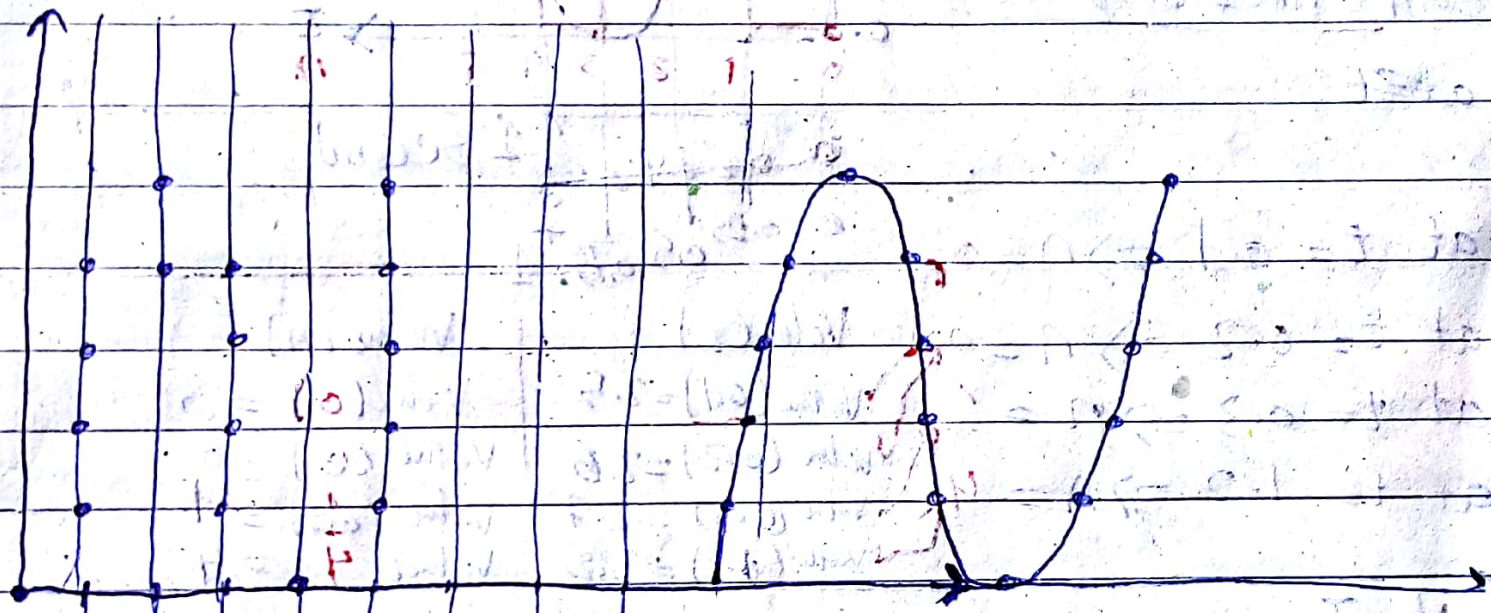
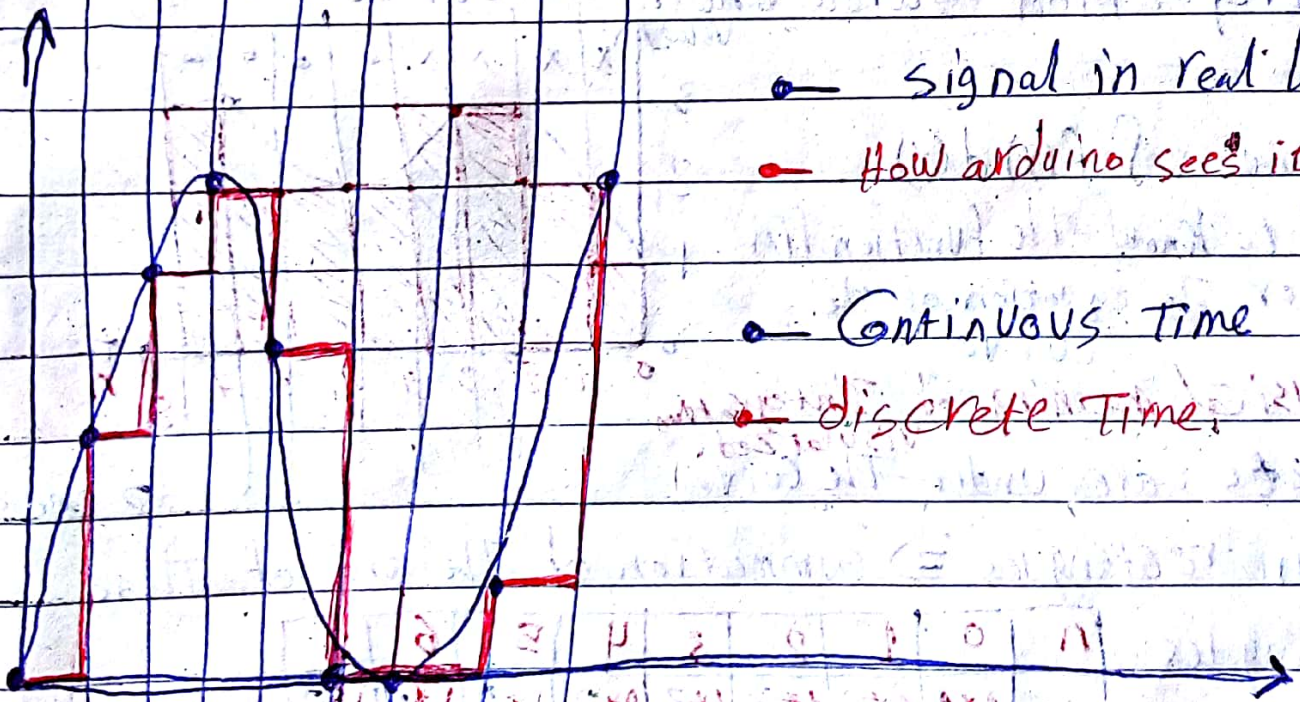
About Signals,



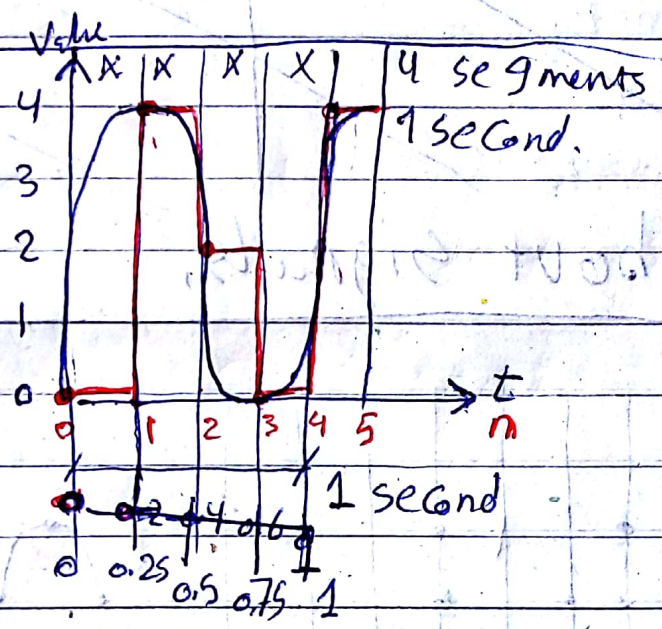


- signal in real life
- How arduino sees it.
- Continuous Time
- discrete Time.

n → no. segments → Sample time.

Math operations in Continuous/ discrete times

1] Current Value



1 second = 4 segments

each segments = 0.25 sec

Sample time = 0.25 sec

~~at t~~

at t = 0.1 ⇒ n = 0

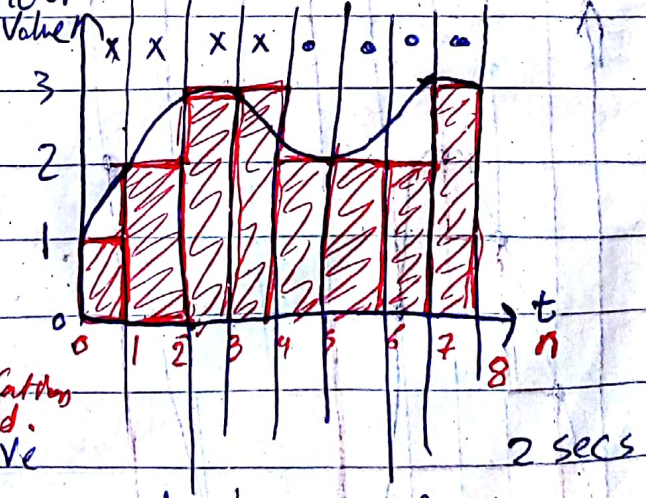
at t = 0.2 ⇒ n = 0

at t = 0.3 ⇒ n = 1

at t = 1.2 ⇒ n = 4

Value(t)	Value(n)
Value(0.1) = 2.5	Value(0) = 0
Value(0.2) = 2.6	Value(0) = 0
Value(0.3) = 2.7	Value(1) = 4
Value(1.2) = 3.8	Value(4) = 4

2] Integration "area under The Curve"



Continuous: $\int f(x)\,d(x)$

need to know The function f(x) or The equation of the Curve

→ Physical meaning of Integration visualized.

discrete: area under The Curve divided in rectangles ⇒ summation of The area of These rectangles.

n	0	1	2	3	4	5	6	7
area	1*1	1*2	1*3	1*3	1*2	1*2	1*2	1*3

n: Value area = summation of all Values

= 1+2+3+3+2+2+2+3 = 18

Slope of straight line

$= m = \frac{y - y_1}{x - x_1} = \frac{y_{last}}{}$

$= \frac{y_{current} - y_{last}}{x_{current} - x_{last}}$

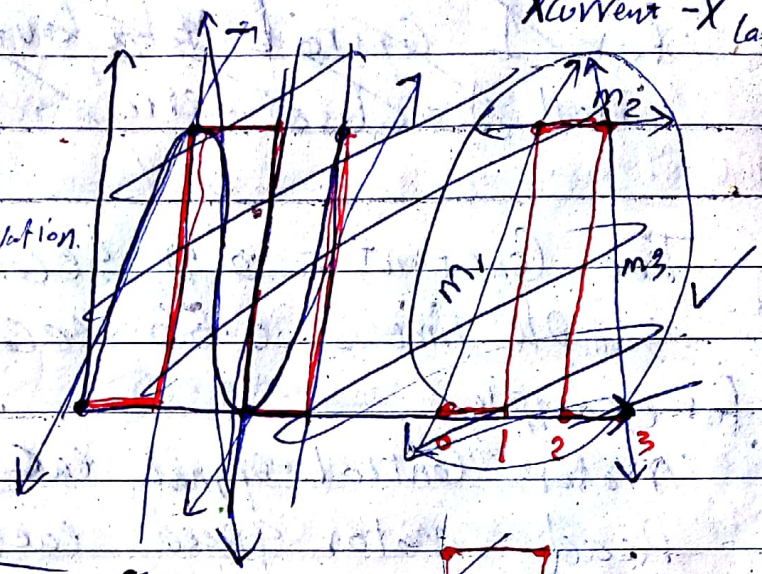
3] Derivation: The Rate of Change

Continuous $= \frac{df(x)}{dx}$

need to know the function or the line equation

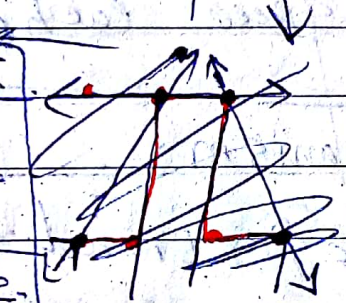
discrete:

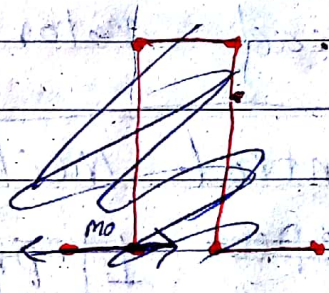
Rate of Change = slope of the line

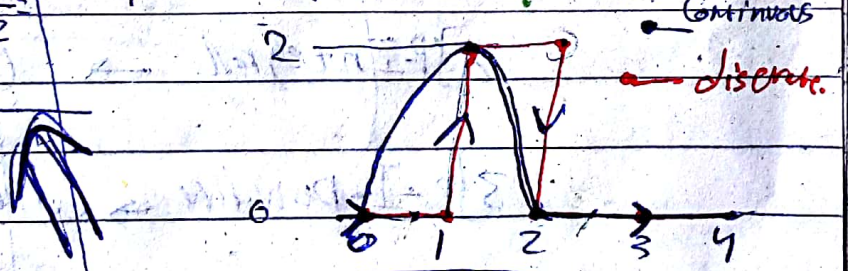


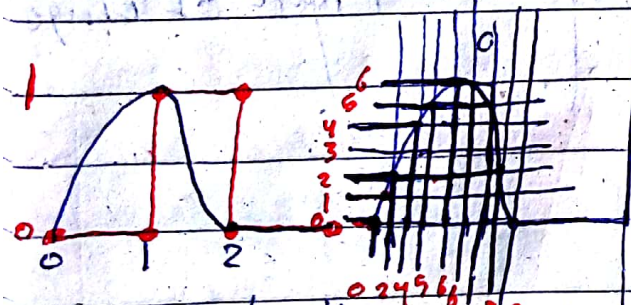
n	0	1	2	3
Current	0	2	0	0
last	0	0	2	0
Rate of Change "slope m"	0	$\frac{y-x_1}{2-0}$ $\frac{1-0}{2-1}$ $=2$	$\frac{0-2}{2-1}$ $=-2$	$\frac{0-0}{3-2}$ 0

No change at the Beginning









n	0	1	2	3
Current	0	2	0	0
last	0	0	2	0
	No change	Change	Change	No change

n	0	1	2
value	0	1	0

n	0	1	2	3	4	5	6	7	8
Value	0	2/6	4/6	5/6	6/6	6/6	5/6	2/6	0/6

but if 1) No. segments increases

2) No. values increases

The accuracy increases.

Continuous is much more accurate

~~but if segments increased for the same amount of time the accuracy~~

Control mechanism in discrete times:

1] ~~Control~~ Proportional Control based on current value. "P Controller"

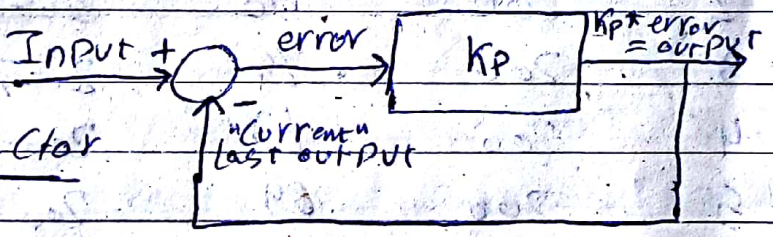
Variables

Input: "we call it set point"

Output: "~~we call it control signal~~"

Kp: proportional term. "used to generate output based on error"

error: difference between last output & Input.



Kp: is just a scale-factor of the error.

error = Input - last output = set point - current

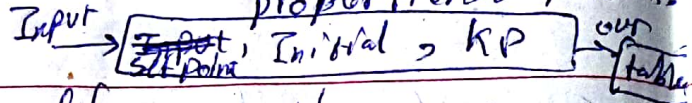
output = error * Kp

idea:

if error is large → out is large

if error is small → out is small

Py Code for proportional problem.



Example: to see How it affects the output.

1

Set Point "Input" = 200

initial Value = 0 "current"

Kp = 0.5 "Randomly selected

n	set Point	Current	error	output
0	200	0	200-0=200	200*0.5= 100
1	200	100	200-100=100	100*0.5 = 50
2	200	50	200-50=150	150*0.5 = 75
3	200	75	200-75=125	125*0.5 = 63 "Int
4	200	63	200-63=137	137*0.5= 69 "Int
5	200	69	200-69=131	131*0.5= 66 "Int
6	200	66	200-66=134	134*0.5 = 67
7	200	67	200-67=133	133*0.5 = 67 "Int

system is now stable but at 67 instead of 200 !

system is at steady state with error [200 - 67]

We can choose different Kp to solve the problem

Problem: steady state error

*So if program stopped the noise will affect the out put which will Change the output, The goal is to keep the out put stable in this noisy environment.

2]

Set Point "Input" = 200

Initial Value "Current" = 0

$K_P = 2.5$

n	Set Point	Current	error	output
0	200	0	200 − 0 = 200	200 * 2.5 = 500
1	200	500	200 − 500 = −300	−300 * 2.5 = −750
2	200	−750	200 − (−750) = +550	+550 * 2.5 = 1375
3	200	1375	200 − 1375 = −1175	−1175 * 2.5 = −2937.5

Range become wider not narrower ⚠

3]

$K_P = 1.5$

n	Set point	Current	error	output
0	200	0	200 − 0 = 200	200 * 1.5 = 300
1	200	300	200 − 300 = −100	−100 * 1.5 = −150
2	200	−150	200 − (−150) = 350	350 * 1.5 = 525
3	200	525	200 − 525 = −325	−325 * 1.5 = −162.5

Range become wider ⚠

4]

$K_P = 1$

n	Set Point	Current	error	output
0	200	0	200 − 0 = 200	200 * 1 = 200
1	200	200	200 − 200 = 0	0 * 1 = 0
2	200	0	200 − 0 = 200	200 * 1 = 200
3	200	200	200 − 200 = 0	0 * 1 = 0

System won't reach steady state as it keep oscillate ⚠ as it should generate 200 on each output because in real life there are alot of noise*

We can't just stop program if output = 200

5]

Set Point "Input" = 200

initial Value "Current" = 0

Kp = 0.9

n → 17, 18

* upper range
x lower range
predict = (80+111)/2 ≈ 96

Range becomes narrower.

n	Set Point	Current	error	output
0	200	0	200-0=200	200*0.9=180.
1	200	180	200-180=20	20*0.9=18 x
2	200	18	200-18=182.	182*0.9=164. "Int"
3	200	164	200-164=36 x	36*0.9=32 x "Int"
4	200	32	200-32=168.	168*0.9=151. "Int"
5	200	151	200-151=49 x	49*0.9=44 x "Int"
6	200	44	200-44=156.	156*0.9=140. "Int"
7	200	140	200-140=60 x	60*0.9=54 x
8	200	54	200-54=146.	146*0.9=131. "Int"
9	200	131	200-131=69 x	69*0.9=62 x "Int"
10	200	62	200-62=138.	138*0.9=124. "Int"
11	200	124	200-124=76 x	76*0.9=68 x "Int"
12	200	68	200-68=132.	132*0.9=119. "Int"
13	200	119	200-119=81 x	81*0.9=73 x "Int"
14	200	73	200-73=127.	127*0.9=114. "Int"
15	200	114	200-114=86 x	86*0.9=77 x "Int"
16	200	77	200-77=123.	123*0.9=111. "Int"
17	200	111	200-111=89 x	89*0.9=80. "Int"
18	200	80	200-80=120.	120*0.9=108

n	setpoint	current	error	output	
19	200	108	200-108=92	92*0.9=83	"Int
20	200	83	200-83=117	117*0.9=105	"Int"
21	200	105	200-105=95	95*0.9=86	"Int"
22	200	86	200-86=114	114*0.9=102	"Int"
23	200	103	200-103=97	97*0.9=87	"Int"
24	200	87	200-87=113	113*0.9=102	"Int"
25	200	102	200-102=98	98*0.9=88	"Int"
26	200	88	200-88=112	112*0.9=101	"Int"
27	200	101	200-101=99	99*0.9=89	"Int"
28	200	89	200-89=111	111*0.9=100	"Int"
29	200	100	200-100=100	100*0.9=90	
30	200	90	200-90=110	110*0.9=99	
31	200	99	200-99=101	101*0.9=91	
32	200	91	200-91=109	109*0.9=98	"Int"
33	200	98	200-98=102	102*0.9=92	"Int"
34	200	92	200-92=108	108*0.9=97	"Int"
35	200	97	200-97=103	103*0.9=93	"Int"
36	200	93	200-93=107	107*0.9=96	"Int"
37	200	96	200-96=104	104*0.9=94	"Int"
38	200	94	200-94=106	106*0.9=95	"Int"
39	200	95	200-95=105	105*0.9=95	"Int"
40	200	95	200-95=105	105*0.9=95	"Int"

system reached steady state

also it took longer to become steady.

Steady state error = 200-95 = 105

Control mechanism

2] proportional - Integral Control based on current values & last values.

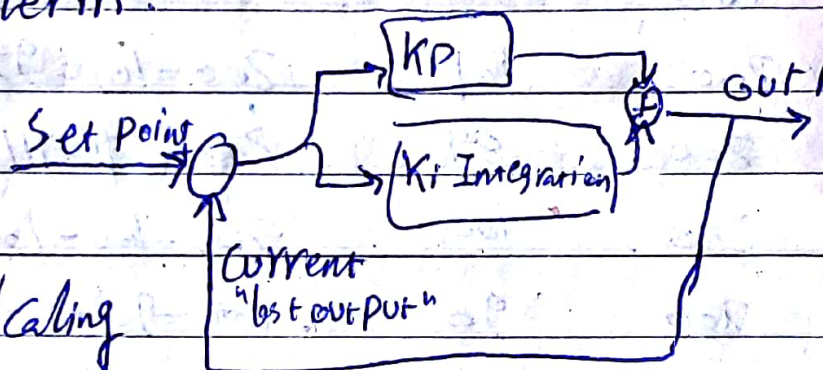
Variables:

Input: "set point"

output: "

Kp : propotional Term

Ki : Integral Term.



both Kp & Ki are scaling factors for different type of errors.

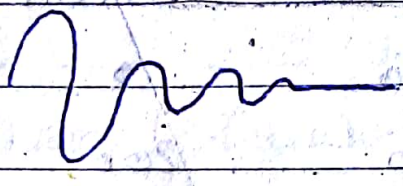
error = set point - current

total error += error

output = Kp * error + Ki * total error

Physical meaning:

if we applied Integration Control for sin wave it will dampen over Time & become steady at certain point.



if we used the proportional Control to achieve oscilations like a sin wave it will make it dampen to a certain point.

Pick Kp, Ki by trail & error!

for these examples Kp, Ki Increase or decrease step = 0.1

examples Revisit:

1] system is stable with 0 steady state error!

Set Point = 200

Initial = 0

Kp = 0.5

Start Ki = Kp = 0.5 "you can start from 0.1 if you want"

n	SetPoint	current error	totalerror	output
0	200	0	200-0=200	200	200*0.5 + 200*0.5 = 200
1	200	200	200-200=0	200	0*0.5 + 200*0.5 = 100
2	200	100	200-100=100	300	100*0.5 + 300*0.5 = 200
3	200	200	200-200=0	300	0*0.5 + 300*0.5 = 150
4	200	150	200-150=50	350	50*0.5 + 350*0.5 = 200
5	200	200	200-200=0	350	0*0.5 + 350*0.5 = 175
6	200	175	200-175=25	375	25*0.5 + 375*0.5 = 200
7	200	200	200-200=0	375	0*0.5 + 375*0.5 = 188 "Int"
8	200	188	200-188=12	387	12*0.5 + 387*0.5 = 200 "Int"
9	200	200	200-200=0	387	0*0.5 + 387*0.5 = 194 "Int"
10	200	194	200-194=6	393	6*0.5 + 393*0.5 = 200 "Int"
11	200	200	200-200=0	393	0*0.5 + 393*0.5 = 197 "Int"
12	200	197	200-197=3	396	3*0.5 + 396*0.5 = 200 "Int"
13	200	200	200-200=0	396	0*0.5 + 396*0.5 = 198
14	200	198	200-198=2	398	2*0.5 + 398*0.5 = 200 "Int"
15	200	200	200-200=0	398	0*0.5 + 398*0.5 = 199
16	200	199	200-199=1	399	1*0.5 + 399*0.5 = 200 "Int"
17	200	200	200-200=0	399	0*0.5 + 399*0.5 = 200 "Int"
18	200	200	200-200=0	399	0*0.5 + 399*0.5 = 200 "Int"

2] Kp is not appropriate! we need Kp that achieves: 1] steady state + error
or 2] oscilated system like no. [4]

3] same!

4] oscilated system: ✓

Note: these examples to examine & practice
The goal is not choosing best Kp & Ki!

Kp set point = 200 intitial = 0 $K_p = 1$, $K_i = K_p = 1$

n	setPoint	current error	total error	output
0	200	0	200-0=200	200	200*1 + 200*1 = 400
1	200	400	200-400=-200	0	-200*1 + 0*1 = -200
2	200	-200	200-(-200)=400	400	400*1 + 400*1 = 800
3	200	800	200-800=-600	-200	-600*1 - 200*1 = -800

Range become wider!

decrease Kp one step $K_i = 0.9$

n	setPoint	current error	total error	output
0	200	0	200-0=200	200	200*1 + 200*0.9 = 380
1	200	380	200-380=-180	20	-180*1 + 20 = -160
2	200	-160	200-(-160)=360	340	360 + 340 = 700
3	200	700	200-700=-500	-160	-500*1 + (-160) = -660

Range become wider

it's better to try decreasing Kp

3] Set Point = 200 $K_p = 0.9$

Initial = 0 $K_i = 0.9$

n	Setpoint	Current error	total error	output
0	200	0	200-0=200	200	200*0.9 + 200*0.9 = 360
1	200	360	200-360=-160	40	-160*0.9 + 40*0.9 = -108
2	200	-108	200-(-108)=308	348	348*0.9 + 348*0.9 = 594
3	200	594	200-594=-394	-43	-394*0.9 + -43*0.9 = -394

Range getting wider

it's better to start small

may be you can test K_p till you find the maximum gain "The one with the oscillations" then restarting program with ~~$K_p = \frac{1}{2}$ max K_p & K_i = small~~ or ~~$K_i = \frac{1}{2}$ max~~

1] $K_i = K_p = \frac{1}{2}$ max K_p, Then increase K_i

or

2] $K_p = \frac{1}{2}$ max K_p & start K_i small Then increase

---

Note:- proportional term eliminates steady state error.

Test numbers to see how the affect

eXamples ReVisit.

Try to make it faster

record 18 steps

1] Set Point = 200

Initial = 0

Kp = 0.5 , Ki = 0.7

n	SetPoint	Current	error	total error	output
0	200	0	200-0=200	200	200*0.5+200*0.7 = 450
1	200	450	200-450=-250	-50	-250*0.5+-50*0.7=-160
2	200	-160	200+160=360	310	360*0.5+310*0.7=

Range get wider

Kp = 0.5 , Ki = 0.6

n	setPoint	Current	error	total error	output
0	200	0	200-0=200	200	200*0.5+200*0.6 = 220
1	200	220	200-220=-20	180	-20*0.5+180*0.6 = 98
2	200	98	200-98=102	282	102*0.5+282*0.6 ≃ 220
3	200	220	200-220=-20	262	-20*0.5+262*0.6 ≃ 147
4	200	147	200-147=53	315	53*0.5+315*0.6 ≃ 216
5	200	216	200-216=-16	299	-16*0.5+299*0.6 ≃ 171
6	200	171	200-171=29	328	29*0.5+328*0.6 ≃ 211
7	200	211	200-211=-11	317	-11*0.5+317*0.6 ≃ 184
8	200	184	200-184=16	333	16*0.5+333*0.6 ≃ 208
9	200	208	200-208=-8	325	-8*0.5+325*0.6 ≃ 191
10	200	191	200-191=9	334	9*0.5+334*0.6 ≃ 205
11	200	205	200-205=-5	329	-5*0.5+329*0.6 ≃ 195
12	200	195	200-195=5	334	5*0.5+334*0.6 ≃ 203
13	200	203	200-203=-3	331	-3*0.5+331*0.6 ≃ 197
14	200	197	200-197=3	334	3*0.5+334*0.6 ≃ 202

n	Set Point	current	error	total error	OUTPUT
15	200	202	200-202=-2	332	-2*0.5+332*0.6≅198
16	200	198	200-198=2	334	2*0.5+334*0.6≅201
17	200	201	200-201=-1	333	-1*0.5+333*0.6≅200
18	200	200	200-200=0	333	0*0.5+333*0.6≅200

not faster! reGrd = 17

Kp=0.4 , Ki = 0.7

n	setPoint	current	Error	total error	output
0	200	0	200-0=200	200	200*0.4+200*0.7=220
1	200	220	200-220=-20	180	-20*0.4+180*0.7=118
2	200	118	200-118=82	262	82*0.4+262*0.7≅174
3	200	174	200-174=26	288	26*0.4+288*0.7≅212
4	200	212	200-212=-12	276	-12*0.4+276*0.7≅188
5	200	188	200-188=12	288	12*0.4+288*0.7≅206
6	200	206	200-206=-6	282	-6*0.4+282*0.7=195
7	200	195	200-195=5	287	5*0.4+287*0.7≅203
8	200	203	200-203=-3	284	-3*0.4+284*0.7≅198
9	200	198	200-198=2	286	2*0.4+286*0.7=201
10	200	201	200-201=-1	285	-1*0.4+285*0.7=199
11	200	199	200-199=1	286	1*0.4+286*0.7=201
12	200	201	200-201=-1	285	-1*0.4+285*0.7=199

steady with error = ±1 reGrd=10

faster

$K_P = 0.3$, $K_i = 0.8$

n	set Point	Current error	total error	output
0	200	0	200-0=200	200	200*.3 + 200*.8 = 220
1	200	220	200-220=-20	180	-20*0.3 + 180*0.8 = 138
2	200	138	200-138=62	242	62*0.3 + 242*0.8 = 212
3	200	212	200-212=-12	230	-12*0.3 + 230*0.8 = 180
4	200	180	200-180=20	250	20*0.3 + 250*0.8 = 206
5	200	206	200-206=-6	244	-6*0.3 + 244*0.8 = 193
6	200	193	200-193=7	251	7*0.3 + 251*0.8 = 203
7	200	203	200-203=-3	248	-3*0.3 + 248*0.8 = 198
8	200	198	200-198=2	250	2*0.3 + 250*0.8 = 201
9	200	201	200-201=-1	249	-1*0.3 + 249*0.8 = 199
10	200	199	200-199=1	250	1*0.3 + 250*0.8 = 200
11	200	200	200-200=0	250	0*0.3 + 250*0.8 = 200

System is stable
at record [10]
with No steady state error

last record with no error
= [16]

This is [faster]

record didn't change but check
The first out put PI → 261
that what we call PID → 222
more damped response reduce over shoot
Control Mechanism

Become steady at the same record with no steady state error

3] Proportional - Integral - Derivative

based on current values, last values & Rate of change of values

set Point = 200 / $K_p = 0.3$ / $K_d = 0.01$

Initial = 0 / $K_I = 0.8$

n	Set Point	current	error	total error	delta error	out put
0	200	0	200-0=200	200		200*0.3+200*0.8+200*0.01 = 222
1	200	222	200-222=-22	178	-22-200=-222	-22*0.3+178*0.8+-222*0.01 ≈ 134
2	200	134	200-134=66	244	66-(-22)=88	66*0.3+244*0.8+88*0.01 ≈ 216
3	200	216	200-216=-16	228	-16-66=-82	-16*0.3+228*0.8-82*0.01 ≈ 177
4	200	177	200-177=23	251	23-(-16)=39	23*0.3+251*0.8+39*0.01 ≈ 208
5	200	208	200-208=-8	243	-8-23=-31	-8*0.3+243*0.8+-31*0.01 ≈ 192
6	200	192	200-192=8	251	8-(-8)=16	8*0.3+251*0.8+16*0.01 ≈ 203
7	200	203	200-203=-3	248	-3-8=-11	-3*0.3+248*0.8-11*0.01 ≈ 197
8	200	197	200-197=3	251	3-(-3)=6	3*0.3+251*0.8+6*0.01 ≈ 202
9	200	202	200-202=-2	249	-2-3=-5	-2*0.3+249*0.8+-5*0.01 ≈ 199
10	200	199	200-199=1	250	1-(-2)=3	1*0.3+250*0.8+3*0.01 ≈ 200
11	200	200	200-200=0	250	0-1=-1	0*0.3+250*0.8+-1*0.01 ≈ 200

Kd yields more damped response

check Plots

PI is sufficient but PID is better

Set Point = 200 / Kp = 0.3 / Kd = 0.03

Initial = 0 / Ki = 0.18

n	setPoint	current	error	delta error	output
0	200	0	200-0=200	200-200	200*0.3+200*0.18+200*0.03 = 226
1	200	226	200-226=-26	-26-200=-226	-26*0.3+174*0.18+(-226*0.03) = 125
2	200	125	200-125=75	75-(-26)=101	75*0.3+249*0.18+101*0.03 = 225
3	200	225	200-225=-25	-25-75=-100	-25*0.3+224*0.18-100*0.03 = 169
4	200	169	200-169=31	31-(-25)=56	31*0.3+255*0.18+56*0.03 = 215
5	200	215	200-215=-15	-15-31=-46	-15*0.3+240*0.18+46*0.03 = 188
6	200	186	200-186=14	14-(-15)=29	14*0.3+254*0.18+29*0.03 = 208
7	200	208	200-208=-8	-8-14=-22	-8*0.3+246*0.18+22*0.03 = 195
8	200	195	200-195=5	5-(-8)=13	5*0.3+251*0.18+13*0.03 = 200
9	200	206	200-206=-6	-6-5=-11	-6*0.3+245*0.18+11*0.03 = 194
10	200	194	200-194=6	6-(-6)=12	6*0.3+251*0.18+12*0.03 = 203
11	200	203	200-203=-3	-3-6=-9	-3*0.3+248*0.18+-9*0.03 = 197
12	200	197	200-197=3	3-(-3)=6	3*0.3+251*0.18+6*0.03 = 202
13	200	202	200-202=-2	-2-3=-5	-2*0.3+249*0.18+-5*0.03 = 198
14	200	198	200-198=2	2-(-2)=4	2*0.3+251*0.18+4*0.03 = 202
15	200	202	200-202=-2	-2-2=-4	-2*0.3+249*0.18-4*0.03 = 198

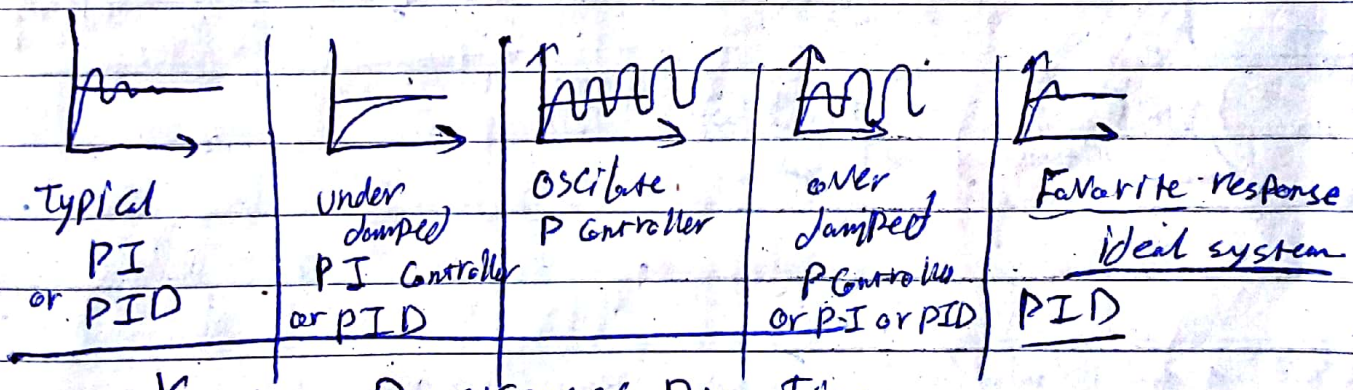
error = ± 2

More damped response is important in real life as it has aggressive response towards sudden changes

as it would be easier for the actuator to transit from last state to next state with lower moment of Inertia

at it is battary friendly " transition is easier"

revise figures "plots" to see how the output changes with different Kp, Ki, Kd values.



Kp : Decreases Rise Time
"make output near set point"

Ki : Eliminates steady-state-error

Kd : Reduce over shoots.
"more damped response"

P, PI, PID in a real life application.

to Control Motor speed.

Input: RPM Scale RPM → Voltage

output: Voltage

Kp = ? Sample time at least 10 smaller Than PID Round Times Const time

PID Const Time = 1 second.

Read motor speed = 0.1 second.

need to test on real Motor !

Comparasion

Numerical example: P Controller

Set Point , Current = Init. Value

→ error = set - current;

output = error * Kp

Current = output;

Real example: Motor Control.

Set Point ⇒ RPM , Current = Init Value ⇒ RPM

error = set - current ⇒ RPM

output = error * Kp ⇒ pwm

→ Clip output (0, 255)

Current = Calculate Motor speed ⇒ "RPM"

"Kp ⇒ pwm / RPM" ①

@ sample time ②

@ Control loop time "Time Constant" ③

Some test

72 RPM → 255 pwm

63 RPM → 200 pwm.

Sampling time not suitable

200 millisecond

@ 60 RPM Read 2 slots in 100 millisecond

Solution: ~~More precise~~ encoder!

200 slots → 1000 milli second

20 slot → 100 milli second

5 slot Readed

Revolution = $\frac{5}{20} = \frac{1}{4}$ R

$\frac{1}{4}$ R → 0.1 sec

R → 60 sec

$\frac{1}{4} * 60 * 10 = 150$ RPM

1 Revolution → 0.1 sec

1 Revolution → 1 minute

1 Revolution → 60 sec

100 slot Readed

Revolution = $\frac{100}{20} = 5$ R

5 Revolution → 0.1 sec

1 Revolution → 60 sec

$\frac{5 * 60}{0.1} = 5 * 60 * 10$

$= 3000$ RPM

Algorithm: Sample time = 500 milli = 0.5 second.
Control loop time = 5 second
set Point = 60
Kp = 7.

each 5 second
Read speed each 0.5 second
~~Avg error~~ =
Avg speed = [speed]/10
Avg error = ~~Avg speed~~ set Point - Avg speed
Control signal = Avg error * Kp
Control signal Constrain [120, 255]
min out that moves the motors
max out can arduino out

Sample time = 0.5 second Min = 120

Control loop time = 5 second Max = 255

set point = 60 Kp = 7

n	set Point RPM	Avg RPM	Avg_error	output
0	60	0.6	59.4	59.4 * 7 = 415.8 => 255
1	60	69	-9	-9 * 7 = -63 => 120
2		39	21	21 * 7 = 147 => 147
3		46	13.2	13.2 * 7 = 92.4 => 120
4		37.8	22.2	22.2 * 7 = 155 => 155
5		49.2	10.8	10.8 * 7 = 75.6 => 120
6		37.2	22.8	22.8 * 7 = 159 => 159
7		50.4	9.6	9.6 * 7 = 67.2 => 120
8		39.6	20.4	20.4 * 7 = 142.8 => 142
9		45	15	15 * 7 = 105 => 120
10		38	21.6	21.6 * 7 = 151.2 => 151
11		48	12	12 * 7 = 84 => 120
12		36.6	23.4	23.4 * 7 = 163 => 163
13		47	12	12 * 7 = 84 => 120