Agri-Stick

urządzenie do monitorowania parametrów gleby w rolnictwie

Agri-Stick to prosty zespół sensorów podstawowych parametrów gleby, które przydatne są w rolnictwie. Urządzenie to zostało opracowane przez Kavinkumara Nkl z Indii, jako sposób na proste i niedrogie zdalne monitorowanie upraw – w ten sposób rolnicy w Indiach nie muszą jeździć na pola, żeby sprawdzić, czy konieczne jest włączenie systemu nawadniającego czy nie. Rozwiązanie opisane w tym artykule może służyć jako inspiracja do opracowania dowolnego systemu akwizycji danych i monitorowania środowiska w oparciu o infrastrukturę chmurową.

Rolnicy spędzają niekończące się godziny na jazdy, po swoich polach, aby sprawdzić stan swoich upraw i ręcznie obsługiwać systemy nawadniające. Jest to czas i praca wymagające znacznych nakładów. Wykorzystanie Internetu Rzeczy (IoT) do dostarczania łatwych w użyciu i wysoce niezawodnych sensorów do precyzyjnego monitorowania i kontroli nawadniania jest jednym z potencjalnych rozwiązań tego problemu.

W poniższym artykule prezentujemy konstrukcję Kavinkumara Nkl – Agri-Stick, która jest zespołem sensorów do pomiaru czterech podstawowych, ważnych dla rolników wartości – temperatury i wilgotności gleby oraz temperatury i wilgotności powietrza. Wartości te są przechowywane w chmurze na serwerach AWS (Amazon), gdzie przesyłane są poprzez interfejs LoRa. W systemie wykorzystano komputer jednopłytkowy Raspberry Pi jako jednokanałową bramkę. Wartości zapisane w chmurze mogą być monitorowane w czasie rzeczywistym zarówno z przeglądarki internetowej, jak i telefonu komórkowego. Umożliwia to rolnikom podejmowanie decyzji na podstawie danych zebranych w terenie. W ten sposób oszczędza się czas i pieniądze.

Wykorzystując czujniki w Agri-Stick i analizując zebrane dane, można uzyskać informacje m.in. na temat nawożenia gleby, wykrywać szkodniki i planować lepiej uprawy, opryski, irygację i zbiory. Zapobiega to nadmiernemu podlewaniu, ogranicza choroby upraw, potwierdza dostarczanie wody (działanie systemu irygacyjnego), a także oszczędza wodę i pracę ludzką.

Jakie elementy są potrzebne do budowy systemu

Agri-Stick oparty jest na module Arduino Nano z mikrokontrolerem ATMega328P i wszystkimi podstawowymi peryferiami. Do modułu tego podłączone są następujące sensory:

- DS18B20 – termometr elektroniczny w wodoszczelnej obudowie; do pomiaru temperatury gleby,
- pojemnościowy czujnik wilgotności gleby,
- DHT22 – zintegrowany termometr i higrometr elektroniczny; do pomiaru temperatury i wilgotności powietrza.

Wszystkie te sensory podłączone są do Arduino. Do tegoż modułu podłączony jest także transceiver LoRa AI Thinker Ra-02, który umożliwia komunikację z bramką (opisaną w dalszej części artykułu).

Elektroniczny termometr do pomiaru temperatury gruntu

Czujnik mierzący temperaturę gleby powinien być odporny na działanie wody i na korozję. Dlatego wykorzystana jest w tym projekcie specjalna sonda, wyposażona w szczelną, metalową obudowę, jak widać na **fotografii 1**. We wnętrzu sondy znajduje się cyfrowy termometr DS18B20 firmy Dallas Semiconductor. Jest to kompaktowy sensor temperatury, wyposażony w interfejs 1-Wire.

Fotografia 1. Elektroniczny sensor temperatury w wodoszczelnej obudowie

Sensor ma tylko trzy wyprowadzenia – zasilanie, masę oraz linię danych. Tylko jeden przewód potrzebny jest do komunikacji z tym układem (stąd nazwa interfejsu, jaki wykorzystuje). Podłączenie sondy do modułu Arduino jest niezwykle proste – zasilanie podłączamy do 5 V, a masę do masy. Linię danych termometru łączymy z pinem D5 w module Arduino.

Pojemnościowy sensor wilgotności gleby

Czujniki wilgotności gleby mierzą objętość wody, jaka znajduje się w zadanej, stałej objętości gleby. Analogicznie jak w przypadku czujnika temperatury, sensor wilgotności gleby powinien być również wodoodporny i zabezpieczony przed korozją. Wszystkie te wymagania spełnia czujnik

Fotografia 2. Pojemnościowy sensor do pomiaru wilgotności

pojemnościowy. Jest on o wiele wytrzymalszy i odporniejszy na warunki otoczenia niż czujnik rezystancyjny, jako że nie musi bezpośrednio stykać się z wilgotną glebą metalowymi elementami. Sensor tego rodzaju zaprezentowano na **fotografii 2**.

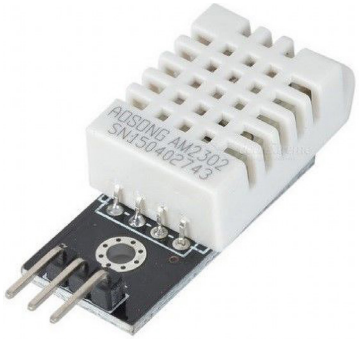
Czujnik taki składa się z pojedynczej sondy zawierającej oddzielne ścieżki miedziane (masa i zasilanie) pokryte izolującą maską lutowniczą. Element ten zachowuje się jak kondensator płaski. Warstwa dielektryczna zapobiega przepływowi prądu elektrycznego. Rolę dielektryka, pomiędzy okładkami kondensatora, odrywa w przypadku tego rodzaju sondy gleba. Zmienia się jej współczynnik przenikalności elektrycznej wraz ze zmianą poziomu jej wilgotności. W ten sposób zmiana wilgotności powoduje zmianę pojemności sensora, która może być mierzona przez moduł Arduino. Zaprezentowany na **fotografii 3** sensor wyposażony jest w układ elektroniczny konwertujący pojemność do napięcia, dzięki czemu na wyjściu z sensora obecne jest jedynie napięcie (o amplitudzie od 0 V do 3 V) proporcjonalne do wilgotności.

Podłączenie opisanego sensora do Arduino jest bardzo proste. Zasilanie i masę sensora łączy się, odpowiednio, z napięciem 5 V i masą modułu, a napięcie wyjściowe podawać będziemy na wejście A1 modułu.

Scalony miernik wilgotności i temperatury powietrza

Do pomiaru temperatury i wilgotności powietrza w projekcie wykorzystany został sensor DHT22 (czasami opisywany jako AM2302) dostępny w postaci modułu. Zastosowano gotowy moduł z własną płytką drukowaną dla uproszczenia konstrukcji systemu.

DHT22 to zintegrowany cyfrowy termometr i higrometr. W jednej obudowie znajdują się dwa niezależne sensory, każdy o rozdzielczości 8 bit. Każdy z czujników w układzie ma własny model kompensacji i jest precyzyjnie skalibrowany. Parametry kalibracyjne zapisywane są w sensorze, więc wszystkie zbierane pomiary cechuje wysoki stopień dokładności – katalogowo poniżej 0,5°C i 2%RH.



Fotografia 3. Sensor wilgotności i temperatury DHT22

Moduł wyposażony jest w interfejs 1-Wire, co umożliwia proste podłączenie sensorów do modułu Arduino. Linie zasilania i masy podłączane są odpowiednio, do zasilania i masy, a linia danych podłączana jest do pinu D4 modułu Arduino.

Łączność – dlaczego LoRa

Krytycznym składnikiem wszystkich systemów Internetu Rzeczy jest komunikacja. Zazwyczaj w systemach tego rodzaju implementuje się systemy łączności bezprzewodowej – komunikacja radiowa jest o wiele wygodniejsza w tego rodzaju implementacjach. Wykorzystuje się zazwyczaj interfejsy takie jak Bluetooth czy Wi-Fi. Ostatnio popularyzują się także interfejsy bezprzewodowe specjalnie dedykowane do systemów Internetu Rzeczy, takie jak LoRa, Sigfox czy NB-IoT.

Urządzenia takie jak Agri-Stick nie mają wielkich wymagań co do prędkości transmisji, za to wymagają często bardzo dużego zasięgu. Wymagania te spełnia system LoRa, oferując wielokilometrowy zasięg przy prędkości transmisji poniżej 50 kb/s. Dodatkowym wymaganiem jest niski pobór mocy – Agri-Stick zasilany jest z baterii, więc moduł transceivera, przesyłającego dane z sensorów do chmury, nie może konsumować zbyt wiele energii. Także to wymaganie spełniane jest przez interfejs LoRa.

LoRa (skrót od *Long Range*, daleki zasięg) to technika modulacji sygnału z widmem rozproszonym, wywodząca się z technologii CSS. Z technicznego punktu widzenia jest to rodzaj modulacji radiowej służący do kodowania informacji za pomocą sygnału świergotowego i wieloznakowego formatu danych. Nazwa LoRa odnosi się również do systemów, które obsługują tego rodzaju modulację, w tym układów, transceiverów i bramek.

LoRa jest w istocie sprytnym sposobem na uzyskanie bardzo dobrej czułości odbiornika i niskiej bitowej stopy błędów (BER) z wykorzystaniem relatywnie niedrogich układów. Oznacza to, że aplikacje o małej szybkości transmisji danych mogą uzyskać znacznie większy zasięg przy użyciu LoRa niż przy użyciu innych porównywalnych cenowo technologii radiowych. Dodatkowo, systemy tego rodzaju charakteryzują się niezwykle niskim poborem energii z uwagi na niewielką moc nadajników potrzebnych do zapewnienia transmisji danych na duże odległości.

Technologia LoRa może być wykorzystywana przez publiczne, prywatne i hybrydowe sieci. Sieci LoRa można z łatwością podłączyć do istniejącej infrastruktury i umożliwić tanie aplikacje IoT z zasilaniem bateryjnym.

Transceiver LoRa

Do komunikacji Agir-Stick wykorzystuje interfejs LoRa. Wykorzystano transceiver

Fotografia 4. Transceiver LoRa wykorzystany w urządzeniu

oparty na układzie scalonym SX1278 – Ai Thinker Ra-02, pokazany na **fotografii 4**. Wykorzystuje on częstotliwość 433 MHz do komunikacji z innymi modułami i bramką. Komunikuje się z mikrokontrolerem w systemie za pomocą interfejsu SPI (Serial Peripheral Interface).

Do modułu należy dołączyć stosowną antenę, która dedykowana jest do pracy w pasmie 433 MHz. W zależności od wykorzystanej obudowy dobrać możemy odpowiednią antenę na to dosyć popularne pasmo.

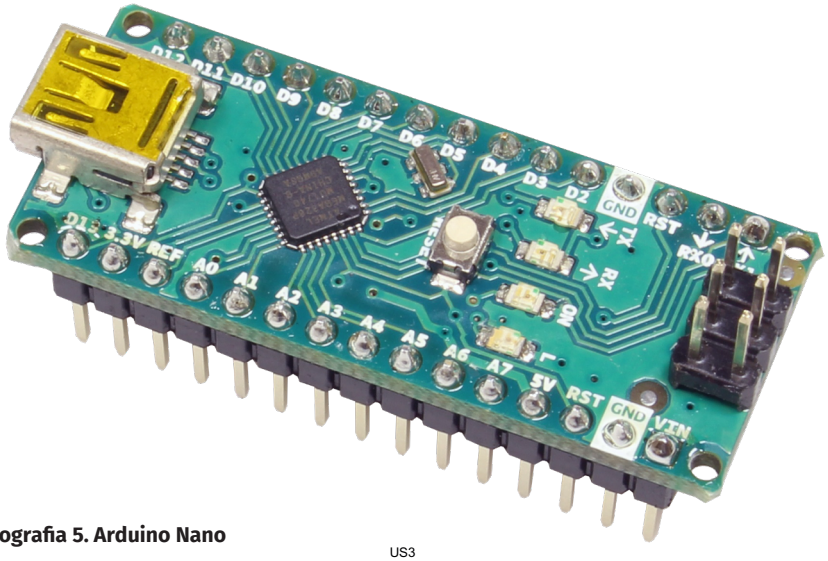
Zasilanie bateryjne i ładowarka

Zasilacz układu oparty jest na pojedynczym ogniwie jonowo-litowym do zapewnienia systemowi długiego czasu pracy przy kompaktowych rozmiarach. System zasilania w Agri-Stick składa się z:

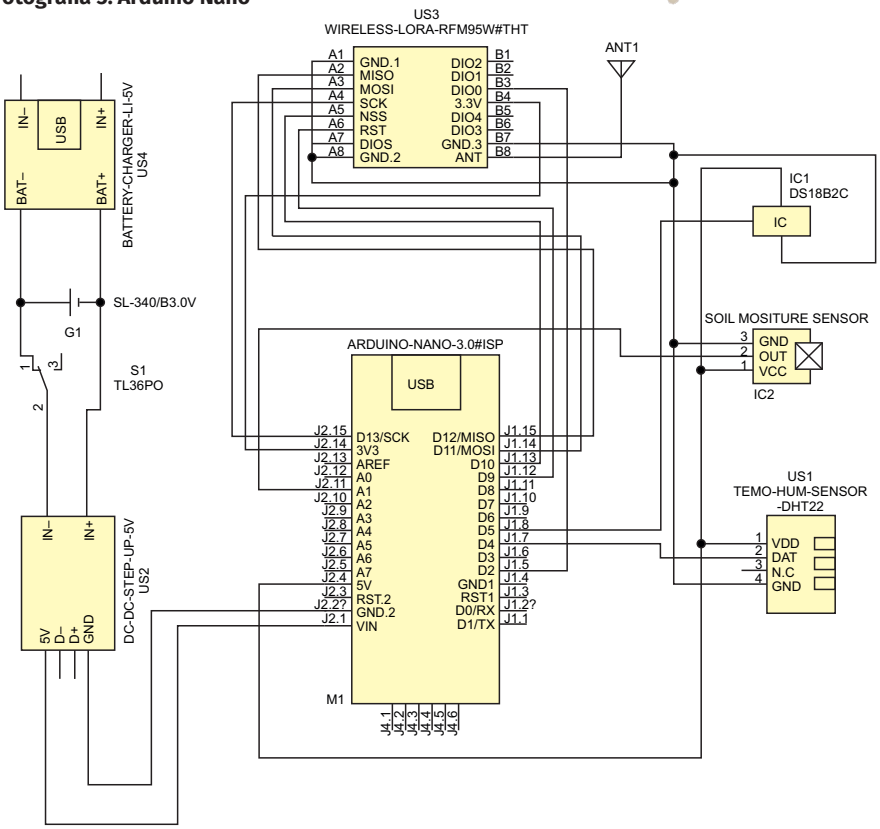
- Pojedynczego ogniwa litowo-jonowego w rozmiarze 18650 o pojemności 2200 mAh.
- Przetwornicy DC-DC typu boost, która generuje napięcie zasilania dla systemu.
- Ładowarki do ogniw litowo-jonowych, opartej na układzi scalonym TP4056.

Nominalne napięcie ogniwa litowo-jonowego wynosi 3,7 V, a maksymalne napięcie, do którego naładować można taki akumulator, wynosi 4,2 V. Z kolei minimalne napięcie, do jakiego rozładować można ogniwo wynosi 3,3 V. Nie powinno się przekraczać tych granicznych napięć z uwagi na negatywny wpływ takiego działania na ogniwo. Ładowarka i przetwornica dbają, aby ich nie przekroczyć podczas pracy.

Przetwornica DC-DC stabilizuje napięcie 5 V na swoim wyjściu. Przetwornica może pracować z szerokim zakresem napięć wejściowych, dzięki czemu jest w stanie stabilizować napięcie zasilania systemu dla dowolnego napięcia ogniwa zasilającego system.



Fotografia 5. Arduino Nano



Rysunek 6. Schemat połączenia poszczególnych modułów w Agri-Stick

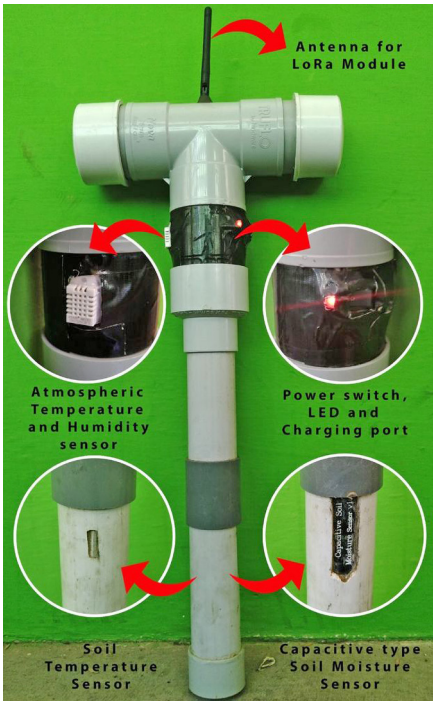
Ładowarka TP4056 jest scalonym, kompletnym systemem ładowania pojedynczego ogniwa litowo-jonowego. Układ ten jest odpowiedzialny za kontrolę wszystkich napięć i prądów podczas ładowania ogniwa. Wejściem ładowarki jest port USB, z którego system pobiera zasilanie.

Arduino – kontroler systemu

Do sterowania całym systemem – zbierania danych z poszczególnych sensorów i wysyłania ich poprzez interfejs LoRa do bramki – wykorzystany został moduł Arduino Nano. Przykładowy moduł tego rodzaju pokazano na **fotografii 5**. Podłączenie poszczególnych sensorów do tego modułu opisano powyżej. Transceiver LoRa, opisany powyżej, podłączany jest do Arduino Nano poprzez interfejs SPI.

Połączenie systemu

Po skompletowaniu wszystkich elementów systemu można przystąpić do integracji wszystkich modułów ze sobą. Poszczególne moduły wyposażone są w miejsce na wlutowanie goldpinów lub bezpośrednio



Fotografia 7. Zdjęcie gotowej konstrukcji w obudowie

przewodów, do połączenia ze sobą poszczególnych elementów. Jak połączyć ze sobą poszczególne moduły, pokazano na **rysunku 6**.

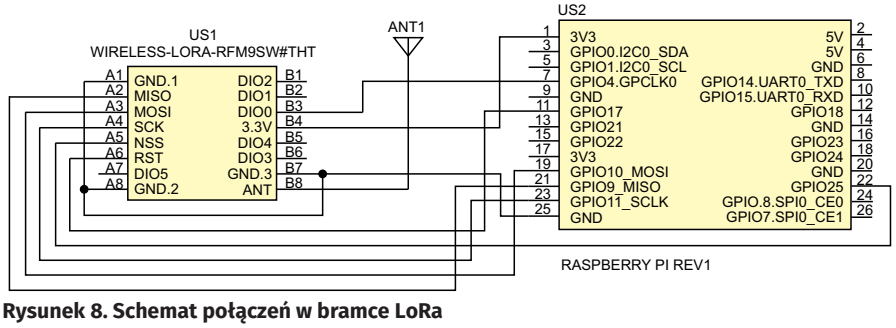
Po połączeniu wszystkich elementów ze sobą całość układu można zainstalować w obudowie. Autor konstrukcji, jak widać na **fotografii 7**, umieścił cały system w plastikowej rurze, co pozwala zachować szczelność, a jednocześnie jest wygodne w użyciu – wystarczy wkopać część rury w ziemię, tak aby termometr i higrometr mogły monitorować parametry gruntu. Po złożeniu systemu i przetestowaniu montażu przystąpić można do konfiguracji środowiska w chmurze i oprogramowania samego modułu Arduino.

Bramka LoRa

Autor opisywanej konstrukcji wykorzystuje zbudowaną samodzielnie bramkę LoRa, opartą na komputerze jednopłytkowym Raspberry Pi i module z transceiverem LoRa. Na **rysuku 8** pokazano schemat połączeń tych elementów.

Na Raspberry Pi uruchomiony jest prosty skrypt, kontrolujący działanie bramki. Znajdziemy go na **listingu 1**.

Aby skrypt ten działał poprawnie, konieczne jest utworzenie odpowiedniej bazy



Rysunek 8. Schemat połączeń w bramce LoRa

Listing 1. Skrypt bramki LoRa na Raspberry Pi

```
from time import sleep
from SX127x.LoRa import *
from SX127x.LoRaArgumentParser import LoRaArgumentParser
from SX127x.board_config import BOARD
import sqlite3
import datetime
import paho.mqtt.publish as publish
# Nazwa bazy danych SQLite
DB_Name =  „/home/pi/FYP/agristick.db”
BOARD.setup()
parser = LoRaArgumentParser(„Continous LoRa receiver.”)
# Klasa menedżera bazy danych
class DatabaseManager():
        def __init__(self):
                self.conn = sqlite3.connect(DB_Name)
                self.conn.execute('pragma foreign_keys = on')
                #self.conn.execute('pragma journal_mode = wal;')
                self.conn.commit()
                self.cur = self.conn.cursor()

        def add_del_update_db_record(self, sql_query, args=()):
                self.cur.execute(sql_query, args)
                self.conn.commit()
                return

        def __del__(self):
                self.cur.close()

class LoRaRcvCont(LoRa):
    def __init__(self, verbose=False):
        super(LoRaRcvCont, self).__init__(verbose)
        self.set_mode(MODE.SLEEP)
        self.set_dio_mapping([0] * 6)

    def on_rx_done(self):
        BOARD.led_on()
        self.clear_irq_flags(RxDone=1)
        payload = self.read_payload(nocheck=True)
        payload = payload[:-1]

        d = datetime.datetime.now()
        d = str(d)
        mystring = „”
        for char in payload:
                mystring = mystring + chr(char)
        #print (mystring)
        soil_temp = mystring[0:7]
        soil_moist = mystring[7:10]
        atmp_temp = mystring[10:17]
        atmp_hum = mystring[17:24]
        print(„datetime: „ + d)
        print(„Soil Temperature: „ + soil_temp)
        print(„Soil Moisture: „ + soil_moist)
        print(„Atmospheric Temperature: „ + atmp_temp)
        print(„Atmospheric Humidity: „ + atmp_hum)
        stringformqtt = d + ',' + soil_temp + ',' + soil_moist + ',' + atmp_temp + ',' + atmp_hum
        print(stringformqtt)
        dbObj = DatabaseManager()
        dbObj.add_del_update_db_record(„insert into sendata (date_time,soil_temp,soil_moist,atmp_temp,atmp_hum)
        values (?,?,?,?,?)”,[d,float(soil_temp),int(soil_moist)$
        del dbObj
        print („Inserted Data into SENSOR DATA Database.”)
        publish.single(„agristick1”, str(stringformqtt), hostname=”13.232.96.33”, auth = {'username':”ubuntu”,  'password':”ceglaeee”})
        self.set_mode(MODE.SLEEP)
        self.reset_ptr_rx()
        BOARD.led_off()
        self.set_mode(MODE.RXCONT)

    def on_tx_done(self):
        print(„\nTxDone”)
        print(self.get_irq_flags())

    def on_cad_done(self):
        print(„\non_CadDone”)
        print(self.get_irq_flags())

    def on_rx_timeout(self):
        print(„\non_RxTimeout”)
        print(self.get_irq_flags())

    def on_valid_header(self):
        print(„\non_ValidHeader”)
        print(self.get_irq_flags())

    def on_payload_crc_error(self):
        print(„\non_PayloadCrcError”)
        print(self.get_irq_flags())

    def on_fhss_change_channel(self):
        print(„\non_FhssChangeChannel”)
        print(self.get_irq_flags())

    def start(self):
        self.reset_ptr_rx()
        self.set_mode(MODE.RXCONT)
        while True:
            sleep(.5)
            rssi_value = self.get_rssi_value()
            status = self.get_modem_status()
            sys.stdout.flush()
            sys.stdout.write(„\r%d %d %d\n” % (rssi_value, status['rx_ongoing'], status['modem_clear']))

lora = LoRaRcvCont(verbose=False)
args = parser.parse_args(lora)
lora.set_mode(MODE.STDBY)
lora.set_pa_config(pa_select=1)
lora.set_freq(433)
lora.set_bw(BW.BW125)
lora.set_spreading_factor(7)
lora.set_rx_crc(True)
```

danych, do której mógłby on zapisywać zbierane pomiary. Aby wygenerować taką bazę danych, musimy skorzystać ze skryptu, umieszczonego w **listingu 2**.

Oprogramowanie dla Arduino

Aby cały system mógł działać potrzebne jest oprogramowanie dla modułu Arduino. Zajmuje się ono zbieraniem i przetwarzaniem danych z sensorów, które mogą być następnie wysłane poprzez sieć LoRa do bramki. Kod programu, odpowiedzialnego za działanie modułu Arduino, znajduje się na **listingu 3**.

Konfiguracja Amazon Web Service (AWS)

Amazon Web Services (AWS) to bezpieczna platforma usług w chmurze, oferująca moc obliczeniową, przechowywanie baz danych, dostarczanie treści i inne funkcje. Miliony klientów wykorzystują obecnie produkty i rozwiązania w chmurze AWS do tworzenia zaawansowanych aplikacji o zwiększonej elastyczności, skalowalności i niezawodności. Amazon Elastic Compute Cloud (EC2) zapewnia serwery wirtualne – zwane instancjami – dla obliczeń. Usługa EC2 oferuje dziesiątki typów instancji o różnych możliwościach i rozmiarach, dostosowanych do konkretnych rodzajów obciążeń i aplikacji, takich jak zadania wymagające dużej

Listing 1. cd

```
lora.set_coding_rate(CODING_RATE.CR4_5)
assert(lora.get_agc_auto_on() == 1)

try:
    lora.start()
except KeyboardInterrupt:
    sys.stdout.flush()
    print(„”)
    sys.stderr.write(„KeyboardInterrupt\n”)
finally:
    sys.stdout.flush()
    print(„”)
    lora.set_mode(MODE.SLEEP)
    print(lora)
    BOARD.teardown()
```

ilości pamięci czy przyspieszone przetwarzanie danych. AWS zapewnia także narzędzie do automatycznego skalowania, które umożliwia dynamiczne zwiększanie pojemności czy ilości instancji w celu utrzymania wysokiej wydajności systemu. Amazon oferuje roczny darmowy pakiet instancji z pewnymi ograniczeniami. Dlatego też autor opisywanej konstrukcji wybrał AWS do obsługi części przetwarzania, które realizowane jest w chmurze.

W opisywanym projekcie wykorzystano instancję Ubuntu Server 18.04 LTS (HVM), SSD Volume Type w instancji typu t2.micro. Zawiera ona jeden wirtualny procesor i jeden GB pamięci RAM. Konfiguracja ta objęta jest rocznym bezpłatnym pakietem od Amazon dla nowych użytkowników. Instancja jest skonfigurowana z zabezpieczeniami, które kontrolując ruch przychodzący do serwera. Wykorzystana konfiguracja używa SSH do zdalnego dostępu do instancji, protokołów HTTP i HTTPS dla ruchu internetowego TCP dla MQTT. Porty powiązane z tymi protokołami to odpowiednio 22, 80, 443 i 1884.

Do instancji dostęp uzyskać można np. przez putty wraz z parą kluczy SSH. Używane są one do łączenia się z instancją bezpiecznym, szyfrowanym kanałem. Uzyskuje się je podczas konfiguracji instancji w Amazonie. Jest to podstawowy mechanizm zapewniający bezpieczne uwierzytelnianie użytkownika instancji.

Subskrypcja MQTT i webowa prezentacja danych

MQTT to lekki system publikowania i subskrybowania danych telemetrycznych. Można za jego pomocą publikować i odbierać wiadomości. MQTT to prosty protokół przesyłania komunikatów, zaprojektowany dla urządzeń o niskiej przepustowości danych. Dlatego jest to idealne rozwiązanie dla aplikacji systemów Internetu Rzeczy.

MQTT umożliwia wysyłanie poleceń do sterowania wyjściami, odczytywanie i publikowanie danych z sensorów i wiele innych. Do działania system ten potrzebuje brokera danych oraz klientów, którzy działają jako subskrybenci i publikatorzy danych.

W opisywanym systemie brama LoRa jest naszym publikatorem danych, a instancja AWS EC2 jest subskrybentem, a także działa jako broker MQTT. Brama Raspberry Pi publikuje wartości czujników do określonego tematu w sieci MQTT, takiego jak na przykład „agrstick1”. Jeśli korzystamy z wielu bramek, możemy publikować na różne tematy w jednej sieci. Broker MQTT przechwytuje te wartości w danym temacie i czeka, aż jakiś subskrybent będzie chciał danych na ten sam temat. Jeśli jakikolwiek klient subskrybuje ten sam temat, broker wyśle do niego zapisane wartości. Dlatego klient subskrybenta w instancji EC2 otrzyma wartości za każdym razem, gdy wydawca opublikuje wiadomość – nowe pomiary.

Dane na instancji AWS EC2 przechowywane są w postaci bazy danych SQLite, takiej samej jak lokalnie, na Raspberry Pi. Aby uruchomić na tej instancji taką bazę, musimy, podobnie jak na Raspberry Pi, uruchomić prosty skrypt. Zapisany jest on na **listingu 4**.

Aby system działał, potrzebna jest na instancji w AWS subskrypcja do MQTT.

Listing 2. Skrypt do tworzenia tabeli dla pomiarów z sensorów w bazie danych

```
import sqlite3
# Nazwa bazy danych SQLite
DB_Name =  „agristick.db”
# Schematy tabel bazy danych SQLite
TableSchema_SD=”””
drop table if exists sendata ;
create table sendata (
  id integer primary key autoincrement,
  date_time text,
  soil_temp float,
  soil_moist int,
  atmp_temp float,
  atmp_hum float
);
„””
# Musimy połączyć się z serwerem bazy, aby stworzyć tabele
conn = sqlite3.connect(DB_Name)
curs = conn.cursor()
# Tworzymy tabelę dla danych z sensorów
sqlite3.complete_statement(TableSchema_SD)
curs.executescript(TableSchema_SD)
# Zamykamy bazę danych
curs.close()
conn.close()
```

Listing 3. Kod programu dla modułu Arduino

```
#include <spi.h> // Do obsługi SPI
#include <lora.h> // Do obsługi modułu LoRa
#include <onewire.h> // Do obsługi DS18b20
#include <dallastemperature.h> Do obsługi DS18b20
#include <stdlib.h> // Dla funkcji floatToString (konwersja)
#include <dht.h>; // Do obsługi DHT22
#define DHTPIN 4     // Definicja pinu podłączenia DHT22
#define DHTTYPE DHT22   // Definicja typu układu DHT 22  (AM2302)
DHT dht(DHTPIN, DHTTYPE); // Inicjalizacja sensora DHT dla Arduini  16MHz
#define ONE_WIRE_BUS 5 // Definicja pinu podłączenia DS18b20
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
String data_string = „ „;
String data_string_p = „ „;
String temp_string = „ „;
String hum_string = „ „;
String airtemp_string = „ „;
String soilmois_string = „ „;
float soil_temp_float; // Przechowuje wartość temperatury gleby
float atm_hum;  // Przechowuje wartość wilgotności
float atm_temp; // Przechowuje wartość temperatury powietrza
int soil_mois; // Przechowuje wartość wilgotności gleby
// Funkcja do konwersji zmiennej typu float na string
String floatToString(float x, byte precision = 4) {
  char tmp[7];
  dtostrf(x, 0, precision, tmp);
  return String(tmp);
}

void setup()
{
  Serial.begin(9600);
  Serial.println(„ Welcome to AgriStick Sensor Node „);
  Serial.println(„ It includes \n Soil temperature,\n Soil moisture,\n Atmospheric
  temperature,\n Atmospheric humidity \n”);
  pinMode(A3, INPUT);
  pinMode(A4, INPUT);
  pinMode(A5, INPUT);
  dht.begin();
  sensors.begin();
  if (!LoRa.begin(433E6)) {
    Serial.println(„Starting LoRa failed!”);
  while (1);
  }
  LoRa.enableCrc();
}
void loop() {
  sensors.requestTemperatures();
  soil_temp_float = sensors.getTempCByIndex(0);
  atm_hum = dht.readHumidity();
  atm_temp = dht.readTemperature();
  soil_mois = analogRead(A1);
  if (soil_temp_float <= 0 || atm_hum <= 0 || atm_temp <= 0 || soil_mois <= 0)
  {
    return;
  }
  temp_string = floatToString(soil_temp_float);
  hum_string = floatToString(atm_hum);
  airtemp_string = floatToString(atm_temp);
  soilmois_string = String(soil_mois);
  data_string = ( temp_string + soilmois_string + airtemp_string + hum_string );
  data_string_p = ( „Soil Temp: „ + temp_string + „ Soil Moist: „ + soilmois_string
  + „ Atmp Temp: „ + airtemp_string + „ Atmp Hum: „ + hum_string );
  Serial.println(data_string_p);
  LoRa.beginPacket();
  uint8_t data[(data_string.length() + 2)];
  data_string.toCharArray(data, (data_string.length() + 2));
  LoRa.write(data, sizeof(data));
  Serial.println(„Packet sent”);
  LoRa.endPacket();
  delay(60000);
}
```

Aby takową uruchomić, konieczny jest kolejny skrypt, zaprezentowany na **listingu 5**. Musimy go jedynie uzupełnić podstawowymi danymi, takimi jak nazwa bazy danych, z której korzystamy, tematy, jakie chcemy subskrybować w MQTT oraz numer IP serwera.

Trzecim, ostatnim skryptem, jaki trzeba uruchomić na instancji w chmurze Amazona, jest webowy panel, pozwalający na prezentację danych. Do stworzenia strony wykorzystano pythonowe biblioteki plotly oraz dash.

Dash to framework do budowania aplikacji internetowych do analizy danych. Jest to wydajne środowisko do budowania aplikacji internetowych. Napisany na podbudowie frameworku Flask oraz Plotly.js i React.js, Dash jest idealny do budowania aplikacji do wizualizacji danych z wysoce niestandardowymi interfejsami użytkownika w czystym Pythonie. Aplikacje Dash są renderowane w przeglądarce internetowej, co pozwala udostępniać aplikacje za pośrednictwem jedynie adresów URL. Ponieważ aplikacje Dash są wyświetlane w przeglądarce internetowej, Dash jest z natury kompatybilny z wieloma platformami i urządzeniami mobilnymi.

Strona renderowana na podstawie kodu w **listingu 6** pokazana jest na **rysunku 9**. Jest to prosty sposób prezentacji danych, bez wizualnych fajerwerków, ale w zupełności wystaczy dla opisywanego zastosowania.

Listing 4. Skrypt inicjalizujący bazę danych na instancji AWS EC2

```
import sqlite3
# Nazwa bazy danych SQLite
DB_Name =  „agristick.db”
# Schemat tabeli w bazie danych SQLite
TableSchema_SD=”””
drop table if exists DATASEN ;
create table DATASEN (
  id integer primary key autoincrement,
  date_time text,
  soil_temp float,
  soil_moist int,
  atmp_temp float,
  atmp_hum float
);
# Musimy połączyć się z serwerem bazy, aby stworzyć tabele
conn = sqlite3.connect(DB_Name)
curs = conn.cursor()
# Tworzymy tabelę dla danych z sensorów
sqlite3.complete_statement(TableSchema_SD)
curs.executescript(TableSchema_SD)
# Zamykamy bazę danych
curs.close()
conn.close()
```

Listing 5. Skrypt subskrypcji do MQTT dla instancji AWS EC2

```
import paho.mqtt.subscribe as subscribe
import sqlite3
# Nazwa bazy danych SQLite
DB_Name = „/home/ubuntu/flaskapp/agristick.db”
sqliteConnection = sqlite3.connect(DB_Name)
sqliteConnection.execute(‘pragma journal_mode=wal;’)
cursorObject = sqliteConnection.cursor()
def on_message(client, userdata, message):
    mqttmessage = str(message.payload)
    topic = str(message.topic)
    mqttmessage = mqttmessage[2:-1]
    print(mqttmessage)
    dt = mqttmessage[0:26]
    st = float(mqttmessage[27:34])
    sm = int(mqttmessage[35:38])
    at = float(mqttmessage[39:46])
    ah = float(mqttmessage[47:54])
    print(mqttmessage)
    print(dt)
    print(st)
    print(sm)
    print(at)
    print(ah)
    sqliteConnection = sqlite3.connect(DB_Name)
    cursorObject = sqliteConnection.cursor()
    sqliteConnection.execute(‘pragma journal_mode=wal;’)
    cursorObject.execute(„INSERT INTO DATASEN (date_time,soil_temp,soil_moist,atmp_temp,atmp_hum) VALUES(?,?,?,?,?)”,[dt,st,sm,at,ah])
    cursorObject.execute(„COMMIT”)
    cursorObject.close()
    print („Inserted Data into DATASEN Database.”)
topics = [„agristick1”,”agristick2”]
subscribe.callback(on_message, topics, hostname=”Ip address of the server”,auth = {‘username’:”ubuntu”, ‘password’:”password set for MQTT”})
```

Listing 6. Kod odpowiadający za renderowanie strony www prezentującej dane z sensorów Agri-Stick

```
import dash_core_components as dcc
import dash_html_components as html
import plotly
from dash.dependencies import Input, Output
import sqlite3
import datetime
from flask import Flask
import requests
server = Flask(__name__)
server.secret_key =’test’
external_stylesheets = [‘https://codepen.io/chriddyp/pen/bWLwgP.css’]
app = dash.Dash(__name__,server=server, external_stylesheets=external_stylesheets)
app.layout = html.Div(style={‘backgroundColor’: ‘#7CFC00’},children=[
    html.H1(
        children=’Agri-Stick’,
        style = {
                ‘textAlign’:’center’,
                ‘color’ : ‘#000000’
        }),
    html.H2(children= ‘A product for better farming’,
        style={
                ‘textAlign’:’center’,
                ‘color’: ‘#FFFFFF’
        }),
        dcc.Graph(id=’live-update-graph’),
        dcc.Interval(
            id=’interval-component’,
            interval=1*3000, # in milliseconds
            n_intervals=3000
        )
        ])
@app.callback(Output(‘live-update-graph’, ‘figure’),
              [Input(‘interval-component’, ‘n_intervals’)])
def update_graph_live(n_intervals):
    # Nazwa naszej bazy danych – taka sama jak w innych skryptach
    conn = sqlite3.connect(‘/home/ubuntu/flaskapp/agristick.db’,check_same_thread=False)
    conn.execute(‘pragma journal_mode=wal;’)
    c = conn.cursor()
    c.execute(‘SELECT * FROM DATASEN ORDER BY id DESC LIMIT 10 OFFSET 0’)
    conn.commit()
    data = c.fetchall()
    conn.close()
    dts0 = data[9][1]
    dts1 = data[8][1]
    dts2 = data[7][1]
    dts3 = data[6][1]
```

Podsumowanie

Zaprezentowany system pozwala na implementację niemalże dowolnej

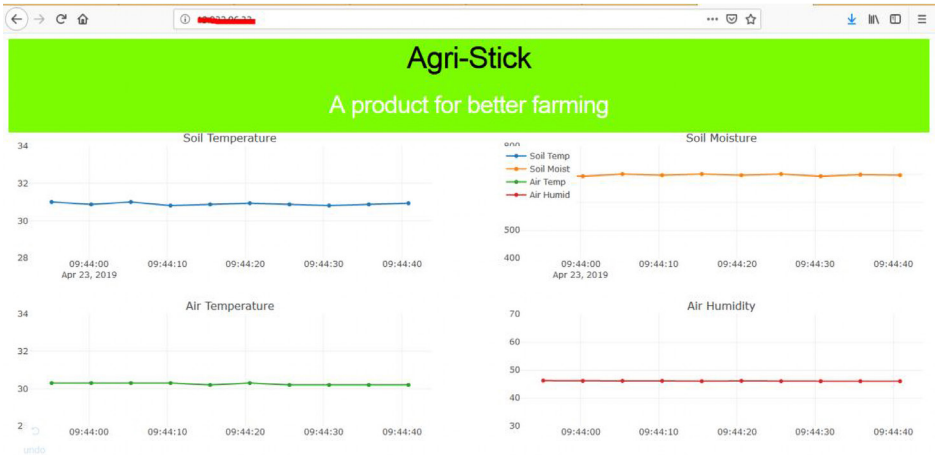
liczby rozproszonych urządzeń z sensorami. Mogą one być wykorzystywane do monitorowania dużej wielkości pola uprawnego, sadu itp. Dodatkowo, poszerzyć można wachlarz sensorów, dostępnych w urządzeniu. Oprócz termometrów i higrometrów, system uzupełnić można o czujniki azotu, fosforu i potasu w glebie, pehametry, oksymetry, mierniki zawartości dwutlenku węgla, nasłonecznienia etc. Jedynie dostępność sensorów i ich kompatybilność z ekosystemem Arduino jest granicą. Przedstawione powyżej skrypty są doskonałym punktem wyjścia do opracowania własnego systemu monitorowania środowiska na bazie o infrastruktury chmurową, oferowaną przez Amazona oraz technologii takich jak LoRa i MQTT.

Nikodem Czechowski

Źródło: ***http://bit.ly/30BorbC***

Listing 6. cd.

```
    dts4 = data[5][1]
    dts5 = data[4][1]
    dts6 = data[3][1]
    dts7 = data[2][1]
    dts8 = data[1][1]
    dts9 = data[0][1]
    dt0 = datetime.datetime.strptime(dts0,'%Y-%m-%d %H:%M:%S.%f')
    dt1 = datetime.datetime.strptime(dts1,'%Y-%m-%d %H:%M:%S.%f')
    dt2 = datetime.datetime.strptime(dts2,'%Y-%m-%d %H:%M:%S.%f')
    dt3 = datetime.datetime.strptime(dts3,'%Y-%m-%d %H:%M:%S.%f')
    dt4 = datetime.datetime.strptime(dts4,'%Y-%m-%d %H:%M:%S.%f')
    dt5 = datetime.datetime.strptime(dts5,'%Y-%m-%d %H:%M:%S.%f')
    dt6 = datetime.datetime.strptime(dts6,'%Y-%m-%d %H:%M:%S.%f')
    dt7 = datetime.datetime.strptime(dts7,'%Y-%m-%d %H:%M:%S.%f')
    dt8 = datetime.datetime.strptime(dts8,'%Y-%m-%d %H:%M:%S.%f')
    dt9 = datetime.datetime.strptime(dts9,'%Y-%m-%d %H:%M:%S.%f')
    soil_temp0 = data[9][2]
    soil_temp1 = data[8][2]
    soil_temp2 = data[7][2]
    soil_temp3 = data[6][2]
    soil_temp4 = data[5][2]
    soil_temp5 = data[4][2]
    soil_temp6 = data[3][2]
    soil_temp7 = data[2][2]
    soil_temp8 = data[1][2]
    soil_temp9 = data[0][2]
    soil_moist0 = data[9][3]
    soil_moist1 = data[8][3]
    soil_moist2 = data[7][3]
    soil_moist3 = data[6][3]
    soil_moist4 = data[5][3]
    soil_moist5 = data[4][3]
    soil_moist6 = data[3][3]
    soil_moist7 = data[2][3]
    soil_moist8 = data[1][3]
    soil_moist9 = data[0][3]
    air_temp0 = data[9][4]
    air_temp1 = data[8][4]
    air_temp2 = data[7][4]
    air_temp3 = data[6][4]
    air_temp4 = data[5][4]
    air_temp5 = data[4][4]
    air_temp6 = data[3][4]
    air_temp7 = data[2][4]
    air_temp8 = data[1][4]
    air_temp9 = data[0][4]
    air_hum0 = data[9][5]
    air_hum1 = data[8][5]
    air_hum2 = data[7][5]
    air_hum3 = data[6][5]
    air_hum4 = data[5][5]
    air_hum5 = data[4][5]
    air_hum6 = data[3][5]
    air_hum7 = data[2][5]
    air_hum8 = data[1][5]
    air_hum9 = data[0][5]
    fig = plotly.tools.make_subplots(rows=2, cols=2,subplot_titles= ('Soil Temperature','Soil Moisture','Air Temperature','Air Humidity'),
    vertical_spacing=0.2)
    fig['layout']['margin'] = {
         'l':30 , 'r': 10, 'b': 20, 't': 20
     }
    fig['layout']['legend'] = {'x': 0.53, 'y': 1, 'xanchor': 'left'}    fig.append_trace({
        'x': [dt0,dt1,dt2,dt3,dt4,dt5,dt6,dt7,dt8,dt9],
        'y': [soil_temp0,soil_temp1,soil_temp2,soil_temp3,soil_temp4,soil_temp5,soil_temp6,soil_temp7,soil_temp8,soil_temp9],
        'name': 'Soil Temp',
        'mode': 'lines+markers',
        'type': 'scatter'
    }, 1, 1)
    fig.append_trace({
        'x': [dt0,dt1,dt2,dt3,dt4,dt5,dt6,dt7,dt8,dt9],
        'y': [soil_moist0,soil_moist1,soil_moist2,soil_moist3,soil_moist4,soil_moist5,soil_moist6,soil_moist7,soil_moist8,soil_moist9],
        'name': 'Soil Moist',
        'mode': 'lines+markers',
        'type': 'scatter'
    }, 1, 2)
    fig.append_trace({
        'x': [dt0,dt1,dt2,dt3,dt4,dt5,dt6,dt7,dt8,dt9],
        'y': [air_temp0,air_temp1,air_temp2,air_temp3,air_temp4,air_temp5,air_temp6,air_temp7,air_temp8,air_temp9],
        'name': 'Air Temp',
        'mode': 'lines+markers',
        'type': 'scatter'
    }, 2, 1)
    fig.append_trace({
        'x': [dt0,dt1,dt2,dt3,dt4,dt5,dt6,dt7,dt8,dt9],
        'y': [air_hum0,air_hum1,air_hum2,air_hum3,air_hum4,air_hum5,air_hum6,air_hum7,air_hum8,air_hum9],
        'name': 'Air Humid',
        'mode': 'lines+markers',
        'type': 'scatter'
    }, 2, 2)
    fig['layout']['yaxis1'].update(range=[28,34])
    fig['layout']['yaxis2'].update(range=[400,800])
    fig['layout']['yaxis3'].update(range=[28,34])
    fig['layout']['yaxis4'].update(range=[45,75])
    return fig
    if __name__ == '__main__':
    app.run_server(debug=True)
```



Rysunek 9. Przykładowa strona www prezentująca dane zbierane przez Agri-Stick. Adres IP strony został zamazany dla ochrony prywatności autora projektu