UNIVERSIDAD PROVINCIAL DEL SUDOESTE

FACULTAD DE LA MICRO, PEQUEÑA

Y MEDIANA EMPRESA

CARRERA DE TECNICATURA UNIVERSITARIA EN TECNOLOGÍAS

DE PROGRAMACIÓN SEDE PUNTA ALTA

TRABAJO FINAL

Autores:

Iván Gonzalo Tapia

Juan Matías Cagna

Docente:

Ing. Ronny Stalin Guevara Cruz

Punta Alta – Argentina

2023

CONTENIDO

RESUMEN

Objetivo

El proyecto tiene como objetivo principal diseñar e implementar un sistema embebido de bajo costo para la automatización del hogar, con un enfoque integral en el control de la iluminación, seguridad, espacios y jardinería.

Objetivos Específicos

Control de la Iluminación: Implementar el encendido y apagado de luces en distintas áreas del hogar mediante comandos desde una terminal virtual.

Seguridad: Integrar un sistema de alarma activado y desactivado a través de la terminal virtual, proporcionando una respuesta visual clara.

Control de Espacios: Automatizar la apertura y cierre de la cochera y el portón mediante un sistema motorizado representado por relés y motores.

Automatización de Jardinería: Desarrollar un sistema de riego controlado desde la terminal virtual para mantener eficientemente el cuidado del jardín.

Interfaz de Usuario: Permitir una interacción amigable con el sistema a través de una interfaz en la terminal virtual, brindando mensajes informativos para una retroalimentación efectiva.

Marco Teórico

Se presenta una introducción al Arduino Uno, una placa de microcontrolador de código abierto, destacando su origen, evolución, características técnicas y funciones.

Procedimiento

El desarrollo del sistema se describe detalladamente, desde la configuración inicial de pines en Arduino Uno hasta la implementación de funciones clave como el control de iluminación, sistema de alarma, control de espacios y automatización de jardinería.

Análisis de Resultados

Se realizaron pruebas exhaustivas para verificar el correcto funcionamiento del sistema. Cada componente respondió según las especificaciones, y la comunicación serial permitió una interacción eficiente entre el usuario y el sistema.

Conclusiones y Recomendaciones

El capítulo final presenta las conclusiones obtenidas durante la implementación y proporciona recomendaciones para posibles mejoras o expansiones futuras del sistema.

Este proyecto busca proporcionar una solución práctica y asequible para la automatización del hogar, explorando y aplicando conceptos de sistemas embebidos con un enfoque en la accesibilidad y funcionalidad del sistema.

INTRODUCCIÓN

En un mundo cada vez más interconectado, la domótica emerge como un campo prometedor, ofreciendo soluciones inteligentes para mejorar la calidad de vida en nuestros hogares. Este proyecto se sumerge en el corazón de la automatización del hogar, proponiendo una solución integral y asequible a través de un sistema embebido basado en la plataforma Arduino Uno.

Contexto y Justificación

La necesidad de optimizar el control de los entornos domésticos ha impulsado la investigación y desarrollo de sistemas embebidos que faciliten la gestión eficiente de diversos aspectos, como la iluminación, seguridad, manejo de espacios y cuidado del entorno exterior. La accesibilidad y simplicidad son pilares fundamentales de este proyecto, destinado a entusiastas de la domótica y personas con diversos niveles de experiencia en tecnología.

Objetivos del Proyecto

El proyecto tiene como objetivo principal diseñar e implementar un sistema embebido de bajo costo, utilizando la versátil plataforma Arduino Uno, para automatizar funciones clave en el hogar. Desde el control inteligente de la iluminación hasta la gestión de espacios, seguridad y jardinería, cada aspecto se aborda con el propósito de ofrecer una solución práctica y accesible.

Estructura del Informe

La estructura del informe se divide en varios capítulos, comenzando con los objetivos generales y específicos que guían el proyecto. A continuación, el Marco Teórico explica las características del Arduino Uno, mostrando su evolución y aplicaciones. El Procedimiento detalla las etapas de implementación y codeo, seguido por un análisis de los resultados obtenidos. Finalmente, las Conclusiones y Recomendaciones cierran el informe, proporcionando recomendaciones y posibles direcciones futuras para la mejora del sistema.

Este proyecto se plantea como un testimonio del avance entre la innovación tecnológica y la vida misma, abriendo las puertas a la automatización accesible de bajo costo para transformar nuestros hogares en entornos inteligentes y modernos.

CAPÍTULO I

Objetivos

Objetivo general

El objetivo principal de este proyecto es diseñar e implementar un sistema embebido de bajo costo para la automatización del hogar, centrándose en el control de la iluminación, la apertura de espacios, la seguridad y la jardinería. Este sistema estará orientado a la simplicidad y accesibilidad para aficionados a la domótica.

Objetivos específicos

Implementar el encendido y apagado de luces en la planta baja mediante comandos desde una terminal virtual.

Extender la funcionalidad al control de la iluminación en la planta alta y el patio.

Integrar un sistema de alarma representado por un LED rojo, activado y desactivado a través de la terminal virtual.

Automatizar la apertura y cierre de la cochera mediante un motor representado por un relé y motor.

Implementar un mecanismo similar para el control del portón.

Desarrollar un sistema de riego representado por un relé y motor, activado y desactivado desde la terminal virtual.

Permitir al usuario interactuar con el sistema a través de una interfaz amigable en una terminal virtual.

Mostrar mensajes informativos después de cada acción realizada para una retroalimentación efectiva.

La realización de este proyecto busca proporcionar a los entusiastas de la domótica una solución práctica y asequible para la automatización de sus hogares. Además, se pretende explorar y aplicar conceptos de sistemas embebidos de manera práctica, incorporando tecnologías de bajo costo para maximizar la accesibilidad.

Este proyecto se centrará en la simulación y programación del sistema embebido, utilizando una plataforma específica proporcionada en la plataforma Moodle. El enfoque estará en la funcionalidad y facilidad de uso del sistema, sin considerar aspectos de implementación hardware.

CAPÍTULO II

Marco teórico

Arduino Uno

El Arduino Uno es una placa de microcontrolador de código abierto diseñada para prototipado y desarrollo de proyectos electrónicos. Desarrollada por Arduino LLC, la placa utiliza el microchip ATmega328P y ofrece una variedad de pines de entrada/salida digital y analógica para conectar periféricos y circuitos adicionales. (colaboradores de Hellbot, 2023)

Origen y Evolución

El proyecto Arduino se originó en el Interaction Design Institute Ivrea (IDII) en Italia como una solución asequible para estudiantes que utilizaban microcontroladores BASIC Stamp. Inicialmente, la plataforma Wiring, creada por Hernando Barragán, sentó las bases. Posteriormente, Massimo Banzi y su equipo bifurcaron el proyecto, lo renombraron como Arduino y añadieron soporte para microcontroladores ATmega8 y ATmega168. El Arduino Uno, lanzado como parte de la serie basada en USB, introdujo el ATmega328P y el ATmega16U2 como convertidor USB a serie.

Características Técnicas

El Arduino Uno presenta las siguientes especificaciones técnicas:

Microcontrolador: Microchip ATmega328P

Voltaje de Funcionamiento: 5 voltios

Voltaje de Entrada: 7 a 20 voltios

Pines de E/S Digitales: 14 (6 de ellos con salida PWM)

Pines de Entrada Analógica: 6

Memoria Flash: 32 KB

Velocidad del Reloj: 16 MHz

Dimensiones: 68.6mm x 53.4mm

Peso: 25g

Funciones y Conectividad

Pines Generales

LED: Controlado por el pin digital 13, proporciona retroalimentación visual.

VIN: Voltaje de entrada desde una fuente externa.

5V: Suministro de voltaje regulado.

3V3: Suministro de 3.3 voltios.

GND: Pines de tierra.

IOREF: Proporciona el voltaje de referencia para el microcontrolador.

Funciones Especiales de Pin

Cada uno de los 14 pines digitales y 6 pines analógicos puede ser configurado como entrada/salida. Algunos pines tienen funciones especializadas, como PWM, interrupciones externas, y soporte para comunicación serie (UART), SPI, y TWI/I²C.

Comunicación y Reinicio Automático

El Arduino Uno se comunica con una computadora a través de UART TTL y USB. Utiliza un ATmega16U2 para canalizar la comunicación serie a través de USB. El reinicio automático se logra mediante un software que activa la línea de reinicio del microcontrolador.

Aplicaciones

El Arduino Uno se utiliza ampliamente en prototipado rápido, desarrollo de proyectos educativos y la creación de dispositivos interactivos. Su versatilidad y facilidad de uso lo convierten en una herramienta fundamental en el campo de la electrónica y la programación.

Conclusiones del Marco Teórico

El Arduino Uno, como componente clave en sistemas embebidos, proporciona una plataforma robusta y accesible para el desarrollo de proyectos electrónicos. Su evolución desde los primeros días hasta el presente refleja su impacto en la comunidad de desarrolladores y estudiantes de electrónica y programación. (colaboradores de Wikipedia, 2023)

CAPÍTULO III

Procedimiento

El desarrollo del sistema embebido para la automatización del hogar se llevó a cabo siguiendo un procedimiento detallado. A continuación, se describen las etapas clave del proceso de implementación:

Configuración Inicial

Se establecieron los pines correspondientes en la placa Arduino Uno para los LEDs y los motores, ver figura 1.

Se implementó la comunicación serial para recibir comandos desde la terminal virtual, ver figura 1.

Inicialización del Sistema

Se programó una secuencia de bienvenida al iniciar el sistema, mostrando mensajes como "Bienvenido" e "Inicializando el sistema".

Se incorporaron retardos (delays) para proporcionar una experiencia de usuario más amigable, ver figura 1.

```
// Definimos los pines para los LEDs y los motores
const int plantaBajaLED = 13;
const int plantaAltaLED = 12;
const int patioLED = 11;
const int alarmaLED = 8;
const int motorCochera = 10;
const int motorPorton = 9;
const int motorRiego = 7;

void setup() {
  // Inicializamos la comunicación serial
  Serial.begin(9600);

  // Inicializamos los pines de salida para LEDs y motores
  pinMode(plantaBajaLED, OUTPUT);
  pinMode(plantaAltaLED, OUTPUT);
  pinMode(patioLED, OUTPUT);
  pinMode(alarmaLED, OUTPUT);
  pinMode(motorCochera, OUTPUT);
  pinMode(motorPorton, OUTPUT);
  pinMode(motorRiego, OUTPUT);

  // Mensajes de inicio
  Serial.println("Bienvenido");
  delay(200);
  Serial.println("Inicializando el sistema");
  delay(200);
  Serial.println("Espere....");
  delay(2000);
  Serial.println("Sistema activo");
}

void loop() {
  // Leemos la entrada serial(lo que ingrese el usuario)
  if (Serial.available()) {
    char comando = Serial.read();
    // Convertimos a minúscula lo ingresado por el usuario antes de ejecutar el comando (con esto manejamos que sea indiferente si el caracter ingresado es una mayuscula)
    char comandoMinuscula = tolower(comando);
    ejecutarComando(comandoMinuscula);
  }
```

Figura 1: captura de código Arduino con configuración inicial e inicialización del Sistema.

Control de Iluminación

Se implementó el encendido y apagado de luces en la planta baja (LEDs amarillos) mediante las teclas A y B en la terminal virtual.

Se replicó el proceso para la iluminación de la planta alta (LEDs morados) y el patio (LEDs aqua).

Se creó una función “encenderLED” que recibe como parámetros el numero de pin de salida, y un mensaje para mostrar y otra función “apagarLED” que recibe los mismos parámetros que la anterior para implementar estas funcionalidades, ver figura 2.

```
// Realizamos funciones para encender y apagar los leds
// Esto nos permite una mejor lectura del código de programación
// Tuvimos que considerar el numero de pin de cada led y el comando elejido por el usuario, para encender las luces de la zona correcta(el comando que ingresa el usuario)
void encenderLED(int pinLED, const char* mensaje) {
  digitalWrite(pinLED, HIGH);
  Serial.println(mensaje);
}
// Funcion para apagar los leds
void apagarLED(int pinLED, const char* mensaje) {
  digitalWrite(pinLED, LOW);
  Serial.println(mensaje);
}
```

Figura 2: código Arduino funciones para encender y apagar LED’s o alarmas.

Sistema de Alarma

Se integró un sistema de alarma representado por un LED rojo, activado mediante el número 1 y desactivado con la tecla 0.

Control de Espacios

Se automatizó la apertura y cierre de la cochera utilizando un motor (relé y motor) que se enciende y apaga con los comandos 2 y 3 respectivamente, tardando un tiempo de dos segundos para abrir o cerrar la puerta de cochera, ver figura 3.

Se implementó un mecanismo similar para el control del portón, el cúal es activado con la tecla número 4 y desactivado con la tecla 5, tardando un tiempo de dos segundos para abrir o cerrar el portón. Para estas últimas se implementó una función llamada “ejecutarMotor” encargada de encender el motor para que se abra o cierre la cochera o portón, ver figura 3.

Automatización de Jardinería

Se desarrolló un sistema de riego representado por un relé y motor. Se activa con el número 6 y se desactiva con la tecla 7, para esta funcionalidad se implementaron dos funciones encargadas de encender y apagar el motor ver figura.

No se estableció un tiempo de encendido específico, ya que en el sistema simulado no se requería un tiempo de ejecución, sino que se implementó con la lógica de que el sistema de riego se encienda y se apague manualmente por el usuario, ver figura 3.

```arduino
// Realizamos una función para ejecutar los motores de porton y cochera
// Esto lo pudimos lograr teniendo en cuenta el numero de pin del motor, para encender el motor correcto(el comando que ingresa el usuario)
void ejecutarMotor(int pinMotor, const char* mensajeInicio, const char* mensajeFin) {

  Serial.println(mensajeInicio);
  digitalWrite(pinMotor, HIGH);
  // Simulamos el tiempo de movimiento del motor (2 segundos)
  delay(2000);
  // Apagamos el motor
  digitalWrite(pinMotor, LOW);
  Serial.println(mensajeFin);
}

void prenderMotor(int pinMotor, const char* mensajeInicio, const char* mensajeFin) {
  Serial.println(mensajeInicio);
  digitalWrite(pinMotor, HIGH);
  Serial.println(mensajeFin);
}

void apagarMotor(int pinMotor, const char* mensajeInicio, const char* mensajeFin) {
  Serial.println(mensajeInicio);
  digitalWrite(pinMotor, LOW);
  Serial.println(mensajeFin);
}
```

Figura 3: código Arduino para funcionalidad de cochera, portón y sistema de riego.

Se utilizó un Switch Case para tener en cuenta los comandos ingresados por el usuario, y también se contemplaron caracteres que no tengan funcionalidad, indicando un mensaje “comando no reconocido”, ver figura 4.

```arduino
void ejecutarComando(char comando) {
  // Ejecutamos la acción correspondiente según el comando recibido para eso utilizamos un switch case
  switch (comando) {
    case 'a':
      encenderLED(plantaBajaLED, "Iluminacion de la planta baja encendida");
      break;
    case 'b':
      apagarLED(plantaBajaLED, "Iluminacion de la planta baja apagada");
      break;
    case 'c':
      encenderLED(plantaAltaLED, "Iluminacion de la planta alta encendida");
      break;
    case 'd':
      apagarLED(plantaAltaLED, "Iluminacion de la planta alta apagada");
      break;
    case 'e':
      encenderLED(patioLED, "Iluminacion del patio encendida");
      break;
    case 'f':
      apagarLED(patioLED, "Iluminacion del patio apagada");
      break;
    case '1':
      encenderLED(alarmaLED, "Alarma encendida");
      break;
    case '0':
      apagarLED(alarmaLED, "Alarma apagada");
      break;
    case '2':
      ejecutarMotor(motorCochera, "Aperturando cochera...", "Cochera aperturada");
      break;
    case '3':
      ejecutarMotor(motorCochera, "Cerrando cochera...", "Cochera cerrada");
      break;
    case '4':
      ejecutarMotor(motorPorton, "Aperturando porton...", "Porton aperturado");
      break;
    case '5':
      ejecutarMotor(motorPorton, "Cerrando porton...", "Porton cerrado");
      break;
    case '6':
      prenderMotor(motorRiego, "Encendiendo sistema de riego...", "Sistema de riego encendido");
      break;
    case '7':
      apagarMotor(motorRiego, "Apagando sistema de riego...", "Sistema de riego apagado");
      break;
    default:
      Serial.println("Comando no reconocido");
  }
```

Figura 4: código Arduino switch case para contemplar acción requerida por usuario.

Análisis de los resultados

Durante la implementación, se realizaron pruebas exhaustivas para verificar el correcto funcionamiento del sistema. Se observó que cada componente, desde la iluminación hasta el control de los motores, respondió según las especificaciones.

El uso de la comunicación serial permitió una interacción eficiente entre el usuario y el sistema. La retroalimentación en la terminal virtual proporcionó información clara sobre el estado de cada acción, como el encendido/apagado de luces, el movimiento de los motores y el estado de la alarma, ver figura 5 y 6.

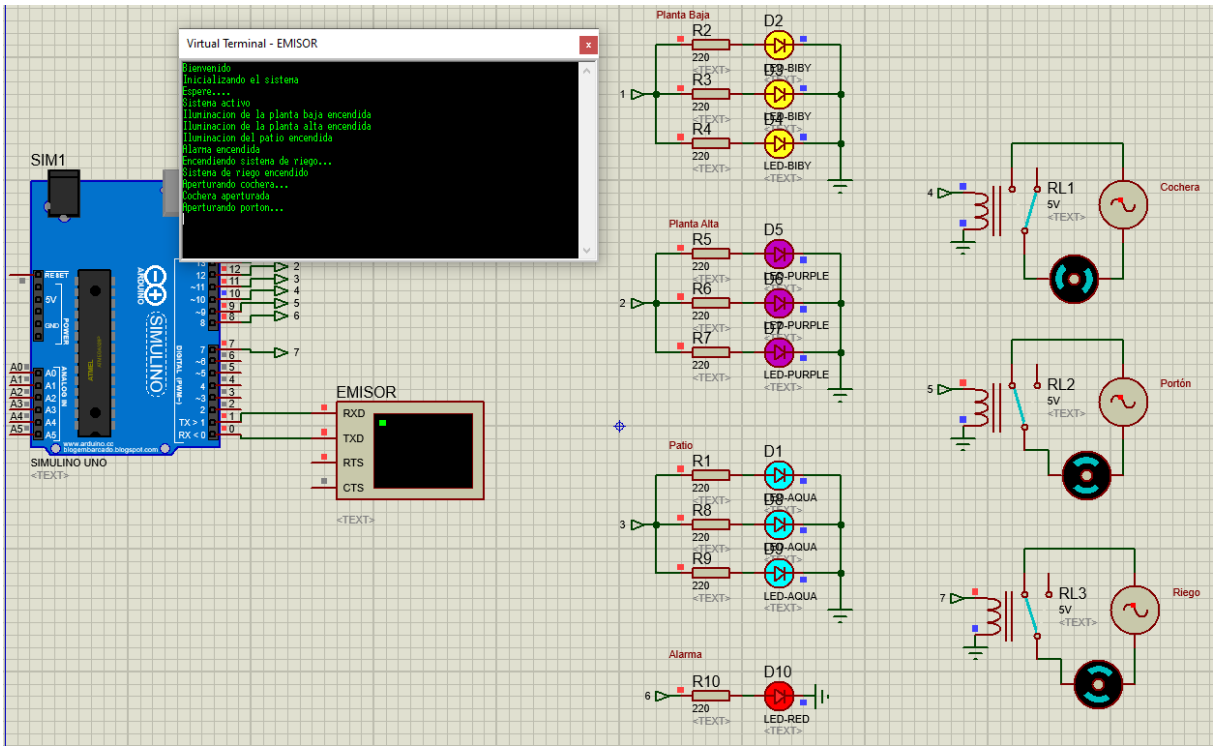

Figura 5: Análisis de los resultados en funcionamiento en Proteus.

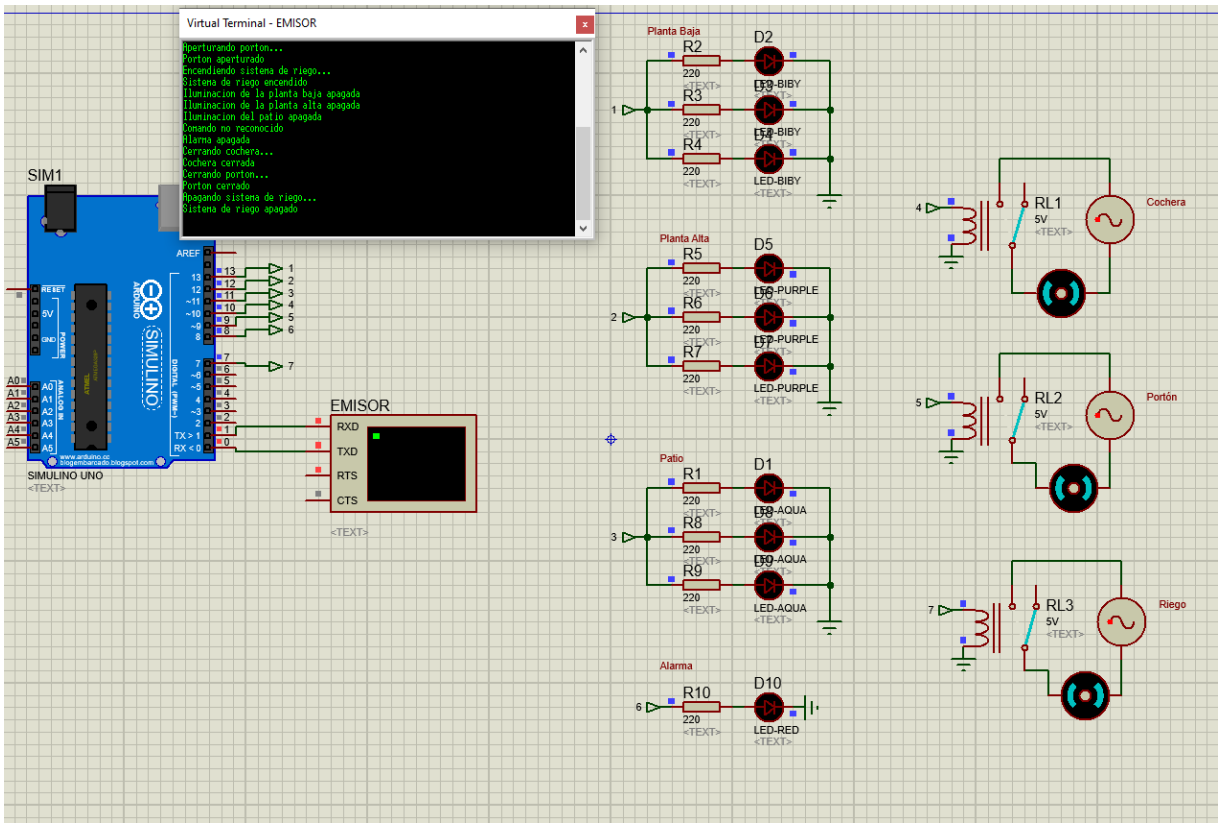

Figura 6: Análisis de los resultados apagando todo lo que estaba funcionando en Proteus.

El procedimiento diseñado logró la automatización efectiva de los aspectos del hogar especificados, cumpliendo con los objetivos planteados en el Capítulo I. Las funciones para encender y apagar LEDs, así como ejecutar los motores, demostraron ser consistentes y confiables durante las pruebas.

Este capítulo presenta el proceso de implementación detallado y el análisis de los resultados obtenidos durante la fase de desarrollo del sistema embebido para la automatización del hogar. En el próximo capítulo, se realizará una revisión de las conclusiones relacionadas con este sistema embebido, y recomendaciones o posibles mejoras para él mismo.

CAPÍTULO IV

Conclusiones y recomendaciones

Conclusiones

Logros del Proyecto

El proyecto logró su objetivo principal al diseñar e implementar un sistema embebido de bajo costo para la automatización del hogar.

Se alcanzó la funcionalidad esperada en el control de iluminación, seguridad, espacios y jardinería, demostrando la viabilidad y eficacia del sistema.

Simplicidad y Accesibilidad

El enfoque en la simplicidad y accesibilidad para aficionados a la domótica se reflejó en la interfaz de usuario y la facilidad de manejo a través de la terminal virtual.

Integración de Arduino Uno:

La elección del Arduino Uno como plataforma embebida demostró ser acertada, proporcionando las capacidades necesarias para la implementación de las funciones requeridas.

Comunicación Serial

La implementación de la comunicación serial facilitó la interacción eficiente entre el usuario y el sistema, permitiendo comandos claros y una retroalimentación efectiva.

Retroalimentación del Usuario

La retroalimentación clara después de cada acción realizada en la terminal virtual contribuyó a una experiencia de usuario mejorada.

Pruebas Exitosas

Las pruebas exhaustivas durante el desarrollo confirmaron el correcto funcionamiento de cada componente, desde la iluminación hasta el control de los motores.

Recomendaciones

Mejoras en la Interfaz

Considerar la mejora de la interfaz de usuario para una experiencia más intuitiva y visual, si es posible, en futuras versiones del sistema.

Optimización del Código

Realizar una revisión del código para identificar oportunidades de optimización y eficiencia en el uso de recursos de la placa Arduino.

Implementación Hardware

Evaluar la posibilidad de llevar a cabo la implementación hardware del sistema para validar su desempeño en un entorno físico.

Inclusión de Temporizadores

En el sistema de riego, considerar la inclusión de temporizadores para automatizar la duración de la irrigación, brindando una mayor autonomía al sistema.

Seguridad y Protección

Incorporar medidas de seguridad adicionales, como contraseñas o autenticación, para proteger el acceso no autorizado al sistema.

Exploración de Tecnologías Adicionales

Investigar nuevas tecnologías y protocolos de comunicación para expandir las capacidades del sistema y su integración con otros dispositivos de domótica.

Documentación Detallada

Elaborar una documentación detallada que incluya instrucciones de instalación, configuración y mantenimiento del sistema para facilitar su uso por parte de los usuarios finales.

Bibliografía

Colaboradores de Hellbot. (2023). *Introducción a hardware de ARDUINO UNO*. Hellbot. https://hellbot.xyz/introduccion-a-hardware-de-arduino-uno/

Colaboradores de Wikipedia. (2023). *Arduino uno*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Arduino_Uno

ANEXOS

Video de presentación

https://www.youtube.com/watch?v=ciw6c30Baqw&ab_channel=ivantapia

Código de programación de la placa Arduino Uno

```
#include <SoftwareSerial.h>

// Definimos los pines para los LEDs y los motores
const int plantaBajaLED = 13;
const int plantaAltaLED = 12;
const int patioLED = 11;
const int alarmaLED = 8;
const int motorCochera = 10;
const int motorPorton = 9;
const int motorRiego = 7;

void setup() {
  // Inicializamos la comunicación serial
  Serial.begin(9600);

  // Inicializamos los pines de salida para LEDs y motores
  pinMode(plantaBajaLED, OUTPUT);
  pinMode(plantaAltaLED, OUTPUT);
  pinMode(patioLED, OUTPUT);
  pinMode(alarmaLED, OUTPUT);
  pinMode(motorCochera, OUTPUT);
  pinMode(motorPorton, OUTPUT);
  pinMode(motorRiego, OUTPUT);

  // Mensajes de inicio
  Serial.println("Bienvenido");
  delay(200);
  Serial.println("Inicializando el sistema");
  delay(200);
  Serial.println("Espere....");
  delay(2000);
  Serial.println("Sistema activo");
}

void loop() {
  // Leemos la entrada serial(lo que ingrese el usuario)
  if (Serial.available()) {
    char comando = Serial.read();
```

```
    // Convertimos a minúscula lo ingresado por el usuario antes de ejecutar el comando (con esto manejamos que sea indiferente si el caracter ingresado es una mayuscula)
    char comandoMinuscula = tolower(comando);
    ejecutarComando(comandoMinuscula);
  }
}

void ejecutarComando(char comando) {
  // Ejecutamos la acción correspondiente según el comando recibido para eso utilizamos un switch case
  switch (comando) {
    case 'a':
      encenderLED(plantaBajaLED, "Iluminacion de la planta baja encendida");
      break;
    case 'b':
      apagarLED(plantaBajaLED, "Iluminacion de la planta baja apagada");
      break;
    case 'c':
      encenderLED(plantaAltaLED, "Iluminacion de la planta alta encendida");
      break;
    case 'd':
      apagarLED(plantaAltaLED, "Iluminacion de la planta alta apagada");
      break;
    case 'e':
      encenderLED(patioLED, "Iluminacion del patio encendida");
      break;
    case 'f':
      apagarLED(patioLED, "Iluminacion del patio apagada");
      break;
    case '1':
      encenderLED(alarmaLED, "Alarma encendida");
      break;
    case '0':
      apagarLED(alarmaLED, "Alarma apagada");
      break;
    case '2':
      ejecutarMotor(motorCochera, "Aperturando cochera...", "Cochera aperturada");
      break;
    case '3':
      ejecutarMotor(motorCochera, "Cerrando cochera...", "Cochera cerrada");
      break;
    case '4':
      ejecutarMotor(motorPorton, "Aperturando porton...", "Porton aperturado");
      break;
    case '5':
      ejecutarMotor(motorPorton, "Cerrando porton...", "Porton cerrado");
```

```
      break;
    case '6':
      prenderMotor(motorRiego, "Encendiendo sistema de riego...", "Sistema de
riego encendido");
      break;
    case '7':
      apagarMotor(motorRiego, "Apagando sistema de riego...", "Sistema de riego
apagado");
      break;
    default:
      Serial.println("Comando no reconocido");
  }
}

// Realizamos funciones para encender y apagar los leds
// Esto nos permite una mejor lectura del código de programación
// Tuvimos que considerar el numero de pin de cada led y el comando elejido por
el usuario, para encender las luces de la zona correcta(el comando que ingresa
el usuario)
void encenderLED(int pinLED, const char* mensaje) {
  digitalWrite(pinLED, HIGH);
  Serial.println(mensaje);
}
// Funcion para apagar los leds
void apagarLED(int pinLED, const char* mensaje) {
  digitalWrite(pinLED, LOW);
  Serial.println(mensaje);
}
// Realizamos una función para ejecutar los motores de porton y cochera
// Esto lo pudimos lograr teniendo en cuenta el numero de pin del motor, para
encender el motor correcto(el comando que ingresa el usuario)
void ejecutarMotor(int pinMotor, const char* mensajeInicio, const char*
mensajeFin) {

  Serial.println(mensajeInicio);
  digitalWrite(pinMotor, HIGH);
  // Simulamos el tiempo de movimiento del motor (2 segundos)
  delay(2000);
  // Apagamos el motor
  digitalWrite(pinMotor, LOW);
  Serial.println(mensajeFin);
}

void prenderMotor(int pinMotor, const char* mensajeInicio, const char*
mensajeFin) {
  Serial.println(mensajeInicio);
  digitalWrite(pinMotor, HIGH);
  Serial.println(mensajeFin);
```

```
}

void apagarMotor(int pinMotor, const char* mensajeInicio, const char*
mensajeFin) {
  Serial.println(mensajeInicio);
  digitalWrite(pinMotor, LOW);
  Serial.println(mensajeFin);
}
```