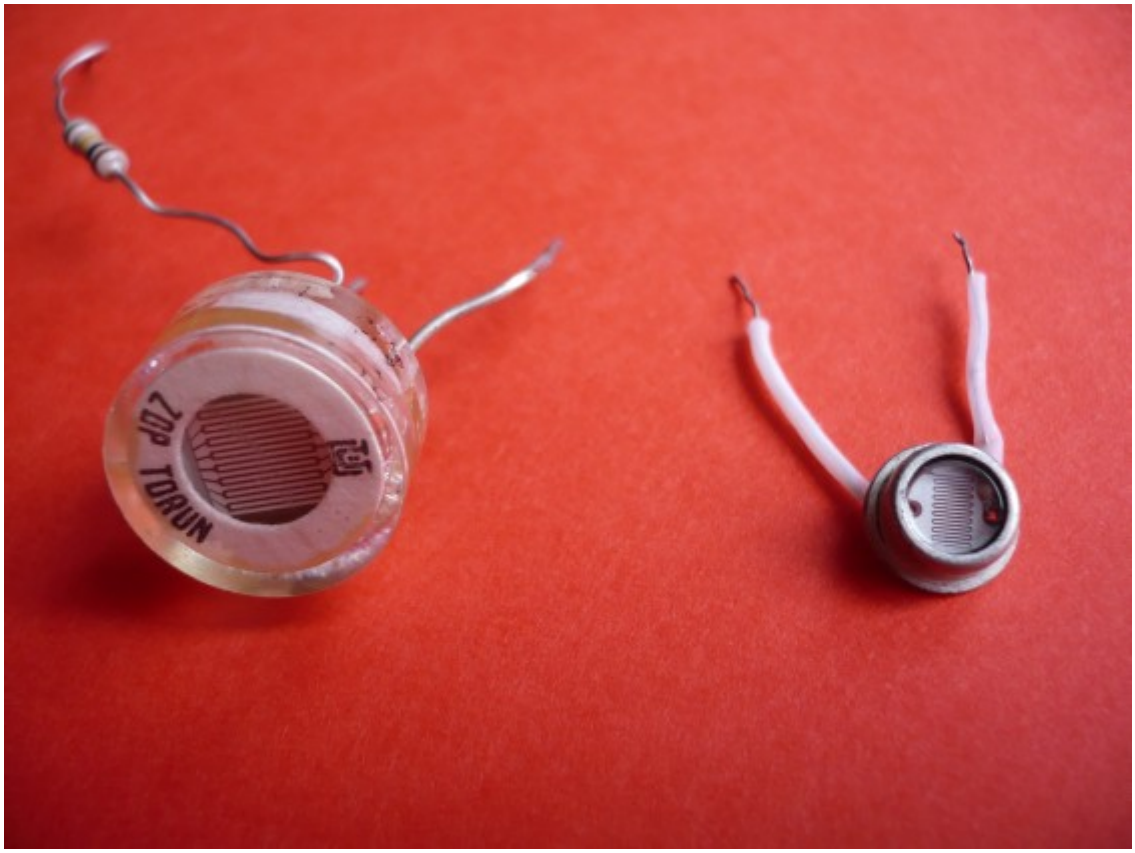
Pomiar natężenia światła (005; 15.07.2009; *arduino, processing*)

Artykuł ten będzie praktycznym wykorzystaniem opisu pomiaru napięcia przy użyciu Arduino. Fotorezystor z dzielnikiem napięcia będzie czujnikiem pomiarowym światła, Arduino urządzeniem pomiarowym, a program napisany w środowisku Processing służyć będzie do prezentacji danych pomiarowych.

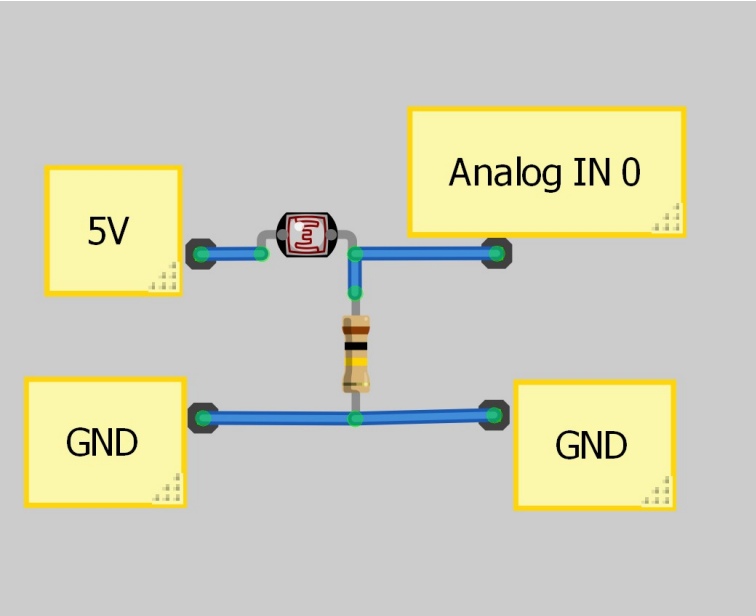
Fotorezystor jako źródło sygnału napięciowego do wejścia analogowego.

Fotorezystor jest elementem półprzewodnikowym którego rezystancja zależy od natężenia padającego światła. Więcej można poczytać tutaj: http://pl.wikipedia.org/wiki/Fotorezystor. Dwa przykładowe fotorezystory wyglądają następująco:



Fotorezystor zastosowany w tym doświadczeniu (ten większy z powyższego zdjęcia) wykazuje opór ok 250Ω przy oświetleniu go z bliska latarką diodową, przy całkowitym zasłonięciu odznacza się oporem większym niż 2MΩ (poza skalą miernika uniwersalnego).

Podłączenie fotorezystora i rezystora do Arduino realizujemy na zasadzie dzielnika napięcia, jak na poniższym schemacie (wykonany w programie Fritzing):



Styki GND przedstawione na powyższym schemacie dotyczą tak naprawdę tego samego styku Arduino – nie wymagane są oddzielne podłączenia.

Teraz trochę teorii, aby odpowiednio dobrać rezystor.

Oznaczmy:

$U_{we} = 5\text{V}$ – napięcie pomiędzy stykami *5V* i *GND*,

U_{wy} – napięcie pomiędzy stykami *Analog IN 0* i *GND*,

R_F – rezystancja fotorezystora, zmienny w zależności od natężenia padającego światła,

R – rezystancja rezystora, stała.

Istnieje zależność:

$$U_{wy} = U_{we} \frac{R}{R_F + R}$$

Ustalmy zatem wartość rezystancji R.

Kalibracja czujnika natężenie światła.

Chcielibyśmy, aby przy minimalnej rezystancji fotorezystora $R_F = 250\,\Omega$ napięcie wyjściowe było możliwie bliskie wejściowemu $5V$. W praktyce wystarczy tu wartość większa od $\frac{254}{255} \cdot 5\text{V} \approx 4{,}98\,V$ (aby otrzymać wartość odczytu z portu analogowego równą 255 po przeliczeniu na 8bitów).

Podstawiając powyższe do zależności: $R = \frac{R_F}{\frac{U_{we}}{U_{wy}} - 1}$ otrzymujemy wartość $R = 62250\,\Omega$.

Po dobraniu rezystora o zbliżonej do obliczonej wartości możemy ustalić dokładnie teoretyczną zależność wartości odczytanej z wejścia analogowego od wartości rezystancji fotorezystora.

Przykładowe wartości dla zastosowanej przeze mnie rezystancji $R = 100\,k\,\Omega$ przedstawia tabela:

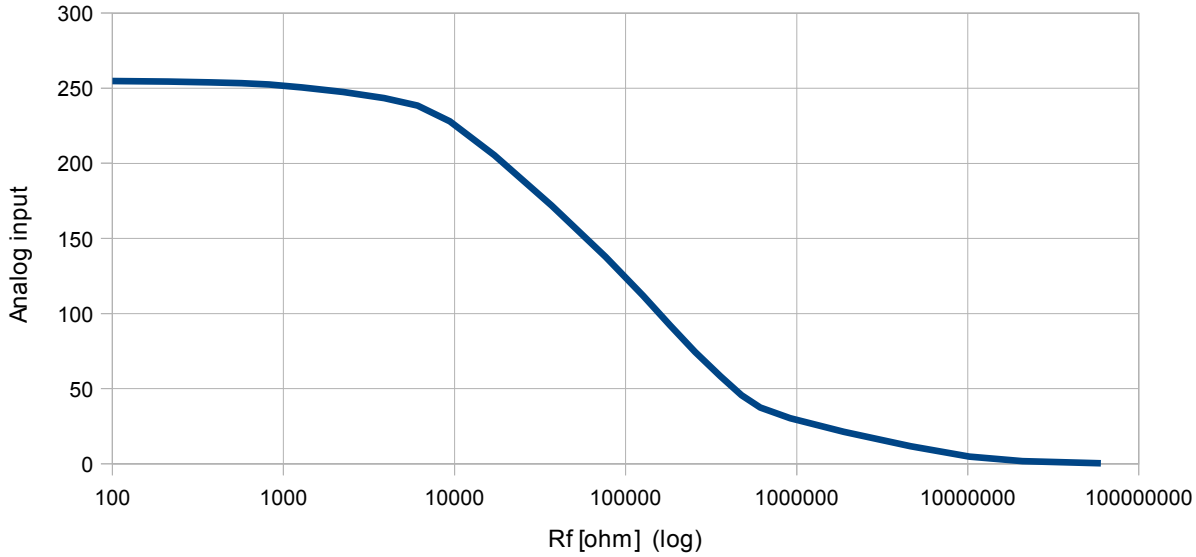
$R_F[\Omega]$	$U_{wy} = U_{we}\frac{R}{R_F + R}$	$\frac{U_{wy}}{U_{we}}$	Odczyt z wejścia analogowego (8bit)
200	4,99002	0,9980	254
2000	4,90196	0,9804	250
100000	2,50000	0,5000	128
1000000	0,45455	0,0909	23
10000000	0,04950	0,0099	3
22000000	0,02262	0,0045	1

Wzory zostały podane, zatem łatwo sprawdzić czy teoretycznie wszystko będzie ok.

Znając zależność rezystancji fotorezystora od natężenia światła i temperatury (np. podane przez producenta) możemy odczyt z wejścia analogowego przeliczyć na konkretną wartość natężenia światła. Nasuwa się również możliwość precyzyjnego pomiaru rezystancji dowolnych elementów, przy czym niezbędny jest rezystor o dokładnie znanej rezystancji.

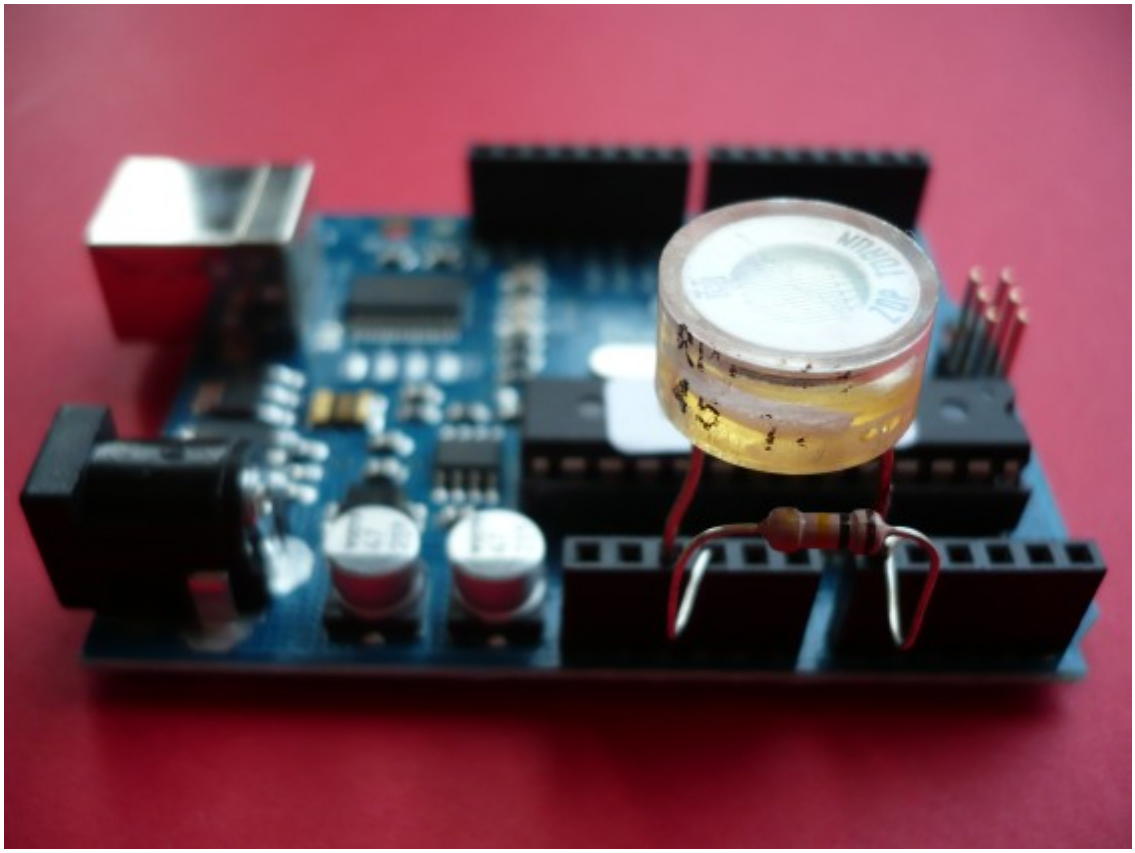
Innym sposobem kalibracji jest podłączenie potencjometru w miejsce rezystora R i dobranie jego ustawienia tak, aby uzyskać skrajne wartości sygnału przy skrajnych stanach oświetlenia (podłączamy go za środkową i jedną skrajną nóżkę). Należy przy tym pamiętać aby potencjometr mógł przewodzić prąd o natężeniu do 50mA (max. możliwy prąd płynący ze złącza 5V).

Ciekawy może wydać się również wykres zależności odczytu z wejścia analogowego od wartości rezystancji fotorezystora (w skali logarytmicznej):



Przydatna tabela do obliczeń wraz z powyższym wykresem do pobrania w formacie arkusza kalkulacyjnego Open Office Calc: http://myinventions.pl/005/Tabela005.ods

A tak wygląda mój układ pomiarowy:



Pomiar i wysłanie danych do komputera.

Kod do Arduino wygląda następująco:

```
int PoziomSygnalu;  //deklaracja zmiennej do przechowywania
                    //wartości z pomiaru
void setup()
  {
  Serial.begin(300); //inicjacja połączenia szeregowego o
                     //prędkości 300b/s
  }
void loop()
{
  PoziomSygnalu = analogRead(0); //zapis do zmiennej wartości
                           //odczytanej z portu analogowego nr 0
  Serial.print(PoziomSygnalu/4, BYTE);
  delay(100);
}
```

Podwójny ukośnik oznacza, że dalsza część linijki jest komentarzem, nie trzeba go kasować, komentarze nie są wgrywane do mikrokontrolera.

Połączenie o przepustowości 300b/s będzie tu wystarczające – przy zastosowanym tu próbkowaniu 10Hz wystarczyło by 10/s*8bitów=80b/s (obliczenie szacunkowe, przy przesyłaniu danych z małymi częstotliwościami nie musimy się o to zazwyczaj martwić i korzystamy ze standardowego 9600. Jeszcze kiedyś poruszę dokładnie temat komunikacji szeregowej).

Sprawdzić można już zatem, jak działa nasz układ oglądając przesłane dane w monitorze portu szeregowego (serial monitor). Tu jednak trzeba zwrócić uwagę na to, że Serial Monitor ma do ustawienia wartość prędkości połączenia w bodach (baud), nie można jej zatem mylić z prędkością ustaloną w kodzie Arduino wyrażoną w b/s.
Prędkość wyrażona w bodach (baud rate) określa liczbę zmiany sygnału na sekundę, a prędkość wyrażona w bitach na sekundę (data rate) określi ilość informacji przesyłanych w ciągu sekundy. Sprawa upraszcza się jednak w naszym przypadku – liczby te będą sobie zazwyczaj równe. Ustawiamy zatem Serial Monitor na 300baud. Należy jednak traktować ustalaną wartość jako „liczbę znaków 8bitowych przesyłanych na sekundę”.

Po uruchomieniu okazuje się, że Serial Monitor pokazuje dziwne znaczki zamiast liczb. Dzieje się tak dlatego, że „kazaliśmy" Arduino wysyłać wartości w postaci pojedynczych bajtów, wyświetlając je Serial Monitor wyświetla odpowiadający im znak z tabeli kodów ASCII (ponieważ ASCII jest 7bitowy, to zastosowane jest jej rozszerzenie o ósmy bit).

Praktycznie do komunikacji między urządzeniami lub programami korzystać będziemy z podanej właśnie funkcji Serial.print(*, BYTE), jednak gdy chcemy aby konsola wyświetlała nam prawidłowo pewną liczbę a w systemie dziesiętnym, zastosować należy: Serial.print(a) lub Serial.print(a,DEC). Polecenia te powodują wysłanie liczby w postaci ciągu znaków ASCII, które odpowiadają za poszczególne cyfry wysyłanej liczby (monitor portu zawsze używa znaków ascii do prezentacji danych).
Warto zwrócić uwagę, że liczba trzycyfrowa może zostać wysłana w postaci jednego bajtu z zastosowaniem polecenia Serial.print(a, BYTE), lub w postaci trzech kolejnych bajtów dla polecenia Serial.print(a).

Odbieranie danych z portu szeregowego w środowisku Processing.

Najprostszy kod pozwalający odczytać bajt wysłany przez Arduino do portu szeregowego, zapisać go do zmiennej i wyświetlić w oknie dialogowym programu (konsoli) przedstawia się następująco (komentarze dotyczą poprzedzających linii kodów):

```
import processing.serial.*;
     //import biblioteki do komunikacji szeregowej
Serial myPort;
     //deklaracja portu szeregowego o nazwie myPort
int BajtOdczytany;
     //deklaracja zmiennej do zapisu  odczytanego bajtu

void setup()
{
  println(Serial.list());
     //wyświetlenie w konsoli listy dostępnych portów szeregowych
  myPort = new Serial(this,Serial.list()[1], 300);
     //przyporządkowanie portu nr 1 na liście dostępnych portów i
     //ustalenie prędkości 300b/s
}

void draw()
{
  if ( myPort.available() > 0)
     // jeśli dostępny jest nieodczytany bajt na porcie szeregowym
     //wykonaj kolejny blok poleceń
    {
      BajtOdczytany = int(myPort.read());
          //odczytanie wartości z portu ze zamianą jej na liczbę
          //całkowitą dziesiętną oraz podstawienie jej pod zmienną
      println(BajtOdczytany);
          //wyświetlenie odczytanej wartości w konsoli w
          //w reprezentacji dziesiętnej
    }
}
```

Oczywiście niezbędna była w powyższym kodzie zamiana odczytanego z portu szeregowego bajtu na zmienną typu integer aby dalsze polecenia (szczególnie print()) rozumiały ją jako liczbę dziesiętną.

Kolejny, bardziej rozbudowany i docelowy kod programu w tym arykule, rysować będzie kwadrat o kolorze zależnym od wartości odczytanej z portu (w skali szarości):

```
import processing.serial.*;
Serial myPort;
int BajtOdczytany;

void setup()
{
  size(300, 300);
     //ustalenie wielkości okna graficznego programu na
     //300x300pixeli
  background(0);
     //ustalenie czarnego koloru tła okna
  myPort = new Serial(this,"COM4", 300);
     //przyporządkowanie do komunikacji portu o nazwie COM4
}

void draw()
{
  if ( myPort.available() > 0)
   {
     BajtOdczytany = int(myPort.read());
     fill(BajtOdczytany);
          //deklaracja koloru wypełnienia każdego obiektu, który
          //powstanie w dalszej części programu,
     rect(100, 80, 100, 140);
          //narysowanie prostokąta w punkcie o współrzędnej 100,80,
          //szerokości 100pikseli i wysokości 140px
     delay(30);
   }
}
```

Ważne spostrzeżenia:

- Instrukcja delay(30) powoduje przerwę między kolejnymi 'przerysowaniami' ekranu, dzięki czemu zmniejsza się obciążenie obliczeniowe komputera, jednakże musi być odpowiednio dobrane (zawsze mniejsze od przerwy między kolejnymi wysłaniami sygnału przez Arduino), aby nie zwiększać opóźnienia pomiędzy sygnałem pojawiającym się na porcie a obrazem. Zbyt duże wartość parametru objawia się rosnącym w czasie użytkowania opóźnieniem między sygnałem wejściowym do Arduino oraz obrazem, co może prowadzić do przepełniania bufora portu szeregowego i utratę części danych.

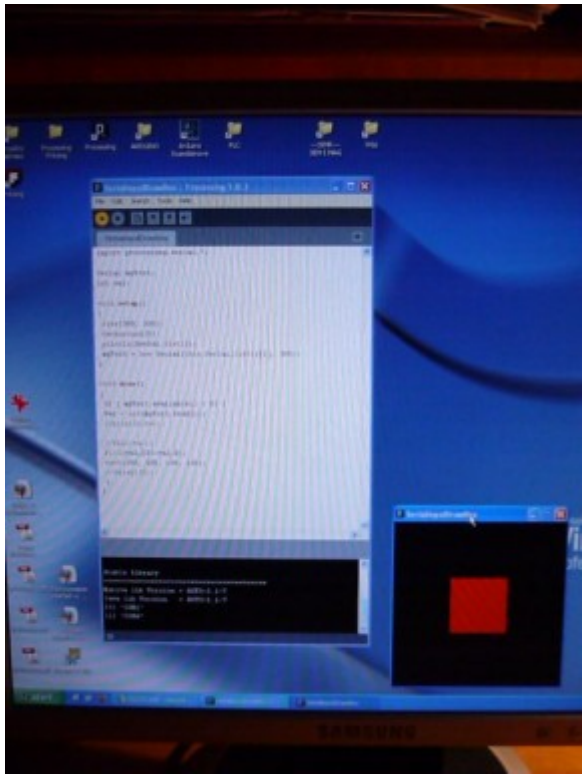
- Znaczny wpływ na czas wykonywania jednej pętli instrukcji ma wielkość obszaru rysunkowego, zatem dla wyższych częstotliwości wysyłania sygnału przez Arduino zalecam małe obszary rysunkowe i jak najkrótksze/najprostrze instrukcje sekcji draw.

Instrukcja fill(V): deklaruje kolor wypełnienia dla obiektu (obiektów) w skali szarości, przy czym dla V=0 mamy kolor czarny, a dla V=255 kolor biały.

Instrukcja fill(R,G,B): deklaruje kolor wypełnienia dla obiektu (obiektów) w formacie RGB, gdzie R – poziom nasycenia czerwienią, G – poziom nasycenia zielenią, B – poziom nasycenia błękitem. Wszystkie wartości z przedziału 0-255.

I tak zastosowanie instrukcji fill(BajtOdczytany, 255-BajtOdczytany, 0) we wcześniejszym programie skutkować będzie zabarwieniem kwadratu na kolor zielony przy braku oświetlenia, a na kolor czerwony przy maksymalnym oświetleniu.

Fotorezystor jest elementem inercyjnym, dlatego też po zasłonięciu lub oświetleniu stałą wartość rezystancji osiąga po pewnym czasie.



Filmik obrazujący działanie opisanego w artykule układu pomiarowego:
http://www.vimeo.com/5605883

Zastosowane w artykule programy:
http://myinventions.pl/005/arduino005.pde
http://myinventions.pl/005/processing005.pde

P.s.: Zwracam szczególną uwagę, że część poleceń, np. dotyczących transmisji szeregowej, różni się w Arduino i w Processingu. Pliki obu programów mają takie samo rozszerzenie, więc zdarzać się może również uruchamianie ich w nieodpowiednim programie. Przydatne jest zatem umieszczanie komentarzy na początku programu z informacją czy jest to skrypt processingu czy arduino.