
بسم الله الرحمن الرحیم

دانشکده مهندسی برق
گروه کنترل

گزارش پروژه مکاترونیک :

سیستم توپ و میله

Ball and Beam System

نگارش :
محمد رضا ابراهیمی
۴۰۱۴۱۱۰۱۲
محمد امین فراهانی فرد
۴۰۱۴۱۳۰۲۹
مهدی رحیم پور
۴۰۱۴۱۲۰۵۲

استاد :
دکتر سعید شمقدری

بهار ۱۴۰۴

فهرست مطالب

مقدمه

در عصر حاضر، طراحی و پیاده‌سازی سامانه‌های کنترل پیشرفته در حوزه‌های متنوعی از مهندسی، از جمله هوافضا، رباتیک، خودرو و مکاترونیک، به عنوان یکی از چالش‌های بنیادین و در عین حال جذاب مطرح است. یکی از شناخته‌شده‌ترین و در عین حال کاربردی‌ترین مسائل آموزشی و پژوهشی در این حوزه، مسئله کنترل موقعیت توپ روی یک میله (Ball and Beam) است. این سیستم که به‌ظاهر ساده به‌نظر می‌رسد، در حقیقت نمونه‌ای کلاسیک و قابل تعمیم از یک سیستم ناپایدار خطی یا غیرخطی با پویایی‌های قابل توجه است. پیچیدگی‌های دینامیکی، تعامل بین ورودی و خروجی، و حساسیت به اغتشاشات خارجی، این سامانه را به بستری مناسب جهت آزمودن و اعتبارسنجی الگوریتم‌های کنترلی مختلف تبدیل کرده است.

هدف از این پروژه، طراحی، پیاده‌سازی و تحلیل یک سامانه کنترلی واقعی برای سیستم Ball and Beam با اتکا به ترکیب روش‌های شناسایی سیستم، مدل‌سازی فیزیکی، و طراحی کنترل‌کننده‌های متنوع نظیر PID ، کنترل کسکید و پیاده‌سازی‌های سخت‌افزاری آن بر پایه‌ی میکروکنترلر Arduino بوده است. در این راستا، ابتدا با تحلیل دقیق نیازمندی‌های سیستم، طراحی مکانیکی، الکترونیکی و نرم‌افزاری آن صورت گرفت. سنسورهای اندازه‌گیری موقعیت مانند اولتراسونیک و ژیروسکوپ (MPU6050) به عنوان ورودی‌های اصلی سامانه در نظر گرفته شدند که با به‌کارگیری فیلترهای نرم‌افزاری نظیر فیلتر میانه و فیلتر کالمن، دقت و پایداری داده‌های حاصل از آن‌ها به‌طور معناداری افزایش یافت.

در گام بعد، جهت طراحی کنترل‌کننده‌ی مناسب، ابتدا با استفاده از روش‌های تجربی مانند شناسایی سیگنال پاسخ پله و شناسایی بر اساس پالس PWM و Pulse Duration ، مدل‌های دقیق سیستم در قالب توابع تبدیل استخراج شدند. این توابع تبدیل، به‌عنوان نمایندگان دینامیکی سیستم، مبنای طراحی و بهینه‌سازی کنترل‌کننده‌های مختلف قرار گرفتند. برای ارزیابی دقت مدل‌های شناسایی‌شده، تحلیل‌های فرکانسی شامل نمودارهای بود (Bode Plots) و پاسخ‌های زمانی در محیط MATLAB و Simulink انجام گرفتند.

از آنجا که سیستم مورد نظر ذاتاً ناپایدار است، طراحی یک کنترل‌کننده‌ی مؤثر نیازمند دقت بالا در مدل‌سازی و لحاظ کردن محدودیت‌های فیزیکی مانند محدوده حرکت میله، زمان پاسخ موتور، دامنه فرمان PWM و دقت سنسورها بود. از این‌رو، مجموعه‌ای از کنترل‌کننده‌ها با درجات پیچیدگی متفاوت شامل کنترلرهای PID کلاسیک، کنترل‌کننده‌های مبتنی بر Ziegler-Nichols ، و در نهایت ساختار کنترل کسکید (Cascade) برای کنترل همزمان زاویه میله و موقعیت توپ طراحی

و ارزیابی گردید. در کنار طراحی کنترل، اجرای الگوریتم‌های داخلی تست و راه‌اندازی اولیه سیستم (BIST)و پیاده‌سازی مراحل بوت سخت‌افزاری برای تضمین عملکرد ایمن و پایدار سیستم از دیگر نقاط قوت پروژه محسوب می‌شوند.

در نهایت، برای تحلیل کامل عملکرد کنترل‌کننده‌ها، مجموعه‌ای از تست‌ها در محیط شبیه‌سازی و محیط واقعی انجام گرفت و نقاط ضعف و قوت هر ساختار کنترلی به‌دقت مستندسازی گردید. نتایج نشان دادند که کنترل‌کننده‌های ساده‌تر مانندPID ، هرچند در شبیه‌سازی‌ها قابل قبول به‌نظر می‌رسند، اما در شرایط واقعی به دلیل پیچیدگی‌های غیرخطی، اصطکاک‌ها و تاخیرات سخت‌افزاری، به خصوص در راه‌اندازی‌های مرتبط با موتور DC پاسخ مناسبی ارائه نمی‌دهند؛ حال آنکه ساختارهای پیشرفته‌تر نظیر کنترل کسکید، توانستند با اعمال منطق کنترل دو حلقه‌ای، پایداری و دقت بالاتری در دنبال کردن مسیر مطلوب توپ فراهم آورند.

این پروژه ضمن تثبیت مفاهیم تئوریک کنترل در یک بستر عملی، بستری مناسب برای گسترش و تحقیق در زمینه‌های پیشرفته‌تر نظیر کنترل مقاوم، کنترل تطبیقی، یا کنترل مبتنی بر یادگیری ماشین در سامانه‌های دینامیکی غیرخطی فراهم می‌آورد.

فصل اول: تشکیلات و راه‌اندازی سیستم

مقدمه: سیستم کنترل موقعیت توپ روی میله (Ball and Beam) به عنوان یکی از مسائل پایه‌ای در مهندسی کنترل، نیازمند ترکیب مناسبی از سخت‌افزارهای دقیق و الگوریتم‌های سنسوری و کنترلی کارآمد است. در این قسمت به بررسی جامع تشکیلات و راه‌اندازی سخت‌افزاری و نرم‌افزاری به کار رفته در این پروژه و تحلیل دقیق دلایل انتخاب هر یک از آنها، چالش‌ها و پرداخته می‌شود

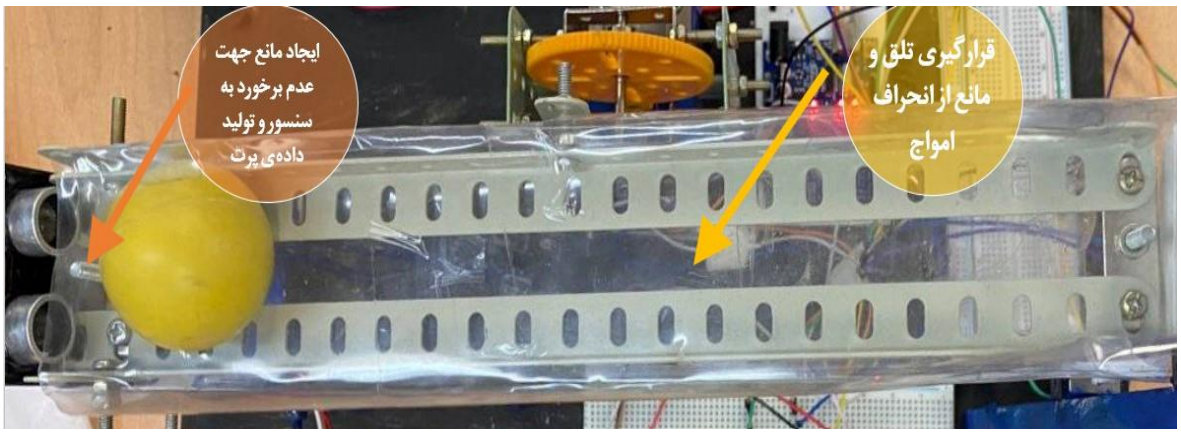
شکل ۱-۱ : (نمای کلی سیستم)

۱-۱. سیستم‌های اندازه‌گیری

جهت دریافت اطلاعات قابل اتکاء و پیاده‌سازی سیستم کنترلی از ۳ سیستم اندازه‌گیری استفاده شده است که در ادامه با جزییات آن‌ها آشنا می‌شویم.

۱-۱-۱. سنسور اولتراسونیک

برای اندازه‌گیری دقیق موقعیت توپ روی میله از سنسور اولتراسونیک استفاده شده است. این انتخاب مبتنی بر مزایای متعددی از جمله دقت قابل قبول در محدوده کاری پروژه، سرعت پاسخگویی مناسب و هزینه بهینه بوده است. به منظور افزایش دقت اندازه‌گیری موقعیت توپ، از یک تلق پلاستیکی نیمه‌شفاف در قسمت فوقانی میله استفاده شد. این تلق با تمرکز و هدایت یکنواخت امواج اولتراسونیک، موجب کاهش پراکندگی و بازتاب‌های ناخواسته امواج شد. در نتیجه، سیگنال بازگشتی از سطح توپ با نویز کمتر و وضوح بالاتر دریافت گردید. این اصلاح ساده در ساختار مکانیکی، منجر به کاهش محسوس خطای اندازه‌گیری و افزایش پایداری داده‌های دریافتی شد که در نتایج شناسایی سیستم نیز بهبود قابل توجهی ایجاد کرد.



شکل ۱-۲ : (اصلاحات مکانیکی برای دریافت خروجی ایده‌آل از سنسور اولتراسونیک)

اساس کار سنسور اولتراسونیک بر مبنای اصل زمان پرواز (Time-of-Flight) استوار است. در این روش، سنسور با ارسال پالس‌های صوتی با فرکانس بالا (معمولاً ۴۰ کیلوهرتز) و اندازه‌گیری زمان بازگشت اکو، فاصله تا شیء مورد نظر را محاسبه می‌کند. دقت این روش به عوامل متعددی از جمله دمای محیط، رطوبت و زاویه قرارگیری سنسور بستگی دارد که در طراحی سیستم به دقت مورد توجه قرار گرفته‌اند.

تابع getDistance() هسته اصلی این سیستم اندازه‌گیری را تشکیل می‌دهد. این تابع با ایجاد یک پالس تحریک ۱۰ میکروثانیه‌ای روی پایه تریگر سنسور، فرآیند اندازه‌گیری را آغاز می‌کند. سپس مدت

زمان انتظار برای دریافت بازتاب صوت با دقت میکروثانیه اندازه‌گیری می‌شود. ثابت ۰/۰۳۴ که در محاسبات استفاده شده، بر اساس سرعت صوت در هوای معمولی (تقریباً ۳۴۰ متر بر ثانیه) تعیین شده است. تقسیم نتیجه بر دو به این دلیل انجام می‌شود که زمان اندازه‌گیری شده شامل مسیر رفت و برگشت امواج صوتی است.

برای بهبود کیفیت اندازه‌گیری‌ها و افزایش قابلیت اطمینان سیستم، از تابع ()getFilteredDistance استفاده شده است. این تابع با بهره‌گیری از تکنیک فیلتر میانه، چندین نمونه متوالی را جمع‌آوری و پردازش می‌کند. روش کار بدین صورت است که ابتدا پنج اندازه‌گیری متوالی با فاصله زمانی ۵ میلی‌ثانیه از یکدیگر انجام می‌شود. سپس این مقادیر با استفاده از الگوریتم بابل سورت مرتب شده و مقدار میانه به عنوان خروجی انتخاب می‌شود. این رویکرد به ویژه در مواجهه با نویزهای لحظه‌ای و اندازه‌گیری‌های نادرست بسیار مؤثر عمل می‌کند.

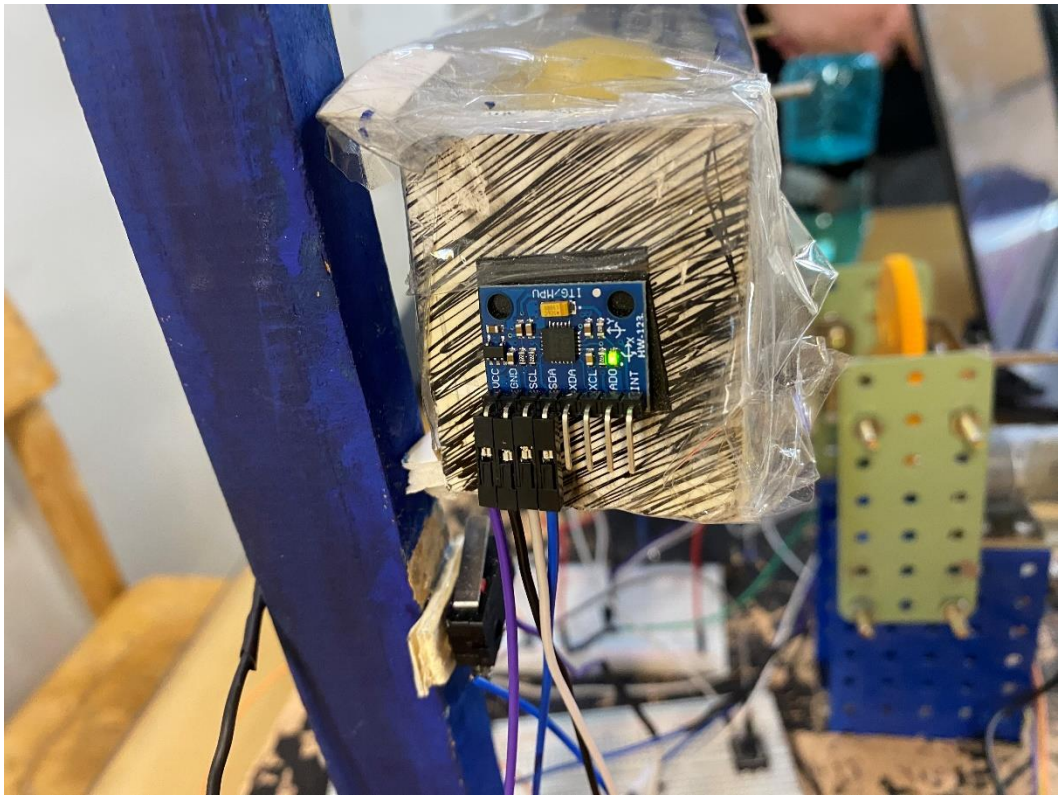
```
float getDistance() {
 digitalWrite(trigPin, LOW);
 delayMicroseconds(2);
 digitalWrite(trigPin, HIGH);
 delayMicroseconds(10);
 digitalWrite(trigPin, LOW);
 unsigned long duration = pulseIn(echoPin, HIGH, 30000);
 return (duration == 0) ? -1 : duration * 0.034 / 2;
}

float getFilteredDistance() {
 const int numReadings = 5;
 float readings[numReadings];

 for (int i = 0; i < numReadings; i++) {
  readings[i] = getDistance();
  delay(5);
 }
 // bubble sort for median
 for (int i = 0; i < numReadings - 1; i++) {
  for (int j = i + 1; j < numReadings; j++) {
   if (readings[i] > readings[j]) {
    float temp = readings[i];
    readings[i] = readings[j];
    readings[j] = temp;
```

```
  }
 }
}
return readings[numReadings / 2];
}
```

۱-۱-۲. سنسور MPU6050

اندازه‌گیری زاویه میله به عهده سنسور تلفیقی MPU6050 گذاشته شده که ترکیبی از ژیروسکوپ و شتاب‌سنج را در یک بسته ارائه می‌دهد. این سنسور در ساختار کنترل کسکید که در ادامه به تفصیل به آن می‌پردازیم، نقش حیاتی ایفا می‌کند، به طوری که حلقه کنترلی داخلی مسئول تنظیم زاویه میله بر اساس داده‌های این سنسور است. دقت قابل‌قبول در اندازه‌گیری زاویه و امکان محاسبه نرخ تغییرات زاویه‌ای از مزایای اصلی این انتخاب محسوب می‌شوند.

سنسور MPU6050 که در انتهای میله نصب شده بود، با وجود مزایای متعدد، در عمل با چالش اساسی مواجه بود که نیاز به راه‌حلی جامع داشت. مهم‌ترین مشکل، وجود نویز ذاتی در خروجی سنسور بود به طوری که حتی در شرایط ساکن نیز خروجی زاویه تا ۰/۵± درجه نوسان داشت. این نویز به صورت تصادفی و با توزیع گاوسی در داده‌ها مشاهده می‌شد.



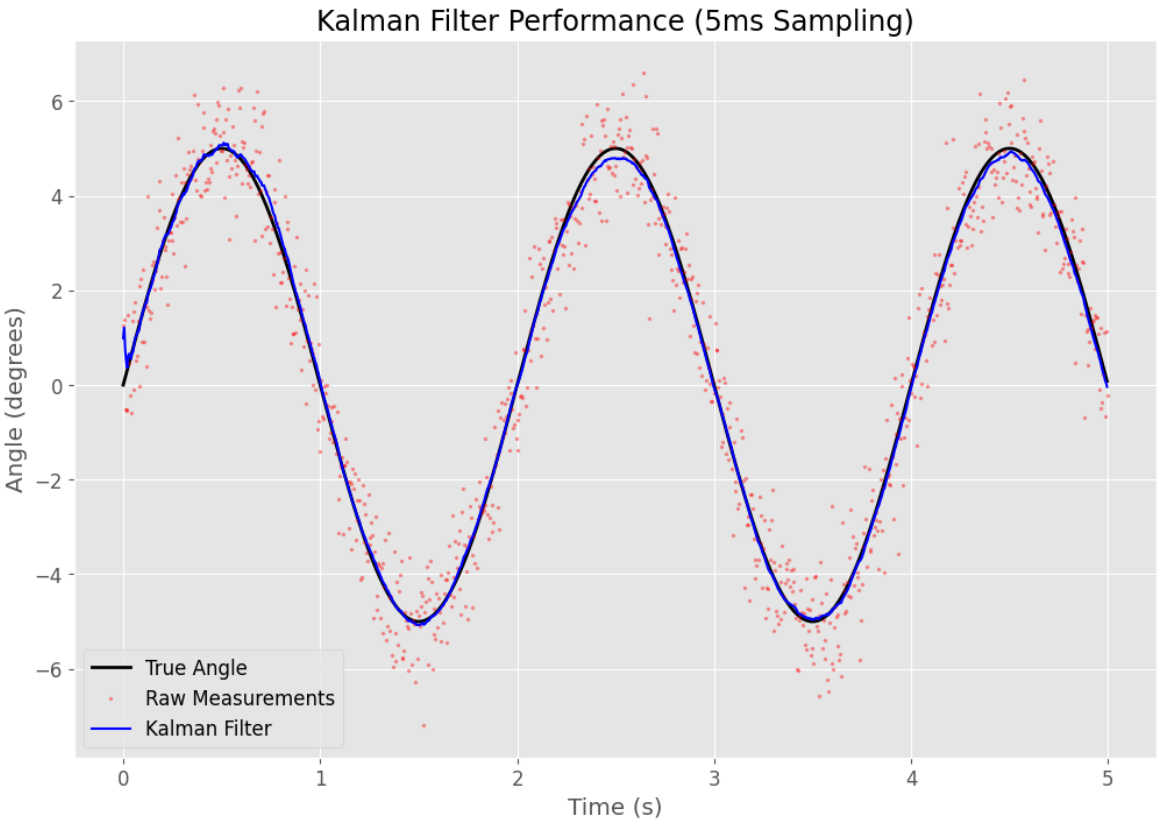
شکل ۱-۳ : (محل قرارگیری سنسور MPU6050)

فیلتر کالمن یک الگوریتم بهینه برای تخمین حالت‌های سیستم‌های دینامیکی است که بر اساس نظریه تخمین بیزین کار می‌کند. این فیلتر در سال ۱۹۶۰ توسط رودلف کالمن معرفی شد و امروزه به عنوان یکی از قدرتمندترین ابزارها در پردازش سیگنال‌های نویزی شناخته می‌شود. در طراحی این سیستم، از فیلتر کالمن برای بهبود دقت اندازه‌گیری‌های سنسور MPU6050 استفاده شده است. این فیلتر بر اساس مدلی از دینامیک میله کار می‌کند که شامل دو متغیر کلیدی است: زاویه فعلی میله (θ) و سرعت زاویه‌ای آن(ω) . این دو متغیر در قالب یک بردار حالت $x = [\theta, \omega]$بیان می‌شوند که اساس کار فیلتر کالمن را تشکیل می‌دهد. فیلتر کالمن در هر چرخه کاری خود دو مرحله اساسی را دنبال می‌کند:

در مرحله پیشبینی، سیستم با استفاده از اطلاعات قبلی و مدل دینامیکی میله، حالت فعلی را تخمین می‌زند. در این مرحله با فرکانس نمونه‌برداری ۲۰۰ هرتز (هر ۵ میلی‌ثانیه)، فیلتر ابتدا با استفاده از داده‌های ژیروسکوپ یک پیش‌بینی از حالت سیستم انجام می‌دهد. این پیش‌بینی بر اساس مدل فیزیکی سینماتیکی میله انجام می‌شود که در آن از جمع زاویه قبلی و حاصلضرب بازه زمانی در سرعت زاویه‌ای به دست می‌آید. همزمان، میزان عدم قطعیت این پیش‌بینی نیز محاسبه می‌شود.

مرحله به‌روزرسانی زمانی اتفاق می‌افتد که اندازه‌گیری جدید از سنسور دریافت می‌شود. در این مرحله، فاکتور کالمن (K) محاسبه می‌شود که در واقع وزن بهینه ای است بین اعتماد به پیش‌بینی مدل و اندازه‌گیری سنسور. این فاکتور به صورت دینامیکی و بر اساس میزان عدم قطعیت هر یک از این دو منبع محاسبه می‌شود. در نهایت، تخمین نهایی حالت سیستم و عدم قطعیت آن به روزرسانی پارامترهای اصلی فیلتر کالمن در این پروژه نیز به دقت تنظیم شده‌اند. ماتریس Q که بیانگر نویز فرایند است، برای زاویه مقدار ۰/۰۰۱ و برای سرعت زاویه‌ای مقدار ۰/۰۰۳ در نظر گرفته شده است. این اختلاف مقادیر نشان‌دهنده آن است که سرعت زاویه‌ای در سیستم ما از نویز بیشتری برخوردار است. ماتریس R نیز که نشان‌دهنده نویز اندازه‌گیری است، مقدار ۰. است. این تنظیمات بیانگر آن است که به اندازه‌گیری‌های ژیروسکوپ اعتماد بیشتری داریم.

نمودار زیر نشان می‌دهد که فیلتر کالمن توانسته است معادل ۸۴/۸٪ کاهش نویز در این سیستم از خود نشان دهد. همچنین دقت اندازه‌گیری زاویه از ۰/۸± درجه به ۰/۱۲±درجه رسیده که بهبودی چشمگیر محسوب می‌شود. همچنین سیستم در برابر لرزش‌ها و اغتشاشات مکانیکی نیز مقاومت بهتری از خود نشان می‌دهد.



شکل ۱-۴ : (عملکرد فیلتر کالمن با سمپلینگ ۲۰۰ هرتز)

```
MPU6050 mpu;

class KalmanFilter {
public:
 float Q_angle = 0.001;
 float Q_bias = 0.003;
 float R_measure = 0.03;
 float angle = 0;
 float bias = 0;
 float P[2][2] = { {0, 0}, {0, 0} };

 float getAngle(float newAngle, float newRate, float dt) {
  angle += dt * (newRate - bias);
  P[0][0] += dt * (dt*P[1][1] - P[0][1] - P[1][0] + Q_angle);
  P[0][1] -= dt * P[1][1];
  P[1][0] -= dt * P[1][1];
  P[1][1] += Q_bias * dt;

  float S = P[0][0] + R_measure;
```

```
  float K[2];
  K[0] = P[0][0] / S;
  K[1] = P[1][0] / S;
  float y = newAngle - angle;
  angle += K[0] * y;
  bias += K[1] * y;

  float P00_temp = P[0][0];
  float P01_temp = P[0][1];

  P[0][0] -= K[0] * P00_temp;
  P[0][1] -= K[0] * P01_temp;
  P[1][0] -= K[1] * P00_temp;
  P[1][1] -= K[1] * P01_temp;

  return angle;
 }
};

KalmanFilter kalmanY;
```

۱-۱-۳. پردازش تصویر

این سیستم با بهره‌گیری از تکنیک‌های پیشرفته پردازش تصویر، موقعیت دقیق توپ روی میله را در لحظه تشخیص می‌دهد. فرآیند کار با دریافت تصویر از وبکم آغاز شده و پس از مراحل پردازشی مختلف، خروجی دقیقی به سیستم کنترل ارسال می‌شود.

در مرحله اول، تصویر دریافتی از فضای رنگی RGB به HSV تبدیل می‌شود. این تبدیل هوشمندانه به سیستم اجازه می‌دهد بدون تأثیرپذیری از تغییرات نور محیط، رنگ توپ را به درستی تشخیص دهد. برای شناسایی توپ زرد، محدوده مشخصی از رنگ (Hue بین ۱۵ تا ۳۵)، غلظت رنگ (Saturation بین ۱۰۰ تا ۲۵۵) و روشنایی (Value بین ۱۰۰ تا ۲۵۵) تعریف شده است.

سپس تصویر وارد مرحله پردازش مورفولوژیکی می‌شود. در این بخش با اعمال عملیات Opening (حذف نویز و پرکردن فضاهای خالی) و Closing (ترکیب پرکردن حفره‌ها و حذف برجستگی‌های کوچک)، ناحیه مربوط به توپ به شکل دقیق‌تری جدا می‌شود. این عملیات با استفاده از یک ماتریس ۵ در ۵ پیکسلی انجام می‌پذیرد.

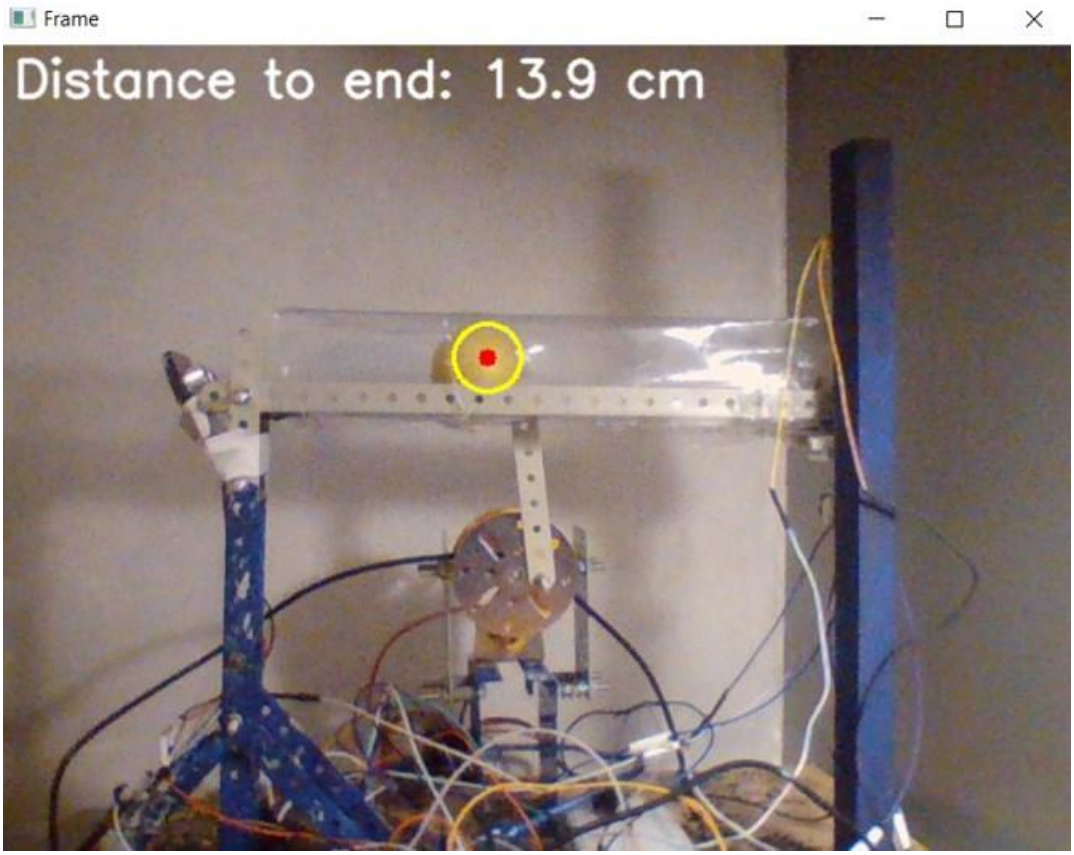
در مرحله بعد، سیستم با استفاده از الگوریتم تشخیص کانتور، تمام نواحی پیوسته باقیمانده را

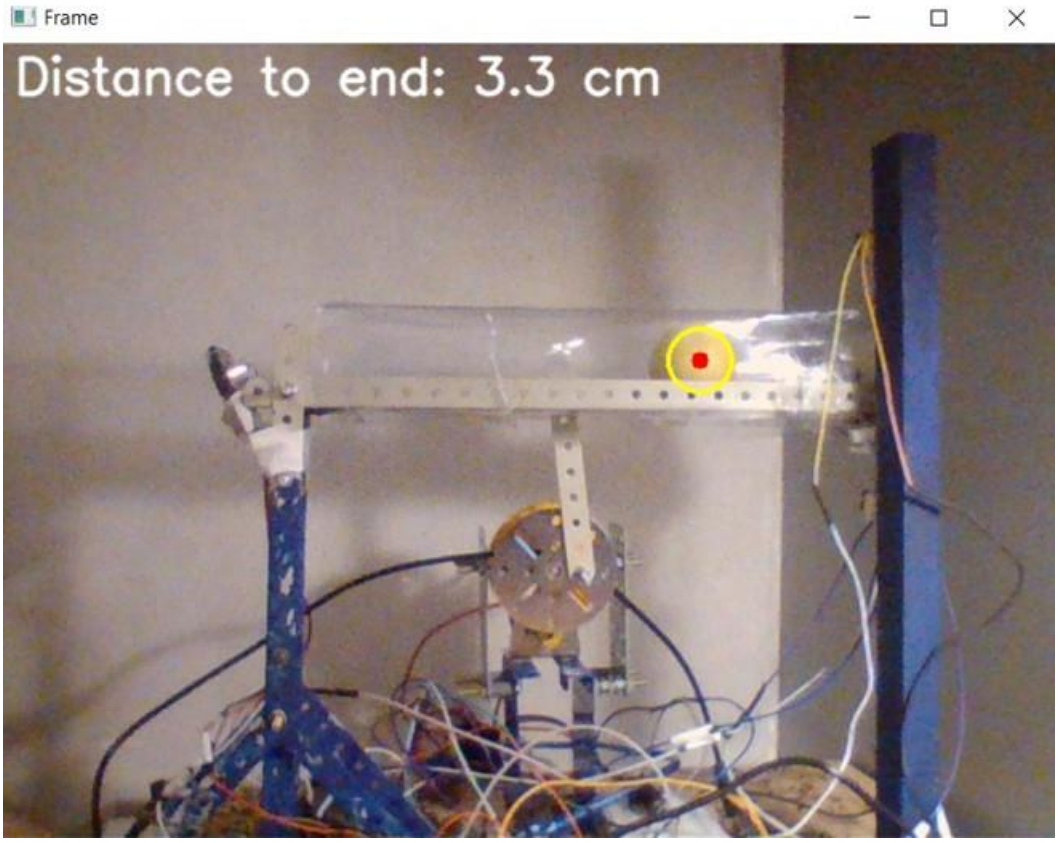
شناسایی می‌کند. از بین این نواحی، بزرگترین ناحیه به عنوان توپ انتخاب می‌شود. برای افزایش دقت، شرط حداقل شعاع ۱۰ پیکسل برای توپ در نظر گرفته شده تا از تشخیص‌های نادرست جلوگیری شود.

برای تعیین دقیق مرکز توپ، از روش Minimum Enclosing Circle استفاده می‌شود که کوچکترین دایره محاطی حول توپ را محاسبه می‌کند. سپس با استفاده از نقاط کالیبره شده (۰ و ۴۰ سانتیمتری روی میله)، موقعیت پیکسلی توپ به مقدار واقعی بر حسب سانتیمتر تبدیل می‌شود.

در نهایت، برای کاهش نوسانات اندازه‌گیری، از فیلتر میانگین متحرک استفاده می‌شود که در آن هر اندازه‌گیری جدید با وزن ۵۰ درصد و تاریخچه قبلی با وزن ۵۰ درصد در محاسبات لحاظ می‌شود. این روش باعث می‌شود خروجی سیستم نرم‌تر و پایدارتر باشد.

داده‌های نهایی با دقت ۰/۱ سانتیمتر و با نرخ ۲۰ فریم بر ثانیه به سیستم کنترل ارسال می‌شوند. این سرعت و دقت برای کنترل بهینه سیستم بال اند بیم کاملاً مناسب است. کل این فرآیند ترکیبی هوشمندانه از پردازش رنگ، عملیات مورفولوژیکی، هندسه تصویر و فیلترهای دیجیتال است که در کنار هم سیستمی دقیق و پایدار برای ردیابی توپ ایجاد کرده‌اند.





شکل ۱-۵ : (عملکرد پردازش تصویر در نواحی مختلف)

```
import cv2
import numpy as np
import serial
import time

# Initialize serial communication (adjust COM port as needed)
ser = serial.Serial('COM6', 9600)  # Adjust COM port as needed
time.sleep(2)  # Wait for serial port to initialize

# Open webcam
cap = cv2.VideoCapture(0)

# Define HSV color range for the yellow ball (more precise range)
lower_yellow = np.array([15, 100, 100])  # Lower bound of yellow hue
upper_yellow = np.array([35, 255, 255])  # Upper bound of yellow hue

# Calibration parameters (to be calibrated precisely)
x_pixel_0cm = 100  # Pixel position at 0cm (calibrate)
```

```
x_pixel_40cm = 600  # Pixel position at 40cm (calibrate)
pixels_per_cm = (x_pixel_40cm - x_pixel_0cm) / 40.0

# Beam parameters
beam_length_cm = 30  # Length of the beam in cm
beam_height_pixels = 50  # Height of the beam in pixels (for visualization)

# Offset calibration (adjust based on setup)
distance_offset_cm = 3.2  # Offset to compensate for camera position

# Moving average filter for distance smoothing
alpha = 0.5  # Smoothing factor (0.0 - 1.0)
smoothed_distance = None

try:
  while True:
    ret, frame = cap.read()
    if not ret:
      break

    # Convert to HSV color space
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Threshold for the yellow ball
    mask_yellow = cv2.inRange(hsv, lower_yellow, upper_yellow)
    mask_yellow = cv2.morphologyEx(mask_yellow, cv2.MORPH_OPEN, np.ones((5, 5),
np.uint8))
    mask_yellow = cv2.morphologyEx(mask_yellow, cv2.MORPH_CLOSE, np.ones((5, 5),
np.uint8))

    # Find contours of the yellow ball
    contours_yellow, _ = cv2.findContours(mask_yellow, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    if contours_yellow:
      # Find the largest contour (assuming it's the ball)
      largest_contour = max(contours_yellow, key=cv2.contourArea)
      (x, y), radius = cv2.minEnclosingCircle(largest_contour)
      if radius > 10:  # Filter out small contours (noise)
        # Draw yellow circle around the ball
```

```
        cv2.circle(frame, (int(x), int(y)), int(radius), (0, 255, 255), 2)
        cv2.circle(frame, (int(x), int(y)), 5, (0, 0, 255), -1)  # Red dot at center

        # Calculate ball distance from the camera reference point
        ball_x_cm = (x - x_pixel_0cm) / pixels_per_cm
        distance_to_end_cm = beam_length_cm - ball_x_cm + distance_offset_cm

        # Apply smoothing to reduce noise
        if smoothed_distance is None:
          smoothed_distance = distance_to_end_cm
        else:
          smoothed_distance = alpha * distance_to_end_cm + (1 - alpha) * smoothed_distance

        # Display the distance on the frame
        font = cv2.FONT_HERSHEY_SIMPLEX
        cv2.putText(frame, f'Distance to end: {smoothed_distance:.1f} cm', (10, 30), font, 1, (255,
255, 255), 2, cv2.LINE_AA)

        # Send data to Arduino
        data = f'{smoothed_distance:.1f}\n'
        ser.write(data.encode())
        time.sleep(0.05)

    # Draw beam for visualization
    cv2.rectangle(frame, (x_pixel_0cm, 100), (x_pixel_0cm + int(beam_length_cm *
pixels_per_cm), 100 + beam_height_pixels), (255, 0, 0), 2)

    # Display the resulting frame
    cv2.imshow("Frame", frame)

    # Exit on ESC key press
    if cv2.waitKey(1) == 27:  # ESC key
      break

except KeyboardInterrupt:
  print("Program interrupted by user.")

finally:
  # Release resources
```

```
cap.release()
cv2.destroyAllWindows()
ser.close()
```

۱-۲. سیستم‌های آغاز به کار و خودآزمایی داخلی (BIST)

سیستم کنترل طراحی شده از یک رویکرد ساختاریافته و ایمن برای راه‌اندازی و عملیات استفاده می‌کند. در این قسمت به تشریح دو فرآیند حیاتی سیستم می‌پردازیم: مرحله آغاز به کار و سیستم خودآزمایی داخلی(BIST) . این مکانیزم‌ها به منظور تضمین عملکرد مطمئن و پایدار سیستم قبل از ورود به حالت کنترل اصلی طراحی شده‌اند. در ادامه، جزییات هر یک از این مراحل شامل ترتیب عملیات، منطق کنترلی و معیارهای سنجش سلامت سیستم به صورت کامل شرح داده خواهد شد. این رویکرد امکان شناسایی و پیشگیری از خطاهای احتمالی را فراهم می‌سازد و بستری امن برای انجام آزمایش‌های کنترل فراهم می‌نماید.

۱-۲-۱. سیستم آغاز به کار

در این پروژه، سیستم Ball and Beam پس از روشن شدن به حالت Initialize می‌رود. در این وضعیت، سیستم در حالت پایدار قرار دارد و هیچ کنترلی روی میله اعمال نمیشود تا زمانی که کاربر دکمه Start را فشار دهد.

الگوریتم نرم‌افزاری:

۱- پس از روشن شدن، وضعیت سیستم به حالت Initialize میرود.

۲- در این حالت، موقعیت اولیه زاویه از MPU6050 خوانده میشود و اگر به شکل پایدار نباشد با اعمال الگوریتم کنترل میله به زاویه تعادل (۰ درجه) می‌رود.

۳- تا زمانی که دکمه Start فشرده نشود، سیستم از حالت Initialize خارج نمیشود.

۴- با فشردن دکمه Start، سیستم وارد حلقه کنترل اصلی میشود.

```
void initializeSystem() {
// 1. تنظیم اولیه سنسورها
Serial.println("System Initializing...");

// 2. کالیبراسیون اولیه MPU6050
mpu.initialize();
```

```
if (!mpu.testConnection()) {
 Serial.println("MPU6050 Initialization Failed!");
 while(1); // توقف سیستم در صورت خطا
}

// 3. تنظیم موتور در حالت خنثی
digitalWrite(IN1, LOW);
digitalWrite(IN2, LOW);
analogWrite(PWM, 0);

// 4. خواندن زاویه اولیه و تنظیم به صفر
float initialAngle = kalmanY.getAngle(0, 0, 0.01);
while(abs(initialAngle) > 0.5) { // حلقه تا رسیدن به تعادل
 anglePIDSetpoint = 0;
 angleInput = initialAngle;
 anglePID.Compute();
 controlMotor(millis());

 initialAngle = kalmanY.getAngle(0, 0, 0.01);
 delay(10);
}

// 5. انتظار برای فشرده شدن دکمه Start
Serial.println("System Ready. Press Start Button...");
while(digitalRead(START_BUTTON_PIN) == HIGH); // حلقه انتظار
Serial.println("Starting Control System...");
}
```

۱-۲-۲. سیستم خودآزمایی داخلی (BIST)

برای اطمینان از سلامت عملکرد سیستم قبل از ورود به کنترل، یک روتین BIST نیز طراحی شده که شامل تست سنسورها و موتور جهت حرکت امن و مطمئن سیستم می‌باشد. روال تست خودکار:
۱- میله در تمام فرآیند کنترل باید در یک زاویه محدود جابه‌جا شود. در انتها و ابتدا این بازه دو لیمیت سوییچ قرار دارند، با بالا و پایین شدن میله در این بازه زاویه‌ای و تحریک لیمیت‌سوییچ‌ها و بررسی مقدار خروجی سنسور MPU6050 به شکل همزمان از سلامت سنسور مطمئن می‌شویم و اگر خطا قابل توجه باشد، سیستم متوقف می‌شود. مثلاً فرض کنید در زمان برخورد میله به لیمیت سوییچ،

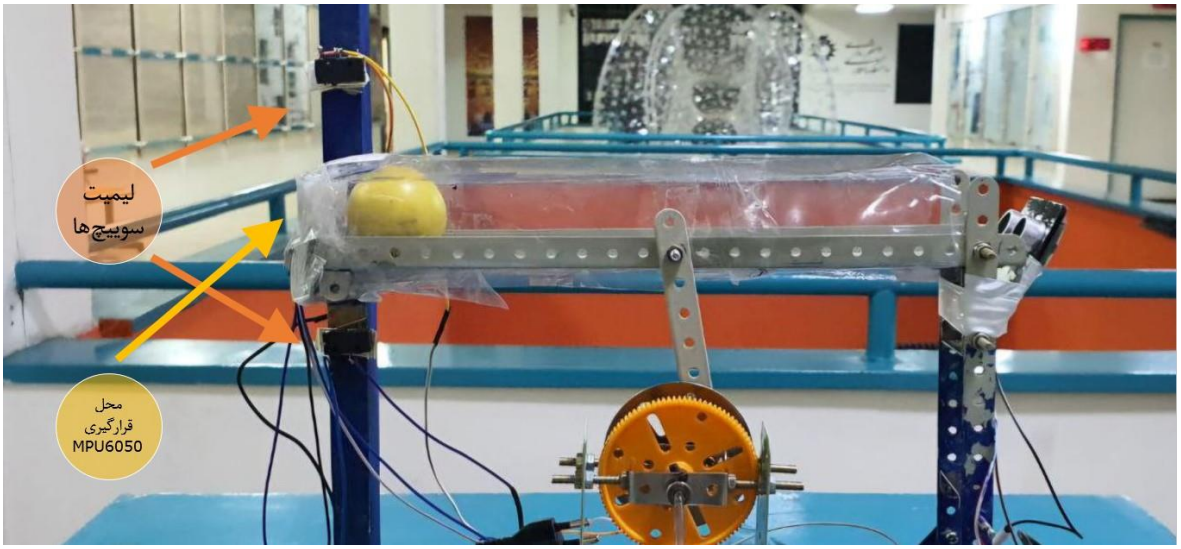
سنسور به ما زاویه‌ای بیشتر از ۱۲ درجه نشان دهد، این رخداد می‌تواند نشان از وجود مشکل در سنسور MPU6050 باشد.

۲- میله با اعمال پالس‌های مشخص به موتور توپ را در نواحی مختلف تکان می‌دهد و دیتای خروجی سنسور اولتراسونیک با دیتای خروجی پردازش تصویر مقایسه می‌شود و اگر اختلاف خروجی دو سیستم اندازه‌گیری قابل توجه باشد، سیستم متوقف می‌شود تا سیستم معیوب شناسایی و رفع ایراد شود.

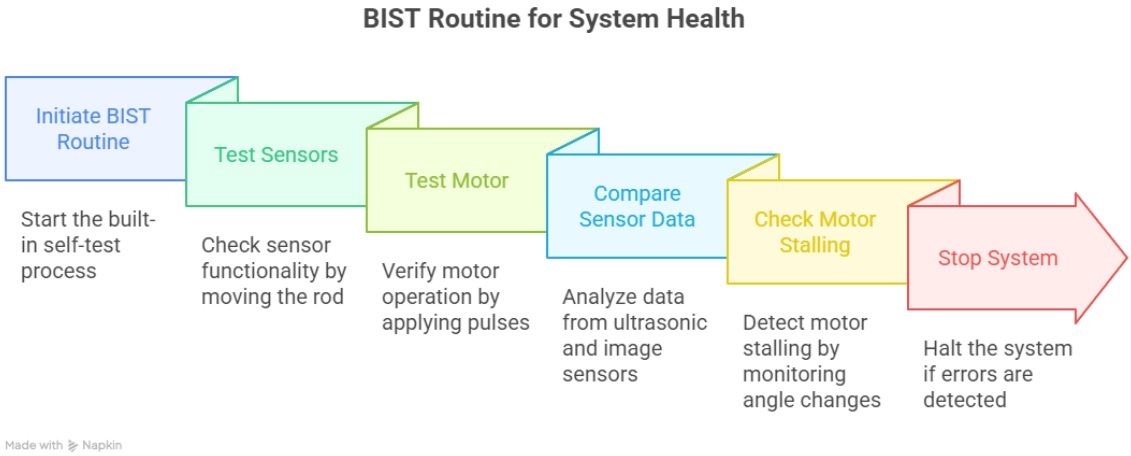
۳- اگر با اعمال PWM های مشخص تغییرات زاویه‌ای که MPU6050 ناچیز و نزدیک صفر باشد، به معنای کار نکردن / استال رفتن موتور تلقی می‌شود و سیستم متوقف می‌شود.

```
bool performBIST() {
 Serial.println("Running Built-In Self Test...");

 // 1. تست سنسور MPU6050 با لیمیت سوییچ‌ها
 Serial.println("Testing MPU6050 with Limit Switches...");
 moveMotorToLimit(HIGH); // حرکت به سمت لیمیت سوییچ بالا
 float angleAtLimit = kalmanY.getAngle(0, 0, 0.01);
 if(abs(angleAtLimit - LIMIT_SWITCH_ANGLE) > 1.0) {
  Serial.println("MPU6050 Test Failed!");
  return false;
 }

 moveMotorToLimit(LOW); // حرکت به سمت لیمیت سوییچ پایین
 angleAtLimit = kalmanY.getAngle(0, 0, 0.01);
 if(abs(angleAtLimit + LIMIT_SWITCH_ANGLE) > 1.0) {
  Serial.println("MPU6050 Test Failed!");
  return false;
 }

 // 2. تست هماهنگی سنسورها
 Serial.println("Testing Sensor Consistency...");
 for(int i = 0; i < 5; i++) {
  float usDistance = getFilteredDistance();
  float camDistance = getCameraDistance(); // تابع فرضی پردازش تصویر
  if(abs(usDistance - camDistance) > 2.0) { // 2 اختلاف مجاز cm
   Serial.println("Sensor Consistency Test Failed!");
   return false;
  }
```

```
 moveMotorRandom(); // حرکت تصادفی برای تست
 delay(500);
}

// 3. تست عملکرد موتور
Serial.println("Testing Motor Functionality...");
unsigned long startTime = millis();
float initialAngle = kalmanY.getAngle(0, 0, 0.01);
analogWrite(PWM, 150); // ثابت PWM اعمال
digitalWrite(IN1, HIGH);
digitalWrite(IN2, LOW);
delay(500); // 500 حرکت به مدت ms

float angleChange = kalmanY.getAngle(0, 0, 0.01) - initialAngle;
if(abs(angleChange) < 0.5) { // تغییر زاویه حداقل 0.5 درجه
 Serial.println("Motor Test Failed!");
 return false;
}

// 4. بازگشت به موقعیت اولیه
returnToZeroPosition();
Serial.println("All BIST Tests Passed!");
return true;
}

// تابع کمکی برای حرکت به سمت لیمیت سوییچ
void moveMotorToLimit(bool direction) {
 digitalWrite(IN1, direction);
 digitalWrite(IN2, !direction);
 analogWrite(PWM, 100);
 while(digitalRead(direction ? UPPER_LIMIT_PIN : LOWER_LIMIT_PIN) != LOW) {
  // انتظار برای فعال شدن لیمیت سوییچ
  if(millis() - startTime > 5000) { // زمان‌بندی اضطراری
   Serial.println("Limit Switch Timeout!");
   break;
  }
 }
 analogWrite(PWM, 0);
}
```



شکل ۱-۶ : (محل قرارگیری لیمیت سوییچ‌ها در انتهای میله اصلی)



شکل ۱-۷ : (روند شماتیک انجام روتین BIST)

۱-۳. سیستم محرکه موتور، گیربکس و تغذیه

در طراحی و پیاده‌سازی سامانه‌های کنترلی نظیر سیستمBall and Beam ، انتخاب و پیاده‌سازی سیستم محرکه، گیربکس و منبع تغذیه از جمله مؤلفه‌هایی است که به‌شدت بر عملکرد کلی، پاسخ دینامیکی، کیفیت کنترل، و پایداری سیستم اثرگذار است. در این پروژه، بخش محرکه شامل موتور، درایور، گیربکس، و مسیر انتقال توان مکانیکی است که پس از چندین مرحله طراحی، آزمایش، و بازطراحی، به ساختار نهایی رسید. انتخاب این اجزا به‌دلیل محدودیت‌های عملی، کمبود مستندات

دقیق در بازار داخلی، و نیاز به پاسخ سریع و دقیق، با چالش‌های فراوانی همراه بود که در ادامه به‌تفصیل بررسی می‌شود.

۱-۳-۱. انتخاب موتور و چالش‌های آن

در مراحل ابتدایی طراحی، موتور انتخاب‌شده یک موتور DC با مشخصات زیر بود:

- مدل: 25GA370
- سرعت بی‌باری: **۶۰۰ دور در دقیقه**
- ولتاژ: **۱۲ ولت**

دلیل اولیه انتخاب موتور DC نسبت به گزینه‌هایی نظیر موتورهای سروو، مربوط به پهنای باند کنترلی بالاتر **و** امکان کنترل پیوسته و دقیق سرعت و موقعیت از طریق مدولاسیون پهنای پالس (PWM) بود. برخلاف سروو‌هایی که حلقه کنترلی داخلی آن‌ها مختص کنترل زاویه بوده و عملاً یک واسط انتزاعی بین فرمان و خروجی ایجاد می‌کنند، موتورهای DC این امکان را می‌دهند تا مستقیماً بر روی جریان، سرعت و در نهایت موقعیت خروجی کنترل دقیق اعمال گردد؛ و در نتیجه، در مراحل طراحی اولیه که کنترل دقیق‌تری مدنظر بود، گزینه DC به‌عنوان انتخاب اول مطرح شد.

با این حال، در مراحل پیاده‌سازی و تست میدانی، مشکلات جدی‌ای در ارتباط با این انتخاب نمایان شد. به‌علت سرعت نسبتاً بالای موتور (600 RPM)و نبود گیربکس داخلی مؤثر (موتور تنها دارای یک گیربکس ساده و نسبتاً ضعیف داخلی بود)، در مقادیر پایینPWM ، موتور در ناحیه **استال (Stall)** قرار می‌گرفت. این مسئله باعث می‌شد که در مقاطع آغازین حرکت یا زمان‌هایی که کنترل‌کننده تلاش می‌کرد نیروی اصلاحی ضعیفی وارد کند، موتور عملاً بی‌حرکت مانده یا واکنش شدیدی نشان دهد که منجر به ایجاد نوسانات یا تأخیرهای خطرناک در پاسخ کنترلی می‌شد. این موضوع به‌ویژه در سیستم‌هایی مانند Ball and Beam که کنترل تعادلی لحظه‌ای حیاتی است، به‌شدت مخرب ظاهر شد.

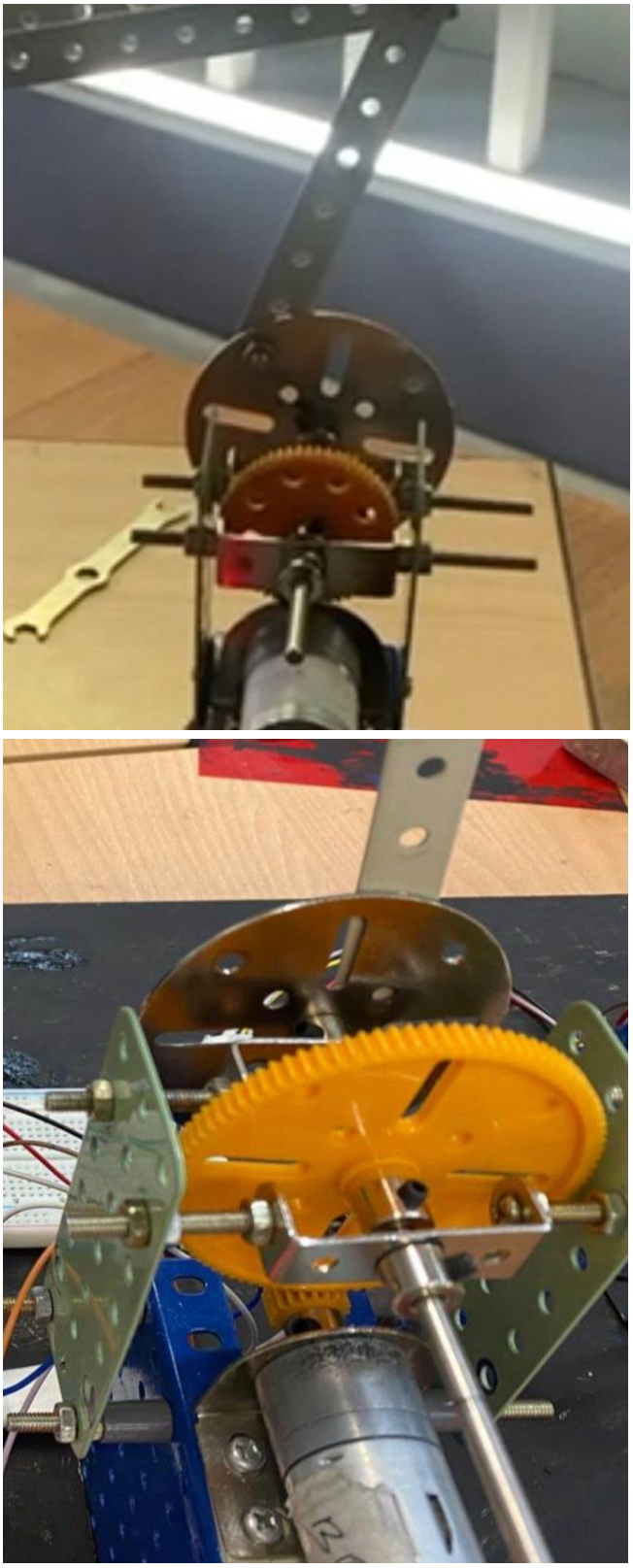
۱-۳-۲. طراحی و اصلاح ساختار گیربکس

برای جبران این مشکل، نیاز به طراحی یک ساختار گیربکس مکانیکی با نسبت کاهش بالا احساس شد. طراحی اصولی و تکنیکال بر اساس مبانی طراحی گیربکس به علت نیاز به اطلاعات دقیق موتور و ضعف جدی دیتاشیت موتورهای موجود در سطح بازار امکان‌پذیر نبود و ناچاراً فرآیند طراحی منجر به آزمون و خطا شد. در ابتدا، یک جفت چرخ‌دنده با نسبت انتقال ۱:۱۹ در مسیر خروجی موتور قرار

داده شد. با وجود بهبود محسوس در پاسخ‌دهی موتور، همچنان سرعت نهایی مجموعه برای نیازهای کنترلی سیستم بیش از حد زیاد بود. به همین جهت، در مرحله‌ی دوم، گیربکس به نسبت ۱:۵۷ افزایش یافت تا گشتاور بهبود یابد و در عین حال پاسخ سریع‌تر و نرم‌تری حاصل شود. با این وجود، همچنان مشخص شد که دینامیک سیستم، به‌ویژه در کنترل‌های ناحیه خطی اطراف نقطه تعادل، از دقت لازم برخوردار نیست. در نهایت، پس از بررسی‌های میدانی و چندین تست تجربی، نسبت انتقال نهایی به ۱:۹۲ افزایش یافت. این مقدار نه‌تنها توانست سرعت سیستم را به ناحیه قابل کنترل کاهش دهد، بلکه استال شدن موتور در مقادیر پایین PWM را تا حد زیادی کاهش داد (هرچند هنوزPWM های زیر ۹۰ رفتار مطلوبی ندارند). در خصوص استفاده از گیربکس‌های حلزونی(Worm Gear) ، علی‌رغم مزایای آن‌ها در فراهم‌سازی نسبت انتقال بالا و توانایی قفل‌شدگی در بارهای ساکن، چند عامل فنی و اجرایی مانع از پیاده‌سازی این نوع گیربکس در سیستم شد.گیربکس‌های حلزونی به‌دلیل اصطکاک لغزشی بالا، از بازده کمتری نسبت به گیربکس‌های دندانه‌دار بهره‌مند هستند که در کاربردهای نیازمند پاسخ سریع، یک نقطه ضعف محسوب می‌شود. از طرفی در حالی که قفل‌شدگی در بسیاری از کاربردها مفید است، در سیستمBall and Beam ، نیاز به حرکت برگشتی سریع و تغییرات لحظه‌ای جهت، این ویژگی را به یک نقطه منفی تبدیل می‌کرد

نکته حائز اهمیت آن است که در جریان این اصلاحات، محل نصب موتور نیز تغییر یافت. در طراحی اولیه، موتور در انتهای میله نصب شده بود. اما به‌دلیل نیاز به کاهش ممان اینرسی و کاهش بار استاتیکی روی شفت و چرخ‌دنده‌ها، در طراحی نهایی، موتور به‌سمت مرکز میله منتقل شد. این جابه‌جایی منجر به بهبود بازده مکانیکی سیستم، کاهش بار بر بلبرینگ‌ها و همچنین افزایش پاسخ‌پذیری مجموعه محرکه در نواحی بحرانی شد.

شکل ۱-۸ : (چرخدنده‌های ۱:۵۷ – ۱:۱۹ – ۱:۹۲ به ترتیب از بالا به پایین)



شکل ۱-۹ : (تغییر گیربکس ۱:۵۷ به ۱:۹۲)

۱-۳-۳. ملاحظات مربوط به منبع تغذیه و درایور

منبع تغذیه انتخابی برای راه‌اندازی موتور، یک منبع ۱۲ ولت، ۳ آمپر با قابلیت تحمل بارهای لحظه‌ای بالا بود. این مقدار جریان به‌صورت تجربی و بر اساس نیاز گشتاور موتور در شرایط حداکثری تعیین شد و در تمام مراحل تست توانست ولتاژ باثباتی را به موتور ارائه دهد. برای کنترل سیگنال توان موتور، درایور L298N به‌کار گرفته شد که با وجود محدودیت‌هایی از جمله افت ولتاژ داخلی و محدودیت جریان، در مقیاس پروژه‌های دانشگاهی انتخابی قابل قبول و در دسترس بود. در مسیر خروجی درایور به موتور، از یک کلید قطع و وصل مکانیکی استفاده شد تا امکان قطع اضطراری جریان یا تست‌های مرحله‌ای فراهم گردد.

۱-۳-۴. مقایسه موتور DC و سروو، و دلایل عملکرد نهایی

یکی از پرسش‌های اساسی در جریان پروژه، این بود که آیا انتخاب موتور DC با وجود آزادی کنترلی بیشتر، در عمل مناسب‌تر از سروو بوده یا خیر. در بررسی‌های انجام‌شده، اگرچه استفاده از موتور DC به ما امکان داد تا الگوریتم‌های کنترلی با دقت و انعطاف بیشتر (مانند کنترل کسکید و کنترل مبتنی بر MPU6050 برای حلقه درونی زاویه) را پیاده‌سازی کنیم، اما در عمل مشاهده شد که نبود حلقه داخلی بسته و آماده مانند سروو موتور باعث شد اجرای الگوریتم‌های کنترل زاویه به‌صورت نرم‌افزاری با تکیه بر(MPU6050) دقت پایین‌تری داشته باشد. به‌عبارتی، کنترل نرم‌افزاری زاویه موتور به‌عنوان حلقه درونی، علی‌رغم استفاده از سنسور دقیق، نتوانست به پایداری و دقت ذاتی حلقه فیدبک داخلی سروو برسد. همچنین چالش‌های مربوط به طراحی گیربکس نیز این نظر را تشدید می‌کند. بنابراین می‌توان گفت که یکی از علل احتمالی در ناپایداری یا عملکرد ضعیف برخی از کنترل‌کننده‌ها، همین پیچیدگی‌های ناشی از استفاده از موتور DC بوده است. تجربه پروژه حاضر نشان داد که انتخاب مؤلفه‌های سخت‌افزاری—به‌ویژه موتور، گیربکس و منبع تغذیه—نه‌تنها باید بر اساس معیارهای تئوریک صورت گیرد، بلکه در عمل، آزمون و خطا و تجربه‌های میدانی نقش تعیین‌کننده‌ای در رسیدن به طراحی بهینه دارند. محدودیت دسترسی به دیتاشیت‌های معتبر، عدم شفافیت در مشخصات فنی موتورها در بازار داخلی، و پیچیدگی‌های تعامل اجزای الکترومکانیکی، ما را وادار کرد تا ساختاری پویا و قابل بازطراحی در پیش بگیریم. نتیجه نهایی، اگرچه با چالش‌های متعدد همراه بود، اما به ساختاری نسبتاً پایدار، قابل تکرار، و تطبیق‌پذیر برای توسعه‌های بعدی منجر شد.

فصل دوم: شناسایی سیستم و طراحی کنترل‌کننده

مقدمه: در این فصل، با هدف مدل‌سازی دقیق دینامیک سیستم Ball and Beam ، فرآیند شناسایی سیستم با استفاده از داده‌های تجربی طی دو روش Pulse Duration و PWM انجام شده است. داده‌های لازم با استفاده از سنسور اولتراسونیک و زاویه‌سنج MPU6050 استخراج گردیده و سپس با بهره‌گیری از جعبه‌ابزار System Identification در نرم‌افزار MATLAB ، توابع تبدیل سیستم در حالات مختلف تخمین زده شده‌اند. برای ارزیابی دقت مدل‌ها، تحلیل پاسخ فرکانسی به‌کار گرفته شده و مرتبه‌های مختلف مدل‌ها مورد بررسی قرار گرفته‌اند. همچنین در ادامه، با تکیه بر مدل‌های به‌دست‌آمده، کنترل‌کننده‌های PID و ساختار کنترلی کسکید طراحی و تحلیل شده‌اند.

۲-۱. شناسایی با Pulse Duration

در این مرحله ما در ابتدا یک فایل از دیتاهای سمپل آماده می کنیم تا بتوانیم عمل شناسایی را برای سیستم با استفاده از متلب انجام دهیم ، برای این کار در Arduino ide یک کد می نویسم تا با استفاده از بتوانیم دیتا هارا برای شناسای سیستم و پیدا کردن تابع تبدیل ان بدست اوریم .
کد در Arduino ide بصورت زیر می باشد:

```
const int IN1 = 7;
const int IN2 = 8;
const int PWM = 9;
const int trigPin = 10;
const int echoPin = 11;

int pwmLevels[] = {-120,120,120,120,-120,-120,120,-120,120};
int currentPWMIndex = 0;
bool pulseActive = false;

unsigned long lastUpdate = 0;
unsigned long pulseStartTime = 0;

const int numDurations = 4;
```

```
unsigned long pulseDurations[] = {50, 100, 650, 300,120, 750 ,70, 400 ,500, 60 ,600 ,150, 700,
                                                                                    200 , 900};

unsigned long waitDurations[] = {100,200,300, 500,700,1000};
int durationIndex = 0;

void setup() {
 Serial.begin(9600);

 pinMode(IN1, OUTPUT);
 pinMode(IN2, OUTPUT);
 pinMode(PWM, OUTPUT);
 pinMode(trigPin, OUTPUT);
 pinMode(echoPin, INPUT);
}

void loop() {
 unsigned long now = millis();

 unsigned long pulseDuration = pulseDurations[durationIndex];
 unsigned long waitDuration  = waitDurations[durationIndex];

 if (!pulseActive && (now - lastUpdate >= waitDuration)) {
  int pwm = pwmLevels[currentPWMIndex];
  applyPWM(pwm);
  pulseStartTime = now;
  pulseActive = true;
  lastUpdate = now;
 }

 if (pulseActive && (now - pulseStartTime >= pulseDuration)) {
  applyPWM(0);
  pulseActive = false;

  //
  float distance = getDistance();
```

```
 /
 Serial.print(millis());
 Serial.print("");
 Serial.print(pwmLevels[currentPWMIndex]);
 Serial.print("");
 Serial.print(distance, 1);
 Serial.print("");
 Serial.print(pulseDuration);
 Serial.print("");
 Serial.println(waitDuration);

 currentPWMIndex = (currentPWMIndex + 1) % (sizeof(pwmLevels) / sizeof(pwmLevels[0]));
 durationIndex = (durationIndex + 1) % numDurations; // رفتن به ست بعدی زمان‌ها
 }
}

void applyPWM(int pwm) {
 if (pwm >= 0) {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
 } else {
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  pwm = -pwm;
 }
 analogWrite(PWM, constrain(pwm, 0, 255));
}

float getDistance() {
 digitalWrite(trigPin, LOW);
 delayMicroseconds(2);
 digitalWrite(trigPin, HIGH);
 delayMicroseconds(10);
 digitalWrite(trigPin, LOW);

 unsigned long duration = pulseIn(echoPin, HIGH, 30000);
 if (duration == 0 || duration >= 30000) return -1;
```

```
float distance = duration * 0.034 / 2;
return distance;
}
```

توضیحات کد:

این کد آردوینو برای کنترل یک موتور DC به کمک سیگنال PWM و اندازه‌گیری فاصله با استفاده از سنسور التراسونیک طراحی شده است. ابتدا پین‌های مرتبط با جهت چرخش موتور (IN1) و(IN2) و پین PWM ، و پین‌های سنسور التراسونیک (trigPin) و (echoPin) تعریف می‌شوند. سپس آرایه‌هایی شامل سطوح مختلفPWM ، زمان‌های پالس، و زمان‌های انتظار تعریف شده‌اند تا سیستم در هر تکرار رفتار متفاوتی از خود نشان دهد. در تابع loop، در صورتی که زمان انتظار به پایان رسیده باشد، یک پالس PWM با مقدار مشخص به موتور اعمال می‌شود و تایمر آغاز می‌گردد. پس از اتمام مدت پالس، موتور متوقف شده و سنسور التراسونیک فعال می‌شود تا فاصله اندازه‌گیری شود. سپس داده‌ها شامل زمان فعلی، مقدارPWM ، فاصله اندازه‌گیری‌شده، مدت پالس و مدت انتظار از طریق پورت سریال ارسال می‌شوند. تابع applyPWM مسئول تعیین جهت چرخش موتور بر اساس علامت PWM و اعمال مقدار آن است، در حالی که getDistance وظیفه‌ی ارسال پالس به سنسور و محاسبه فاصله بر حسب سانتی‌متر را بر عهده دارد. این کد امکان تست رفتار سیستم مکانیکی در شرایط مختلف زمانی و با مقادیر متفاوت PWM را فراهم می‌کند و داده‌های لازم برای تحلیل عملکرد آن را فراهم می‌سازد.

حال که دیتا های لازم را جمع کردیم ،وقت آن است که به سراغ متلب برویم و با استفاده از آن و جعبه‌ابزار شناسایی مربوط به آن تابع تبدیل سیستم را به دست آوریم.

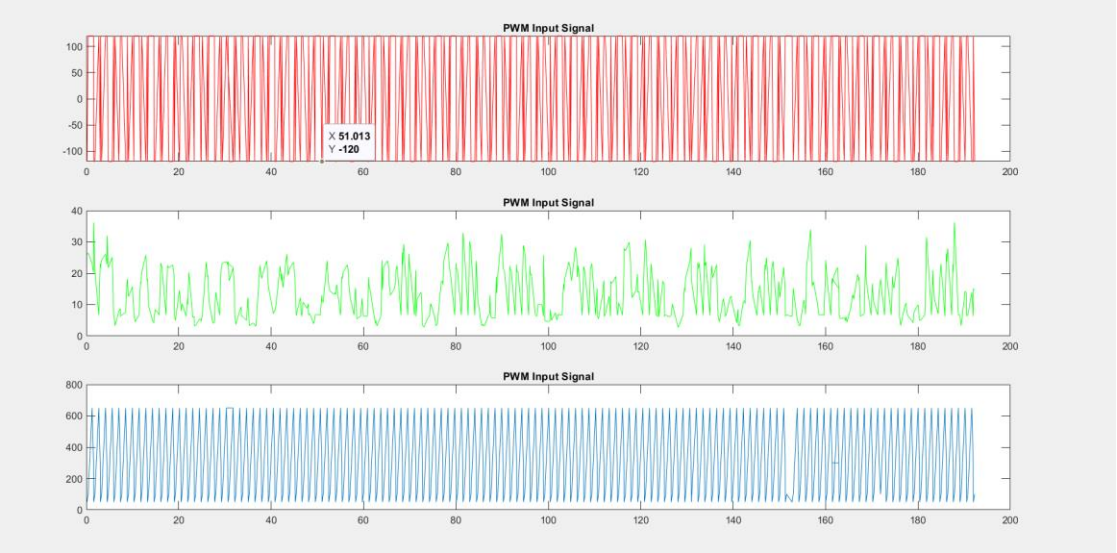
در متلب با استفاده از کد زیر می خواهیم سیستم را شناسایی کنیم:

```
% MATLAB Code for Sinusoidal Input Identification with Pulse Duration

% File to load data
filename = 'فtestt.txt';

% Load Data
data = readmatrix(filename);
time_data = data(:, 1) / 1000; % Convert to seconds
```

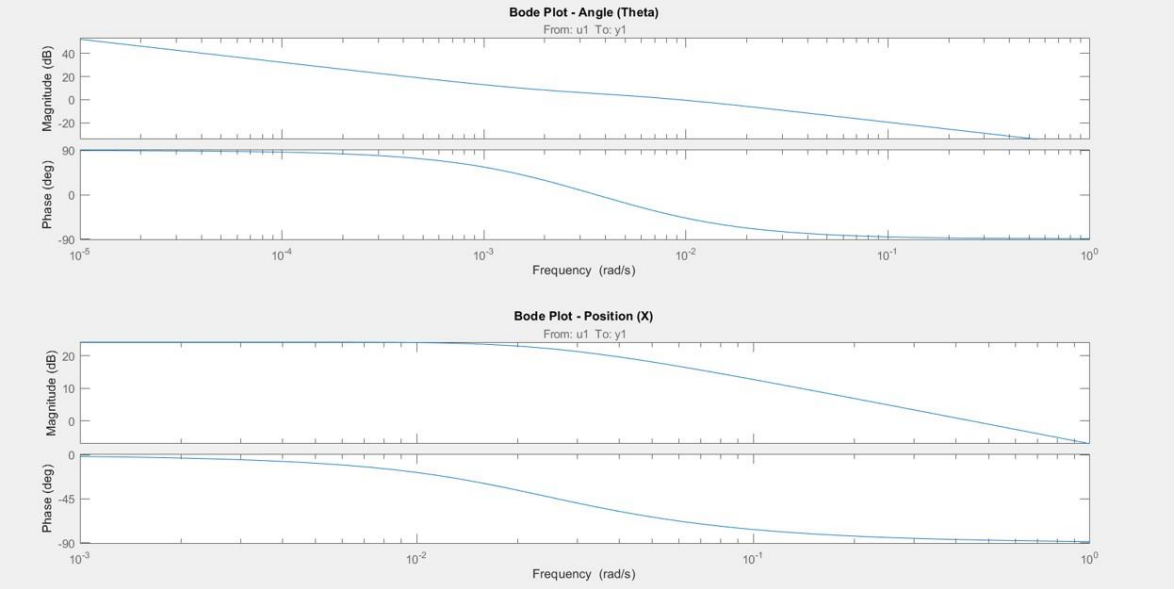
```
pwm_data = data(:, 2);
angle_data = data(:, 3);
distance_data = data(:, 4);
pulse_duration = data(:, 5);

% Sample Time Calculation
Ts = mean(diff(time_data));

% Create iddata objects
theta_data_id = iddata(angle_data, pwm_data, Ts);
x_data_id = iddata(distance_data, pwm_data, Ts);

% Model Order Selection
order = [2 1];

% Transfer Function for Angle (Theta)
tf_theta = tfest(theta_data_id, order(1), order(2));

% Transfer Function for Position (X)
tf_x = tfest(x_data_id, order(1), order(2));

% Display Transfer Functions
disp('Transfer Function for Angle (Theta):');
disp(tf_theta);

disp('Transfer Function for Position (X):');
disp(tf_x);

% Frequency Response Analysis
figure;
subplot(2, 1, 1);
bode(tf_theta);
title('Bode Plot - Angle (Theta)');

subplot(2, 1, 2);
bode(tf_x);
title('Bode Plot - Position (X)');
```

توضیحات کد:

این کد MATLAB در ابتدا فایل داده‌ای با نام 'ف 'testt.txtبارگذاری می‌شود که شامل اطلاعات مربوط به زمان (که از میلی‌ثانیه به ثانیه تبدیل می‌شود)، ورودیPWM ، زاویه(theta) ، فاصله (position)و مدت‌زمان پالس است. سپس با استفاده از اختلاف بین نمونه‌های زمانی، زمان نمونه‌برداری (Ts) محاسبه می‌شود. با استفاده از داده‌های زاویه و فاصله به عنوان خروجی، و PWMبه عنوان ورودی، دو شیء iddataساخته می‌شود که برای شناسایی سیستم استفاده می‌گردند. مدل‌سازی به صورت تابع تبدیل خطی با استفاده از دستور tfest انجام می‌شود که با مشخص کردن مرتبه مدل (۲ قطب و ۱ صفر) دو تابع تبدیل برآورد می‌گردد: یکی برای زاویه و دیگری برای موقعیت. نتایج به‌دست‌آمده با دستور disp چاپ می‌شوند و سپس نمودار بود Bode) (Plotبرای هر دو مدل ترسیم می‌شود تا رفتار فرکانسی سیستم‌ها بررسی گردد. این تحلیل به‌ویژه در کاربردهایی که ورودی‌ها به صورت پالس‌های متناوب یا سینوسی هستند اهمیت دارد و کمک می‌کند تا دینامیک سیستم به صورت دقیق مدل‌سازی شود. نمودار های تحلیلی بصورت زیر است:



شکل ۱-۱۰ : ((نمودار های دیتا های ورودی برای تحیل)



شکل ۱-۱۱ : (نمودار بود)

در طی فرایند شناسایی حالات مختلفی را برای شناسایی سیستم در نظر گرفتم که تمام مراحل را در ادامه توضیح خواهیم داد؛ در اینجا ما از روشی به نام پالس دیوریشن استفاده کرده‌ایم. در این روش می‌خواهیم به موتور PWM های ثابت بدهیم با این تفاوت که عرض پالس را ما انتخاب می کنیم، این یعنی ما به موتور می‌گوییم که به چه اندازه باید کار بکند. تابع تبدیل های به دست آمده در این فرآیند شناسایی در ادامه آمده است.

تابع تبدیل (X)و نتایج مربوط به آن:

```
Transfer Function for Position (X):
  idtf with properties:

       Numerator: [0.4431 0.0064]
     Denominator: [1 0.0387 3.9653e-04]
        Variable: 's'
         IODelay: 0
       Structure: [1×1 pmodel.tf]
   NoiseVariance: 2.0267e+04
      InputDelay: 0
     OutputDelay: 0
       InputName: {'u1'}
       InputUnit: {''}
      InputGroup: [1×1 struct]
      OutputName: {'y1'}
      OutputUnit: {''}
     OutputGroup: [1×1 struct]
           Notes: [0×1 string]
        UserData: []
            Name: ''
              Ts: 0
        TimeUnit: 'seconds'
    SamplingGrid: [1×1 struct]
```

شکل ۱-۱۲ (شناسایی تابع تبدیل درجه ۲ برای فاصله)

تابع تبدیل θ و نتایج مربوط به آن:

```
Model Properties
>> ppppppppp1
Transfer Function for Angle (Theta):
  idtf with properties:

       Numerator: [0.0122 -0.0122]
     Denominator: [1 0.7845 0.1729]
        Variable: 's'
         IODelay: 0
       Structure: [1×1 pmodel.tf]
   NoiseVariance: 10.1435
      InputDelay: 0
     OutputDelay: 0
       InputName: {'u1'}
       InputUnit: {''}
      InputGroup: [1×1 struct]
      OutputName: {'y1'}
      OutputUnit: {''}
     OutputGroup: [1×1 struct]
           Notes: [0×1 string]
        UserData: []
            Name: ''
              Ts: 0
        TimeUnit: 'seconds'
    SamplingGrid: [1×1 struct]
          Report: [1×1 idresults.tfest]
```

شکل ۱-۱۳ (شناسایی تابع تبدیل درجه ۲ برای زاویه)

برای سیستم از شناسایی درجه ۳ و درجه ۴ نیز استفاده کرده‌ایم. ولی به دلیل حاشیه پایداری نامناسب پاسخ مطلوب ما را ایجاد نمی‌کردند. تابع تبدیل مربوط به هرکدام را در ادامه قرار می دهیم:

```
Transfer Function for Position (X):
  idtf with properties:

         Numerator: [9.0986 433.2385]
       Denominator: [1 32.5098 224.9942 57.7244]
          Variable: 's'
           IODelay: 0
         Structure: [1×1 pmodel.tf]
     NoiseVariance: 0.6871
        InputDelay: 0
       OutputDelay: 0
         InputName: {'u1'}
         InputUnit: {''}
        InputGroup: [1×1 struct]
        OutputName: {'y1'}
        OutputUnit: {''}
       OutputGroup: [1×1 struct]
             Notes: [0×1 string]
          UserData: []
              Name: ''
                Ts: 0
          TimeUnit: 'seconds'
      SamplingGrid: [1×1 struct]
            Report: [1×1 idresults.tfest]
```

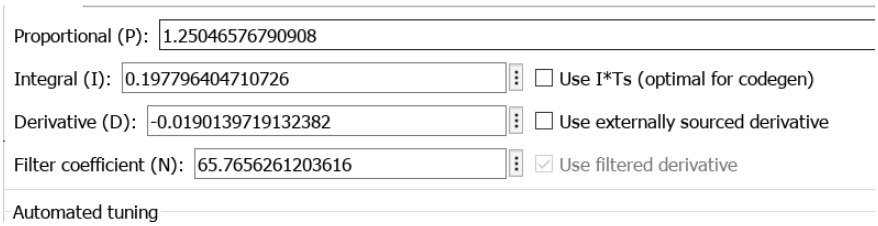
شکل ۱-۱۴ : (شناسایی درجه ۳)

```
Transfer Function for Angle (Theta):
  idtf with properties:

        Numerator: [22.9908 -1.2970e+03]
      Denominator: [1 43.6868 2.1044e+03 3.2847e+04]
         Variable: 's'
          IODelay: 0
        Structure: [1×1 pmodel.tf]
    NoiseVariance: 0.4308
       InputDelay: 0
      OutputDelay: 0
        InputName: {'u1'}
        InputUnit: {''}
       InputGroup: [1×1 struct]
       OutputName: {'y1'}
       OutputUnit: {''}
      OutputGroup: [1×1 struct]
            Notes: [0×1 string]
         UserData: []
             Name: ''
               Ts: 0
         TimeUnit: 'seconds'
     SamplingGrid: [1×1 struct]
           Report: [1×1 idresults.tfest]
```

شکل ۱-۱۵ :(شناسایی درجه ۴)

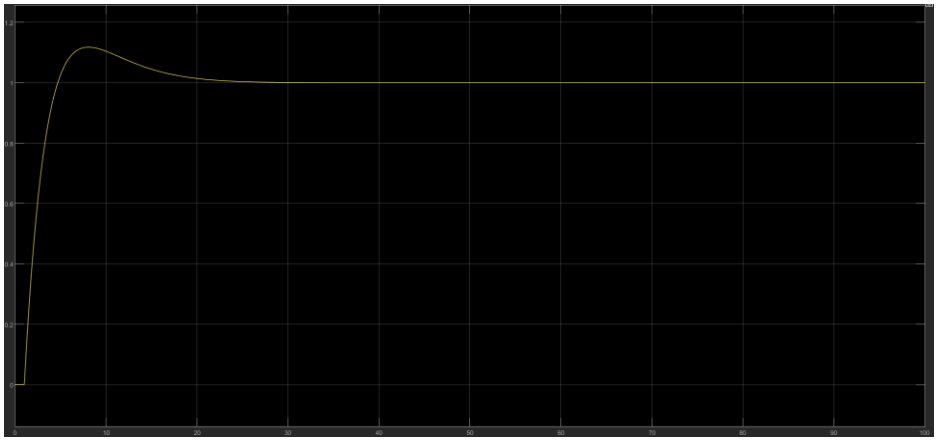
برای کنترل سیستم چند روش را امتحان کردیم :

در ابتدا با تنها با داشتن موقعیت و طراحی PID قصد کنترل سیستم را داشتیم که نتیجه بصورت زیر می باشد:



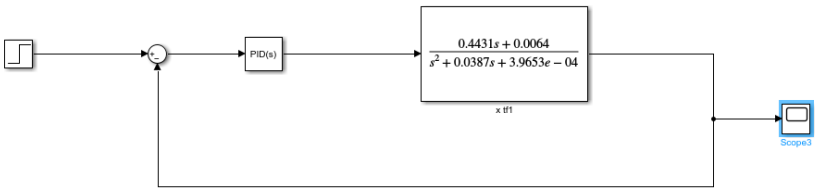
شکل ۱-۱۶ (ضرایب PID برای موقعیت)

و پاسخ آن برای setpoint = 1 :



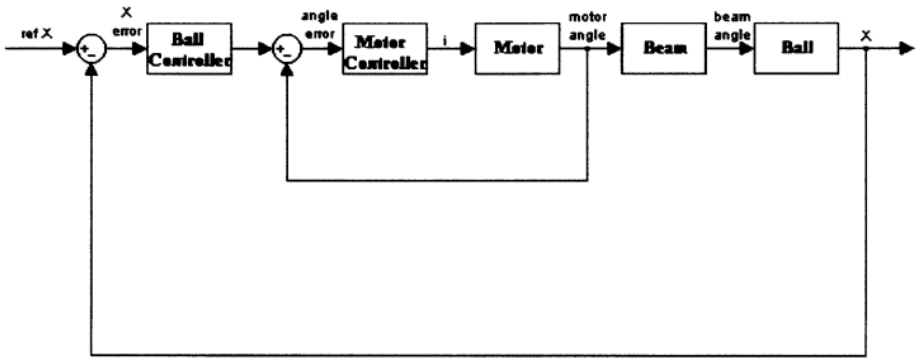
شکل ۱-۱۷ (پاسخ سیستم بعد کنترل به ورودی ۱)

و بلوک دیاگرام آن در سیمولینک بصورت زیر است:



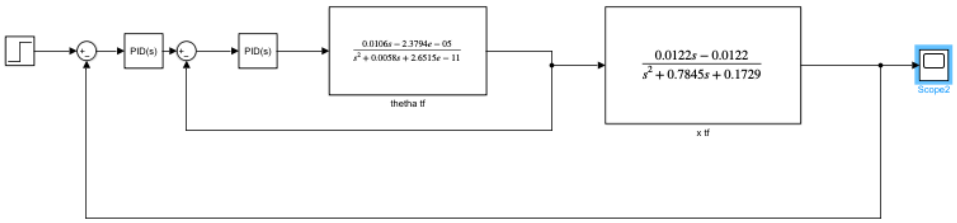
شکل ۱-۱۸ (بلوک دیاگرام در سیمولینک)

این حالت مطلوب ما نبود و نمی توانستیم توپ را بروی میله با آن کنترل کنیم؛ پس برای کنترل بسراغ کنترل کننده کسکید رفتیم و دوباره فرایند طراحی کنترل کننده کسکید را برای آن پیاده کردیم.



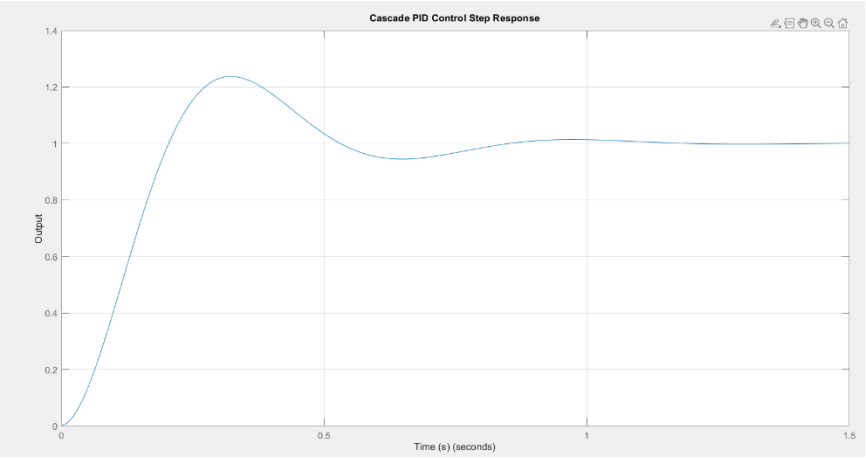
شکل ۱-۱۹ (نمونه ای از کنترل کننده کسکید)

بلوک دیاگرام ان در سیمولینک بصورت زیر است:



شکل ۱-۲۰ (بلوک دیاگرام کسکید در سیمولینک)

پاسخ سیستم با کنترل کننده کسکید بصورت زیر درآمد:



شکل ۱-۲۱ (پاسخ سیستم کسکید به ورودی پله واحد)

و ضرایب PID های آن بصورت زیر می باشد:

```
کنترل‌کننده خارجی (برای x):
  Kp = 30.0000
  Ki = 1.0000
  Kd = 0.1000

کنترل‌کننده داخلی (برای θ)
 Kp = -31.4669
 Ki = -5.2588
 Kd = -33.8086
```

برای دقت بیشتر از روش زیگلر نیکولز هم استفاده کردیم که بصورت زیر است:

تابع انتقال سیستم:

$$Gx(s) = \frac{0.4431s + 0.0064}{s^2 + 0.0387s + 3.9653 \times 10^{-4}}$$

سپس ضرایب کنترل کننده PID به صورت زیر محاسبه می‌شوند:

$$K_u = 85.40$$
$$T_u = 4.32 \text{ (ثانیه)}$$

محاسبه ضرایبPID :

$$K_p = 0.6 \cdot K_u$$
$$K_i = \frac{2K_p}{T_u}$$
$$K_d = \frac{K_p \cdot T_u}{8}$$

با جایگذاری مقادیر:

$$K_p = 0.6 \cdot 85.40 = 51.24$$
$$K_i = \frac{2 \cdot 51.24}{4.32} = 23.73$$
$$K_d = \frac{51.24 \cdot 4.32}{8} = 27.63$$

برای زاویه داریم:

$$G_\theta(s) = \frac{0.0122s - 0.0122}{s^2 + 0.7845s + 0.1729}$$

مقادیر به‌دست‌آمده از شبیه‌سازی:

$$K_u = 4.00,$$
$$T_u = 1.80 \text{(ثانیه)}$$

جایگذاری در فرمول‌ها:

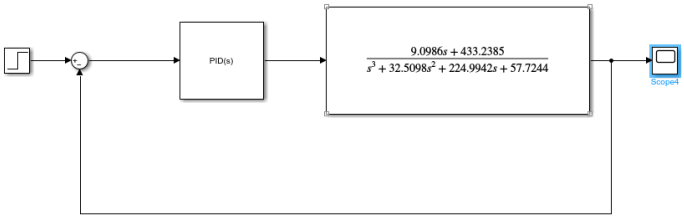
$$K_p = 0.6 \cdot 4.00 = \boxed{2.40}$$
$$K_i = \frac{2 \cdot 2.40}{1.80} = \boxed{2.67}$$
$$K_d = \frac{2.40 \cdot 1.80}{8} = \boxed{0.54}$$

۲-۲. شناسایی با PWM

در این قسمت کاملا مشابه قسمت قبل است و با این تفاوت که از PWM های متغیر استفاده کرده ایم و دیگر مثل pulse duration به موتور فرمان نمی دهیم به چه مدت روی یک PWM یکسان کار کند.

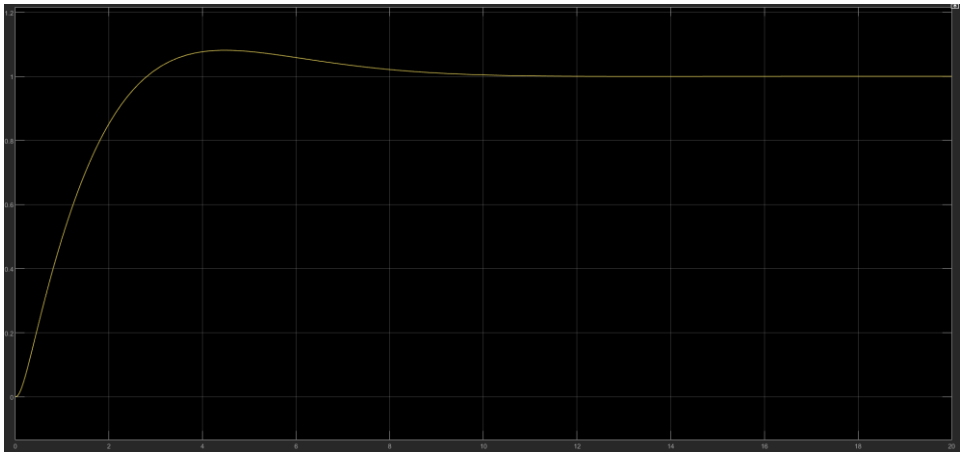
```
Transfer Function for Position (X):
  idtf with properties:

       Numerator: [-0.2871 13.0882]
     Denominator: [1 4.9252 3.8915e-10]
        Variable: 's'
         IODelay: 0
       Structure: [1×1 pmodel.tf]
   NoiseVariance: 0.5673
      InputDelay: 0
     OutputDelay: 0
       InputName: {'u1'}
       InputUnit: {''}
      InputGroup: [1×1 struct]
      OutputName: {'y1'}
      OutputUnit: {''}
     OutputGroup: [1×1 struct]
           Notes: [0×1 string]
        UserData: []
            Name: ''
              Ts: 0
        TimeUnit: 'seconds'
    SamplingGrid: [1×1 struct]
          Report: [1×1 idresults.tfest]
```

شکل ۱-۲۲ (تابع تبدیل شناسایی شده از PWM)



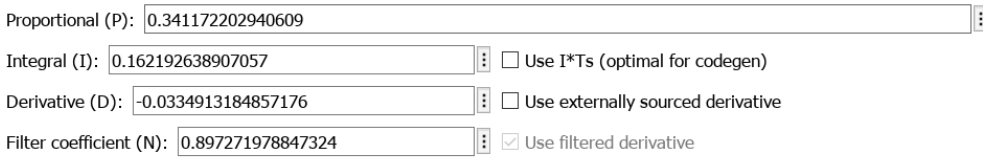
شکل ۱-۲۳ (سیستم در سیمولینک)

پاسخ سیستم به setpoint=1:



شکل ۱-۲۴ (پاسخ سیستم به ورودی ۱)

و برای ضرایب PID:



شکل ۱-۲۵ (ضرایب **PID**)

مشابها برای شناسایی درجه ۳ نیز داریم:

```
Transfer Function for Position (X):
  idtf with properties:

       Numerator: [9.0986 433.2385]
     Denominator: [1 32.5098 224.9942 57.7244
        Variable: 's'
         IODelay: 0
       Structure: [1×1 pmodel.tf]
   NoiseVariance: 0.6871
      InputDelay: 0
     OutputDelay: 0
       InputName: {'u1'}
       InputUnit: {''}
      InputGroup: [1×1 struct]
      OutputName: {'y1'}
      OutputUnit: {''}
     OutputGroup: [1×1 struct]
           Notes: [0×1 string]
        UserData: []
            Name: ''
              Ts: 0
        TimeUnit: 'seconds'
    SamplingGrid: [1×1 struct]
          Report: [1×1 idresults.tfest]
```

شکل ۱-۲۶ (شناسایی درجه ۳)

همچنین نمونه ای از فایل های txt که به متلب داده ایم:

```
152 -120 26.0 50 100
402 120 26.5 100 200
1251 120 23.1 650 300
1553 120 20.2 300 500
1606 -120 36.2 50 100
1855 -120 21.0 100 200
2704 120 6.7 650 300
3006 -120 23.0 300 500
3058 120 21.7 50 100
3308 -120 24.2 100 200
4159 120 26.1 650 300
4460 120 22.8 300 500
4513 120 32.0 50 100
4762 -120 21.8 100 200
5612 -120 25.1 650 300
5913 120 6.7 300 500
5964 -120 6.7 50 100
6214 120 3.3 100 200
7064 -120 8.0 650 300
7366 120 8.7 300 500
7416 120 6.3 50 100
7666 120 6.7 100 200
```

```
1056 0 27.0 50 100
1705 120 22.7 600 100
1808 -120 22.3 100 100
1870 -120 21.9 60 100
1960 0 19.7 50 100
2610 120 14.0 600 100
2711 0 4.7 100 100
2772 -120 6.2 60 100
2863 120 9.2 50 100
3513 -120 23.2 600 100
3614 -120 6.3 100 100
3677 0 28.3 60 100
3766 120 25.1 50 100
4416 120 16.1 600 100
4518 0 13.8 100 100
4580 120 13.5 60 100
4670 -120 11.3 50 100
5319 -120 5.7 600 100
5420 0 4.2 100 100
5481 120 3.3 60 100
5573 0 29.3 50 100
6221 -120 5.4 600 100
6322 120 6.3 100 100
```

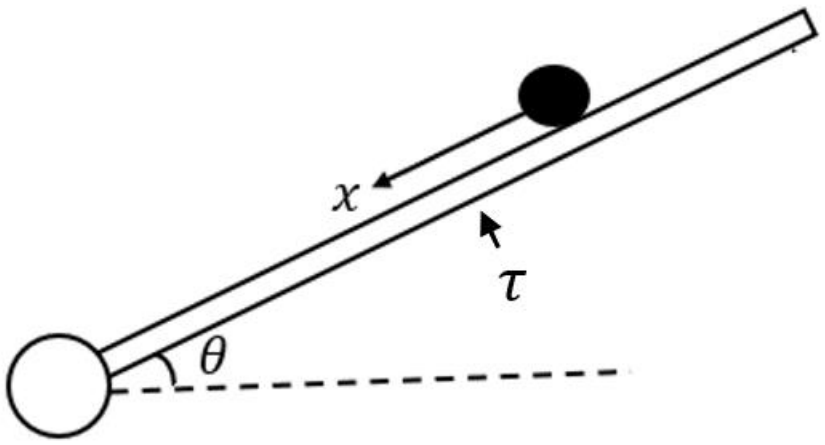
```
150 0.21 27.5
200 15.37 23.6
250 -2.77 23.5
-250 3.70 25.6
-200 5.83 23.1
-150 21.78 26.4
-120 1.75 6.7
120 -0.02 23.2
150 -2.02 23.1
200 1.30 22.0
250 -4.96 21.2
-250 -2.08 22.5
-200 2.85 27.9
-150 -4.18 29.6
-120 4.61 6.7
120 -5.45 22.0
150 -3.30 22.6
200 1.27 18.8
250 -3.22 17.3
-250 -1.76 18.7
-200 5.96 21.3
-150 -3.28 19.7
```

فصل سوم: شبیه‌سازی

مقدمه: شبیه‌سازی به‌عنوان یک مرحله واسط بین طراحی تئوریک و پیاده‌سازی عملی، نقشی حیاتی در ارزیابی عملکرد سیستم و اعتبارسنجی کنترل‌کننده‌های طراحی‌شده ایفا می‌کند. در این فصل، مدل‌های استخراج‌شده از فرآیند شناسایی سیستم در محیط MATLAB/Simulink پیاده‌سازی شده‌اند و رفتار دینامیکی سیستم تحت شرایط مختلف کنترلی مورد بررسی قرار گرفته است. هدف از این شبیه‌سازی‌ها، تحلیل دقیق پاسخ زمانی، ارزیابی پایداری، و مقایسه عملکرد کنترل‌کننده‌ها در شرایط ایده‌آل و نیمه‌واقعی بوده است تا پیش از اجرای فیزیکی، نقاط ضعف احتمالی در طراحی آشکار گردد.

۳-۱. مدل سازی به روش لاگرانژ

در این پروژه برای مدل‌سازی دینامیکی سیستم Ball and Beam از روش لاگرانژ استفاده شده است. این روش بر پایه انرژی‌های جنبشی و پتانسیل سیستم بنا شده و از معادلات لاگرانژ برای استخراج معادلات حرکت بهره می‌برد. با تعریف مختصات تعمیم‌یافته مانند زاویه میله و موقعیت توپ، انرژی کل سیستم محاسبه شده و سپس با اعمال اصل کمینه‌سازی ، معادلات دیفرانسیل غیرخطی حاکم بر سیستم استخراج می‌شوند و سپس با استفاده از مبانی خطی‌سازی به توابع تبدیل مناسب می‌رسیم. این مدل‌سازی پایه‌ای برای تحلیل پویای سیستم و طراحی کنترل‌کننده‌های دقیق فراهم می‌سازد.



شکل ۱-۲۷ (مدل شماتیک سیستم توپ و میله)

$$L = T - V$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) - \frac{\partial L}{\partial q} = Q$$

انرژی جنبشی توپ در راستای x :

$$T = \frac{1}{2}m\dot{x}^2$$

انرژی جنبشی توپ به علت چرخش دور خودش:

$$T = \frac{1}{2}J\omega^2 = \frac{1}{2}\left(\frac{2}{5}mr^2\right)\left(\frac{\dot{x}}{r}\right)^2 = \frac{1}{5}m\dot{x}^2$$

انرژی جنبشی توپ به علت گشتاور وارد به میله:

$$T = \frac{1}{2}mx^2\dot{\theta}^2$$

انرژی جنبشی میله:

$$T = \frac{1}{2}I\dot{\theta}^2$$

انرژی جنبشی کل:

$$T = \frac{7}{10}m\dot{x}^2 + \frac{1}{2}I\dot{\theta}^2 + \frac{1}{2}mx^2\dot{\theta}^2$$

انرژی پتانسیل کل:

$$V = mgx\,sin\,\theta + Mg\frac{L}{2}\sin\theta$$

$$L = T - V$$

$$\frac{\partial L}{\partial \dot{x}} = \frac{7}{5}m\dot{x}$$

$$\frac{\partial L}{\partial x} = mx\dot{\theta}^2 - mg\sin\theta$$

$$\frac{\partial L}{\partial \theta} = mgx\cos\theta + Mg\frac{L}{2}\cos\theta$$

$$\frac{\partial L}{\partial \dot{\theta}} = I\dot{\theta} + mx^2\dot{\theta}$$

معادله لاگرانژ نسبت به x:

$$\frac{7}{5}m\ddot{x} - mx\dot{\theta}^2 + mg\sin\theta = -b\dot{\theta}$$

برای اصطکاک $b\dot{\theta}$ را در نظر می گیریم

معادله لاگرانژ نسبت به θ

$$I\ddot{\theta} + mx^2\ddot{\theta} - mgx\cos\theta - Mg\frac{L}{2}\cos\theta = \tau$$

در خطی سازی تتا حول صفر است در نتیجه:

$$\cos\theta = 1 \qquad sin\theta = \theta$$

همچنین هر توان بیشتر از یک را صفر در نظر می گیریم.

پس خواهیم داشت:

$$\frac{7}{5}m\ddot{x} + mg\theta = -b\dot{\theta}$$

$$\ddot{x} = -\frac{5}{7m}b\dot{\theta} - \frac{5}{7}g\theta$$

$$I\ddot{\theta} + mx^2\ddot{\theta} + mgx + Mg\frac{L}{2} = \tau$$

$$\ddot{\theta} = \frac{\tau - mgx - Mg\frac{L}{2}}{I + mx^2}$$

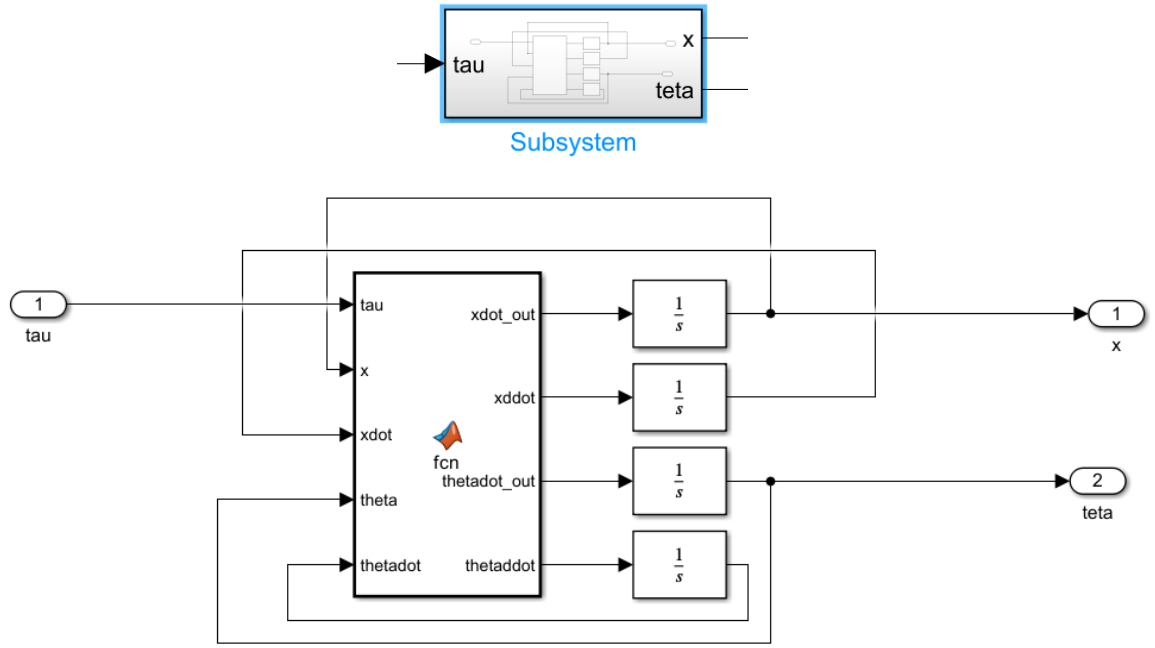
مقادیر پارامتر های سیستم توپ و میله:

M جرم توپ	0.037 kg
M جرم میله	0.092 kg
G شتاب گرانشی	9.81 ms^2/
L طول میله	0.29 m
I ممان اینرسی میله	0.000644 $kg.m^2$
b ضریب اصطکاک	0.005 N.m.s/rad

۳-۲. مدل خطی در متلب

ابتدا مدل سازی خطی را در متلب سیمولینک پیاده سازی کردیم سپس کنترل کننده هایی که در بخش طراحی کنترل کننده با روش های شناسایی مختلف به دست آورده‌ایم را بر روی این مدل اعمال می کنیم و نتیجه را مشاهده می کنیم.

مدل خطی را در یک subsystem قرار می‌دهیم با ورودی گشتاور و خروجی های قابل مشاهده مکان و به این صورت subsystem همانند تابع تبدیل مدل خطی ما خواهد بود و می‌توانیم اثر کنترل کننده را روی آن مشاهده کنیم.



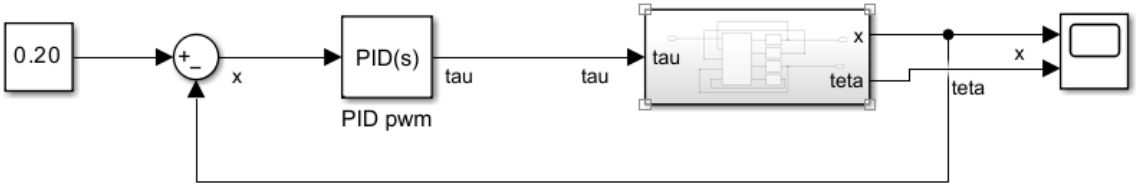
nonlinear ▸ Subsystem2 ▸ MATLAB Function

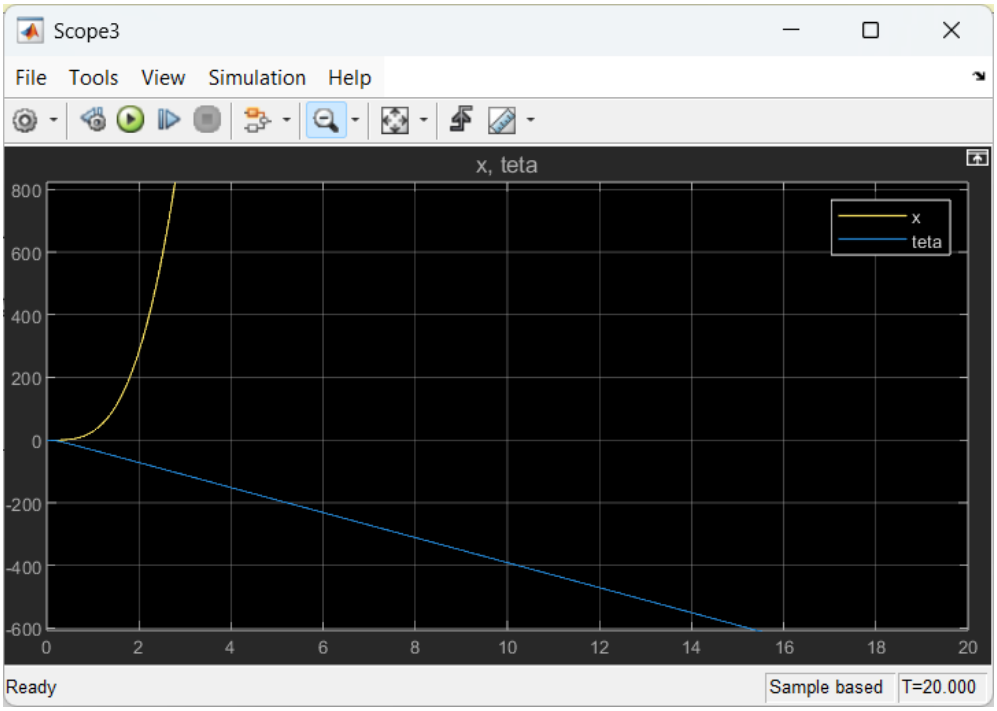
```
1   function [xdot_out,xddot,thetadot_out,thetaddot] = fcn(tau,x,xdot,theta,thetadot)
2
3   m=0.037; %ball mass
4   M=0.092; %beam mass
5   L=0.29; %beam length
6   b=0.005; %اصطکاک
7   g=9.81;
8   I= 0.000644; %ممان اینرسی
9
10  xddot=(-5*b/(7*m))*thetadot + (-5*g/7)*theta;
11  thetaddot = (tau-(M*g*L/2) -(m*g*x)) /(I + m*x*x)  ;
12
13  xdot_out=xdot;
14  thetadot_out=thetadot;
15  end
16
```

شکل ۱-۲۸ : (پیاده‌سازی مدل خطی بدست آمده از مدل‌سازی لاگرانژ در متلب)

۳-۲-۱. کنترل کننده بر اساس شناسایی PWM

پس از اعمال ضرایب PID که در بخش شناسایی بر اساس PWM بدست آورده بودیم به مدل خطی در متلب، به نتیجه مطلوبی نرسیدیم و توپ و میله رفتار ناپایدار داشتند.

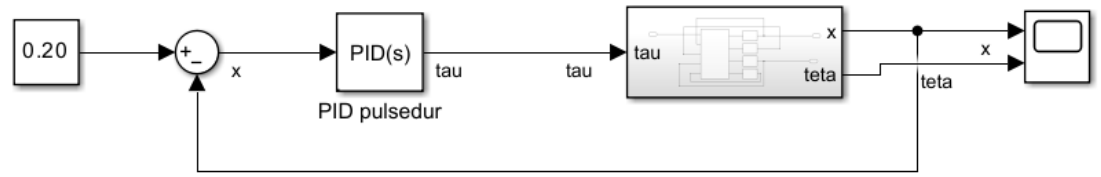


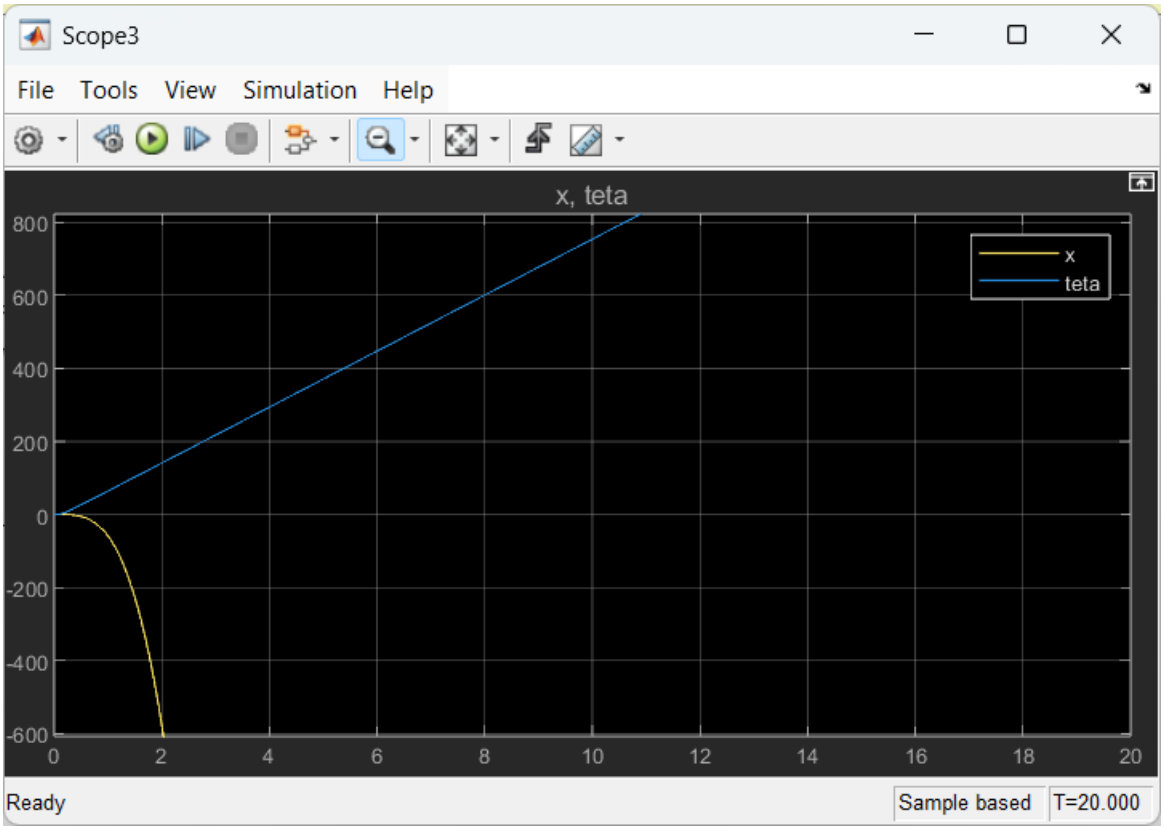


شکل ۱-۲۹ : (بلوک دیاگرام و رفتار سیستم پس از اعمال کنترل‌کننده PWM)

۳-۲-۲. کنترل کننده بر اساس شناسایی Pulse Duration

پس از اعمال ضرایب PID که در بخش شناسایی بر اساس Pulse Duration بدست آورده بودیم به نتیجه مطلوبی نرسیدیم و توپ و میله رفتار ناپایدار داشتند.

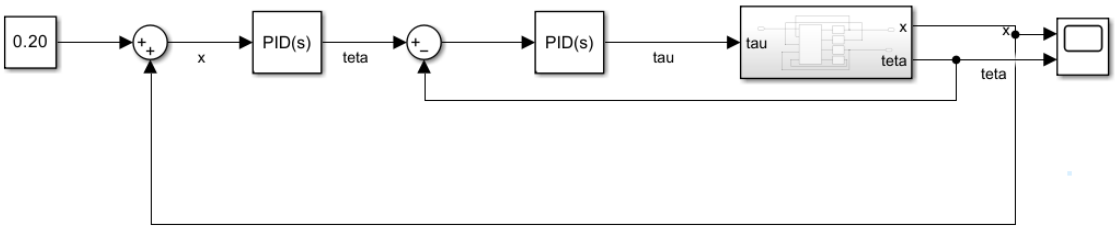


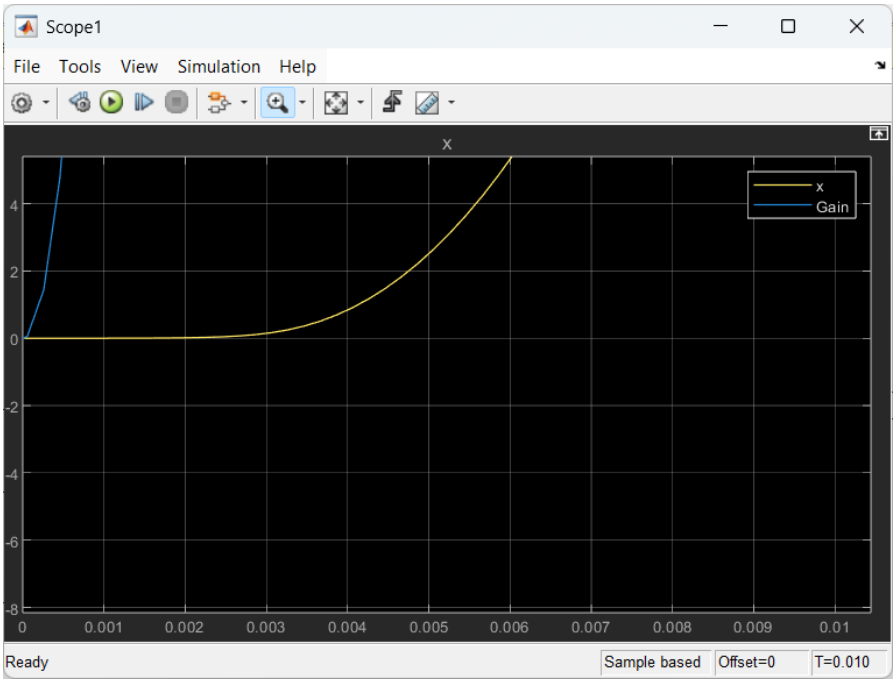


شکل ۱-۳۰ : (بلوک دیاگرام و رفتار سیستم پس از اعمال کنترل‌کننده Pulse Duration)

۳-۲-۳. کنترل کننده کسکید بر اساس شناسایی PulseDuration

پس از اعمال ضرایب PID دو کنترل کننده که در بخش شناسایی بر اساس Pulse Duration بدست آورده بودیم به نتیجه مطلوبی نرسیدیم و توپ و میله رفتار نسبت به دو مدل قبل رفتار شدیدا ناپایدارتری داشتند.

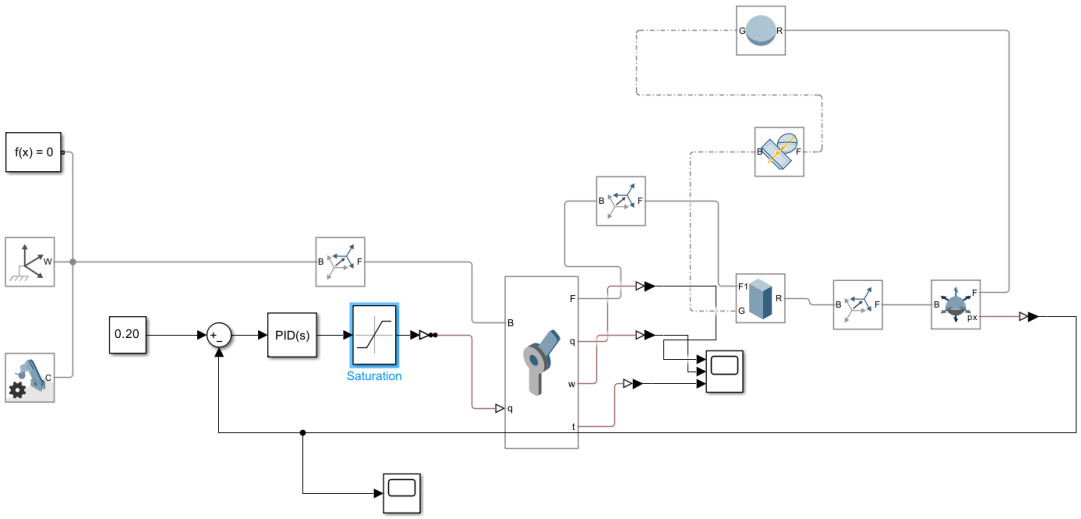


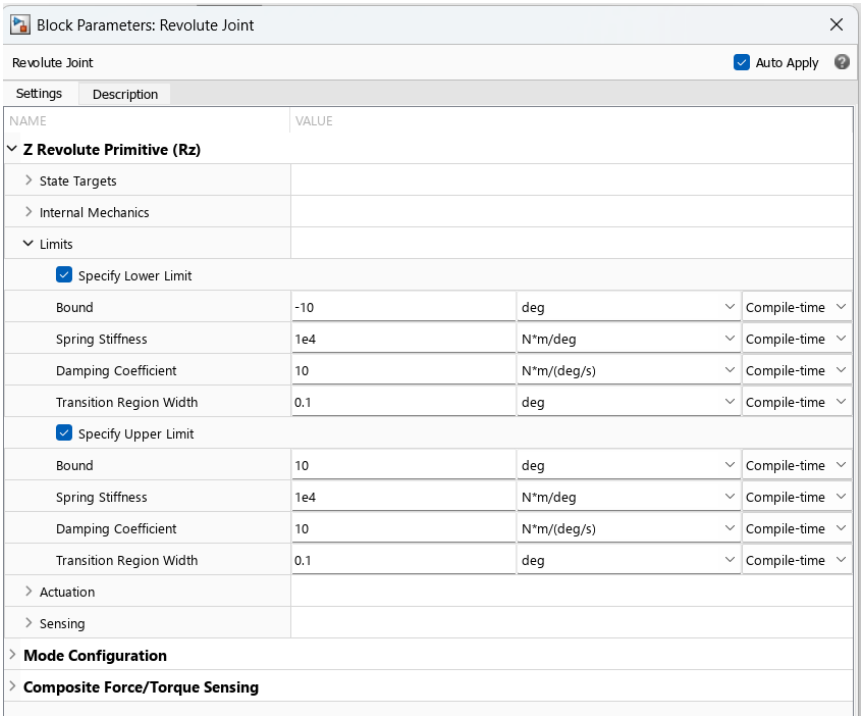


شکل ۱-۳۱ : (بلوک دیاگرام و رفتار سیستم پس از اعمال کنترل‌کننده کسکید)

۳-۳. مدل غیرخطی در Simscape

در محیط Simscape سیستم توپ و میله را شبیه سازی می‌کنیم. تغییراتی که نسبت به پروپوزال در سیستم خود داده ایم را بر روی Simscape اعمال می‌کنیم.

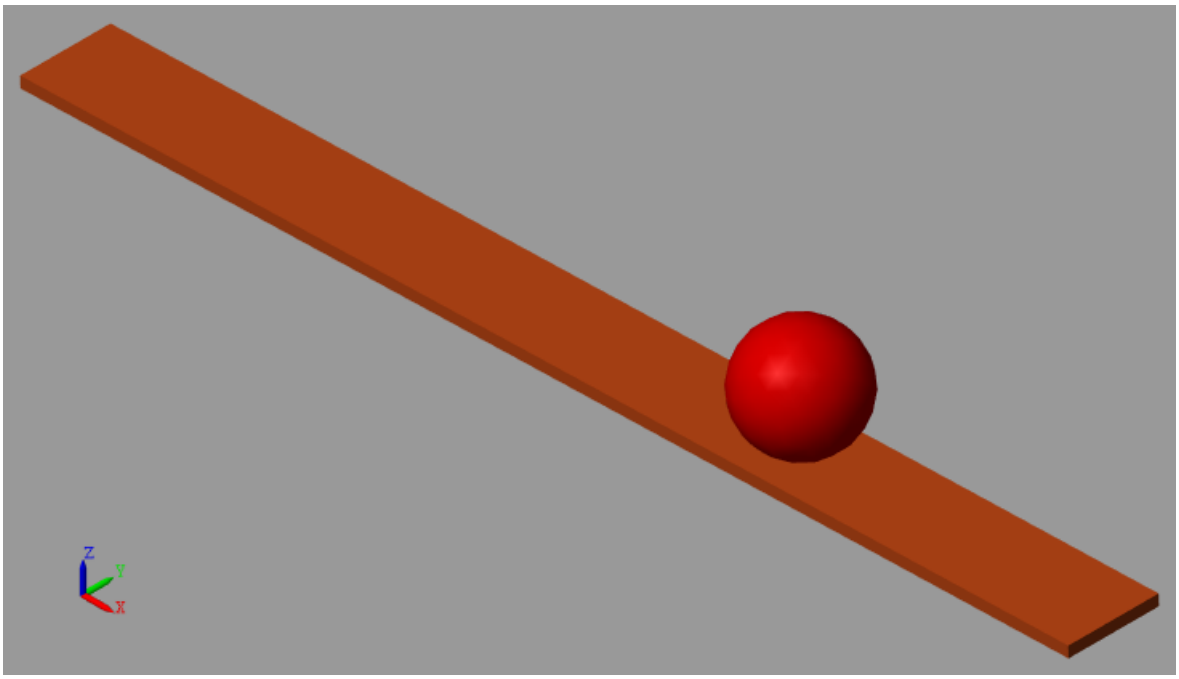




شکل ۳-۱ :(پیاده‌سازی سیستم توپ و میله در Simscape و بروزرسانی فاکتورها)

بلوک saturation را اضافه می‌کنیم تا خروجی PID را محدود کند و سیستم به واقعیت نزدیک تر

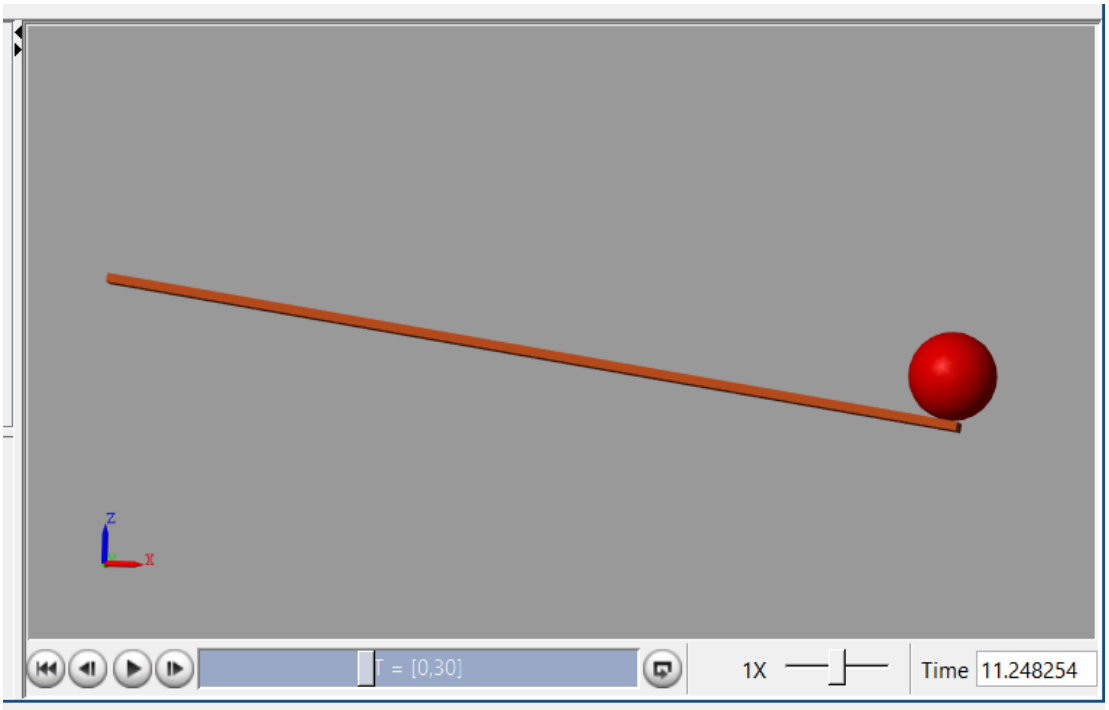
شود. لیمیت های joint را مانند واقعیت سیستم روی مثبت و منفی 5 درجه تنظیم می کنیم و ضریب اصطاک را نیز تغییر می‌دهیم. طول و جرم میله و جرم و حجم توپ را نیز مطابق سیستم واقعی خود تغییر می‌دهیم.



به این صورت مدل شبیه‌سازی غیرخطی ما آماده خواهد بود و می‌توانیم اثر کنترل کننده را روی آن مشاهده کنیم.

۳-۳-۱. کنترل کننده بر اساس شناسایی PWM

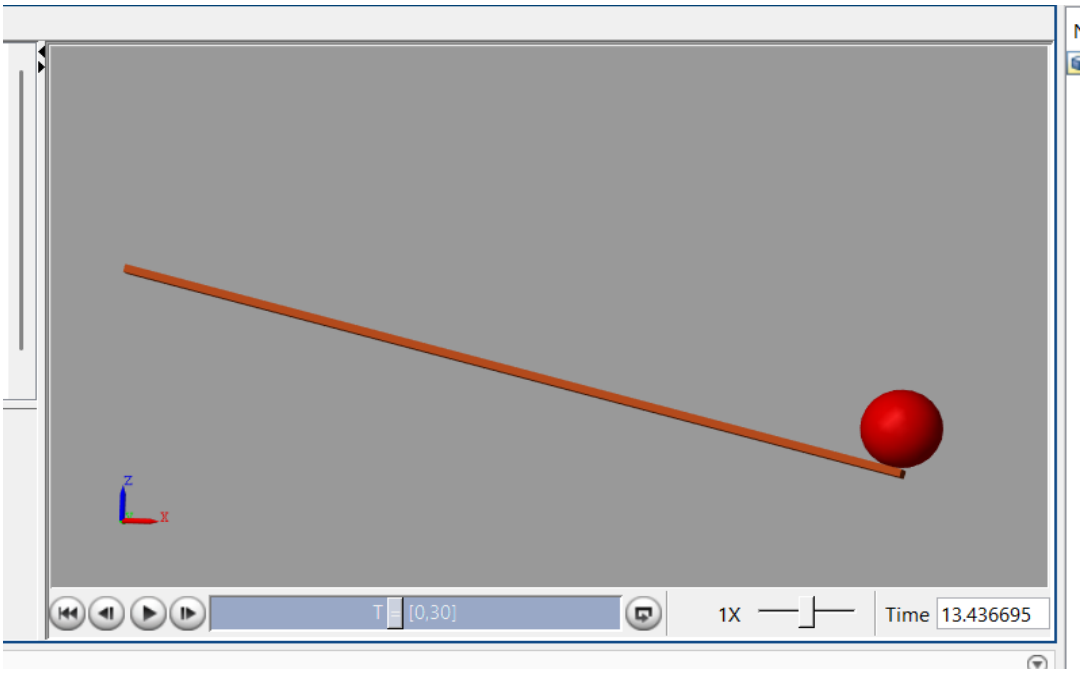
پس از اعمال ضرایب PID که در بخش شناسایی بر اساس PWM بدست آورده بودیم به نتیجه مطلوبی نرسیدیم و توپ از میله افتاد.



شکل ۱-۳۳ : (شکست کنترل‌کننده PWM در شبیه‌سازی)

۳-۳-۲. کنترل کننده بر اساس شناسایی Pulse Duration

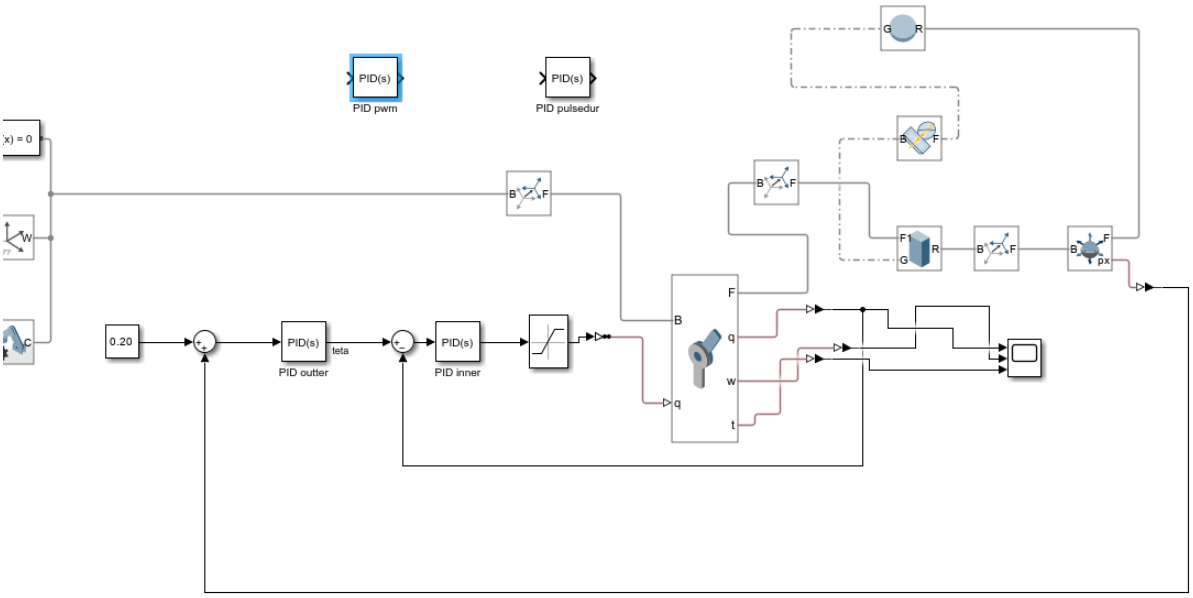
پس از اعمال ضرایب PID که در بخش شناسایی بر اساس Pulse Duration بدست آورده بودیم به نتیجه مطلوبی نرسیدیم و توپ از میله افتاد.



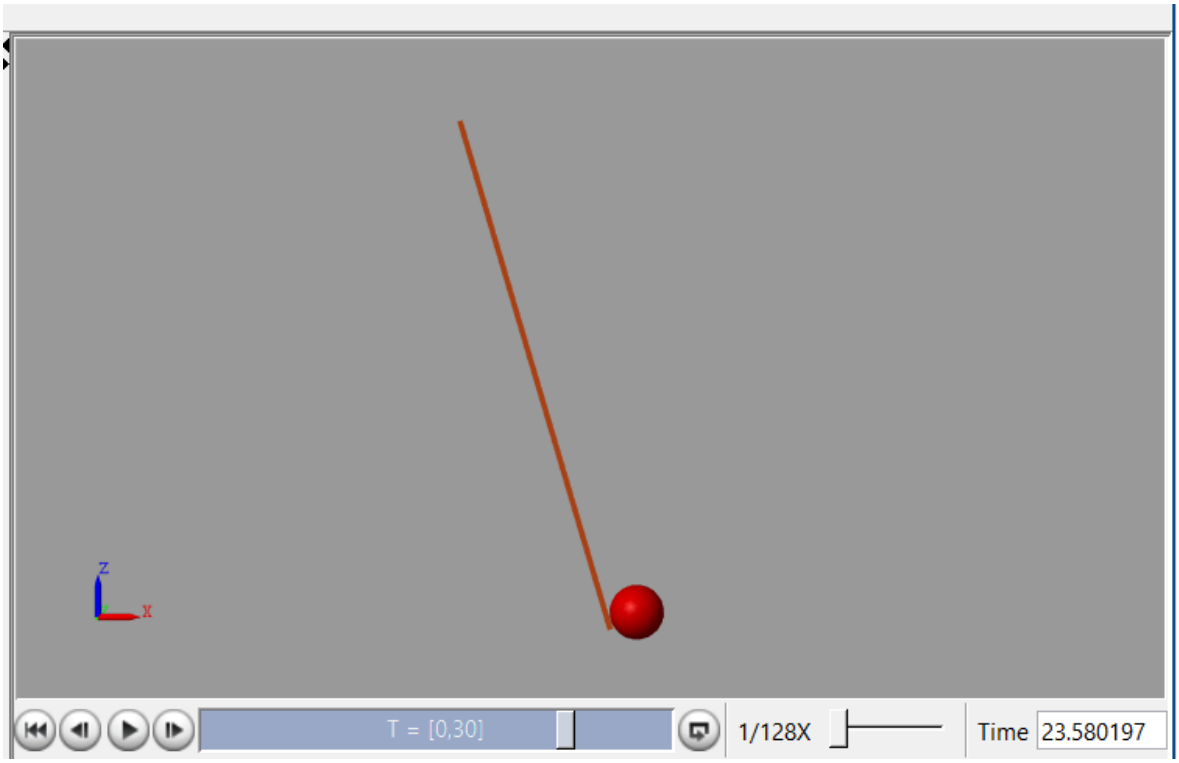
شکل ۱-۳۴ : (شکست کنترل‌کننده Pulse Durationدر شبیه‌سازی)

۳-۳-۳. کنترل کننده کسکید بر اساس شناسایی Pulse Duration

پس از اعمال ضرایب PID که در بخش شناسایی بر اساس Pulse Duration بدست آورده بودیم به نتیجه مطلوبی نرسیدیم و توپ از میله افتاد.



شکل ۱-۳۵ : (پیاده‌سازی کنترل‌کننده کسکید در Simscape)



شکل ۱-۳۶ : (شکست کنترل‌کننده کسکید در شبیه‌سازی)

۳-۴. جمع‌بندی

مشاهده شد که تمام کنترل کننده هایی که از طریق شناسایی بر روی سیستم واقعی طراحی شدند بر روی مدلسازی ها و شبیه سازی ها نتایج نا امید کننده ای داشتند. از دلایل این عدم تطابق می‌توان به غیرخطی بودن ذات سیستم توپ و میله، تفاوت های عملکردی موتور DC در برابر عملگر متلب simscape و تفاوت های شبیه سازی با سیستم واقعی (برای مثال دو سر بسته میله اجازه افتادن توپ را نمی‌دهد و اوورشوت های بالا مانع کنترل توپ نمی‌شود) اشاره کرد.

این کنترل کننده ها عملکرد بهتری روی سیستم واقعی نسبت به نتایج شبیه سازی داشتند که در فصل آینده به آنها خواهیم پرداخت.

فصل چهارم: پیاده‌سازی کنترل‌کننده و بررسی عملکرد

مقدمه: پس از طراحی و شبیه‌سازی کنترل‌کننده‌ها، مرحله پیاده‌سازی عملی نقش تعیین‌کننده‌ای در سنجش کارایی واقعی سیستم و میزان تطابق آن با مدل‌های تئوریک ایفا می‌کند. در این فصل، کنترل‌کننده‌های طراحی‌شده در بستر سخت‌افزاری شامل میکروکنترلرArduino ، موتور، سنسورها و درایور راه‌اندازی پیاده‌سازی شده‌اند و با اعمال سیگنال‌های کنترلی در شرایط عملی، عملکرد آن‌ها مورد ارزیابی قرار گرفته است. بررسی دقیق پاسخ سیستم، تحلیل پایداری و نوسانات، رفتار گذرا و ماندگار، و مقایسه داده‌های واقعی با نتایج شبیه‌سازی از جمله محورهای اصلی این فصل هستند که در راستای اعتبارسنجی نهایی طراحی کنترلی انجام گرفته‌اند.

۴-۱. کنترل با PWM

روش کنترلی این کنترل کننده اعمال PWM های مختلف به موتور بر اساس میزان خطا و اجرای آنها به مدت ۵۰ میلی ثانیه و توقف ۲۰۰ میلی ثانیه ای برای بازخوانی سنسور ها و بسته شدن حلقه کنترلی می‌باشد.

علامت مثبت یا منفی خروجی PID جهت چرخش موتور را تعیین می‌کرد و در صورتی که میله به لیمیت سوییچ ها برخورد می‌کرد جهت چرخش موتور تغییر می‌کرد. همچنین در صورتی که PID مقادیر PWM زیر ۹۰ را اعمال می‌کرد، یک if آنها را به ۹۰ تبدیل می‌کرد چون PWM های زیر ۹۰ موتور رو به حالت stall می‌برد.

در این روش در فرآیند کنترل تنها از سنسور موقعیت توپ (التراسونیک/پردازش تصویر) استفاده شد و کنترل تنها بر اساس مکان بود. یک deadband ۱ سانتی متری نیز برای کنترل توپ در نظر گرفته شد تا سیستم دچار over control نشود.

این کنترل کننده موفق نبود چون PWM های بالای 160 با 50 میلی ثانیه فعال بودن باعث چرخش 360 درجه ای شفت و بازگشت میله به زاویه ابتدایی و بی فایده بودن کنترل می‌شد. در ادامه بخش هایی از کد قرار گرفته اند.

ضرایب PID تنظیم شده:

```
double Kp = 0.45;
double Ki = 0.15;
double Kd = 0.3;

PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);
```

و در این قسمت بخش مربوط به deadband و بررسی پایداری به مدت ۳ ثانیه پیش از توقف کنترل و در نهایت دستور کنترلی به موتور:

```
  myPID.SetMode(AUTOMATIC);
  myPID.SetOutputLimits(-255, 255);
}

void loop() {
  Input = getDistance();

  if (Input >= 0) {
    error = Setpoint - Input;
    if (abs(error) <= 2.0) {
      // deadband zone
      if (!inDeadband) {
        inDeadband = true;
        deadbandStartTime = millis();
      } else {
        if (millis() - deadbandStartTime >= 3000 && !controlStopped) {
          controlStopped = true;
          applyPWM(0);
          Serial.println("Target reached. Control stopped.");
        }
      }
      if (!controlStopped)
        Output = 0;
    } else {
      inDeadband = false;
      controlStopped = false;
      myPID.Compute();
    }
    if (!controlStopped)
      applyPWM(Output);
```

```
void applyPWM(double pwmVal) {
  int pwm = abs(pwmVal);
  if (pwm > 0 && pwm < 90) pwm = 90;

  if (pwmVal == 0) {
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, LOW);
    analogWrite(PWM, 0);
    return;
  }

  if (pwmVal > 0) {
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, HIGH);
  } else {
    digitalWrite(IN1, HIGH);
    digitalWrite(IN2, LOW);
  }

  analogWrite(PWM, constrain(pwm, 0, 255));
}
```

۴-۲. کنترل با Pulse Duration

پس از دیدن نتایج کنترلی بوسیله PWM به این نتیجه رسیدیم که PWM های بالا برای کنترل مناسب نیستند و ثابت کردن مقدار PWM (PWM=120) و تغییر دادن Pulse Duration که همان مدت فعال بودن موتور می‌باشد نتایج smooth تر و پایدارتری دارد.

در این روش کنترل نیز ۲۰۰ میلی ثانیه جهت توقف بین فرمان های کنترلی و یک deadband ۱ سانتی متری قرار می‌دهیم.

این روش کنترلی نتیجه بهتری داشت اما همچنان در کنترل توپ ناکام بود، چون هنگامی که توپ به ست پوینت می‌رسید زاویه دار بودن میله باعث سر خوردن توپ می‌شد. پس ما به سراغ کنترل زاویه و مکان (کسکید) رفتیم.

۴-۳. کنترل‌کننده کسکید

کنترل کننده کسکید متشکل از یه کنترل کننده داخلی و یک کنترل کننده بیرونی است که کنترل کننده داخلی وظیفه کنترل زاویه میله و کنترل کننده خارجی وظیفه کنترل توپ را دارد.

کنترل کننده خارجی ست پوینت (مکان توپ) را می‌گیرد و ست پوینت کنترل کننده درونی رو خروجی می‌دهد، کنترل کننده درونی ست پوینت زاویه میله را دریافت می‌کند (برای مثال اگر مکان

توپ خطای منفی داشته باشد به این معنی است که توپ ابتدای میله است و کنترل کننده خارجی یک زاویه منفی به عنوان ست پوینت به کنترل کننده درونی می‌دهد تا میله به پایین خم شود و توپ به سمت ست پوینت برود) و pulse duration را خروجی می‌دهد. نکته حائز اهمیت این روش سریع تر بودن کنترل کننده درونی نسبت به کنترل کننده خارجی است.

در این روش نیز ۲۰۰ میلی ثانیه جهت توقف بین فرمان های کنترلی و یک deadband ۱ سانتی متری قرار دادیم. همچنین یک deadband ۰/۲ درجه ای نیز برای زاویه قرار دادیم.

این کنترل کننده در برخی نقاط موفق به کنترل توپ با زمان نشست بالای 20 ثانیه می‌شود.

این روش، روش نهایی و بهترین نتیجه ای بود که ما به آن رسیدیم.

یک دکمه نیز بر روی ستاپ قرار داده شده که جدا از دکمه on-off موتور وضعیت سیستم را تغییر می‌دهد، در حالت اولیه زاویه میله کنترل می‌شود و روی صفر درجه تنظیم می‌شود. در صورت فشرده شدن دکمه کنترل موقعیت توپ شروع می‌شود و در صورت دوباره فشرده شدن به حالت قبل برگشته و زاویه میله روی صفر درجه تنظیم می‌شود. در ادامه بخش هایی از کد قرار گرفته اند.

در ابتدا ضرایب PID و مقدار ثابت PWM و حداکثر مقدار pulse duration را تعیین می کنیم:

```
PID anglePID(&angleInput, &angleOutput, &anglePIDSetpoint, 13, 0.4, 3, DIRECT);
PID positionPID(&positionInput, &positionOutput, &positionPIDSetpoint, 0.095, 0.0099, 0.09, DIRECT);
const int fixedPWM = 120;
const int maxPulseDuration = 100;
```

تعیین محدوده‌های حرکتی برای میله:

```
anglePID.SetOutputLimits(-maxPulseDuration, maxPulseDuration);
positionPID.SetMode(AUTOMATIC);
positionPID.SetOutputLimits(-5.0, 5.0);
```

و تغییر وضعیت سیستم با توجه به وضعیت دکمه:

```
115  void loop() {
116    bool reading = digitalRead(buttonPin);
117    if (reading != lastReading) {
118      lastDebounceTime = millis();
119    }
120    if ((millis() - lastDebounceTime) > debounceDelay) {
121      if (reading == LOW && buttonState == HIGH) {
122        mode = !mode;
123        safetyStop();
124      }
125      buttonState = reading;
126    }
127
128    lastReading = reading;
129    if (mode != lastMode) {
130      if (mode) {
131        Serial.println("حالت کاسکید فعال: کنترل موقعیت و زاویه");
132
133      }
134       else {
135       Serial.println("حالت ساده فعال: کنترل زاویه با ست پوینت 0 درجه");
136        anglePIDSetpoint = angleSetpointSimple;
137      }
138      lastMode = mode;
139    }
140    if (mode) {
141      controlCascade();
142    } else {
143
144      controlAngleSimple();
145    }
146  }
```

در این تابع کنترل زاویه میله صورت می گیرد و ست پوینت را صفر تنظیم می کند:

```
149  void controlAngleSimple() {
150    unsigned long now = millis();
151    anglePIDSetpoint = 0.0;
152
153    if (now - lastSensorRead >= sensorInterval) {
154      float dt = (now - prevMPUTime) / 1000.0;
155      prevMPUTime = now;
156      lastSensorRead = now;
157      readMPU6050(dt);
158    }
159    angleInput = kalAngleY;
160    anglePID.Compute();
161    controlMotor(now);
162
163    if (now - lastPrint >= printInterval) {
164      lastPrint = now;
165      Serial.print("حالت ساده | زاویه: ");
166      Serial.print(kalAngleY, 1);
167      Serial.print("° | خطا: ");
168      Serial.print(anglePIDSetpoint - kalAngleY, 1);
169      Serial.print("° | خروجی PID: ");
170      Serial.print(angleOutput, 1);
171      Serial.print(" | مدت پالس: ");
172      Serial.print(abs(angleOutput));
173      Serial.println("ms");
174    }
175  }
```

و موتور برحسب Pulse Duration کنترل می‌شود:

```
220  void controlMotor(unsigned long now) {
221    if (kalAngleY > -2 && kalAngleY <10) {
222      if (!motorRunning && abs(kalAngleY - anglePIDSetpoint) > angleTolerance && now - lastControlTime >= controlInterval) {
223        if (angleOutput > 0) {
224          runMotor(fixedPWM);
225        } else if (angleOutput < 0) {
226          runMotor(-fixedPWM);
227        }
228        motorStartTime = now;
229        lastControlTime = now;
230        motorRunning = true;
231      }
232
233      if (motorRunning && now - motorStartTime >= abs(angleOutput)) {
234        stopMotor();
235        motorRunning = false;
236      }
237    } else {
238      safetyStop();
239    }
240  }
```