

# Cyfrowe przetwarzanie obrazów „Algorytmy wykrywania krawędzi” dokumentacja końcowa

Piotr Jastrzębski

## 1 Wstęp

Wykrywanie krawędzi jest jedną z podstawowych operacji cyfrowego przetwarzania obrazów, pozwalającą na uzyskanie informacji o obrazie i zawartych w nim obiektach. Na ich podstawie możliwe jest ustalanie cech obiektów i dalsze przetwarzanie - np. śledzenie obiektów w scenie, rozpoznawanie obiektów, analizowanie logiczne zawartości, wspomaga procesy wektoryzacji itp.

Istnieją liczne algorytmy pozwalające na analizowanie obrazów pod tym kątem. W tym projekcie użyte zostały algorytmy zaawansowane – Canny’ego oraz operator Sobela, opisane w punkcie 3 oraz prosty algorytm autorski oparty na progowaniu w obrazie, streszczony w punkcie 2.2 dokumentacji. Wszystkie te rozwiązania operują na zadanym zbiorze obrazów o różnym rodzaju i stopniu zaszumienia. Program pozwalający na wprowadzenie zaszumienia w obrazie został przygotowany, a wyniki jego działania zostały użyte do tworzenia obrazów testowania pozwalających na badanie jakości algorytmów wykrywania krawędzi.

## 2 Algorytm prosty

### 2.1 Działanie

Działanie programu dla wybranego atrybutu filtr można opisać następująco:

1. Inicjalizacja – Pobranie aktualnie obrabianego obrazka, wczytanie filtra. Zastosowany filtr jest postaci:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad (1)$$

Suma wartości poszczególnych elementów filtru jest równa 0. Dzięki czemu obszary obrazka o jednakowej barwie po zastosowaniu filtru będą miały wartość zerową, im gwałtowniejsza będzie natomiast zmiana barwy na obrazku (różnica pomiędzy sąsiednimi pikselami), tym wynik działania filtru będzie większy.

2. Filtracja – Dla każdego piksela, poza brzegowymi (nie można tam zastosować pełnego filtru), obliczany jest wynik działania filtru, i zapisywany w strukturze pomocniczej.
3. Normalizacja – Wartości otrzymane w poprzednim punkcie normalizowane są do jedynki, a następnie rozciągane na zakres 0-256. Dzięki temu różnice pomiędzy poszczególnymi wartościami stają się bardziej znaczące. Następnie wartości ze struktury pomocniczej przenoszone są na raster.
4. Progowanie – Uzyskany w poprzednim punkcie raster poddawany jest progowaniu, pikselom o wartościach przekraczających zadany próg przypisywany jest kolor czarny, a pozostałym biały. Uzyskany w ten sposób obrazek dzięki dużemu kontrastowi pozwala użytkownikowi w łatwy sposób rozróżnić krawędzie wykryte przy użyciu filtru.
5. Wyświetlenie nowego obrazka

Wynik działania kroków 1-4 zostaje wyświetlony w oknie programu.

Zastosowany algorytm charakteryzuje się niezwykłą prostotą i pozwala w zadowalający sposób wykrywać zarówno pionowe jak i poziome krawędzie w niezaszumionych obrazach. Słabo sprawdza się natomiast dla obrazów zaszumionych (szczególnie wrażliwe, na szum typu salt and pepper).

## 2.2 Struktura logiczna

Klasa główna Wczytywanie:

- wczytuje obrazek(git.jpg) jako BufferedImage
- wywołuje klasę GraphicsX, jako argument podaje wczytany obrazek

Klasa GraphicsX:

- zwraca tablicę pikseli obrazka (getCalosc())
- filtruje obrazek na podstawie wzoru (filtr()):  $p(x, y) = 8 * p(x, y) - p(x - 1, y - 1) - p(x - 1, y) - p(x - 1, y + 1) - p(x, y - 1) - p(x, y + 1) - p(x + 1, y - 1) - p(x + 1, y) - p(x + 1, y + 1)$
- wywołuję klasę Progowanie

- zapisuję obraz wynikowy na dysku(save()) jako git1.jpg

Klasa Progowanie:

- przyjmuję jako argumenty obrazek(BufferedImage) oraz próg
- dla pikseli większych od progu przypisuje wartość 255
- dla pikseli mniejszych od progu przypisuje wartość 0

Klasa ObrazFrame:

- tworzy ramkę programu
- umieszcza w ramce obraz wejściowy oraz obraz wyjściowy po przekształceniach

### 3 Algorytmy zaawansowane

#### 3.1 Detektor Sobela

Detektor Sobela wykorzystywany jest, w cyfrowym przetwarzaniu obrazów, do wykrywania krawędzi. Swoje działanie opiera o dyskretne różniczkowanie umożliwiające aproksymację pochodnych kierunkowych intensywności obrazu w ośmiu kierunkach, co 45°. Jego implementację zaczerpnięto z forum internetowego allexperts.com2 stosując jednak pewną modyfikację. Zmieniono parametr level, dzięki czemu wykryte krawędzie na obrazie, są lepiej widoczne. W działaniu, algorytm ten świetnie realizuje swoje zadanie, co prezentują przykładowe wyniki.

#### 3.2 Detektor krawędzi Cannyego

Detektor Cannyego jest wieloetapowym mechanizmem wykrywania krawędzi w obrazie. Spośród wszystkich detektorów krawędzi, detektor Cannyego w [3] uważany jest za optymalny, gdyż zapewnia:

- dobrą detekcję – wykrywa tak dużo krawędzi, jak to tylko możliwe
- dobre umiejscowienie – wykryta krawędź leży możliwie blisko krawędzi rzeczywistej
- minimalną odpowiedź – szum z obrazu wejściowego nie powinien tworzyć krawędzi fałszywych, a znalezione krawędzie są oznaczane tylko jeden raz

Powysze wymagania są spełnione, gdyż detektor Cannyego wykorzystuje rachunek wariancyjny za pomocą którego wyszukiwana jest funkcja optymalizująca dany funkcjonał. Optymalna funkcja opisana jest przez sumę czterech wykładniczych warunków, ale może być przybliżona pierwszą pochodną funkcji Gaussa.

### 3.2.1 Redukcja szumu

Detektor krawędzi Cannego wykorzystuje filtr bazujący na pierwszej pochodnej funkcji Gaussa, ponieważ jest czuły na obecność szumu w surowym nieobrobionym obrazie. Początkowym krokiem, jest zatem dokonanie splotu obrazu z filtrem Gaussa. Efektem tego działania jest rozmazany obraz, który charakteryzuje się brakiem znacznych zakłóceń. Macierz rozmycia Gaussa może być dowolną macierzą kwadratową. We wzorze 2 przedstawiono przykładową macierz filtra  $5 \times 5$  wykorzystywanego do obróbki obrazu ze współczynnikiem  $\sigma = 1.4$ . Zgodnie z [2], odchylenie standardowe filtra Gaussa jest pierwszym z sześciu parametrów metody. Im jest ono większe, tym mniej "fałszywych" krawędzi zostanie rozpoznanych, jednocześnie wydłużając czas samej konwolucji.

$$B = \frac{1}{59} \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * A \quad (2)$$

### 3.2.2 Wyszukiwanie natężenia gradientu

W związku z faktem, że wykryta w obrazie krawędź może być skierowana w różnych kierunkach, algorytm Cannego wykorzystuje cztery filtry do detekcji poziomych, pionowych oraz ukośnych krawędzi na wstępnie wygładzonym obrazie. Inne operatory detekcji krawędzi (np. Krzyż Roberts'a, Prewitt, Sobel) zwracają wartości pierwszej pochodnej jedynie dla kierunku poziomego ( $G_y$ ) i kierunku pionowego ( $G_x$ ). Gradient krawędzi oraz jej kierunek mogą być określony na podstawie wzorów: odpowiednio

$$G = \sqrt{G_x^2 + G_y^2} \quad (3)$$

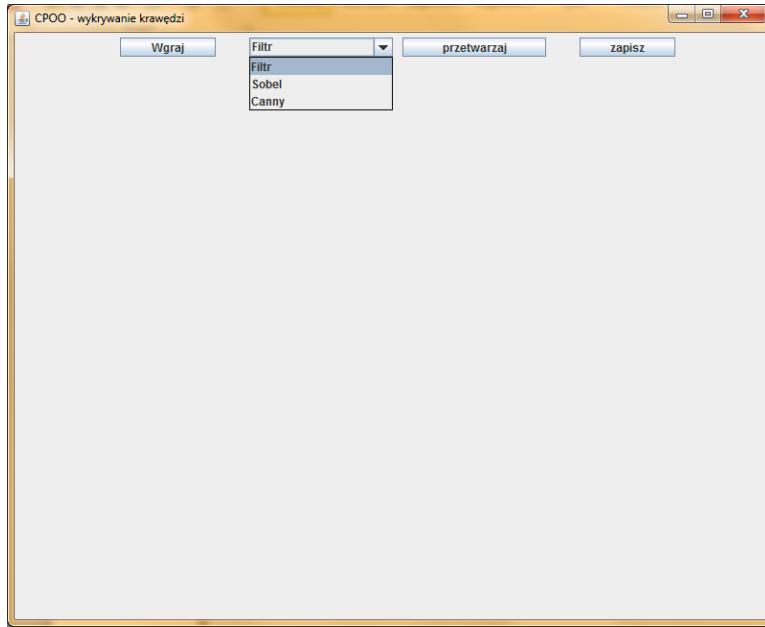
oraz

$$\Theta = \arctg\left(\frac{G_y}{G_x}\right). \quad (4)$$

Zastosowanie konwolucji z maskami np. Sobela, pozwala na odnalezienie modułu gradientu w obrazie. Kąt wykrytej krawędzi zaokrąglony jest do  $45^\circ$  dając cztery podstawowe przypadki reprezentujące odpowiednio: pion ( $N - S$ ), poziom ( $E - W$ ) oraz dwie przekątne ( $SE - NW$ ) i ( $SW - NE$ ).

### 3.2.3 Usuwanie niemaksymalnych pikseli

Usuwanie niemaksymalnych pikseli w wykrytych krawędziach sprowadza się do "pocieniań" krawędzi w sposób zapewniający ich ciągłość i zachowanie szerokości jednego piksela. Dla każdego z pikseli rozpatrywane są dwa piksele sąsiednie wybrane na podstawie najmniejszego, uprzednio



Rysunek 1: Wyniki symulacji (pominięto pozycje z  $t \approx 0$ )

obliczonego gradientu. Jeśli właściwy piksel nie ma intensywności większej niż sąsiedzi, jest zerowany.

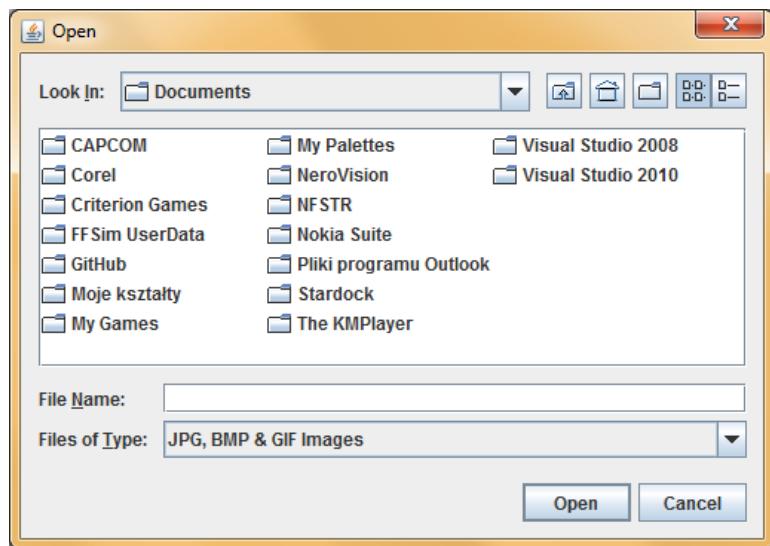
### 3.2.4 Progowanie z histerezą

Progowanie z histerezą wymaga podania dwóch wartości progowych gradientu,  $T_1$  oraz  $T_2$ . Jeśli intensywność krawędzi jest większa niż  $T_2$ , krawędź uznaje się za pewną, zaś jeśli jest mniejsza niż  $T_1$ , jest ona usuwana z obrazu. Dla wartości pośrednich, do już wykrytych krawędzi są dołączane następne piksele mimo spadku intensywności, aż do osiągnięcia dolnego progu wykrywania. Takie postępowanie zapobiega dzieleniu krawędzi w miejscach słabszego kontrastu.

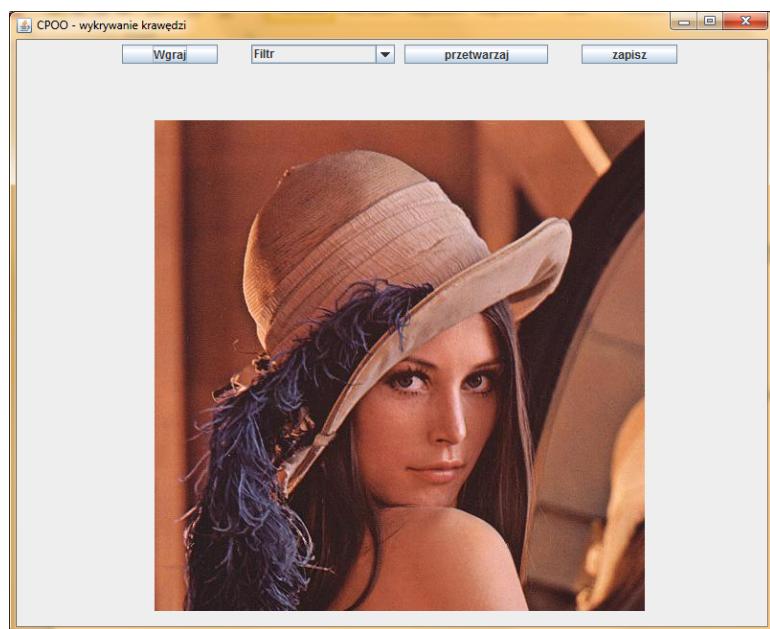
## 4 Instrukcja

Po uruchomieniu programu użytkownikowi ukazuje się następujące okno: Dobrze widoczny, prosty interfejs umożliwia wykorzystanie i prezentację wyników działania zaimplementowanych algorytmów. Poszczególne elementy interfejsu odpowiadają za następujące działania:

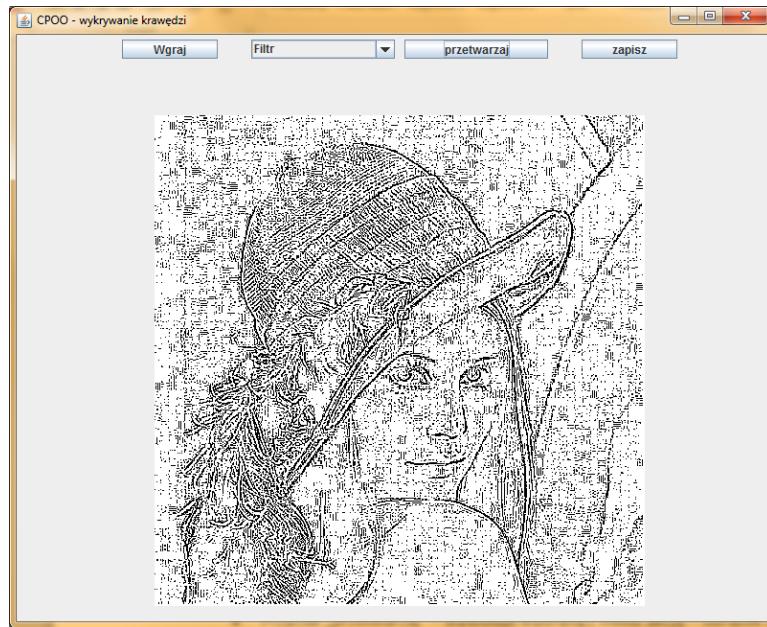
- Przycisk „Wgraj” — umożliwia wskazanie pliku graficznego do obróbki. (Rysunek 2) Po wczytaniu obrazka jest on widoczny w oknie aplikacji. (Rysunek 3)



Rysunek 2: Wskazanie pliku graficznego do obróbki



Rysunek 3: Podgląd obrazka



Rysunek 4: Rysunek

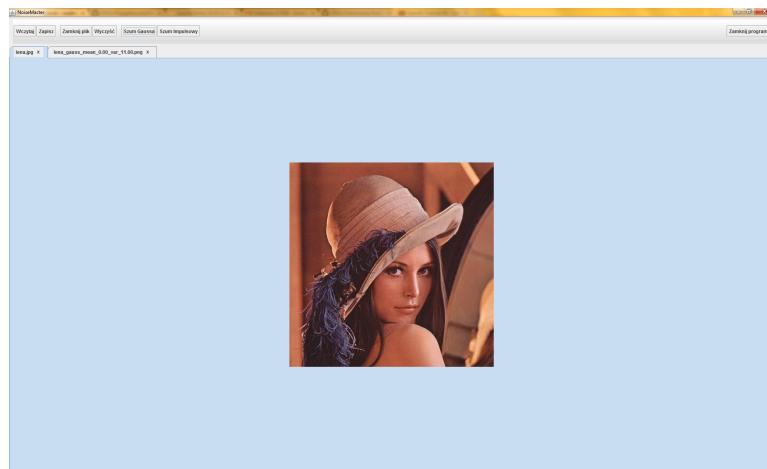
- Menu rozwijane — umożliwia wybór spośród dostępnych algorytmów
- Przycisk „przetwarzaj” — wywołuje wybraną z menu akcję – obrazek źródłowy widoczny w oknie aplikacji zostaje zastąpiony wynikiem działania algorytmu. Obraz po przetworzeniu przedstawiono na rysunku 4.
- Przycisk „zapisz” — pozwala na zapisanie aktualnie widocznej wersji obrazka na dysku twardym

Oprócz opisanej powyżej funkcjonalności właściwej aplikacji, do przetestowania poszczególnych algorytmów użyta została zewnętrzna paczka do generacji szumu. Jest ona dostępna pod adresem: <http://db.apache.org/derby/releases/release-10.10.1.1.cgi>.

Nabudowany interfejs użytkownika przedstawiono na rysunku 5.

Podobnie jak aplikacja właściwa, interfejs programu jest niezwykle prosty w obsłudze:

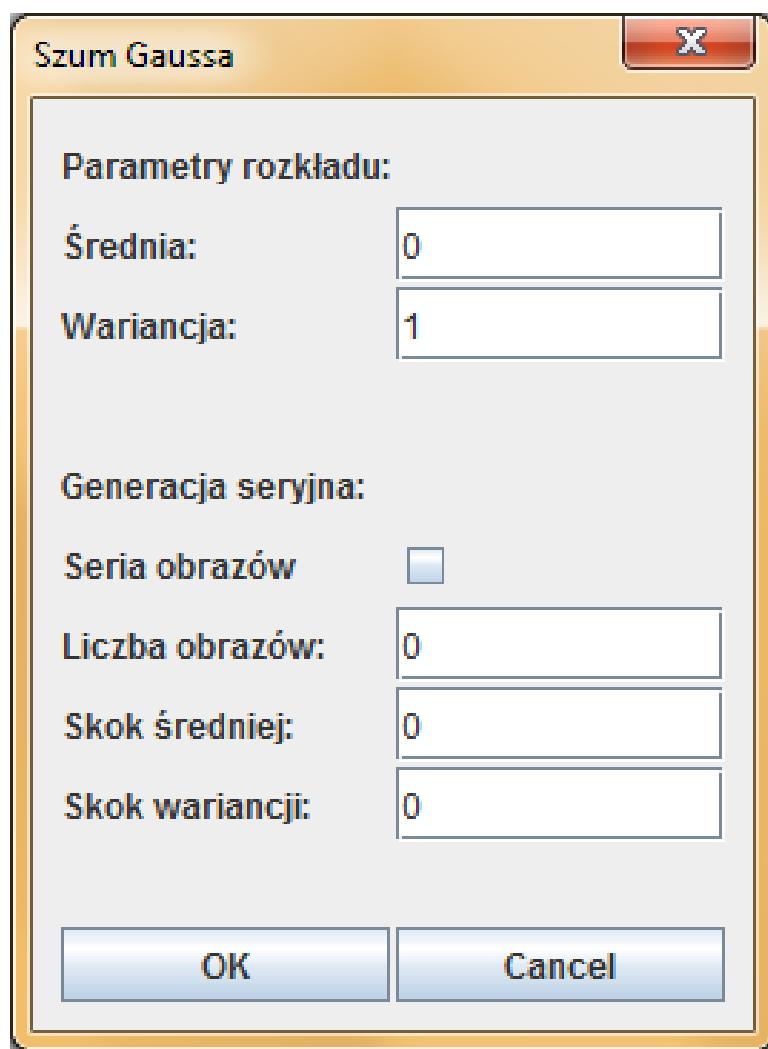
- Przycisk „Wczytaj” — umożliwia wczytanie obrazka z pliku
- Przycisk „Zapisz” — umożliwia zapisanie obrazka do pliku
- Przycisk „Zamknij plik” — zamyka aktywne okno podglądu
- Przycisk „Wyczyść” — zamyka wszystkie zakładki



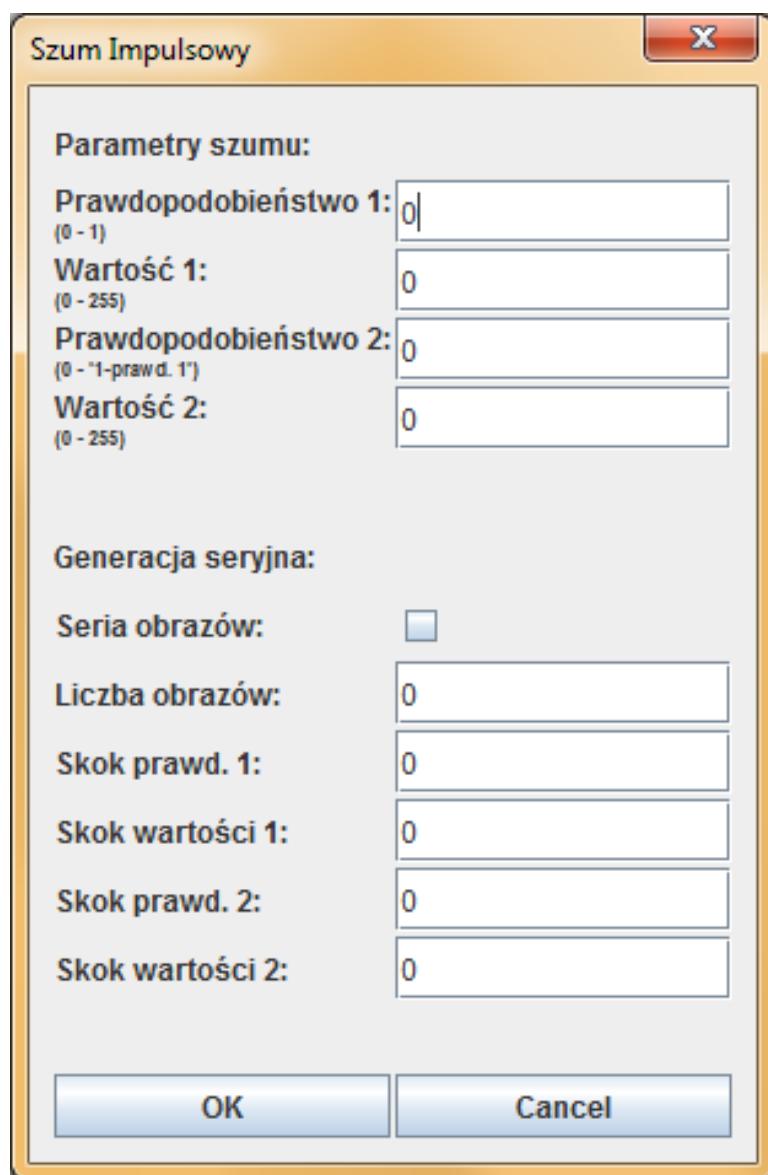
Rysunek 5: Interfejs użytkownika

- Przycisk „Szum Gausa” — pozwala określić parametry szumu, i nanieść go na aktywny obrazek. Opcje szumu przedstawiono na rysunku 6.
- Przycisk „Szum Impulsowy” — pozwala określić parametry szumu, i nanieść go na aktywny obrazek. Obrazek z naniesionym szumem jest dostępny w nowej zakładce programu, dzięki czemu możliwe jest niezależne przeglądanie, porównywanie i zapisywanie wybranego obrazka, co widać na rysunku 7.

## 5 Testy



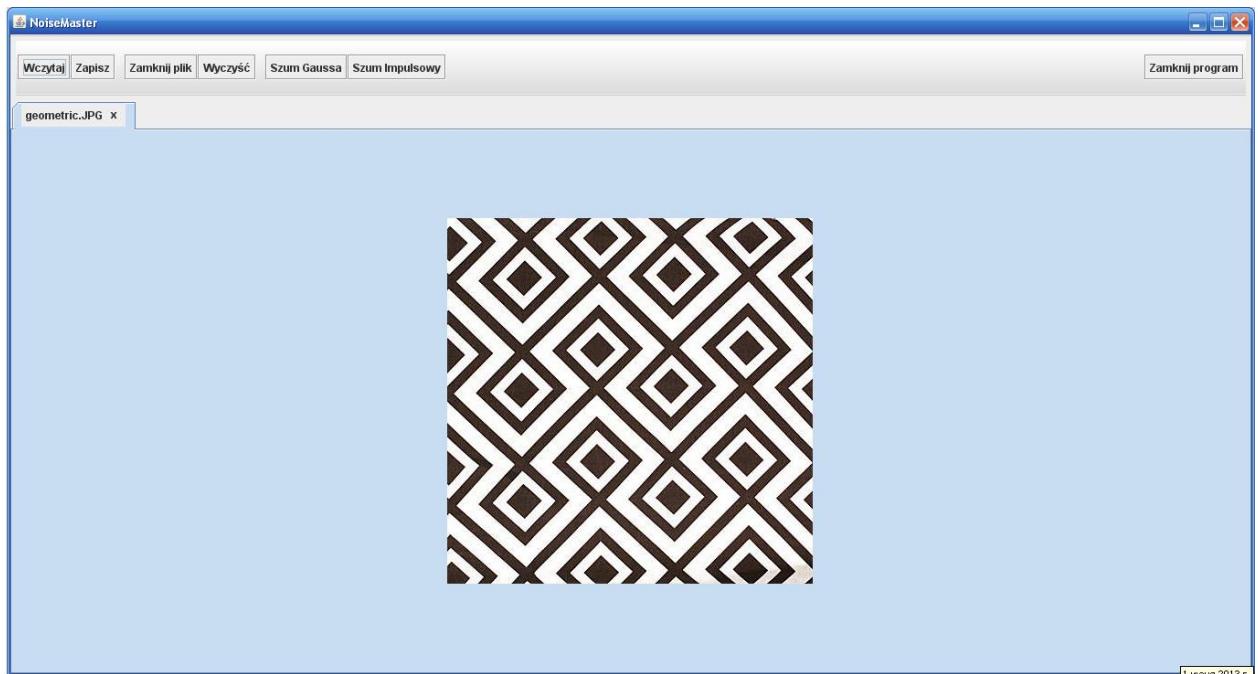
Rysunek 6: Opcje szumu Gaussa



Rysunek 7: Zaszumiony obraz

## 5 Testy

Narzędzie do generacji szumu *NoiseMaster* pozwala na dodawanie szumu Gaussowskiego oraz szumu impulsowego. Szum Gaussa odawany jest do obrazu poprzez dodawanie do wartości każdego z pikseli obrazu losowej wartości losowanej przy pomocy rozkładu normalnego. Szum impulsowy jest dodawany poprzez dodawanie do wartości pikseli obrazu jednej z dwóch zadanych wartości wariancji  $v_1$  i  $v_2$ , przy określaniu dwóch wartości prawdopodobieństw wystąpienia zmiany wartości  $p_1$  i  $p_2$ .



Rys.1 Główne okno programu do zaszumienia obrazków NoiseMaster.





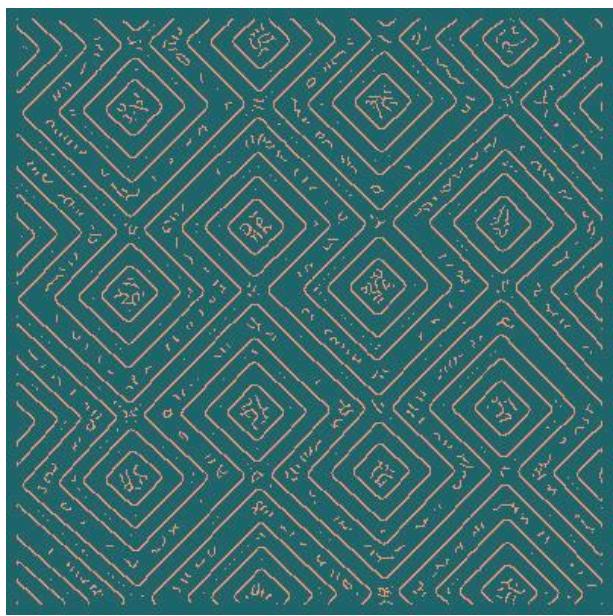
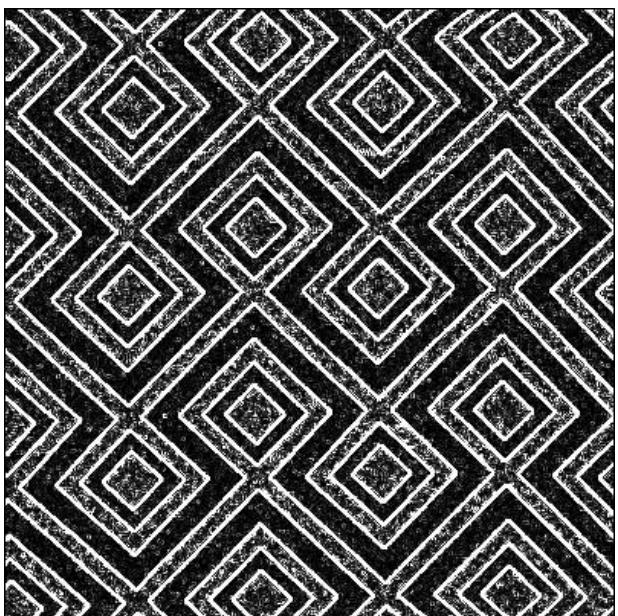
Rys.1.1.1. Zestawienie wyników detekcji krawędzi na obrazie nie posiadającym zaszumień oraz zakłóceń. Lewy górny rysunek to obraz wejściowy, prawy gorny – to obraz otrzymany po zastosowaniu prostego algorytmu do wykrywania krawędzi. Lewy dolny – to obraz otrzymany po zastosowaniu filtra Soblera. Prawy dolny – to obraz otrzymany po zastosowaniu algorytmu Canny'ego.



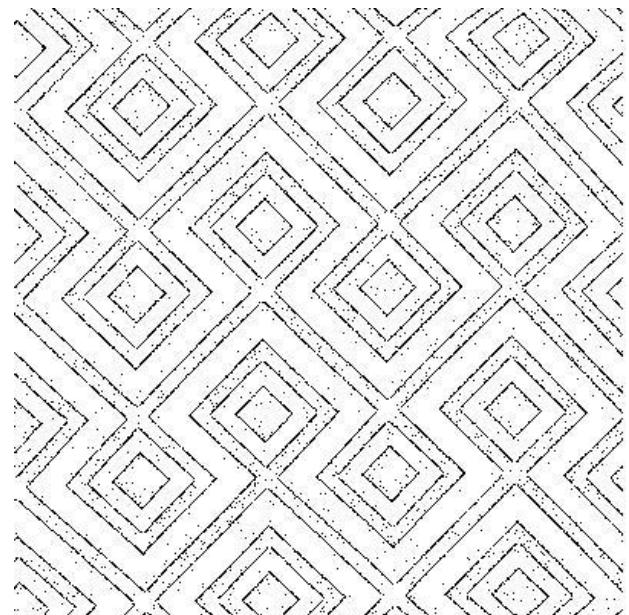
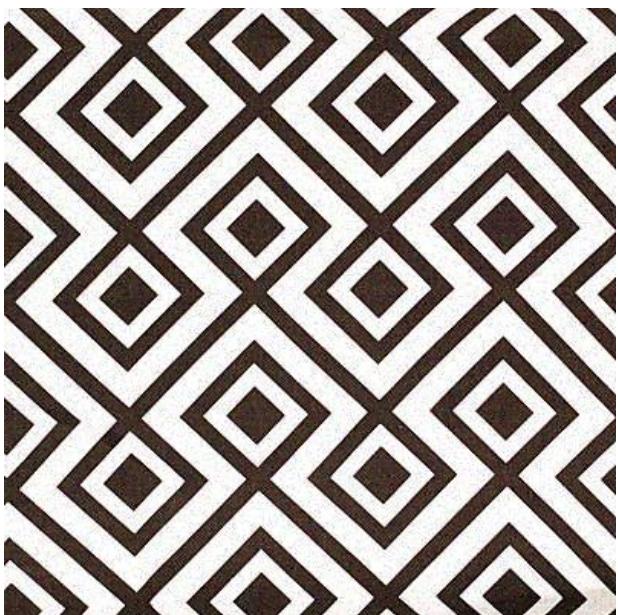


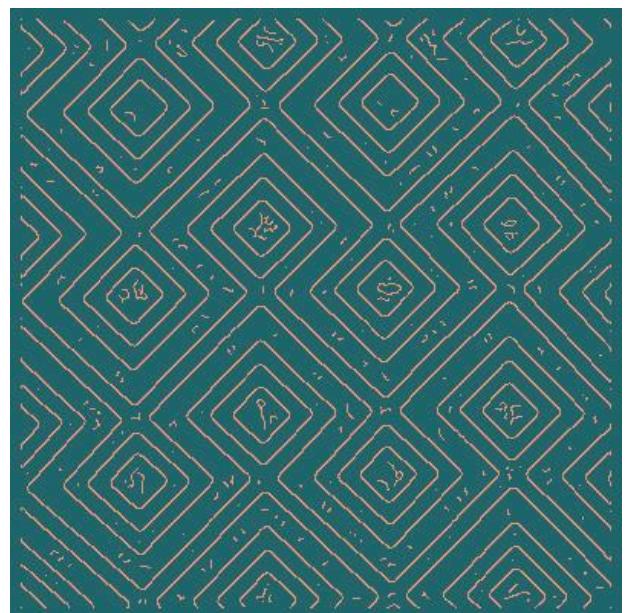
Rys. 1.1.2. Zestawienie wyników detekcji krawędzi na obrazie posiadającym zakłocenia o rozkładzie normalnym  $N(0,10)$ . Kolejność obrazów jak na Rys.1.1.1.



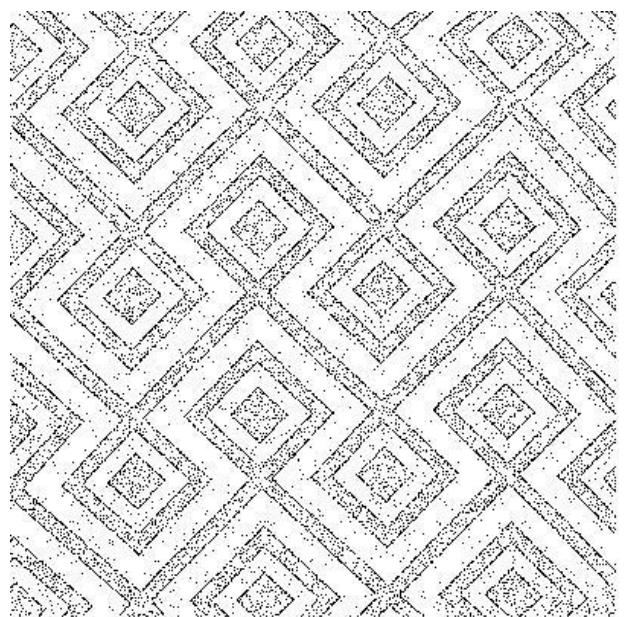
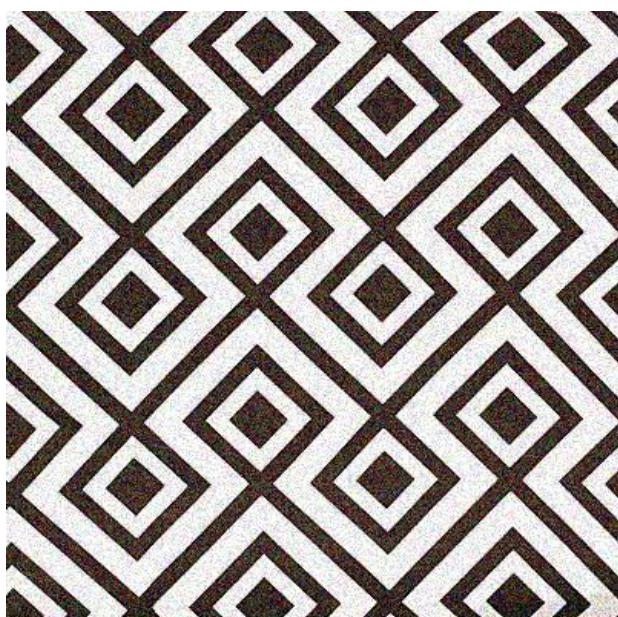


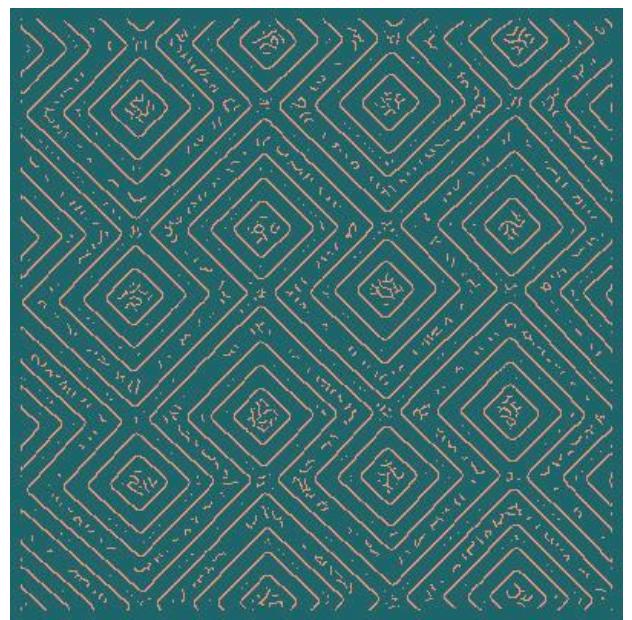
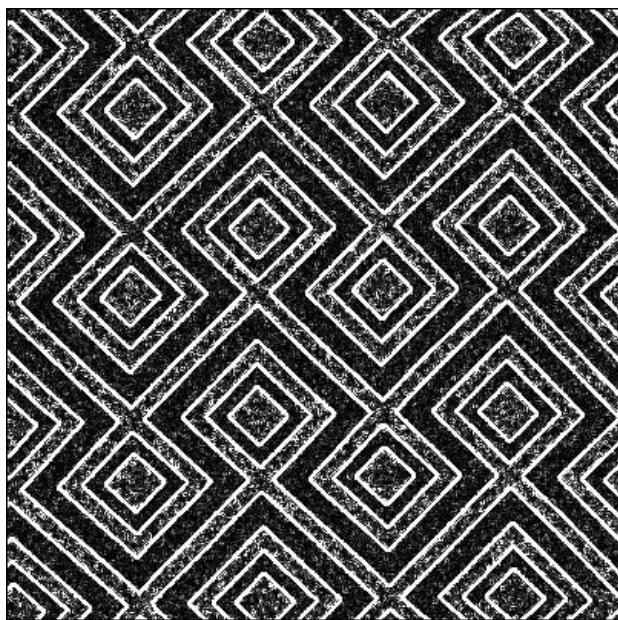
Rys. 1.1.3. Zestawienie wyników detekcji krawędzi na obrazie posiadającym zakłocenia o rozkładzie normalnym N (0,20). Kolejność obrazów jak na Rys.1.1.1.



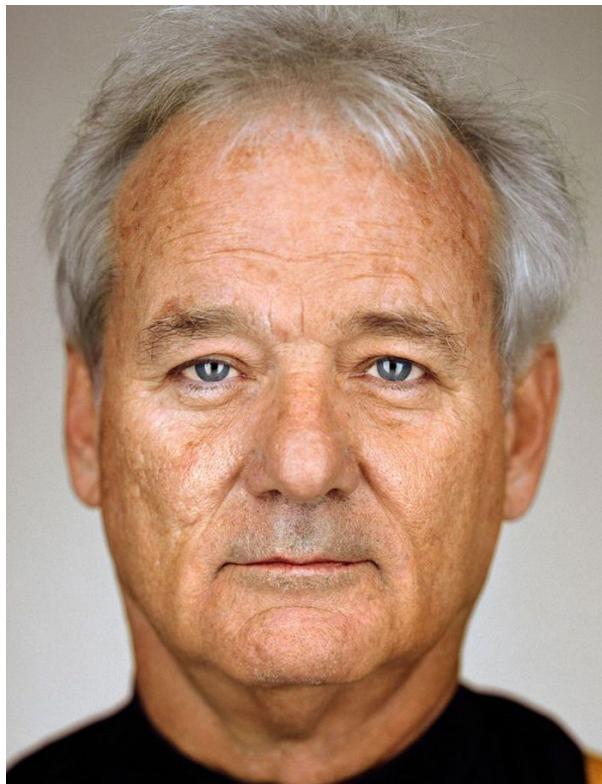


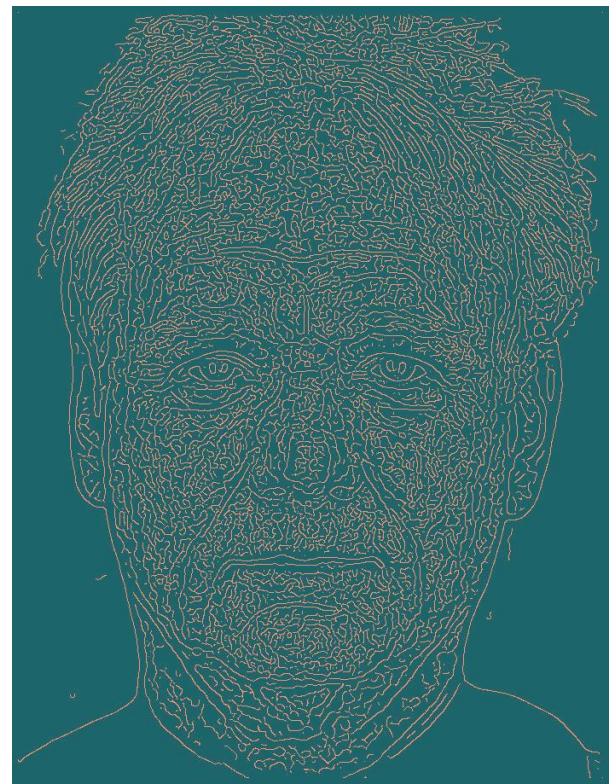
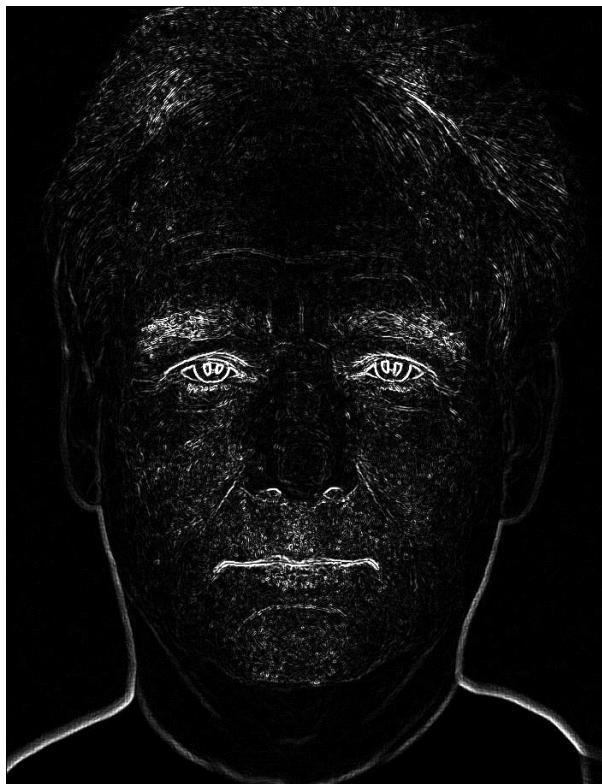
Rys.1.1.4. Zestawienie wyników detekcji krawędzi na obrazie posiadającym zakłucenia impulsowe o parametrach  $p1=0.2$ ,  $v1=30$ ,  $p2=0$ ,  $v2=0$ . Kolejność obrazów jak na Rys.1.1.1.



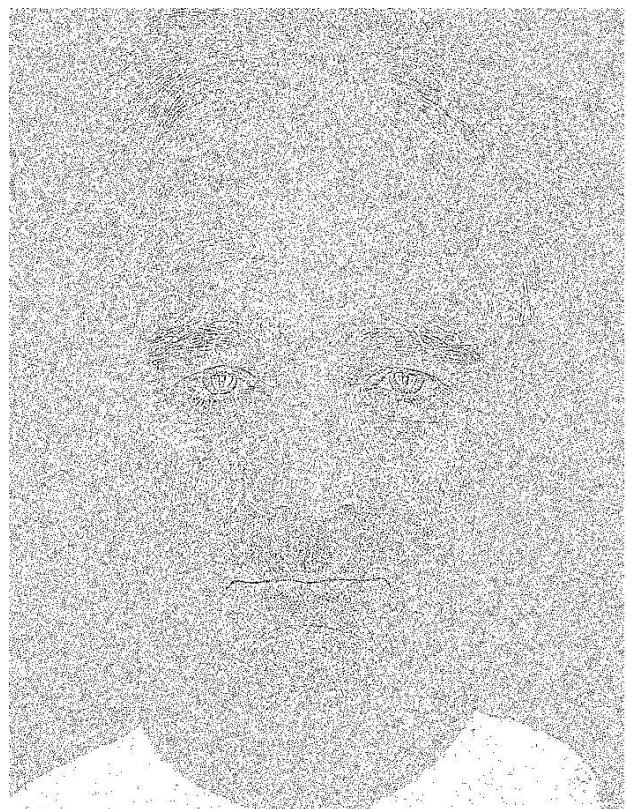
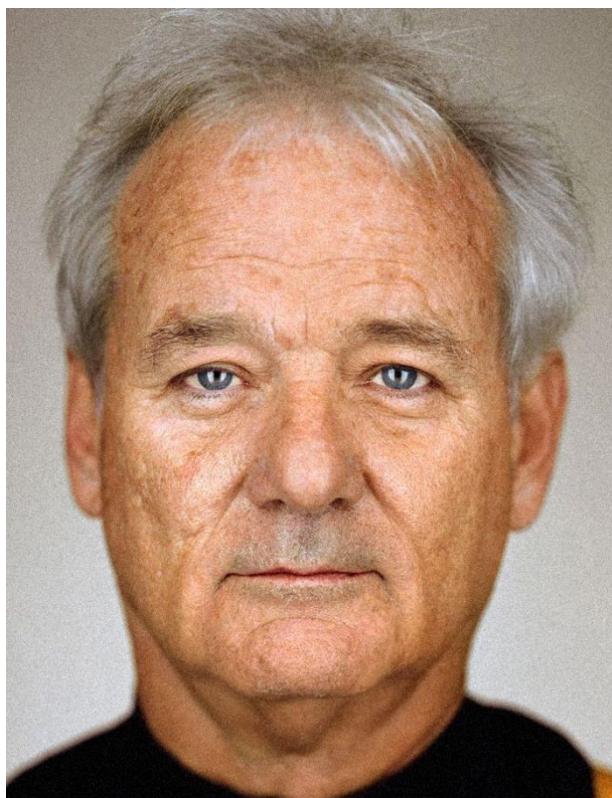


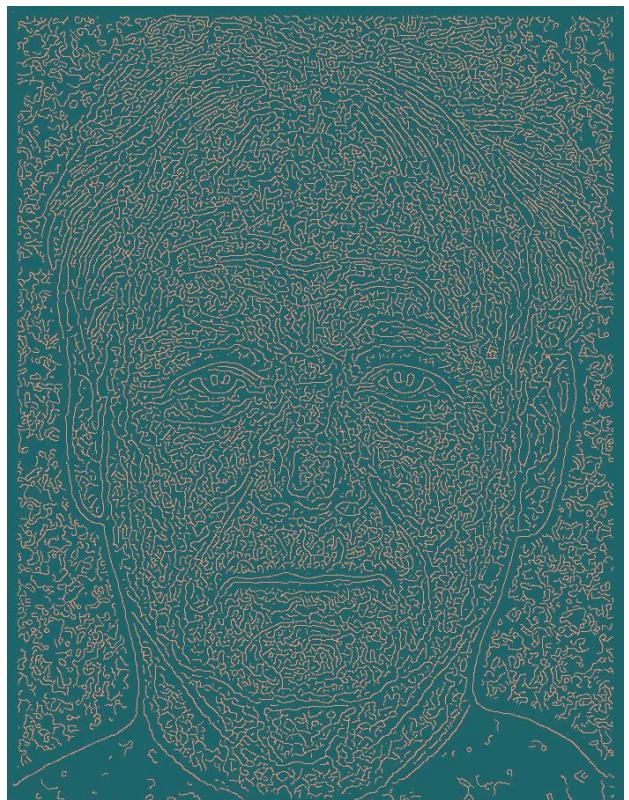
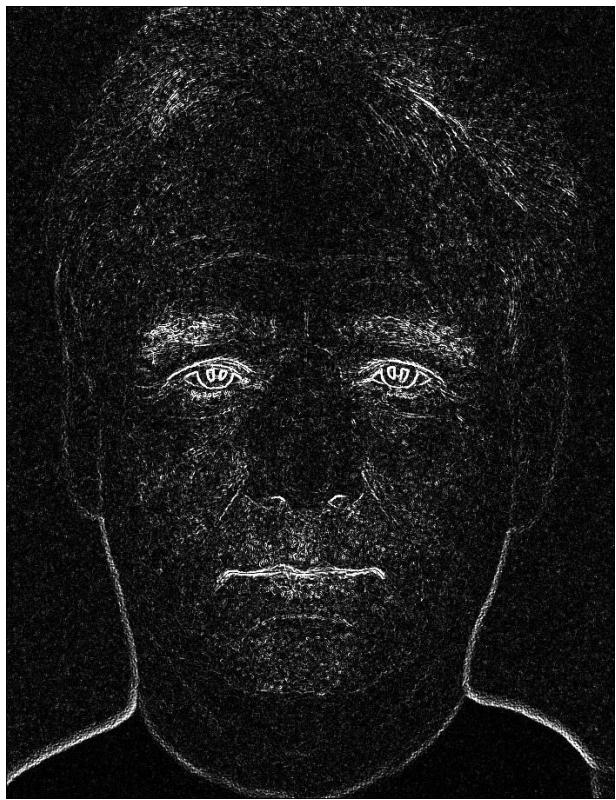
Rys.1.1.5. Zestawienie wyników detekcji krawędzi na obrazie posiadającym zakłocenia impulsowe o parametrach  $p1=0.3$ ,  $v1=40$ ,  $p2=0.3$ ,  $v2=40$ . Kolejność obrazów jak na Rys.1.1.1.



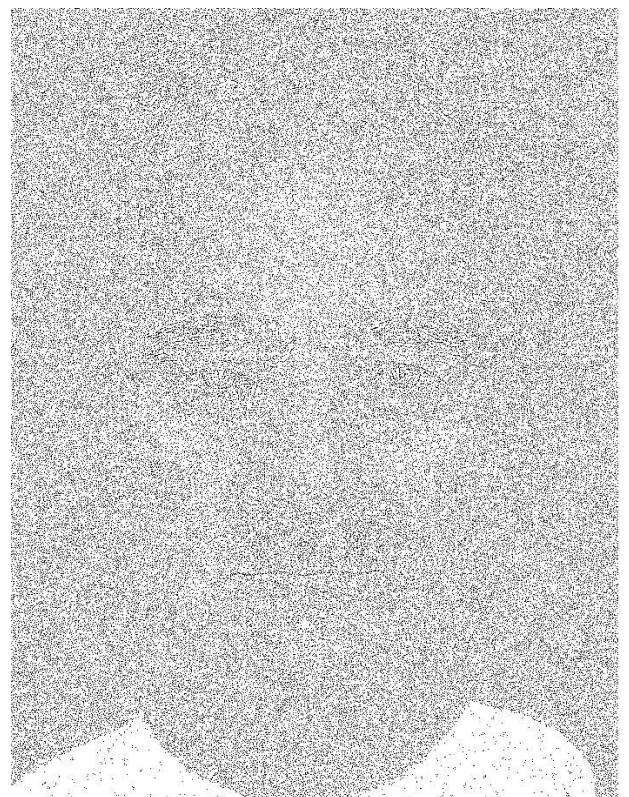
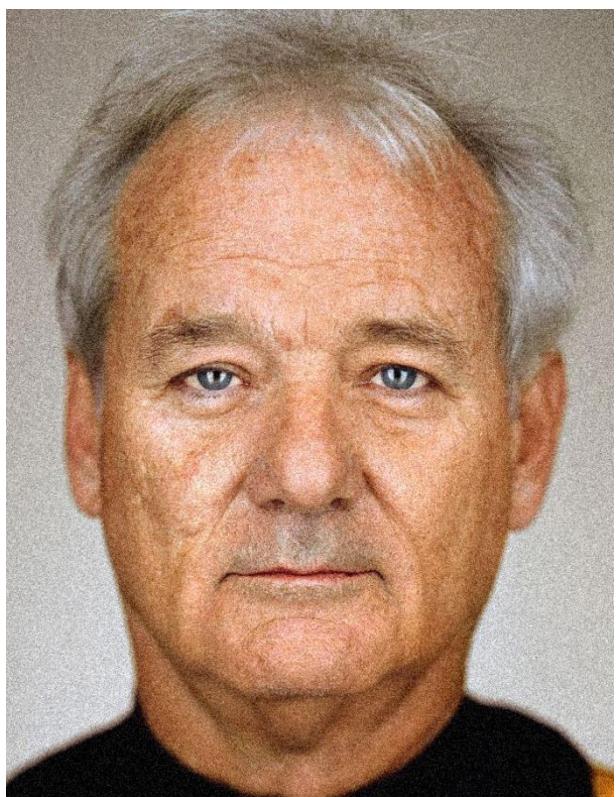


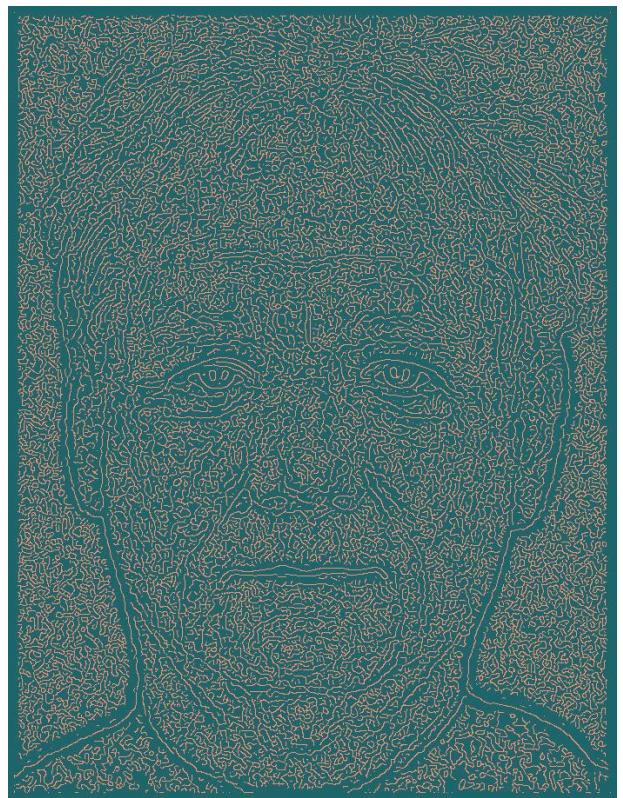
Rys.1.2.1. Zestawienie wyników detekcji krawędzi na obrazie nie posiadającym zaszumień oraz zakłóceń. Kolejność obrazów jak na Rys.1.1.1.



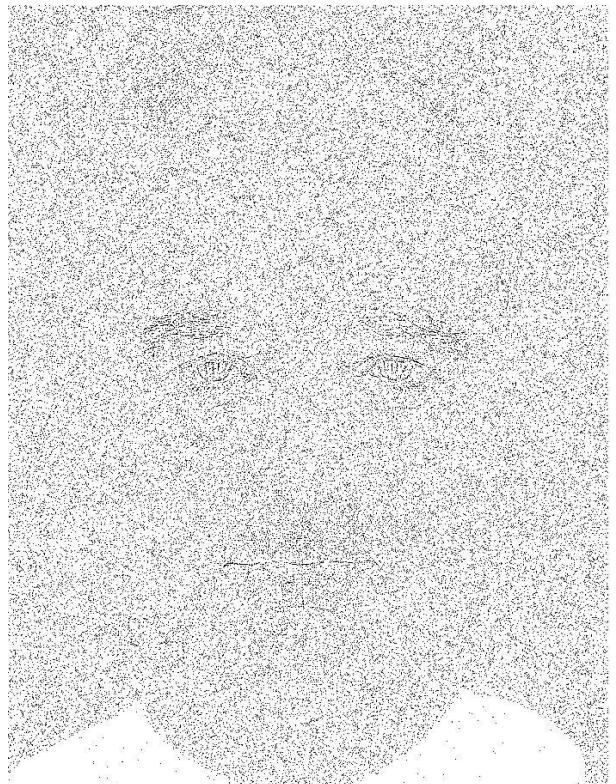
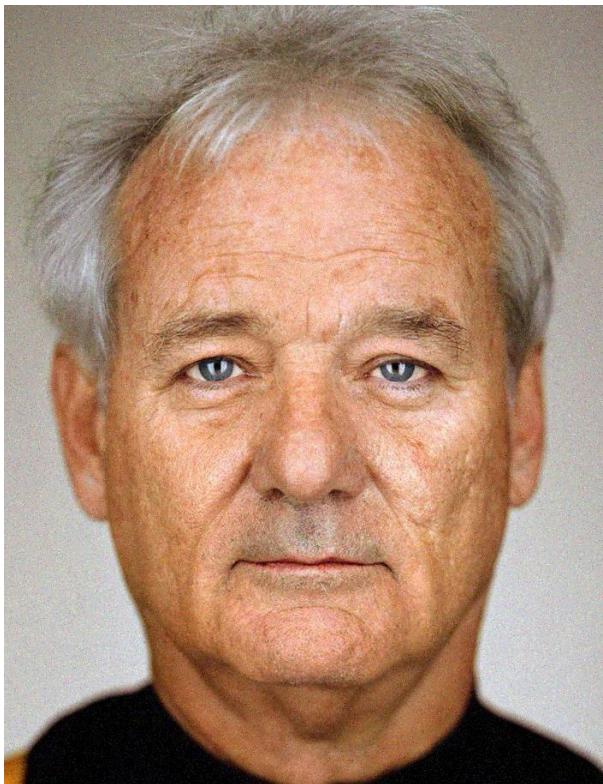


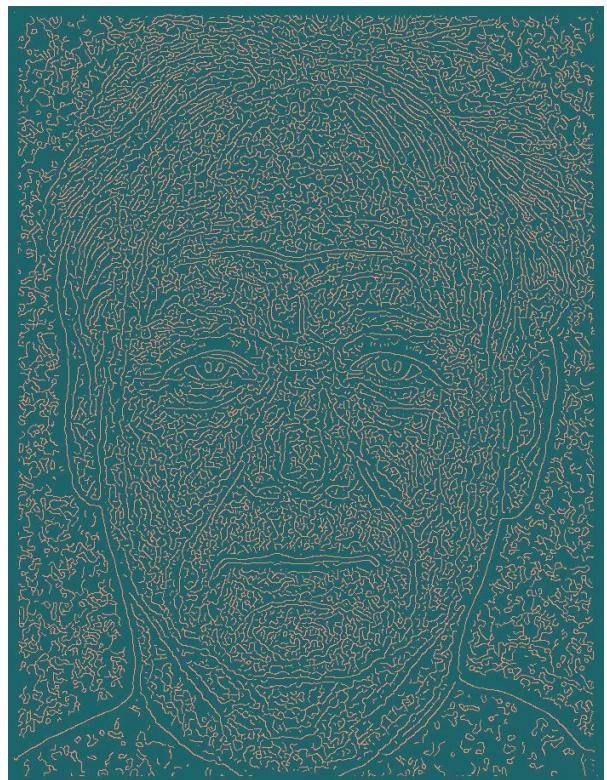
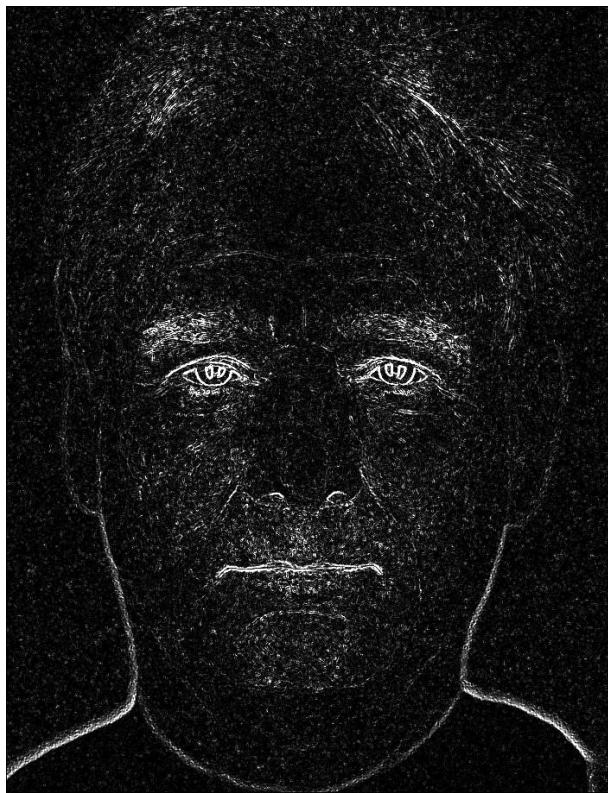
Rys. 1.2.2. Zestawienie wyników detekcji krawędzi na obrazie posiadającym zakłocenia o rozkładzie normalnym  $N(0,10)$ . Kolejność obrazów jak na Rys.1.1.1.



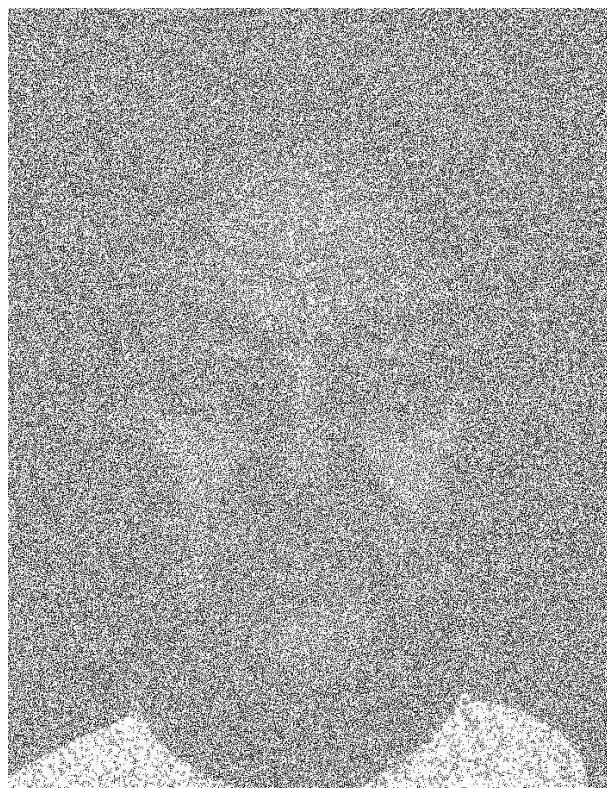
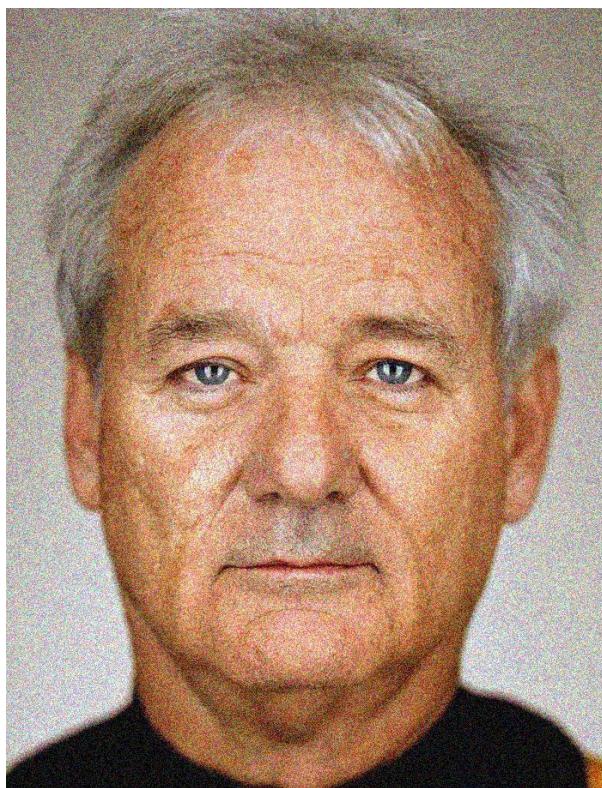


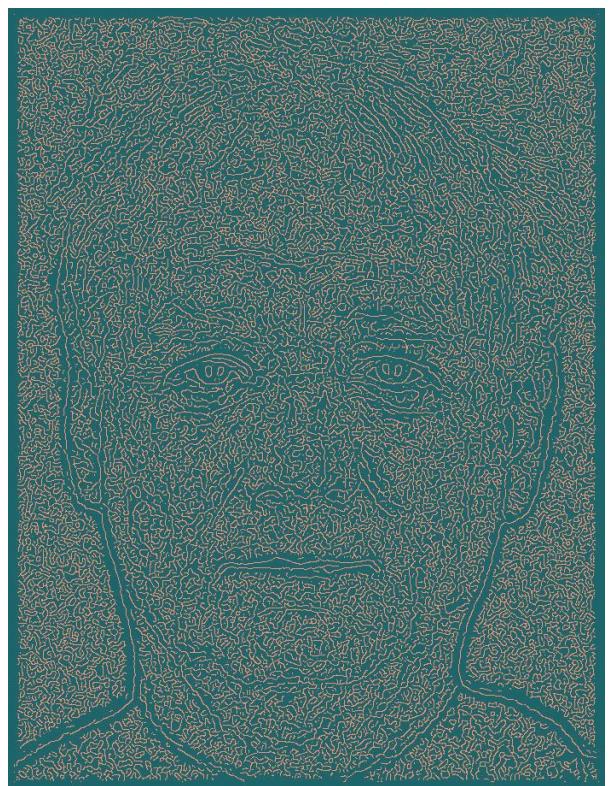
Rys. 1.1.3. Zestawienie wyników detekcji krawędzi na obrazie posiadającym zakłocenia o rozkładzie normalnym  $N(0,20)$ . Kolejność obrazów jak na Rys.1.1.1.



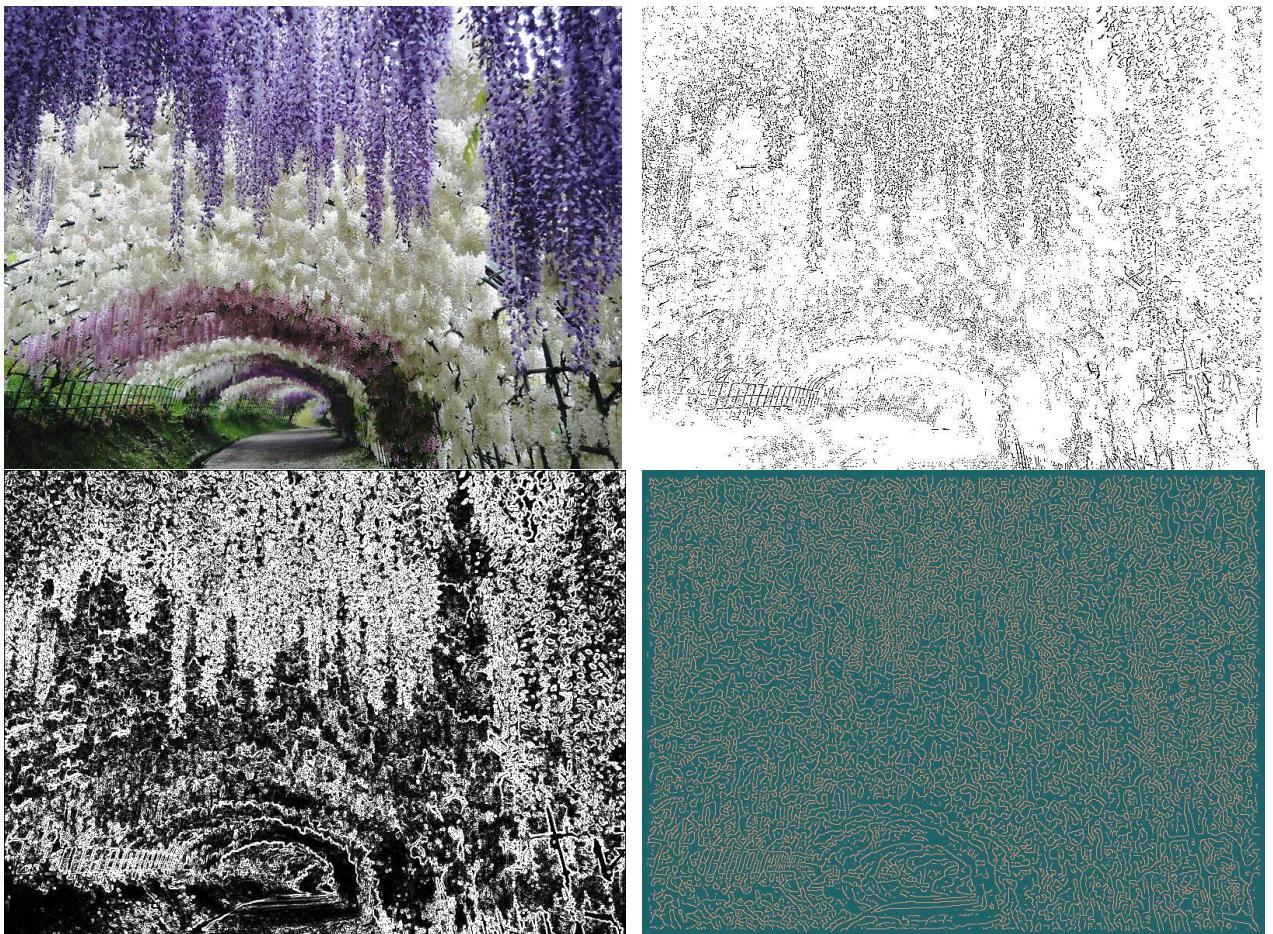


Rys.1.2.4. Zestawienie wyników detekcji krawędzi na obrazie posiadającym zakłucenia impulsowe o parametrach  $p_1=0.2$ ,  $v_1=30$ ,  $p_2=0$ ,  $v_2=0$ . Kolejność obrazów jak na Rys.1.1.1.

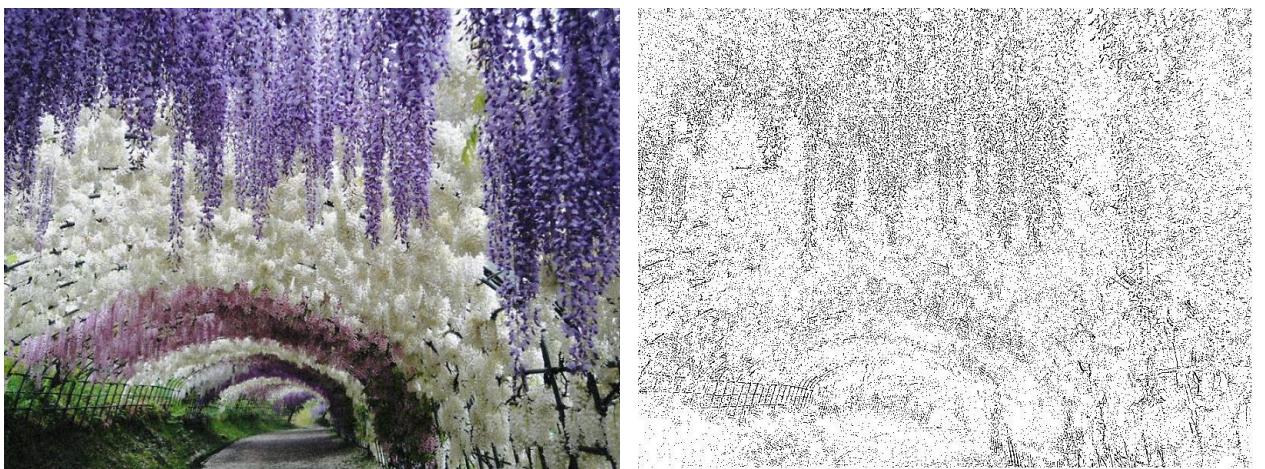


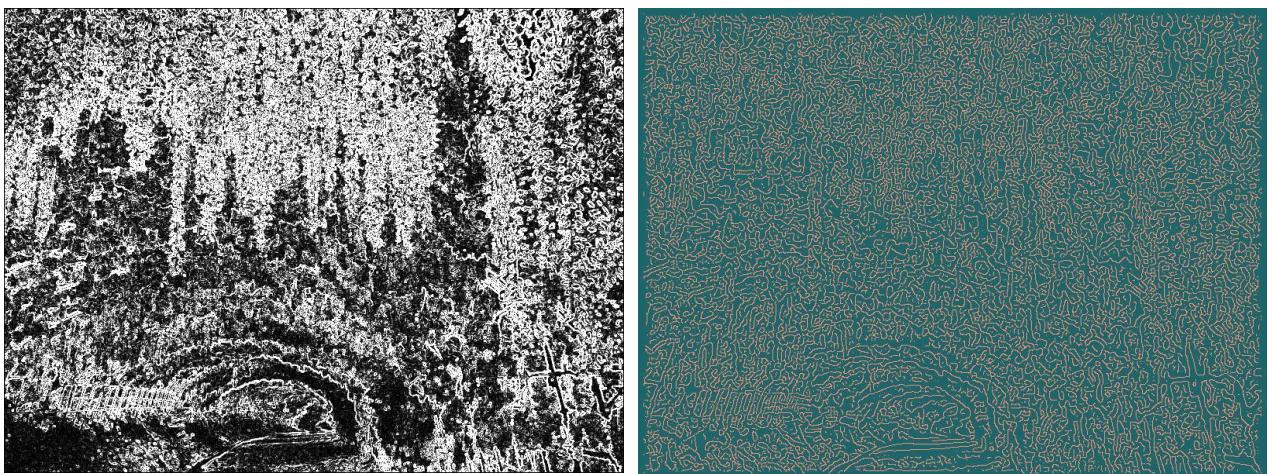


Rys.1.1.5. Zestawienie wyników detekcji krawędzi na obrazie posiadającym zakłucenia impulsowe o parametrach  $p_1=0.3$ ,  $v_1=40$ ,  $p_2=0.3$ ,  $v_2=40$ . Kolejność obrazów jak na Rys.1.1.1.

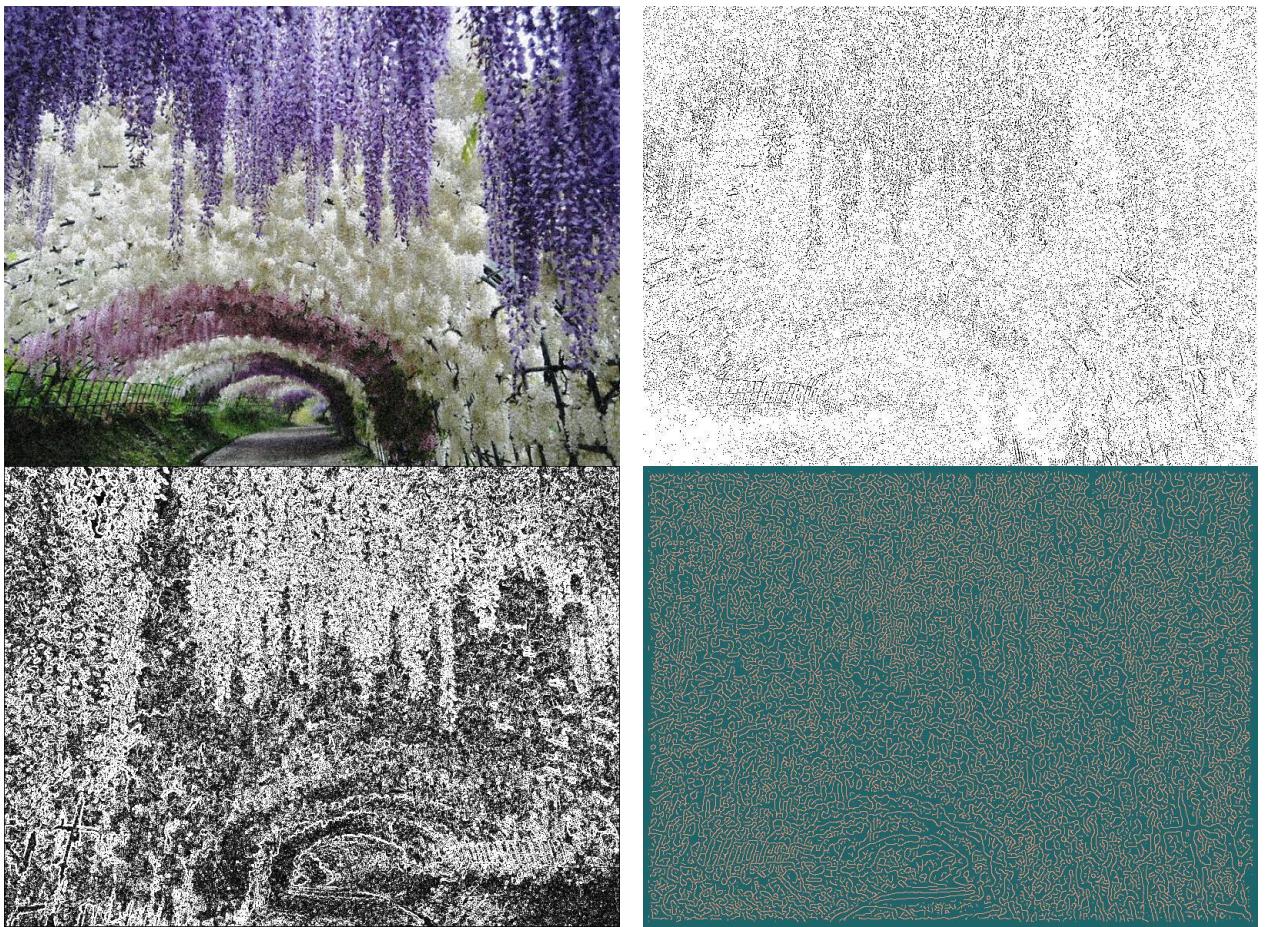


Rys.1.3.1. Zestawienie wyników detekcji krawędzi na obrazie nie posiadającym zaszumień oraz zakłóceń. Kolejność obrazów jak na Rys.1.1.1.

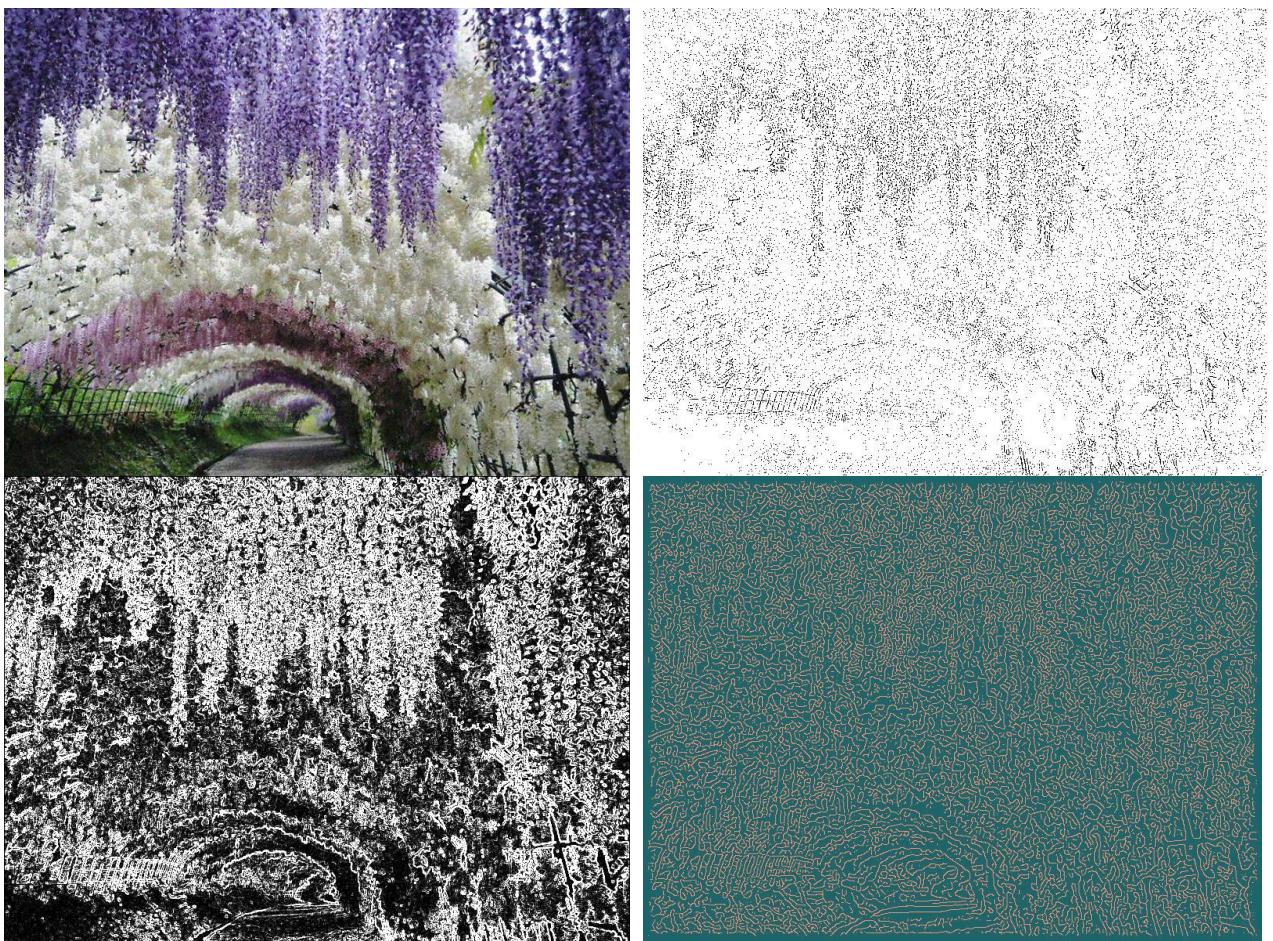




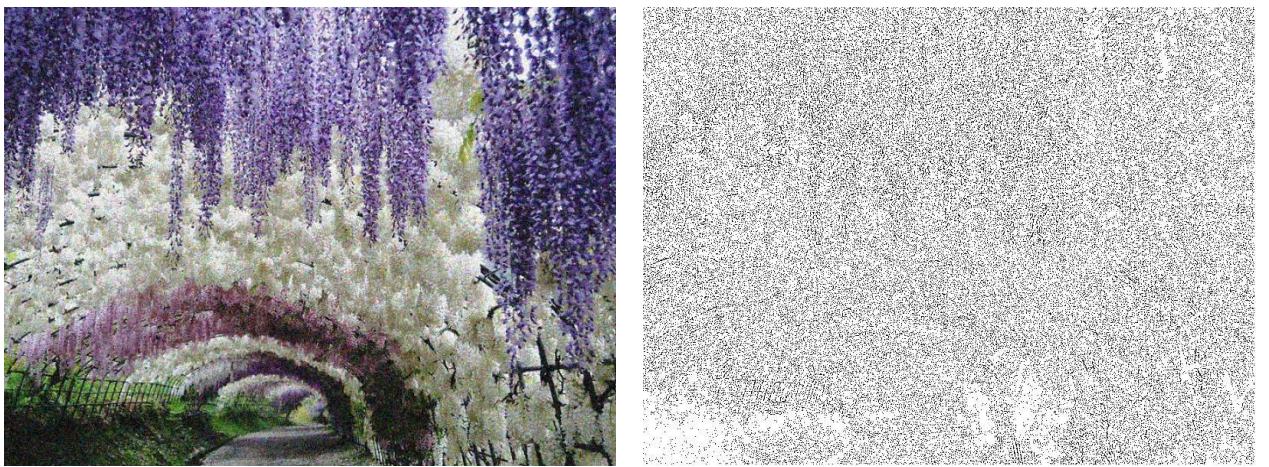
Rys. 1.3.2. Zestawienie wyników detekcji krawędzi na obrazie posiadającym zakłocenia o rozkładzie normalnym  $N(0,10)$ . Kolejność obrazów jak na Rys.1.1.1.

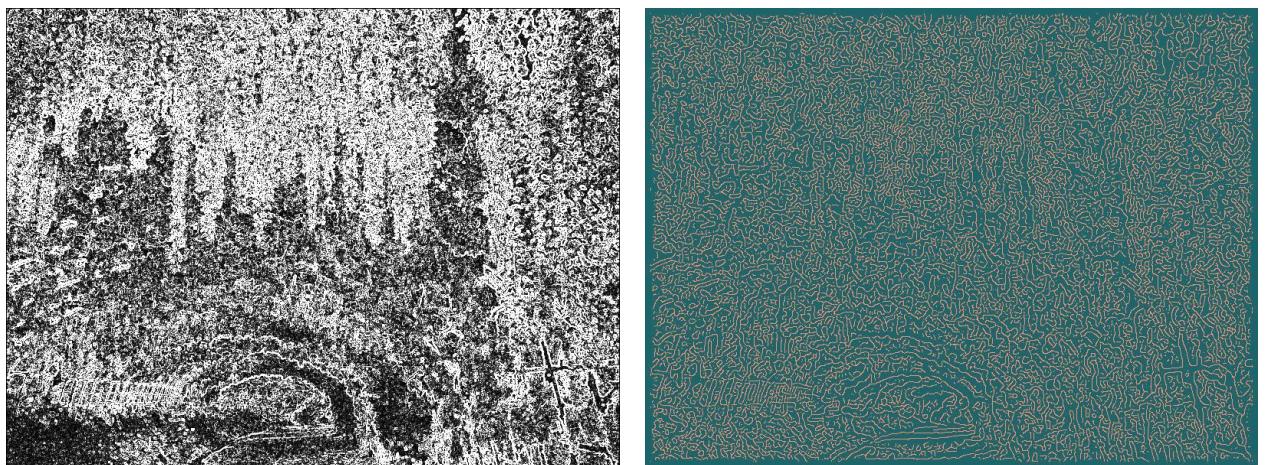


Rys. 1.3.3. Zestawienie wyników detekcji krawędzi na obrazie posiadającym zakłocenia o rozkładzie normalnym  $N(0,20)$ . Kolejność obrazów jak na Rys.1.1.1.

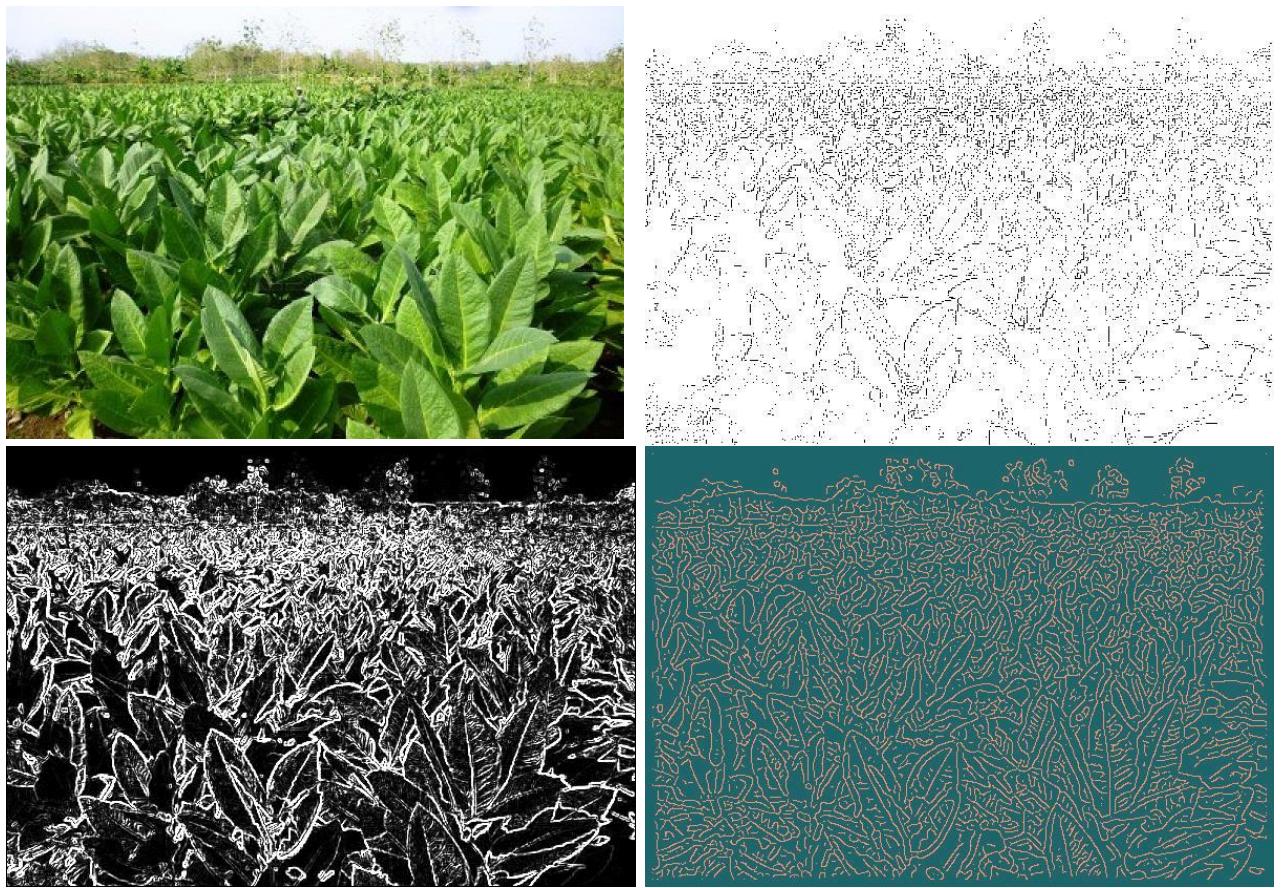


Rys.1.3.4. Zestawienie wyników detekcji krawędzi na obrazie posiadającym zakłucenia impulsowe o parametrach  $p_1=0.2$ ,  $v_1=30$ ,  $p_2=0$ ,  $v_2=0$ . Kolejność obrazów jak na Rys.1.1.1.

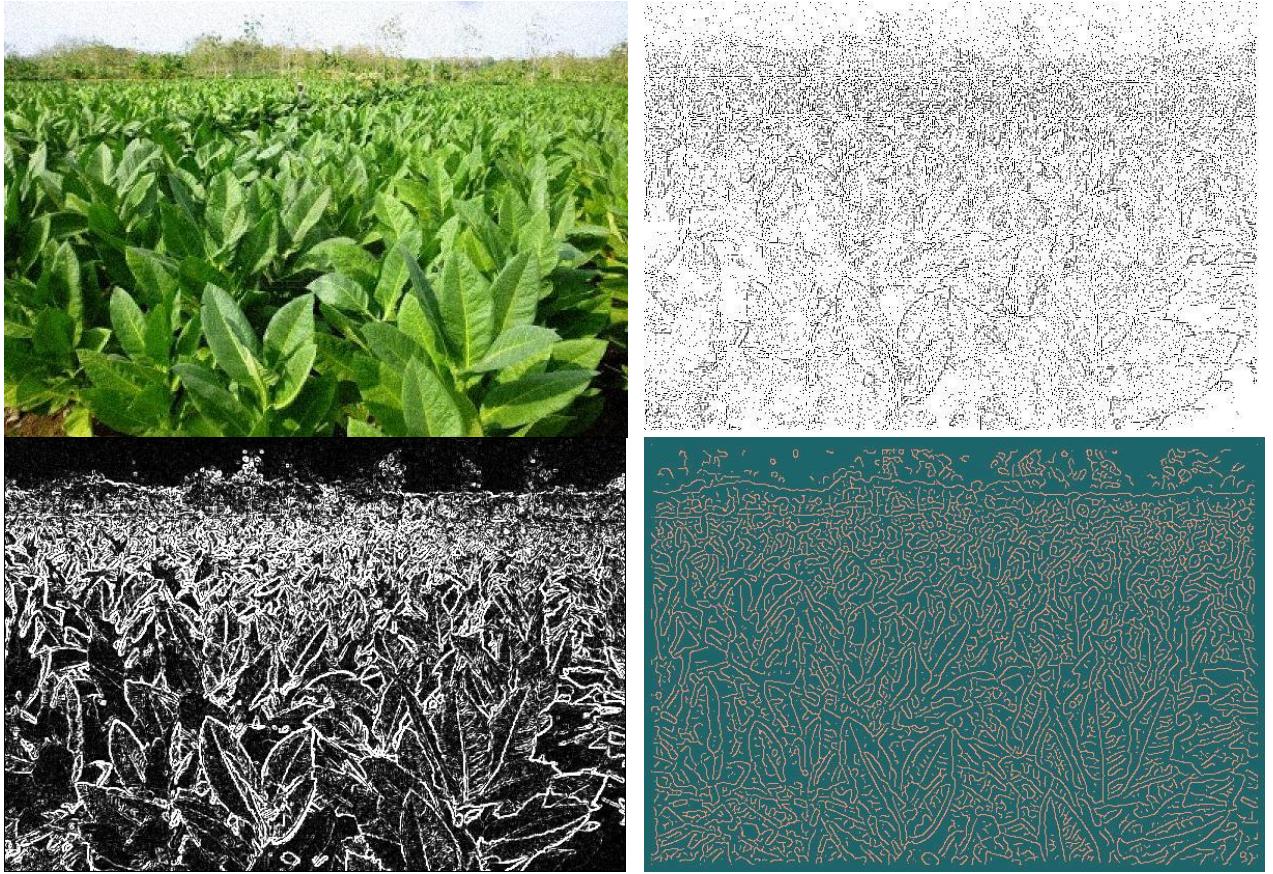




Rys. 1.3.5. Zestawienie wyników detekcji krawędzi na obrazie posiadającym zakłocenia impulsowe o parametrach  $p_1=0.3$ ,  $v_1=40$ ,  $p_2=0.3$ ,  $v_2=40$ . Kolejność obrazów jak na Rys.1.1.1.

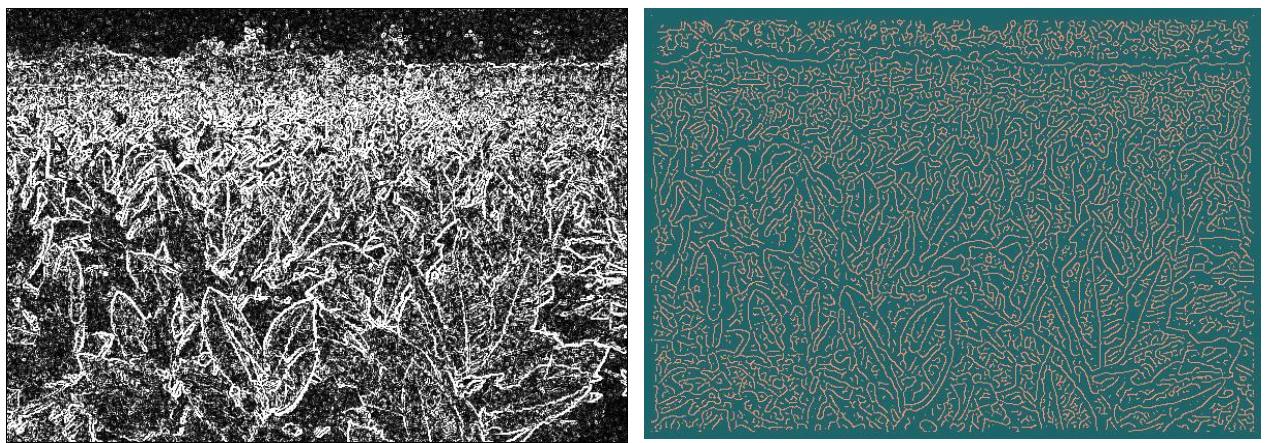


Rys.1.4.1. Zestawienie wyników detekcji krawędzi na obrazie nie posiadającym zaszumień oraz zakłóceń. Kolejność obrazów jak na Rys.1.1.1.

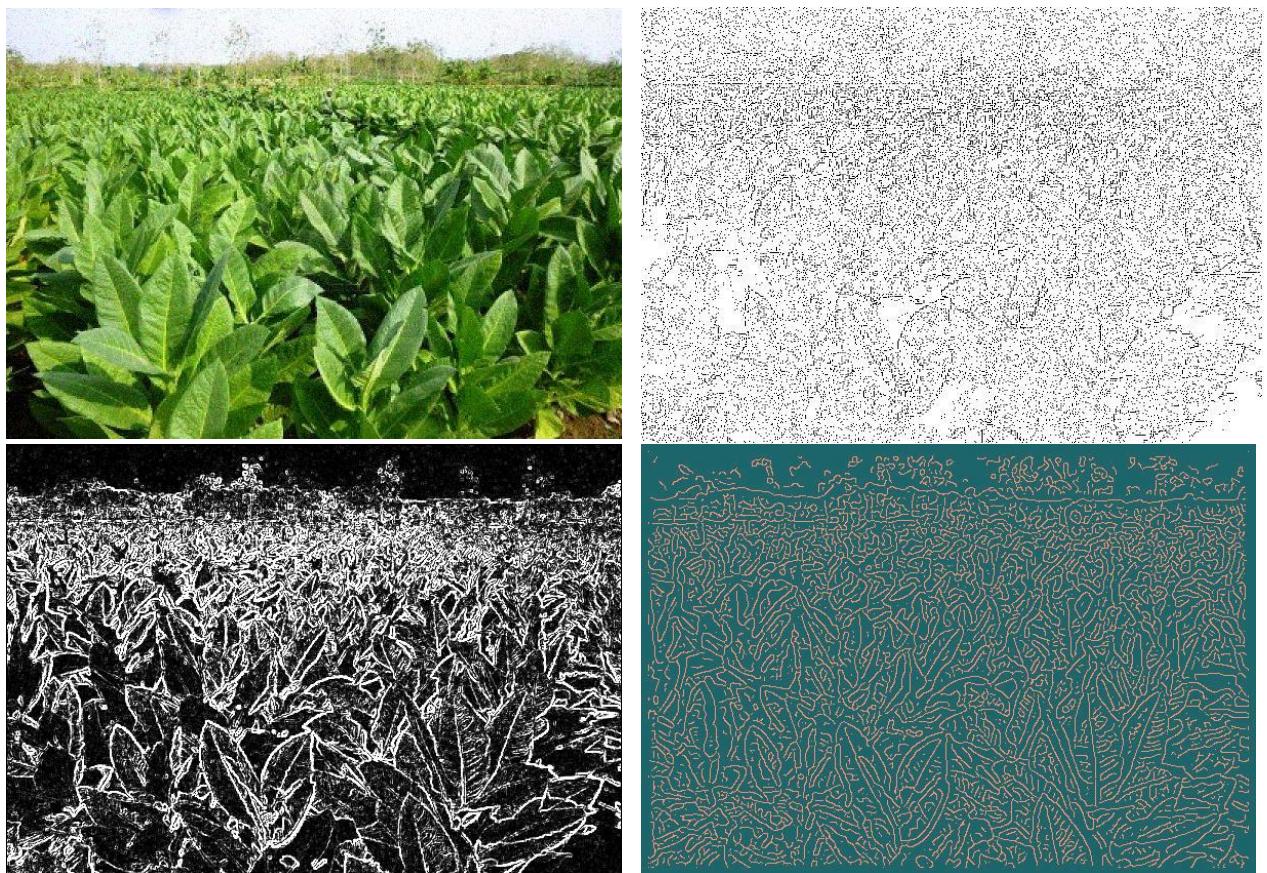


Rys. 1.4.2. Zestawienie wyników detekcji krawędzi na obrazie posiadającym zakłocenia o rozkładzie normalnym N (0,10). Kolejność obrazów jak na Rys.1.1.1.

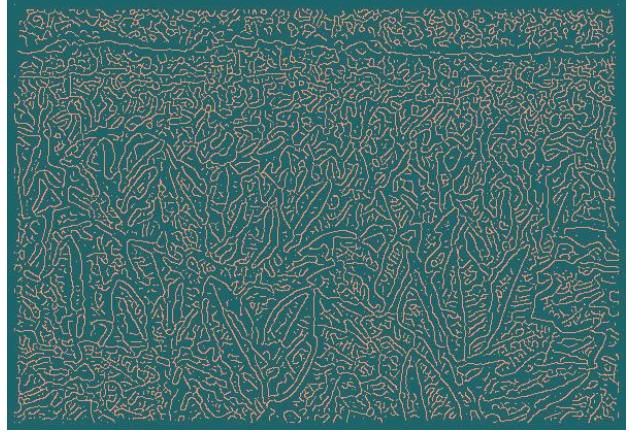
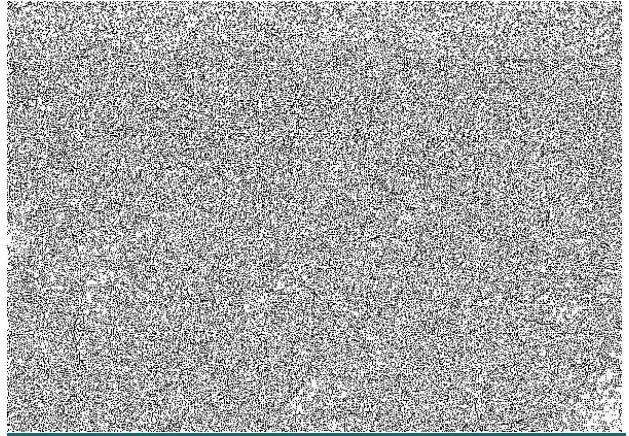




Rys. 1.4.3. Zestawienie wyników detekcji krawędzi na obrazie posiadającym zakłocenia o rozkładzie normalnym N (0,20). Kolejność obrazów jak na Rys.1.1.1.



Rys.1.4.4. Zestawienie wyników detekcji krawędzi na obrazie posiadającym zakłocenia impulsowe o parametrach p1=0.2, v1=30, p2=0, v2=0. Kolejność obrazów jak na Rys.1.1.1.



Rys.1.1.5. Zestawienie wyników detekcji krawędzi na obrazie posiadającym zakłucenia impulsowe o parametrach  $p_1=0.3$ ,  $v_1=40$ ,  $p_2=0.3$ ,  $v_2=40$ . Kolejność obrazów jak na Rys.1.1.1.

## 6 Wyniki i wnioski

Algorytm prosty oraz algorytm Sobela są algorytmami bardziej podatnymi na zakłócenia niż algorytm Canny'ego. Przy zakłócenach typu Gaussa znaczne pogorszenie dla prostego algorytmu są widoczne stosunkowo wcześnie. Występuje dużo artefaktów, czyli fragmenty obrazu, które nie są krawędziami. Taka sama obserwacja ma miejsce przy filtrze Sobela. Algorytm Canny'ego zostawia po sobie znacznie mniej artefaktów, obraz krawędzi jest bardziej czysty. Dodatkową zmianą, jakie wnoszą zakłócenia Gaussa jest znaczne pogorszenie jakości wykrytych krawędzi mniej widocznych. Te delikatne krawędzie są najlepiej rozpoznawane algorytmem Canny'ego nawet przy zakłócenach, podczas gdy pozostałe algorytmy sobie z tym nie radzą. Z kolei mocne krawędzie, bardziej wyróżniające się, są wykrywane nawet przy zakłócenach, zarówno w przypadku algorytmu Canny'ego jak i Sobela (gorzej niż w przypadku algorytmu Canny'ego, ale można zastosować odpowiedni filtr celem poprawy), natomiast prosty algorytm zwyczajnie sobie nie radzi (krawędzie są postrzępione i dookoła powstaje dużo artefaktów).

Zakłócenia impulsowe zdają się mieć mniejszy wpływ na jakość detekcji niż szum Gaussowski. Wynika to z tego, że mają one mniejszą siłę (nie są nałożone na cały obrazek, a jedynie na wybrane piksele). Podobnie jak w przypadku zakłóceń Gaussa, najlepiej spisują się algorytmy Canny'ego i Sobela. Krawędzie są dobrze rozróżnialne, a dodatkowo w przypadku algorytmu Canny'ego jest najmniej artefaktów. Niewiele artefaktów posiada także efekt działania algorytmu prostego. Jest on dość odporny na zakłócenia tego typu. Obrazy otrzymane za pomocą tego algorytmu są bardzo do siebie zbliżone. Algorytm Sobela także poprawnie rozpoznaje krawędzie, jednakże podobnie jak w zakłócenach z rozkładem Gaussa, wzmacnia artefakty i powoduje powstawanie „fałszywych” krawędzi.

Jak widać, zaawansowane metody detekcji krawędzi (zwłaszcza algorytm Canny'ego) dają najlepsze rezultaty. Obrazy z filtru Sobela, po niewielkim nakładzie pracy także dały by bardzo dobry efekt. Nieco gorzej zachowuje się prosty algorytm, jednakże jego zaletą jest prostota wykonania i szybkość obliczeń, co może być także przesłanką do jego stosowania.

## 7 Skład zespołu

- **Piotr Jastrzębski** – prowadzenie korespondencji z prowadzącym, koordynacja prac, podział ról w zespole, zarządzanie procesem
- **Tomasz Adamczyk, Piotr Chmiel, Stanislav Tymkiv** – opracowanie i implementacja prostego algorytmu autorskiego
- **Paweł Przybysz, Sylwester Tor, Eryk Solecki** – wdrożenie algorytmów zaawansowanych

- **Zuzanna Jarosz, Małgorzata Bosowska** – analiza artykułów dotyczących wybranych algorytmów, stworzenie streszczenia na podstawie kilku artykułów opisujących działanie algorytmu
- **Mariana Ruzhytska** – zaszumienie obrazów, przeprowadzenie testów

## 8 Załączniki

1. Artykuł nr 1. – plik *Artykuly//Artykuł1\_Algorytm\_Canny'ego.pdf*
2. Artykuł nr 2. – plik *Artykuly//Artykuł2\_Analiza\_obrazów\_i\_modelowanie\_wirtualne\_w\_konstruowaniu\_protez\_kości\_czaszki*
3. Kod źródłowy i plik wykonywalny implementacji prostego algorytmu wraz z wdrożeniem zaawansowanych algorytmów (razem) – katalog *Aplikacje/WykrywanieKrawędzi*
4. Kod źródłowy i plik wykonywalny programu zaszumiania obrazów – katalog *Aplikacje/Zaszumianie*

## Literatura

- [1] IEEE Xplore. Dostępne w Internecie: <http://ieeexplore.ieee.org>, data dostępu: 29.05.2013.
- [2] OpenCV. Dostępne w Internecie: <http://opencv.willowgarage.com/wiki/>, data dostępu: 29.05.2013.
- [3] C. Bołdak. Cyfrowe przetwarzanie obrazów. Dostępne w Internecie: <http://aragorn.pb.bialystok.pl/~boldak/DIP/Wyklady.html>, data dostępu: 29.05.2013.
- [4] Korohoda P. Tadeusiewicz R. *Komputerowa analiza i przetwarzanie obrazów*. Wydawnictwo Fundacji Postępu Telekomunikacji, 1997.