POBR

Sprawozdanie z projektu

Dominika Ziółkiewicz

1. Wstęp

Celem projektu było wykonanie algorytmu do rozpoznawania wybranego obiektu na zdjęciach. Wybranym przeze mnie obiektem było logo firmy PZU S.A., które wygląda następująco:

Przed przystąpieniem do realizacji przyjęto kilka założeń:

- Język programowania: C++,
- Korzystanie ze struktur danych z biblioteki OpenCV, natomiast nie można korzystać z zaimplementowanych tam funkcji do przetwarzania obrazów,
- Logo nie może być uzyskane bezpośrednio cyfrowo za pomocą programu komputerowego,
- Nie jest uwzględniany gradient w otoczce na logu. Do opracowania algorytmu wykorzystane maja być litery na nim się znajdujące.

2. Implementacja

Po wykonaniu zdjęć rzeczywistego obiektu, przystąpiono do implementacji rozwiązania, które składa się z kilku etapów: przetwarzania obrazu, segmentacji i filtracji segmentów za pomocą niezmienników momentowych oraz detekcji.

Do wszystkich kroków i testów używane były następujące zdjęcia:

(a)

(b)

(c)

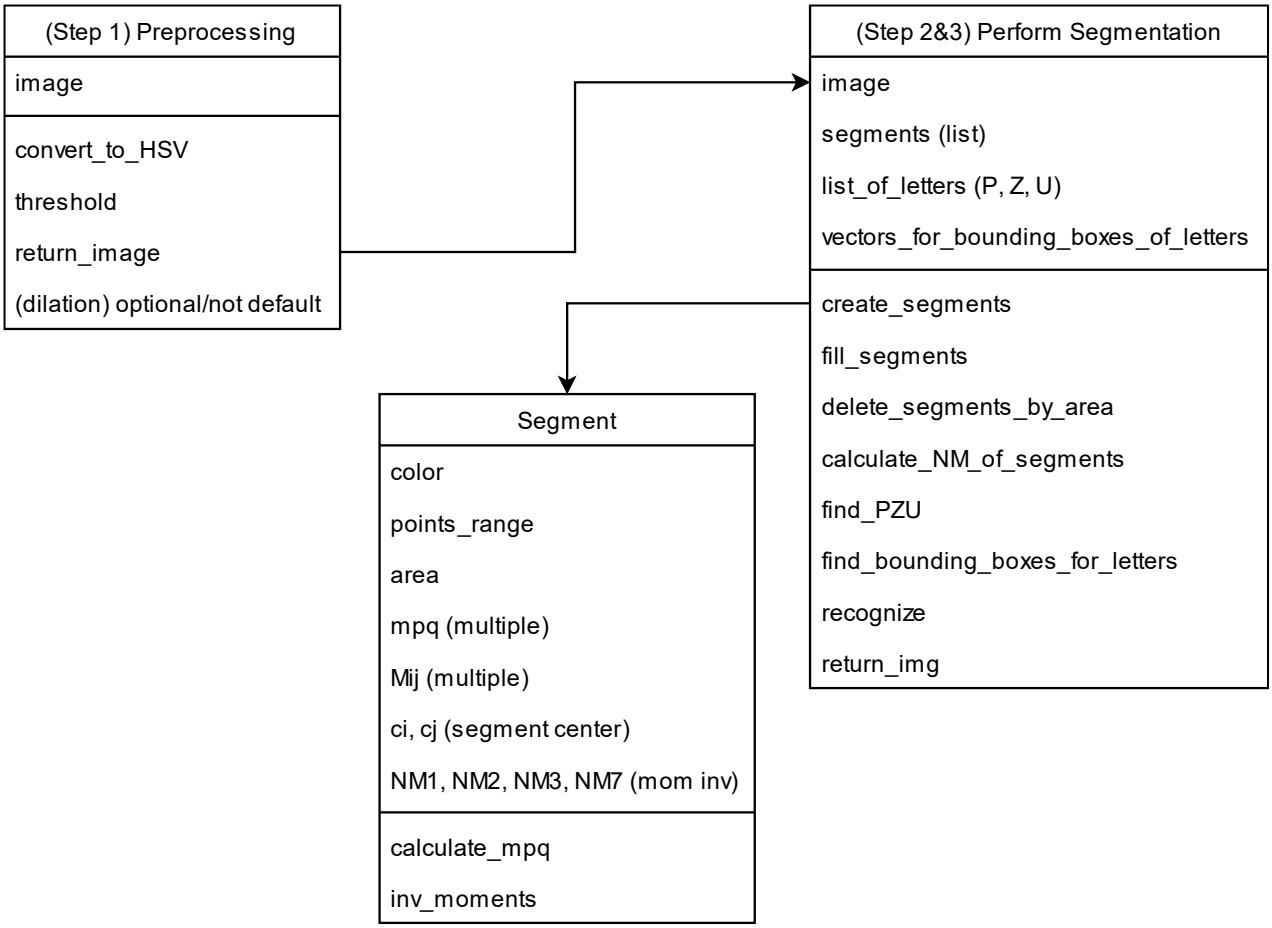
(d)

(e)

(f)

Warto zwrócić uwagę, że wszystkie zostały zrobione o tej samej porze jednego dnia a więc przy podobnych warunkach oświetleniowych. Przyjęto minimalny rozmiar obiektu jak ten na zdjęciu (a) oraz maksymalne pochylenie „w głąb” jak na zdjęciu (b).

Całość projektu można przedstawić za pomocą następującego schematu, do którego będą odniesienia w kolejnych podpunktach:



1.1 Przetwarzanie wstępne

Po zaciągnięciu obrazu, przechodzi on najpierw przez fazę przetwarzania wstępnego. Etap ten został zrealizowany w ramach jednej klasy *„Preprocessing”,* w której obraz zostaje:

1. Przekonwertowany z przestrzeni RGB na HSV
2. Przefiltrowany pod kątem *hue* – odcienia światła, *saturation* – nasycenia barwy i *value* – jasności, tak aby pozostawić na zdjęciu jedynie elementy odpowiadające konkretnemu odcieniowi granatowego, który występuje na logu
3. Poddany progowaniu – białe logo, reszta czarna

Ostatecznie uzyskane wartości filtra barwy granatowej wynoszą:

$$70 < hue < 240$$

$$40 < saturation < 255$$

$$60 < value < 200$$

1.2 Segmentacja

Segmentacja została zrealizowana w taki sposób, że program przechodzi po całym zdjęciu i dla każdego białego piksela rozpoczyna wypełnianie segmentu, do którego ten piksel należy. Obiekt klasy *„Segment”* dodawany jest na listę. Przechowuje on jednocześnie informacje o konkretnych pikselach, które do niego należą. Operacja ta jest zaimplementowana w klasie „*PerformSegmentation*” w ramach metod *create_segments* i *fill_segments.*

Kolejnym krokiem jest przefiltrowanie „odstających” elementów na obrazie. Po pierwsze - pod katem ich pola. Odrzucane są te, które są zbyt małe i zbyt duże żeby móc być poszukiwanym logiem. Po drugie – pod kątem kształtu. Tutaj przeprowadzono analizę czterech niezmienników momentowych (NM1, NM2, NM3 i NM7) i określono dla jakich przedziałów ich wartości, uzyskiwana jest każda z trzech liter. Jednocześnie, podczas tej filtracji, dorzucano litery (segmenty) na odpowiednie listy (P, Z, U) i wyrzucano z listy wszystkich segmentów te, które nie pasują. Operacje te zaimplementowane są w metodach *delete_segments_by_area, calculate_NM_of_segments* i *find_PZU.*

1.3 Detekcja

Mając określone segmenty, które są literami, odszukano dla każdej z nich tzw. *„bounding boxa”,* a właściwie tylko lewy górny i prawy dolny jego róg. Algorytm rozpoznania pojedynczego loga wygląda następująco:

Krok 1: Weź literę P.

Krok 2: Oblicz środek jej bounding boxa.

Krok 3: Weź literę Z.

Krok 4: Oblicz środek jej bounding boxa.

Krok 5: Przejdź po literach U i znajdź taką, o najmniejszej odległości do Z na podstawie środków bounding boxów.

Krok 6: Sprawdź, czy odległość między P i Z jest zbliżona do odległości miedzy Z i U.

Jeśli TAK:

Krok 7: Sprawdź, czy suma odległości między P i Z oraz Z i U jest zbliżona do odległości miedzy P i U.

Jeśli TAK:

Znalezione LOGO -> rysuj bounding box.

Jeśli NIE:

Przejdź do Kroku 3.

Jeśli NIE:

Przejdź do Kroku 3.

Powyższe kroki wykonują się do wyczerpania liter P. Operacje detekcji zostały zaimplementowane w metodach *find_bounding_boxes_for_letters* i *recognize.*

3. Wyniki i wnioski

Dla wszystkich wymienionych wcześniej zdjęć, przetestowano działanie programu uzyskując następujące wyniki:

Widać, że na wszystkich zdjęciach użytych do implementacji rozwiązania, program sobie radzi. Sprawdzono jeszcze wynik, gdy kilka obiektów rozpoznawanej klasy znajduje się na jednym zdjęciu:

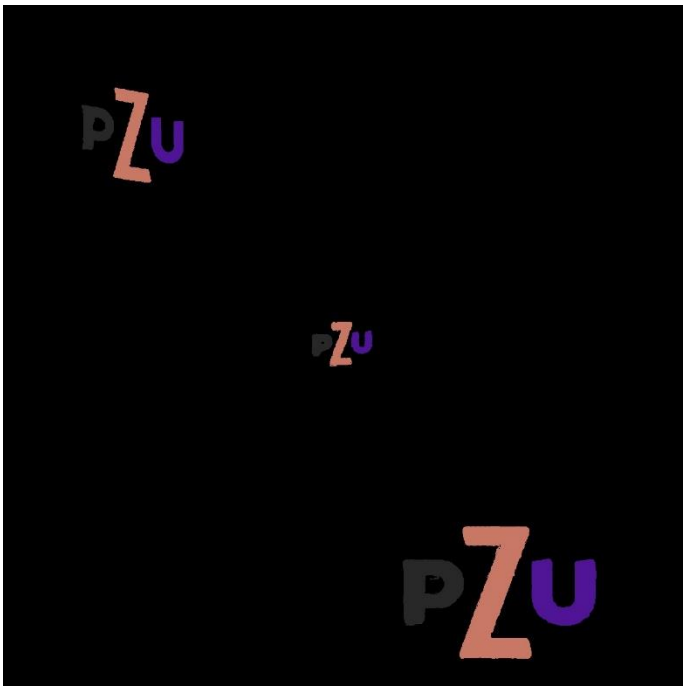
Jak można zauważyć, mimo umieszczenia trzech log w różnych wariantach ustawienia na jednym obrazku, algorytm wykrył wszystkie trzy poprawnie. Wykonano w takim razie jeszcze bardziej wymagający test, a wyniki umieszczono poniżej:

Tutaj widać, że algorytm nie poradził sobie w dwóch sytuacjach:

1. Logo na samej górze z lewej jest zbyt słabej rozdzielczości (w dodatku w formacie .jpg).
2. Logo w środku jest zbyt pomniejszone względem założonego minimum.

W tym pierwszym przypadku, po analizie jak faktycznie wygląda rozdzielenie na segmenty odpowiednich liter, okazało się, że są przypadki gdy litera U jest klasyfikowana jako P lub odwrotnie. Dzieje się tak, ponieważ wszystkie niezmienniki momentowe, które są używane, w pewnych sytuacjach przyjmują wartości podobne zarówno dla jednej jak i drugiej litery. Z tego wynikała trudność odpowiedniego ustawienia przedziałów wartości dla każdego z nich aby klasyfikacja przebiegała jak najpoprawniej.

Generalnie miejsca, w których trzeba było ręcznie dostosowywać parametry były dwa. Pierwszym było progowanie po kolorze już w przestrzeni barw HSV. Ten zakres dobierano doświadczalnie, ponieważ kryteria dla koloru granatowego dostępne w sieci były w innych skalach niż potrzebne. Hue podaje się w zakresie 0-360° a saturation i value w %, natomiast program potrzebował wartości z przedziału 0-255, także informacje znalezione na temat koloru granatowego w Internecie posłużyły tylko jako pierwsze przybliżenie. Po filtracji koloru, pola i segmentacji z pozostawieniem samych liter P, Z, U obraz z trzema logami wyglądał następująco:



Drugim takim miejscem, gdzie wartości były ustawiane ręcznie to zakresy niezmienników momentowych. Dla litery Z wartości zwłaszcza NM1 odstawały bardzo mocno od reszty, dlatego tutaj filtracja była najskuteczniejsza. Natomiast jak już wspomniano najciężej proces doboru zakresów, przebiegał dla liter P i U. W idealnych warunkach wszystkie niezmienniki były odróżnialne ale gdy logo było przechylone lub oddalone to potrafiły się pokrywać.

Podsumowując, opisane przeze mnie rozwiązanie detekcji wybranej klasy obiektu za pomocą niezmienników momentowych, może być skuteczne przy zajściu odpowiednich warunków, zwłaszcza oświetleniowych. Wybrany obiekt nie może być zbyt oddalony lub przechylony, ponieważ wtedy wartości niezmienników momentowych zaczynają się jednak mocniej zmieniać i przy filtracji kształtów taki obiekt może zostać odrzucony. Rozwiązanie to ma jednak swoją zaletę, mianowicie nie wymaga dużej ilości zdjęć (brak treningu jak np. przy klasyfikatorach) oraz jest względnie proste w implementacji.