

گزارشکار Anti windup

در پوشه PID Anti-Windup دو پوشه وجود دارد به نام های ۱ و ۲

در پوشه ۱ پیاده سازی PID در script صورت گرفته است که برای نشان دادن نمودار های PID با امگا های مختلف هست.

در پوشه ۲ پیاده سازی Anti Windup به دو روش script و Simulink انجام شده است که در script با تغییر Ta ها و رسم نمودار های حالت های مختلف خروجی تاثیر افزایش و کاهش ثابت زمانی Ta که از بازه ۱ تا ۱۰ هست بر روی ناحیه اشباع نشان داده شده است.

در ادامه به توضیح هر کدام می پردازیم:

بخش اول:

پاک سازی و شروع دوباره و بستن پروژه ها و کار های قبلی در مطلب:

```
% Clearing and Closing Everything
clc; clear; close all;
%-----
```

ساخت تابع تبدیل $G(s) = 1 / s (s+1)$:

```
% Making Transfer Function for G(s)
s = tf('s');
G = 1 / (s * (s + 1));
%-----
```

ست کردن یا تیون کردن پارامتر های که PID (Kp، Kd و Ki) با استفاده از تابع pidtune برای تابع $G(s)$:

```
% Tuning G(s) for PID
% Other Options = P, PI, PD
C = pidtune(G, 'PID');
%-----
```

از این تابع pidtune می توان برای ست کردن پارامتر ها برای هر ۴ روش P، PI، PD و PID استفاده کرد و فقط کافیست mode آن را تغییر داد.

در بخش بعدی کد ضرایب PID (K_p ، K_d و K_i) در خروجی نمایش داده شده اند:

```
% Accessing the Individual Coefficients
Kp = get(C, 'Kp');
Kd = get(C, 'Kd');
Ki = get(C, 'Ki');
x = sprintf('Kp = %f',Kp);
disp(x);
y = sprintf('Kd = %f',Kd);
disp(y);
z = sprintf('Ki = %f',Ki);
disp(z);
%-----
```

که مقادیر خروجی به شرح زیر می باشد:

$K_p = 2.109126$

$K_d = 1.834131$

$K_i = 0.538966$

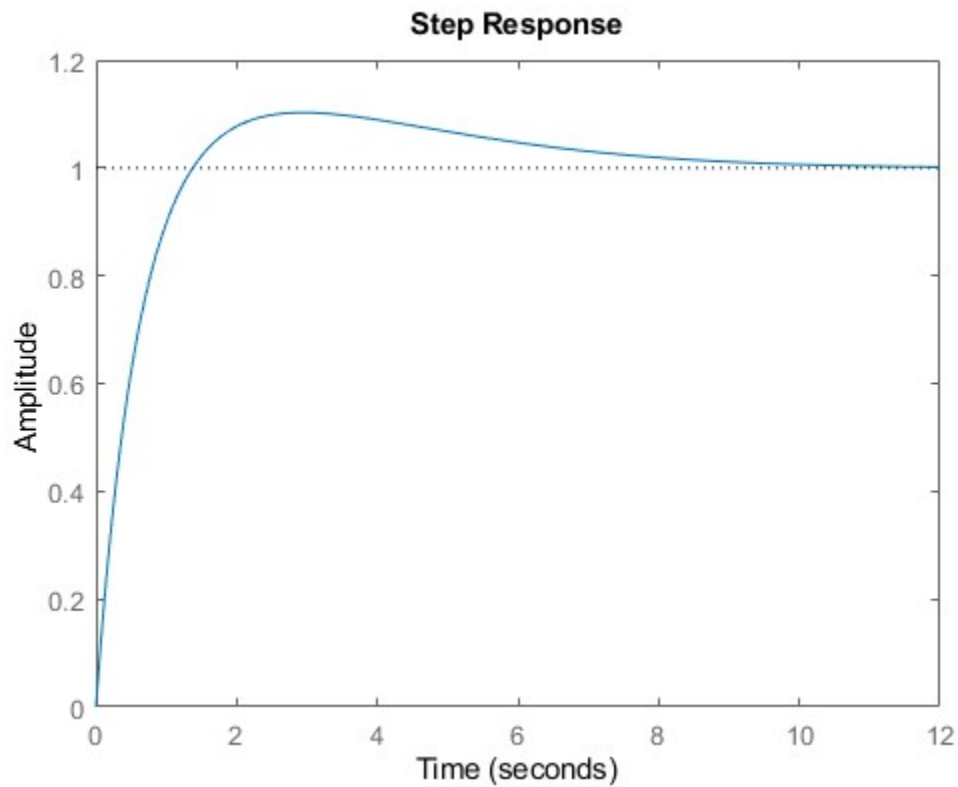
در ادامه سیستم PID را به صورت حلقه بسته با فیدبک ۱ می سازیم که در نهایت تابع تبدیل T می شود:

```
% Closed-Loop Transfer Function
T = feedback(C * G, 1);
%-----
```

در مرحله بعدی کد نمودار پاسخ پله تابع تبدیل T را با دستور زیر رسم می کنیم:

```
% Step Response Plot
figure(1);
step(T);
%-----
```

نمودار آن نیز به شکل زیر می شود:



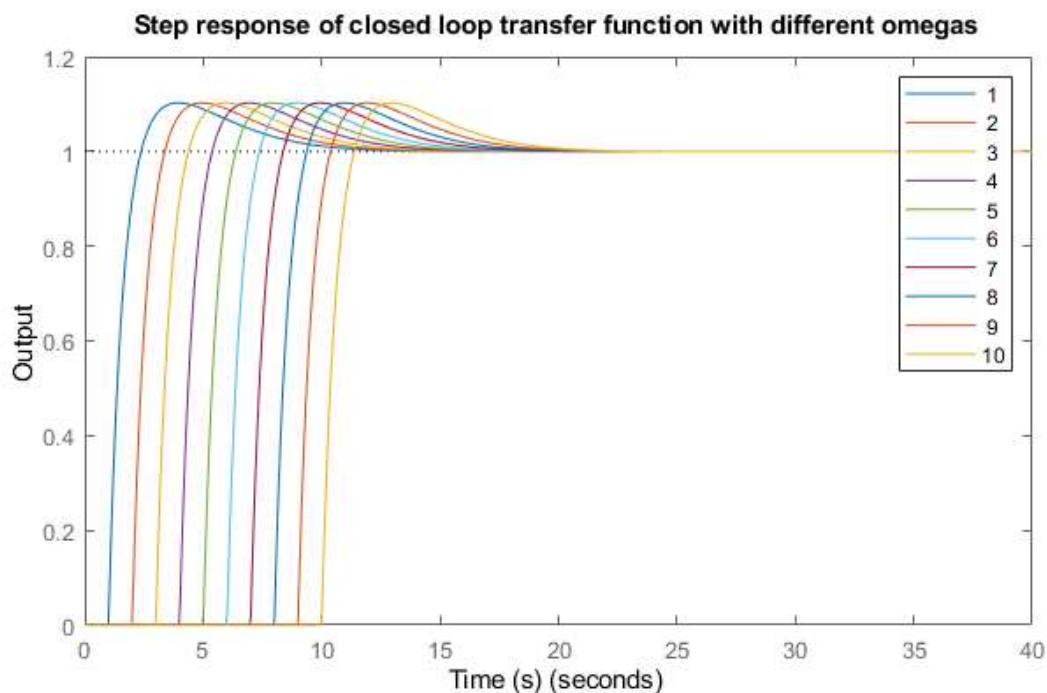
در بخش بعدی کد figure 1 که همان نمودار پاسخ پله تابع تبدیل T هست مختصاتی را اخذ می کند تا بتوان figure 2 را در کنار آن در قسمت های بعدی کد به صورت دو پنجره کنار هم گذاشت:

```
%-----positioning figure 1-----  
pos1 = get(gcf,'Position'); % get position of Figure(1)  
set(gcf,'Position', pos1 - [pos1(3)/2,0,0,0]) % Shift position of Figure(1)  
%-----
```

در بخش بعدی کد نمودار پاسخ پله تابع تبدیل T را به ازای $\omega = (1, 10, 1)$ یعنی از ۱ تا ۱۰ با گام های یک رسم می کنیم:

```
% Showing different Omegas
omega_min = 1; % minimum omega value
omega_max = 10; % maximum omega value
omega_step = 1; % omega increment
omega = omega_min:omega_step:omega_max; % omega vector
figure(2); % create a new figure
hold on; % hold the plot
for i = 1:length(omega)
    T_omega = T * exp(-omega(i) * s); % apply the time delay
    step(T_omega); % plot the step response
end
hold off; % release the plot
legend(num2str(omega)); % add a legend with omega values
xlabel('Time (s)'); % add x-axis label
ylabel('Output'); % add y-axis label
title('Step response of closed loop transfer function with different omegas'); % add title
%-----
```

نمودار حاصله رسم شده مطابق شکل زیر می باشد:



در نهایت هم *figure 1* و *figure 2* در کنار همدیگه قرار داده شده اند:

```
%---positioning figures 1, 2 side by side---
set(gcf, 'Position', get(gcf, 'Position') + [0,0,150,0]); % When Figure(2) is not the same size as Figure(1)
pos2 = get(gcf, 'Position'); % get position of Figure(2)
set(gcf, 'Position', pos2 + [pos1(3)/2,0,0,0]) % Shift position of Figure(2)
%-----
```

شما می توانید برای اطلاعات دقیق تر و نحوه عملکرد کد نوشته شده فایل *html* پابلیش شده توسط خود برنامه متلب را در همین پوشه ۱ ملاحظه بفرمایید.

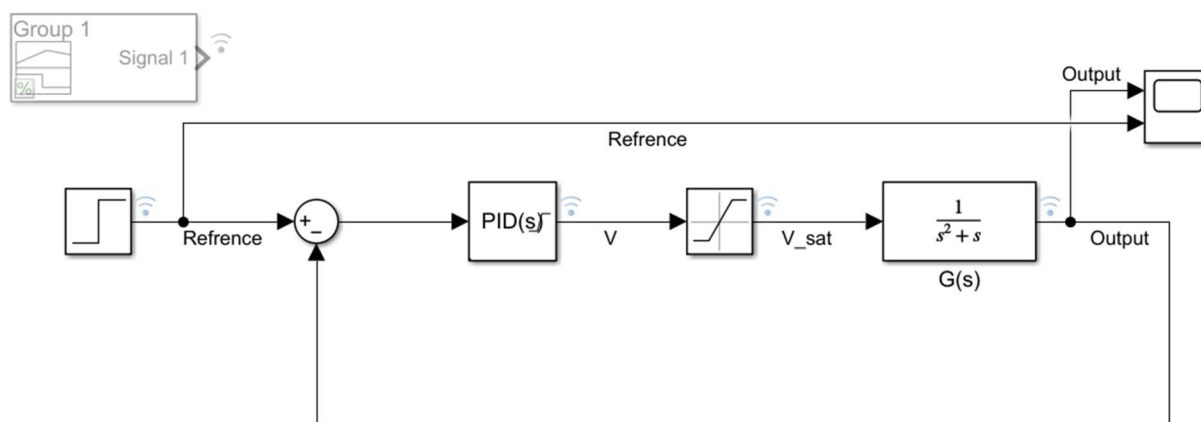
بخش دوم:

در این بخش ابتدا یک فایل سیمولینگ هست که شبیه سازی در آن برای $Kb = 1 / Ta$ ثابت انجام شده است و شما می توانید نتیجه را هم در اسکوپ و هم توسط *inspector* مشاهده نمایید.

ابتدا در مورد شماتیک و بلوک دیاگرام ساخته شده و نحوه پیاده سازی آنتی وینداپ در سیمولینک توضیح مختصری می دهیم و در ادامه کد *k2.m* مورد بررسی قرار می گیرد که در آن کنترلر *pid* با روش آنتی وینداپ پیاده سازی شده و با تغییر مقادیر Ta رفتار تابع نهایی در ناحیه اشباع بررسی می شود.

بخش Simulink:

بلوک دیاگرام آن به شکل زیر است:



در اینجا در ورودی تابع پله را داریم که بعد به یک جمع کننده می رود و خروجی جمع کننده به سمت *PID* رفته در نهایت خروجی *PID* را دچار *saturation* می کنیم و آن را به تابع تبدیل $G(s)$ که داشتیم تحویل می دهیم.

در بخش طراحی این بلوک دیاگرام در تنظیمات *PID Controller* داریم:

Block Parameters: PID Controller

PID 1dof (mask) (link)

This block implements continuous- and discrete-time PID control algorithms and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the 'Tune...' button (requires Simulink Control Design).

Controller: PID Form: Parallel

Time domain:

☒ Continuous-time

☐ Discrete-time

Discrete-time settings

Sample time (-1 for inherited): -1

Compensator formula

$$P + I \frac{1}{s} + D \frac{N}{1 + N \frac{1}{s}}$$

Main Initialization Output Saturation Data Types State Attributes

Controller parameters

Source: internal

Proportional (P): 2.08797641119552

Integral (I): 0.485356670842282

Derivative (D): 1.56681262957778

☒ Use filtered derivative

Filter coefficient (N): 6.49505028525306

OK Cancel Help Apply

که این مقادیر P ، I ، D همان مقادیر K_p ، K_d و K_i در تابع تبدیل PID هستند که با استفاده از خاصیت Automated tuning خود Simulink انجام شده است که با مقایسه کردن با مقادیر برنامه اول می توانید ببینید که پارامتر های PID به همدیگر نزدیک هستند:

Automated tuning

Select tuning method: Transfer Function Based (PID Tuner App) Tune...

☒ Enable zero-crossing detection

روش انتخاب شده برای Automated Tuning با استفاده از تابع تبدیل انجام می شود که در واقع $G(s)$ گذاشته شده در ادامه و فیدبک تاثیر خود را بر روی ست کردن پارامتر و مقدار های آن ها دارند.

تا به اینجا کار کنترلر PID و تابع تبدیل $G(s)$ که در برنامه اول داشتیم شبیه سازی شد. همچنین پارامتر های PID نیز مانند برنامه اول ست شد که مقادیری بسیار نزدیک به برنامه اول را دارند.

حال در ادامه می خواهیم پیاده سازی روش آنتی وینداپ را در Simulink توضیح دهیم.

برای این منظور در تنظیمات PID Controller به بخش Output Saturation می رویم:

Block Parameters: PID Controller

windup, external reset, and signal tracking. You can tune the PID gains automatically using the 'Tune...' button (requires Simulink Control Design).

Controller: PID Form: Parallel

Time domain:

☒ Continuous-time

☐ Discrete-time

Discrete-time settings

Sample time (-1 for inherited): -1

Compensator formula

$$P + I \frac{1}{s} + D \frac{N}{1 + N \frac{1}{s}}$$

Main Initialization Output Saturation Data Types State Attributes

Output saturation

☒ Limit output

Source: internal

Upper limit: 34

Lower limit: -34

☒ Ignore saturation when linearizing

Anti-windup

Anti-windup Method: back-calculation

Back-calculation coefficient (Kb): 1

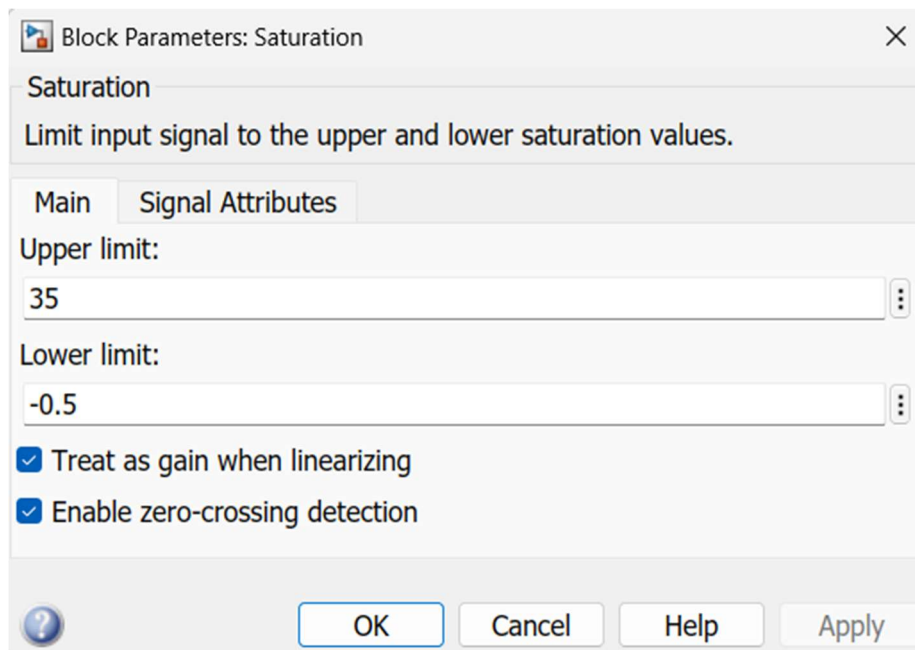
OK Cancel Help Apply

گزینه *limit output* را تیک زده و برای سادگی بر روی *internal* نگه می داریم و مقدار آن را با توجه به حدودا نصف اندازه نشان داده شده در خروجی ست می کنیم و بخش مهم ست کردن آنتی وینداپ هست که باید با روش *Back Calculation* انجام شود

این ضریب *Kb* که در اینجا دیده می شود همان $1/Ta$ هست و عملا داریم: $Kb = 1/Ta$

مقادیر مناسب برای Ta از ۱ تا هر مقدار مثبتی که بخواهیم هست که همین نتیجه می دهد که *Kb* باید مقداری بین ۰ و ۱ داده شود.

برای *Saturation* هم مقدار زیر را ست می کنیم:



Block Parameters: Saturation

Saturation

Limit input signal to the upper and lower saturation values.

Main Signal Attributes

Upper limit:

35

Lower limit:

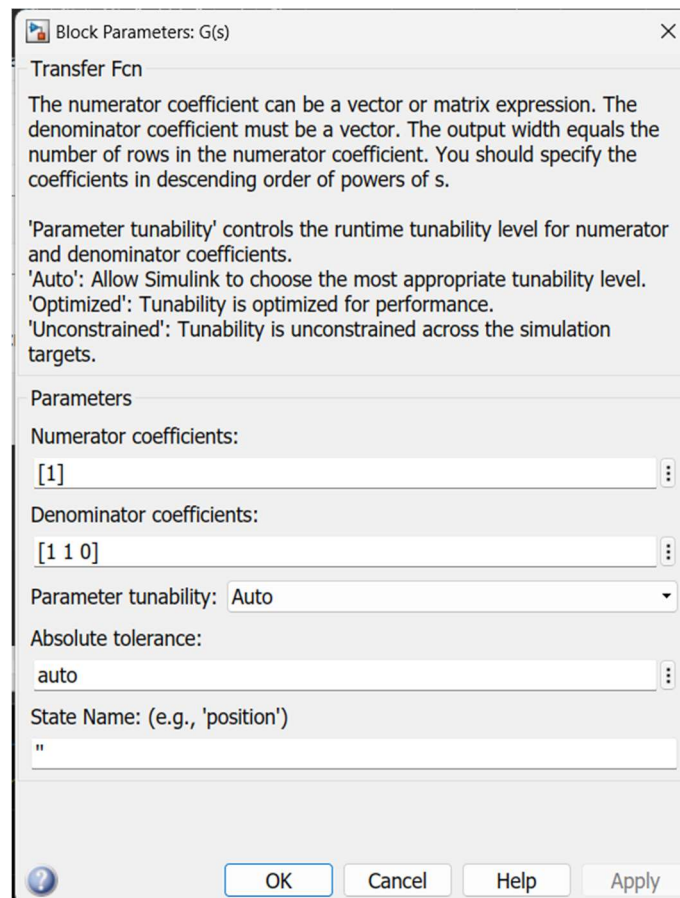
-0.5

☒ Treat as gain when linearizing

☒ Enable zero-crossing detection

OK Cancel Help Apply

برای ساخت تابع $G(s)$ هم باید مقادیر زیر را قرار دهیم:



Block Parameters: G(s)

Transfer Fcn

The numerator coefficient can be a vector or matrix expression. The denominator coefficient must be a vector. The output width equals the number of rows in the numerator coefficient. You should specify the coefficients in descending order of powers of s.

'Parameter tunability' controls the runtime tunability level for numerator and denominator coefficients.

'Auto': Allow Simulink to choose the most appropriate tunability level.

'Optimized': Tunability is optimized for performance.

'Unconstrained': Tunability is unconstrained across the simulation targets.

Parameters

Numerator coefficients:

[1]

Denominator coefficients:

[1 1 0]

Parameter tunability: Auto

Absolute tolerance:

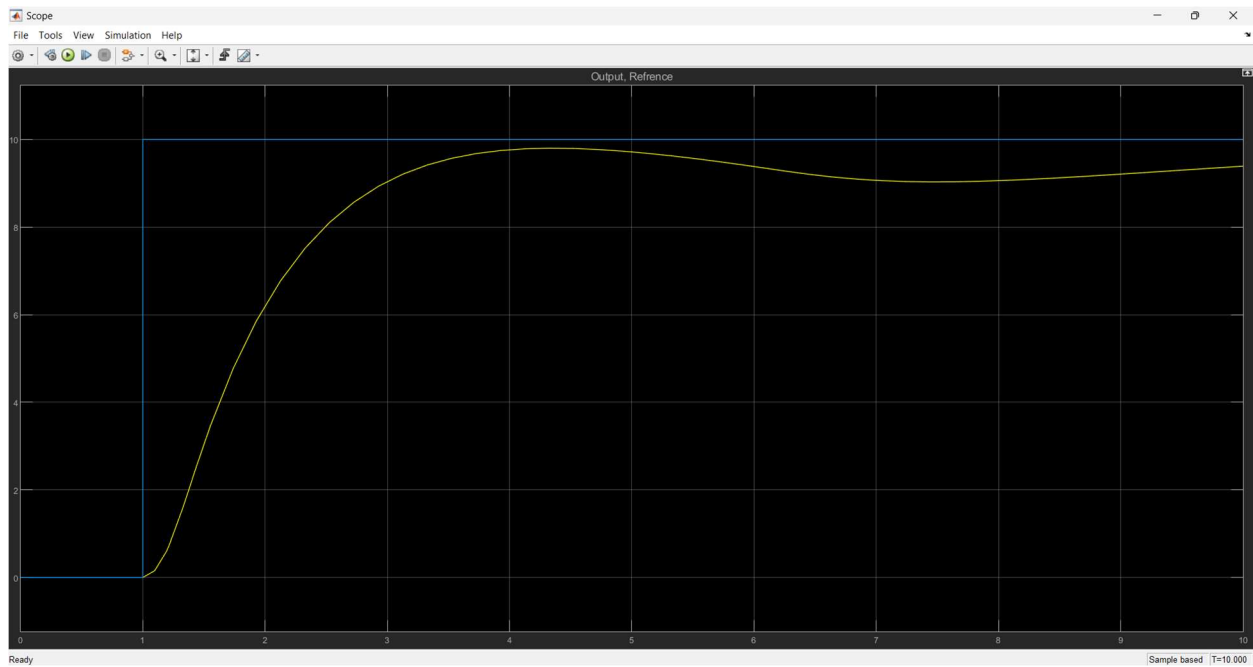
auto

State Name: (e.g., 'position')

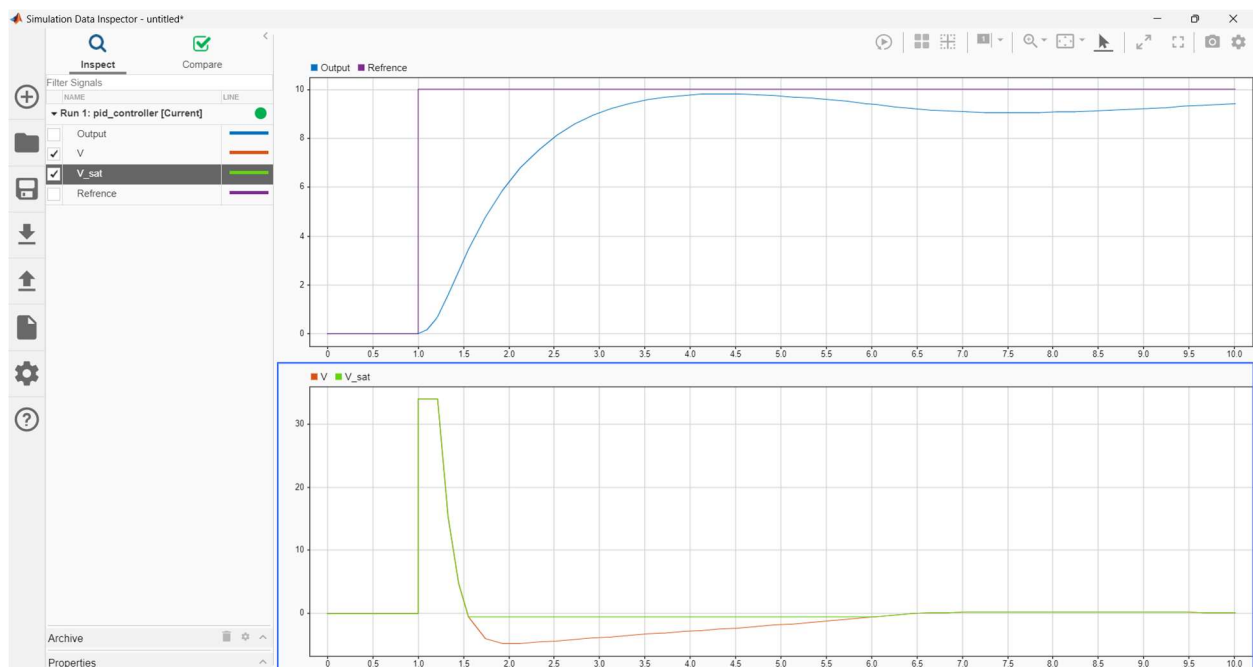
"

OK Cancel Help Apply

در نهایت با استفاده از اسکوپ خروجی و ورودی پله را بر روی اسکوپ می توان دید:



همچنین با توجه به نام گذاری و متغییر دادن به هر محل اتصال این نمودار و نمودار مقایسه V_{sat} و V می توان آن را در *Inspector* نمایش داد:



بخش *Simulink* نشان داده شده فقط برای پیاده سازی آنتی وینداپ به صورت بلوک دیگرام بوده و خروجی آن را می توان بررسی کرد اما برای نشان دادن و دیدن تغییرات Ta و تأثیرات آن بر روی ناحیه اشباع برنامه دوم در *script* ای به نام *k2.m* نوشته شده است.

در ادامه به توضیح برنامه دوم و توضیح نمودارهای آن می پردازیم.

بخش برنامه دوم:

ابتدا تمامی برنامه ها و خروجی های قبلی را می بندیم و پاک می کنیم:

```
% Clearing and Closing Everything
clc; clear; close all;
%-----
```

در بخش بعدی درست مثل برنامه اول که نوشته بودیم تابع $G(s)$ و C را تشکیل می دهیم ضرائب PID هم مانند قبل با تابع $pidtune$ ست می کنیم و پس از نمایش ضرایب K_p ، K_d و K_i ، تابع تبدیل T را با فیدبک ۱ و $C*G$ تشکیل می دهیم (درست مانند برنامه اول که توضیحات را دادیم):

```
% Making Transfer Function for G(s)
s = tf('s');
G = 1 / (s * (s + 1));
%-----

% Tuning G(s) for PID
% Other Options = P, PI, PD
C = pidtune(G, 'PID');
%-----

% Accessing the Individual Coefficients
Kp = get(C, 'Kp');
Kd = get(C, 'Kd');
Ki = get(C, 'Ki');
x = sprintf('Kp = %f', Kp);
disp(x);
y = sprintf('Kd = %f', Kd);
disp(y);
z = sprintf('Ki = %f', Ki);
disp(z);
%-----

% Closed-Loop Transfer Function
T = feedback(C * G, 1);
%-----
```

مقادیر K_p ، K_d و K_i به شکل زیر هستند:

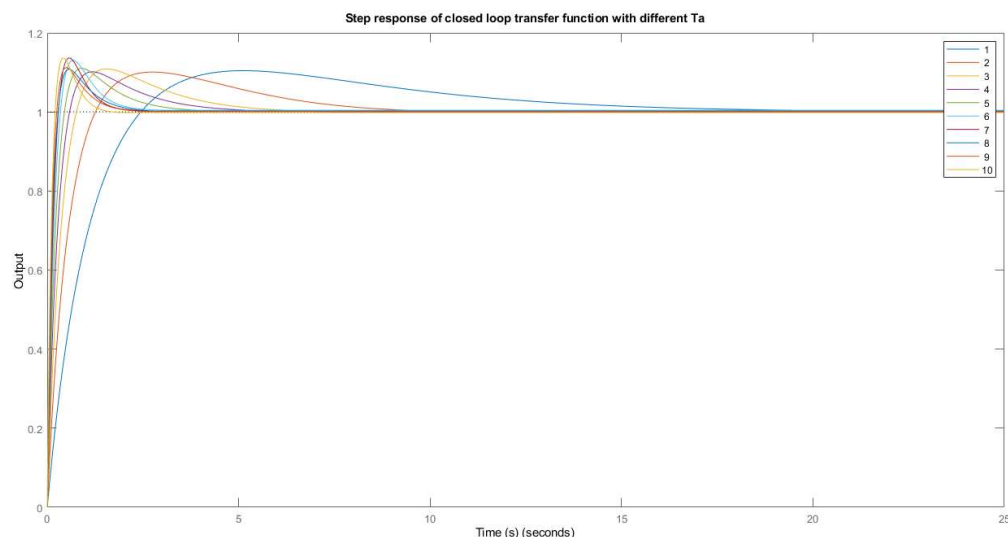
```
Kp = 2.109126
Kd = 1.834131
Ki = 0.538966
```

در ادامه در بخش آخر برنامه دوم مقادیر مینیمم (۱) و ماکسیمم (۱۰) را برای Ta تعیین می کنیم و با استفاده از گام Ta (۱) حلقه for را طی می کنیم.

در حلقه for پارامتر سوم $pidtune$ که همان Ta هست را برابر با مقدار i در هر دور لوپ قرار می دهیم و تابع C را دوباره با مقدار جدید Ta تیون می کنیم و در نهایت دوباره تابع تبدیل T را دوباره بازسازی می کنیم و نمودار پاسخ ضربه آن T خاص با Ta مورد نظر را رسم می کنیم:

```
% Showing different Ta
Ta_min = 1; % minimum Ta value
Ta_max = 10; % maximum Ta value
Ta_step = 1; % Ta increment
Ta = Ta_min:Ta_step:Ta_max; % Ta vector
figure('WindowState','maximized'); % create a new figure
hold on; % hold the plot
for i = 1:length(Ta)
    C = pidtune(G, 'PID',i); % C with different Ta
    T = feedback(C * G, 1); % The new Transfer Function
    step(T); % plot the step response
end
hold off; % release the plot
legend(num2str(Ta')); % add a legend with omega values
xlabel('Time (s)'); % add x-axis label
ylabel('Output'); % add y-axis label
title('Step response of closed loop transfer function with different Ta'); % add title
%-----
```

حال نمودار نهایی را مشاهده می کنیم که به شکل زیر رسم شده است:



در این نمودار می بینیم که هر چه مقدار Ta بزرگتر می شود به عبارتی Kb کوچکتر می شود نمودار رسم شده در ناحیه اشباع به مقدار تابع پله خیلی نزدیک می شود و عملاً اثر انتگرال گیری های زیادی کم می شود و تابع رفتاری بسیار نزدیک به پله خواهد داشت.

آن قوس اولیه ای که وجود دارد ناشی از بخش PD کنترل PID هست و عملاً می توان گفت که با افزایش T_a رفتار PID را به PD نزدیک می کنیم با این تفاوت که ترم ثابتی که بر اثر خطا در PD وجود دارد عملاً در PID جبران می شود و رفتار PID با وینداپ در کاربرد های مختلف برای حذف نیروهای ثابت مثل گرانش و disturbance های مثل باد، زلزله و شوک و بسیار کاربردی هست و عملاً با وجود سیستم آنتی وینداپ می توان تا حد خوبی از اشباع شدن انتگرال گیر در سیستم PID جلوگیری کرد.