BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE INGENIERÍA QUÍMICA

"ESTUDIO PRELIMINAR PARA LA IMPLEMENTACIÓN DE CONTROL PID AUTÓNOMO DE NIVEL BASADO EN ARDUINO PARA UN TANQUE DE PROCESO ATMOSFÉRICO"

T E S I S D E M A E S T R Í A

QUE PARA OBTENER EL GRADO DE
MAESTRA EN INGENIERÍA QUÍMICA
PRESENTA:

I. Q. MARÍA ISABEL LÓPEZ BADILLO

ASESOR:

M. I. Q. FRANCISCO MANUEL PACHECO AGUIRRE

H. PUEBLA DE ZARAGOZA, DICIEMBRE 2014

BUAP

MAESTRÍA EN INGENIERÍA QUÍMICA
BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

ACTA DE REVISIÓN, LIBERACIÓN E IMPRESIÓN DE TESIS

Nombre del estudiante: I. Q. María Isabel López Badillo

Matrícula: 212470797

Nombre del Asesor: M. I. Q. Francisco Manuel Pacheco Aguirre

Nombre del Coasesor:

Título de la tesis:

ESTUDIO PRELIMINAR PARA LA IMPLEMENTACIÓN DE CONTROL PID AUTÓNOMO DE NIVEL BASADO EN ARDUINO PARA UN TANQUE DE PROCESO ATMOSFÉRICO

Por medio de la presente, los integrantes de la Comisión Revisora expresamos que hemos leído y revisado el manuscrito de esta tesis de maestría, por lo que **estamos de acuerdo en que se proceda con su impresión definitiva y que el estudiante presente su defensa y examen de grado en horario y lugar que se indica más abajo.**

Presidente:	Dr. Irving Israel Ruiz López	Firma:	
Secretario:	Dra. Mayra Ruiz Reyes	Firma:	
Vocal (1):	M. I. Q. Francisco M. Pacheco Aguirre	Firma:	
Vocal (2):		Firma:	

FECHA DE EXAMEN: 08/12/2014

HORA: 12:00 p.m.

DÍA DE LA SEMANA: Lunes

LUGAR: Facultad de Ingeniería Química

Contenido

Resumen

En este trabajo se estudia la aplicación de una plataforma electrónica libre basada en software y hardware denominada Arduino y su correspondencia funcional como herramienta aplicada al control de nivel de líquido, el cual, es comúnmente requerido como una rutina propia de control de los procesos en ingeniería, ya que ciertos sectores de la industria requieren establecer y procurar valores específicos en los equipos de proceso (Jyothish, 2013).

A través de este estudio, será entonces posible contar con información acertada y actualizada para esta aplicación que permita discernir en sí misma, la factibilidad de la tecnología Arduino como prototipo de controlador proporcional-integral-derivativo (PID) que en tiempo real y de manera autónoma controle el nivel de líquido en un tanque de proceso atmosférico.

El desarrollo central del algoritmo y su funcionamiento a través de la placa electrónica, ha sido denominado para fines prácticos como CoNArd-01 (controlador de nivel Arduino), que corresponde de manera funcional al conjunto de sensor ultrasónico, válvula principal de carga, válvula de descarga adicional o sobre nivel, bomba e indicadores gestionados automáticamente a partir de señales comparadas.

Siendo un diseño inicial, CoNArd-01 es altamente perfectible electrónicamente (siendo ingenieros químicos los autores) y cabe la posibilidad de acoplarlo en otras aplicaciones de control de procesos. Una de sus finalidades es la libertad de ser copiado, estudiado, adaptado o modificado de acuerdo a las necesidades y objetivos particulares de estudiantes, académicos, investigadores, etc. Su implementación en un sistema real, tiene por objetivo evaluar las características propias de un controlador, como estabilidad, calidad de respuesta y robustez, a través de las cuales se discute el desempeño de la placa Arduino en combinación de un control PID para el rubro de control automático en la ingeniería, aportando así información significativa en esta área. Los resultados auguran al microcontrolador de Arduino como una alternativa real para el estudio de los elementos de control de nivel y sus aplicaciones en ingeniería.

Lista de Figuras

Lista de Tablas

Lista de Símbolos

d_{sup}	Distancia entre el sensor ultrasónico y una superficie delante (cm).
$e(t)$	Error calculado en el tiempo t.
h	Nivel de líquido (cm).
H	Altura de instalación del sensor respecto al fondo del tanque (cm).
h_c	Nivel de líquido mínimo en el tanque TA-101 colindante al volumen de control.
h_{sp}	Nivel de líquido de referencia (*setpoint*).
K_d	Ganancia derivativa.
K_i	Ganancia integral.
K_p	Ganancia proporcional.
Q_i	Flujo de entrada (L s^{-1}).
Q_o	Flujo de salida (L s^{-1}).
t	Tiempo (s).
$u(t)$	Señal de respuesta del controlador. Acción de control.
v_{sonido}	Velocidad del sonido (1/29 cm μs^{-1}).
Δh_c	Altura del volumen de control.
τ_{Echo}	Duración del pulso de respuesta del sensor ultrasónico (µs).

Capítulo

1

Antecedentes y marco teórico

En este Capítulo se aborda el concepto de Arduino desde la perspectiva fundamental del qué es y cómo funcionan en conjunto (1) una placa electrónica basada en un microcontrolador (hardware) y (2) un lenguaje de programación Arduino con su entorno de desarrollo (software).

1.1 Antecedentes

Los avances tecnológicos y la alta competitividad industrial han llevado a diseñar e implementar procesos cada vez más eficientes desde el punto de vista energético, productivo y operacional (Desborough y Miller, 2002); comúnmente aquellos procesos en los que se implementa el control automático pueden lograr este tipo de metas (Smith y Corripio, 1991).

El estudio de nuevos dispositivos capaces de ejecutar control en aplicaciones de ingeniería es dominante en países como China, EE. UU. o Japón (Cheng y Mok, 2001). Ejemplo de estos nuevos dispositivos son los denominados sistemas de control embebido (*ECS*), los cuales, han resultado ser una opción atractiva para muchas aplicaciones de control automático. Los microcontroladores de pocos bits que se utilizan en los *ECS* han venido incrementando su capacidad y bajando su

costo dramáticamente, demostrando que tienen potencial para implementarse en aplicaciones de relativa complejidad. No obstante, en muchos casos, aún se prefiere el uso de tarjetas de control y adquisición de datos de muy alta capacidad de memoria y alto desempeño computacional a pesar de su elevado costo y de que normalmente no se producen en México (Bautista *et al.*, 2010).

Arduino es un microcontrolador de pocos bits y fácil acceso en México, en base a los trabajos desarrollados en torno a él, ha demostrado ser una opción práctica para controlar sistemas de relativa complejidad. La mayoría de los trabajos acoplan comúnmente a este microcontrolador como interface para la adquisición de datos, tanto en dispositivos autónomos como dispositivos en ejecución con software externo como *LabVIEW* o *Matlab*.

En bibliografía se encuentra el desarrollo de dispositivos electrónicos basados en Arduino, como el nominado "*Ardudrop 1.0*", que a través de varios sensores gestionados por Arduino, adquiere distintos tipos de datos ambientales (De Pablo y De Pablo, 2010). También es común encontrar aplicaciones sencillas de automatización, como el encender/apagar una autoclave de esterilización (Arízaga *et al.*, 2012) o para abrir/cerrar una válvula de gas a cuatro posiciones preestablecidas (Sethuramalingam y Karthighairasan, 2012).

La placa Arduino en ejecución con software especializado, se adapta comúnmente sirviendo de interface para la adquisición y transmisión de datos, de este modo, ha formado parte del control de motores de corriente directa (*DC*) (Aguilar y Granados, 2013; Naveenkumar y Krishna, 2013), control de nivel de agua en tanques (Giri *et al.*, 2013; Jyothish, 2013) y control en sistemas de aeronáutica (Alarcón y Lajo, 2013).

Prototipos como el denominado "*UberFridge*" (Jacobs, 2013) que controla la temperatura de fermentación de cerveza o el denominado "*SousViduino*" (Earl, 2013) para controlar la cocción de alimentos, podrían incluso ser de gran interés comercial. A pesar de tan variados proyectos en torno a la placa Arduino, se carece todavía de estudio formal enfocado al control de los procesos en ingeniería, empleando a esta placa no solo como interfaz de datos, sino también como dispositivo de control neto.

El presente trabajo desarrolla un sistema de control basado en Arduino, como opción práctica para controlar un sistema de relativa complejidad: el control de nivel en un tanque de proceso. El contar con información para su configuración e instauración, promueve una alternativa que podría ampliar las expectativas de operación en este campo de estudio: haciendo uso de este tipo de microcontroladores en el rubro de control en ingeniería.

1.2 Plataforma Arduino

Arduino es una plataforma electrónica para la creación de prototipos de código abierto, está basada en software y hardware flexibles, que además, son fáciles de usar. Arduino puede interactuar con el entorno controlando y teniendo acceso a toda una gama de dispositivos como por ejemplo, sensores, luces, motores y otros actuadores (Arduino, 2014).

Por lo general, los proyectos basados en Arduino requieren poca o ninguna habilidad programando o conocimientos de la teoría electrónica, Arduino fue diseñado para gente que no necesariamente tiene un fondo técnico en ese sentido, la mayoría de las veces estos se van adquiriendo sobre la marcha (Evans, 2011).

1.2.1 Placa Arduino (hardware)

El hardware de Arduino básicamente es una placa electrónica que se conforma de un microcontrolador soldado a un conjunto de componentes simples como resistencias, diodos, capacitores, puertos, entre otros. Una vez programado su microcontrolador, Arduino puede controlar y tener acceso a distintos dispositivos electrónicos para una variedad de proyectos.

Existen varias versiones de placas Arduino, se pueden construir a mano o comprarlas pre-ensambladas. Cada versión cuenta con un botón de reinicio (para el programa precargado en su microcontrolador), conectores de alimentación y diferentes puertos, entre ellos: puertos digitales de entrada/salida (*I/O*) y analógicos

de entrada (la versión Arduino Due es la única placa Arduino que contiene dos puertos analógicos de salida); puerto USB (*Universal Serial Bus*); puertos de alimentación. Por ejemplo, en la Figura 1, se observa la placa Arduino Mega 2560, destacando en ella algunos componentes electrónicos y puertos de comunicación comunes entre las versiones de placas Arduino.

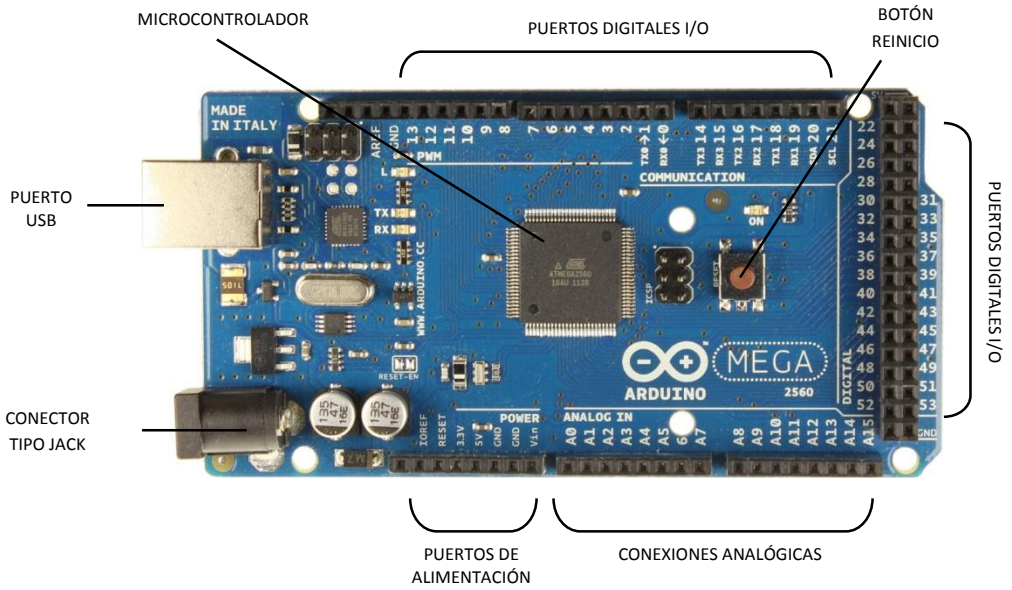


FIGURA 1 Puertos electrónicos básicos en la placa Arduino Mega 2560.

Las distintas versiones de placas Arduino pueden diferir en capacidad de memoria, número de puertos de comunicación, componentes electrónicos y modos de operación; la selección adecuada de placa Arduino a utilizar en determinado proyecto recae en estas características que tendrán que cubrir las necesidades del proyecto en cuestión. En la Tabla 1 se reportan las características básicas que destacan en cada una de las versiones de placas Arduino.

Por ejemplo, los proyectos que utilizan librerías complejas, manipulen muchas variables o datos, ocuparán mucha memoria (*Electrically Erasable Programmable Red-Only Memory*, *EEPROM*; *Static Random Access Memory*, *SRAM*; *Flash memory*) y

ésta sería en muchos casos más importante que la velocidad de la unidad central de procesamiento (*CPU*). Una librería es un código que facilita la programación en proyectos de dispositivos complejos como un sensor de humedad, una pantalla, un servomotor, etc. En determinados proyectos hay que tener en cuenta el número de puertos digitales *I/O*, ya que éstos se utilizan para conectar sensores y actuadores; también hay que tomar en cuenta la cantidad de puertos digitales *I/O* disponibles que se pueden utilizar con *PWM* (*Pulse-Width Modulation*) si se necesitara para por ejemplo, manejar varios motores con control de velocidad y varios servomotores como en un brazo robótico o proyectos más sofisticados. El dispositivo *UART* (*Universal Asynchronous Receiver-Transmitter*) mantiene y controla la comunicación bilateral serial.

TABLA 1 Principales características en placas Arduino (Arduino, 2014).

Arduino	Procesador	Alimentación (V) mínima/máxima	Velocidad CPU (MHz)	I/O analógica	IO / PWM digital	EEPROM (kB)	SRAM (kB)	Flash (kB)	USB	UART
Uno	ATmega328	5 / 7-12	16	6 / 0	14 / 6	1	2	32	Regular	1
Due	AT91SAM3X8E	3.3 / 7-12	84	12 / 2	54 / 12	-	96	512	2 Micro	4
Leonardo	ATmega32u4	5 / 7-12	16	12 / 0	20 / 7	1	2.5	32	Micro	1
Mega 2560	ATmega2560	5 / 7-12	16	16 / 0	54 / 15	4	8	256	Regular	4
Mega ADK	ATmega2560	5 / 7-12	16	16 / 0	54 / 15	4	8	256	Regular	4
Micro	ATmega32u4	5 / 7-12	16	12 / 0	20 / 7	1	2.5	32	Micro	1
Mini	ATmega328	5 / 7-9	16	8 / 0	14 / 6	1	2	32	-	-
Nano	ATmega168	5 / 7-9	16	8 / 0	14 / 6	0.512	1	16	Mini-B	1
	ATmega328					1	2	32		
Ethernet	ATmega328	5 / 7-12	16	6 / 0	14 / 4	1	2	32	Regular	-
Esplora	ATmega32u4	5 / 7-12	16	-	-	1	2.5	32	Regular	-
ArduinoBT	ATmega328	5 / 2.5-12	16	6 / 0	14 / 6	1	2	32	-	1
Fio	ATmega328P	3.3 / 3.7-7	8	8 / 0	14 / 6	1	2	32	Mini	1
Pro (168)	ATmega168	3.3 / 3.35-12	8	6 / 0	14 / 6	0.512	1	16	-	1
Pro (328)	ATmega328	5 / 5-12	16	6 / 0	14 / 6	1	2	32	-	1
Pro Mini	ATmega168	3.3 / 3.35-12	8	6 / 0	14 / 6	0.512	1	16	-	1
		5 / 5-12	16							
LilyPad	ATmega168V	2.7-5.5	8	6 / 0	14 / 6	0.512	1	16	-	-
	ATmega328V									
LilyPad USB	ATmega32u4	3.3 / 3.8-5	8	4 / 0	9 /4	1	2.5	32	Micro	-
LilyPad Simple	ATmega328	2.7-5.5 /2.7-5.5	8	4 / 0	9 / 4	1	2	32	-	-
LilyPad SimpleSnap	ATmega328	2.7-5.5 / 2.7-5.5	8	4 / 0	9 / 4	1	2	32	-	-

1.2.1.1 Microcontrolador de Arduino

El microcontrolador es el elemento más importante en la placa, es donde se instala y ejecuta el código de programa que se haya elaborado. Un microcontrolador es un circuito integrado (chip) que incluye en su interior las tres unidades funcionales de una computadora: *CPU*, memoria y puertos de *I/O* para su interacción con el entorno. En la actualidad existen muchas compañías (Motorola, Microchip, Atmel, Intel, etc.) que fabrican microcontroladores y por lo tanto muchos tipos con características diferentes.

Cada placa Arduino cuenta con al menos un microcontrolador de la marca Atmel, siendo los más usados el ATmega328, ATmega32u4, ATmega168. Existen diferentes tipos de memorias en el microcontrolador usado por Arduino: *Flash*, *SRAM* y *EEPROM*. La memoria *Flash* es donde Arduino almacena el *sketch*, siendo éste el nombre que le ha dado Arduino al programa que compila y ejecuta. La *SRAM* es el espacio donde los programas almacenan y manipulan las variables y los datos cuando el programa es ejecutado, cuando Arduino pierde alimentación, esta información es eliminada. Hay que optimizar al máximo cada programa, ya que si la *SRAM* se quedara sin espacio, lo programado fallaría de forma imprevista. La *EEPROM* puede ser utilizada para almacenar información a largo plazo, si Arduino pierde información, la información almacenada permanecerá.

El único modelo de placa Arduino que es reparable fácilmente, cambiando el microcontrolador, es Arduino Uno, en el resto de las versiones, los microcontroladores están soldados directamente a la placa, por lo que no hay posibilidad de repararlos en caso de avería. Al no poder retirar el microcontrolador del circuito, si lo que se necesita es programarlo, esto puede hacerse por su conector *ICSP* (*In Circuit Serial Programming*), lo que realmente se programa es el gestor de arranque (*bootloader*) del microcontrolador.

1.2.1.2 Conexiones de alimentación

Las placas Arduino pueden alimentarse de tensión para su funcionamiento a través de su puerto *USB*, conector tipo *jack* y puertos de alimentación. El puerto *USB* (mini

o regular) además de recibir tensión para alimentar la placa, carga el *sketch* en la memoria *Flash* del microcontrolador desde el software de Arduino. El conector de alimentación recibe tensión a partir de un adaptador *AC/DC* o de baterías (3.3-12V). Las baterías también pueden ser conectadas utilizando los puertos de alimentación GND y Vin, donde se conectan ánodo y cátodo respectivamente.

Si el puerto Vin no está siendo utilizado como entrada de alimentación, a través de él se puede acceder a la tensión suministrada a la placa por puerto USB o conector tipo *jack*. Los puertos de alimentación 5V, 3.3V y GND funcionan de manera similar, pero proporcionando los voltajes que sus respectivos nombres indican. El puerto de alimentación IOREF proporciona el voltaje con la que opera el microcontrolador. Esta versatilidad de alimentación permite alimentar algunos o todos los dispositivos que se necesiten ejecutar con la placa Arduino.

1.2.1.3 Puertos analógicos

La mayoría de los sensores necesitan conectarse a puertos de señal analógica. Una señal analógica es aquella que toma todo valor intermedio entre dos valores cualesquiera, es decir, para pasar de un nivel a otro la señal pasa por todos los valores intermedios (Donate, 1995).

En la gran mayoría de las placas, los pines analógicos solamente pueden ser de entrada, la placa Arduino Due es la única versión que tiene dos salidas analógicas porque cuenta con dos convertidores digital a analógico (*DAC*), esta ventaja sobre las otras se presta a por ejemplo, proyectos que necesitan reproducir sonido de calidad. Hay forma de simular salidas analógicas través de los puertos digitales *PWM*. El voltaje de referencia para las entradas analógicas puede modificarse utilizando el puerto *AREF*.

1.2.1.4 Conexiones digitales

Los puertos digitales de la placa Arduino pueden ser configurados de manera individual para funcionar como entradas o salidas (*I/O*) de señales digitales. Una

señal digital varía en un intervalo entre el cual no toma valores intermedio, es decir, solo posee dos niveles posibles de estado: tensión más alta y tensión más baja (Donate, 1995). Dentro de los puertos digitales *I/O*, algunos son reservados para funciones programables especializadas, por ejemplo:

a. RX y TX, son utilizados para recibir (RX) y transmitir (TX) datos serie, esta comunicación serial permite, por ejemplo, conectarse a otras placas y compartir información.
b. PWM, proporciona salidas analógicas, esto se logra con modulación por ancho de pulso (*PWM*), la cual, es una técnica que modifica el ancho del pulso de una señal digital en encendido (tensión más alta) y con suficiente velocidad para que el voltaje de salida sea analógico (realmente no es un voltaje analógico, sino que se simula haciendo repeticiones). El pulso que proporcionan las placas Arduino son de tipo *TTL* () con 5V como tensión más alta y a una frecuencia de aproximadamente 490Hz.
c. LED, es un puerto que se encuentra conectado con un diodo emisor de luz (*LED*) de la placa. Cuando se le asigna un valor de tensión alta, el *LED* se encenderá y manera contraria, si se le asignara un valor de tensión baja el *LED* se apagará.
d. RESET, imita el funcionamiento del botón de reinicio si suministramos un valor de tensión baja.

1.2.2 Lenguaje y entorno de programación Arduino (software)

El software de Arduino es multiplataforma, funciona en Windows, Macintosh OSX y Linux. Los proyectos basados en Arduino pueden ejecutarse sin necesidad de conectarse a una computadora, lo cual brinda cierta autonomía al sistema; aunque cabe mencionar que se tiene la posibilidad de comunicarse con diferentes paquetes de software externo ejecutados desde la computadora, como por ejemplo: Matlab, LabVIEW, Flash, Processing, MaxMSP (Arduino, 2014).

Programar el microcontrolador de Arduino es muy sencillo y accesible, además se cuenta con la ayuda de la comunidad Arduino, lo que lo hace aún más fácil. Dicha comunidad está conformada por los usuarios de Arduino que comparten resultados

y habilidades adquiridas producto del desarrollo de proyectos personales en torno a la placa, de esta manera la comunidad se enriquece a partir de la producción de conocimiento (WordPress, 2009).

El microcontrolador de la placa Arduino se programa mediante (1) el lenguaje de programación Arduino, el cual, está basado en lenguaje de alto nivel C/C++ y (2) el entorno de desarrollo Arduino, software que se puede descargar en forma gratuita en la página oficial de Arduino. Todo *sketch* en Arduino se compone de las dos funciones principales siguientes:

```
void setup(){     ..................................  Fn. (a)
}
void loop(){      ..................................  Fn. (b)
}
```

La función `setup()` contiene un código de configuración inicial de variables, puertos, librerías, bits, etc. y solo se ejecuta una sola vez cuando el *sketch* da inicio ya sea por alimentación de tensión o reinicio en la placa Arduino. La función `loop()` se ejecuta después de `setup()` y se mantiene ejecutando en manera de ciclo hasta que la placa Arduino se deje de energizar o reinicie; dentro de esta función Arduino hace uso de datos a través de sus puertos. El programa que ejecuta Arduino se puede dividir en tres partes principales: estructura, valores (constantes y variables) y operaciones (Arduino, 2014).

1.2.2.1 Estructuras

Las estructuras de control son instrucciones que nos permiten tomar decisiones y hacer diversas repeticiones en el programa en base a parámetros leídos a través de los puertos o calculados por el mismo programa en base a dichas lecturas. Para su construcción, es necesario hacer uso de distintos operadores y escribir en su correcta sintaxis.

La información respecto al lenguaje de programación Arduino, es bastante amplia y compleja, no obstante, en las Tablas 2, 3 y 4, se presenta parte del lenguaje de

programación para estructuras básicas y por lo tanto comunes del programa, el acompañamiento de un ejemplo ayuda a su entendimiento.

TABLA 2 Estructuras de control del lenguaje de programación Arduino.

Estructura de control	Descripción	Ejemplo
`if` (si)	Se utiliza en conjunción con un operador de comparación para determinar si se ha alcanzado cierta condición.	`if (var > 50) //acción si es verdadero;` `if (x == y) //acción si es verdadero;`

`if...else`
(si...entonces)
	Permite mayor control sobre el código a diferencia de `if`, al permitir múltiples condiciones.	`if (pinFiveInput < 500) {`
		`  //acción A si es verdadero;`
		`}`
		`else {`
		`  //acción B si es falso;`
		`}`

`for`
(por)
	Útil para cualquier operación repetitiva.	`for (int x=0; x<100; x++){`
		`  println(x); //imprimirá de 0 a 99`
		`}`

`switch case`
(interruptor en caso de)
	Controla el flujo de los programas al especificar diferentes códigos a ejecutar en varias condiciones.	`switch(var){`
		`  case 1:`
		`  //acción cuando var = 1;`
		`  brake;`
		`  case 2:`
		`  //acción cuando var = 2;`
		`  brake;`
		`}`

`while`
(mientras)
	Es un bucle que se repetirá de forma continua hasta que una expresión declarada se convierte en falsa.	`while(var < 200){`
		`  //se repetirá 200 veces`
		`  var++;`
		`}`

`do...while`
(hacer... mientras)
	El loop do funciona de la misma manera que el `while`, con la excepción de que la condición se comprueba al final del bucle, por lo que éste se ejecutará al menos una vez.	`do {`
		`  delay(50);`
		`  x = readSensor();`
		`} while (x < 100);`

`break`
(descanso)
	Para salir de un do, `for` o `while`, sin pasar por el estado normal de bucle. También se usa para salir de la sentencia `switch`.	`for (x=0; x < 255; x++){`
		`  digitalWrite(PWMpin, x);`
		`  sens = analogRead(sensorPin);`
		`  if(sens < 100){`
		`  x=0;`
		`  break;`
		`  }`
		`}`

`continue`
(continuar)
	Salta el resto de la iteración actual de un bucle (do, `for` o `while`). Continúa con la expresión condicional del bucle y continua con las iteraciones posteriores.	`for (x=0; x<255; x++) {`
		`  if (x>40 && x<120){`
		`  continue; //salta estos valores`
		`  }`
		`  digitalWrite(PWMpin, x);`
		`}`

TABLA 2 Estructuras de control del lenguaje de programación Arduino *(continuación)*.

Estructura de control	Descripción	Ejemplo

`return`
(retorno)
	Termina una función y devuelve el valor de una función a otra función que se ha llamado, si se desea.	`int checkSensor(){`
		`  if(analogRead(0)>400){`
		`  return 1;`
		`  else{`
		`  return 2;`
		`  }`
		`}`

`goto`
(ir a)
	Transfiere el flujo del programa a un punto marcado en el programa.	`if(analogRead(0)>250) goto bailout;`
		`//...;`
		`bailout:`
		`//...;`

TABLA 3 Sintaxis del lenguaje de programación Arduino.

Sintaxis	Aplicación	Ejemplo
`;` (punto y coma)	Se utiliza para terminar una declaración.	`int a = 13;`

`{}`
(llaves)
	Los principales usos son en funciones, loops, y declaraciones condicionales (ver Tabla 2)	while (time < 200) {
		a = 4;
		}

`//`
(comentario)
	Para una sola línea del programa, cualquier escrito después de // es un comentario, éste es ignorado por el compilador y no ocupan espacio en la memoria.	`x = 5; // comentario 1`
		`// comentario 2`
		`// comentario 3`

`/**/`
(comentario multilínea)
	Se utiliza para un comentario o comentarios de mayor extensión a una línea del programa.	`x = 5; /*comentario 1`
		`comentario 2`
		`comentario 3*/`

`#define`
(definir)
	Permite dar un nombre a un valor constante antes de compilar el programa y así no ocupar espacio en la memoria con esa constante.	`#define ledPin 3 /*el compilador`
		`reemplazará cualquier mención`
		`de ledPin con el valor de 3 en`
		`tiempo de compilación*/`

`#include`
(incluir)
	Se utiliza para incluir bibliotecas fuera del *sketch*.	`#include <avr/pgmspace.h> /*de este`
		`modo el programa incluye la`
		`biblioteca especificada*/`

TABLA 4 Operadores aritméticos, de comparación y compuestos del lenguaje de programación Arduino.

Operador aritmético	Aplicación	Ejemplo
= (asignación)	Evalúa cualquier valor o expresión a su lado derecho y lo almacena en la variable a su izquierda.	`var = analogRead(0); /*almacena la entrada del pin analógico 0 en var*/`

+ (suma)
- (resta)
* (multiplicación)
/ (división)
	Respectivamente, suma, resta, multiplica y divide los operandos.	y = y + 3;
		x = x – 7;
		i = j * 6;
		r = r / 5;

% (módulo)	Calcula el residuo cuando un entero se divide por otro. *residuo = dividendo % divisor.*	`x = 7 % 5;  // x = 2`
		`x = 9 % 5;  // x = 4`
		`x = 5 % 5;  // x = 0`
		`x = 4 % 5;  // x = 4`

Operador de comparación	**Aplicación**	**Ejemplo**

== (igual a)
!= (diferente de)
< (menor que)
> (mayor que)
<= (menor o igual que)
>= (mayor o igual que)
	Se utilizan en conjunto de la función `if` para saber si se ha alcanzado cierta condición.	`if(x != 120)`
		`  digitalWrite(LEDpin, HIGH);`
		`if (y > 120)`
		`  digitalWrite(LEDpin, LOW);`

Operador *booleano*	**Aplicación**	**Ejemplo**

&& (y)
|| (o)
! (no)
	Pueden ser utilizados dentro de la condición de una sentencia `if`.	`if(digitalRead(2)==HIGH && digitalRead(3)==HIGH) //... ; //es verdad si ambas entradas son HIGH`
		`if (x > 0 || y > 0) //... ; //es verdad si x o y son mayores de 0`
		`if(!x) //... ; //es verdad si x es falsa (p.e. x = 0)`

++ (incremento)
-- (decremento)
	Incrementa o decrementa una variable.	`x++; /*incrementa x por uno y devuelve el valor antiguo de x*/`
		`++x; /*incrementa x por uno y devuelve el valor nuevo de x*/`
		`x--; /*decrementa x por uno y devuelve el valor antiguo de x*/`
		`--x; /*decrementa x por uno y devuelve el valor nuevo de x*/`

+= (suma compuesta)
-= (resta compuesta)
*= (multiplicación compuesta)
/= (división compuesta)
	Realiza una operación matemática en una variable con otra constante o variable.	x += y; //equivale a escribir x = x + y ;
		x -= y; //equivale a escribir x = x – y;
		x *= y; //equivale a escribir x = x * y;
		x /= y; //equivale a escribir x = x / y;

1.2.2.2 Valores

Los datos que almacena y manipula el programa de Arduino son de valor constante o variable. A diferencia de las constantes, las variables son todo dato o conjunto de datos que cambian su valor con la ejecución del programa y los puede haber de

distintos tipos. Las variables presentadas en la Tabla 5, son estados predefinidos en el lenguaje Arduino que se utilizan para hacer que los programas sean más fáciles de leer, definiendo y caracterizando a los puertos *I/O* con los que se trabaja.

TABLA 5 Estados constantes en el lenguaje de programación Arduino.

Constantes *booleanos*	
`False` (falso)	Se define como 0 (cero).
`true` (verdadero)	Se define como 1 (uno), aunque cualquier número entero distinto a cero es cierto, en un sentido *booleano*.
Constantes para definir nivel de puertos	
`HIGH` (alto)	Como entrada, indica recibir una tensión de 3V o más. Como salida, indica estar a 5V.
`LOW` (bajo)	Como entrada, indica recibir una tensión de 2V o menos. Como salida, indica estar a 0V.
Constantes para definir puertos digitales	
`INPUT`	Los puertos configurados están en un estado de alta impedancia, es decir, con una gran resistencia en serie.
`INPUT_PULLUP`	Los puertos configurados invierten el comportamiento, donde HIGH significa que el sensor está apagado y LOW significa que está activado.
`OUTPUT`	Los puertos configurados estarán en un estado de baja impedancia, es decir, con mínima resistencia en serie.
`LED_BUILTIN`	Declara manualmente el puerto conectado a un LED en serie con una resistencia, en la mayoría de las placas es el puerto 13.

En la Tabla 6, se presentan los tipos de datos que son frecuentemente utilizados, no son todos los tipos de valores que hay, pero sí los más comunes.

TABLA 6 Tipo de datos en el lenguaje de programación Arduino.

Dato	Aplicación	Declaración

`void`	Se utiliza sólo en la declaración de funciones. Indicando que se espera que la función no devuelva información alguna cuando se llama.	`void setup () {`
		`}`
		`void loop () {`
		`}`

`boolean`	Un dato *booleano* tiene uno de dos valores: verdadero o falso. Cada variable ocupa un byte de memoria.	`boolean running = false;`
`char`	Para almacenar un valor en forma de letra.	`char myChar = 'A'`
`unsigned char`	Codifica números del 0 al 255.	`unsigned char myChar = 240;`
`int`	Variable principal para el almacenamiento de números enteros.	int ledPin = 13; int var = 25;
`float`	Números que tienen un punto decimal.	float myfloat; float x = 1.117;
Convertidor	**Aplicación**	**Declaración**
`int()`	Convierte un valor a un dato tipo `int`.	int (myfloat); int(x); //donde x = 1.117;
`float()`	Convierte un valor a un dato tipo `float`.	float(2/var);

1.2.2.3 Funciones

Una función realiza una tarea específica y puede retornar un valor. Las funciones son utilizadas para descomponer grandes problemas en tareas simples. Una función puede estar formada por líneas de código hasta su forma más simple presentada bajo una instrucción matemática. De igual modo que en las secciones anteriores, se presenta en la Tabla 7 algunas de las funciones básicas.

TABLA 7 Funciones en lenguaje de programación Arduino.

Función digital I/O	Aplicación	Ejemplo
`pinMode()`	Configura el pin especificado como entrada o como salida.	`int ledPin = 13;` `pinMode (ledPin, OUTPUT);`
`digitalWrite()`	Escribe un valor HIGH o LOW en un puerto digital.	`int ledPin = 13;` `digitalWrite(ledPin, HIGH)`
`digitalRead()`	Lee el valor de un puerto digital, pudiendo ser éste HIGH o LOW.	`int  inPin = 7;` `val = digitalRead(inPin);`
`analogReference(type )`	Configura la tensión de referencia utilizado para la entrada analógica.	`analogReference(DEFAULT);`
`analogRead()`	Lee el valor del pin analógico especificado.	`analogRead (puerto)`
`analogWrite()`	Escribe un valor analógico (PMW) a un puerto.	`analogWrite(puerto, valor)`
Tiempo	Aplicación	Sintaxis
`millis()`	Devuelve el número de milisegundos desde que la placa Arduino empezó a correr el programa actual.	`time = millis();`
`micros()`	Devuelve el número de microsegundos desde que la placa Arduino empezó a correr el programa actual.	`time = micros();`
`delay()`	Hace una pausa en el programa para la cantidad de tiempo (en milisegundos) especificado como parámetro.	`delay(1000);`
`delayMicroseconds()`	Hace una pausa en el programa para la cantidad de tiempo (en microsegundos) especificado.	`delayMicroseconds(50);`
Matemáticas	Aplicación	Sintaxis
`min()`	Calcula el mínimo de dos números.	`sensVal = min(sensVal, 100);`
`max()`	Calcula el máximo de dos números.	`sensVal = min(sensVal, 100);`
`abs()`	Calcula el valor absoluto de un número.	`var = analogRead(0);` `var = abs(var);`
`constrain()`	Restringe a un número a estar dentro de un rango.	`constrain(sensVal, 10, 150);` `//limita a sensVal entre 10 y 150.`
`map()`	Mapea un número de un rango a otro.	`y = map(x, 1, 50, 50, 1);`
`sqrt()`	Calcula la raíz cuadrada de un número.	`sqrt(234);`
Trigonométricas	Descripción	Ejemplo
`sin()`	Calcula el seno de un ángulo (en radianes).	`sin(rad);`
`cos()`	Calcula el coseno de un ángulo (en radianes).	`cos(rad);`
`tan()`	Calcula la tangente de un ángulo (en radianes).	`tan(rad);`

Capítulo

2

Metodología

El estudio preliminar para la implementación de la tecnología Arduino en el control de nivel de un taque, se organizó en etapas consecutivas que a continuación se presentas en el mismo orden de ejecución. Las etapas primera y cuarta están presentes en el Capítulo 1, la segunda y tercera en el Capítulo 3, mientras que las restantes quedan implícitas en el Capítulo 4. Exceptuando la primera y cuarta, el resultado final de cada etapa implica regresar a la anterior para ir perfeccionando, la metodología del estudio se ilustra en la Figura 2.

2.1 Analizar la electrónica Arduino

En esta primera etapa se recopilaron publicaciones de trabajos en torno a la placa Arduino, específicamente en aplicaciones relacionadas al control automático y se manejan como antecedentes.

Aunque la página oficial de Arduino es muy completa, se documentó de información adicional como guías de uso, videos y publicaciones de la comunidad Arduino, ya que el usuario suele no ser un experto; esta información es sintetizada en el Capítulo 1.

Esta etapa incluyó además el estudio de la información técnica de los elementos básicos que integran el hardware de Arduino, así como de los posibles dispositivos que pueden conectarse a él.

2.2 Desarrollar la ingeniería básica

La ingería básica parte de conceptualizar en conjunto: la problemática y la propuesta de solución. La propuesta de solución contempla las capacidades y alcances del proyecto propuesto, estos dependerán de los recursos a disposición, tanto materiales como humanos, y de la capacidad tecnológica de la placa Arduino (analizada en la etapa anterior), a este conjunto se le ha nombrado ingeniería conceptual.

En base a la ingeniería conceptual, surge el desarrollo formal de la ingeniería básica con documentos propios como son:

1. Diagrama de Flujo de Proceso (DFP).
2. Diagrama de Tuberías e Instrumentación (DTI).
3. Filosofías de proceso.
4. Filosofía de control.
5. Diseño y/o adaptaciones de componentes.
6. Diseño de circuitos o cableado.
7. Diagrama isométrico.

2.3 Ensamble del sistema de control

En base a los documentos propios de la ingeniería básica, el ensamble del equipo se realiza de manera sistemática. Sobre la marcha estos documentos y el ensamble pueden irse modificando si se encuentra necesario, la debida concordancia es el resultado final de ellos, completando dicha documentación con un diagrama isométrico.

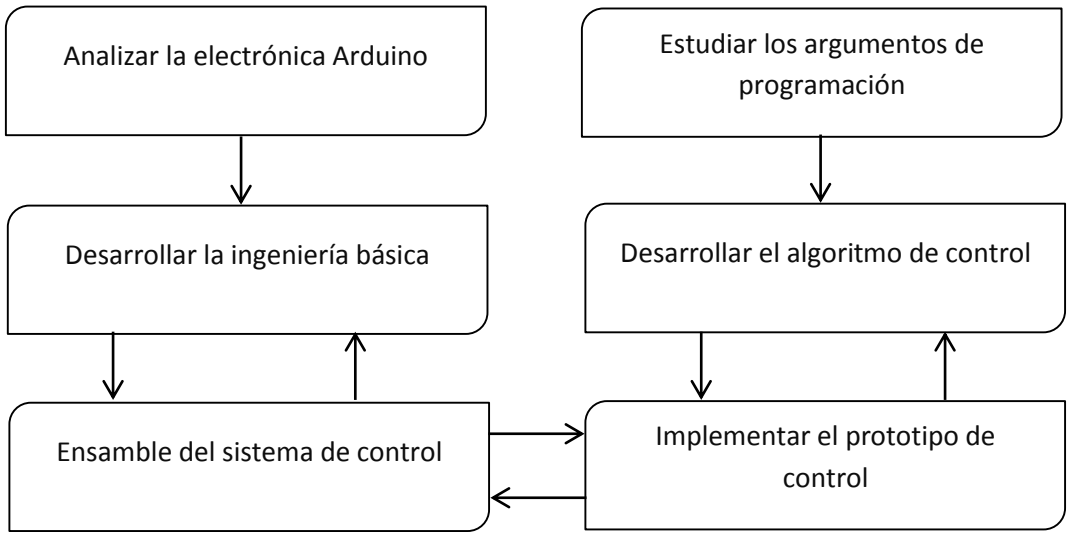


FIGURA 2 Metodología para el estudio preliminar de la implementación de Arduino en el control de nivel.

2.4 Estudiar los argumentos de programación

El lenguaje de programación de Arduino está basada en el lenguaje de programación C/C++, el estudio de sus argumentos se enfocó a los dispositivos a controlar. El resultado de este estudio se sintetiza en las tablas del Capítulo 1 para el mejor manejo de información, y se plasma en la programación del algoritmo de control diseñado en la siguiente etapa.

2.5 Desarrollar el algoritmo de control

El algoritmo de control deberá tener la lógica necesaria para actuar de manera autónoma ante los posibles escenarios que puedan surgir bajo la filosofía de proceso desarrollada en la ingeniería básica y con los elementos implementados en ella. Este algoritmo mantendrá el nivel de líquido deseado en el tanque de proceso implementando un control proporcional-integral-derivativo (PID).

2.6 Implementar el prototipo de control

Esta fase refiere a las pruebas de control con el equipo ensamblado en la tercera etapa, operando bajo el algoritmo programado en la quinta. Esta última etapa va de la mano con la anterior, ya que el algoritmo de control es reajustado paralelamente a su implementación, el algoritmo final es con el cual los resultados se han reportado.

Capítulo

3

Ingeniería, diseño y construcción

Los proyectos de ingeniería exigen de sus respectivas ingenierías conceptual y básica como bases para el diseño, construcción y futura puesta en marcha de prototipos, equipos, sistemas, etc. Es así como el presente Capítulo se encuentra documentado con: diagramas de bloques, DFP, DTI e isométrico; filosofías de proceso y control; diseño de componentes y esquemas electrónicos. Con esto se describe la planeación y propuesta de desarrollo para la implementación del sistema a controlar.

3.1 Ingeniería conceptual

La ingeniería conceptual presenta la problemática a resolver y la solución propuesta en este proyecto de manera generalizada pero muy concreta. Se tiene un tanque de proceso a presión atmosférica, y el proceso requiere que el nivel de líquido en dicho tanque permanezca en un valor deseado por la exigencia misma del proceso, pudiendo ser variante a través del tiempo.

La Figura 3 muestra un diagrama de bloques del sistema de control propuesto, donde el nivel del líquido en el tanque de proceso es monitoreado con un sensor ultrasónico que retroalimenta a la placa Arduino con datos de altura. Un control es

programado y compilado en la placa Arduino Mega 2560, por ser una de las versiones que cuenta con mayor número de puertos digitales *I/O* y alta capacidad de memoria. El controlador mandará señales hacia las válvulas de control que permitan la carga y descarga de líquido en el tanque de proceso, y al actuador de una bomba que desplazará el líquido de proceso desde un tanque de almacenamiento.

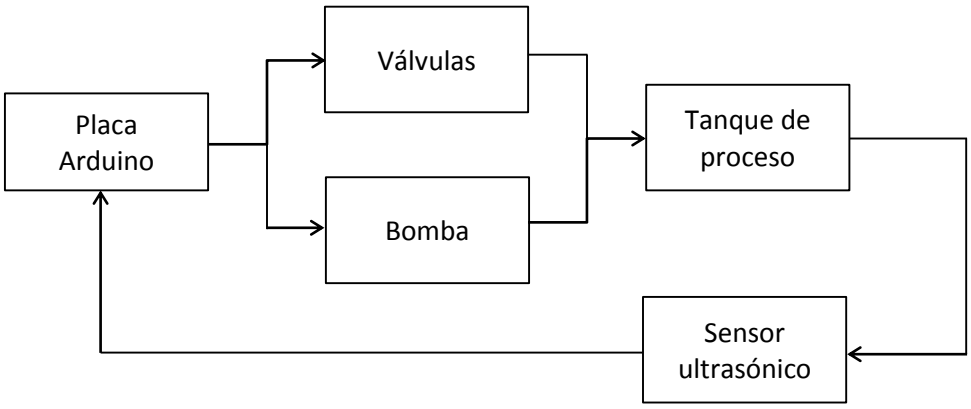


FIGURA 3 Diagrama de bloques del sistema de control.

3.2 Diseño de componentes del sistema de control

3.2.1 Sensor de nivel

El sensor de nivel es un dispositivo que por medio de ráfagas ultrasónicas, determina distancias y en consecuencia la altura de líquido en un taque.

En la Figura 4 se muestra el sensor ultrasónico empleado (HC-SR04), éste consta de cuatro pines definidos como: Vcc, Trig, Echo, GND; con los pines Vcc y GND se conecta el dispositivo a 5 V de *DC* y tierra respectivamente para su alimentación, por el pin Trig se emite una señal (pulso 10 μs *TTL*) que activa el emisor de ultrasonidos, mientras que el pin Echo provee de una señal de respuesta (pulso *TTL* proporcional a la distancia medida) que termina al haber recepción de los mismos. En la Tabla 8, se presentan éstas y otras especificaciones del dispositivo HC-SR04.

FIGURA 4 Sensor ultrasónico HC-SR04.

TABLA 8 Especificaciones del dispositivo HC-SR04 (Elecfreaks, 2013).

Especificación	Medida
Voltaje de operación	5 V de *DC*
Corriente de operación	15 mA
Frecuencia de operación	40 kHz
Rango Máximo	4 m
Rango Mínimo	2 cm
Ángulo de cobertura	15 °
Señal de disparo	Pulso 10 µs TTL
Señal de respuesta	Pulso TTL proporcional a la distancia medida
Dimensión	4.5x2x1.5 cm

El sensor emite una ráfaga ultrasónica corta (40 kHz), la cual, al chocar con una superficie delante, retorna. Justo en el momento en que la ráfaga es emitida, el sensor envía por uno de sus pines, un pulso de respuesta, que termina cuando la ráfaga que retorna es detectada por el receptor de ultrasonidos del sensor. Por lo tanto, la duración del pulso de respuesta (τ_{Echo}) es doblemente proporcional a la distancia entre el sensor y la superficie delante (d_{sup}); esto último queda implícito en la siguiente ecuación:

$$d_{sup} = \tau_{Echo}\, v_{sonido}\, 2^{-1} \qquad \text{..... Ec. (1)}$$

donde v_{sonido} es la constante de velocidad del sonido.

En la Figura 5 se presenta el arreglo de tanque y sensor ultrasónico, de este modo y aplicando la Ec. (1), d_{sup} corresponde a la distancia entre el sensor y la superficie de líquido en el tanque; y el nivel de líquido (h) es calculado como

$$h = H - d_{sup} \qquad \text{..... Ec. (2)}$$

donde H refiere a la altura a la cual esté instalado el sensor respecto al fondo del tanque.

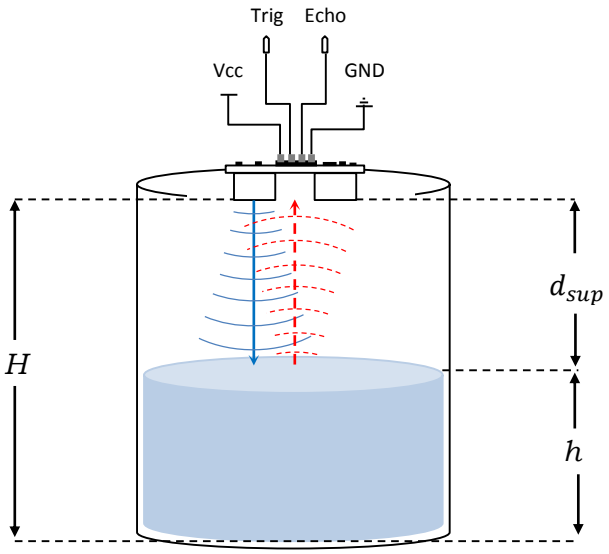


FIGURA 5 Arreglo de tanque y sensor de nivel.

3.2.2 Válvula reguladora de carga

La válvula de control principal regula el flujo de entrada hacia el tanque de proceso, el diseño de este dispositivo está conformado por:

1. Obturador concéntrico de mariposa.
2. Cuerpo de la válvula.
3. Servomotor electromecánico.
4. Junta homocinética.

El actuador de esta válvula se compone de un servomotor acoplado sobre el eje de la mariposa, mediante la junta homocinética. Como se muestra en la Figura 6, este mecanismo permite la apertura, cierre y posiciones intermedias del obturador de mariposa. Este dispositivo está diseñado para sostener una posición Normalmente Abierta (NA o NO) y su cierre se alcanza posicionando la mariposa 85° en dirección opuesta a las manecillas de un reloj.

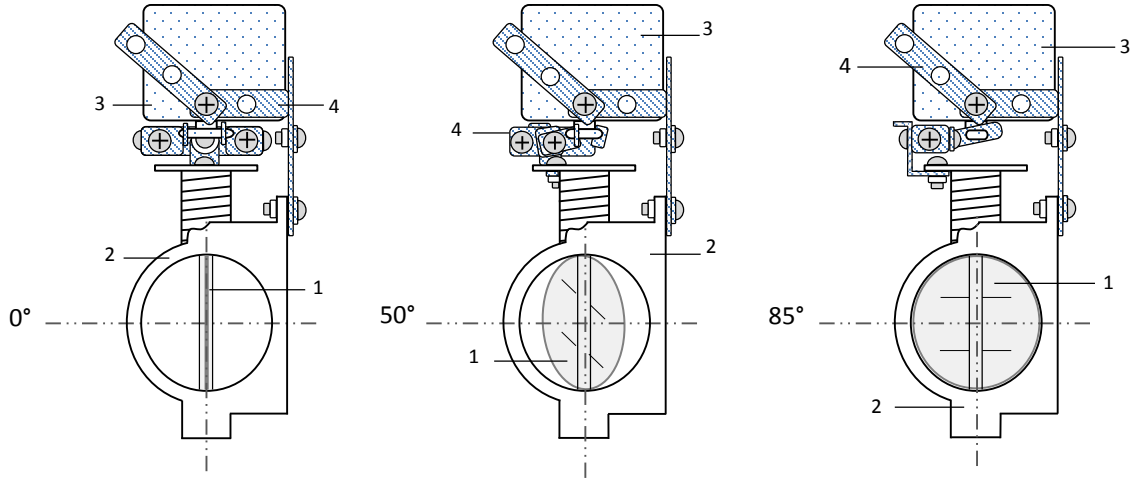


FIGURA 6 Detalle de la válvula de control de carga, donde se muestra el arreglo actuador-válvula motricidad (vista frontal al sentido del flujo). 1. Obturador, 2. Cuerpo de la válvula, 3. Servomotor, 4. Junta homocinética.

3.2.3 Válvula todo/nada de descarga

La válvula de control secundaria permite la descarga de fluido en el tanque de proceso con su apertura. Parte del diseño original de una válvula antirretorno (*check*) que permite manipular la posición del obturador independientemente de la función antirretorno.

Como se muestra en la Figura 7, de manera normal la válvula actúa como una *check* simple impidiendo la salida de flujo en el tanque, sin embargo, es posible elevar el obturador (~30°) a través de una línea conectada a un servomotor electromecánico, el cual, eleva su eje 130°.

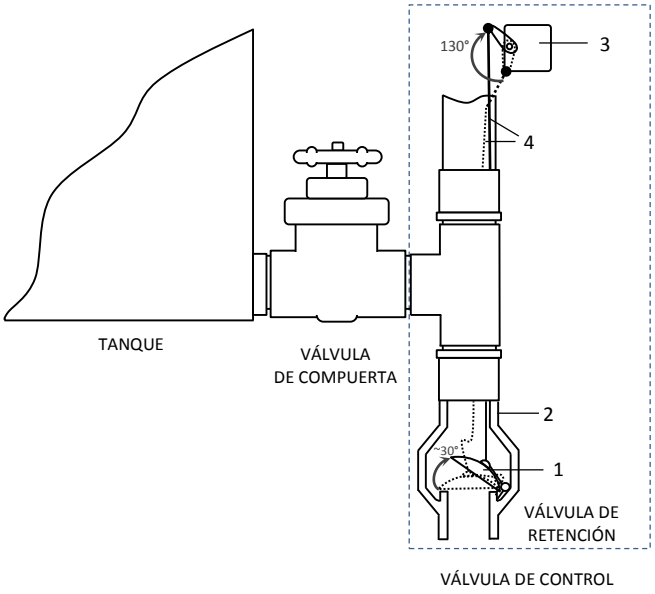


FIGURA 7 Arreglo general que muestra la línea de transmisión entre el actuador y el obturador de una válvula *check*. 1. Obturador, 2. Cuerpo de la válvula, 3. Servomotor, 4. Línea de actuación.

3.3 Ingeniería básica

El desarrollo de documentos propios de la ingeniería básica, requiere de una adecuada identificación de los objetivos, requisitos y aplicaciones que los equipos involucrados deberán satisfacer.

La simbología empleada (Norma Pemex No. 2.451.03) para los componentes (como equipos, instrumentos e indicadores) de diagramas DFP y DTI se muestra en la Tabla 9. El fluido de proceso, agua potable, se identifica con las letras APO, mientras que las denominaciones en letra para la identificación de los equipos y las funciones de los instrumentos se presentan en la Tabla 10.

El diseño de los diagramas DFP y DTI que se presentan en las Figuras 8 y 9 respectivamente, parten de la misma distribución espacial y son interpretados en la filosofía de proceso y control.

TABLA 9 Simbología de equipos de proceso, instrumentación e indicadores (Norma Pemex No. 2.451.03).

<table>
<tr><td colspan="4">Bomba y equipo de almacenamiento</td></tr>
<tr><td colspan="2">Bomba (todo tipo)</td><td colspan="2">Tanque atmosférico de almacenamiento</td></tr>
<tr><td colspan="4">Flujo e instrumentación</td></tr>
<tr><td>Tubería principal
Tubería auxiliar</td><td>Balance de materia
Corriente | 1 | 2 | 3
C_1
C_2
Total Lb.
Total Gal.
Temperatura
Presión</td><td>Rotámetro</td><td>Válvula de compuerta
Válvula de mariposa
Válvula de bola
Válvula de retención</td></tr>
<tr><td>Señal eléctrica</td><td colspan="2">Inst. montado localmente</td><td></td></tr>
<tr><td>E
Motor eléctrico</td><td>Operación manual</td><td>Medidor de nivel de líquido</td><td>LC
Controlador de nivel</td></tr>
<tr><td colspan="4">Otros indicadores</td></tr>
<tr><td colspan="2">Identificador de línea de corriente</td><td colspan="2">L. B.
Límite de batería</td></tr>
</table>

TABLA 10 Sistema de denominación de equipos y función de instrumentos (Norma Pemex No. 2.451.03).

Denominación de equipos y función de instrumentos					
TA	__	Tanque acumulador	VM	__	Válvula manual
TC	__	Tanque subterráneo (cisterna)	VC	__	Válvula de control
BA	__	Bomba centrífuga	LC	__	Controlador de nivel

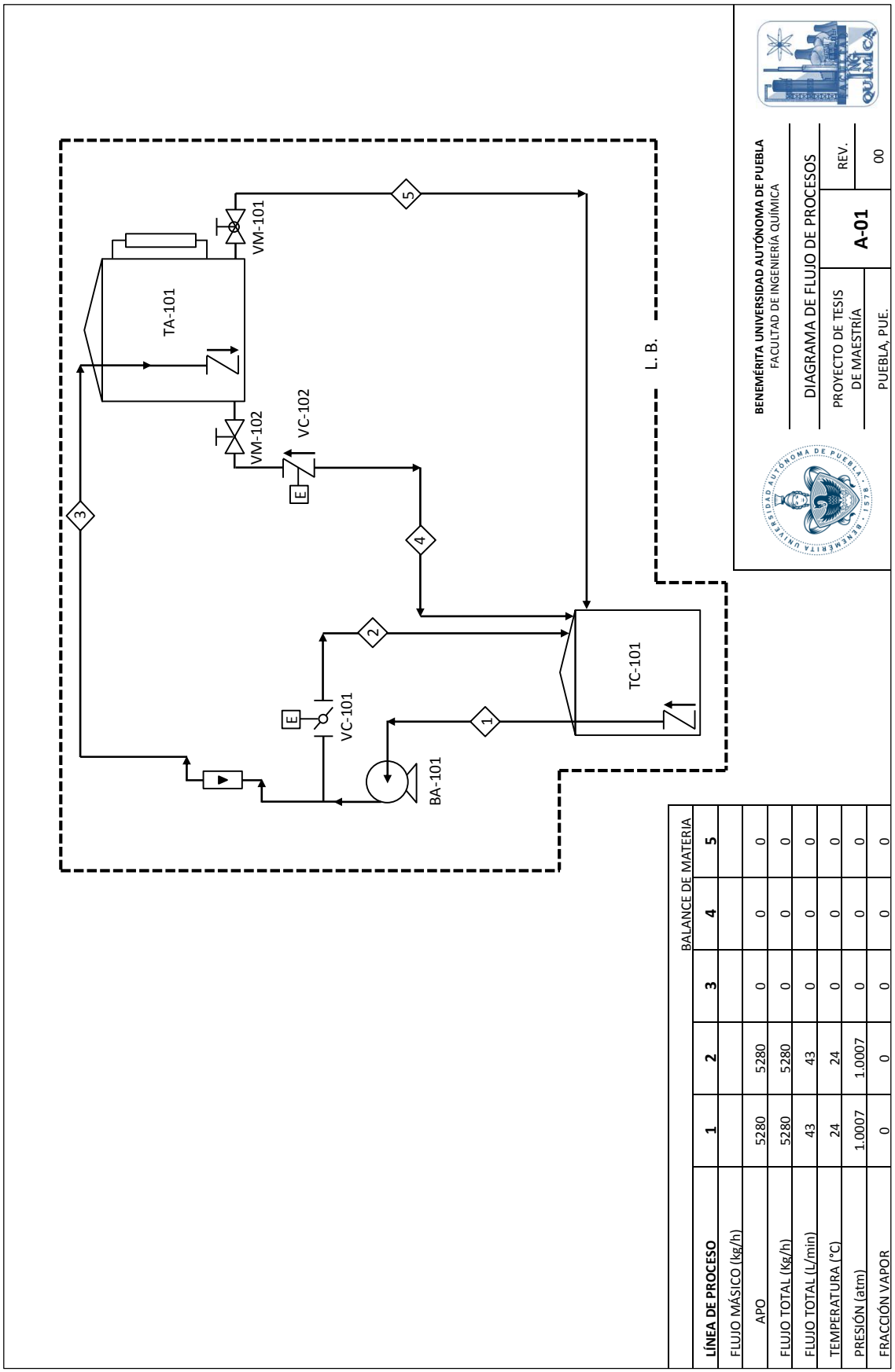
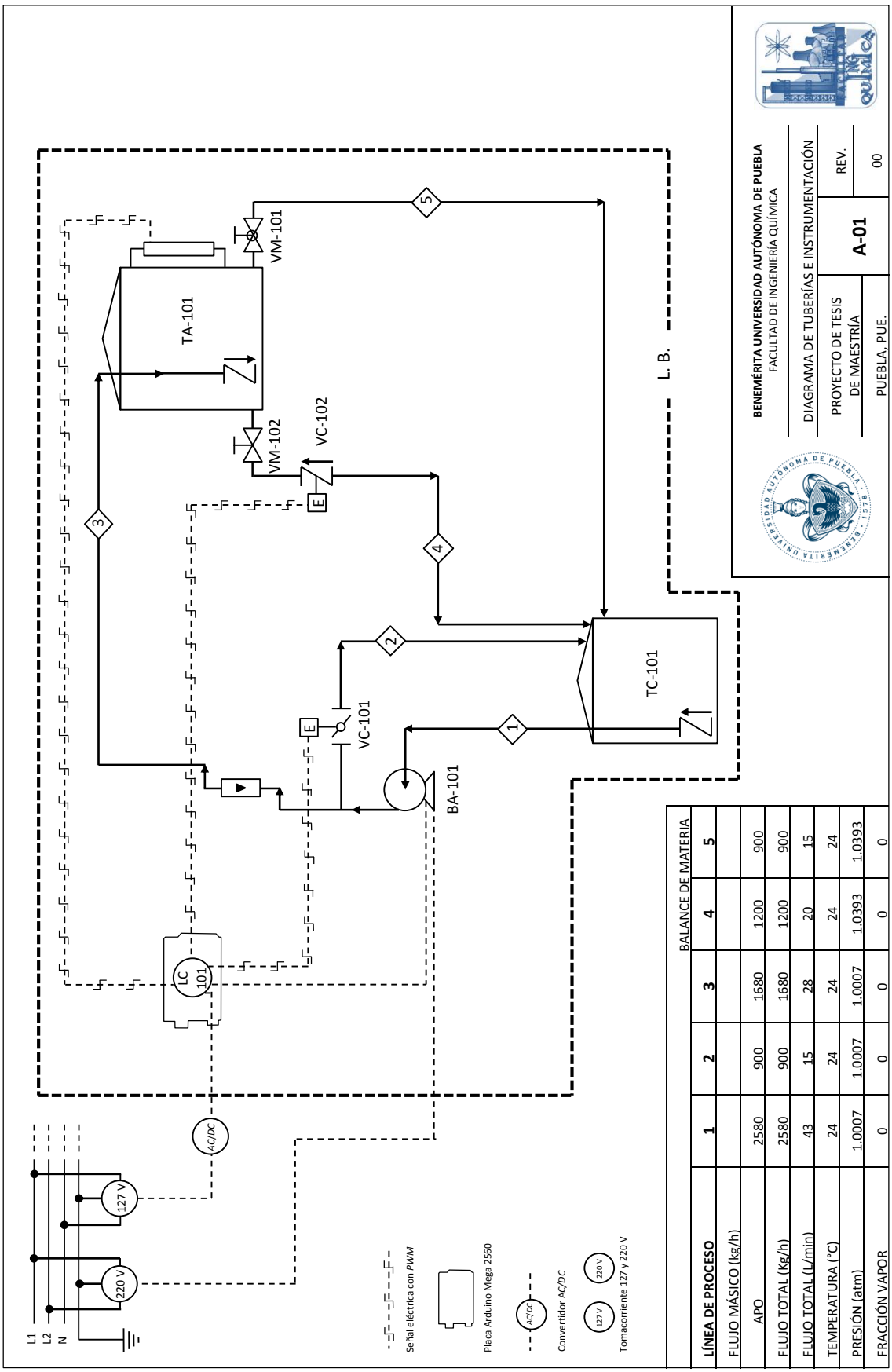


FIGURA 8 Diagrama de Flujo de Proceso (DFP) del sistema de control.



BALANCE DE MATERIA

LÍNEA DE PROCESO	1	2	3	4	5
FLUJO MÁSICO (kg/h)					
APO	2580	900	1680	1200	900
FLUJO TOTAL (Kg/h)	2580	900	1680	1200	900
FLUJO TOTAL (L/min)	43	15	28	20	15
TEMPERATURA (°C)	24	24	24	24	24
PRESIÓN (atm)	1.0007	1.0007	1.0007	1.0393	1.0393
FRACCIÓN VAPOR	0	0	0	0	0

FIGURA 9 Diagrama de Tuberías e Instrumentación (DTI) para el sistema de control.

3.3.1 Filosofía de proceso

El proceso comienza con el tanque cisterna TC-101 (100 L) en el cual se almacena el fluido de proceso APO. La bomba BA-101 (1 hp) succiona el fluido de proceso en el TC-101, su descarga deriva a través de las líneas 3 y/o 2 por efecto de un arreglo de tuberías (*loop* de tuberías) y el porcentaje de cierre de la válvula reguladora VC-101.

La línea 2 retorna el fluido al TC-101, mientras que el flujo en la línea 3 tiene salida hacia el tanque atmosférico TA-101 (80 L). La VC-101 está instalada en la línea de retorno, y con su total apertura, la descarga de la BA-101 fluye al TC-101. De este modo, con el porcentaje de cierre de la VC-101 se va regulando el flujo de entrada (Q_i) hacia el TA-101.

El TA-101 es el tanque principal, contiene el nivel de líquido especificado por la exigencia misma del proceso, tiene una entrada que conecta a la línea 3 y dos salidas que conectan a una válvula de bola VM-101 y una válvula de compuerta VM-102.

La VM-102 (siempre abierta durante el proceso) se conecta con una válvula todo/nada VC-102 que con su apertura produce un flujo (Q_1) de salida en el TA-101 que se direcciona hacia el TC-101 por la línea 4. La VM-101 es una válvula reguladora que con su apertura parcial o total, produce un flujo (Q_2) de salida en el TA-101 que se direcciona hacia el TC-101 por la línea 5. Por lo que el flujo de salida (Q_o) en el tanque puede generarse abriendo por lo menos una de ellas.

3.3.2 Filosofía de control

Se puede mantener un nivel deseado de líquido en el tanque atmosférico TA-101. Para ello, un sensor instalado en el tanque, monitorea el nivel real y envía una señal eléctrica al controlador automático LC-101. El LC-101 compara el nivel real con el deseado y corrige cualquier error enviando una señal eléctrica hacia la válvula electromecánica VC-101 la cual accionará ajustando su apertura o cierre.

Si al inicio o durante el control de nivel, el valor real supera al deseado, el LC-101 enviará una señal eléctrica a la válvula todo/nada VC-102 para abrirla, de lo contrario ésta cierra.

Para la operación de la bomba BA-101, el LC-101 le enviará una señal eléctrica, de lo contrario la bomba se apaga.

La perturbación en el sistema refiere a la descarga de líquido en TA-101 por la apertura parcial o total de la válvula VM-101 operada manualmente. Esta perturbación deberá ser siempre menor a la capacidad de bombeo, reflejando que a pesar de tener la máxima descarga, el sistema de control sea capaz de corregir la desviación del valor medido respecto al valor deseado.

3.4 Interfaz de comunicación

Se diseña la interfaz de comunicación del prototipo de controlador de nivel basado en Arduino. La interfaz comprende las conexiones electrónicas entre la placa Arduino Mega 2560 y los dispositivos a gestionar.

La presentación del montaje y circuito electrónico de las Figuras 10 y 11, se lleva a cabo con la ayuda de Fritzing, el cual, es un software de diseño para prototipos basados en Arduino (Fritzing, 2014).

Los componentes con los que se comunica el centro de control son:

a. Placa Arduino Mega 2560 (ARD1).
b. Sensor ultrasónico HC-SR04 (HC1).
c. Servomotores (VC1 y VC2) para el control de las válvulas de control VC-101 y VC-102.
d. Relevador de estado sólido (KA1) para el accionamiento de la bomba BA-101.
e. Interruptor de inicio (SB1) y potenciómetros (R1 y R2) que funcionan como interfaz de entrada con los operarios.
f. *Display LCD* (LCD1) e indicadores de luz (LED1, LED2, LED3 y LED4) que actúan como interfaces de salida con los operarios.

El KA1 forma parte del circuito de control de la bomba BA-101 y como puede observarse en las Figuras 10 y 11, se encuentra conectado a una línea (L2) de *AC* perteneciente a una toma trifásica de 220 V, tensión que requiere la bomba para su accionamiento. En la Figura 12, se presentan los circuitos de fuerza y control del motor (M) de la bomba; un contactor (KM1) en serie con un guardamotor (FR1) forman parte del sistema de actuación de la bomba además del KA1.

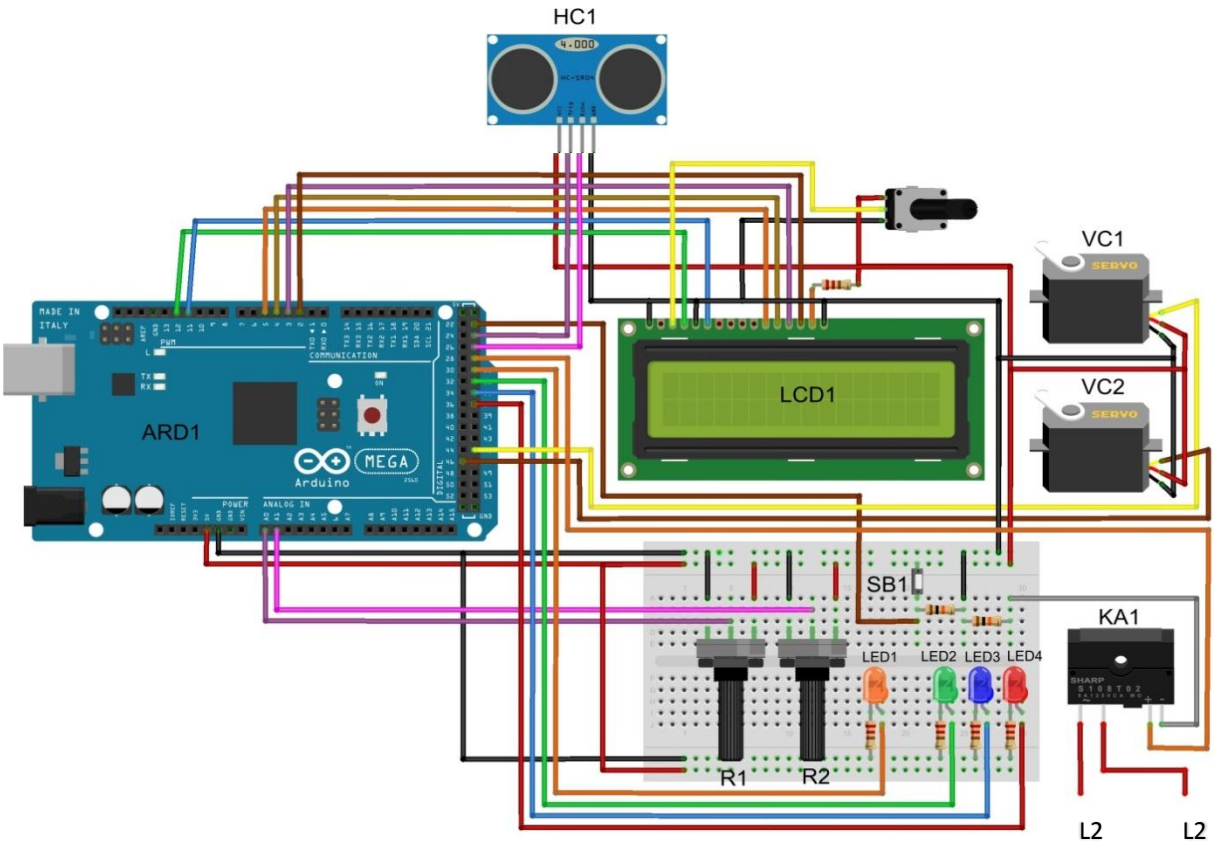


FIGURA 10 Diseño electrónico secuencial basado en un preliminar (*protoboard*). L2 identifica a una línea de corriente alterna (*AC*).

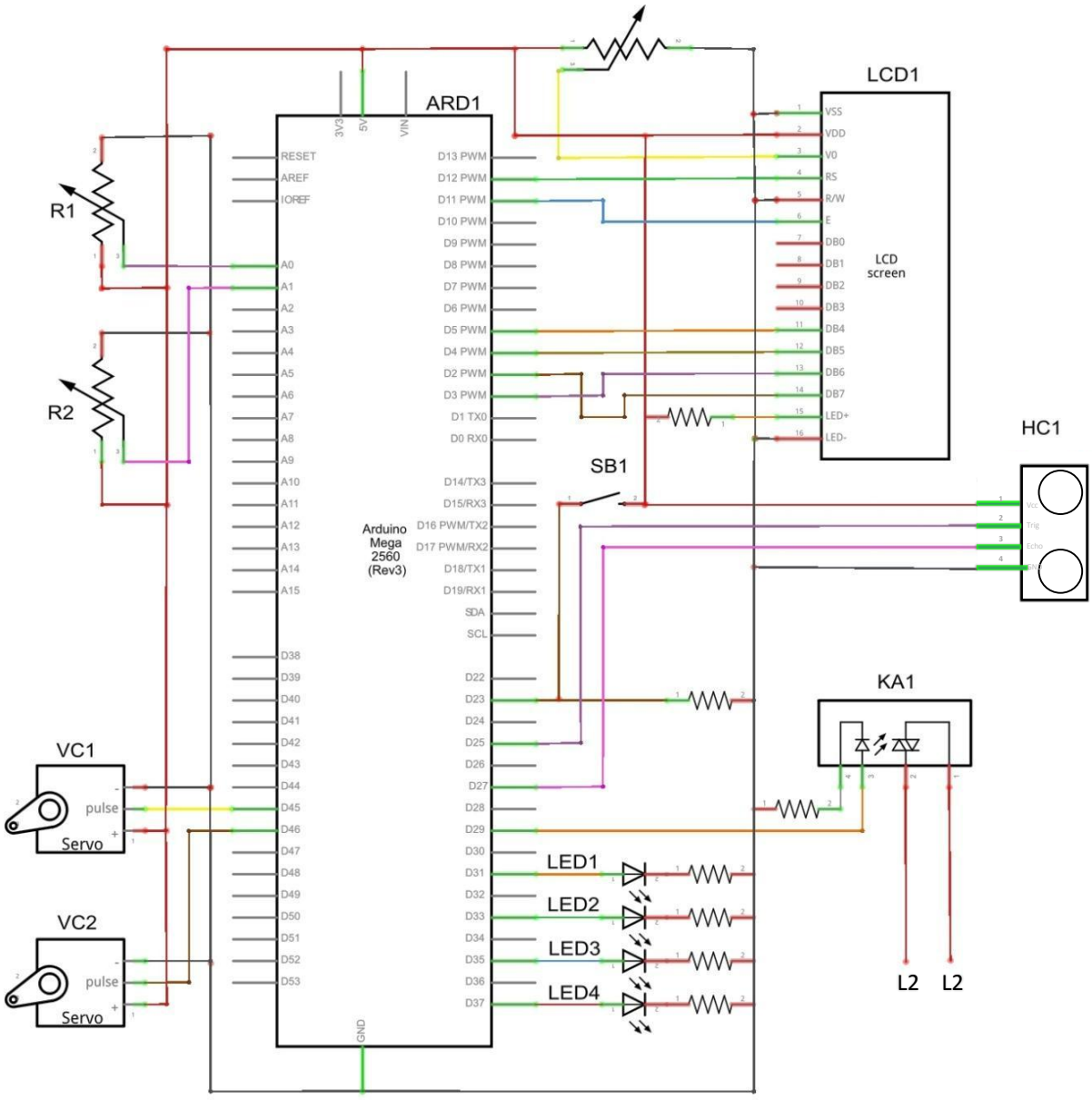
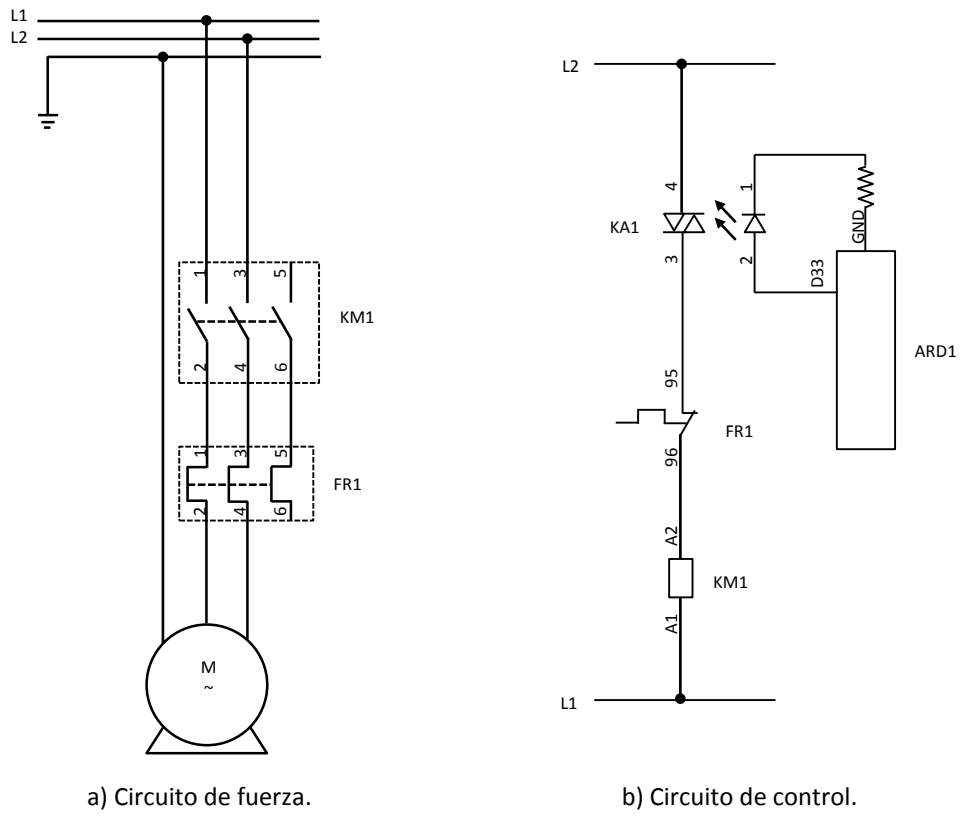


FIGURA 11 Diagrama bifilar simple de las conexiones entre la placa Arduino y demás dispositivos. L2 identifica a una línea de corriente alterna (*AC*).



a) Circuito de fuerza. b) Circuito de control.

FIGURA 12 Circuito de fuerza y circuito de control para el accionamiento del motor (M) de la bomba BA-101.

3.5 Ensamble del sistema a controlar

En la Figura 13, se muestra el diagrama isométrico de la red de tuberías que corresponde al sistema de pruebas, con el cual, el prototipo de controlador de nivel opera; podemos apreciar el arreglo de tanque, bomba y *loop* de tuberías.

Como resultado del arreglo isométrico, en la Figura 14, se presenta el comportamiento isoporcentual (a partir de ~11°) del flujo de entrada (Q_i) al tanque atmosférico TA-101 en función del ángulo de cierre en la mariposa de la válvula de regulación VC-101. En consecuencia, para el control automático de VC-101, la mariposa de dicha válvula tendrá un posicionamiento de 10 a 85°, tomando a 10° como totalmente abierta (0% cerrada) y a 85° como totalmente cerrada (100% cerrada).

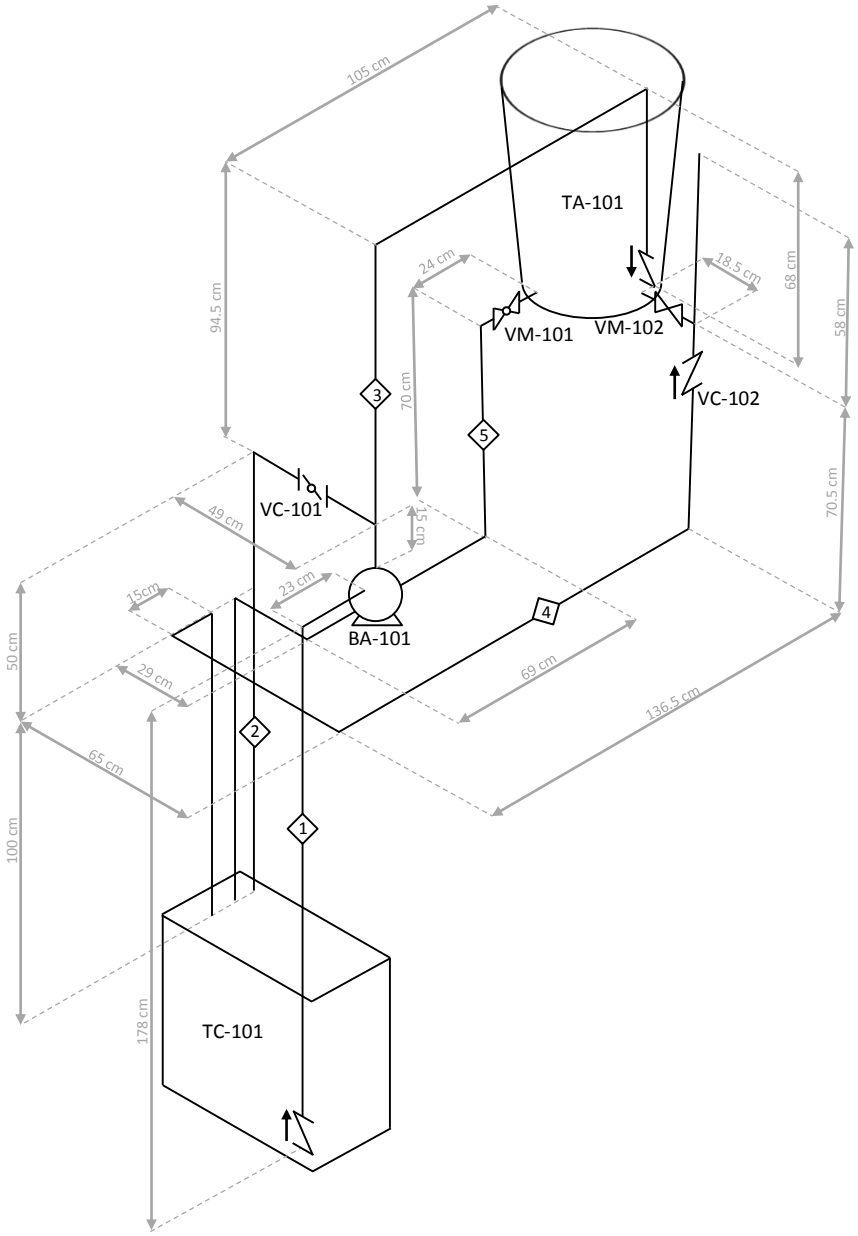


Figura 13 Isométrico de red de tuberías que corresponde al sistema de pruebas.

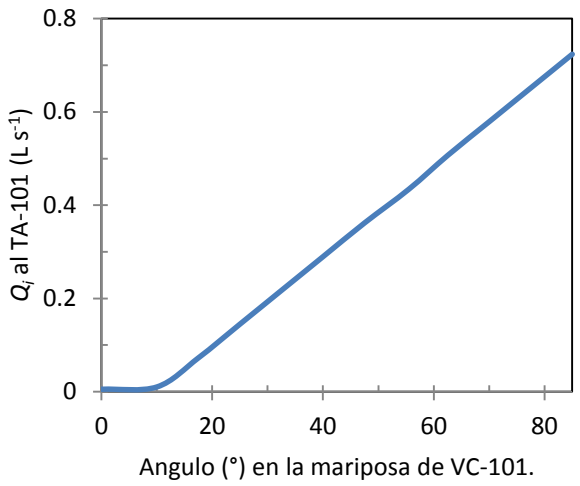


FIGURA 14 Comportamiento del flujo de proceso hacia TA-101 respecto al posicionamiento de la válvula de control VC-101.

El prototipo de controlador se presenta en la Figura 15 como tablero de control, donde se encuentra la interfaz de comunicación con los operarios. El prototipo de controlador de nivel y sistema de pruebas se presentan en la Figura 16, en conjunto constituyen el equipo del sistema de control para estudiar la implementación de control PID de nivel basado en Arduino.

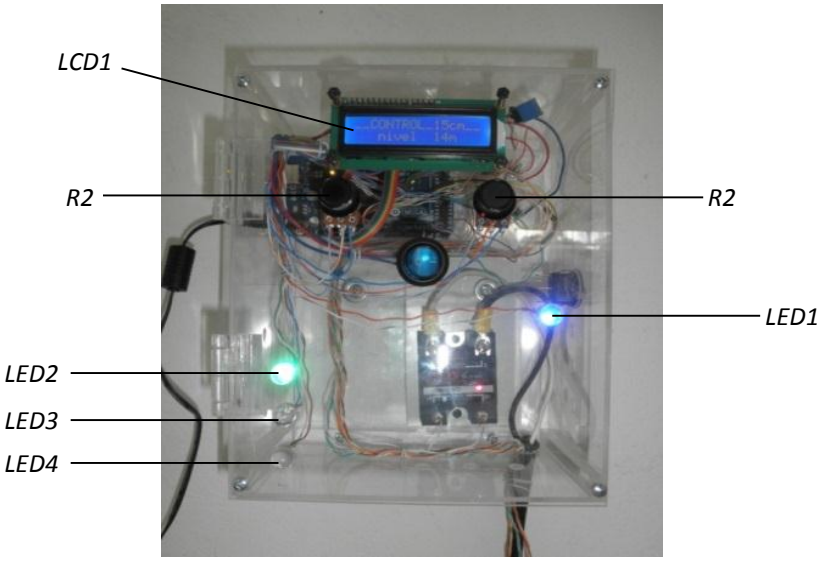


Figura 15 Tablero de control del prototipo de controlador de nivel.

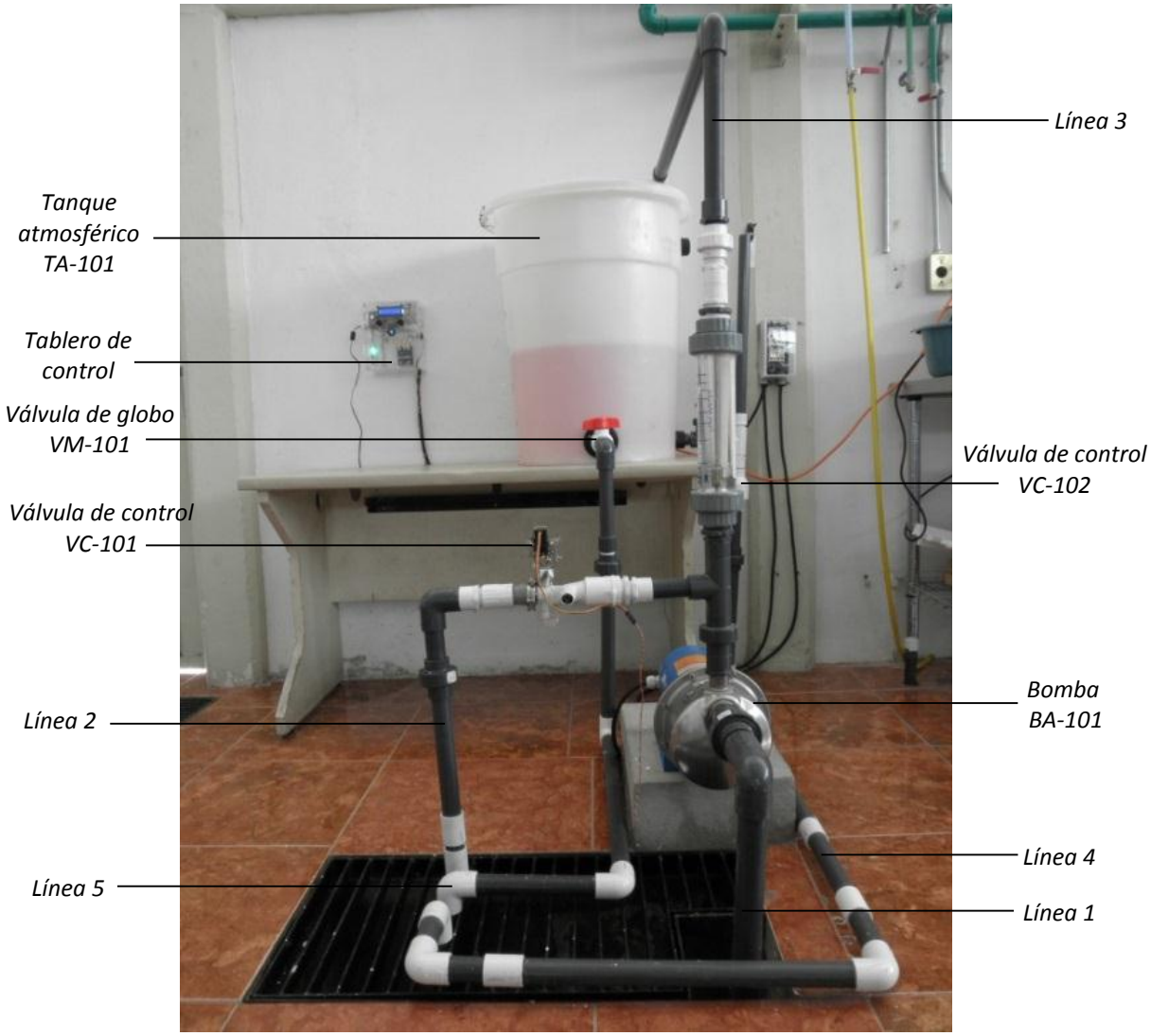


Figura 16 Sistema real del proceso a controlar en el cual se señalan algunos de los elementos que lo componen.

Capítulo

4

Programación del algoritmo de control

El prototipo de controlador de nivel basado en Arduino es un controlador digital programado para ser interpretado por el microcontrolador de la placa Arduino Mega 2560 (microcontrolador ATmega2560), tiene la capacidad de ejecutar sus secuencias lógicas sin la intervención de un equipo de cómputo. En este Capítulo se describen las bases para el desarrollo del algoritmo de control, así como su traducción en el lenguaje de programación Arduino que permita darle vida al sistema de control.

4.1 Sistema de control

Por definición, un sistema de control es una combinación de componentes que interactúan juntos y cumplen con un objetivo determinado (Ogata, 1998). En la Figura 17 se presenta, el diagrama de bloques del sistema de control de nivel propuesto, el cual, se conforma por:

a. Controlador digital automático (basado en Arduino).
b. Actuadores (de válvulas de control y bomba).
c. Planta (tanque atmosférico).
d. Sensor (sensor ultrasónico).

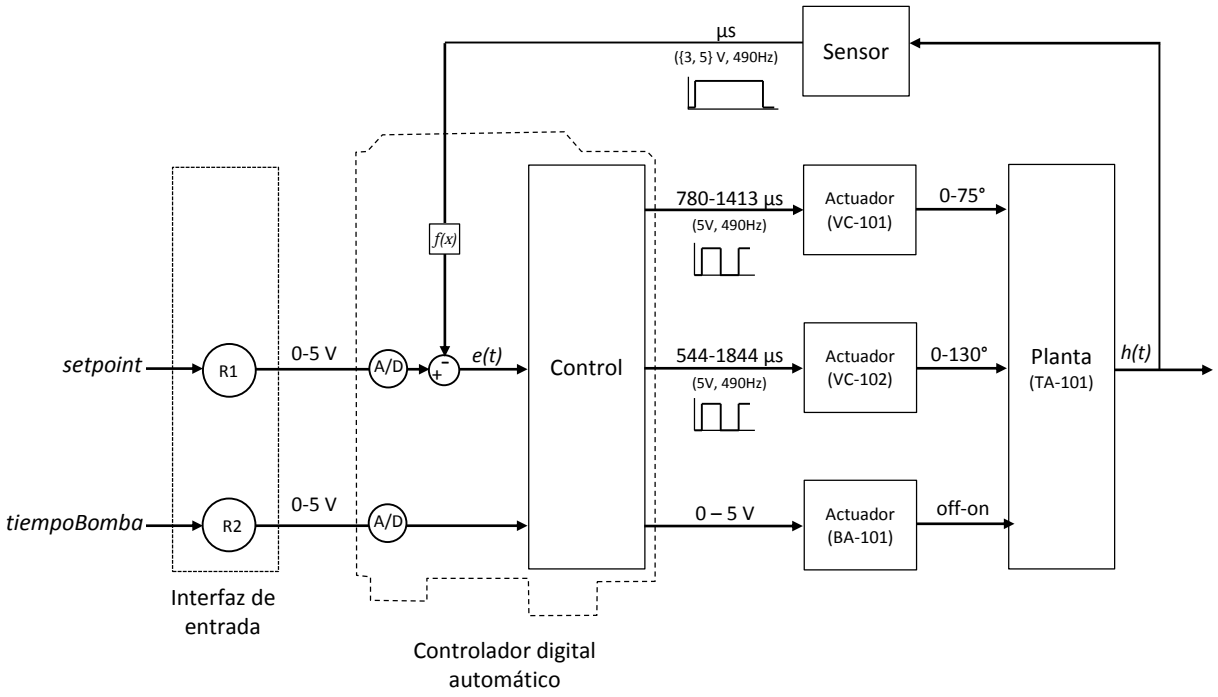


FIGURA 17 Diagrama de bloques del sistema de control de nivel propuesto, formado por controlador, actuadores y sensor.

El controlador detecta la señal de error y la procesa, generando salidas digitales que alimentan a los actuadores. Cada actuador producirá una entrada a la planta de acuerdo con la salida de su respectivo control, con el fin de que en combinación, la señal de salida de la planta (*h(t)*), medida por el sensor, se aproxime a la señal de referencia (*setpoint*). El error es la diferencia entre la variable medida y el *setpoint*:

$$e(t) = setpoint - h(t) \qquad \text{..... Ec. (3)}$$

La interfaz de comunicación que contempla el sensor (HC1) y los dispositivos que forman parte de los actuadores de válvulas (VC1, VC2) y bomba (KA1), entre otros, se presentó en el Capítulo 3 (Sección 3.4). La gestión de estos componentes se hace a partir de la programación de los puertos de Arduino a los cuales han sido conectados. Para facilitar la programación, establecer una nomenclatura para cada puerto de manera igual o semejante al dispositivo conectado, resulta ser menos engorroso a la hora de hacer reajustes en el programa. Es por ello, que utilizamos `#define` para dar nombre al valor constante con el cual se identifica cada puerto:

```
#define R1 0        // Potenciómetro para setpoint
#define R2 1        // Potenciómetro para tiempoBomba
#define SB1 23      // Botón de on/off
#define pinTrig 25  // Pin Trig de HC1
#define pinEcho 27  // Pin Echo de HC1
#define KA1 29      // Relevador de estado sólido
#define LED1 31     // LED indicador de bomba encendida
#define LED2 33     // LED indicador de nivel alcanzado
#define LED3 35     // LED indicador de nivel medio
#define LED4 37     // LED indicador de nivel bajo
#define VC1 44      // Actuador de válvula VC-101
#define VC2 45      // Actuador de válvula VC-102
```

4.1.1 Planta (tanque atmosférico)

En el sistema de control de nivel, la planta se compone de un tanque en el cual se tiene por objetivo, mantener un nivel deseado (*setpoint*) de líquido. La estructura geométrica del equipo TA-101 pude observarse en la Figura 18, se conforma de un tanque de la forma cono truncado en posición vertical, el cual mide 71 cm de altura, 43.5 cm de diámetro en su base y 50 cm de diámetro en su parte superior. Se trata de un taque de proceso atmosférico (abierto), que tiene una entrada y dos salidas de flujo con sección transversal circular de 2.54 cm de diámetro.

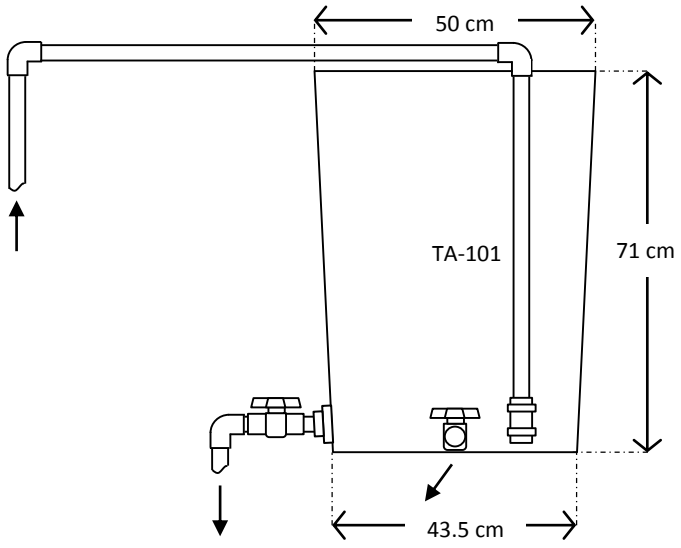


FIGURA 18 Estructura geométrica del tanque atmosférico TA-101 (vista lateral del sistema).

4.1.2 Sensor de nivel

El sensor que retroalimenta el sistema de control de nivel es el descrito en la Subsección 3.2.1. El procedimiento con el cual se realiza una medición se programa bajo el siguiente orden:

1. Se manda un pulso de activación de 5 V por 10 µs hacia el sensor a través de su pin Trig; esto requiere mandar 0V laterales al pulso de activación.
2. Se mide el tiempo que dura un pulso de alto rango (3-5 V) que envía el pin Echo del sensor.
3. Con la duración del pulso de respuesta se calcula la distancia hacia la superficie de líquido en el tanque. El pulso de respuesta es leído en µs, es por ello, que al aplicar la Ec. (1), v_{sonido}=1/29 cm µs^{-1}.
4. Con la distancia hacia la superficie de líquido se calcula el nivel de líquido en el tanque aplicando la Ec. (2) con H=69.5 cm.

El anterior procedimiento es programado bajo la función `ip()`:

```
float ip (int inPinTrig, int inPinEcho){      ..........................................  Fn. (c)
  float duracion, distancia;
  digitalWrite(inPinTrig, LOW);
  delayMicroseconds(2);
  digitalWrite(inPinTrig, HIGH);                // Pulso de activación
  delayMicroseconds(10);
  digitalWrite(inPinTrig, LOW);
  delayMicroseconds(2);
  duracion = pulseIn(inPinEcho, HIGH);          // Duración del pulso de respuesta
  distancia = (duración*(1/29))/2;              // Ec. (1)
  float outInput = distanciaSensor - distancia; // Ec. (2)
  return outInput;
}
```

la cual se manda a llamar dentro del algoritmo una vez por cada ciclo de la siguiente manera:

```
#define pinTrig 25  // Pin Trig de HC1
#define pinEcho 27  // Pin Echo de HC1

#define distanciaSensor 69.50   // H en Ec. (2)

float input;
```

```
void setup(){
  pinMode(pinTrig, OUTPUT);
  pinMode(pinEcho, INPUT);
}
void loop() {
  input = ip(pinTrig, pinEcho);  // Se llama a la función ip()
}
```

4.1.3 Actuadores: válvulas y bomba

La válvula de control VC-101 es una válvula que regula el flujo de entrada (Q_i) del tanque atmosférico TA-101. Su obturador de mariposa puede tener un posicionamiento de entre 0-85°, pero para su control automático se trabaja entre 10-85° (ver Figura 14). Para facilitar su control, el servomotor (VC1) se acopló de tal manera que el posicionamiento de 0-75° en su eje, corresponde con el de la mariposa de 10-85°.

La válvula de control VC-102 es una válvula que permite descargar líquido en el tanque TA-101. Su control es de dos posiciones: totalmente abierto/cerrado. Para ello, un servomotor (VC2) se posiciona normalmente en 0° y cuando es requerida la descarga en 130°.

El control de la bomba BA-101 es de dos posiciones: arranque/paro. La bomba opera (arranque) al alimentar el relevador (KA1) que cierra su circuito de fuerza y se apaga (paro) cuando el KA1 no recibe esa tensión.

Para programar el accionamiento del VC1 y el VC2 se incluye una librería para manejar servomotores en el programa y se manda una señal con `write()`, especificando el ángulo deseado para cada servomotor, por ejemplo: `vc101.write(40)` mandará un posicionamiento de 40° al servomotor VC1 y `vc102.write(130)` mandará un posicionamiento de 130° al servomotor de VC2. La puesta en marcha de BA-101 se hace enviando una señal de arranque con `digitalWrite(KA1, HIGH)` y su apagado ocurre enviando señal de paro con `digitalWrite(KA1, LOW)`. La estructura generalizada del programa para el accionamiento de estos dispositivos es la siguiente:

```
#define KA1 29      // Relevador de estado sólido
#define VC1 44      // Actuador de válvula VC-101
#define VC2 45      // Actuador de válvula VC-102

#include <Servo.h>  // Se llama a librería
Servo vc101;
Servo vc102;

#define distanciaSensor 69.50  // H en Ec. (2)
#define min101 0               // Ángulo mínimo de control para VC-101
#define max101 75              // Ángulo máximo de control para VC-101
#define min102 0               // Ángulo mínimo de control para VC102
#define max102 130             // Ángulo máximo de control para VC102

void setup(){
  pinMode(KA1, OUTPUT);
  vc101.attach(VC1, 730, 2300);
  vc102.attach(VC2);
  vc101.write(min101);
  vc102.write(min102);
}

void loop(){
  digitalWrite(KA1, "   ");  // HIGH o LOW
  vc101.write("      ");     // Entre min101 y max101
  vc102.write("      ");     // min102 o max102
}
```

4.1.4 Controlador basado en Arduino

En la Figura 19 se muestra el diagrama de bloques del controlador, esta generalidad aplica para el control individual de válvulas y bomba, los cuales responden ante un mismo error *e(t)*, que genera una señal *u(t)* particular hacia cada uno de ellos.

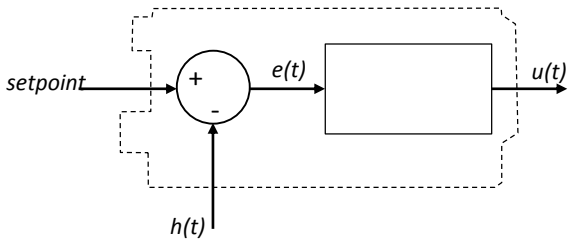


FIGURA 19 Diagrama general de bloques del controlador digital.

La acción de control *u(t)* hacia el relevador KA1 (arranque/paro) y el servomotor VC2 (abierto/cerrado) es de dos posiciones, mientras que para VC1 puede ser de una posición (totalmente cerrada) o de tipo PID (proporcional-integral-derivativo).

El valor de *setpoint* es fijado por el operador a través de un potenciómetro (R1) que funciona bajo la siguiente función llamada `sp()`:

```
int sp (float inSetpoint){      ..... Fn. (d)
    int leeSetpoint = analogRead(inSetpoint);
    int outSetpoint = map(leeSetpoint, 1023, 0, 4, 40);
    return outSetpoint;
}
```

La anterior función retorna un valor entre 4 a 40 que es asignado a la variable de `setpoint` dentro del algoritmo como se ejemplifica en el siguiente código:

```
#define R1 0        // Potenciómetro para setpoint

#define distanciaSensor 69.50   // H en Ec. (2)

int setpoint;

void setup(){
}

void loop(){
  setpoint = sp(R1);  // Se llama a la función sp()
}
```

4.1.4.1 Control de BA-101

Para la bomba BA-101 y su control de dos posiciones (arranque/paro), la señal de salida del controlador *u(t)* permanece en un valor de 0 V (`LOW`) si la señal de error *e(t)* en unidades absolutas es menor o igual a la tolerancia permisible del error (*tolError*) y además se mantenga así como mínimo un tiempo establecido por el operario (*tiempoBomba*), de lo contrario *u(t)* permanece en un valor de 5 V (`HIGH`).

La *tolError* es un margen de error preestablecido de 1.5 cm. El *tiempoBomba* es fijado por el operador a través de un potenciómetro (R2) bajo la función `onB()` que retorna un valor entre 5 y 15:

```
int onB(int inTiempoBomba){
  int leeTiempoBomba = analogRead(inTiempoBomba);
  int outTiempoBomba = map(leeTiempoBomba, 1023, 10, 5, 15);
  return outTiempoBomba;
}
```

..... Fn. (e)

Código de control para la bomba:

```
#define R1 0        // Potenciómetro para setpoint
#define R2 1        // Potenciómetro para tiempoBomba
#define pinTrig 25  // Pin Trig de HC1
#define pinEcho 27  // Pin Echo de HC1
#define KA1 29      // Relevador de estado sólido

#define distanciaSensor 69.50  // H en Ec. (2)
#define tolError 1             // Tolerancia permisible del error
#define deltaTiempo 1 // Intervalo de tiempo(s) para la salida del controlador PID

int setpoint, input, tiempoBomba, contador;
float error;

void setup(){
  pinMode(pinTrig, OUTPUT);
  pinMode(pinEcho, INPUT);
  pinMode(KA1, OUTPUT);
}

void loop() {
  input = ip(pinTrig, pinEcho); // Se llama a la función ip()
  setpoint = sp(R1);            // Se llama a la función sp()
  tiempoBomba = onB(R2);        //Se llama a la función onB()
  error = setpoint - input;
  if(abs(error) <= tolError){
    contador ++;
  }
  else {
    contador = 0;
  }
  if(contador*deltaTiempo >= tiempoBomba*60){
    digitalWrite(KA1, LOW);  // Paro
    delay(5000);             // Espera 5 s. Protección paro-arranque para el motor
  }
  else{
    digitalWrite(KA1, HIGH); // Arranque
  }
```

```
  delay(1000); // Espera 1 s
}
```

Las funciones `ip()` y `sp()`llamadas dentro del algoritmo, son las respectivas Fn. (c) y (d) que se han descrito con anterioridad. El algoritmo tiene una espera de 1 s por cada ciclo, por ello, la variable `contador` tendría esta misma unidad y al ser comparada con la variable `tiempoBomba`, esta última es multiplicada por 60 haciendo referencia de los segundos que hay en un minuto. También debe notarse que el programa anterior mantiene una espera de 5 s entre paro y arranque (`LOW`/`HIGH`) con la finalidad de proteger el motor de la bomba BA-101 de un apagado y encendido rápidos.

4.1.4.2 Control de VC-101

La válvula de control VC-101 modifica el porcentaje de su cierre ya sea ante una acción de control de tipo PID o de una posición totalmente cerrada, dependiendo del nivel (*h*) en el cual se encuentre el líquido respecto al valor de referencia (*setpoint*). La acción de control PID se ejecuta dentro y por arriba de un volumen de control, de altura constante (5cm) por debajo del valor de *setpoint*, como se muestra en la Figura 20, mientras que la acción de control con posición totalmente cerrada se ejecuta siempre por debajo del volumen de control.

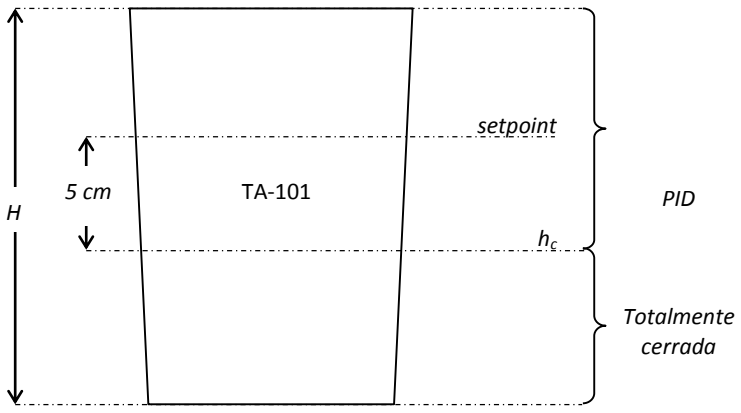


FIGURA 20 Acciones de control para VC-101 respecto al valor de *setpoint*.

La acción de control PID se genera con la siguiente ecuación:

$$u(t)=K_p e(t)+K_i \int_0^t e(t)dt+K_d \frac{de(t)}{dt} \quad \text{..... Ec. (4)}$$

en donde cada sumando corresponde en orden de aparición, a las acciones proporcional, integral y derivativa. K_p, K_i y K_d se denominan ganancias, las cuales son ajustables y constantes a cada sistema de control (Ogata, 1998). El control convencional PID es comúnmente utilizado en el control de nivel, pero los parámetros de estos controladores deben ser ajustados por el método apropiado de sintonización para satisfacer sus actuaciones requeridas (Hasan *et al.*, 2011).

Pese a que existen en la literatura métodos y reglas de ajuste para las ganancias de este tipo de controladores (Romero *et al.*, 2012; Ogata, 1998), al implementarse en sistemas reales, no necesariamente operan de manera cercana a la ideal como la generada por simuladores, o lo hacen solo bajo condiciones particulares. Es por ello y por la razón de que el presente trabajo no se enfoca al desarrollo de una técnica óptima de ajuste, que las ganancias se ajustan de manera empírica cuidando los siguientes aspectos:

a. La salida del controlador debe estar entre el intervalo de operación 0-75°.
b. Respetar un margen de error (*tolError*) de *setpoint* no mayor a 1cm una vez alcanzado el régimen permanente.
c. Minimizar en lo posible sobreoscilaciones en la respuesta de la planta.
d. Reducir en lo posible el tiempo de establecimiento a régimen permanente.

Las respuestas que se presentarán en las Figuras 21-25, se han simulado a partir del comportamiento isoporcentual del flujo de entrada (Q_i) al tanque TA-101 respecto al cierre de la válvula VC-101 (Figura 14), el error *e(t)* y la capacidad promedio de líquido en el TA-101 (1.62 L cm^{-1}); estas respuestas corresponden a la dinámica teórica de nivel de líquido, dentro del volumen de control. El valor de *setpoint*, es igual en todas las simulaciones, y como se pretende visualizar el comportamiento teórico del nivel de líquido, no se requiere presentar unidades. En el Anexo A, se describe el proceso matemático tabulado en Microsoft Excel 2010.

Como se muestra en la Figura 21, un controlador proporcional sería suficiente si el flujo de salida del tanque es de valor cero constante, de lo contrario habría una

diferencia de nivel entre el valor deseado (*setpoint*) y el nivel real de líquido, esta diferencia de nivel se conoce técnicamente como error de *offset*, y será tanto mayor como sea el flujo de salida.

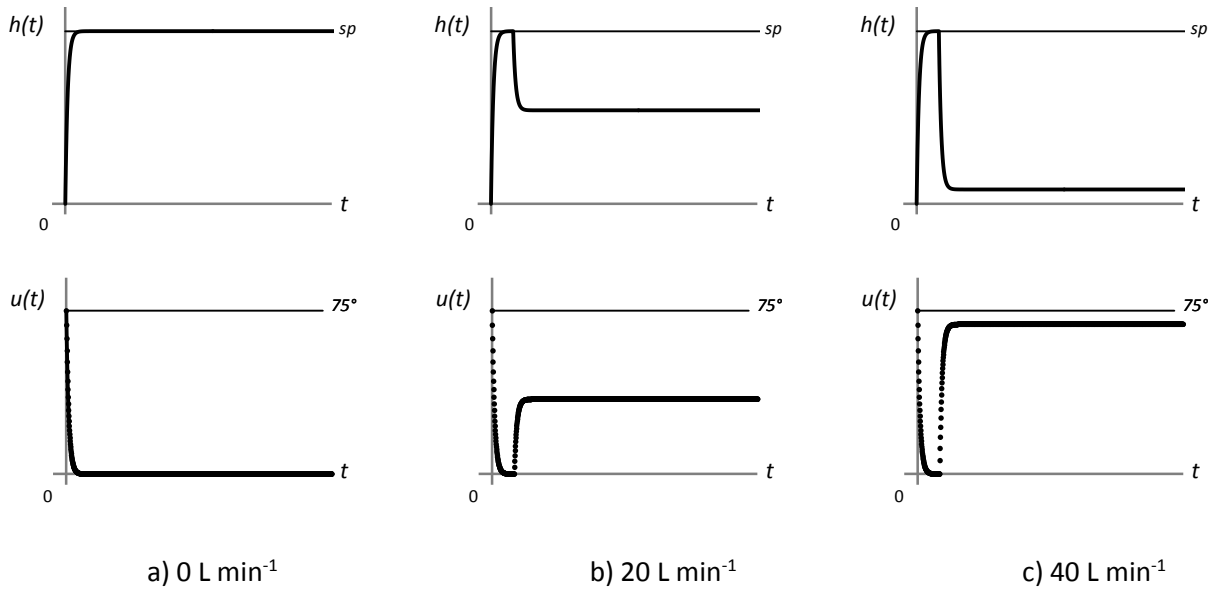
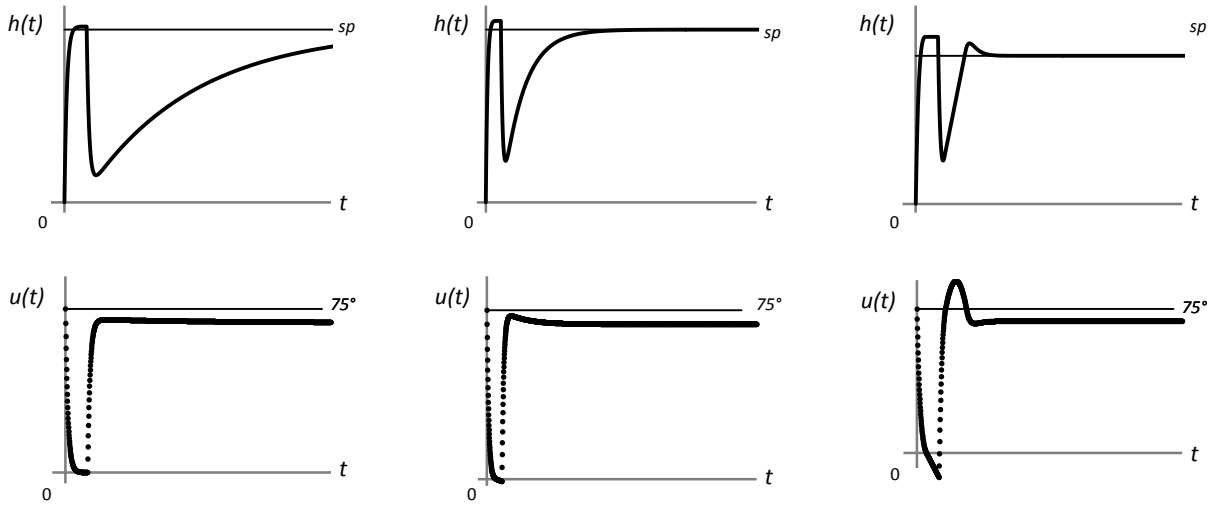


FIGURA 21 Comportamiento teórico del sistema de control por acción proporcional (K_p=15) en respuesta a una misma entrada escalón y distintas perturbaciones (0, 20 y 40 L min^{-1} de flujo de salida en el tanque) en iguales tiempos.

El proceso de ajuste de parámetros del controlador PID se propone complementando un controlador proporcional con un término integral, para luego, complementar el controlador proporcional-integral con un término derivativo cuidando los aspectos antes listados. Es de este modo, que en la Figura 22 se presentan las curvas *h(t)* y *u(t)* contra *t* en donde se muestra el ajuste empírico del término integral variando K_i con K_p =15 para el controlador de tipo PI.

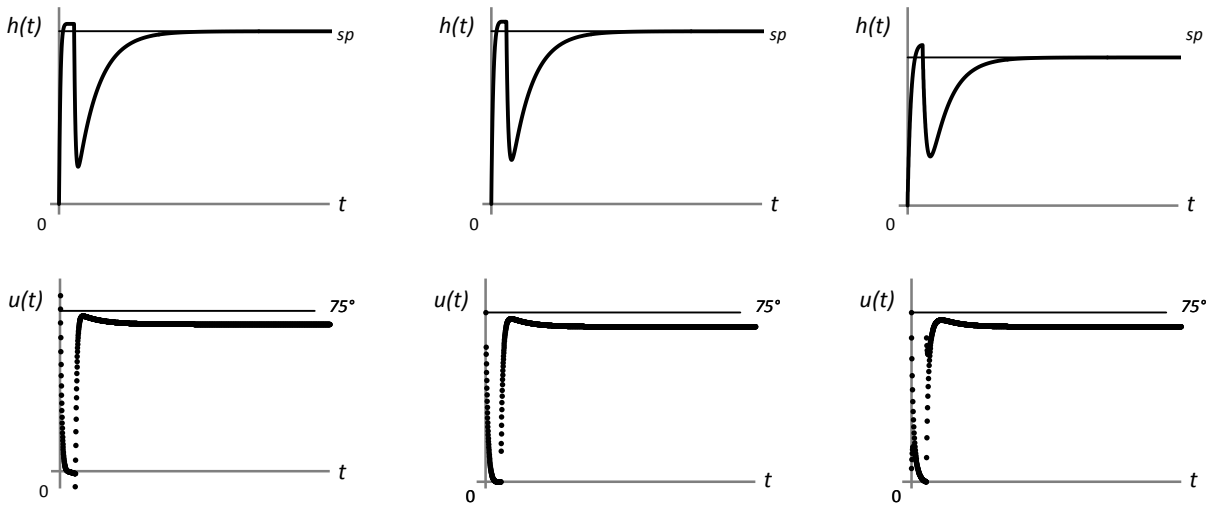


a) K_i = 0.03 b) K_i = 0.1 c) K_i = 0.35

FIGURA 22 Comportamiento teórico del sistema de control a una misma entrada escalón e igual perturbación máxima (Q_d = 40 L min^{-1} de flujo de salida en el tanque) en respuesta al ajuste del término integral con K_p=15.

En el ajuste del término integral se observa que al aumentar la ganancia K_i, el tiempo de establecimiento a régimen permanente reduce, sin embargo, el margen de error de *setpoint* tiende a aumentar antes de presentarse una perturbación y el valor de salida del controlador se aleja del intervalo deseado.

En la Figura 23 se presentan de igual manera las curvas *h(t)* y *u(t)* contra *t* en donde se muestra el ajuste empírico del término derivativo variando K_d con K_p=15 y K_i=0.1 para un controlador de tipo PID.

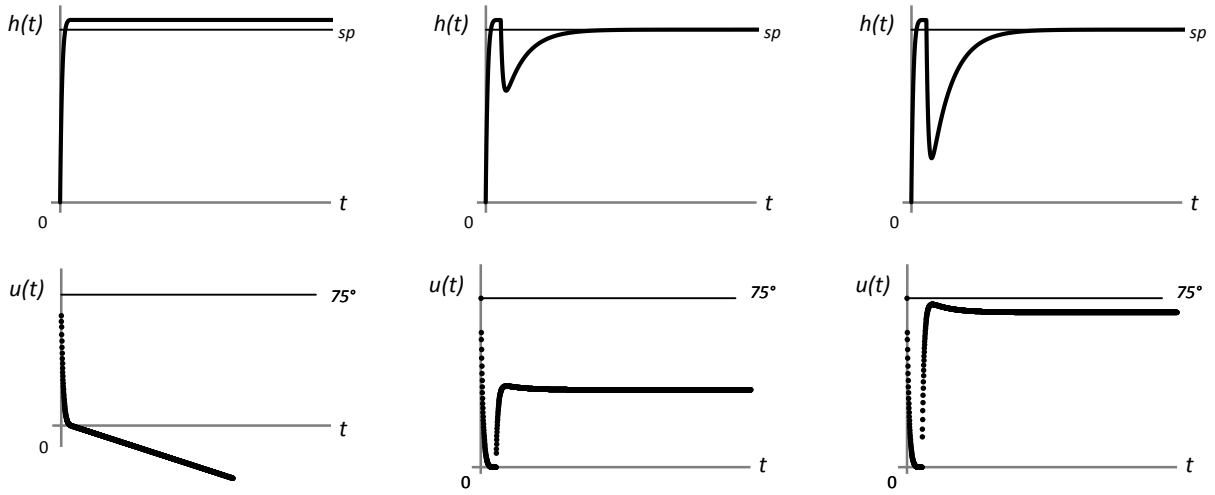


a) K_d = -30 b) K_d = 20 c) K_d = 140

FIGURA 23 Comportamiento teórico del sistema de control a una misma entrada escalón e igual perturbación (40 L min^{-1} de flujo de salida) en el tanque en respuesta al ajuste del término derivativo con K_p=15 y K_i=0.1.

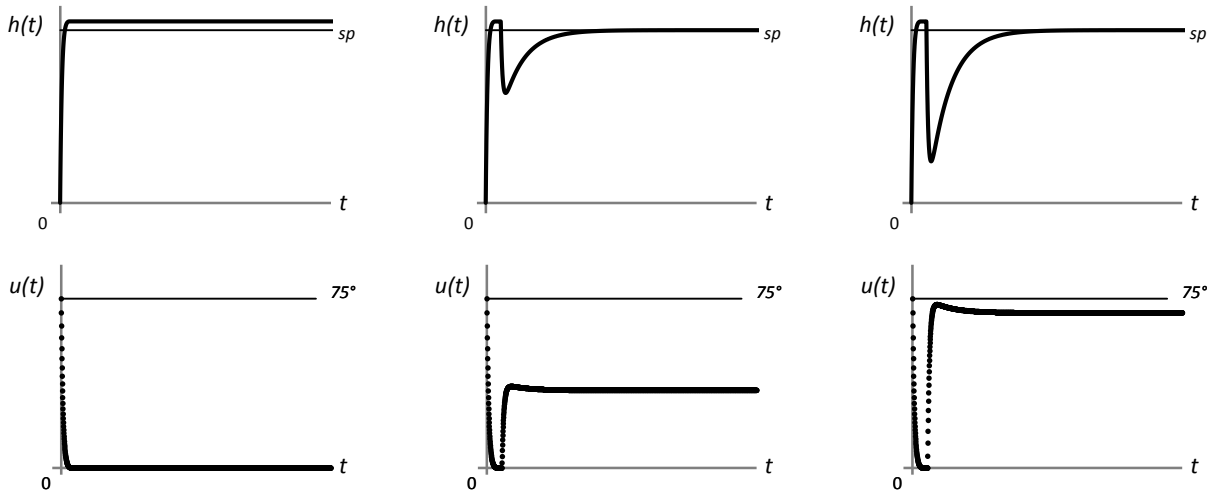
En el ajuste del término derivativo puede observarse que la variación de la ganancia K_d, no altera la salida de la planta, sin embargo, la salida del controlador sí se ve afectada de manera tal, que a valores demasiado pequeños se aleja del intervalo deseado, y a valores muy grandes muestra regiones en donde el controlador pareciera ser inestable.

A continuación se presenta el comportamiento teórico del sistema de control con modo PI y PID, Figuras 24 y 25 respectivamente, donde los comportamientos parecieran ser prácticamente iguales.



a) 0 L min^{-1} b) 20 L min^{-1} c) 40 L min^{-1}

FIGURA 24 Comportamiento teórico del sistema de control por acción proporcional-integral-derivativa (K_p=15, K_i=0.1, K_d=20) en respuesta a una misma entrada escalón y distintas perturbaciones (0, 20 y 40 L min^{-1} de flujo de salida) en el tanque en iguales tiempos.



a) 0 L min^{-1} b) 20 L min^{-1} c) 40 L min^{-1}

FIGURA 25 Comportamiento teórico del sistema de control por acción proporcional-integral (K_p=15, K_i=0.1) en respuesta a una misma entrada escalón y distintas perturbaciones (0, 20 y 40 L min^{-1} de flujo de salida) en el tanque en iguales tiempos.

Ambos controles (totalmente cerrada/PID) generan un posicionamiento a la válvula cada determinado tiempo preestablecido (*deltaTiempo*), la cual debe ser mayor o igual al tiempo por ciclo (1s). El siguiente es el código de control de VC-101:

```
#define R1 0        // Potenciómetro para setpoint
#define pinTrig 24  // Pin Trig de HC1
#define pinEcho 27  // Pin Echo de HC1
#define VC1 44      // Actuador de válvula VC-101

#include <Servo.h>
Servo vc101;

#define distanciaSensor 69.50
#define min101 0       // Ángulo en el actuador para apertura total en VC-101
#define max101 75      // Ángulo en el actuador para cierre total en VC-101
#define deltaTiempo 1  // Intervalo de tiempo en s para la acción del control
#define deltaHc 5      // Altura en cm del volumen de control
#define kp 15          // Ganancia proporcional
#define ki 0.1         // Ganancia integral
#define kd 20          // Ganancia derivativa

int setpoint, tiempo, ultTiempo;
float input, error, sumError, ultError, derivError, output;

void setup(){
  pinMode(pinTrig, OUTPUT);
  pinMode(pinEcho, INPUT);
  vc101.attach(VC1, 780, 2300); //Pulso para girar el servo de 0 a 180°
  vc101.write(min101);
}

void loop(){
  setpoint = sp(R1);
  error = setpoint - input;
  if((tiempo =! 0) && (error <= deltaHc)) sumError += error;
  if(tiempo == 0 || tiempo == ultTiempo + deltaTiempo){
    derivError = (error - ultError)/deltaTiempo;
    if(error > deltaHc){
      output = max101;
    }
    else {
    output = kp*error + ki*sumError + kd*derivError;
    if(output < 0) output = 0;                        // Impedir sobresaltos
    if(output > max101) output = max101;              // Impedir sobresaltos
    }
    vc101.write(output);
    ultTiempo = tiempo;
    ultError = error;
  }
  delay(1000);  // Tiempo de espera por cada ciclo (1 s)
  tiempo ++;
}
```

En el código, las funciones `ip()` y `sp()` que son llamadas dentro del algoritmo, son las respectivas Fn. (c) y (d) que se han descrito con anterioridad. Los términos derivativo e integral iniciales en el control PID son de valor cero: el valor 0 utilizado como dividendo, divisor o multiplicando genera este mismo valor. Aunque el valor calculado de la variable *output* resulte en un valor negativo, ésta no realiza este tipo de giro, por lo que cualquier valor de salida negativo ejecuta un valor cero, de manera similar, cualquier valor de salida mayor a 75 ejecuta un valor de 75.

4.1.4.3 Control de VC-102

Para la válvula de control VC-102 y su control de dos posiciones abierto/cerrado, la señal de salida del controlador *u(t)* permanece en un valor máximo (`max102`) si la señal de error es menor a menos una tolerancia permisible *tolVc102* constante de 1.5 cm, de lo contrario, *u(t)* permanece en un valor mínimo (`min102`). Al igual que en códigos anteriores, las funciones `ip()` y `sp()` que son llamadas dentro del algoritmo, son las respectivas Fn. (c) y (d).

```
#define R1 0        // Potenciómetro para setpoint
#define pinTrig 25  // Pin Trig de HC1
#define pinEcho 27  // Pin Echo de HC1
#define VC2 45      // Actuador de válvula VC-102

#include <Servo.h>  // Se llama a librería
Servo vc102;

#define distanciaSensor 69.50
#define min102 0               // Ángulo en el actuador para cierre de vc102
#define max102 130             // Ángulo en el actuador para apertura de vc102
#define tolVc102 1.5           // Tolerancia permisible (cm) para activar vcC102

int setpoint;
float input, error;

void setup(){
  pinMode(pinTrig, OUTPUT);
  pinMode(pinEcho, INPUT);
  vc102.attach(VC2);
  vc102.write(min102);
}

void loop() {
  input = ip(pinTrig, pinEcho);  // Se llama a la función ip()
  setpoint = sp(R1);             // Se llama a la función sp()
```

```
  error = setpoint - input;
  if(error < -tolVc102){
    vc102.write(max102);        // Abierta
  }
  else{
    vc102.write(min102);        // Cerrada
  }
}
```

4.2 Algoritmo de control y escenarios posibles

El diagrama de flujo presentado en la Figura 25, representa el algoritmo del controlador que permite operar la bomba y las válvulas de control de manera continua. En un inicio, en base a la exigencia misma del proceso, el operador especifica el valor de nivel deseado (*setpoint*) y el tiempo en el cual estará funcionando la bomba BA-101 (*tiempoBomba*) una vez alcanzado el *setpoint*. Especificados *setpoint* y *tiempoBomba*, el control inicia con la lectura del nivel en el tanque y así en cada ciclo:

a. La válvula de control VC-102 se mantendrá abierta si el error medido es menor al valor negativo de una tolerancia permisible (*tolVc102*), de lo contrario, ésta se mantendrá cerrada.
b. La válvula VC-101 tiene un cierre total si el error medido es mayor a la altura de control (Δh_c), de lo contrario, el porcentaje de cierre en la válvula VC-101 estará determinado por un control PID.
c. La bomba BA-101 encenderá y se mantendrá en este control mientras el valor absoluto del error sea menor o igual a *tolError* y *contador*++ sea menor a *tiempoBomba* de lo contrario, la bomba BA-101 permanece apagada.
d. Una vez apagada la bomba, para proteger su motor de un encendido inmediato, este control (apagado) se mantiene como mínimo 5 segundos.

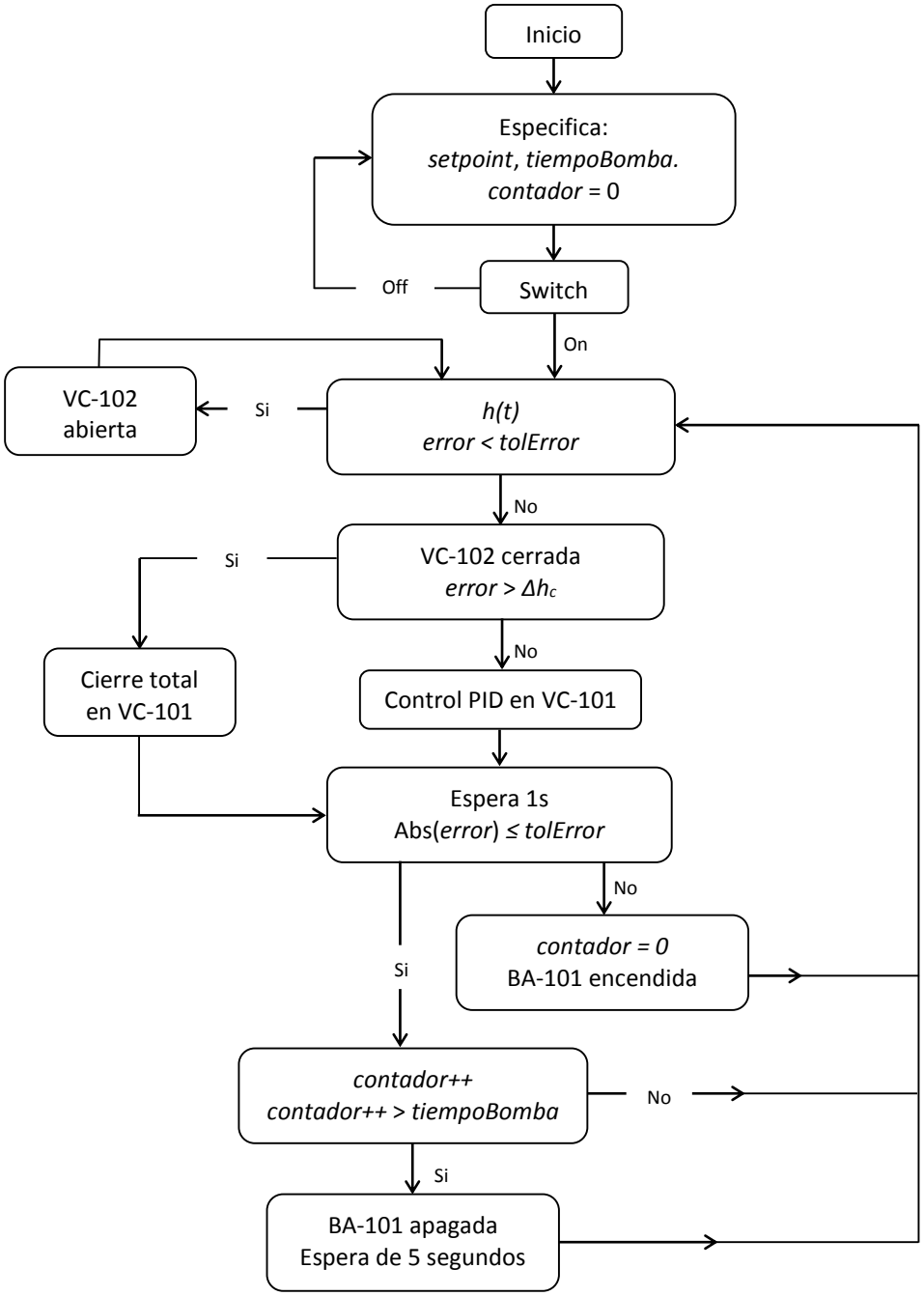


FIGURA 26 Diagrama de flujo que refiere a la lógica de programación del controlador.

Este algoritmo de control tiene como fin actuar de manera lógica e inmediata ante los escenarios a continuación:

i. Q_o=0 y *h<setpoint*. El primer escenario planteado sería cuando en la planta de proceso la perturbación sea nula (Q_o=0) y el nivel líquido se localice por debajo del valor deseado (*h<setpoint*).

ii. 0<Q_o<Q_d) y *h<setpoint*. El segundo escenario planteado sería cuando en la planta de proceso la perturbación es distinta de cero (0<Q_o<Q_d) y el nivel de líquido se localice por debajo del valor deseado (*h<setpoint*).

iii. Q_o=0 y *h>setpoint*. El tercer escenario planteado sería cuando en la planta de proceso la perturbación sea nula (Q_o=0) y el nivel de líquido se localice por arriba del valor deseado (*h>setpoint*).

iv. 0<Q_o<Q_d y *h>setpoint*. El cuarto escenario planteado sería cuando en la planta de proceso la perturbación es distinta de cero (0<Q_o<Q_d) y el nivel se localice por arriba del valor deseado (*h>setpoint*).

En el sistema de pruebas se ejecutarán de manera individual los escenarios i y ii, con *setpoint* arbitrario de 30 cm; se evaluará la conveniencia entre el modo PI y PID, además del efecto que tiene modificar el intervalo de ejecución de cada control, probando con 1s y prueba aparte cada 5s. Posteriormente se simularán los cuatro escenarios posibles con el control e intervalo de ejecución convenientes.

Resultados

A partir de los trabajos realizados que iniciaron con el análisis de la tecnología Arduino, como son: el desarrollo de ingeniería básica, el diseño de componentes en base a las necesidades espaciales del equipo de pruebas y el desarrollo del algoritmo de control para su funcionamiento a través de la placa Arduino (denominado CoNArd-01), se obtuvieron como productos:

1. *Diagramas DFP y DTI* en donde se muestra la dinámica propuesta del proceso de llenado hacia un tanque atmosférico (DFP) y su correspondiente control de nivel (DTI).
2. *Implementación de un sensor ultrasónico como medidor de nivel*, que interpreta con su programación la duración de recepción de sus ultrasonidos en datos de altura de líquido.
3. *Adaptaciones actuador-válvula* que tienen accionamiento inmediato en cuanto a impedir, pasar o regular el flujo a través de ellas.
4. *Diseño hidráulico de red de tuberías* en donde se muestra el arreglo de bomba, válvulas, tanques y tubería del sistema de pruebas.
5. *Diagrama bifilar* y su funcionalidad en acción con el equipo hidráulico producto de la ingeniería básica responden de manera sincrónica bajo el algoritmo de control programado. Trabajar con la placa electrónica Arduino Mega 2560 implicó utilizar 2 puertos analógicos de entrada de los 16 disponibles y 16 puertos digitales *I/O* (2 de ellos PWM) dejando libres 38 de ellos.
6. *Desarrollo de un algoritmo de control* que permite vincular entradas *e(t)* y salidas *u(t)* de información en el controlador. El tamaño binario del *sketch* es de 9,932 bytes de un máximo de 258,048 bytes, manipula 29 variables declaradas y utiliza 2 librerías.
7. *Equipo del sistema de control* conformado por el funcionamiento conjunto del sistema de pruebas y el prototipo de controlador de nivel (hardware y software).

A continuación se reportan el comportamiento y las respuestas generadas por la implementación del controlador, el cual, es sometido a distintas pruebas ante cuatro escenarios que abarcan desde el llenado convencional sin perturbaciones hasta un escenario que promueve continuamente disturbios durante el llenado y control del sistema.

Escenarios:

i. Q_o=0 e *input*<*setpoint*.
ii. 0<Q_o<Q_d e *input*<*setpoint*.
iii. Q_o=0 e *input*<*setpoint*.
iv. 0<Q_o<Q_d e *input*>*setpoint*.

Los resultados del escenario *i* se presentan en las Figuras 27 a 30. Los resultados del escenario *ii*, son los presentados en las Figuras 31 a 34; para tener una mejor visualización, se presentan a partir del volumen de control donde es ejecutado el control PID o PI, además de que se presenta la respuesta simulada (Sim.), comprobando la teoría con la que fueron ajustadas las ganancias K_p, K_i y K_d descrita en la Subsección 3.1.4.2.

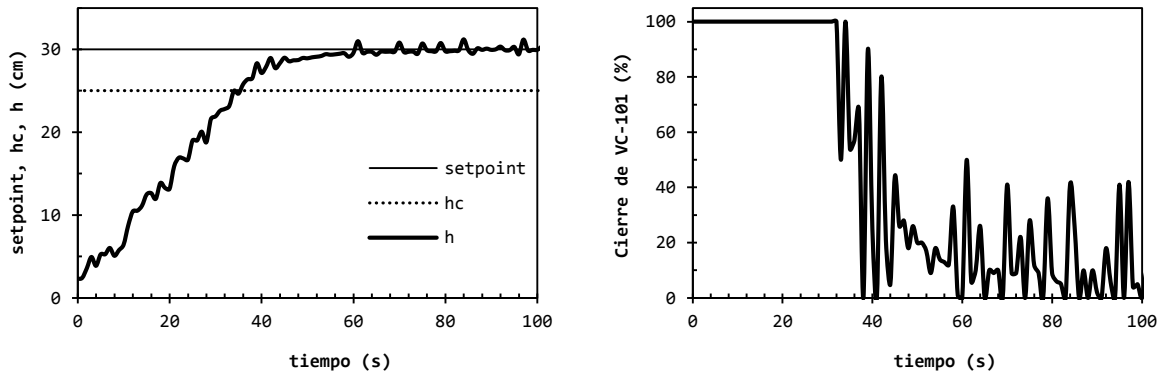


FIGURA 27 Respuesta del sistema de control ante una entrada escalón sin salida de flujo en el tanque y controlador PID ejecutado cada 1 s (escenario *i*).

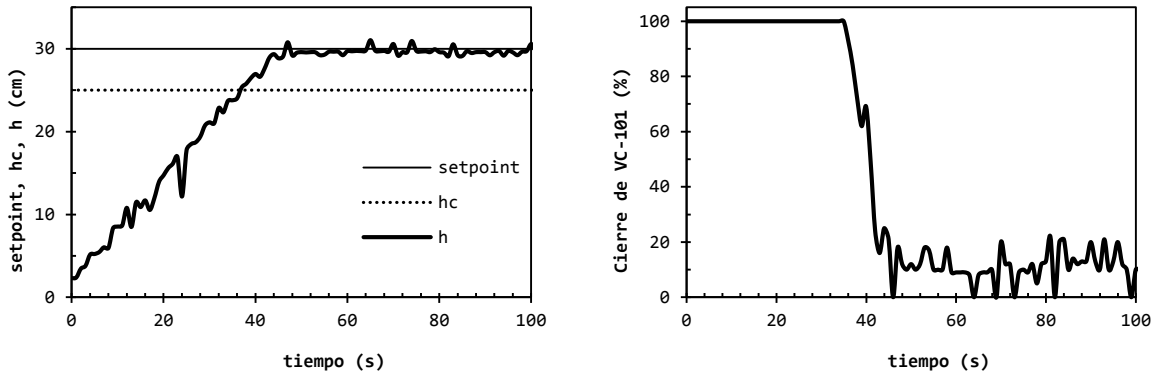


FIGURA 28 Respuesta del sistema de control ante una entrada escalón sin salida de flujo en el tanque y controlador PI ejecutado cada 1 s (escenario *i*).

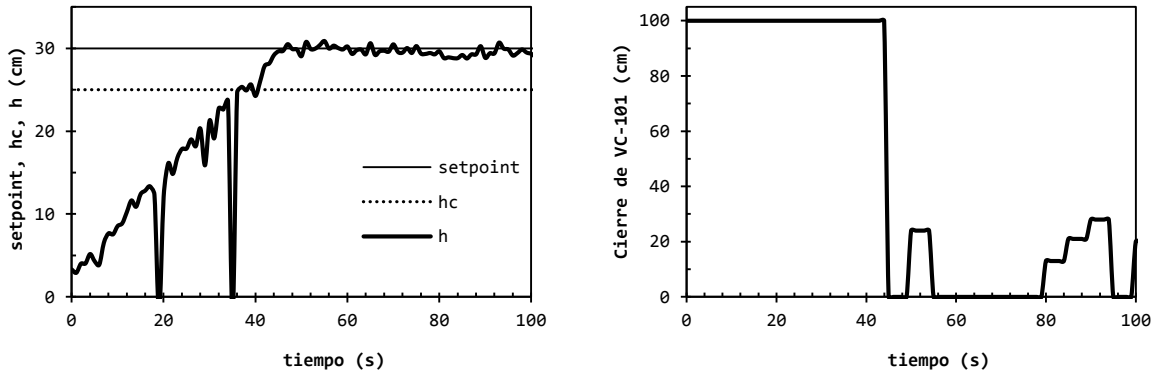


FIGURA 29 Respuesta del sistema de control ante una entrada escalón sin salida de flujo en el tanque y controlador PID ejecutado cada 5 s (escenario *i*).

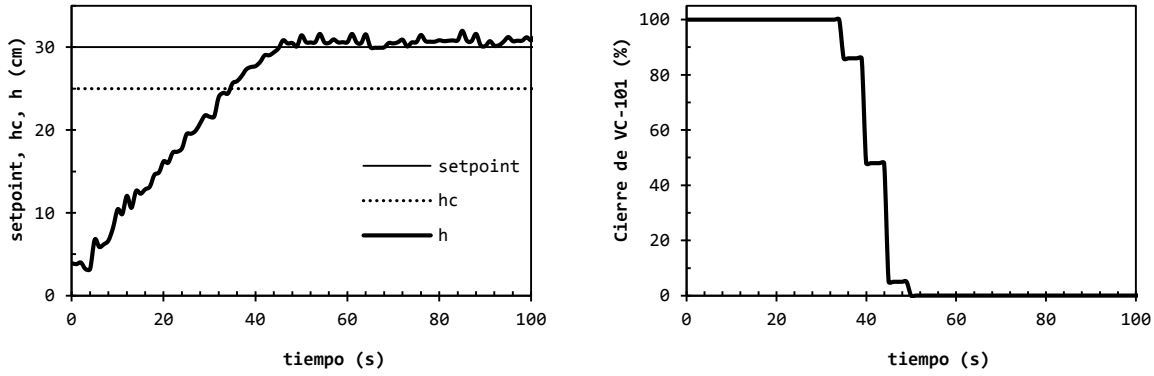


Figura 30 Respuesta del sistema de control ante una entrada escalón sin salida de flujo en el tanque y controlador PI ejecutado cada 5s (escenario *i*).

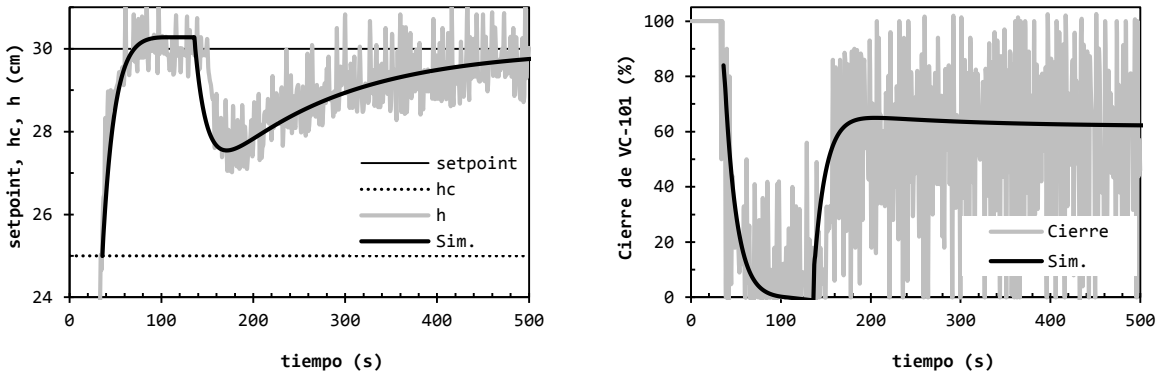


Figura 31 Respuesta del sistema de control ante una entrada escalón con salida de flujo en el tanque a los 100 s y controlador PID ejecutado cada 1 s (escenario *ii*).

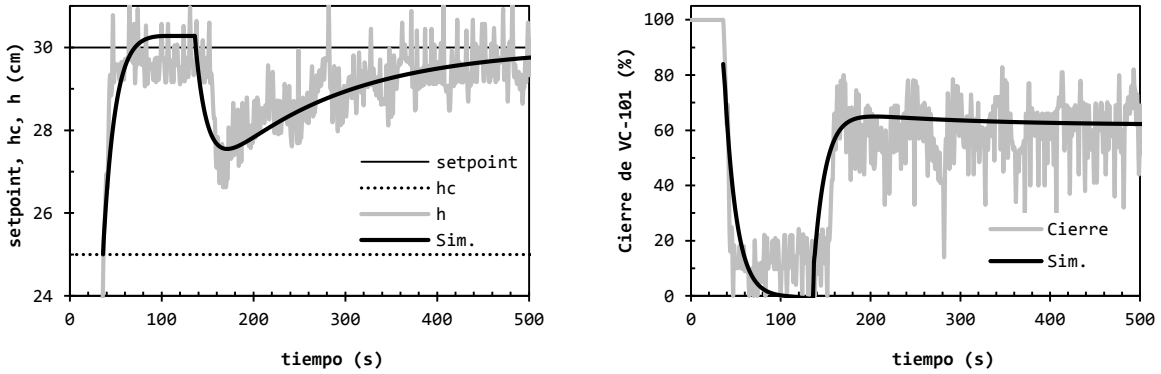


FIGURA 32 Respuesta del sistema de control ante una entrada escalón con salida de flujo en el tanque a los 100 s y controlador PI ejecutado cada 1 s (escenario *ii*).

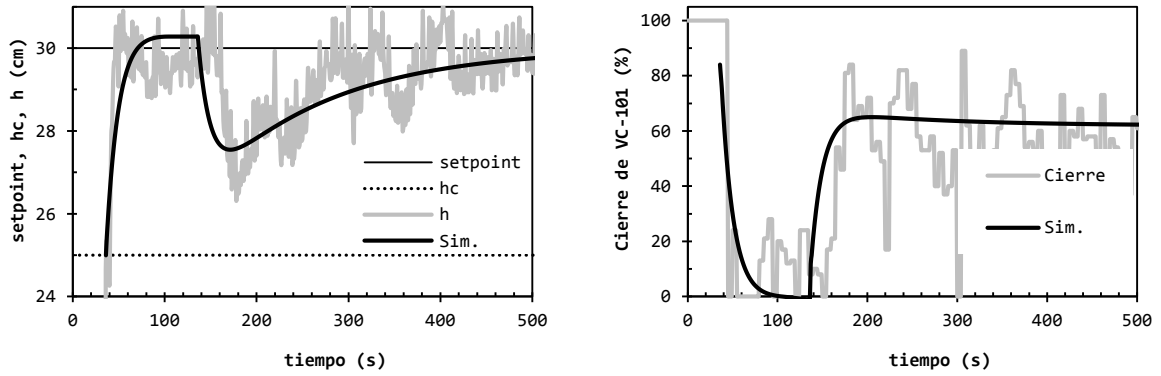


FIGURA 33 Respuesta del sistema de control ante una entrada escalón con salida de flujo en el tanque a los 100 s y controlador PID ejecutado cada 5 s (escenario *ii*).

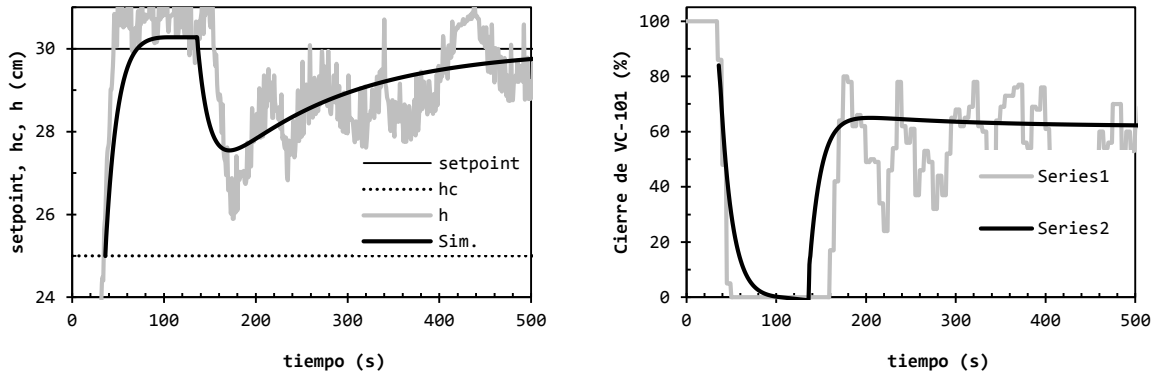


Figura 34 Respuesta del sistema de control ante una entrada escalón con salida de flujo en el tanque a los 100 s y controlador PI ejecutado cada 5 s (escenario *ii*).

En la Figura 35 se presenta la salida de la planta (*h*) ante cambios en Q_o y *setpoint*; se utiliza un controlador PI que se ejecuta una vez por ciclo en el algoritmo de control (cada 1s) y se presentan los cuatro escenarios posibles de manera continua. El proceso inicia con el escenario *i*, a los 150 s se presenta el escenario *iii*, a los 350 s el escenario *ii* y a los 1000 s el escenario *iv*.

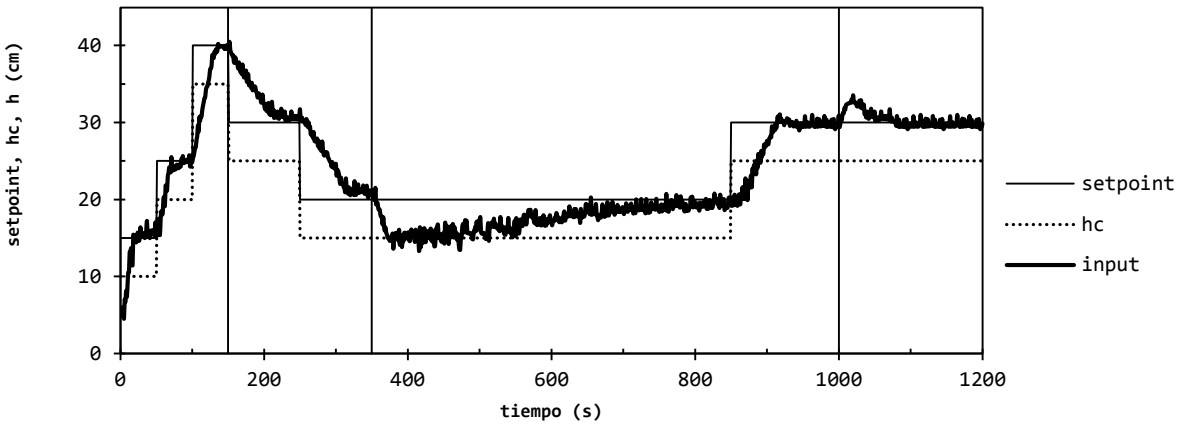


Figura 35 Respuesta del sistema de control ante variantes de Q_o y *setpoint* (inicia escenario: *i* en 0 s, *iii* en 150 s, *ii* en 350 s, *iv* en 1000 s) utilizando un controlador PI ejecutado cada 1s.

En cuanto a los resultados de la ejecución del algoritmo ante los escenarios planteados se tiene:

a. Producción de ruido en las lecturas del sensor ultrasónico, que aumenta dependiendo del oleaje provocado por la descarga de líquido en el tanque.
b. El incrementar el intervalo de ejecución del control PID o PI, puede sobrepasar el valor de *setpoint* por el ruido del sensor de nivel que puede dar una acción de control de una lectura altamente sobrepasada.
c. La acción derivativa que como fin tiene una acción anticipada, no se ve reflejada en la respuesta de la planta, pero sí en la acción del actuador, generando inestabilidades en este último.
d. A pesar del ruido en el sensor de nivel, *input* se mantiene en la tolerancia permisible cuando llega al valor de *setpoint* con control PI.
e. Parece conveniente trabajar con un controlador tipo PI ejecutado a intervalos cortos de 1s para la rectificación oportuna de lecturas sobrepasadas del sensor.
f. Un intervalo mucho menor estaría restringida por el accionamiento de los componentes a controlar, en este caso, por los servomotores conectados.
g. Aunque Q_o sea de valor máximo, el nivel de líquido no desciende más debajo de h_c, por la acción de control de válvula VC-101 totalmente cerrada por debajo del volumen de control y por la restricción $0<Q_o<Q_d$.

Conclusiones y recomendaciones

El presente trabajo concluye, en base al estudio, con los siguientes puntos:

1. La placa electrónica de Arduino puede considerarse como una opción viable y funcional para instaurar un algoritmo de control PID de nivel en tanques atmosféricos como el reportado en el presente trabajo de tesis.
2. El algoritmo desarrollado permite establecer una respuesta teórica y su correspondencia física desarrollada.
3. El consumo eléctrico del equipo hace económica su operación, ya que es alimentado por corriente eléctrica doméstica y un convertidor *AC/DC* de 12V y 1A.
4. Los resultados obtenidos a partir del funcionamiento del equipo de control basado en Arduino (CoNArd-01) ante los escenarios planteados:
   i. Q_o=0 e *input*<*setpoint*.
   ii. 0<Q_o<Q_d e *input*<*setpoint*.
   iii. Q_o=0 e *input*<*setpoint*.
   iv. 0<Q_o<Q_d e *input*>*setpoint*.

   muestran un comportamiento estable, por lo tanto son aceptables.
5. Los documentos propios de la ingeniería básica permiten reproducir el sistema de control físico para el estudio de control de nivel.
6. La aplicación de este prototipo de controlador representa una alternativa tangible que permite el estudio de los elementos de control de nivel y sus aplicaciones en ingeniería.
7. Esta alternativa de controlador refleja ser económica, funcional, flexible y confiable para el control automático de nivel en taques de proceso.

Recomendaciones:

1. El estudio podría complementarse con un controlador complejo que esté ajustado bajo un método formal de sintonización.

2. Las oscilaciones que se presentan en las lecturas del sensor podrían minimizarse con la instalación de mamparas en la descarga del líquido hacia el tanque.
3. Alimentar los dispositivos externos a la placa (servomotores, sensor, *display*, indicadores LED, etc.) con voltaje que no derive de ésta, es decir, no utilizar sus puertos de voltaje (5Vcc, Vin y GND), sobre todo si se desea expandir el proyecto.

Bibliografía

Aguilar, F. y Granados, V. (2013). Using open-source platform for trajectory control of DC motors. *Institute of Electrical and electronics Engineers*.

Alarcón, G. y Lajo, D. (2013). Diseño de un controlador PID con interfaz gráfica de control para un sistema de equilibrio bi-hélice. *Eleventh LACCEI Latin American and Caribbean Conference for Engineering and Technology*. Cancun.

Arduino. (2014). Obtenido de Arduino: http://www.arduino.cc

Arízaga, A., Calleja, J., Hernández, R. y Benitez, A. (2012). Automatic control for laboratory sterilization process based on Arduino hardware. *IEEE*, 130-133.

Bautista, R., Domínguez, R. y Domínguez, J. (2010). ¿El uso de microcontroladores de 8-bits es una opción práctica en sistemas de control automático relativamente complejos? *Congreso Anual 2010 de la Asociación de México de Control Automático*.

Cheng, D. y Mok, A. (2001). Developing new generation of process control system. *IEEE Real-Time Embedded System Workshop*.

De Pablo, M. y De Pablo, C. (2010). Ardudrop 1.0: dispositivo electrónico para el estudio de la humedad del suelo. *Tecnologí@ y desarrollo*.

Desborough, L. y Miller, R. (2002). Increasing customer value of industrial control performance monitoring -honeywell's experience. *AIChE syposium series*, 169-189.

Donate, A. H. (1995). *Electrónica digital práctica: tecnología y sistemas*. Barcelona: Marcombo.

Earl, B. (06 de Octubre de 2013). *Sous-vide controller powered by Arduino. The SousViduino*. Obtenido de Adafruit Learning System: http://learn.adafruit.com/sous-vide-powered-by-arduino-the-sous-viduino?view=all

Elecfreaks. (2013). *HC-SR04 ultrasonic module user guide.* Recuperado el 2014, de Elecfreaks: http://www.elecfreaks.com/5464.html

Evans, B. (2011). *Beginning Arduino programming.* Apress.

Fritzing. (2014). Obtenido de Fritzing electronics made easy: www.fritzing.com

Giri, G., Moses, K., Suresh, C. y Jagadeesh, K. (2013). *Water tank depth sensor using Arduino-LabVIEW interface.* India: Department of Electrical and Electronics Engineering.

Hasan, S., Kabir, S. y Alam, S. (2011). Design and implementation of a neural control system and performance characterization with PID controller for water level control. *International Journal of Artificial Intelligence and Application*, 79-88.

Jacobs, E. (2013). *UberFridge: controlling beer fermentation temperature from a web interface.* Obtenido de Uberfridge: http://www.elcojacobs.com/uberfridge

Jyothish, S. Y. (2013). Design and implementation of fuzzy controller on embedded computer for water level control. *International Journal of Scientific and Engineering Research*, 51-56.

Naveenkumar, R. y Krishna, P. (2013). Low cost data acquisition and control using Arduino prototyping platform and LabVIEW. *International Journal of Science and Research*, 366-369.

Ogata, K. (1998). *Ingeniería de control moderna.* México: Pearson.

Romero, J., O., A., Padula, F., G., R., Garcia, S. y Balaguer, P. (2012). Estudio comparativo de algoritmos de auto-ajuste de controladores PID. *Revista Iberoameticana de Automática e Informática industrial*, 182-193.

Sethuramalingam, T. K. y Karthighairasan, M. (2012). Automatic gas valve control system using Arduino hardware. *Bonfring International Journal of Power System and Integrated Circuits*, 18-21.

Smith, C. A. y Corripio, A. B. (1991). *Control automático de procesos.* México: Limusa.

WordPress. (2009). Recuperado el 2013, de Proyecto Arduino: http://proyectoarduino.wordpress.com/

Anexo A

Dinámica teórica del nivel de líquido ante control PID

1. A partir de la Figura 13 y el acoplamiento de los ejes actuador-mariposa descrito en la Subsección 3.1.3, se genera la Figura A1.

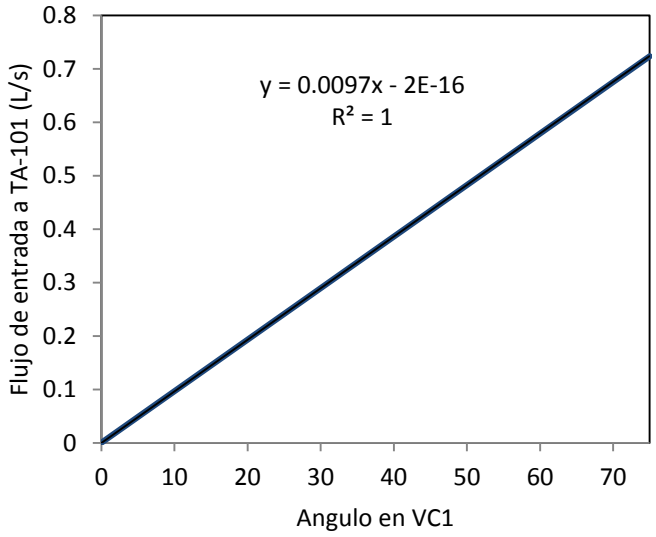


FIGURA 36 Comportamiento del flujo de proceso hacia TA-101 respecto al posicionamiento del actuador VC1.

2. Se establecen los parámetros con los que se simulará la dinámica, en la Tabla A1 se muestran.

Tabla 11 Parámetros para la dinámica teórica del nivel de líquido ante control PID.

L/cm	1.62
deltah (cm)	5
tSensa (s)	1
kp	15
ki	0.1
kd	20

- ✓ L/cm: capacidad del tanque TA-101.
- ✓ deltah: Δh_c.
- ✓ tSensa: tiempo de sensado.
- ✓ kp, ki, kd: ganancias proporcional, integral y derivativa, respectivamente.

3. Se establecen los valores iniciales con los que inician los cálculos, en la Tabla A2 se muestran.

Tabla 12 Valores iniciales para la dinámica teórica del nivel de líquido ante control PID.

tiempo	0
L	0
nivel	0
error	5
sumError	0
derivError	0
ang_ejec	75
L/s_entra	0.73
L/s_sale	>=0
L/s_total	L/s_entra – L/s_sale

4. El procedimiento de cálculo a partir de tiempo=1, es en el estricto orden que se muestra en la Tabla A3.

Tabla 13 Procedimiento de cálculo para la dinámica teórica del nivel de líquido ante control PID con tSensa=1.

1	tiempo	tiempo + tSensa
2	L (en tanque)	L + L/s_total
3	nivel (h(t))	L / (L/cm)
4	ultError	error
5	error	deltah – nivel
6	sumError	sumError + error
7	derivError	(ultError – error)/tSensa
8	ang_calc	Kp*error+ki*sumError+kd*deltaError
9	ang_ejec	*Si* ang_calc<0, ang_calc=0 *Si* ang_calc>75, ang_calc=75
10	L/s_entra	0.0097*cierre_ejec
11	L/s_sale	< 0.7 *(arbitrario)*
12	L/s_total	L/s_entra – L/s_sale

- ✓ tiempo: tiempo en ejecución.
- ✓ L: litros que contiene TA-101 en el tiempo de ejecución.
- ✓ nivel: nivel de líquido en el tanque TA-101.
- ✓ ultError: ultimo error calculado.
- ✓ error: error de término proporcional.
- ✓ sumError: suma del error del término integral.
- ✓ derivError: derivada del error del término derivativo.
- ✓ ang_calc: ángulo calculado para VC1.
- ✓ cierre_ejec: ángulo teórico que ejecuta VC1. Se cuida que siempre sea igual a ang_calc.
- ✓ L/s entra: L/s que por ang_ejec, entra al tanque TA-101.
- ✓ L/s_sale: perturbación en el sistema, L/s que salen del tanque TA-101.
- ✓ L/s_total: velocidad de llenado del tanque TA-101.

5. Se repite el paso anterior tantas veces sea necesario, hasta alcanzar nivel = deltah.

El procedimiento con tSensa = 5, o algún otro valor entero mayor a 1, tiene mismos valores iniciales y sigue la misma lógica, pero al modificar los pasos 1, 4 y 8 se debe tener cuidado al tabular los datos (ver Tabla A4).

Tabla 14 Procedimiento de cálculo para la dinámica teórica del nivel de líquido ante control PID con tSensa>1 de valor entero.

1	tiempo	tiempo + 1
2	L (en tanque)	L + L/s_total
3	nivel (h(t))	L / (L/cm)
4	ultError	*Si* tiempo es múltiplo de tSensa: error correspondiente a tiempo=tiempo – tSensa
5	error	deltah – nivel
6	sumError	sumError + error
7	derivError	(ultError – error)/tSensa
8	ang_calc	*Si* tiempo es múltiplo de tSensa: Kp*error+ki*sumError+kd*deltaError
9	ang_ejec	*Si* ang_calc<0, ang_calc=0 *Si* ang_calc>75, ang_calc=75
10	L/s_entra	0.0097*ang_ejec
11	L/s_sale	< 0.7 *(arbitrario)*
12	L/s_total	L/s_entra – L/s_sale

En modo de ejemplo, en la Tabla A5 se presenta parte del resultado utilizando el procedimiento en Tabla A3; la Figura A2 deriva de este mismo procedimiento.

Tabla 15 Variables calculadas utilizando el procedimiento mostrado en la Tabla A3.

tiempo	L	nivel	ultError	error	sumError	derivError	ang_calc	ang_ejec	L/s_entra	L/s_sale	L/total
0	0.00	0.00		5	0	0.00	0.00	75.00	0.73	0	0.73
1	0.73	0.45	5.00	4.55	4.55	-0.45	59.74	59.74	0.58	0.00	0.58
2	1.31	0.81	4.55	4.19	8.74	-0.36	56.62	56.62	0.55	0.00	0.55
3	1.86	1.15	4.19	3.85	12.60	-0.34	52.29	52.29	0.51	0.00	0.51
4	2.36	1.46	3.85	3.54	16.14	-0.31	48.47	48.47	0.47	0.00	0.47
5	2.83	1.75	3.54	3.25	19.39	-0.29	44.90	44.90	0.44	0.00	0.44
6	3.27	2.02	3.25	2.98	22.37	-0.27	41.59	41.59	0.40	0.00	0.40
7	3.67	2.27	2.98	2.73	25.11	-0.25	38.53	38.53	0.37	0.00	0.37
8	4.05	2.50	2.73	2.50	27.61	-0.23	35.68	35.68	0.35	0.30	0.05
9	4.09	2.53	2.50	2.47	30.08	-0.03	39.55	39.55	0.38	0.30	0.08
10	4.18	2.58	2.47	2.42	32.50	-0.05	38.55	38.55	0.37	0.30	0.07
11	4.25	2.62	2.42	2.38	34.88	-0.05	38.22	38.22	0.37	0.30	0.07
12	4.32	2.67	2.38	2.33	37.21	-0.04	37.84	37.84	0.37	0.30	0.07
13	4.39	2.71	2.33	2.29	39.51	-0.04	37.50	37.50	0.36	0.30	0.06
14	4.45	2.75	2.29	2.25	41.76	-0.04	37.17	37.17	0.36	0.30	0.06
15	4.51	2.79	2.25	2.21	43.97	-0.04	36.87	36.87	0.36	0.30	0.06
16	4.57	2.82	2.21	2.18	46.15	-0.04	36.59	36.59	0.35	0.30	0.05
17	4.62	2.85	2.18	2.15	48.30	-0.03	36.33	36.33	0.35	0.30	0.05
18	4.68	2.89	2.15	2.11	50.41	-0.03	36.09	36.09	0.35	0.30	0.05
19	4.73	2.92	2.11	2.08	52.49	-0.03	35.86	35.86	0.35	0.30	0.05

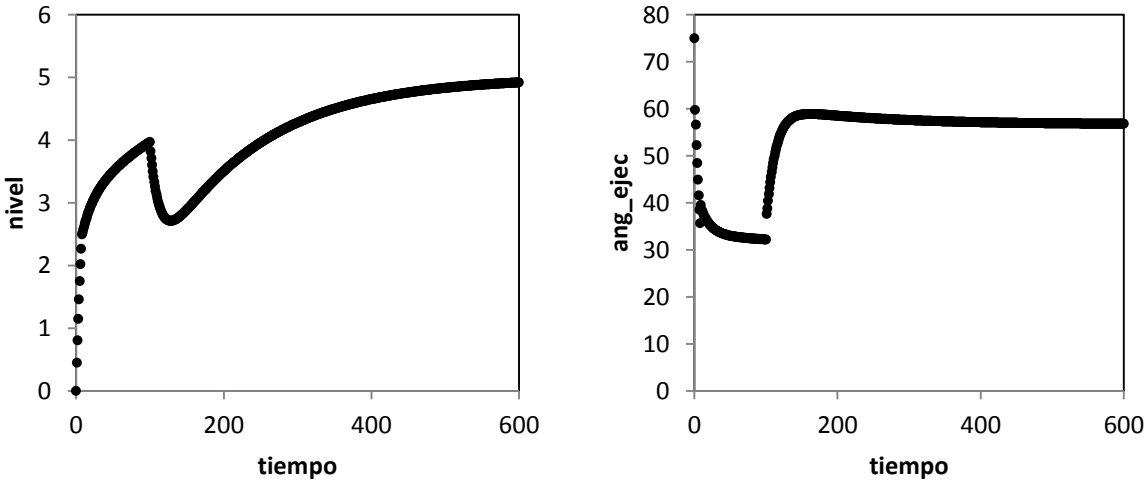


FIGURA 37 Comportamiento teórico del nivel de líquido y ángulo que ejecuta VC1, como resultado del procedimiento descrito en la Tabla A3.