Telekomunikacja

SERF - Self-embedding fragile watermarking based on DCT and fast fractal coding.

Skład zespołu: **Dominik Rosiek, Piotr Ścibor.**

Cel

Celem projektu jest implementacja algorytmu do samo-rekonstrukcji obrazów cyfrowych z wykorzystaniem kodowania fraktalnego. Samo-rekonstrukcja (ang. self-recovery lub self-embedding) pozwala na weryfikację integralności zdjęcia oraz na odtworzenie jego oryginalnej treści w oparciu o cyfrowy znak wodny. Opracowaną w ramach projektu aplikację należy wyposażyć w graficzny interfejs użytkownika pozwalający na dostosowanie parametrów algorytmu oraz na zabezpieczanie i weryfikację integralności obrazów cyfrowych.

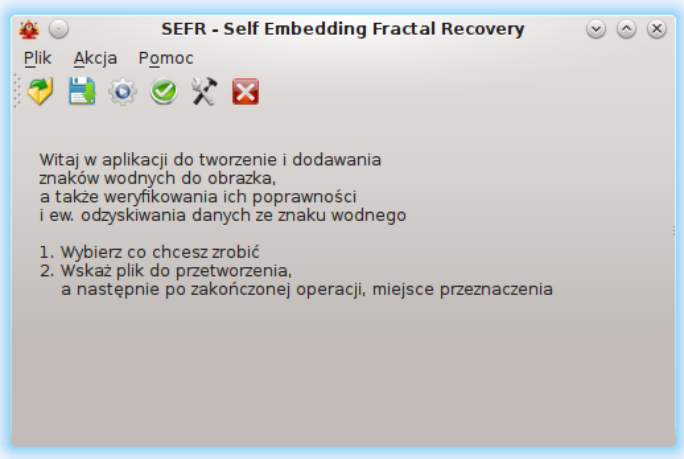
Realizacja

Tworząc naszą aplikację opieraliśmy się na wiedzy i informacjach zawartych w artykule : ***„Zhang, Xuanping, Yangyang Xiao, and Zhongmeng Zhao. "Self-embedding fragile watermarking based on DCT and fast fractal coding." Multimedia Tools and Applications, doi: 10.1007/s11042-014-1882-9”.*** Zaczęliśmy od przeczytania powyższego artykułu, przetłumaczenia niezrozumiałych dla nas fragmentów oraz od wspólnego omówienia planu. Po zapoznaniu się z artykułem podzieliliśmy się obowiązkami i zaczęliśmy pisać silnik aplikacji. Wybraliśmy język programowania python.

Śledząc krok po kroku informacje zawarte w artykule, wzięliśmy się za pisanie odpowiednich funkcji, klas i całego kodu. Dokumentacja naszych funkcji i klas znajduje się w dołączonych plikach (funkcje.html, main.html, gui.html).

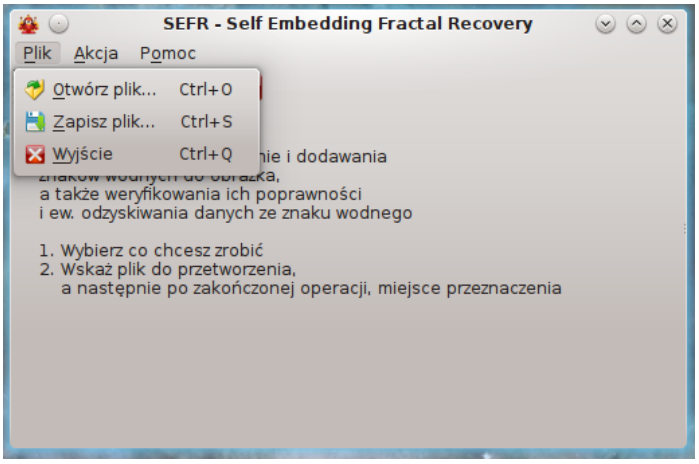
Po wstępnym napisaniu kodu zaczęliśmy, jeszcze konsolowo, testować działanie naszej aplikacji. Mieliśmy problemy z jakością uzyskanej rekonstrukcji. Problem ten wymagał od nas ponownego przejrzenia artykułu i poprawienia kodu. Zadanie to jednak okazało się trudniejsze niż zakładaliśmy. Poprawienie jakości rekonstrukcji uszkodzonego obrazu udało nam się w niewielkim stopniu. Wciąż jakość ta znacząco odbiegała od tej zaprezentowanej w artykule. Nie potrafiliśmy znaleźć rozwiązania problemu.

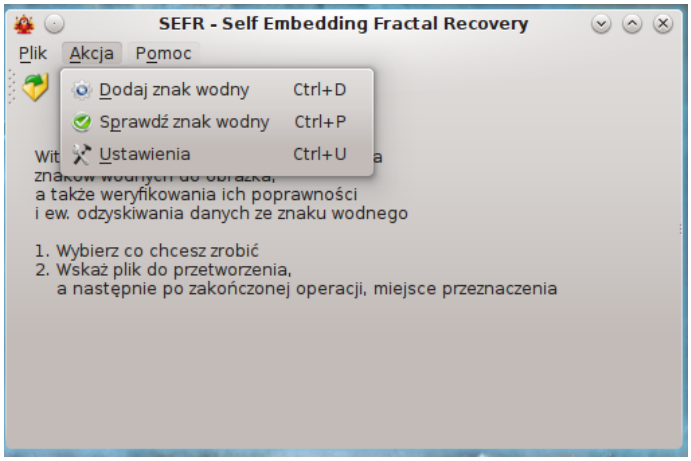
Zostawiając powyższą kwestię otwartą przeszliśmy do tworzenia graficznego interfejsu użytkownika. Zadanie to było czasochłonne jednak nie sprawiało nam większych problemów. Postawiliśmy na prostą strukturę okna, bez niepotrzebnych udziwnień. Po skończeniu całość wyglądała następująco:



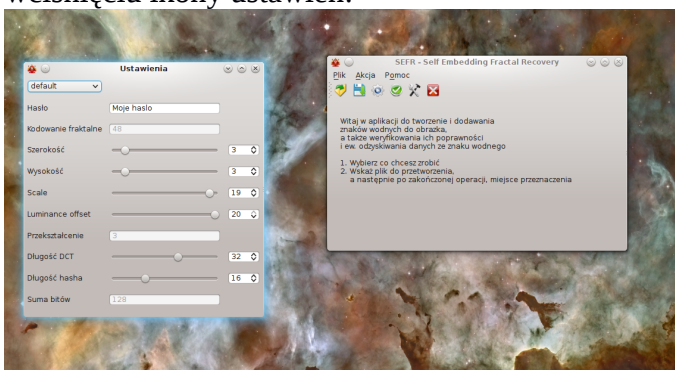
Wyposażyliśmy naszą aplikację w przyciski szybkiego dostępu ułatwiające mi. załadowanie obrazu do aplikacji, zapisanie zakodowanego obrazu ze znakiem wodnym, zamknięcie aplikacji i inne.

Wszystkie te funkcje są oczywiście dostępne również z poziomu menu (Plik i Akcja).





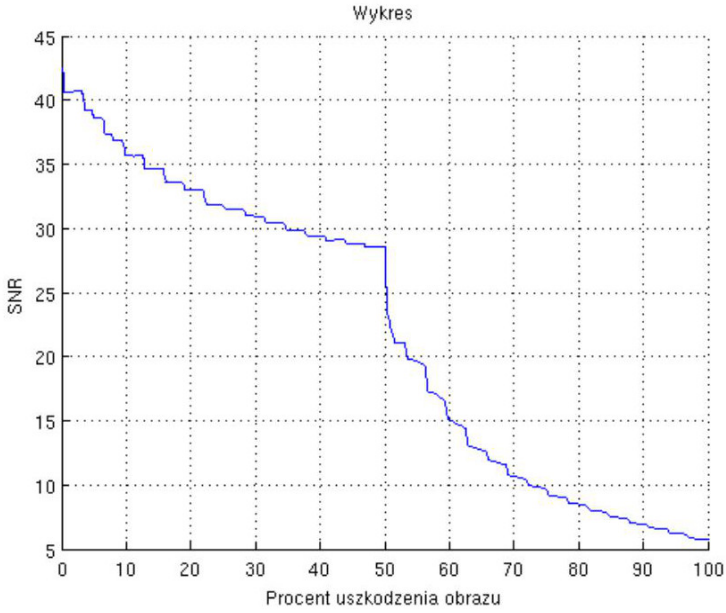
Następnie w celu ulepszenia naszego projektu wyposażyliśmy aplikację w dodatkową funkcję ustawień. Dzięki temu możliwe jest samodzielne ustawianie i rozporządzanie przeznaczeniem bitów używanych do kodowania znaku wodnego. Dodaliśmy dodatkowe okno ustawień dostępne po wciśnięciu ikony ustawień:



Po zakończeniu pisania graficznego interfejsu i po konsultacji z Panem dr inż. Pawłem Korusem wróciliśmy do problemu jakości rekonstrukcji obrazu. Po zastosowaniu się do wskazówek Pana doktora udało nam się poprawić jakość pracy naszej aplikacji.

Testy

Poniżej przedstawiamy wykres pokazujący jakość rekonstrukcji obrazu w zależności od poziomu zniszczeń. Użyte ustawienia to po 4 bity na x i y, 10 na offset, 11 na jasność oraz 40 dla DCT. Jak widać przy uszkodzeniach obrazu powyżej 50% drastycznie spada. Powodem tego jest sposób uszkadzania obrazu. Bity obrazu były zerowane od lewej do prawej, więc bardzo szybko były uszkadzane bloki bazowe potrzebne do rekonstrukcji fraktalnej.



Przyszłość

Nasz projekt można rozwijać. Rozkład współczynników można zoptymalizować za pomocą algorytmu Lloyda-Maxa. Niestety nie udało się nam tego zrealizować w wyznaczonym czasie. Kolejnym elementem jest wykonanie dokładniejszych testów i udokumentowanie ich. Cały program można również wzbogacić o watermarking obrazów kolorowych. Wystarczy potraktować każdy zestaw barw (R, G, B) jako paletę kolorów czarnobiałych.