Analysis of Control Systems

Ines Garcia Oliver Hernandez Paul Ramirez

Tecnologico de Monterrey, Campus Guadalajara

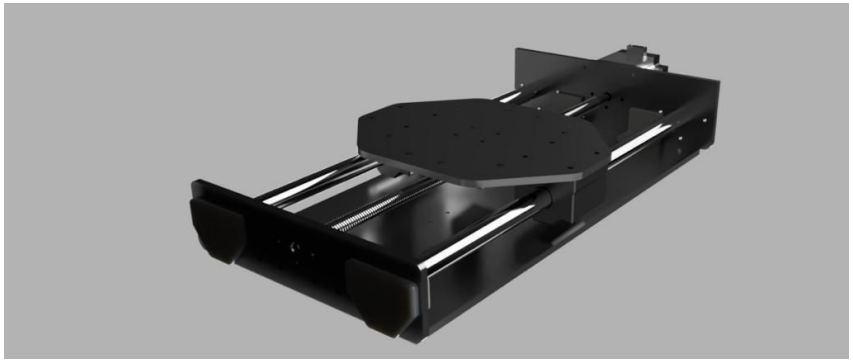
October 20, 2023

Table of Contents

Introduction

During the course we developed a linear axis as well as its control. Previously, we presented the construction of the plant, so now we shall devote this presentation mainly to the design and implementation of the PID control. Lastly, we will provide some of the needed theory to understand the decisions behind our plant.

Construction of The Plant



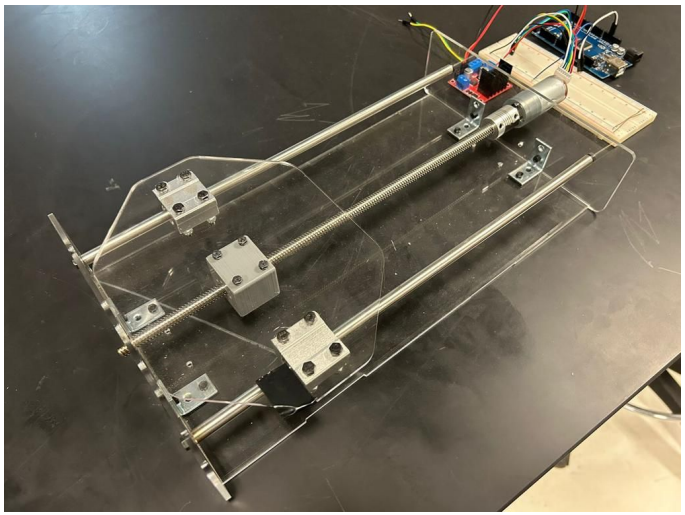


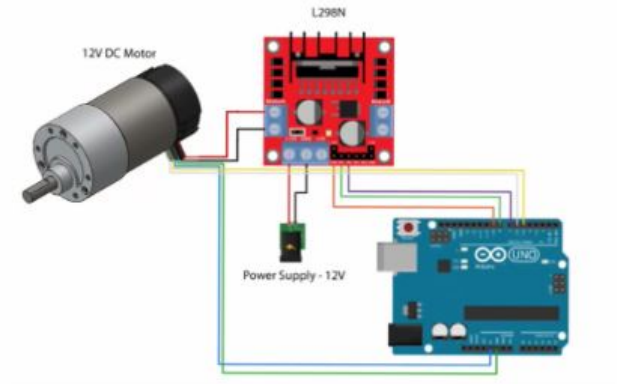
Figure: Render and Photograph of the Constructed Plant

Mechanics

- Linear Shaft: The linear shaft is a long rod-shaped component that supports linear motion and power transmission. It guides and adjusts motion
- Size and Precision: The size and precision of the linear shaft are determined by factors such as the load and specific requirements of our application.
- Screw Mechanism: Our system features a screw mechanism that converts the rotational motion of the electric motor into linear motion.
- Mounting Options: Linear shafts can be mounted in two ways: continuous support and end support. The choice depends on the load and application

Electronics

- 12-Volt Motor: Our project features a robust 12-volt motor
- L298N Motor Controller: We've integrated the L298N motor controller
- Arduino Mega: The Arduino Mega serves as our computational powerhouse
- Encoder: The motor is equipped with an encoder, delivering real-time feedback on position and velocity.



Software

- Purpose: The ‘PositionMotorControl‘ class is designed to provide advanced control for our linear axis. It allows us to set and control the position, velocity, and direction of the motor.
- Initialization: The class features methods like ‘init()‘ to initialize the motor and encoder pins, setting up the necessary parameters for control.
- Control Methods: With methods like ‘control()‘ and ‘controlPID()‘, we can fine-tune the motor’s behavior, making it responsive to our specific needs.
- Position and Velocity: The class enables us to set the target position, monitor the current position, and adjust the motor’s velocity as required.

Software

- Error Handling: It incorporates error handling mechanisms, such as the ‘error‘ and ‘errores‘ variables, for precise control and responsiveness.
- Hardware Integration: The class interacts with the motor through pins, supports encoder feedback, and leverages mechanical constants to ensure accurate motion.
- Interrupt Handling: To manage encoder changes, the class uses interrupt-driven functions, enhancing the overall efficiency of the system.
- Custom Libraries: We’ve developed custom libraries, ‘PositionMotorControl.h‘ and ‘PositionMotorControl.cpp‘, to encapsulate the functionality and simplify integration into our project.

Identification of The Plant

In our project, we implemented a system identification process to uncover the characteristics of our motor system. To achieve this, we utilized a carefully designed code that randomized the PWM signals applied to the motor while simultaneously measuring the encoder output at consistent time intervals.

- **Data Acquisition**: The first step involved collecting extensive data by applying varying PWM inputs and capturing corresponding encoder responses.
- **Regression Analysis**: With our dataset in hand, we harnessed the power of MATLAB's System Identification Toolbox. Through regression analysis, we approximated the plant function that governs the behavior of our motor system.

Derivation of The Difference Equation

Consider the PID controller with an s-domain transfer function

$$\frac{U(s)}{X(s)} = G_c(s) = K_P + \frac{K_I}{s} + K_D s$$

We can determine a digital implementation of this controller using a discrete approximation for the derivative and integration. For the time derivative, we use the **backward difference rule**

$$u(kT) = \left.\frac{dx}{dt}\right|_{t=kT} = \frac{1}{T}(x(kT) - x((k-1)T))$$

Derivation of The Difference Equation

The z-transform of the equation is then

$$U(z) = \frac{1 - z^{-1}}{T} X(z) = \frac{z - 1}{Tz} X(z)$$

The integration of $x(t)$ can be represented by the **forward rectangular integration** at $t = kT$ as

$$u(kT) = u((k-1)T) + Tx(kT)$$

where $u(kT)$ is the output of the integration at $t = kT$. The z-transform is

$$U(z) = z^{-1}U(z) + TX(z)$$

Derivation of The Difference Equation

The transfer function is then

$$\frac{U(z)}{X(z)} = \frac{Tz}{z-1}$$

Hence, the z-domain transfer function of the **PID controller** is

$$G_c(z) = K_P + K_I \frac{Tz}{z-1} + K_D \frac{z-1}{Tz}$$

The complete difference equation algorithm that provides the PID controller is obtained by adding the three terms to obtain (we use $x(kT) = x(k)$)

$$u(k) = \left[K_P + K_I T + \frac{K_D}{T}\right] x(k) - K_D T x(k-1) + K_I u(k-1)$$

Derivation of The Difference Equation

The previous derivation was retrieved from a textbook. [1] However, the used difference equation is shown below. The gains are a substitution. If desired, look up for their definitions in the project's specifications.

$$\Delta u = K_0 e(kT) + K_1 e[(k-1)T] + K_2 e[(k-2)T]$$

[1] Modern Control Systems by Dorf and Bishop

Syntonization of The Control

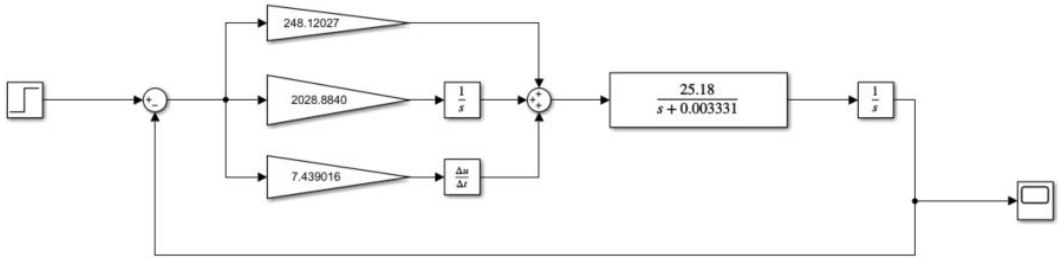


Figure: Block diagram

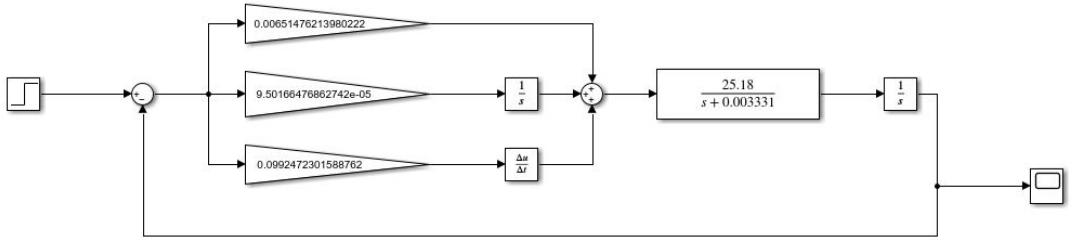


Figure: Block diagram slower

Implementation of The Control

```
void PositionMotorControl::controlPID()
{
  error = targetPosition -
    (currentPosition * 2 * 3.1416 / 493.9);
  errores[1] = errores[0];
  errores[2] = errores[1];
  errores[0] = error;
  prevPwm = motorSpeed;
  ...
```

Implementation of The Control

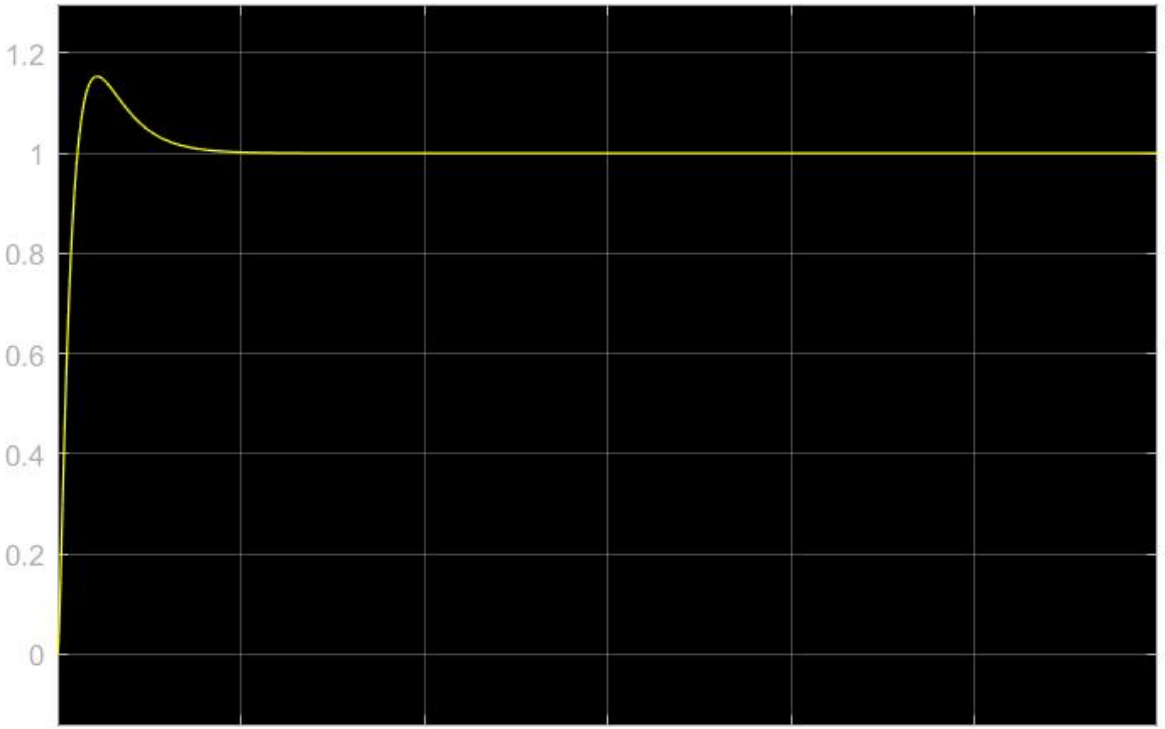
```
...
double kp = 248;
double ki = 2300;
double kd = 7.44;
double a = (kp + kd/0.02)*errores[0];
double b = (-kp + (ki*0.02) - (2*kd/0.02))*errores[1];
double c = (kd/0.02) * errores[2];

int pwmSpeed = prevPwm + a + b + c;
motorSpeed = pwmSpeed;
...
```

Implementation of The Control

```
if (error <= -1) {
  motorDirection = 1;
}
else if (error >= 1) {
  motorDirection = -1;
}
else {
  motorDirection = 0;
  pwmSpeed = 0;
}
...
```

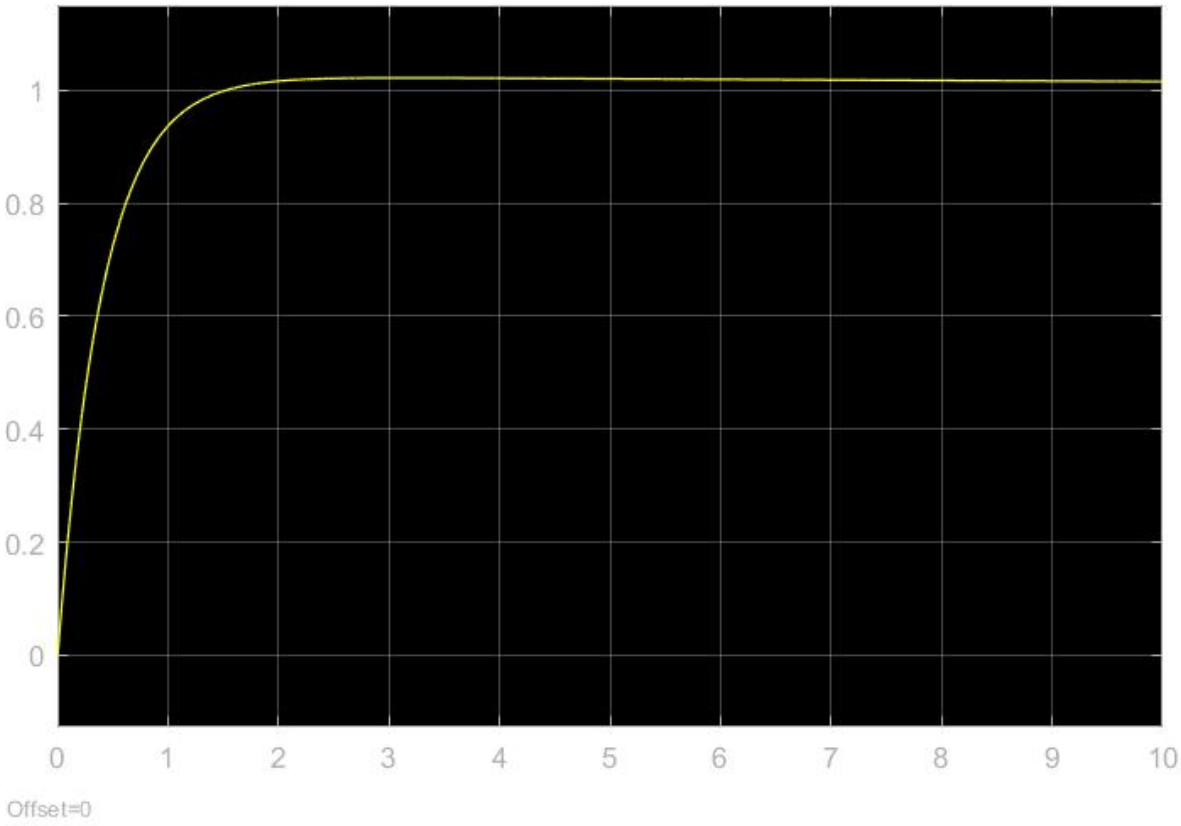
Results of The Control

Our PID control approach for the linear axis achieve a maximum impulse of just 15% and a rapid steady-state response achieved within 0.4 seconds. This combination of precise control and fast response time is a testament to the effectiveness of our PID tuning



Results of The Control

Another aproach with a maximum impulse of 1% and a slower steady-state response achieved within 4 seconds.



Conclusion

In conclusion, our project's success can be attributed to the synergy of mechanical, electrical, and software elements working together seamlessly. We've designed and constructed a precise linear axis control system that meets our specific requirements. We've developed a custom software solution using the 'PositionMotorControl' library, which enables us to achieve accurate control and position feedback. This software, along with our tailored electronics and mechanical components, brings us closer to our project objectives.

For a more detailed look at our project, including the code, hardware specifications, and documentation, please visit our project repository on GitHub:

`https://github.com/Ineso1/Linear-Axis-Control`