به نام او

گزارش‌کار پروژه کنترل صنعتی

اعضای گروه: امین رزاقی (۹۵۲۳۰۴۶) – هلیا گهرباونگ (۹۵۲۳۱۰۶)

شبیه‌سازی

مرحله اول: تنظیم سرعت موتور DC طبق ساختار مرسوم

در این بخش میخواهیم سرعت یک موتور DC را در متلب با استفاده از سیگنال PWM کنترل کنیم. به همین منظور حلقه کنترلی مربوطه را در بخش Simulink متلب شبیه‌سازی کردیم. در این پروژه چون جهت چرخش موتور تغییر نمیکند، از یک کلید قدرت به عنوان درایور استفاده کردیم و استفاده‌ای از H-Bridge نشد. همچنین کنترل‌کننده PID بود که ضرایب آن را با استفاده از بخش PIDTuning بدست آوردیم که در ادامه نشان داده می‌شود.

روشی که با استفاده از آن تابع تبدیل سیستم بدست آمد two-point بود. طبق این روش با دادن یک Set Point مشخص و دیدن خروجی برنامه، در نمودار جایی که مقدار ۶۳٪ مقدار نهایی دیده میشـــد، t_1 و جایی که ۲۸٪ آن دیده میشـــد را t_2 نامیدیم و طبق فرمول های زیر، مقدار τ و t_0 بدست آوردیم.

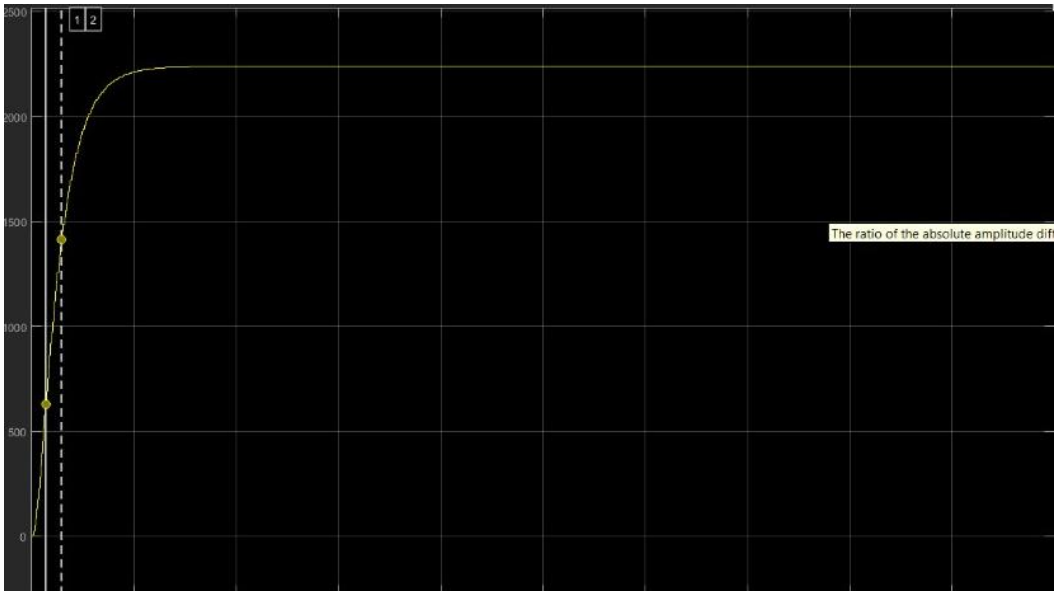
$$\tau = 1.5\,(t_1 - t_2)$$

$$t_0 = t_1 - \tau$$

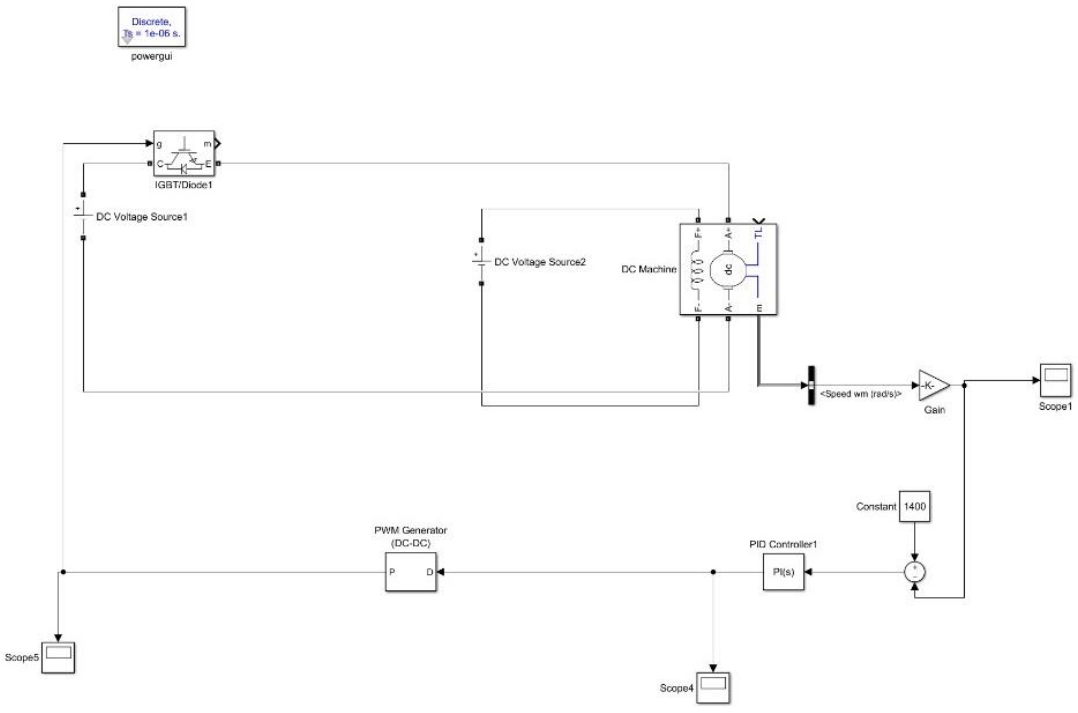
مقدار $t_1 = 1.049$ و $t_2 = 0.355$ مطابق نمودار بدست آمد. از این مقادیر داریم:

$\tau = 1.041$

$t_0 = 0.008$



حال با این مقادیر تابع تبدیل را بدست آورده و با PIDTuning ضرایب کنترل‌کننده مناسب را پیدا کردیم و مدار زیر را در متلب شبیه‌سازی کردیم.



شکل مدار اول

در نتیجه ضرایب مورد نظر بدست آمد: Kp = 1, Ki =0.02, Kd = 0

استفاده از PWM

حال از بلوک پالسی PWM Generator برای آتش کردن ماسفت استفاده میکنیم. در اصل PWM موجی مربعی است که در برخی زمان ها ۰ و برخی زمان ها ۱ است و این ۰ و ۱ شدن ها با فرکانس مرتبی تکرار می شود. PWM مانند سایر امواج، دارای دامنه ی Amplitude، دور تناوب یا Period و فرکانس اســت. عبارت دیگری که در PWM مورد اســتفاده قرار می گیرد Duty Cycle است. دیوتی سایکل مدت زمان ۱ بودن به مدت زمان کل پریود در هر سیکل موج است که معمولا بر حسب درصد (٪) نمایش داده می شـود. به فرض مثال اگر Duty Cycle یک موج PWM برابر با ٪۴۰ باشـد بدان معنی اسـت که در هر سیکل ٪۴۰ ولتاژ برابر VCC و در ٪۶۰ اوقات ولتاژ برابر ۰ است. همانگونه که می دانید در چنین حالتی ولتاژ موثر یا Vrms برابر با ٪۴۰ VCC خواهد بود. به فرض مثال شما اگر با یک میکرو با تغذیه ۵ V، موج PWM با دیوتی سایکل ٪۵۰ ایجاد نمایید ولتاژ RMS شما برابر ٪۵۰ VCC یا به عبارتی ۲,۵ ولت خواهد بود.

با اعمال موج های PWM بالا به یک ماســفت می خواهیم توان یک موتور DC را بین ۰ تا ٪۱۰۰ تغییر دهیم که این تغییر توان تاثیر مستقیمی بر سرعت موتور خواهد گذاشت.

استفاده از IGBT

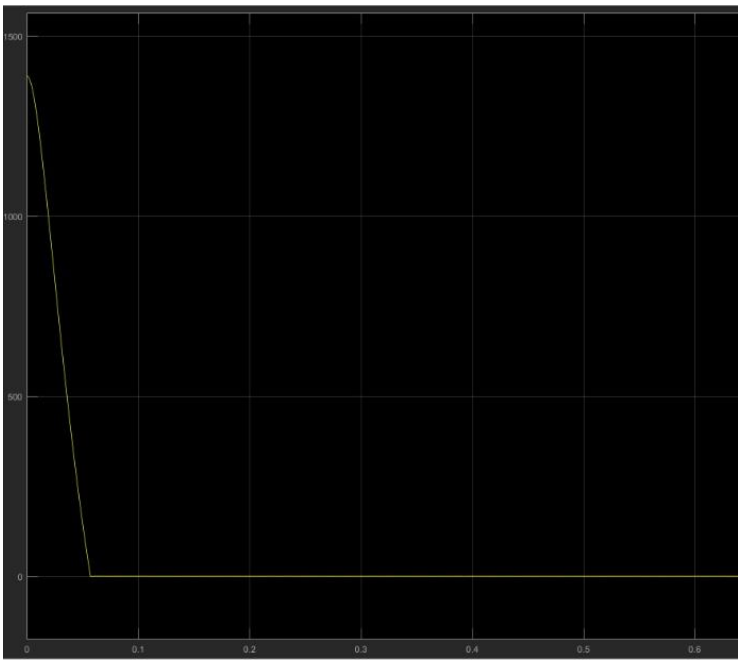
IGBT که ترکیبی از BJT و ماسفت است، با ولتاژ یا همان سیگنال gate کنترل میشود. استفاده اصلی آن محافظت از موتور در برابر جریان راه‌اندازی و اتصـال کوتاه اسـت. این مدل به عنوان سـوئیچ به کار رفته و با خاموش و روشـن کردن سـریع منبع امکان استفاده از پالس PWM برای کنترل را به ما می‌دهد.

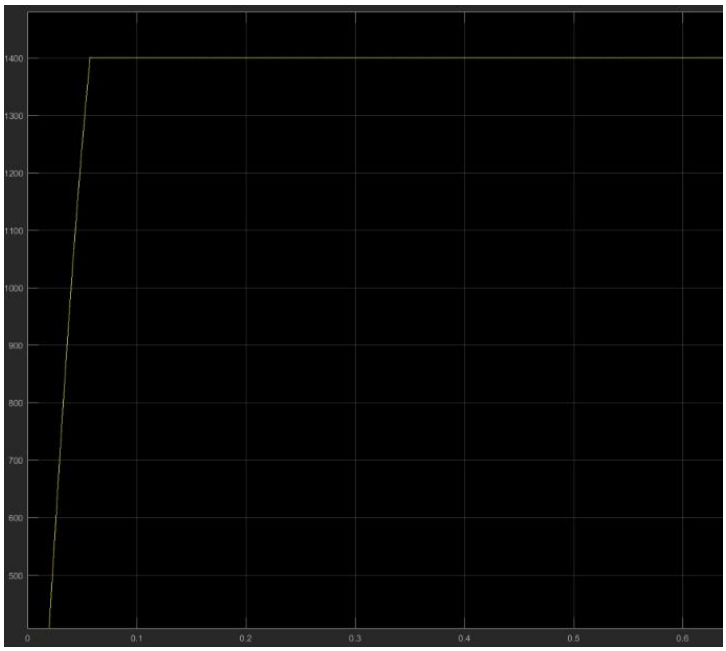
چالش‌ها

به دلیل سوئیچینگ های زیاد، ریپل بالایی داشتیم، به همین دلیل از مدار RLC با مقادیر R = 1, C = 1-e6, L = 1-e3 در مدار قرار دادیم.

همچنین برای به دست آوردن اطلاعات موتور از BusSelector استفاده کردیم که این المان ۵ داده میدهد که ما فقط سرعت و ولتاژ خروجی را میخواهیم.

خروجی‌ها



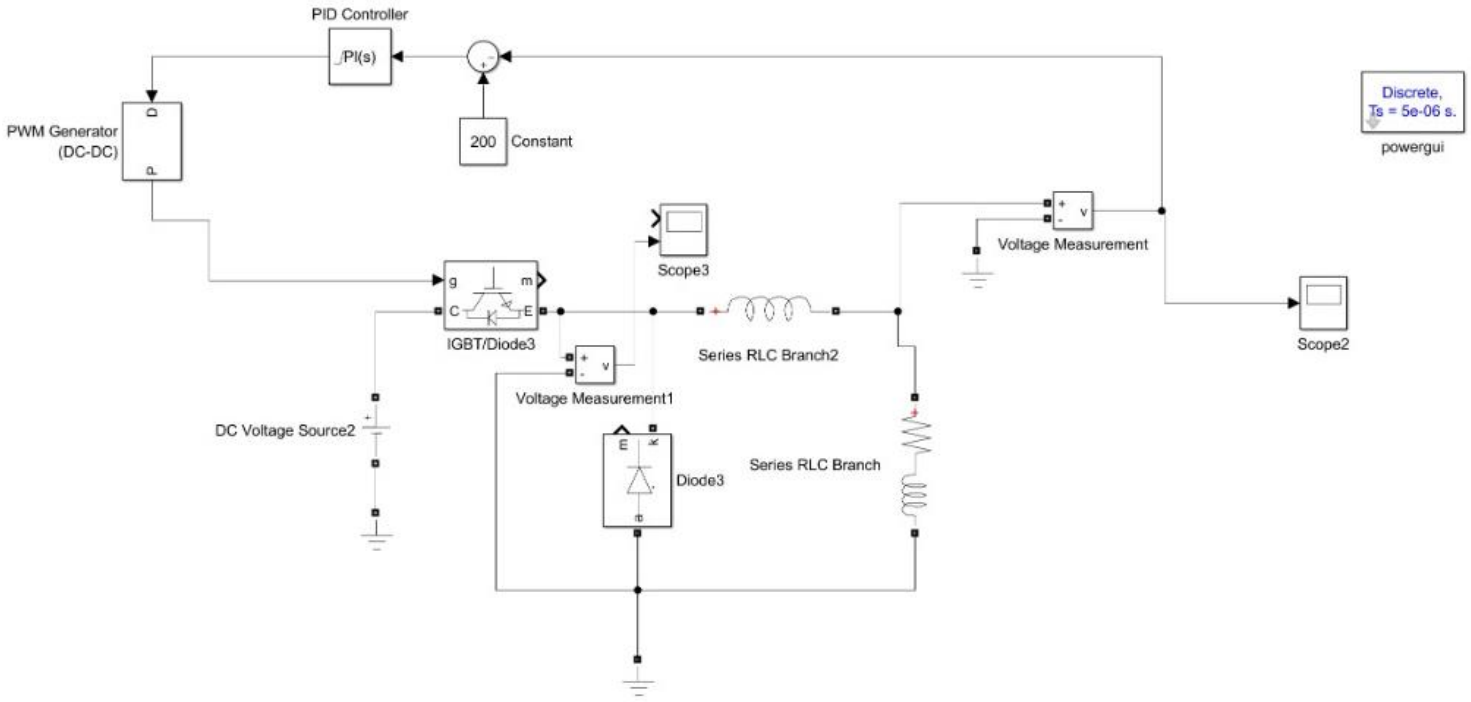


لازم به ذکر است که موتور استفاده شده در سیمولینک با موتور واقعی ما در مدار فیزیکی متفاوت بود. همچنین ما در این بخش، موفق شدیم سرعت را کنترل کنیم، که در ادامه در بخش کنترل آبشاری یا Cascade، حلقه‌ی بیرونی خواهد بود. در مرحله بعد میخواهیم ولتاژ را کنترل کنیم.

مرحله دوم: طراحی مبدل DC-DC

همانطور که میدانیم کنترل سرعت موتور با تنظیم ولتاژ پایانه های آن انجام میشود. در این پروژه ما از طریق مبدل Buck ولتاژ ورودی که از منبع ولتاژ ۳۱۰ که از یک سو سازی برق شهر تامین شده به سطح ولتاژ مطلوب تبدیل میکنیم. مقادیر RLC لازم برای مدار باک از قبل به ما داده شده بود. اما امکان دیگر ا ستفاده از روابط موجود در درس الکترونیک صنعتی بود. خروجی این مبدل نیز مانند duty cycle در صدی خواهد بود که از نسبت ولتاژ خروجی به ولتاژ ورودی بد ست می آید. برای بد ست آوردن مقادیر PID این بخش از سعی و خطا استفاده کردیم.

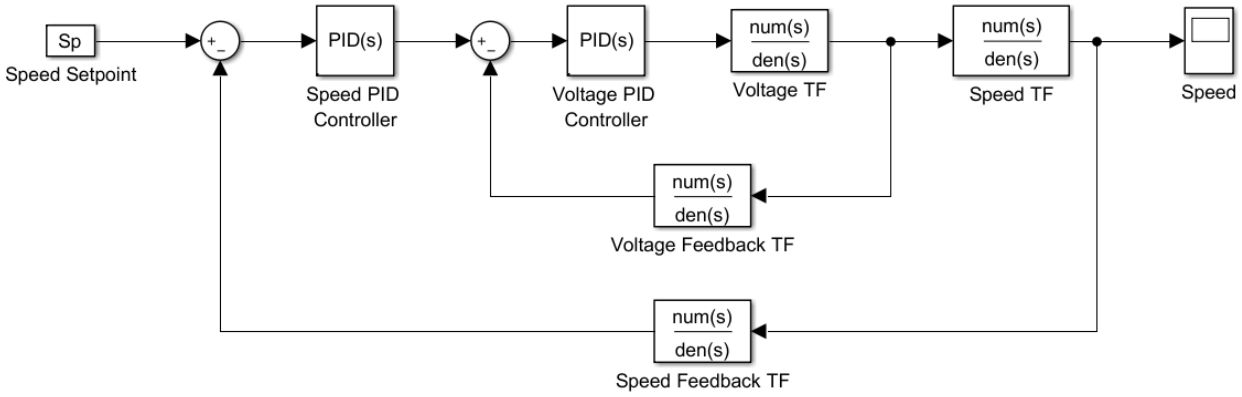
مدار نهایی:



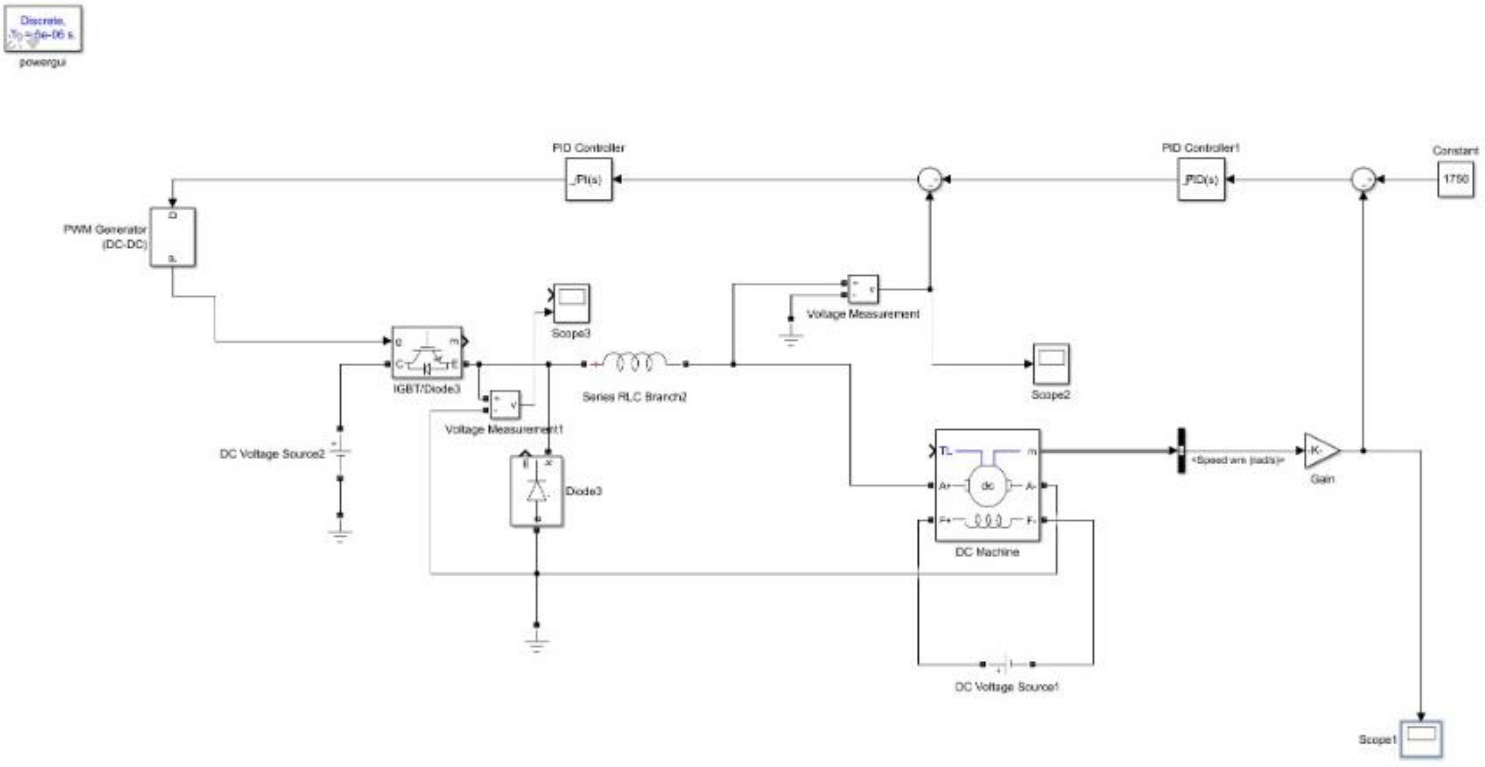
شکل مدار دوم

مرحله سوم: کنترل موتور DC با استفاده از مبدل طراحی شده

در این مرحله میخواهیم بخش اول و دوم را ترکیب کنیم و به کنترل کننده نهایی موتور DC برسیم. مطابق شکل زیر دو مدار ترکیب میشوند. به این ساختار Cascade گفته میشد. مزیت این ساختار این است که اگر در ولتاژ خطا یا disturbance داشته باشیم، دیگر لازم نیست نتیجه آن را در سرعت موتور ببینیم، بلکه در همان حلقه کنترلی ولتاژ این خطا جبران و کنترل می شود و دیگر در سرعت و دور موتور خروجی اثری ناشی از ولتاژ اشتباه نمیبینیم.



مدار نهایی:



شکل مدار سوم

مرحله چهارم: پیاده‌سازی کنترل‌کننده سرعت به صورت سخت‌افزاری

در این مرحله میخواهیم الگورتیم کنترلی بخش یک را بصورت عملی روی میکروپروسسور ARM پیاده سازی کنیم. در گام اول تایمر های لازم را تنظیم کردیم. برای این پروژه دو تایمر نیاز هست، یکی برای PWM و دیگر برای Encoder. دور هر سیکل بر اساس دانسته ها ۲ میلی ثانیه است. Timer 1 را به PWM و Timer 2 را به Encoder اختصاص دادیم.

در بخش اصلی (main) ابتدا فلگ Interrupt های Update را صفر میکنیم. چون در حالت عادی فعال هستند نمیخواهیم که در زمانی غیر از زمان مشخص شده‌ی خودمان، Interrupt داشته باشیم.

در مرحله بعد میخواهیم Interrupt های بخش Update را فعال کنیم. در ادامه تایمر PWM از Channel م شخص شده و همچنین Encoder را از دو Channel متفاوت را راه اندازی میکنیم.

در این بخش کد PIDCalculate را هم زدیم که با مقادیر کنترلی Kp = 8000 و Ki = 0.5 یک کنترلر PI تعیین کردیم که این تابع با استفاده از SetPoint و سرعت انکودر، Error را بدست آورده و بصورت ساده، یعنی حساب کردن ITerm و PTerm و جمع آنها (چون کنترلر موازی است)، سرعت آن را کنترل میکند. ابتدا از تایمر ۲ مقدار پالس را میخوانیم و داخل متغیر Pulse ذخیره میکنیم. سپس مقدار سرعت را از رابطه زیر بدست آوردیم:

■ $Speed = 65535 \times Revolve + Pulse - Ppulse$

در ادامه توضیح داده میشود که Revolve چیست. همچنین Ppulse پالس سیکل قبلی است. حال Ppulse این سیکل را برابر چالس همین سیکل قرار میدهیم متغیر Revolve را برابر صفر قرار میدهیم.

متغیر MyError که از اختلاف ست پوینت و سرعت بدست می آید را در مراحل بعد در ITerm و PTerm استفاده کردیم.برای جلوگیر از Wind up همچنین برای متغیر ITerm محدوده گذاشتیم که از نصف بیشترین مقدار ممکن خروجی همیشه کمتر با شد. برای خروجی نیز چون نمیخواهیم از ۴۸۰۰۰ بالاتر برود محدوده قرار میدهیم که باعث نشود موتور برعکس بچرخد. در انتهای این تابع، مقدار خروجی کنترل کننده را روی چنل ۱ تایمر ۱ ست میکنیم.

از تابع PIDCalculate در تابع CallBack استفاده کردیم. این تابع برای زمانی است که Interrupt داریم. در این تابع چک میکنیم که Interrupt در تایمر ۱ اتفاق افتاده یا تایمر ۲. اگر در تایمر ۱ یا همان PWM سرریز داشته باشیم، Interrupt فعال شده و تابع PIDCalculate صدا زده میشود. اما اگر تایمر ۲ یا انکودر باشد، متغیر Revolve را برابر ۱ قرار میدهیم.

اما متغیر Revolve چیست؟ در ابتدای سیکل، متغیر Pulse صفر خواهد بود. بنابراین سرعت یا همان Speed در فرمول ■ با کم شدن از پالس قبلی منفی خواهد بود و باعث میشود که سرعت منفی داشته باشیم. از آنجایی که در این پروژه موتور صرفا در یک جهت میچرخد، نیازی به سرعت منفی نداریم و به همین منظور در ابتدای هر سیکل متغیر Revolve را برابر یک قرار میدهیم تا برای محاسبه‌ی سرعت بجای Pulse از یک ۶۵۵۳۵ کم شود و سرعت مطلوب را به ما بدهد.

مرحله اختیاری: پیاده‌سازی کنترل‌کننده موقعیت

در این حالت نیز دوباره سرعت را کنترل میکنیم، اما با این تفاوت که حلقه داخلی حلقه کنترلی سرعت و حلقه خارجی، حلقه‌ی کنترل موقعیت اســت. همچنین متغیر موقعیت ما نیز همان پالس در مرحله قبل و متغیر ســرعت همان Speed در مرحله قبل اســت. ابتدا کنترل کننده را برای ســـرعت طراحی کردیم. ســـپس همان کنترل کننده را داخل حلقه به عنوان حلقه ی داخلی میگذاریم و SetPoint این حلقه نیز همان خروجی کنترل کننده موقعیت اســـت. از نکاتی که باید در کد به آن توجه میکردیم، مثبت و منفی بودن خطای موقعیت بود؛ که مطابق با این خطا یا Error جهت گردش موتور تغییر میکند. (ساعت‌گرد/پادساعت‌گرد)

همچنین در این بخش نیز به Wind Up توجه میشود و برای بخش انتگرال گیر محدوده مشخص کردیم. در کل این بخش مشابه با بخش ۳ نوشته شد.