SMARTGARDEN

Autor – Armando Rial Michalski, Eng
Serviço Nacional de Aprendizagem Comercial do Rio Grande do Sul
Curso de Sistemas para Internet
Rua Coronel Genuíno 130 – Campus Poa I - CEP: 90010-350 – Porto Alegre / RS– Brasil
Telefone: (51) 99992-4221 – Email: armando.rial@gmail.com

Orientador – Luciano Zanuz, PhD
Serviço Nacional de Aprendizagem Comercial do Rio Grande do Sul
Rua Coronel Genuíno 130 – Campus Poa I - CEP: 90010-350 – Porto Alegre / RS– Brasil
Telefone: (51) 3322-9400 – Email: lzanuz@senacrs.com.br / professor.zanuz@gmail.com

Resumo. *O SmartGarden é uma solução de irrigação automatizada que visa facilitar o cuidado com plantas em cenários onde a presença constante do cuidador não é possível. Este projeto tem como objetivo criar um sistema funcional, acessível e escalável, capaz de automatizar a rega de plantas com base na umidade do solo ou por controle remoto via interface web.*

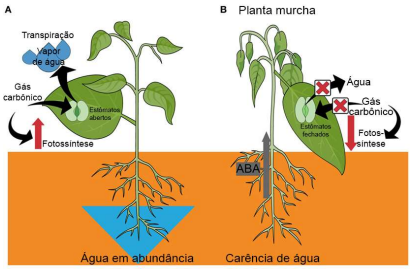
Palavras-chave: *Jardinagem inteligente; Microprocessador; Arduino, Programação C++.*

Abstract. *The SmartGarden is an automated irrigation solution designed to make plant care easier in situations where the constant presence of a caregiver is not possible. This project aims to create a functional, accessible, and scalable system capable of automating plant watering based on soil moisture or through remote control via a web interface.*
This second water reservatory feeds the flush toilets and the taps from external areas of the residence. In order to ensure the functioning of these system during drought periods, there is a connection between the conventional water reservatory and the reservatory of recovered water. The system also allows the recovery of wasted water in the transient of heating water for the shower.

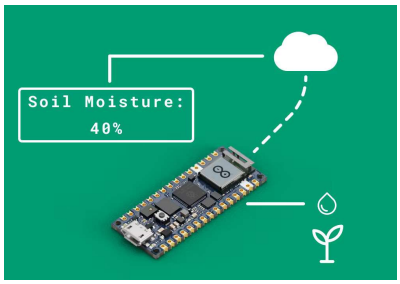
Key-words: *Smart garden; Microprocessor; Arduino, C++ programming.*

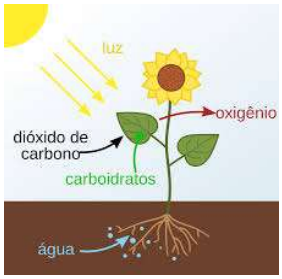
1. INTRODUÇÃO

1.1) Problema: A irrigação manual de plantas e jardins exige tempo e dedicação contínua por parte do responsável. Esse método está sujeito a falhas humanas, como excesso ou falta de água, o que pode comprometer o desenvolvimento saudável das plantas. Além disso, em áreas extensas ou em situações de ausência prolongada, a prática manual torna-se inviável, ocasionando desperdício de recursos hídricos e dificultando a manutenção adequada da vegetação.

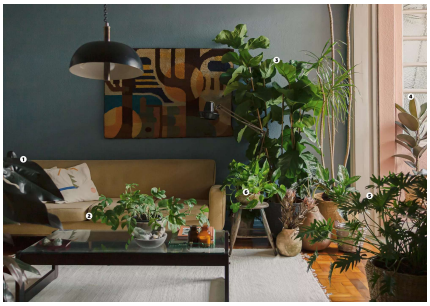


1.2) Proposta: Diante desse cenário, este projeto propõe a criação de um sistema automatizado de irrigação, desenvolvido a partir de um microcontrolador integrado a sensores capacitivos de umidade do solo e atuadores como motobomba e válvulas solenóides. O sistema será capaz de monitorar em tempo real as condições do solo e acionar a irrigação apenas quando necessário, garantindo maior eficiência no uso da água e autonomia no cuidado das plantas. Essa abordagem busca aliar tecnologia e sustentabilidade, oferecendo uma solução para otimizar o processo de irrigação.





1.3) Objetivo: O principal objetivo deste trabalho é automatizar a irrigação, reduzindo a necessidade de intervenção manual e assegurando níveis adequados de umidade para o crescimento saudável das plantas. Entre os objetivos específicos, destacam-se: minimizar o desperdício de água por meio de acionamento inteligente, disponibilizar monitoramento e controle remoto por meio de interface digital e contribuir para práticas mais sustentáveis na gestão de recursos hídricos. Dessa forma, espera-se que o sistema represente um avanço na modernização de técnicas de irrigação.

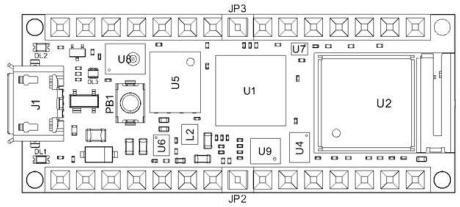


2. PLANEJAMENTO

O planejamento do projeto iniciou-se com a definição clara do problema e dos objetivos a serem alcançados. A partir dessa delimitação, foram estabelecidos os requisitos funcionais e técnicos do sistema de irrigação automatizado, considerando aspectos como eficiência no uso da água, confiabilidade dos componentes e facilidade de operação. Nessa etapa, também foram analisados dados que evidenciam a necessidade de soluções mais eficazes, bem como sistemas semelhantes já existentes, o que permitiu identificar lacunas e oportunidades de inovação. Essa fase inicial foi fundamental para orientar as escolhas posteriores e garantir que o desenvolvimento estivesse alinhado às necessidades identificadas.

2.1) Componentes: Em seguida, realizou-se o levantamento dos recursos necessários para a implementação do sistema. Foram analisadas diferentes opções de microcontroladores, sensores de umidade do solo e atuadores, levando em conta critérios de custo, disponibilidade no mercado e compatibilidade técnica. O microcontrolador escolhido, Arduino RP2040 Connect, destaca-se por oferecer conectividade Wi-Fi e Bluetooth, possibilitando integração com aplicativos móveis e dashboards de monitoramento. Os sensores capacitivos de umidade foram selecionados por sua precisão e durabilidade, já que não sofrem corrosão. Relés e válvulas solenóides foram escolhidos pela compatibilidade com a tensão de operação da motobomba, assegurando confiabilidade no acionamento. Além disso, foram definidos materiais auxiliares, como tubulações e reservatórios, que asseguram a integração adequada entre hardware e ambiente físico. Essa análise comparativa permitiu selecionar os componentes mais adequados para o protótipo, conciliando viabilidade técnica e financeira.

2.1.1. Microcontrolador:

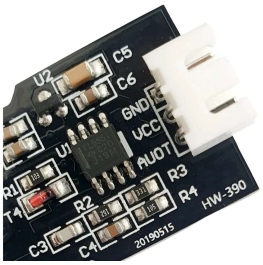


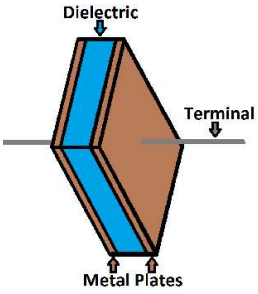
Front View of Arduino Nano RP2040 Connect Topology

Ref.	Description	Ref.	Description
U1	Raspberry Pi RP2040 Microcontroller	U2	Ublox NINA-W102-00B Wi-Fi/Bluetooth® Module
U3	N/A	U4	ATECC608A-MAHDA-T Crypto IC
U5	AT25SF128A-MHB-T 16MB Flash IC	U6	MP2322GQH Step-Down Buck Regulator
U7	DSC6111HI2B-012.0000 MEMS Oscillator	U8	MP34DT06JTR MEMS Omnidirectional Microphone IC
U9	LSM6DSOXTR 6-axis IMU with Machine Learning Core	J1	Male Micro USB Connector
DL1	Green Power On LED	DL2	Builtin Orange LED
DL3	RGB Common Anode LED	PB1	Reset Button
JP2	Analog Pin + D13 Pins	JP3	Digital Pins

- Modelo: ABX00053
- Descrição: Arduino Nano RP2040 Connect
- Processador: 133 MHz 32bit Dual Core Arm® Cortex®-M0+
- Wifi module: U-blox® Nina W102
- Alimentação: 5 Vdc

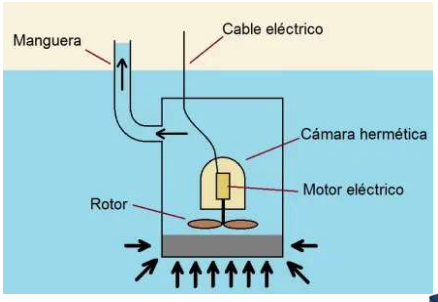
2.1.2. **Sensor capacitivo de úmidade**:





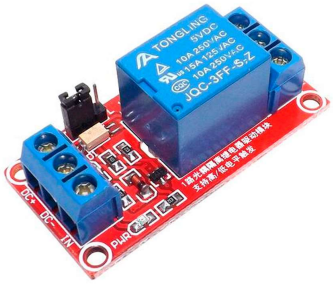
- Modelo: HW-390
- Descrição: Sensor capacitivo de úmidade
- Portas: VCC, GND, AOUT
- Alimentação: 5 Vdc

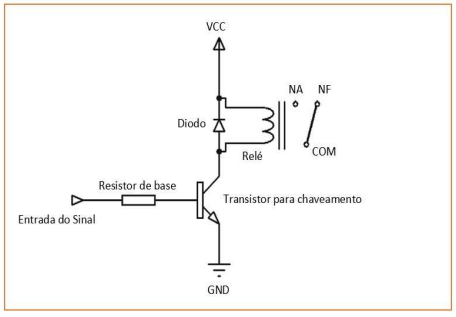
2.1.3. **Bomba submerssível**:



- Modelo: Mini-bomba aquário 5V
- Vazão: 80-100 litros / hora
- Elevação máxima: 40-100 cm
- Alimentação: 3-5 Vdc
- Magueira: 8 mm

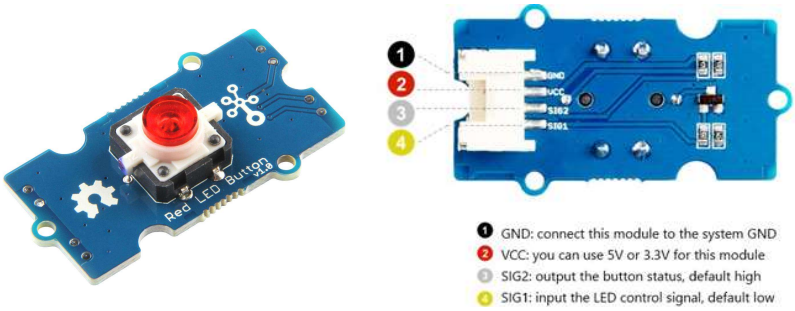
2.1.4. **Modulo relé**:

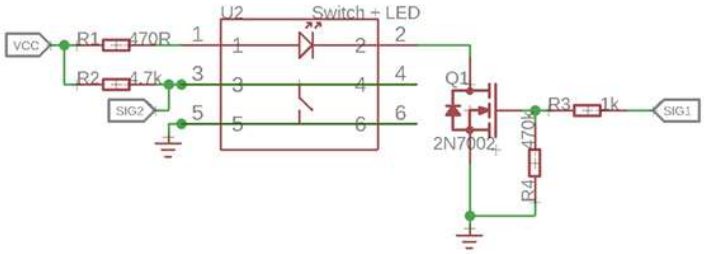




- Modelo: MOD-R1X-05V
- Alimentação 5 Vdc: Portas DC+, DC-
- Acionamento: Porta IN (0 ou 1), alterando jumper (NPN ou PNP)
- Saída: COM (Entrada Comum), NA (Saída Normalmente-Aberta), NF (Saída Normalmente-Fechada)

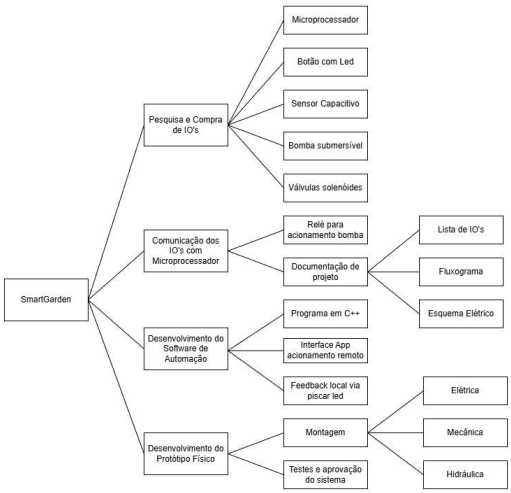
2.1.5. **Modulo botão com led**:





- Modelo: MOD-R1X-05V
- Alimentação: 5 Vdc (Portas VCC, GND)
- Porta SIG1: Entrada do sinal de controle do LED, padrão baixo
- Porta SIG2: Saída do status do BOTÃO, padrão alto

2.2) Cronograma: Por fim, elaborou-se um cronograma de atividades que contemplava as fases de desenvolvimento, testes e ajustes do sistema. O cronograma foi desenvolvido a partir do digrama EAP (estrutura analítica de projeto) abaixo. O planejamento metodológico incluiu a divisão das tarefas em etapas sequenciais: projeto eletrônico, programação do microcontrolador, montagem física e validação experimental. A arquitetura modular do sistema foi considerada nesse processo, permitindo futuras expansões sem necessidade de grandes investimentos adicionais. Essa organização possibilitou maior controle sobre o andamento do trabalho e facilitou a identificação de possíveis riscos ou atrasos, garantindo maior eficiência na execução da pesquisa e maior sustentabilidade do projeto.

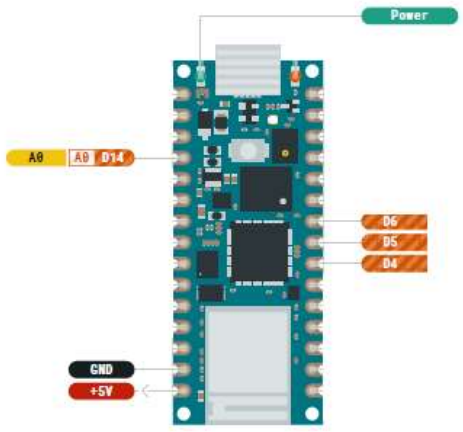


3. DESENVOLVIMENTO

3.1) Hardware

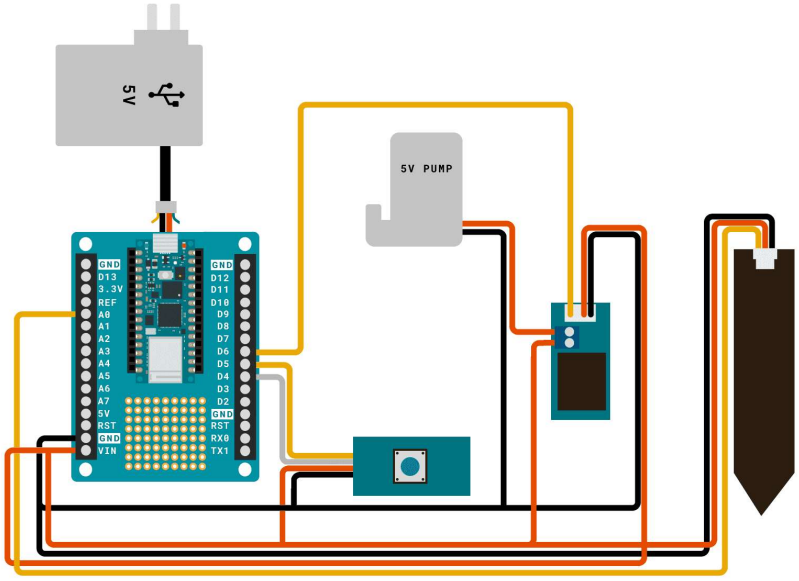
3.1.1. Lista de I/O:

- Entradas Digitais = Porta D4 (Botão Físico NA Não-Rentivo)
- Entradas Analógicas = Porta A0 (Sensor Capacitivo de Umidade)
- Saídas Digitais = Porta D5 (Led), Porta D6 (Relé 5Vdc PNP)
- Saída Analógicas = Não utilizada
- Fonte de alimentação = 5V 2A



3.1.2. Esquema elétrico:

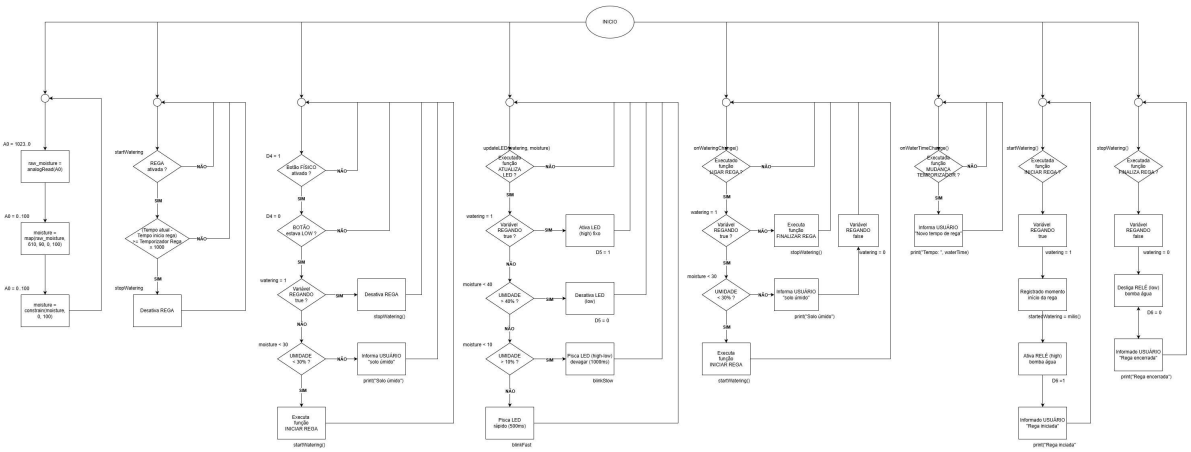
Com objetivo de orientar a montagem elétrica do sistemas, e facilitar o comissionamento das conexões elétricas, foi desenvolvido o esquema elétrico abaixo.



3.2) Software

3.2.1. Diagrama funcionamento do sistema:

Com objetivo de facilitar o entendimento do sistemas foi desenvolvido o diagrama de software abaixo. Ele mostra as funcionalidades locais e remotas, e ocorre a interação entre essas variáveis e funções.



3.2.2. Programa de controle (Código C++):

Nesta primeira parte do código definimos as portas do microcontrolador, bibliotecas, variáveis, constantes e setup inicial (configuração da comunicação serial, configuração da comunicação com ArduinoCloud, condições iniciais das variáveis do sistema). Para informar ao usuário que o setup foi finalizado com sucesso, e podemos então iniciar a interação com sistema, foi inserido um piscar de 4 vezes do led (linhas 45 ao 52).

```
1  /* ------------ PARÂMETROS DE CONFIGURAÇÃO ------------ */
2
3  constexpr int BUTTON_PIN = 4;                    // Entrada digital para botão
4  constexpr int LED_PIN    = 5;                    // Saída digital para LED do botão
5  constexpr int RELAY_PIN  = 6;                    // Saída digital para relé
6  constexpr int MOIST_PIN  = A0;                   // Entrada analógica para sensor de umidade
7
8  int raw_moisture = 0;                            // Valor bruto do sensor
9  int moisture = 0;                                // Percentual de umidade
10
11 // Constantes simbólicas para limites de umidade (otimizar)
12 constexpr int MOISTURE_WET = 40;                  // Acima de 40% = solo úmido
13 constexpr int MOISTURE_DRY = 10;                  // Abaixo de 10% = solo muito seco
14
15 // Configuração dos tempos de piscar
16 int blinkSlow = 1000;                            // período em ms (devagar)
17 int blinkFast = 500;                             // período em ms (rápido)
18
19 /* ------------ BIBLIOTECAS E VARIÁVEIS ------------ */
20
21 #include "thingProperties.h"                     // Conexão com Arduino IoT CLoud
22 #include <Bounce2.h>                             // Controle de botão com debounce
23
24 Bounce b;                                        // Instância do botão com debounce
25 unsigned long startedWatering;                   // Marca o início da rega
26
27 /* ------------ CONFIGURAÇÃO INICIAL ------------ */
28
29 void setup() {
30   Serial.begin(9600);                            // Inicia a comunicação serial com o computador a 9600 bps
31   delay(1500);                                   // Aguarda 1,5 segundos para estabilização do sistema
32
33   b.attach(BUTTON_PIN, INPUT_PULLUP);            // Configura o botão com resistor pull-up interno usando a biblioteca Bounce2
34   b.interval(25);                                // Define intervalo de debounce de 25 ms para evitar leituras falsas do botão
35   pinMode(LED_PIN, OUTPUT);                      // Define o pino do LED como saída
36   pinMode(RELAY_PIN, OUTPUT);                    // Define o pino do relé como saída
37
38   stopWatering();                                // Garante que a bomba esteja desligada ao iniciar o sistema
39
40   initProperties();                              // Inicializa as propriedades vinculadas à Arduino IoT Cloud
41   ArduinoCloud.begin(ArduinoIoTPreferredConnection);  // Inicia a conexão com a nuvem usando a configuração preferida
42   setDebugMessageLevel(4);                       // Define o nível de detalhamento das mensagens de debug (4 = detalhado)
43   ArduinoCloud.printDebugInfo();                 // Imprime informações de debug da conexão com a nuvem
44
45   // Pisca o LED 5 vezes para indicar que o sistema está rodando corretamente
46   for (int i = 0; i <= 4; i++) {
47     digitalWrite(LED_PIN, HIGH);                 // Acende o LED
48     delay(200);                                  // Aguarda 200 ms
49     digitalWrite(LED_PIN, LOW);                  // Apaga o LED
50     delay(200);                                  // Aguarda 200 ms
51   }
52 }
```

Segundamente, definimos duas funções que serão utilizadas no Loop Principal do programa: “INICIAR REGAR” (*startWatering*) e “FINALIZAR REGA “ (*stopWatering*). Essas funções, além de acionar o relé, alteram uma variável interna de controle chamada “REGANDO” (*watering*) e informam ao usuário via comunicação serial da “Rega iniciando” ou “Rega encerrada”.

```
/* ------------ FUNÇÕES DE CONTROLE DE REGA ------------ */

void startWatering() {
  watering = true;                        // Define a variável de controle como verdadeira (rega ativa)
  startedWatering = millis();             // Registra o momento em que a rega começou (em milissegundos)
  digitalWrite(RELAY_PIN, HIGH);          // Liga o relé, ativando a bomba de água
  Serial.println("Rega iniciada.");       // Exibe mensagem no monitor serial indicando início da rega
}

void stopWatering() {
  watering = false;                       // Define a variável de controle como falsa (rega encerrada)
  digitalWrite(RELAY_PIN, LOW);           // Desliga o relé, interrompendo a bomba de água
  Serial.println("Rega encerrada.");      // Exibe mensagem no monitor serial indicando fim da rega
```

Terceiramente, entramos no Loop Principal do programa onde temos a leitura do sensor capacitivo de umidade (convertendo valor bruto, para valor pencentual), leitura do botão do físico (com debounce, que evita falsos positivos ao apertar o botão rapidamente), controle do temporizador de acionamento da bomba de água via relé (desliga bomba, após superado o tempo ajustado), controle do led para feedback da umidade durante rega (piscar lento ou rápido), comunicação serial (umidade e temporizador).

```
/* ------------ LOOP PRINCIPAL ------------ */

void loop() {
  ArduinoCloud.update();              // Atualiza a conexão com o Arduino IoT Cloud e sincroniza variáveis

  // Lê o sensor de umidade e converte o valor bruto para percentual (0% seco, 100% úmido)
  raw_moisture = analogRead(MOIST_PIN);         // Lê o valor analógico do sensor de umidade
  moisture = map(raw_moisture, 610, 90, 0, 100);    // Converte o valor para escala percentual invertida
  moisture = constrain(moisture, 0, 100);       // Garante que o valor fique entre 0% e 100%

  // Exibe os dados no monitor serial com marcação de tempo
  Serial.print("Umidade: ");              // Imprime o texto "Umidade: "
  Serial.print(moisture);                 // Imprime o valor percentual da umidade
  Serial.print("% - Tempo: ");            // Imprime o texto "% - Tempo: "
  Serial.println(millis());               // Imprime o tempo atual em milissegundos desde que o Arduino foi ligado

  // Atualiza LED conforme status
  updateLED(watering, moisture);

  // Verifica se o tempo de rega já passou e interrompe se necessário      // Se variável 'watering' for verdadeira (rega ativa),
  if (watering && (millis() - startedWatering) >= waterTime * 1000) {      // E o tempo atual menos o tempo de início da rega, for maior ou igual ao tempo
    stopWatering();                       // Encerra-se a rega após o tempo configurado
  }

  // Atualiza o estado do BOTÃO FÍSICO com debounce              // Verifica se houve mudança no botão HIGH → LOW (pressionado) ou LOW → HIGH (solto),
  b.update();                                                    // Como botão INPUT_PULLUP, o estado normal (não pressionado) é HIGH,
  if (b.changed() && b.read() == false) {                        // Se o botão foi pressionado e nível lógico LOW,
    if (watering) {                                              // Se já estiver regando
      stopWatering();                                            // Interrompe a rega
    } else {                                                     // Senão verifica
      if (moisture < 30) {                                       // Se umidade estiver abaixo de 30%
        startWatering();                                         // Inicia a rega
      } else {                                                   // Senão supõe-se que umidade está acima de 30%
        Serial.println("Solo úmido — rega local não iniciada."); // E informa usuário que solo esta úmido, e ignora comando
      }
    }
  }
}
```

Por último, temos a declaração da função controle do led para feedback da umidade durante rega (piscar lento ou rápido), utilizada no Loop Principal. Temos também neste trecho de código abaixo as funções que verificam solicitações remotas via aplicação móvel (ArduinoCloud) e acionam essas funções internas “LIGAR REGA” (onWateringChange) e “EDITAR TEMPORIZADOR” (onWaterTimeChange). Essas rotinas permitem que o sistema seja controlado tanto de forma automática quanto manual, oferecendo maior flexibilidade ao usuário. Além disso, a integração com a nuvem garante que os comandos possam ser executados em tempo real, mesmo à distância. Dessa maneira, o projeto se aproxima de uma solução inteligente e conectada, alinhada às demandas atuais de automação residencial e agrícola.

```
/* ------------ FUNÇÃO DE CONTROLE DO LED ------------ */
// millis() = Retorna o tempo em milissegundos desde que o Arduino foi ligado
// % blinkSlow  = Calcula o resto da divisão do tempo atual pelo período de piscar (ex: 1000 ms)
// < (blinkSlow / 2) = Compara se esse tempo está na primeira metade do ciclo

void updateLED(bool watering, int moisture) {
  if (watering) {
    digitalWrite(LED_PIN, HIGH);                  // LED acesso
  } else if (moisture > MOISTURE_WET) {           // Umidade < 40%
    digitalWrite(LED_PIN, LOW);                   // LED apagado
  } else if (moisture > MOISTURE_DRY) {           // Umidade < 10%
    digitalWrite(LED_PIN, (millis() % blinkSlow) < (blinkSlow / 2));    // Pisca devagar = 1000ms = 1s
  } else {
    digitalWrite(LED_PIN, (millis() % blinkFast) < (blinkFast / 2));    // Pisca rápido = 500ms = 0,5s
  }
}

/* ------------ FUNÇÕES DE EVENTO DA NUVEM ------------ */

// Evento de mudança na variável watering
void onWateringChange() {                         // Função de rega via app
  if (watering) {                               // Verifica se a variável 'watering' foi definida como TRUE (pedido para iniciar a rega)
    if (moisture < 30) {                          // Se umidade estiver abaixo de 30%
      startWatering();                            // Inicia a rega
    } else {                                    // Senão
      Serial.println("Solo úmido — rega remota ignorada.");     // Informa usuário que o solo está úmido, e ignora o comando
      watering = false;                           // Corrige o estado da variável na nuvem para refletir que a rega não foi iniciada
    }
  } else {                                    // Senão supõe-se que 'watering' é FALSE,
    stopWatering();                             // E interrompe a rega
  }
}

// Evento de mudança na variável waterTime
void onWaterTimeChange() {
  Serial.print("Novo tempo de rega configurado: ");         // Imprime uma mensagem indicando que o tempo de rega foi alterado
  Serial.print(waterTime);                                  // Imprime o novo valor da variável 'waterTime' (em segundos)
  Serial.println(" segundos.");                             // Finaliza a linha com a unidade de tempo e quebra de linha
}

```

3.2.3. Comentários sobre trechos do programa (Código C++):

Com objetivo de obter um maior esclarecimente sobre o código acima, trago abaixo três trechos que acredito caberem maiores explicações. O primeiro trecho é sobre o piscar do led de feedback visual. Para controle desse piscar geralmente utiliza-se a função delay, entretanto essa estrágia poderia afetar o Loop Pricipal, gerando atrasos indesejados no funcionamento do sistema. Como alternativa e solução deste problemas, optou-se pela utilização do algoritmo descrito abaixo.

Sabendo:

- milis() = Tempo atual (ms) desde que o arduino foi ligado.
- BlinkSlow = Variável que armazena o período de piscar lento.
- % blinkSlow = Resto da divisão (%) do tempo atual pelo período de piscar
- < (blinkSlow/ 2) = Compara se o tempo anterior (resto da divisão, do tempo atual pelo tempo setado como período lento) a comparação (menor que) está na primeira metade do ciclo (tempo setado divido por dois). Essa comparção deve então retornar TRUE ou FALSE, a depender do resultado da comparação.

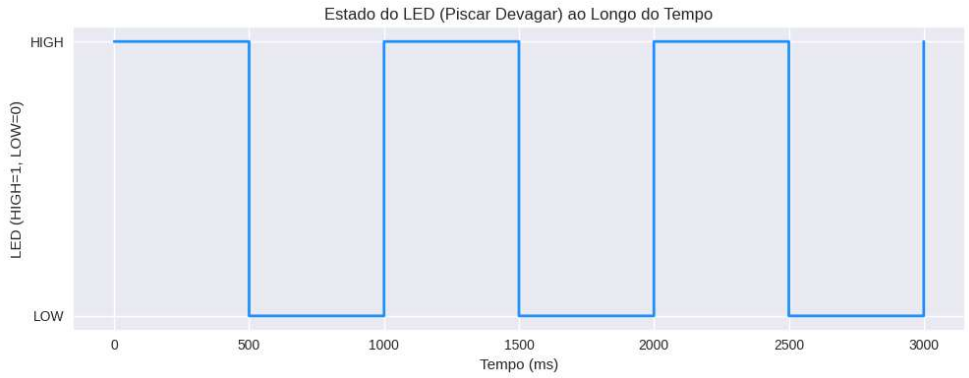
Formulou-se:

- blinkSlow = 1000
- digitalWrite(LED_PIN, (millis() % blinkSlow) < (blinkSlow/2));

Esse método permite que o LED pisque de forma contínua sem bloquear a execução das demais rotinas do programa. Além disso, garante maior responsividade do sistema, já que o loop principal permanece livre para processar outras tarefas em tempo real.

Logo:

- LED on (millis() % 1000) < 500)→ primeira metade (500ms) do ciclo.
- LED off (millis() % 1000) >= 500) → segunda metade (500ms) do ciclo.



Assim:

- Não é necessario utilizar o delay, então o Loop Principal roda sem atrasos.
- Operador % (resto da divisão) reinicia o tempo a cada ciclo de 1000 ms.

milis() [ms]	milis() % 1000	< 500	Resultado	LED
0	0	TRUE	HIGH	Ligado (on)
200	200	TRUE	HIGH	Ligado (on)
499	499	TRUE	HIGH	Ligado (on)
500	500	FALSE	LOW	Desligado (off)
700	700	FALSE	LOW	Desligado (off)
999	999	FALSE	LOW	Desligado (off)
1000	0 (reinicia ciclo)	TRUE	HIGH	Ligado (on)
1500	500	FALSE	LOW	Desligado (off)
2300	300	TRUE	HIGH	Ligado (on)
2700	700	FALSE	LOW	Desligado (off)

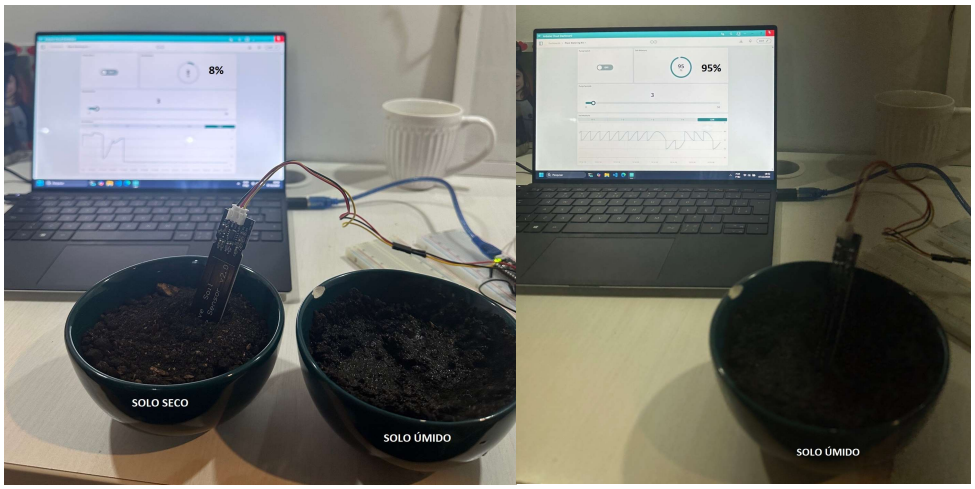
O segundo trecho do código interressante de comentar é a função debounce do botão físico. Na figura abaixo é possível observar a necessidade de utlização de uma biblioteca específica (Bounce2.h). A partir disso podemos definir uma instância do botão com debounde, e definir no setup inicial um tempo de 25ms. Lembrando que também é necessário definir que nesse sistema, o botão utilizado é do tipo não-retentivo, normalmente aberto de retorno automático.

```
/* ------------ BIBLIOTECAS E VARIÁVEIS ------------ */

#include "thingProperties.h"                    // Conexão com Arduino IoT Cloud
#include <Bounce2.h>                            // Controle de botão com debounce

Bounce b;                                       // Instância do botão com debounce
unsigned long startedWatering;                  // Marca o início da rega

/* ------------ CONFIGURAÇÃO INICIAL ------------ */

void setup() {
  Serial.begin(9600);                           // Inicia a comunicação serial com o computador a 9600 bps
  delay(1500);                                  // Aguarda 1,5 segundos para estabilização do sistema

  b.attach(BUTTON_PIN, INPUT_PULLUP);           // Configura o botão com resistor pull-up interno usando a biblioteca Bounce2
  b.interval(25);                               // Define intervalo de debounce de 25 ms para evitar leituras falsas do botão
  pinMode(LED_PIN, OUTPUT);                     // Define o pino do LED como saída
  pinMode(RELAY_PIN, OUTPUT);                   // Define o pino do relé como saída
```

O terceiro trecho do código que gostaria de chamar atenção é a conversão do sinal de leitura do sensor capacitivo de úmidade. Conforme é possível observar, o valor bruto é lido pelo sensor e jogado diretamente para variável raw_moisture (line 60). Sensores analógicos tipicamente apresentam valores entre 0 e 1023. Conforme a umidade aumenta, o valor registrado diminui. Na segunda linha (line 61) temos função *map* que pega o intervalo descrescente do sensor (entre 610 e 90) e converte para valores percentuais crescentes 0 até 100%. Onde 610 corresponde 0% (seco), 90 corresponde a 100% (úmido).

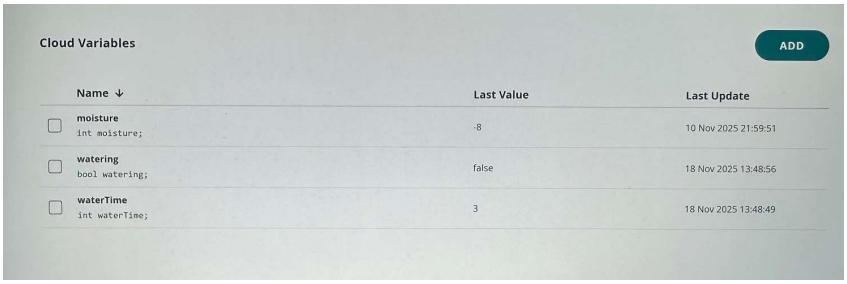
```
59  // Lê o sensor de umidade e converte o valor bruto para percentual (0% seco, 100% úmido)
60  raw_moisture = analogRead(MOIST_PIN);          // Lê o valor analógico do sensor de umidade
61  moisture = map(raw_moisture, 610, 90, 0, 100);    // Converte o valor para escala percentual invertida
62  moisture = constrain(moisture, 0, 100);        // Garante que o valor fique entre 0% e 100%
63
```

A terceira linha (line 62) garrante que a leitura não ultrapasse o limite inferior de 0%, e o limite superior de 100%.



3.2.4. Declaração de variáveis globais:

No ArduinoCloud é necessário definir variáveis que serão compartilhadas entre o Código C++ de controle, e o Dashboard IHM (Interface Homem Máquina). Para que isso aconteça é necessário criarmos uma biblioteca chamada “thingProperties.h” que deve ser importada no programa C++ de controle.

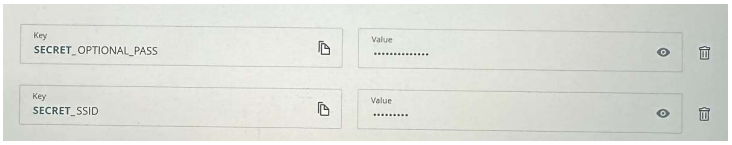


3.2.5. Interface Homem-Máquina (aplicação remota):

Com objetivo de conseguir comunicar o dashboard com o microcontrolador, cria-se um ponto de acesso a partir do Wi-Fi onde o sistema vai ser instalado.

```
#include <ArduinoIoTCloud.h>
#include <Arduino_ConnectionHandler.h>

const char SSID[]     = SECRET_SSID;    // Network SSID (name)
const char PASS[]     = SECRET_OPTIONAL_PASS;    // Network password (use for WPA, or use as key for WEP)

void onWaterTimeChange();
void onWateringChange();

int moisture;
int waterTime;
bool watering;

void initProperties(){

  ArduinoCloud.addProperty(moisture, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(waterTime, READWRITE, ON_CHANGE, onWaterTimeChange);
  ArduinoCloud.addProperty(watering, READWRITE, ON_CHANGE, onWateringChange);

}

WiFiConnectionHandler ArduinoIoTPreferredConnection(SSID, PASS);
```

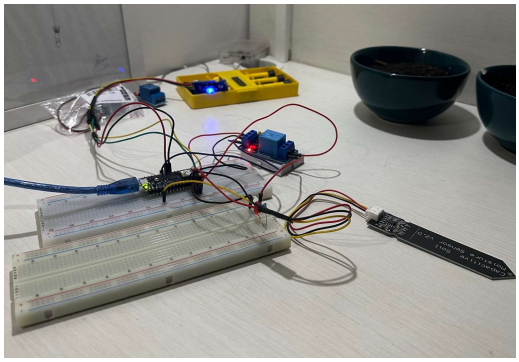
Para não deixar o Id e Senha expostas no código, as mesmas são inseridas separamentes em outro arquivo.



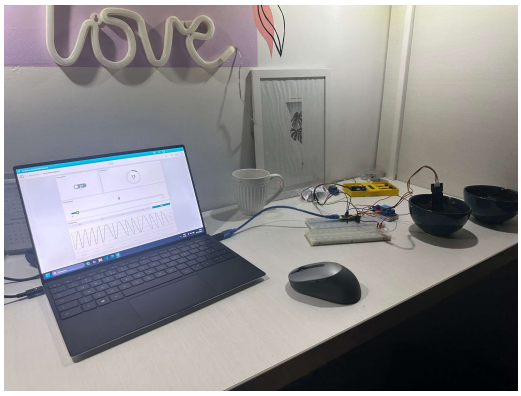
Neste dashboard foram inserido quatro componentes: um botão virtual digital de acionamento do relé e motobomba, de acordo com a lógica estabelecida no programa C; um display da variável analógica do sensor capacitivo de umidade; um temporizador do período de acionamento do relé; um gráfico da leitura da umidade (eixo x) ao longo do tempo (eixo y). Esse histórico cartesiano tem como objetivo principal visualizar o decaimento da úmidade ao logo dos dias. A partir do uso prático desse sistema é possível então revisar os valores internos de umidade, que consideram um sistema muito seco quando umidade fica abaixo de 10% e umido quando está acima de 40%, de acordo com cada solo ou planta. Lembrando que a leitura de 0 até 100%, depende da calibração correta do sensor de acordo com solo utilizado.

3.3) Montagem:

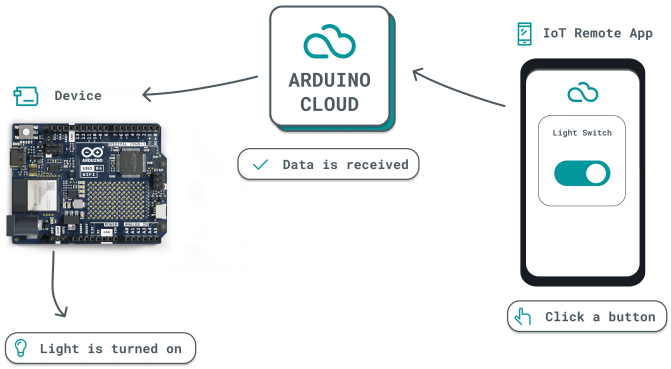
Inicialmente foi realizado e montagem elétrica do sistema, de acordo com os diagramas mostrados acima.



Em em segundo momento, é testado a integração do software com o hardware, a partir de um acionamento de um simples Led. Na figura abaixo a comunicação via cabo, de maneira serial.

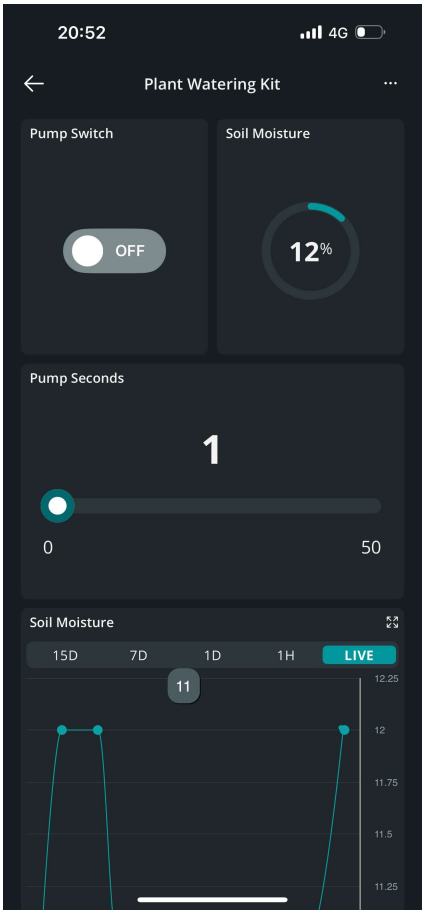


Em um terceiro momento, testa-se a comunicação via Wi-Fi eliminando então a necessidade de um computador rodando com os softwares da Arduino.



Em um quarto momento, conecta-se ao sistema a parte hidraulica, para realização de testes práticos e ajuste do temporizador conforme necessidade.

Por ultimo, testa-se a aplicação via aplicativo móvel fora da rede de Wi-Fi onde o microcontrolador esta inserido.



4. RESULTADOS

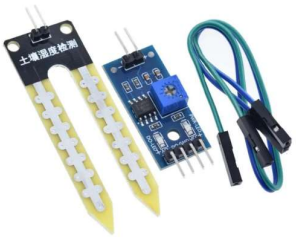
4.1) Custos do protótipo:

- Microprocessador = R$420
- Placa com bornes = R40
- Módulo relé = R$40
- Motobomba = R20
- Sensor de úmidade = R$20
- Módulo botão com led = R$15
- Fonte = R$15
- Protoboard = R$15
- Fios = R$10
- Mangueira = R$5

TOTAL = R$600

4.2) Testes:

Durante a montagem elétrica do sistema, se tornou evidente a necessidade de calibração do sensor de úmidade. Mesmo com a calibração do sensor, houveram sensores que performaram melhor que outros. Foram testados três tipos de sensores de umidade, conforme é possível observar na figura abaixo. O sensor que melhor performou foi o terceiro, que acabou sendo escolhido para integrar o projeto.



Durante a montagem hidraulica do sistema, notou-se que para o correto funcionamento do sistema, o reservatório de água deve localiza-se abaixo do nível da planta. Essa necessidade se torna evidente, uma vez que não existe uma válvula de retenção de fluxo de retorno, na saída da bomba. A adição de uma válvula retenção entra então como uma possível melhoria do sistema. Nos testes realizados com planta e reservatório em mesmo nível, cria-se um vaso comunicante entre os reservatórios, e alturas de água tendem a se estabilizar por conta da ação da pressão atmosférica.

Cabe lembrar que após a instalação dessa eventual vávlula de retenção, uma nova calibragem do temporizador do relé será necessária. Eventuais alterações no comprimento ou diâmetro da maneira, impactam também diretamente no ajuste deste temporizador. Conforme recomendação do manual da bomba, recomenda-se uma diferença de altura de até 1 metro.

4.3) Melhorias:

- Carenagem de proteção, para proteção dos componentes eletrônicos.
- Display para retorno local da leitura dessas variáveis.
- Válvula de retenção na saída da motobomba.
- Sensor de nível baixo, para proteção da motobomba:
  - Ou sensor tipo boia, sendo necessário instalação dentro do reservatório,
  - Ou sensor capactivo, podendo ser instalado do lado de fora do reservatório,
  - Ou sensor pressão, podendo ser instalado abaixo do reservatório.
- Expansão do sistema para controle irrigação de até 8 plantas:
  - Sensores de úmidade para novas plantas (7 unidades).
  - Válvulas solenóides para direcionamento de fluxo (7 unidades).
  - Eventual necessidade de redimensionamento da motobomba, tendo em vista adição de novas resistências hidraulicas ao sistema.
- Luzes UV para controle da iluminação artificial.
- Sensor de chuva para detecção de eventuais precipitações de umidade do ar.
- Sensor de gás carbonico para monitoramento das condições do ambiente.
- Sensor de oxigênio para monitoramento da melhora do ambiente.
- Sensor de peso para controle do crescimento da planta.
- Camera para verificação da evolução do crecimento no formato time-lapse.
- Placa solar para recarregamento da bateria de alimentação do sistema.

5. CONCLUSÃO

O sistema de rega automática desenvolvido com o microcontrolador Arduino RP2040 Connect demonstrou viabilidade técnica e financeira, atendendo aos objetivos propostos de automatizar a irrigação e reduzir a necessidade de intervenção manual. A integração entre sensores capacitivos de umidade, relés e válvulas solenóides possibilitou medições precisas e acionamento confiável dos atuadores, garantindo níveis adequados de umidade para o crescimento saudável das plantas. Dessa forma, o projeto mostrou-se uma solução prática e eficiente para otimizar o uso da água e assegurar o cuidado contínuo da vegetação.

Além de contribuir para a redução do desperdício de recursos hídricos, o sistema apresenta potencial para aplicação em diferentes contextos, desde pequenos jardins residenciais até áreas agrícolas de maior porte. Sua arquitetura modular e de baixo custo favorece a replicabilidade e a expansão futura, tornando-o uma alternativa acessível e sustentável. A utilização de conectividade Wi-Fi e Bluetooth amplia as possibilidades de monitoramento remoto, aproximando o projeto das demandas contemporâneas por soluções inteligentes e integradas.

Embora os resultados obtidos sejam satisfatórios, o sistema pode ser aprimorado com a incorporação de novas funcionalidades, como a adição de múltiplos sensores de umidade para ampliar a precisão das medições em diferentes pontos do solo, o uso de válvulas solenóides adicionais para direcionar o fluxo de água de forma segmentada e personalizada, e a inclusão de um display LCD 16x2 para visualização local dos níveis de umidade. Além disso, melhorias como a integração com previsões meteorológicas, o uso de energia renovável para alimentar os componentes e a aplicação de algoritmos de aprendizado de máquina podem otimizar ainda mais o processo de irrigação. Tais avanços ampliariam a autonomia e a eficiência do sistema, consolidando-o como uma ferramenta relevante para práticas agrícolas e ambientais sustentáveis.

6. REFERÊNCIAS

- ALLEN, R. G.; PEREIRA, L. S.; RAES, D.; SMITH, M. *Crop Evapotranspiration – Guidelines for Computing Crop Water Requirements*. FAO Irrigation and Drainage Paper 56. Rome: FAO, 1998.
- ARDUINO. *Arduino RP2040 Connect Documentation*. Disponível em: <https://docs.arduino.cc/hardware/rp2040-connect>. Acesso em: 25 nov. 2025.
- BANZI, Massimo. *Getting Started with Arduino*. 4. ed. Sebastopol: Make Community, 2022.
- CRAFT, Brock. *Arduino Projects For Dummies*. Hoboken: Wiley Publishing, 2013.
- CHEICH, Mike. *Arduino Book for Beginners*. Programming Electronics Academy, 2021.
- DESAI, Pratik. *Python Programming for Arduino*. Birmingham: Packt Publishing, 2015.
- EMBRAPA. *Tecnologias de irrigação para agricultura sustentável*. Brasília: Embrapa, 2018.
- FRIZZARIN, Fernando Bryan. *Arduino*. São Paulo: Casa do Código, 2016.
- FRIZZARIN, Fernando Bryan. *Arduino prático*. São Paulo: Casa do Código, 2016.
- JOHNSON, Noelle. *The Water-Smart Garden*. Franklin: Cool Springs Press, 2025.
- KAUL, Lukas. *Practical Arduino Robotics*. Birmingham: Packt Publishing, 2023.
- KARVINEN, Tero. *Make a Mind-Controlled Arduino Robot*. Sebastopol: Make Community, 2011.
- KURNIAWAN, Agus. *IoT Projects with Arduino*. Cham: Apress, 2021.
- MARKEN, Bill. *Container Gardening For Dummies*. Hoboken: Wiley Publishing, 2010.
- MISRA, Yogesh. *Programming and Interfacing with Arduino*. Boca Raton: CRC Press, 2021.
- NUSSEY, John. *Arduino For Dummies*. Hoboken: Wiley Publishing, 2013.
- NUSSEY, John. *Arduino For Dummies*. 2. ed. Hoboken: Wiley Publishing, 2018.
- OLIVEIRA, Claudio. *Arduino descomplicado*. São Paulo: Érica, 2015.
- PEREIRA, L. S.; CORDERY, I.; IACOVIDES, I. *Coping with Water Scarcity: Addressing the Challenges*. Cham: Springer, 2009.
- PURDUM, Jack. *Arduino Projects for Amateur Radio*. New York: McGraw-Hill, 2014.
- SILVA, J. R.; OLIVEIRA, A. P. Sistemas automatizados de irrigação com sensores de umidade. *Revista Engenharia Agrícola*, Campinas, v. 40, n. 2, p. 123-134, 2020.
- VERMA, Vijay. *Electronic Components for Arduino*. Independently Published, 2021.
- WAHER, Peter. *Learning IoT with Arduino Building*. Birmingham: Packt Publishing, 2016.