Ultradźwiękowy miernik odległości

Mikrokontrolery - projekt

Grupa 23

1. Borowy Szymon
2. Kołodziej Dominik
3. Kozak Marcin

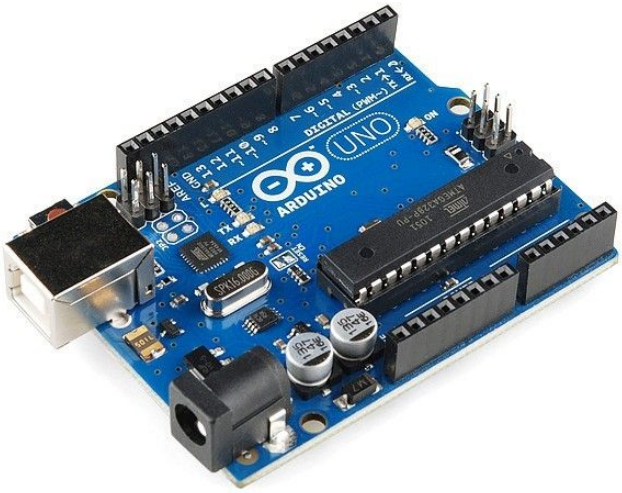
Informatyka WEAIiIB
2 rok studiów
2018/19
AGH Kraków

1. WPROWADZENIE

Celem przeprowadzonego ćwiczenia było skonstruowanie urządzenia mierzącego odległość do dowolnego, dostatecznie dużego obiektu. Zdajemy sobie sprawę, że pomiar taki jest obarczony dużym błędem m.in ze względu na podzespoły o ograniczonej dokładności, jednak za pomocą algorytmów chcemy, aby ten pomiar jak najbliższy rzeczywistej odległości.

2. SCHEMAT POŁĄCZEŃ PŁYTKI I DODATKOWYCH MODUŁÓW

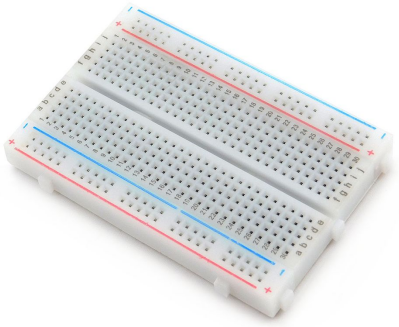
a) WYKORZYSTANE MODUŁY



[1]

Płytka z mikrokontrolerem Arduino Uno

[1] https://www.crazypi.com/image/cache/data/arduino-uno-r3-1-600x600.jpg



[2]

Płytka stykowa

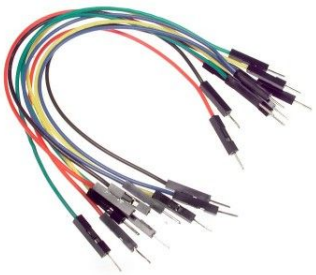
[3]

Ultradźwiękowy czujnik odległości

[4]

TM1638 controller - wyświetlacz LED i przyciski

[2] https://nettigo.pl/system/images/66/original.jpg
[3] https://images-na.ssl-images-amazon.com/images/I/61-2fYKuyKL._SY355_.jpg
[4] https://www.robotics.org.za/image/cache/catalog/generic/TM1638/TM1638%20-%20Main-500x500.jpg



[5]

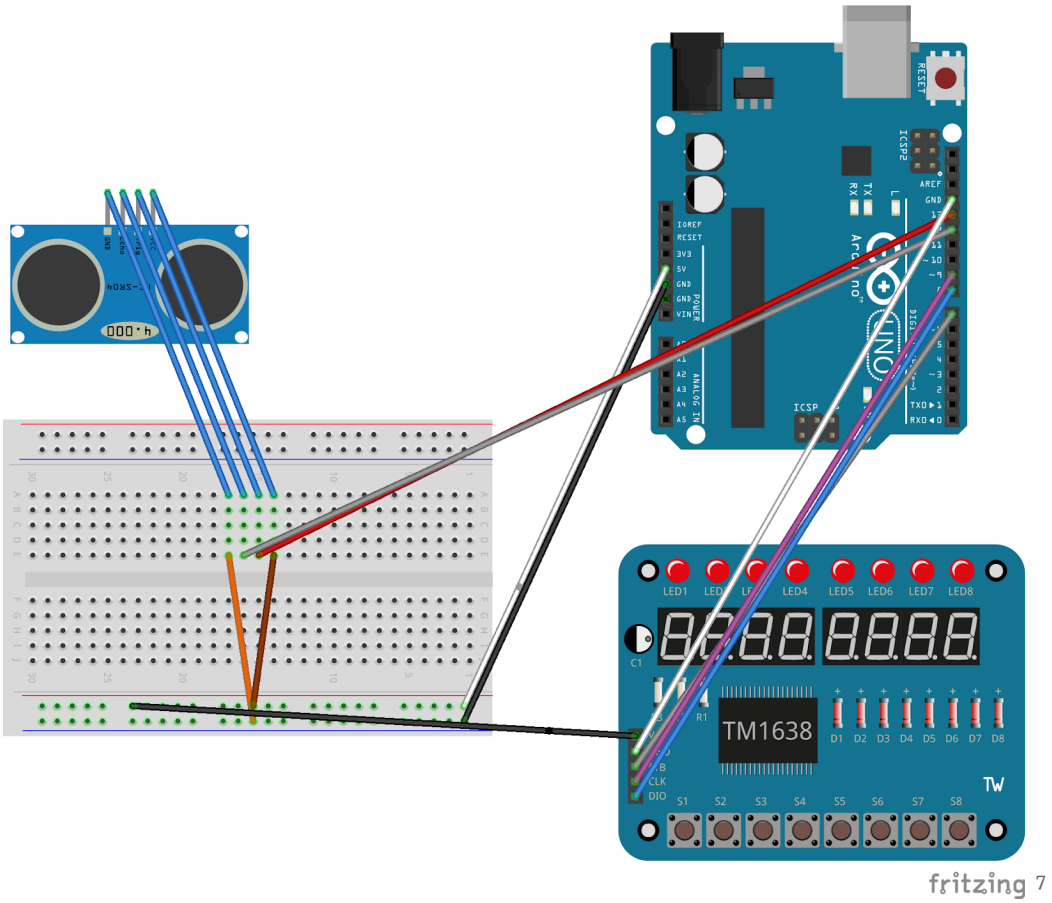
Kable

[6]

Kabel USB Arduino

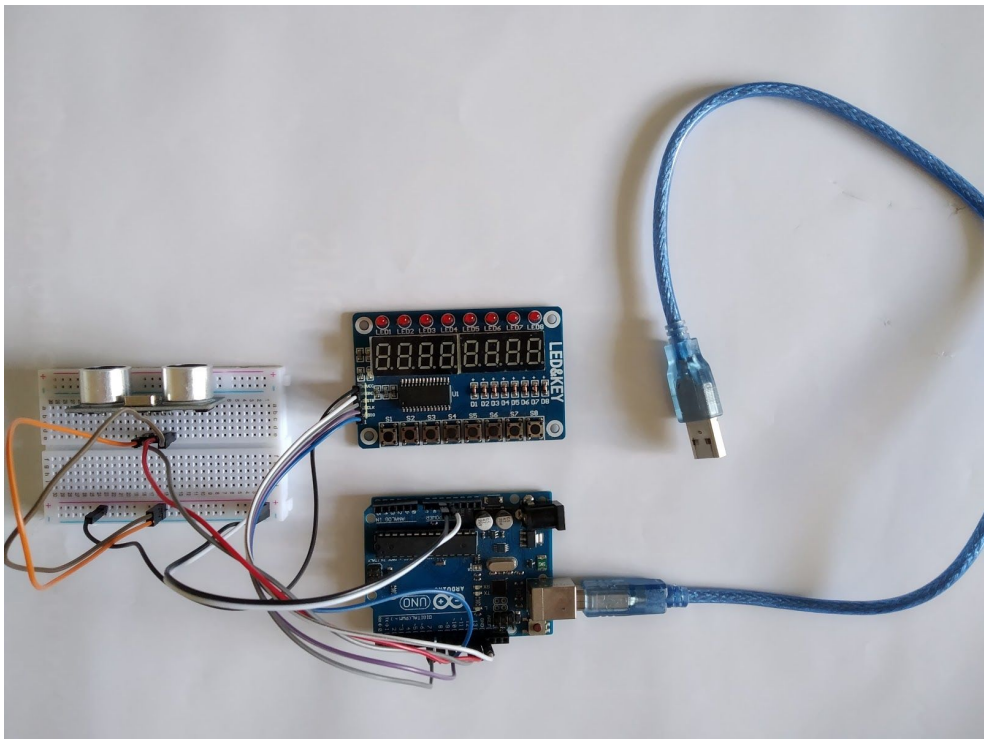
[5] https://www.exploringarduino.com/wp-content/uploads/2013/05/45040-dscn0624-400x300.jpg
[6] https://rc-planeta.pl/2174-large_default/kabel-do-arduino-uno-usb.jpg

b) SCHEMAT POŁĄCZENIA



Schemat symboliczny

[7] Opracowanie własne. Użyto programu fritzing



[8]

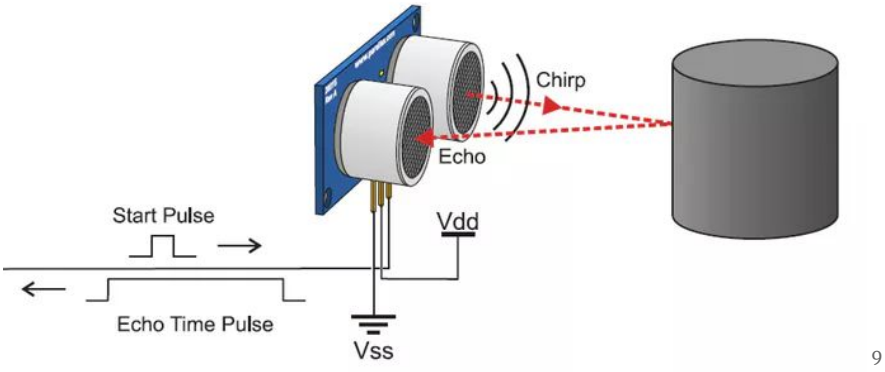
Zdjęcie zbudowanego układu

3. OPIS ALGORYTMU I ISTOTNYCH ELEMENTÓW PROGRAMU

a) OPIS DZIAŁANIA CZUJNIKA ULTRADŹWIĘKOWEGO

Moduł SR-04F posiada dwa “bębny” w swojej obudowie. Za pomocą jednego z nich wysyłana jest fala ultradźwiękowa, a drugi jest odpowiedzialny za jej przechwycenie spowrotem.

[8] Opracowanie własne



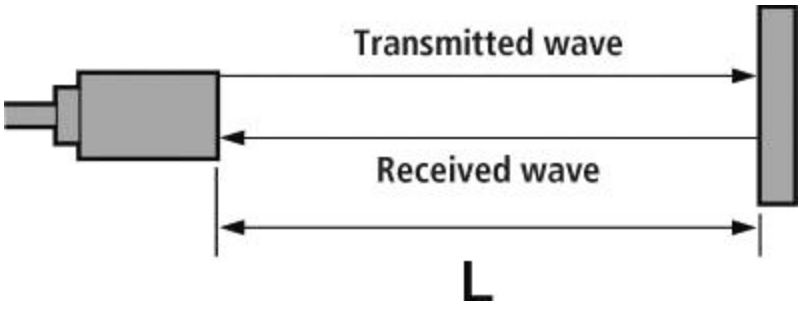
[9]

Emiter wysyła falę przez 10 mikrosekund, jeżeli zostanie podany na wejściu wysoki sygnał *trigPin*. Po zakończeniu nadawania sygnał *trigPin* ustawiany jest na niski. Następnie czekamy na falę i gdy ta zostaje zarejestrowana przez *echoPin* znamy czas, w którym fala wróciła do urządzenia.

Z podstawowych praw fizyki wiemy, że

*Dystans = Prędkość fali * Całkowity czas drogi*

Fala musi jednak pokonać drogę z urządzenia do badanej powierzchni, a następnie od tej powierzchni do urządzenia.



Dlatego więc nasz wzór na dystans przyjmuje postać:

*Dystans = Prędkość fali * (Całkowity czas drogi / 2)*

*Dystans = 29.1 * (Całkowity czas drogi / 2)*

[9] https://hackster.imgix.net/uploads/attachments/226415/SensorPingOperation.png

gdzie 29.1 to prędkość fali w centymetrach na mikrosekundy[10]

b) OPIS UŻYTEGO ALGORYTMU

W głównej pętli *loop()* wykonujemy wszystkie obliczenia. Sprawdzamy aktualny stan urządzenia i sprawdzamy, czy stan ten nie został zmieniony poprzez naciśnięcie przycisków.

Wyróżniamy 2 stany: czujnik aktywny (*turn_on = true*), oraz nieaktywny (*turn_on = false*).

W stanie aktywnym mamy również stan mierzenia i gotowości.

W stanie mierzenia na początku wykonujemy 10 pomiarów za pomocą pętli for. Wszystkie te pomiary zachowujemy w tablicy *duration[]*.

Aby pozbyć się odstających wyników zdecydowaliśmy, że do obliczeń nie będziemy brać najmniejszego oraz największego z pomiarów, a z reszty 8 pomiarów obliczymy średnią.

Przykład.

Mamy 10 pomiarów:

*586.00, 617.00, **617.00**, 542.00, **536.00**, 587.00, 611.00, 593.00, 593.00, 592.00,*

Z nich usuwamy najmniejszy (536.00) i największy (617.00):

586.00, 617.00, 542.00, 587.00, 611.00, 593.00, 593.00, 592.00,

A z reszty obliczamy średnią czasu:

590.13

Którą przetwarzamy dalej według wzorów opisanych wcześniej na milimetry:

*(590.13 / 2) / (29.1 * 0.1) ≈ 101 [mm]*

[10] https://create.arduino.cc/projecthub/rztronics/ultrasonic-range-detector-using-arduino-and-sr-04f-8a804d

Dalej przesyłamy obliczany wynik w milimetrach do napisanej przez nas funkcji *displayNumber()*, która najpierw wyciąga cyfry z liczb całkowitej int i przypisuje je do tablicy czteroelementowej, a następnie ustawia te cyfry na odpowiednich pozycjach na wyświetlaczu. Dodatkowo funkcja sprawdza czy wartości zmierzone nie są większe od 10m i nie są ujemne. Jeżeli taki wypadek występuje, to na wyświetlaczu zamiast cyfr pojawia się "----".

Po tym wykonaniu pomiarów i wyświetleniu wyniku na wyświetlaczu miernik nie będzie wykonywał pomiarów i będzie czekał na naciśnięcie przycisku, aby zmienić stan.

Natomiast w stanie nieaktywnym nie są wykonywany pomiary, ale jedynie wyświetla się "----" przez 0.2s, a następnie pusty ekran przez 0.2s co daje nam miganie ekranu.

4. WYKORZYSTANE BIBLIOTEKI

Została wykorzystana jedna dodatkowa biblioteka: TM1638.h[11], która służy do interakcją z płytką Led&Key.

Dzięki niej możemy wyświetlać cyfry, bądź inne znaki za pomocą funkcji *setDisplay()* przesyłając do niej liczby na każde pole, zamiast wysyłać po jednym bicie na każdy element ekranu.

Również możemy dostać się do naciśniętych klawiszy w przystępny sposób za pomocą funkcji *getButtons()*.

5. SPOSÓB BUDOWY KODU WYKONALNEGO

Cały program został napisany w programie Arduino 1.8.9[12]

Zawartość programu zapisywana jest w pliku z rozszerzeniem *.ino, a jego składnia jest podobna do popularnego języka C.

Za pomocą tego programu dostępna jest również kompilacja programu wraz z

[11] https://github.com/rjbatista/tm1638-library
[12] Program został pobrany ze strony https://www.arduino.cc/

dołączonymi bibliotekami (TM1638.h) oraz wgranie programu na płytkę Arduino.

6. KOD ŹRÓDŁOWY

```
#define trigPin 13
#define echoPin 12

#define offset 4
#define ile_prob 10

#include <TM1638.h>
TM1638 module(8, 9, 7); //number of pins to led&key

byte displayDigits[] = {63,6,91,79,102,109,125,7,127,103 };
byte values[] = { 0,0,0,0,0,0,0,0 };
int theDigits[] = { 0,0,0,0 };
bool turn_on = false;
bool clicked = false;

void setup()
{
  //Serial.begin(9600); //for printing and debugging, might be deleted
later
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  module.setDisplay(values);
  module.setupDisplay(true, 2); // where 7 is intensity (from 0 to 7)
}

void loop()
{
  byte keys = module.getButtons();
  if(keys == 1){
    turn_on = true;
  }
  if(keys == 2){
    turn_on = false;
    clicked = false;
  }
  if (turn_on && !clicked){
    double duration[ile_prob], distance;
```

```
    int distance_mm;

    for (int i=0; i<ile_prob; i++){
      digitalWrite(trigPin, LOW);
      delayMicroseconds(2);
      digitalWrite(trigPin, HIGH);
      delayMicroseconds(10);
      digitalWrite(trigPin, LOW);
      duration[i] = pulseIn(echoPin, HIGH);
      delay(10);

    }


    double max_value = duration[0], min_value = duration[0], duration_mean
= 0;
    int max_index = 0 , min_index = 0;

    for (int i = 1; i < ile_prob; i++){
      if (duration[i] >= max_value){
          max_value = duration[i];
          max_index = i;
        }
      else if (duration[i] <= min_value){
          min_value = duration[i];
          min_index = i;
        }
    }

    for (int i=0; i<ile_prob;i++){
        if (i!= min_index && i!= max_index){
          duration_mean+=duration[i];
        }
    }


    duration_mean /= (ile_prob-2);
    distance = (duration_mean/2) / (29.1*0.1);
    distance_mm = (int) distance;

    delay(500);
    displayNumber(distance_mm);
```

```
    clicked = true;
  }
  else if (!clicked){
    displayNothing();
  }
}

void displayNothing(){

 for(int i = 0; i < 8; i++)
  values[i] = 64;

 module.setDisplay(values);
 delay(200);

 for(int i = 0; i < 8; i++)
  values[i] = 0;

 module.setDisplay(values);
   delay(200);
}

void displayNumber(int number_mm){

 // Extract the digits from this number.
 theDigits[0] = (int)(number_mm/1000);
 theDigits[1] = (int)((number_mm - (1000*theDigits[0])) / 100);
 theDigits[2] = (int)((number_mm - (1000*theDigits[0]) -
(100*theDigits[1]))/10);
 theDigits[3] = (int)(number_mm - (1000*theDigits[0]) - (100*theDigits[1])
- (10*theDigits[2]));

 // Update the values in the values array used by the display.
 values[0+offset] = displayDigits[theDigits[0]];
 values[1+offset] = displayDigits[theDigits[1]];
 values[2+offset] = displayDigits[theDigits[2]] + 128;
 values[3+offset] = displayDigits[theDigits[3]];

 // Make sure that the number passed to the function was >= 0 or <100,
otherwise show an error with ----.
 if(number_mm < 0 or number_mm > 9999){
    for(int i = 0; i < 4; i++)
```

```
      values[i+offset] = 64;
 }

 // Update the display itself with the new values.
 module.setDisplay(values);
}
```

7. ZAJMOWANA PAMIĘĆ

Szkic używa 5686 bajtów (17%) pamięci programu. Maksimum to 32256 bajtów.

Zmienne globalne używają 282 bajtów (13%) pamięci dynamicznej, pozostawiając 1766 bajtów dla zmiennych lokalnych. Maksimum to 2048 bajtów.

8. PODSUMOWANIE

Jesteśmy zadowleni z wyników pomiaru urządzenia, które skonfrontowaliśmy z dalmierzem laserowym. Potwierdziliśmy, że możemy zbudować sami takie urządzenie, a przy nim dużo się nauczyć.