Manuale utente

Aqua&Terra v1.0

Scopo del documento

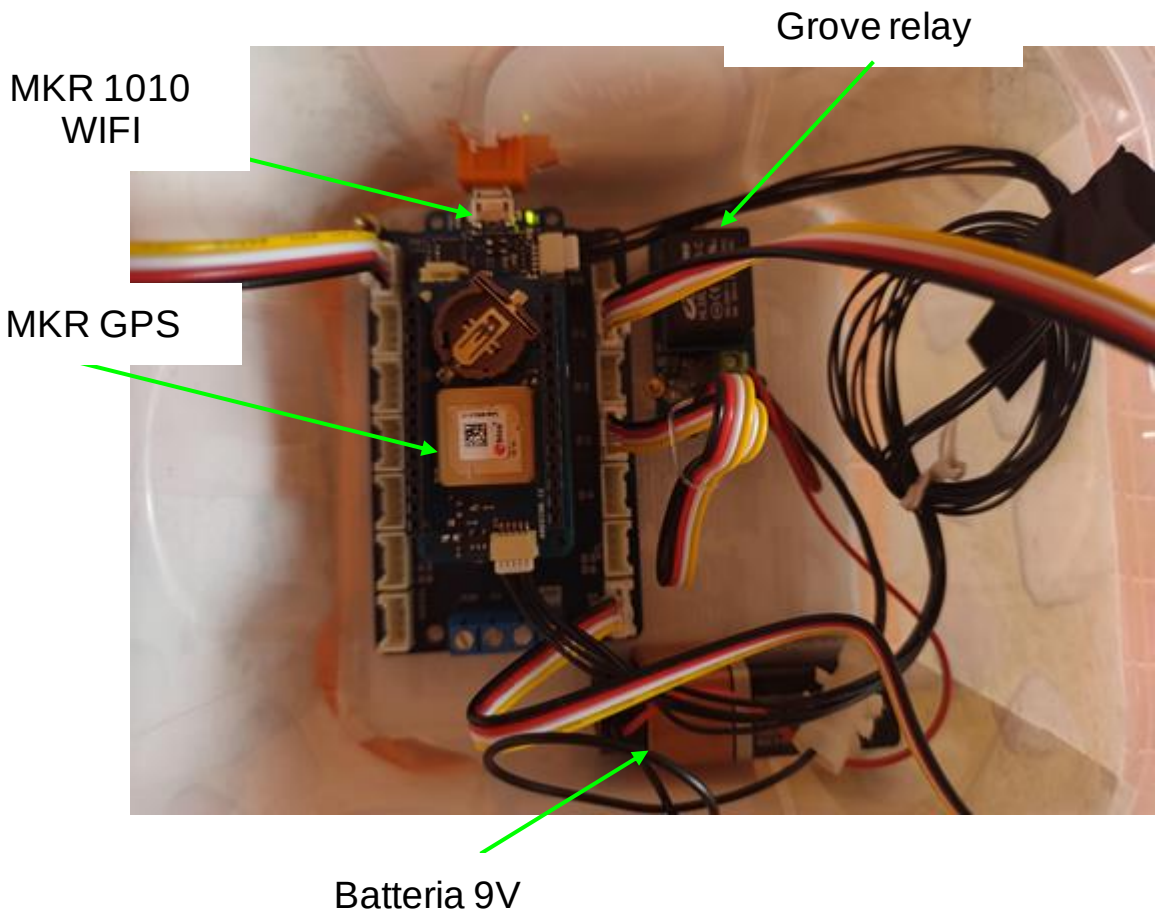
Il presente documento descrive l'uso da parte del progetto Aqua&Terra, presentato per il Maker Faire 2023.

Introduzione

Il progetto Aqua&Terra si prefigge di sviluppare, mediante la piattaforma HW e SW Arduino, un dispositivo di irrigazione delle piante basato su dati locali (umidità del terreno) e su dati remoti (previsioni meteorologiche).
Il dispositivo prevede il controllo da remoto mediante l'integrazione con la piattaforma IoT messa a disposizione da Arduino.

Il dispositivo Aqua&Terra

L'immagine seguente mostra il dispositivo, comprensivo di tutte le componenti hardware necessarie al suo funzionamento:

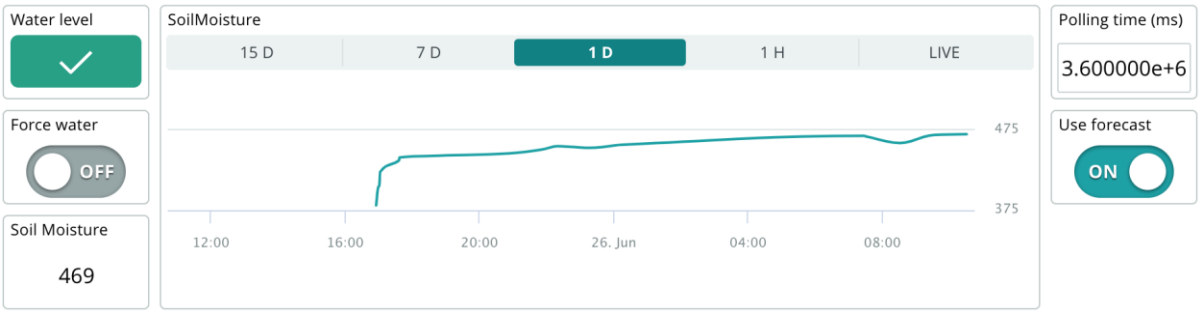


I componenti sono i seguenti:

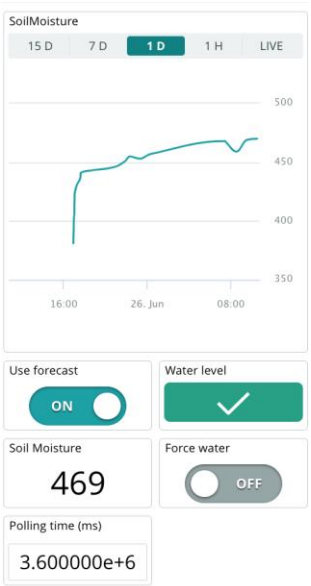
- 1 Grove - Moisture Sensor per la rilevazione dell'umidità

- 1 Grove - Water Sensor per rilevare il livello dell'acqua
- 1 Grove - OLED Display 0.96" per visualizzare i parametri ambientale attuali
- 1 Grove - Relay per permettere l'attivazione dell'elettropompa
- 1 elettropompa Brunner da 12V 10 l/min
- 1 shield Arduino MKR WiFi 1010 per l'elaborazione del workflow di lavoro, la connessione alla piattaforma cloud di Arduino e l'invocazione dei servizi di previsione meteo
- 1 board Arduino MKR GPS per la rilevazione della posizione
- 1 batteria da 9V

Unitamente a tali componenti, vi è la possibilità di controllare il dispositivo mediante la piattaforma IoT di Arduino attraverso la seguente dashboard:



Dashboard di controllo su PC



Dashboard di controllo su dispositivo mobile

La dashboard è composta da diversi elementi grafici che permettono di:

1. controllare lo stato del livello dell'acqua nel serbatoio da cui viene attinta (Water level)
2. controllare lo storico dei valori di umidità del suolo mediante il grafico. Valori sotto la soglia di 300 danno luogo a irrigazione della pianta
3. ultimo valore di umidità del terreno rilevato (Soil Moisture)
4. controllo sull'uso delle previsione meteo per decidere se procedere con l'irrigazione (Use forecast)

5. possibilità di forzare manualmente l'irrigazione qualora se ne rilevi la necessità (Force water)
6. controllo del tempo di lettura delle informazioni dai sensori (Polling time ms)

Mediante tale dashboard è possibile, quindi, monitorare ed eseguire azioni in tempo reale sul dispositivo.

L'ambiente di sviluppo

Il codice relativo alla gestione del dispositivo è stato sviluppato con l'IDE ufficiale di Arduino (Arduino IDE) versione 2.0.
Unitamente al software di sviluppo, sono state utilizzate le seguenti librerie ufficiali:

- Arduino JSON
- WIFININA
- SeeedOLED
- ArduinoCloudThing
- ArduinoIoTCloud
- ArduinoMKRGPS

Tali librerie sono necessarie per poter compilare il software di gestione.

Vengono riportati di seguito alcuni link di documentazione dei sensori utilizzati:

- https://wiki.seeedstudio.com/Grove-Moisture_Sensor
- https://wiki.seeedstudio.com/Grove-Water_Sensor
- https://wiki.seeedstudio.com/Grove-Relay

Il codice è presente su repository Git ospitato sul sito BitBucket.

Il codice

Il codice del progetto è racchiuso in uno sketch diviso principalmente in 5 parti:

- dichiarazione variabili globali e costanti
- funzione di setup
- funzione di loop
- funzioni ausiliarie
- funzione di integrazione con la piattaforma IoT

Il cuore dell'algoritmo è la funzione di loop, realizzata mediante un watchdog (senza uso della funzione delay) implementato tramite l'uso della funzione millis.
In tal modo il flusso di esecuzione risulta essere molto pronto ad eventuali eventi provenienti dall'utente mediante azioni sulla dashboard.

Il codice della versione 2.0 è condiviso nell'Appendice A.

Appendice A

```
/*
* Progetto Aqua&Terra v2.0
* Codice sorgente
*/

#include <Wire.h>
#include <SeeedOLED.h>
#include <WiFiNINA.h>
#include "thingProperties.h"
#include <ArduinoJson.h>
#include <Arduino_MKRGPS.h>

#define WATER_KO 1
#define WATER_OK 0

#define DRY_SOIL_MIN 0
#define DRY_SOIL_MAX 300
#define HUMID_SOIL_MIN 300
#define HUMID_SOIL_MAX 700
#define WATER_SOIL_MIN 700
#define WATER_SOIL_MAX 950

#define NORMAL_POLLING 60000               // Data is updated every minute
#define HIGH_FREQ_POLLING 10000            // During water plants the polling time
is 10 seconds
#define WEATHER_FORECAST_POLLING 10800000  // Update weather forecast every 3hrs

#define WEATHER_API_KEY "77aa52994dce79dd192770375870e9da"  // OpenWeatherMap API
key

// In case no GPS
#define DEFAULT_LATITUDE 42.0
#define DEFAULT_LONGITUDE 12.0

// WIFI params

char ssid[] = SECRET_SSID;           // your network SSID (name)
char pass[] = SECRET_OPTIONAL_PASS;  // your network password (use for WPA, or use
as key for WEP)
int keyIndex = 0;                    // your network key Index number (needed only
for WEP)
```

```
int status = WL_IDLE_STATUS;
// if you don't want to use DNS (and reduce your sketch size)
// use the numeric IP instead of the name for the server:
//IPAddress server(74,125,232,128);  // numeric IP for Google (no DNS)
char server[] = "api.openweathermap.org";  // name address for Google (using DNS)

// Initialize the Ethernet client library
// with the IP address and port of the server
// that you want to connect to (port 80 is default for HTTP):

WiFiClient client;

// Soil moisture params

const int soilMoisturePin = A1;  // Soil Moisture sensore Analog Pin
int soilMoistureValue = 0;       // Initial value

// Water level sensor params

const int water_sensor_pin = 1;
bool water_level;

// Relay params
const int relay_pin = 3;

// Other variables

unsigned long timestamp = millis();          // Initial timestamp
unsigned long weather_timestamp = millis();  // Weather forecast timestamp
String last_weather_forecast;
bool useForecastLocal;

int polling_time_local = NORMAL_POLLING;


// GPS default lacation
float latitude = DEFAULT_LATITUDE;
float longitude = DEFAULT_LONGITUDE;


void setup() {
 Serial.begin(9600);

 // Defined in thingProperties.h
 initProperties();
```

```
 // Connect to Arduino IoT Cloud
 ArduinoCloud.begin(ArduinoIoTPreferredConnection);

 /*
    The following function allows you to obtain more information
    related to the state of network and IoT Cloud connection and errors
    the higher number the more granular information you'll get.
    The default is 0 (only errors).
    Maximum is 4
*/
 setDebugMessageLevel(4);
 ArduinoCloud.printDebugInfo();

 // Pins setup
 pinMode(water_sensor_pin, INPUT);
 pinMode(relay_pin, OUTPUT);

 // Components setup
 OLED_setup();
 WIFISetup();
 GPSSetup();

 ReadValuesFromSensors();

 DisplayValues();

 useForecast = true;
 useForecastLocal = useForecast;

 GetWeatherData(latitude, longitude);

 polling_time = polling_time_local;
}

void loop() {
 // Update variables & states on Arduino Cloud
 ArduinoCloud.update();


 // Check if it's time to give water to plants
 if (millis() - timestamp >= polling_time_local) {  // Time to update values is
over

   ReadValuesFromSensors();
```

```
   DisplayValues();

   // If no rain is expected or on weather data error or force water plants.
   if (forceWater || GetWeatherData(latitude, longitude) <= 0) {

     if (water_level == WATER_OK) {  // Check the water level
       Serial.println("Water level is OK");
       if (forceWater || soilMoistureValue <= DRY_SOIL_MAX) {  // If soil moisture
level is low or force water plants
         Serial.println("Start water!!!");
         digitalWrite(relay_pin, HIGH);  // Pump is ON
         polling_time_local = HIGH_FREQ_POLLING;
       } else {
         digitalWrite(relay_pin, LOW);  // Pump is OFF
         Serial.println("Stop water!!!");
         forceWater = false;
         polling_time_local = polling_time;
       }
     } else {  // No water
       Serial.println("Water level is KO");
       digitalWrite(relay_pin, LOW);  // Pump is OFF
       forceWater = false;
       polling_time_local = polling_time;
     }
   }

   timestamp = millis();  // Current timestamp is updated
 }
 delay(10);
}

// Utility functions

// Function used to read values from the sensors and update params from cloud
void ReadValuesFromSensors() {
 // Read water level
 water_level = digitalRead(water_sensor_pin);
 waterLevelOK = !water_level;
 Serial.print("Water level ");
 Serial.println(!water_level);

 soilMoistureValue = analogRead(soilMoisturePin);
 soilMoisture = soilMoistureValue;
 Serial.print("Soil moisture ");
```

```
 Serial.println(soilMoistureValue);

 // Getting GPS position
 if (GPS.available()) {
   Serial.println("GPS available. Getting position");
   latitude = GPS.latitude();
   longitude = GPS.longitude();
 }
}

// Function used to display sensors values to OLED display
void DisplayValues() {

 String soilMessage = "Soil Moist: ";
 String gpsMessage = "GPS ";

 // Display soil moisture values
 SeeedOled.clearDisplay();
 SeeedOled.setTextXY(0, 0);
 soilMessage.concat(String(soilMoistureValue));
 SeeedOled.putString(soilMessage.c_str());  //print soil data converted

 // Water level
 SeeedOled.setTextXY(2, 0);
 SeeedOled.putString("Water level ");
 SeeedOled.putNumber(!water_level);

 // Display GPS values
 SeeedOled.setTextXY(4, 0);
 gpsMessage.concat(String(latitude));
 gpsMessage.concat(",");
 gpsMessage.concat(String(longitude));
 SeeedOled.putString(gpsMessage.c_str());
}

// GPS board setup
void GPSSetup() {

 // GPS Initialization
 if (!GPS.begin()) {
   Serial.println("Failed to initialize the GPS!");
   Serial.println("GPS not available. Using default position: latitude " +
String(DEFAULT_LATITUDE) + ", longitude " + String(DEFAULT_LONGITUDE));
 } else {
   Serial.println("GPS initialized");
```

```
 }
}

// OLED DISPLAY setup
void OLED_setup() {
 Wire.begin();  //initialize I2C in master mode

 SeeedOled.init();  //initialize the OLED

 SeeedOled.clearDisplay();  //clear the screen and set start position to top left
corner

 SeeedOled.setNormalDisplay();  //Set display to normal mode (i.e non-inverse mode)

 SeeedOled.setPageMode();
}

void WIFISetup() {
 // check for the WiFi module:

 if (WiFi.status() == WL_NO_MODULE) {

   Serial.println("Communication with WiFi module failed!");

   // don't continue

   while (true)
     ;
 }

 String fv = WiFi.firmwareVersion();

 if (fv < WIFI_FIRMWARE_LATEST_VERSION) {

   Serial.println("Please upgrade the firmware");
 }

 // attempt to connect to Wifi network:

 while (status != WL_CONNECTED) {

   Serial.print("Attempting to connect to SSID: ");

   Serial.println(ssid);
```

```
    // Connect to WPA/WPA2 network. Change this line if using open or WEP network:

    status = WiFi.begin(ssid, pass);

    // wait 10 seconds for connection:

    delay(10000);
  }

  Serial.println("Connected to wifi");
}

/*
Function used to get weather data forecasts
@param lat latitude of the device
@param lon longitude of the device
@return 0 if no rain, 1 if rain or -1 on error
*/
int GetWeatherData(double lat, double lon) {

  if (!useForecastLocal) {  // Don't want to use weather forecasts
    Serial.println("Forecast service is disabled");
    return -2;
  }

  // Time to update weather forecast is not over
  if (last_weather_forecast.length() > 0 && millis() - weather_timestamp <=
WEATHER_FORECAST_POLLING) {
    Serial.println("No need to update the weather forecast");

    Serial.print("Weather forecast is ");
    Serial.println(last_weather_forecast);

    SeeedOled.setTextXY(6, 0);
    SeeedOled.putString("Forecast ");
    SeeedOled.putString(last_weather_forecast.c_str());

    if (last_weather_forecast == "Rain") {
      return 1;
    } else {
      return 0;
    }
  }

  // Update weather forecast
```

```
 weather_timestamp = millis();  // Update weather time stamp

 String latS = String(lat);
 String lonS = String(lon);

 String url = "GET /data/2.5/forecast?lat=" + latS + "&lon=" + lonS + "&appid=" +
WEATHER_API_KEY + "&cnt=1 HTTP/1.1";

 Serial.println("\nGetting weather data");

 if (client.connect(server, 80)) {

   Serial.println("connected to server");

   // Make a HTTP request:

   client.println(url);

   client.println(F("Host: api.openweathermap.org"));

   client.println(F("Connection: close"));

   if (client.println() == 0) {
     Serial.println(F("Failed to send request"));
     client.stop();
     return -1;
   }

   // Check HTTP status
   char status[32] = { 0 };
   client.readBytesUntil('\r', status, sizeof(status));
   if (strcmp(status, "HTTP/1.1 200 OK") != 0) {
     Serial.print(F("Unexpected response: "));
     Serial.println(status);
     client.stop();
     return -1;
   }

   // Skip HTTP headers
   char endOfHeaders[] = "\r\n\r\n";
   if (!client.find(endOfHeaders)) {
     Serial.println(F("Invalid response"));
     client.stop();
     return -1;
   }
```

```
  // Convert to JSON object

  StaticJsonDocument<2048> doc;

  DeserializationError error = deserializeJson(doc, client);

  if (error) {
    Serial.print("deserializeJson() failed: ");
    Serial.println(error.c_str());
    return -1;
  }

  const char* cod = doc["cod"];  // HTTP code should be "200" if request is OK

  if (strcmp(cod, "200") == 0) {
    Serial.println("Weather data request OK");
    // Process data

    JsonObject weatherData = doc["list"][0];

    const char* forecast = weatherData["weather"][0]["main"];

    Serial.print("Weather forecast is ");
    Serial.println(String(forecast));

    SeeedOled.setTextXY(6, 0);
    SeeedOled.putString("Forecast ");
    SeeedOled.putString(forecast);

    last_weather_forecast = forecast;

    if (last_weather_forecast == "Rain") {
      Serial.println("Rain is expected");
      return 1;  // Rain is expected
    }

  } else {
    Serial.println("Error on getting weather data");
    return -1;
  }
}

Serial.println("No rain is expected");
return 0;  // No rain
```

```
}



/*
 Since ForceWater is READ_WRITE variable, onForceWaterChange() is
 executed every time a new value is received from IoT Cloud.
*/
void onForceWaterChange() {
 if (forceWater) {
   Serial.println("Force water plants!!!");
   digitalWrite(relay_pin, HIGH);  // Pump is ON
   polling_time_local = HIGH_FREQ_POLLING;
 } else {
   Serial.println("Stop water plants");
   digitalWrite(relay_pin, LOW);  // Pump is OFF
   polling_time_local = polling_time;
 }
}

/*
 Since UseForecast is READ_WRITE variable, onUseForecastChange() is
 executed every time a new value is received from IoT Cloud.
*/
void onUseForecastChange() {
 useForecastLocal = useForecast;
 Serial.print("Weather forecast use is ");
 Serial.println(useForecastLocal);
}



/*
 Since PollingTime is READ_WRITE variable, onPollingTimeChange() is
 executed every time a new value is received from IoT Cloud.
*/
void onPollingTimeChange() {
 if (polling_time_local != polling_time) {
   Serial.print("Polling time changed from " + String(polling_time_local) + " to "
+ String(polling_time) + " ms");
   // Polling time has changes
   polling_time_local = polling_time;  // In milliseconds
 }
}
```