Amirkabir University of Technology
(Tehran Polytechnic)

کنترل سرعت موتور DC

سینا ربیعی، علیرضا فقیه علی‌آبادی، معین نصیری

دکتر افشار

پاییز ۱۴۰۱

فهرست مطالب

۱. مشخصات سیستم

برای کنترل موتور DC نیاز به خود موتور، انکودر (افزایشی یا مطلق، اثر هال یا نوری) برای فیدبک موقعیت یا سرعت از موتور، درایور موتور DC (برای ارسال داده و راه‌اندازی موتور به کمک میکروکنترلر) و میکروکنترلر برای پیاده‌سازی الگوریتم PID و کنترل داریم.

۱. موتور DC و انکودر

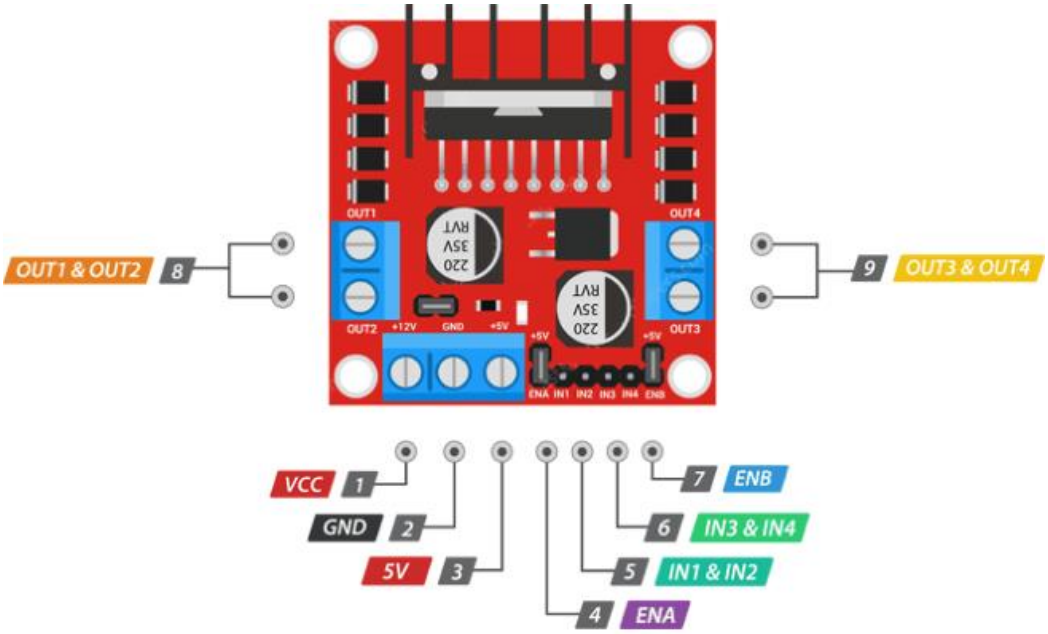
موتور DC انتخاب شده دارای ولتاژ نامی ۶V (به این منظور که در صورت استفاده از ولتاژهای کمتر از ۶ ولت، دور خروجی، توان و در نهایت گشتاور تولیدی موتور کاهش می‌یابد.) و حداکثر دور ۱۵۰rpm است. در صورت استفاده از ولتاژهای بیشتر، ممکن است بتوان گشتاور و سرعت را به اندازه کمی افزایش داد. توجه داریم که این کار توصیه نمی‌شود و باعث کاهش عمر موتور می‌شود. همچنین امکان داغ کردن موتور و خطرات به همراه آن در سیستم‌های حساس وجود دارد.

در دیتاشیت این موتور، وزن ۳۰ گرم، جریان بدون بار ۰/۰۴ آمپر و طول شفت ۴ میلی‌متر بیان شده. همچنین انکودر آن دارای رزولوشن ۴ پالس بر دور می‌باشد و ثابت زمانی ۰/۰۶ ذکر شده و مدل را بدون تاخیر در نظرگرفته است.

برای بدست آوردن موقعیت موتور از یک انکودر دو کانله اثرهال با دقت ۴ppr (چون دو کانال داریم در کل ۸ppr)استفاده می‌شود. این دو کانال سیگنال‌های متمم یکدیگر تولید می‌کنند، یعنی در زمانی که کانال ۱، High باشد، کانال ۲، Low است و برعکس. برای بدست آوردن سرعت می‌توان از داده‌های این سنسور مشتق گرفت یا در صورت دیجیتال بودن کنترلر، موقعیت دریافت شده در دو بازه زمانی متوالی را بر طول بازه تقسیم کنیم.

الگوریتم PID نوشته شده در میکرو یک Duty Cycle خروجی می‌دهد که مربوط به سیگنال PWM ورودی موتور است. دقت داریم که نباید موتور را به طور مستقیم از برد درایو کرد (در صورت افزایش ولتاژ یا جریان‌کشی بیش از حد و ناگهانی امکان آسیب رسیدن به برد وجود دارد) و نیاز به درایور DC داریم. به همین منظور از برد L298N استفاده می‌کنیم. این برد یک درایور DC دو کاناله و ۱۲V است.

۲. درایور (L298N)

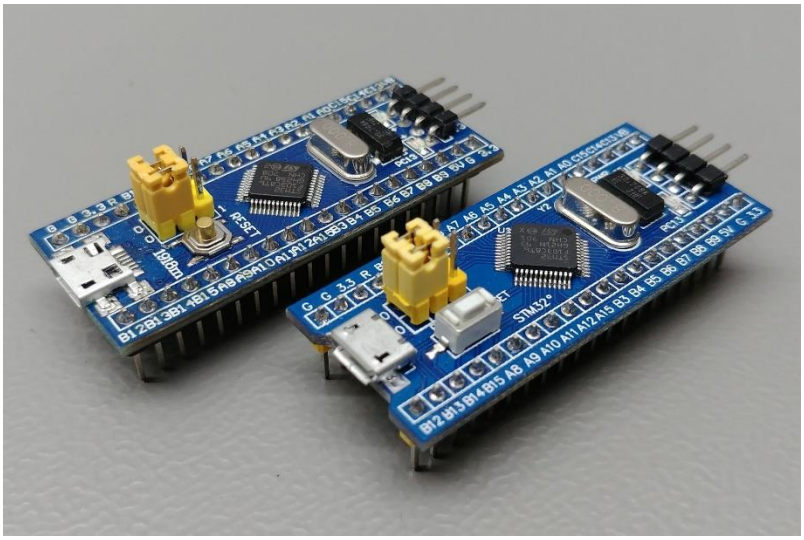


شکل ۱: داریور L298N

همانطور که در تصویر مشاهده می‌شود، پایه‌های ENA، IN1 و IN2 به ترتیب مربوط به کنترل سرعت OUT1 و OUT2 به کمک PWM و کنترل جهت چرخش موتور است. این داریور در چیپ داخلی خود دارای ماژول H-bridge است و به همین ترتیب امکان کنترل جهت چرخش موتور را دارد. این درایور دو ورودی ۱۲V و ۵V دارد که به ترتیب برای راه‌اندازی موتور ها و تغذیه مدار logic موجود در برد استفاده می‌شود. نیازی به برقراری اتصال ۵V نیست و می‌توان از رگولاتور موجود روی برد استفاده نمود که ورودی ۱۲V را به ۵V مورد نیاز برای مدارات داخلی تبدیل می‌کند. برای کنترل سرعت موتور باید جامپر موجود روی پین‌های ENA را حذف کنیم.

۳. میکرو کنترلر (STM32F103 BluePill)

در نهایت برای پیاده‌سازی PID از میکروکنترلر STM32F103C8T6 استفاده می‌کنیم. برد استفاده شده با نام تجاری bluepill موجود می‌باشد. این میکرو دارای ۴ تایمر (۲ تایمر جنرال و ۲ تایمر پیشرفته) و رجیسترهای ۳۲ بیتی می‌باشد. کلاک این میکرو از یک کریستال ۸ هرتز موجود روی برد تامین می‌شود و همه‌ی تایمرهای آن با فرکانس ۷۲MHz کار می‌کنند که برای کنترل سرعت موتور کافی می‌باشد (در تنظیمات برد می‌توان به کمک رجیسترهای PSC و ARR این مقدار را کاهش داد).



شکل ۲: برد bluepill

این برد دارای پروگرامر و تغذیه on-board نیست. برای تغذیه از ارتباط micro-USB استفاده می‌کنیم که تغذیه ۵V آن را تامین می‌کند. همچنین برای برنامه‌ریزی برد از پروگرامر خارجی ST-LINK استفاده می‌کنیم. برای برقراری ارتباط این پروگرامر با برد باید تغذیه و زمین آن را از برد با ۳/۳V و GND تامین کنیم. همچنین پایه SWDIO برد و پروگرامر را به هم متصل کرده و ارتبط SWCLK پروگرامر را با پایه متناظر روی برد برقرار می‌کنیم. سپس با استفاده از نرم‌افزار Cube IDE میکرو مورد نظر را انتخاب کرده و برنامه مورد نظر را روی آن لود می‌کنیم. توجه داریم که جامپرهای بوت برد در BOOT0 و BOOT1 باید در موقعیت صفر (نشان داده شده در تصویر) قرار بگیرند تا برد آماده برنامه‌ریزی باشد.

شکل ۳: پروگرامر ST-LINK

۲. تست حلقه‌باز و پارامترهای سیستم

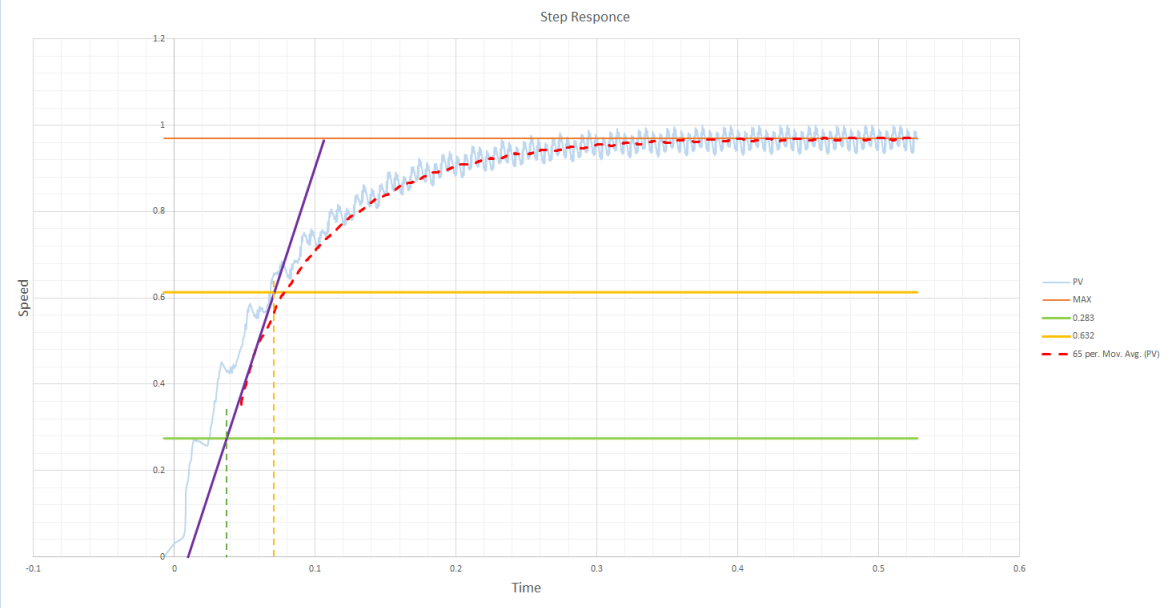
برای محاسبه ثابت زمانی سیستم و تاخیر و دیگر پارامترهای سیستم از تست حلقه باز موتور استفاده می‌کنیم. توجه داریم که تست نوسان کامل استفاده نشده زیرا امکان آسیب رسیدن به موتور با این کار وجود دارد. برای این منظور یک ورودی پله را به موتور داده و خروجی انکودر را رسم می‌کنیم. توجه داریم که به دلیل نویز موجود در سیستم و زمان نمونه برداری کوچک (۱۰ ms)، امکان بررسی داده‌های خروجی به شکل بدست آمده وجود ندارد. ابتدا برای نمونه برداری و بررسی پارامترهای سیستم هر ۳ پریود نمونه برداری داده‌ها را ثبت می‌کنیم (۳۰ ms) و به داده‌های بدست آمده یک خط فیت می‌کنیم. در ادامه داده نمودارهای بدست آمده از تحلیل داده‌ها در MATLAB، Cube monitor و Excel آورده شده. داده‌های استفاده شده در از لاجیک آنالایزر (Logic Analayzer) ۱۶۸MHz شرکت saleae بدست آمده. چون نرخ نمونه‌برداری این آنالایزر بسیار بالا است، نیاز به فیلتر قوی‌تری بود. به همین دلیل از فیلتر Moving Average استفاده کرده و منحنی تقریبی را بدست آوردیم.

۱. پارامترهای سیستم

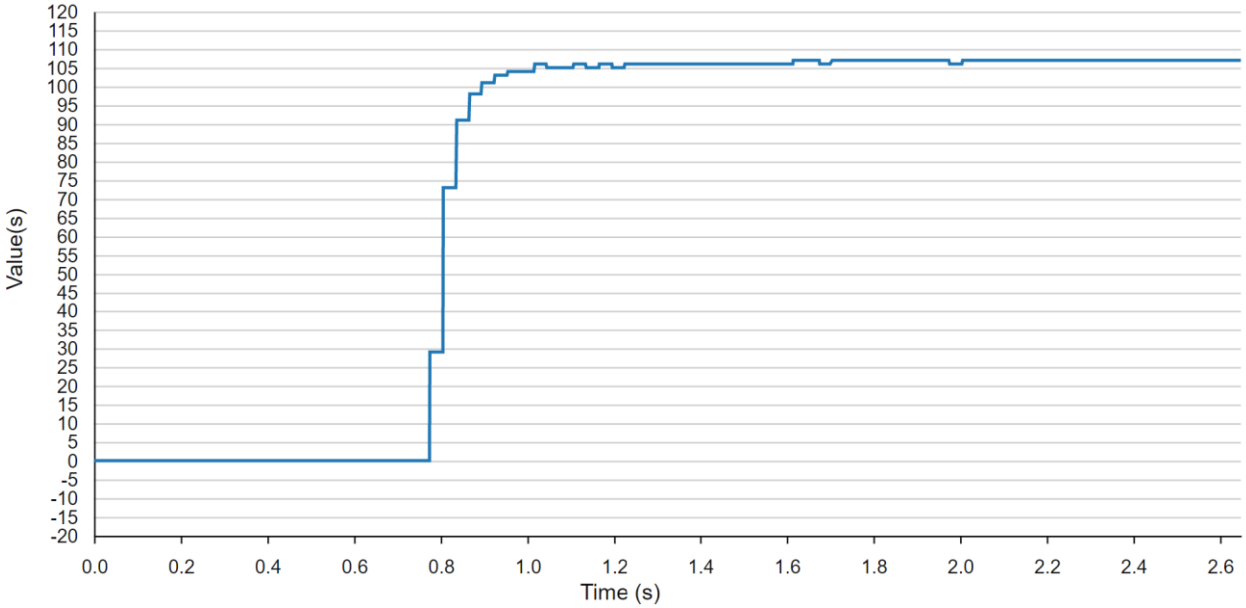
حال برای بدست آوردن مقادیر t_0 و τ می‌توان از ۳ روشی که در اسلایدهای درس آمده است استفاده کرد. برای اینکه کنترلر ما محافظه‌کارانه‌تر باشد باید از روش دو نقطه‌ای (two point method) استفاده کنیم. در روش two point method نسبت $p_u = \frac{t_0}{\tau}$ در مقایسه با بقیه روش‌ها بزرگتر بدست می‌آید. این نسبت نشان‌دهنده میزان کنترل‌ناپذیری سیستم می‌باشد درنتیجه هرچه بزرگتر باشد سیستم ما کنترل ناپذیرتر خواهد بود. حال اینکه ما سیستم خود را کنترل ناپذیرتر فرض می‌کنیم در واقع کنترلر خود را محافظانه‌تر محاسبه می‌کنیم.

در این روش به دو نقطه در نمودار بدست آمده نیاز داریم. یک نقطه در جایی که نمودار به ۶۳,۲ درصد مقدار نهایی خود می‌رسد (t_1) در واقع این نقطه جایی است که زمان برابر مقدار τ است. نقطه دیگر زمانی است که نموار به ۲۸,۳ درصد مقدار نهایی خود می‌رسد(t_2) این نقطه نیز جایی قرار دارد که زمان برابر $\frac{\tau}{3}$ است. حال براساس زمان نقاط بدست آمده و به کمک فرمول زیر مقدارهای t_0 و τ بدست می‌آوریم.

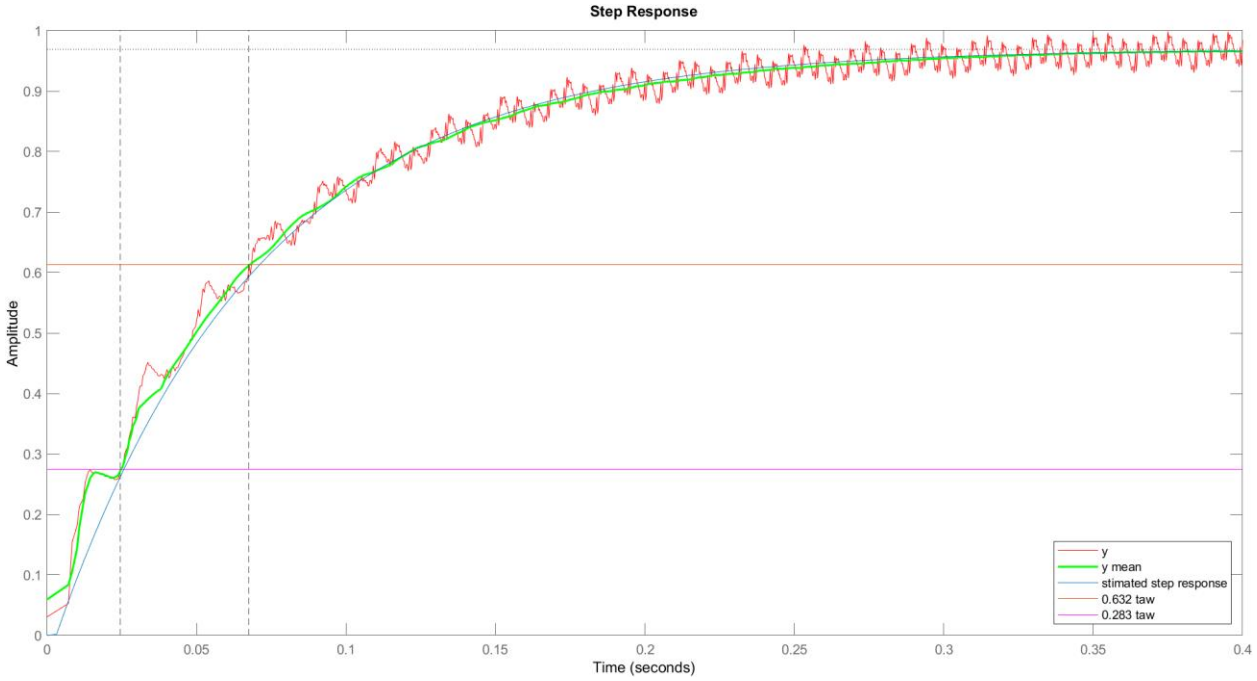
$$\tau = 1.5 * (t_1 - t_2) \quad , \quad t_0 = t_1 - \tau$$



شکل ۴: نمودار بدست آمده از Excel



شکل ۵: نمودار بدست آمده از Cube Monitor



شکل ۶: نمودار بدست آمده از MALAB

مشخصات پاسخ پله تقریبی رسم شده برای داده‌ها به شرح زیر می‌باشد:

$$\tau = 0.068, \quad t_0 = 0.003, \quad k = 0.969$$

داده‌های بدست آمده از طریق خطوط رسم شده روی نمودار با توجه به روش Tangent and two point به صورت زیر می‌باشد:

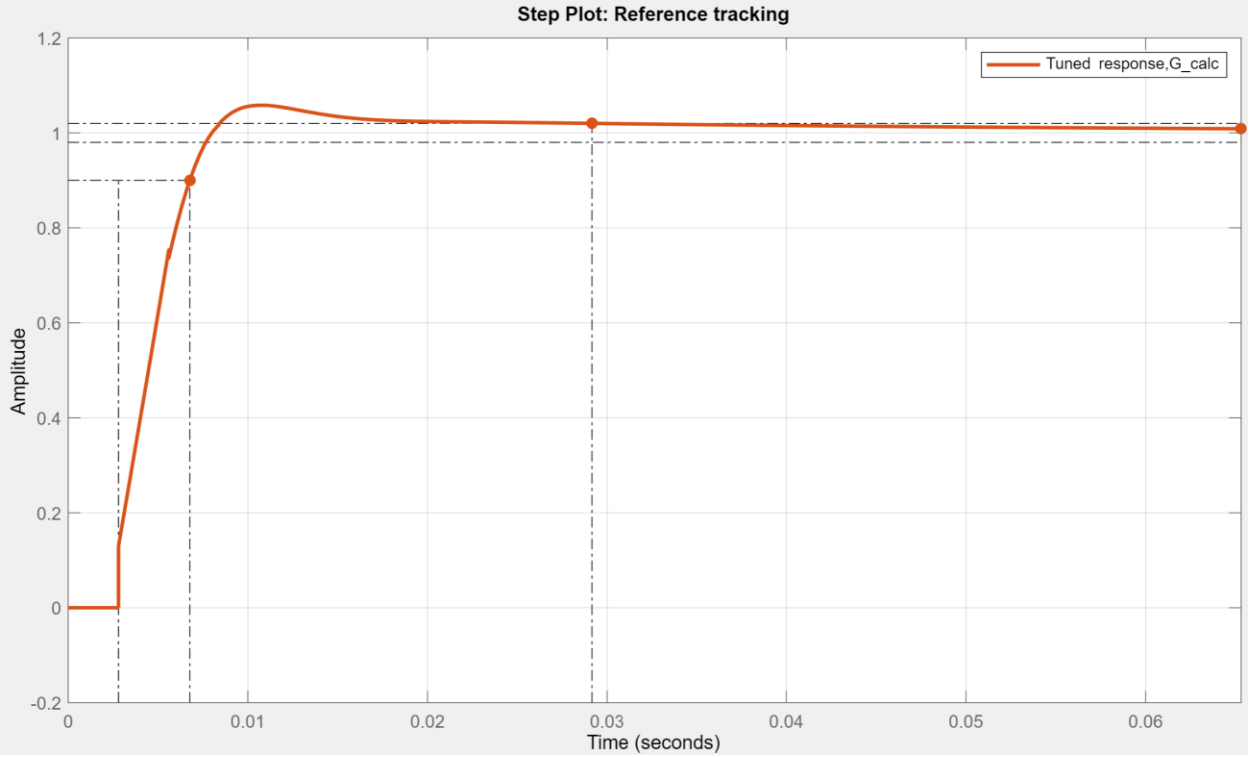
$$t_2 = 0.0244, \quad t_1 = 0.0675, \quad k = 0.9703$$

$$\tau = 1.5 * (0.0675 - 0.0244) \cong 0.0647 \Rightarrow t_0 = 0.0675 - 0.0647 = 0.0028$$

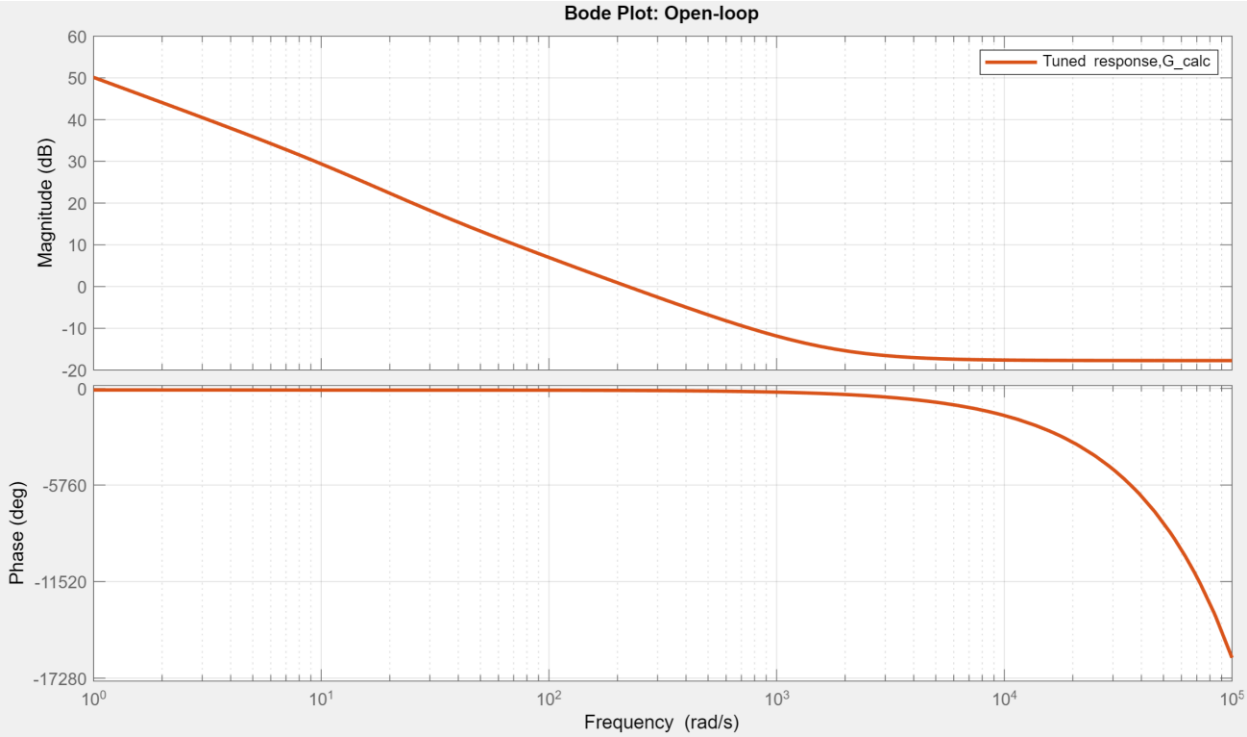
داده‌ها با تقریب بسیار خوبی برابر می‌باشند.

۲. طراحی PID با MATLAB

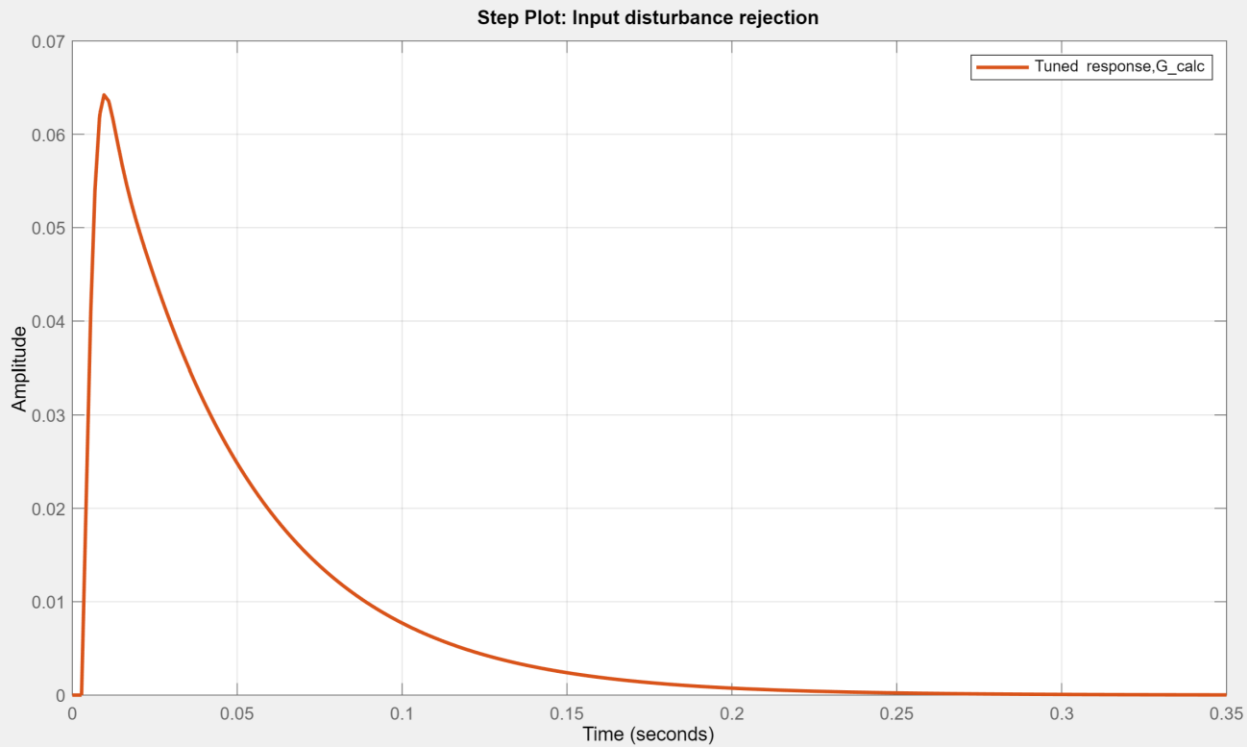
در مرحله بعدی مدل FOPTD را در MATLAB تعریف می‌کنیم و به کمک PID Tuner ضرایب PID و نوع آن را مشخص می‌کنیم. در ادامه نتایج بدست آمده از این toolbox آورده شده.



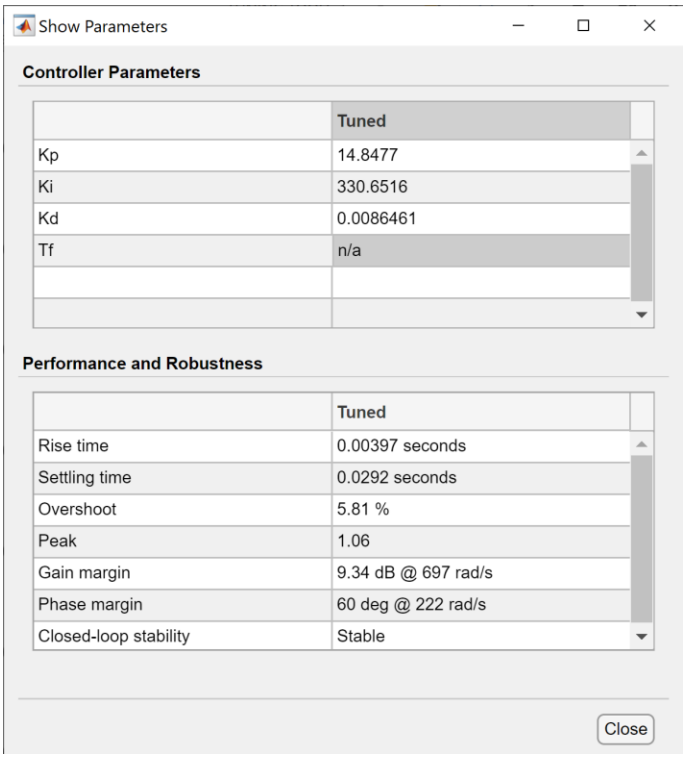
شکل ۷: پاسخ پله حلقه بسته سیستم به همراه کنترلر طراحی شده



شکل ۸: نمودار Bode سیستم به همراه کنترلر طراحی شده



شکل ۹: نمودار دفع اغتشاش توسط سیستم و کنترلر طراحی شده



شکل ۱۰: پارامترهای بدست آمده از PID Tuner

همانطور که در تصویر بالا مشاهده می‌شود، اورشوت سیستم تقریبا ۶درصد، زمان نشست آن کمتر از ۰/۰۳ ثانیه و حاشیه فاز آن ۶۰ درجه می‌باشد. مشخصات بدست آمده با این کنترلر مطلوب می‌باشد. توجه داریم که مقادیر بدست آمده برای PID موازی است.

۳. طراحی PID با روش QDR

برای طراحی PID می‌توان از روش‌های Ziegler-Nicholes و Quarter Decade Ratio استفاده کرد.
برای این منظور از جدولی که در ادامه آورده شده استفاده می‌کنیم. و نوع آن را مشخص می‌کنیم. در ادامه نت

Controller type	K_c	T_I	T_D
P	τ/Kt_0	-	-
PI	$0.9\tau/Kt_0$	$o.33t_0$	-
PID (series)	$1.2\tau/Kt_0$	$2t_0$	$o.5t_0$
PID (parallel)	$1.2\tau/Kt_0$	$2.5t_0$	$o.4t_0$

شکل ۱۱: جدول طراحی مقادیر کنترلر با روش QDR

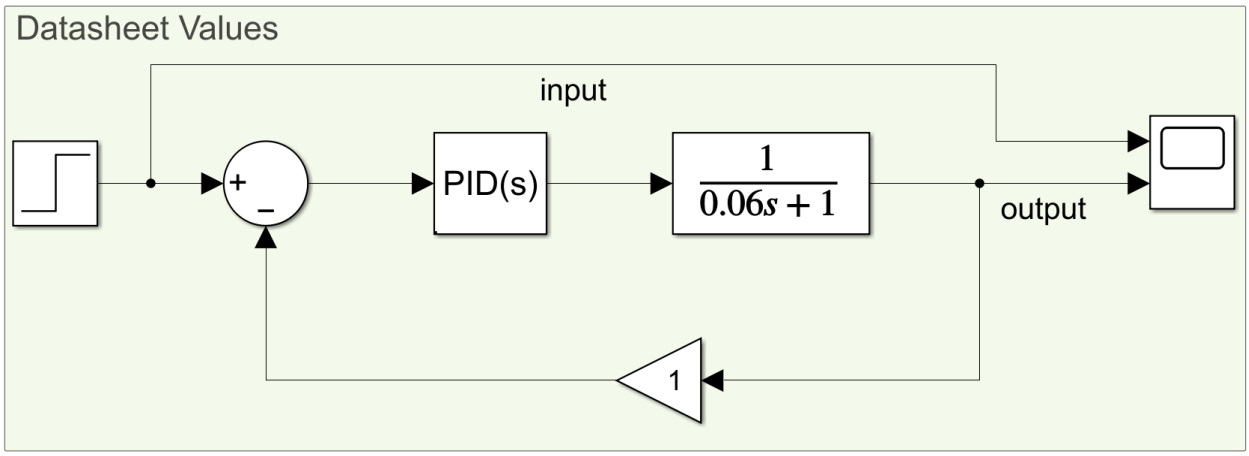
با توجه به مقادیر بدست آمده، ضرایب PID موازی به صورت زیر حاصل می‌شود:

$$\begin{cases} K_C = \dfrac{1.2 \times 0.0647}{0.9703 \times 0.0028} \cong 25.577 \Rightarrow K_i = \dfrac{K_C}{T_i} \cong 3653.857 \Rightarrow K_d = K_C T_d \\ T_i = 2.5 \times 0.0028 \cong 0.007 \\ T_d = 0.4 \times 0.0028 \cong 0.001 \\ \cong 0.0256 \end{cases}$$

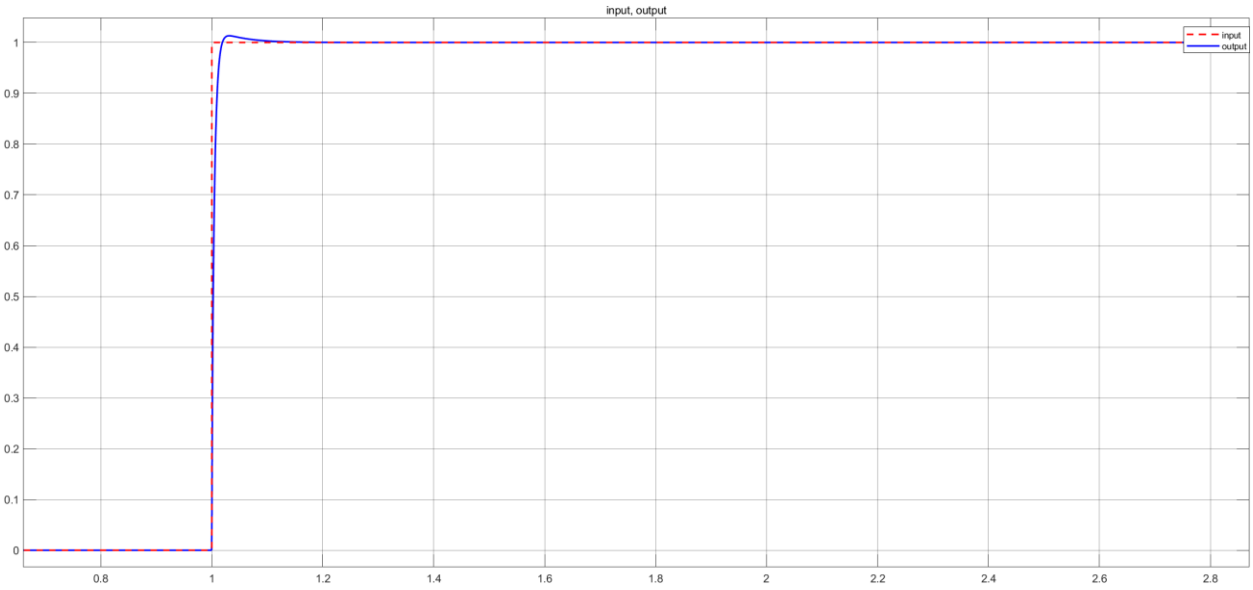
۳. شبیه‌سازی و مقایسه کنترل‌کننده‌ها

۱. شبیه‌سازی کنترلر برای سیستم مدل شده با مقادیر موجود در دیتاشیت

برای شبیه‌سازی سیستم در Simulink به صورت زیر عمل می‌کنیم. توجه داریم که پارامترهای بلوک PID در Simulink دارای فیلتر برای مشتق‌گیر است (زیرا مشتق‌گیر تنها علی نیست و در واقعیت پیاده‌سازی نمی‌شود). برای کاهش اثر قطب این فیلتر، ضریب N را بزرگ قرار می‌دهیم.



شکل ۱۲: سیستم تشکیل داده شده با پارامترهای دیتاشیت

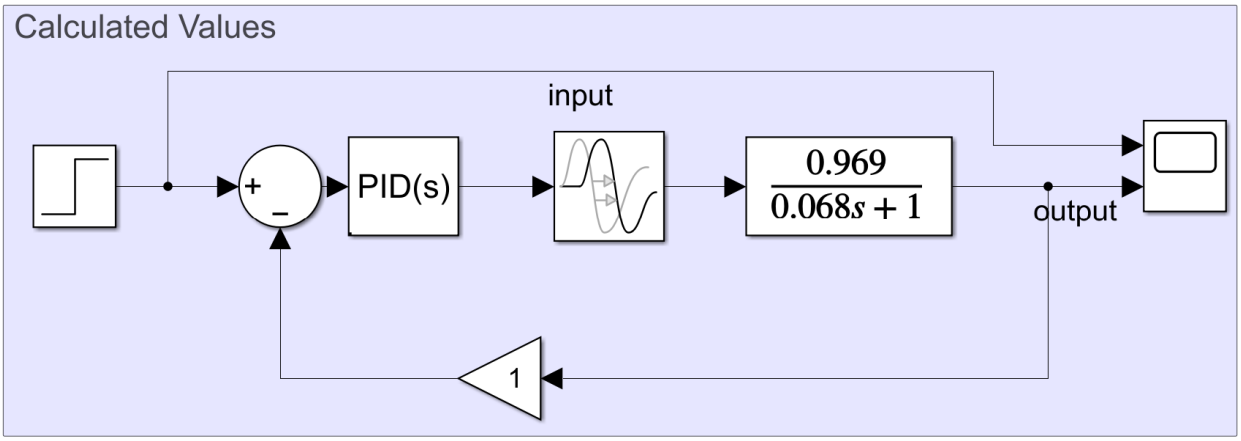


شکل ۱۳: خروجی سیستم با پارامترهای دیتاشیت و PID طراحی شده

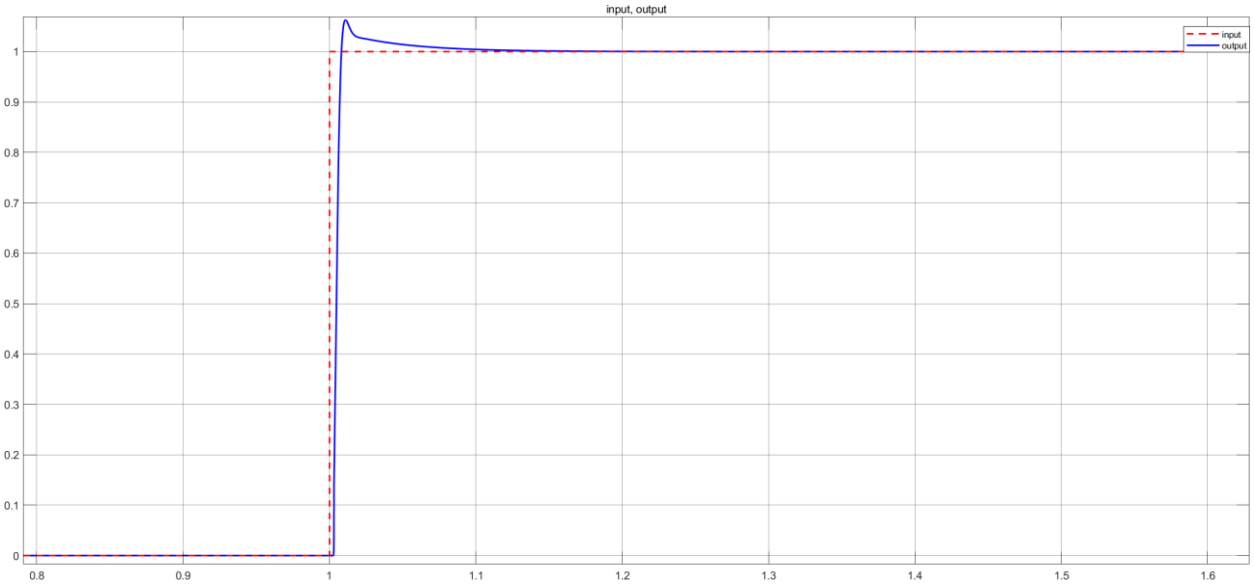
همانطور که مشاهده می‌شود، PID طراحی شده برای سیستم بدون دیلی و بدون تاخیر نیز به خوبی عمل می‌کند.

۲. شبیه‌سازی کنترلر برای سیستم مدل شده با مقادیر بدست آمده از tow-point method

در ادامه برای سیستم با تاخیر از بلوک Transport Delay برای اعمال تاخیر ۰/۰۰۳ ثانیه‌ای در سیستم استفاده می‌کنیم. توجه داریم که می‌توانستیم از تقریب‌های پاده استفاده کرد یا حتی با توجه به نسبت تاخیر و ثابت زمانی سیستم، به کل آن را نادیده گرفت، ولی برای دقیق‌تر بودن شبیه‌سازی‌ها از بلوک Delay استفاده می‌کنیم.



شکل ۱۴: سیستم تشکیل داده شده با پارامترهای محاسبه شده

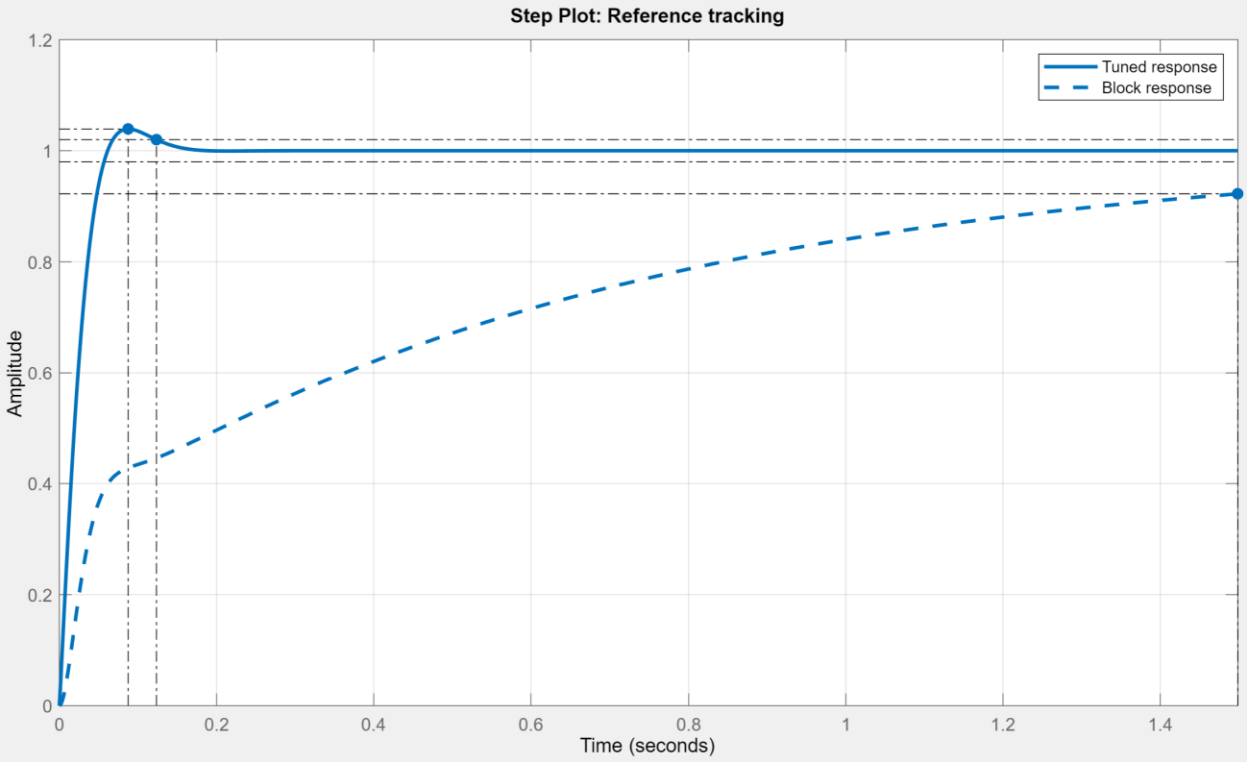


شکل ۱۵: خروجی سیستم با پارامترهای محاسبه شده و PID طراحی شده

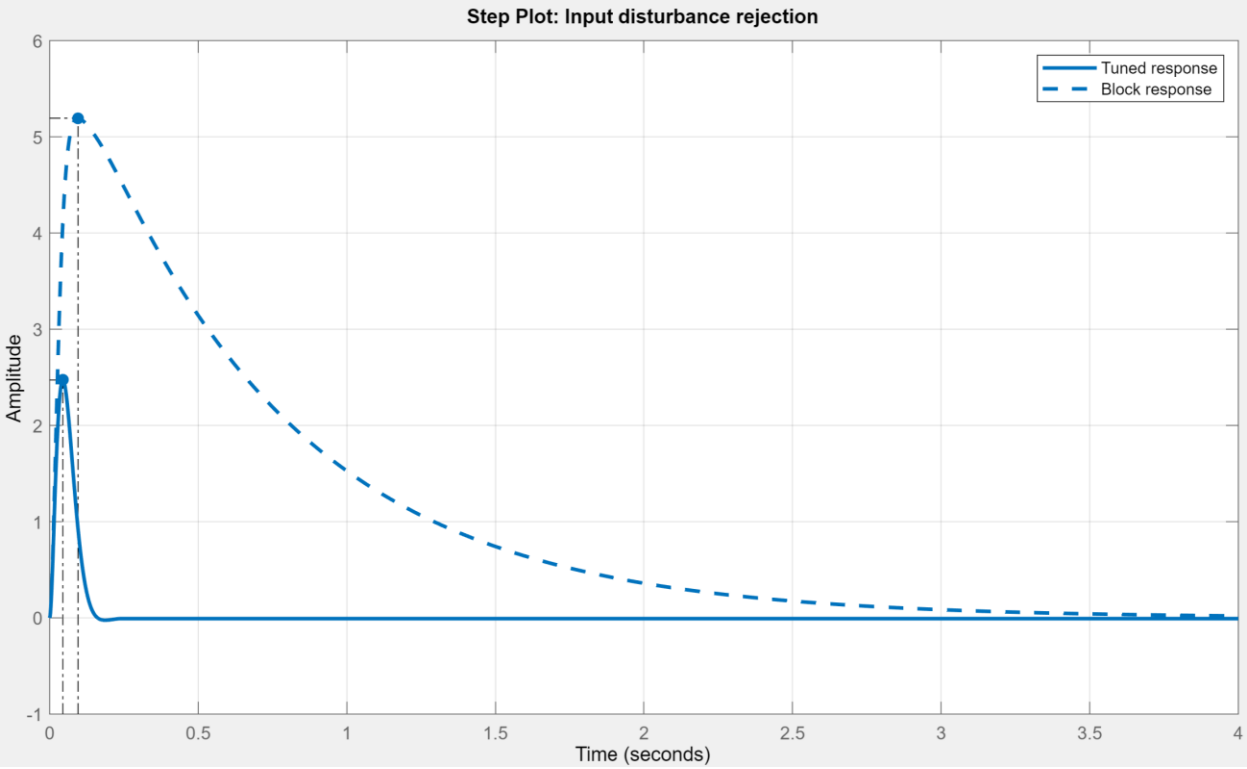
همانطور که مشاهده می‌شود، دو سیستم با تقریب خوبی عملکرد مشابهی دارند، ولی بالازدگی مدل به دست آمده از دیتاشیت کمتر است که به دلیل وجود تاخیر در سیستم دوم است. برای بهبود عملکرد در حضور تاخیر می‌توان از smith predictor استفاده کرد.

۳. شبیه‌سازی کنترلر سرعت برای موتور پیشنهای

با توجه به مشخصات بیان شده برای موتور DC از بلوک DC machine استفاده می‌کنیم. برای کنترل سرعت این بلوک از یک بلوک PID استفاده می‌کنیم. ورودی این کنترلر سیگنال خطا بدست آمده از مقایسه SP و باس سرعت خروجی موتور است. خروجی این کنترلر به یک Controlled Voltage Source وصل می‌شود که پایه منفی آن از پایه منفی آرمیچر گرفته شده و خروجی آن به پایه مثبت آرمیچر متصل می‌شود. به این ترتیب با کنترل ولتاژ آرمیچر، سرعت را کنترل می‌کنیم. در ادامه نمای سیستم تعریف شده و نتایج خروجی و تلاش کنترلی و مشخصات PID بدست آمده از طریق PID tuner آورده شده. توجه داریم که ولتاژ تغذیه filed باتوجه به مشخصات موتور انتخابی ۳۰۰ ولت قرار داده شده.



شکل ۱۶: پاسخ پله حلقه بسته سیستم به همراه کنترلر tune شده



شکل ۱۷: نمودار دفع اغتشاش توسط سیستم و کنترلر طراحی شده

Show Parameters

Controller Parameters

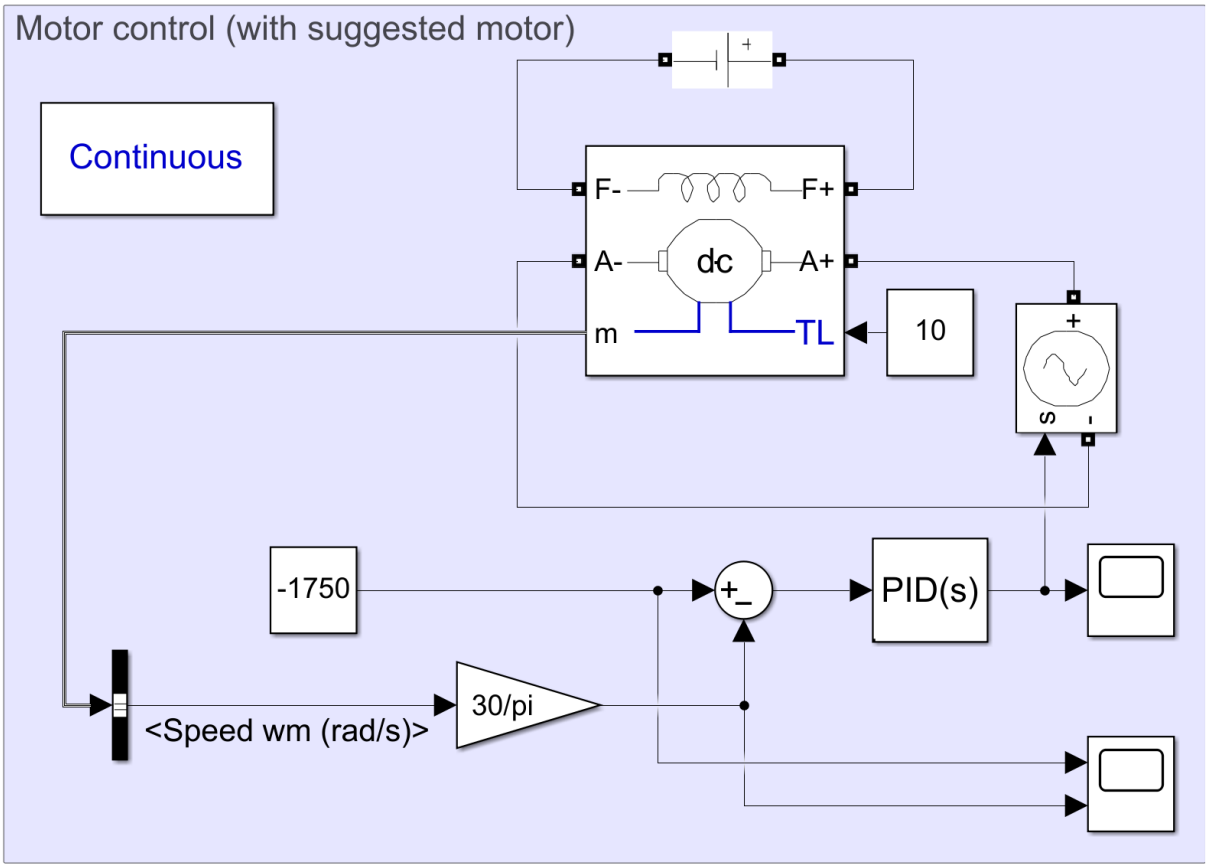
	Tuned	Block
P	0.21198	0.061458
I	5.7195	0.23723
D	0.0017099	0.0011205
N	4571.3119	23.2685

Performance and Robustness

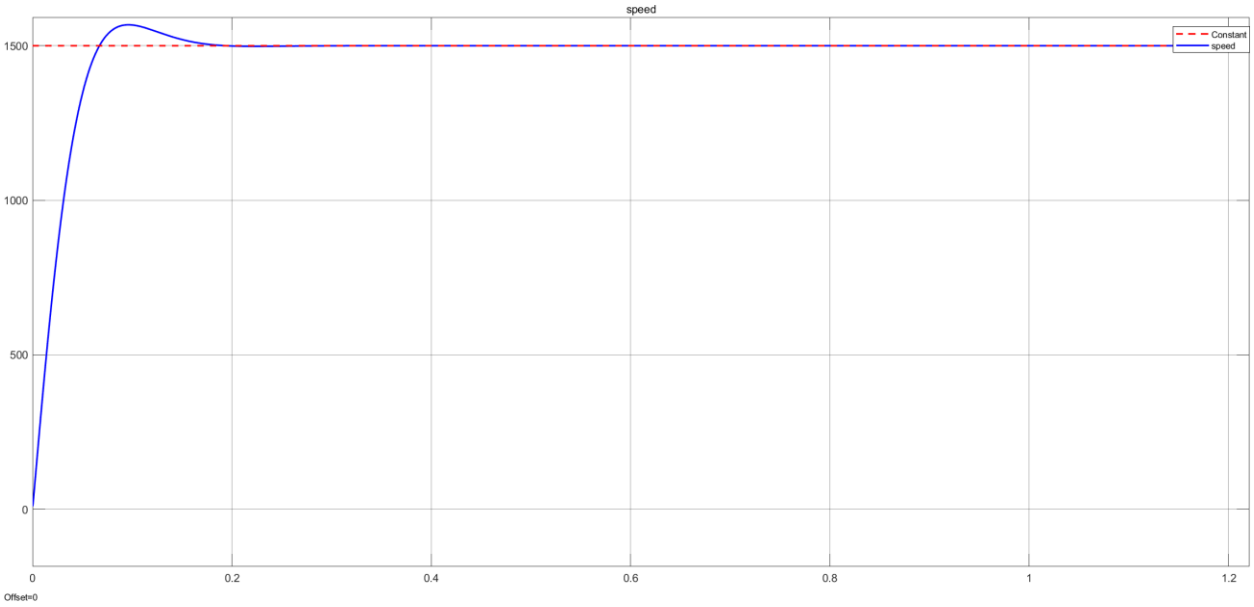
	Tuned	Block
Rise time	0.0415 seconds	1.31 seconds
Settling time	0.124 seconds	2.44 seconds
Overshoot	3.91 %	0 %
Peak	1.04	0.999
Gain margin	Inf dB @ Inf rad/s	Inf dB @ Inf rad/s
Phase margin	74.5 deg @ 40 rad/s	116 deg @ 2.44 rad/s
Closed-loop stability	Stable	Stable

Close

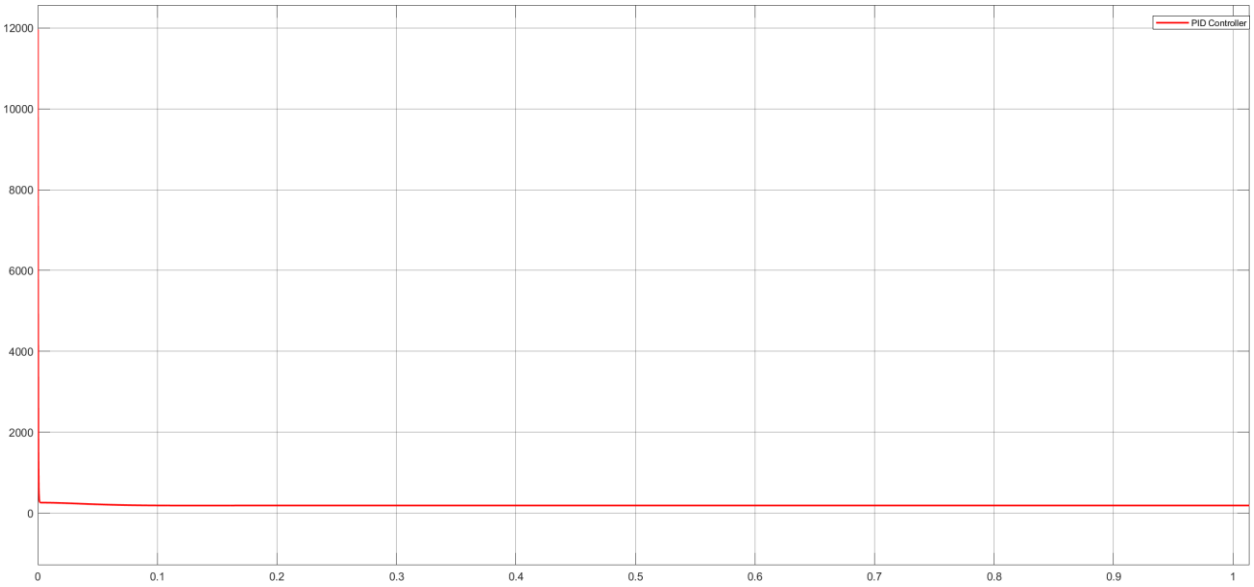
شکل ۱۸: پارامترهای بدست آمده از PID Tuner



شکل ۱۹: سیستم تشکیل داده شده با موتور پیشنهادی



شکل ۲۰: خروجی سیستم با موتور پیشنهادی و PID طراحی شده

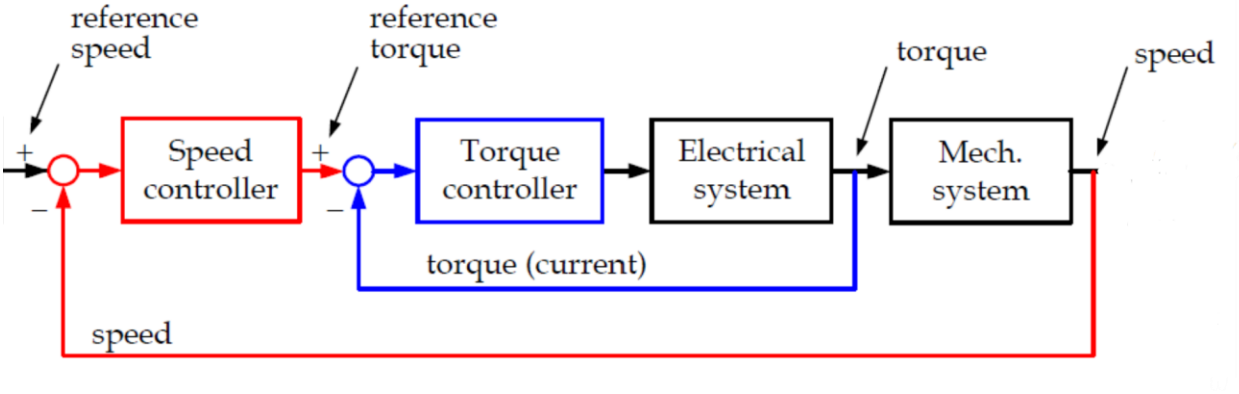


شکل ۲۱: تلاش کنترلی

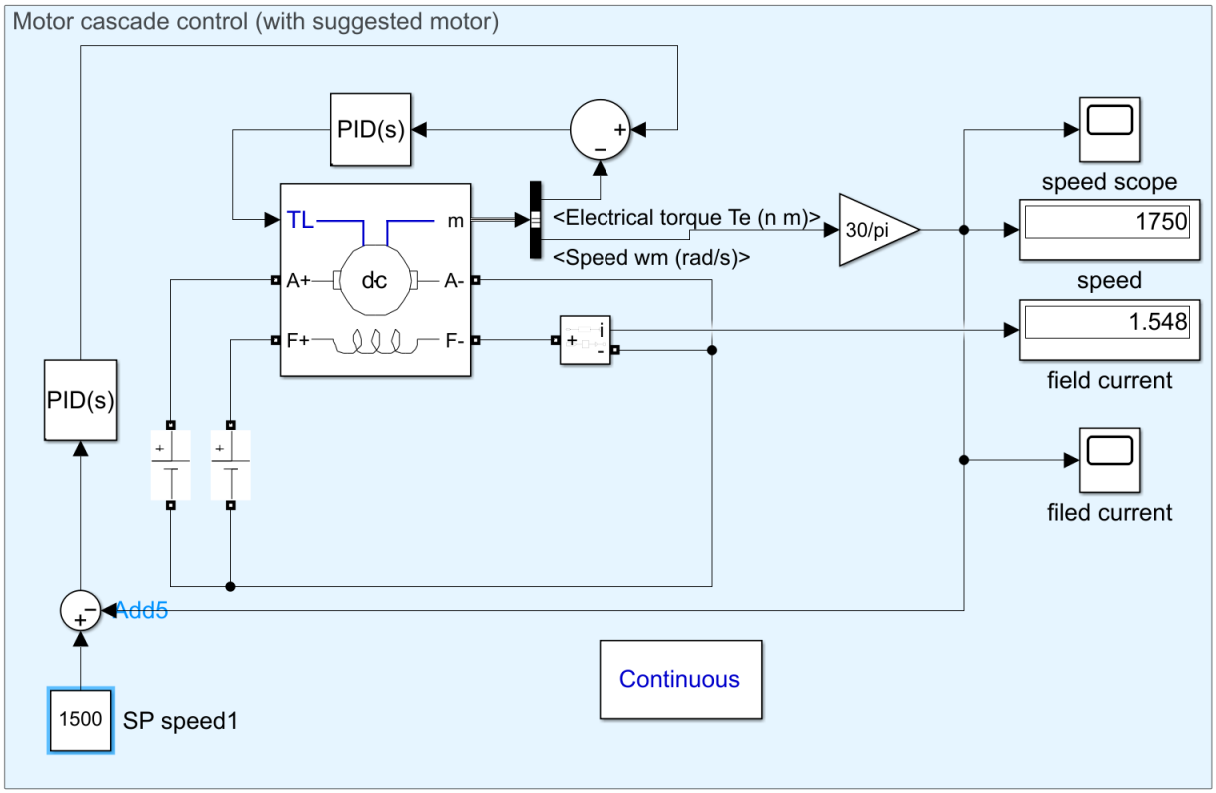
همانطور که مشاهده می‌شود، تلاش کنترلی از حد مجاز سرعت موتور فراتر رفته و actuator قابلیت این موضوع را ندارد. به همین دلیل، می‌توان با کاهش K_p و کندتر کردن سیستم، این مشکل را برطرف کرد.

۴. شبیه‌سازی کنترلر cascade برای موتور پیشنهای

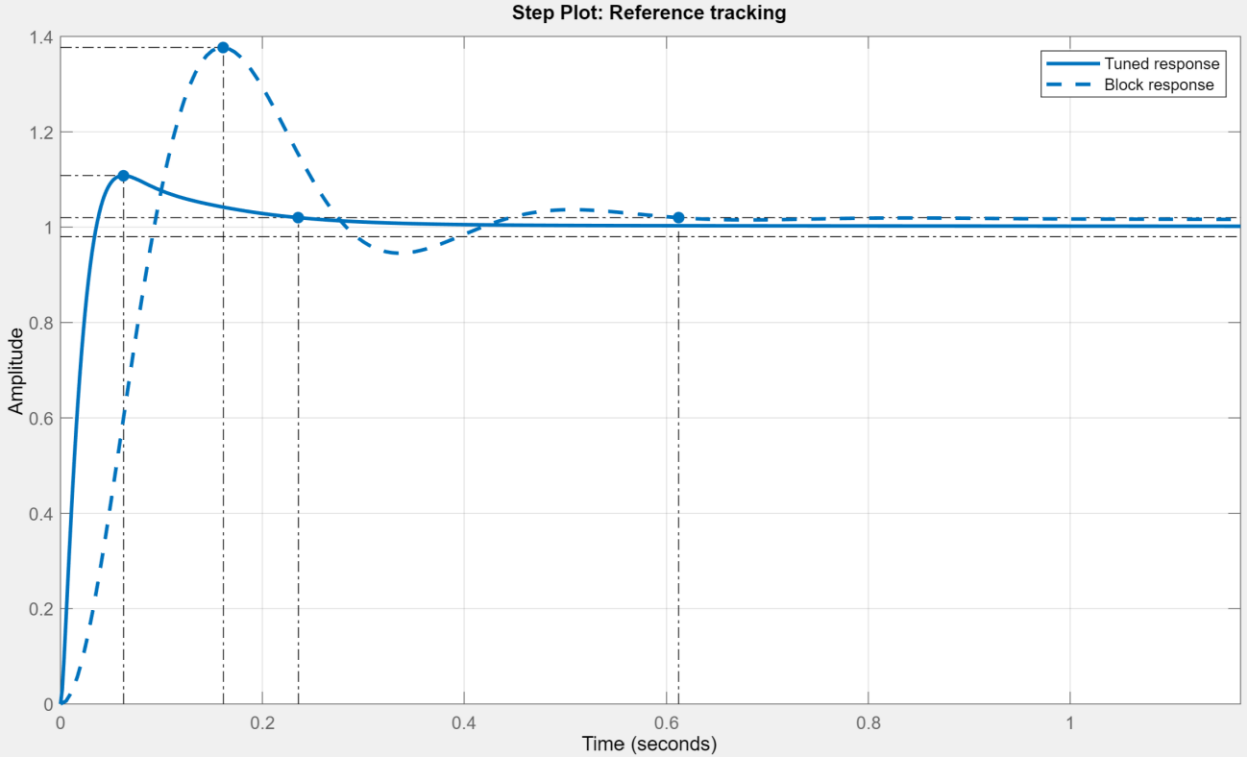
در سیستم قبلی لوپ داخلی کنترل جریان (گشتاور) را اضافه می‌کنیم. به این ترتیب ورودی موتور گشتاور متغییر خواهد شد که در تصویر پایین نمای بلوک دیاگرامی سیستم و نمای اصلی در شبیه‌سازی آورده شده. در این سیستم ولتاژ تغذیه آرمیچر نیز ثابت و برابر ۲۴۰ ولت است.



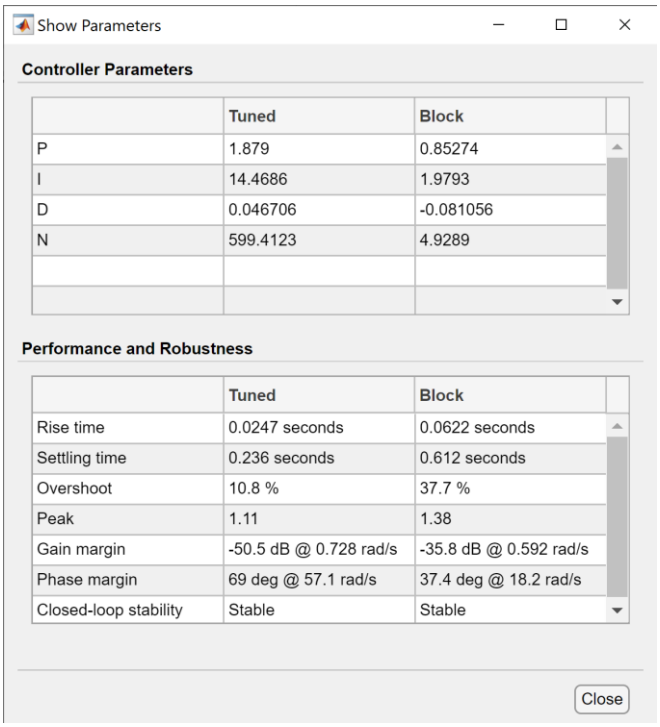
شکل ۲۲: بلوک دیاگرام کنترل سرعت با لوپ cascade روی جریان یا گشتاور



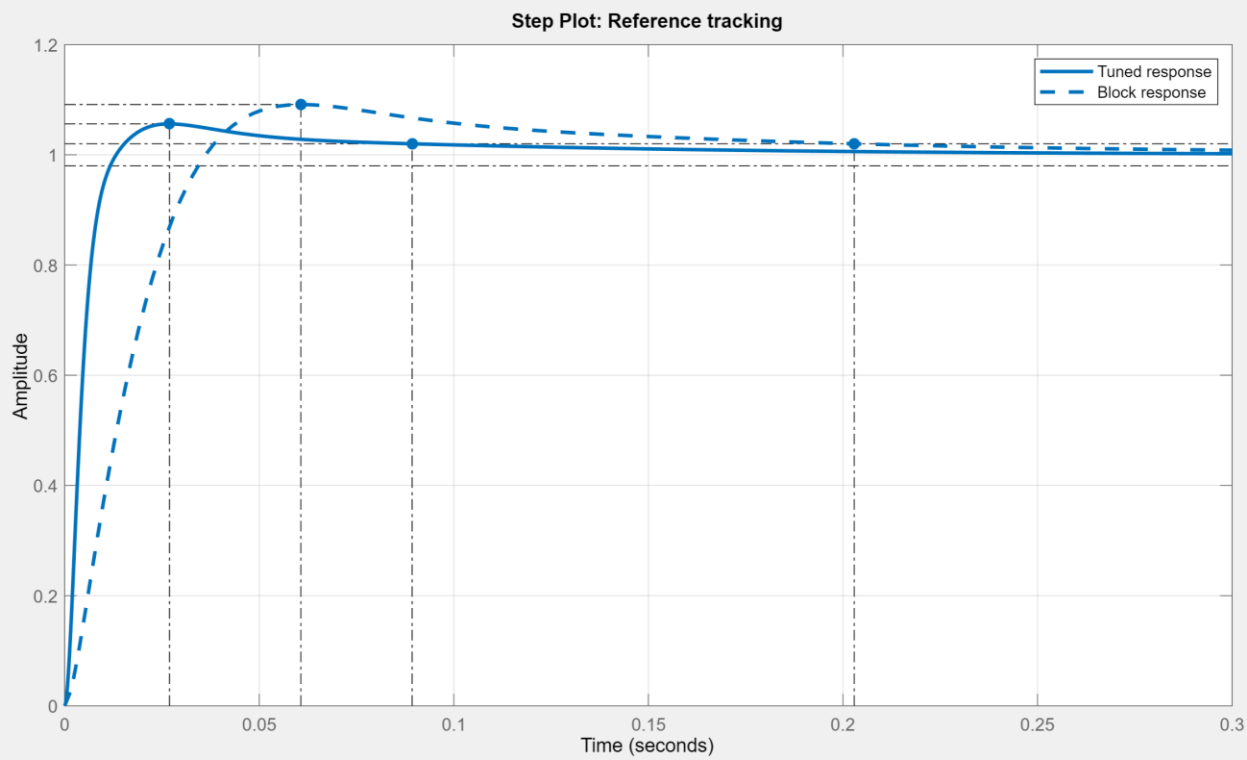
شکل ۲۳: سیستم cascade تشکیل داده شده با پارامترهای محاسبه شده



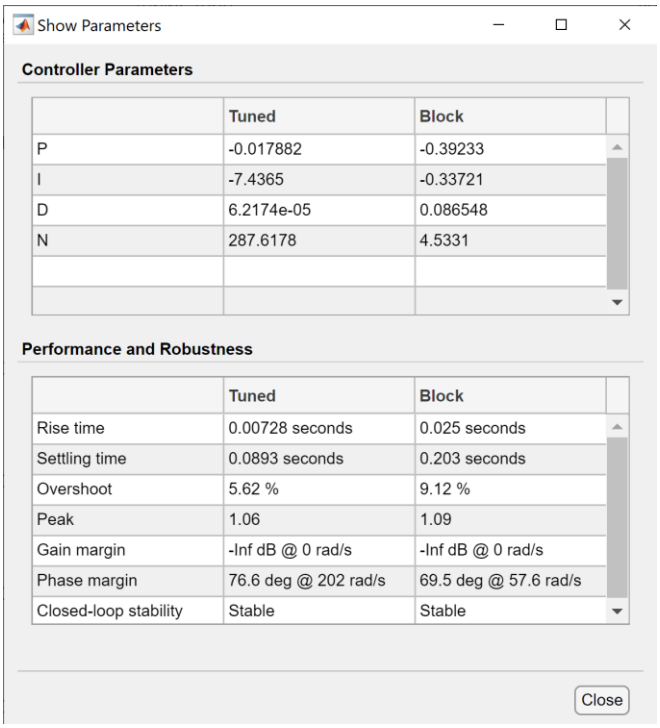
شکل ۲۴: پاسخ پله حلقه بسته سیستم به همراه کنترلر tune شده در لوپ خارجی سیستم



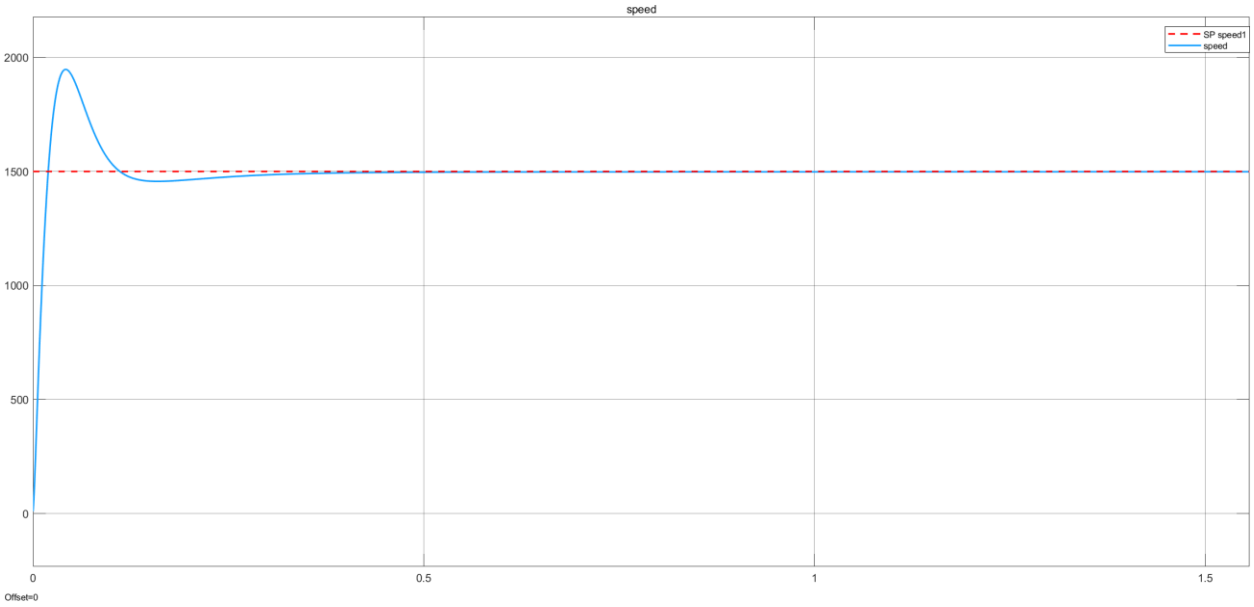
شکل ۲۵: پارامترهای بدست آمده از PID Tuner برای لوپ خارجی



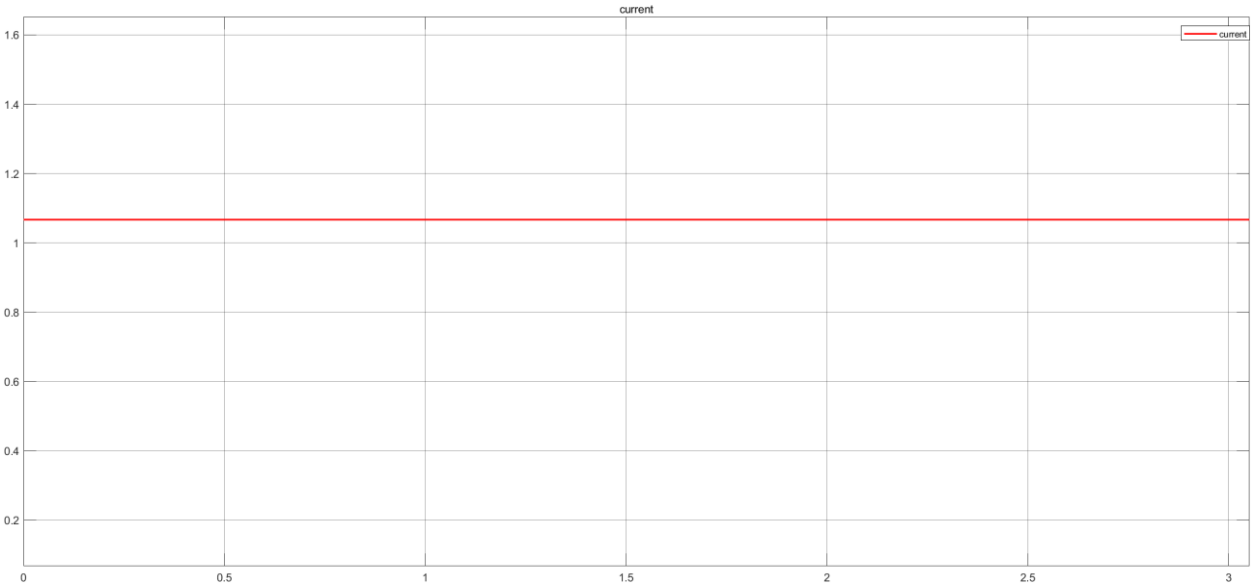
شکل ۲۶: پاسخ پله حلقه بسته سیستم به همراه کنترلر tune شده در لوپ داخلی سیستم



شکل ۲۷: پارامترهای بدست آمده از PID Tuner برای لوپ داخلی



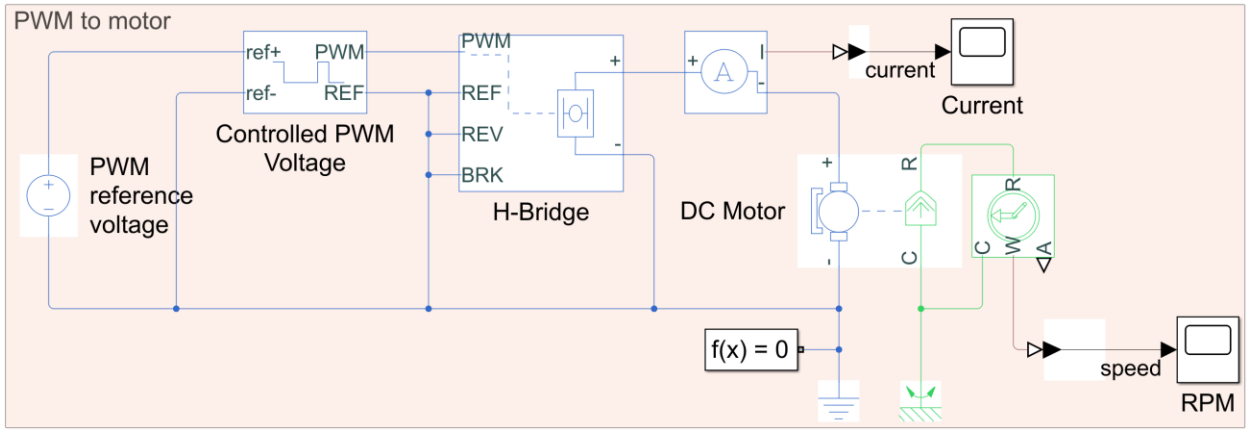
شکل ۲۸: خروجی سرعت با PIDهای طراحی شده



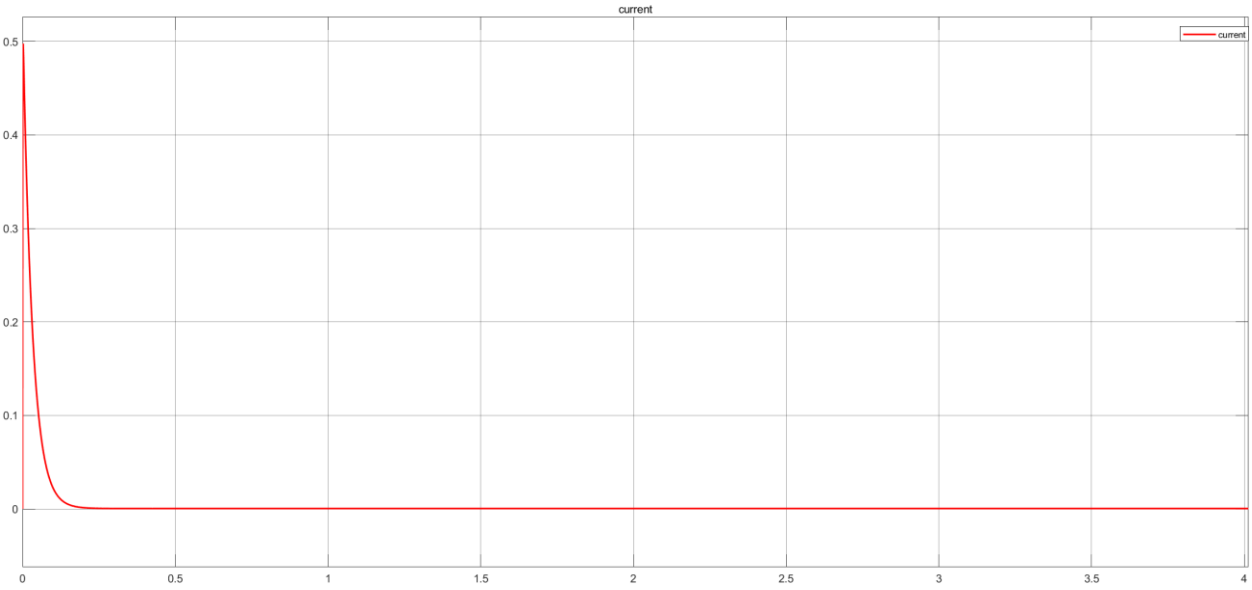
شکل ۲۹: خروجی جریان سیستم با PIDهای طراحی شده

۵. شبیه‌سازی H-bridge

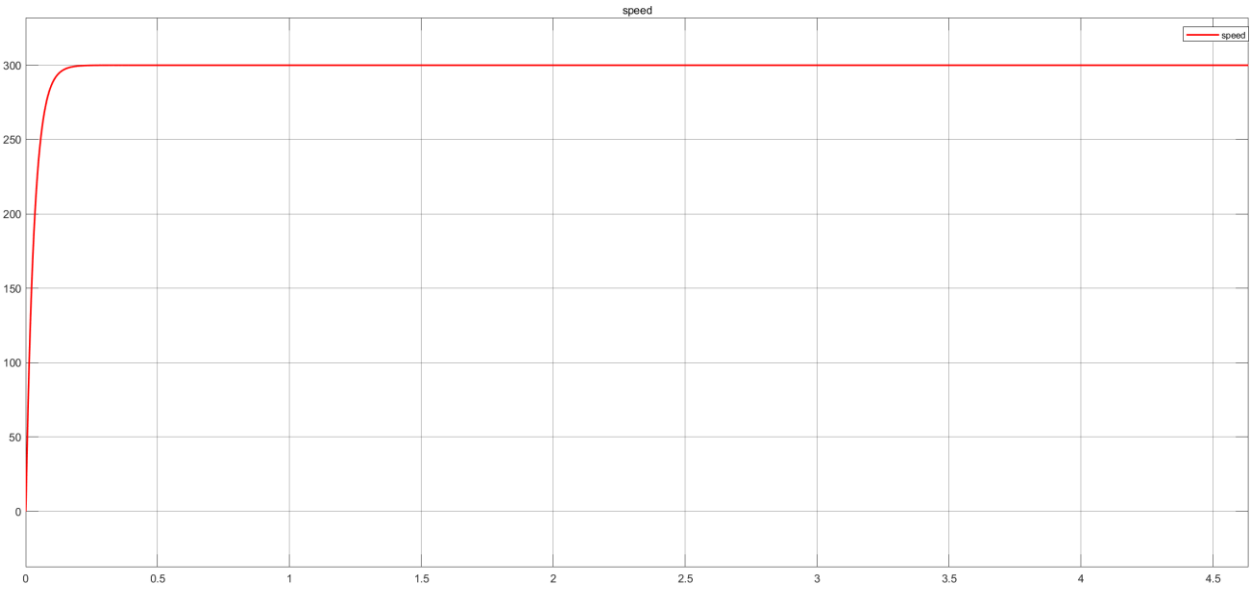
برای شبیه‌سازی H-bridge و PWM موتور مانند تصویر زیر عمل می‌کنیم. در این مدل یک ولتاژ مرجع به PWM با فرکانس مشخص تبدیل شده و به H-bridge داده می‌شود. خروجی این ماژول به یک جریان‌سنج و موتور داده می‌شود و تغییرات سرعت موتور پلات می‌شود.



شکل ۳۰: سیستم راه‌اندازی موتور با H-bridge و PWM



شکل ۳۱: خروجی جریان

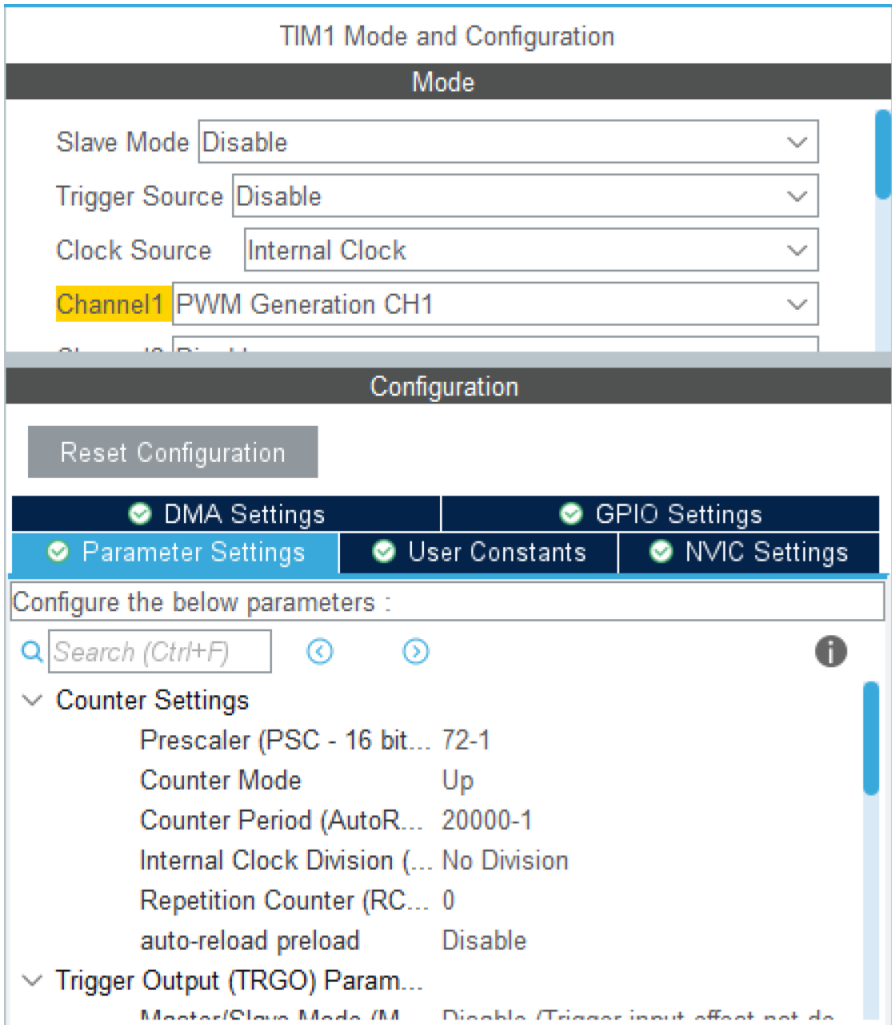


شکل ۳۲: خروجی سرعت

۴. کد پیاده‌سازی شده در میکروکنترلر

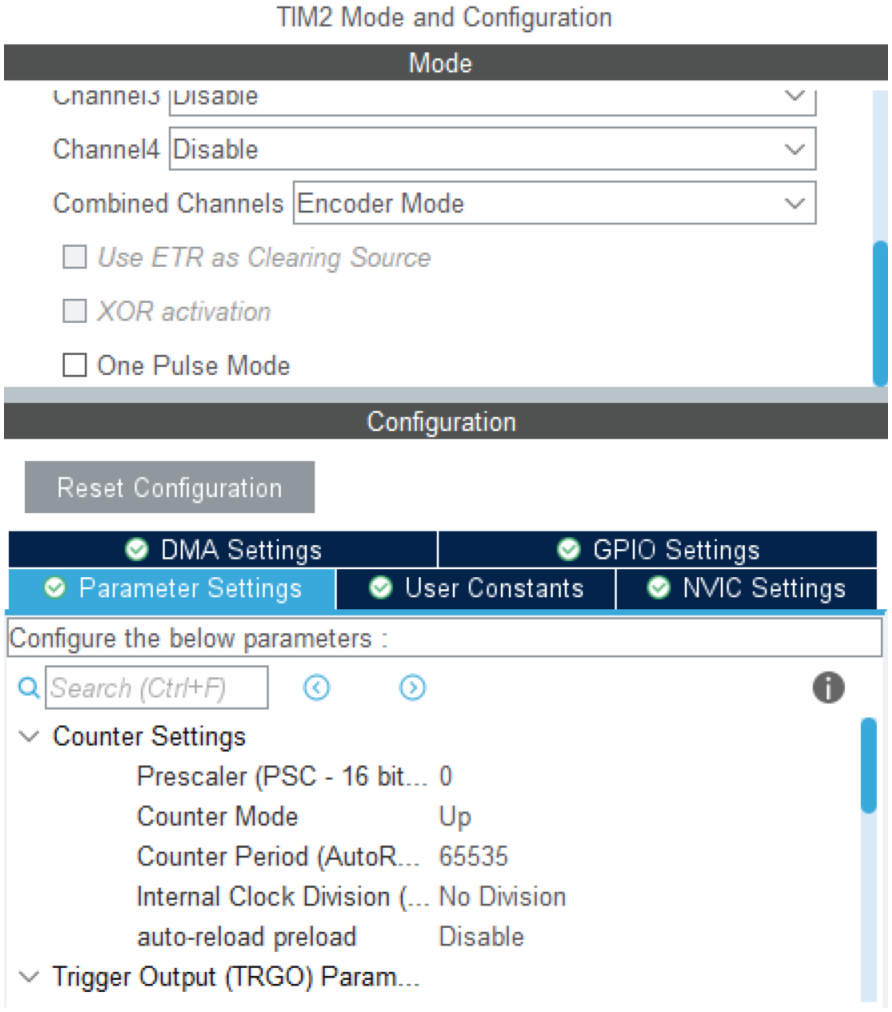
۱. پایه‌های میکروکنترلر

برای دریافت داده‌های انکودر و ارسال PWM و مشخص کردن زمان نمونه‌برداری از سه تایمر استفاده می‌کنیم. تایمر TIM1 برای تولید PWM استفاده می‌شود. به این ترتیب که رجیستر PSC آن ۷۱ (کاهش فرکانس به ۱MHz) و رجیستر ARR آن ۱۹۹۹۹ (برای افزایش رزولوشن سیگنال ایجاد شده، زیرا تا این شماره می‌شمارد) تنظیم می‌شود. کانال ۱ این تایمر (پایه PA8) برای تولید این سیگنال اشغال می‌شود. این پایه به ENA درایور متصل می‌شود.



شکل ۳۳: تنظیمات تایمر۱

تایمر TIM2 برای دریافت مقادیر انکودر استفاده می‌شود. به این ترتیب که رجیستر PSC آن ۰ (عدم کاهش فرکانس برای دقت بیشتر نمونه‌برداری) و رجیستر ARR آن به ماکزیمم ۶۵۵۳۵ (برای افزایش رزولوشن سیگنال ایجاد شده) تنظیم می‌شود. از این تایمر در مود Combined channel و در حالت Encoder mode استفاده می‌شود. کانال ۱ و ۲ این تایمر (پایه‌های PA0 و PA1) به دو سیگنال خروجی از انکودر متصل می‌شوند.

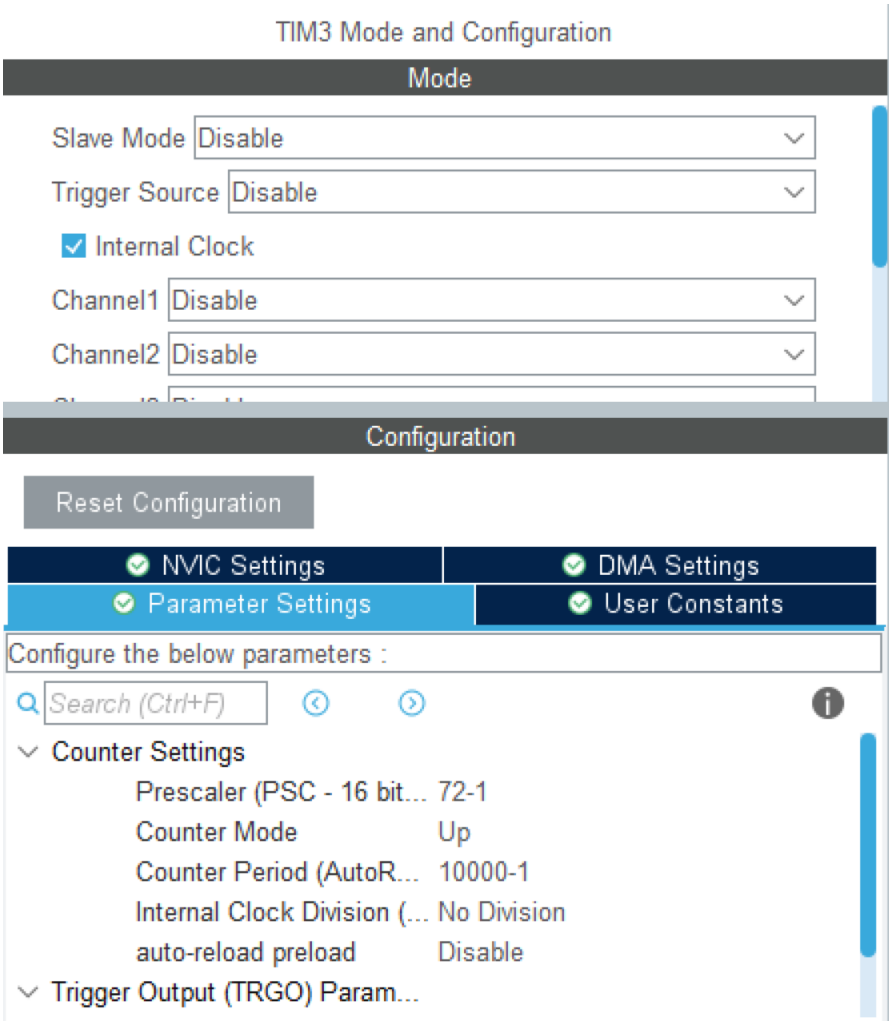


شکل ۳۴: تنظیمات تایمر۲

تایمر TIM3 برای محاسبه زمان‌نمونه برداری استفاده می‌شود. به این ترتیب که رجیستر PSC آن ۷۱ و رجیستر ARR آن ۹۹۹۹ تنظیم می‌شود. در این تایمر از هیچ پورتی استفاده نمی‌شود. با توجه به رابطه آورده شده در ادامه می‌توان فرکانس نمونه برداری را محاسبه نمود:

$$f_{out} = f_{in} \times \frac{1}{PSC+1} \times \frac{1}{ARR+1} = 72MHz \times \frac{1}{72} \times \frac{1}{10000} = 100Hz$$

$$T_s = \frac{1}{f_{out}} = 10\ ms$$



شکل ۳۵: تنظیمات تایمر۳

تایمرهای ۲ و ۳ به صورت interrupt تنظیم می‌شوند. زیرا می‌خواهیم سر زمان نمونه‌برداری سرعت را محاسبه کنیم و همچنین در صورت overflow کردن تایمر ۲، آن را شناسایی کنیم و جهت را مشخص کنیم. در صورت استفاده از قابلیت interrupt می‌توان کنترلر سیستم را به صورت real-time پیاده‌سازی نمود. همچنین در این صورت باید کدهای مورد نظر خود را در تابع callback مربوط به وقفه رخ داده بنویسیم.

در این کد از یک ADC برای دریافت مقدار پتانسیومتر و تبدیل آن به مقدار دیجیتال دیجیتال استفاده می‌کنیم که پایه مربوط به آن PA5 است. توجه داریم که داده‌های بدست آمده از این روش دارای نویز و تغییرات بسیاری می‌باشند، به همین دلیل از یک خازن برای دمپ کردن این تغییرات استفاده می‌کنیم.

دو پایه PA6 و PA7 به پایه‌های IN1 و IN2 درایور متصل می‌شوند که مسئول مشخص کردن جهت چرخش موتور می‌باشند. در صورتی که اگر به ترتیب High و Low باشند، موتور ساعتگرد و اگر به ترتیب Low و High باشند، موتور پادساعتگرد می‌چرخد.

برای تغذیه برد انکودر، مدار تقسیم مقاومتی پتانسیومتر، تغذیه پروگرامر و روشن کردن LED از پایه‌های ۳/۳V موجود روی برد استفاده می‌کنیم. توجه داریم که برای یکی شدن زمین‌ها در کل سیستم باید GND برد را به زمین تغذیه موتور متصل کنیم در غیر این صورت کنترلر به نحوه مطلوب عمل نخواهد کرد.

۲. الگوریتم PID

برای پیاده‌سازی این الگوریتم در حالت کلی دو متغییر PV و direction را کنترل می‌کنیم. در این الگوریتم تمامی متغیرها را به ۰ تا ۱۰۰ درصد scale می‌کنیم که توانایی مقایسه آن‌ها با یکدیگر را داشته باشیم.

در ابتدا با ورود به تابع interrupt بررسی می‌کنیم کدام تایمر باعث ایجاد این امر شده. اگر تایمر ۲ باشد، یکی یک دور کامل چرخیده‌ایم و یک flag را ۱ می‌کنیم. اگر تایمر ۴ باشد، یعنی به زمان‌نمونه برداری رسیده ایم و الگوریتم کنترلی باید پیاده شود.

```
if(htim == &htim2)
	flag = 1;
if (htim == &htim4){
	.
	.
	.
```

کد شماره ۱: مشخص کردن تایمر مسئول ایجاد اینتراپت

در حلقه کنترلی ابتدا موقعیت را در متغیر Position ذخیره می‌کنیم (این مقدار از شمارنده تایمر ۲ بدست آمده). سپس SetPoint از ADC پایه PA5 بدست می‌آید. مقدار آنالوگ ورودی به این پایه از یک تقسیم مقاومتی به کمک پتانسیومتر بدست می‌آید. تبدیل شده این ورودی به دیجیتال مقداری بین ۰ تا ۴۰۰۰ دارد، پس برای تبدیل آن به بازه ۱۰۰- تا ۱۰۰ درصد از آن ۲۰۰۰ واحد کم کرده و بر ۴۰ تقسیم می‌کنیم. برای دمپ کردن تغییرات پتانسیومتر و نویزهای آن، از یک خازن بین زمین و خروجی آن استفاده می‌کنیم. حالا باید مقدار PV که در این مثال سرعت است را بدست بیاوریم. برای این کار مطابق زیر عمل می‌کنیم که تاثیرات overflow و multi turn بودن سیستم را نیز در نظر گرفته باشیم.

```
Position = __HAL_TIM_GET_COUNTER(&htim2);
SP = HAL_ADC_GetValue(&hadc1)-2000/40.;
if (flag == 0)
	PV = ((float)Position - (float)LastPosition)/(42) * 100;
else if(Position >  LastPosition)
	PV = ((float)Position - 65535 - (float)LastPosition)/(42) * 100;
else
	PV = ((float)Position + 65535 - (float)LastPosition)/(42) * 100;
flag = 0;
LastPosition = Position;
```

کد شماره ۲: محاسبه سرعت در هر نمونه‌برداری با احتساب چرخش بیش از یک دور

محاسبه PV به درصد به صورت زیر انجام می‌شود:

$$Max\ PV = Max\ RPM \times \frac{1\ min}{60\ s} \times \frac{1\ s}{100\ T_s} = Max\ RPT_s = \frac{1}{40}$$

$$Max\ PPT_s = PPR \times Max\ RPT_s = 1150 \times \frac{1}{40} = 28.75$$

PPR همان تعداد پالس‌ها در هر دور چرخش موتور است که با تست‌های انجام شده، ۱۱۵۰ است. برای تبدیل شدن PV به درصد باید آن را به ماکزیمم تقسیم کنیم. با مشاهده عملی متوجه شدیم که موتور در ولتاژهای بالاتر با سرعتی نزدیک RPM می‌چرخد، به همین دلیل با آزمون و خطا، مقدار ۴۲ را برای ماکزیمم پالس دریافت شده در هر بازه نمونه‌برداری بدست آوردیم. پس سرعت به درصد برحسب رابطه زیر بدست می‌آید:

$$PV_{\%} = PV \times \frac{100}{42}$$

پس از این مرحله، ارور را محاسبه می‌کنیم. توجه داریم که برای محاسبه ارور از Reverse control action استفاده می‌کنیم. به این ترتیب با افزایش SP خطا افزایش یافته و باعث افزایش ولتاژ ورودی و Duty Cycle می‌شود. همچنین DErr را برای در نظر گرفتن derivative overrun لحاظ می‌کنیم.

```
Err = SP - PV;  // Reverse control action
DErr = -PV;         // derivative overrun
```

کد شماره ۳: محاسبه خطا و با در نظر گرفتن derivative overrun

ابتدا چک می‌کنیم که در ناحیه P.B. هستیم یا خیر. در صورتی که خارج از P.B. باشیم باید از کنترلر on/off استفاده کنیم. برای محاسبه این باند به روش زیر عمل می‌کنیم:

$$P.B. = \frac{100}{K_p} = \frac{100}{14.85} = 6.734\%$$

پس اگر ارور بزرگتر از ۳/۳۶۷ درصد باشد موتور با ماکزیمم سرعت در جهت مثبت و اگر کمتر از ۳/۳۶۷- درصد باشد، خروجی صفر می‌شود.

حال اگر در ناحیه P.B. باشیم باید از کنترلر PID استفاده کنیم. بخش انتگرال‌گیر در هر مرحله با ضرب ارور در زمان نمونه‌برداری جمع می‌شود. برای جلوگیری از اثر windup در هر مرحله مقدار انتگرال‌گیر باید چک شود تا در صورتی که از مقدار max یا min خود عبور کرد به ترتیب برابر همان مقادیر max و min قرار بگیرد.

$$T_i = \frac{K_p}{K_i} \cong \frac{14.85}{330.65} \cong 0.045$$

همچنین درصورتی که علامت Err با integral متفاوت باشد، به معنای تغییر جهت موتور می‌باشد. در این حالت برای افزایش سرعت کنترلر، integral را پاک می‌کنیم.

اگر حداکثر ۷۰ درصد بازه خروجی را به فعالیت انتگرال‌گیر اختصاص دهیم، می‌توان باند انتگرال‌گیر را به صورت زیر تعریف کنیم:

$$Max\ int = \frac{70}{T_S \times K_i} = \frac{70}{0.01 \times 330.65} \cong 21.17$$

بعد از بررسی انتگرال‌گیر به سراغ محاسبه مشتق‌گیر می‌رویم. مشتق‌گیر برابر است با اروری که در این مرحله داریم منهای ارور مرحله قبل تقسیم بر زمان نمونه‌برداری. برای جلوگیری از اثر overrun مشتق‌گیر باید SP را از محاسبه ارور مشتق‌گیر حذف کنیم تا در صورت تغییر ناگهانی ورودی، این تغییرات باعث ایجاد خروجی‌های بزرگ در مشتق‌گیر نشوند.

$$T_d = \frac{K_d}{K_p} \cong 5.822 \times 10^{-4} \cong 0$$

طبق فرمول PID با استفاده از ضرایب Kp ، Td و Ti و همچنین مقادیری که برای مشتق‌گیر و انتگرال‌گیر بدست آوردیم خروجی کنترلر را محاسبه می‌کنیم. در این PID از آرایش موازی استفاده می‌کنیم، پس خورجی آن به صورت زیر محاسبه می‌شود:

$$PID = K_P(e + \frac{1}{T_i}\int e\,dt + T_d\frac{de}{dt})$$

برای اثر اشباع کنترلر باید خروجی کنترلر را چک کنیم تا در صورتی که از مقدار max یا min سیستم خود عبور کرد مقدار آن را به ترتیب همان مقادیر max یا min قرار دهیم. توجه داریم که در این سیستم min همان مقدار max در جهت دیگر را دارد.

خروجی PID مقداری بین ۰ تا ۱۰۰ درصد می‌باشد. برای تبدیل این مقدار به PWM از counter period تایمر ۱ که مسئول ایجاد PWM است استفاده می‌کنیم. ورودی این تایمر بین ۰ تا ۲۰۰۰۰ است که به همان ۰ تا ۱۰/۲ ولت مپ می‌شود. به همین ترتیب با ضریب خروجی‌ها در ۲۰۰ این مقدار درست ایجاد می‌شود.

```
if (Err > 3.367){          //OUT OF porpotinal band
      direction = 1;
      TIM1 -> CCR1 = out_max*200;
}
else if (Err < -3.367){
      TIM1 -> CCR1 = 0;
}
else{ // inside the porpotional band
      integral += (float)Err/100.0;
      if ((Err < 0 && integral > 0) || (Err > 0 && integral < 0))
            integral = 0;
      if (integral > int_max) // integral windup
            integral = int_max;
      else if (integral < -int_max)
            integral = -int_max;
      else{
            derivative = ((float)DErr - (float)DLastErr)*100.0;
            PID = Kp*(Err + integral/Ti + derivative*Td);

            if (PID > out_max){// output saturation band
                  direction = 1;
                  TIM1 -> CCR1 = out_max*200;
            }
            else if (PID < -out_max){
                  direction = -1;
```

```
                TIM1 -> CCR1 = out_max*200;
        }
        else if (PID > 0){
                direction = 1;
                TIM1 -> CCR1 = PID*200;
        }
        else{
                direction = -1;
                TIM1 -> CCR1 = PID*200;
        }
    }
}
```

کد شماره ۴: کد الگوریتم PID با در نظر گرفتن integral windup و derivative overrun و output saturation

توجه داریم که ماکزیمم خروجی درایور برای PWM با Duty Cycle برابر با ۱۰۰ درصد، حدودا ۱۰/۲ ولت می‌باشد.به همین دلیل out_max تعریف شده درون کد را برابر ۶۰ قرار می‌دهیم. با این انجام این کار قرار گرفتن ولتاژ تغذیه موتور بین ۰ تا ۶ ولت (ولتاژ نامی موتور) را تضمین می‌کنیم.

برای تعیین جهت چرخش از علامت خروجی کنترلر استفاده می‌کنیم. درصورتی که مثبت باشد جهت را برابر ۱ (ساعتگرد) و در صورتی که علامت آن منفی باشد جهت را برابر ۱- (پادساعتگرد) قرار می‌دهیم.

```
if (direction == -1)
{
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_SET);
}
else
{
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);
}
```

کد شماره ۵: تعیین جهت چرخش موتور

در تست‌های عملی، مقادیر بدست آمده برای PID با شبیه‌سازی باعث نوسانی شدن و گاها ناپایداری سیستم می‌شود. این موضوع با توجه به اصول practical controller tuning tips موجود در اسلایدها (بند ۵) قابل توجیه است. ضریب کنترل ناپذیری این سیستم (P_u) بسیار کوچک می‌شود که نشانه کنترل‌پذیری بالای این سیستم است. این موضوع باعث می‌شود در تنظیم PID پارامترها بزرگ بدست آیند و از پاسخ به حالت مطلوب نزدیک نشود. در این حالت از تخمین استفاده می‌کنیم. به نظر می‌رسد که با کاهش K_P و کاهش سرعت پاسخ سیستم و همچنین افزایش P.B.، می‌توان این مشکل را برطرف کرد. ضریب بخش Porpotional را برابر ۲ قرار می‌دهیم.

$$P.B. = \frac{100}{K_p} = \frac{100}{2} = 50\%$$

$$T_i = 1$$

$$T_d = \frac{K_d}{K_p} \cong 0$$

کد شماره ۶، الگوریتم پیاده شده در ARM به طور کامل نمایش می‌دهد.

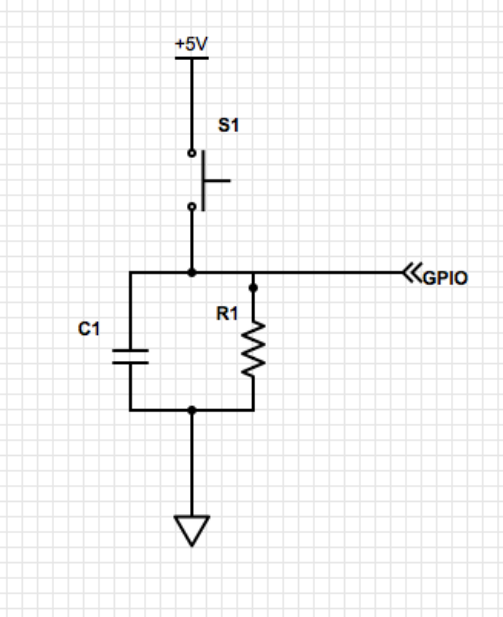
```
int out_max = 60;
int int_max = 21.78;
int cnt = 0;

int Err = 0;
int LastErr = 0;
int DErr = 0;
int DLastErr = 0;
float integral = 0;
float derivative = 0;
int SP = 0;
int PV = 0;
float PID = 0;

float Ti = 1;
float Kp = 2;
float Td = 0;
int flag = 0;

int direction = 0;
int Position = 0;
int LastPosition = 0;

int16_t val = 0;

HAL_TIM_Base_Start_IT(&htim3);
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
HAL_TIM_Encoder_Start_IT(&htim2,TIM_CHANNEL_1);
HAL_TIM_Encoder_Start_IT(&htim2,TIM_CHANNEL_2);
HAL_ADC_Start(&hadc1);
TIM1 -> CCR1 = 2000;

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
	if(htim == &htim2)
		flag = 1;
	if (htim == &htim3){
			Position = __HAL_TIM_GET_COUNTER(&htim2);
			val = HAL_ADC_GetValue(&hadc1);
			SP = (val-2000)/20.;
			if (flag == 0)
				PV = ((float)Position - (float)LastPosition)/(42.0) *
100;
			else if(Position >  LastPosition)
				PV = ((float)Position - 65535 -
(float)LastPosition)/(42.0) * 100;
			else
				PV = ((float)Position + 65535 -
(float)LastPosition)/(42.0) * 100;
			flag = 0;
			LastPosition = Position;

			Err = SP - PV;  // Reverse control action
			DErr = -PV;         // derivative overrun
```

```
                    //PID
                    if (Err > 25){            //OUT OF porpotinal band
                          direction = 1;
                          TIM1 -> CCR1 = out_max*200;
                    }
                    else if (Err < -25){
                          direction = -1;
                          TIM1 -> CCR1 = out_max*200;
                    }
                    else{ // inside the porpotional band
                          integral += (float)Err/100.0;
                                if ((Err < 0 && integral > 0) || (Err > 0 &&
integral < 0))
                                      integral = 0;
                          if (integral > int_max) // integral windup
                                integral = int_max;
                          else if (integral < -int_max)
                                integral = -int_max;
                          else{
                                derivative = ((float)DErr -
(float)DLastErr)*100.0;
                                PID = Kp*(Err + integral/Ti + derivative*Td);

                                if (PID > out_max){       // output saturation
band
                                      direction = 1;
                                      TIM1 -> CCR1 = out_max*200;
                                }
                                else if (PID < -out_max){
                                      direction = -1;
                                      TIM1 -> CCR1 = out_max*200;
                                }
                                else if (PID > 0){
                                      direction = 1;
                                      TIM1 -> CCR1 = PID*200;
                                }
                                else{
                                      direction = -1;
                                      TIM1 -> CCR1 = -PID*200;
                                }
                          }
                    }

                    if (direction == -1)
                    {
                           HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET);
                           HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_SET);
                    }
                    else
                    {
                           HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);
                           HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);
                    }
      }
}
```

کد شماره ۷: کد کامل پیاده‌سازی شده در stm32f103c8

با تعریف ورودی SP با پتانسیومتر، قابلیت تغییر سرعت برای سیستم نیز فراهم می‌باشد. برای در نظر گرفتن مود manual به سیستم یک کلید دیگر اضافه می‌کنیم. توجه داریم که برای bumpless transfer بین دو مود موجود، زمانی که سیستم در مود دستی قرار می‌گیرد، خطای انتگرال‌گیر همچنان وجود خواهد داشت. به این ترتیب که SP تنظیم شده در حالت اتوماتیک دریافت شده و خطای آن با خروجی کنونی سیستم مقایسه شده و خطای آن در انتگرال‌گیر انباشته می‌شود. کد شرط مربوط به این مورد در ادامه آورده شده. در این مدار چون امکان تنظیم دستی ولتاژ وجود ندارد، در حالت manual فقط خطا توسط انتگرال‌گیر دنبال می‌شود.

برای debounce کردن این کلید از یک مدار RC با ثابت زمانی تقریبا میلی‌ثانیه استفاده می‌کنیم.



شکل ۳۶: مدار RC برای debounce کلیدهای مدار

```
if (Auto_Man == 1){
        PID CODE
}
else
{
       if(htim == &htim2)
              flag = 1;
       if (htim == &htim3){
                     Position = __HAL_TIM_GET_COUNTER(&htim2);
                     val = HAL_ADC_GetValue(&hadc1);
                     SP = (val-2000)/20.;
                     if (flag == 0)
                            PV = ((float)Position - (float)LastPosition)/(42.0) *
100;
                     else if(Position >  LastPosition)
                            PV = ((float)Position - 65535 -
(float)LastPosition)/(42.0) * 100;
                     else
                            PV = ((float)Position + 65535 -
(float)LastPosition)/(42.0) * 100;
                     flag = 0;
                     LastPosition = Position;
                     Err = SP - PV;  // Reverse control action
```

```
                    integral += (float)Err/100.0;
        }
}
```

کد شماره ۸: شرط اضافه شده برای تشخیص مود کاری

۵. کنترل موقعیت

برای کنترل موقعیت، همان سیستم قبلی و پایه‌های مشخص شده را استفاده می‌کنیم با این تفاوت که PV الآن موقعیت موتور می‌باشد و PID برای این کنترل باید باز تنظیم شود.

۱. تغییر مد کاری

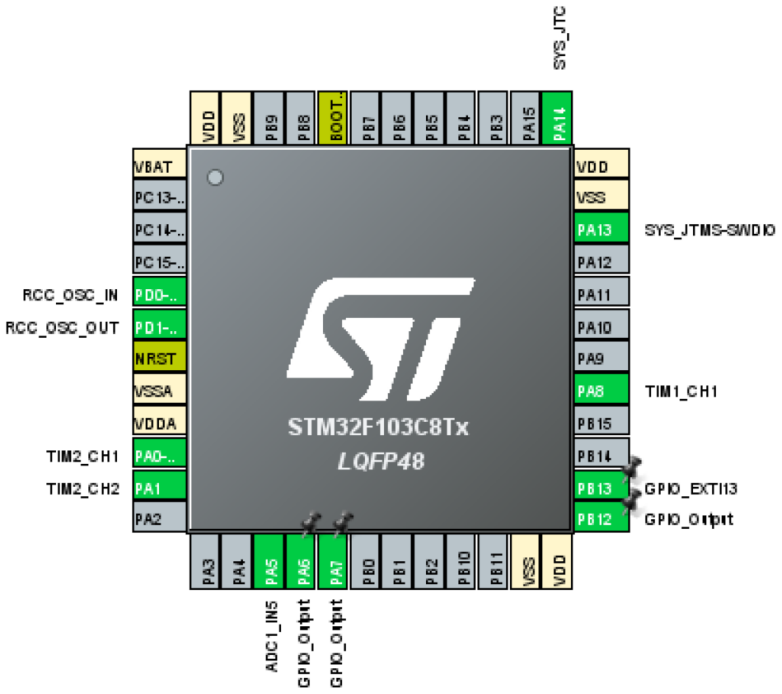
برای تغییر مد از یک کلید استفاده می‌کنیم (با یک مدار RC مشابه debounce می‌شود). این کلید به پایه PB13 متصل می‌شود و روی حالت EXTI (External Intrrupt) تنظیم می‌شود. به این ترتیب با برقراری ارتباط کلید، تابع زیر اجرا می‌شود. متغییر mode از نوع GPIO_PinState تعریف می‌شود.

شکل ۳۷: فعال‌سازی وقفه‌های خطوط ۱۰ تا ۱۵

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
        UNUSED(GPIO_Pin);
        if (GPIO_Pin == GPIO_PIN_13)
        {
                if (mode == 0){
                        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, GPIO_PIN_SET);
                        mode = 1;
                }
                else{
                        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, GPIO_PIN_RESET);
                        mode = 0;
                }
        }
}
```

کد شماره ۹: تابع اینتراپت کلید تغییر مد

اگر کلید زده شود، مد از حالت ۱ به ۰ یا از ۰ به ۱ می‌رود. زمانی که mode برابر ۱ باشد، LED مربوط به تغییر وضعیت روشن می‌شود که نشان‌دهنده مد کنترل موقعیت می‌باشد. زمانی که مد برابر ۰ باشد، LED خاموش و نشان‌دهنده مد کنترل سرعت است.



شکل ۳۸: پایه‌های اختصاص فعال شده در میکرو

۲. الگوریتم PID کنترل موقعیت

در این حالت خطای حالت ماندگار اهمیت زیادی دارد. به همین دلیل ضریب K_i را افزایش می‌دهیم (با کاهش T_i این کار را انجام می‌دهیم). همچنین بخش مشتق‌گیر را حذف می‌کنیم و یک کنترلر PI قرار می‌دهیم. برای کنترل دقیق‌تر روی موقعیت از تایمر ۴ در مود انکودر با تنظیمات مشابه تایمر ۲ استفاده می‌کنیم. با این تفاوت که رجیستر ARR آن تا ۱۱۵۰ (یک دور چرخش موتور) بالا می‌رود.

```
if (htim == &htim3){
	Position_pos = __HAL_TIM_GET_COUNTER(&htim4);
	PV_pos = (float)Position_pos/1150*100.0;
	val_pos = HAL_ADC_GetValue(&hadc1);
	SP_pos = val_pos/40.;
	Err_pos = SP_pos - PV_pos;  // Reverse control action

	 //PID
	if (Err <3 && Err>-3)
		TIM1 -> CCR1 = 0;
	else if (Err_pos > 7){			//OUT OF porpotinal band
		direction = 1;
		TIM1 -> CCR1 = 7*200;
	}
	else if (Err < -7){
		direction = -1;
		TIM1 -> CCR1 = 7*200;
	}
	else{ // inside the porpotional band
		integral_pos += (float)Err_pos/100.0;
		if ((Err_pos < 0 && integral_pos > 0) || (Err_pos > 0 && integral_pos
< 0))
		integral_pos = 0;
		if (integral_pos > int_max) // integral windup
			integral_pos = int_max;
```

```
            else if (integral_pos < -int_max)
                  integral_pos = -int_max;
            else{
                  PID_pos = Kp_pos*(Err_pos + integral_pos/Ti_pos);

                  if (PID_pos > 7){          // output saturation band
                        direction = 1;
                        TIM1 -> CCR1 = 7*200;
                  }
                  else if (PID_pos < -7){
                        direction = -1;
                        TIM1 -> CCR1 = 7*200;
                  }
                  else if (PID_pos > 0){
                        direction = 1;
                        TIM1 -> CCR1 = PID_pos*200;
                  }
                  else{
                        direction = -1;
                        TIM1 -> CCR1 = -PID_pos*200;
                  }
            }
      }

      if (direction == -1)
      {
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_SET);
      }
      else
      {
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);
      }
}
```

کد شماره ۱۰: کنترل موقعیت موتور

۶. نکات نهایی

در انتها، ذکر چند نکته برای درک بهتر موانع پروژه نحوه رفع آن‌ها حائز اهمیت می‌باشد.

- با توجه به ثابت زمانی موتور، می‌توان به‌جای PID از کنترلرهای PI استفاده نمود. زیرا سیستم نیازی به کاهش زمان پاسخ ندارد.
- زمان نمونه‌برداری با یک قاعده سرانگشتی ۱۰ میلی‌ثانیه انتخاب شده. با توجه به ماکزیمم سرعت موتور حداکثر حرکت دورانی در یک زمان نمونه‌برداری:

$$Max\ rotational\ movement = Max\ speed \times T_s$$
$$= 150\ RPM \times \frac{1\ min}{60\ sec} \times \frac{1\ sec}{100\ ms} = 0.025 = 9°$$

به این ترتیب برای دقت بیشتر (در صورت توانایی انکودر در اندازه‌گیری) می‌توان این زمان را ۱ میلی‌ثانیه هم کاهش داد.

- انکودر در حالت ۱ کاناله دارای ۴ پالس بر دور بود که با استفاده از دو کانال این دقت به ۸ پالس بر دور افزایش یافت. حتی با استفاده از دو کانال، این دقت کمتر از دقت یک انکودر نوری افزایشی معمولی (بیش از ۳۰ پالس) است. پس استفاده از انکودرهای نوری (در صورت امکان) دقت را به شدت افزایش می‌دهد.