Przetwarzanie Cyfrowe Obrazów

Mateusz Węcławski

Politechnika Warszawska

30/05/2025

Sprawozdanie z projektu: Wykrywanie Tęczowego Logo Apple

1. Wstęp

Celem detekcji było dawne logo przedsiębiorstwa Apple – symbolu przypominające ugryzione jabłko, i podzielonena 6 kolorowych pasków przypiminając tęczę. Analiza opiera się na detekcji kształtu i kolorów.

2. Algorytm detekcji

Algorytm detekcji wygląda następująco:

1) Wczytanie zdjęcia
2) Przekonwertowanie zdjęcia z RGB na HSV w celu lepszego rozpoznawania kolorów (oddzielona luminancja od chrominancji)
3) Wydzielenie masek dla każdego koloru – tzn. wszystkie barwy na obrazie które są np. zielone – zostają przekonwertowane na kolor biały (255), a wszystkie inne barwy na czarny (0). Dzięki temu dostajemy 6 różnych masek dla 6 różnych pasków

3B*) Filtracja poszczególnych masek – liczone są niezmienniki momentowe i porównywame z wzorcowymi niezmiennikami dla poszczególnych kształtów w logo Apple

4) Połączenie wszystkich masek w jedno – dzięki temu mamy oddzielone jabłko od tła, widoczny jest biały obiekt na czarnym tle
5) Rozpoznawanie dużych skupisk białych obiektów i nadawanie im etykiet – np. obiekt 1, obiekt 2, etc. Dzięki filtracji zazwyczaj będzie jeden duży obiekt (jabłko) i ewentualnie obiekty które przypominają kształtem pasek

6) Liczenie dla każdego obiektu jego bounding boxa i pola powierzchni
7) Sortowanie obiektów malejąco ze względu na rozmiar pola powierzchni
8) Iteracja po każdym obiekcie (w programie zaznaczyłem iterację po 5 obiektach w celu szybszego działania programu) – wyznaczanie 7 niezmienników momentowych i porównywanie ich z wzorcowymi niezmiennikami momentowymi dla loga apple (został ustawiony próg maksymalnie 2 niezgodnych niezmienników, tzn. wystarczy że 5 niezmienników jest zgodna, ponadto bufor zgodności został ustawiony na +- 1.5, tzn. jeśli wzorcowy niezmiennik wynosi 5, to wykryty niezmiennik maksymalnie może wynosić <3.5,6.5>.
9) Jeśli została wykryta zgodność w niezmiennikach, następuje detekcja kolorów na wyciętym obrazie (obiekcie wysegmentowanym)
10) Po detekcji, porównywana jest zgodność wykrytych kolorów z kolejnością wzorcową, tzn. np. że po kolorze fioletowym występuje kolor niebieski. Podczas detekcji przypisana została mediana wartości pikseli każdego koloru, która służy do stwierdzenia, który kolor występuje w której kolejności w wykrywanym obiekcie (większe wartości Y są „niżej" – kolor zielony będzie miał najmniejszą medianę, a kolor niebieski największą) – Jeśli zgodność pokrywa się w 5 na 6 kolorów – zostało pomyslnie wykryte logo.
11) Nakreślenie bounding boxa na obrazie i przedstawienie wyniku użytkownikowi

3. Wybrane zastosowane techniki i funkcje

```python
def is_rainbow_in_order(detected_colors):
    # funkcja do sprawdzenia czy kolory są po kolei
    isOrder=False
    correct_order = ['green', 'yellow', 'orange', 'red', 'purple', 'blue']

    # sortowanie wykrytych kolorow tylko po wsp Y (na samej gorze po lewo jest 0,0 wiec powinno byc rosnąco)
    # tzn ze green ma najmniejsza wartosc a niebieski najwieksza
    sorted_colors = sorted(detected_colors.items(), key=lambda x: x[1])

    if len(detected_colors) < 5:
        print(f"Not enough colors detected ({len(detected_colors)} < 5).")
        return False

    matchedOrderBuffor=0
    for i in range (0,len(sorted_colors)-1):
        sorted_name,sorted_value = sorted_colors[i]
        sorted_name1,sorted_value1 = sorted_colors[i+1]
        if(sorted_value>=sorted_value1):
            matchedOrderBuffor=matchedOrderBuffor+1
    if(matchedOrderBuffor<2):
        isOrder=True
    # 5 a nie wszystkie 6 kolorow, wiekszy bufor, moge nie patrzec tak stricte na wszystkie kolory *max 1 koloru moze brakowac*
    # max jeden kolor moze byc w innej kolejnosci
    return isOrder
```

Funkcja służąca do detekcji poprawnej kolejności kolorów

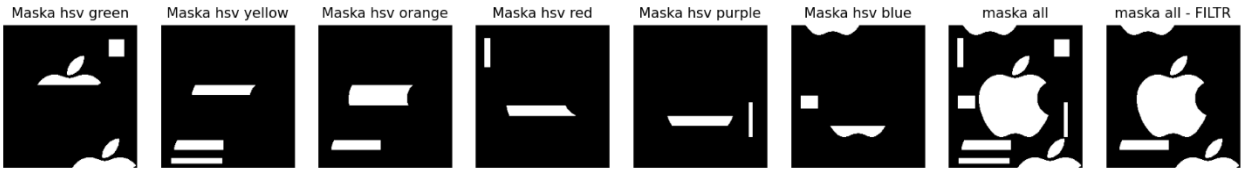
```python
def myConnectedComponentsWithStats(binary_image, connectivity=8):
    # Labelowanie komponentow (grup pikseli)
    labels, num_features = label(binary_image, structure=np.ones((3,3)) if connectivity == 8 else np.array([[0,1,0],[1,1,1],[0,1,0]], dtype=int))

    # Ini stats
    stats = np.zeros((num_features + 1, 5), dtype=np.int32) # x, y, width, height, area

    # petla po kazdych labelach
    for i in range(1, num_features + 1):
        component_mask = (labels == i) # maska boolean dla label i
        rows, cols = np.where(component_mask) # pobieranie pikseli komponentu

        if len(rows) > 0:
            y_min, y_max = np.min(rows), np.max(rows)
            x_min, x_max = np.min(cols), np.max(cols)

            x_bbox = x_min
            y_bbox = y_min
            w_bbox = x_max - x_min + 1
            h_bbox = y_max - y_min + 1
            area = np.sum(component_mask)

            stats[i, :] = [x_bbox, y_bbox, w_bbox, h_bbox, area]

        else:
            stats[i, :] = [0, 0, 0, 0, 0]


    return num_features + 1, labels, stats # +1 dla tla label 0
```

Funkcja odpowiedzialna za nadawanie etykiet dużym skupiskom pikseli, a następnie obliczanie bounding boxa i pola powierzchni dla obiektów w masce złożonej z poszczególnych masek

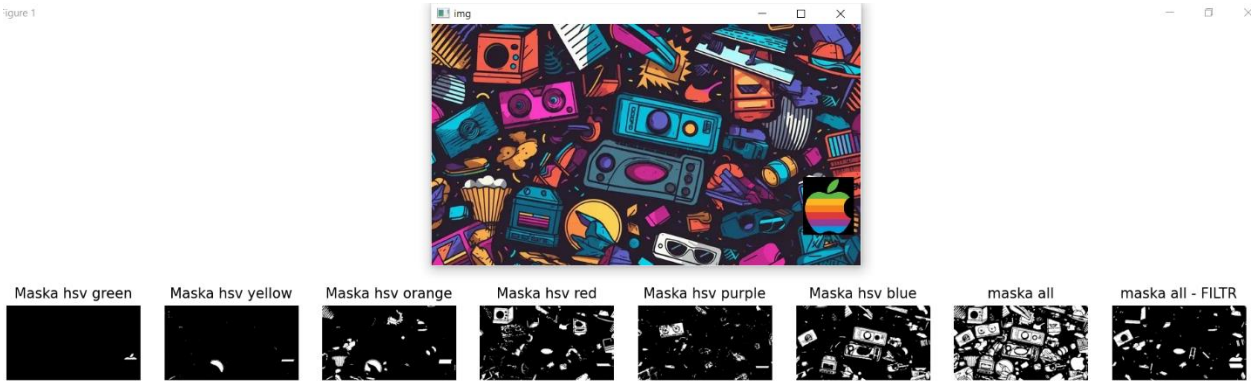
4. Wnioski z implementacji i testowania

1) Błędny wzór na niezmiennik momentowy nr 7 w literaturze – posługiwałem się dokumentem przedstawionym na laboratorium i wszystkie niezmienniki były obliczane zgodnie z ogólnodostępną fukcją w openCV, oprócz niezmiennika 7. Przejrzałem internet i tam wzór na ten niezmiennik jest znacznie inny – znacznie dłuższy, po zastosowaniu wzoru z internetu niezmiennik był już obliczany poprawnie
2) Branie największego pola – na początku implementacji brałem tylko największy wykryty obiekt – rozumowanie błędne ponieważ po 1) mógłby być po prostu na obrazie duży niebieski kwadrat i on byłby poddany działaniom detekcji, a potencjalne logo już nie. Po 2) nie uwzględniało to tego, że na obrazie może być więcej niż 1 logo. W celu naprawy tego błędu, używam teraz pętli i poddaje analizie 5 największych wykrytych obiektów – nie jest to idealny sposób, ale ograniczyłem to na 5 obiektów ze względu na przyspieszenie działania programu.
3) Zostały dodane filtry poszczególnych masek kolorów – obliczane są niezmienniki momentowe obiektów znajdujących się wewnątrz tych masek, i odrzucane są te, które nie przypominają pasków. Wadą dodania filtrów jest bardzo zwiększony czas działania programu (iteracja po wszystkich elementach danej maski)
4) Algorytm odporny jest na obroty, skalowanie, itp. – dzięki niezmiennokom

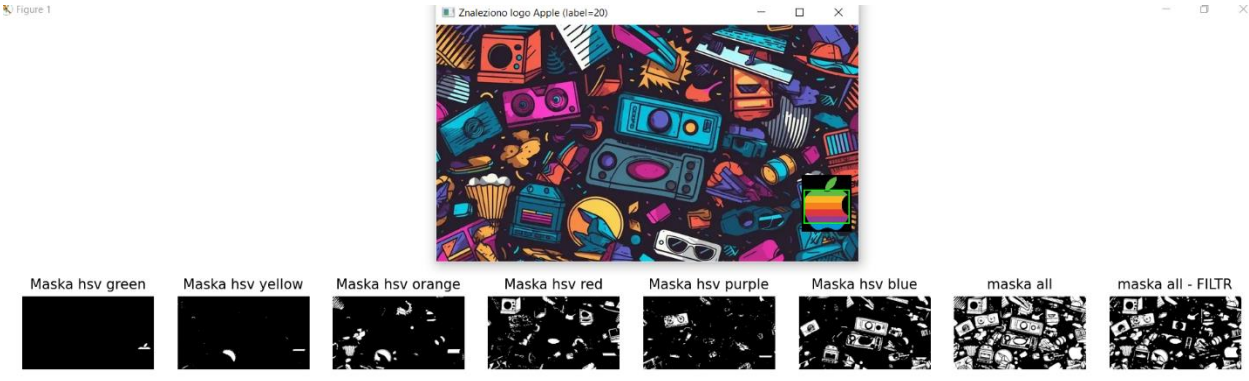
5. Wyniki



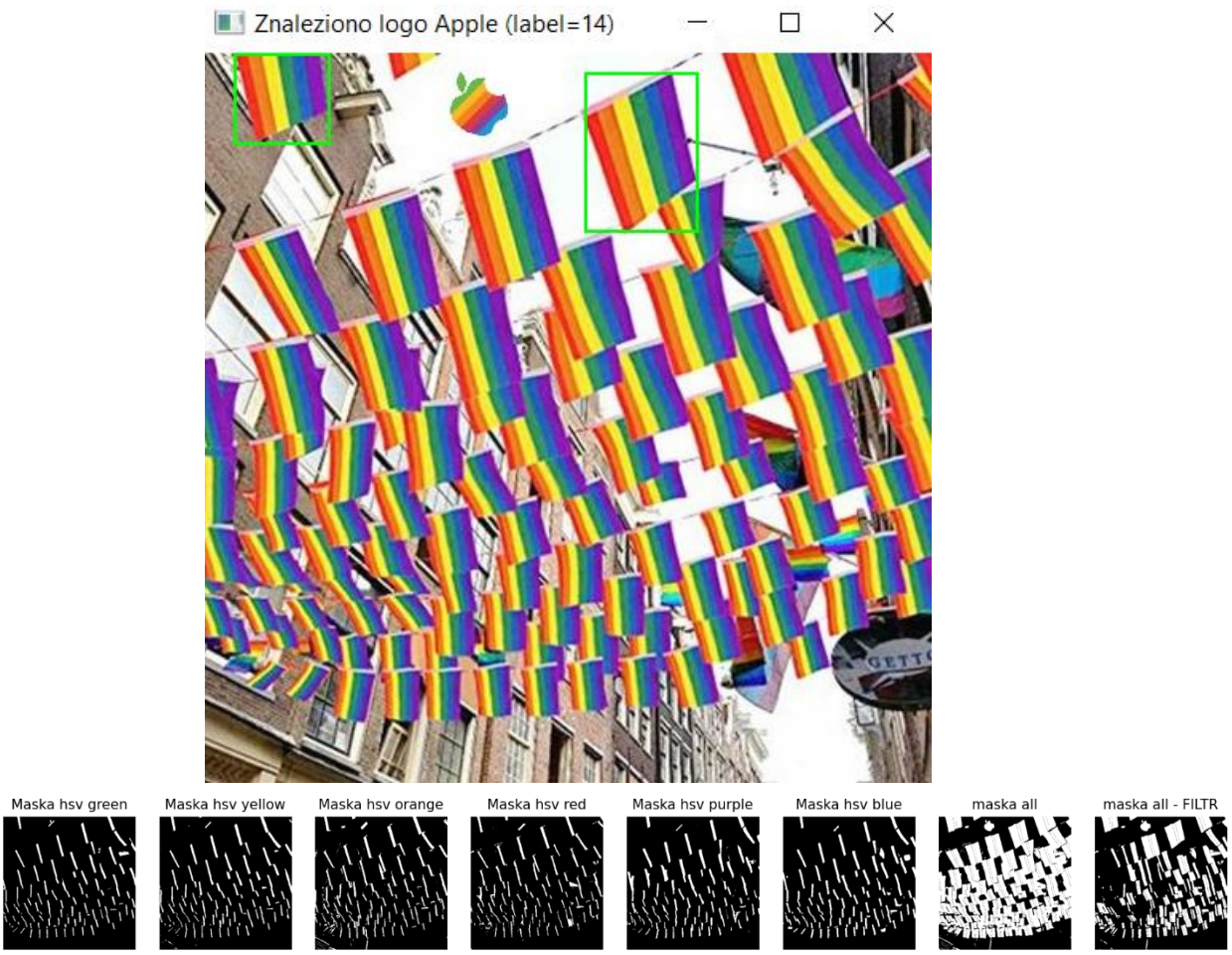
Jak widać, kolorowe fragmenty, które nie przypominają wystarczająco poszczególnych pasków zawartych w logu Apple, zostały odfiltrowane. Algorytm poradził sobie dobrze.



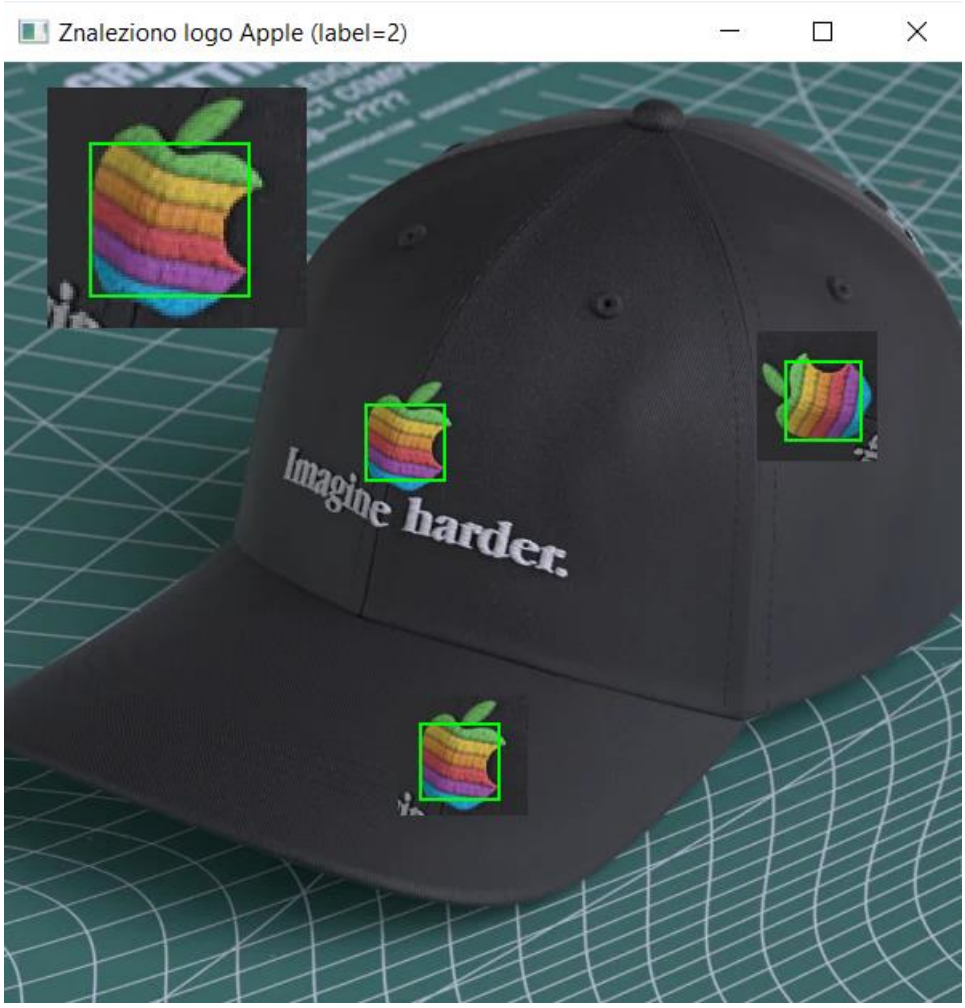
Dla tego obrazu niestety filtracja okazała się za bardzo restrykcyjna – została odfiltrowana niebieska i pomarańczowa część jabłka – kwestia rozszerzenia zakresu.

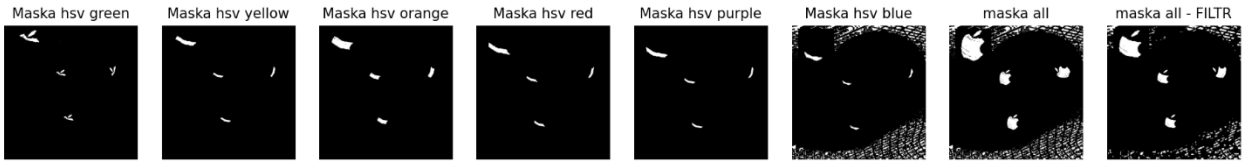


Dzięki zwiekszonemu zakresu, logo zostało poprawnie wykryte.



Dla tego obrazu, algorytm sobie niestety nie poradził (2 fałszywe wykrycia) – filtr źle tu działa, tzn. dla tego obrazu zakresy niezmienników momentowych powinny być znacznie bardziej restrykcyjne.





Najlepszym sposobem na poprawę działania algorytmu byłoby używanie oddzielnych docelowych zakresów niezmienników dla każdego z fragmentów logo, a nie tak jak jest teraz, używanie tych samych dla każdej maski.