Contents

[**1.** **PERFIL DE PROYECTO** - 3 -](#_Toc443055037)

[**1.1. Introducción.** - 4 -](#_Toc443055038)

[**1.2. Objetivos.** - 4 -](#_Toc443055039)

[**1.3. Antecedentes.** - 5 -](#_Toc443055040)

[**1.4. Estructura Organizacional (Organigrama).** - 5 -](#_Toc443055041)

[**1.5. Justificación.** - 5 -](#_Toc443055042)

[**1.6. Descripción del Problema.** - 5 -](#_Toc443055043)

[**1.7. Alcance del Proyecto.** - 6 -](#_Toc443055044)

[**2.** **FLUJO DE TRABAJO: REQUISITOS** - 12 -](#_Toc443055045)

[**2.1. Identificar Casos de Usos y Actores** - 13 -](#_Toc443055046)

[**2.2. Priorización de Casos de Uso** - 14 -](#_Toc443055047)

[**2.3. Detallar Casos de Uso** - 15 -](#_Toc443055048)

[**2.4. Diagrama General de Caso de Uso** 23](#_Toc443055049)

[**3.** **FLUJO DE TRABAJO: ANALISIS** 24](#_Toc443055050)

[**3.1.- Análisis de la Arquitectura** 25](#_Toc443055051)

[**3.1.2. Vista de Paquetes** 26](#_Toc443055052)

[**3.2. Analizar Casos de Uso (Diagrama de Colaboración)** 27](#_Toc443055053)

[**3.3 ANÁLISIS DE PAQUETES** 30](#_Toc443055054)

[**3.4 Organización Por capas del Análisis** 30](#_Toc443055055)

[**4.** **FLUJO DE TRABAJO: DISEÑO** 31](#_Toc443055056)

[**4.1. Diseño de la Arquitectura** 32](#_Toc443055057)

[**4.2 Detalle de Usuarios** 32](#_Toc443055058)

[**4.3 Diseñar Casos de Uso** 34](#_Toc443055059)

[**4.4. Diseño de Interfaz Humana** 38](#_Toc443055060)

[**5.** **FLUJO DE TRABAJO: IMPLEMENTACION** 39](#_Toc443055061)

[**5.1 Implementación de la Arquitectura** 40](#_Toc443055062)

[**5.2. Estándar de Codificación de Java** 44](#_Toc443055063)

[**6.** **FLUJO DE TRABAJO: PRUEBA** 52](#_Toc443055064)

[**6.1. Caso de Prueba: Avanzar** 53](#_Toc443055065)

[**7.** **MANUAL DE USUARIO** 54](#_Toc443055066)

[**7.1 MOVIMIENTO Y LUCES** 55](#_Toc443055067)

[**8. BIBLIOGRAFÍA** 56](#_Toc443055068)

# **PERFIL DE PROYECTO**

## **1.1. Introducción.**

El desarrollo del software ha cambiado desde sus inicios, lo que desde sus inicios fue considerada una rama poco organizada, debido a la gran falta de normativas, procesos y metodologías.

Los grandes avances en el área de las comunicaciones han dejado atrás por mucho a los avances en los procesos de desarrollo del software, llámese ingeniería del software, han llegado hace no mucho, remontándonos en la historia, el primer avance fue el proceso de desarrollo en cascada. Este es un proceso que se popularizó a nivel mundial, convirtiendo al desarrollo del software en una disciplina, elevándola al nivel de ingeniería.

Otro gran avance dado es la unificación de los distintos métodos de modelado, integrados con un proceso de desarrollo marco, el cual es el Proceso Unificado de Desarrollo de Software.

## **1.2. Objetivos.**

**1.2.1. Objetivo General.-**

* Desarrollar un software de control remoto “Arduino - WIFI” mediante el dispositivo ESP8266 utilizando el protocolo HTTP

**1.2.2. Objetivos Específicos:**

1. Realizar una minuciosa captura de requerimientos, identificando primeramente a los posibles usuarios del software, para luego poder determinar requerimientos de acuerdo a las expectativas de los usuarios.
2. Analizar los Requerimientos obtenidos para definir los requisitos del software.
3. Diseñar una adecuada Arquitectura Lógica (Modelo 2 Capas).
4. Implementar una interfaz capaz de ofrecer al usuario una visión cómoda del Software desde diferentes perspectivas.
5. Realizar el respectivo análisis y diseño mediante la metodología de desarrollo de Software conocido como PUDS y herramientas UML.
6. Implementar el software en un lenguaje de programación adecuado “Arduino, HTML y JavaScript”.
7. Realizar las pruebas necesarias para el óptimo funcionamiento del software.

## **1.3. Antecedentes.**

Existe gran cantidad de software y ejemplos de los mismos cuando se quiere realizar el control remoto de dispositivos basados en Arduino mediante Bluetooth, lo que implica que este queda limitado a unos cuantos metros, dependiendo de dispositivo Bluetooth con que se cuenta, también está limitada librerías propias del dispositivo controlador, como ser un celular con Android y que este cuente con las librerías necesarias para realizar la comunicación Bluetooth.

## **1.4. Estructura Organizacional (Organigrama).**

Browser



RED Ethernet

**incer**

SERVIDOR

ESP8266-Arduino

***Figura 1: Estructura Organizacional (Organigrama)***

## **1.5. Justificación.**

Debido a las limitaciones en distancia que se tiene mediante el uso del Bluetooth, surge la necesidad de utilizar un elemento nuevo (ESP8266) mediante el cual se puede realizar la comunicación con un Access Point y el uso de HTTP para las comunicaciones, lo cual incrementa la distancia de acceso a la capacidad de la red wifi que se monte.

## **1.6. Descripción del Problema.**

Nos centraremos en el uso del componente ESP8266.

La comunicación del Arduino con el ESP8266.

Conexión del ESP8266 a la red WIFI mediante un Access Point.

Montar un servidor web para alojar la aplicación web.

Enviar información web del servidor hacia el ESP8266

Transformar las operaciones en acciones en el Arduino y componentes.

## **1.7. Alcance del Proyecto.**

**1.7.1. Requerimientos funcionales**

El sistema debería permitir:

* Comunicación HTTP mediante el módulo ESP8266.
* Avanzar.
* Retroceder.
* Parar
* Girar a la izquierda.
* Girar a la derecha.
* Encender luces
* Apagar luces.
* Mostrar estado mediante el display LiquidCristal

**1.7.2. Plataforma de desarrollo**

* El sistema se desarrollará bajo la plataforma **HTML, CSS, JQUERY, ARDUINO IDE**
* Para el servidor web se utilizara: **XAMPP**.
* El software desarrollado está en paradigma del Proceso Unificado de Desarrollo de Software (**PUDS**).
* Para visualizar, especificar, construir y documentar utilizamos el Lenguaje Unificado de Modelado (UML).
* La Arquitectura está basada en el paradigma Cliente / Servidor

**1.8. MARCO TEÓRICO**

**1.8.1. SISTEMAS DISTRIBUIDOS**

***Introducción***

Los sistemas de computo están sufriendo una revolución desde 1945 cuando comenzó la era de la computadora moderna hasta cerca de 1985 las computadoras eran grandes y caras incluso las mini computadoras costaban por lo general cientos de miles de dólares cada una como resultado la mayor parte de las organizaciones tenia tan solo un puñado de computadoras y por carecer de una forma para conectarlas estas operaban por lo general en forma independiente entre si. Sin embargo a partir de la mitad de la década de 1980 dos avances tecnológicos comenzaron a cambiar esta situación.

El primero fue el desarrollo de poderosos microprocesadores. En principio se disponía de maquinas de 8 bits pero se volvieron comunes las CPU de 16 32 e incluso 64 bits

***¿Qué es un Sistema distribuido?***

Un sistema distribuido es una colección de computadoras independientes que aparecen ante los usuarios del sistema como una computadora.

Esta definición tiene dos aspectos:

* *El primero* se refiere al hardware las maquinas son autónomas.
* *El segundo* se refiere al software los usuarios piensan que el sistema es como una única computadora, ambos son esenciales.

*Como primer ejemplo:* una red de estaciones de trabajo en un departamento de una universidad o compañía, además de cada estación de trabajo personal podría existir una pila de procesadores en el cuarto de máquinas que no estén asignados a usuarios específicos sino que se utilicen de manera dinámica conforme sea necesario.

Ventajas de los sistemas distribuidos con respecto de las PC independientes, puesto que los microprocesadores constituyen una forma económica de trabajo ¿por qué no se ofrece a cada persona su propia PC y se la deja trabajar de manera independiente?

Imaginémonos el caso de empleados de reservaciones en las líneas aéreas necesitan tener acceso a la base de datos maestras de los vuelos y reservaciones existentes, en caso de que cada uno usara una copia particular, los empleados no llegarían a saber quien vendió que asiento. Los datos compartidos son absolutamente esenciales para esta y otras aplicaciones de modos que las máquinas deben estar conectadas entre si. La conexión de las maquinas conduce a un sistema distribuido.

***Desventajas de los sistemas distribuidos***

Aunque los sistemas distribuidos tienen sus aspectos fuertes también tienen sus debilidades como por ejemplo:

* Un problema potencial es el debido a las redes de comunicación, estas pueden perder mensajes lo cual requiere un software especial para su manejo y puede verse sobrecargado, al saturarse la red esta debe remplazarse o añadir una segunda, en ambos casos hay que tender cables en una parte de uno o mas edificios con gran costo o bien hay que remplazar las tarjetas de interfaz de red (por ejemplo por fibras ópticas).
* Una vez que el sistema llega a depender de la red la perdida o saturación de esta puede negar algunas de las ventajas que el sistema distribuido debía conseguir.
* Por ultimo el hecho ya descrito de que los datos sean fáciles de compartir es una ventaja pero se pude convertir en un arma de dos filos. Si las personas pueden tener acceso a los datos en todo el sistema entonces también pueden tener acceso a los datos con los que no tiene nada que ver. En otras palabras la seguridad es con frecuencia un problema. Para que los datos se mantengan en secreto a toda costa es preferible tener una computadora personal aislada sin conexiones de red con las demás maquinas y mantenerlas en un cuarto cerrado con un mueble seguro donde guardar todos los discos flexibles.

***Comunicación en los sistemas distribuidos***

La diferencia mas importante entre un sistema distribuido y un sistema con un procesador es la comunicación entre procesos. En un sistema con un sistema con un procesador, la mayor parte de la comunicación entre procesos supone de manera implícita la existencia de la memoria compartida. Un ejemplo típico es el problema de los productores y consumidores, donde un proceso escribe en un buffer compartido y otro proceso lee de el.

***Protocolo con capas***

Debido a la ausencia de memoria compartida, toda la comunicación en los sistemas distribuidos se basa en la transferencia de mensajes. Cuando el proceso A quiere comunicarse con el proceso B, construye primero un mensaje en su propio espacio de direcciones. Entonces ejecuta una llamada al sistema para que el sistema operativo busque el mensaje y lo envié a través de la red hacia B, aunque parezca sencillo deben coincidir en el significado de los bits que se envíen.

***Arquitectura de aplicaciones multicapas orientadas a objetos***

Según [LAR-99] una arquitectura multicapas se adecua a los sistemas de información orientados a los objetos e incluye la división de las responsabilidades que encontramos en la arquitectura clásica de tres capas. Las responsabilidades se asignan a objetos de software. Uno de los beneficios del paradigma orientado a objetos es que fomenta la reutilización del software: en efecto, conceptos como encapsulación y herencia facilitan la construcción de clases que podemos desarrollar solamente una vez para incorporarlas en varios sistemas.

Una aplicación multicapa consta al menos de 3 capas, que habitualmente son las siguientes:

* Presentación, en la que residen las ventanas que se encargan de mostrar la información y con las que interactúa el usuario.
* Dominio (también llamada de negocio o de procesamiento), en la que reside toda la lógica que describe los mecanismos para resolver el problema.
* Almacenamiento, en la que reside la posible base de datos gestionada por la aplicación, mas un conjunto de clases adicionales.
* Pueden existir capas adicionales que se encarguen de otras funcionalidades, como la gestión de las comunicaciones con otros sistemas.

***Sistemas Distribuidos (Desarrollo)***

* *Modelo de igual a igual (peer topeer)*
* Redes de igual a igual o de pares o peer to peer (P2P)
* Comunicación entre sistemas finales iguales
* Ningún sistema final tiene mayor prioridad que otro
* No existen clientes y servidores fijos
* Cada proceso participante es cliente (solicitudes) y servidor (respuestas), y a menudo, simultáneamente
* Cliente envía solicitudes de servicio
* Servidor escucha de forma pasiva y proporciona servicios como
* respuestas

Aplicación:

Napster: Primera aplicación P2P de compartición y transferencia de ficheros (de música) basada en un servidor central de gestión que asocia ficheros con máquinas y organiza el tráfico de ficheros entre iguales. Para preguntar por un fichero hay que consultar al servidor central.

Emule/eDonkey: Basadas en un servidor central tipo Nasper.

GNUtella/ KaZaa: Conexión directa entre iguales sin un servidor central mediante un protocolo de búsqueda distribuida

MSN Messenger: Mensajería instantánea (multiuserchat) en tiempo real para usuarios del sistema Windows

SKYPE: Telefonía entre pares por Internet.

* *Modelo tradicional de cliente y servidor*
* Redes de cliente y servidor

• Un proceso es cliente y otro servidor

* El proceso servidor puede, a su vez, convertirse en cliente de otro proceso servidor igual

Aplicación:

Sistema RPC: software interno basado en un nivel intermedio de comunicaciones nivel intermedio de comunicaciones estructurado en 2 módulos de comunicaciones o STUBS estructurado en 2 módulos de comunicaciones o STUBS que independizan a las independizan a las implementaciones reales del cliente y servidor implementaciones reales del cliente y servidor(permitiendo que puedan (permitiendo que puedan ser escritas en diferentes lenguajes) para ser escritas en diferentes lenguajes) para realizar las llamadas al interfaz de sockets realizar las llamadas al interfaz de sockets

# **FLUJO DE TRABAJO: REQUISITOS**

## **2.1. Identificar Casos de Usos y Actores**

**Caso de Estudio:**

Software para Control Remoto.

****

1. **Identificar Actores**

* **Usuario:** Es la persona que se encarga de ejecutar las acciones.

1. **Identificar Casos de Uso**

|  |  |
| --- | --- |
| **Nro.** | **Caso de Uso** |
| **C.U. 1** | **Inicializar Dispositivo** |
| **C.U. 2** | **Avanzar** |
| **C.U. 3** | **Retroceder** |
| **C.U. 4** | **Parar** |
| **C.U. 5** | **Girar a la izquierda** |
| **C.U. 6** | **Girar a la derecha** |
| **C.U. 7** | **Encender Luces** |
| **C.U. 8** | **Apagar Luces** |
| **C.U. 9** | **Mostrar estado mediante Display** |

## **2.2. Priorización de Casos de Uso**

|  |  |  |
| --- | --- | --- |
| **Nro.** | **Caso de Uso** | **Prioridad** |
| **C.U. 1** | **Inicializar Dispositivo** | **A** |
| **C.U. 2** | **Avanzar** | **B** |
| **C.U. 3** | **Retroceder** | **B** |
| **C.U. 4** | **Parar** | **B** |
| **C.U. 5** | **Girar a la izquierda** | **B** |
| **C.U. 6** | **Girar a la derecha** | **B** |
| **C.U. 7** | **Encender Luces** | **B** |
| **C.U. 8** | **Apagar Luces** | **B** |
| **C.U. 9** | **Mostrar estado mediante Display** | **B** |

**A: Alta.**

**B: Media.**

**C: Baja.**

## **2.3. Detallar Casos de Uso**

**CU1: Inicializar Dispositivo**

****

|  |  |
| --- | --- |
| **Nro.: 1** | **Caso de uso:** Inicializar Dispositivo |
| **Propósito:** Poner en marcha el Arduino, ESP8266 y componentes. | |
| **Descripción:** Esto ocurre cuando enciende conecta el Arduino a una fuente de Energia. | |
| **Actores:** Usuario | |
| **Iniciador:** Usuario | |
| **Procesos:**   * **Encender o conectar el Arduino a una fuente de Alimentación.** | |
| **Casos Especiales:** | |

**CU2: Avanzar**

****

|  |  |
| --- | --- |
| **Nro.: 2** | **Caso de uso:** Avanzar |
| **Propósito:** Permite al dispositivo avanzar. | |
| **Descripción:** Al elegir esta opción pondrá en marcha el motor del dispositivo en modo de avance. | |
| **Actores:** Usuario | |
| **Iniciador:** Usuario | |
| **Procesos:**   * **Avanzar** * Seleccionar la opción Avanzar. * Pone en marcha el motor activando en puente H en modo de Avance. | |
| **Casos Especiales:** | |

**CU3: Retroceder**

****

|  |  |
| --- | --- |
| **Nro.: 3** | **Caso de uso:** Retroceder |
| **Propósito:** Permite al dispositivo Retroceder | |
| **Descripción:** Al elegir esta opción pondrá en marcha el motor del dispositivo en modo de retroceso. | |
| **Actores:** Usuario | |
| **Iniciador:** Usuario | |
| **Procesos:**   * **Retroceder** * Selecciona la opción Retroceder. * Pone en marcha el motor activando en puente H en modo de Retroceso. | |
| **Casos Especiales:** | |

**CU4: Parar**

****

|  |  |
| --- | --- |
| **Nro.:4** | **Caso de uso:** Parar |
| **Propósito:** Permite al dispositivo detenerse. | |
| **Descripción:** Al elegir esta opción detendrá el motor del dispositivo. | |
| **Actores:** Usuario | |
| **Iniciador:** Usuario | |
| **Procesos:**   * **Parar** * Selecciona la opción Parar. * Detiene el motor | |
| **Casos Especiales:** | |

**CU5: Girar Izquierda**

****

|  |  |
| --- | --- |
| **Nro.:5** | **Caso de uso:** Girar Izquierda |
| **Propósito:** Permite al dispositivo girar a la izquierda. | |
| **Descripción:** Al elegir esta opción el servo motor del dispositivo girara hacia la izquierda. | |
| **Actores:** Usuario | |
| **Iniciador:** Usuario | |
| **Procesos:**   * **Girar a la izquierda** * Selecciona la opción girar a la izquierda. * Pone en marcha el servo motor y hace que gire a la izquierda | |
| **Casos Especiales:** | |

**CU6: Girar Derecha**

****

|  |  |
| --- | --- |
| **Nro.:6** | **Caso de uso:** Girar Derecha |
| **Propósito:** Permite al dispositivo girar a la derecha. | |
| **Descripción:** Al elegir esta opción el servo motor del dispositivo girara hacia la derecha. | |
| **Actores:** Usuario | |
| **Iniciador:** Usuario | |
| **Procesos:**   * **Girar a la derecha** * Selecciona la opción girar a la derecha. * Pone en marcha el servo motor y hace que gire a la derecha | |
| **Casos Especiales:** | |

**CU7: Encender Luces**

****

|  |  |
| --- | --- |
| **Nro.:7** | **Caso de uso:** Encender Luces |
| **Propósito:** Permite al dispositivo encender las luces led. | |
| **Descripción:** Al elegir esta opción el dispositivo enciende las luces led que tiene incorporado. | |
| **Actores:** Usuario | |
| **Iniciador:** Usuario | |
| **Procesos:**   * **Encender Luces** * Selecciona la opción Encender Luces. * Enciende las luces led que tiene el dispositivo | |
| **Casos Especiales:** | |

**CU8: Apagar Luces**

****

|  |  |
| --- | --- |
| **Nro.:8** | **Caso de uso:** Apagar Luces |
| **Propósito:** Permite al dispositivo apagar las luces led. | |
| **Descripción:** Al elegir esta opción el dispositivo apaga las luces led que tiene incorporado. | |
| **Actores:** Usuario | |
| **Iniciador:** Usuario | |
| **Procesos:**   * **Apagar Luces** * Selecciona la opción Apagar Luces. * Apaga las luces led que tiene el dispositivo | |
| **Casos Especiales:** | |

**CU9: Mostrar Estado**

****

|  |  |
| --- | --- |
| **Nro.:9** | **Caso de uso:** Mostrar Estado |
| **Propósito:** Permite al dispositivo mostrar datos mediante el Display. | |
| **Descripción:** Al elegir esta opción el dispositivo activara el Display y mostrar en texto el estado del dispositivo. | |
| **Actores:** Arduino | |
| **Iniciador:** Arduino | |
| **Procesos:**   * **Mostrar Estado** * Arduino al inicializar el dispositivo, obtiene el ip de conexión. * Muestra la ip en el display LiquidCrystal | |
| **Casos Especiales:** | |

## **2.4. Diagrama General de Caso de Uso**

****

# **FLUJO DE TRABAJO: ANALISIS**

## **3.1.- Análisis de la Arquitectura**

**3.1.1. Identificar Paquetes**

**Paquete: Control**

****

**Paquete: Arduino**



**Dependencia de Paquetes**



## **3.1.2. Vista de Paquetes**

**Paquete: Control**

**Paquete: Arduino**

****

**Paquete: Control**



## **3.2. Analizar Casos de Uso (Diagrama de Colaboración)**

**CU1: Inicializar Dispositivo**

****

**CU2: Avanzar**

****

**CU3: Retroceder**

****

**CU4: Parar**

****

**CU5: Girar Izquierda**

****

**CU6: Girar Derecha**

****

**CU7: Encender Luces**

****

**CU8: Apagar Luces**

****

**CU9: Mostrar Estado**

****

## **3.3 ANÁLISIS DE PAQUETES**

**3.4.1. Dependencia entre paquetes**

****

## **3.4 Organización Por capas del Análisis**

****

# **FLUJO DE TRABAJO: DISEÑO**

## **4.1. Diseño de la Arquitectura**

**4.1.1. Diagrama de Paquete**

****

## **4.2 Detalle de Usuarios**

1. **Usuario1: Arduino**

Es el dispositivo programable que puede realizar las siguientes acciones:

* Inicializar Dispositivo.
* Mostrar Estado

1. **Usuario2: Usuario**

Es la persona que puede realizar las siguientes acciones.

* Inicializar Dispositivo.
* Mostrar Estado.
* Avanzar.
* Retroceder.
* Parar.
* Encender Luces.
* Apagar Luces
* Girar Izquierda
* Girar Derecha

**4.1.2. Diagrama de Despliegue**

 ****

## **4.3 Diseñar Casos de Uso**

**4.2.1. Diagrama de Secuencia**

**CU1: Inicializar Dispositivo**

****

**CU2: Avanzar**

****

**CU3: Retroceder**

****

**CU4: Parar**

****

**CU5: Girar Izquierda**

****

**CU6: Girar Derecha**

****

**CU7: Encender Luces**

****

**CU8: Apagar Luces**

****

**CU9: Mostrar Estado**

****

## **4.4. Diseño de Interfaz Humana**

**INTERFAZ DEL SERVIDOR**

**4.3.1. Formulario de Control**



# **FLUJO DE TRABAJO: IMPLEMENTACION**

## **5.1 Implementación de la Arquitectura**

 ****

**5.1.1. Elección del lenguaje de programación**

**Arduino**

**Estructura de un programa**

La estructura básica del lenguaje de programación de Arduino es bastante simple y se compone de al menos dos partes. Estas dos partes necesarias, o funciones, encierran bloques que contienen declaraciones, estamentos o instrucciones.

void setup() //Primera Parte

{

estamentos;

}

void loop() //Segunda Parte

{

estamentos;

}

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

La función de configuración (**setup**) 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 en serie y otras.

La función bucle (**loop**) siguiente 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.

**setup()**

La función **setup()** se invoca una sola vez cuando el programa empieza. Se utiliza para inicializar los modos de trabajo de los pins, o el puerto serie. Debe ser incluido en un programa aunque no haya declaración que ejecutar. Así mismo se puede utilizar para establecer el estado inicial de las salidas de la placa.

void setup()

{

pinMode(pin, OUTPUT); // configura el 'pin' como salida

digitalWrite(pin, HIGH); // pone el ‘pin’ en estado HIGH

}

**loop()**

Después de llamar a **setup()**, la función **loop()** hace precisamente lo que sugiere su nombre, se ejecuta de forma cíclica, lo que posibilita que el programa esté respondiendo continuamente ante los eventos que se produzcan en la placa.

void loop()

{

digitalWrite(pin, HIGH); // pone en uno (on, 5v) el 'pin'

delay(1000); // espera un segundo (1000 ms)

digitalWrite(pin, LOW); // pone en cero (off, 0v.) el 'pin'

delay(1000);

}

**funciones**

Una función es un bloque de código que tiene un nombre y un conjunto de instrucciones que son ejecutadas 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 devuelve 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.

type nombreFunción(parámetros)

{

instrucción;

}

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 retornaría al programa principal. Esta función cuando se ejecuta devuelve el valor de tipo entero 'v'.

int delayVal()

{

int v; // crea una variable temporal 'v'

v= analogRead(pot); // lee el valor del potenciómetro

v /= 4; // convierte 0-1023 a 0-255

return v; // devuelve el valor final

}

**{} entre llaves**

Las llaves sirven para definir el principio y el final de un bloque de instrucciones. Se utilizan para los bloques de programación setup(), loop(), if.., etc.

type funcion()

{

instrucciones;

}

Una llave de apertura “{“ siempre debe ir seguida de una llave de cierre “}”, si no es así el programa dará errores.

El entorno de programación de Arduino incluye una herramienta de gran utilidad para comprobar el total de llaves. Sólo tienes que hacer click en el punto de inserción de una llave abierta e inmediatamente se marca el correspondiente cierre de ese bloque (llave cerrada).

**; punto y coma**

El punto y coma “;” se utiliza para separar instrucciones en el lenguaje de programación de Arduino. También se utiliza para separar elementos en una instrucción de tipo “bucle for”.

int x = 13; /\* declara la variable 'x' como tipo entero de valor 13 \*/

Nota: Olvidaos de poner fin a una línea con un punto y coma o se producirá en un error de compilación. El texto de error puede ser obvio, y se referirá a la falta de una coma, o puede que no. Si se produce un error raro y de difícil detección lo primero que debemos hacer es comprobar que los puntos y comas están colocados al final de las instrucciones.

**/\*… \*/ bloque de comentarios**

Los bloques de comentarios, o comentarios multi-línea son áreas de texto ignorados por el programa que se utilizan para las descripciones del código o comentarios que ayudan a comprender el programa. Comienzan con / \* y terminan con \* / y pueden abarcar varias líneas.

/\* esto es un bloque de comentario no se debe olvidar

cerrar los comentarios estos deben estar equilibrados \*/

Debido a que los comentarios son ignorados por el compilador y no ocupan espacio en la memoria de Arduino pueden ser utilizados con generosidad. También pueden utilizarse para "comentar" bloques de código con el propósito de anotar informaciones para depuración y hacerlo mas comprensible para cualquiera.

Nota: Dentro de una misma línea de un bloque de comentarios NO se puede escribir otro bloque de comentarios (usando /\*..\*/).

**// línea de comentarios**

Una línea de comentario empieza con // y terminan con la siguiente línea de código. Al igual que los comentarios de bloque, los de línea son ignoradas por el programa y no ocupan espacio en la memoria.

// esto es un comentario

Una línea de comentario se utiliza a menudo después de una instrucción, para proporcionar más información acerca de lo que hace ésta o para recordarla más adelante.

## **5.2. Estándar de Codificación de Java**

***Espacios en blanco***

Los espacios en blanco y líneas vacías proporcionan mayor legibilidad al código que se escribe, tanto porque las sentencias se visualizan mucho más claramente como porque las secciones de código diferentes se encuentran separadas adecuadamente.

***Líneas en blanco.-***Se debe utilizar siempre una línea en blanco de separación en las siguientes circunstancias:

* Entre métodos
* Entre las variables locales en un método y su primera sentencia
* Antes de un bloque de código o un comentario de una línea
* Entre secciones lógicas dentro de un método para incrementar la legibilidad del código

Es conveniente el uso de dos líneas en blanco en las circunstancias que se indican:

* Entre secciones dentro de un fichero fuente
* Entre las definiciones de clases e interfaces

***Espacios en blanco.-*** Se deberían utilizar espacios en blanco de separación en las siguientes circunstancias:

* Una palabra clave seguida por un paréntesis debería ir separada por un espacio en blanco, bien antes o después del paréntesis. Por ejemplo:

while (true) {

...

}

Aunque también sería admisible,

while( true ) {

...

}

Entre el nombre de un método y su paréntesis de apertura, no debe colocarse ningún espacio en blanco, sino después del paréntesis, tal como el segundo ejemplo anterior. Esto ayuda a distinguir claramente las palabras clave de la llamada al método.

* Se debe colocar un espacio de separación entre los argumentos de la lista de comandos, después de las comas.
* Todos los operadores, excepto el punto (.), debería separarse de sus operandos mediante espacios. No obstante, espacios en blanco no deben separar nunca operadores unarios como el menos unario, o los operadores unarios de incremento y decremento, de sus operadores.

a += c + d;

a = (a + b) / (c \* d);

while (d++ = s++) {

n++;

}

prints("size is " + foo + "\n");

* Las expresiones en una sentencia for deben ir separadas mediante espacios en blanco.

for (expr1; expr2; expr3)

* Los moldeos (casting) deben ir seguidos de un espacio en blanco. Por ejemplo:

myMethod( (byte) aNum, (Object) x );

myFunc( (int) (cp + 5), ((int) (i + 3)) + 1 );

**Convenciones de nomenclatura**

Las convenciones de nomenclatura hacen que los programas sean más estándar y fáciles de leer, ya que cualquier programador está acostumbrado a tratar con código escrito de forma semejante. Además, proporcionan información sobre la funcionalidad del identificador; por ejemplo, si es una constante, un paquete o una clase, lo que también redunda en una ayuda adicional a la hora de entender el código.

Las convenciones que se indican aquí son de muy alto nivel y muy generales. Probablemente sea necesario, en proyectos de al menos una envergadura mediana, proporcionar convenciones más específicas.

|  |  |  |
| --- | --- | --- |
| ***Identificador*** | ***Reglas de Nomenclatura*** | ***Ejemplo*** |
| *Clases* | Los nombres de clases deben ser palabras completas, en mayúsculas y minúsculas, con la primera letra de cada palabra en mayúscula. Los nombres de clases deben ser simples y descriptivos, utilizando palabras completas y acrónimos o abreviaturas (a no ser que la abreviatura sea ampliamente conocida, como URL o HTML). | class Raster;  class ImageSprite; |
| *Interfaces* | Los nombre de interfaces deberías seguir las mismas reglas indicadas para las clases. | interface RasterDelegate;  interface Storing; |
| *Métodos* | Los métodos deberían ser verbos, en mayúsculas y minúsculas, con la primera letra en minúscula, y la primera letra de cada una de las palabras internas en mayúscula. | run();  runFast();  getBackground(); |
| *Variables* | Todas los nombres de variables de instancia o de clase deben estar constituidos por palabras con la primera letra de la primera palabra en minúscula y la primera letra de las palabras internas en mayúscula.  Los nombres de variables deben ser cortos y significativos. La elección de un nombre de variable debe ser mnemotécnica, es decir, pensado para que un lector casual al verla comprenda su uso. Se deben evitar las variables de una sola letra, excepto en variables temporales de corto uso. Nombres comunes para este tipo de variables son: i, j, k, m y n para enteros; c, d, y e para caracteres. | int i;  char \*cp;  float myWidth; |
| *Constantes* | Los nombres de variables declaradas como constantes de clase y constantes ANSI, deberías escribirse siempre en mayúsculas, con las palabras internas separadas por el signo de subrayado ("\_"). Las constantes ANSI se deben evitar en lo posible, para facilitar la depuración del código. | int MIN\_WIDTH = 4;  int MAX\_WIDTH = 999;  int GET\_THE\_CPU = 1; |

**Técnicas de programación**

Estas son algunas sugerencias acerca de cuestiones de programación más habituales y comunes.

***Proporcionar acceso a variables de clase e instancia.-*** No hacer nunca una instancia o una variable de clase pública sin una buena razón. Normalmente, las variables de instancia necesitan que sus valores sean fijados o recogidos explícitamente; a menudo esto es un efecto lateral de la llamada al método.

Un ejemplo de uso adecuado de las variables de instancia públicas es el caso en que la clase es esencialmente una estructura de datos, sin ninguna funcionalidad. Es decir, si se desea utilizar una estructura en lugar de una clase (si la estructura está soportada por Java), entonces sí es apropiado el uso de variables de instancia públicas.

***Referencias variables y métodos de clase.-*** Evitar el uso de un objeto para acceder a una variable de clase (estática) o método. Utilizar el nombre de la clase en su lugar. Por ejemplo:

classMethod(); // OK

AClass.classMethod(); // OK

anObject.classMethod(); // EVITARLO!

***Asignación de variables.-*** Las constantes numéricas (literales) no deben codificarse directamente, excepto para –1, 0 y 1; porque pueden aparecer en un bucle for como valores de contador.

***Técnicas varias.-*** Evitar la asignación de diferentes variables con el mismo valor en una misma sentencia. Esto es muy difícil de leer. Por ejemplo:

fooBar.fChar = barFoo.lchar = 'c'; // EVITARLO!

No utilizar el operador de asignación en un lugar donde puede confundirse fácilmente con un operador condicional. Por ejemplo:

if (c++ = d++) { // EVITARLO! Java lo rechaza

...

}

Debería escribirse de la siguiente forma:

if ((c++ = d++) != 0) {

...

}

No utilizar asignaciones embebidas, en un intento de mejorar el rendimiento en tiempo de ejecución. Esto es un trabajo que debe realizar el compilador, y actualmente, rara vez es una ayuda. Por ejemplo:

d = (a = b + c) + r; // EVITARLO!

Debería escribirse de la siguiente forma:

a = b + c;

d = a + r;

***Paréntesis.-*** Es siempre una buena idea el uso abundante de paréntesis en expresiones que involucren a varios operadores para evitar los problemas generados por la precedencia de operadores. Incluso aunque la precedencia de los operadores parezca clara al programador, es posible que no lo sea tanto para otros, por lo que se debe asumir que esa precedencia no es bien conocida por todo el mundo. Y además, los paréntesis son gratis, así que pueden usarse libremente.

if (a == b && c == d) // EVITARLO!

if ((a == b) && (c == d)) // CORRECTO

***Retorno de valores.-*** Intentar que la estructura del programa coincida con la intención, sin abundar en lo ya evidente.

if (expresionBooleana) {

return TRUE;

} else {

return FALSE;

}

Debería escribirse de la siguiente forma:

return expresionBooleana;

De forma semejante,

if (condicion) {

return x;

}

return y;

Debería escribirse de la siguiente forma:

return (condicion ? x : y);

***Expresión antes del ? de una condición.-*** Si una expresión conteniendo un operador binario se coloca antes del interrogante en una expresión ternaria del tipo ?: debería colocarse entre paréntesis. Por ejemplo:

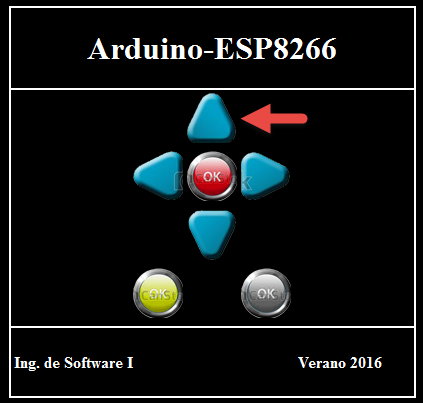
(x >= 0) ? x : -x

***Comentarios especiales.-*** Utilizar XXX, o alguna otra secuencia convenida, como comentario e indicador de algo que se está probando y que aparentemente funciona. Usar FIXME, o alguna otra secuencia convenida, como comentario e indicador de algo que se está probando y funciona mal, erráticamente o se está mejorando.

# **FLUJO DE TRABAJO: PRUEBA**

## **6.1. Caso de Prueba: Avanzar**

**Componente de Prueba**



**Entrada**

* Nombre del Servidor: localhost
* IP del Servidor: 192.168.43.100
* Nro. De Puerto: 80

**Resultado**

* El dispositivo arranca el motor y avanza.

**Condiciones**

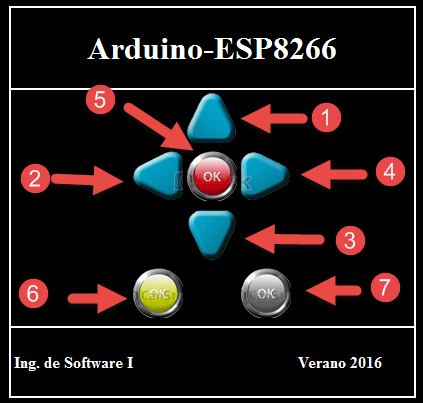
1. Que la Dirección del IP establecida este bien escrita y en el rango específicos.
2. Que el puerto sea numérico y que también se encuentre entre los números de puertos disponibles.

**Procedimiento de Prueba**

1. Conectar el Arduino a una fuente de alimentación o en su caso a la computadora.
2. Se acciona el boton avanzar.
3. Verificar que el dispositivo acciona el motor y tienen la posibilidad de avanzar.

# **MANUAL DE USUARIO**

## **7.1 MOVIMIENTO Y LUCES**



**1.-** Permite al carro avanzar, lo cual activa el motor que tiene el carro y permite el avance del mismo.

**2.-** Permite al carro girar a la izquierda, que junto con el avance activado, logran un giro hacia la izquierda

**3.-** Permite al carro retroceder, lo cual activa el motor que tiene el carro y permite el retroceso del mismo.

**4.-** Permite al carro girar a la derecha, que junto con el avance activado, logran un giro hacia la derecha.

**5.-** Permite que el carro se detenga, con lo cual queda estático para una futura acción.

**6.-** Permite al carro encender las luces led que tienen incorporadas.

**7.-** Permite al carro apagar las luces led que tienen incorporadas.

# **8. BIBLIOGRAFÍA**

* Programacion Arduino: http://playground.arduino.cc/ArduinoNotebookTraduccion/Structure
* HTML-JQUERY

https://api.jquery.com/jquery.get/

* Javier García de Jalón, José Ignacio Rodríguez, Iñigo Mingo, Aitor Imaz, Alfonso Brazales, Alberto Larzabal , Jesús Calleja , Jon García -- APRENDA JAVA COMO SI ESTUVIERA EN PRIMERO
* Libros, revistas y artículos
* Patrones de diseño aplicados a JAVA Stephen Stelting / Olav Maassen
* Patrick Niemeyer, Jonathan Knudsen - CURSO DE JAVA
* Pressman, R., “Ingenieria del Software”: Un enfoque Practico. Mc Graw Hill/INTERAMERICANA DE ESPAÑA, S.A.U, 2001.
* Bertrand Meyer, “Orientado a Objetos”. Segunda Edición, Prentice Hall.