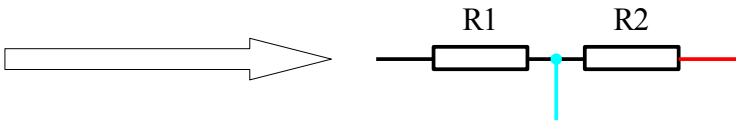
Oscyloskop (007; 20.07.2009; *arduino; processing*)

Sposób prezentacji danych pomiarowych w środowisku Processing opisany w artykule o pomiarze natężenia światła jest obrazowy, jednak mało przydatny. Przedstawię zatem sposób na rysowanie prostych wykresów w czasie rzeczywistym.

Potencjometryczny dzielnik napięcia.

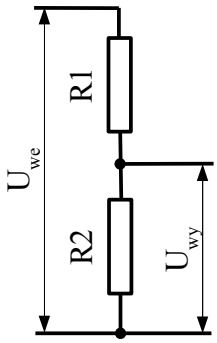
Jeśli chodzi o sam pomiar napięcia, możemy zastosować wcześniej opisany dzielnik napięcia z fotorezystorem lub też potencjometr pełniący funkcję sterowanego dzielnika napięcia.

Rodzajów potencjometrów jest niezliczona ilość, zajmijmy się najpopularniejszym, potencjometrem obrotowym, więc stwórzmy jego model następująco:

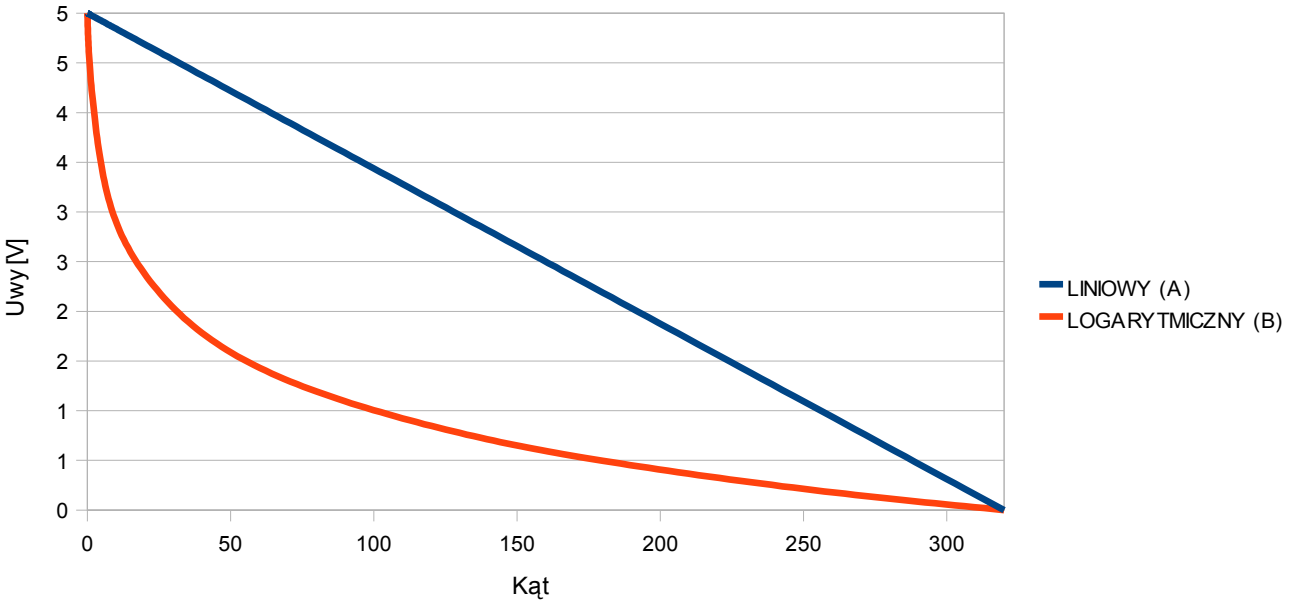


Oczywiście istnieje zależność: R1+R2=R, gdzie R jest wartością oporu potencjometru podaną na obudowie. Obrót gałki potencjometru powoduje wzrost jednej rezystancji i spadek drugie.
Wyróżniamy dwa typy potencjometrów: liniowe (ozn. A) i logarytmiczne (ozn. B).

Zastosujmy potencjometr jako dzielnik napięcia następująco:

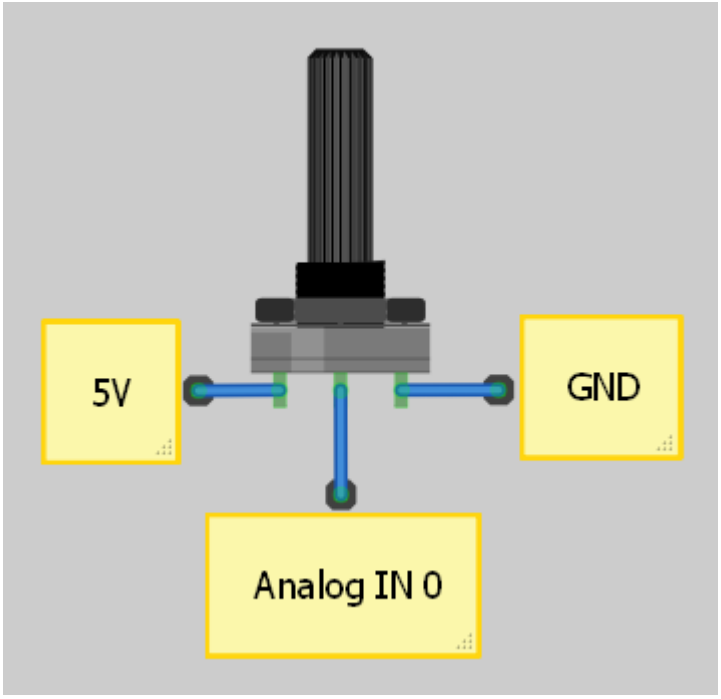


Zakładając typ potencjometru łatwo można teraz określić zależność napięcia wyjściowego od kąta obrotu. Poniższy wykres przedstawia przykładowe przebiegi (napięcie wejściowe: 5V, rezystancja potencjometru 100kOhm, pełen kąt obrotu 320stopni):



Arkusz kalkulacyjny z tabelami obliczeniowymi i powyższymi wykresami do pobrania:
http://myinventions.pl/007/RezystorowyDzielnikNapiecia.ods

Sposób podłączenia potencjometru do Arduino przedstawia się następująco:



Pomiar i wysłanie danych do komputera.

Kod do Arduino wygląda następująco:

```
void setup()
  {
  Serial.begin(9600);
  }
void loop()
{
  Serial.print(analogRead(0)/4, BYTE);
  delay(20);
}
```

Program ten realizuje pomiar napięcia z częstotliwością 50Hz i wysyłanie wartości jednobitowych na port szeregowy.

Prezentacja danych na komputerze (wykres w środowisku Processing).

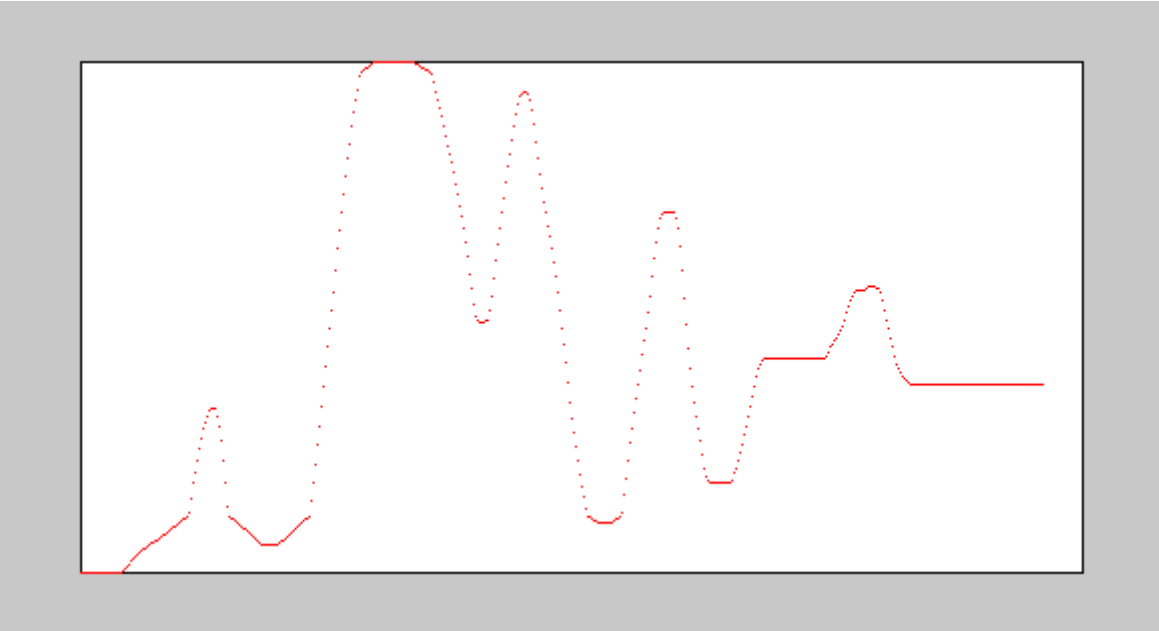
Najprostszy kod pozwalający rysować wykres odebranych danych wygląda następująco:

```
import processing.serial.*;
Serial myPort;
int X=40;
int Y;

void setup()
{
  size(580, 316);
  background(200);
  rect(40,30,500,255);
  stroke(255,0,0);
  myPort = new Serial(this,"COM4", 9600);
}
void draw()
{
  if (myPort.available() > 0)
  {
    Y = int(285-myPort.read());
    point(X,Y);
    X++;
    if (X>=540){
      X=40;
      stroke(0);
      rect(40,30,500,255);
      stroke(255,0,0);
    }
  }
}
```

Obszar wykresu jest prostokątem o szerokości 500px i wysokości 255px, przy czym przesunięty jest względem krawędzi okna o 40px w prawo i 30px w dół. W każdej pętli programu po sprawdzaniu warunku dostępności danych na porcie szeregowym obliczana jest współrzędna Y punktu wykresu (poprzez odejmowanie, bo chcemy mieć skierowaną w górę oś Y; wartości tak dobrane aby osiągać krańce obszaru wykresu), rysowany jest punkt o współrzędnych X i Y, zwiększana jest o jeden wartość współrzędnej X. Gdy wykres osiągnie brzeg obszaru rysowania (X=540) następuje przerysowanie nowego obszaru rysunkowego i zmiana współrzędnej X.

Uwaga: Jeśli dostrzeżemy rosnące w czasie opóźnienie między zmianą położenia potencjometru a zachowaniem wykresu, należy zmniejszyć częstotliwość próbkowania sygnału przez Arduino (program nie nadąża odbierać danych z portu szeregowego i rysować wykresu). Uruchomienie wygaszacza ekranu podczas działa programu również może spowodować opóźnienie.

Przykładowy wykres otrzymany otrzymany z zastosowaniem opisanego programu:

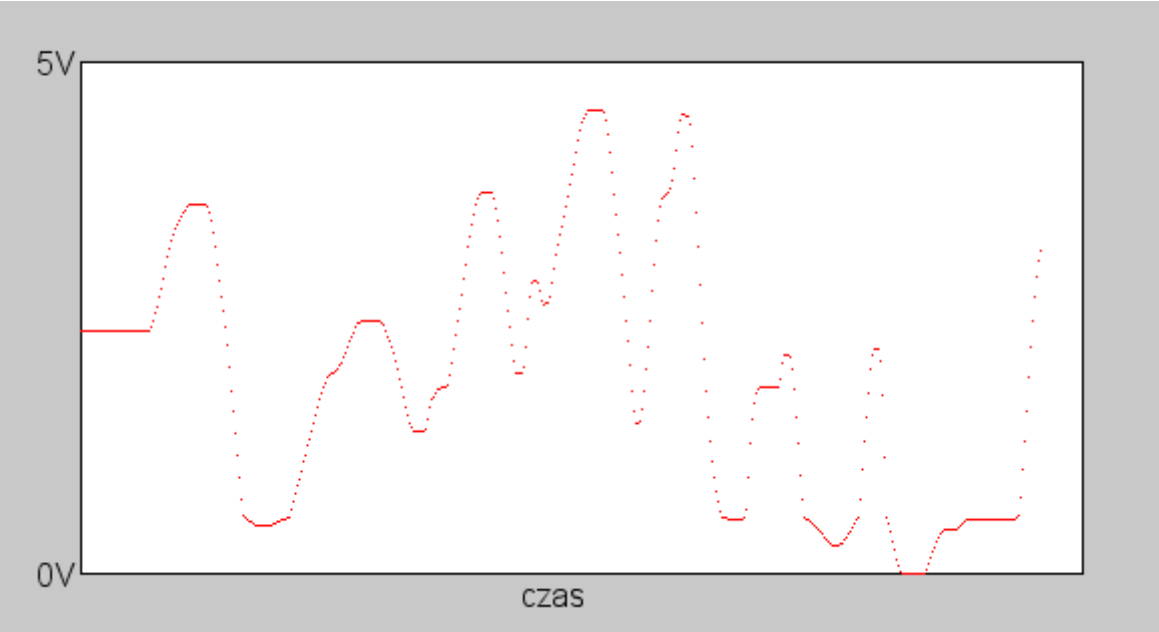


Dokonajmy opisu osi wykresu, oraz sprawdźmy ile czasu trwa narysowanie jednego pełnego wykresu. Na początek konieczne jest utworzenie czcionki dla programu. Wybieramy z menu *Tools/create font...* wybieramy czcionkę, ustalamy jej rozmiar na 16 i zatwierdzamy. W katalogu głównym pisanego programu (menu *sketch/show sketch folder*) pojawi się katalog data, a w nim utworzona czcionka. Czcionka w formacie *.vlw opisuje każdy znak w postaci bitmapy o rozdzielczości dostosowanej do podanej w czasie jej tworzenia wielkości.

```
import processing.serial.*;
Serial myPort;
int X=40;
int Y;
float t1 = 0;
float t2 = 0;

void setup()
{
  size(580, 316);
  background(200);
  rect(40,30,500,255);
  PFont font;                  //utworzenie zmiennej typu czcionka
  font = loadFont("ArialMT-16.vlw");
         //wczytanie do zmiennej podanej czcionki
  textFont(font,16);
         //ustawienie czcionki 'font' jako używanej w programie,
         //oraz ustalenie jej rozmiaru na 16 (wpisanie innego
         //rozmiaru niż podstawowy skutkuje rozciągnięciem
         //czcionki - utratą je jakości)
  fill(0);
  text("5V",18,37); //wstawienie tekstu 5V w punkcie o współ. 18,37
  text("0V",18,292);
  text("czas",260,302);
  myPort = new Serial(this,"COM4", 9600);

}
void draw()
{
  if ( myPort.available() > 0)
  {
    Y = int(285-myPort.read());
    stroke(255,0,0);
    point(X,Y);
    X++;
    if (X>=540){
      X=40;
      background(200);
      text("5V",18,37);
      text("0V",18,292);
      text("czas",260,302);
      stroke(0);
      fill(255);
      rect(40,30,500,255);
      t2 =float(millis())/1000;
      print("długość kreślenia przebiegu [s]: ");
      println(t2-t1);
      t1=float(millis())/1000;
      fill(0);
    }
  }
}
```

Pomiar czasu dokonany został z użyciem polecenia millis() zwracającego liczbę milisekund od startu programu. Obserwując konsolę w czasie pracy programu zauważamy, że pierwsza wartość jest większa od później podanych, co wynika z wliczenia do niej czasu uruchomienia programu. Program działa prawidłowo, jeśli kolejne wartości są do siebie zbliżone i wynoszą około 10sekund (500 punktów wykresu przy podawaniu sygnału z częstotliwością 50Hz).

Przykładowy przebieg wygląda następująco:



Wysokie częstotliwości.

Zwiększając częstotliwość wysyłania sygnału przez Arduino możemy doprowadzić do sytuacji, w której program nie nadąży rysować punkty na wykresie. Najprostszym sposobem pokonania tej trudności będzie dokonaniu pewnej ilości pomiarów z dużą częstotliwością, a następnie realizacja przerwy w wysyłaniu danych, aby mogły one zostać odebrane z bufora portu szeregowego i naniesione na wykres. Nie będzie to już niestety oscyloskop czasu rzeczywistego.

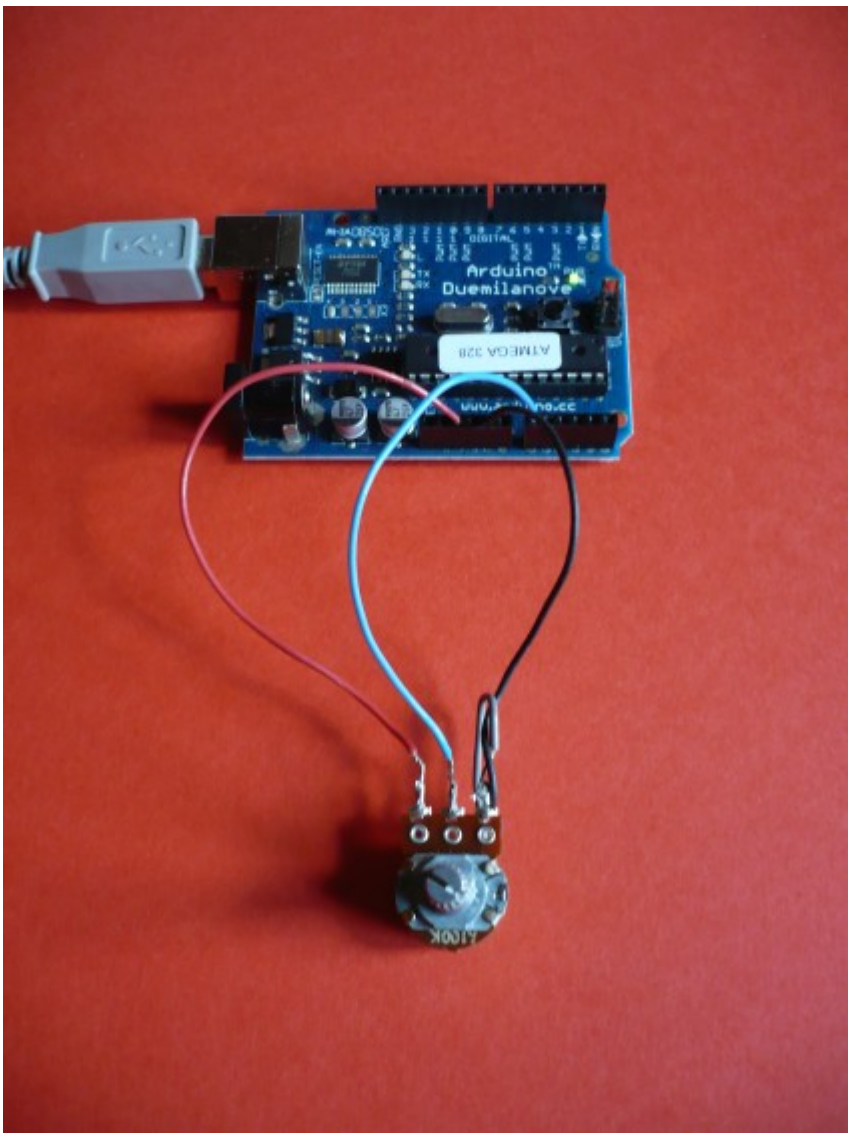
Przykładowy kod do Arduino dla próbkowania 200Hz, przerwy między pomiarami 20 sekund, oraz stanu wysokiego na wyjściu cyfrowym nr 13 na czas pomiaru (w celu zaświecenia diody):

```
void setup()
{
  Serial.begin(9600);
  pinMode(13, OUTPUT);
}

void loop()
{
  delay(2000);
  digitalWrite(13, HIGH);
  for(int i=1 ; i<=500;i++)
  {
    Serial.print(analogRead(0)/4, BYTE);
    delay(5);
  }
  digitalWrite(13, LOW);
  delay(20000);
}
```

Do tego zmodyfikowany program w processingu opisujący dodatkowo oś czasu i opóźniający czyszczenie wykresu:

```
import processing.serial.*;
Serial myPort;
int X=40;
int Y;

void setup()
{
  myPort = new Serial(this,"COM4", 9600);
  size(580, 316);
```

```
  background(200);
  stroke(0);
  rect(40,30,500,255);
  PFont font;
  font = loadFont("ArialMT-16.vlw");
  textFont(font);
  fill(0);
  text("5V",18,37);
  text("0V",18,290);
  text("0s",36,302);
  text("2,5s",532,302);
}
void draw()
{
  if (myPort.available() > 0)
  {
    Y = int(285-myPort.read());
    stroke(255,0,0);
    point(X,Y);
    X++;
  }
  else
  {
    delay(5000);
    X=40;
    background(200);
    stroke(0);
    fill(255);
    rect(40,30,500,255);
    fill(0);
    text("5V",18,37);
    text("0V",18,292);
    text("0s",36,302);
    text("2,5s",532,302);
  }
}
```



Na zdjęciach dostrzec można dodatkowy szary przewód – łączy on obudowę potencjometru z masą.

Filmik obrazujący działanie opisanego w artykule oscyloskopu czasu rzeczywistego:

http://www.vimeo.com/5678042

Zastosowane w artykule programy:

Oscyloskop czasu rzeczywistego:

http://myinventions.pl/007/OscyloskopRTarduino.pde

http://myinventions.pl/007/OscyloskopRTprocessing.pde

Oscyloskop 200Hz:

http://myinventions.pl/007/Oscyloskop200arduino .pde

http://myinventions.pl/007/Oscyloskop200processing.pde