UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE GUAYAQUIL

FACULTAD DE INGENIERÍAS

CARRERA DE INGENIERÍA ELECTRÓNICA MENCIÓN EN SISTEMAS COMPUTACIONALES

TESIS PREVIA A LA OBTENCIÓN DEL TÍTULO DE:

INGENIERO ELECTRÓNICO CON MENCIÓN EN SISTEMAS COMPUTACIONALES

TEMA:

"EVALUACIÓN DE LA PLATAFORMA ARDUINO E IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL DE POSICIÓN HORIZONTAL"

AUTORES:

CARLOS HIPÓLITO TAPIA AYALA
HECTOR MAURICIO MANZANO YUPA

TUTOR:
ING. RICARDO CAJO

GUAYAQUIL, OCTUBRE 2013

DECLARATORIA DE RESPONSABILIDAD

Nosotros CARLOS TAPIA AYALA y HECTOR MAURICIO MANZANO YUPA Alumnos de la Universidad Politécnica Salesiana, Facultad de Ingenierías, carrera de Ingeniería Electrónica, libre y voluntariamente declaramos que la presente tesis ha sido realizada en su totalidad por los Autores, por tal razón asumimos la Responsabilidad de la Autoría.

Guayaquil, octubre del 2013

CARLOS TAPIA AYALA

HECTOR MAURICIO MANZANO

DEDICATORIA

Dedico este proyecto y toda mi carrera universitaria a Dios porque me ha ayudado día tras día en seguir adelante. Le agradezco a mi madre Marina Yupa Acosta y a mi hermana Elizabeth Galarza Yupa porque ellas me dieron su apoyo en los momentos más difíciles de mi carrera, ese cariño y calor humano necesario, ya que ellas son las que han velado por mi salud, mis estudios y mi educación. A mis abuelos maternos ya que lamentablemente no los tengo con vida pero fueron el motor de mi niñez y adolescencia para poder ser una persona de valores y principios. A mi novia que me ayudó mucho en la culminación de este proyecto de tesis. En fin a todos los que están vinculados a mí y me ayudaron de forma desinteresada, gracias.

HECTOR MAURICIO MANZANO YUPA

Dedico la conclusión de este proyecto y toda mi carrera Universitaria a Dios y la Virgen María por darme la perseverancia, la lucidez, paciencia y sobre todo el conocimiento y la Sabiduría ya que el camino ha sido difícil hasta llegar al objetivo final. Le agradezco de manera puntual a mi madre Carmen Ayala Arguello, que sin ella definitivamente yo no sería una persona de bien y un profesional, porque gracias a su cariño, y comprensión he finalizado esta etapa de mi vida. También agradecer a mis Hermanos que sin ellos tampoco hubiese alcanzado la meta.

CARLOS HIPÓLITO TAPIA AYALA

ÍNDICE GENERAL

CAPÍTULO I. EL PROBLEMA

CAPÍTULO II. MARCO TEÓRICO

CAPÍTULO III. ELEMENTOS DE CONTROL Y CONSTRUCCÍON DEL DISENO MECÁNICO

ÍNDICE DE TABLAS

ÍNDICE DE GRÁFICOS

ÍNDICE DE ANEXOS

ABSTRACT

AÑO	ALUMNO/S	DIRECTOR DE TESIS	TEMA TESIS
2013	• CARLOS HIPÓLITO TAPIA AYALA • HECTOR MAURICIO MANZANO YUPA	ING. RICARDO CAJO	" EVALUACIÓN DE LA PLATAFORMA ARDUINO E IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL DE POSICIÓN HORIZONTAL"

En el desarrollo y avance de la tecnología electrónica de los microcontroladores existen múltiples modelos y diseños para cada uno de los problemas e innovaciones de las empresas, sin embrago muchas soluciones resultan demasiado costosas por su estructura y forma del proyecto. Por este motivo surge la necesidad de encontrar herramientas tecnológicas que ayuden al desarrollo del campo industrial y robótico por ende hemos dado a conocer la plataforma Arduino que en su diseño de hardware tiene un microcontrolador Atmega, entradas/salidas digitales, salidas PWM, entradas analógicas, conexión USB, botón reinicio y múltiples cualidades de la placa, de igual manera tiene su propio software de instalación donde será cargado la programación y enviado a la tarjeta. Por todas estas características favorables de la Plataforma Arduino hemos desarrollado la presente tesis: **"EVALUACIÓN DE LA PLATAFORMA ARDUINO E IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL DE POSICIÓN HORIZONTAL",** La implementación esta enlazado mecánicamente con un eje transversal al servomotor donde obedecerá las tareas enviadas por Arduino y comparadas con el sensor de posición, en el cual el alerón se moverá y formará el ángulo que se ordene, mediante una interfaz gráfica de Labview el usuario ingresará un valor en grados y escogerá el eje con respecto a las x, positivo o negativo para que se posicione el alerón.

PALABRAS CLAVES:

Microcontroladores, Plataforma Arduino, Atmega, programación, eje transversal, servomotor, sensor de posición, alerón, Labview.

INTRODUCCIÓN

En el desarrollo de la tecnología de los microcontroladores tenemos una gran variedad de tipos y modelos para cada uno de los problemas e innovaciones de las industrias y empresas en forma general, sin embargo muchas soluciones resultan muy costosas para nuestro medio, por lo que es necesario buscar otras alternativas y herramientas tecnológicas. Al realizar la implementación de tesis utilizaremos la tarjeta "Arduino uno" que consta de un microcontrolador Atmega 328 y este a la vez está diseñado con entradas analógicas, entradas y salidas digitales, entradas y salidas PWM y con una fuente de poder de 5 y 3.3 v dc.

El Sistema de posición horizontal está basado principalmente en un alerón que va rotar dependiendo del ángulo que se ordene. Va estar enlazada mecánicamente con el eje transversal donde estará ubicado el servomotor.

El diseño consta de las siguientes etapas:

Tarjeta Arduino (controlador digital)

Servomotor (fuerza)

Alerón (mecánico)

Sensor (adquisición de datos)

El sistema inicia cuando se ejecuta el software de Arduino, donde contiene la programación y tareas que se deben de cumplir cuando se ingrese los datos en la interfaz del usuario y a la vez se energizan los dispositivos electrónicos que intervienen en el sistema. Iniciado el Sistema, el sensor tomará la posición inicial 0°, paralelo a la superficie, además el usuario elegirá el cuadrante donde se desee que el alerón se incline formando el ángulo respectivo al eje de las x. EL usuario ingresará un número en grados, el Arduino enviará una señal de activación al servomotor, y este, a su vez comparará el dato ingresado por el usuario versus la señal tomada del sensor en tiempo real. Cuando los dos datos sean iguales, el servomotor se detendrá.

Cabe mencionar que en la interfaz Labview se visualizará el ingreso de datos (ángulo) y capturará la posición del sensor en un indicador gráfico.

CAPÍTULO I

EL PROBLEMA

1.1 PLANTEAMIENTO DEL PROBLEMA

Nosotros como estudiantes de la UNIVERSIDAD POLÍTECNICA SALESIANA damos a conocer bajo nuestra tesis una nueva herramienta tecnológica llamada "Arduino" a nivel de hardware y software enfocado al área de microcontroladores, bajo nuestra implementación incluimos el manejo, utilidad de un servomotor y el funcionamiento de un sensor que se basa en tecnología patentada de efecto Hall, la cual permite por primera vez la medición de ángulo/posición sin contacto y con eje pasante. Dentro de nuestro proyecto también utilizamos el software de programación grafica Labview que es muy utilizado por los estudiantes de Ingeniería electrónica y Eléctrica como herramienta para la adquisición de datos.

1.2 OBJETIVOS

1.2.1 Objetivo General

Diseñar e implementar un sistema de control de posición horizontal utilizando la plataforma Arduino.

1.2.2 Objetivos Específicos

- Analizar e interpretar el funcionamiento de hardware y software de la plataforma Arduino.
- Realizar prototipos de pruebas con la tarjeta Arduino y verificar el comportamiento de este con los distintos dispositivos a realizar la tesis.
- Desarrollar tareas en el software Arduino capaz de cumplir el funcionamiento entre la tarjeta y los componentes que actúan en el sistema de control.

- Implementar y explicar de modo practico-experimental la comunicación entre el software de simulación Labview (interfaz gráfica usuario) y la plataforma Arduino.

- Demostrar que al ingresar un número en grados, el alerón se detendrá cuando el servo motor detecte el ángulo obtenido.

- Verificar el funcionamiento del servo motor por órdenes enviadas por la tarjeta Arduino.

- Comprobar que el sistema debe ser capaz de corregir perturbaciones externas, al ingresar un número en grados, por ejemplo 30º, el usuario mueve el alerón el sistema debe oponerse al movimiento y finalmente buscar la posición en que se seteó.

- Fortalecer el desarrollo académico de la materia teoría control II, asignatura fundamental para los estudiantes de las carreras de ingeniería eléctrica y electrónica industrial.

1.3 JUSTIFICACIÓN

El avance de la Electrónica basado en los Sistemas de Control Autónomos, ha hecho que a diario se vayan desarrollando tarjetas e interfaces electrónicas que cumplan con las necesidades y requerimientos tecnológicos para las grandes industrias que tienen Sistemas que son controlados y operados por personas.

A nivel Mundial, Los Sistemas de Control Autónomos, en su infraestructura de hardware, los microcontroladores son el núcleo principal en el proceso y ejecución de tareas.

Las plataformas Arduino son microscontroladores, chips sencillos y de bajo costo que permiten el desarrollo de múltiples diseños de control. Al ser open-hardware

tanto su diseño como distribución es libre, pueden utilizarse libremente para el desarrollo de cualquier tipo de proyecto sin haber adquirido ninguna licencia.

En la comunidad educativa, la plataforma Arduino se está extendiendo y complementado enormemente en los últimos años para la enseñanza de diferentes materias tales como:

- Electrónica digital
- Programación en C/C++/java
- Microcontroladores
- Control automático

Por todas estas características favorables, proporcionaremos a la UNIVERSIDAD POLITECNICA SALESIANA en aportar a la investigación y desarrollo de una implementación basado en la plataforma Arduino para los estudiantes de las carreras técnicas, dándoles a conocer sus bondades y beneficios para la materia de microcontroladores.

1.4 MARCO METODOLÓGICO

- Realización del documento tesis, mediante la recopilación de la información Investigada.
- Diseño y construcción de la implementación que acopla lo mecánico, electrónico y programación; particular importancia tienen las pruebas realizadas por cada elemento de la estructura de la tesis.
- Elaboración y aplicación de los objetivos para la creación del documento de tesis, a partir de las pruebas ejecutadas en la implementación construida.
- Encuentros, reportes y sugerencias del tutor encargado de tesis.
- Elaboración de presupuestos y costos.
- Entrega de la documentación de tesis al consejo de carrera.

CAPÍTULO II

MARCO TEÓRICO

2.1 ARDUINO

Arduino es una plataforma de hardware de código abierto, basada en una sencilla placa de circuito impreso que contiene un microcontrolador de la marca "ATMEL" que cuenta con entradas y salidas, analógicas y digitales, en un entorno de desarrollo que está basado en el lenguaje de programación processing. El dispositivo conecta el mundo físico con el mundo virtual, o el mundo analógico con el digital controlando, sensores, alarmas, sistemas de luces, motores, sistemas comunicaciones y actuadores físicos.

Hay muchos otros microcontroladores y plataformas disponibles para la computación física donde las funcionalidades y herramientas son muy complicadas de programar Arduino simplifica el proceso de trabajar con microcontroladores, ofrece algunas ventajas y características respecto a otros sistemas.

Factible: Las placas Arduino son más accesibles y factibles comparadas con otras plataformas de microcontroladores.

Multi-Plataforma: El software de Arduino funciona en los sistemas operativos Windows, Macintosh OSX y Linux. La mayoría de los entornos para microcontroladores están limitados a Windows.

Ambiente de programación sencillo y directo: El ambiente de programación de Arduino es fácil de usar para los usuarios, Arduino está basado en el entorno de programación de processing con lo que el usuario aprenderá a programar y se familiarizará con el dominio de desarrollo Arduino.

Software ampliable y de código abierto: El software Arduino está publicado bajo una licencia libre y preparada para ser ampliado por programadores y desarrolladores experimentados. El lenguaje puede ampliarse a través de librerías de C++ y modificarlo a través del lenguaje de programación AVR C en el que está diseñado.

Hardware ampliable y de código abierto: Arduino está basado en los microcontroladores ATMEGA168, ATMEGA328 y ATMEGA1280. Los planos de los módulos están publicados bajo licencia *creative commons***,** por lo que los diseñadores de circuitos pueden hacer su propia versión del módulo, ampliándolo u optimizándolo facilitando el ahorro.

Figura 2.1: Tarjeta Arduino "Uno"

Fuente: http://Arduino.cc

2.2 CARACTERÍSTICAS DE LA PLACA ARDUINO

2.2.1 Hardware

Arduino en su diseño de hardware es una placa electrónica que se puede adquirir ensamblada o construirla directamente porque se encuentran los planos electrónicos y la licencia del producto en el internet.

Las placas han ido evolucionando como su software, al inicio las primeras placas utilizaban un chip FTDI "FT232RL" para comunicarse por puerto USB al computador y un procesador para ser programado, luego se utilizó un microcontrolador especial para cumplir esta función como en el caso de Arduino "uno", que tenían un micro para ser programado y otro para la comunicación, en la actualidad se usa un único microcontrolador que se compromete en llevar a cabo la comunicación y sobre el que también se descargan las instrucciones a ejecutar.

2.2.2 Descripción de la placa Arduino

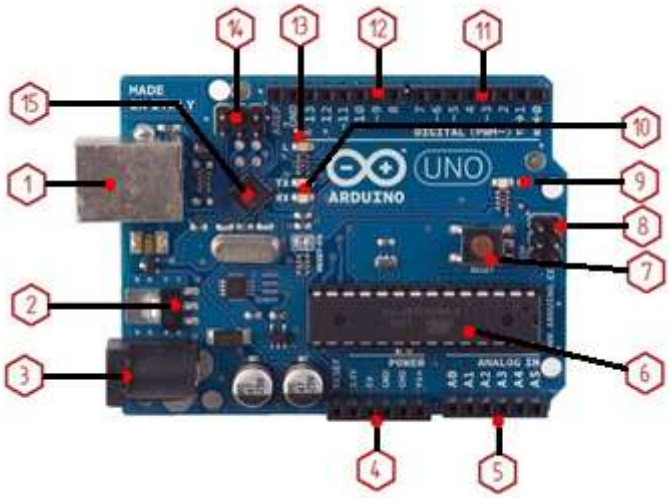


Figura 2.2: Descripción de los componentes de la placa Arduino "uno"

Fuente: Los Autores

1. Conector USB: proporciona la comunicación para la programación y la toma de datos, también provee una fuente de 5VDC para alimentar al Arduino, pero de baja corriente por lo que no sirve para alimentar motores gran potencia.

2. Regulador de voltaje de 5V: se encarga de convertir el voltaje ingresado por el plug 3, en un voltaje de 5V regulado necesario para el funcionamiento de la placa y para alimentar circuitos externos.

3. Plug de conexión para fuente de alimentación externa: Es el voltaje que se suministra que debe ser directo y estar entre 6V y 18Vo hasta 20V, generalmente se debe de tener cuidado de que el terminal del centro del plug quede conectado a positivo ya que algunos adaptadores traen la opción de intercambiar la polaridad de los cables.

4. Puerto de conexiones: Es constituido por 6 pines de conexión con las funciones de RESET que permite resetear el microcontrolador al enviarle un cero lógico. Pin 3.3V provee de una fuente de 3.3VDC para conectar dispositivos externos como en la protoboard por ejemplo. Pin 5V es una fuente de 5VDC para conectar dispositivos externos. Dos pines GND que permite la salida de cero voltios para dispositivos externos. Pin Vin, este pin está conectado con el positivo del plug 3 por lo que se usa para conectar la alimentación de la placa con una fuente externa de entre 6 y 12VDC en lugar del plug 3 o la alimentación por el puerto USB.

5. Puertos de entradas análogas: lugar donde se conectan las salidas de los sensores análogos. Estos pines solo funcionan como entradas recibiendo voltajes entre cero y cinco voltios directos.

6. Microcontrolador Atmega 328: Implementado con los Arduino uno en la versión SMD del Arduino uno R2, se usa el mismo microcontrolador pero en

montaje superficial, en este caso las únicas ventajas son la reducción del peso y ganar un poco de espacio.

7. Botón reset: Permite resetear el microcontrolador haciendo que reinicie el programa.

8. Pines de programación ICSP: son usados para programar microcontroladores en protoboard o sobre circuitos impresos sin tener que retirarlos de su sitio.

9. Led ON: enciende cuando el Arduino está encendido.

10. Leds de recepción y transmisión: se encienden cuando la tarjeta se comunica con el PC. El Tx indica transmisión de datos y el Rx recepción.

11. Puertos de conexiones de pines de entradas o salidas digitales: La configuración como entrada o salida debe ser incluida en el programa. Cuando se usa la terminal serial es conveniente no utilizar los pines cero (Rx) y uno (Tx). Los pines 3, 5 y 6 están precedidos por el símbolo ~, lo que indica que permiten su uso como salidas controladas por ancho de pulso PWM.

12. Puerto de conexiones 5 entradas o salidas adicionales: Las salidas 9, 10 y 11 permiten control por ancho de pulso; la salida 13 es un poco diferente pues tiene conectada una resistencia en serie, lo que permite conectar un led directamente entre ella y tierra. Finalmente hay una salida a tierra GND y un pin AREF que permite ser empleado como referencia para las entradas análogas.

13. Led pin 13: indica el estado en que se encuentra.

14. Pines de programación ICSP: son usados para programar microcontroladores en protoboard o sobre circuitos impresos sin tener que retirarlos de su sitio.

15. Chip de comunicación: Permite la conversión de serial a USB.

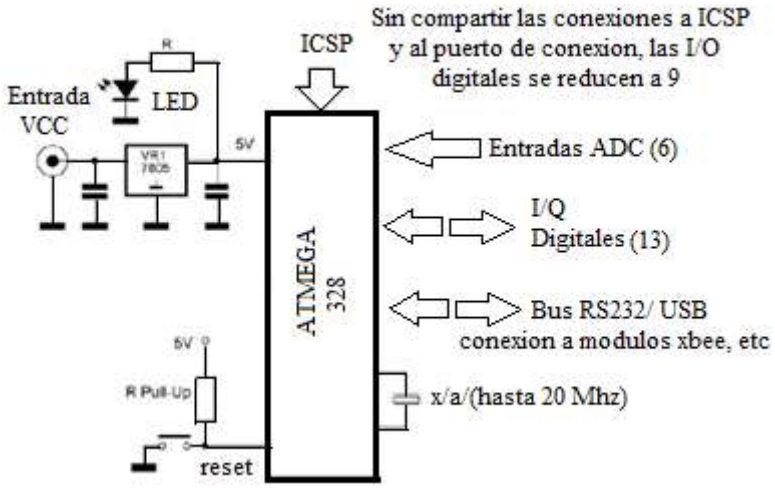


Figura 2.3: Diagrama de bloques sencillo de una placa Arduino

Fuente: Los Autores

2.3 ATMEL

Las tarjetas Arduino son placas que contienen un microcontrolador de la marca Atmel denominada AVR® 8-Bit RISC, esta línea de microcontroladores está formada por varios grupos, entre los cuales se encuentra Atmega. La diferencia entre miembros de una misma familia, radica en que para cada una de ellas puede variar el número y tipo de periféricos que incluyen la cantidad de memoria de programa y de datos.

La característica que tienen en común las familias pertenecientes a la línea AVR® 8-Bit RISC es que están basadas en la arquitectura AVR, mostrada en la figura 2.4.

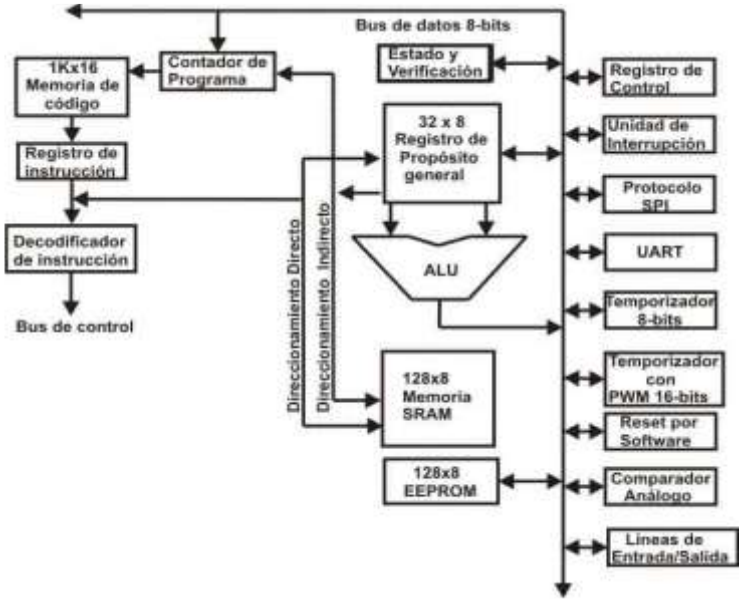


Figura 2.4: Diagrama de la arquitectura AVR

Fuente: http://www.sc.ehu.es

Los AVR son una familia de microcontroladores RISC en sus siglas significa (Reduced Instruction Set Computer) su diseño está basado en la arquitectura Harvard que se refiere a que la unidad central de proceso tiene por separado la memoria de programa y la memoria de datos. Los comienzos de la arquitectura AVR fué pensada por los estudiantes Alf Egil Bogen y Vegard Wollan en el Norwegian Institute of Technology, y posteriormente refinada y desarrollada en ATMEL en 1995.

El Núcleo del AVR cuenta con 32 registros en forma general, todos estos registros son conectados directamente a la unidad lógico-aritmética proporcionando que dos registros puedan ser localizados en una sola instrucción (en un ciclo de reloj), y que el resultado de la operación sea almacenado en uno de estos registros, lo que permite una mejor eficiencia y un mejor rendimiento al comparar con otros microprocesadores convencionales. El núcleo AVR cuenta con memoria de programa tipo flash, lo que implica que no se necesita una memoria externa para contener el código del programa. La memoria de programa puede ser programada de dos modos: en modo serial y en modo paralelo. En la mayoría de los dispositivos que

incluyen un núcleo AVR, la memoria de datos está constituida por una memoria EEPROM y una memoria RAM (SRAM).

Los procesadores AVR cuentan con algunos modos de operación, como el modo "Sleep y Powerdown", para ahorrar energía cuando el procesador no esté trabajando, además los procesadores AVR en su estructura realizan interrupciones internas y externas. Contiene un reset por software (wacthdog timer) con oscilador independiente, el cual es utilizado para su recuperación cuando suceda un problema de software, o también puede ser utilizado en algunas otras aplicaciones.

La mayoría de instrucciones son de 16 bits (2 bytes) de longitud en su código de operación y toman una localidad de la memoria de programa.

Figura 2.5: Microcontrolador ATMEGA 328

Fuente: http: //www.hobbytronics.co.uk

2.3.1 Sistema de memoria

El mapa de memoria para la familia de microcontroladores AVR, Como la mayoría de los microcontroladores está formado por una memoria de código y una memoria de datos.

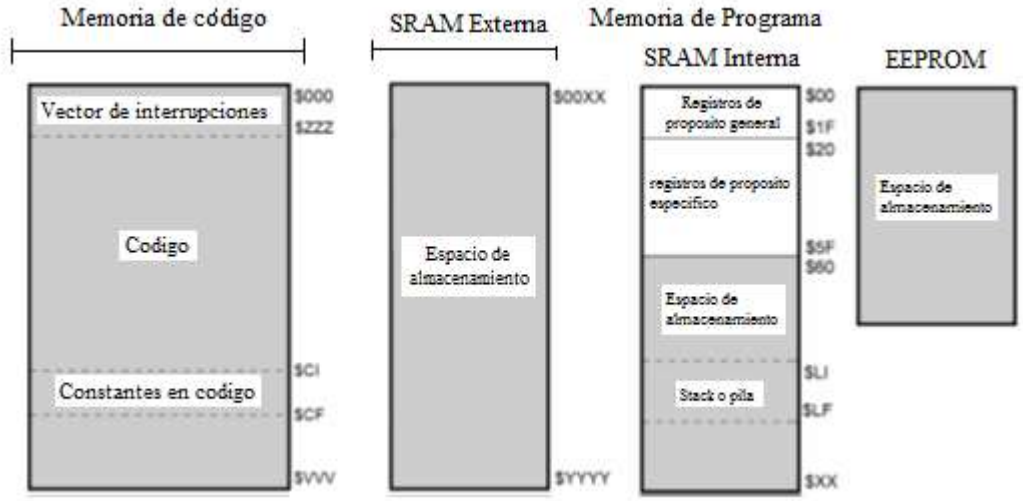


Figura 2.6: Mapa de memoria AVR

Fuente: Los Autores

2.3.1.1 Memoria de Código o Programa

La memoria de código en un microcontrolador AVR dentro la subfamilia MEGA puede contener entre los 4 kbytes y 256 kbytes y está formada en localidades de 16 bits, es direccionada por el contador de programa (PC, Program Counter) y tiene por función principal albergar las instrucciones para realizar una tarea específica. Como funciones alternativas, en el espacio de memoria contiene al vector de interrupciones y en este se pueden declarar constantes. Algunos microcontroladores AVR tienen un espacio que permite el auto-programado.

2.3.1.2 Memoria de datos

La memoria de datos tiene por función principal contener un espacio de almacenamiento temporal. La memoria de datos agrupa a tres bloques: SRAM interna, SRAM externa y EEPROM. Actualmente en el mercado está disponible el set de instrucciones AVR en diferentes dispositivos que comparten el mismo núcleo pero tienen distintos Periféricos y cantidades de RAM y ROM. La familia de Tiny AVR ATtiny11de microcontrolador contiene 1KB de memoria flash y sin RAM (sólo los 32 registros), y 8 pines, hasta el microcontrolador de la familia Mega AVRATmega2560 con 256KB de memoria flash, 8KB de memoria RAM, 4KB de memoria EEPROM, conversor análogo digital de 10 bits y 16 canales, temporizadores, comparador analógico, JTAG, etc.

Los microcontroladores AVR tienen dos etapas (cargar y ejecutar), que les permite ejecutar la mayoría en un ciclo de reloj, lo que los hace relativamente rápidos entre los microcontroladores de 8 bit.

2.3.2 Clasificación de los microcontroladores Atmel

ATXMEGA: Son procesadores muy potentes con 16 a 384 kB de memoria flash programable, encapsulados de 44, 64 y 100 pines, capacidad de DMA, eventos, criptografía y amplio conjunto de periféricos con DACs.

ATMEGA: Los microcontroladores AVR contienen 4 a 256 kB de memoria flash programable, encapsulados de 28 a 100 pines, conjunto de instrucciones extendido (multiplicación y direccionamiento de programas mayores) y amplio conjunto de periféricos.

ATTINY: son pequeños microcontroladores AVR con 0,5 a 8 kB de memoria flash programable, encapsulados de 6 a 20 pines y un limitado set de periféricos.

AT90USB: Atmega integrado con controlador USB

AT90CAN: Atmega con controlador de bus CAN

AT90S: tipos obsoletos, los AVRs clásicos

Modelos placas Arduino	**Modelos microcontroladores uC**
Arduino Due	AT91SAM3X8E
Arduino Leonardo	Atmega 32U4
Arduino uno	Atmega 328
Arduino Duemilanove	Atmega 328
Arduino Pro 3.3V/8MHz	Atmega 328
Arduino Pro 5V/16MHz	Atmega 328
Arduino Mega 2560 R3	Atmega 2560
Arduino Mega	Atmega 1280
Mega Pro 3.3V	Atmega 2560
Mega Pro 5V	Atmega 2560
Arduino Mini 05	Atmega 328
Pro Micro 5V/16MHz	Atmega 32U4
Pro Micro 3.3V/8MHz	Atmega 32U4
LilyPad Arduino 328 MainBoard	Atmega 328

Tabla 2.1: Modelos de placas Arduino / Modelos microcontroladores

Fuente: Los Autores

2.3.3 Microcontrolador Atmega 328

El microcontrolador ATMEGA 328 es un circuito integrado que contiene las partes funcionales de una Pc, como CPU, memorias (RAM) para datos, memorias (ROM, PROM, EPROM) para escribir el programa,pines de entrada y salida para la comunicación con el mundo exterior y algunas periféricos (comunicación serial, temporizador, convertidor A/D, etc.)

Se conoce que ATMEL fabrica los microcontroladores de la familia AVR, por lo que el microcontrolador es un chip que contiene memoria flash reprogramable. Las principales características de microcontroladores ATMEL son sus 28 pines y su estructura interna cuentan con 16k bytes de In-flash, sistema programable con lectura y escritura, 512 bytes de EEPROM, 1k bytes de SRAM, 23 líneas para propósito general I/O, 32 registros para propósito de trabajo general, temporizador y contadores.

El modo de la energía-abajo guarda el contenido del registro, pero se congela el oscilador, desactivando todas las funciones del chip de otra hasta el siguiente reinicio. En el modo de ahorro de energía, el contador asíncrono sigue funcionando, lo que permite al usuario mantener una base de temporizador, mientras que el resto dispositivo está durmiendo. Esto permite muy rápida puesta en marcha con el bajo consumo de energía.

Características físicas de los periféricos

- Interface serie SPI maestro/esclavo.
- ADC de 10 bit y 8 canales.
  - 8 canales de terminación simple
  - 7 canales diferenciales
  - 2 canales diferenciales con ganancia programable a 1x, 10x o 200x.
- 2 canales de PWM de 8 bit.

- Modulador de comparación de salida.
- 6 canales PWM con resolución programable desde 2 a 16 bits.
- Interface serie de dos hilos orientada a byte.
- Dos timer/contadores de 8 bits con prescaler separado y modo comparación.
- Dos timer/contadores extendidos de 16 bits con prescaler separado, modo Comparación y modo de captura.
- Comparador analógico On-Chip
- Contador en tiempo real con oscilador separado
- Doble USART serie programable

Voltajes de funcionamiento

- 2.7- 5.5V (ATMEGA 328L)
- 4.5- 5.5V (ATMEGA 328)

Niveles de velocidad

- 0 – 8 MHZ (ATMEGA 328L)
- 0 – 16 MHZ (ATMEGA 328L)

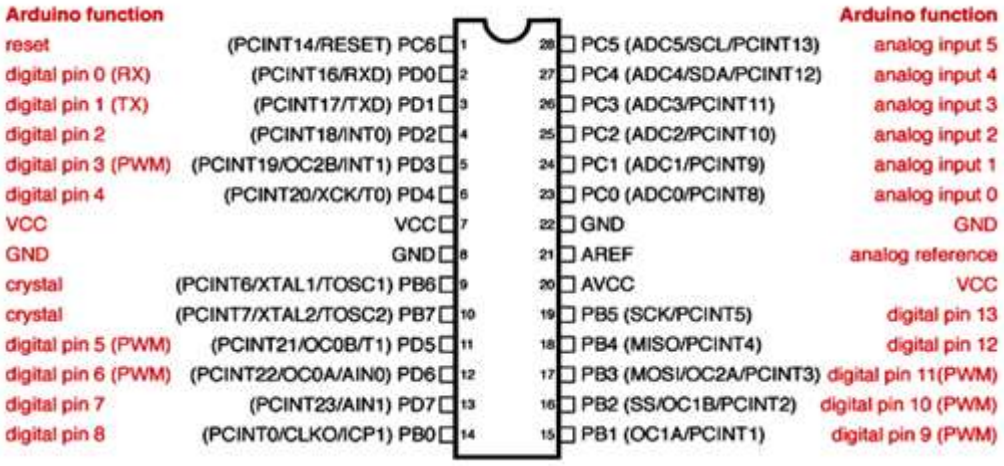


Figura 2.7: Mapa de pines microcontrolador Atmega 328

Fuente: http://Arduino.cc

2.4 MODELOS DE TARJETAS ARDUINO

En la comunidad Arduino existen una gran variedad de tarjetas Arduino creadas oficial y no oficiales creadas por terceros pero con características similares.

Las placas Arduino responden a las diferentes extensiones que se han venido realizando de las primeras placas de acuerdo con las demandas específicas de los usuarios y las investigaciones ya que son de tecnología abierta. La función de una u otra placa dependerá del tamaño del proyecto a realizar. Las placas Arduino más comunes utilizadas en el campo de la tecnología son (figura 2.8)

Figura 2.8: Modelos de las tarjetas Arduino

Fuente: http://Arduino.cc

2.4.1 Arduino uno

Arduino uno es una de las placas más utilizadas en los proyectos tecnológicos de robótica y contiene un microcontrolador ATmega328 que tiene 32 KB de memoria flash para almacenar el código de los cuales 0,5 KB es utilizado por el gestor de arranque. También dispone de 2 KB de SRAM y 1 KB de EEPROM ,cuenta con 14 entradas y salidas / digitales de los cuales 6 son utilizados como salidas PWM aparte tenemos 6 entradas analógicas, un cristal de 16 MHZ oscilador, una conexión USB, un conector de alimentación, una cabecera ICSP, y el botón de reinicio.

El diseño ha hecho posible que el microcontrolador se pueda conectar por medio de un cable USB al ordenador o el poder con un adaptador AC-DC o batería para empezar.

Voltaje de operación	5V
Voltaje de Entrada (recomendado)	7-12 v
Digital pines I / O	14 (de los cuales 6 proporcionan salida PWM)
Pines de entrada analógica	6
Corriente DC	40 mA
Corriente continúa	3.3V Pin 50 mA
Memoria Flash	32 KB (ATmega328) de los cuales 0,5 KB utilizado por gestor de arranque
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Velocidad del reloj	16 MHz.

Tabla 2.2: Características básicas de la placa Arduino uno

Fuente: Los Autores

Figura 2.9: Tarjeta Arduino

Fuente: http://Arduino.cc

2.4.2 Arduino Duemilanove

Arduino Duemilanove tuvo su lanzamiento en el año 2009 en Italia. Es una de las placas más populares dentro de las series con USB, puede contar con el microcontrolador Atmega328 o el ATmega168 que tiene 16 KB de memoria flash para almacenar el código de los cuales 2 KB se utiliza para el cargador de arranque y 1 KB de SRAM y 512 bytes de EEPROM. Tiene 14 pines con entradas/salidas digitales 6 de las cuales pueden ser usadas como salidas PWM, también tiene 6 entradas analógicas, un cristal oscilador a 16Mhz, una conexión USB, una cabecera ISCP y un botón de reset.

Voltaje de operación	5V
Voltaje de Entrada (recomendado)	7-12V
Pines Digital I / O	14 (de los cuales 6 proporcionan salida PWM)
Pines de entrada analógica	6
Corriente DC	40 mA
Corriente continua	3.3V 50 mA
Memoria flash	16 KB (ATmega168) o 32 KB (ATmega328) de las cuales 2 KB las usa el gestor de arranque(bootloader)
SRAM	1 KB (ATmega168) o 2 KB (ATmega328)
EEPROM	512 bytes (ATmega168) o 1 KB (ATmega328)
Velocidad del reloj	16 MHz

Tabla 2.3: Características básicas de la placa Arduino Duemilanove

Fuente: Los Autores

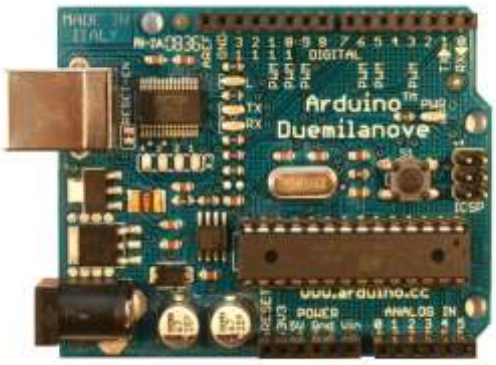


Figura 2.10: Tarjeta Duemilanove

Fuente: http://Arduino.cc

2.4.3 Arduino BT (bluetooth)

El Arduino BT es una de las placas diseñadas en el amplio mundo del hardware, originalmente se la construyó con el microcontrolador ATmega168, pero en la actualidad se equipa con el Atmega328, la función principal es la comunicación inalámbrica a través de Bluetooth de serie pero no es compatible con auriculares Bluetooth u otros dispositivos de audio. Cuenta con 14 pines de entradas / salidas digitales de las cuales 6 se puede utilizar como salidas PWM, 6 entradas analógicas, un oscilador de cristal de 16 MHz, terminales de tornillo para alimentación, una cabecera ICSP, y un botón de reinicio. El microprocesador se puede programar de forma inalámbrica a través de la conexión Bluetooth. La comunicación entre la placa Arduino con los ordenadores, teléfonos y otros dispositivos Bluetooth se la hace atreves del módulo Bluegiga WT11 este comunica con el ATmega328 vía serial (compartido con los pines RX y TX de la placa) y Viene configurado para la comunicación 115200 baudios. El módulo es detectable por los conductores del bluetooth de su sistema operativo y deber ser configurado porque este proporciona un puerto COM virtual para el uso de otras aplicaciones. El software de Arduino consta con un monitor de puerto serie que accede a los datos de texto simples para ser enviados hacia y desde la placa Arduino través de la conexión bluetooth.

Voltaje de operación	5V
Digital pines I / O	14 (de los cuales 6 proporcionan salida PWM)
Pines de entrada analógica	6
Corriente DC	40 mA
Corriente continua	3.3V 500 mA(con una fuente de energía capaz 1.5 A)
Corriente DC	5V 1000 mA
Memoria Flash	32 KB(de los cuales 2 KB usados por bootloader)
SRAM	2KB
EEPROM	1KB
Velocidad del reloj	16 MHz
BT modulo	2.1 WT1li-A-AI4

Tabla 2.4: Características básicas de la placa Arduino BT

Fuente: Los Autores

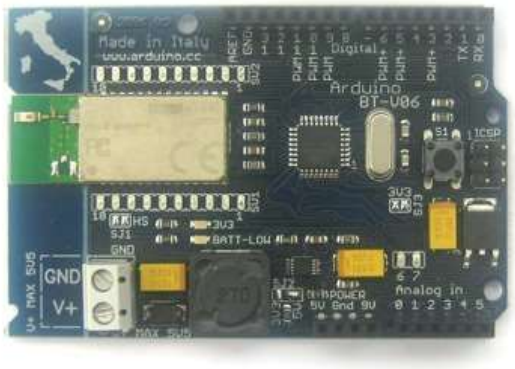


Figura 2.11: Tarjeta BT (bluetooth)

Fuente: http://Arduino.cc

2.4.4 Arduino Lilypad

La placa Arduino Lilypad es construida para el campo industrial de la aérea textil y diseñada para coser con un hilo conductor prendas y accesorios dinámicos e interactivos, se monta de manera igual las fuentes de alimentación, sensores y actuadores. Se basa en el microcontrolador ATmega168 o el ATmega328. La placa Arduino LilyPad ha sido desarrollada por Leah Buechley y la versión comercial del kit por "SPARKFUN" electrónica.

Voltaje de operación	2.7 a 5.5 V
Voltaje de entrada	2.7 a 5.5 V
Digital pines E / S	14 (de los cuales 6 proporcionan salida PWM)
Pines de entrada analógica	6
Corriente DC	40 mA
Memoria Flash	16 KB(de los cuales 2 KB usados por bootloader)
SRAM	1KB
EEPROM	512 bytes
Velocidad del reloj	8 MHz.

Tabla 2.5: Características básicas de la placa Arduino Lilypad

Fuente: Los Autores

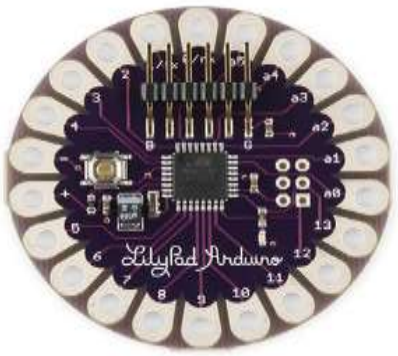


Figura 2.12: Arduino LilyPad

Fuente: http://Arduino.cc

2.4.5 Arduino Mega/2560

El Arduino Mega/2560 es una placa grande y más potente, electrónicamente esta basado en el microcontrolador Atmega 2560 tiene 256 KB de memoria flash para almacenar código de los cuales 8 KB se utiliza para el gestor de arranque, 8 KB de SRAM y 4 KB de EEPROM. Tiene 54 pines digitales de entrada / salida de los cuales 15 se pueden utilizar como salidas PWM, además 16 entradas analógicas, 4 puertas seriales, un oscilador de 16MHz, una conexión USB, un conector de alimentación, una cabecera ICSP, y un botón de reinicio. Para empezar a trabajar con el microcontrolador basta con conectarlo a un ordenador con un cable USB o el poder con un adaptador AC-DC o batería.A diferencia de las demás tarjetas Arduino esta puede funcionar con un suministro externo de 6 a 20 voltios.

Voltaje de Operación	7 a 12 V
Voltaje de Entrada (limites)	6-20 V
Digital pines E / S	54 (de los cuales 15 proporcionar salida PWM)
Pines de entrada analógica	16
Corriente DC	40 mA
Corriente CC	3.3 v 50 mA
Memoria Flash	256 KB(de los cuales 8 KB usados por bootloader)
SRAM	8KB
EEPROM	4KB
Velocidad del reloj	16 MHz.

Tabla 2.6: Características básicas de la placa Arduino Mega/2560

Fuente: Los Autores

Figura 2.13: Arduino Mega/2560

Fuente: http://Arduino.cc

2.4.6 Arduino Fio

La placa Arduino Fio está diseñado para aplicaciones inalámbricas, consta con un microcontrolador ATmega328P tiene 32 KB de memoria flash para el almacenamiento de código de los cuales 2 KB se utiliza para el gestor de arranque y Dispone de 2 KB de SRAM y 1 KB de EEPROM. Cuenta con 14 pines de entrada / salida digital de las cuales 6 se puede utilizar como salidas PWM, 8 entradas analógicas, un resonador de a bordo, un botón de reinicio y dispone un circuito de carga a través de USB e incluye para sus conexiones una batería de polímero de litio. Para la comunicación de la tarjeta el microcontrolador ATmega328P proporciona comunicación serie UART TTL, que está disponible en los pines digitales.

Voltaje de Operación	3.3 V
Voltaje de Entrada	3.35 - 12 V
Digital pines E / S	14 (de los cuales 6 proporcionar salida PWM)
Pines de entrada analógica	8
Corriente DC	40 mA
Memoria Flash	32KB(de los cuales 2 KB usados por bootloader)
SRAM	2KB
EEPROM	1KB
Velocidad del reloj	8 MHz.

Tabla 2.7: Características básicas de la placa Arduino Fio

Fuente: Los Autores

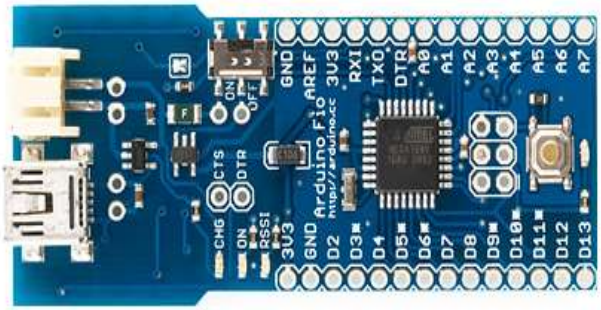


Figura 2.14: Arduino Fio

Fuente: http://Arduino.cc

CAPITULO III

ELEMENTOS DE CONTROL Y CONSTRUCCÍON DEL DISEÑO MECÁNICO

3.1 SERVOMOTOR

El servomotor es un dispositivo electrónico y mecánico en el cual está conformado por un motor de corriente continua, una caja reductora (engranajes) y un circuito de control que tiene la capacidad de ubicarse en cualquier posición dentro de su rango de operación y mantenerse estable en dicha ubicación. En las actividades diarias se utilizan los servos para posicionar superficies de control como el movimiento de palancas, pequeños ascensores, robots y timones. Los servomotores son pequeños, tiene internamente una circuitería de control y son sumamente poderosos en su torque para su tamaño.

Un servo estándar como el HS-311 de HITEC tiene 42 onzas por pulgadas o 3 kg por cm de torque, que es bastante fuerte para su tamaño como el que se está utilizando en nuestro proyecto.

Figura 3.1: Servomotor

Fuente: http: // Dungun-electronics.blogspot.com

Figura 3.2: Servomotor desmontado

Fuente: http:// platea.pntic.mec.es

3.1.1 Estructura interna

3.1.1.1 Motor de corriente contínua

El motor cc es una máquina que transforma la energía eléctrica en mecánica, provocando un movimiento rotatorio en el cual le brinda movilidad al servo. Cuando se aplica energía a sus dos terminales, este motor gira en un sentido a su velocidad máxima y la orientación de giro también se invierte si el voltaje aplicado en sus dos terminales es inverso.

3.1.1.2 Engranajes reductores

Los engranajes reductores convierten gran parte de la velocidad de giro del motor en torsión (torque).

3.1.1.3 Circuito de control

El circuito de control es el que asume la posición del motor. Recibe los pulsos de entrada y ubica al motor en su nueva posición dependiendo de los pulsos recibidos.

3.1.1.4 Terminales

Los servomotores tienen 3 terminales:

- Terminal positivo: recibe la alimentación del motor (4 a 6 voltios)
- Terminal negativo: referencia tierra del motor (0 voltios)
- Entrada de señal: recibe la señal de control

3.1.2 Funcionamiento

La forma del funcionamiento del servo depende del circuito de control y el potenciómetro (una resistencia variable) en la cual está conectada al eje central del servo motor. Este potenciómetro autoriza a la circuitería de control supervisar el ángulo actual del servo. Si el eje está en el ángulo correcto el motor gira en la dirección adecuada hasta llegar al ángulo designado. Un servo normal se utiliza para controlar un movimiento angular de entre 0 y 180 grados, no es mecánicamente capaz de retroceder a su lugar si hay un mayor torque que el sugerido por las especificaciones del fabricante.

El voltaje de alimentación puede estar comprendido entre los 4 y 8 voltios. El ángulo de ubicación del motor depende de la duración del nivel alto de la señal. Cada servo motor, dependiendo de la marca y modelo utilizado, tiene sus propios márgenes y características de operación.

El ángulo está establecido por la duración de un pulso que se aplica al sistema de control a esto nos referimos como PCM (modulación codificada de pulsos). El servo espera ver un pulso cada 20 milisegundos y la longitud del pulso determinara los giros del motor. Por ejemplo un pulso de 1.5 ms, hará que el motor se torne a la posición de 90 grados (llamado posición neutra). Si el pulso es menor de 1.5 ms, entonces el motor se acercara a los 0 grados. Si el pulso es mayor de 1.5 ms, el eje se acercara a los 180 grados.

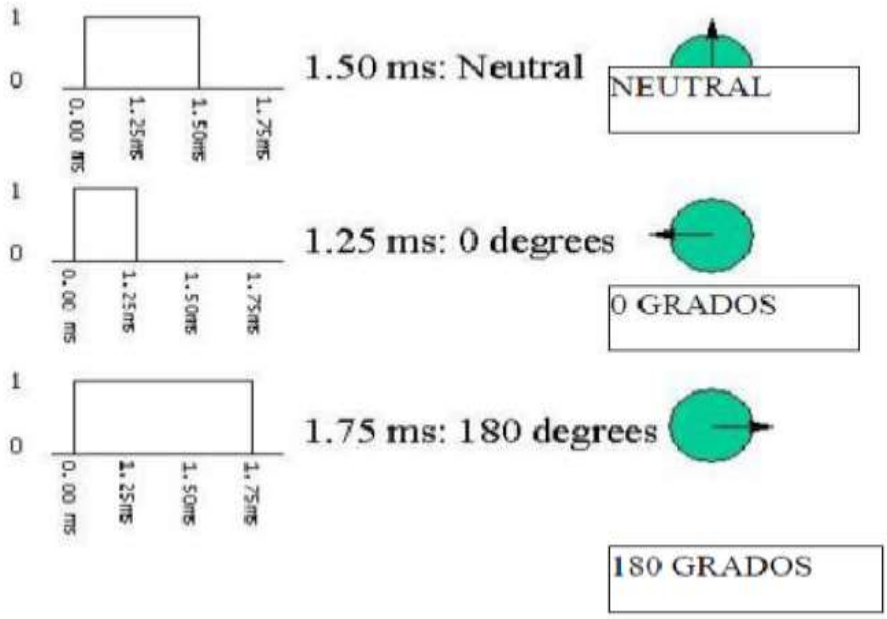


Figura 3.3: Movimiento de un servomotor

Fuente: http:// https://info-ab.uclm.es

Como se observa en la figura, la duración del pulso indica o dictamina el ángulo del eje y los tiempos reales dependen del fabricante del motor.

Para los HITEC: 0.50 ms = 0 grados

1.50 ms = 90 grados

2.5 ms = 180 grados

Como podemos observar para un servomotor estándar la duración del pulso alto para alcanzar un ángulo de posición θ estará dado por siguiente la fórmula:

$t = 0{,}3 + \theta/100$

Dónde: t está dada en milisegundo y θ en grados.

Duración del nivel alto [ms]	Ángulo [grados]
0,3	0
1,2	90
2,1	180
0,75	45

Tabla 3.1: Ejemplo de algunos valores usados en un servomotor

Fuente: Los Autores

Al enviarle continuamente una señal en la posición establecida, el servomotor se bloqueara, de esta forma el conservara su posición y se resistirá a fuerzas externas que intenten cambiarlo de ubicación. Si los pulsos no se envían, el servomotor queda liberado, y cualquier fuerza externa puede cambiarlo de posición fácilmente.

3.1.3 Control

Frecuentemente los pulsos en el tiempo off pueden interferir con el sincronismo interno del servo pero esto no significa que sea critico puede estar alrededor de los 20 ms y podría escucharse un sonido de zumbido o alguna vibración en el eje. Si el espacio de pulso es mayor que 50 ms (depende el fabricante), entonces el servo podría estar en modo SLEEP entre los pulsos. Entraría a funcionar en pasos pequeños y el rendimiento no sería el óptimo.

3.1.4 Modificaciones a los servos

Unas de las ventajas de los servomotores es la capacidad que tiene en convertirse en un motor de corriente continua normal, eliminada su restricción de giro y permitirle dar giros completos, pues es necesario eliminar el circuito de control. Debido que los

engranajes reductores se conservan luego de la modificación, el motor obtenido mantiene la fuerza y velocidad que tenían servo inicial, la ventaja de modificarlos es que tiene menos inercia que los motores de corriente continua comerciales, los que los hace conveniente para ciertas aplicaciones

3.2 SENSOR

Un sensor es un dispositivo eléctrico y/o mecánico capacitado para detectar acciones o estímulos (magnitudes físicas o químicas) en valores medibles para las magnitudes eléctricas. Las fases de transformación se las realiza en tres pasos:

- Al ser medido un fenómeno físico este es captado por el sensor en el cual muestra en su salida una señal eléctrica dependiente del valor de la variable física.
- La señal eléctrica es cambiado por un sistema de acondicionamiento de señal, cuya salida es un voltaje.
- El sensor tiene una circuitería que transforma y/o amplifica el voltaje de salida, la cual pasa a un conversor A/D conectado a un PC. El convertidor A/D transforma la señal de voltaje continua en una señal discreta.

3.2.1 Descriptores estáticos de un sensor

Los descriptores estáticos detallan el comportamiento permanente del sensor:

- **Rango**: valores máximos y mínimos para las variables de entrada y salida de un sensor.
- **Exactitud**: Es la desviación de la lectura de un sistema de medida respecto a una entrada conocida. El mayor error esperado entre las señales ideal versus la señal medida.
- **Repetitividad**: La lectura tiene que ser medida con una precisión dada.
- **Reproducibilidad**: Tiene el mismo sentido que la repetitividad excepto que se utiliza cuando se toman medidas distintas bajo condiciones diferentes.
- **Resolución**: Es la cantidad de medida más corto que se pueda detectar.

- **Error**: Es la diferencia entre el valor medido y el valor real.
- **No linealidades**: La desviación de la medida de su valor real, supuesto que la respuesta del sensor es lineal. No-linealidades típicas: saturación, zona muerta e histéresis.
- **Sensibilidad**: Es la razón de cambio de la salida frente a cambios en la entrada: $s = \partial V / \partial x$
- **Excitación**: Es la cantidad de corriente o voltaje requerida para el funcionamiento del sensor.
- **Estabilidad:** Es la capacidad de que la medida del sensor pueda mostrar la misma salida en un rango en que la entrada permanece constante.

3.2.2 Descriptores dinámicos de un sensor

Tiempo de retardo; td: Es el tiempo que tarda la salida del sensor en alcanzar el 50% de su valor final.

Tiempo de subida; tr: Es el tiempo que tarda la salida del sensor hasta alcanzar su valor final. => velocidad del sensor, es decir, lo rápido que responde ante una entrada.

Tiempo de pico; tp: El tiempo que tarda la salida den sensor en alcanzar el pico máximo de su sobre oscilación.

Pico de sobre oscilación; Mp: Expresa cuanto se eleva la evolución temporal de la salida del sensor respecto a su valor inicial.

Tiempo de establecimiento; ts: El tiempo que tarda la salida del sensor en entrar en la banda del 5% alrededor del valor final y ya no vuelve a salir de ella.

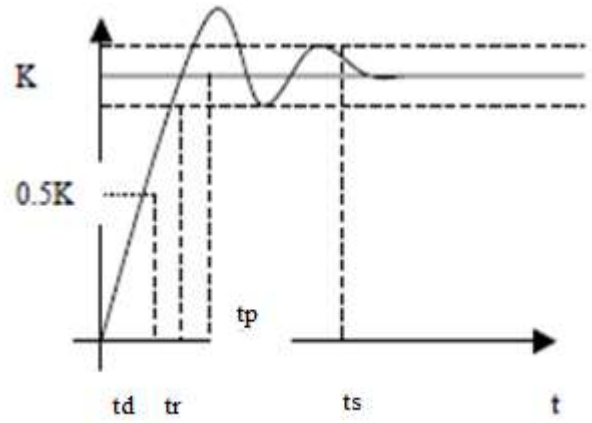


Figura 3.4: descriptores dinámicos de un sensor

Fuente: http://www.fim.umich.mx

3.2.3 Sensores de posición

Al estudiar los sensores de posición encontramos un rango y gamas, de los cuales pueden ser analógicos o digitales

- **Analógicos**: potenciómetros, resolvers, LVDT, rotativo SMD
- **Digitales**: encoders (absolutos e incrementales).

3.2.3.1 Potenciómetros

El potenciómetro angular es un transductor de tipo absoluto y con salida analógico, está diseñado básicamente con una resistencia de hilo bobinado y compartido a lo largo de un soporte en forma de arco. El movimiento del eje arrastra el cursor induciendo cambios de resistencia entre él y los extremos. Al alimentar los extremos con una tensión constante vo aparece en la toma de medida una tensión proporcional al ángulo girado a partir del origen. Dinámicamente el potenciómetro es un elemento

Proporcional sin retardo, pero la frecuencia de funcionamiento suele quedar limitada a 5 Hz por motivos mecánicos.

Figura 3.5: Potenciómetro

Fuente: http://img.directindustry.es

Ventajas: Facilidad de uso y bajo precio.

Desventajas: Para poder medir el desplazamiento deben estar fijados al dispositivo con una precisión limitada.

3.2.3.2 *Resolvers* (captadores angulares de posición)

Los *Resolvers* tienen semejanza a pequeños motores pero básicamente son transformadores rotativos diseñados de tal forma que su coeficiente de acoplamiento entre el rotor y el estator varía según sea la posición angular del eje. Constan de una bobina solidaria al eje excitada por una portadora y dos bobinas fijas situadas a su alrededor. La bobina móvil excitada con tensión gira un ángulo θ induce en las bobinas fijas las tensiones.

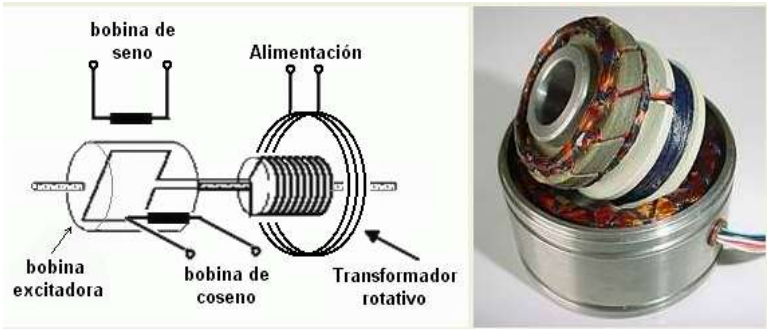


Figura 3.6: *Resolvers*

Fuente: http://robots-argentina.com.ar

3.2.3.3 Sensores lineales de posición (LVDT)

Es un dispositivo muy parecido a un transformador en el cual sensa la posición que provee un voltaje de salida proporcional al desplazamiento de su núcleo que pasa a través de sus arrollamientos, consta de un bobinado primario, secundario y un núcleo magnético, se aplica al bobinado primario una corriente alterna conocida como señal de portadora en el cual dicha corriente produce un campo magnético variable a alrededor del núcleo, este campo magnético induce un voltaje alterno en el bobinado secundario que está cerca del núcleo. Los sensores de posición lineal LVDT miden movimientos de pocas millonésimas partes de pulgada hasta varias pulgadas. Dinámicamente el sensor LVDT está auto limitada por los efectos inerciales de la masa del núcleo. Una de las ventajas de este dispositivo es que la salida es incremental, en caso de pérdida de alimentación los datos de la posición no se perderían. Cuando el sistema de medición es reiniciado, el valor LVDT de salida será el mismo que tuvo antes del fallo de alimentación.

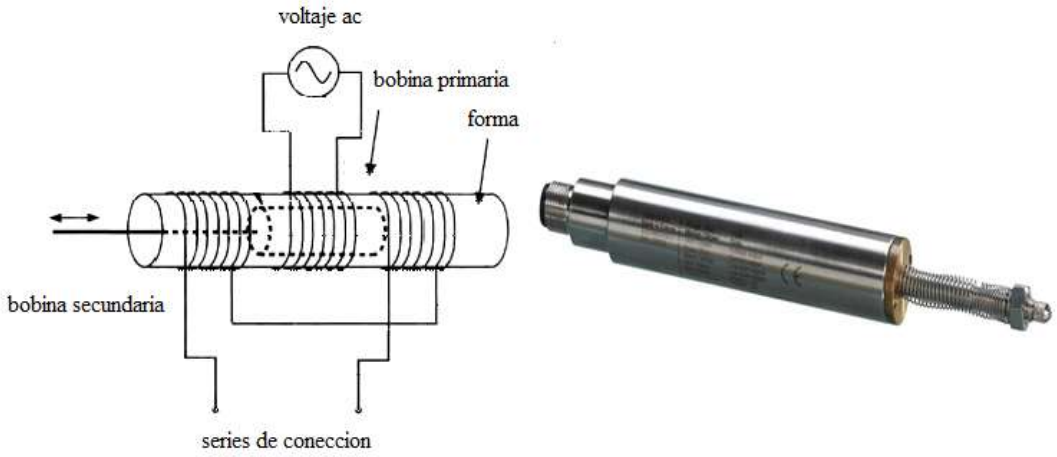


Figura 3.7: Sensor lineal de posición

Fuente: http://www.sensores-de-medida.es

3.2.3.4 *Encoders* (codificadores angulares de posición)

Es un codificador rotatorio o generador de pulso, normalmente suelen ser electromecánico, su función principal es convertir la posición angular de un eje a un código digital, sus partes internas constan de un disco transparente con una serie de marcas opacas colocadas radialmente y equidistantes entre sí, de un sistema de iluminación y de un elemento foto receptor.

El eje que se quiere medir va acoplado al disco, a medida que el eje gira se van generando pulsos en el receptor cada vez que la luz atraviese las marcas, llevando una cuenta de estos pulsos es posible conocer la posición del eje.

La resolución depende del número de marcas que se pueden poner físicamente en el disco. Hay dos tipos de *encoders* principales: absoluto e incremental.
El funcionamiento de un *encoder* absoluto es que el disco se divide en un número de sectores, codificándose cada uno de ellos con un código binario (código Gray).
La resolución es fija y viene dada por el número de anillos que posea el disco granulado. El *encoder* incremental matemáticamente proporciona formas de ondas cuadradas y desfasadas entre sí en 90^0, su precisión depende de factores mecánicos y

eléctricos entre los cuales el error se manifiesta en distintas causas la excentricidad del disco, falla electrónica de lectura, pueden ser imprecisiones de tipo óptico.

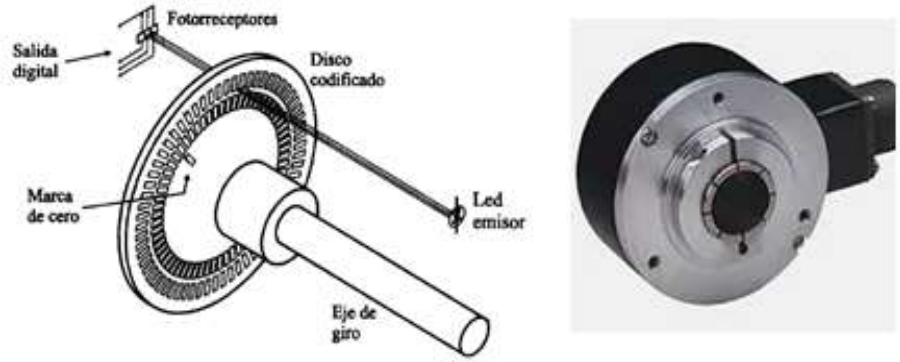


Figura 3.8: Encoders

Fuente: http://ab.rockwellautomation.com

3.2.3.5 Sensor de posición rotativo SMD (MTS360)

Es un sensor sin contacto (MTS360) que proporciona un verdadero avance al combinar la medición de posición absoluta de 360 grados de giro a través de un tamaño altamente reducido. Con su tamaño pequeño de solo 6mm x 17mm x 18mm, los ingenieros pueden ahora integrar un sensor rotativo de altas especificaciones directamente en el PCB sin encontrar problemas de empaquetado que normalmente afectan a encoders y otros sensores de posición absolutos. El excepcional bajo perfil permite al MTS encajar fácilmente en lugares previamente demasiado pequeños para otros dispositivos de posición.

El MTS360 se basa con tecnología patentada de efecto Hall, la cual proporciona por primera vez la medición de ángulo/posición sin contacto y con eje pasante, utilizando características SMD estándar.

Esta nueva tecnología ofrece medición de ángulos de hasta 360º sin "banda muerta" y con linealidades a partir de ±0.5%. Está preparado para trabajar a temperaturas de -

40ºC hasta +150ºC y puede operar a velocidades de hasta 300 rpm. La señal de salida es posible seleccionarla entre analógica, PWM de 12 bits ó SPI de 14 bits e incluye una segunda salida asignada a un interruptor programable.

El sensor MTS360 es un modelo óptimo para la estabilización de imágenes y dispositivos biomédicos de precisión utilizados en: Sistemas de climatización, control de equipos marino, feedback motor de velocidad, sistemas robóticos y de automatización.

Figura 3.9: Sensor de posición rotativo SMD (MTS360)

Fuente: http:// directindustry.es

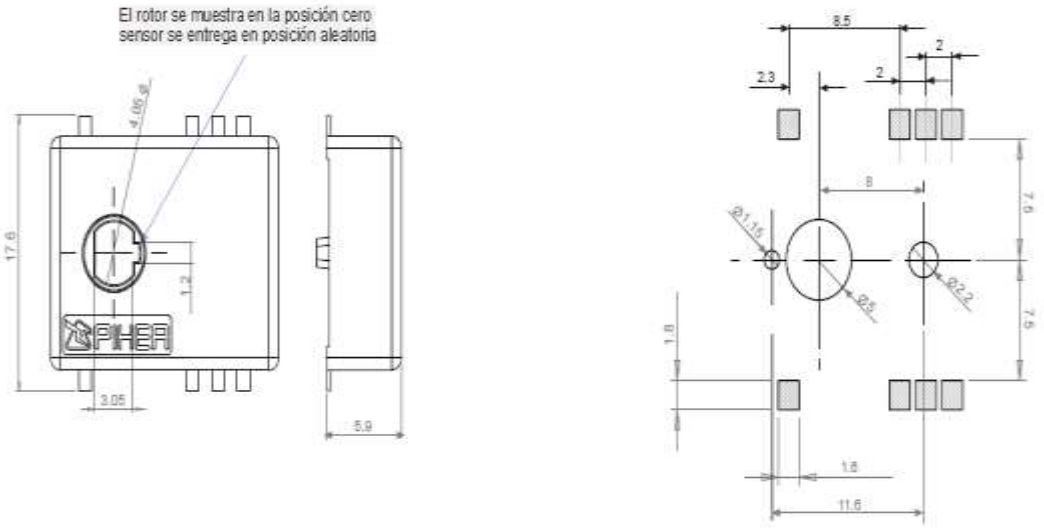


Figura 3.10: Dimensiones del sensor MTS360

Fuente: http:// directindustry.es

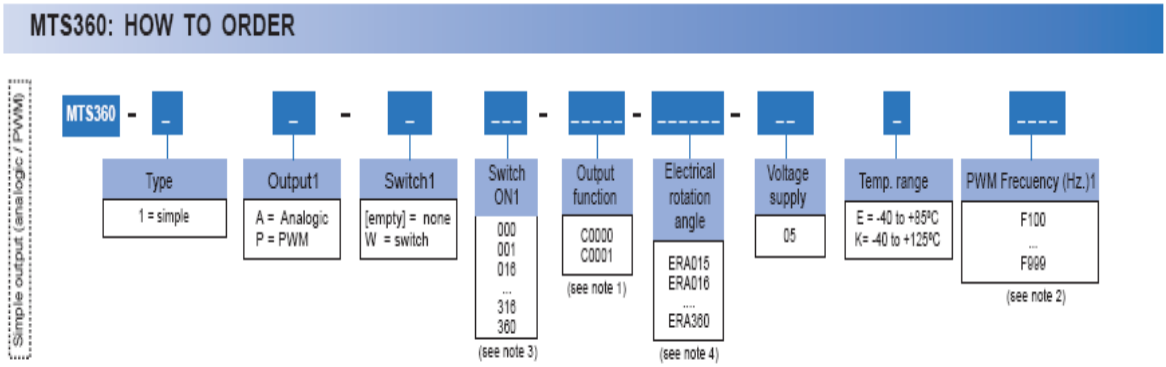


Figura 3.11: Características del sensor MTS360

Fuente: http://www.piher.net

Bajo estas especificaciones del sensor, se escogió con las siguientes características:

MTS-360-1A-W-360-0000-E

3.2.3.5.1 Configuración de pines SMD (MTS360)

	Output Types	Terminal 1	Terminal 2	Terminal 3	Terminal 4	Terminal 5	Terminal 6	Terminal 7	Terminal 8
SIMPLE	A-	Ground	Ground	Signal Output	Signal Output	Ground	Ground	Supply Voltage	Supply Voltage
	A-W	Ground	Ground	Signal Output	Signal Output	Switch Output	Switch Output	Supply Voltage	Supply Voltage
	P-	Ground	Ground	Signal Output	Signal Output	Ground	Ground	Supply Voltage	Supply Voltage
	P-W	Ground	Ground	Signal Output	Signal Output	Switch Output	Switch Output	Supply Voltage	Supply Voltage
	S-								
	AA-								

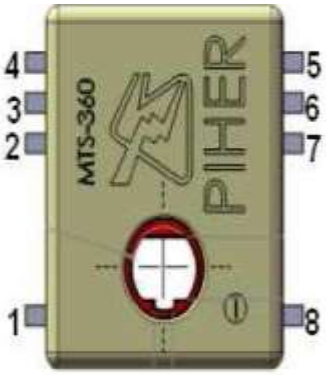


Figura 3.12: Configuración de pines sensor MTS360

Fuente: http://www.piher.net

3.3 DISEÑO Y CONSTRUCCÍON DE LA ESTRUCTURA MECÁNICA

A continuación detallamos cada paso de la construcción de nuestro diseño mecánico.

En nuestro diseño, se diseñó una mesa capaz de soportar y fijar cada uno de los elementos de la implementación con las medidas correspondientes:

Base = 1.50 mts
Altura= 1.35 mts
Ancho= 1 m

Figura 3.13: Medidas de la Mesa de soporte

Fuente: Los Autores

La mesa tiene cuatro ruedas para su traslado.

Figura 3.14: Mesa de soporte

Fuente: Los Autores

La mesa tiene 2 brazos que sostienen el eje transversal.

El brazo 1 tiene una altura con respecto a la mesa de 45 cm y tiene un ruliman para el movimiento del eje.

Figura 3.15: Brazo 1

Fuente: Los Autores

El brazo 2 tiene las mismas características del brazo 1 con altura respecto a la mesa de 19 cm de altura.

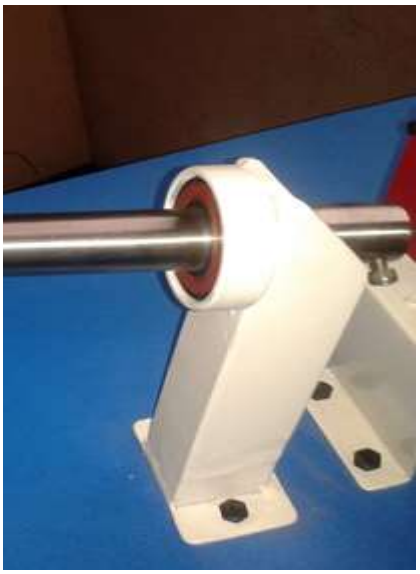


Figura 3.16: Brazo 2

Fuente: Los Autores

Estos 2 brazos sostienen al eje transversal con los que se encuentran perpendiculares al mismo.

Las características de El eje transversal son:

- Masa = 1,18 kg
- Diámetro = 2,4 cm
- Longitud = 1,6 m

Figura 3.17: Eje transversal

Fuente: Los Autores

Las medidas de los alerones son:

Alerón 1 = 19,5 cm

Figura 3.18: Alerón 1

Fuente: Los Autores

Alerón 2 = 36,5 cm

Figura 3.19: Alerón 2

Fuente: Los Autores

Adicional a los brazos, eje transversal y alerones incluimos dos micros-bases que sostienen al servomotor y a la tarjeta electrónica del sensor SMD.

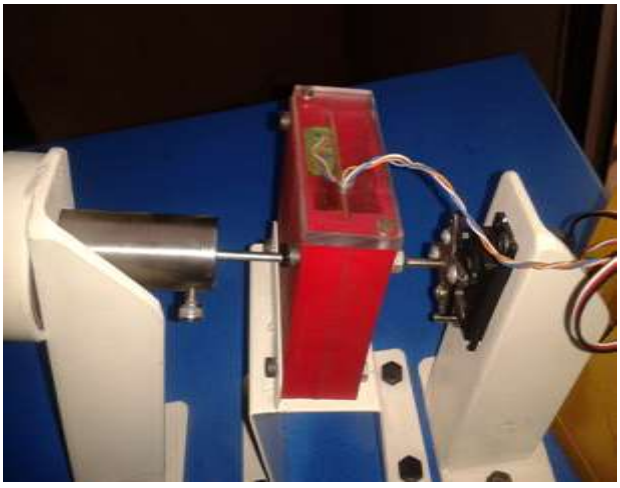


Figura 3.20: Base de Soporte para Tarjeta del Sensor

Fuente: Los Autores

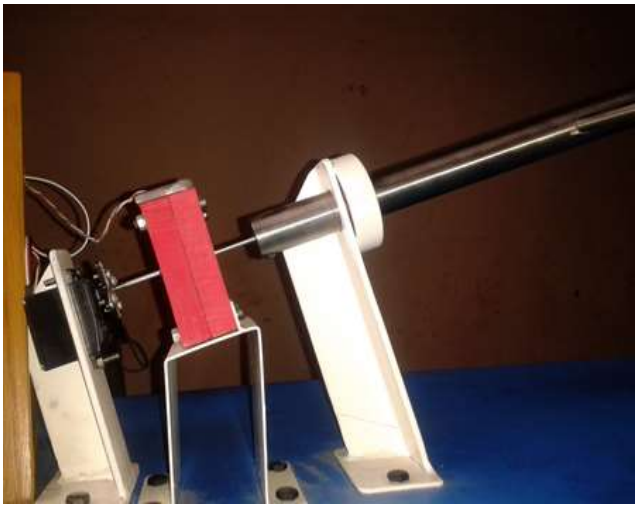


Figura 3.21: Parte frontal micro base

Fuente: Los Autores

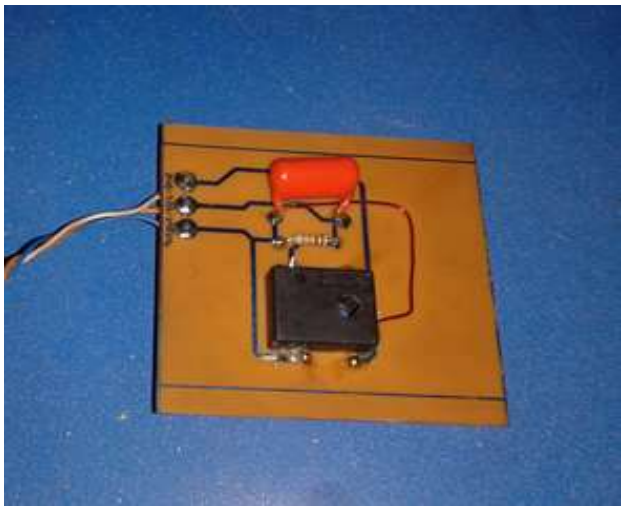


Figura 3.22: Tarjeta electrónica del sensor

Fuente: Los Autores

El servomotor en su estructura contiene brazos de rotación en el cual a éste se adicionó un micro-eje que atraviesa al sensor y al eje transversal para que a su vez se posesione en el ángulo ordenado.

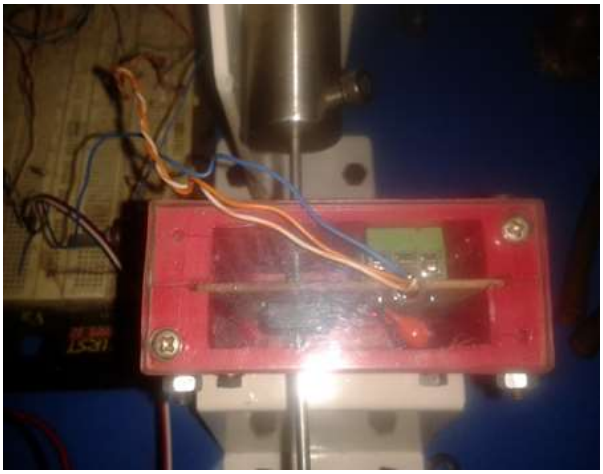


Figura 3.23: Acoplamiento Tarjeta del Sensor

Fuente: Los Autores

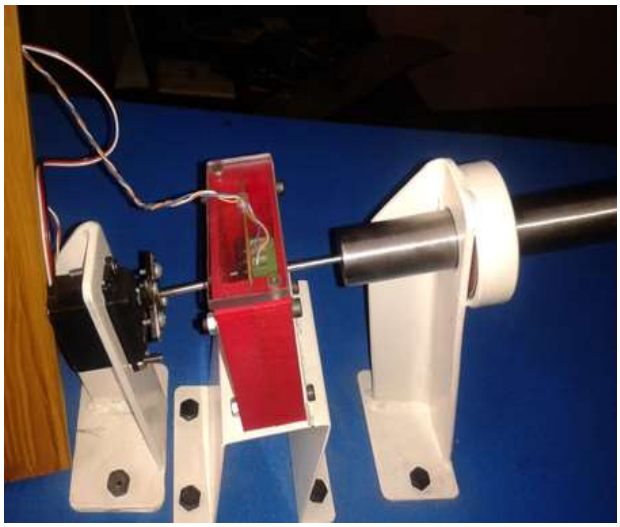


Figura 3.24: Servo, tarjeta del sensor y Eje transversal

Fuente: Los Autores

CAPÍTULO IV

DESARROLLO DEL SOFTWARE DE LA IMPLEMENTACÍON ENTRE ARDUINO Y LABVIEW

4.1 LABVIEW

Labview es un software de programación gráfica para aplicaciones que impliquen adquisición, control, análisis y presentación de datos. Labview emplea la programación gráfica o lenguaje G para crear programas basados en diagramas de bloques. Labview es compatible con herramientas e instrumentos similares a los sistemas de desarrollo comerciales que utilizan el lenguaje C o BASIC y puede trabajar con programas, aplicaciones de otras áreas como por ejemplo la plataforma Arduino. Labview tiene extensas librerías de funciones y subrutinas. Además de las funciones básicas de todo lenguaje de programación, Labview incluye librerías específicas para la adquisición de datos, control de instrumentación VXI, GPIB y comunicación serie, análisis presentación y guardado de datos.

Las ventajas al trabajar con Labview son las siguientes:

- El sistema tiene un compilador gráfico que se ejecuta a la máxima velocidad posible.
- Incorpora aplicaciones y diseños escritos en otros lenguajes.
- El software tiene un sistema donde se integran las funciones de adquisición, análisis y presentación de datos.
- Su utilización ayuda a los usuarios a crear soluciones completas y complejas en sus proyectos.
- Las aplicaciones se reduce al menos de 4 a 10 veces su desarrollo ya que es muy intuitivo y fácil de aprender.
- Dota de gran flexibilidad al sistema, permitiendo cambios y actualizaciones tanto del hardware como del software.

4.1.1 Programación gráfica de Labview

Básicamente al diseñar el programa con Labview se crea algo denominado VI (instrumento virtual), este VI puede utilizarse en cualquier otra aplicación como una sub función dentro de un programa general. Los VI's se caracterizan por ser un cuadrado con su respectivo símbolo relacionado con su funcionalidad, tener una interfaz con el usuario, tener entradas con su color de identificación de dato, tener una o varias salidas y por supuesto ser reutilizables. Debido al lenguaje gráfico, el compilador Labview es más cambiable, ya que sobre el mismo código se puede ver fácilmente el flujo de datos así como su contenido.

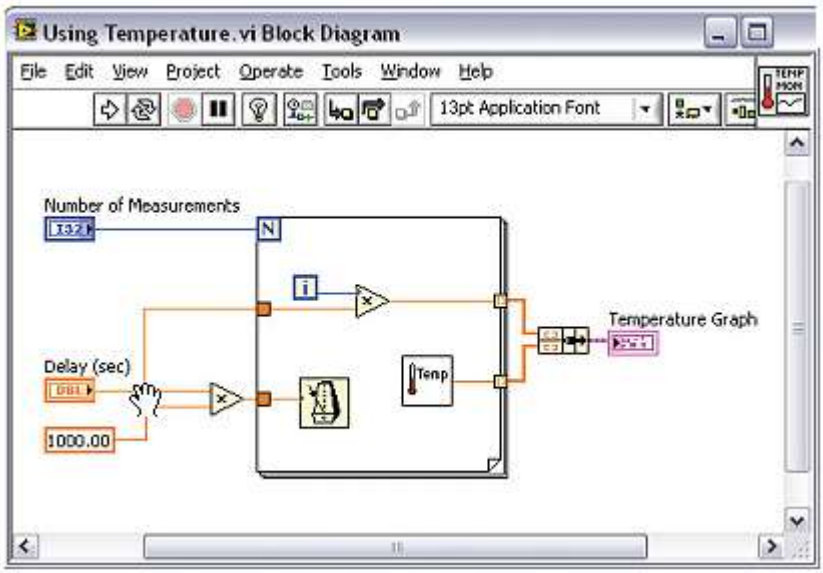


Figura4.1: Programación grafica de Labview

Fuente: Los Autores

4.1.2 Entorno Labview

La programación grafica de Labview consta de un panel frontal y un panel de código como se mencionó antes, en este panel frontal es donde se diseña la interface de usuario y se ubican los indicadores y controladores. En el panel de código se encuentran las funciones. Cada control que se utiliza en la interfaz tiene un representación en el panel de código, igualmente los indicadores necesarios para entregar la información procesada al usuario tienen un código que los identifica en el panel de código o programación. Los controles pueden ser booleanos, numérico,

strings, un arreglo matricial y los indicadores pueden ser como tablas, graficas en 2D o 3D, browser todo pudiéndolos visualizar.

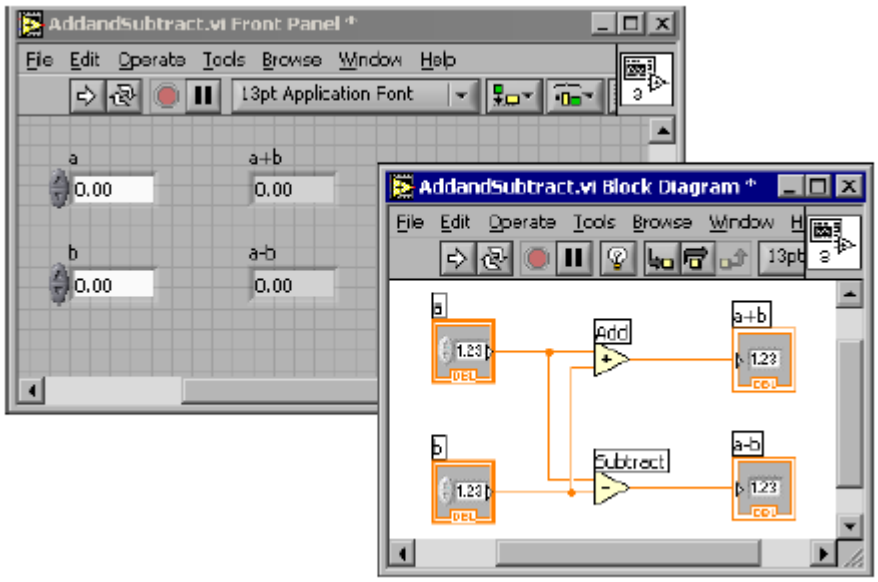


Figura 4.2: Programación gráfico de Labview

Fuente: Los Autores

4.2 SOFTWARE ARDUINO

La plataforma Arduino se programa mediante el uso de un lenguaje propio basado en la popular programación de alto nivel processing. El entorno es un código abierto, libre en la cual hace fácil escribirlo y cargarlo a la placa E/S, funciona con los sistemas operativos de Windows, Mac OS X y Linux. Sin embargo es posible utilizar otros lenguajes de programación y aplicaciones populares en Arduino. Arduino está basado en C y soporta todas las funciones del estándar C y algunas de C++.

Algunos ejemplos son:

- Java
- Flash (mediante ActionScript)
- Processing

- Pure Data
- MaxMSP (entorno gráfico de programación para aplicaciones musicales, de audio y multimedia)
- Adobe Director
- Python
- Ruby
- C
- C++ (mediante libSerial o en Windows)
- C#
- Cocoa/Objective-C (para Mac OS X)
- Linux TTY (terminales de Linux)
- 3DVIAVirtools (aplicaciones interactivas y de tiempo real)
- SuperCollider (síntesis de audio en tiempo real)
- InstantReality (X3D)
- Liberlab (software de medición y experimentación)
- BlitzMax (con acceso restringido)
- Squeak (implementación libre de Smalltalk)
- Mathematica
- Matlab
- Minibloq (entorno gráfico de programación, corre también en OLPC)
- Isadora (interactividad audiovisual en tiempo real)
- Perl
- PhysicalEtoys (entorno gráfico de programación usado para proyectos de robótica educativa)
- Scratchfor Arduino (S4A) (entorno gráfico de programación, modificación del entorno para niños Scratch, del MIT)
- Visual Basic .NET
- VBScript
- Gambas

Figura 4.3: Logo software Arduino

Fuente: http://arduino.cc

4.2.1 Entorno Arduino

4.2.1.1 Barra de herramientas

Figura 4.4: Barra de herramientas del entorno Arduino

Fuente: http://arduino.cc

Verify/Compile

Chequea el código en busca de errores.

Stop

Para el _Serial monitor_, o minimiza otros botones.

New

Crea una nueva rutina.

Open

Muestra un menú con todas las rutinas de tu sketchbook.

Save

Guarda las rutinas.

Upload to I/O board

Carga tu código a la placa Arduino I/O.

Serial Monitor

Muestra datos serie enviados desde la placa Arduino (placa serie o USB).

Tab Menu

Permite gestionar las rutinas con más de un archivo (cada uno de los cuales aparece en su propia pestaña). Estos pueden ser:
Archivos de código de Arduino (sin extensión).
Archivos de C (extensión .c).
Archivos de C++ (extensión .cpp).
Archivos de cabecera (extensión .h)

4.2.2 Manual de programación – Arduino

La estructura básica del lenguaje de programación de Arduino se compone de al menos dos partes y son las siguientes:

En donde setup **()** es la parte encargada de recoger la configuración y loop **()** es la que contienen el programa que se ejecutará cíclicamente (de ahí el termino loop –bucle-). Ambas funciones son necesarias para que el programa trabaje.

La función de configuración (**s**etup **()**) debe contener la declaración de las variables. Es la primera función a ejecutar en el programa, se ejecuta sólo una vez y se utiliza para configurar o inicializar pinMode (modo de trabajo de las E/S), configuración de la comunicación enserie y otras.

La función bucle (**l**oop **()**), contiene el código que se ejecutara continuamente (lectura de entradas, activación de salidas, etc.). Esta función es el núcleo de todos los programas de Arduino y la que realiza la mayor parte del trabajo.

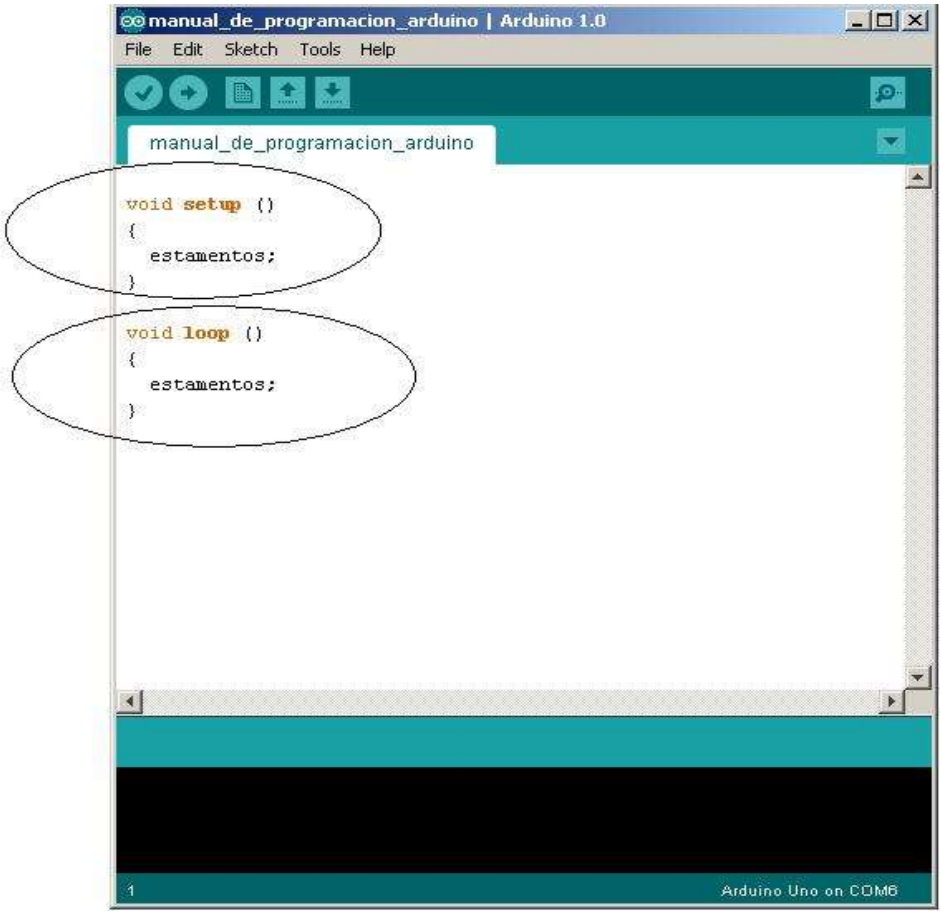


Figura 4.5: Estructura de programación (void setup, void loop)

Fuente: Los Autores

4.2.2.1 Setup ( )

La función setup () se invoca una sola vez cuando el programa empieza. Se utiliza para inicializar los modos de trabajo de los pines, o el puerto serie. Debe ser incluido en un programa aunque no haya declaración que ejecutar.

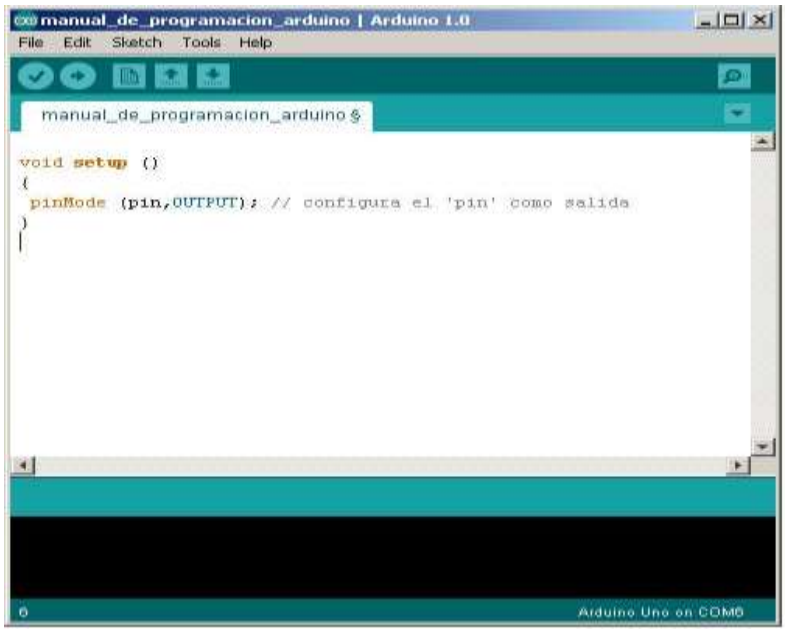


Figura 4.6: Inicializando con la función void setup

Fuente: Los Autores

4.2.2.2 Loop ( )

La función loop () hace precisamente lo que sugiere su nombre, se ejecuta de forma cíclica, lo que posibilita que el programa este respondiendo continuamente ante los eventos que se produzcan en la tarjeta.

Figura 4.7: Llamando a la función void loop

Fuente: Los Autores

4.2.2.3 Funciones

Una función es un bloque de código que tiene un nombre y un conjunto de líneas de programación que son ejecutados cuando se llama a la función. Son funciones setup () y loop () de las que ya se ha hablado. Las funciones de usuario pueden ser escritas para realizar tareas repetitivas y para reducir el tamaño de un programa. Las funciones se declaran asociadas a un tipo de valor "type". Este valor será el que devolverá la función, por ejemplo 'int' se utilizará cuando la función devuelva un dato numérico de tipo entero. Si la función no devuelve ningún valor entonces se colocará delante la palabra "void", que significa "función vacía". Después de declarar el tipo de dato que devuelve la función se debe escribir el nombre de la función y entre paréntesis se escribirán, si es necesario, los parámetros que se deben pasar a la función para que se ejecute.

La función siguiente devuelve un número entero, delayVal () se utiliza para poner un valor de retraso en un programa que lee una variable analógica de un potenciómetro conectado a una entrada de Arduino. Al principio se declara como una variable local, ´v´ recoge el valor leído del potenciómetro que estará comprendido entre 0 y 1023, luego se divide el valor por 4 para ajustarlo a un margen comprendido entre 0 y 255, finalmente se devuelve el valor ´v´ y se retornará al programa principal. Esta función cuando se ejecuta devuelve el valor de tipo entero ´v´.

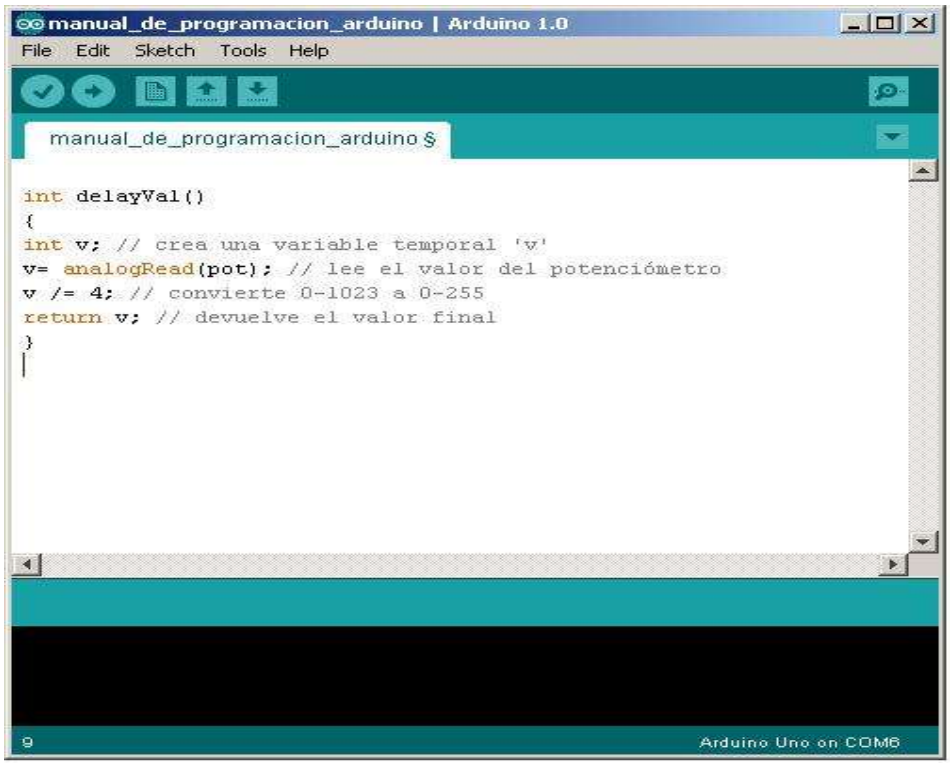


Figura 4.8: Función delayVal ()

Fuente: Los Autores

4.2.2.4 Declaración de variable

Todas las variables tienen que declararse antes de que puedan ser utilizadas. Para declarar una variable se comienza por definir su tipo como int (entero), long (largo), float (coma flotante), etc. asignándoles siempre un nombre, y, opcionalmente, un valor inicial. Esto sólo debe hacerse una vez en un programa, pero el valor se puede cambiar en cualquier momento usando aritmética y reasignaciones diversas. El siguiente ejemplo declara la variable entradaVariable como una variable de tipo entero "int", y asignándole un valor inicial igual a cero. Esto se llama una asignación.

```
int entradaVariable = 0;
```

4.2.2.5 Variables

Una variable es una manera de nombrar y almacenar un valor numérico para su uso posterior por el programa. Como su nombre indica, las variables son números que se pueden variar continuamente en contra de lo que ocurre con las constantes cuyo valor nunca cambia. Una variable debe ser declarada y, opcionalmente, asignarle un valor. El siguiente código de ejemplo declara una variable llamada variableEntrada y luego le asigna el valor obtenido en la entrada analógica del PIN2:

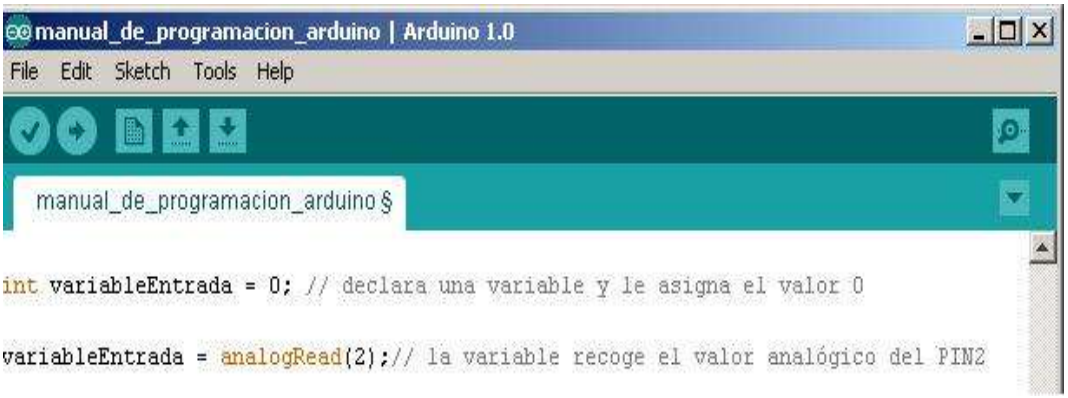


Figura 4.9: Declaración de variable

Fuente: Los Autores

VariableEntrada es la variable en sí. La primera línea, declara que será de tipo entero "int". La segunda línea fija a la variable el valor correspondiente a la entrada analógica PIN2. Esto hace que el valor de PIN2 sea accesible en otras partes del código. Una vez que una variable ha sido asignada, o re-asignada, se puede probar su valor para ver si cumple ciertas condiciones (instrucciones if...), o puede utilizar directamente su valor.

Como ejemplo ilustrativo ver tres operaciones útiles con variables: El siguiente código prueba si la variable "entradaVariable" es inferior a 100, si es cierto se asigna el valor 100 a "entradaVariable" y, a continuación, establece un retardo (delay) utilizando como valor "entradaVariable" que ahora será como mínimo de valor 100:

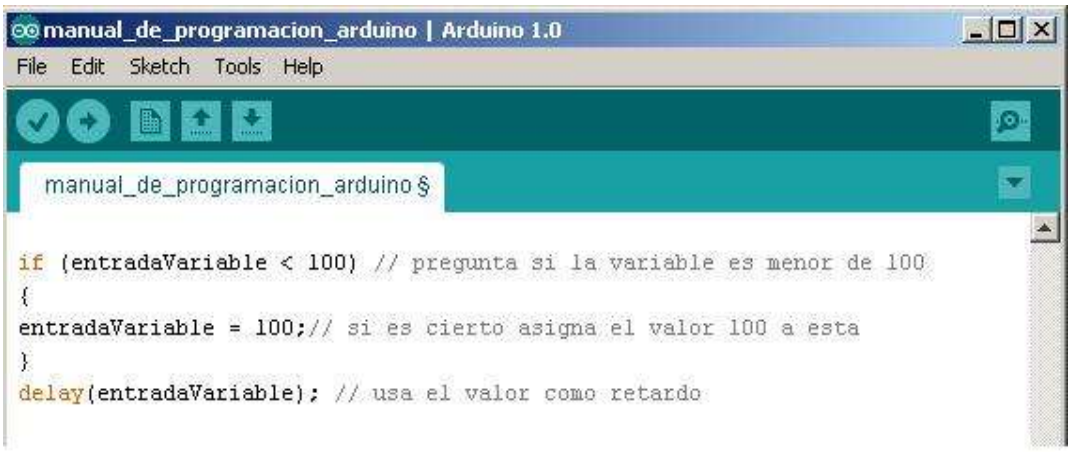


Figura 4.10: Declaración de variable

Fuente: Los Autores

Nota: Las variables deben tomar nombres descriptivos, para hacer el código más legible. Nombres de variables pueden ser "contactoSensor" o "pulsador", para ayudar al programador y a cualquier otra persona a leer el código y entender lo que representa la variable. Nombres de variables como "var" o "valor", facilitan muy poco que el código sea inteligible. Una variable puede ser cualquier nombre o palabra que no sea una palabra reservada en el entorno de Arduino.

4.2.2.6 Utilización de una variable

Una variable puede ser declarada al inicio del programa antes de la parte de configuración setup (), a nivel local dentro de las funciones, y, a veces, dentro de un bloque, como para los bucles del tipo if...for..., etc. En función del lugar de declaración de la variable así se determinara el ámbito de aplicación, o la capacidad de ciertas partes de un programa para hacer uso de ella.

Una variable global es aquella que puede ser vista y utilizada por cualquier función de un programa. Esta variable se declara al comienzo del programa, antes de setup ().

Una variable local es aquella que se define dentro de una función o como parte de un bucle. Sólo es visible y sólo puede utilizarse dentro de la función en la que se declaró.

Por lo tanto, es posible tener dos o más variables del mismo nombre en diferentes partes del mismo programa que pueden contener valores diferentes. La garantía de que sólo una función tiene acceso a sus variables dentro del programa simplifica y reduce el potencial de errores de programación.

El siguiente ejemplo (figura 3.11) muestra cómo declarar a unos tipos diferentes de variables y la visibilidad de cada variable:

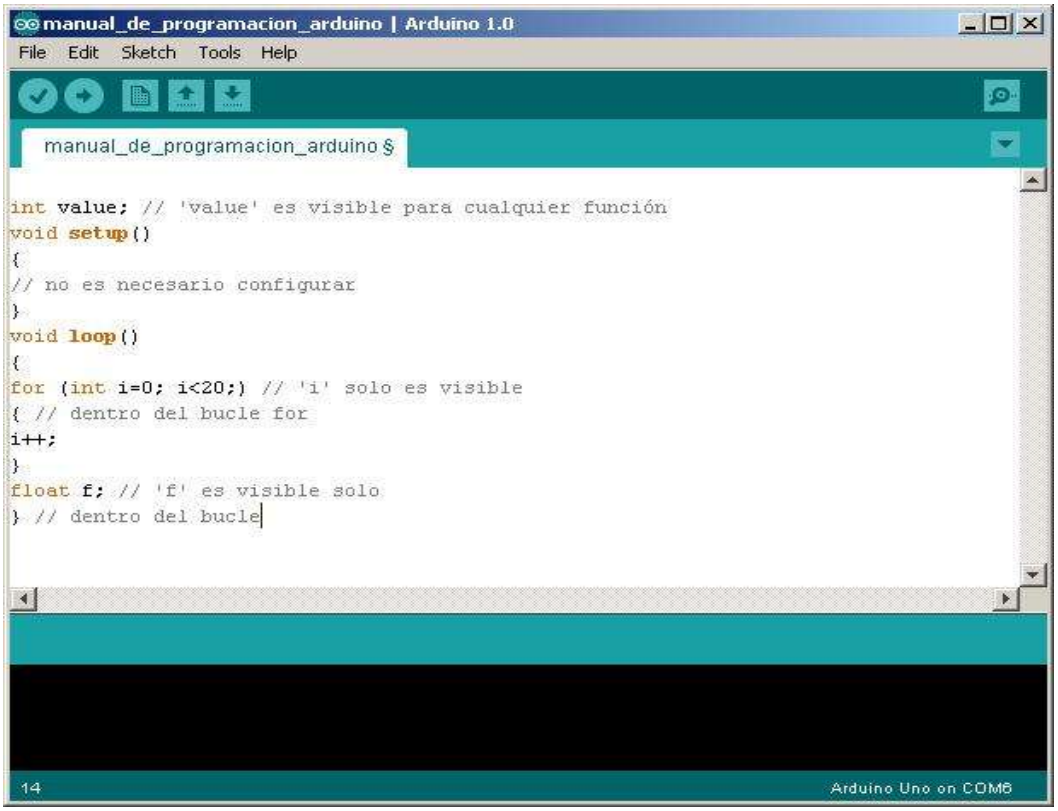


Figura 4.11: Declaración de diferentes tipos de variable

Fuente: Los Autores

4.2.2.7 Tipos de variables

4.2.2.7.1 Byte

Byte almacena un valor numérico de 8 bits sin decimales. Tienen un rango entre 0 y 255.

4.2.2.7.2 Int

Enteros son un tipo de datos primarios que almacenan valores numéricos de 16 bits sin decimales comprendidos en el rango 32,767 a -32,768.

4.2.2.7.3 Long

El formato de variable numérica de tipo extendido “long” se refiere a números enteros (tipo 32 bits) sin decimales que se encuentran dentro del rango -2147483648 a 2147483647.

4.2.2.7.4 Float

El formato de dato del tipo “punto flotante” “float” se aplica a los números con decimales. Los números de punto flotante tienen una mayor resolución que los de 32 bits con un rango comprendido 3.4028235E +38 a +38-3.4028235E.
Float una Variable = 3.14; // declara 'unaVariable' como tipo flotante.

Nota: Los números de punto flotante no son exactos, y pueden producir resultados extraños en las comparaciones. Los cálculos matemáticos de punto flotante son también mucho más lentos que los del tipo de números enteros, por lo que debe evitarse su uso si es posible

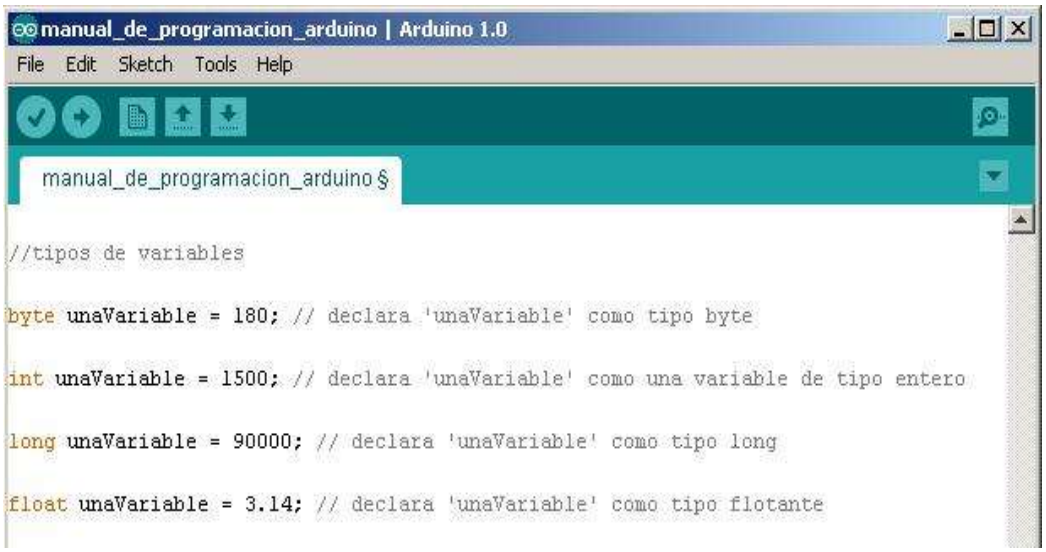


Figura 4.12: Declaración de la variable Float

Fuente: Los Autores

4.2.2.8 Arrays

Un array es un conjunto de valores a los que se accede con un número índice. Cualquier valor puede ser recogido haciendo uso del nombre de la matriz y el número del índice.

El primer valor de la matriz es el que está indicado con el índice 0, es decir el primer valor del conjunto es el de la posición 0. Un array tiene que ser declarado y opcionalmente asignados valores a cada posición antes de ser utilizado.

Del mismo modo es posible declarar una matriz indicando el tipo de datos y el tamaño y posteriormente, asignar valores a una posición especifica:

int miArray [5]; // declara un array de enteros de 6 posiciones
int miArray [3] = 10; // asigna el valor 10 a la posición 4

Para leer de un array basta con escribir el nombre y la posición a leer:
X = miArray [3]; // x ahora es igual a 10 que está en la posición 3 del array

Las matrices se utilizan a menudo para líneas de programación de tipo bucle, en los que la variable de incremento del contador del bucle se utiliza como índice o puntero del array. El siguiente ejemplo usa una matriz para el parpadeo de un led.

Utilizando un bucle tipo for, el contador comienza en cero 0 y escribe el valor que figura en la posición de índice 0 en la serie que hemos escrito dentro del arrayparpadeo [], en este caso 180, que se envía a la salida analógica tipo PWM configurada en el PIN10, se hace una pausa de 200 ms y a continuación se pasa al siguiente valor que asigna el índice "i".

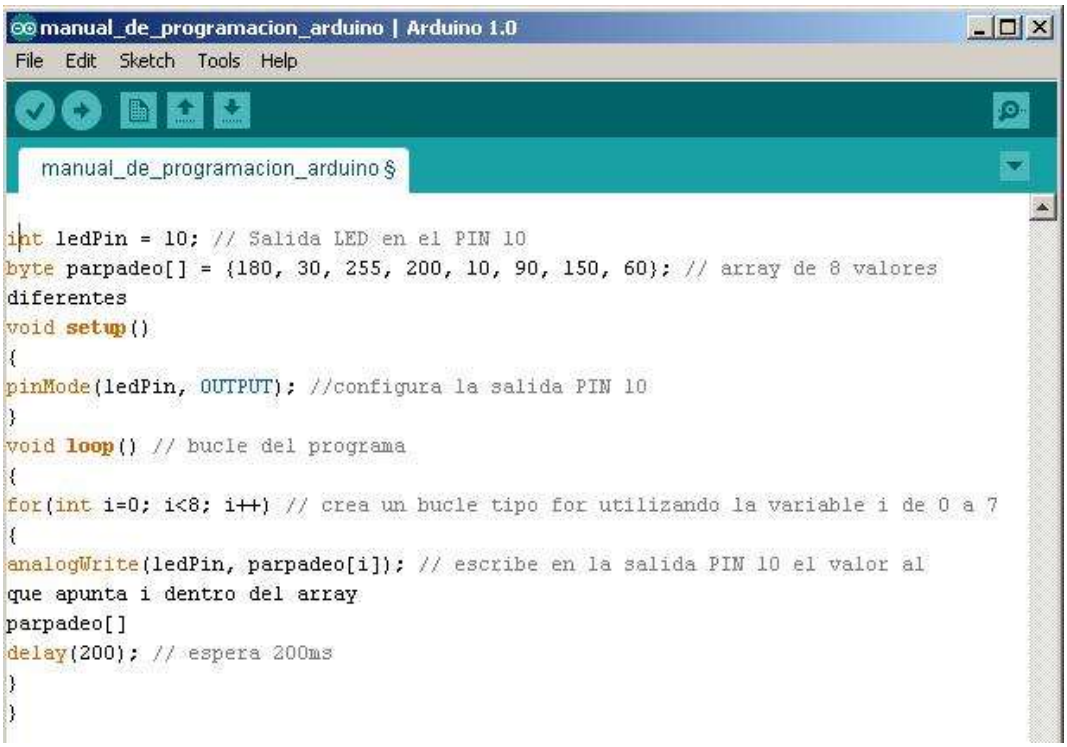


Figura 4.13: Declaración de un array

Fuente: Los Autores

4.2.2.9 Aritmética

Los operadores aritméticos que se incluyen en el entorno de programación son suma, resta, multiplicación y división. Estos devuelven la suma, diferencia, producto, o cociente (respectivamente) de dos operando

```
Y= Y + 3;
X= X – 7;
J= J * 6;
R= R / 5;
```

La operaciones se efectúa teniendo en cuenta el tipo de datos que está definido para los operandos (int, dbl, float, etc...), por lo que, por ejemplo, 9 y 4 como enteros "int", 9 / 4 devuelve de resultado 2 en lugar de 2,25 ya que el 9 y 4 se valores de tipo entero "int" (enteros) y no se reconocen los decimales con este tipo de datos.

Esto también significa que la operación puede sufrir un desbordamiento si el resultado es más grande que lo que puede ser almacenada en el tipo de datos.

Si los operandos son de diferentes tipos, para el cálculo se utilizará el tipo más grande de los operandos en juego. Por ejemplo, si uno de los números (operandos) es del tipo float y otra de tipo integer, para el cálculo se utilizará el método de float es decir el método de coma flotante.

Elegir el tamaño de las variables de tal manera que sea lo suficientemente grande como para que los resultados sean lo precisos que usted desea. Para las operaciones que requieran decimales utilice variables tipo float, pero sea consciente de que las operaciones con este tipo de variables son más lentas a la hora de realizarse el cómputo.

Nota: Utilizar el operador (int) myFloat para convertir un tipo de variable a otro sobre la marcha. Por ejemplo, i = (int) 3,6 establecerá i igual a 3.

4.2.2.10 Asignaciones compuestas

Las asignaciones compuestas combinan una operación aritmética con una variable asignada. Estas son comúnmente utilizadas en los bucles tal como se describe más adelante. Estas asignaciones compuestas pueden ser:

```
X++   // igual que X = X + 1,       o incrementar X en + 1
X--   // igual que X = X - 1,       o incrementar X en - 1
X+= Y   // igual que X = X + Y,   o incrementar X en +Y
X *= Y // igual que X = X + 1,     o multiplicar X por Y
```

Nota: Por ejemplo, x * = 3 hace que x se convierta en el triple del antiguo valor x y por lo tanto x es reasignada al nuevo valor.

4.2.2.11 Operadores de comparación

Las comparaciones de una variable o constante con otra se utilizan con frecuencia en las estructuras condicionales del tipo if...para testear si una condición es verdadera. En los ejemplos que siguen en las próximas páginas se verá su utilización práctica usando los siguientes tipo de condicionales:

```
X = = Y   // X es igual a Y
X < Y     //  X es menor que Y
X > Y     //  X es mayor  que Y
```

4.2.2.12 Operadores lógicos

Los operadores lógicos son usualmente una forma de comparar dos expresiones y devolver un VERDADERO o FALSO dependiendo del operador. Existen tres operadores lógicos, AND (&&), OR (||) y NOT (!), que a menudo se utilizan en líneas de programación de tipo if...:

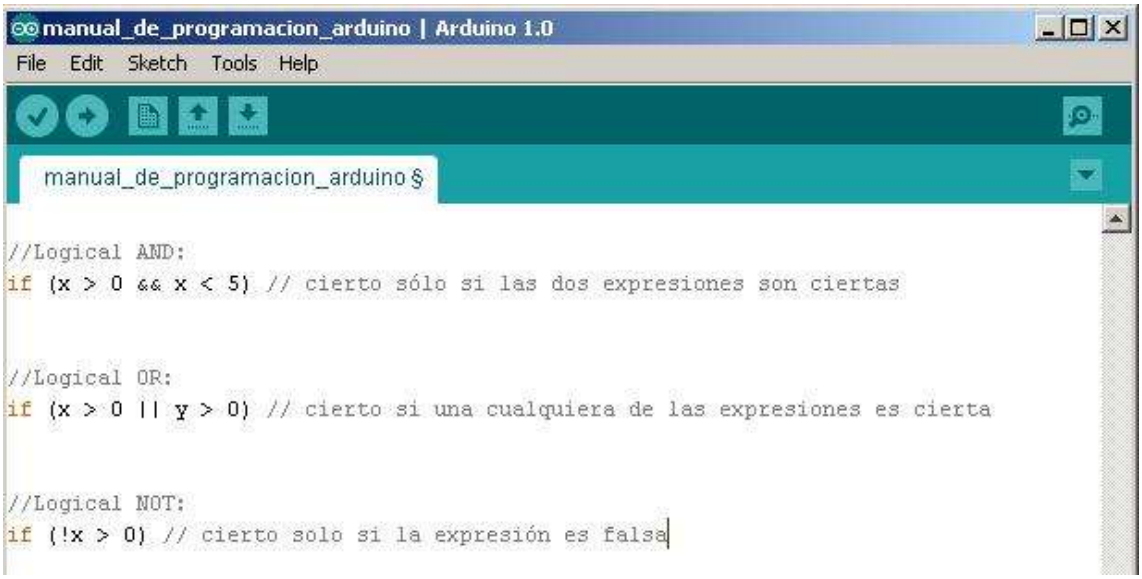


Figura 4.14: Operadores lógicos

Fuente: Los Autores

4.2.2.13 Constantes

El lenguaje de programación de Arduino tiene unos valores predeterminados, que son llamados constantes. Se utilizan para hacer los programas más fáciles de leer. Las constantes se clasifican en grupos.

4.2.2.14 Cierto/falso (true/false)

Estas son constantes booleanas que definen los niveles HIGH (alto) y LOW (bajo) cuando estos se refieren al estado de las salidas digitales. FALSE se asocia con 0 (cero), mientras que TRUE se asocia con 1, pero TRUE también puede ser cualquier

otra cosa excepto cero. Por lo tanto, en sentido booleano, -1, 2 y -200 son todos también se definen como TRUE.

4.2.2.15 Input / Output /If /If else….

Estas constantes son utilizadas para definir, al comienzo del programa, el modo de funcionamiento de los pines mediante la instrucción pinMode de tal manera que el pin puede ser una entrada INPUT o una salida OUTPUT.

PinMode (13, OUTPUT); // designamos que el PIN 13 es una salida

If es un estamento que se utiliza para probar si una determinada condición se ha alcanzado, como por ejemplo averiguar si un valor analógico está por encima de un cierto número, y ejecutar una serie de declaraciones (operaciones) que se escriben dentro de llaves, si es verdad. Si es falso (la condición no se cumple) el programa salta y no ejecuta las operaciones que están dentro de las llaves, El formato para if es el siguiente:

En el ejemplo anterior se compara una variable con un valor, el cual puede ser una variable o constante. Si la comparación, o la condición entre paréntesis se cumple (es cierta), las declaraciones dentro de los corchetes se ejecutan. Si no es así, el programa salta sobre ellas y sigue.

Nota: Tener en cuenta el uso especial del símbolo '=', poner dentro de if (x = 10), podría parecer que es válido pero sin embargo no lo es ya que esa expresión asigna el valor 10 a la variable x, por eso dentro de la estructura if se utilizaría X==10 que en este caso lo que hace el programa es comprobar si el valor de x es 10... Ambas cosas son distintas por lo tanto dentro de las estructuras if, cuando se pregunte por un valor se debe poner el signo doble de igual “= =”

If… else viene a ser un estructura que se ejecuta en respuesta a la idea "si esto no se cumple haz esto otro". Por ejemplo, si se desea probar una entrada digital, y hacer una prueba si la entrada fuese alta o hacer otra prueba si la entrada fue baja, se escribiría que de esta manera:

```
if (inputPin = = HIGH) // si el valor de la entrada inputPin es alto
{
   InstruccionesA; // ejecuta si se cumple la condición
}
else
{
    InstruccionesB; // ejecuta si no se cumple la condición
 }
```

Else puede ir precedido de otra condición de manera que se pueden establecer varias estructuras condicionales de tipo unas dentro de las otras (anidamiento) de forma que sean mutuamente excluyentes pudiéndose ejecutar a la vez. Es incluso posible tener un número ilimitado de estos condicionales. Recordar sin embargo qué sólo un conjunto de declaraciones se llevará a cabo dependiendo de la condición probada:

```
if (inputPin < 500)
{
   InstruccionesA; // ejecuta las operaciones A
}
else if (inputPin >= 1000)
{
    InstruccionesB; // ejecuta las operaciones B
}
else
{
    InstruccionesC; // ejecuta las operaciones C
 }
```

Nota: Un estamento de tipo if prueba simplemente si la condición dentro del paréntesis es verdadera o falsa. Esta declaración puede ser cualquier declaración válida. En el anterior ejemplo, si cambiamos y ponemos (inputPin == HIGH). En este caso, el estamento if sólo chequearía si la entrada especificado esta en nivel alto (HIGH), o +5 v.

4.2.2.16 For

La declaración For se usa para repetir un bloque de sentencias encerradas entre llaves un número determinado de veces. Cada vez que se ejecutan las instrucciones del bucle se vuelve a testear la condición. La declaración For tiene tres partes separadas por (;) vemos el ejemplo de su sintaxis:

```
For (inicialización; condición; expresión)
{
ejecutaInstruciones;
}
```

La inicialización de una variable local se produce una sola vez y la condición se testea cada vez que se termina la ejecución de las instrucciones dentro del bucle. Si la condición sigue cumpliéndose, las instrucciones del bucle se vuelven a ejecutar. Cuando la condición no se cumple, el bucle termina.

El siguiente ejemplo inicia el entero i en el 0, y la condición es probar que el valor es inferior a 20 y si es cierto i se incrementa en 1 y se vuelven a ejecutar las instrucción es que hay dentro de las llaves:

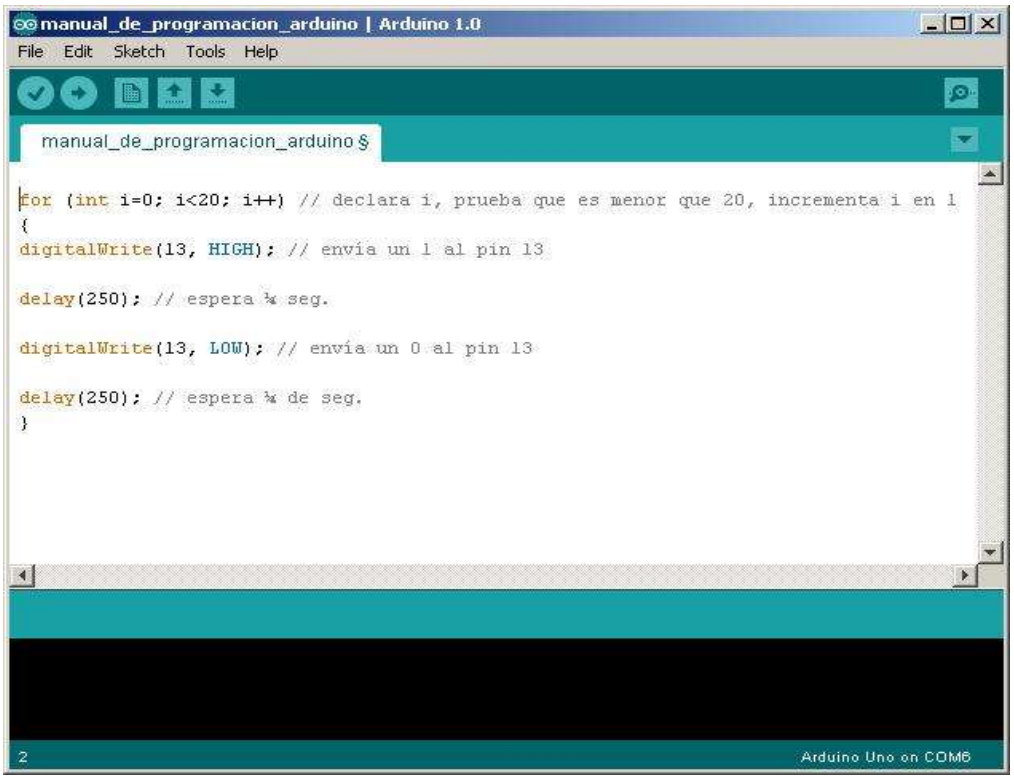


Figura 4.15: Declaración FOR

Fuente: Los Autores

Nota: El bucle en el lenguaje C es mucho más flexible que otros bucles encontrados en algunos otros lenguajes de programación, incluyendo BASIC. Cualquiera de los tres elementos de cabecera puede omitirse, aunque el punto y coma es obligatorio. También las declaraciones de inicialización, condición y expresión puede ser cualquier estamento válido en lenguaje C sin relación con las variables declaradas. Estos tipos de estados son raros pero permiten disponer soluciones a algunos problemas de programación raras.

4.2.2.17 While

Un bucle del tipo while es un bucle de ejecución continua mientras se cumpla la

Expresión colocada entre paréntesis en la cabecera del bucle. La variable de prueba tendrá que cambiar para salir del bucle. La situación podrá cambiar a expensas de una expresión dentro el código del bucle o también por el cambio de un valor en una entrada de un sensor.

```
While (unavariable < 200)   // testea si es menor que 200
{

    Instrucciones;     //ejecuta las instrucciones entre llaves
    Unavariable++;  // incrementa la variable en 1

}
```

El siguiente ejemplo asigna el valor leído lee Sensor () a la variable 'x', espera 50 Milisegundos y luego continúa mientras que el valor de la 'x' sea inferior a 100:

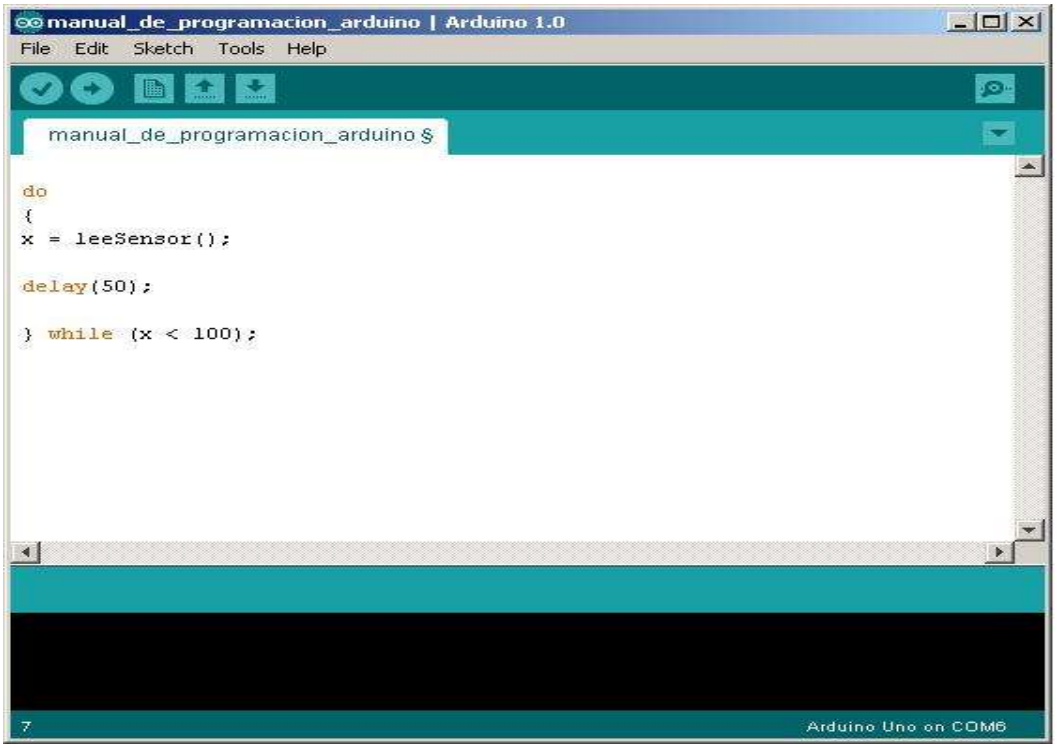


Figura 4.16: Bucle de tipo “while”

Fuente: Los Autores

4.2.2.18 Pinmode (pin, mode)

Esta instrucción es utilizada en la parte de configuración setup () y sirve para configurar el modo de trabajo de un PIN pudiendo ser INPUT (entrada) u OUTPUT (salida).

Pinmode (pin, OUTPUT); // configura pin como salida

Los terminales de Arduino, por defecto, están configurados como entradas, por lo tanto no es necesario definirlos en el caso de que vayan a trabajar como entradas. Los pines configurados como entrada quedan, bajo el punto de vista eléctrico, como entradas en estado de alta impedancia.

Estos pines tienen a nivel interno una resistencia de 20 KΩ a las que se puede acceder mediante software. Estas resistencias acceden de la siguiente manera:

Pinmode (pin, INPUT);
digitalwrite(pin, HIGH);

Las resistencias internas normalmente se utilizan para conectar las entradas interruptores. En el ejemplo anterior no se trata de convertir un pin en salida, es Simplemente un método para activar las resistencias interiores.

Los pines configurado como OUTPUT (salida) se dice que están en un estado de baja impedancia estado y pueden proporcionar 40 mA (miliamperios) de corriente a otros dispositivos y circuitos. Esta corriente es suficiente para alimentar un diodo LED (no olvidar poner una resistencia en serie), pero no es lo suficiente grande como para alimentar cargas de mayor consumo como relés, solenoides, o motores.

Un cortocircuito en las patillas Arduino provocará una corriente elevada que puede dañar o destruir el chip Atmega. A menudo es una buena idea conectar en la OUTPUT (salida) una resistencia externa de 470 o de 1000 Ω

4.2.2.19 Digitalread (pin)

Lee el valor de un pin (definido como digital) dando un resultado HIGH (alto) o LOW (bajo). El pin se puede especificar ya sea como una variable o una constante (0-13).

Valor = digitalread (Pin); // hace que valor sea igual al estado leído en pin.

4.2.2.20 Digitalwrite (pin, value)

Envía al ´pin´ definido previamente como OUTPUT el valor HIGH o LOW (poniendo en 1 o 0 la salida). El pin se puede especificar ya sea como una variable o como una constante (0-13).

El siguiente ejemplo lee el estado de un pulsador conectado a una entrada digital y lo escribe en el ´pin´de salida led:

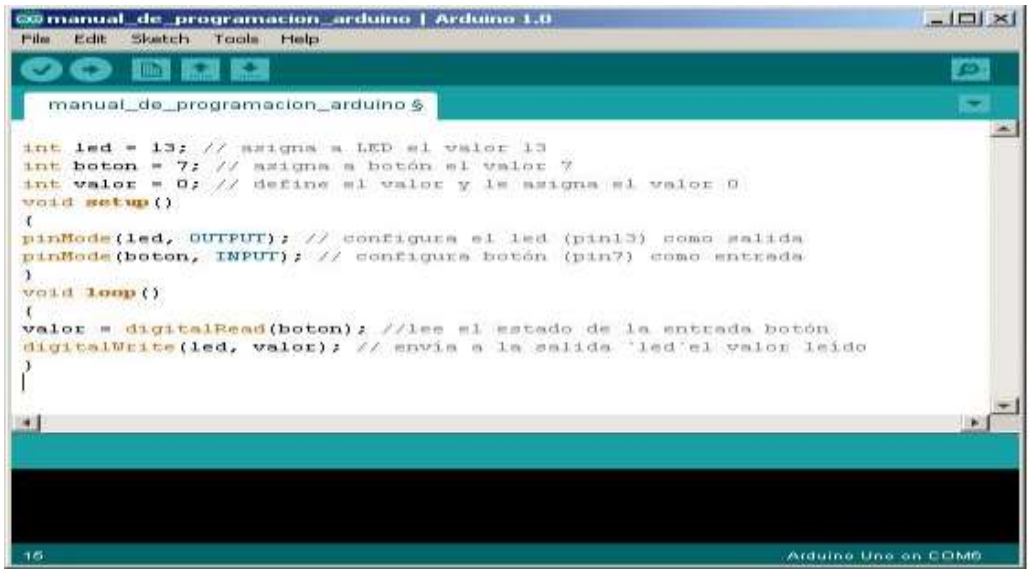


Figura 4.17: Leer el estado de un pulsador a una entrada digital

Fuente: Los Autores

4.2.2.21 Analogread (pin)

Lee el valor de un determinado pin definido como entrada analógica con una resolución de 10 bits. Esta instrucción sólo funciona en los pines (0-5). El rango de valor que puede leer oscila de 0 a 1023.

Valor = analogread (pin); // asigna a valor lo que lee en la entrada pin

Nota: Los pines analógicos (0-5) a diferencia de los pines digitales, no necesitan ser declarados como INPUT u OUPUT ya que son siempre INPUT´s.

4.2.2.22 Analogwrite (pin, value)

Esta instrucción sirve para escribir un pseudo-valor analógico utilizando el procedimiento de modulación por ancho de pulso (PWM) a uno de los pines de Arduino marcados como “pin PWM”. El más reciente Arduino, que implementa el chip ATmega168, permite habilitar como salidas analógicas tipo PWM los pines 3, 5,

6, 9, 10 y 11. Los modelos de Arduino más antiguos que implementan el chip ATmega8, solo tiene habilitadas para esta función los pines 9, 10 y 11. El valor que se puede enviar a estos pines de salida analógica puede darse en forma de variable o constante, pero siempre con un margen de 0-255.

Analogwrite (pin, valor); // escribe valor en el pin definido como analógico

Si enviamos el valor 0 genera una salida de 0 voltios en el pin especificado; un valor de 255 genera una salida de 5 voltios de salida en el pin especificado. Para valores de entre 0 y 255, el pin saca tensiones entre 0 y 5 voltios - el valor HIGH de salida equivale a 5v (5 voltios). Teniendo en cuenta el concepto de señal PWM, por ejemplo, un valor de 64 equivaldrá a mantener 0 voltios de tres cuartas partes del tiempo y 5 voltios a una cuarta parte del tiempo; un valor de 128 equivaldrá a mantener la salida en 0 la mitad del tiempo y 5 voltios la otra mitad del tiempo, y un valor de 192 equivaldrá a mantener en la salida 0 voltios una cuarta parte del tiempo y de 5 voltios de tres cuartas partes del tiempo restante.

Debido a que esta es una función de hardware, en el pin de salida analógica (PWN) se generará una onda constante después de ejecutada la instrucción analogWrite hasta que se llegue a ejecutar otra instrucción analogWrite (o una llamada a digitalReadodigitalWrite en el mismo pin).

Nota: Las salidas analógicas a diferencia de las digitales, no necesitan ser declaradas como INPUT u OUTPUT.

El siguiente ejemplo lee un valor analógico de un pin de entrada analógica, convierte el valor dividiéndolo por 4, y envía el nuevo valor convertido a una salida del tipo PWM o salida analógica:

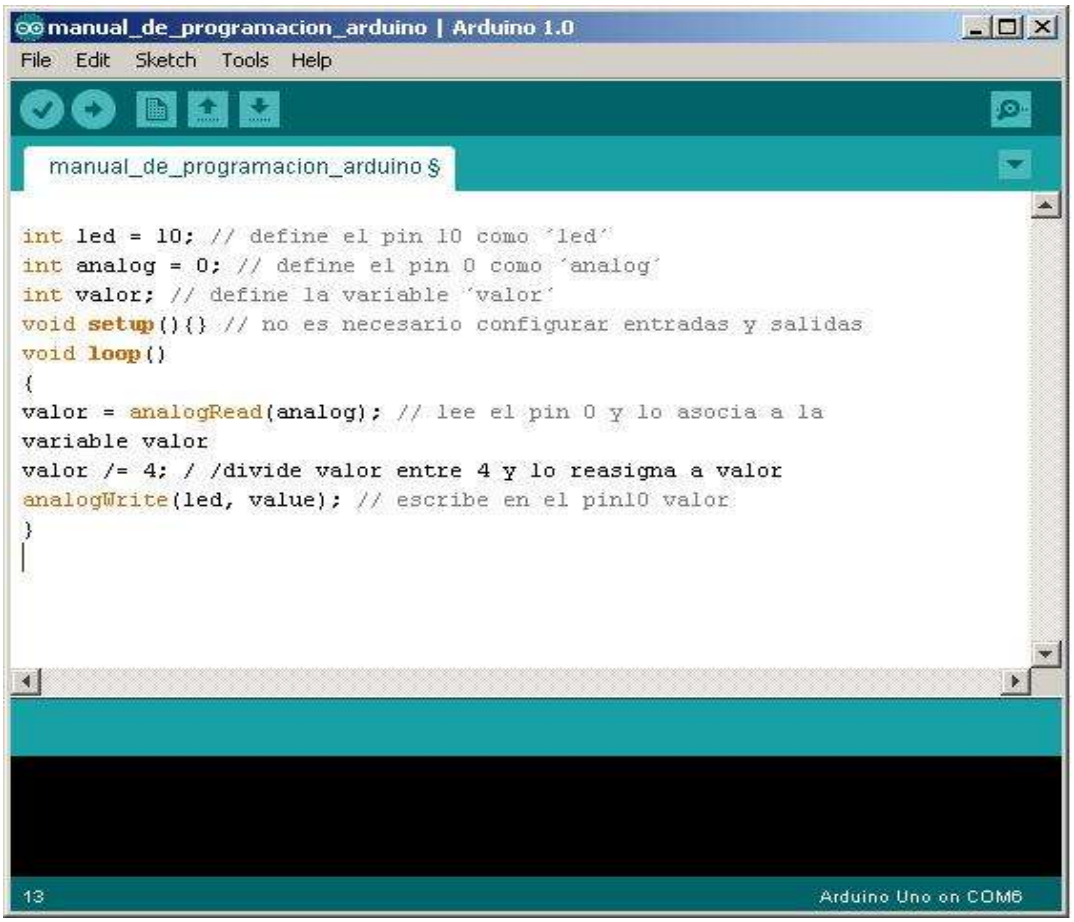


Figura 4.18: Lee un valor analógico y convierte en una salida del tipo PWM

Fuente: Los Autores

4.2.2.23 Delay (MS)

Detiene la ejecución del programa la cantidad de tiempo en ms que se indica en la propia instrucción. De tal manera que 1000 equivale a 1seg.

Delay(1000); // espera 1 segundo

4.2.2.24 Serial.begin (rate)

Abre el puerto serie y fija la velocidad en baudios para la transmisión de datos en serie. El valor típico de velocidad para comunicarse con el ordenador es 9600, aunque otras velocidades pueden ser soportadas.

```
Void setup ()
{
      Serial.begin (9600);    // abre el puerto serie
                              // Configurando la velocidad en 9600 bps
}
```

Nota: Cuando se utiliza la comunicación serie los pines digital 0 (RX) y 1 (TX) no puede utilizarse al mismo tiempo.

4.2.2.25 Serial.println (data)

Imprime los datos en el puerto serie, seguido por un retorno de carro automático y salto de línea. Este comando toma la misma forma que Serial.println (), pero es más fácil para la lectura de los datos en el Monitor Serie del software.

```
Serial.println (analogvalue);   // envía el valor analogvalue al puerto
```

Nota: Para obtener más información sobre las distintas posibilidades de Serial.println () y Serial.print () puede consultarse el sitio web de Arduino.

El siguiente ejemplo toma de una lectura analógica pin0 y envía estos datos al Ordenador cada 1 segundo.

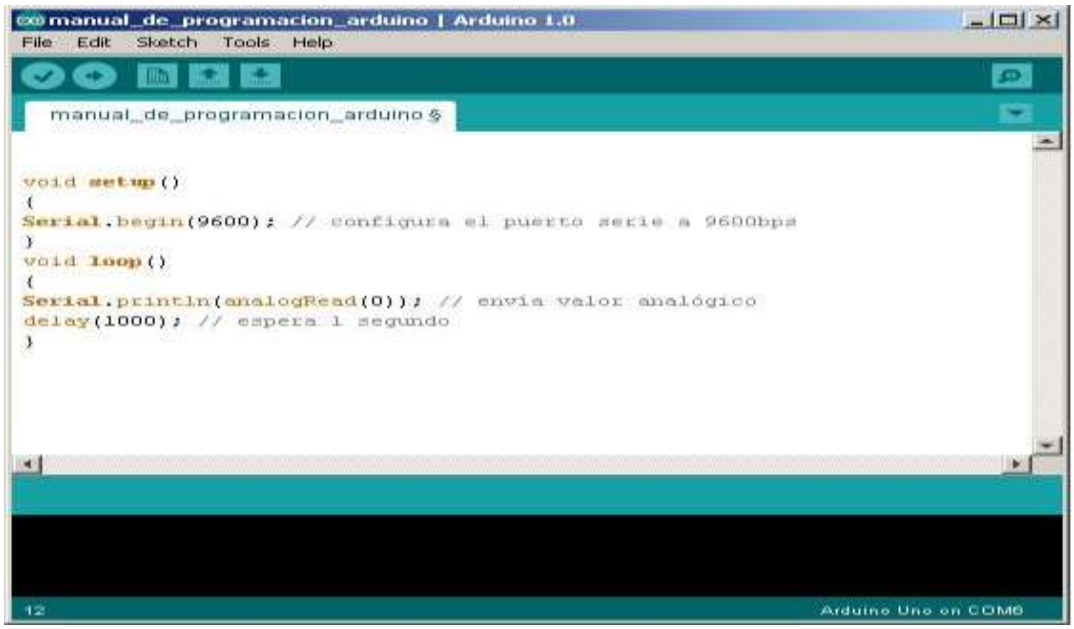


Figura 4.19: Utilización del comando “Serial.print”

Fuente: Los Autores

4.2.2.26 Serial.println (data, data type)

Envía un número o una cadena de caracteres al puerto serie, seguido de un caractér de retorno de carro "CR" (ASCII 13, or '\r') y un caractér de salto de línea "LF"(ASCII 10, or '\n'). Toma la misma forma que el comando Serial.print ().

Serial.println (b) vuelca o envía el valor de b como un número decimal en caracteres ASCII seguido de "CR" y "LF".

Serial.println (b, DEC) vuelca o envía el valor de b como un número decimal en caracteres ASCII seguido de "CR" y "LF".

Serial.println (b, HEX) vuelca o envía el valor de b como un número hexadecimal en caracteres ASCII seguido de "CR" y "LF".

Serial.println (b, OCT) vuelca o envía el valor de b como un número Octal en caracteres ASCII seguido de "CR" y "LF".

Serial.println (b, BIN) vuelca o envía el valor de b como un número binario en caracteres ASCII seguido de "CR" y "LF".

Serial.print (b, BYTE) vuelca o envía el valor de b como un byte seguido de "CR" y "LF".

Serial.println (str) vuelca o envía la cadena de caracteres como una cadena ASCII seguido de "CR" y "LF".

Serial.println () sólo vuelca o envía "CR" y "LF". Equivale a printNewline ().

4.2.2.27 Serial.print (data, data type)

Envía un número o una cadena de caracteres, al puerto serie. Dicho comando puede tomar diferentes formas, dependiendo de los parámetros que utilicemos para definir el formato de volcado de los números.

Parámetros:

data: el número o la cadena de caracteres a volcar o enviar.

Datatype: determina el formato de salida de los valores numéricos (decimal, octal, binario, etc...) DEC, OCT, BIN, HEX, BYTE, si no se vuelva ASCII

Ejemplos:
Serial.print (b)

Vuelca o envía el valor de b como un número decimal en caracteres ASCII. Equivale printInteger ().
int b = 79; Serial.print(b); // prints the string "79".

Serial.print (b, DEC)

Vuelca o envía el valor de b como un número decimal en caracteres ASCII. Equivale a printInteger ().
int b = 79;
Serial.print (b, DEC); // prints the string "79".

Serial.print (b, HEX)

Envía el valor de b como un número hexadecimal en caracteres ASCII. Equivale a printHex ();

int b = 79;
Serial.print (b, HEX); // prints the string "4F".

Serial.print (b, OCT)

Vuelca o envía el valor de b como un número Octal en caracteres ASCII. Equivale printOctal ();

int b = 79;
Serial.print (b, OCT); // prints the string "117".

Serial.print (b, BIN)

Vuelca o envía el valor de b como un número binario en caracteres ASCII. Equivale a printBinary ();

int b = 79;
Serial.print (b, BIN); // prints the string "1001111".

Serial.print (b, BYTE)

Vuelca o envía el valor de b como un byte. Equivaldría a printByte ();

int b = 79;
Serial.print (b, BYTE); // Devuelve el caractér "O", el cual representa el caractér ASCII del valor 79.

Serial.print (str)

Vuelca o envía la cadena de caracteres como una cadena ASCII. Equivale a printString ().

Serial.print ("Hello World!"); // envía "Hello World!"

4.2.2.28 Serial.avaible ( )

intSerial.available ()

Obtiene un número entero con el número de bytes (caracteres) disponibles para leer o capturar desde el puerto serie. Equivale a la función serial.available ().
Devuelve Un entero con el número de bytes disponibles para leer desde el buffer serie, o 0 si no hay ninguno. Si hay algún dato disponible, SerialAvailable () será mayor que 0. El buffer serie puede almacenar como máximo 64 bytes.

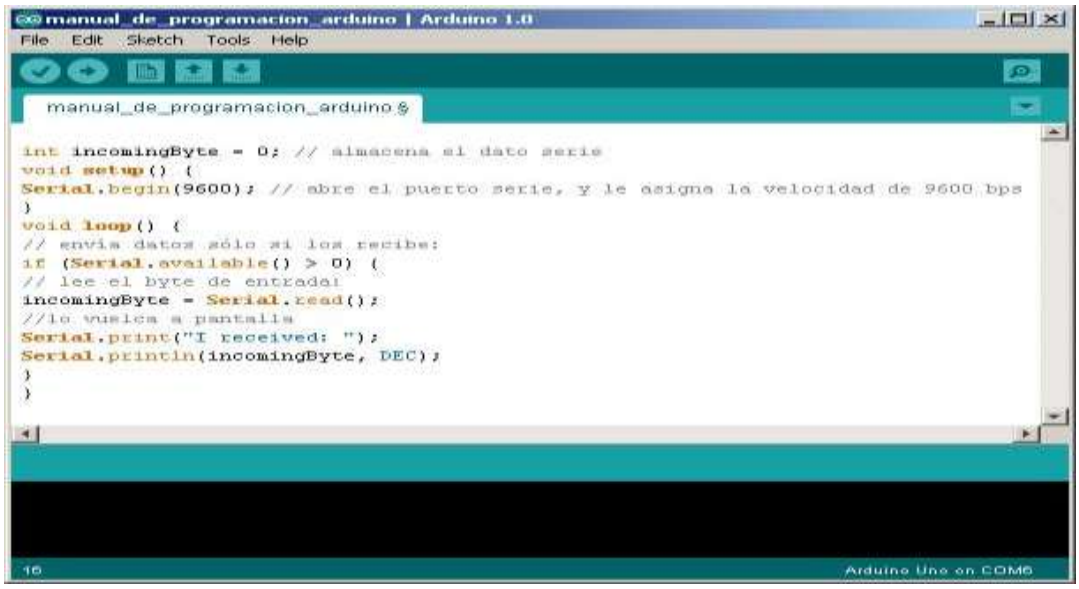


Figura 4.20: Función "Serial.avaible"

Fuente: Los Autores

4.2.2.29 Serial.read ( )

IntSerial.Read () Lee o captura un byte (un caractér) desde el puerto serie. Equivale a la función serialRead ().

Devuelve: El siguiente byte (carácter) desde el puerto serie, o -1 si no hay ninguno.

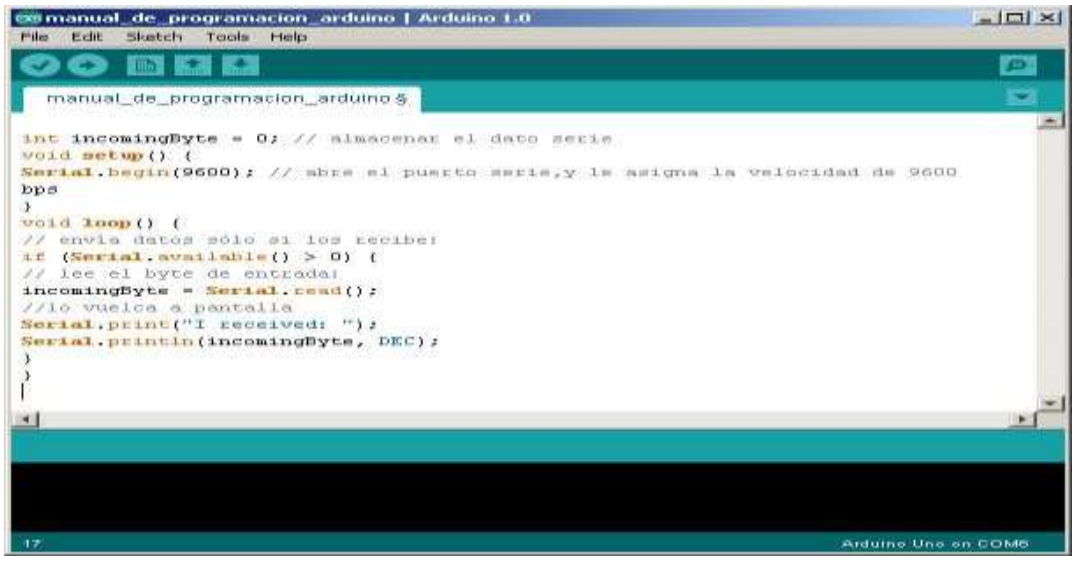


Figura 4.21: Función "serialread"

Fuente: Los Autores

4.4 CONFIGURACIÓN DE UNA PLACA ARDUINO EN UN "DAQ" UTILIZANDO LABVIEW.

Gracias a National Instruments, han creado componentes para Labview (VI) y el firmware necesario para convertir una placa de Arduino en una tarjeta de adquisición de datos y poder manejarla por USB desde Labview.

Requisitos necesarios:

- Un Arduino (Obvio) basado en el ATMEGA328 o superior (en el ATMEGA168 no cabe el firmware). Vale un Duemilanove, pero en nuestro caso será un Arduino UNO.
- El IDE de Arduino (versión 1)
- Labview 2009 o superior (vale la versión trial)
- Kit de drivers VISA.
- VIPM (VI package manager).
- Labview Interface for Arduino (en adelante LIFA)

Lo primero que se necesita es instalar Labview.

Descargar la versión de evaluación del siguiente enlace, aunque para hacerlo primero se tendrá que registrar.

http://www.ni.com/trylabview/

Una vez instalado Labview, proceder a instalar el pack de drivers de VISA, que se debe descargarlo desde la siguiente página. Se ha instalado la versión 5.1.1, pero funciona con la versión 5.0.3 y ya han sacado la 5.1.2.

NI-VISA 5.1.1 (http://joule.ni.com/nidu/cds/view/p/id/2659/lang/en)

Después instalar el gestor de paquetes VI (VIPM).Descargar la versión de la comunidad (free) de la siguiente página. Este programa será el que descargue e instale los VI de Arduino en Labview.

JKI VI Package Manager. (http://www.jki.net/vipm/download)

Una vez instalado todo esto, ejecutar el gestor de VI (VIPM) y buscar "Arduino". Solo se tiene que seleccionar el paquete de la lista e instalarlo. Los paquetes de componentes del LIFA se pueden descargar e instalar manualmente, pero esta forma es mucho más fácil. Cuando termine, se habrá completado la parte de instalación del Labview, nos queda la parte del Arduino.

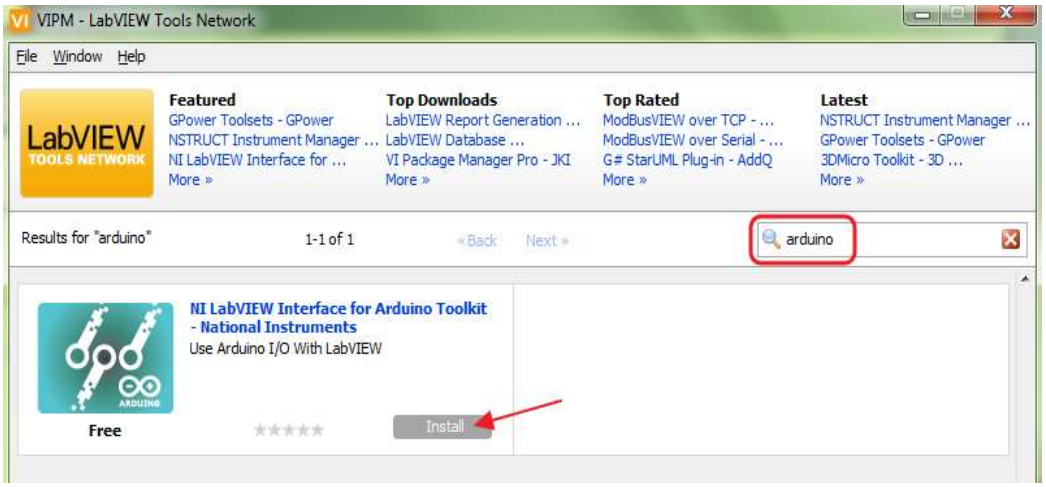


Figura 4.22: Reconocimiento e instalación de Labview y Arduino

Fuente: Los Autores

El IDE de Arduino se descarga directamente de la página de Arduino. En la página de Arduino en Español, los enlaces a las últimas versiones no están actualizados, se quedaron en la versión 0019.se ha utilizado la versión 1.0. En el siguiente enlace está la lista de las últimas versiones del IDE. (www.arduino.org)

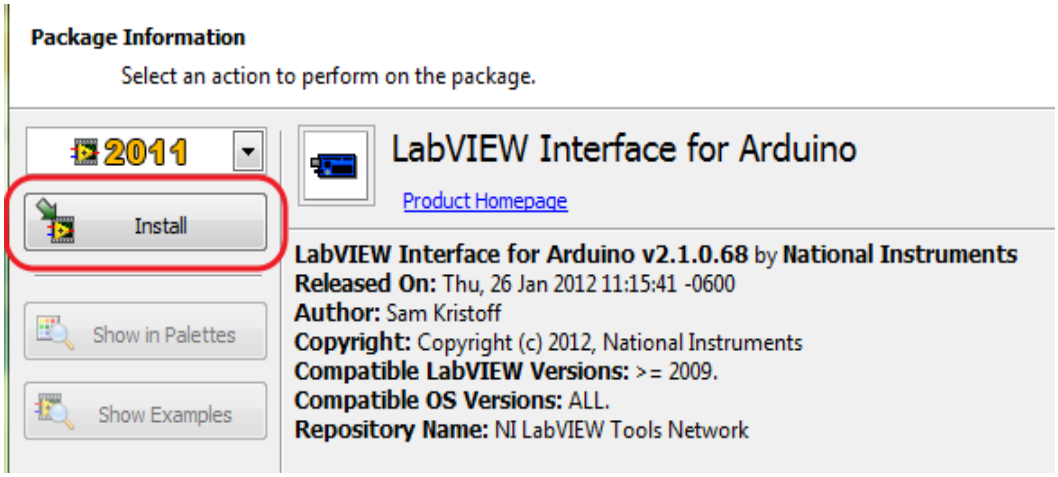


Figura 4.23: Instalación de la interface Labview y Arduino

Fuente: Los Autores

Una vez descargado lo descomprimimos y se crea un enlace en el menú de programas o el escritorio si lo prefieren. Con esto ya está todo instalado y solo se tiene que cargar el firmware del LIFA al Arduino para que se pueda comunicar con Labview.

Lo primero es conectar la placa de Arduino al USB. Si es la primera vez que se lo realiza se deberá instalar los drivers de Windows (si es que todavía se usa este SO). No se preocupe que el IDE de Arduino ya los trae, solo hay que decirle al sistema en que carpeta se encuentran. Conectar el Arduino y se aparece el típico globo que avisa de que no se han encontrado los drivers del nuevo dispositivo, hacer clic en él y da la opción de buscar los drivers. Si no aparece, ir directamente al administrador de dispositivos haciendo clic derecho sobre "Equipo" (Mi PC en XP) y seleccionando Administrar en el menú emergente. Una vez abierto el administrador de dispositivos nos encontraremos con el Arduino desconocido.

Figura 4.24: Administrador de dispositivo para encontrar Arduino

Fuente: Los Autores

Hacer clic con el botón derecho sobre el dispositivo desconocido y elegir la opción "Actualizar software de controlador". En la siguiente ventana elegir la opción "Buscar software de controlador en el equipo". Ahora se pedirá que se indique en que carpeta están los drivers de Arduino. Se tendrá que seleccionar la carpeta "drivers" que hay dentro del directorio de Arduino.

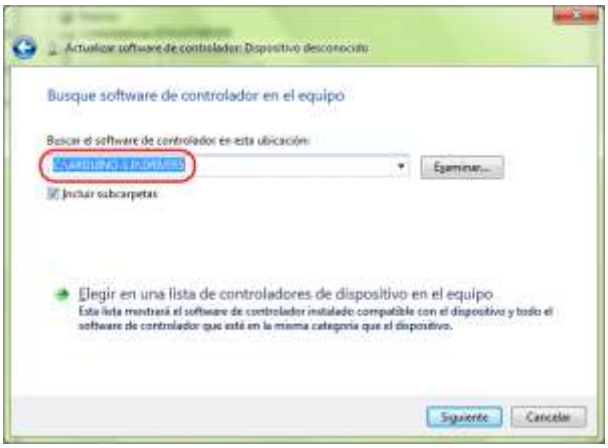


Figura 4.25: Buscar software de controlador en el equipo

Fuente: Los Autores

Y aceptamos la instalación del driver, tras lo cual, nuestro Arduino estará reconocido en el sistema como un puerto serie. El número de puerto serie lo necesitaremos en los siguientes pasos.

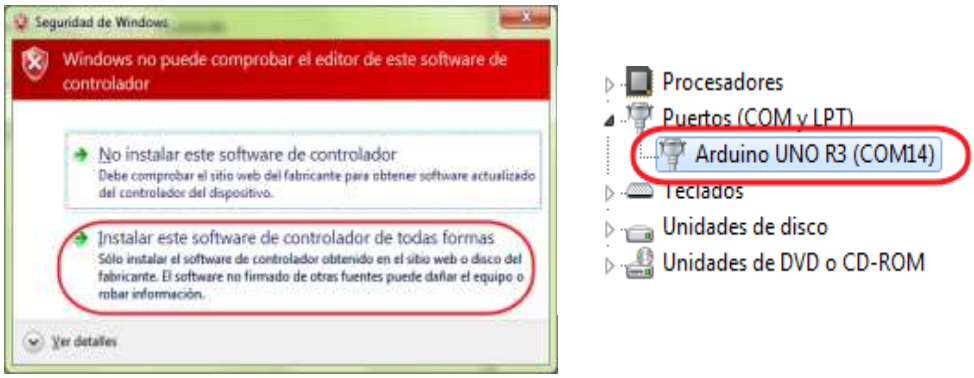


Figura 4.26: Instalación de driver y reconocimiento del puerto serial

Fuente: Los Autores

Se tiene Arduino conectado al USB y reconocido por el sistema, ahora toca programar el Arduino con el sketch que comunica la placa con Labview. El sketch de Arduino se instala con LIFA, por lo que tenemos que ir a la carpeta donde se ha instalado. La ruta se muestra en la siguiente imagen (figura 4.27). En el directorio LVIFA_Base, se encuentra el archivo LVIFA_Base.pde que es el sketch que debemos abrir. Se abrirán además todas las librerías. Cuando todo esté abierto se programará en la placa de Arduino. Primero seleccionar el tipo de placa que se está utilizando, que en este caso es Arduino UNO. Se seleccionará desde el menú *tools*, en la opción *Board*.

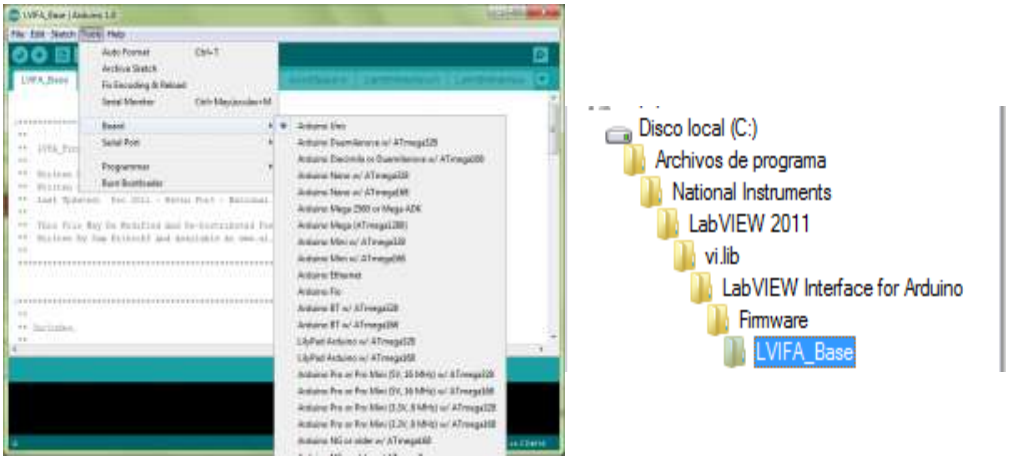


Figura 4.27: Instalación Arduino con el sketch que comunica la placa con Labview.

Fuente: Los Autores

Ahora elegir el puerto serie (COM) que utiliza el sistema para comunicarse con la placa de Arduino. En este caso es el COM14, pero el sistema puede haber asignado cualquier otro. Lo puede averiguar en el Administrador de dispositivos. El puerto se selecciona en la opción "Serial Port" del menú Tools.

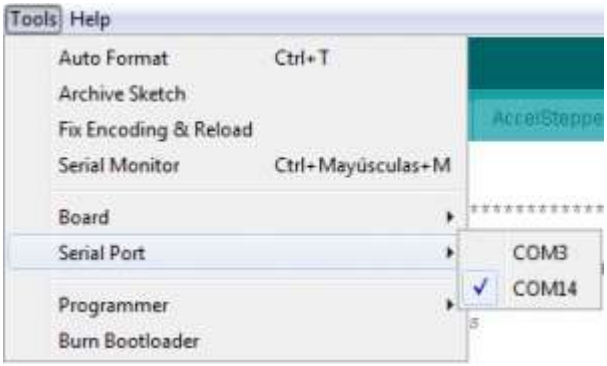


Figura 4.28: Seleccionando el puerto para la comunicación con la placa Arduino

Fuente: Los Autores

Y se puede programar la placa con el botón de la flecha de la parte superior.

Figura 4.29: Selección del botón para programar la placa Arduino

Fuente: Los Autores

Eso es todo. Con esto ya tenemos lista la placa de Arduino para comunicarse con Labview y tenemos montones de VI que nos van a permitir hacer fácilmente aplicaciones para controlar procesos usando Arduino como tarjeta de adquisición de datos.

4.4 ENVIÓ DE UNA SEÑAL ANALÓGICA DESDE DE UN POTENCIÓMETRO HACIA LABVIEW. .

4.4.1 Componentes utilizados

- Protoboard – que incluye un potenciómetro de 10K Ohmio.
- Tarjeta Arduino Uno, convertida en DAQ
- Tarjeta Arduino Uno, que se convertirá en una tarjeta controladora (donde ésta poseerá la programación)
- Laptop – que incluye instalación de Labview y sus módulos para Arduino.

4.4.2 Objetivo

El objetivo de esta fase de prueba es conocer el entorno de programación de Arduino y como es la interacción entre Arduino y Labview

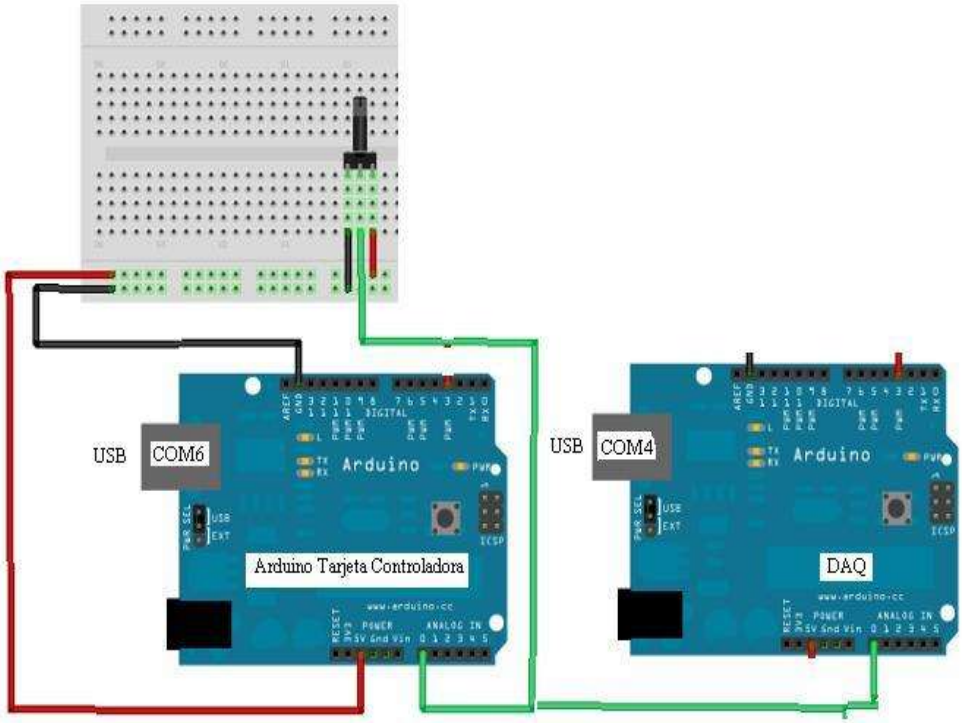


Figura 4.30: Envió de una señal analógica (potenciómetro) a la placa Arduino.
Fuente: Los Autores

Como se puede mostrar en la imagen anterior (figura 4.30), ese es el modo de conexión de los componentes físicos a utilizar en esta práctica.

Cuando el sistema arranque en Labview, al momento de mover el potenciómetro se visualizara el cambio de la señal analógica de 0 a 5 V ac.

4.4.3 Pantalla programación de Arduino - potenciómetro

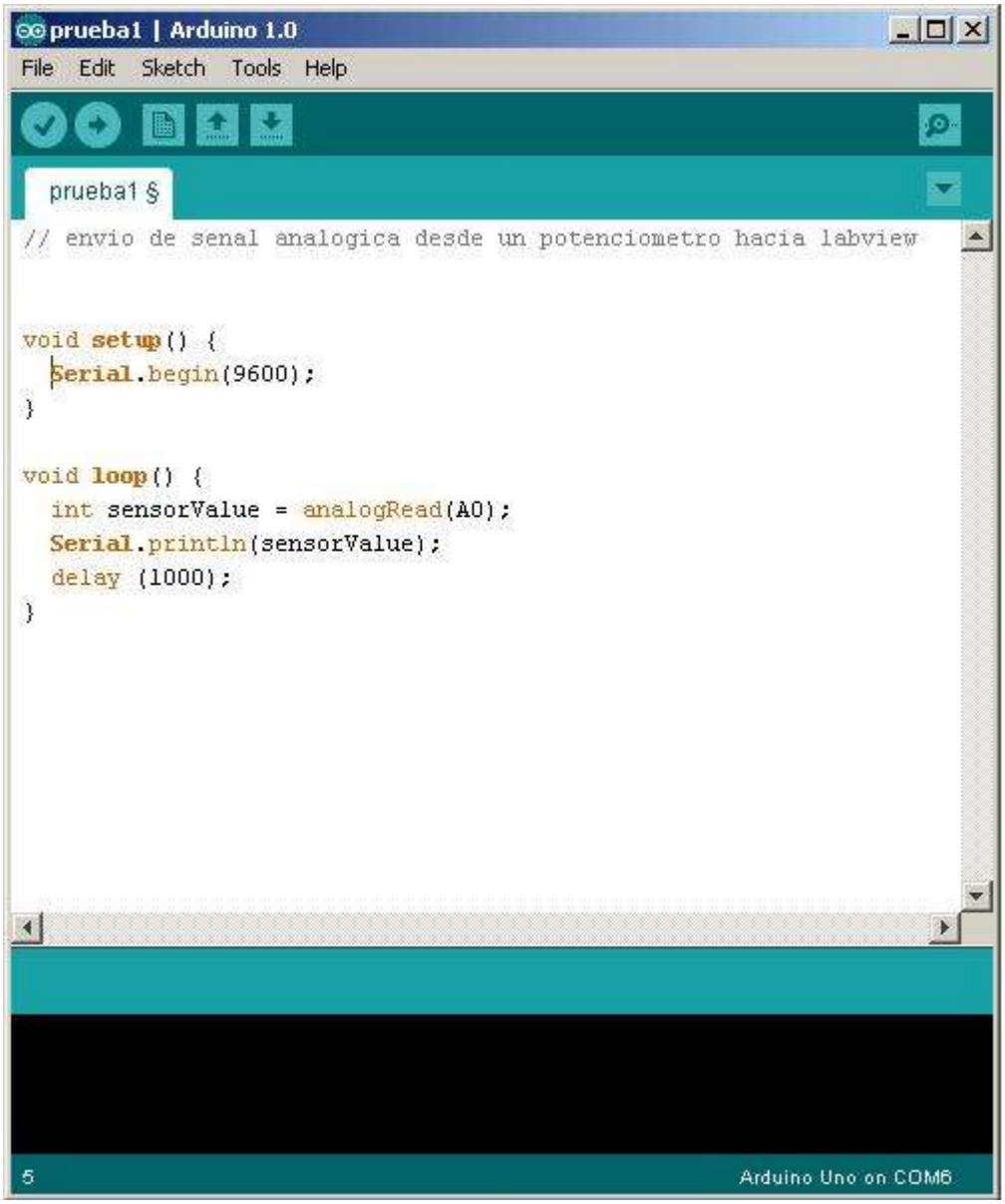


Figura 4.31: Programación con el software Arduino

Fuente: Los Autores

4.4.4 Pantalla Labview: Panel frontal

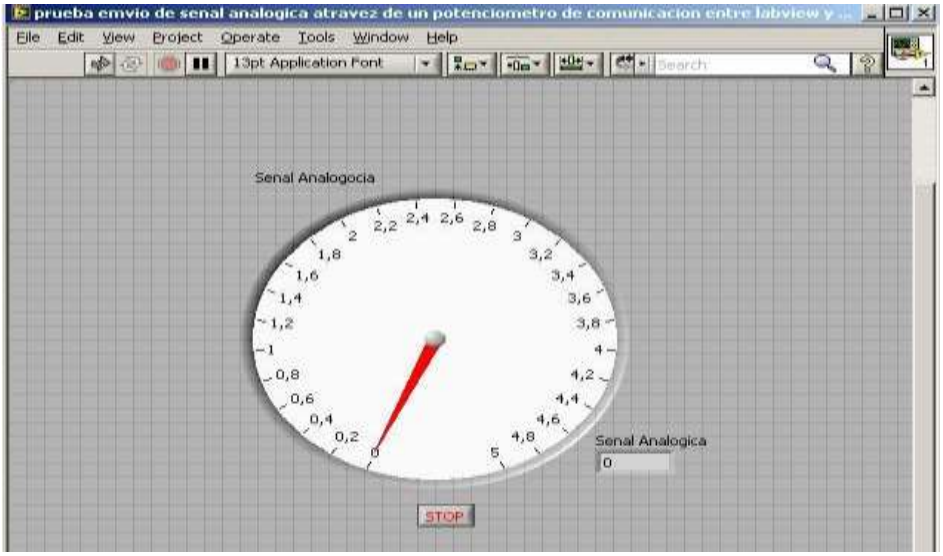


Figura 4.32: Software Labview

Fuente: Los Autores

4.4.5 Pantalla Labview: Diagrama de bloques

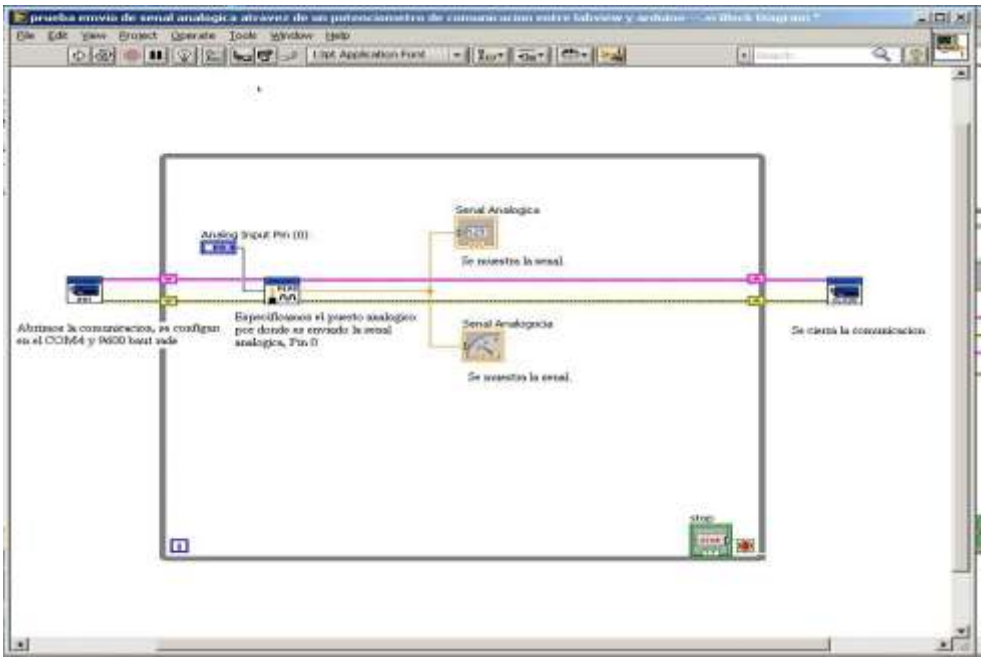


Figura 4.33: Diagrama bloque del software Labview

Fuente: Los Autores

4.5 INTERACCIÓN Y COMUNICACION ENTRE ARDUINO Y EL SERVOMOTOR.

Tomando en cuenta con la gran variedad de ejemplos y tutoriales que posee esta potente herramienta llamado Arduino en software y hardware. Se toma los ejemplos para la comunicación e interacción entre Arduino y el Servomotor. A continuación su explicación:

4.5.1 Componentes utilizados

- Protoboard – que incluye un potenciómetro de 10K Ohmio.
- Tarjeta Arduino Uno, que se convertirá en una tarjeta controladora (donde ésta poseerá la programación).
- Servo motor estándar HITEC 311 (180º)
- Fuente de Poder (5V dc, 1A).

4.5.2 Objetivo

El objetivo de esta prueba es conocer más el entorno de programación de Arduino y como es la interacción entre Arduino y los compones de hardware a utilizar en la presente tesis.

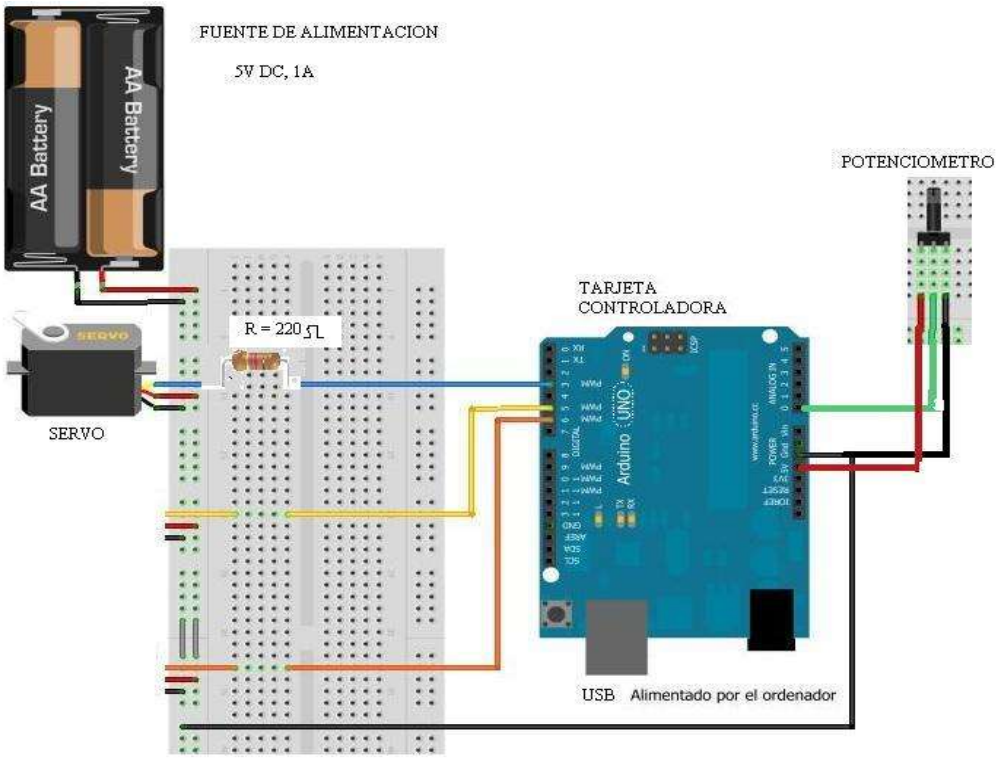


Figura 4.34: Interacción y comunicación entre Arduino y el servomotor

Fuente: Los Autores

Como se puede mostrar en la imagen anterior, ese es el modo de conexión de los componentes físicos a utilizar en esta práctica.

Descripción:

Cuando el sistema se energice, el Servomotor, se moverá siempre cuando se varié la señal enviada a través del potenciómetro. Cabe señalar que el Servomotor cubre un ángulo máximo de 180°.

4.5.3 Pantalla programación de Arduino-servomotor

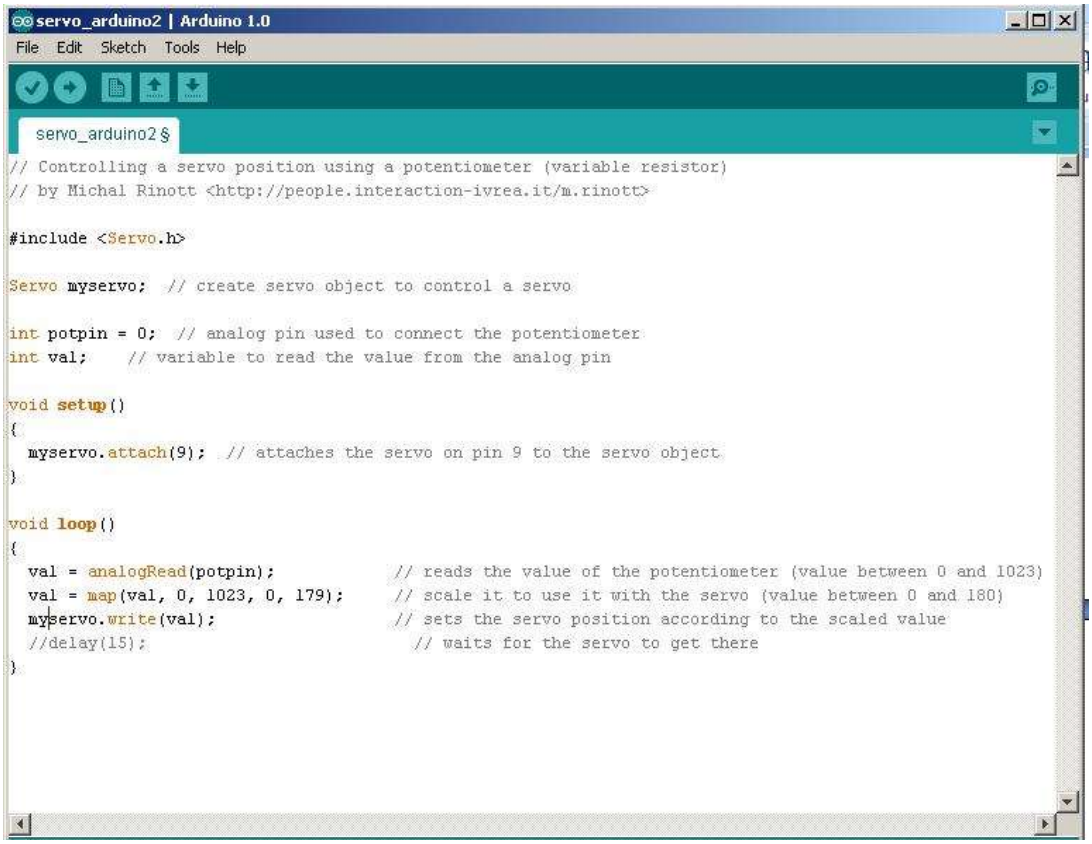


Figura 4.35: Programación del servomotor con el software Arduino

Fuente: Los Autores

4.6 INTERACCIÓN Y COMUNICACIÓN ENTRE ARDUINO Y EL SENSOR "MTS360"

La siguiente fase de prueba es comunicar el Sensor MTS360 con la tarjeta Arduino.

4.6.1 Componentes utilizados

- Protoboard – que incluye un potenciómetro de 10K Ohmio.
- Tarjeta Arduino Uno, que se convertirá en una tarjeta controladora (donde ésta poseerá la programación).
- Sensor MTS 360
- Fuente de Poder (5V dc, 1A).

4.6.2 Objetivo

El objetivo de esta fase prueba es conocer más el entorno de la programación de Arduino con y su interacción con los diferentes hardware (Sensor MTS360)

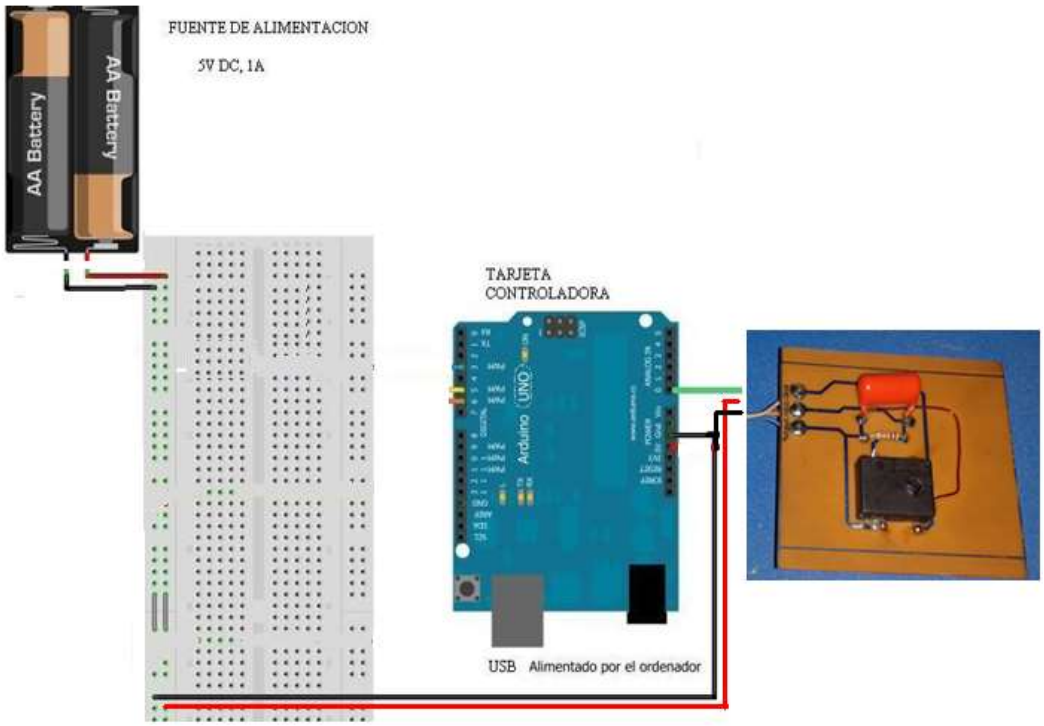


Figura 4.36: Interacción entre Arduino y el Sensor MTS360

Fuente: Los Autores

Como se puede mostrar en la imagen anterior, ese es el modo de conexión de los componentes físicos a utilizar en esta práctica.

Descripción:

En el visor de la comunicación Serial se mostrara la señal generada por el sensor en la cual está conectada a la entrada analógica del Pin A0. Para que se refleje la variación de la señal del sensor se tiene que introducir un eje pasante en la parte del dispositivo.

4.6.3 Pantalla programación de Arduino-sensor

Figura 4.37: Programación del sensor MTS360 con el software Arduino

Fuente: Los Autores

4.7 EXPLICACÍON GENERAL DEL FUNCIONAMIENTO DEL SISTEMA DE POSICÍON HORIZONTAL

Tomar en cuenta lo siguiente:

- El servomotor tiene un rango de operación de 0 a 180º.
- El sensor tiene un rango de operación de 0 a 360º, pero como el sistema solo cubre la mitad de una vuelta, se lo programará para que nos arroje un valor de 0 a 180º.

4.7.1 Inicio del sistema

El sistema inicia en la Posición 90º, que es la posición del servomotor, que para el usuario es 0º con respecto al Eje X. como se muestra en la siguiente figura(4.38).

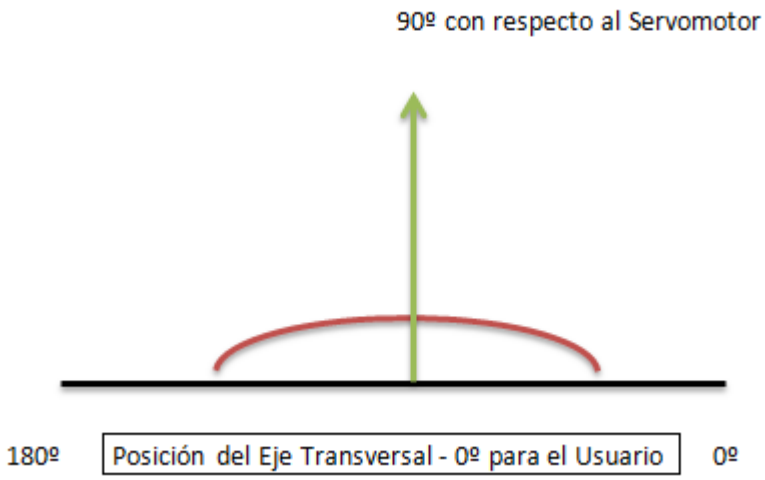


Figura 4.38: Posición del servomotor con respecto al sistema implementado
Fuente: Los Autores

Partiendo de este punto se puede hablar de las variables que cumplen una función específica y las operaciones matemáticas que estas deben de realizan (revisar diccionario de Datos). El usuario elegirá el eje: positivo o negativo

El usuario deberá ingresar un numero entre 0 a 90. Con el switch ON/OFF, el sistema iniciara y el eje transversal se moverá y formará un ángulo con el alerón.

Nuestra implementación es un sistema de control de lazo cerrado, en la cual se demostrará su estabilidad y su error (valor de entrada menos valor de salida).

4.8 DIAGRAMA DE FLUJO

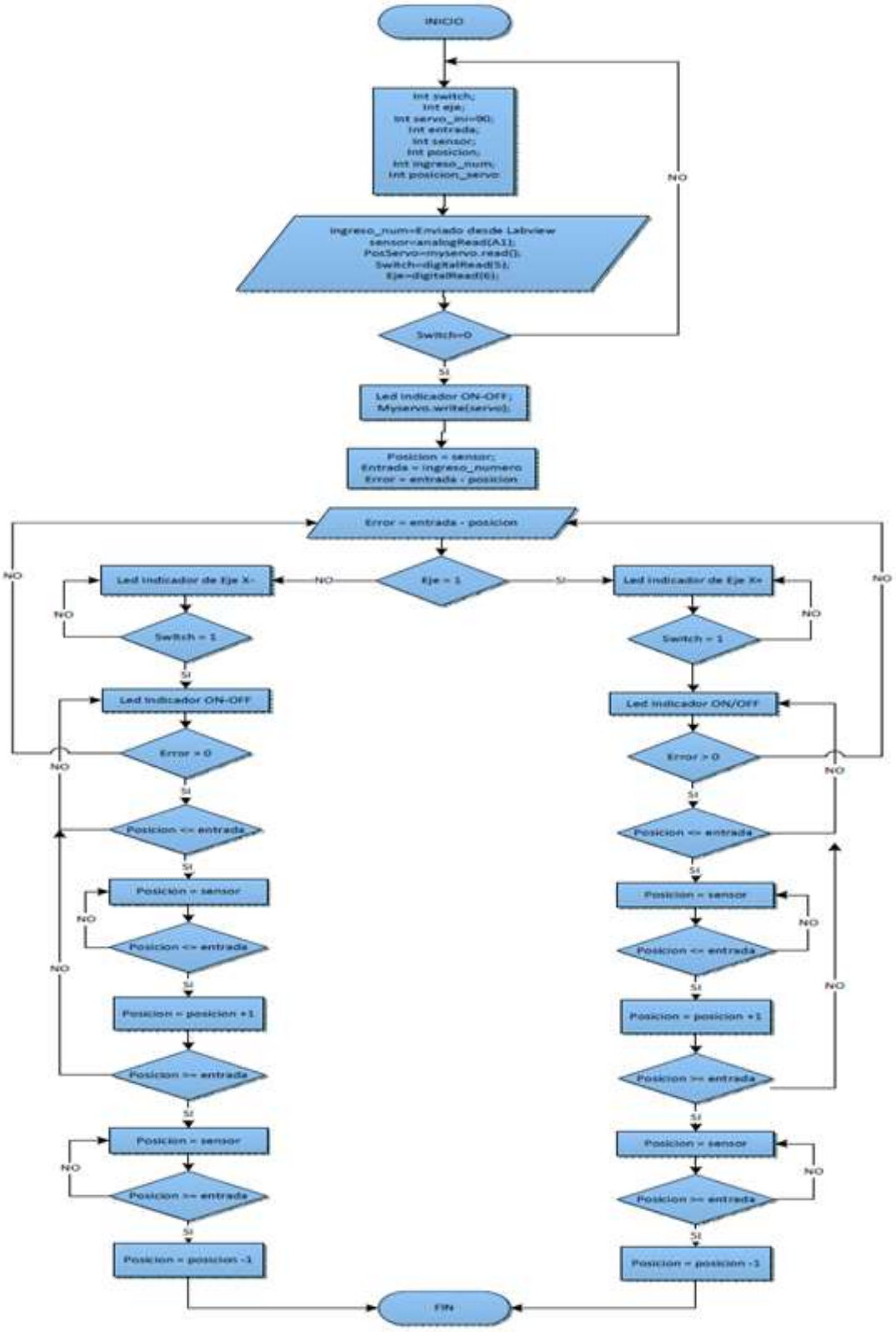


Figura 4.39: Diagrama de flujo del programa" sistema de posición horizontal"

Fuente: Los Autores

4.9 ESTRUCTURA DE LA PROGRAMACIÓN – ARDUINO – SISTEMA DE POSICÍON HORIZONTAL.

La estructura del lenguaje de programación de Arduino está formada de 3 partes principales:

1) Creación de variable e invocación de librerías.
2) La Función Void setup () es llamada una vez al iniciar el programa. Es la primera función al ejecutar el programa y se utiliza para establecer el modo de los pines (entrada o salida) o inicializar la comunicación serial.
3) La Función void loop () incluye el código que se ejecutará de forma continua, permitiendo que el programa responda a cambios y el control de la tarjeta Arduino.

Bajo estos conceptos, a continuación la programación de Arduino utilizada para nuestro proyecto.

4.9.1 Creación de Variables e invocación de librerías

```
#include <Servo.h> //invocación de librería para Servomotor
Servo myservo;     //creación de variable para servomotor
int sensor0=0, sensor180=0;   // variable para determinar función  lineal del sensor
int pin2=2, pin3=3, pin4=4, pin7=7, pin8=8, pin10=10, pin11=11, pin12=12;
//creación de variables de tipo entero
//para asignar el número de Puerto en DAQ
int entra2=0, entra3=0, entra4=0, entra7=0, entra8=0, entra10=0, entra11=0,
entra12=0;
int sumatotal=0;
int suma1=0, suma2=0, suma3=0, suma4=0, suma5=0, suma6=0, suma7=0,
suma8=0;
int Valsensor, i=0, Total=0, Promedio=0;   //filtro sensor
int Lectura[1];   // filtro sensor
```

```
int servo; //posición de inicio del servo
int pin5=5; // entrada del binario desde Labview EJE
int pin6=6; // entrada del binario desde Labview SWITCH
int eje ;       // Eje + , - Labview
int switch1 ;   // ON-OFF Labview
int     posfinal=0,error=0,entrada=0,sensor=0,sensor_artificio=0,posicion=0,error2=0,
sensor_artificio2=0;
int cont=0, suma=0, sumato=0, cont2=0, cont3=0, cont4=0;
```

Este es el inicio de la programación, En esta etapa se crea las diferentes variables que se vayan a utilizar en el transcurso de la programación y además la invocación de las librerías como se puede ver:

```
#include <Servo.h>   --  Invocación de librería para manejo del Servomotor.

Servo myservo;         --  Creación de Variable basada en la librería.
```

Y el resto de variable son de tipo de entero para guardar números enteros o decimales o valores binarios (1,0).

4.9.2 Inicio de la función void setup ()

```
void setup()
{
myservo.attach(9);  // indica que la entrada PWM del pin 9, es del SERVO
Serial.begin(9600);  // abrir la comunicación del puerto serie
for (i=0; i<1; i++) //filtro sensor
Lectura[i]=0;        // filtro sensor
i=0;                 //filtro sensor
}
```

En la función void setup se programa el pin donde se va ingresar la señal del servomotor en el Arduino controlador.

Además se abre la comunicación serial para leer el Arduino controlador lo que está procesando.

Un pequeño lazo for aplicado en la lectura del sensor, más conocido como filtro, que más adelante se explicara con detalles.

```
void loop()
{
servo=myservo.read();     // La lectura de la posicion del servomotor se guarda en la
                             variable servo
eje=digitalRead(pin5);      // La entrada digital dle pin 5 se guarda en la variable eje –
                              seleccion del eje + o -
switch1=digitalRead(pin6); // la entrada digital dle pin 6 se guarda en la variable
                                switch1-  Inicio del Sistema ON / OF
Serial.println("Sensor Sin Filtro");
delay(100);
Serial.println(analogRead(A1));
//Serial.println(Valsensor);
delay(1000);
if (switch1==0)
{
 myservo.write(90);  // Posicionamos al servo en 90°
 delay(100);            // retardo de 100 mili segundos
                          //Serial.println("SW");
}
```

4.9.3 Inicio y explicación de la estructura de programación

La variable eje contiene la salida digital (0 y 1) que envía el Labview a través del Arduino DAQ. Lo mismo se puede decir para la siguiente variable switch1. Que es el ON/OFF en el Labview.

La variable servo guarda la posición en él que se encuentra en cualquier instancia el Servomotor.

En la sentencia IF, se pregunta por la variable switch1 si es igual 0, si eso es verdad al servo motor lo posicionamos en 90°, y le damos un retardo de 100 mili segundos.

```
Serial.println("ANGULO SERVO");  // Imprime Mensaje mensaje
delay(100);                    // retardo    100 mili segundos
Serial.println(myservo.read());  //
delay(100);
Total = Total - Lectura[i];    //filtro sensor
Lectura[i]=analogRead(A1);     // la variable Lectura guarda la señal del sensor
Total = Total + Lectura[i];    //filtro sensor
i=i+1;                         //filtro sensor
if (i>=1)                      //filtro  sensor
{                              //filtro sensor
 i=0;                          //filtro sensor
Promedio = Total/1;            //filtro sensor
Valsensor map (Promedio, 236, 578, 0,180);
//función map es una ecuación lineal que toma el máximo y mínimo del sensor y lo presenta entre 0 a 180.
}    //filtro sensor
 Serial.println("Sensor Valor Real con Filtro");    // Imprime mensaje

delay(100);          // retardo de 100 mili segundos
```

```
Serial.println(Valsensor);   // Imprime el Valor del Sensor con el filtro.
delay(100);    // retardo de 100 mili segundos
```

En ésta parte de la programación se habla del filtro aplicado en el sensor. Actualmente este filtro solo toma un valor y saca el promedio para la cantidad de valores tomados. La variable lectura guarda los valores ingresados. La función map cumple lo que realiza una ecuación lineal. Como se explica en la programación. Las siguientes líneas de programación imprime el valor del sensor aplicado el filtro.

```
entra2=digitalRead(pin2);    //lectura de un bits menos significativo enviado desde Labview (8 bits)
entra3=digitalRead(pin3);
entra4=digitalRead(pin4);
entra7=digitalRead(pin7);
entra8=digitalRead(pin8);
entra10=digitalRead(pin10);
entra11=digitalRead(pin11);
entra12=digitalRead(pin12); ////lectura de un bits más significativo enviado desde
                                    Labview (8 bits)
                                  // Conversión de un numero Binario a Decimal
if (entra2==1)
{
suma1=1;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
}
else
{
suma1=0;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
 }
 if (entra3==1)
 {
```

```
suma2=2;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
 }
else
{
suma2=0;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
}
 if (entra4==1)
{
suma3=4;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
}
else
{
suma3=0;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
 }
 if (entra7==1)
 {
 suma4=8;
 sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
 }
 else
 {
 suma4=0;
 sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
 }
 if (entra8==1)
 {
 suma5=16;
 sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
}
else
```

```
 {
 suma5=0;
 sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
 }
 if (entra10==1)
{
 suma6=32;
 sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
}
else
{
suma6=0;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
}
 if (entra11==1)
{
suma7=64;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
 }
else
{
 suma7=0;
 sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
 }
 if (entra12==1)
{
 suma8=128;
 sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
 }
 else
 {
 suma8=0;
 sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
 } // Fin de la Conversión de Binario a Decimal
```

Las pantallas de programación anteriores realiza la conversión de un binario a decimal. El número binario es de 8 bits enviado desde Labview hacia el Arduino DAQ y este a su vez conectado a las entradas digitales del Arduino controlador. Las variables pin indica el número de PIN en el cual se está ingresando el número binario desde el Arduino DAQ. A continuación la siguiente imagen de conexión.

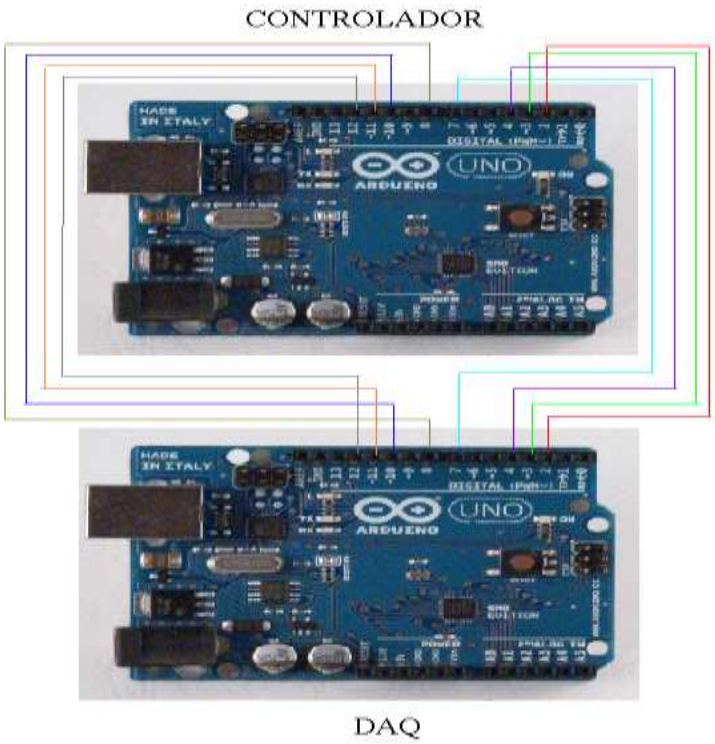


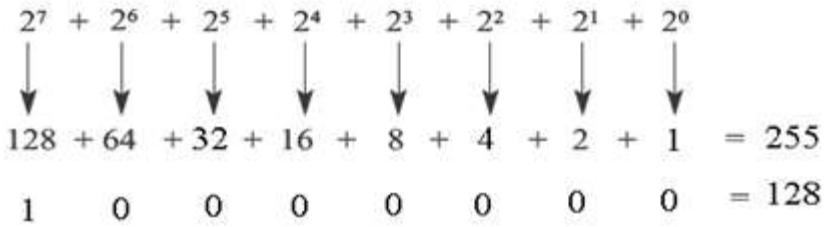
Figura 4.40: conexión entre controlador (placa Arduino) y DAQ

Fuente: Los Autores

Las variables pin2, pin3, pin4, pin7, pin8, pin10, pin11, pin12 guardan el número del pin del Arduino controlador. Las variables entra2, entra3, entra4, entra7, entra10, entra11, entra12 almacenan el número binario (1 o 0) enviado desde el Arduino DAQ.

En los IF consiguientes a la programación se pregunta si es un alto o un bajo que se está recibiendo y según el orden (bits menos significativos al más significativo) se otorgará el valor de 1, 2, 4, 8, 16, 32, 64, 128, La variable sumatotal es el resultado final de la conversión. Tomar en cuenta que la elección de programar un número de 7 bits fue debido que en la Interfaz del Usuario (Labview), el usuario deberá ingresar un número no mayor a 90. Como se describe en la siguiente imagen.

Sistema de numeración en base 2



Como última parte y la más importante de la programación en el sistema de control es poder darle al sistema la autonomía para que, por si solo pueda ejecutar movimientos arriba hacia abajo o viceversa.

Se preguntara por la variable switch1 ON/OFF que se encuentra en la interfaz de Labview y dependiendo del eje Positivo o negativo nuestro eje transversal se moverá de acuerdo al ángulo ingresado. Al llamar a la variable sensor se asigna el valor de la Variable Valsensor **(**osea el Valor real acondicionado al rango de operación 0 – 180º). Se crea además una variable sensor_artificio. También tenemos el número ingresado por el usuario en la variable sumato que se le asigna en la variable entrada. La variable posición tomara el valor de la variable sensor_artificio, y esta a su vez será la que se encuentre en la línea de programación (myservo.write(posición)) para que el servomotor se posicione.

```
if (eje ==1) // Pregunta posicíon del Eje + (Labview)
{

 if (switch1==1)  // Pregunta posicíon del ON/OFF (Labview)
{
sensor=Valsensor;
sensor_artificio=90-sensor;   //
Serial.println("Sensor con Artificio");
delay(100);
Serial.println(sensor_artificio);
delay(100);
posición=sensor_artificio;
suma = 0;
suma = sumatotal + 90 ;
sumato=suma -90;
entrada=sumato;
error=entrada-sensor_artificio;
error2=abs(error);
```

Teniendo estos datos aplicamos nosotros la diferencia entre entrada y sensor_artificio y obtenemos el valor del error, como se muestra en la siguiente figura de nuestro sistema de control de lazo cerrado.

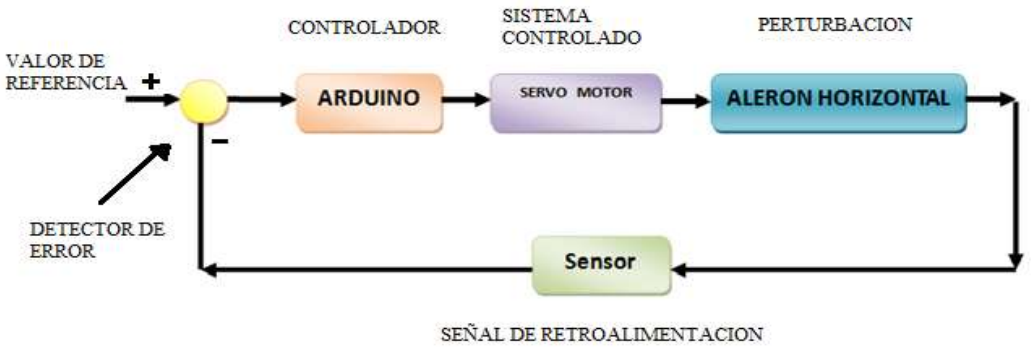


Figura 4.41: Sistema de control de lazo cerrado

Fuente: Los Autores

Preguntaremos por la variable error**,** si es mayor a cero entrara a un IF y preguntara si la posición es mayor o igual a la entrada si eso se cumple ingresara a un lazo for y el servomotor se posicionará dependiendo del valor que tenga la variable posición. La posición aumentará cada vez que las condiciones antes dichas se cumplan y así el servomotor cambiara paulatinamente su posición y éste se detendrá hasta que el error sea igual a cero, Tomar en cuenta que el movimiento del servo en esta línea de programación es de un solo sentido.

```
if (error > 0)
{
 if (posición<=entrada)
{
for (posición=sensor_artificio;posición<=entrada; posición+=1)
{
myservo.write(posicíon+90+0);       // posiciona al servo dependiendo del valor
ingresado
delay(100);
Serial.println("POSICÍON");
delay(10);
Serial.println(posicíon+90);
delay(10);
}

}

}
```

El sistema de control será más estable cuando el error se acerque más al cero.

A continuación si el error es menor a 0 y la posición es mayor o igual a la entrada, ingresara a un lazo for y el Servomotor se posicionará dependiendo del valor que tenga la variable posición. La variable posición disminuirá cada vez que las

condiciones antes dichas se cumplan y así el servomotor cambiará paulatinamente su posición y este se detendrá hasta que el error sea igual a cero. De esta manera el usuario cuando elija el eje Positivo el Servomotor podrá moverse en los dos sentidos, de arriba hacia abajo o viceversa. A continuación las líneas de programación.

```
 if (error < 0)
{
if (posición>=entrada)
{
delay(100);
for (posición=sensor_artificio;posicion>=entrada; posición-=1)
{
myservo.write(posicion+90-5);   // posiciona al servo dependiendo del valor
                                    ingresado
delay(100);
Serial.println("POSICIÓN");
delay(10);
Serial.println(posicion+90-5);
delay(10);
```

Definitivamente el sistema no iniciaré si el Switch se encuentra en estado OFF, aunque se haya elegido el eje y se haya ingresado el valor para posicionar al servomotor.

Como lo dicho anteriormente, si la variable switch1 es igual a cero (el sistema no iniciará) y se mantendrá el servomotor en la posición 90°, que para el usuario es 0° con respecto al eje de las X, a continuación la líneas de programación.

```
else if (switch1==0)
{
//Serial.println("VALOR CONT2");
```

```
suma=0;
myservo.write(90);
delay(10);
delay(100);
```

Lo mismo se puede decir cuando el usuario elija el eje negativo, se aplica la misma lógica. A continuación las siguientes pantallas de programación.

```
else if (eje==0)  // Pregunta posición del Eje - (Labview)
{
if (switch1==1)
suma=0;
sensor=Valsensor;
sensor_artificio=sensor-90;
sensor_artificio2=abs(sensor_artificio);
posición=sensor_artificio;
suma=0;
suma=90-sumatotal;    // forma el ángulo según el valor que ingresa con respecto al eje –X
sumato=90-suma;
entrada=sumato;
error=entrada-sensor_artificio
error2=abs(error);
Serial.println("ERROR");
delay(100);
Serial.println(error2);
delay(100);
 if(error>0)
{
if(posición<=entrada)
{
for (posicion=sensor_artificio;posicion<=entrada; posición+=1)
```

```
{
myservo.write(90-posicion-3); // posiciona al servo dependiendo del valor ingresado
delay(100);
Serial.println("POSICION");
delay(10);
Serial.println(90-posicion-3);
delay(10);
Serial.println("ERROR");
Serial.println(error2);
}
}
}
if (error < 0)
{
 if(posición>=entrada)
{
for (posición=sensor_artificio;posición>=entrada; posición-=1)
{
myservo.write(90-posición+6); // posiciona al servo dependiendo del valor ingresado
delay(100);
Serial.println("POSICIÓN ");
delay(10);
Serial.println(90-posición+6);
delay(10);
Serial.println("ERROR")
//delay(100);
Serial.println(error2);
//delay(100);
}
}
}
}
else if (switch1==0)
{
```

```
cont2=0;
suma=0;
myservo.write(90+5);
delay(100);
}
}
}
```

4.9.4 Diagrama esquemático de conexión.

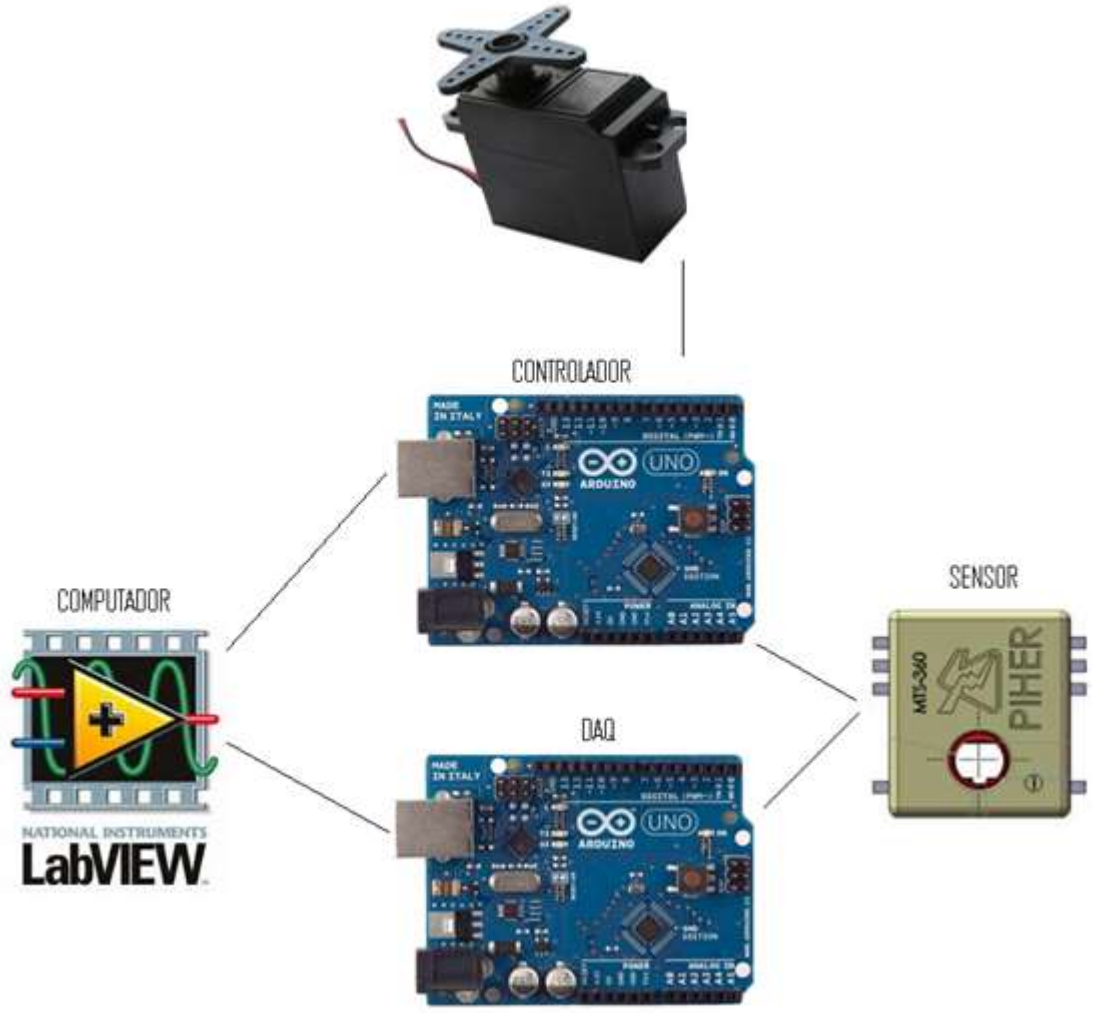


Figura 4.42: Diagrama Esquemático de Conexión

Fuente: Los Autores

4.10 PROGRAMACIÓN EN PLATAFORMA LABVIEW SISTEMA DE POSICIÓN HORIZONTAL.

Labview (acrónimo de Laboratory Virtual Instrumentation Engineering Workbench) es una plataforma y entorno de desarrollo para diseñar sistemas, con un lenguaje de programación visual gráfico. Está dividido en dos bloques fundamentales:

- **Interfaz gráfica de usuario -** Esta interfaz utilizada para interactuar con el usuario cuando el programa se está ejecutando.
- **Diagrama bloques. (programación Grafica) -** En este diagrama se define su funcionalidad, aquí se colocan iconos (VI) que realizan una función determinada y se interconectan con otros iconos (VI) para cumplir una tarea o una función específica.

Tomar en Cuenta que la compatibilidad de Arduino y Labview es desde la versión 2009 en adelante de Labview. Antes de empezar con el diseño de la interfaz gráfica para el usuario y el diagrama de bloques utilizado en el sistema de Control, se tiene que tener instalado lo siguiente:

- Instalación del Software Labview 2009 en adelante (Licenciado o Student Edition).
- Instalación del paquete Labview Interface for Arduino es totalmente gratuito y se puede descargar desde la página del fabricante: https://decibel.ni.com/content/groups/labview-interface-for-arduino.
- Instalación del Firmware de comunicación entre la Interfaz de Labview y la Tarjeta Arduino. (Revisar capitulo en el documento de Tesis).

Cumpliendo estos requerimientos se podrá decir que se estará listo para realizar la comunicación entre Labview y Arduino. Cabe indicar que la plataforma Labview está siendo utilizada como un visor de datos y no como un controlador, ya que como en nuestro proyecto está enfocado más en el estudio e implementación de la herramienta Arduino. La tarjeta Arduino en este caso será la controladora en el cual

lleve la programación que cumplirá las diferentes etapas y procesos de nuestro sistema de control.

4.10.1 Interfaz gráfica de usuario

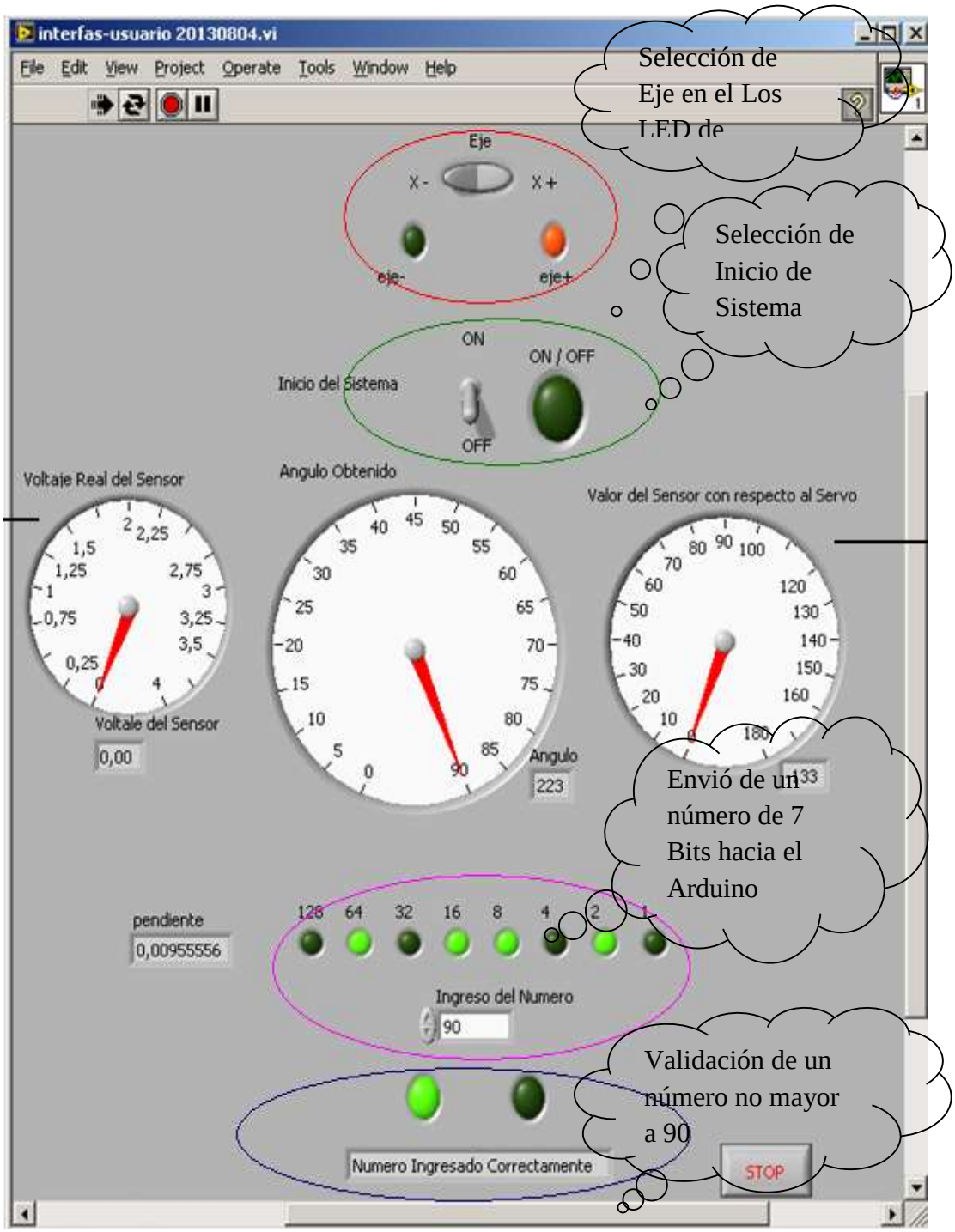


Figura 4.43: Interfaz gráfica de usuario en Labview

Fuente: Los Autores

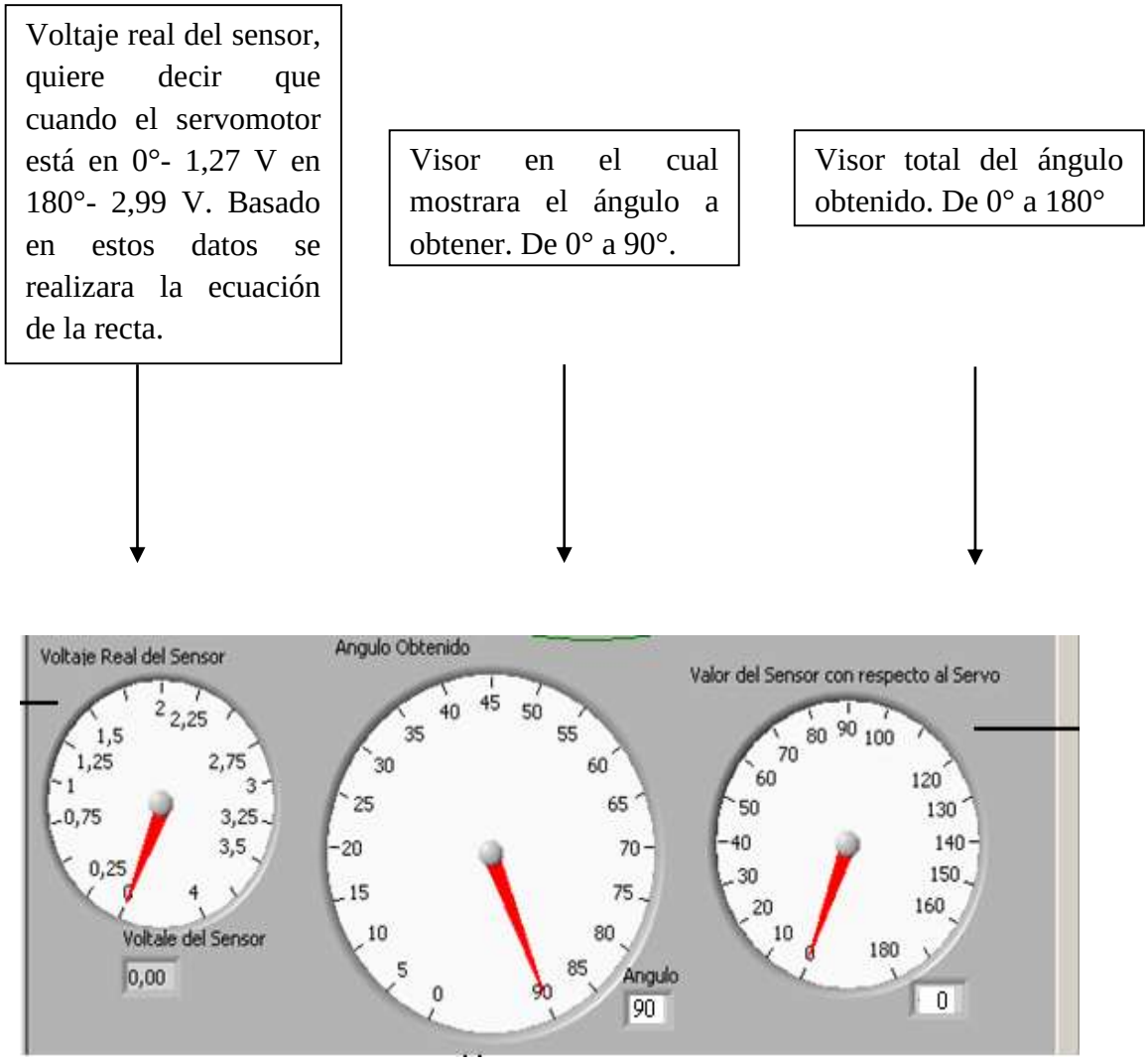


Figura 4.44: Interfaz gráfica de usuario en Labview (voltaje-ángulo obtenido-valor del sensor respecto al servo)

Fuente: Los Autores

4.10.2 Diagrama de bloques

Agregamos una estructura de tipo While Loop,

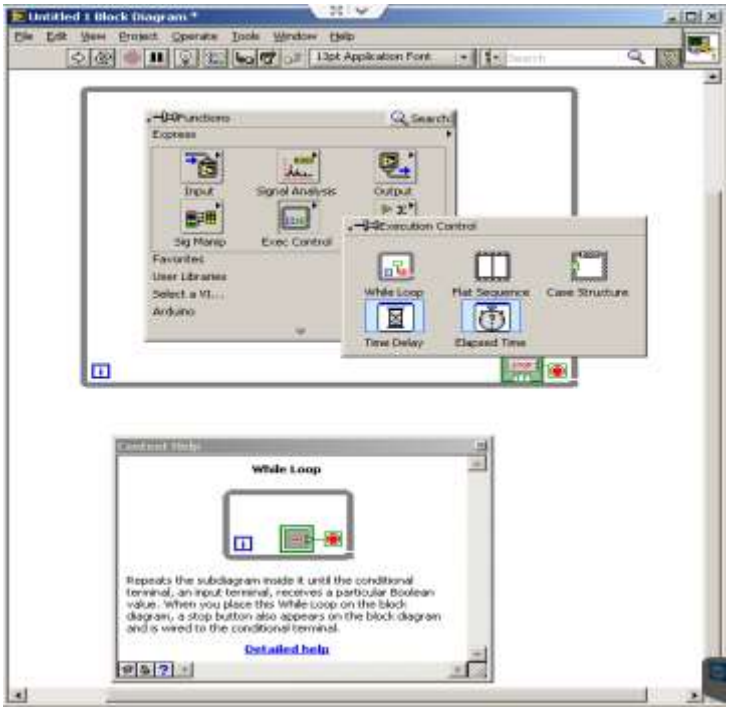


Figura 4.45: estructura de tipo While Loop en Labview

Fuente: Los Autores

4.10.3 Selección de eje

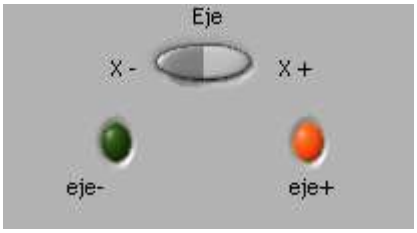


Figura 4.46: Interfaz gráfica de usuario selección de eje

Fuente: Los Autores

Inicializar y abrir la comunicación, agregando el VI Init, este bloque se configura:

Puerto de Comunicación →COM10

Velocidad de Transmisión →9600

Tipo de Tarjeta Arduino →Arduino Uno

Tipo de puerto de comunicación →USB/Serial

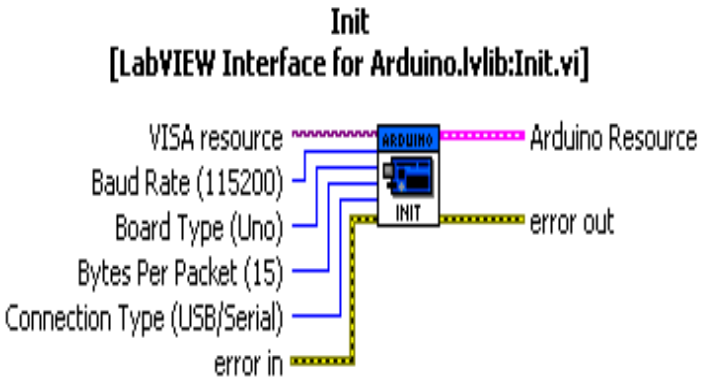


Figura 4.47: Comunicación gráfica "VI Init"

Fuente: Los Autores

Hacer doble clic en el bloque y configurar los parámetros antes dichos.

4.10.4 Envió de una señal digital a través del Labview hacia el Arduino DAQ/selección de eje.

Agregamos un bloque VI Set Digital Pin Mode y configuramos lo siguiente:

Número de Pin →5

Entrada o salida Digital →OutPut

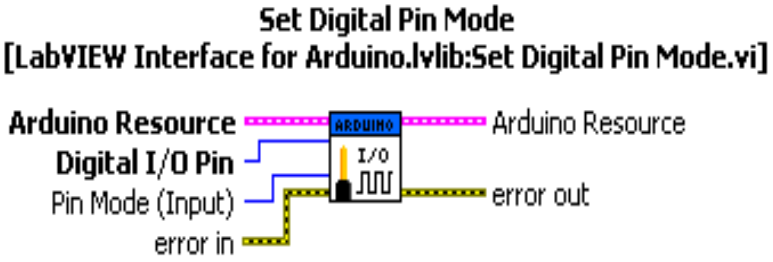


Figura 4.48 Comunicación gráfica "VI Set Digital Pin Mode"

Fuente: Los Autores

Agregar el bloque VI Digital Write Pin y configurar lo siguiente:

Número de Pin →5

Dato a ingresar →eje

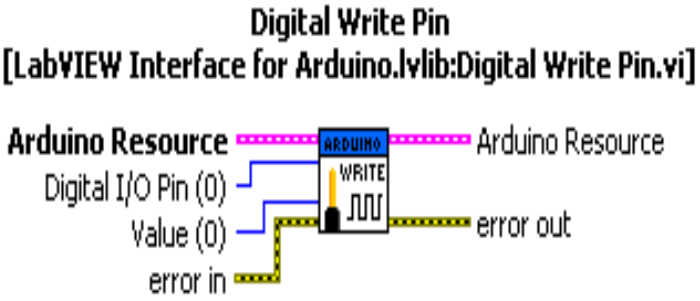


Figura 4.49: comunicación gráfica "VI Digital Write Pin"

Fuente: Los Autores

Y finalmente las conexiones quedarían de la siguiente manera:

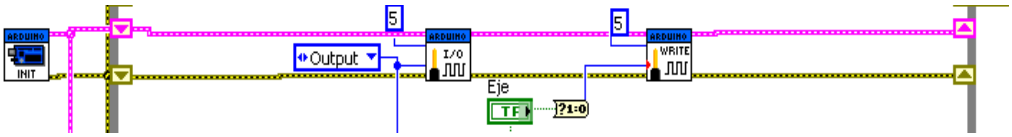


Figura 4.50: Diagrama de bloques

Fuente: Los Autores

En este proyecto en los pines 12 y 13 se conectarán los LED que represente los ejes en nuestro proyecto:

Led eje + →pin 12

Led eje - →pin 13

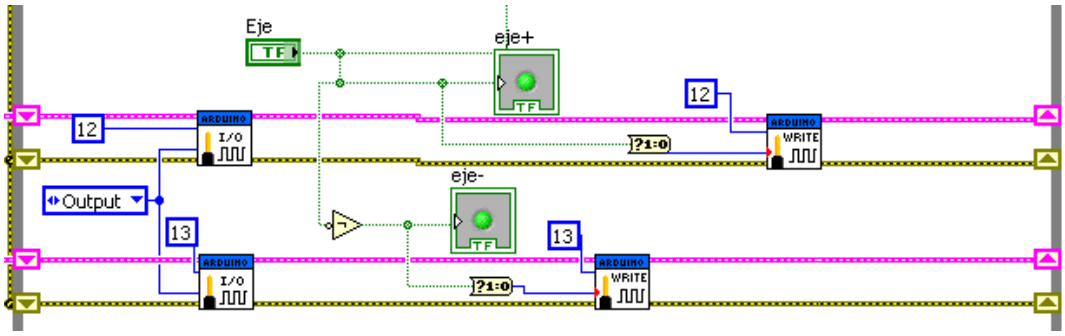


Figura 4.51: conexión de los pines 12 y 13 en el diagrama de bloques

Fuente: Los Autores

4.10.5 Switch ON/OFF

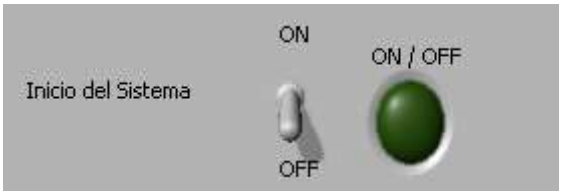


Figura: 4.52: Switch" ON/OF"

Fuente: Los Autores

4.10.6 Envió de una señal digital a través del Labview hacia Arduino DAQ/switch ON/OFF

Agregar un bloque VI Set Digital Pin Mode y configurar lo siguiente:

Numero de pin →**6**

Entrada o salida digital →**OutPut**

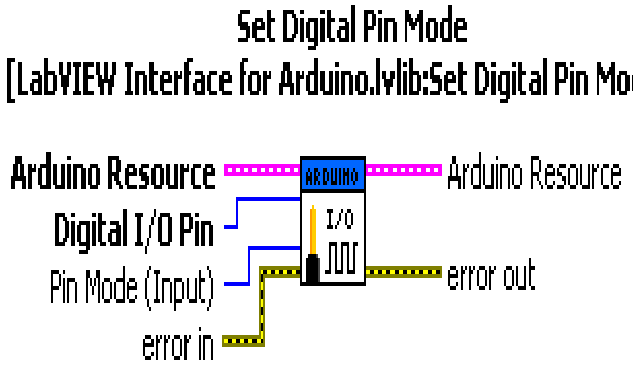


Figura 4.53: Comunicación gráfica "VI Set Digital Pin Mode"

Fuente: Los Autores

Agregar el bloque VI Digital Write Pin y configuramos lo siguiente:

Numero de Pin →**6**

Dato a ingresar →**Inicio del Sistema**

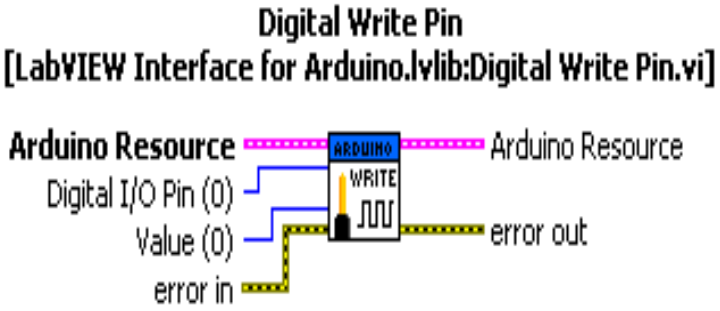


Figura 4.54: Comunicación gráfica "VI Digital Write Pin"

Fuente: Los Autores

Finalmente quedará:

Led ON/OFF → pin 6

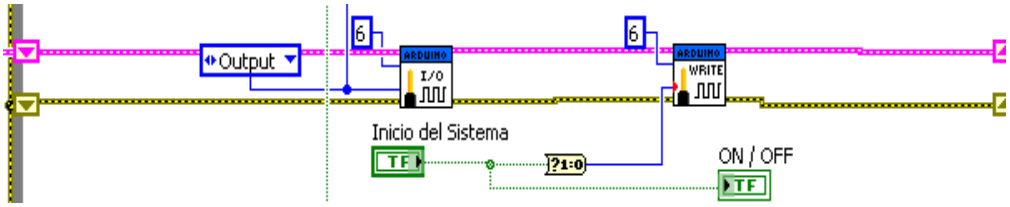


Figura 4.55: Diagrama de bloques

Fuente: Los Autores

4.10.7 Enviar un número de 8 bits desde Labview hacia el Arduino DAQ.

En esta etapa de la programación en Labview necesitaremos el bloque VI Index Array.

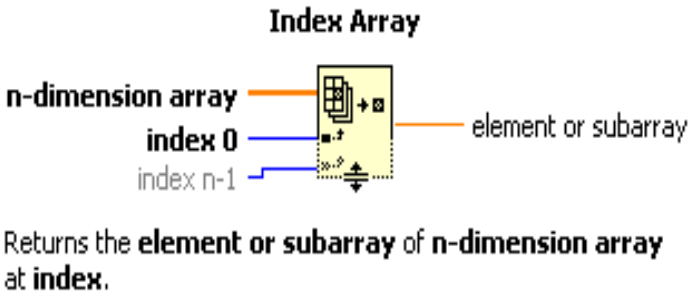


Figura 4.56: Comunicación gráfica "VI Index Array"

Fuente: Los Autores

La funcionalidad de este VI es convertir un número decimal en binario. Es decir 8 bits del menos significativo al más significativo con ocho salidas digitales que se configuran en el Arduino DAQ. Como se muestra en la siguiente imagen:

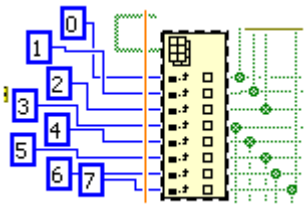


Figura 4.57: Bloque de conversión de un número decimal en binario

Fuente: Los Autores

En el pin n-dimension array, se ingresa el número decimal y en los pines element or subarray es la salida digital después de la conversión del número decimal.

Ejemplo: 90 = 00101101

Figura 4.58: Leds muestran la conversión decimal en binario

Fuente: Los Autores

Los nombres de los LED representan las salidas digitales del index array

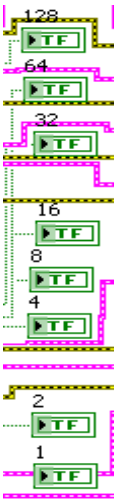


Figura 4.59: Bloque de conversión de un número decimal en binario

Fuente: Los Autores

Ahora declararemos las salidas digitales en nuestro Arduino DAQ con su respectivo número de Pines. Nos quedaría de la siguiente manera:

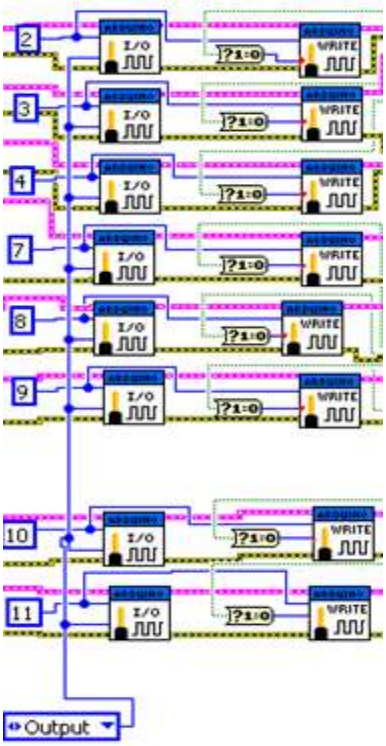


Figura 4.60: Bloque de conversión de un número decimal en binario

Fuente: Los Autores

Contando con las salidas digitales proceder a conectar a los bloques Digital Write Pin y la conexión final quedará de la siguiente manera:

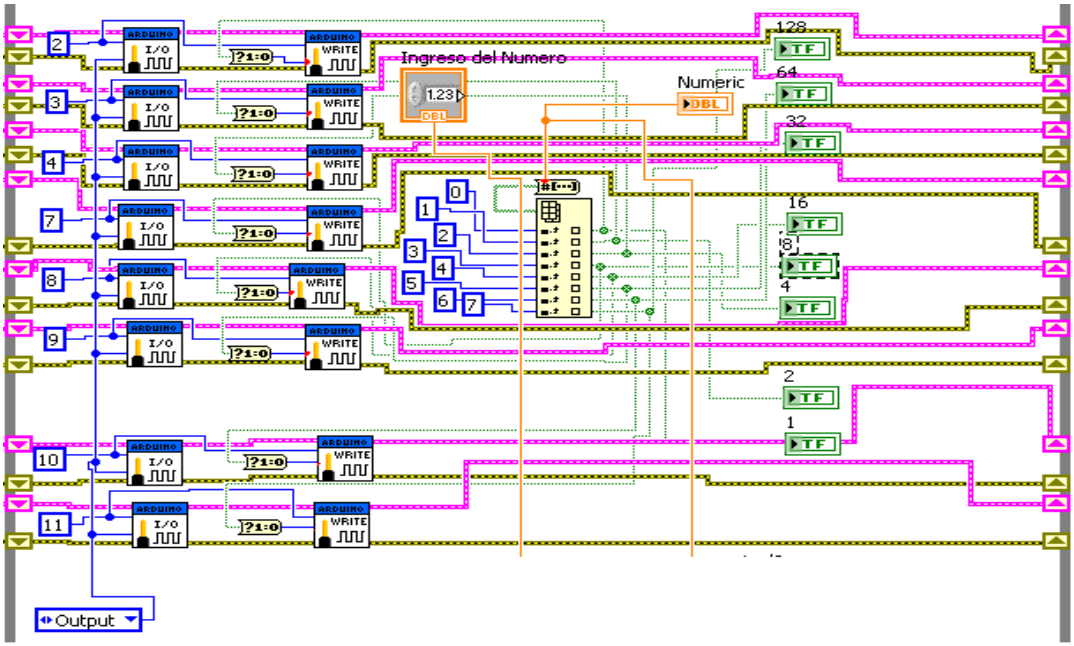


Figura 4.61: Conexión de bloques

Fuente: Los Autores

4.10.8 Visor de voltaje real del sensor de posición

En este visor se obtendrá el voltaje real del Sensor, dicho voltaje es con respecto a la posición del servomotor. La relación entre el servomotor y el sensor es la siguiente:

0° - 2,99 V ---- 180° - 127 V

Figura 4.62: Visor de voltaje real del sensor

Fuente: Los Autores

4.10.9 Lectura de señal analógica del sensor de posición en Labview

Agregar un bloque VI Analog Read Pin y configurar lo siguiente:

Numero de pin en la entrada analógica, en este caso es entrada A0.

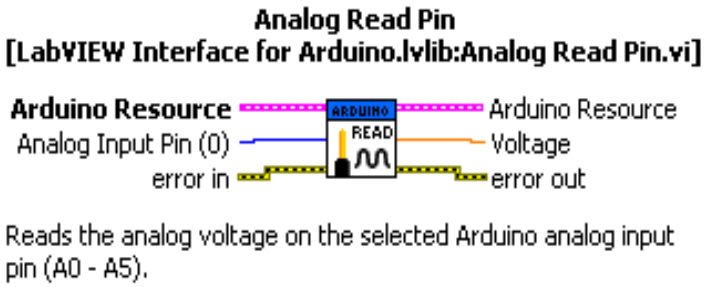


Figura 4.63: Comunicación gráfica "VI Analog Read Pin"

Fuente: Los Autores

Y nos quedará de la siguiente manera:

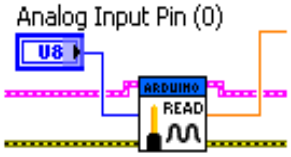


Figura 4.64: Comunicación gráfica "VI Analog Read Pin"

Fuente: Los Autores

En el pin de voltaje, se debe conectar el visor de voltaje.

4.10.9.1 Visor de la posición del sensor de 0 a 180 grados

En este visor se muestra la posición con respecto al servomotor. Esta relación es la siguiente.

Voltaje del sensor	Posición del sensor
2,99 V	0
1,27 V	180

Tabla 4.1: Relación entre voltaje del sensor y posición del sensor

Fuente: Los Autores

Se obtendrá la siguiente gráfica:

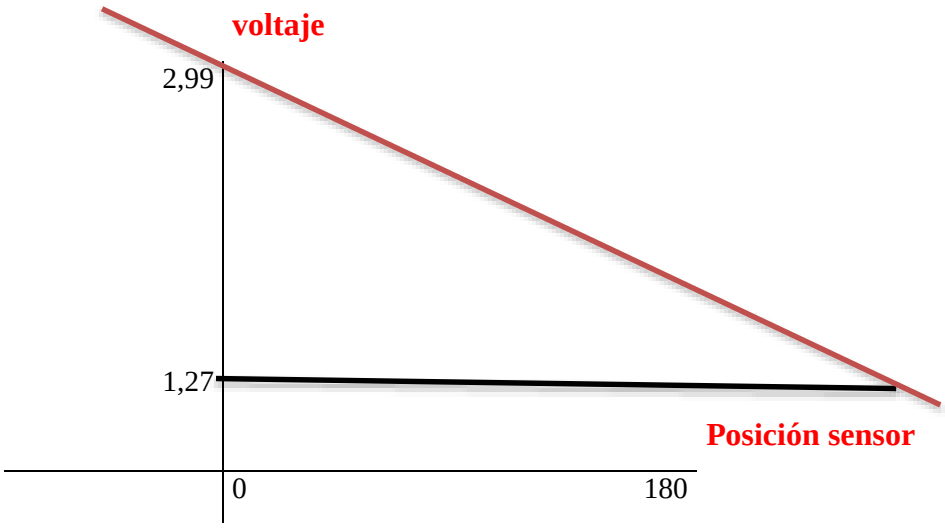


Figura 4.65: Gráfica de relación entre voltaje y sensor

Fuente: Los Autores

$Y = mx + b \rightarrow m = 0{,}00955$

Esta relación esta agregada en Labview y es la siguiente:

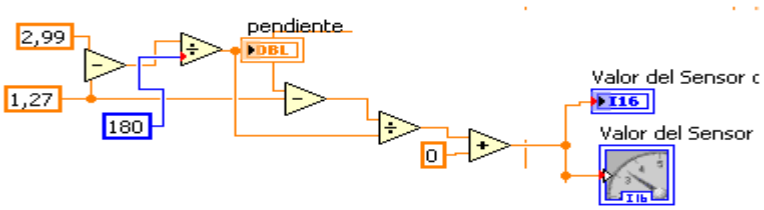


Figura 4.66: Relación entre voltaje y sensor en Labview

Fuente: Los Autores

Figura 4.67: Visor del sensor con respecto al servo

Fuente: Los Autores

4.10.10 Visor del ángulo obtenido

En este visor Indicará el ángulo obtenido, previamente ingresado la posición a obtener, la posición actual del sensor, la diferencia entre la posición del sensor menos la posición a obtener y ese resultado es el error (Sistema de control de lazo cerrado) y dependiendo de ese resultado es la orden que se le envía al Servomotor para que se mueva o se detenga. Todo este proceso lo realiza el Arduino Controlador**.** Labview es solo simplemente un visor de procesos.

Figura 4.68: Visor del ángulo obtenido

Fuente: Los Autores

Y la programación para este visor es la siguiente:

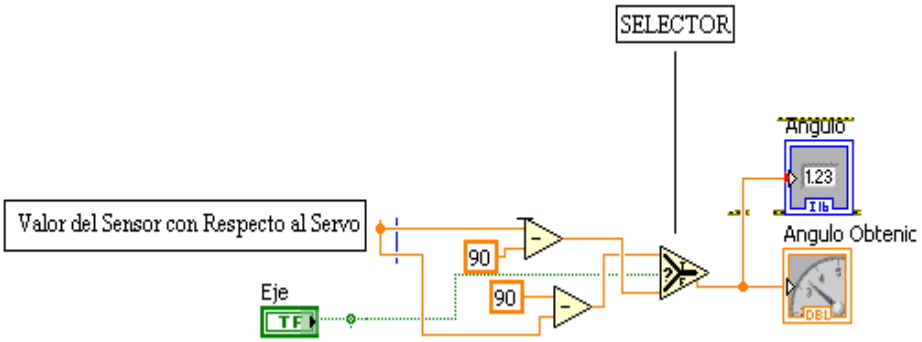


Figura 4.69: Programación del ángulo obtenido

Fuente: Los Autores

4.10.11 Visor del error

Como se ha explicado en la programación de la tarjeta Arduino, Es la diferencia entre valor ingresado y el valor del sensor.

A continuación el siguiente visor:

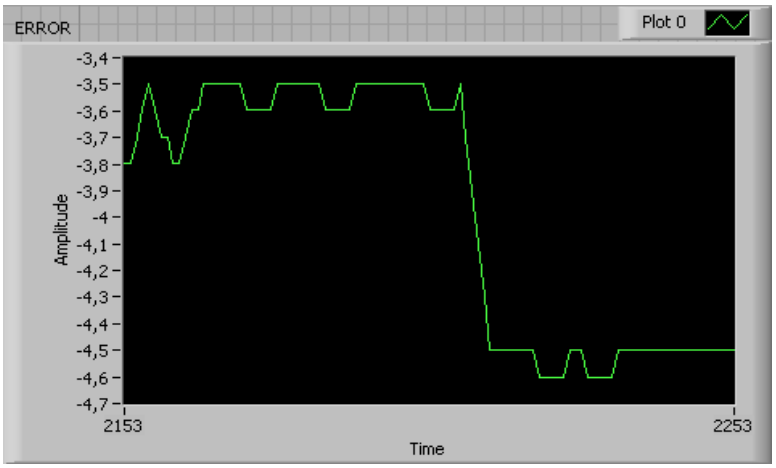


Figura 4.70: Gráfica del error en Labview

Fuente: Los Autores

La programación para este visor es la siguiente:

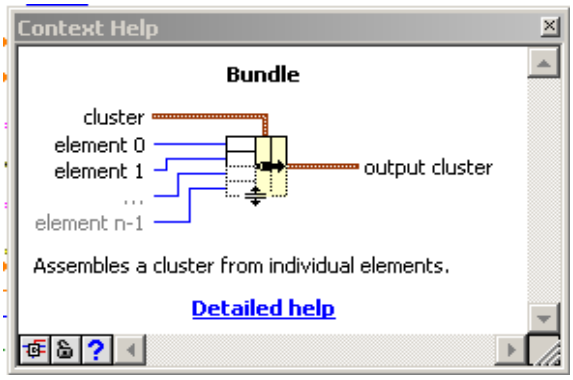


Figura 4.71: Visor de programación del error Labview

Fuente: Los Autores

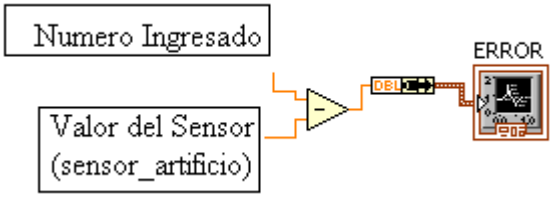


Figura 4.72: Visor de programación del error Labview

Fuente: Los Autores

CAPÍTULO V

ANÁLISIS Y CÁLCULOS MATEMÁTICOS DE LA IMPLEMENTACÍON

5.1 MOMENTO DE INERCIA

El momento de inercia de un cuerpo es la medida de la resistencia que presenta ante un cambio de su movimiento de rotación, y depende de la distribución de su masa respecto al eje de rotación. Para calcular el momento de inercia I se divide el cuerpo en una cantidad infinitesimal de elementos Δm, a distancia del eje de rotación y se realiza la sumatoria sobre todos los elementos de masa:

$I = \Sigma \Delta m * r^2$

Para cuerpos con distribución de masa continua y homogénea, la sumatoria se puede expresar como una integral de todos los Δm.

$I = M. (1/V) \int r^2 dv$

Donde *M* es la masa total, V el volumen del objeto y *r* la distancia del elemento de volumen *dv* al eje de rotación. El cálculo de la integral se simplifica si se toman cuerpos con simetría de rotación rotando entorno de su eje de simetría, como por ejemplo el eje transversal de nuestro proyecto que es un cilindro sólido y macizo de masa M e igual radio R por lo que la ecuación se transforma en:

$I = ½ MR^2$

$V = \pi R^2 . H$ y $dv = 2\pi r H dr$

5.2 TORQUE

En el instante que se le aplica fuerza a un objeto rígido, éste tiende a realizar un movimiento giratorio en torno a su eje de rotación. Esta propiedad de la fuerza de hacer girar a un objeto se le conoce como torque. El torque se define como el producto vectorial de una fuerza tangencial **F** que actúa sobre un cuerpo, y la un posición **r** respecto al origen de rotación de cuerpo rígido.

$$\tau = r \times F$$

También es posible relacionar el torque con el trabajo mecánico efectuado por una fuerza aplicada sobre un objeto durante cierto desplazamiento. Se define como la integral del producto escalar del vector fuerza por el vector desplazamiento.

$$W = \int F\, dx$$

Por otro lado, la potencia (en watts) se puede definir como la rapidez con el que se realiza el trabajo, como se aprecia en la ecuación.

$$P = \frac{dW}{dt} = F\frac{dx}{dt}$$

En el caso particular de los motores eléctricos, de cualquier tipo, estos corresponden a máquinas giratorias por lo que la ecuación se puede expresar en coordenadas polares:

$$W = \int F\, r d\theta$$

Ahora bien, como anteriormente se mencionó el torque corresponde al producto vectorial de fuerza tangencial y la distancia r desde el origen. La magnitud de torque está dado por:

$$|\tau| = rFsen(\theta)$$

Donde θ corresponde al ángulo entre vector fuerza y vector posición. Por lo tanto para obtener un torque máximo el sen (θ)=1, con esto la ecuación se convierte en

$$|\tau|_{máx} = rF$$

Ahora en la ecuación anterior se puede sustituir la fuerza por la relación del torque máximo, para obtener:

$$W = \int \tau d\theta$$

De la expresión de la ecuación anterior se puede deducir que $dW = \tau d\theta$, y con esto se puede obtener una ecuación que relaciona al torque con la potencia aplicada y la velocidad angular del servo motor, de la siguiente forma:

$$P = \frac{dW}{dt} = \tau \frac{d\theta}{dt} = \tau\omega$$

Donde ω es la velocidad angular en radianes por segundo. Por consiguiente, el torque puede calcularse partiendo de la potencia mecánica si además se conoce la velocidad del servo motor.

El torque además se relaciona con el momento de inercia. Dado que el torque es Fuerza tangencial multiplicada por el radio desde el origen de rotación de un objeto rígido, el torque se puede expresar como

$$|\tau| = Fr = m.ar$$

Donde "m" corresponde a la masa del objeto y "a" es la aceleración tangencial del cuerpo rígido. Como la aceleración tangencial está relacionada con la aceleración lineal a través de la relación:

$$a = r\alpha$$

Entonces, el torque puede expresarse como:

$$\tau = m.r^2\alpha$$

Y si se recuerda de la ecuación que mr2 es el momento de inercia de una partícula sobre su eje de rotación hasta el origen. Entonces se tiene una expresión para el torque que es proporcional a la aceleración angular:

$$|\tau| = J\alpha$$

El servomotor en las especificaciones tiene un valor determinado con respecto al torque en el cual realizamos un cálculo matemático para saber cuánto torque se necesita para mover nuestro eje transversal.

5.3 INERCIA DEL EJE TRANSVERSAL

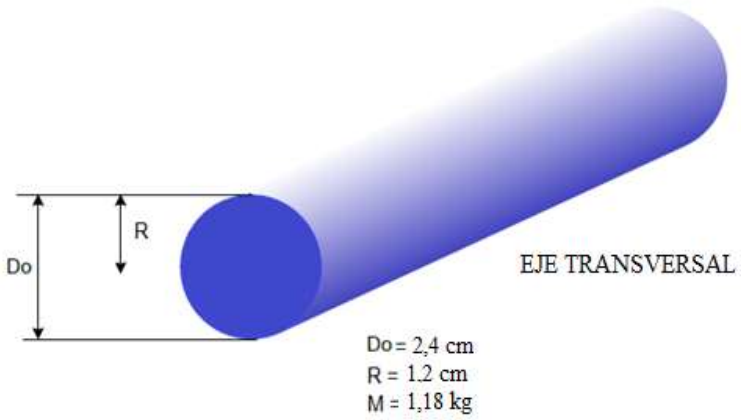


Figura 5.1: Medidas del eje transversal

Fuente: Los Autores

Como el eje transversal es de forma cilíndrica y maciza tiene una masa 0,83 kg e incluir las masas de los alerones es de alerón 1 = 0.06 kg y alerón 2 = 0.02 kg para calcular la inercia del cuerpo del total.

Masa total = 0,91 kg

La fórmula de la inercia es $I = ½ MR^2$

$I = ½ (0.91 kg) (1,2 cm)^2$

I = 0,6552

5.3.1 Velocidad y aceleración angular

w = velocidad angular

α = aceleración angular

En las especificaciones del servomotor tenemos que la velocidad de operación es 0.19seg / 60^0

Primero realizamos la conversión de la 60^0 a radianes

60^0_X (1 radian /$57,29^0$) = 1,047 radianes

w = 1,047/0, 19 = 5, 51 rad/s

$w = 2 \pi / t$

$t = 2 \pi / w$

$t = 2(3.1416)/5,51$

$t = 1,14$ s

Ahora Calcular la aceleración angular y viene dada de la siguiente fórmula.

$\alpha = w/t^2$

$\alpha = 5,51/(1,14)^2$

$\alpha = 4,24$ rad/s^2

Ahora aplicar la fórmula de torque.

$\tau = I \alpha$

Donde el momento de inercia del eje transversal es multiplicado por la aceleración angular.

$\tau = I \alpha$

$\tau = (0,6552 kg.cm^2) (4,24 rad/s^2)$

Al aplicar la fórmula el torque o par de fuerza que se necesita para mover el eje es de:

$\tau = 2,77$ kg.cm

5.4 SISTEMA DE CONTROL AUTOMÁTICO

Un sistema automático de control es un conjunto de componentes físicos conectados o relacionados entre sí, de manera que regulen o dirijan su actuación por sí mismos, es decir sin intervención de agentes exteriores (incluido el factor humano), corrigiendo además los posibles errores que se presenten en su funcionamiento.

Actualmente, cualquier mecanismo, sistema o planta industrial presenta una parte actuadora, que corresponde al sistema físico que realiza la acción, y otra parte de mando o control, que genera las órdenes necesarias para que esa acción se lleve o no a cabo.

5.4.1 Necesidad y aplicaciones de los sistemas automáticos de control

En la actualidad los sistemas automáticos juegan un gran papel en muchos campos, mejorando nuestra calidad de vida.

En los procesos industriales:

- Aumentando las cantidades y mejorando la calidad del producto, gracias a la producción en serie y a las cadenas de montaje.
- Reduciendo los costes de producción.
- Fabricando artículos que no se pueden obtener por otros medios.

En los hogares:

- Mejorando la calidad de vida. Podríamos citar desde una lavadora hasta un control inteligente de edificios (domótica).
- Para los avances científicos: Un claro ejemplo lo constituyen las misiones espaciales.
- Para los avances tecnológicos: por ejemplo en automoción es de todos conocidos los Limpiaparabrisas inteligentes, etc.

Como se puede observar las aplicaciones son innumerables. De esta manera surge toda una teoría, la regulación automática, dedicada al estudio de los sistemas automáticos de control.

5.4.2 Representación de los sistemas de control

5.4.2.1 Diagrama de bloques

Un proceso o sistema de control es un conjunto de elementos interrelacionados capaces de realizar una operación dada o de satisfacer una función deseada.

Los sistemas de control se pueden representar en forma de diagramas de bloques, en los que se ofrece una expresión visual y simplificada de las relaciones entre la entrada y la salida de un sistema físico.

A cada componente del sistema de control se le denomina elemento, y se representa por medio de un rectángulo.

El diagrama de bloques más sencillo es el bloque simple, que consta de una sola entrada y de una sola salida.

La interacción entre los bloques se representa por medio de flechas que indican el sentido de flujo de la información

En estos diagramas es posible realizar operaciones de adición y de sustracción, que se representan por un pequeño círculo en el que la salida es la suma algebraica de las entradas con sus signos. También se pueden representar las operaciones matemáticas de multiplicación y división como se muestra en la siguiente figura 5.2:

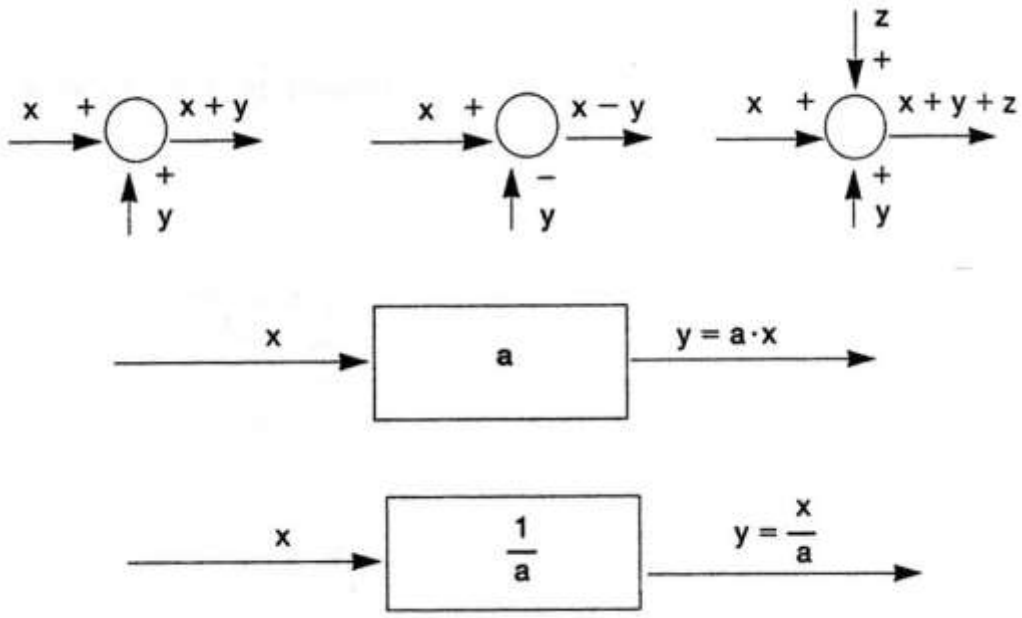


Figura 5.2: Diagramas de bloques

Fuente: Los Autores

5.4.2.2 Tipos de sistemas de control

Los sistemas de regulación se pueden clasificar en:

Sistemas de bucle o lazo abierto: son aquellos en los que la acción de control es independiente de la salida.

Sistemas de bucle o lazo cerrado: son aquellos en los que la acción de control depende en cierto modo, de la salida.

5.4.2.3 Sistemas de control en lazo abierto

Un sistema de control en lazo o bucle abierto es aquél en el que la señal de salida no influye sobre la señal de entrada. La exactitud de estos sistemas depende de su calibración, de manera que al calibrar se establece una relación entre la entrada y la salida con el fin de obtener del sistema la exactitud deseada.

El diagrama de bloque de un sistema en lazo abierto es:

Figura 5.3: Diagrama de bloques de un sistema de lazo abierto

Fuente: Los Autores

5.4.2.4 Sistemas de control en lazo cerrado

Si en un sistema en lazo abierto existen perturbaciones, no se obtiene siempre la variable de salida deseada. Conviene, por tanto, utilizar un sistema en el que haya una relación entre la salida y la entrada.

Un sistema de control de lazo cerrado es aquél en el que la acción de control es, en cierto modo, dependiente de la salida. La señal de salida influye en la entrada. Para esto es necesaria que la entrada sea modificada en cada instante en función de la salida. Esto se consigue por medio de lo que llamamos realimentación o retroalimentación (feedback).

La realimentación es la propiedad de un sistema en lazo cerrado por la cual la salida o cualquier otra variable del sistema que esté controlada, se compara con la entrada del sistema (o una de sus entradas), de manera que la acción de control se establezca como una función de ambas.

A veces también se le llama a la realimentación transductor de la señal de salida, ya que mide en cada instante el valor de la señal de salida y proporciona un valor proporcional a dicha señal.

Por lo tanto podemos definir también los sistemas de control en lazo cerrado como aquellos sistemas en los que existe una realimentación de la señal de salida, de manera que ésta ejerce un efecto sobre la acción de control.

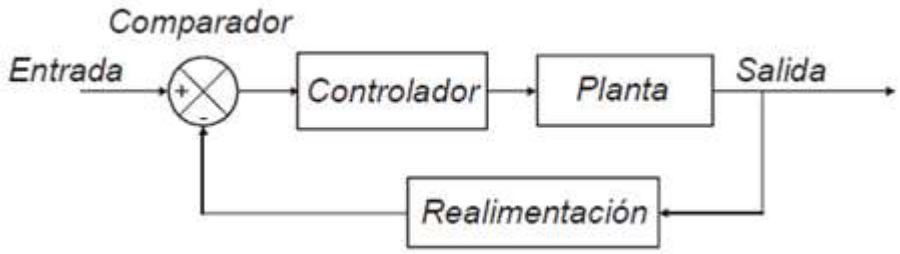


Figura 5.4: Diagramas de bloques de un sistema de lazo cerrado

Fuente: Los Autores

El controlador está formado por todos los elementos de control y a la planta también se le llama proceso.

En este esquema (figura 5.4) se observa cómo la salida es realimentada hacia la entrada. Ambas se comparan, y la diferencia que existe entre la entrada, que es la señal de referencia o consigna (señal de mando), y el valor de la salida (señal realimentada) se conoce como error o señal de error. La señal que entrega el controlador se llama señal de control o manipulada y la entregada por la salida, señal controlada.

El error, o diferencia entre los valores de la entrada y de la salida, actúa sobre los elementos de control en el sentido de reducirse a cero y llevar la salida a su valor correcto. Se intenta que el sistema siga siempre a la señal de consigna.
El diagrama de bloques anterior se puede sustituir por el siguiente:

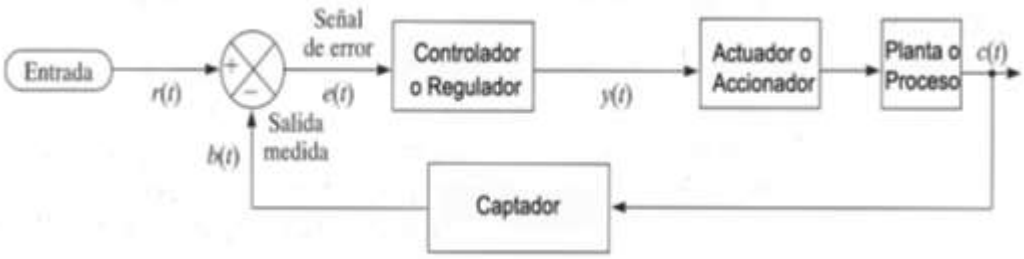


Figura 5.5: Diagramas de bloques de un sistema de lazo cerrado

Fuente: Los Autores

La salida del sistema de regulación se realimenta mediante un captador. En el comparador o detector de error, la señal de referencia (salida del transductor) se compara con la señal de salida medida por el captador, con lo que se genera la siguiente señal de error:
e(t) = r(t) – b(t), donde e(t) es la señal de error, r(t) la señal de referencia y b(t) la variable realimentada. Pueden suceder dos casos:
- Que la señal de error sea nula. En este caso la salida tendrá exactamente el valor previsto.
-Que la señal de error no sea nula. Esta señal de error actúa sobre el elemento regulador que a su salida proporciona una señal que, a través del elemento accionador, influye en la planta o proceso para que la salida alcance el valor previsto y de esta manera el valor se anule

5.4.2.5 Función de transferencia

Para determinar la respuesta de un elemento en función del tiempo, se aplican señales conocidas a la entrada del sistema o elemento y se evalúan las señales que

aparecen en la salida. La respuesta obtenida así se llama respuesta transitoria. Normalmente la señal de entrada es una señal de entrada en forma de escalón.

También se puede estudiar la respuesta matemáticamente mediante la función de transferencia o respuesta en frecuencia. Por medio de la función de transferencia se puede conocer:

-La respuesta del sistema frente a una entrada determinada.

-La estabilidad del sistema (si la respuesta del sistema se va a mantener dentro de unos límites determinados).

-Qué valores se pueden aplicar al sistema para que permanezca estable.

Se define función de transferencia G(s) de un sistema como el cociente entre las transformadas de Laplace de las señales de salida y entrada.

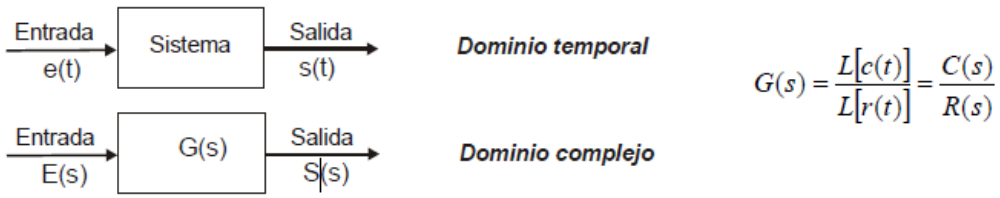


Figura 5.6: Diagramas de bloques

Fuente: Los Autores

Las características de la función de transferencia dependen únicamente de las propiedades físicas de los componentes del sistema, no de la señal de entrada aplicada.

La función de transferencia viene dada como el cociente de dos polinomios en la variable compleja s de Laplace, uno, N(s) (numerador) y otro D(s) (denominador).

$$G(s) = \frac{N(s)}{D(s)} = \frac{b_0.s^m + b_1.s^{m-1} + ... + b_m}{a_0.s^n + a_1.s^{n-1} + ... + a_n}$$

Una vez calculada la transformada de Laplace de la entrada, conocer de forma inmediata la transformada de Laplace de la salida. Calculando la trasformada inversa se obtiene la respuesta en el tiempo del sistema ante esa entrada determinada.

5.5 MODELADO MATEMÁTICO

5.5.1 Función de transferencia del servomotor

Figura 5.7: Servomotor HITEC HS-311

Fuente: http://www.servodatabase.com

El servomotor está formado por un motor dc, la reducción del engranaje y la realimentación, todo en la misma caja pequeña. Teniendo el servo un margen máximo de operación de 180^0aproximadamente.

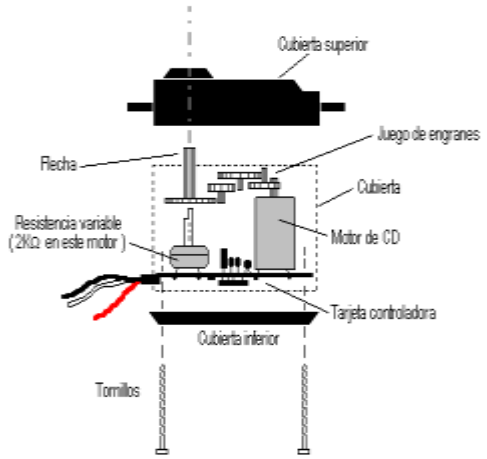


Figura 5.8: Servomotor partes internas

Fuente: Los Autores

Especificaciones del servomotor HITEC HS-311:

Sistema de control: pulsos PWM

Voltaje de operación: 4.8-6 v
Velocidad de operación: 0.19seg / 60 grados a 4.8 v
Torque: 3 kg-cm (42 oz-in) a 4.8 v y 3,7 kg-cm (51 oz-in) a 6 v
Tamaño: 40 x 20 x 37 mm
Peso: 43 g

La Función de transferencia de un servomotor está representada de la siguiente ecuación.

$$\frac{\Theta(s)}{Ea(s)} = \frac{k0\ k1\ n\ km\ /\ Tm\ .}{s2+1/Tm\ s\ +\ k0\ k1\ n\ km\ /Tm}$$

Donde
km = k2 / (Ra b0 + k2 k3)
Tm = Ra J/ (Ra b0 + k2 k3)

Se puede definir la constante

$$K1 = \frac{K2}{Ra}$$

Sabemos a su vez que la función de transferencia del servomotor incluyendo el torque del mismo tiene la siguiente forma.

$$\frac{Ra}{K2} Tm(s) + K3S\,\Theta(s) = Ea(s)$$

De la cual se puede llevar a cabo el despeje que relacione el torque con el voltaje de armadura del servomotor.

$$\frac{Tm(s)}{Ea(s)} = K1 - SK1K3\frac{\Theta(s)}{Ea(s)}$$

5.5.1.1 Ganancia del detector (k0)

Se divide el voltaje de referencia entre el ángulo de movimiento del potenciómetro.

Vref = 2.5 volts

$\theta = \pi$ rad

k0 = Vref /θ

k0 = 2.5/π

k0 = 0.79577 volts/rad

5.5.1.2 Tren de engranes (n)

El servomotor cuenta con un tren de engranes que le sirven para transmitir energía, adaptando su velocidad angular y el par mecánico.

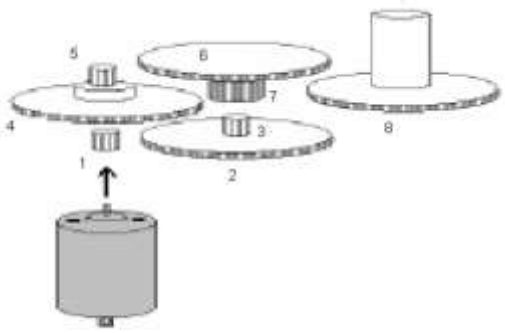


Figura 5.9: Tren de engranajes del servomotor

Fuente: Los Autores

Numero de engrane y diámetro (cm)

1. 0.32
2. 1.57
3. 0.38
4. 1.53
5. 0.51
6. 1.45
7. 0.75
8. 1.71

La relación de engranes n está determinada por n = N1 / N2 y para un tren de engranes está determinado por:

nT = n1 n2 n3 n4

n1 = N1 /N2 = 0.32/1.57

n2 = N3 /N4 = 0.38/1.53

n3 = N5 /N6 = 0.51/1.45

n4 = N7 /N8 = 0.75/1.71

nT = 1/128

5.5.1.3 Amplificador (k1)

El servomotor tiene dos circuitos integrados que se modelaron como una etapa amplificadora, por lo tanto definimos la ganancia de este amplificador con un valor de:

k1 = 1

Ra = 7,6 ohms

Figura 5.10: Motor DC

Fuente: Los Autores

5.5.1.4 Velocidad y aceleración angular

En las especificaciones del motor tiene una velocidad de S = 9590 rpm, ésta velocidad en un minuto de movimiento, se obtiene su velocidad a un t = 0.23s.

9590 ------ 60 s

X ------ 0.23 s

x = (0.23 *9590)/60 =36.76166 rpm

La velocidad angular se determina por

$\omega = (2\pi) *S/t$

S = x = 36.76166 rpm

$\omega = (2\pi) (36.76166)/ 0.23 = 1004.2622$ rad/s

y la aceleración angular como

$\alpha = (2\pi) *S / t2$

$\alpha = (2\pi) (36.76166) / 0.232 = 4366.35$ rad/s2

5.5.1.5 Constante par torsión (k2)

La constante de par torsión se define como:

τ = torque del motor

ia= corriente a máxima eficiencia

$k2 = \tau / ia$

k2 = 0.38 m / 0.18 = 2.1111111 m Nm/Amp

5.5.1.6 Coeficiente de fricción (b0)

τ = torque del motor

ω = velocidad angular del motor

$b0 = \tau / \omega$

b0 = 0.38 m / 1004.26 = 0.37838 μ Nms/rad

5.5.1.7 Momento de inercia (J)

Se determina por

$J = \tau / \alpha$

J = 0.38 m / 4366.35 = 0.087029 μ Kgm2

5.5.1.8 Constante fuerza-contraelectromotriz (k3)

Para determinar esta constante se requiere tener el valor de la fuerza Contraelectromotriz, ya que este valor no se puede medir físicamente se realiza una estimación de este valor con la siguiente formula.

ea= voltaje administrado al motor

R_a = resistencia de la armadura del motor

ia= corriente de armadura del motor

eb= ea - ia Ra

Tomando los valores de máxima eficiencia

eb= 4.5 – (0.184)(7.6) = 3.1016 volts

Por lo que la constante k3 es

$k3 = eb / \omega$

k3 = 3.1016 / 1004.26 = 3.08844 m Vrad/s

5.5.2 FUNCIÓN DE TRANSFERENCIA DEL ALERÓN

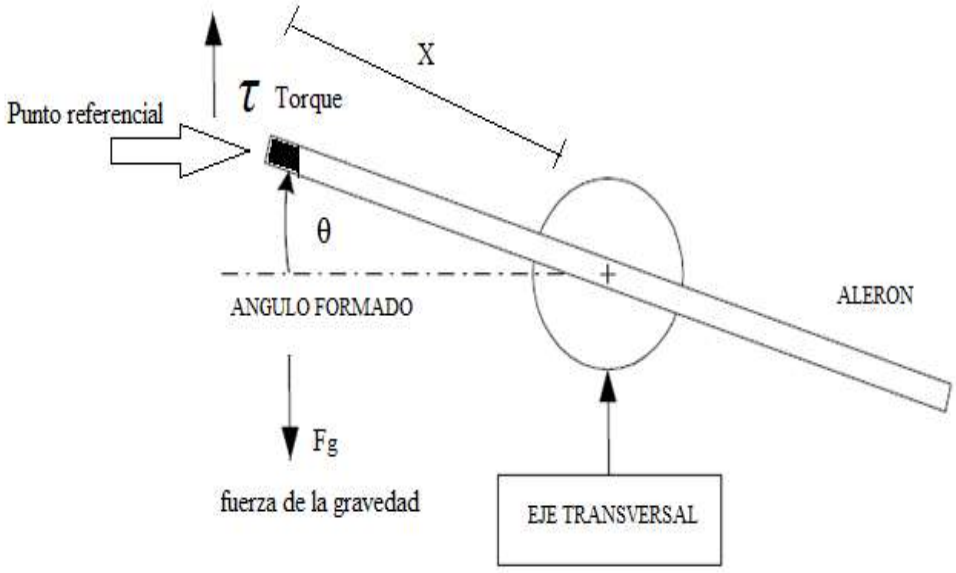


Figura 5.11: Diagrama frontal del eje transversal y alerón

Fuente: Los Autores

El movimiento a controlar en este proyecto es el de rotación de un sólido alrededor de su eje central de inercia. La variación del estado de rotación de un sólido viene determinada por la variación de su velocidad angular por lo que, si se quiere describir el movimiento de rotación se debe encontrar una ecuación que permita calcular la aceleración angular del mismo, para ello la siguiente ecuación:

$$\Sigma F = I\alpha$$

Para obtener la ecuación de movimiento que caracteriza a este proyecto se tendrá en cuenta que el eje de inercia de la barra móvil está ubicado justo en su centro de gravedad, razón por la cual la fuerza resultante sobre la barra será nula, al estar todas las fuerzas de un lado y del otro, de su eje de inercia, compensadas entre sí.

Una vez planteadas las consideraciones principales, estamos en condiciones de aplicar la ecuación.

F (servo) $X–XF_g sen\theta = I\alpha$

$T - XF_g sen\theta = I\alpha$

Donde;

T: torque aplicado al alerón

θ: posición de la barra respecto al eje de giro, se consideraran desplazamientos muy pequeños por lo que **sin θ≈ θ** en [**rad**].

X: longitud desde el centro de la barra a uno de sus extremos.

Fg: fuerza de gravedad que actúa sobre el alerón.

I: momento de inercia del sistema.

α: aceleración angular

$T - XF_g\theta = I\alpha$

$T - XF_g\theta = I\ d^2\ \theta/dt^2$

Aplicar las transformadas de Laplace para obtener la función de transferencia:

$L\ (T) – L\ (XF_g\theta) = I\ (L)\quad L\ d^2\ \theta/dt^2$

$T(s) – XF_g\theta(s) = I\ s^2\theta(s)$

$T(s) = I\ s^2\theta(s) + XF_g\theta(s)$

$T(s) = \theta(s)\ (I\ s^2 + XF_g)$

La función de transferencia con respecto al alerón.

$$\frac{\theta\,(S)}{T\,(S)} = \frac{1}{Is^2 + F_g\,X}$$

CRONOGRAMA

	1 Mes	2 Mes	3 Mes	4 Mes	5 Mes	6 Mes	7 Mes	8 Mes	9 Mes	10 Mes	11 Mes	12 Mes	13 Mes	14 Mes	15 Mes	16 Mes	17 Mes	18 Mes
Investigación y estudio del proyecto																		
Elaboración de costos y presupuestos																		
Análisis de información y selección de dispositivos electrónicos y mecánicos																		
Pruebas de placa Arduino (Hardware y Software)																		
Pruebas de Arduino + Dispositivos Electrónicos																		
Análisis, cálculos diseño del Sistema de Posición Horizontal																		
Construcción del diseño mecánico																		
Implementación, pruebas y corrección de errores surgidos de los cálculos																		
Integración de toda la parte mecánica eje transversal, servo y tarjetas Arduino																		
Instalación del Software de Arduino y Labview																		
Desarrollo del programa sistema control de posición horizontal																		
Acoplamiento de los dispositivos electrónicos con la programación del proyecto																		
Pruebas de funcionamiento en campo																		
Análisis y evaluación de resultados obtenidos																		
Elaboración del instructivo																		
Presentación de la implementación terminada.																		
Presentación de la documentación de tesis																		
Correcciones de la documentación de tesis																		

PRESUPUESTO

DESCRIPCION	CANTIDAD	VALOR
LAPTOP HP PAVILON 2GB RAM, 250 GB DISCO DURO	1	$ 900
TARJETAS ARDUINO	4	$ 80
SERVOMOTOR	6	$120
SENSOR DE POSICÍON MTS 360	4	$240
MULTÍMETRO	1	$50
COMPONENTES ELECTRÓNICO VARIOS		$100
CONTRUCCÍON MESA DE SOPORTE Y EJE TRANSVERSAL	1	$450
RECURSOS DE OFICINA		$150
TOTAL		**$2090**

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES:

- La tarjeta controladora “Arduino uno” aportó conocimientos y experiencia para la implementación de la tesis donde se realizaron pruebas de adquisición de datos con respecto a Labview con Arduino.

- En la práctica se comprobó que el sensor MTS 360, cumplió las condiciones del diseño y por ende se realizaron pruebas en la programación del software Arduino.

- En el principio para familiarizarse con el entorno de programación, se hizo una gran variedad de códigos de ejemplos con el servomotor y sensor realizando pequeños programas para comprobar que independientemente unas de otras, funcionaban. Posteriormente se juntó todas las tareas y se adaptó a un mismo programa.

- La función de la tarjeta Arduino no es solo controladora, investigamos que podíamos convertirla en una tarjeta de adquisición de datos (DAQ), donde fue de mucho provecho y ahorro de dinero. En la tesis se utilizó dos tarjetas Arduino uno como controlador y otra como DAQ.

- El uso de Labview para la implementación fué solo de visor y conversión de datos binarios a decimales.

- La realización de esta tesis es para promover el uso de las tarjetas Arduino y el mundo de microcontroladores, ya que son piezas muy asequibles y con gran versatilidad, con las que se pueden desarrollar infinitos proyectos sea en el campo robótico como en el industrial.

RECOMENDACIONES:

- Es muy importante la conexión de la tarjeta Arduino y el voltaje recomendado de los dispositivos, asegurando la fiabilidad y el uso de los mismos.

- Tener en cuenta al momento de la instalación la versión del IDE de Arduino y kits de drivers VISA al convertirla en una tarjeta de adquisición de datos.

- Para la conexión del sensor MTS 360 se recomienda leer las instrucciones ya que funciona bajo la tecnología patentada del efecto hall.

- Se recomienda utilizar un servomotor de marca reconocida en el mercado a fin de no tener problemas con respecto al peso y torque a utilizar.

BIBLIOGRAFÍA
DIRECCIONES ELECTRÓNICAS

http://www.arduino.cc/es/

http://arduino.cc/es/Main/Software

http://www.monografias.com/trabajos60/servo-motores/servo-motores.shtml

http://www.servodatabase.com/servo/hitec/hs-311

http://www.piher.net

http:// directindustry.es

http://www.ni.com/labview/esa/

http://sine.ni.com/nips/cds/view/p/lang/es/nid/209835

http://www.hobbytronics.co.uk/arduino-tutorial2-servos

http://trollmaker.com/article3/arduino-and-hc-sr04-ultrasonic-sensor

http://fisicajm.es.tl/libros-de-fisica-mecanica.htm

http://www.sapiensman.com/control_automatico/control_automatico6.htm

http://www.atmel.com/Images/8161s

ANEXO 1

CÓDIGO DE PROGRAMACÍON

```
#include <Servo.h> //invocación de librería para Servomotor
Servo myservo;     //creación de variable para servomotor
int sensor0=0, sensor180=0;   // variable para determinar función línea del sensor
int pin2=2, pin3=3, pin4=4, pin7=7, pin8=8, pin10=10, pin11=11, pin12=12;
                                        //   Creación de variables de tipo entero
int entra2=0, entra3=0, entra4=0, entra7=0, entra8=0, entra10=0, entra11=0,
entra12=0;
int sumatotal=0;
int suma1=0, suma2=0, suma3=0, suma4=0, suma5=0, suma6=0, suma7=0,
suma8=0;
int Valsensor, i=0, Total=0, Promedio=0;   //filtro sensor
int Lectura[1];   // filtro sensor
int servo; //posición de inicio del servo
int pin5=5; // entrada del binario desde Labview EJE
int pin6=6; // entrada del binario desde Labview SWITCH
int eje ;       // eje + , - Labview
int switch1 ;   // ON-OFF Labview
int     posfinal=0,error=0,entrada=0,sensor=0,sensor_artificio=0,posicion=0,error2=0,
sensor_artificio2=0;
int cont=0, suma=0, sumato=0, cont2=0, cont3=0, cont4=0;
void setup()
{
myservo.attach(9);  // indica que la entrada PWM del pin 9, es del SERVO
Serial.begin(9600);  // abrir la comunicación del puerto serie
for (i=0; i<1; i++)   //filtro sensor
Lectura[i]=0;          //filtro sensor
i=0;                       //filtro sensor
```

```
}
void loop()
{
servo=myservo.read();      // La lectura de la posición del servomotor se guarda en la variable servo
eje=digitalRead(pin5);    // La entrada digital del pin 5 se guarda en la variable eje - selección del eje + o -
switch1=digitalRead(pin6); //la entrada digital del pin 6 se guarda en la variable switch1-  Inicio del Sistema ON / OF
Serial.println("Sensor Sin Filtro");
delay(100);
Serial.println(analogRead(A1));
delay(1000);
if (switch1==0)
{
myservo.write(90);  // Posicionamos al servo en 90°
delay(100);     // retardo de 100 mili segundos
}
Total = Total - Lectura[i];    //filtro sensor
Lectura[i]=analogRead(A1);     // en la variable Lectura guarda la señal del sensor
Total = Total + Lectura[i];   //filtro sensor
i=i+1;                         //filtro sensor
if (i>=1)                      //filtro  sensor
{                              //filtro sensor
i=0;                           //filtro sensor
Promedio = Total/1;            //filtro sensor
Valsensor = map (Promedio, 236, 578, 0,180);  // la función map es una ecuación lineal que toma el máximo y mínimo
 // del sensor y lo presenta entre 0 a 180.
}
Serial.println("Sensor Valor Real con Filtro");    // Imprime mensaje
delay(100);         // retardo de 100 mili segundos
Serial.println(Valsensor);   // Imprime el Valor del Sensor con el filtro.
delay(100);    // retardo de 100 mili segundos
```

```
entra2=digitalRead(pin2);    //lectura de un bits menos significativo enviado desde Labview (8 bits)
entra3=digitalRead(pin3);
entra4=digitalRead(pin4);
entra7=digitalRead(pin7);
entra8=digitalRead(pin8);
entra10=digitalRead(pin10);
entra11=digitalRead(pin11);
entra12=digitalRead(pin12); ////lectura de un bits más significativo enviado desde Labview (8 bits)
// Conversión de un numero Binario a Decimal
if (entra2==1)
{
suma1=1;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
}
Else
{
 suma1=0;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
}
if (entra3==1)
{
suma2=2;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
 }
else
{
suma2=0;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
}
if (entra4==1)
{
suma3=4;
```

```
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
}
else
{
suma3=0;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
}
if (entra7==1)
{
suma4=8;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
}
Else
{
suma4=0;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
}
if (entra8==1)
{
suma5=16;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
}
else
{
suma5=0;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
}
 if (entra10==1)
{
suma6=32;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
}
else
{
```

```
suma6=0;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
}
if (entra11==1)
{
suma7=64;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
}
Else
{
suma7=0;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
}
if (entra12==1)
{
suma8=128;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
}
Else
{
suma8=0;
sumatotal= suma1+suma2+suma3+suma4+suma5+suma6+suma7+suma8;
 } // Fin de la Conversión de Binario a Decimal
 if (eje ==1) // Pregunta posición del Eje + (Labview)
{
if (switch1==1)  // Pregunta posición del ON/OFF (Labview)
{
sensor=Valsensor;
sensor_artificio=90-sensor;
Serial.println("Sensor con Artificio");
delay(100);
Serial.println(sensor_artificio);
delay(100);
posición=sensor_artificio;
```

```
suma = 0;
suma = sumatotal + 90;
sumato=suma -90;
entrada=sumato;
error=entrada-sensor_artificio;
error2=abs(error);
Serial.println("ERROR");
delay(100);
Serial.println(error2);
delay(100);
if (error > 0)
{
if (posición<=entrada)
{
for (posición=sensor_artificio; posición<=entrada; posición+=1)
{
myservo.write(posicion+90+0); // posiciona al servo dependiendo del valor ingresado
delay(100);
Serial.println("POSICIÓN");
delay(10);
Serial.println(posición+90);
delay(10);
Serial.println("ERROR");
Serial.println(error2);
 }
 }
 }
if (error < 0)
{
if (posición>=entrada)
{
delay(100);
for (posición=sensor_artificio; posición>=entrada; posición-=1)
```

```
{
myservo.write(posicion+90-5);     // posiciona al servo dependiendo del valor ingresado
delay(100);
Serial.println("POSICIÓN");
delay(10);
Serial.println(posicion+90-5);
delay(10);
Serial.println("ERROR");
Serial.println(error2);
}
}
}
}
else if (switch1==0)
{
suma=0;
myservo.write(90);
delay(10);
delay(100);
int pos=0;
if (pos>=90)
{
for(pos = sensor_artificio; pos<=90; pos += 1)
{
 myservo.write(pos);
delay(100);
}
}
}
}

else if (eje==0)  // Pregunta posición del Eje - (Labview)
{
```

```
 if (switch1==1)
{
suma=0;
sensor=Valsensor;
sensor_artificio=sensor-90;
sensor_artificio2=abs(sensor_artificio);
posición=sensor_artificio;
suma=0;
suma=90-sumatotal;    // forma el angulo según el valor que ingresa con respecto al eje –X
sumato=90-suma;
entrada=sumato;
error=entrada-sensor_artificio;
error2=abs(error);
Serial.println("ERROR");
delay(100);
Serial.println(error2);
delay(100);
if(error>0)
{
if(posición<=entrada)
{
for (posición=sensor_artificio; posición<=entrada; posición+=1)
{
 myservo.write(90-posicion-3);              // posiciona al servo dependiendo del valor ingresado
delay(100);
Serial.println("POSICIÓN");
delay(10);
Serial.println(90-posicion-3);
delay(10);
Serial.println("ERROR");
Serial.println(error2);
}
```

```
}
}
if (error < 0)
{
if(posición>=entrada)
{
for (posición=sensor_artificio; posición>=entrada; posición-=1)
{
myservo.write(90-posicion+6); // posiciona al servo dependiendo del valor ingresado
delay(100);
Serial.println("POSICIÓN ");
delay(10);
Serial.println(90-posición+6);
delay(10);
Serial.println("ERROR");
Serial.println(error2)
}
}
}
else if (switch1==0)
{
cont2=0;
suma=0;
myservo.write(90+5);
delay(100);
}
}
}
```

ANEXO 2

DICCIONARIO DE DATOS

Variables:

ValSensor ➔ Valor Real del Sensor (0 – 5 V), el cual cuando se conecta a las entradas analógica del Arduino (A0…A5), este lo convierte a un número de 10 bits. Dicha funcionalidad es propia de la Tarjeta Arduino. Y la relación es la siguiente:

Voltaje real	Numero de 10 bits
3,09 V	236
1,33 V	578

Después con la función map **(**ecuación de la recta**),** se aplica la siguiente relación:

Numero de 10 bits	Posición del Sensor
236	0
578	180

Sensor ➔ Sensor es la equivalencia de la variable Valsensor.

Sensor_artificio ➔ Es una variable que realiza dos operaciones dependiendo del eje a escoger:

Eje positivo: Es la diferencia entre 90 (El valor de 90 es un artificio aplicado para que el valor del sensor se encuentre en 0º, debido a que nuestro servomotor empieza

con 90 y por ende la variable ValSensor toma el mismo valor) y el dato de la variable sensor.

Eje negativo: Es la diferencia entre la variable sensor y 90 (El valor de 90 es un artificio aplicado para que el valor del sensor se encuentre en 0º, debido a que nuestro Servomotor empieza con 90 y por ende la variable ValSensor toma el mismo valor).

Posición → Es la equivalencia a la variable sensor_artificio.

Sumatotal → Es la variable que guarda el numero decimal ingresado por el usuario desde Labview.

Suma → Es la suma entre Sumatotal y 90.

Sumato → Es la diferencia entre la variable suma y 90.

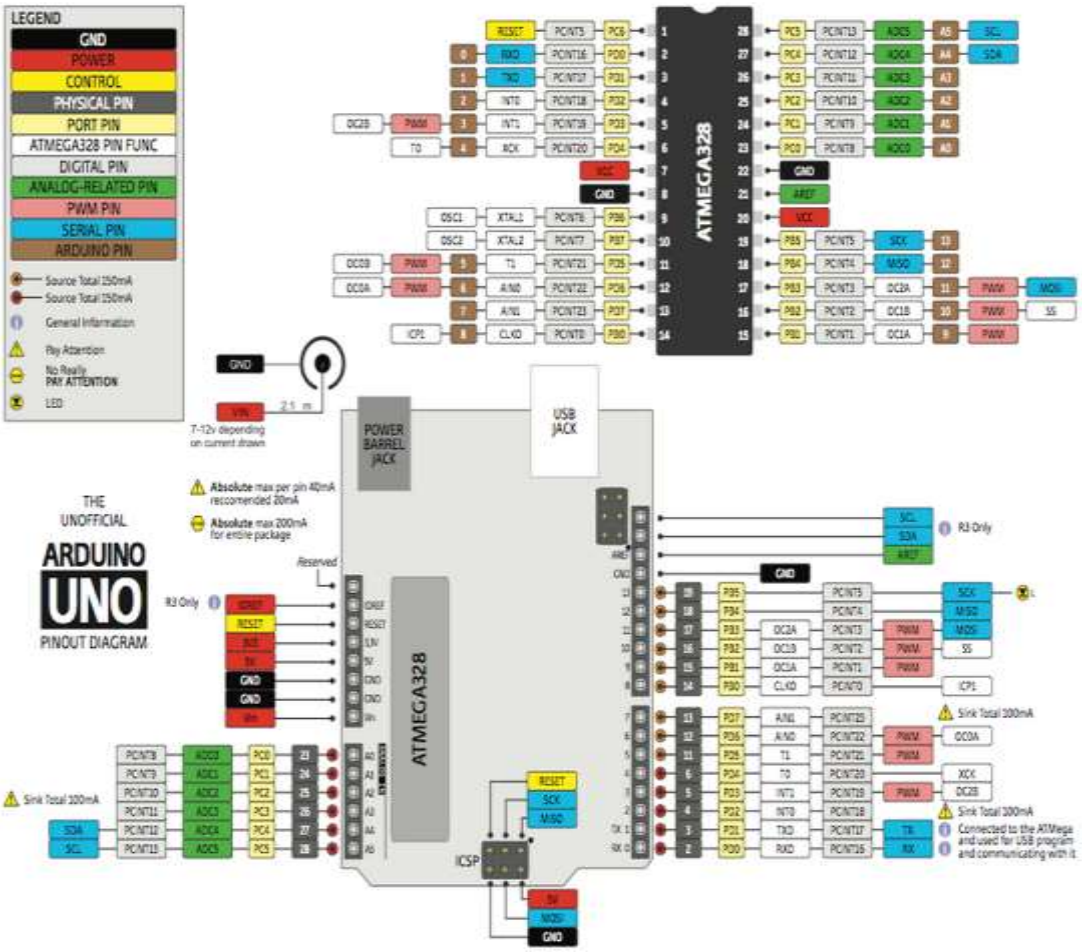
Entrada → Es la equivalencia a la variable sumato.

Error → Es la diferencia entre las variables Entrada y sensor_artificio.

Error2 → Es el valor absoluto de la variable error.

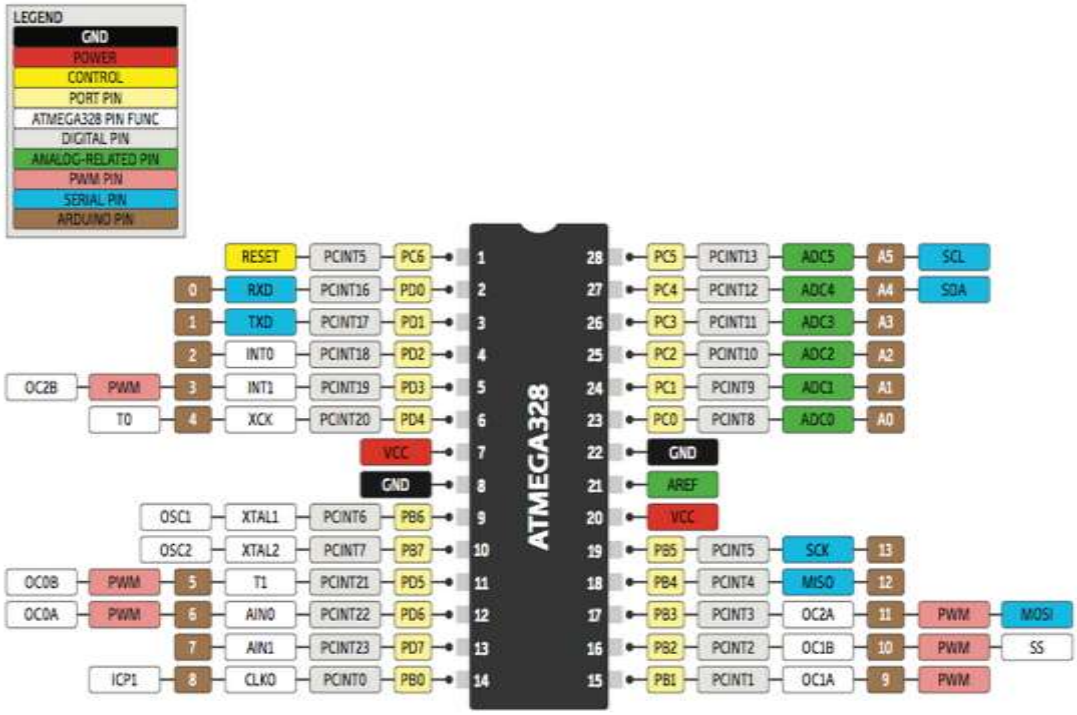
ANEXO 3

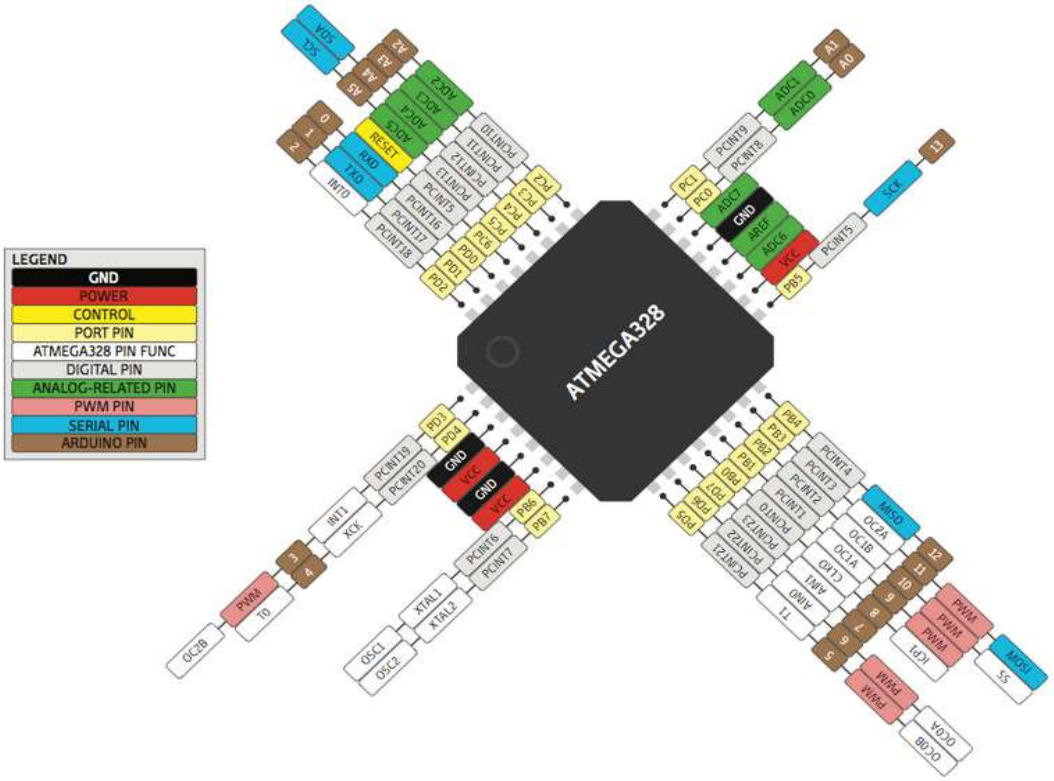
DIAGRAMA ESQUEMÁTICA ARDUINO UNO



ANEXO 4

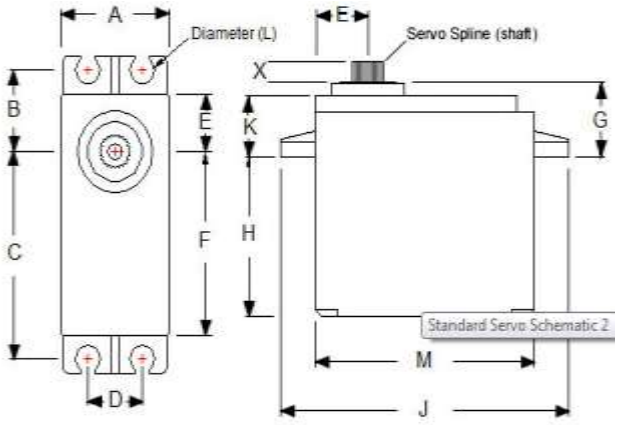
DATASHEET ATMEGA 328P DIP/SMD





ANEXO 5

ESPECIFICACIONES TÉCNICAS SERVOMOTOR



Sistema de control: + ancho de pulso de control 1500usec Neutral

Pulso requerido: 3-5 voltios pico a pico de onda cuadrada

Tensión de funcionamiento: 4,8 a 6,0 voltios

Rango de temperatura de funcionamiento: -20 a +60 grados C

Velocidad de funcionamiento (4.8V): 0.19sec/60 ° sin carga

Velocidad de funcionamiento (6.0V): 0.15sec/60 ° sin carga

Torsión de la parada (4.8V): 42 oz / in (3,0 kg / cm)

Torsión de la parada (6.0V): 51 oz / in (3,7 kg / cm)

Consumo de corriente (4.8V): 7.4mA/idle, 160 mA sin carga operativa

Consumo de corriente (6.0V): 7.7mA/idle, 180 mA sin carga operativa

Anchura de banda muerta: 5usec

Dirección: Multi-direccional

Tipo de motor: con núcleo de metal Brush

Potenciómetro Drive: 4 Control deslizante / de accionamiento directo

Continua rotación modificable: Sí

Conector de cable Longitud: 11.81 "(300 mm)

Peso: 1,52 oz (43 g)