{tot} \quad \text{.....(3.29)}$$

This equation will still be applicable for non-integrating systems, since the term ($a_T - a_{T-1}$) will be zero if the system has reached steady state and will therefore disappear.

The second problem that can occur is if an error is caused by a disturbance variable that has an integral action on the system. In this case the error will continue to increase and the assumption of a constant error for the prediction horizon will not be valid. The problem is that an error is usually caused by a number of different factors, especially in multivariable systems. Part of the error may therefore be caused by an integrator error, but part may also be due to a constant error. The current measured error (the difference between the measured output and the calculated output) is therefore divided into two parts. The one part (specified as a percentage of the total error) will then be due to an integral error, and the rest will be due to a constant error. This will have to be specified for each output variable and can be seen as an extra tuning parameter. It is now assumed that the future error will also consist of a constant error portion and an integral error portion. It is assumed that the integral error portion will continue to increase in the future with a constant slope (see Figure 3-6). This slope can be determined by dividing the current integral error portion with the time interval of the controller, since it is assumed that the integral part of the error was zero when the previous control action was taken. The predicted future error will now be the sum of the constant error portion and the integral error portion.

$$e_j = \frac{x_I(c_m - \hat{c}_n)}{\Delta t} \cdot (j+1) \cdot \Delta t + (1 - x_I)(c_m - \hat{c}_n) = (c_m - \hat{c}_n)(j \cdot x_I + 1) \quad \text{.....(3.30)}$$

where: x_I = fraction of the total error attributed to integral action

$j = 1$ to V

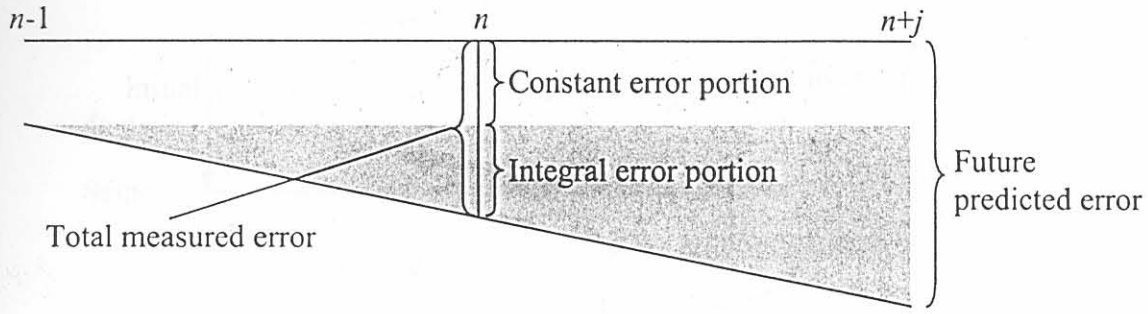

Figure 3-6: Integral error

Equation 3.14 can therefore be rewritten by incorporating this dynamic error as follows:

$$c_j^f = (c_m - \hat{c}_n)(j \cdot x_I + 1) + \bar{c}_n + \sum_{i=1}^{T} a_{j+i} \Delta m_{n+1-i} \qquad \text{where: } j = 1 \text{ to } V \text{ .....(3.31)}$$

3.6 Handling Noise

It is important for any controller to handle noise effectively, otherwise the controller may try to control the noise, which will lead to excessive movement of the manipulated variables. The usual way to handle noise is to use a filter to eliminate the noise. The problem with this method is that it adds extra dynamics to the system that will make it more sluggish to respond.

MPC can make use of target areas or variable regions of uncertainty for control (Cutler, 1982) to handle noise. A variable region of uncertainty is a region around the setpoint that is specified by the user. If the output is within this region, the error (difference between the setpoint and the output) is set to zero and no control action is taken to correct this error

(Angus, 1989). This region can be specified for the whole prediction horizon (see Figure 3-7). If the output is outside this region, the new error will be set equal to the distance between the predicted output value and the nearest uncertainty region boundary. This will result in less strict control and also less movement of the manipulated variable.

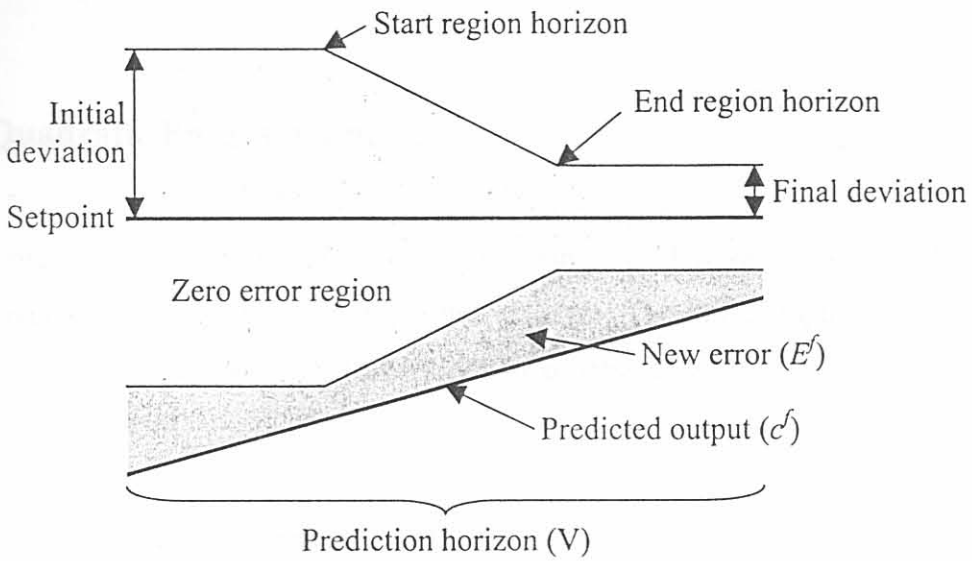

Figure 3-7: Variable region of uncertainty

3.7 Process Constraints

Almost all practical control problems have constraints on both the manipulated and controlled variables. This is due to physical limitations of the plant equipment. These constraints have to be incorporated in the control strategy to ensure effective control (Rao & Rawlings, 1999).

One way of dealing with process constraints is to adjust the tuning parameters. The selection of tuning parameters will then be an iterative procedure until all the constraints are met. This procedure, however, is not always satisfactory, since it will not guarantee that the constraints are never violated.

A more direct approach is to modify the optimisation problem by adding the constraints explicitly. Either a linear or quadratic programming approach can be followed. If a linear programming method is followed, the objective function has to be changed to a linear form.

The advantage of linear programming is that its computational effort can be four to five times less than that of the quadratic programming method. It was, however, decided to use quadratic programming since it is normally easier to tune than linear programming (Seborg *et. al.*, 1989). The reason for this is because with quadratic programming the controlled variables can be weighted in the quadratic performance index. Quadratic programming furthermore typically results in smoother responses than the linear programming approach.

3.8 Quadratic Programming

In real processes, the computed values of the manipulated variables may not be obtainable due to physical constraints or limits on the variables. The limits imposed on the controlled variables may also be violated. Three kinds of process constraints are usually encountered (Garcia *et. al.*, 1986):

1. Manipulated variable constraints (valve saturation).
2. Controlled variable constraints (overshoots in the controlled variables past allowable limits must be avoided).
3. Associated variables (key process variables that are not directly controlled but that must be kept within bounds).

Ideally, the model predictive controller should now predict violations and prescribe moves that would keep these variables within bounds. With the least-squares method, the controller does not check whether the computed manipulated variable moves is within the limits and it furthermore does not check if the outputs will violate the limits. The quadratic programming technique uses an advanced algorithm that explicitly ensures that these constraints are not violated. First, it will be shown how al the constraints can be written in matrix form. These matrix equations will then be combined into a quadratic programming problem as will be shown later.

The constraints on the manipulated variables can be written as follows:

$$m_{i,\min} \leq m_{i,n} + \sum_{j=1}^{U} \Delta m_{i,n+j-1}^{f} \leq m_{i,\max} \ , i = 1 \ldots n_m \quad \ldots\ldots\ldots\ldots \quad (3.32)$$

where n_m = number of manipulated variables. This equation can be written in matrix notation as follows:

$$\begin{bmatrix} 1_L & & \\ & \mathrm{O} & \\ & & 1_L \\ -1_L & & \\ & \mathrm{O} & \\ & & -1_L \end{bmatrix} \Delta m^f \leq \begin{bmatrix} (m_{1,\max} - m_{1,n})\overline{1} \\ \mathrm{M} \\ (m_{n_m,\max} - m_{n_m,n})\overline{1} \\ (m_{1,n} - m_{1,\min})\overline{1} \\ \mathrm{M} \\ (m_{n_m,n} - m_{n_m,\min})\overline{1} \end{bmatrix} \quad \ldots\ldots\ldots\ldots \quad (3.33)$$

with $\overline{1} = \begin{bmatrix} 1 & 1 & \Lambda & 1 \end{bmatrix}^T$ of length U and 1_L an $U \times U$ lower triangular identity matrix:

$$1_L = \begin{bmatrix} 1 & 0 & 0 & \Lambda & 0 \\ 1 & 1 & 0 & \Lambda & 0 \\ \mathrm{M} & & & & \mathrm{M} \\ 1 & 1 & 1 & \Lambda & 1 \end{bmatrix} \quad \ldots\ldots\ldots\ldots \quad (3.34)$$

Actually only the first change in the manipulated variable needs to be constrained to prevent violation of the constraints, since only the first change in the manipulated variable is implemented. Constraining all the manipulated variable projections will however improve the performance of the MPC controller (Garcia *et. al.*, 1986).

Constraints on the controlled variables will typically look as follows:

$$c_{\min} \cdot \overline{1} \leq c \leq c_{\max} \cdot \overline{1} \quad \ldots\ldots\ldots\ldots \quad (3.35)$$

where c is a vector of the values of the controlled variable at the current or future instances. This time the values of the future variables are calculated by looking at the past and future

predicted moves of the manipulated variables. The values of the controlled variables due to changes in the past at each future instant j (c^f_{n+j}) are calculated according to Equation **3.14**. The additional changes that the controlled variables will undergo if the future changes of the manipulated variable are incorporated ($\hat{c}$), are calculated using Equation **3.16** ($\hat{c} = A\Delta m^f$).

The total change of the controlled variable in the future (c_{n+j}) will therefore be the sum of the two future predicted values:

$$c = c^f + A\Delta m^f \quad \text{.....(3.36)}$$

c^f can be written as follows in terms of the open loop error (E^f) as follows:

$$c^f = c^{sp} - E^f \quad \text{.....(3.37)}$$

and substituted into Equation **3.36**:

$$c = c^{sp} - E^f + A\Delta m^f \quad \text{.....(3.38)}$$

The constraint Equation (**3.35**) can now be written as follows:

$$(c_{\min} - c^{sp})\bar{1} + \hat{E} \le A\Delta m^f \le (c_{\max} - c^{sp})\bar{1} + \hat{E} \quad \text{.....(3.39)}$$

or

$$\begin{bmatrix} A \\ -A \end{bmatrix} \Delta m^f \le \begin{bmatrix} (c_{\max} - c^{sp})\bar{1} + \hat{E} \\ (c^{sp} - c_{\min})\bar{1} - \hat{E} \end{bmatrix} \quad \text{.....(3.40)}$$

It is furthermore also possible for quadratic programming to keep associated variables within bounds using the same technique as with the controlled variables. A new projection variable

for the associated variable will have to be created. The constraints on the projections of a single associated variable (v) can be written analogous to Equation **3.40**:

$$\begin{bmatrix} B \\ -B \end{bmatrix} \Delta m^f \leq \begin{bmatrix} (v_{\max})\bar{1} - \hat{E}_v \\ \hat{E}_v - (v_{\min})\bar{1} \end{bmatrix} \quad \text{..... (3.41)}$$

with

$$\hat{E}_v = \begin{bmatrix} v^*_{k+1} + (v_m - v^*_k) \\ \text{M} \\ v^*_{k+V} + (v_m - v^*_k) \end{bmatrix} \quad \text{..... (3.42)}$$

B is the dynamic matrix for the associated variables (the same as A for the controlled variables) and v_m is the measured value at the current instant.

All these equations can now be combined into the following quadratic problem (Garcia *et. al.*, 1986):

$$\min_{\Delta mv} F = \tfrac{1}{2}\left(\Delta m^f\right)^T \cdot H \cdot \Delta m^f - g^T \cdot \Delta m^f \quad \text{..... (3.43)}$$

subject to: $D \cdot \Delta m^f \leq d$

where: $H = A^T W_1^T W_1 A + W_2^T W_2$ (The quadratic program Hessian matrix)

$g = A^T W_1^T W_1 E^f$ (The quadratic program gradient vector)

D is obtained by combining the matrix of equations **3.40**, **3.41** and **3.33**. d is obtained by combining the vectors to the right of the inequality sign in equations Error! Not a valid link. and Error! Not a valid link..

3.8.1 Tuning of a Quadratic Controller

When using quadratic programming, there are some additional tuning parameters for tuning the MPC controller than the ones described earlier. The projection interval that should be constrained can be chosen. In practice, only a subset of all the V projections in Equations **3.40** and **3.41** are constrained, starting with the l^{th} projection, where $l \geq 1$. This subset of future projections forms a "constraint window" over which the quadratic controller will prevent constraint violations from occurring. Performance is often improved by moving the "constraint window" further down the horizon (Garcia *et. al.*, 1986). The reason for this is that any projection violation is handled rigorously by the quadratic program. Severe input moves in the face of non-minimum phase characteristics are sometimes required to correct for violations in the earlier projections.

3.8.2 Solving the Quadratic Program

Quadratic programming represents a special class of non-linear programming in which the objective function is quadratic and the constraints are linear. The functions usually look as follows:

Minimise: $c^T x + \frac{1}{2} x^T H x$

Subject to: $Ax \leq b$

$x \geq 0$

where c is an n vector, b is an m vector, A is an $m \times n$ matrix and H is an $n \times n$ symmetric matrix. To get the quadratic programming problem for MPC (equation 3.43) in this form, the equations have to be transformed. To ensure that x is always positive as required be the equations above, x was set equal to $(\Delta m + [m_{min} - m_{min}])$, since the change in the manipulated variable (Δm) can never be larger than the difference between the maximum and the minimum value for that manipulated variable $(m_{min} - m_{min})$.

A more general set of linear constraints can easily be cast in this format using standard linear transformations. These functions can be written as the Karush-Kuhn-Tucker conditions (Bazaraa *et al*, 1993):

$$Ax + y = b$$
$$-Hx - A^T u + v = c$$
$$x^T v = 0, \qquad u^T y = 0$$
$$x, y, u, v \geq 0$$

These equations can be rewritten as the linear complementary problem (Bazaraa *et al*, 1993):

$$w - Mz = q \qquad w^T z = 0 \qquad w, z \geq 0$$

with

$$M = \begin{bmatrix} 0 & -A \\ A^T & H \end{bmatrix}, \qquad q = \begin{bmatrix} b \\ c \end{bmatrix}, \qquad w = \begin{bmatrix} y \\ v \end{bmatrix}, \qquad z = \begin{bmatrix} u \\ x \end{bmatrix}$$

The complementary pivoting algorithm (Bazaraa *et al*, 1993) can thus be used to solve this linear complementary problem. The values of x will then be the optimum solution for the quadratic programming problem that lies within the constraints. The algorithm for the complementary pivoting algorithm is as follows:

Initialisation step:

If $q \geq 0$ then $(w, z) = (q, 0)$ is a complementary basic feasible solution. Otherwise, write the linear complementary system in the following tableau format:

	W	Z	z_0	RHS
w	I	$-M$	-1	q

RHS = right hand side

Let $-q_s = \max\{-q_i : 1 \leq i \leq p\}$, $\qquad p = length(q)$

The tableau is updated next by pivoting row s and the z_0 column. Let $y_s = z_s$ and go to the main step.

***Main Step*:**

1. Let d_s be the updated column in the current tableau under the variable y_s. If $d_s \leq 0$, go to step 4. Otherwise, determine the index r by the following minimum ratio test:

$$\frac{q_r}{q_{rs}} = \min_{1 \leq i \leq p} \left\{ \frac{q_i}{d_{is}} : d_{is} > 0 \right\}$$

   where q is the updated right hand side column denoting the values of the basic variables. If the basic variable (column whose row entries are zero, except row r, which is one) at row r is z_0, go to step 3. Otherwise, go to step 2.
2. The basic variable at row r is either w_l or z_l for some $l \neq s$. The variable y_s enters the basis and the tableau is updated by pivoting at row r and the y_s column. If the variable that just left the basis is w_l, then $y_s = z_l$; and if the variable that just left the basis is z_l, then $y_s = w_l$. Return to step 1.
3. Here y_s enters the basis and z_0 leaves the basis. Pivot at the y_s column and the z_0 row, producing a complementary basic feasible solution. Stop.
4. Stop with ray termination. A ray $R = \{(w, z, z_0) + \lambda d : \lambda \geq 0\}$ is found such that every point in R satisfies the linear complementary problem.

Pivoting:

To pivot row r and column c in the tableau (t), the following procedure is followed:

1. Divide the r^{th} row by t_{rj}.
2. Multiply the new r^{th} row by t_{ij} and subtract from the i^{th} constraint row, for $i = 1, \text{K}, m$, $i \neq r$.

3.9 PI-Control for Comparison Purposes

A brief overview of proportional-integral (PI) control is given here since it will be used to compare MPC with and it is also used in the input module of the simulator. The general structure of a PI feedback controller is shown in Figure 3-8.

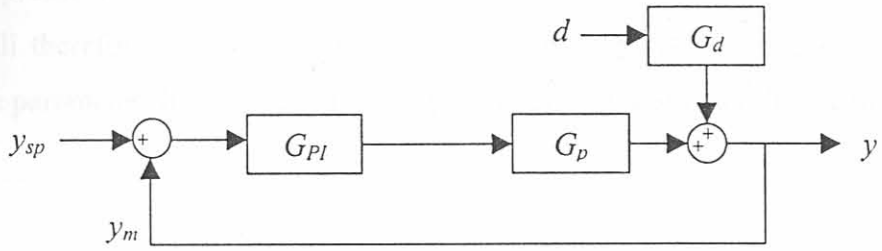

Figure 3-8: PID-controller structure

The input to the controller box is the error (difference between the measured value and the setpoint of the controlled variable). The output of the PI-controller is the change in the manipulated variable (MV).

The general equation for the continuous transfer function of the PI-controller is as follows (Stephanopoulos, 1984):

$$G_{PI} = K_c\left(1+\frac{1}{\tau_I s}\right) \qquad (3.44)$$

Reasonable parameters for the PI controller (K_c, τ_I) can be obtained from step response data using the Cohen-Coon method. After a model has been determined from step response data, the following equations can be used to calculate the parameters for the PI controller (these equations were developed for a one-quarter decay ratio, a minimum offset and a minimum integral square error):

$$K_c = \frac{1}{K}\frac{\tau}{t_d}\left(0.9+\frac{t_d}{12\tau}\right) \qquad (3.45)$$

$$\tau_I = t_d \left(\frac{30 + \frac{3t_d}{\tau}}{9 + \frac{20t_d}{\tau}} \right) \quad \text{..... (3.46)}$$

These parameters for the controller are only approximate values, since the Cohen-Coon method approximates the process response as a first order system with dead time. The method will therefore not work well for higher order responses or responses with integral action. The parameters have to be retuned further to get the best controller setting.