دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

عنوان

طراحی و پیاده سازی کنترل کننده موتور DC با استفاده از یک میکروکنترلر ARM

درس

کنترل صنعتی

نام استاد

جناب آقای دکتر افشار

نگارش

مارال مرداد

محمد برآبادی

سجاد قدیری

محیا حقگو

قسمت اول:

مشخصات موتور به صورت زیر به دست آمده است:

فایل دیتاشیت درایور

Armature resistance(Ra) = 0.78

Armature inductance(La) = 0.0160

Damping (Viscous) Friction(B) = 0.02

Motor inertia(J) = 5

Input voltage(Va) = 240

Load torque(TL) = 30

Back EMF = 1.4

برای به دست آوردن معادلات موتور به صورت زیر می‌نویسیم:

می‌دانیم گشتاور ایجادشده توسط موتور ضریبی از جریان موتور است:

$$T = K_t i$$

EMF موتور هم ضریبی از سرعت زاویه ای موتور می‌باشد:

$$e = K_e \dot{\theta}$$

حال با استفاده از قانون دوم نیوتن معادلات موتور به صورت زیر است:

$$Ki = J\ddot{\theta} + b\dot{\theta}$$

$$L\frac{di}{dt} + Ri = V - K\dot{\theta}$$

با ساده‌سازی معادلات بالا تابع تبدیل به صورت زیر به دست می‌آید:

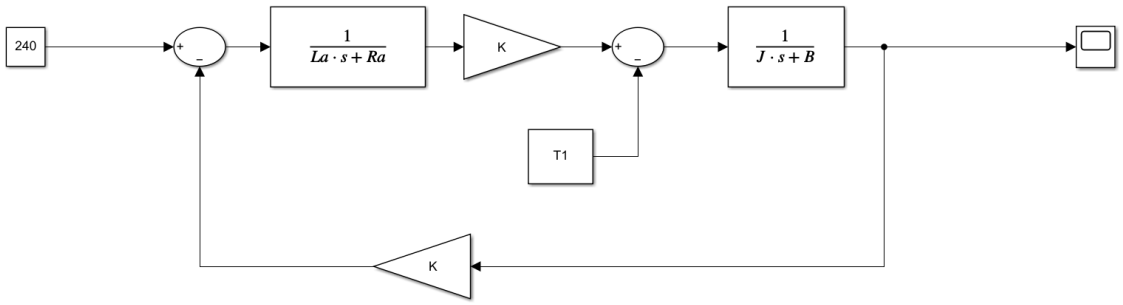
$$\frac{\dot{\theta}}{V} = \frac{K}{LJs^2 + (JR + bL)s + (bR + K^2)}$$

که با جاگذاری پارامترها تابع تبدیل به صورت زیر می‌شود:

$$G = \frac{3 * S + 1.7}{0.7 * S^2 + 4.272 * S + 2.38}$$

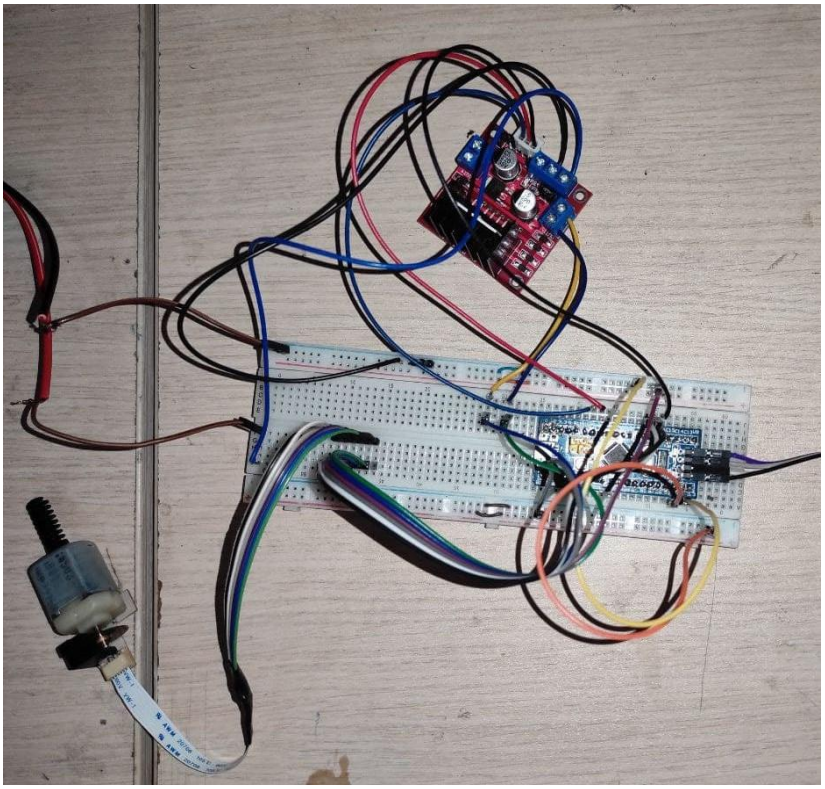
مدل موتور در محیط سیمولینک به صورت زیر بسته شده است:

(مربوط به فایل سیمولینک Motor_Model)



قسمت دوم:

مدار حلقه باز به صورت زیر بسته شده است:

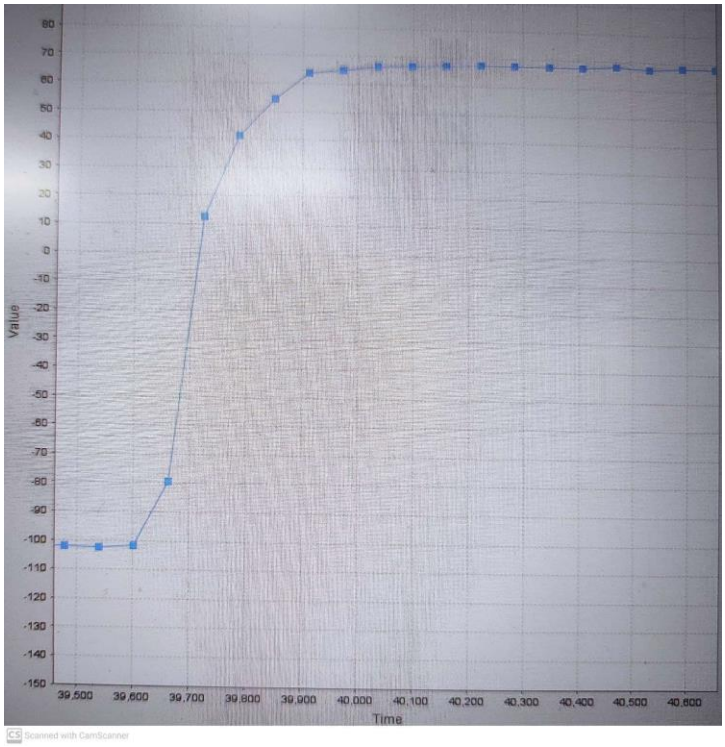


با توجه به کد زده شده برای موتور بازه ی تغییرات ورودی و خروجی به صورت زیر است:

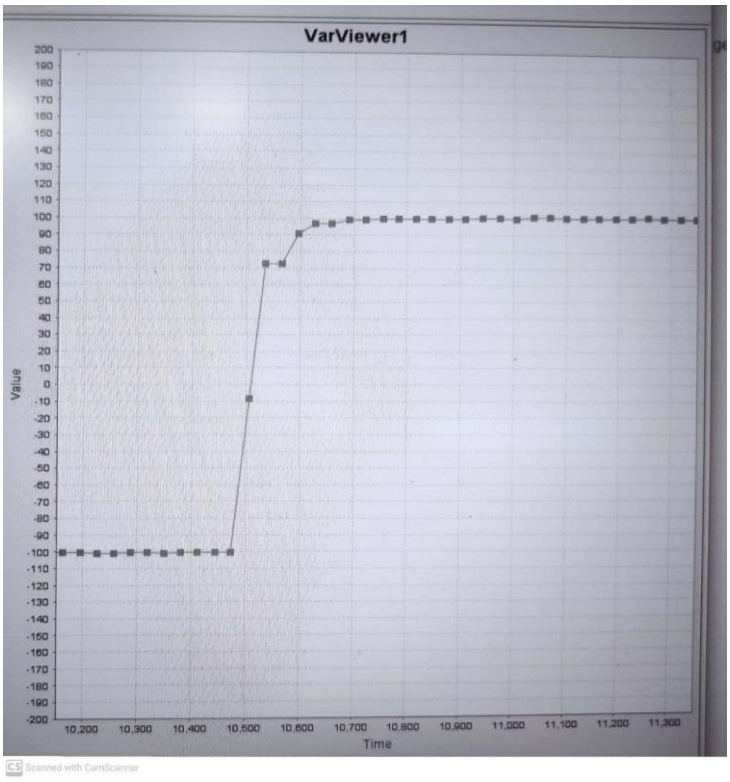
$$voltage = [-80\,,80]$$

$$speed = [-100{,}100]$$

برای مشاهده تغییرات خروجی دو ولتاژ ۸۰ و ۵۰ ولت را به موتور داده ایم. پاسخ ها به صورت زیر است:



$$voltage = 50$$



$$voltage = 80$$

با استفاده از Zigler-Nichols first method (QDR) و روش Two-point می‌دانیم:

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

$$t_0 = t_1 - \tau$$

که t_1 و t_2 به ترتیب زمان‌هایی هستند که پاسخ به ۶۳,۲٪ و ۲۸,۲٪ مقدار نهایی خود می‌رسد.

با توجه به پاسخ مدار حلقه باز مقدارهای t_1 و t_2 به صورت زیر می‌باشند.

$$t_1 = 43\ ms$$

$$t_2 = 18\ ms$$

همچنین برای محاسبه ی k داریم:

$$k = \frac{\Delta c}{\Delta m}$$

که Δc و Δm به ترتیب تغییرات خروجی و تغییرات فرمان کنترلی هستند. در این مدار رنج تغییرات خروجی که سرعت می‌باشد، $(100و100-)$ و رنج تغییرات فرمان کنترلی $(80$ و $80-)$ می‌باشد.

از روابط بالا داریم:

$$\tau = 37.5\ ms$$

$$t_0 = 5.5\ ms$$

$$k = 0.8$$

تابع تبدیل FOPDT به صورت زیر می‌باشد:

$$G = \frac{0.8\ e^{-0.0055s}}{1 + 0.0375s}$$

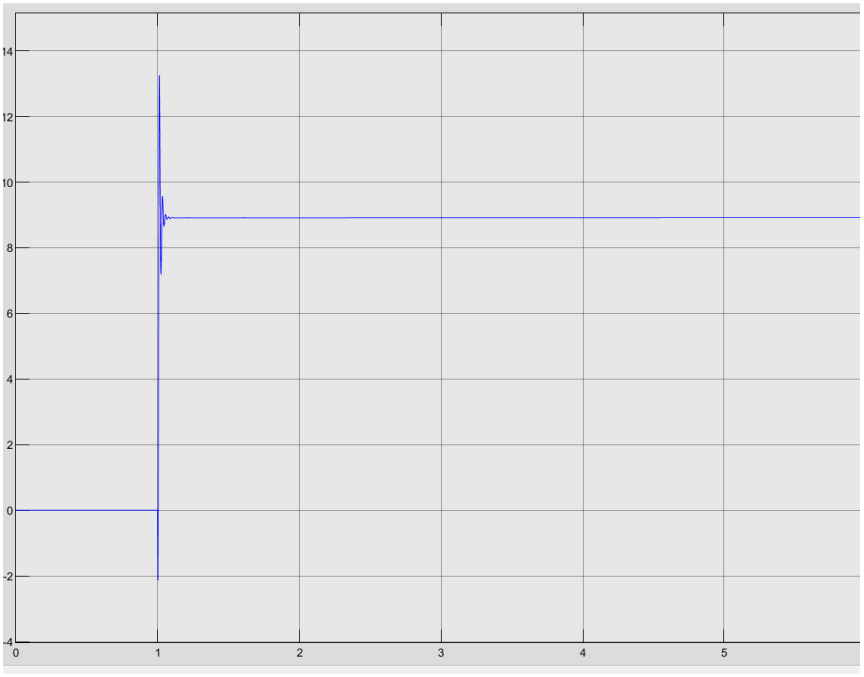
با داشتن τ و t_0 و k می‌توانیم ضرایب کنترلر PID موازی را به دست آوریم که به صورت زیر می‌باشد:

$$k_p = 10.22$$

$$T_I = 13.5\ ms$$

$$T_D = 2.2\ ms$$

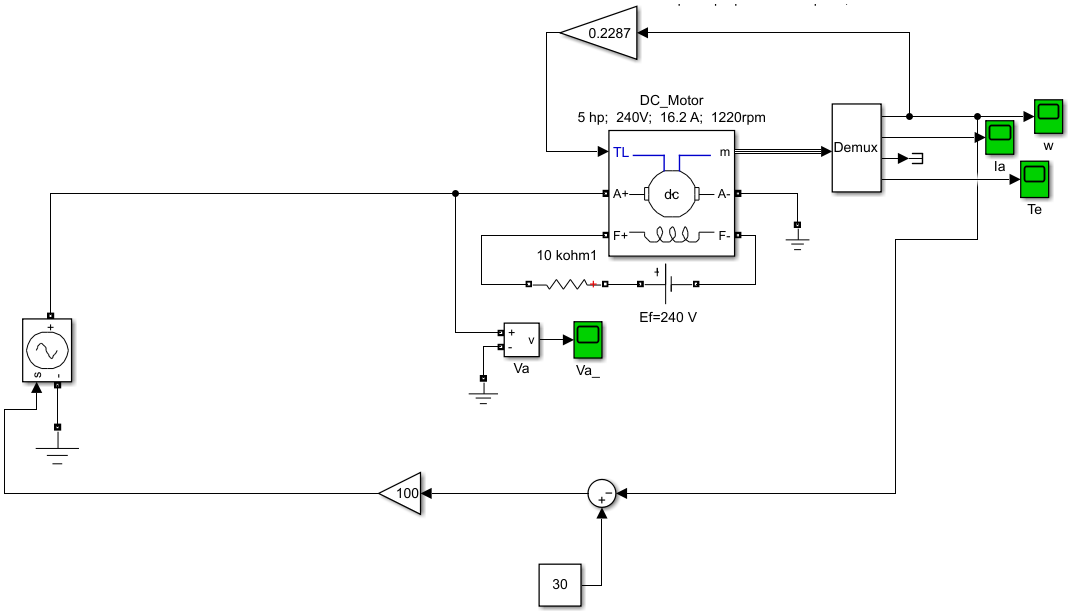
کنترلر به دست آمده را در مدار قرار می‌دهیم. پاسخ سیستم حلقه بسته به صورت زیر است:



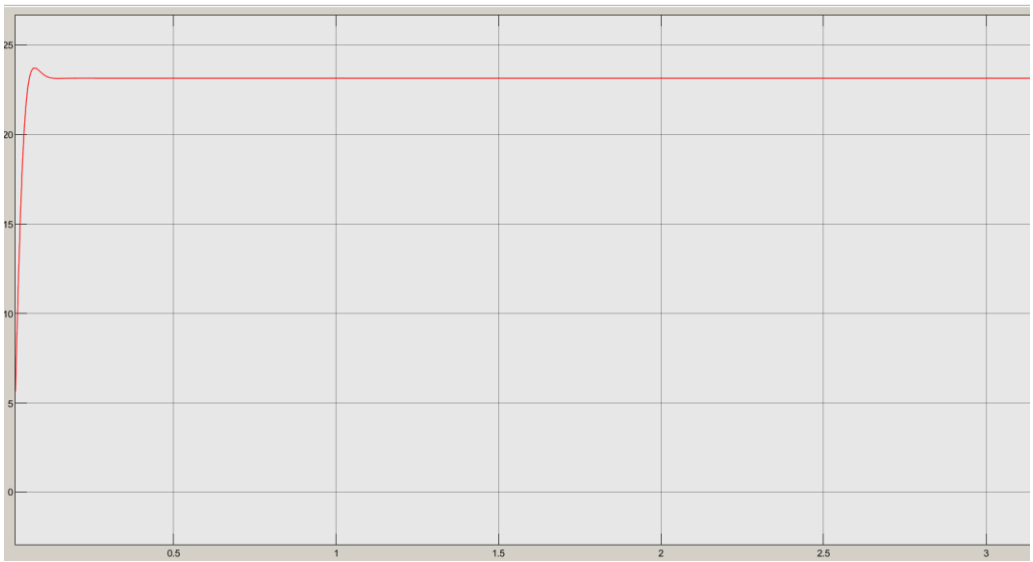
برای مدار حلقه بسته اگر پاسخ پله سیستم نوسانی کامل شود، با استفاده از روش Continuous cycling می‌توان ضرایب Ku و Pu را به دست آورد و از روی این ضرایب کنترلر طراحی کرد. با توجه به معادله مشخصه تابع تبدیل موتور DC دو قطب دارد. طبق نمودار بودی فاز سیستم با دو قطب هیچوقت به منفی ۱۸۰ درجه نمی‌رسد. همچنین طبق نمودار نایکوئیست هیچگاه به حالت نوسانی کامل نخواهد رسید و ضرایب Ku وPu به دست نخواهد آمد.

مدار حلقه بسته به صورت زیر بسته شده است:

(مربوط به فایل سیمولینک close-loop)



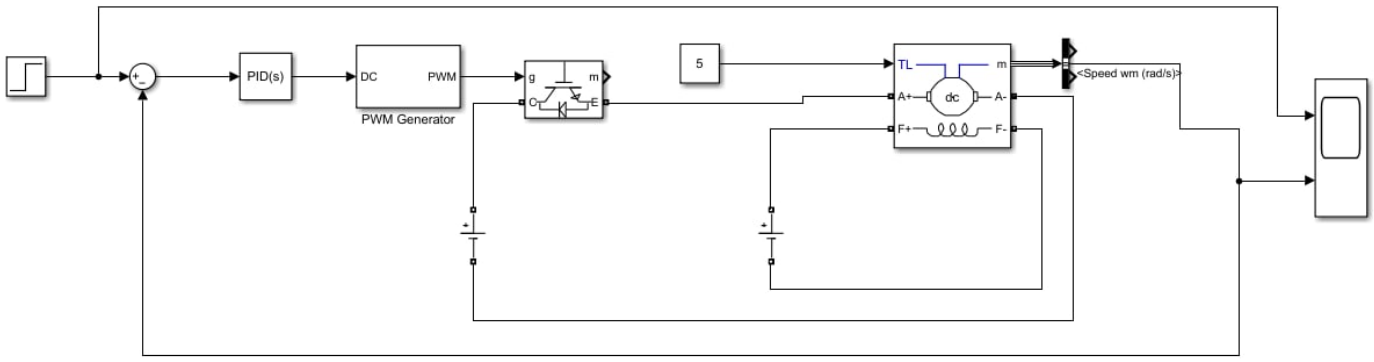
پاسخ سیستم حلقه بسته سیستم به صورت است:



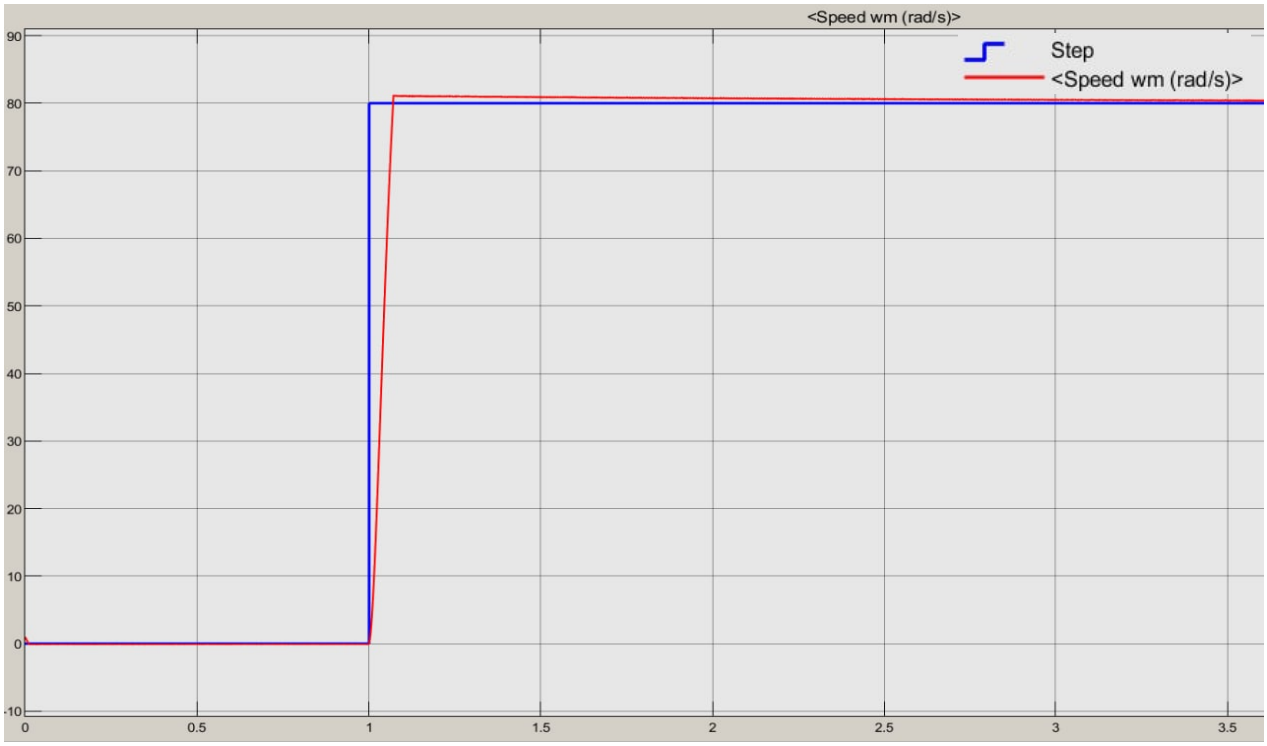
قسمت سوم :

(مربوط به فایل سیمولینک PWM_Control)

در این بخش با استفاده از PWM و طراحی یک کنترل کننده PI سیستم را کنترل می‌کنیم. مدار را مطابق شکل زیر در سیمولینک می‌بندیم.

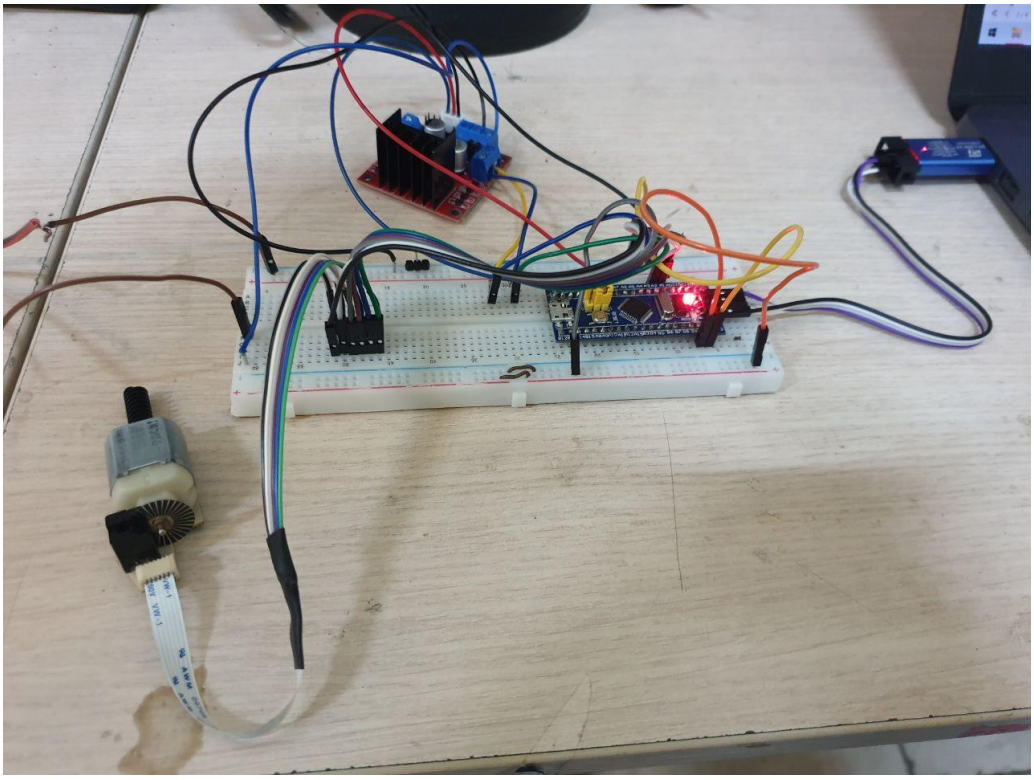


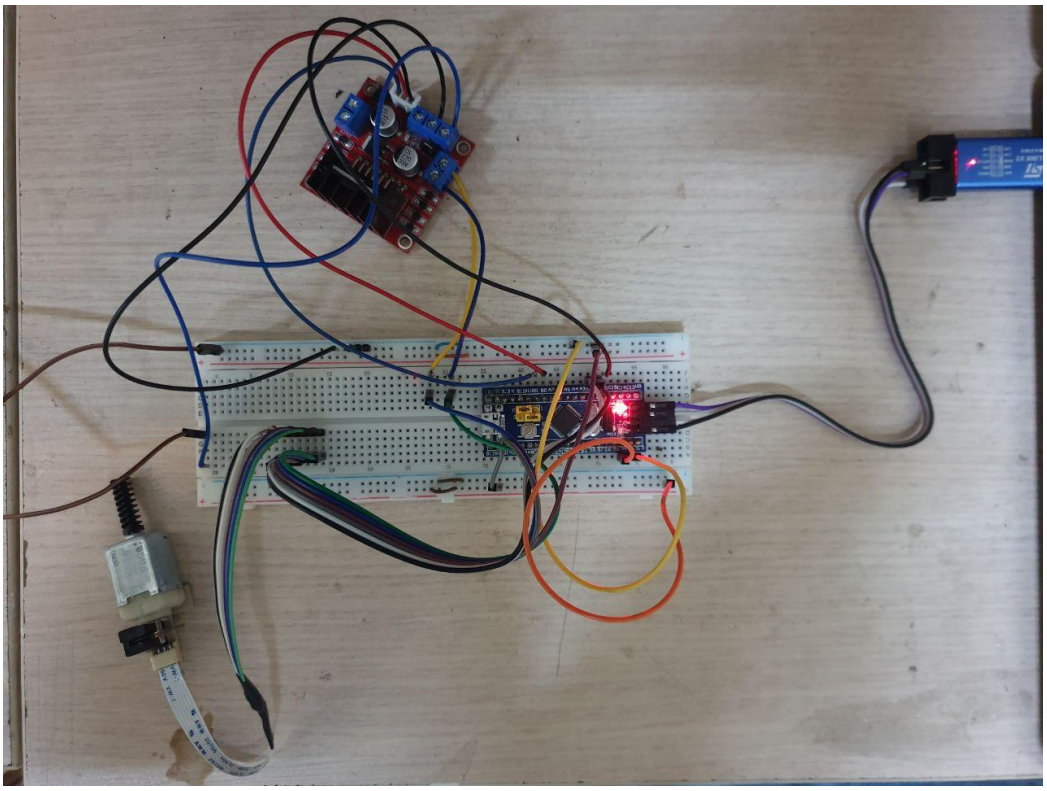
Set point را برابر ۸۰ قرار دادیم. همچنین ضرایب PI به ترتیب $k_I = 4.5$, $k_p = 11$ می‌باشد.



بخش عملی:

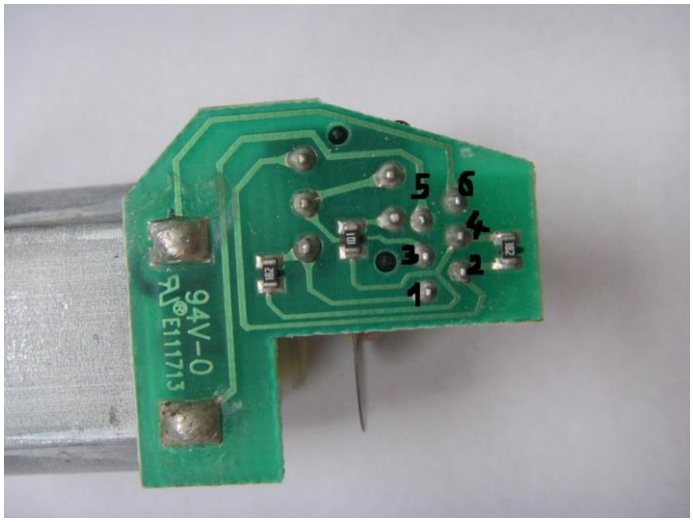
در این قسمت مدار شکل زیر بسته شده است





همان‌طور که در شکل قابل مشاهده است. پایه‌های IN1 و IN2 درایور به ولتاژ به دوتا از پایه‌های میکرو متصل هستند که فقط مقدار صفر و یک می‌گیرند و پایه enable درایور به پایه‌ای از میکرو متصل شده که PWM تولید می‌کند.

برای این بخش از یک موتور DC با کدCN503 استفاده شده است. اتصالات موتور به همراه انکدر در شکل زیر قابل مشاهده است.



۱- A Pulse

۲- B Pulse

۳- Vcc

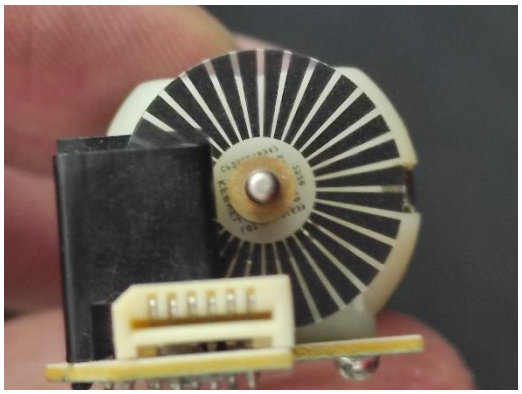
۴- GND

۵- Motor

۶- Motor

که در مدار ما دو پایه خروجی درایور به پایه‌های ۵ و ۶ موتور متصل شده است.

شیارهای انکدر ۳۲ تا سفید ۳۲ تا مشکی هستند.



برای شمارش پالس ها در کد به شکل زیر عمل شده است:

پایه‌های پالس A و B انکدر موتور به پین‌های A1 و A2 میکروکنترلر متصل شده اند.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if(GPIO_Pin == GPIO_PIN_1){ // PA1  -> A
    int16_t PA2 = HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_2); // -> B
    if(PA2 == GPIO_PIN_RESET){
     pulse_count++;
      return;
    }else if (PA2 == GPIO_PIN_SET){
     pulse_count--;
      return;
    }
  }
  if(GPIO_Pin == GPIO_PIN_2){ // PA2  -> B
    int16_t PA1 = HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_1); // -> A
    if(PA1 == GPIO_PIN_RESET){
     pulse_count--;
      return;
    }else if (PA1 == GPIO_PIN_SET){
     pulse_count++;
      return;
    }
  }

}
```

هروقت که اینتراپت تمام شود این تابع اجرا شده و تشخیص می‌دهد کدام یک از پالس‌های A یا B فعال شده اند. طبق تنظیمات انجام شده در cube انکدر حساس به لبه بالارونده است پس هروقت یک لبه بالا رونده اتفاق افتاد تشخیص می‌دهد که پالس دیگر set یا reset است که اگر set بود، شمارنده پالس را یکی کم و اگر reset بود شمارنده پالس را یکی زیاد می‌کند و برای پالس دیگر برعکس آن است.

در مرحله بعد لازم بود که برای فهمیدن سرعت فعلی و به دست آوردن فاصله‌ی سرعت فعلی با سرعت مطلوب از طریق پالس‌های شماره شده توسط انکدر سرعت را محاسبه کنیم که برای این منظور کد زیر پیاده سازی شد.

```
void update_speed(){
  int32_t delta_pulse = pulse_count - last_pulse_count;
  speed = delta_pulse / (float) delta_T ; // pulse per second
  speed = speed / PPR; // round per second
  last_pulse_count = pulse_count;
}
```

این تابع در callback تایمر که هر ۵۰ میلی ثانیه یکبار اجرا می‌شود، فراخوانی می‌شود و سرعت توسط آن محاسبه می‌شود. برای محاسبه سرعت اختلاف تعداد پالس‌های شمارش شده انکدر با سری قبلی که تابع فراخوانی شده است(یعنی ۵۰ میلی ثانیه قبل) بر زمان تقسیم شده و این گونه سرعت برحسب پالس در ثانیه به دست می‌آید و با تقسیم آن بر PPR (که تعداد شیارهای انکدر در یک دور چرخش کامل است.) سرعت فعلی به دست می‌آید.

توسط تابع زیر خروجی‌ای که از کنترلر به دست می‌آید به درایور موتور داده می‌شود.

```
void set_voltage(float duty_cycle){
  if(duty_cycle > 100){
    duty_cycle = 100;
  }else if(duty_cycle < -100){
    duty_cycle = -100;

  }
  if(duty_cycle> 0){
  TIM2->CCR1 = (duty_cycle/100.0 * counter_period);
    HAL_GPIO_WritePin(IN1_pin_GPIO_Port,IN1_pin_Pin,GPIO_PIN_RESET);
  HAL_GPIO_WritePin(IN2_pin_GPIO_Port,IN2_pin_Pin,GPIO_PIN_SET);

  }else{
    TIM2->CCR1 = (-1 * duty_cycle/100.0 * counter_period);
  HAL_GPIO_WritePin(IN1_pin_GPIO_Port,IN1_pin_Pin,GPIO_PIN_SET);
  HAL_GPIO_WritePin(IN2_pin_GPIO_Port,IN2_pin_Pin,GPIO_PIN_RESET);
  }


}
```

در ابتدا بررسی می‌شود که duty cycle که بر حسب درصد است از محدوده خود خارج نشده باشد. و بعد اگر duty cycle مثبت بود به صورت درصد در مقدار کل ضرب می‌شود و موتور در جهت مثبت می‌چرخد و اگر منفی بود موتور در جهت منفی می‌چرخد.

در نهایت سرعت به شکل کد زیر کنترل شده است که در ادامه توضیح داده می‌شود.

```
float error ;
float last_error = 0;
float integrator = 0;
float derivative = 0;

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
  float Kp = 1.5;
  float Ki = 15;
  float Kd = 0;

  update_speed();
   error = desired_speed - speed;
  integrator += integrator + Ki * delta_T * error;
  derivative = Kd * (error - last_error) / delta_T;
  last_error = error;

  if(integrator > 50 ){
    integrator = 50;
  }else if(integrator < -50){
    integrator = -50;
  }
  float out = Kp* error + integrator + derivative;
  set_voltage(out);
}
```

هر ۵۰ میلی ثانیه یکبار این تابع اجرا شده و سرعت فعلی را با تابع update_speed می‌یابد و از اختلاف این سرعت با مقدار مطلوب ارور را تشکیل می‌دهد. با ضرب کردن Kp در مقدار ارور ترم تناسبی ساخته می‌شود و با ضرب کردن Ki در مجموع ارورها ترم انتگرال گیر به دست می‌آید و با ضرب کردن Kd در تغییرات اروری که در بازه ۵۰ میلی ثانیه اتفاق افتاده است ترم مشتق گیر به دست می‌آید.

برای انتگرال گیر شرط اشباع روی ۵۰ در نظر گرفته شده است(نصف کل بازه)

در ابتدا ضرایب Ki و Kd صفر گذاشته شد و نتیجه این بود که با Kp بزرگتر از ۳ سیستم به ناپایداری می‌رسد پس ضریب K بحرانی برای این سیستم تقریبا ۳ است.

سیستم با کنترلر تناسبی به تنهایی دارای یه خطای دائم نزدیک ۱۰ بود به همین منظور انتگرال گیر اضافه شد که این خطای دائم رو به حدود ۱ تا ۲ کاهش داد و در نهایت چون سیستم سرعت مطلوبی داشت و اورشوت بالایی نداشت کنترلر PI برای آن در نظر گرفته شد.

به همراه فایل گزارش سه ویدیو نیز از نتایج قسمت عملی آپلود شده است که در ویدیو اول کنترلر فقط P است و به همین دلیل یک خطای حالت دائم داریم برای رفع این خطا در ویدیوهای بعدی کنترلر PI استفاده شده است.

در ویدیو دوم مشاهده می‌کنید که set piont روی ۴۰ تنظیم شده و سیستم با کنترلر PI با ضرایب اولیه

اولیه سرعت را حدود ۵۰ نگه می‌دارد یعنی دارای خطای ۱۰ است.

در ویدیو سوم ضریب Ki تیون شده و خطا را حدود صفر نگه می‌دارد و سرعت را حدود ۸۰ که مقدار set point در نظر گرفته شده است.

کلیه کد ها در فولدر stm32 می‌باشند.