EEE4320-01

# Digital Control Engineering

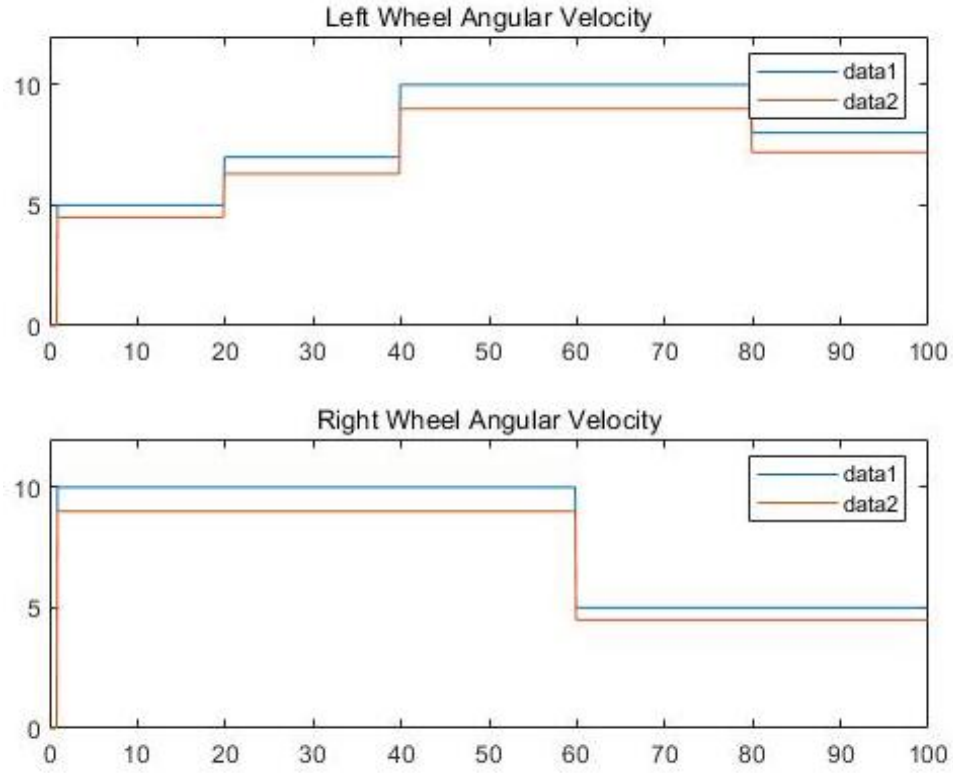
**HW#3**

2018142059

김서영

1.

- P control ($k_p$=9, $k_i$=0, $k_d$=0)

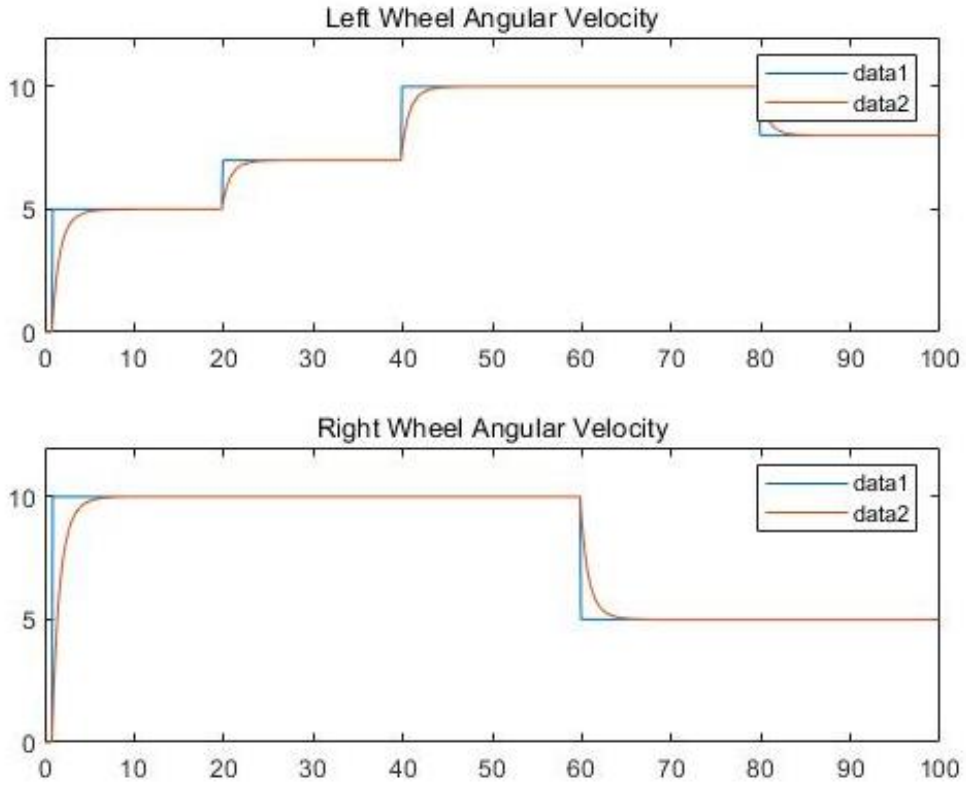



P control에서 조정할 수 있는 parameter는 $K_p$ 이다. 이를 증가시킬수록 더 빠르게 desired value에 가까워지지만, discrete time에서의 P control에서는 그 값이 너무 높으면 overshoot가 생긴다. 따라서 overshoot가 생기지 않는 선에서 그 값을 정하였다. P control에서 final value theorem을 이용하여 t가 무한으로 갈 때 C(t)의 값을 구하면 다음과 같이 된다.

$$\lim_{t\to\infty} C(t) = \lim_{s\to 0} sC(s) = \frac{K_p}{c + K_p}$$

따라서 $K_p$가 클수록 1에 가까워지지만, discrete time에서의 overshoot가 생기지 않게 하고, 과전류를 방지하기 위해서는 desired value인 C(t)=1에 도달할 수 없다. 결국 그래프와 같이 desired value에 도달하지 못하고 steady state error가 발생한다.

- PI control ($k_p$=1, $k_i$=1, $k_d$=0)




P control의 Steady State error를 없애기 위하여 integral control 부분을 추가한 것이 PI control이다. 이 때 C(s)는 다음과 같이 쓸 수 있다.

$$C(s) = \frac{K_p s + K_I}{[Is^2 + (c + K_p)s + K_I]R(s)} + \frac{s}{[Is^2 + (c + K_p)s + K_I]D(s)}$$

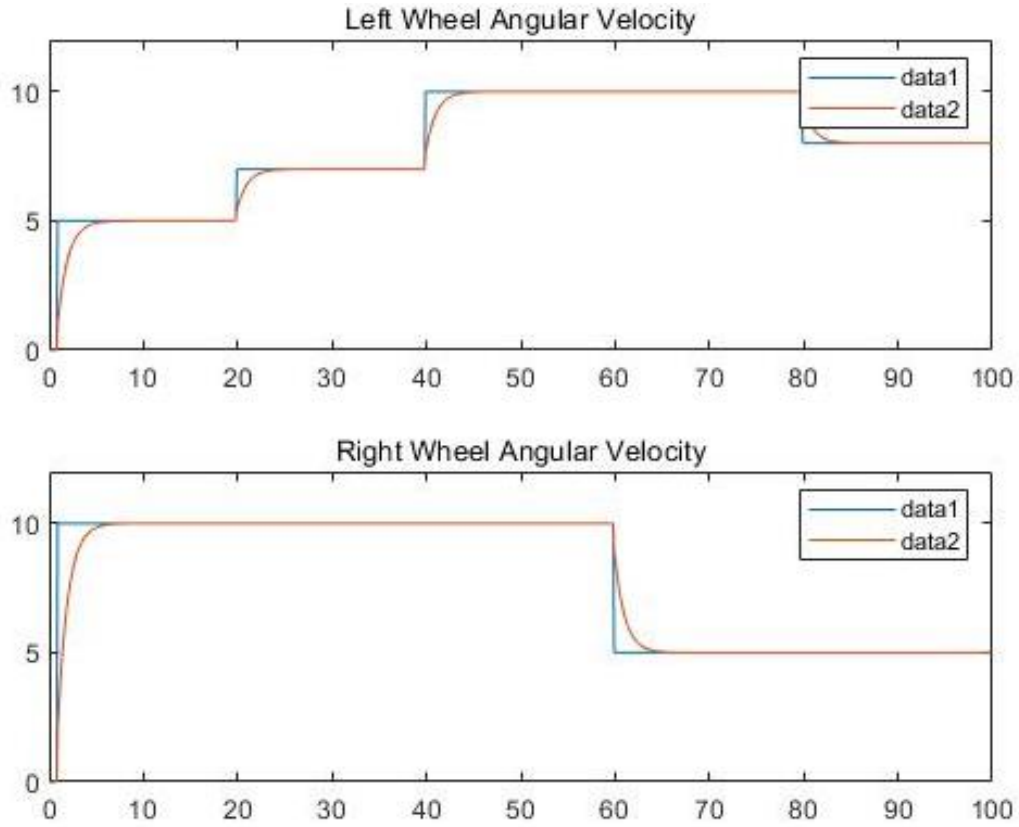
따라서 Overshoot를 없애기 위해서는 다음을 만족하여야 한다.

$$(c + K_p)^2 - 4K_I > 0$$

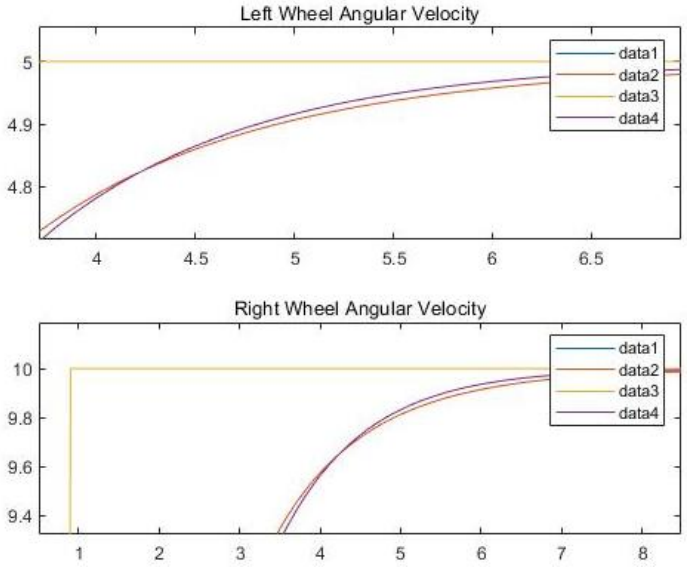
또한, 앞에서와 같은 방법으로 t가 무한으로 갈 때 C(t)의 값을 final value theorem을 통하여 구하면 1이 되므로 Steady State Error는 의도했던 것과 같이 0이 된다.

이를 모두 만족하도록 parameter의 값들을 각각 $K_p = 1, K_i = 1$로 설정하였을 때, 위와 같은 그래프가 나타난다. Steady state error는 확연히 눈에 띄지 않고, overshoot도 나타나지 않는다.

- PID control ($k_p$=1, $k_i$=1, $k_d$=0.1)

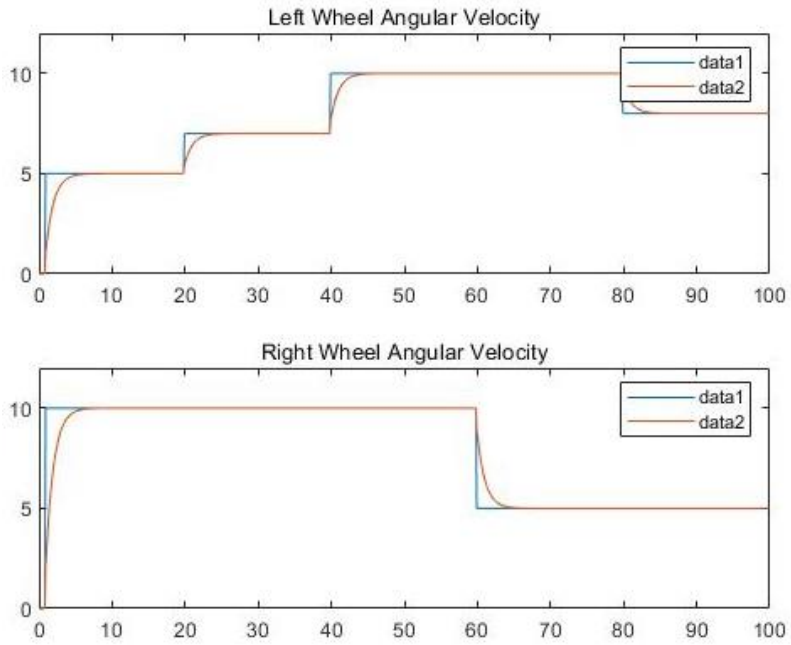



PI Control에서는 디자인하기에 따라 Steady State Error도 존재하지 않고, Overshoot도 없앨 수 있다. 그러나 여기서 transient response를 감소시키기 위해 derivative 파트를 추가한 것이 PID Control이다. 각각의 parameter를 $K_p = 1, K_i = 1, K_d = 0.1$로 지정하였을 때 각 Wheel의 Angular Velocity는 위의 그래프와 같다. 전체적으로 보기에는 앞선 PI control의 그래프와 눈에 띄는 차이가 보이지 않지만, 이를 확대하면 아래와 같이 transient response가 감소한 것을 확인할 수 있다.



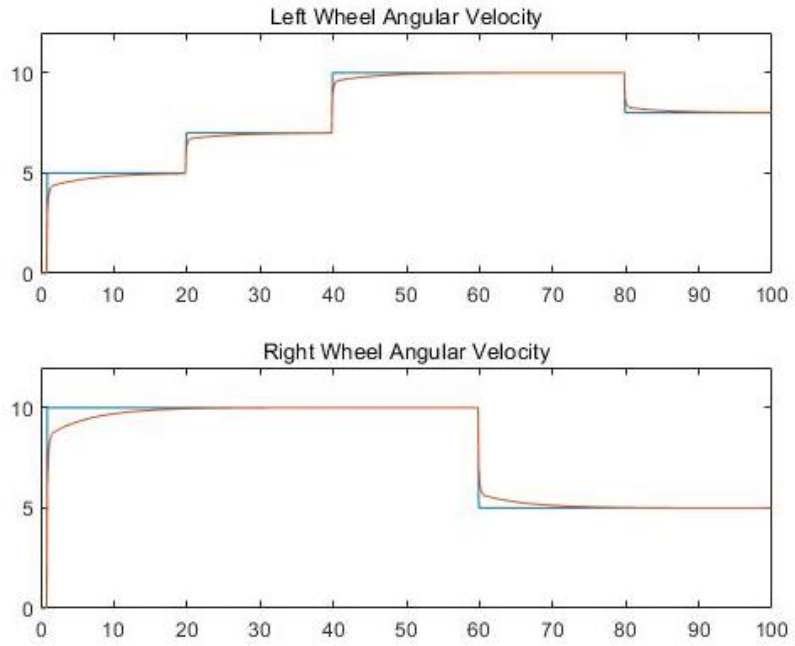
2. Transient angular speeds in time course

- PID critical damped case ($k_p$=1, $k_i$=1, $k_d$=0.1)



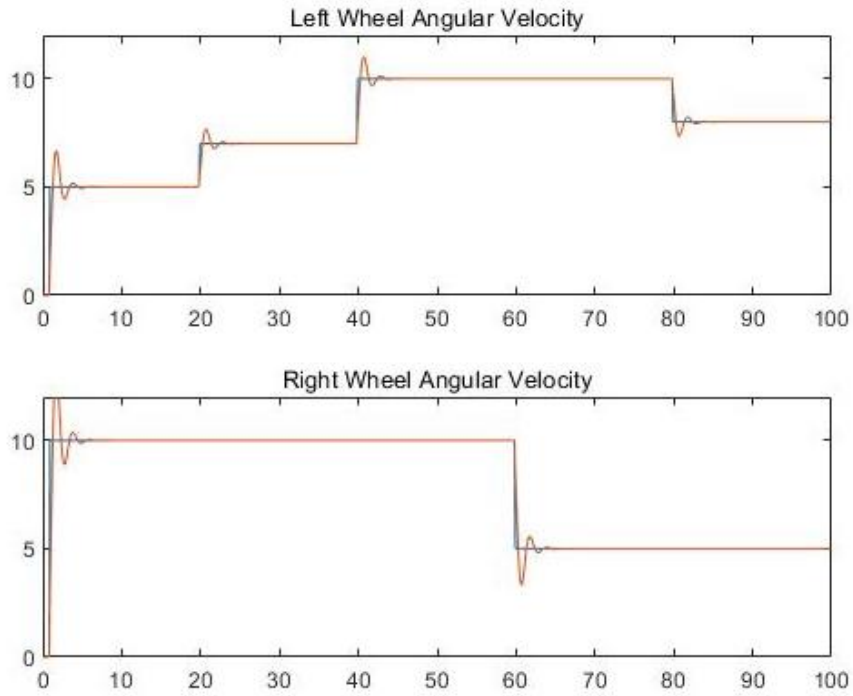

앞에서 parameter를 지정한 PID를 기반으로 parameter를 수정하여 Overdamped case와 underdamped case를 구성하였다.

- Overdamped case ($k_p$=5, $k_i$=1, $k_d$=0.1)




$K_p$를 적절히 변경하면 위와 같이 overdamped case를 나타난다. Overdamped case에서는 Angular velocity가 desired value에 도달하는데 매우 오랜 시간이 걸린다.

- Underdamped case ($k_p$=1, $k_i$=10, $k_d$=0.1)

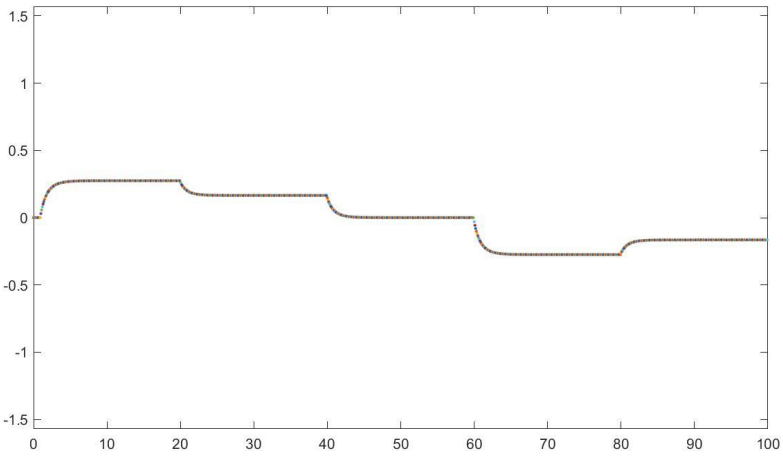



$K_i$를 증가시켜 위와 같은 underdamped case를 나타내었다. Underdamped case에서는 Angular velocity가 너무 빠르게 변화하여 desired value에 도달하기 위해 oscillation이 발생한다.

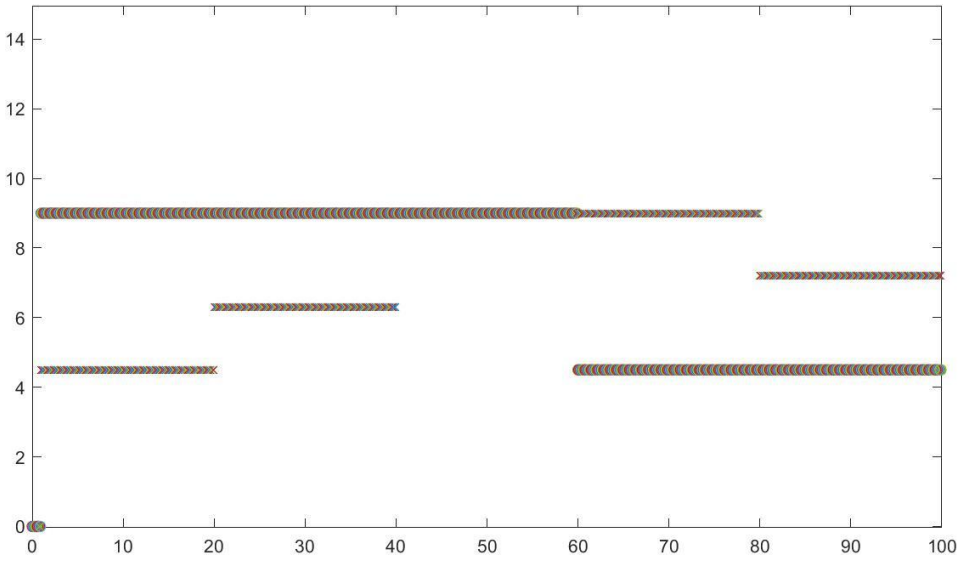
3. Robot movement simulation

- P control ($k_p$=9, $k_i$=0, $k_d$=0)

P control에서 로봇의 각속도와 로봇 바퀴의 각속도는 다음과 같이 나타난다.

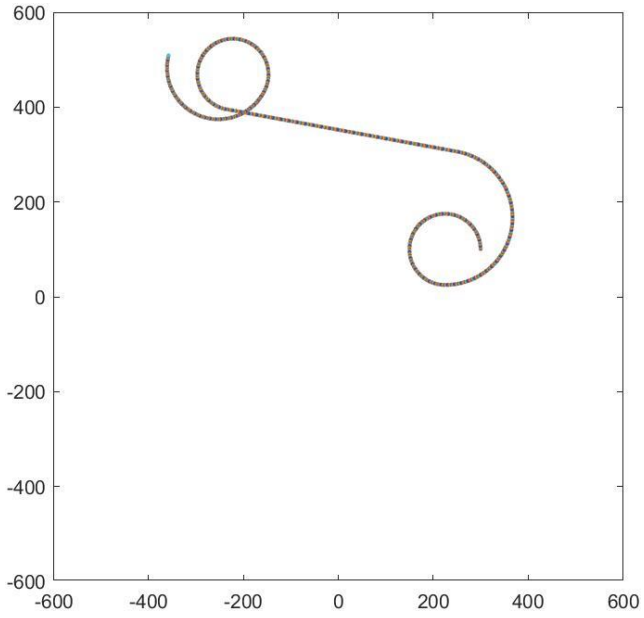


▲ Robot Angular Velocity (P control)




▲ Left Wheel/Right Wheel Angular Velocity (P control)

P control의 steady state error 때문에 wheel angular velocity가 각 desired velocity보다 작은 값을 갖고, 이에 따라 로봇의 각속도 또한 의도한 바와 다르게 변화하는 것을 확인할 수 있다.

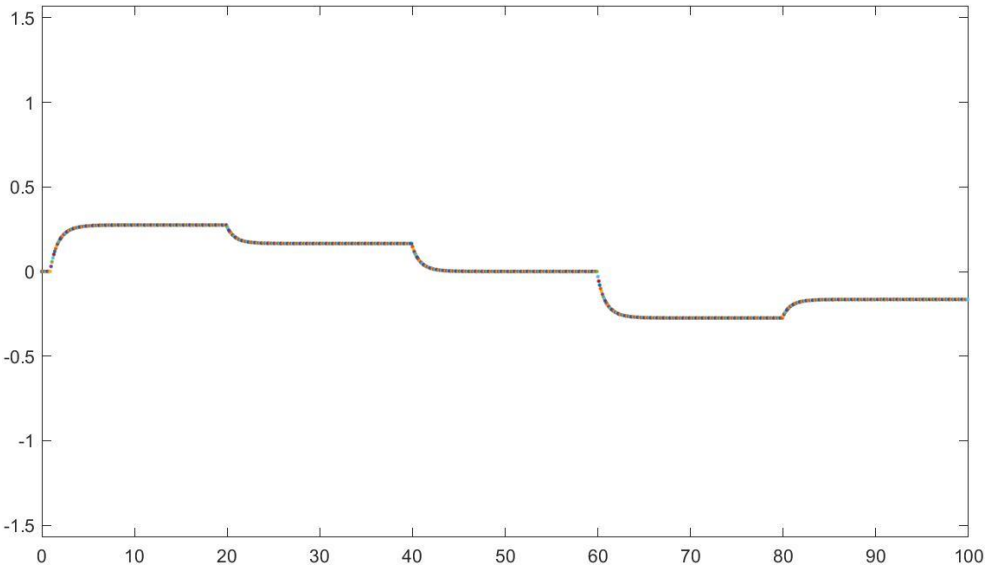



▲ Robot Trajectory (P control)

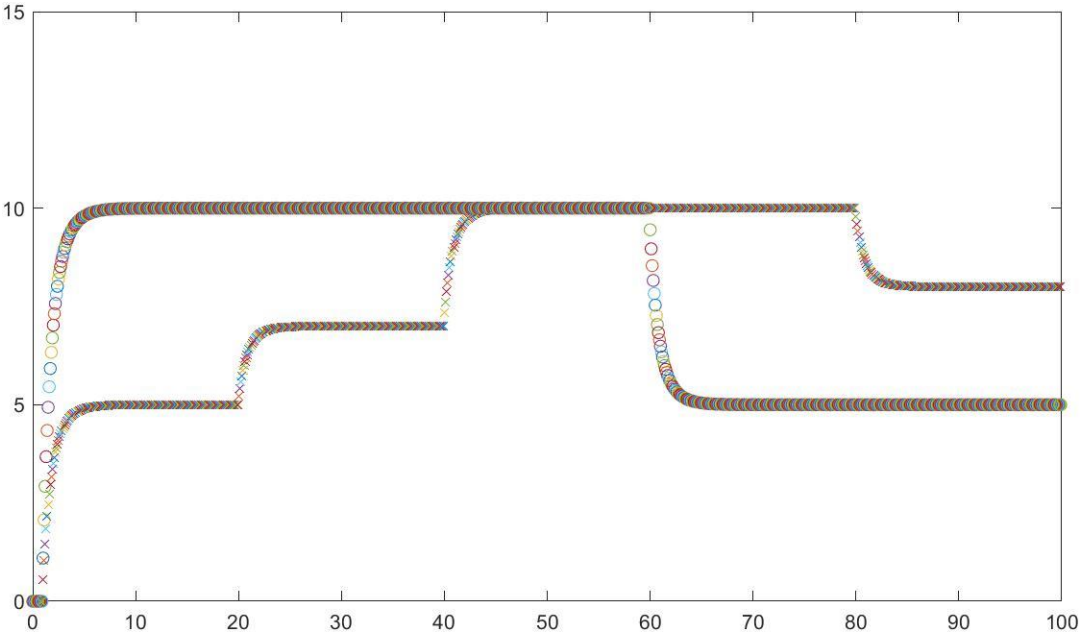
이러한 error는 로봇의 이동경로에도 반영되어 위와 같은 경로가 나타난다. 후술할 PI와 PID control을 사용했을 때의 이동경로와 확연히 다른 경로로 움직이는 것을 확인할 수 있다.

- PI control ($k_p$=1, $k_i$=1, $k_d$=0)

PI control을 사용하였을 때 다음과 같은 angular velocity 그래프들을 얻을 수 있었다.

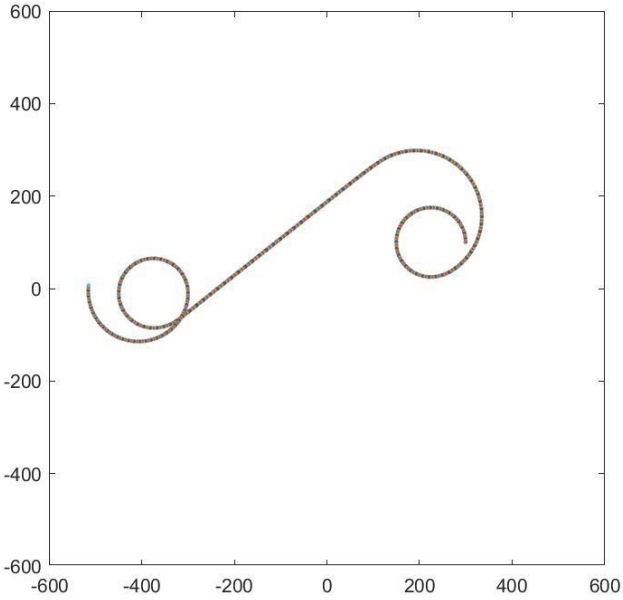



▲ Robot Angular Velocity (PI Control)




▲ Left Wheel/Right Wheel Angular Velocity (PI control)

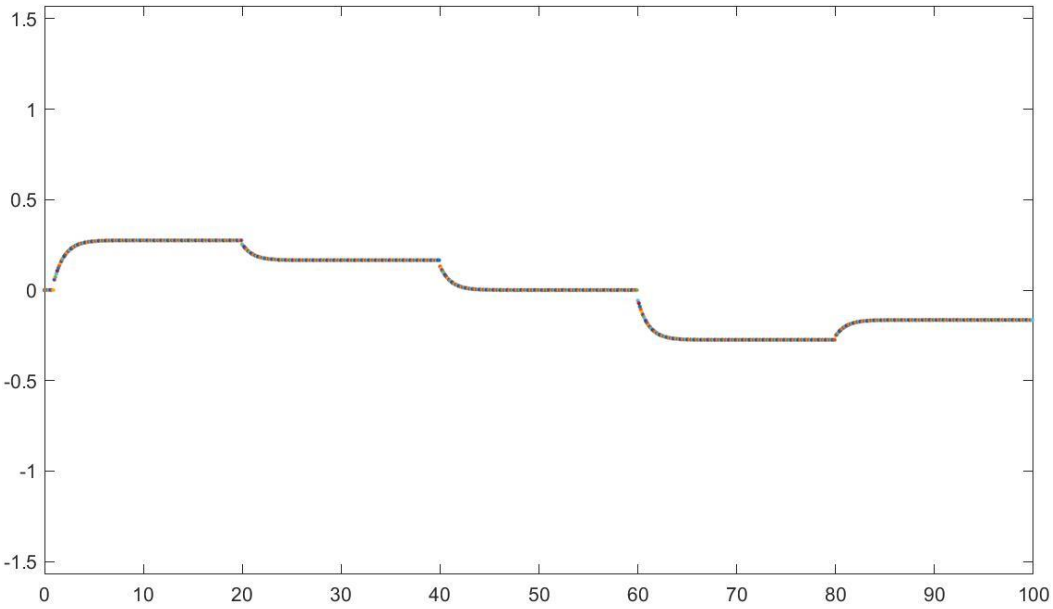
P control에 Integral component를 더하여 약간의 transient response 이후 steady state error 없이 desired value에 도달하는 것을 확인할 수 있다. 이는 로봇의 이동경로에도 영향을 미쳐 다음과 같이 이후 서술될 PID control의 경로와 P control에서보다 훨씬 비슷한 경로로 로봇이 이동하는 것을 확인할 수 있다.



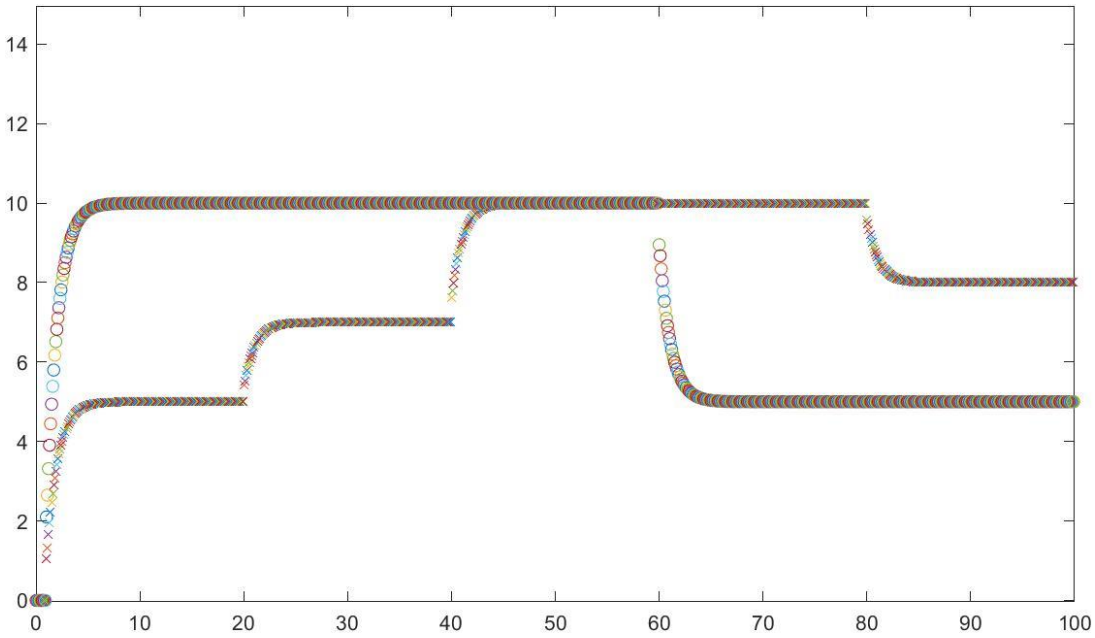

▲ Robot Trajectory (PI control)

- PID control ($k_p$=1, $k_i$=1, $k_d$=0.1)

Transient response를 줄이기 위해 PI control에서 derivative part를 추가한 PID control로 simulation을 진행하면 로봇과 각 바퀴의 angular velocity는 다음과 같이 나타난다.

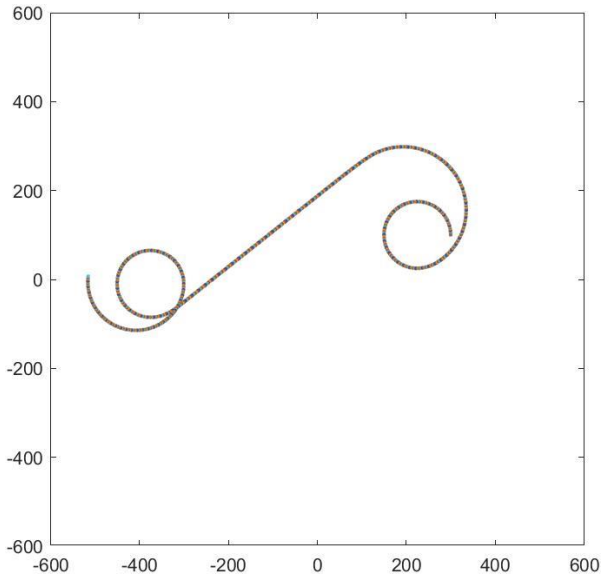



▲ Robot Angular Velocity (PID control)




▲ Left Wheel/Right Wheel Angular Velocity (PID control)

앞의 1번에서 확인한 것과 같이 PI control을 사용했을 때와 비교하였을 때 transient response가 감소하긴 하였으나 그 차이가 크지 않아 눈에 띄지는 않는다. Steady state도, overshoot도 발생하지 않았다. 이를 반영한 이동경로를 나타낸 것은 다음과 같다.

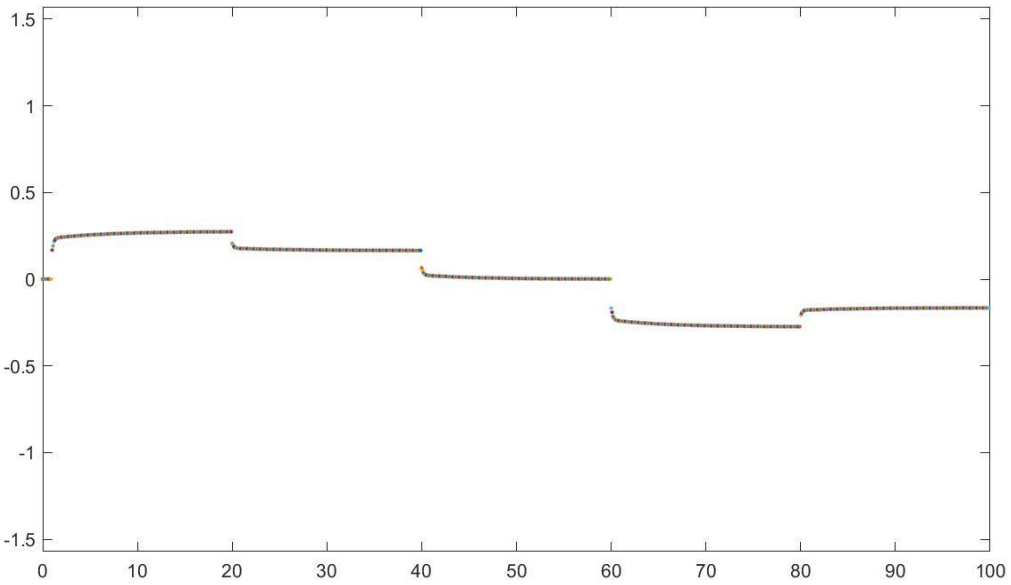



▲ Robot Trajectory (PID control)

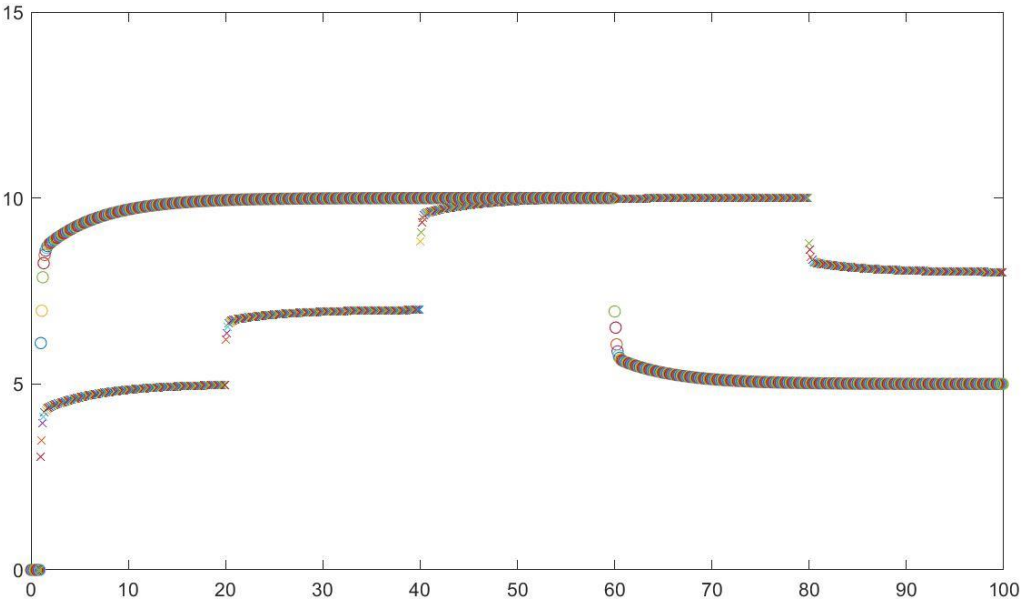
이동경로에서도 PI control을 사용하였을 때와 거의 비슷하게 움직이는 것을 알 수 있다. 그러나 이론상 각 바퀴의 angular velocity가 PI control보다 step function으로 이루어진 desired value에 가깝기 때문에 그 경로는 command로 결정된 경로에 더욱 가까울 것이다.

-Overdamped case ($k_p$=5, $k_i$=1, $k_d$=0.1)

$K_p$ 의 값을 바꿔 overdamped 상태로 simulation을 수행하면 다음과 같은 angular velocity 그래프를 얻을 수 있다.

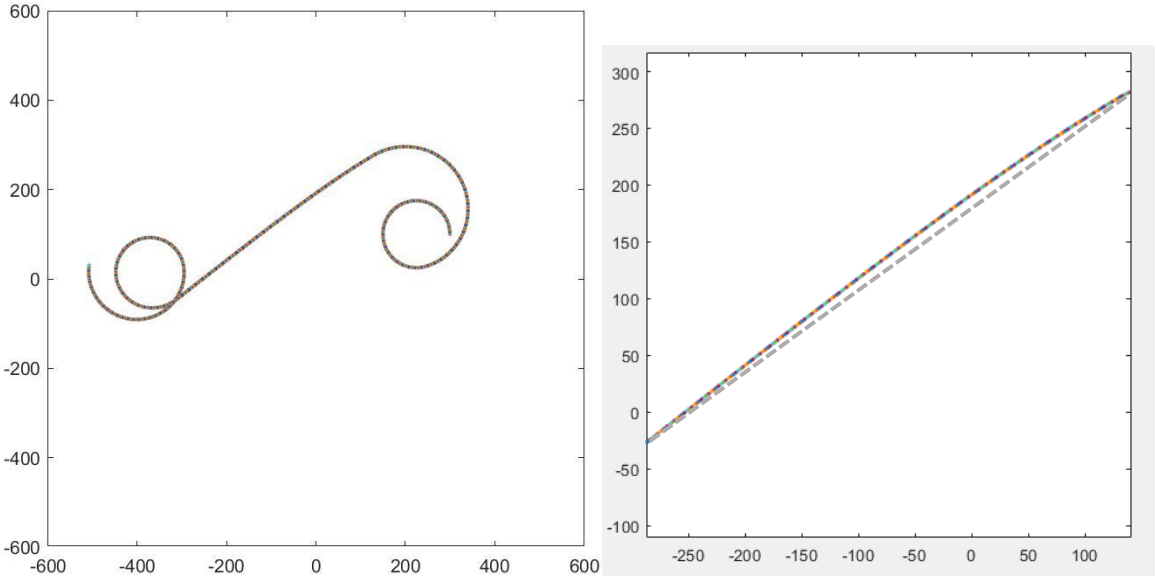



▲ Robot Angular Velocity (PID Overdamped)




▲ Left Wheel/Right Wheel Angular Velocity (PID Overdamped)

Overdamped case에서는 desired value에 가까워져도 steady state에 도달하는 데 오랜 시간이 걸린다. 그래프에서도 이것이 나타나서 desired value에 가까워진 상태에서도 계속 증가하는 것을 확인할 수 있다. 로봇의 각속도 또한 0이여야 하는 부분에서도 계속 값이 변화한다.

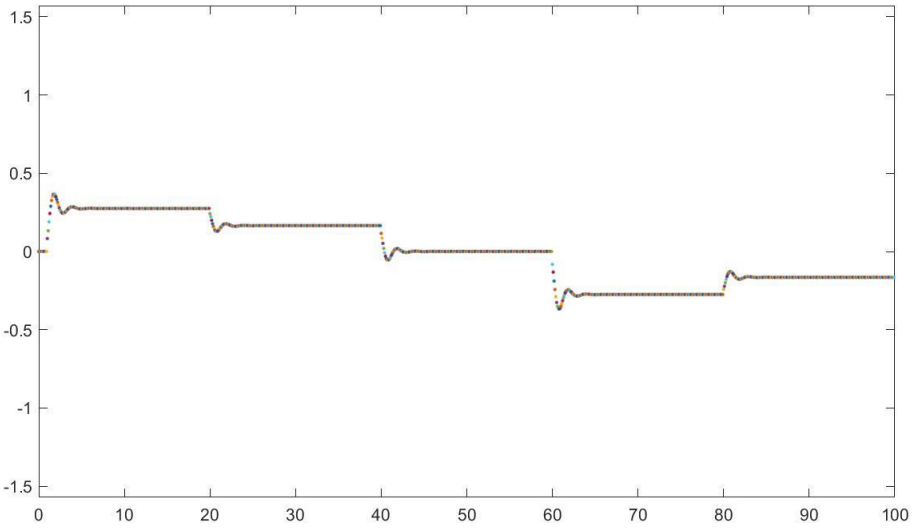



▲ Robot Trajectory (PID Overdamped)

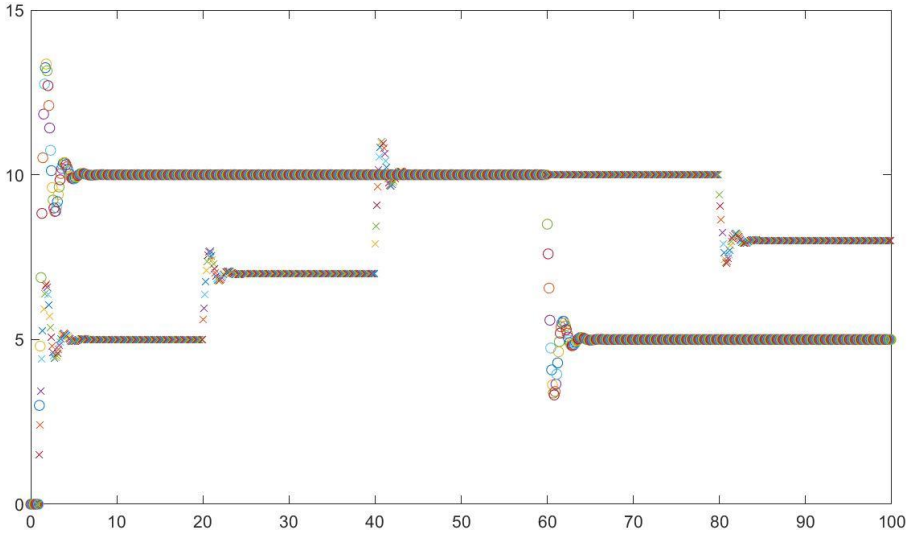
각속도가 계속 변화하는 것은 로봇의 이동경로에서도 볼 수 있는데 critical damped의 경우와 달리 양 바퀴의 desired angular velocity가 같아서 로봇이 직선으로 움직여야 하는 구간에서도 경로가 휘어있는 것을 확인할 수 있다.

-Underdamped case ($k_p$=1, $k_i$=10, $k_d$=0.1)

$K_i$의 값을 조정하여 underdamped 상태로 simulation을 진행하면 각속도 그래프는 다음과 같이 나온다.



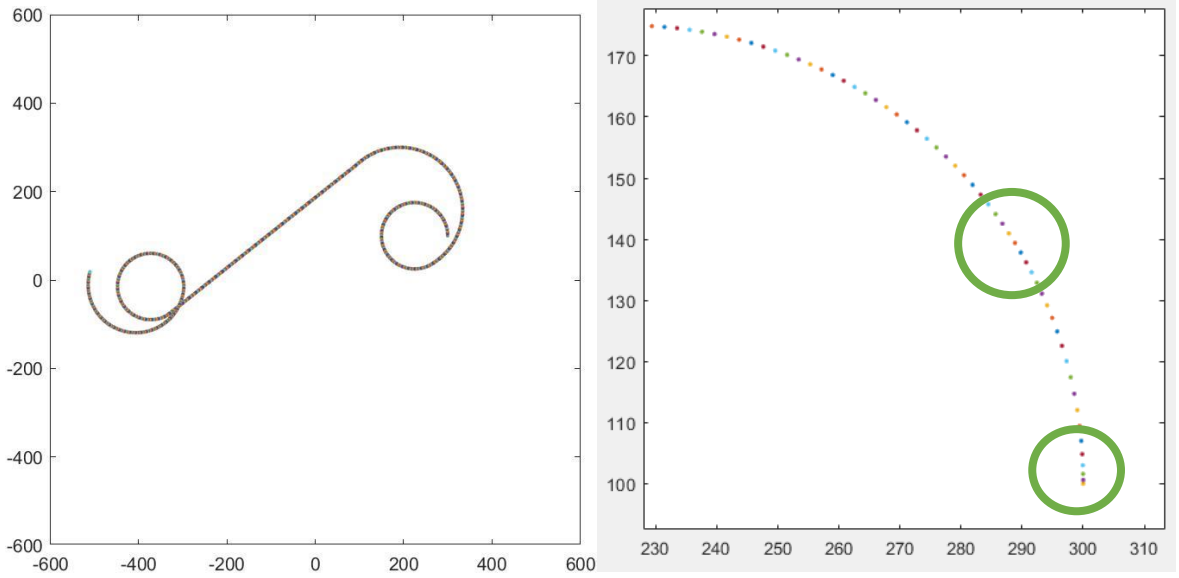

▲ Robot Angular Velocity (PID underdamped)




▲ Left Wheel/Right Wheel Angular Velocity (PID underdamped)

Underdamped case의 특징인 Oscillation이 바퀴의 각속도에서 발생하며 이는 로봇의 각속도에도 영향을 미친다. 로봇의 각속도가 Oscillate하는 것은 로봇이 회전할 때 부드러운 경로로 일정하게 움직이지 못하는 것을 뜻한다.

이는 로봇의 trajectory에서도 나타나는데, overdamped case와 다르게 oscillate 이후에는 desired value로 움직이기 때문에 전체적인 경로에서는 눈에 띄지 않지만, 경로를 확대하였을 때 일정하지 않은 것을 확인할 수 있다. (동그라미 친 부분)




▲ Robot Trajectory (PID underdamped)

<Source code>

y2018142059_1.m

```
%
% P/PI/PID controller simulation
%
close

dt=0.1; I=1; c=1; % wheel motor parameters
% kp=9; ki=0; kd=0;   %P parameters
% kp=1; ki=1; kd=0;   %PI parameters
kp=1; ki=1; kd=0.1; % PID parameters

wr(1)=0; wr(2)=0; wl(1)=0; wl(2)=0; % initial setting for wheel speed
tr(1)=0; tr(2)=0; tl(1)=0; tl(2)=0; % initial setting for torque

r=zeros(1000,1);
r(10:600)=10;
r(600:1000)=5;

l=zeros(1000,1);
l(10:200)=5;
l(200:400)=7;
l(400:800)=10;
l(800:1000)=8;

er(1)=0; er(2)=0;
el(1)=0; el(2)=0;

for n=3:1000, % we start with n=3 (why?, because of e(n-2))
er(n) = r(n)-wr(n-1);   el(n) = l(n)-wl(n-1);
%
tr(n)=tr(n-1)+kp*(er(n)-er(n-1))+ki*er(n)*dt+kd*(er(n)-2*er(n-1)+er(n-
2))/ dt;
tl(n)=tl(n-1)+kp*(el(n)-el(n-1))+ki*el(n)*dt+kd*(el(n)-2*el(n-1)+el(n-
2))/ dt;
%
wr(n)=wr(n-1) + dt/I*(-c*wr(n-1) + tr(n));
wl(n)=wl(n-1) + dt/I*(-c*wl(n-1) + tl(n));
% transient angular speed
WR=wr(n); WL=wl
end;

subplot(2,1,1);
plot((1:1000)*dt - dt, l); hold on;
plot((1:1000)*dt - dt, wl);
legend();
title('Left Wheel Angular Velocity');
axis([0 100 0 12])

subplot(2,1,2);
plot((1:1000)*dt - dt, r); hold on;
plot((1:1000)*dt - dt, wr);
legend();
title('Right Wheel Angular Velocity');
axis([0 100 0 12])
```

y2018142059_2.m

```
%
% PID controller simulation (critical, underdamped, overdamped)
%
close

dt=0.1; I=1; c=1; % wheel motor parameters
% kp=1; ki=1; kd=0.1;   % PID parameters
% kp=5; ki=1; kd=0.1;     % Overdamped
kp=1; ki=10; kd=0.1;  % Underdamped

wr(1)=0; wr(2)=0; wl(1)=0; wl(2)=0; % initial setting for wheel speed
tr(1)=0; tr(2)=0; tl(1)=0; tl(2)=0; % initial setting for torque

r=zeros(1000,1);
r(10:600)=10;
r(600:1000)=5;

l=zeros(1000,1);
l(10:200)=5;
l(200:400)=7;
l(400:800)=10;
l(800:1000)=8;

er(1)=0; er(2)=0;
el(1)=0; el(2)=0;

for n=3:1000, % we start with n=3 (why?, because of e(n-2))
er(n) = r(n)-wr(n-1);   el(n) = l(n)-wl(n-1);
%PID
tr(n)=tr(n-1)+kp*(er(n)-er(n-1))+ki*er(n)*dt+kd*(er(n)-2*er(n-1)+er(n-
2))/ dt;
tl(n)=tl(n-1)+kp*(el(n)-el(n-1))+ki*el(n)*dt+kd*(el(n)-2*el(n-1)+el(n-
2))/ dt;
wr(n)=wr(n-1) + dt/I*(-c*wr(n-1) + tr(n));
wl(n)=wl(n-1) + dt/I*(-c*wl(n-1) + tl(n));
WR=wr(n); WL=wl% transient angular speed of left motor
end;

subplot(2,1,1);
plot((1:1000)*dt - dt, l); hold on;
plot((1:1000)*dt - dt, wl);
title('Left Wheel Angular Velocity');
axis([0 100 0 12])

subplot(2,1,2);
plot((1:1000)*dt - dt, r); hold on;
plot((1:1000)*dt - dt, wr);
title('Right Wheel Angular Velocity');
axis([0 100 0 12])
```

y2018142059_3.m

```
%
% Robot Simulation
%

close all;

head =90*pi/180 ; x=300; y=100; % starting position and heading angle
diameter = 55; radius = diameter /2; % robot diameter
wdiameter=5.5; wradius = wdiameter /2;%wheel diameter
B = 50; % distance between two wheels
t = 0:0.1:2*pi+0.2; % to draw robot body
dt=0.1;
I=1; c=1; % wheel motor parameters
% kp=9; ki=0; kd=0;     % P parameteres
% kp=1; ki=1; kd=0;     % PI parameters
% kp=1; ki=1; kd=0.1;   % PID parameters
kp=5; ki=1; kd=0.1;   % PID parameters overdamped
% kp=1; ki=10; kd=0.1;  % PID parameters underdamped

eL=0; eL1=0; eL2=0;
eR=0; eR1=0; eR2=0;



scrsz = get(0,'ScreenSize');
figure('Position',[100 100 scrsz(3)*0.8 scrsz(4)*0.8])
time=0;
Numberofloop=1000;
wL=0;% anguler velocity of left wheel
wR=0;% anguler velocity of right wheel
tL=0;
tR=0;
w = 0;
savex=zeros(1,Numberofloop);
savey=zeros(1,Numberofloop);
saveR=zeros(1,Numberofloop);
savehead = zeros(1,Numberofloop);
savewL=zeros(1,Numberofloop);
savewR=zeros(1,Numberofloop);



for N=1:1:Numberofloop

 %Robot Drawing
 rx = x + radius * cos(t); ry = y + radius * sin(t);
 subplot(2,2,1),plot(rx, ry)% draw robot body
 axis([-600 600 -600 600])
%  title('Robot Location');
 daspect([1 1 1])
 line([x x+radius*cos(head)], [y y+radius*sin(head)])

 %Robot Trajectory
 subplot(2,2,2),plot(x, y,'.');
 axis([-600 600 -600 600]),hold on;
%  title('Robot Trajectory');
 daspect([1 1 1])
```

```
 %Robot Angular Velocity
 subplot(2,2,3),plot(time,w ,'.');
 axis([0 Numberofloop*dt -pi/2 pi/2]),hold on;
%  title('Robot Angular Velocity');
%  grid on;
%  grid minor;

 %Wheel Angular velocity
 subplot(2,2,4),plot(time, wL,'x'),hold on
 plot(time, wR,'o'),
%  grid on;
%  grid minor;
%  title('Wheel Angular Velocity');
%  legend('Left Wheel', 'Right Wheel');
 axis([0 Numberofloop*dt 0 15]),hold on;

 %%%%
 %you may determine desired angular velocity here.
 if (N==1)
 dwL = 0; dwR = 0;
 elseif (N==10)
 dwL = 5; dwR = 10;
 elseif (N==200)
 dwL = 7; dwR = 10;
 elseif (N==400)
 dwL = 10; dwR = 10;
 elseif (N==600)
 dwL = 10; dwR = 5;
 elseif (N==800)
 dwL = 8; dwR = 5;
 end
 %%%%
 %%%%

 %you may determine torque here.
 %left
 eL2=eL1;
 eL1=eL;
 eL=dwL-wL;
 %right
 eR2=eR1;
 eR1=eR;
 eR=dwR-wR;

 %PID
 tL=tL+kp*(eL-eL1)+ki*eL*dt+kd*(eL-2*eL1+eL2)/dt;
 tR=tR+kp*(eR-eR1)+ki*eR*dt+kd*(eR-2*eR1+eR2)/dt;
 %%%%
 %%%%
 %you may apply torque to robot system here
 wL = wL+dt/I*(-c*wL+tL);
 wR = wR+dt/I*(-c*wR+tR);
 %%%%
 VL=wL*wradius;%wheel speed
 VR=wR*wradius;%wheel speed

 if(VL==VR)
 x=x+VL*dt*cos(head);
 y=y+VL*dt*sin(head);
```

```
 else
 w=(VR-VL)/B;
 R=(B*(VR+VL))/((VR-VL)*2);
 x=R*sin(w*dt+head)+x-R*sin(head);
 y=-R*cos(w*dt+head)+y+R*cos(head);
 head=mod(head+w*dt,2*pi);
 end

 pause(0.01);
 savex(1,N)=x;
 savey(1,N)=y;
 savehead(1,N)=head;
 savewL(1,N)=wL;
 savewR(1,N)=wR;
 time=time+dt;
end
```