3. Control System

The main control item of the tracker system is the microcontroller Atmega16L loaded with PID controller algorithm.

3.1 Microcontroller :(atmega16L):

The ATmega16 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega16 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed. The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers. The ATmega16 AVR is supported with a full suite of program and system development tools including: C compilers, macro assemblers, program debugger/simulators, in-circuit emulators, and evaluation kits. The boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega16 is a powerful microcontroller that provide highly-flexible and cost-effective solution to many embedded control applications.

3.1.1 MCU Block Daigram:

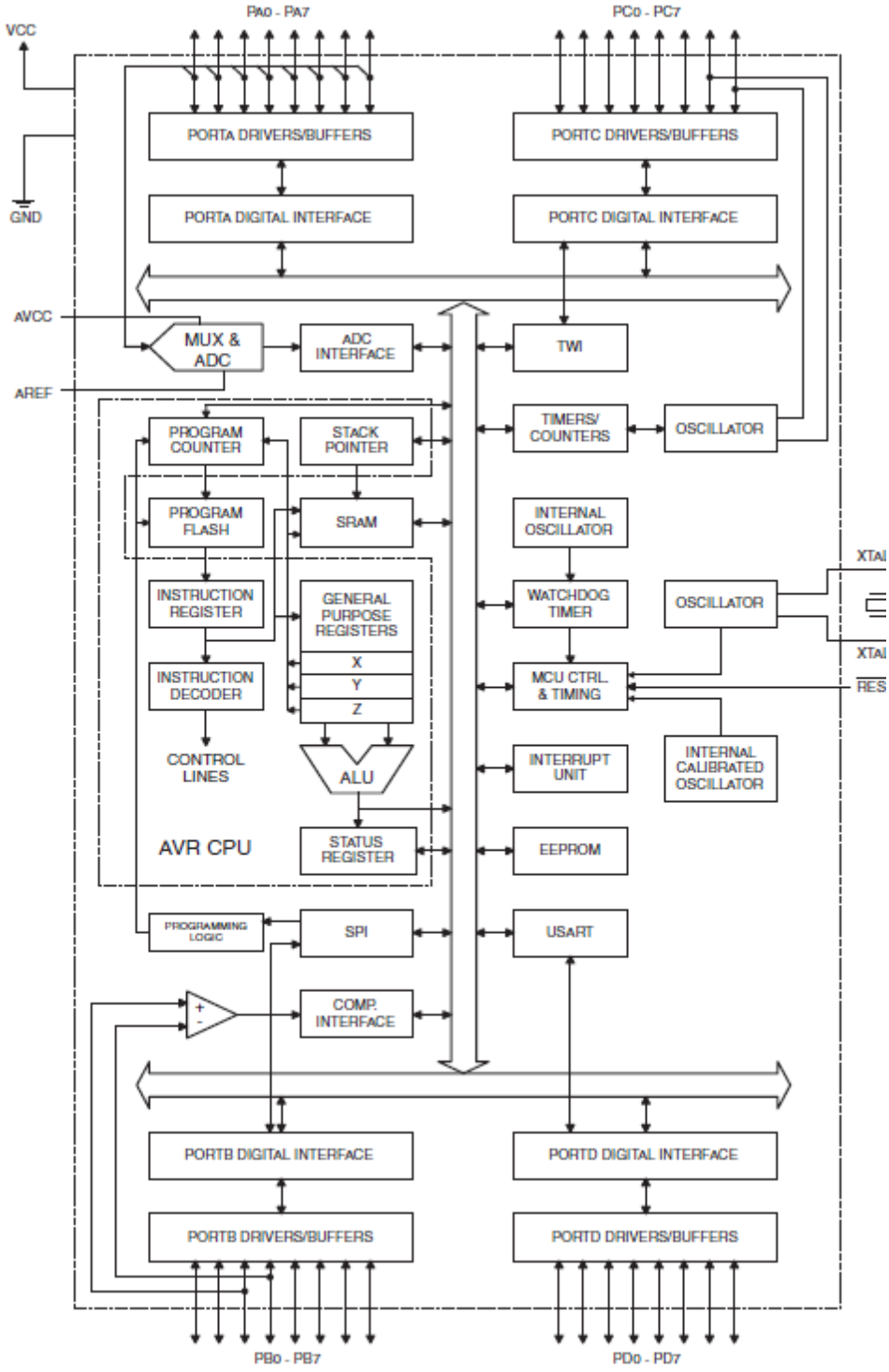

Figure (3.1) atmega16 block diagram

3.1.2 Pin Descriptions:

VCC Digital supply voltage.

GND Ground.

Port A (PA7..PA0) Port A serves as the analog inputs to the A/D Converter.

Port A also serves as an 8-bit bi-directional I/O port, if the A/D Converter is not used. Port pins can provide internal pull-up resistors (selected for each bit). The Port A output buffers have symmetrical drive characteristics with both high sink and source capability. When pins PA0 to PA7 are used as inputs and are externally pulled low, they will source current if the internal pull-up resistors are activated. The Port A pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port B (PB7..PB0) Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active,

even if the clock is not running.

Port C (PC7..PC0) Port C is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port C output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active,

even if the clock is not running. If the JTAG interface is enabled, the pull-up resistors on pins PC5(TDI), PC3(TMS) and PC2(TCK) will be activated even if a reset occurs.

Port D (PD7..PD0) Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active,

even if the clock is not running.

RESET Input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running.Shorter pulses are not guaranteed to generate a reset.

XTAL1 Input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

XTAL2 Output from the inverting Oscillator amplifier.

AVCC is the supply voltage pin for Port A and the A/D Converter. It should be externally connected to VCC, even if the ADC is not used. If the ADC is used, it should be connected to VCC through a low-pass filter.

AREF is the analog reference pin for the A/D Converter. Figure (4.2).

3.1.3 Data Registers I\O:

Port A Data Register – PORTA

Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Data Direction Register – DDRA

Bit	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Input Pins Address – PINA

Bit	7	6	5	4	3	2	1	0	
	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Port B Data Register – PORTB

Bit	7	6	5	4	3	2	1	0	
	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port B Data Direction Register – DDRB

Bit	7	6	5	4	3	2	1	0	
	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port B Input Pins Address – PINB

Bit	7	6	5	4	3	2	1	0	
	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Port C Data Register – PORTC

Bit	7	6	5	4	3	2	1	0	
	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port C Data Direction Register – DDRC

Bit	7	6	5	4	3	2	1	0	
	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port C Input Pins Address – PINC

Bit	7	6	5	4	3	2	1	0	
	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Port D Data Register – PORTD

Bit	7	6	5	4	3	2	1	0	
	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port D Data Direction Register – DDRD

Bit	7	6	5	4	3	2	1	0	
	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port D Input Pins Address – PIND

Bit	7	6	5	4	3	2	1	0	
	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

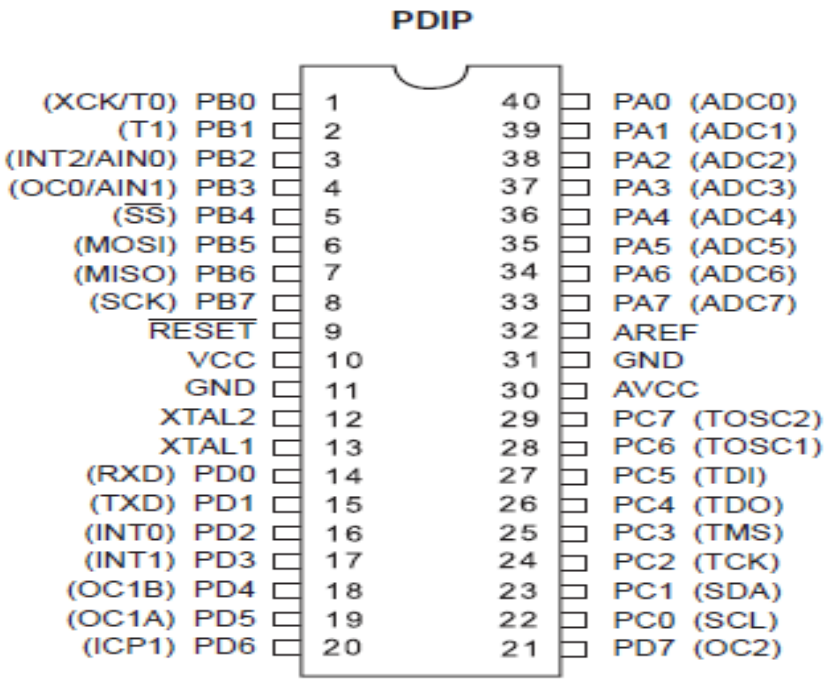

Figure (3.2) Shows PIN out of ATmega16L

3.2 PID controller algorithm:

PID controllers date to 1890s governor design. PID controllers were subsequently developed in automatic ship steering. One of the earliest examples of a PID-type controller was developed by Elmer Sperry in 1911, while the first published theoretical analysis of a PID controller was by Russian American engineer Nicolas Minorsky, in (Minorsky 1922). Minorsky was designing automatic steering systems for the US Navy, and based his analysis on observations of a helmsman, observing that the helmsman controlled the ship not only based on the current error, but also on past error and current rate of change; this was then made mathematical by Minorsky. His goal was stability, not general control, which significantly simplified the problem. While proportional control provides stability against small disturbances, it was insufficient for dealing with a steady disturbance, notably a stiff gale (due to droop), which required adding the integral term. Finally, the derivative term was added to improve control. [8]

Trials were carried out on the USS *New Mexico*, with the controller controlling the *angular velocity* (not angle) of the rudder. PI control yielded sustained yaw (angular error) of ±2°, while adding D yielded yaw of ±1/6°, better than most helmsmen could achieve. [8]

The Navy ultimately did not adopt the system, due to resistance by personnel. Similar work was carried out and published by several others in the 1930s. [8]

In the early history of automatic process control the PID controller was implemented as a mechanical device. These mechanical controllers used a

lever, spring and a mass and were often energized by compressed air. These pneumatic controllers were once the industry standard. [8]

Electronic analog controllers can be made from a solid-state or tube amplifier, a capacitor and a resistor. Electronic analog PID control loops were often found within more complex electronic systems, for example, the head positioning of a disk drive, the power conditioning of a power supply, or even the movement-detection circuit of a modern seismometer. Nowadays, electronic controllers have largely been replaced by digital controllers implemented with microcontrollers or FPGAs. [8]

Most modern PID controllers in industry are implemented in programmable logic controllers (PLCs) or as a panel-mounted digital controller. Software implementations have the advantages that they are relatively cheap and are flexible with respect to the implementation of the PID algorithm. PID temperature controllers are applied in industrial ovens, plastics injection machinery, hot stamping machines and packing industry. [8]

Variable voltages may be applied by the time proportioning form of pulse-width modulation (PWM)—a cycle time is fixed, and variation is achieved by varying the proportion of the time during this cycle that the controller outputs +1 (or −1) instead of 0. On a digital system the possible proportions are discrete—e.g., increments of 0.1 second within a 2 second cycle time yields 20 possible steps: percentage increments of 5%; so there is a discretization error, but for high enough time resolution this yields satisfactory performance. [8]

This algorithm is being used to reduce the time required to minimize the error to be as close as the reference value and also for accurate calculation of the number of steps required to the correct position of the solar panel.

The PID control scheme, figure (4.3), is named after its three correcting terms, whose sum constitutes the manipulated variable (MV). The proportional, integral, and derivative terms are summed to calculate the output of the PID controller. Defining $u(t)$as the controller output, the final form of the PID algorithm is:

$$\mathrm{u}(t) = \mathrm{MV}(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{d}{dt} e(t)$$

where

K_p: Proportional gain, a tuning parameter

K_i: Integral gain, a tuning parameter

K_d: Derivative gain, a tuning parameter

e(t) Error = r(t)setpoint – y(t)measured value

t: Time or instantaneous time (the present)

τ: Variable of integration; takes on values from time 0 to the present t

.

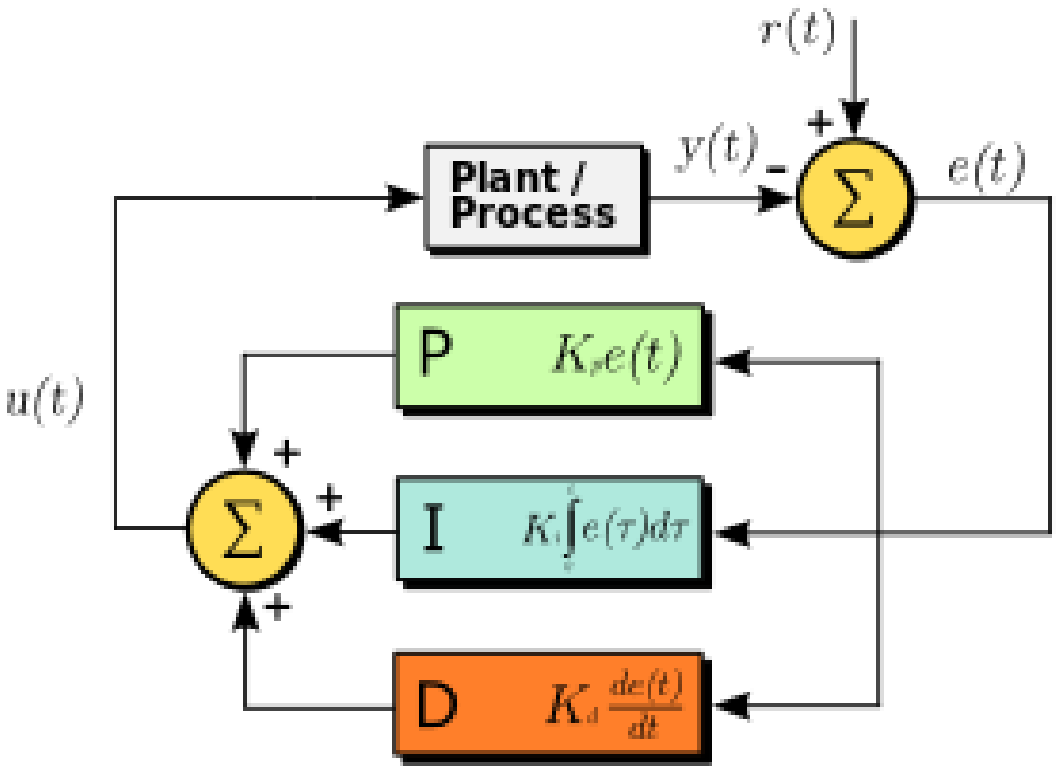

Figure (3.3) System representation of PID equation

3.2.1 Proportional term:

The proportional term produces an output value that is proportional to the current error value. The proportional response can be adjusted by multiplying the error by a constant K_p, called the proportional gain constant.

The proportional term is given by:

$$P_{\text{out}} = K_p\, e(t)$$

A high proportional gain results in a large change in the output for a given change in the error, figure (4.4). If the proportional gain is too high, the system can become unstable. In contrast, a small gain results in a small

output response to a large input error, and a less responsive or less sensitive controller. If the proportional gain is too low, the control action may be too small when responding to system disturbances. Tuning theory and industrial practice indicate that the proportional term should contribute the bulk of the output change. [9]

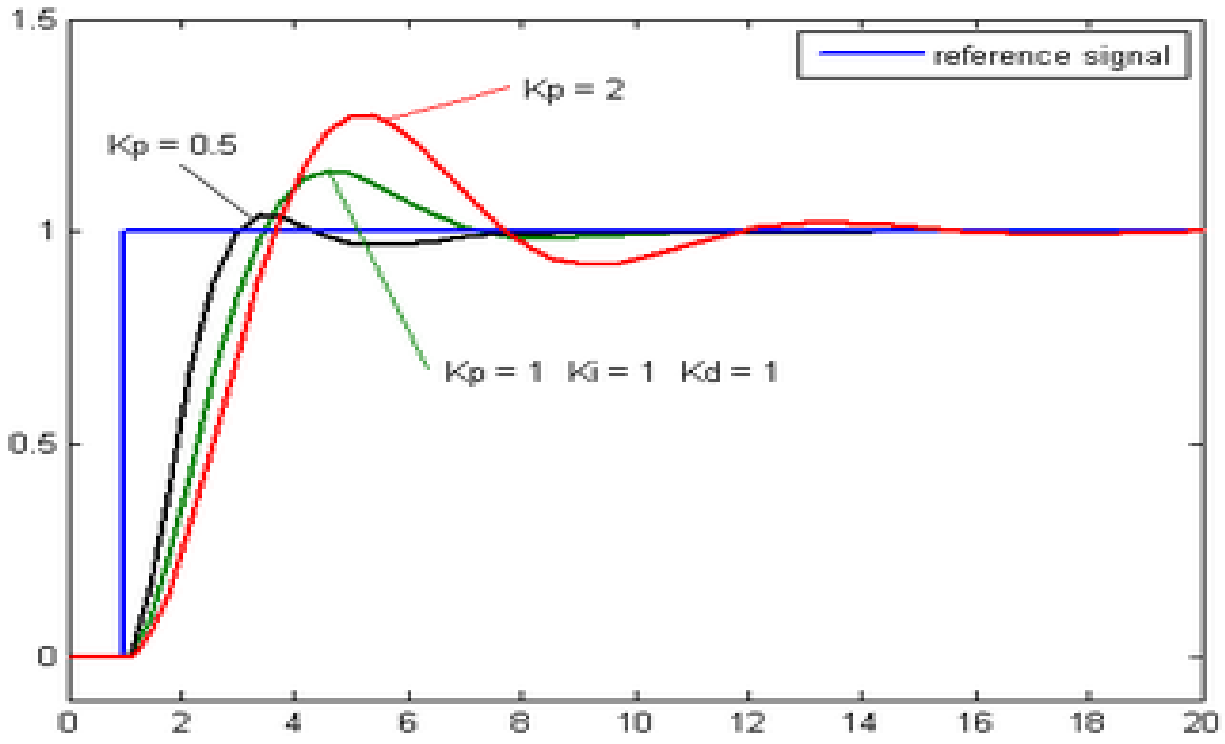

Figure (3.4) Plot of y(t) vs time, for three values of K_p (K_i and K_d held constant)

3.2.2 Integral term:

The contribution from the integral term is proportional to both the magnitude of the error and the duration of the error. The integral in a PID controller is the sum of the instantaneous error over time and gives the accumulated offset that should have been corrected previously, figure (4.5). The

accumulated error is then multiplied by the integral gain (K_i) and added to the controller output.

The integral term is given by:

$$I_{\text{out}} = K_i \int_0^t e(\tau)\, d\tau$$

The integral term accelerates the movement of the process towards setpoint and eliminates the residual steady-state error that occurs with a pure proportional controller. However, since the integral term responds to accumulated errors from the past, it can cause the present value to overshoot the setpoint value. [9]

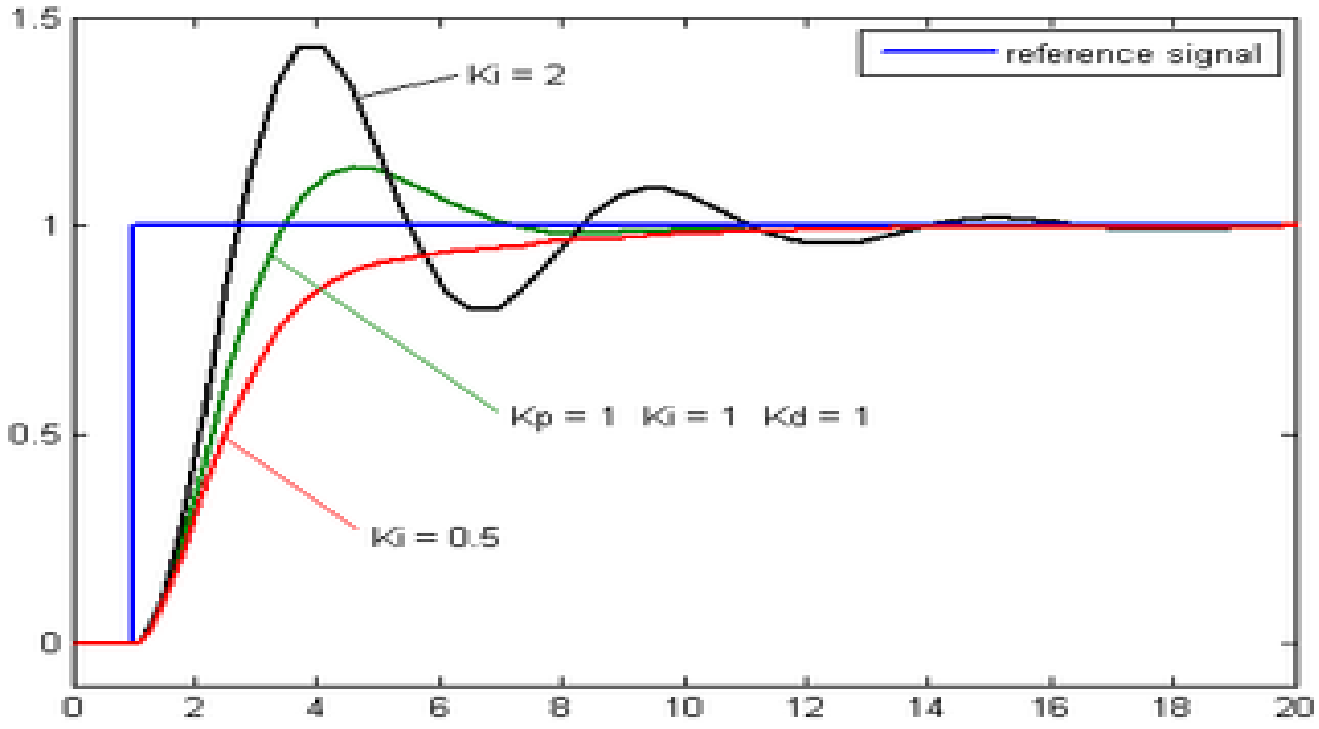

Figure (3.5) Plot of y(t) vs time, for three values of K_i (K_p and K_d held constant)

3.2.3 Derivative term:

The derivative of the process error is calculated by determining the slope of the error over time and multiplying this rate of change by the derivative gain K_d. The magnitude of the contribution of the derivative term to the overall control action is termed the derivative gain, K_d.

The derivative term is given by:

$$D_{\text{out}} = K_d \frac{d}{dt} e(t)$$

Derivative action predicts system behavior and thus improves settling time and stability of the system. Derivative action, however, is seldom used in practice because of its inherent sensitivity to measurement noise, figure (4.6). If this noise is severe enough, the derivative action will be erratic and actually degrade control performance. Large, sudden changes in the measured error (which typically occur when the set point is changed) cause a sudden, large control action stemming from the derivative term, which goes under the name of *derivative kick*. This problem can be ameliorated to a degree if the measured error is passed through a linear low-pass filter or a nonlinear but simple median filter. [9]

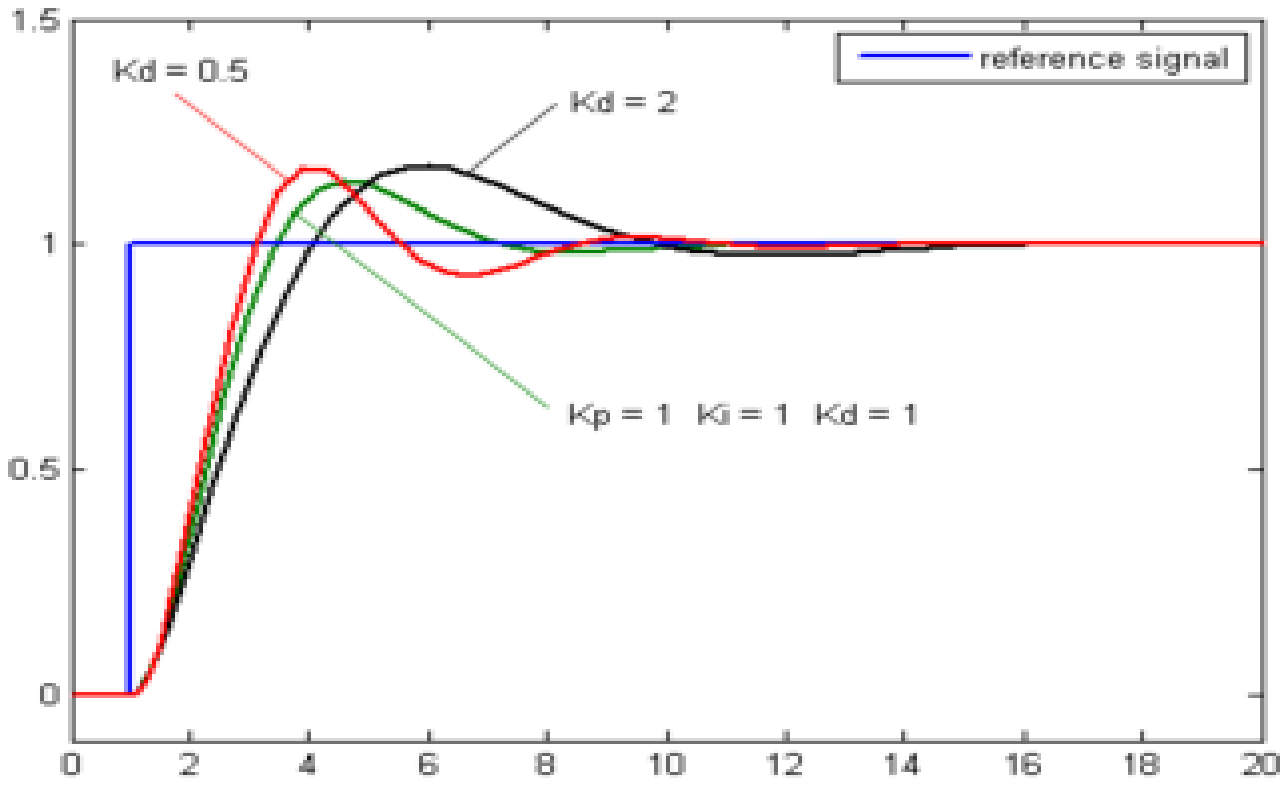

Figure (3.6) Plot of PV vs time, for three values of K_d (K_p and K_i held constant)

3.2.4 Loop tuning:

Tuning a control loop is the adjustment of its control parameters (proportional band/gain, integral gain/reset, derivative gain/rate) to the optimum values for the desired control response. Stability (bounded oscillation) is a basic requirement, but beyond that, different systems have different behavior, different applications have different requirements, and requirements may conflict with one another. [9]

PID tuning is a difficult problem, even though there are only three parameters and in principle is simple to describe, because it must satisfy complex criteria within the limitations of PID control. There are accordingly various methods for loop tuning. [9]

Designing and tuning a PID controller appears to be conceptually intuitive, but can be hard in practice, if multiple (and often conflicting) objectives such as short transient and high stability are to be achieved. Usually, initial designs need to be adjusted repeatedly through computer simulations until the closed-loop system performs or compromises as desired. [9]

Some processes have a degree of nonlinearity and so parameters that work well at full-load conditions don't work when the process is starting up from no-load; this can be corrected by gain scheduling (using different parameters in different operating regions). PID controllers often provide acceptable control using default tunings, but performance can generally be improved by careful tuning, and performance may be unacceptable with poor tuning. [9]

3.2.5 Stability:

If the PID controller parameters (the gains of the proportional, integral and derivative terms) are chosen incorrectly, the controlled process input can be unstable, i.e., its output diverges, with or without oscillation, and is limited only by saturation or mechanical breakage. Instability is caused by *excess* gain, particularly in the presence of significant lag. [9]

Generally, stabilization of response is required and the process must not oscillate for any combination of process conditions and setpoints, though sometimes marginal stability (bounded oscillation) is acceptable or desired. [9]

3.2.6 Optimum behavior:

The optimum behavior on a process change or setpoint change varies depending on the application.

Two basic requirements are regulation (disturbance rejection – staying at a given setpoint) and command tracking (implementing setpoint changes) – these refer to how well the controlled variable tracks the desired value. Specific criteria for command tracking include rise time and settling time. Some processes must not allow an overshoot of the process variable beyond the setpoint if, for example, this would be unsafe. Other processes must minimize the energy expended in reaching a new set point. [9]

3.2.7 Overview of tuning methods:

There are several methods for tuning a PID loop. The most effective methods generally involve the development of some form of process model, then choosing P, I, and D based on the dynamic model parameters. Manual tuning methods can be relatively inefficient, particularly if the loops have response times on the order of minutes or longer. [9]

The choice of method will depend largely on whether or not the loop can be taken "offline" for tuning, and on the response time of the system. If the system can be taken offline, the best tuning method often involves subjecting the system to a step change in input, measuring the output as a function of time, and using this response to determine the control parameters. [9]

Method such as :

I. Manual tuning:

If the system must remain online, one tuning method is to first set $K_i$and $K_d$values to zero. Increase the $K_p$until the output of the loop oscillates, then the $K_p$should be set to approximately half of that value for a "quarter amplitude decay" type response. Then increase $K_i$until any offset is corrected in sufficient time for the process. However, too much $K_i$will cause instability. Finally, increase K_d, if required, until the loop is acceptably quick to reach its reference after a load disturbance. However, too much $K_d$will cause excessive response and overshoot. A fast PID loop tuning usually overshoots slightly to reach the setpoint more quickly; however, some systems cannot accept overshoot, in which case an over-damped closed-loop system is required, which will require a $K_p$setting significantly less than half that of the $K_p$setting that was causing oscillation. [9]

Effects of *increasing* a parameter independently

Parameter	**Rise time**	**Overshoot**	**Settling time**	**Steady-state error**	**Stability**[9]
K_p	Decrease	Increase	Small change	Decrease	Degrade
K_i	Decrease	Increase	Increase	Eliminate	Degrade
K_d	Minor change	Decrease	Decrease	No effect in theory	Improve if $K_d$small

II. Ziegler–Nichols method:

Another heuristic tuning method is formally known as the Ziegler–Nichols method, introduced by John G. Ziegler and Nathaniel B. Nichols in the 1940s. As in the method above, the $K_i$and $K_d$gains are first set to zero. The proportional gain is increased until it reaches the ultimate gain, K_u, at which the output of the loop starts to oscillate. $K_u$and the oscillation period $P_u$are used to set the gains as shown: [9]

Ziegler–Nichols method

Control Type	K_p	K_i	K_d
P	$0.50K_u$	-	-
PI	$0.45K_u$	$1.2K_p/P_u$	-
PID	$0.60K_u$	$2K_p/P_u$	$K_pP_u/8$

These gains apply to the ideal, parallel form of the PID controller. When applied to the standard PID form, the integral and derivative time parameters $T_i$and $T_d$are only dependent on the oscillation period P_u. Please see the section "Alternative nomenclature and PID forms". [9]

3.2.8 Limitations of PID control:

While PID controllers are applicable to many control problems, and often perform satisfactorily without any improvements or even tuning, they can perform poorly in some applications, and do not in general provide optimal control. The fundamental difficulty with PID control is that it is a feed*back* system, with *constant* parameters, and no direct knowledge of the process, and thus overall performance is reactive and a compromise – while PID control is the best controller with no model of the process, better performance can be obtained by incorporating a model of the process. [9]

The most significant improvement is to incorporate feed-forward control with knowledge about the system, and using the PID only to control error. Alternatively, PIDs can be modified in more minor ways, such as by changing the parameters (either gain scheduling in different use cases or adaptively modifying them based on performance), improving measurement (higher sampling rate, precision, and accuracy, and low-pass filtering if necessary), or cascading multiple PID controllers. [9]