# Automatyczne układanie puzzli na podstawie zdjęć

**Alicja Figas, Marcin Gruza**
Wydział Informatyki i Zarządzania
Politechnika Wrocławska
Wyb. Wyspiańskiego 27, 50-370 Wrocław
{238442, 256875}@student.pwr.edu.pl

## 1 Opis problemu

W ramach projektu postanowiliśmy zająć się automatycznym układaniem puzzli o nieregularnym kształcie rzeczywistych puzzli z wypustkami. Skorzystać przy tym chcemy z klasycznych metod, aby poznać ich mechanikę działania, aniżeli gotowych już modeli sieci neuronowych, które to można wykorzystać w podobnych zadaniach rekostrukcji.

Jest to złożony problem widzenia maszynowego ze względu na kilka podproblemów, które się na niego składają zaczynając od segmentacji, porównywaniu do siebie krawędzi części obrazów i składanie obrazu w całość za pomocą algorytmu zachłannego.

Na podstawie dostępnych źródeł literaturowych jest to również problem niewyeksplorowany, szczególnie dla obrazów rzeczywistych.

Celem programu jest stworzenie działającego programu, który umożliwi wczytanie pliku ze zdjęciem puzzli, a następnie przedstawienie rozwiązania w postaci ułożonych już puzzli.

Sam problem należy do dziedziny problemów segmentacji obrazu, a następnie

## 2 Przyjęte założenia i ograniczenia

Należało przyjąć następujące założenia:

- **Kompletność puzzli** - ze względu na algorytm układania, który jest algorytmem z nawrotami przyjęliśmy założenie o kompletności puzzli na wczytywanym obrazie, co oznacza że używany obraz powinien zawierać wszystkie kawałki potrzebne do ułożenia kompletnego rozwiązania oraz tylko takie, które z tego obrazu pochodzą.
- **Mała ilość kawałków** - wczytywania obrazów, jak i późniejsze przetwarzanie wymaga dobrej jakości zdjęć, inaczej poprawna segmentacja, jak i późniejsze układanie. Jest to również związane z mocą obliczeniową.
- **Jednorodne tło** Zadanie segmentacji jest mocno utrudnione, kiedy tło jest niejednorodne, co utrudnia ekstrację krawędzi.
- **Puzzle wygenerowane komputerowo lub rzeczywiste zdjęcia puzzli** ostatnim poruszonym warunkiem działania naszego programu są dane - zdjęcia puzzli posiadających wypustki.
- **Nie uwzględniamy perspektywy** Problem perspektywy przy rozpoznawaniu obrazów i transformacji, tak żeby zachować parametry obecnego obrazu umożliwiając ułożenie kompletnego rozwiązania samo w sobie jest złożonym problemem i automatyczne rozpoznawanie perspektywy i transformacja były na tyle dużym zadaniem, że na potrzeby rozwiązania na ten kurs zdecydowaliśmy się go pominąć.

- **Puzzle nie mogą nachodzić na siebie** Ważne, aby na wykonanym zdjęciu puzzle były oddzielone od siebie. Ich połączenie niemożność poprawnego rozpoznania puzzli, a następnie jego krawędzi, a więc uniemożliwia stworzenie rozwiązania.

## 3 Wykorzystane komponenty obce i ich użyta funkcjonalność

Do stworzenia programu wykorzystano język programowania Python wraz z implementacją biblioteki OpenCV[1] dla tego języka. Biblioteka ta wspiera zadania widzenia komputerowego umożliwiając wczytywanie obrazów, ich reprezentację za pomocą macierzy złożonych oraz szereg implementacji klasycznych algorytmów z których będziemy korzystać i które umożliwiają nam rozwiązanie przedstawionego problemu.

Sposób w jaki moduły są wykorzystywane jest opisane w dalszych rozdziałach z opisem metodologii.

## 4 Architektura zaimplementowanego programu

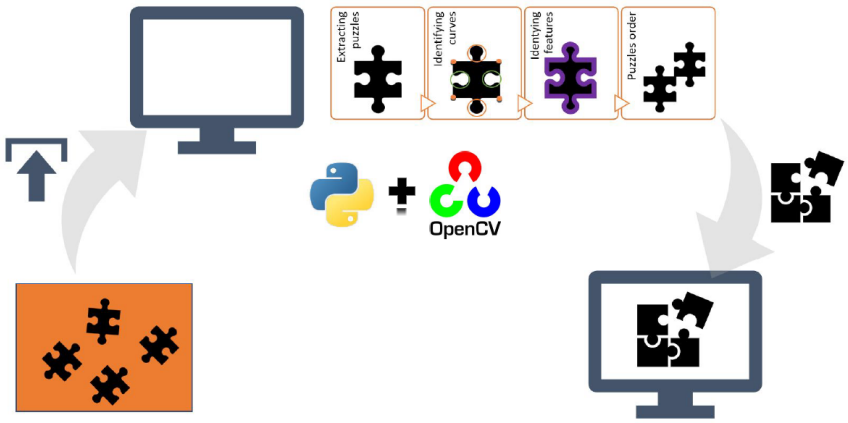
Rysunek 1 przedstawia schemat naszego rozwiązania - jest to program komputerowy, który umożliwia po wczytaniu obrazu i przeprocesowaniu zwraca propozycję ułożonego obrazu.

Implementacja składa się z 4 głównych komponentów, są one podzielone na 4 części pod kątem logicznym, a funcjonalnie na dwie. Pierwszy z nich pozwala na wyciągnięcie kawałków puzzli z obrazkaz za pomocą odpowiednich transformacji, a następnie stworzenie maski zawierającej kontury puzzli, aby umożliwić ich dalsze procesowanie.

Drugi komponent umożliwia rozpoznanie już dla każdego puzzla z osobno jego krawędzi - sprawdzamy, jakie kawałek puzzla ma krawędzie - wklęsłe, wypukłe lub też proste, jeśli pochodzą z krawędzi obrazka. Poza kształtem krawędzi rozpoznawane są również same rogi puzzla. W ten sposób otrzymujemy informacje o położeniu i typie krawędzi, co stanowi wejście do kolejnego kompotentu, gdzie na postawie zadanych danych znajdujemy reprezentacje wektorową danej krawędzi.

Po znalezieniu tej miary dla każdej z krawędzi w ostatnim komponencie programu następuje próba ułożenia obrazu na podstawie różnych miar podobieństw opisanym w rozdziałach poniżej przy opisie metodologii naszego rozwiązania, aż do uzyskania finalnego rozwiązania.

Tak znalezione rozwiązanie jest prezentowane użytkownikowi. Program jest możliwy do uruchomienia na standardowym komputerze osobistym, nie wymaga dodatkowego sprzętu, a użyte oprogramowanie jest ogólnodostępne - cała implementacja jest stworzona w języku programowania Python[3] za pomocą Jupyter Notebook[2] oraz wykorzystuje bibliotekę OpenCV[1].




Rysunek 1: Architektura programu

## 5 Wkład w dziedzinę

Dziedzina rozpoznawania elementów obrazu, a następnie jego układania ma zastosowanie w kilku praktycznych dziedzinach między innymi wspieraniu procesów przemysłowych, tak aby automatyzować pracę np. fabrykach, aby maszyny samodzielnie mogły dopasować elementy przy składaniu nie wymagając ustawiania bardzo precyzyjnych parametrów, gdzie każda zmiana może prowadzić do złego łączenia elementów, co niesie uszkodzenie elementów i powoduje koszty.

Innym z zastosowań tej dziedziny problemu jest wsparcie dla osób niewidomych i pomoc w ich codziennym życiu.

My postanowiliśmy się zająć aspektem nieco mniej praktycznym Zaczynając pracę nad projektem zaczęliśmy od krótkiego przeglądu dostępnych rozwiązań.

Sam problem rozwiązywanie czy też uzupełnienia obrazu na podstawie modeli generatywnych, który umożliwia rekonstrukcję obrazu jest obecny. Jednak samo zagadnienie układania puzzli jest mniej bogate w źródła. Większość dostępnych artykułów naukowych skupia się na puzzlach kwadratowych, gdzie zadanie jest proste i nie wymaga segmentacji obrazu - puzzle mają określoną długość i szerokość i są generowane komputerowo [4].

[6] prezentuje podejście do układania puzzli o różnych kształtach. Podobnie jak my stosuje porównywanie krawędzi za pomocą kształtu i koloru. Autorzy używają w niej innych miar podobieństwa kształtów (np. odległości pomiędzy narożnikiem a każdym innym punktem krawędzi oraz szerokości główek) oraz punktowego porównania kolorystycznego. Ich podejście dało jednak prawidłowe ułożenie jedynie dla małych zestawów puzzli (6-24 elementy). [7] prezentuje podejście umożliwiające ułożenie nawet tysiąca puzzli, jednak autorzy uprościli problem poprzez skanowanie puzzli zamiast robienia im zdjęć, co ułatwiło problem ektrakcji cech.

Naszym wkładem w dziedzinę jest więc poszerzenie dostępnych rozwiązań o metodyki rozpoznawania i układania puzzli o nieregularnym kształcie rzeczywistych puzzli, a także miary jakości rozwiązania puzzli.

Dodatkowo podjęliśmy próbę rozwiązania zadania dla rzeczywistych obrazów, a nie jedynie komputerowych.

## 6 Segmentacja puzzli

Pierwszym krokiem działania naszego systemu była segmentacja puzzli - z obrazu należało wyekstrahować kawałki puzzli, tak aby możnabyło założyć je w kompletną całość i przedstawić rozwiązanie.

Zadanie to jest o tyle nietrywialne, że puzzle mają różne kształty, jak i wielkość zdjęcia może być różna.

### 6.1 Ekstracja wszystkich krawędzi

Aby dokonać segmentacji puzzli należało rozpoznać ich krawędzie na tle tła. W tym celu zdecydowaliśmy się użyć filtra Canny'ego [5].Działanie algorytmu składanie się z kilku kroków. Najpierw z obrazu usuwany jest szum. Następnie dla obrazu wyliczany jest gradient, na którego podstawie szukane są lokalne maksima obrazu, które następnie są wygaszane (ich wartość jest zmieniana na zero). Na końcu następuję ograniczenie ze względu na histerezę, jest to krok w którym sprawdzane jest, co rzeczywiście jest krawędzią, a co nie. W ten sposób uzyskujemy wszystkie krawędzie obecne na obrazie.

Potencjalnym problemem tego rozwiązania jest to, że krawędzie mogą nie łączyć się ze sobą, tak więc będziemy mieć kilka krawędzi zewnętrznym oraz dużo krawędzi wewnątrz puzzla. Jak również obecność krawędzi w tle w puzzlach rzeczywistych. Te problemy rozwiązują dwa kolejne kroki.

### 6.2 Domykanie kształtów

Użycie filtra Canny'ego miało za zadanie wyszczególnić wszystkie krawędzie ze zdjęcia, tak, aby później na ich podstawie wybrać kontur puzzla. Zadanie to jednak jest uciążliwe gdyż możliwym jest,

że znaleziony kształt jest niezamknięty, co uniemożliwi dalsze znajdowanie kompletnego kształtu puzzli.

W tym celu zastosowano dwie transformacje morfologiczne: erozji oraz dylatacji, które umożliwiają poprawne dalsze przetwarzanie obrazu, dzięki temu, że przez ich zastosowanie domykamy kształty puzzli.

### 6.3 Znajdowanie konturów

Ostatni element algorytmu jest już ekstrakcją właściwą kawałków. Mając obraz dla którego mamy zestaw krawędzi obrazu za pomocą algorytmu binarnego Satoshi Suzuki'iego dostępnego w OpenCV jako *findCountour*. Algorytm ten umożliwia hierarchiczność krawędzi, tak więc wszystkie krawędzi wykryte algorytmem Canny'ego po domknięciu operacjami morfoloficznymi znajdujące się wewnątrz kawałków puzzli zostają oznaczone w hierarchii jako krawędzie wewnętrzne. Dzięki otrzymanemu wynikowi możemy wyfiltrować ze zbioru wszystkie zewnętrzne krawędzie.

Dodatkowo na końcu pozbywamy się krawędzi zewnętrznych, które nie były puzzlami tj. krawędź stołu, małe odbicia światła, które odfiltrowywujemy na podstawie założenia z progu z dołu wielkości konturu. W przypadku krawędzi stołu jedna współrzędna będzie bliska zeru, więc algorytm ją odrzuci, w przypadku odbić światła, obie krawędzie będą na tyle niewielkie, że algorytm je odrzuci.

Na podstawie znalezionych krawędzi tworzymy maskę binarną puzzli, która może byc przekazana do dalszego procesowania.

## 7 Preprocessing puzzli

W celu skutecznego przeprowadzenia algorytmu układania puzzli konieczne było ich odpowiednie przetworzenie i rozpoznanie ich elementów charakterystycznych:

- **Kontur** - dokładna linia wyznaczająca całą zewnętrzną krawędź puzzla.
- **Punkty wklęsłe** - punkty najbardziej oddalone od otoczki wypukłej.
- **Narożniki** - punkty oddzielające boczne krawędzie puzzla
- **Główki** - okrągłe wypukłe fragmenty puzzla
- **Rodzaje krawędzi** - prosta/z główką/z otworem

W tej sekcji zostaną opisane metody rozpoznawania poszczególnych elementów puzzla.

### 7.1 Kontur

Odnalezienie konturu było stosunkowo prostym zadaniem. Wystarczyło wyznaczenie punktów otaczających wcześniej wysegmentowany puzzel. W przypadku realistycznych zdjęć puzzli koniecznie jest dodatkowe wygładzenie oraz erozja krawędzi puzzla, ze względu na obecność cieni i pozostałości tła przy krawędziach.

### 7.2 Punkty wklęsłe

W kolejnym etapie wyznaczane były punkty wklęsłe na krawędzi puzzla. Są to konkretnie punkty na samym środku otworu puzzla oraz punkty w których rozpoczyna się główka puzzla. Punkty te były odnajdywane poprzez wyznaczenie punktów najbardziej oddalonych od otoczki wypukłej wyznaczonej z konturu puzzla. Minimalna odległość takiego punktu od otoczki jest parametrem algorytmu zależnym od rozdzielczości zdjęcia i stosunku wielkości puzzla do zdjęcia.

### 7.3 Narożniki

Wyznaczenie narożników jest konieczne w celu wyznaczenia granic krawędzi oraz punktów złączeń pomiędzy kolejnymi puzzlami. W celu ich odnalezienia zastosowano algorytm detekcji narożników Harrisa z wysoką czułością (parametr $k$), dzięki czemu nawet nieco zaokrąglone narożniki puzzli zostały prawidłowo wyznaczone. Oczywiście zwiększenie czułości detektora powoduje wiele fałszywych wskazań narożników, np. na końcówkach główek lub w miejscach delikatnych załamań

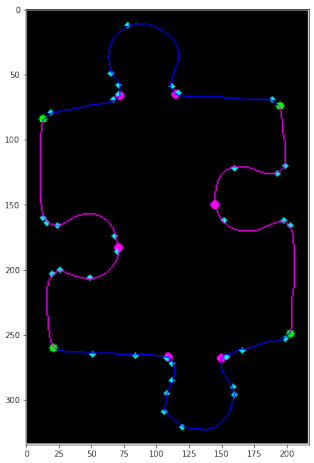
krawędzi wynikających z zaszumień zdjęcia. Aby odfiltrować nieprawidłowe wskazania, jako narożniki puzzla wybieraliśmy te 4 punkty, które tworzyły czworobok o największym polu powierzchni.

### 7.4 Główki

Opisana wyżej metoda detekcji narożników może działać nieprawidłowo w przypadku puzzli z mocno wysuniętymi główkami. Powodują one często, że pole powierzchni czworoboku wyznaczonego przez punkty z wysuniętej główki jest większe niż czworoboku wyznaczonego przez prawidłowe narożniki. Z tego powodu konieczne było wyznaczenie fragmentów konturu puzzla, które są główkami i zignorowanie potencjalnych narożników wyznaczonych na nich. W celu wyznaczenia fragmentów główek użyto miary *roundness* określonej jako:

$$roundness = \frac{P}{O^2}$$

gdzie $P$ to pole powierzchni, a $O$ to obwód figury. Dla figur zbliżonych do okręgu miara ta przyjmuje wartości $>\approx 0.07$. W naszym algorytmie sprawdzaliśmu miarę $roundness$ dla figur wyznaczonych przez otoczkę wypukłą punktów leżących na konturze puzzla pomiędzy dwoma kolejnymi punktami wklęsłymi puzzla. Jeśli miara ta przyjmuje odpowiednio dużą wartość to oznacza to, że dany fragment puzzla jest główką.




Rysunek 2: Wizualizacja preprocessingu przykładowego puzzla. Punkty zielone przedstawiają rozpoznane narożniki, niebieskie punkty to narożniki potencjalne, różowe punkty to punkty wklęsłe. Kolory krawędzi reprezentują jej typ.

### 7.5 Rodzaje krawędzi

Ostatnim etapem rozpoznawania puzzla było wyznaczenie rodzaju krawędzi określonych przez wcześniej wyznaczone narożniki. Rodzaj krawędzi jest wprost określony przez liczbę punktów wklęsłych leżących na niej. Brak takich punktów oznacza krawędź prostą, a więc zewnętrzną. Jeden punkt wklęsły oznacza krawędź z otworem, a dwa punkty - krawędź z główką.

## 8 Porównywanie krawędzi

Po wyznaczeniu krawędzi puzzli konieczne było określenie miary podobieństwa pomiędzy puzzlami. W naszym projekcie wybraliśmy dwie miary: bazującą na kształcie i kolorze. Po ich obliczeniu wyliczaliśmy średnią ważoną tych miar do wyliczenia ostatecznego podobieństwa krawędzi.

### 8.1 Podobieństwo geometryczne

Podobieństwo geometryczne krawędzi jest zdefiniowane jako średnia odległość euklidesowa pomiędzy kolejnymi punktami krawędzi po ich nałożeniu na siebie. W celu nałożenia na siebie krawędzi

wykorzystano wektory wyznaczone przez dwa graniczne punkty porównywanych krawędzi. Najpierw odnaleziono obrót oraz przesunięcie wektora tak by nałożył się na drugi, a następnie odnalezioną transformację nakładano na całą krawędź puzzla. Zaletą tej miary jest jej nieparametryzowalność - działa na każdym zbiorze puzzli. Wadą jest brak użyteczności tej miary w przypadku puzzli bez zniekształconych krawędzi, gdzie każde dwie krawędzie o pasującym typie są do siebie równie podobne.

### 8.2 Podobieństwo kolorystyczne

Podobieństwo kolorystyczne jest zdefiniowane jako średnia odległość w przestrzeni barw pomiędzy kolejnymi segmentami kolorystycznymi krawędzi. Jej wyznaczenie odbywa się w kilku krokach:

1. Wyznaczenie wektora kolorystycznego krawędzi. Odbywa się poprzez przechodzenie wzdłuż krawędzi filtrem (pełną macierzą jedynek) o określonym rozmiarze uśredniającym kolory pikseli
2. Podział wyznaczonego w poprzednim kroku wektora na określoną parametrem liczbę segmentów
3. Obliczenie średniej odległości w przestrzeni barw pomiędzy kolejnymi segmentami

Poza rozmiarem filtra oraz liczbą segmentów, porównanie to wymaga również określenia przestrzeni barw, w której puzzle będą porównywane.

## 9 Algorytm

Po określeniu miary podobieństwa kolejnym krokiem było stworzenie samego algorytmu układania puzzli. W naszym projekcie użyliśmy **algorytmu z nawrotami**. Rozpoczynając od lewego górnego narożnika, dokładaliśmy puzzle do prawej krawędzi, uzupełniając kolejne wiersze. Taki sposób układania jest prostszy od układania rozpoczynając od dowolnego puzzla ponieważ nie wymaga sprawdzania gdzie w danym kroku możemy dołożyć puzzel bez naruszania liczby wierszy/kolumn całego zestawu.

## 10 Zbiór danych

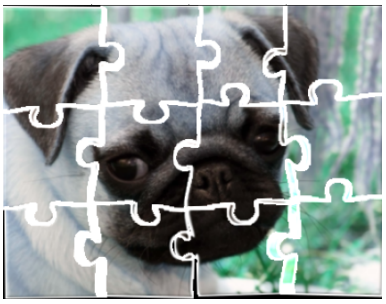
W celu walidacji naszego podejścia wykorzystaliśmy zarówno puzzle wygenerowane sztucznie jak i prawdziwe, sfotografowane aparatem. Wykorzystane zestawy są przedstawione na rysunku 3. Prawdziwe puzzle zostały oddzielone od w miarę jednolitego tła (zielony obrus oraz niebieska kartka).

## 11 Wyniki eksperymentalne

Początkowe eksperymenty pokazały, że algorytm dobrze radzi sobie z układaniem puzzli sztucznych. Brak szumów, charakterystyczne kształy i kolory umożliwiły ich łatwe ułożenie bez większego trudu związanego z wyszukiwaniem odpowiednich parametrów. Puzzle prawdziwe okazały się jednak problematyczne. Ręczne dostrajanie parametrów nie przyniosło porządanego efektu. Algorytm albo nie odnajdywał żadnego rozwiązania, albo wskazywał wiele nieprawidłowych rozwiązań. Z tego powodu podeszliśmy do problemu od drugiej strony: ręcznie wyznaczyliśmy prawidłowe rozwiązania dla obu zestawów puzzli prawdziwych, a następnie sprawdziliśmy jak bardzo algorytm zbliża się do nich w zależności od parametrów.

W celu sprawdzenia jakości algorytmu, badaliśmy dwa rozkłady:

1. **Rozkład rankingów prawidłowych dopasowań**. Obliczenie odległości pomiędzy wszystkimi krawędziami daje nam dla każdej krawędzi listę jego potencjalnych dopasowań posortowanych rosnąco według odległości. Ranking dopasowania danej krawędzi jest zdefiniowany jako pozycja prawidłowego dopasowania na tej liście. W idealnym algorytmie ranking prawidłowego dopasowania dla każdej krawędzi jest równy 0.



(a) Zestaw #1 Puzzle wygenerowane, 12 elementów

(b) Zestaw #2 Puzzle wygenerowane, 30 elementów

(c) Zestaw #3 Puzzle prawdziwe, 20 elementów

(d) Zestaw #4 Puzzle prawdziwe, 54 elementy

Rysunek 3: Przykładowe zestawy puzzli ułożone przez nasz algorytm

2. **Rozkład odległości pomiędzy prawdłowo dopasowanymi krawędziami** - odległości te powinny być istotnie mniejsze od odległości pomiędzy niedopasowanymi krawędziami.

W badaniach sprawdziliśmy kilka parametrów:

- **Paleta barw** używana podczas porównywania kolorystycznego krawędzi.
- **Liczba segmentów** na które była dzielona krawędź podczas porównywania kolorystycznego. Każdy segment był reprezentowany przez dokładnie jeden kolor.
- **Rozmiar filtra** za pomocą którego uśrednialiśmy kolor wzdłuż krawędzi.

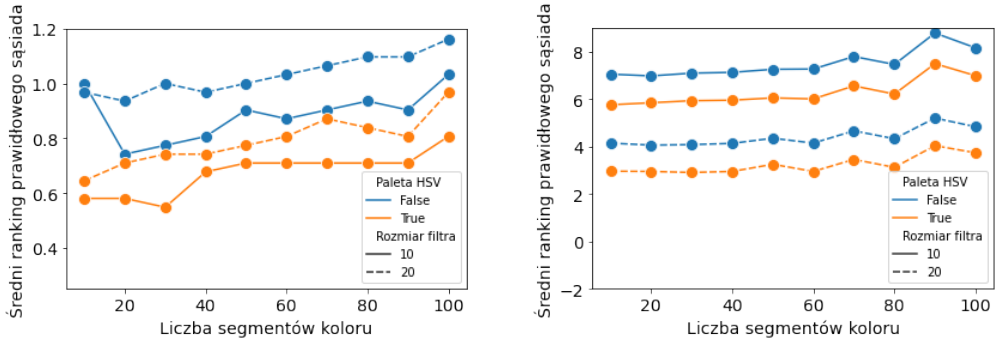
Wyniki dla badań zależności jakości algorytmu od liczby segmentów kolorów, rozmiaru filtra koloru oraz palety barw są przedstawione na rysunkach 4 i 5. W tych eksperymentach uwzględnialiśmy jedynie podobieństwo kolorystyczne krawędzi.

W przypadku liczby segmentów, dla obu zestawów puzzli optymalne okazały się wartości z zakresu 10-30. W przypadku zestawu puzzli #4 różnice nie są znaczące, jakość algorytmu spadała dopiero przy więcej niż 90 segmentach koloru.

Istotnym czynnikiem okazała się paleta bartw wykorzystywana przy porównywaniu krawędzi. W przypadku palety barw HSV wykorzystaliśmy jedynie dwa kanały: Hue oraz Saturation. We wszystkich eksperymentach takie podejście dało istotnie lepsze wyniki niż porównywanie za pomocą palety RGB.

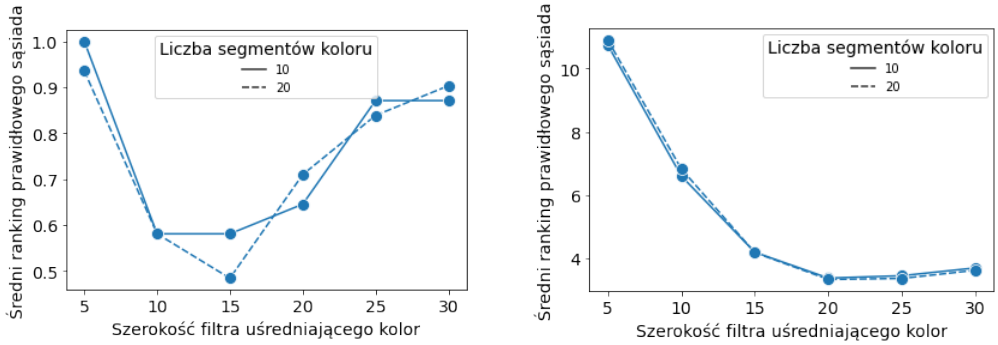
Rozmiar filtra również był istotnym czynnikiem wpływającym na algorytm. W przypadku zestawu #3 najbardziej optymalnym okazał się filtr rozmiaru 15x15, natomiast dla zestawu #4 20x20. Widać tutaj konieczność dostosowania tego parametru dla każdego zestawu puzzli z osobna, gdyż np. choć rozmiar filtra 10x10 daje dobre wyniki dla zestawu #3, to w przypadku zestawu #4 rozmiar ten okazuje się stanowczo za mały.

Rysunek 6 przedstawia rozkład rankingów prawidłowych dopasowań dla najlepszych odnalezionych parametrów dla obu zestawów puzzli, dla algorytmu wykorzystującego jedynie podobieństwo kolorystyczne. Na wykresach widać, że algorytm w większości przypadków prawidłowo wskazał dopasowaną krawędź. Zdarzały się jednak istotne pomyłki. W przypadku zestawu #4 kilka prawidłowych dopasowań znalazło się dopiero na ponad 40 pozycji listy najbliższych sąsiadów. To w




(a) Dla zestawu puzzli #3 (b) Dla zestawu puzzli #4

Rysunek 4: Zależność średniego rankingu prawidłowego dopasowania od liczby segmentów wektora koloru. Uwzględniona jest jedynie kolorystyczna miara podobieństwa.

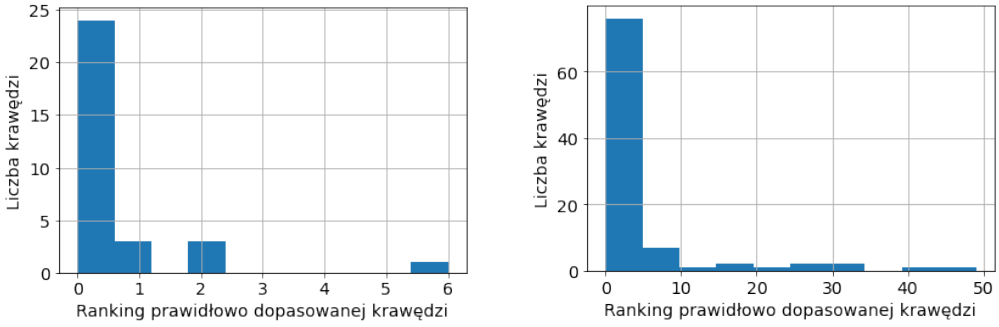



(a) Dla zestawu puzzli #3 (b) Dla zestawu puzzli #4

Rysunek 5: Zależność średniego rankingu prawidłowego dopasowania od rozmiaru filtra uśredniającego kolor krawędzi. Uwzględniona jest jedynie kolorystyczna miara podobieństwa.

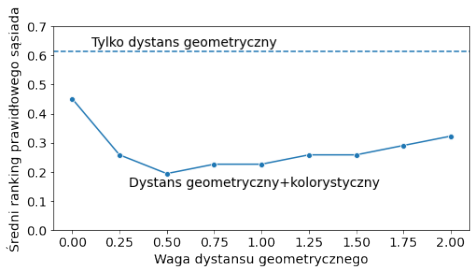
praktyce spowodowałoby niemożliwość uzyskania rozwiązania, dlatego konieczne było dodatkowe zastosowanie miary odległości geometrycznej.

Po wyznaczeniu optymalnych parametrów miary kolorystycznej, dodaliśmy do algorytmu miarę geometryczną. Rysunek 7 przedstawia wykresy zależności średniego rankingu prawidłowego dostosowania od wagi przydzielonej dystansowi geometrycznemu, przy ustalonej wadze podobieństwa kolorystycznego ($= 1.0$). Eksperymenty pokazały, że waga ta ma ogromny wpływ na ostateczną

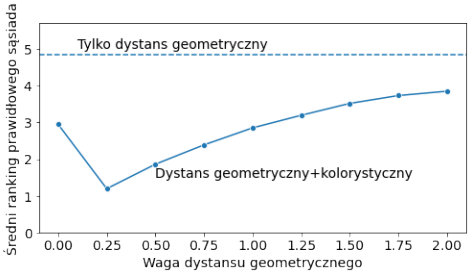



(a) Zestaw puzzli #3 (b) Zestaw puzzli #4

Rysunek 6: Rozkład rankingów najlepszych dopasowań dla najlepszej znalezionej konfiguracji parametrów. Uwzględniona jest jedynie kolorystyczna miara podobieństwa.




(a) Zestaw puzzli #3




(b) Zestaw puzzli #4

Rysunek 7: Zależność średniego rankingu prawidłowego dopasowania od wagi przydzielonej dystansowi geometrycznemu.

jakość algorytmu. W przypadku zestawu #3 najbardziej optymalną wagą dla dystansu geometrycznego okazała się wartość 0.5, zaś dla zestawu #4 - 0.25. W obu przypadkach prawidłowe dobranie wagi podobieństwa geometrycznego zmniejszyło średni ranking prawidłowego dopasowania ponad dwukrotnie.

## 12 Ocena końcowa

Zaimplementowany algorytm okazał się skuteczny dla małych zestawów puzzli zarówno sztucznych jak i rzeczywistych. Radzi sobie z puzzlami różnego typu, np. mocno zniekształconymi o dużej różnorodności geometrycznej, ale też prostymi, wykorzystując podobieństwo kolorystyczne. Wadą algorytmu jest jego silna zależność od parametrów, które trzeba ustawić dla każdego zestawu osobno. Innym problemem, z którym nie zdążyliśmy się zmierzyć, jest zniekształcenie perspektywiczne wynikające z perspektywy, z której wykonano zdjęcie puzzli. W małych zestawach, jakimi się zajmowaliśmy, nie było ono jednak istotnym problemem.

W dalszym rozwoju algorytmu warto byłoby wyznaczyć parametry najbardziej wpływające na jakość algorytmu oraz ich najbardziej racjonalne zakresy, dzięki czemu użytkownik mógłby je w wygodny sposób dostroić. Warto byłoby również zmodyfikować algorytm układania tak, by zaczynał nie od górnego lewego rogu, a od aktualnie najbardziej optymalnego możliwego połączenia. To przyspieszyłoby jego działanie a także zmniejszyło liczbę ewentualnych nieprawidłowych rozwiązań

## 13 Podsumowanie

Raport przedstawia automatyczny algorytm układania puzzli stworzony w ramach projektu Analiza obraz i wideo. Algorytm jest przeznaczony do automatycznego układania niewielkich zestawów puzzli ze zdjęć. Algorytm działa w sposób zachłanny wykorzystując podobieństwo kolorystyczne oraz geometryczne krawędzi. Wyniki eksperymentalne wskazują na jego skuteczność jednak silne uzależnienie od doboru parametrów.

## Literatura

[1] Dokumentacja biblioteki OpenCv. `https://docs.opencv.org/master`. [Dostęp 04-02-2021].

[2] Dokumentacja Jupyter Notebook. `https://jupyter-notebook.readthedocs.io/en/stable/`. [Dostęp 04-02-2021].

[3] Dokumentacja języka Python. `https://docs.python.org/3.8/`. [Dostęp 04-02-2021].

[4] Zbiór danych generowanych komputerowo puzzli z Kaggle. `https://www.kaggle.com/shivajbd/jigsawpuzzle`. [Dostęp 04-02-2021].

[5] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[6] D. A. Kosiba, P. M. Devaux, S. Balasubramanian, T. L. Gandhi, and K. Kasturi. An automatic jigsaw puzzle solver. In *Proceedings of 12th International Conference on Pattern Recognition*, volume 1, pages 616–618 vol.1, 1994.

[7] P. Ondruska. Automatic assembly of jigsaw puzzles from digital images. 2011.