HydroPlant
System automatycznego nawadniania i monitorowania roślin

Bartłomiej Szałach
Krzysztof Kruczyński

14 kwietnia 2017

Spis treści

1 Plan projektu

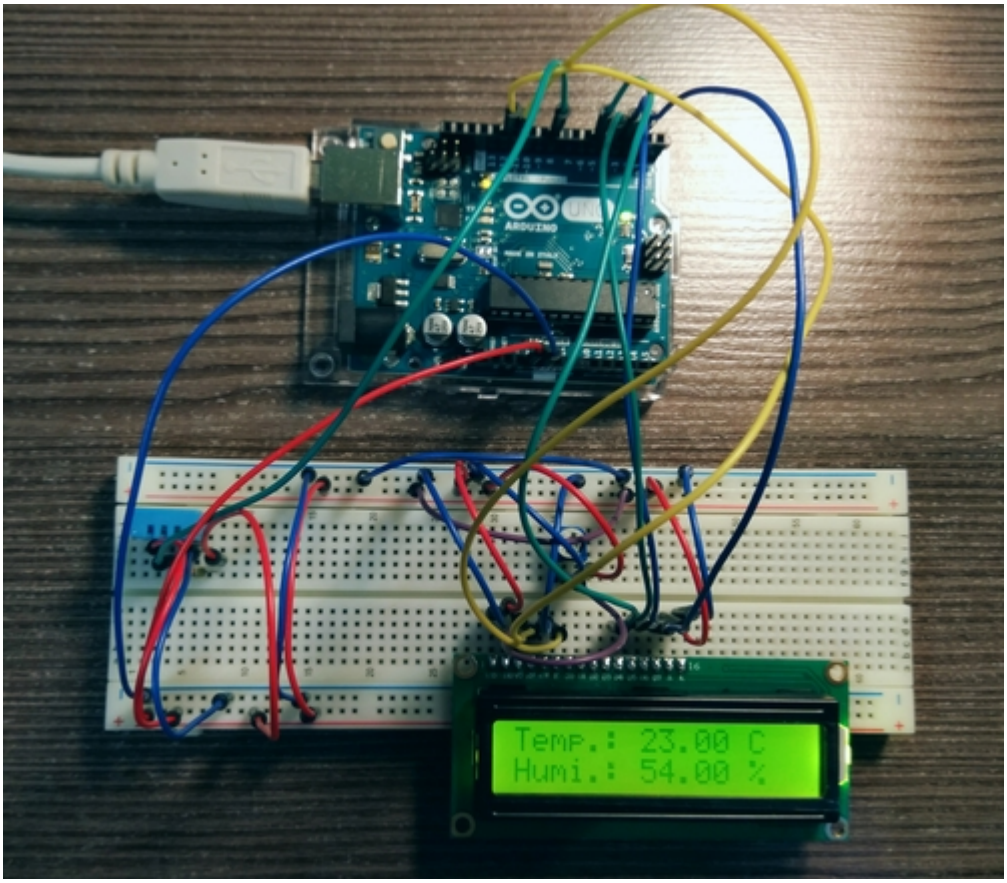
Projekt zakłada stworzenie systemu zawierającego czujniki pozwalające na monitoring wilgotności gleby oraz temperatury miejsca w którym umieszczona jest roślina. Chcemy zaimplementować obsługę czujników oraz algorytm rozstrzygający kiedy i ile wody zaaplikować do kwiatka. Pod uwagę zostanie wzięty m.in. problem porcjowania wody, w celu podania jej odpowiedniej ilości. Dodatkowo, w systemie będzie się znajdywał przycisk do ręcznego uruchomienia pompki.

2 Główne komponenty

- Arduino UNO R3 (ATMEGA 328P)
- Czujnik temperatury i wilgotności powietrza DHT11
- Czujnik wilgotności gleby
- Pompa wodna RS-360SH
- Zbiornik na wodę z rurkami
- Zestaw diod

3 Wstępny miniprojekt z czujnikiem DHT11

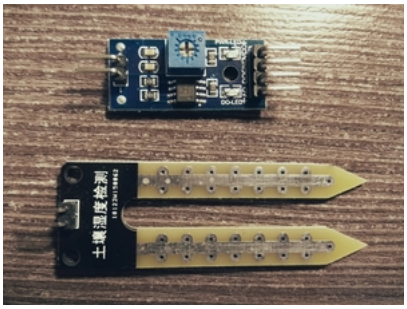
Aby zapoznać się z platformą Arduino oraz podstawową obsługą czujników wykonaliśmy prosty projekt pokazujący aktualną temperaturę oraz wilgotność powietrza odczytaną za pomocą czujnika DHT11. W tym projekcie użyliśmy oficjalnej biblioteki dołączonej do czujnika, którą w prawdziwym projekcie zaimplementujemy samodzielnie.



Rysunek 1: Aktualne pomiary czujnika DHT11 wypisywane na ekran

4 Obsługa czujnika wilgotności gleby

Zakupiony czunik składa się z trzech części: sondy pomiarowej, modułu detektora oraz przewodów. Zasilany jest napięciem od 3,3 V do 5 V. Posiada dwa wyjścia: cyfrowe (D0) oraz analogowe (A0).
Wyjście D0 jest domyślnie w stanie wysokim, natomiast po wykryciu wilgotności przechodzi w stan niski. Czułość możemy regulować za pomocą wbudowanego potencjometru. Przy zmianie oporności potencjometru możemy zmienić próg wilgotności przy którym na wyjściu cyfrowy będzie wartość 0. Przy domyślnym ustawieniu potencjometra uruchomiliśmy następujący program.



Rysunek 2: Sonda pomiarowa oraz detektor

```
int sensor_A0 = A0;
int sensor_D0 = 2;
int val_A0;
int val_D0;

void setup() {
  Serial.begin(9600);
  pinMode(2, INPUT);
}

void loop() {
  val_A0 = analogRead(sensor_A0);
  val_D0 = digitalRead(sensor_D0);
  Serial.print("D0: ");
  Serial.print(val_D0);
  Serial.print(" -- A0: ");
  Serial.println(val_A0);
  delay(500);
}
```

Po wgraniu programu na procesor sprawdzaliśmy pomiary na monitorze szeregowym. W trakcie działania do sondy przyłożyliśmy mokrą chusteczkę, co od razu zostało przez nią wykryte.

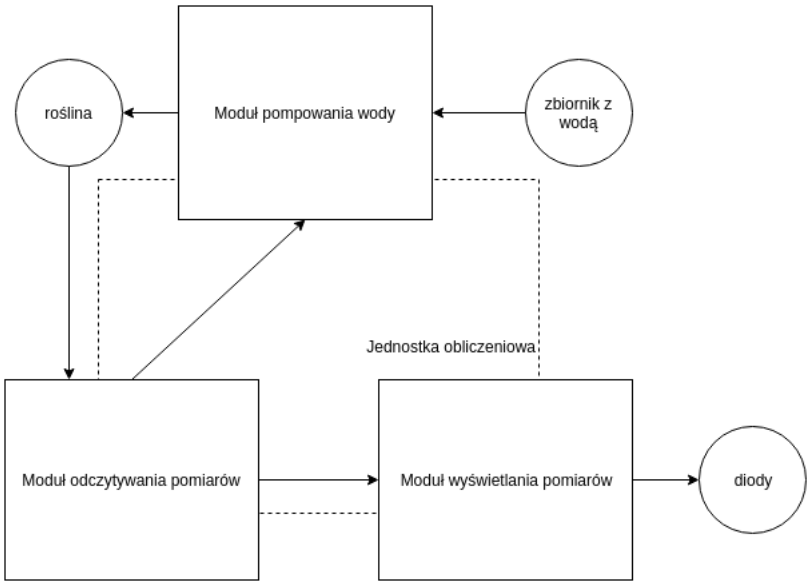
```
D0: 1   -- A0: 1019
D0: 1   -- A0: 1019
D0: 1   -- A0: 1017
D0: 1   -- A0: 1018
D0: 1   -- A0: 1018
D0: 1   -- A0: 782
D0: 1   -- A0: 623
D0: 0   -- A0: 499
D0: 0   -- A0: 438
D0: 0   -- A0: 399
D0: 0   -- A0: 401
D0: 1   -- A0: 534
D0: 0   -- A0: 453
D0: 0   -- A0: 499
D0: 0   -- A0: 502
```

5 Czujnik temperatury LM35

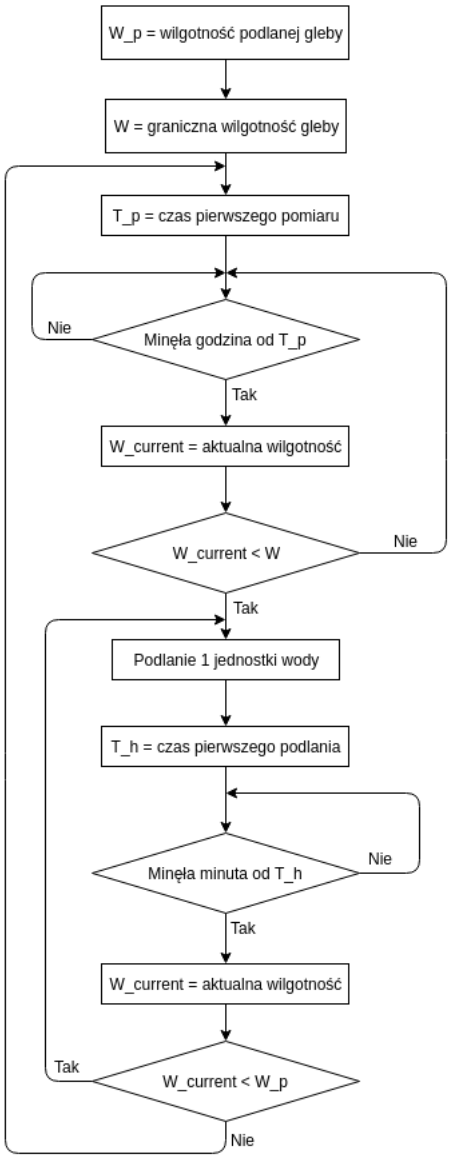
Pomimo, że został już omówiony inny czujnik temperatury, chcemy zapoznać się także z drugim posiadanym przez nas, czyli LM35. Dzięki temu w projekcie będziemy mogli użyć tego, który najlepiej spełnia nasze oczekiwania lub nawet kombinacji kilku czuników celem optymalizacji algorytmu.
Temperaturę z czujnika pobieramy i skalujemy według prostego wzoru:

$$temp = (5.0 * analogRead(tempPin) * 100.0)/1024$$

6 Diagram systemu



7 Algorytm podawania wody



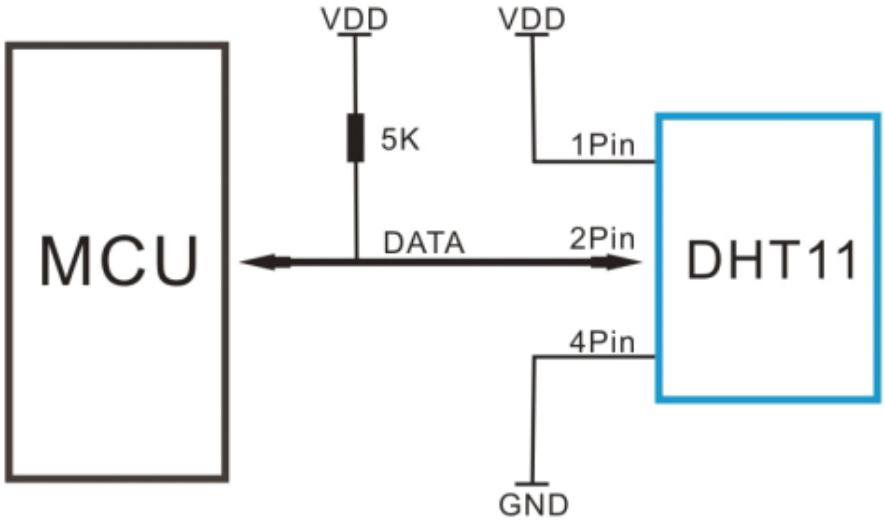
8 Implementacja prostego algortmu na podstawie pomiaru gleby

```
const int dhtPin = 2;
const int moisturePin = A0;
const int lightPin = A1;
const int thresholdMoisture = 900;
const int minMoisture = 500;
const int timeout = 1000;        //todo change to an hour
const int wateringTimeout = 100; //todo change to a minute

char *getCurrentTime();
void waterOneUnit();
int readMoisture() ;

void setup() {
    Serial.begin(57600);
}

void loop() {
    int currentMoisture = readMoisture();
    Serial.print("current moisture: ");
    Serial.println(currentMoisture);

    if (currentMoisture > thresholdMoisture) {
        Serial.println("i'm inside if");

        do {
            Serial.print("water time: ");
            Serial.println(millis());
            waterOneUnit();
            delay(wateringTimeout);
            currentMoisture = readMoisture();

        } while(currentMoisture > minMoisture);
//        int dhtData = DHT11.read(dhtPin);
//        float t = DHT11.temperature;
//        float h = DHT11.humidity;
//        int light = analogRead(lightPin);
```

```
    }
    Serial.println("loop end");
    delay(timeout);
}

void waterOneUnit() {

}

int readTemperature() {
    int dhtData = DHT11.read(dhtPin);
    return (int) DHT11.temperature;
}

int readHumidity() {
    int dhtData = DHT11.read(dhtPin);
    return (int) DHT11.humidity;
}

int readMoisture() {
    const int numberOfMoistureSamples = 15;
    const int moistureSamplingTimeout = 1000;
    int moistureSamplesSum = 0;
    for (int i = 0; i < numberOfMoistureSamples; i++) {
        moistureSamplesSum += (int) analogRead(moisturePin);
        delay(moistureSamplingTimeout);
    }
    return moistureSamplesSum / numberOfMoistureSamples;
}

int readLight() {
    return (int) analogRead(lightPin);
}
```

9 Biblioteka do obsługi czujnika DHT11

Czujnik podłączony jest przy pomocy trzech z czterech dostępnych pinów, gdyż według dokumentacji pin o numerze 3 nie jest używany. Podczas gdy dwa piny odpowiadają za zasilanie modułu (VDD/GND), trzeci z nich jest używany jako jedyny do komunikacji oraz synchronizacji z mikroprocesorem w Arduino. Jest on podłączony do czunika, Arduino oraz do napięcia (VDD) poprzez rezystor, dzięki czemu możliwe jest wysyłanie komunikatów oraz danych poprzez modyfikację napięcia oraz naprzemienne czytanie lub wysyłanie poprzez Arduino na ten pin.

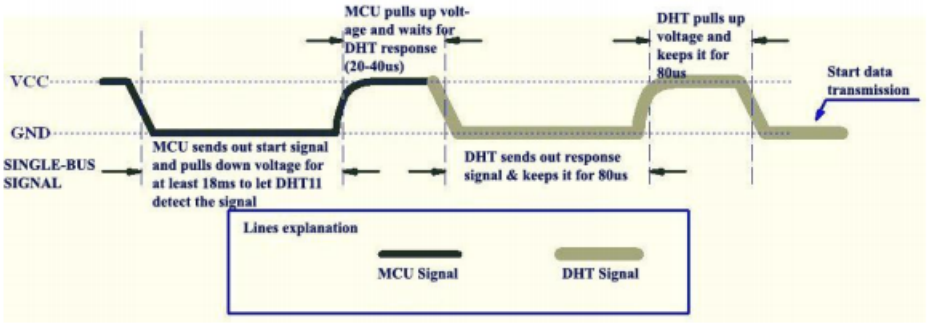


Transmitowane dane składają się z 40 bitów, po 1 bajt na kolejno:

- Część dziesiętną wartości wilgotności
- Część całkowitą wartości wilgotności
- Część dziesiętną wartości temperatury
- Część całkowitą wartości temperatury
- Sumę kontrolną, będącą sumą powyższych czterech

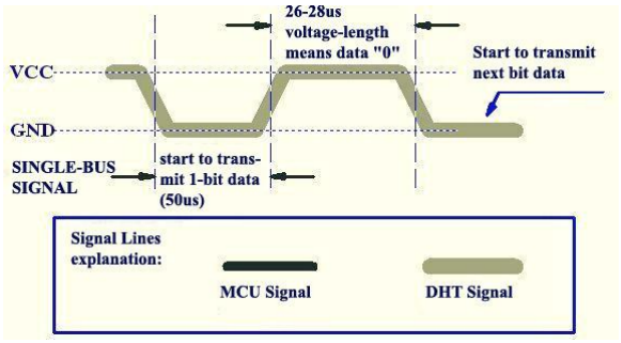
Bity w każdym bajcie są zaczynają być wysyłane od najstarszego. Zarówno biblioteka dostępna na oficjalnej stronie Arduino, jak i zaimplementowana tutaj nie używają części dziesiętnych.

Po podłączeniu układu, wartość magistali służącej do komunikacji jest w stanie wysokim, co onacza wolne pasmo i brak komunikacji. Aby to zmienić, mikrokontroler wysyła wysyła przez 18ms sygnał niski, aby czujnik mógł wykryć jego obecność, a następnie przez 40μs ponownie ustawia magistralę w tryb wysoki. Gdy czujnik wykryje sygnał startowy, odpowiada, wysyłając przez 80μs sygnał niski, a następnie przez taki sam okres czasu sygnał wysoki, co oznacza, że przygotowuje się do transmisji. Biblioteka sprawdza, czy DHT11 poprawnie nadał swoją odpowiedź i, jeżeli nie, sygnalizuje stosowny błąd.

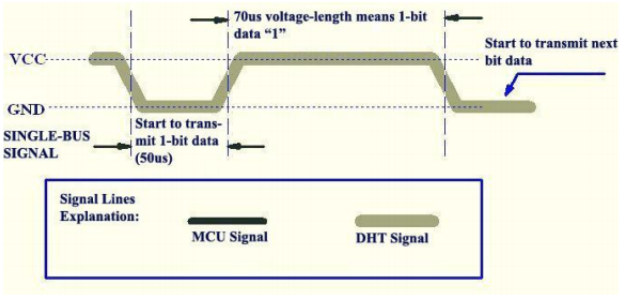


Rysunek 3: Handshake pomiędzy mikrokontrolerem i czujnikiem

Gdy po wymianie sygnałów magistrala jest w stanie niskim ustawionym przez czujnik, może on zacząć transmitować dane. Przed wysłaniu każdego bitu, magistrala jest ustawiana na 50μs w stan niski. Kolejne bity są przesyłane poprzez odpowiednią długość sygnału wysokiego: 26μs - 28μs dla "0"i 70μs dla "1".



Rysunek 4: Wysyłanie bitu "0"



Rysunek 5: Wysyłanie bitu "1"

Wszystkie otrzymane 40 bitów jest na końcu zapisywane do buffera, który jest zwracany użytkownikowi.
Odczytywanie odebranych danych jest realizowane w następujący sposób:

```
// DHT sends 5 bytes ie. 40 bits as output
for (int i = 0; i < 40; i++) {

    if (!check(pin, LOW)) return DHTLIB_ERROR_TIMEOUT;

    unsigned long t = micros();

    if (!check(pin, HIGH)) return DHTLIB_ERROR_TIMEOUT;

    // Below is 0, above is 1
    if ((micros() - t) > 40) {
        buffer[current_byte] |= (1 << bit_shift);
    }

    // If counter = 0 move to the next byte,
    // otherwise decrease counter to set next bit of requested bits

    if (bit_shift == 0) {
        bit_shift = 7;
        current_byte++;
    } else {
        bit_shift--;
    }

}
```

10 Czujnik światła

Zmodyfikowaliśmy algorytm podawania wody, biorąc pod uwagę czujnik światła tak, by podlewanie nie odbywało się nocą, gdyż jest to roślinom zbędne. Zatem jeśli jest noc, struktura algorytmu nie wchodzi do pierwszego bloku warunkowego.
Dodatkowo implementacja została usprawniona w taki sposób, by procesy oczekiwania jakiegoś czasu nie były blokujące, tzn. by program mógł na bieżąco bez przerwy działać.
Ostateczny algorytm podlewania wygląda następująco:

```
    if (measure.moisture > maxMoisture) {
        watering.waterTime = true;
        watering.start = getCurrentTime();
    }
    if (watering.isNow && hasTimePassed(watering.start, wateringUnit)) {
        putWaterAway();
    }
```

```
if (watering.waterTime && hasTimePassed(watering.start, wateringTimeout)) {
    watering.start = getCurrentTime();
    if (!measure.night) {
        waterPlant();
    }
    if (measure.moisture < minMoisture) {
        watering.waterTime = false;
    }
}

delay(timeout);
```

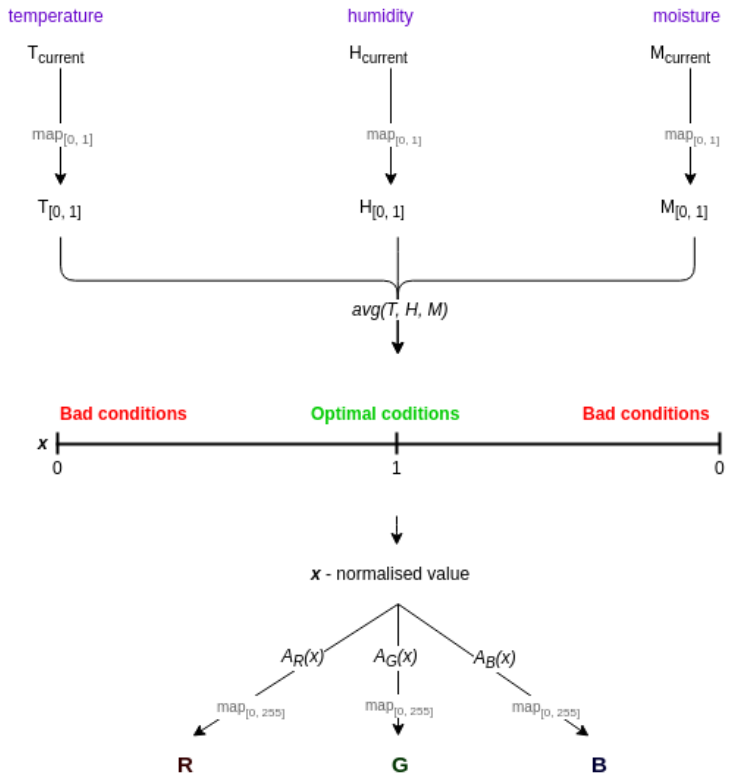
11 Podsystem monitorowania rośliny

Ostateczna forma tego podsystemu prezentuje aktualny stan rośliny (1) na dołączonym module ekranu cyfrowego, (2) na komputerze za pomocą wysyłanych danych poprzez dołączony moduł ethernet oraz (3) przy pomocy kolorowej diody RGB informującej o stanie rośliny. Podsystem znajduje się na diagramie systemu z punktu 6.
Implementacja tego podsystemu bez implementacji konkretnych funkcji:

```
updateMeasurements();

printToLcd(measure);

PlantStateColorCalculator calculator;
rgb_color newColor = calculator.calcPlantStateColor(measure.temperature, measure.humidity,
    measure.moisture);
fader f(redPin, greenPin, bluePin);
f.fade(currentColor, newColor);
currentColor = newColor;
```

12 Dioda prezentująca warunki roślinne

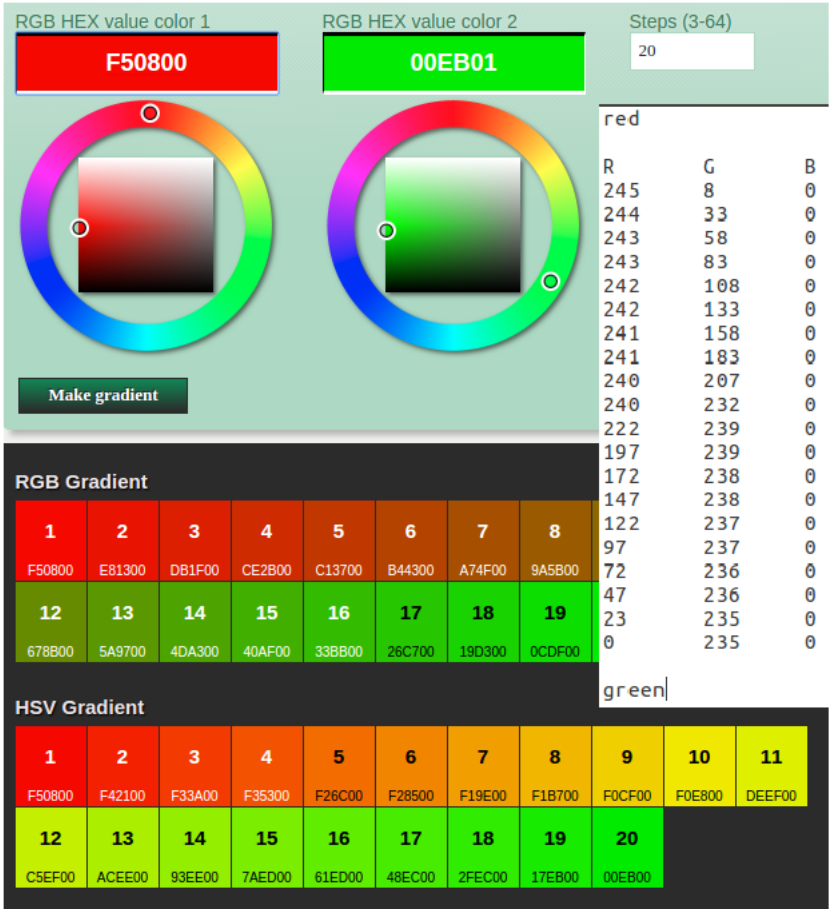
Warunki roślinne wyświetlane na diodzie RGB to: temperatura (T), wilgotność powietrza (H) i wilgotność gleby (M).
Barwa diody waha się między zielonym (dobre warunki) a czerwonym (złe warunki).
Proces przekształcania wejściowych pomiarów na odpowiedni kolor RGB przedstawia poniższa grafika.



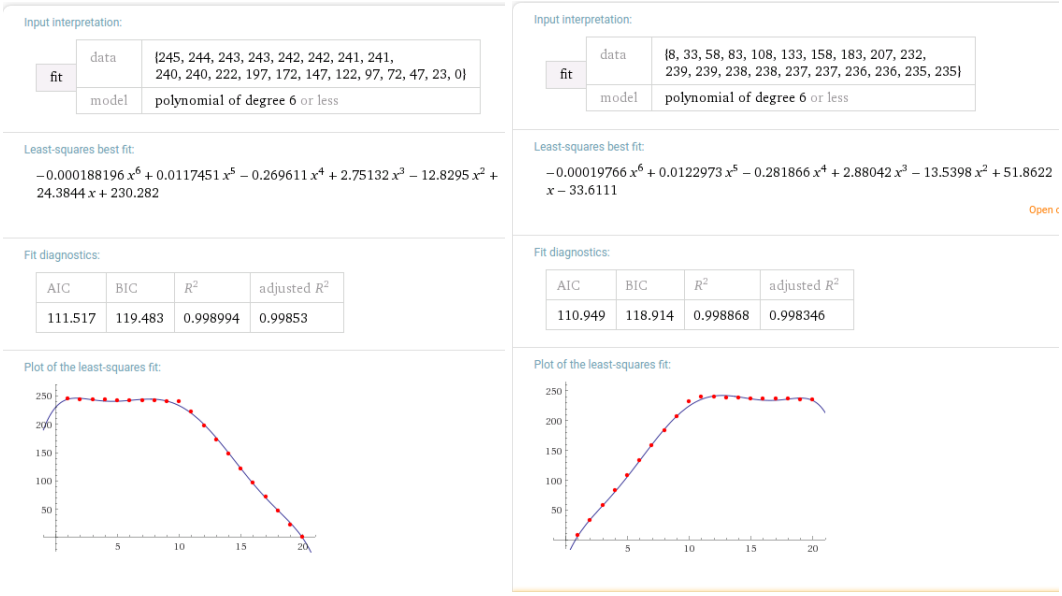
Rysunek 6: Konwersja pomiarów na stan rośliny w kolorze RGB

Funkcje oznaczone przez A(x) oznaczają funkcje aproksymujące wartość danego koloru ze względu na jakość podanego argumentu w przedziale [0, 1], gdzie 0 to najniższa jakość, a 1 najwyższa. Tym argumentem jest średnia ważona wszystkim pomiarów (odpowiednio znormalizowanych) z konkretnymi wagami. Te wagi to $w_T = 2$, $w_H = 2$ i $w_M = 6$.

Do implementacji funkcji aproksymujących kolory wykorzystaliśmy najpierw 20 próbek kolorów gradientowych pomiędzy zielonym a czerwonym, wygenerowanych przy pomocy aplikacji webowej.



Następnie przy pomocy innej aplikacji dopasowaliśmy wielomian 6. stopnia do tych próbek tak, by dało się obliczyć każdą z trzech barw RGB w zależności od jakości argumentu wejściowego.



(a) Czerwona (b) Zielona

Rysunek 7: Krzywa dopasowana do próbek danej barwy

Ponadto została zaimplementowana optymalizacja łagodnego przejścia z jednego koloru w drugi, bo jeśli okazałoby się, że trzeba byłoby wyświetlić w krótkim czasie dwa mocno różne kolory, to dzięki tej optymalizacji przejście między nimi będzie płynne dla oka.

13 Bibliografia

- http://playground.arduino.cc/Main/DHT11Lib
- http://botland.com.pl/content/140-arduino-i-obsluga-czujnika-temperatury-i-wilgotnosci-dht11
- http://www.arduino.cc/en/Tutorial/HelloWorld?from=Tutorial.LiquidCrystal
- http://botland.com.pl/content/150-arduino-w-polaczeniu-z-czujnikiem-wilgotnosci-gleby
- http://playground.arduino.cc/Main/LM35HigherResolution
- http://www.micropik.com/PDF/dht11.pdf