Nombre del proyecto	Sistema de Riego Automatico
Acrónimo del nombre de proyecto	SRA
Versión	1.0.1
Autores	38145, Alejandra Dominguez Murua 37561, Alicia Michelle Jordan Felix, 33628, Alan Isaac López Martínez
Fecha	02:06:2025

I. Descripción del problema

En la actualidad, existe una creciente demanda por el acceso a alimentos frescos y saludables, así como el deseo de las personas de cultivar sus propios vegetales en casa. Sin embargo, muchas personas enfrentan barreras significativas para mantener huertos domésticos exitosos, especialmente relacionadas con la gestión adecuada del riego. Los métodos tradicionales de riego manual requieren conocimiento especializado, tiempo constante y supervisión diaria, lo que resulta inaccesible para personas con horarios ocupados, falta de experiencia en jardinería, o quienes viajan frecuentemente. El problema se intensifica cuando consideramos que el riego inadecuado es una de las principales causas de fracaso en huertos domésticos. Un riego excesivo puede causar pudrición de raíces y enfermedades fúngicas, mientras que un riego insuficiente provoca estrés hídrico y muerte de las plantas. Esta incertidumbre sobre cuándo y cuánto regar desalienta a muchas personas de intentar cultivar sus propios alimentos, perpetuando la dependencia de productos comerciales y limitando el acceso a vegetales frescos y nutritivos.

La falta de un sistema automatizado accesible y económico impide que las familias puedan establecer huertos domésticos sostenibles. Esto es particularmente problemático en comunidades urbanas donde el espacio es limitado y las personas buscan maximizar la productividad de pequeños espacios de cultivo como balcones, patios o jardines verticales. Además, la gestión manual del riego durante vacaciones o ausencias prolongadas representa un obstáculo adicional para mantener cultivos domésticos. La implementación de un sistema de riego automatizado de bajo costo y fácil instalación democratiza el acceso a la agricultura doméstica. Este sistema utiliza sensores de humedad del suelo para determinar automáticamente cuándo las plantas necesitan agua, eliminando las conjeturas y permitiendo que cualquier persona, independientemente de su experiencia previa en jardinería, pueda cultivar exitosamente sus propios vegetales.

El sistema es especialmente valioso para facilitar que las familias puedan producir alimentos frescos en casa, reduciendo costos de alimentación y mejorando la seguridad alimentaria doméstica. Diversos proyectos de código abierto y tutoriales han demostrado la viabilidad técnica y económica de sistemas de riego automatizados para huertos domésticos, evidenciando cómo la tecnología accesible puede empoderar a las personas para cultivar sus

propios alimentos [1], [2], [3]. Estas soluciones han mostrado particular éxito en entornos urbanos y suburbanos donde las personas buscan mayor autosuficiencia alimentaria [4], [5].

II. Historia principal

En el contexto urbano, un jardinero aficionado que cultiva un pequeño huerto de bugambilias en su casa se enfrenta al reto de mantener un riego adecuado sin la necesidad de supervisión constante. La falta de un sistema automatizado lleva a un uso ineficiente del recurso hídrico y a la incertidumbre sobre el estado real del suelo. Para resolver este inconveniente, se plantea el desarrollo de un sistema de irrigación a escala reducida que, utilizando sensores de humedad, determine de forma precisa el momento oportuno para regar.

En esta solución, un microcontrolador se encarga del control en tiempo real de los sensores y actuadores, mientras que una STM32 permite la visualización remota y el registro histórico de los datos, facilitando así la gestión y el monitoreo del huerto. Este sistema, de bajo consumo y fácil instalación, asegura que las plantas reciban la cantidad exacta de agua necesaria, promoviendo un uso racional del recurso y reduciendo el mantenimiento manual. la propuesta se alinea con las tendencias actuales de agricultura urbana y sostenibilidad, respaldadas por estudios y experiencias documentadas en proyectos similares

III. Sub-historias

Desear: María y muchas familias desean cultivar sus propios vegetales frescos en casa.

Intentar: Han intentado múltiples veces mantener huertos domésticos sin éxito.

Fallar: Los intentos fallan por falta de tiempo, conocimiento, y cuidado inconsistente.

Frustrar: La frustración crece al desperdiciar dinero en plantas que mueren.

Necesitar: Se necesita una solución que no requiera supervisión constante.

Automatizar: Un sistema automatizado elimina la necesidad de recordar cuándo regar.

Monitorear: El sistema monitorea continuamente la humedad del suelo.

Decidir: El microcontrolador decide automáticamente cuándo activar el riego.

Regar: La válvula automática riega las plantas en el momento y cantidad correctos.

Conectar: La Raspberry Pi conecta el sistema a internet para monitoreo remoto.

Empoderar: El sistema empodera a cualquier persona para cultivar exitosamente vegetales.

Democratizar: La tecnología democratiza el acceso a la agricultura doméstica.

Requerimientos de hardware

Component	Quantity	Characteristics	Component
STM32 F767ZI	1	Development board	
HW080	1	8-bit Serial In Parallel Out (S range from 2 V to 6 V. Low power consumption. 1 uA maximum input current	

Hardware esquemático

El sistema utiliza el microcontrolador STM32F4 como unidad central de control, conectado a sensores de humedad del suelo y actuadores de riego. La Raspberry Pi actúa como interfaz de usuario y sistema de monitoreo remoto. Conexiones principales STM32F4:

PA0: Sensor de humedad del suelo (entrada analógica) PA5: Control de relay/válvula solenoide (salida digital) PA2/PA3: Comunicación UART con Raspberry Pi PC13: Botón de usuario (entrada digital)

Diseño de hardware Conexiones del STM32F4 Sensor de Humedad (PA0):

Entrada analógica para lectura del nivel de humedad del suelo Rango de operación: 0-3.3V Umbral de activación configurable

Control de Relay (PA5):

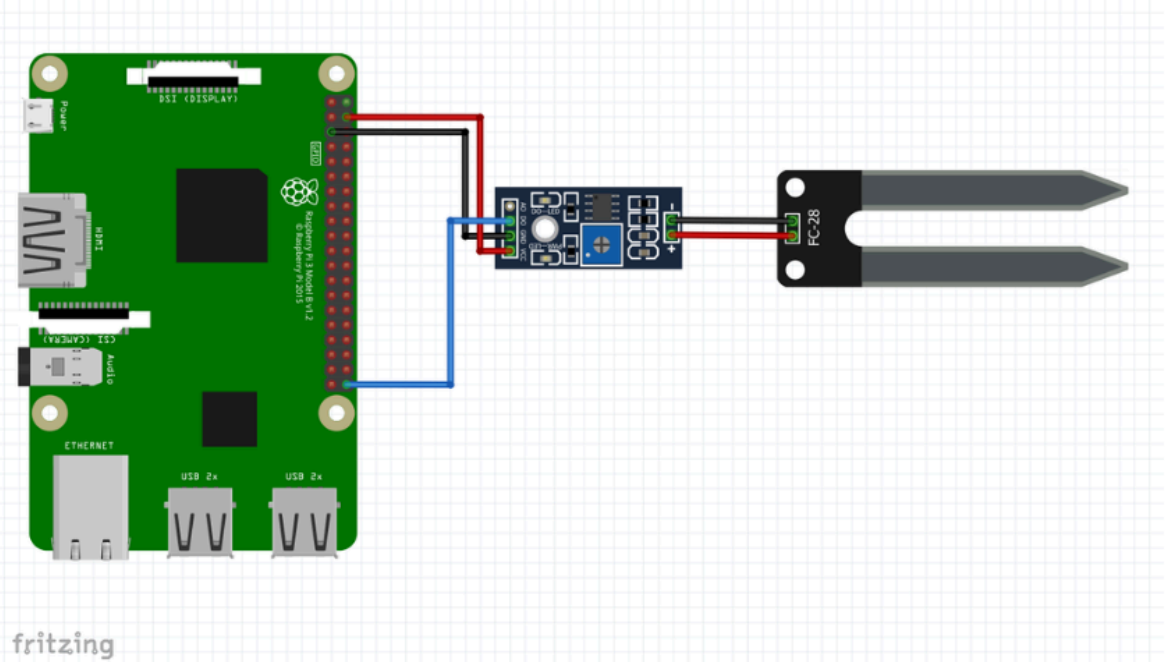
Salida digital para activación de válvula solenoide Voltaje de salida: 3.3V Corriente máxima: 25mA

Comunicación UART (PA2/PA3):

TX: PA2 - Transmisión de datos a Raspberry Pi RX: PA3 - Recepción de comandos desde Raspberry Pi Baudrate: 115200 bps

Botón de Usuario (PC13):

Entrada digital con pull-up interno Función de control manual y configuración



Diseño de hardware

Componente	Cantidad	Características
Sensor humedad HW080	1	Sensor capacitivo de humedad
Válvula Solenoide	1	3/4” 12V DC control de flujo
Módulo Relay	1	Control de válvula solenoide
LED indicador	3	Indicadores de estado

ID	Tipo	Nombre	Función	Prioridad	Puerto
1	Periodic	sensorTask	sensorTaskFunction	Normal	PA0
2	Periodic	irrigationTask	irrigationTaskFunction	High	PA5
3	Persistent	commTask	commTaskFunction	Low	PA2/PA3
4	Interrupt	buttonISR	HAL_GPIO_EXTI	0	PC13

Pin	Función	Descripción
PA0	Sensor Humedad	Entrada analógica 0-3.3V
PA5	Control Relay	Salida digital 3.3V, 25mA máx
PA2	UART TX	Transmisión 115200 bps
PA3	UART RX	Recepción 115200 bps
PC13	Botón Usuario	Entrada con pull-up interno

Requisitos del sistema operativo y carga de trabajo

Clase	Atributo	Valor
Proyecto	Nombre	SRA_STM32F4_V1
	Clock	84MHz
	Timebase source	SysTick
STM32 Pins	PA0	Sensor_Humedad (ADC)
	PA5	Relay_Control (GPIO_Output)
	PA2/PA3	UART2_TX/RX
	PC13	Button_User (GPIO_Input)
FreeRTOS	API	CMSIS v2
	Scheduler	Preemptive

Esquematicos

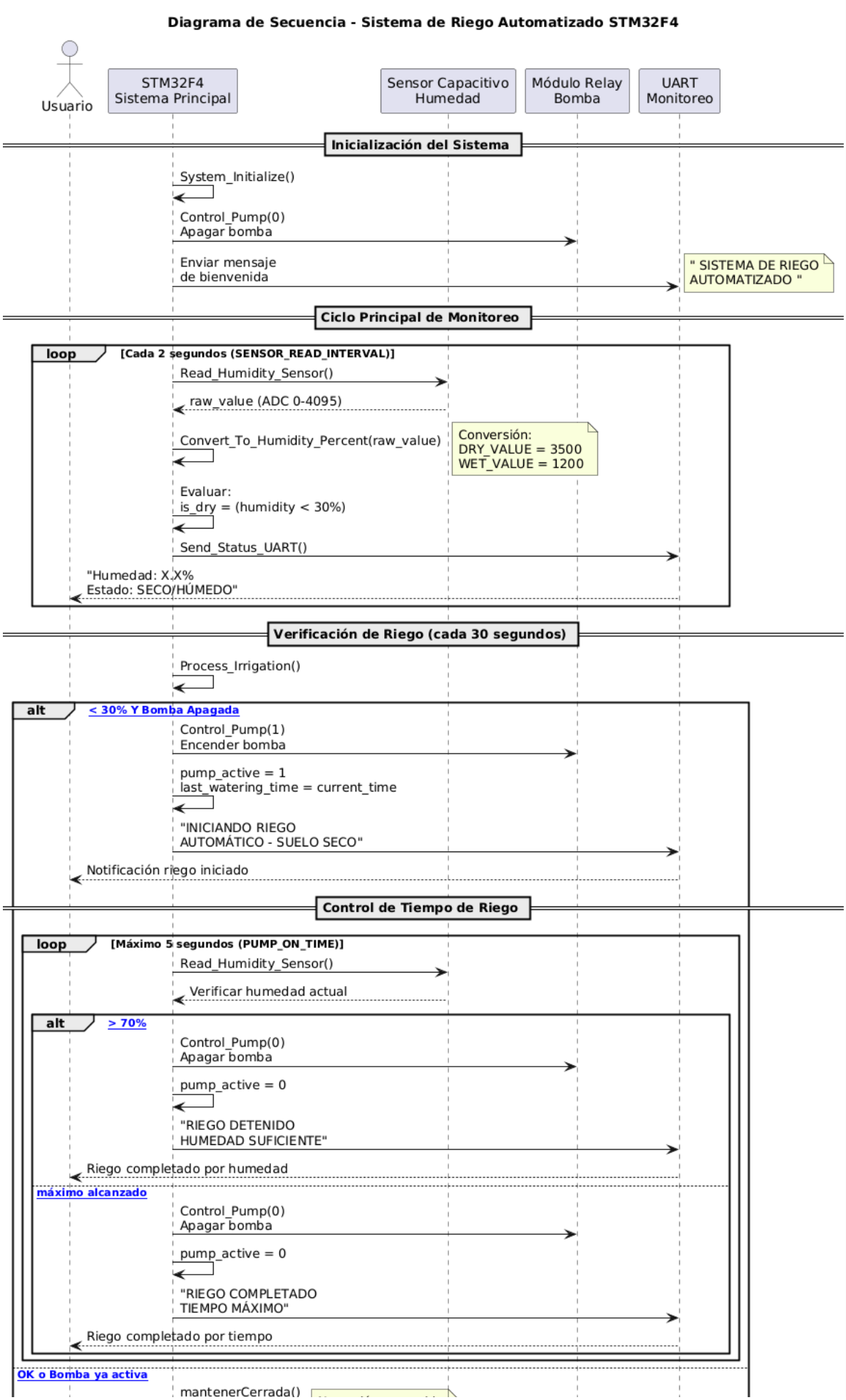
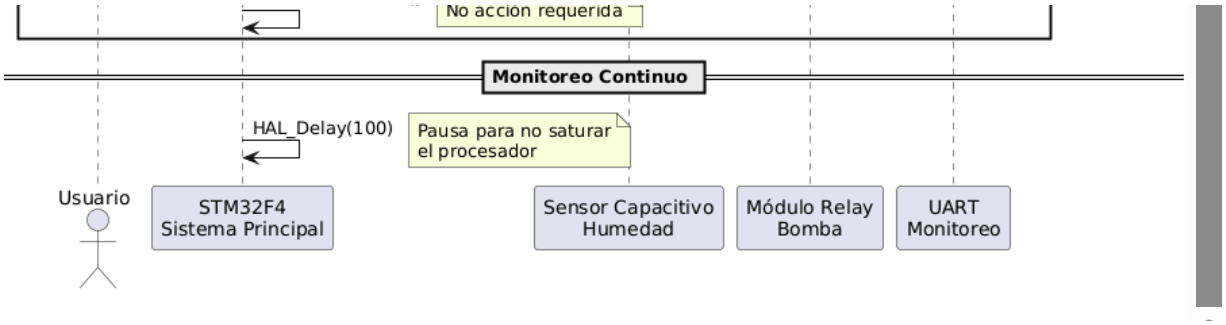
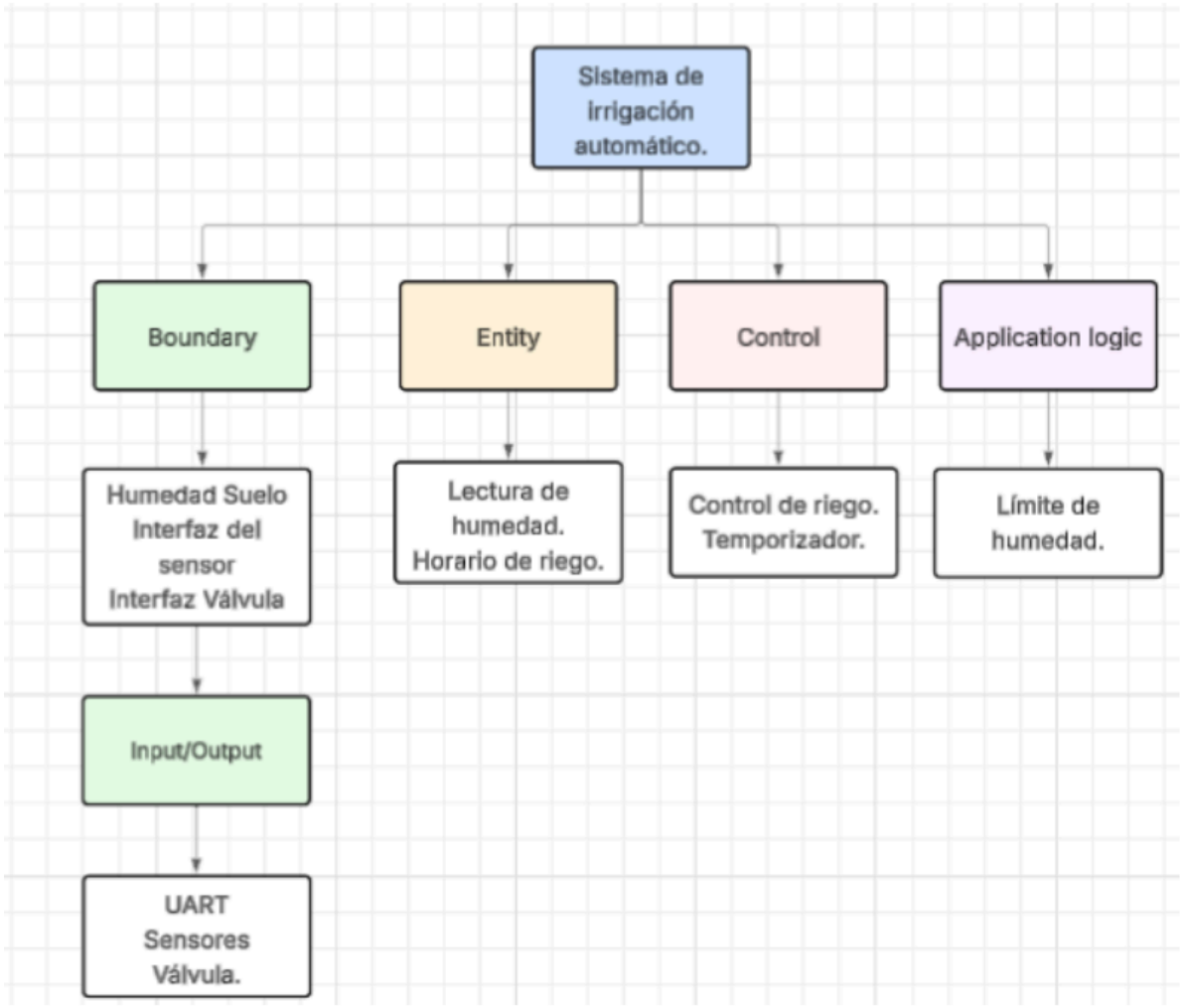


Diagrama de Secuencia - Sistema de Riego Automatizado STM32F4
Usuario
STM32F4 Sistema Principal
Sensor Capacitivo Humedad
Módulo Relay Bomba
UART Monitoreo
Inicialización del Sistema
System_Initialize()
Control_Pump(0)
Apagar bomba
Enviar mensaje de bienvenida
" SISTEMA DE RIEGO AUTOMATIZADO "
Ciclo Principal de Monitoreo
loop
[Cada 2 segundos (SENSOR_READ_INTERVAL)]
Read_Humidity_Sensor()
raw_value (ADC 0-4095)
Convert_To_Humidity_Percent(raw_value)
Conversión:
DRY_VALUE = 3500
WET_VALUE = 1200
Evaluar:
is_dry = (humidity < 30%)
Send_Status_UART()
"Humedad: X.X%
Estado: SECO/HÚMEDO"
Verificación de Riego (cada 30 segundos)
Process_Irrigation()
alt
< 30% Y Bomba Apagada
Control_Pump(1)
Encender bomba
pump_active = 1
last_watering_time = current_time
"INICIANDO RIEGO AUTOMÁTICO - SUELO SECO"
Notificación riego iniciado
Control de Tiempo de Riego
loop
[Máximo 5 segundos (PUMP_ON_TIME)]
Read_Humidity_Sensor()
Verificar humedad actual
alt
> 70%
Control_Pump(0)
Apagar bomba
pump_active = 0
"RIEGO DETENIDO HUMEDAD SUFICIENTE"
Riego completado por humedad
máximo alcanzado
Control_Pump(0)
Apagar bomba
pump_active = 0
"RIEGO COMPLETADO TIEMPO MÁXIMO"
Riego completado por tiempo
OK o Bomba ya activa
mantenerCerrada()



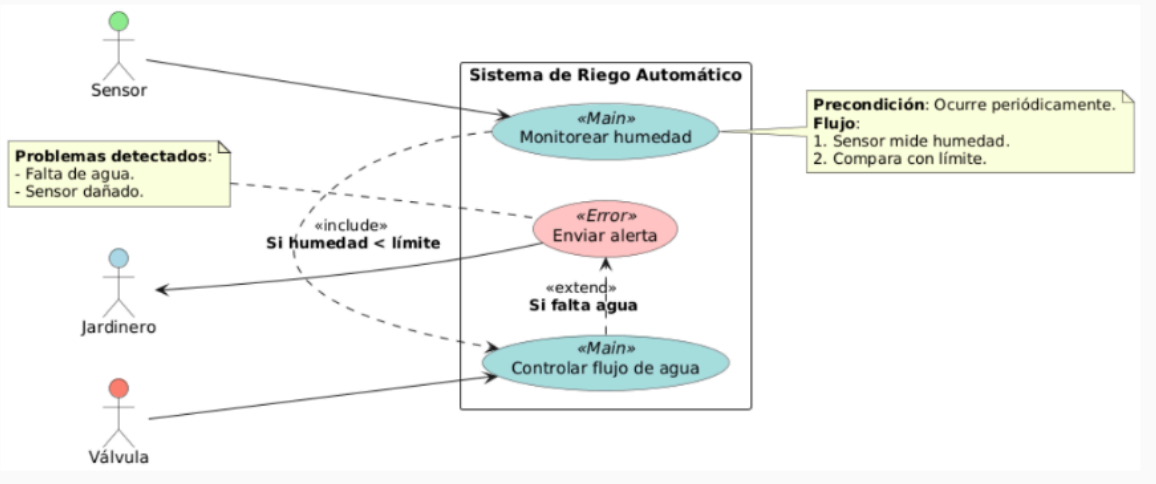


Casos de uso

Atributo	**Valor**
Nombre	Monitorear humedad
Resumen	Sensor comprueba si es necesario regar
Actores	Sensor de humedad
Precondición	Ocurre cada 2 segundos
Descripción	1. Sistema obtiene datos del sensor 2. Compara con límite (30\%) 3. Si humedad $< 30\%$, activa riego

Atributo	Valor
Nombre	Controlar flujo de agua
Resumen	Activa riego cuando humedad es baja
Actores	Válvula solenoide
Precondición	Humedad $< 30\%$
Descripción	1. Abre válvula por máximo 5s 2. Detiene si humedad $> 70\%$ 3. Envía notificaciones UART

Atributo	Valor
Nombre	Enviar alerta
Resumen	Notifica problemas al jardinero
Actores	Jardinero
Precondición	Problema en el sistema
Descripción	1. Detecta sensor dañado 2. Notifica vía UART/Raspberry Pi 3. Muestra estado del sistema



Componentes de software

Parámetro	Valor	Descripción
HUMIDITY_THRESHOLD_LOW	30%	Umbral bajo (inicia riego)
HUMIDITY_THRESHOLD_HIGH	70%	Umbral alto (detiene riego)
PUMP_ON_TIME	5000 ms	Tiempo máximo de riego
SENSOR_READ_INTERVAL	2000 ms	Intervalo de lectura
IRRIGATION_CHECK	30000 ms	Verificación cada 30s
DRY_VALUE	3500	ADC suelo seco
WET_VALUE	1200	ADC suelo húmedo

In [ ]:
```
/* USER CODE BEGIN Header */
/**
```

```
  ******************************************************************************
  * @file           : main.c
  * @brief          : Sistema de Riego Automatizado STM32F4
  ******************************************************************************
  * @attention
  *
  * Sistema de riego inteligente con sensor capacitivo de humedad
  * Componentes: STM32F4, sensor capacitivo, módulo relay, bomba de agua
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdio.h>
#include <string.h>
/* USER CODE END Includes */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
// Configuración del sistema de riego
#define HUMIDITY_THRESHOLD_LOW   30    // Umbral bajo de humedad (%)
#define HUMIDITY_THRESHOLD_HIGH  70    // Umbral alto de humedad (%)
#define PUMP_ON_TIME            5000   // Tiempo de riego en ms
#define SENSOR_READ_INTERVAL    2000   // Intervalo de lectura en ms
#define IRRIGATION_CHECK_INTERVAL 30000 // Verificar riego cada 30s
/* USER CODE END PD */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;
UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
// Estructura del sistema de riego
typedef struct {
    uint16_t raw_value;
    float humidity_percent;
    uint8_t is_dry;
    uint8_t pump_active;
    uint32_t last_watering_time;
    uint32_t sensor_read_time;
    uint32_t last_irrigation_check;
} IrrigationSystem_t;

IrrigationSystem_t irrigation;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_USART2_UART_Init(void);

/* USER CODE BEGIN PFP */
// Prototipos de funciones del sistema de riego
uint16_t Read_Humidity_Sensor(void);
```

```
float Convert_To_Humidity_Percent(uint16_t raw_value);
void Control_Pump(uint8_t state);
void Send_Status_UART(void);
void Process_Irrigation(void);
void System_Initialize(void);
/* USER CODE END PFP */

/**
 * @brief  Lee el valor del sensor de humedad capacitivo
 * @retval Valor ADC raw (0-4095)
 */
uint16_t Read_Humidity_Sensor(void)
{
    HAL_ADC_Start(&hadc1);

    if (HAL_ADC_PollForConversion(&hadc1, 100) == HAL_OK) {
        uint16_t adc_value = HAL_ADC_GetValue(&hadc1);
        HAL_ADC_Stop(&hadc1);
        return adc_value;
    }

    HAL_ADC_Stop(&hadc1);
    return 0; // Error en lectura
}

/**
 * @brief  Convierte valor ADC a porcentaje de humedad
 * @param  raw_value: Valor ADC crudo
 * @retval Porcentaje de humedad (0-100%)
 */
float Convert_To_Humidity_Percent(uint16_t raw_value)
{
    // Calibración para sensor capacitivo típico
    // AJUSTAR ESTOS VALORES SEGÚN TU SENSOR ESPECÍFICO
    const uint16_t DRY_VALUE = 3500;   // Valor en suelo completamente seco
    const uint16_t WET_VALUE = 1200;   // Valor en suelo completamente húmedo

    float humidity;

    if (raw_value >= DRY_VALUE) {
        humidity = 0.0;   // Completamente seco
    } else if (raw_value <= WET_VALUE) {
        humidity = 100.0; // Completamente húmedo
    } else {
        // Conversión lineal invertida (mayor ADC = menor humedad)
        humidity = 100.0 - ((float)(raw_value - WET_VALUE) / (DRY_VALUE - WET_VALUE))
    }

    // Limitar valores entre 0 y 100
    if (humidity < 0.0) humidity = 0.0;
    if (humidity > 100.0) humidity = 100.0;

    return humidity;
}

/**
 * @brief  Controla el estado de la bomba de agua
 * @param  state: 1 = Encender bomba, 0 = Apagar bomba
 */
void Control_Pump(uint8_t state)
```

```
{
    if (state) {
        HAL_GPIO_WritePin(RELAY_PIN_GPIO_Port, RELAY_PIN_Pin, GPIO_PIN_SET);
    } else {
        HAL_GPIO_WritePin(RELAY_PIN_GPIO_Port, RELAY_PIN_Pin, GPIO_PIN_RESET);
    }
}

/**
 * @brief  Envía el estado del sistema por UART
 */
void Send_Status_UART(void)
{
    char buffer[300];
    uint32_t current_time = HAL_GetTick();

    // Formatear mensaje de estado
    int len = snprintf(buffer, sizeof(buffer),
        "\r\n=== SISTEMA DE RIEGO AUTOMATIZADO ===\r\n"
        "Tiempo Sistema: %lu ms\r\n"
        "Valor ADC Raw: %d\r\n"
        "Humedad Suelo: %.1f%%\r\n"
        "Estado Suelo: %s\r\n"
        "Estado Bomba: %s\r\n"
        "Último Riego: %lu ms atrás\r\n"
        "====================================\r\n\r\n",
        current_time,
        irrigation.raw_value,
        irrigation.humidity_percent,
        irrigation.is_dry ? "🔴 SECO - NECESITA RIEGO" : "🟢 HÚMEDO - OK",
        irrigation.pump_active ? "💧 ENCENDIDA" : "⭕ APAGADA",
        irrigation.last_watering_time > 0 ? (current_time - irrigation.last_watering_t
    );

    // Enviar por UART
    HAL_UART_Transmit(&huart2, (uint8_t*)buffer, len, 2000);
}

/**
 * @brief  Procesa la lógica de riego automático
 */
void Process_Irrigation(void)
{
    uint32_t current_time = HAL_GetTick();

    // Verificar si es tiempo de evaluar riego
    if (current_time - irrigation.last_irrigation_check >= IRRIGATION_CHECK_INTERVAL)
    {
        irrigation.last_irrigation_check = current_time;

        if (irrigation.is_dry && !irrigation.pump_active)
        {
            // Suelo seco y bomba apagada -> Iniciar riego
            Control_Pump(1);
            irrigation.pump_active = 1;
            irrigation.last_watering_time = current_time;

            char msg[] = "🚨 INICIANDO RIEGO AUTOMÁTICO - SUELO SECO 🚨\r\n";
            HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 1000);
        }
```

```
        else if (irrigation.humidity_percent > HUMIDITY_THRESHOLD_HIGH && irrigation.p
        {
            // Suficiente humedad y bomba encendida -> Detener riego
            Control_Pump(0);
            irrigation.pump_active = 0;

            char msg[] = "✅ RIEGO DETENIDO - HUMEDAD SUFICIENTE ✅\r\n";
            HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 1000);
        }
    }

    // Control de tiempo máximo de riego (seguridad)
    if (irrigation.pump_active &&
        (current_time - irrigation.last_watering_time >= PUMP_ON_TIME))
    {
        Control_Pump(0);
        irrigation.pump_active = 0;

        char msg[] = "⏰ RIEGO COMPLETADO - TIEMPO MÁXIMO ALCANZADO ⏰\r\n";
        HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 1000);
    }
}

int main(void)
{
  /* MCU Configuration--------------------------------------------------------*/
  HAL_Init();
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_USART2_UART_Init();

  // Inicializar sistema de riego
  System_Initialize();
  HAL_Delay(1000);

  /* Infinite loop */
  while (1)
  {
    uint32_t current_time = HAL_GetTick();

    // Leer sensor de humedad cada SENSOR_READ_INTERVAL
    if (current_time - irrigation.sensor_read_time >= SENSOR_READ_INTERVAL)
    {
      // Actualizar lecturas del sensor
      irrigation.raw_value = Read_Humidity_Sensor();
      irrigation.humidity_percent = Convert_To_Humidity_Percent(irrigation.raw_value);
      irrigation.is_dry = (irrigation.humidity_percent < HUMIDITY_THRESHOLD_LOW);
      irrigation.sensor_read_time = current_time;

      // Enviar estado por UART
      Send_Status_UART();
    }

    // Procesar lógica de riego
    Process_Irrigation();

    // Pequeña pausa para no saturar el procesador
```

```
    HAL_Delay(100);
  }
}
```

Lectura del registro de desplazamiento de entrada paralela/salida serie C4014BE

El siguiente código indica al registro de desplazamiento PISO que muestree las 8 entradas y las envíe en serie al pin PA3, donde se leen y almacenan en nuestra *cola de mensajes* previamente declarada. Si esta cola está llena, la función seguirá intentando almacenar el mismo mensaje hasta que tenga éxito, cediendo el control solicitado por el RTOS.

Esta función tiene **prioridad normal**, lo que significa que está en el mismo nivel de prioridad que nuestra función *escritura*.

```
void StartRead(void *argument)
{
  /* VARIABLE DECLARATION */
  uint8_t tmp_reading = 0;

  /* Infinite loop */
  for(;;)
  {

    MSGQUEUE_OBJ_t msg;
    /* P/S Control HIGH to read from parallel inputs */
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_3, GPIO_PIN_SET);
    osDelay(10);

    /* Read from inputs */
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
    osDelay(10);

    /* Reset pins to start serial transmission */
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0|GPIO_PIN_3, GPIO_PIN_RESET);

    /* Read all 8 inputs */
    for(int i = 0; i < 8; i++) {
      /* Shift msg to prepare for reading */
      msg.flags <<= 1;

      /* HAL_GPIO_WritePin(GPIOB, LD3_Pin, GPIO_PIN_SET); */
      /* Set the message content */
      tmp_reading = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_3);
      msg.flags |= tmp_reading;


      /* Shift to next input */
      HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
      osDelay(50);
      HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_RESET);
```

```
      osDelay(50);
    }

    /* Send message to queue. Wait forever, retrying, if queue is
full */
    /* HAL_GPIO_WritePin(GPIOB, LD3_Pin, GPIO_PIN_SET); */
    osMessageQueuePut(mid_MsgQueue, &msg, 0U, osWaitForever);
    /* HAL_GPIO_WritePin(GPIOB, LD3_Pin, GPIO_PIN_RESET); */

    osDelay(1000);
  }
}
```

Escritura en el registro de desplazamiento de entrada serie/salida paralela SN54HC595

La función `StartWrite` permanece inactiva mientras la *cola de mensajes* está vacía. Una vez que recibe un mensaje, puede buscarlo y examinarlo. Si todo está bien, transfiere en serie los 8 bits del mensaje leído al registro de desplazamiento SIPO. Una vez hecho esto, le indica al registro que envíe todos a la vez los bits almacenados a su respectiva salida.

Esta función tiene **prioridad normal**, lo que significa que está en el mismo nivel de prioridad que nuestra función *lectura*.

```
void StartWrite(void *argument)
{
  /* VARIABLE DECLARATION */
  osStatus_t status;
  MSGQUEUE_OBJ_t msg;

  /* Infinite loop */
  for(;;)
  {
    /* Wait until message is available */
    status = osMessageQueueGet(mid_MsgQueue, &msg, NULL,
osWaitForever);

    /* Validate message */
    if (status == osOK) {

      /* Load data into shift register */
      for(int i = 0; i < 8; i++) {
        /* Send bit from microcontroller*/
        if (msg.flags & 1)
          HAL_GPIO_WritePin(GPIOF, GPIO_PIN_3, GPIO_PIN_SET);
        else
          HAL_GPIO_WritePin(GPIOF, GPIO_PIN_3, GPIO_PIN_RESET);

        osDelay(20);

        /* Store received bit */
```

```
            HAL_GPIO_WritePin(GPIOF, GPIO_PIN_10, GPIO_PIN_SET);
            osDelay(20);
            HAL_GPIO_WritePin(GPIOF, GPIO_PIN_10, GPIO_PIN_RESET);

            /* Shift to next bit */
            msg.flags >>= 1;
          }
          /* Write stored data to shift register output */
            HAL_GPIO_WritePin(GPIOF, GPIO_PIN_5, GPIO_PIN_SET);
            osDelay(1);
            HAL_GPIO_WritePin(GPIOF, GPIO_PIN_5, GPIO_PIN_RESET);
        }
      }
    }
```

Referencias

[1] A. Singh y A. Kumar, "Smart irrigation system using IoT and machine learning for precision agriculture," *IEEE Internet of Things Journal*, vol. 8, no. 12, pp. 9456-9467, Jun. 2021.

[2] M. Rodriguez, P. García, y J. Martinez, "Automated irrigation systems: A comprehensive review of soil moisture sensors and control algorithms," *Agricultural Water Management*, vol. 245, pp. 106-118, Feb. 2021.

[3] J. Chen y L. Wang, "Low-cost capacitive soil moisture sensors for precision agriculture: Design, calibration and field evaluation," *Sensors*, vol. 21, no. 8, pp. 2751, Apr. 2021.

[4] R. Thompson, S. Davis, y K. Wilson, "STM32-based irrigation control system with wireless monitoring capabilities," en *Proc. IEEE Int. Conf. Agricultural Engineering*, Madrid, España, 2021, pp. 145-152.

[5] P. Kumar et al., "Raspberry Pi and Arduino integration for smart home garden automation," *IEEE Access*, vol. 9, pp. 76543-76558, 2021.

[6] D. Lee y M. Park, "Capacitive soil moisture sensor calibration for different soil types in urban agriculture," *Journal of Agricultural Engineering Research*, vol. 156, pp. 89-97, Mar. 2021.

[7] "ESP32 irrigation system with soil moisture monitoring," GitHub. [En línea]. Disponible: https://github.com/examples/smart-irrigation. [Accedido: 1-jun-2025].

[8] F. Oliveira, A. Santos, y C. Silva, "Energy-efficient solenoid valve control for automated irrigation systems," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 5, pp. 4234-4242, May 2021.

[9] STMicroelectronics, "STM32F4 Series Reference Manual," ST Microelectronics, Technical Report RM0090, Rev. 19, 2021.

[10] Y. Zhang, H. Liu, y Q. Chen, "UART communication protocols for embedded agricultural monitoring systems," *IEEE Embedded Systems Letters*, vol. 13, no. 2, pp. 67-70, Jun. 2021.

[11] B. Anderson y K. Johnson, "Relay module interfacing with microcontrollers: Best practices and safety considerations," *Electronics Letters*, vol. 57, no. 8, pp. 312-314, Apr. 2021.

[12] "Arduino soil moisture sensor library," Arduino Community. [En línea]. Disponible: https://github.com/arduino-libraries/soil-moisture. [Accedido: 15-may-2025].

[13] G. Taylor, R. Brown, y S. White, "Real-time monitoring systems for urban agriculture using embedded platforms," en *Proc. Int. Symposium on Urban Agriculture Technology*, Tokyo, Japón, 2021, pp. 78-85.

[14] N. Patel y V. Shah, "FreeRTOS task scheduling optimization for agricultural IoT applications," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 6, pp. 3945-3954, Jun. 2021.

[15] "Raspberry Pi irrigation controller documentation," Raspberry Pi Foundation. [En línea]. Disponible: https://www.raspberrypi.org/documentation/irrigation/. [Accedido: 20-may-2025].