LABORATORIO
Ingeniería de Control

REPORTE DE PRÁCTICA
Reporte Proyecto Final

Instructores: César Cantú
Horario: 11:30 - 14:30
Nombre del equipo: E5
Integrantes:

Rodrigo Merino Silva	**A01632344**
Alexis Ruiz B.	**A00819813**
Sebastian Esquer Gaitan	**A00820249**

Evaluación:

Sección	Ponderación	A	B	C	D
Encabezado y Objetivos	5				
Introducción	10				
Contenido o desarrollo	55				
Conclusión Personal	15				
Bibliografía y Anexos	5				
Presentación	10				
Total	**100**				

Objetivos

Controlar la velocidad de giro de un motor de DC, el cual tiene incluido un encoder para medir la velocidad de giro del motor. Los pulsos del encoder son registrados en el Arduino y a su vez, el Arduino controla la velocidad del motor a través de un voltaje generado. En este caso, el Arduino generará un voltaje que llegará al puente H (el cual servirá de interfaz entre la sección de control y la sección de potencia), y se ejercerá un control a través de un PWM sobre el motor de DC.

Introducción

En este proyecto se implementará un sistema de control a un motor de DC utilizando un arduino y un encoder. Se le implementarán las variables necesarias para que el sistema PID se comporte de la manera más óptima. Así mismo se compararán las variables calculadas teóricamente con las variables calculadas automáticamente por el software MATLAB.

Contenido

Para capturar los datos del motor se realizó un código por medio de interrupciones en arduino que tiene como salida el tiempo transcurrido y las revoluciones por minuto. Gracias a que el motor de tinkercad incluye un lector de revoluciones por minuto, este se utilizó en paralelo para la comprobación de los resultados.

Para calcular las variables del PID se utilizó el método de Ziegler Nichols

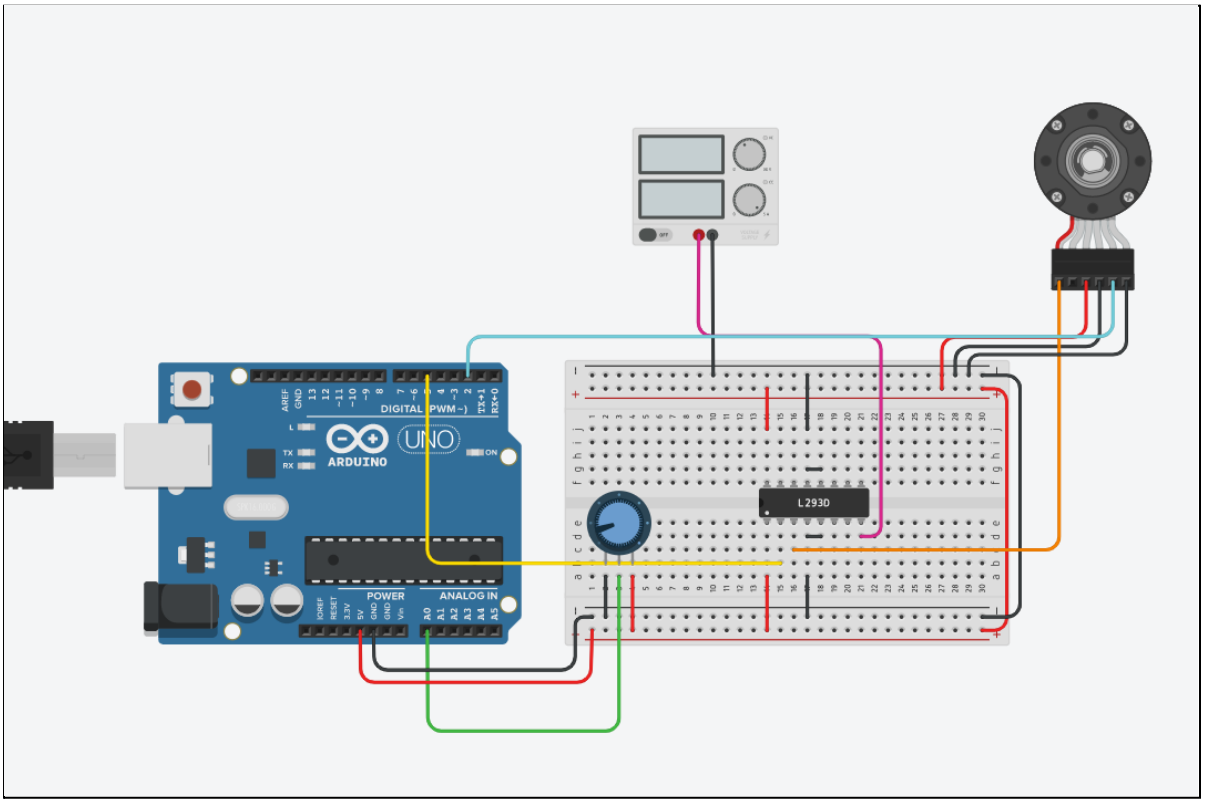

Figura 1. Circuito en tinkercad.

Diagrama de conexiones del circuito

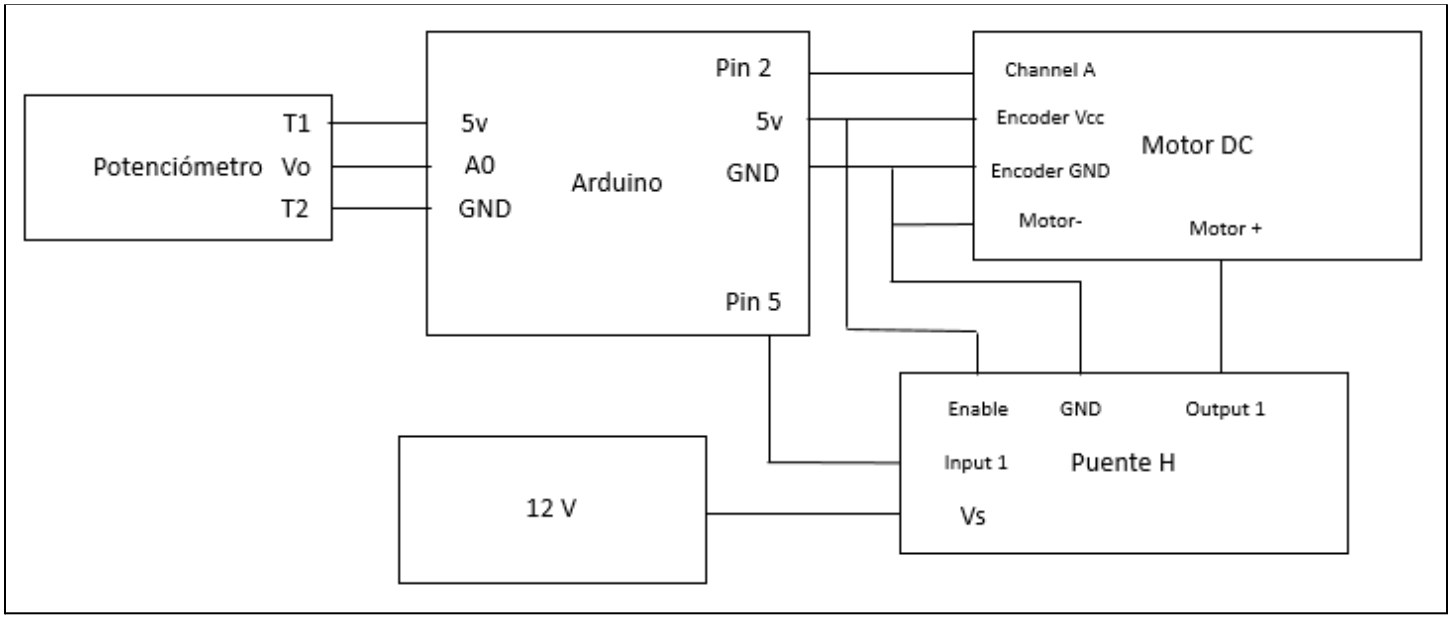

Figura 2. Diagrama de conexiones de Arduino con Motor DC

El circuito consiste en un control de un motor DC con encoder. Para variar la velocidad del motor se utiliza un potenciómetro conectado a un arduino. El microcontrolador le va a mandar una señal modulada por el ancho del pulso para controlar la velocidad. Es requisito que el motor cuente con un encoder para que el arduino pueda utilizarlo como sensor y recibir retroalimentación para el control.

Diagrama de bloques del sistema de control

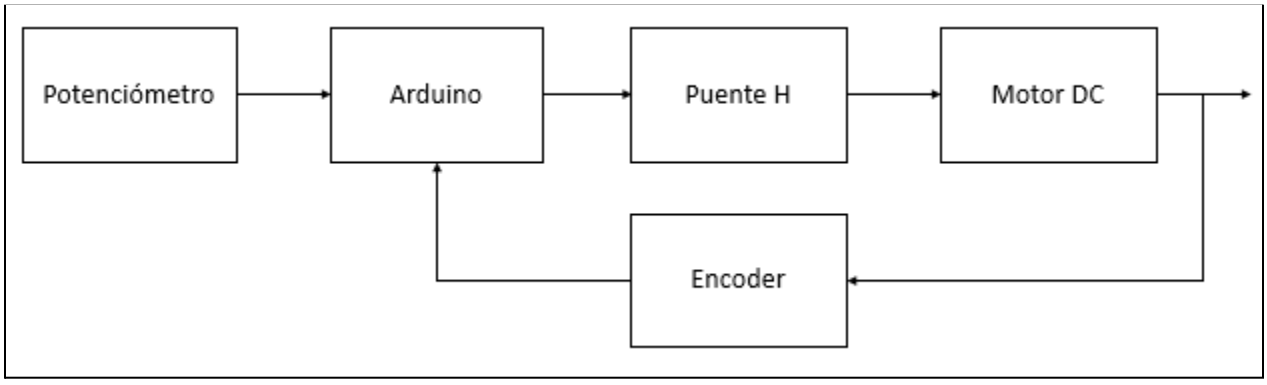

Figura 3. Diagrama de bloques de circuito controlador

Prueba Escalón (gráficas y parámetros obtenidos del sistema de primer orden)
De esta simulación obtuvimos los siguientes datos:

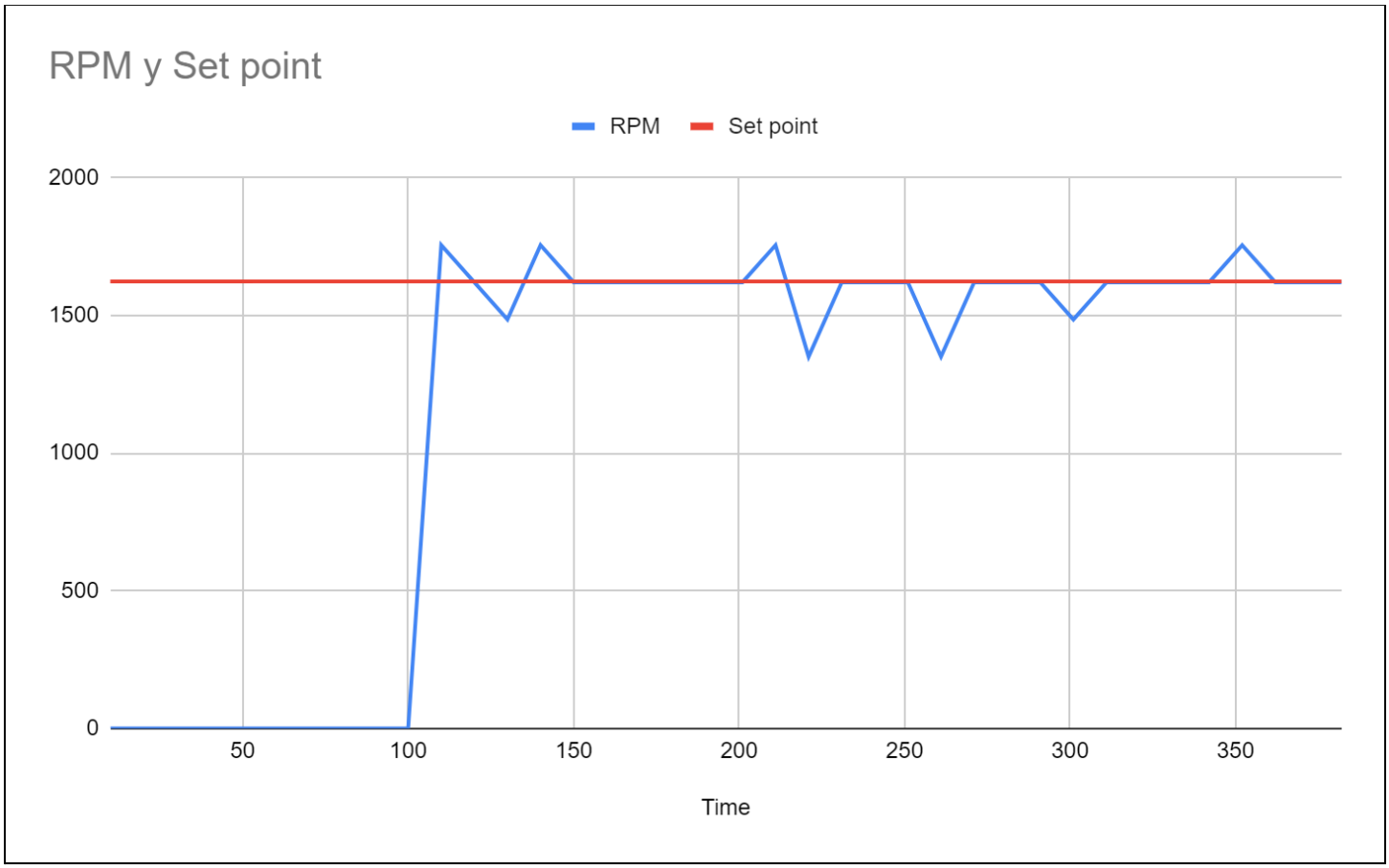

Figura 4. Gráfica de la simulación

El código se encuentra en el anexo [1]

Debido a un problema de tinkercad no se pudieron generar variables suficientemente confiables. El código se realizó de distintas maneras y con diferentes librerías sin poder obtener una buena lectura. Los datos presentados son los que tuvieron mejor confiabilidad para esta prueba.

Una vez con estos datos se migraron a MATLAB para hacer la comparación entre variables teóricas y automáticas

Tablas de parámetros de sintonía del PID obtenidos por ZN y Criterios integrales

MAX RPM	2319
STEP	0.7
Resolución	1 ms

Tabla 1. Parámetros

k	0.997967104
θ	100
τ	10

Tabla 2. Variables

Steps	Expected Speed	k
14	1623.3	115.95
15	1623.3	108.22
16	1623.3	101.45625
		108.5420833

Tabla 3. Datos arduino.

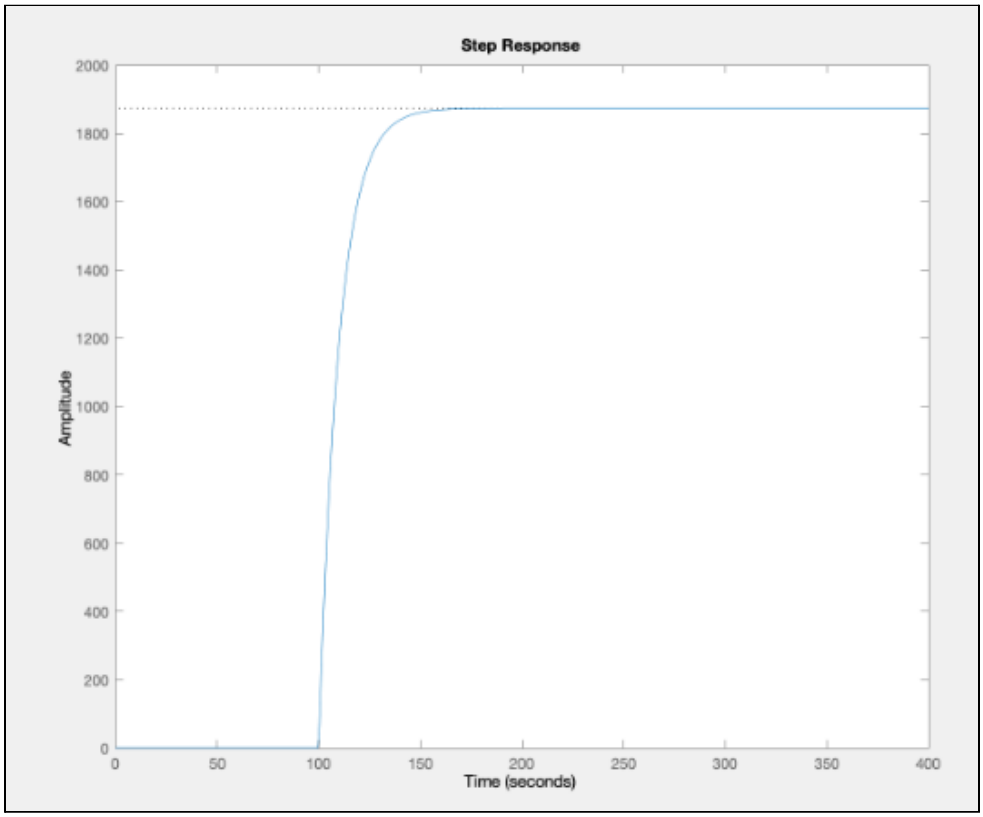

Figura 5. Respuesta al sistema con datos obtenidos de arduino.

Pruebas en Matlab

En la figura 6 podemos observar la comparación de todas las simulaciones realizadas. Corrimos dos métodos integrales, IAE e ITAE para los tres tipos de controladores P, PI y PID. Se puede apreciar como todos tienen un tiempo de respuesta parecido, pero todas llegan a tener un comportamiento diferente.

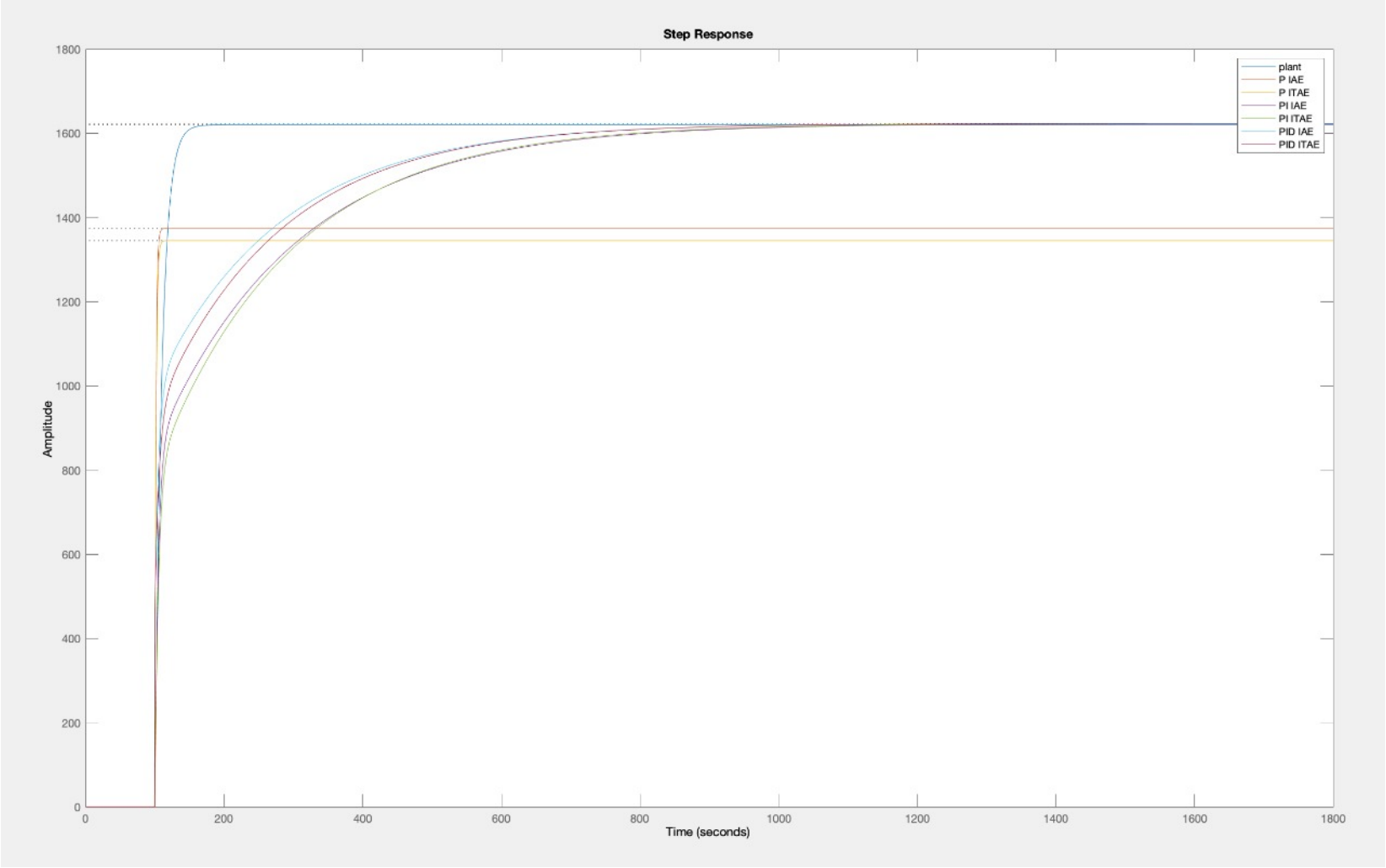

Figura 6. Comparación de métodos de sintonización

Resultados

Basándonos en los resultados podemos observar que cuando se utiliza un control puramente proporcional el tiempo de elevación es mucho más corto que utilizando otros tipos de control, pero debemos de tener en cuenta que esto pudiera tener efectos negativos en algún overshoot o en cambios drásticos considerando ruido en el sistema.
Al utilizar controladores PI observamos que el tiempo de elevación es de los más lentos y el tiempo en el que se tarda en estabilizarse es mucho comparado con los demás.

Los controladores PID obtuvieron un tiempo un poco más real y relativamente más bajo que los controladores PI y un menor tiempo de estabilización. Con estos datos en mente concluimos que para nuestro sistema es mejor utilizar un controlador PID con sintonización IAE ya que se obtuvieron buenas métricas de desempeño y al mismo tiempo se va a poder llevar a cabo un control que no sea muy variable cuando se le hagan cambios de setpoint o con pruebas de ruido.

	RiseTime	Overshoot	SettlingTime	SettlingMin	SettlingMax	Undershoot	Peak	PeakTime
plant	21.9741	0	139.1222	0.9020	0.9980	0	0.9980	443.9384
P IAE	3.3778	0	106.0164	0.7680	0.8462	0	0.8462	153.2252
P ITAE	3.7692	0	106.7125	0.7462	0.8285	0	0.8285	158.4707
PI IAE	315.2947	0	743.0815	0.9002	0.9999	0	0.9999	1.9525e+03
PI ITAE	315.2892	0	729.3588	0.9001	0.9998	0	0.9998	1.5823e+03
PID IAE	249.0036	0	645.3026	0.9000	0.9997	0	0.9997	1.4047e+03
PID ITAE	260.8540	0	651.0334	0.9001	0.9999	0	0.9999	1.6326e+03

Figura 7. Resultados de comparación

Conclusión personal (10pts)

Alexis Ruiz B

Se cumplieron los objetivos del proyecto final y se logró obtener el comportamiento esperado. Se tuvieron que ejecutar cosas de una manera diferente a la inicialmente considerado porque tuvimos problemas al trabajar con las simulaciones en tinkercad. Se obtenían valores que no deberían resultar y el comportamiento del circuito no era el esperado. Logramos realizar una prueba escalón y con esos datos graficarlos para poder obtener parámetros del sistema e identificarlo. A partir de esos datos proseguimos a trabajar en matlab y pudimos cumplir exitosamente con el proyecto.

Rodrigo Merino:

Este proyecto final recopila lo aprendido en el laboratorio de control. A pesar de los contratiempos y problemas que tuvimos con la plataforma tinkercad, los objetivos se cumplieron de manera satisfactoria. Con ayuda de herramientas aprendidas en el curso se logró pasar ese obstáculo y concluir este proyecto de la mejor manera.

Sebastian Esquer Gaitan

Este proyecto fue bastante interesante, pudimos simular el comportamiento de un motor de corriente directa utilizando tinkercad, pero debido al simulador hubo algunas carencias que nos obligaron a terminar la simulación en MATLAB. A pesar de esto logramos realizar prueba de escalón, la cual en base a las tablas de sintonización de IAE e ITAE pudimos obtener diferentes configuraciones para un controlador.

MATLAB nos permite simular de manera sencilla la respuesta del sistema a estos controladores y compararlos en una sola gráfica, algo que hubiera sido complicado con tinkercad.

A pesar de que la planta originalmente carecía de error, la respuesta del sistema es bastante instantánea, y con ayuda de los controladores logramos que la respuesta fuera un poco más lenta, lo que creo que es más útil en un motor en la vida real ya que respuestas instantáneas podrían crear comportamiento no deseado.

En comparación entre los distintos sistemas obtenidos no hubo mucha diferencia, solo los que utilizaron solamente la parte proporcional de pid, tuvieron una respuesta que no llega al nivel deseado, el resto de las pruebas difieren poco solamente en el tiempo de estabilización.

Bibliografía

MathWorks. (s.f.) Stepinfo. Recuperado el 6 de junio de 2021 de:
https://la.mathworks.com/help/control/ref/lti.stepinfo.html

Anexo:
[1] Código Arduino

```
#define encoderPin 2
#define MOTOR_IN1 5

#define MAX_SPEED 255
#define MAX_RPM 2319
#define STEP 0.7

float dt = 0.01;
int target = int(MAX_SPEED * STEP);
float targetRPM = map(target, 0, MAX_SPEED, 0, MAX_RPM);

double kP = 0.1;
double kI = 0.0;
double kD = 0.0;
double error;
double error_last = 0.0;
double P;
double I = 0.0;
double D;
int U;

int actualSpeed;

unsigned long lPrevMillis = 0;
unsigned long lCurrMillis = 0;

volatile int steps = 0;

void encoderFunc() {
  steps++;
}

void setup() {
  pinMode(encoderPin, INPUT_PULLUP);
  pinMode(MOTOR_IN1, INPUT);
  attachInterrupt(digitalPinToInterrupt(encoderPin), encoderFunc, RISING);

  Serial.begin(9600);

  analogWrite(MOTOR_IN1, MAX_SPEED * STEP);
}
```

```
void loop() {
  delay(10);
  lCurrMillis = millis();
  lPrevMillis = lCurrMillis;
  actualSpeed = steps * 115;

  Serial.print(lCurrMillis);
  Serial.print(" ");
  Serial.println(actualSpeed);

  steps = 0;
}
```

Código matlab

```
clear

MAX_RPM = 2319;
STEP = 0.7;

k = 0.997967104;
theta = 100;
tao = 10;

sys = tf(k, [tao, 1]);

% PI IAE
[kppiIAE, kipiIAE] = piIAE(k, tao, theta);
piIAEControlled = feedback(pid(kppiIAE, kipiIAE) * sys, 1);
% PI ITAE
[kppiITAE, kipiITAE] = piITAE(k, tao, theta);
piITAEControlled = feedback(pid(kppiITAE, kipiITAE) * sys, 1);
% PID IAE
[kppidIAE, kipidIAE, kdpidIAE] = pidIAE(k, tao, theta);
pidIAEControlled = feedback(pid(kppidIAE, kipidIAE, kdpidIAE) * sys, 1);
% PID ITAE
[kppidITAE, kipidITAE, kdpidITAE] = pidITAE(k, tao, theta);
pidITAEControlled = feedback(pid(kppidITAE, kipidITAE, kdpidITAE) * sys, 1);
% P IAE
pIAEControlled = feedback(pid(kppiIAE * 5) * sys, 1);
% P ITAE
pITAEControlled = feedback(pid(kppiITAE * 5) * sys, 1);

sys.InputDelay = theta;
```

```
piIAEControlled.InputDelay = theta;
piITAEControlled.InputDelay = theta;
pidIAEControlled.InputDelay = theta;
pidITAEControlled.InputDelay = theta;
pIAEControlled.InputDelay = theta;
pITAEControlled.InputDelay = theta;

opt = stepDataOptions('StepAmplitude', STEP * MAX_RPM);
hold on;
step(sys, pIAEControlled, pITAEControlled, piIAEControlled, piITAEControlled, pidIAEControlled,
pidITAEControlled, opt);
legend('plant', 'P IAE', 'P ITAE', 'PI IAE', 'PI ITAE', 'PID IAE', 'PID ITAE');
```

Código matlab (tunning vars)

```
RowNames = {'P IAE';'P ITAE';'PI IAE';'PI ITAE';'PID IAE';'PID ITAE'};
kp = [kppiIAE * 5; kppiITAE * 5; kppiIAE; kppiITAE; kppidIAE; kppidITAE];
ki = [0; 0; kipiIAE; kipiITAE; kipidIAE; kipidITAE];
kd = [0; 0; 0; 0; kdpidIAE; kdpidITAE];
tuningTable = table(kp, ki, kd, 'RowNames', RowNames);

hold on;
uitable('Data',tuningTable{:,:},'ColumnName',tuningTable.Properties.VariableNames,...
    'RowName',tuningTable.Properties.RowNames,'Units', 'Normalized', 'Position',[0, 0, 1, 1]);
```

Código matlab (PID IAE)

```
function [kP,kI,kD] = pidIAE(k, t0, T)
    kP = (1.086/k)*((t0/T)^(-0.869)) / 5;
    kI = 1 / (T)/(0.74-0.130*(t0 / T));
    kD = 0.348 * T * ((t0 / T) ^ 0.9292);
end
```

Código matlab (PID ITAE)

```
function [kP,kI,kD] = pidITAE(k, t0, T)
    kP = (0.965/k)*((t0/T)^(-0.855)) / 5;
    kI = 1 / (T)/(0.796-0.147*(t0 / T));
    kD = 0.348 * T * ((t0 / T) ^ 0.9292);
end
```

Código matlab (PI IAE)

```
function [kP,kI] = piIAE(k, t0, T)
    kP = (0.758/k)*((t0/T)^(-0.861)) / 5;
    kI = 1 / (T)/(1.02-0.323*(t0 / T));
end
```

Código matlab (PID ITAE)

```
function [kP,kI] = piITAE(k, t0, T)
    kP = (0.586/k)*((t0/T)^(-0.916)) / 5;
    kI = 1 / (T)/(1.03-0.165*(t0 / T));
end
```

Código matlab (Performance)

```
plantInfo = stepinfo(sys);
pIAEInfo = stepinfo(pIAEControlled);
pITAEInfo = stepinfo(pITAEControlled);
piIAEInfo = stepinfo(piIAEControlled);
piITAEInfo = stepinfo(piITAEControlled);
pidIAEInfo = stepinfo(pidIAEControlled);
pidITAEInfo = stepinfo(pidITAEControlled);

RowNames = {'plant'; 'P IAE'; 'P ITAE'; 'PI IAE';'PI ITAE';'PID IAE';'PID ITAE'};
RiseTime = [plantInfo.RiseTime; pIAEInfo.RiseTime; pITAEInfo.RiseTime; piIAEInfo.RiseTime;
piITAEInfo.RiseTime; pidIAEInfo.RiseTime; pidITAEInfo.RiseTime];
Overshoot = [plantInfo.Overshoot; pIAEInfo.Overshoot; pITAEInfo.Overshoot; piIAEInfo.Overshoot;
piITAEInfo.Overshoot; pidIAEInfo.Overshoot; pidITAEInfo.Overshoot];
SettlingTime = [plantInfo.SettlingTime; pIAEInfo.SettlingTime; pITAEInfo.SettlingTime; piIAEInfo.SettlingTime;
piITAEInfo.SettlingTime; pidIAEInfo.SettlingTime; pidITAEInfo.SettlingTime];
SettlingMin = [plantInfo.SettlingMin; pIAEInfo.SettlingMin; pITAEInfo.SettlingMin; piIAEInfo.SettlingMin;
piITAEInfo.SettlingMin; pidIAEInfo.SettlingMin; pidITAEInfo.SettlingMin];
SettlingMax = [plantInfo.SettlingMax; pIAEInfo.SettlingMax; pITAEInfo.SettlingMax; piIAEInfo.SettlingMax;
piITAEInfo.SettlingMax; pidIAEInfo.SettlingMax; pidITAEInfo.SettlingMax];
Undershoot = [plantInfo.Undershoot; pIAEInfo.Undershoot; pITAEInfo.Undershoot; piIAEInfo.Undershoot;
piITAEInfo.Undershoot; pidIAEInfo.Undershoot; pidITAEInfo.Undershoot];
Peak = [plantInfo.Peak; pIAEInfo.Peak; pITAEInfo.Peak; piIAEInfo.Peak; piITAEInfo.Peak; pidIAEInfo.Peak;
pidITAEInfo.Peak];
PeakTime = [plantInfo.PeakTime; pIAEInfo.PeakTime; pITAEInfo.PeakTime; piIAEInfo.PeakTime;
piITAEInfo.PeakTime; pidIAEInfo.PeakTime; pidITAEInfo.PeakTime];
performanceTable = table(RiseTime, Overshoot, SettlingTime, SettlingMin, SettlingMax, Undershoot, Peak,
PeakTime, 'RowNames', RowNames);

hold on;
uitable('Data',performanceTable{:,:},'ColumnName',performanceTable.Properties.VariableNames,...
    'RowName',performanceTable.Properties.RowNames,'Units', 'Normalized', 'Position',[0, 0, 1, 1]);
```