Podstawy Automatyki i Sterowania IV

Wprowadzenie do programowania mikrokontrolerów

Projekt sonaru ultradźwiękowego

Damian Grabowski

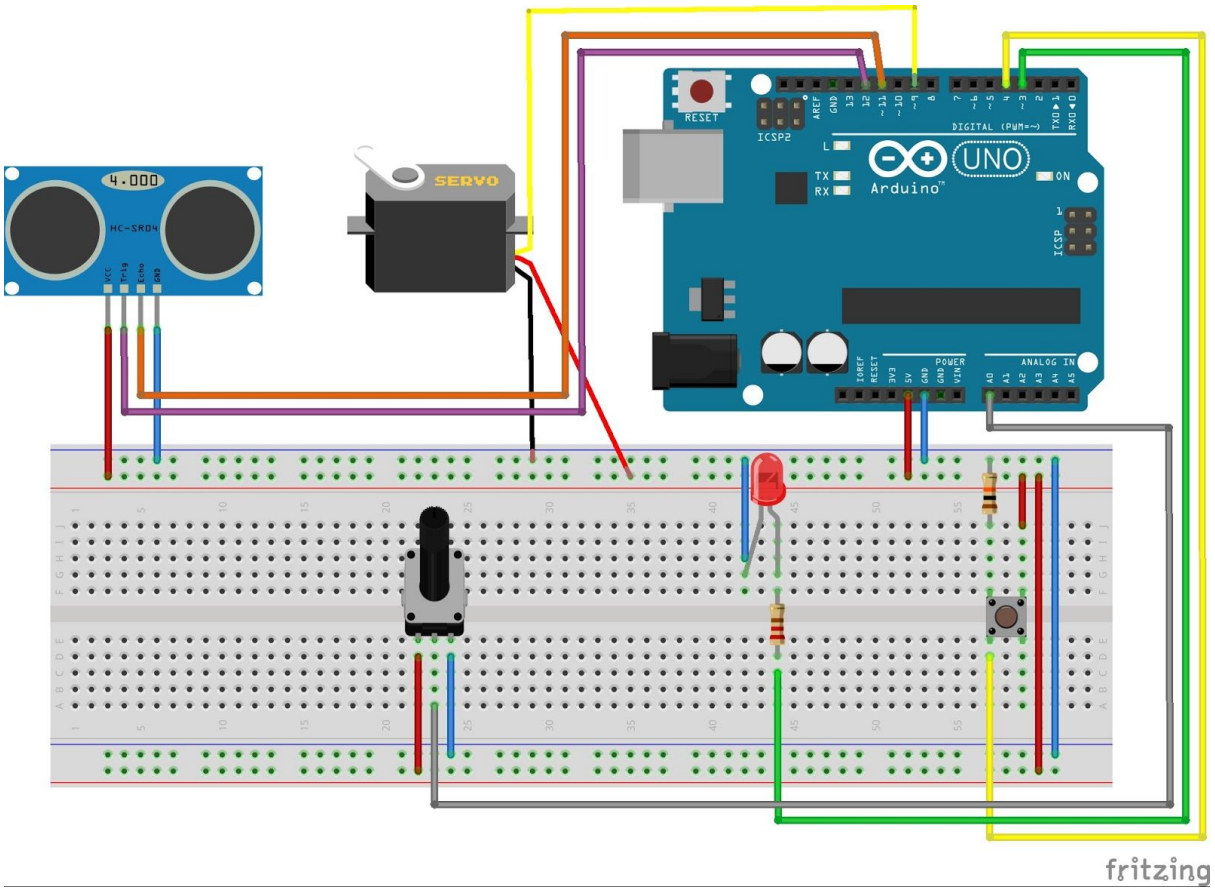
Założenia projektu

Celem projektu było utworzenie sterowanego mikrokontrolerem (***Arduino UNO***) obracającego się czujnika ultradźwiękowego (**HC-SR04**) na serwomechanizmie (**powerHD HD-1160A-medium**) - sumarycznie urządzenie tworzyłoby efekt podobny do sonaru działającego np. na okrętach morskich.W trybie automatycznym czujnik poruszałby się w zakresie od 0° do 180°, pobierając odległość co 1 stopień, zaś w trybie ręcznego sterowania (po naciśnięciu przycisku) obrót serwa jest kontrolowany przez obrót potencjometru. Tryb ręczny jest sygnalizowany przez palącą się czerwoną diodę.

Wykonanie można podzielić na 3 główne części:

- zmontowanie układu
- oprogramowanie mikrokontrolera
- stworzenie programu na stacji bazowej (tj. komputerze) obrabiającego pozyskane dane oraz adekwatne ich wyświetlanie

Konstrukcja układu



Rys. 1 - Konstrukcja układu

Na rysunku 1 nie przedstawiono oczywistego połączenia czujnika z serwem.

Schemat połączeń z płytką:

Czujnik HC-SR04:	Trigger	-	pin 12	-	Output
	Echo	-	pin 11	-	Input
Serwo	-	pin 9	-	Output	
Dioda	-	pin 3	-	Output	
Potencjometr	-	pin 0	-	Input	
Przycisk	-	pin 2	-	Input	

Do diody jest dołączony rezystor 220Ω, a do przycisku 10kΩ. Potencjometr również jest 10kΩ.

Oprogramowanie kontrolera

Program powstał w dedykowanym dla rodziny urządzeń *Arduino IDE*. Jedyną dołączoną biblioteką jest **servo.h**, zajmujące się obsługą serwomechanizmu. Program komunikuje się z komputerem za pomocą portu szeregowego (prędkość 9600 baudów, brak bitu parzystości, wiadomość długości 8 bitów, 1.0 bit-stop).

Program korzysta z jednego obiektu - **servo**, typu *Servo (do obsługi serwomechanizmu)*- oraz 3 zmiennych globalnych: **buttonState** (tryb ręczny / automatyczny), **pos** (początkowa pozycja), **change** (różnica położenia pomiędzy poszczególnymi inkrementacjami).

W części **void setup()**, następuje inicjalizacyjna konfiguracja urządzenia - ustawienia portów (Input/Output), podlinkowanie obiektu servo, oraz uruchomienie komunikacji portu szeregowego.

W głównym ciele programu **void loop()** powtarzają się następujące czynności:

- Sprawdzenie wejścia przycisku - jeżeli jest naciśnięty zmiana **buttonState**
- W zależności od buttonState:
    - jeżeli jest równy 0 (tryb automatyczny) - wczytanie pozycji serwa, a następnie jej zmiana o **change**
    - jeżeli jest równy 1 (tryb ręczny) - zapalenie diody, odczytanie stanu potencjometru (0-1023), mapowanie zczytanej wartości na przedział (0-180) oraz ustawienie pozycji serwa
- Niezależnie od krok poprzedniego następuje sformułowanie i wysłanie komunikatu do komputera:
    - zczytanie pozycji serwa, pomiar czasu na odpowiedź na czujniku odległości
    - przeliczenie czasu odpowiedzi na odległość (**w centymetrach**)

- formowanie komunikaty (zmienna string)
- wysłanie komunikatu

$$pXXdY\,Y$$

Komunikaty wysyłane mają powyższą formę, gdzie XX oznacza pozycję serwa, a YY oznacza zmierzoną odległość.
Przykładowo ***p10d25*** oznacza obiekt w odległości 25cm pod kątem 10°.

Główna pętla programu jest powtarzana co 17 ms (1s/60) co skutkuje w częstotliwości odświeżania 60 klatek na sekundę (dokładniej opisane w dalszej części)

Oprogramowanie stacji bazowej

Program powstał w środowisku *Processing* (opartym na języku java). Jego głównym zadaniem jest wizualna reprezentacja otrzymywanych danych.

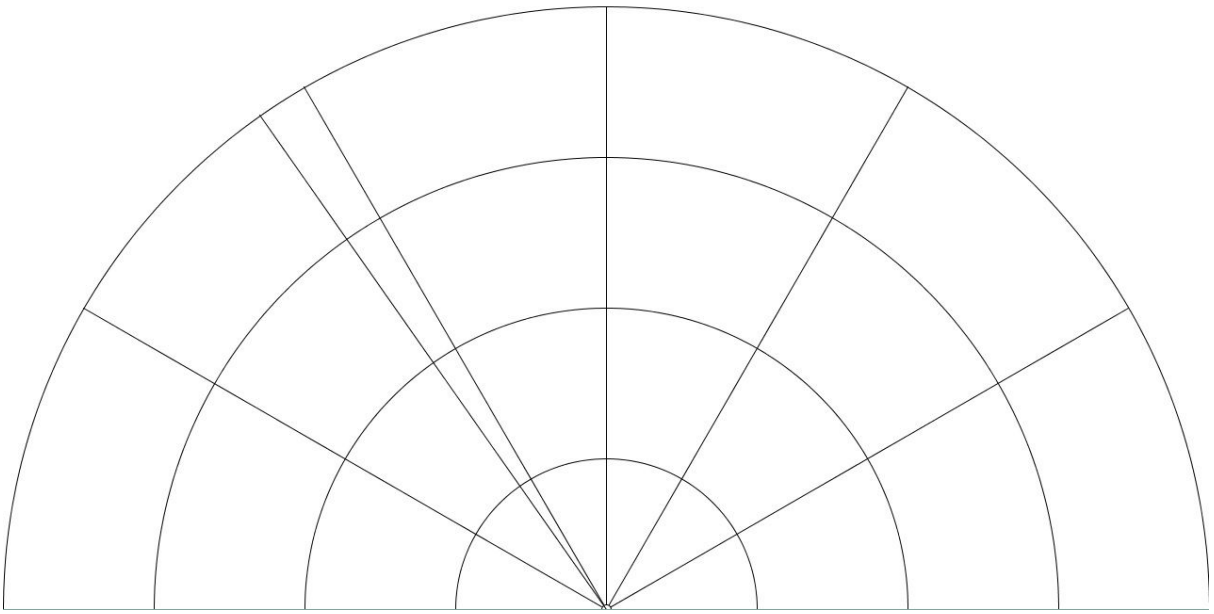
Główne zmienne:

- **M** - główny parametr skalujący , determinujący zasięg wyświetlanych pomiarów (najbardziej skrajny łuk, teoretycznie czujnik ma zasięg 200 cm, jednakże w praktyce jego zakres wiarygodności ogranicza się do 50 cm)
- **CurrentPosition[ ]** - tablica przechowująca na bieżąco otrzymywane pomiary
- **MeanPosition[ ][ ]** - dwuwymiarowa tablica przechowująca następujące dane:
  - **MeanPosition[0][ ]** - numer ostatniego pomiaru
  - **MeanPosition[1][ ]** - średnia wszystkich poprzednich pomiarów
  - **MeanPosition[2][ ]** - średnia kwadratów wszystkich poprzednich pomiarów

W części inicjalizacyjnej **void setup()** następuje wywołanie okna (o rozdzielczości 1280x720) oraz uruchomienie komunikacji przez port szeregowy (identyczne parametry jak w poprzednim programie).

Główna część **void draw()** rozpoczyna się od wygenerowania obrazu pomiarowego, stworzonego na wzór prawdziwych sonarów.

Łuk najbardziej skrajny odpowiada odległości **M**, zdefiniowanej wcześniej. Kolejne łuki zmierzając do środka to 0.75M, 0.5M i 0.25M. Główne linie biegnące od środka do krawędzi są ulokowane co 30°. Linia znajdująca się w okolicy 120° to linia wskazująca obecny kierunek zwrotu czujnika.



Rys. 2 - Graficzna reprezentacja przestrzeni pomiarowej

Następnym etapem działania programu jest odebranie komunikatu od kontrolera i wgranie ich do odpowiednich struktur. Pomiar bieżący jest wczytywany bezpośrednio do tablicy, zaś średnia i średnia kwadratów jest obliczana na bieżąco zgodnie z zależnościami:

$$S_{n+1} = \frac{n \cdot S_n + x_{n+1}}{n+1}$$

$$S^2_{n+1} = \frac{n \cdot S^2_n + (x_{n+1})^2}{n+1}$$

Gdzie:

n - numer pomiaru

S_n - Średnia z n pomiarów

S^2_n - Średnia kwadratów z n pomiarów

x_n - n-ta zmierzona wartość

Każdorazowa pętla główna kończy się generowaniem punktów na wykresie. Pierwsze są generowane punkty średnich położeń (większa średnica), których barwa zależy od stosunku odchylenia standardowego do odległości średniej. Innymi słowy, im statystycznie pewniejszy pomiar tym kolor bardziej “odpowiadający poprawności”

$\frac{2\sigma}{M} < 25\%$ - kolor zielony

$50\% > \frac{2\sigma}{M} > 25\%$ - kolor żółty

$\frac{2\sigma}{M} > 50\%$ - kolor czerwony

$$\sigma = \sqrt{S_n^2 - (S_n)^2}$$

Gdzie:

σ - odchylenie standardowe

Następnie generowane są punkty bieżącego położenia (ostatnio otrzymane z kontrolera), są to punkty o mniejszej średnicy niż punkty położeń średnich. Ich kolor zależy od:

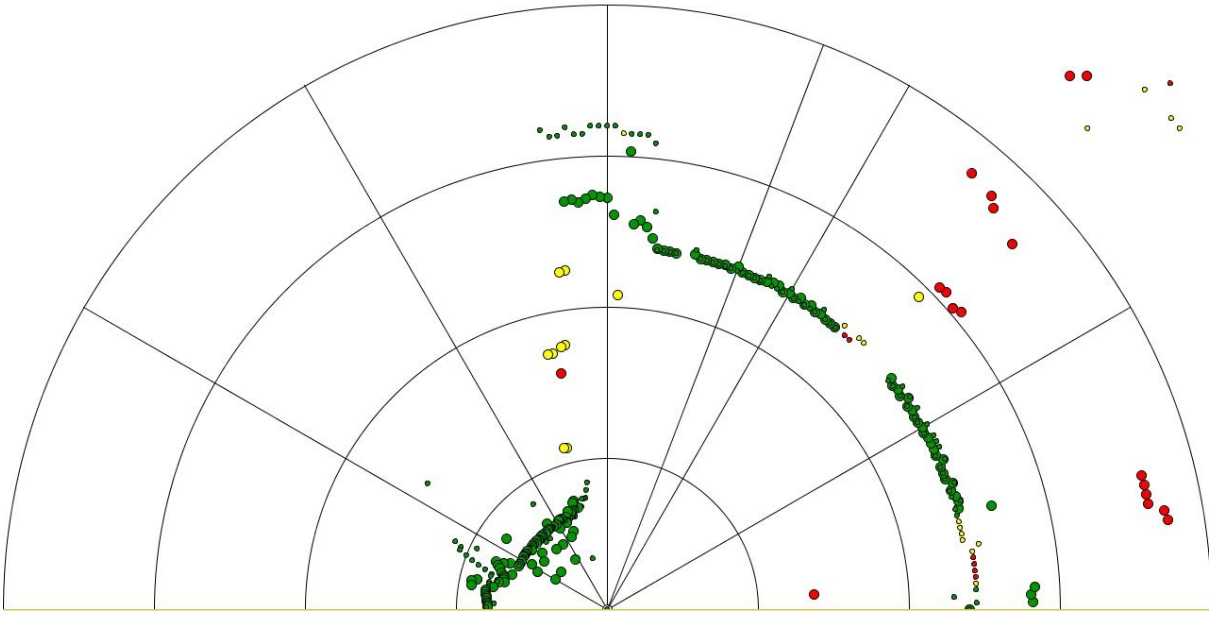
$\frac{|r - S_n|}{M} < 25\%$ - kolor zielony

$50\% > \frac{|r - S_n|}{M} > 25\%$ - kolor żółty

$\frac{|r - S_n|}{M} > 50\%$ - kolor czerwony

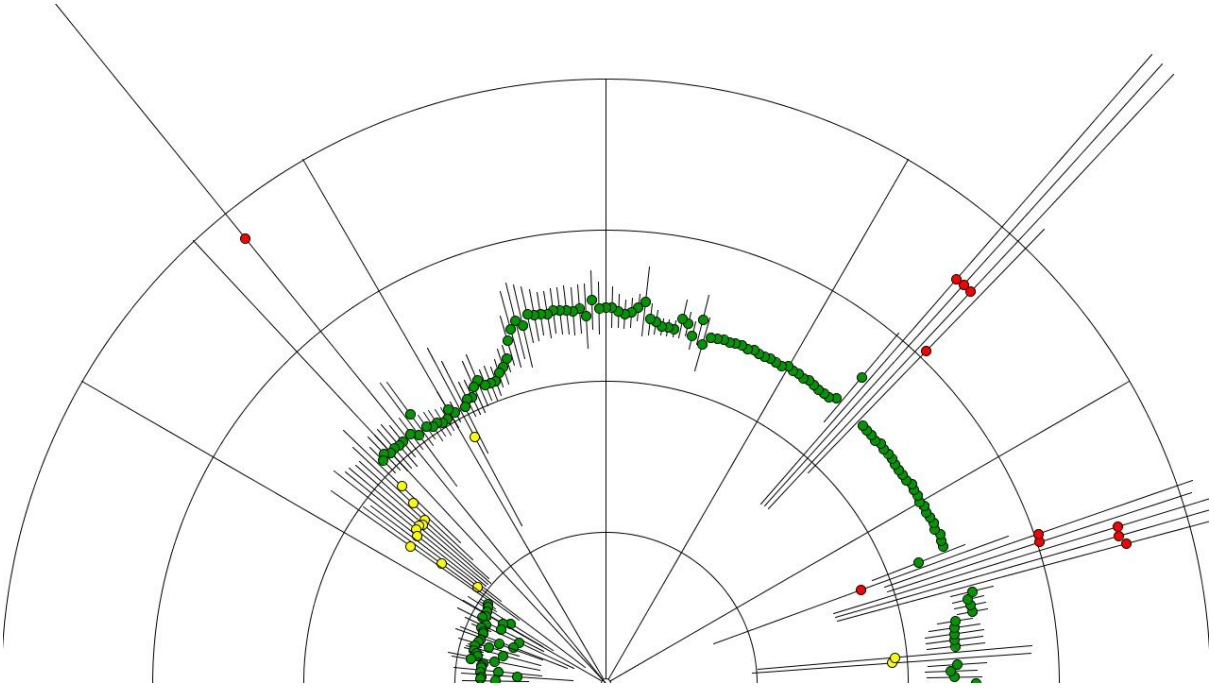
Gdzie:

r - ostatnio zmierzona odległość



Rys. 3 - Przykładowe pomiary sonaru

Program również może generować “linie prawdopodobieństwa”, czyli wartości średnie +/- odchylenie standardowe. Domyślnie ta opcja jest wyłączona gdyż w dużym stopniu zaciemnia wyświetlane pomiary.



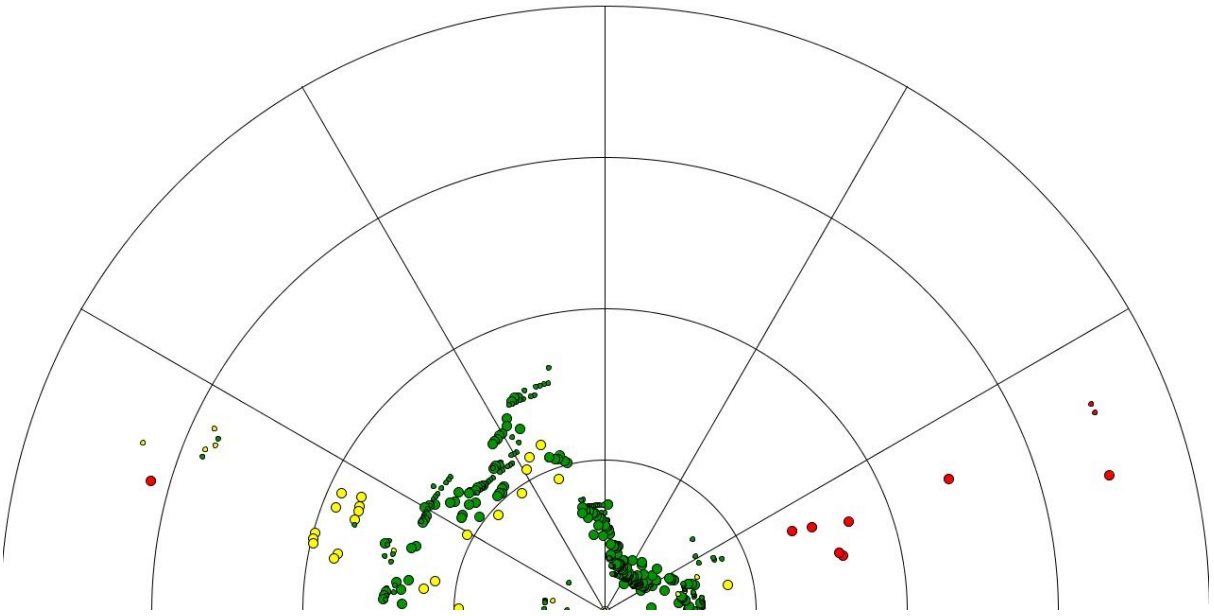
Rys. 4 - Wyświetlane pomiary z uruchomioną opcją pola prawdopodobieństwa

Linia pokazująca obecne położenie czujnika, jest odświeżana z częstotliwością równą wysyłanych komunikatów przez arduino. Częstotliwość 17ms, które daje częstotliwość 60Hz (60fps) sprawia złudzenie kompletnie płynnej.

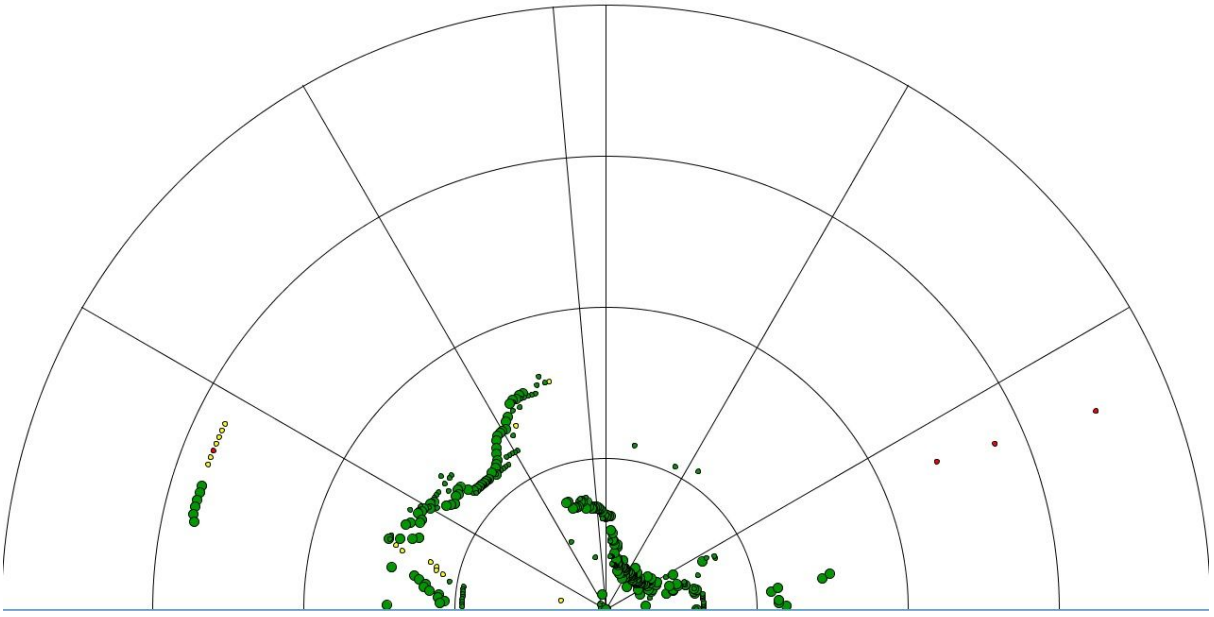
Wnioski

Analiza otrzymywanych wykresów, pozwala wysunąć interesujące wnioski - długo trwające pomiary niekoniecznie zwiększają pewność pomiarów (oczywiście zakładając stateczność otoczenia), zwłaszcza gdy aktywny jest ręczny tryb sterowania (dany punkt jest aktualizowany dużo częściej wartość średnia danego punktu “eksploduje” i wylatuje z zakresu widzialnego. Dzieje się tak dlatego że następuje akumulacja błędów, czyli pomiarów które zdecydowanie wykraczają poza akceptowalny zakres. Wprowadzenie średniej ucinanej, czyli takiej która nie akceptuje pomiarów wykraczających poza dane otoczenie punktu, usuwa ten błąd. Z drugiej strony użycie tej metody uniemożliwia jakąkolwiek analizę środowiska zmiennego

Jako rozwinięcie projektu można by stworzyć program tworzący mapę przeszkód na podstawie pomiarów, albo korzystając z kilku identycznych sonarów mapować powierzchnie.



Rys. 5 - Wyniki pomiarów po dłuższej chwili



Rys. 6 - Wyniki pomiarów po dłuższej chwili z użyciem średniej ucinanej