ENGG4420 -- LECTURE 10
SECTION 1.4. DDC CONTROL -- IMPLEMENTATION OF THE REAL-TIME CONTROL ALGORITHMS

IMPLEMENTATION OF THE CONTROLL TASK
○ PID controller task algorithm – rectangular method
○ Synchronization of the control loop
○ Bumpless transfer
○ Velocity PID algorithm
○ Saturation and integral wind-up action

DIRECT DIGITAL CONTROL (DDC) -- LOOP CONTROL
• In DDC, the computer is in the feedback loop as is shown in figure below -- actuators are labelled as A.
• **IN DDC,** the computer is in the feedback loop. As a result, it forms a critical component in terms of the reliability of the system. We need to ensure that, in event of the failure or malfunction of the computer, the plant remains in a safe condition.

• Common safety procedures:
  ○ Limit the DDC unit to making incremental changes to the actuators on the plant; and
  ○ Limit the rate of change of the actuator settings.

• The advantages of DDC over the analog control are:
  ○ **Cost** – a single digital computer can control a large number of loops. With the introduction of microprocessors a single-loop DDC unit can be cheaper than an analog unit.
  ○ **Performance** – digital control offers simpler implementation of a wide range of control algorithms, improved controller accuracy and reduced drift.
  ○ **Safety** – modern digital hardware is highly reliable with long mean-time-between-failures and hence can improve the safety of systems. However, the software used in programmable digital systems may be much less reliable than the hardware.
DDC APPLICATIONS

- DDC may be applied either to a single-loop system implemented on a small microprocessor or to a large system involving several hundred loops.

- The development of integrated circuits (IC) and the microprocessors have ensured that in terms of cost the digital solution is now cheaper than the analog. Single-loop controllers used as stand-alone controllers are now based on the use of digital techniques and contain one or more microprocessor chips which are used to implement DDC algorithms.

- Many computer control implementations have simply taken over the well-established analog PID (Proportional + Integral + Derivative) algorithm.

- There is much more to computer control than simple DDC and the use of DDC is not limited to the PID control algorithm -- algorithms developed using various digital control design techniques can be equally effective and a lot more flexible than the three-term controller.

- However, the art of tuning PID controllers is well established and the techniques give a well-behaved controller.

- Also, the PID controller performs adequately in 90% of all control problems, it provides a strong deterrent to the adoption of new control system design techniques.
CONTINUOUS FEEDBACK CONTROL

The control tasks are directly related to the control scheme used. The above is a simplified block diagram of the feedback control part of the continuous control system. Also, some of the symbols and terms used are defined.

In this control diagram, we have a plant, a controller and a feedback loop:
- \( r(t) \) = set point; \( c(t) \) = controlled variable; \( e(t) = r(t) - c(t) \) = error; and \( m(t) \) = manipulated variable (used to determine the change in the controlled variable).

**Q1?:** In the case of our hot air plant what are the \( r(t) \), \( c(t) \), and \( m(t) \)?
PID CONTROLL ALGORITHM

- PI and PID control prove to give very good control performance for a wide range of industrial processes. Improvements to these control methods are not significant and the trade off is increased complexity.
- The PID algorithm can be expressed in many forms.
- The manipulated variable is a function of the (error) + (the integral of the error) + (the derivative of the error). That is why the three term controller is called PID.

\[ m(t) = K_p[e(t) + \frac{1}{T_i} \int_0^t e(t) \, dt + T_d \frac{de(t)}{dt}] \]  

- \( c(t) \): measured value
- \( r(t) \): reference value
- \( e(t) = r(t) - c(t) \): the error
- \( K_p \): overall controller gain
- \( T_i \): integral action time
- \( T_d \): derivative action time

For the majority of systems PI control is enough.
P CONTROL

- P control implies using a control signal that is made proportional to the error between the desired value of an output and the actual value of the output.
- The ratio between the control signal and the error signal can be adjusted using the proportionality constant (gain) $K_p$.
- Choosing the value of $K_p$ involves a compromise:
  - a high value of $K_p$ gives a small steady-state error and a fast response, but the response will be oscillatory and may be unacceptable in many applications;
  - a low value gives a slow response and a large steady-state error.
- These are some facts derived from control theory:
  - the size of the offset is directly proportional to the size of the load changes and inversely proportional to the $K_p$.
  - a higher value of $K_p$ giving a steeper gradient and so a smaller error change needed to accommodate a loop change.
PI CONTROL

- By adding the integral action term (the control signal is proportional to the integral of the error) the steady-state error can be reduced to zero since the integral action term integrates the error signal with respect to time.
- For a given error value the rate at which the integral term increases is determined by $T_i$.
- The integral term compensates for changes that occur in the process being controlled.
- A purely P controller operates correctly only under one particular set of process conditions; that is changes in the load on the process or some environmental condition will result in a steady-state error.
- The integral term compensates for these changes and reduces the error to zero.

PID CONTROL

- For processes that are subjected to sudden disturbances the addition of the derivative term can give improved performance.
- Because derivative action produces a control signal that is related to the rate of change of the error signal, it anticipates the error and hence acts to reduce the error that would otherwise arise from the disturbance.
SAMPLED FEEDBACK CONTROL SYSTEMS

GOALS:
1. Present controller algorithms that can be generated from the time domain version,
2. Present algorithms that are derived from the Z transform representation of a controller.

• This figure presents a simplified block diagram of a sampled feedback control system.
• In this diagram, the symbols $c(nT)$, $r(nT)$, $e(nT)$, and $u(nT)$ are the sampled values of $c(t)$, $r(t)$, $e(t)$, and $u(t)$, respectively, at sample times $nT$ where $n$ is an integer and $T$ is the sampling interval.
• Note that the DAC (digital to analog converter) and ADC (analog to digital converter) blocks are used to convert $u(nT)$ to $u(t)$ and $c(t)$ to $c(nT)$, respectively.
• Since around the plant the signals are all analog a digitization process is applied in order to interface to the computer.
In the PID Eq. (1) replace the differential and integral terms by their discrete equivalents (that use sampled values) by using the following relationships:

\[
\frac{df}{dt} \Bigg|_k = \frac{f_k - f_{k-1}}{\Delta t}, \quad \int e(t) dt = \sum_{k=1}^{n} e_k \Delta t
\]

\[
m(n) = K_p \left[ T_d \left( \frac{e(n) - e(n-1)}{\Delta t} \right) + e(n) + \frac{1}{T_i} \sum_{k=1}^{n} e_k \Delta t \right]
\]

As a result, Eq. (1) is transformed in the discrete form, Eq. (3)

- where m(n) represents the value of m at nΔt, and e(n) is the error at nΔt.
- By introducing the new parameters:
  - \( K_i = K_p \left( \frac{T_s}{T_i} \right) \); and \( K_d = K_p \left( \frac{T_d}{T_s} \right) \);
  - where \( T_s = \Delta t = \) the sampling interval,
- Then Eq. (3) can be expressed as a discrete algorithm of the form:

\[
s(n) = s(n-1) + e(n);
\]

\[
m(n) = K_p \cdot e(n) + K_i \cdot s(n) + K_d \cdot [e(n) - e(n-1)];
\]

Where s(n) is the sum of errors taken over the interval 0 to nT_s.
IMPLEMENTATION OF THE BASIC PID ALGORITHM

• The basic code statements for implementing the PID algorithm (4) are:

  >> sn = sn + en
  >> mn = Kp*en + Ki*sn + Kd*(en –enold)
  >> enold = en

• In implementing this algorithm we have to assume that:
  ○ the plant output is obtained by using an ADC to sample and convert the output signal,
  ○ the actuator control signal is outputted through a DAC to the actuator; and
  ○ procedures ADC and DAC are available to read the ADC and to send values to the DAC.

• Then a PID control module can be written.

• **Assignment:** Write a C-code or MATLAB code to implement the PID algorithm.
IMPLEMENTATION NOTES

• The calculations of the PID control algorithm must be synchronized with real-time.
• For correct operation the sampling interval $T_s$ should not be computer speed dependent and it should be fixed since the coefficients $K_i$ and $K_d$ are both calculated assuming a specific value of sampling interval.
• Or, the algorithm should be designed to include the sampling interval $T_s$ as a variable.

• Other problems to consider when implementing the PID algorithm are:
  ○ synchronization of the control loop;
  ○ bumpless transfer – that is, smooth transfer from manual to automatic control;
  ○ actuator limiting and other forms of saturation – this lead to integral wind-up; and
  ○ measurement and process noise.
• The program also has the controller parameters built in as program constants; hence modification of the controller settings requires re-compilation of the program.
SYNCRONIZATION OF THE CONTROL LOOP

• A typical feature of real-time programs is that once they have been started they run continuously until some external event occurs to stop them.
• Synchronization can be achieved by several different means such as:
  ○ polling; external interrupt signals; ballast coding; and real-time clock signals.

POLLING
• Synchronization can be achieved through polling -- The idea of polling is to have a synchronization procedure that reads a signal from the plant interface (i.e., sample time).
• When the synchronization procedure is called the computer waits in a loop until the interface signal is true. This wait is referred to as a “busy wait” since no other computation can be carried out while the computer is waiting.
• The polling method is easy to program and easy to design and use; however, because of the “busy wait” its use is restricted to small dedicated systems.
• An alternative method suitable for simple, dedicated, control systems is to use ballast coding.
BALLAST CODE

• The idea of the ballast coding is to make the loop time completely dependent on the internal operations of the computer and independent of external timing or synchronization signals.

• The method involves finding the time taken to execute each possible path in the control loop of the program and adding code statements – ballast code – to make the execution time for each path equal.
  ○ If necessary a further block of ballast code is added at the end to make the total execution time for the control loop equal to some desired execution time (i.e., sample time).

• The method minimizes the amount of external hardware required and is thus cost effective for systems that are to be produced in large quantities. An obvious problem is that any change in the code results in the need to adjust the ballast code segments.

• Also, the technique cannot be used if interrupts are being used (WHY NOT?) and the code will have to be modified if the CPU clock rate is changed. As is the case with polling, the use of the ballast code technique prevents the computer system being used to carry out any other calculations while it is waiting to carry out the next control calculation.
  ○ A: Interrupts are random events and the execution time on the paths is not controllable anymore (is non-deterministic).
• The ballast coding method can be illustrated by considering the program structure shown in this figure.
• For each path (for example: A, A1, A1.1) the computational time for that particular path is calculated or measured and ballast code is added to each so as to make the computational time for each path equal.
• For path A, A1, A1.1 ballast A1.1 is added. Further ballast code can be added to make the total computational time equal to the sample interval; this is shown as Ballast B.
SYNCHRONIZATION THROUGH EXTERNAL INTERRUPTS

- For small systems with a limited number of DDC loops (or other actions that requires synchronization), use of an external interrupt for synchronization can be very effective.
- The control loop is written as an interrupt which is associated with a particular interrupt line. The interrupt line is activated by some external device – typically a clock (set at intervals of $T_s$ or multiple of $T_s$).
- While the control loop is waiting to be activated other programs can be running. This form of operation is typically referred to as a foreground-background operation.

REAL-TIME CLOCK

- The most general solution to the problem of timing a control loop is provided by adding a real-time clock to the computer system. Provision of a real-time clock involves the addition of some hardware components and some software. Today's microcontrollers normally provide a real-time clock that can be accessed.
- The time is returned in terms of the number of ticks of the clock, a tick being the resolution of the clock, that is the smallest interval of time the clock can measure.
- Tasks are synchronized with the system's tick.
BUMPLESS TRANSFER

• In digital algorithm (4), the controlled variable \( m(n) \) is equal to the value of the integral term \( K_i * s(n) \) when the error is zero (steady state). Ideally, in the steady state with zero error we would like the integral term to be zero, which would mean that \( m(n) = 0 \).

• In many applications the steady-state operation conditions require that \( m(n) \) has some value other than zero -- For Example:
  ○ A steam boiler may require the fuel line valves to be half open.
  ○ For hot-air blower a non-zero voltage has to be applied to the heater input for the heater to provide heat output.

• One solution to have the integral term zero is to modify equation \( m(t) \) by adding a constant term \( M \) representing the value of the manipulated variable at the steady-state operation point, changing the general equation to:

\[
m(t) = K_p [e(t) + 1 / T_i \int_0^t e(t) dt + T_d d(e(t) / dt)] + M \tag{5}
\]

• The quantity \( M \) can be thought of as setting the operating point for the controller. If it is omitted and integral action is present, the integral term will compensate for its omission. However, changing from manual to automatic control will not be smooth. There will also be the danger that on change-over, a large change (for example, in a valve position) will be demanded.
METHODS TO ACHIEVE BUMPLESS TRANSFER

• Plant operating requirements usually demand that manual/automatic change-over be made in bumpless manner. Bumpless transfer can be achieved by using various techniques: 1) preset change-over value; 2) tracking of operator setting; 3) velocity algorithm.

1) PRESET CHANGE-OVER VALUE. The value of \( M \) is calculated for a given steady-state operating pointing and is inserted either as a constant in the program or by the operator prior to the change-over from manual to automatic mode.

• The transfer to automatic mode is made when the value of the error is zero; at the time of change-over the integral term is set to zero and the output \( m(n) \) is equaled to \( M \).

• The main drawbacks of this technique are:
  ○ the predetermined value of \( M \) is correct only for a specified load.
  ○ If the load is varying it may not be possible or convenient to make the change-over at the predetermined load value.
  ○ Also, if the error is not zero on change-over there will be a sudden change in the value of the manipulated variable due to the proportional action.
1) TRACKING THE OPERATOR SETTING -- during manual control the manipulated variable $m$ is set from the operator’s control panel -- the computer system can track this value in real-time in $mc$.
   ○ We can obtain an analog or digital readout from the operator’s control panel, or we can read the value of the input to the control actuator on the plant.
   • At the point at which change-over is made, from the algorithm point of view the value of $m$ is stored in a variable $mc$. Two algorithmic methods of transfer can be used:
     ○ METHOD 1 -- $M$ is not preset and change-over is made when the error $e(t)$ is zero; then $M = mc$.
     ○ METHOD 2 -- $M$ is preset to a value appropriate for the nominal level and change-over is made when the error is not zero. The integral action term needs to be set to an initial value: $s(0) = mc - Kp*ec - M$; where $ec = error$ value at change-over.

2) VELOCITY ALGORITHM -- An alternative form of the PID algorithm which is widely used to provide automatic bumpless transfer.
VELOCITY ALGORITHM
• The PID algorithm given by the equations (1) is called the positional algorithm because it calculates the absolute value of the actuator position.
• An alternative form of the PID algorithm is the velocity algorithm that can be used to solve the automatic bumpless transfer problem.
  ○ The velocity algorithm generates the value of the manipulated variable change at each sample time rather than the absolute value of the variable. As a result, the change is smaller and incremental.
• The continuous time equation of the velocity algorithm can be obtained by differentiating the positional PID model equation with respect to time:

\[ \frac{dm(t)}{dt} = K_p \left( \frac{de(t)}{dt} + \frac{1}{T_i} e(t) + T_d \frac{d^2 e(t)}{dt^2} \right) \]  

(6)

One method to obtain the difference equation is to use Eq. (6) and apply backward differences.
DIFFERENCE EQUATION FOR THE VELOCITY ALGORITHM -- can be easily obtained by finding \( m(n) - m(n-1) \) from Eq. (4) which is repeated below.

Approximations used for generating Eq. (4).

\[
\frac{de}{dt} \bigg|_k = \frac{e_k - e_{k-1}}{\Delta t} = \text{slope of the curve}
\]

\[
\int e(t) \, dt = \sum_{k=1}^{n} e_k \Delta t = \text{rectangular area approx.}
\]

(4) \[
\begin{align*}
\Delta m(n) &= m(n) - m(n-1) \\
\Delta m(n) &= K_p \Delta e(n) + K_i \int \Delta e(n) \, dt + K_d (e(n) - e(n-1))
\end{align*}
\]

\[
\begin{align*}
m(n) &= \Delta m(n) + m(n-1) \\
&= \Delta m(n) + K_p \Delta e(n) + K_i \int \Delta e(n) \, dt + K_d \Delta e(n) + K_d (e(n-1) - e(n-2))
\end{align*}
\]

\[
\begin{align*}
&= K_p \Delta e(n) + K_i \int \Delta e(n) \, dt + K_d (e(n) - 2e(n-1) + e(n-2)) \\
\Delta m(n) &= K_p \Delta e(n) + K_i \int \Delta e(n) \, dt + K_d (e(n) - 2e(n-1) + e(n-2)) \\
\end{align*}
\]

(7) \[
\Delta m(n) = K_p \Delta e(n) + K_i \int \Delta e(n) \, dt + K_d (e(n) - 2e(n-1) + e(n-2))
\]

(8) \[
\begin{align*}
\Delta m(n) &= K_1 e(n) + K_2 e(n-1) + K_3 e(n-2) \\
\text{Where} \quad K_1 &= K_p \left(1 + \frac{T_s}{T_i} + \frac{T_d}{T_s}\right) \\
K_2 &= - \left(1 + 2\frac{T_d}{T_s}\right) \\
K_3 &= T_d \frac{1}{T_s}
\end{align*}
\]
PROGRAMMING EQUATION FOR THE VELOCITY ALGORITHM

• \( \Delta m = K_1 \cdot e(n) + K_2 \cdot e(n-1) + k_3 \cdot e(n-2) \) -- which is simple and can be easily programmed.
  - Where: \( K_1 = K_p \left( 1 + T_s/T_i + T_d/T_s \right) \); \( K_2 = -(1 + 2T_d/T_s) \); \( K_3 = T_d/T_s \); and \( T_s \) is the sampling interval; \( T_i \) is the integral action time and \( T_d \) is the derivative action time.

• Because Eq. (7) outputs only a change in the controller position this algorithm automatically provides bumpless transfer. However, in cases with large standing error in the moment of change-over the response of the controller may be slow, particularly when \( T_i \) is large (long integral action time).

COMPARISON OF POSITION AND VELOCITY ALGORITHMS

• Eq. (4) VERSUS Eq. (8):
  1. the velocity algorithm is simpler and safer -- no large changes in demanded actuator position will occur.
  2. To protect against sudden large changes the maximum value which \( m(n) \) can take can be limited. For example, in valve position or motor speed large changes are avoided.
  3. Other sudden changes in both algorithms can occur if: a) the measured signal is noisy, b) the set point is changed or, c) by changing the PID parameters on-line.
METHOD for dealing with noisy measurements is to use a fourth-order difference algorithm to approximate $\text{de}/\text{dt}$ (explained later).

METHOD to solve the disturbance caused by set point changes -- modifying the algorithm to use the set point $r$ and the measured output $c$ rather than the error signal $e$.

- For example, in the velocity algorithm, based on the use of error $e$, the value of the set point appears in the derivative term and any change in value is differentiated (normally a change in set point is larger); hence a sudden step change can cause a large disturbance.
  - If in Eq. (7) we assume $r$ (set point) constant and $e(n) = r - c(n)$ -- then we get:
    \[
    \Delta m(n) = K_p \left( [c(n-1) - c(n)] + \frac{T_s}{T_i} (r - c(n)) \right) + \frac{T_d}{T_s} [2c(n-1) - c(n-2) - c(n)]
    \]

- From this algorithm form, changes in the set point are included only in the constant term $(T_s/T_i) \cdot r$ by changing $r$.
- The set point $r$ is now part of the integral term and hence the controller must always include integral action. In addition, $T_s/T_i$ should not be set to zero or some very small value.

- In summary, the effects of changing $r$ from the differentiation term can be removed by assuming $r$ constant (before change) and put the effects into the integral term.
METHOD FOR SOLVING disturbances caused by on-line parameter changes. In a digital algorithm such as the positional PID algorithm, a large parameter change can be introduced in a single computing step.

\[ s(n) = s(n-1) + e(n); \]
\[ m(n) = K_p \cdot e(n) + K_i \cdot s(n) + K_d \cdot [e(n) + e(n-1)]; \]

- In the algorithm above, the parameter \( e(n) \) is not necessarily large but \( s(n) \) can be much larger than \( e(n) \).
- As a result, if we make a change in the integral action time \( T_i \) there can be a significant step change in the output since \( K_i = K_p \frac{T_s}{T_i} \).
  - A solution to this problem is to limit the rate at which a parameter can change or we can alter the algorithm as shown below:
    \[ s(n) = s(n-1) + e(n) \frac{T_s}{T_i}; \]
    \[ m(n) = K_p \cdot e(n) + K_p \cdot s(n) + K_d \cdot [e(n) + e(n-1)]; \]
- The parameter \( \frac{T_s}{T_i} \) had been moved into the sum equation from the integral term and we assume that \( K_p \) is not changed.
- By writing the algorithm in this form the effects of making a change to \( T_i \) are much reduced if \( K_p \) is not changed.
  - (NOTE: Now \( T_i \) appears only in the \( e(n) \) term which is much smaller than \( s(n) \).
- Changes to \( K_p \) and \( K_d \) cause much smaller disturbances unless the error is large and/or the rate of change of error is large.
SATURATION AND INTEGRAL ACTION WIND-UP

- In practice, physical constraints of the actuator will limit the value of manipulated variable $m(n)$.
  - For example, a valve has a fully open and fully closed position even if the manipulated variable provides signals beyond these limits.
  - Also, an electric heater can supply only a given maximum amount of heat and cannot supply negative heat.
- If the value of the manipulated variable exceeds the input range limits corresponding to the actuator's output limits, effective feedback control is lost -- good designs should avoid these situations.

EXAMPLE -- BUILDING HEATING SYSTEM

- If extreme environmental conditions take place, the heating system will not be able to keep the desired temperature.
ANALYSIS. Due to the large standing error, the integral term of a PI controller will continue to grow; that is, the value of $s(n)$ in Eq. (4) will increase with each sample time.

- If the environmental conditions come back to normal, our integral term $s(n)$ will still be large until the building temperature will oscillate back and forth around the reference value and settle to a steady state condition.
- The integral term will continue to keep the demanded heat output at its maximum value even though the building temperature is now higher than the desired temperature until the error changes sign.
- The effect is called integral WIND-UP or integral saturation and results in the controller having a poor response when it comes out of a constrained condition.
TECHNIQUES FOR DEALING WITH THE PROBLEMS OF THE INTEGRAL WIND-UP

- 1) fixed limits on integral term;
- 2) stop summation on saturation;
- 3) integral subtraction;
- 4) use of velocity algorithm.

FIXED LIMITS

- Integral summation $s(n)$ cannot accumulate values beyond some fixed limits (maximum and minimum). Beyond this limits the integral summation is reset to the fixed limits.
- These limits can be set to the maximum/minimum values of the manipulated variable -- thus if $s_{\text{max}} = m_{\text{max}}$ and $s_{\text{min}} = m_{\text{min}}$ then the coding in the function PIDControl could be:

```c
void PIDcontrol (mn, en, enold, sn, kp, ki, kd, smax, smin)
float *mn, en, *enold, sn, kp, ki, kd, smax, smin;
{
    if sn > smax
        sn = smax;
    else if sn < smin
        sn = smin;
    *mn = kp*en + ki*sn + kd*(en - *enold);
    *enold = en;
}
```
STOP SUMMATION

• In this method, the value of the integrator sum is frozen during the saturation of the control actuator.

• The scheme can be implemented either: a) by freezing the summation term when the manipulated variable falls outside the range \([m_{\text{min}}, m_{\text{max}}]\) or, b) by the use of a digital input signal from the actuator which indicates that it is at a limit.

• NOTE: The drawback of these techniques is that the value of the integral term, when the system emerges from saturation, does not relate to the dynamics of the plant under full power. Consequently, the controller offset (provided by the integral term) lags behind the offsets required by the plant and load as the set point is reached.

• The stop summation technique gives a better response if the integral term is unfrozen once the error changes. The sign of the error will change before the actuator comes out of saturation. Assume that a positive error drives the actuator towards the upper limit; then the behaviour required is presented in the above table.

<table>
<thead>
<tr>
<th>Actuator</th>
<th>Error</th>
<th>Integral summation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upper limit</td>
<td>+</td>
<td>Stopped</td>
</tr>
<tr>
<td>Upper limit</td>
<td>-</td>
<td>Active</td>
</tr>
<tr>
<td>Normal</td>
<td>+</td>
<td>Active</td>
</tr>
<tr>
<td>Normal</td>
<td>-</td>
<td>Active</td>
</tr>
<tr>
<td>Lower limit</td>
<td>+</td>
<td>Active</td>
</tr>
<tr>
<td>Lower limit</td>
<td>-</td>
<td>stopped</td>
</tr>
</tbody>
</table>

```c
void PIDcontrol ...
{
  stopsummation =
  ((*mn > mnmax) && (en > 0.0))
  || ((*mn < mnmin) && (en < 0.0));
  if (!stopsummation)
    sn = sn + en;
    *mn = kp*en + ki*sn + kd*(en – eold);
    enold = en;
}
```
INTEGRAL SUBTRACTION

• The idea behind this method is that the integral value is decreased by an amount proportional to the difference between the calculated value of the manipulated variable and the maximum value allowable.

• The integral summation expression $s(n) = s(n-1) + e(n)$ is replaced by $s(n) = s(n-1) - K[m(n) - m_{max}] + e(n)$.
  ○ The integral sum is thus decreased by the excess actuation and increased by the error. The rate of decrease is dependent on the choice of the parameter $K$; if it is not properly chosen then a continual saturation/desaturation oscillation can occur.

• The method can be modified to stop the addition of the error part during saturation if a logic signal from the actuator indicating saturation or no saturation is available.
  ○ In this case, the value of the integral sum begins to decrease as soon as the actuator enters saturation and continues to decrease until it comes out of saturation at which point integral summation begins again (method applied while in saturation).

• The benefit of this method is that the system comes out of saturation as quickly as possible; there is, however, no attempt to match the integral term to the requirements of the plant and the value of $K$ must be chosen by experience rather than by reference to the plant characteristics.
VELOCITY ALGORITHM

- In general, the integral wind-up can be avoided by the use of the velocity algorithm. In this algorithm, the integral action is obtained by a summation of the increments in the output device (meaning that the output device accumulates the increments from each step since -- incremental action that is built upon the previous step -- the velocity algorithm itself doesn't have a summation term).
- This summation takes place either at the actuator or at a device connected to the actuator, and it is this device which is subject to limiting. As a result, there is an automatic integral limit which prevents a build-up or error. However, as soon as the error changes sign the actuator will come of its limit and hence at desaturatation the integral cumulative value is lost.
- Note that with cascaded controllers the velocity algorithm will not be able to prevent integral wind-up.