**Sprawozdanie z laboratorium: Komunikacja człowiek-komputer**

**Przetwarzanie obrazu –Wykrywanie liczby oczek na kostce**

Zajęcia czwartkowe, 15:10

**Abstrakt**

W dokumentacji prezentujemy rozwiązanie klasycznego problemu wykrywania oczek na kostkach. Opisujemy rozwiązanie pozwalające wyodrębnić dobrą klatkę z filmu przedstawiającego rzucanie kośćmi, z którego algorytm wykrywa kości, ich pozycje i oblicza liczbę oczek na górnej ściance. Rozwiązanie obejmuje wiele funkcji z biblioteki Open Source „OpenCV” oraz hierarchicznego grupowania biblioteki Scipy. Opisujemy również wyniki eksperymentów, wyzwania, przyjęte założenia, zalety danego rozwiązania i jego ograniczenia. Dodatkowo sprawozdanie zawiera propozycję dalszych kroków mających na celu poprawę wyników.

**Metodologia**

Główne wyzwania, które są wyjątkowe w tym problemie, są następujące:

• Filmy przedstawiające rzucanie kostką nie są jednakowe, ponieważ długość każdego filmu i liczba klatek na sekundę jest inna w każdym filmie.

• Filmy, które zostały ustawione na wyższym poziomie, aby uwzględnić otoczenie tacy lub stołu, również powodują wykrycie w nich obiektów o okrągłym kształcie.

• Inne ustawienie kamery oznacza, że ​​boczne powierzchnie kostek są również uruchamiane w celu wykrycia oczek, a obiektyw typu rybie oko wymaga wstępnej obróbki obrazu uzyskanego przez taki obiektyw przed uruchomieniem właściwego algorytmu.

• Szybsze, bardziej energiczne rzuty kostką.

• Niewyraźne i hałaśliwe filmy z różnymi oświetleniami zwiększają złożoność.

• Ciała obce przypominające kształtem i kolorem kostki z fałszywymi plamkami w tle prowadzą do fałszywych trafień.

• Tło / taca i kostki w różnych kolorach oraz przezroczyste kostki są trudne do wykrycia i sklasyfikowania.

Logiczny podział kroków podjętych w tym zadaniu jest następujący:

1) Ładowanie wideo, przeglądanie każdej klatki i wykrywanie nieruchomej klatki   
2) Znalezienie obszaru zainteresowania, w którym powinno zostać uruchomione wykrywanie ‘oczek’.  
3) Wstępne przetwarzanie wyodrębnionej nieruchomej ramki, aby można było łatwo wykryć ‘oczka’.  
4) Wyodrębnienie ‘oczek’ z każdej kostki i przypisanie ich do właściwej kostki wraz z pozycją kostki.

**Pozyskiwanie ramek**

Ponieważ uwzględnienie każdej klatki kosztowałoby dużo pamięci i czasu, ta metoda dostosowuje mechanizm, w którym tylko dwie klatki na obraz są wybierane na każdą sekundę wideo. Jednym z założeń przyjętych w tym kroku jest to, że kostki pozostaną nieruchome przez co najmniej jedną sekundę w filmie. W ten sposób odczytywane są wszystkie ramki, a numer każdej ramki jest zapisywany w zmiennej. Klatka jest brana pod uwagę przy wyodrębnianiu prawej klatki tylko wtedy, gdy liczba klatek jest wielokrotnością połowy szybkości klatek wideo. Ponadto kilka pierwszych klatek i kilka ostatnich klatek jest ignorowanych, aby krok wyodrębniania klatek koncentrował się tylko na środkowej części każdego wideo. Takie podejście jest stosowane, dlatego widać, że każdy film zaczyna się od ręki, która ma rzucić kostką na tacę, a kończy ręką, która zbiera wszystkie kości z tacy.

Uzyskane w ten sposób klatki są zapisywane na liście po przekonwertowaniu ich na skalę szarości za pomocą metody cvtColor openCV. Obrazy są konwertowane do skali szarości, ponieważ to podejście nie jest zależne od żadnych cech zależnych od koloru, aby wyodrębnić liczbę oczek lub wykryć kostki. Ramki na liście są rozpatrywane parami w kolejności od początku do końca.

Po tych wszystkich krokach liczba niezerowych pikseli w otrzymanym ostatecznym obrazie jest przechowywana na osobnej liście dla każdej pary obrazów. Lista niezerowych pikseli dla każdej pary ramek przedstawia wielkość ruchu, jaki wystąpił między dwiema klatkami w każdej parze. Zatem druga klatka z pary, która miała minimalny ruch między nimi, reprezentowałaby klatkę, która nie miała żadnych ruchów, a zatem ramkę nieruchomych kości po ułożeniu.

**Określenie regionu zainteresowania**

Po uzyskaniu właściwej klatki obraz z nieruchomymi kostkami jest przepuszczany przez dwustronny filtr, aby zredukować efekty szumu uzyskiwanego z hałaśliwych kamer. Do tego zadania wybrano filtr dwustronny (bilateral filter), ponieważ po rozmyciu daje lepsze obrazy niż jakikolwiek inny algorytm rozmycia, który ślepo uwzględniłby wszystkie piksele w sąsiedztwie piksela. Filtr dwustronny zachowuje ważne cechy, takie jak krawędzie i rogi, ponieważ pobiera średnią ważoną sąsiednich pikseli, zwiększając wagę poszczególnych cech. Z tym filtrem należy obchodzić się ostrożnie i odpowiednio go dostrajać, ponieważ złe zastosowanie tego filtra może spowodować fałszywe krawędzie.

Rozmyty obraz jest dopasowany do stałej ramki w kolorze czarnym, aby uniknąć wykrywania obiektów o okrągłym kształcie, które nie leżą na tacy (a więc nie należą do żadnej matrycy). Piksele w 6% góry, 9% dołu i 5% każdej prawej i lewej strony obrazu są ustawiane na czarno, co daje efekt nałożenia czarnej ramki na obraz.

**Wykrywanie oczek**

Obraz z ustawionym regionem zainteresowania byłby używany do wykrywania ‘oczek’ za pomocą algorytmu wykrywania ‘bloba’(oczek/plam) z openCV. Algorytm działa w następujący sposób:  
1) Konwersja obrazu początkowego na wiele obrazów binarnych za pomocą funkcji ‘tresholding’ (ustawianie dowolnego piksela o wartości wyższej od progu na kolor, a mniejszych na inny kolor) z różnymi wartościami progowymi, począwszy od progu minimalnego do progu maksymalnego.  
2) Grupowanie białych pikseli, które są bliżej siebie, w jedną całość, tak aby plamki można było łatwo zidentyfikować na każdym z obrazów binarnych.  
3) Rozważane są środki każdego z obiektów w każdym obrazie binarnym, a inne obiekty w sąsiedztwie są łączone z początkowym obiektem, jeśli obiekt znajduje się w minimalnym progu odległości. Ten krok pozwala uniknąć traktowania dużego obiektu jako dwóch oddzielnych obiektów.  
4) Na koniec zwraca się położenie środka tak rozpoznanych ‘oczek’ i promienia odpowiednich ‘oczek’.

**Parametry**

**(min\_threshold,max\_threshold,min\_area,max\_area,min\_circularity,min\_inertia\_ratio)**

Funkcja wykrywania plamek w openCV oferuje opcję filtrowania wykrytych w ten sposób plamek za pomocą koloru, rozmiaru i kształtu (które są definiowane przez to, jak blisko okręgu znajduje się obiekt, jak wypukły jest on i jak płaski jest okrąg). Zakłada się, że kostki rzucane w filmach nie są mniejsze niż określony rozmiar (określony przez obszar minimalny) i nie większy niż określony rozmiar (określony przez obszar maksymalny). Innym subtelnym założeniem jest to, że filmy nie są nagrywane w pozycji bardzo pochylonej (określonej przez współczynnik minimalnej bezwładności).

Z uwagi na to, że testowane przez nas filmy zawierały różne rodzaje kostek oraz różną odległość kamery od obiektów parametry nie są stałe w całym badaniu. Gdy badaliśmy kostki szklane, o różnych kolorach parametry przedstawiały się tak: min\_threshold = 50, max\_threshold = 200, min\_area = 20, max\_area = 68, min\_circularity = .3,   
min\_inertia\_ratio = .4 natomiast badając kostki drewniane z czarnymi oczkami parametry ustawiliśmy na takie: min\_threshold = 50, max\_threshold = 300, min\_area = 20, max\_area = 90, min\_circularity = .2, min\_inertia\_ratio = .2

**Określanie lokalizacji kości i liczby oczek**

Po wykryciu wszystkich oczek należy je oddzielić w odniesieniu do różnych kostek, a odpowiednie oczka przypisać do właściwej kostki. Do tego zadania używany jest hierarchiczny algorytm grupowania biblioteki Scipy. Algorytm hierarchicznego grupowania traktuje każde oczko jako oddzielną grupę(cluster) i łączy oczka, które są bliżej siebie, w jedną grupę w miarę wzrostu progu. Wartość thresh=39 jest używana w tym zastosowaniu do oddzielenia oczek, które są daleko i grupowania tych, które są bardzo blisko siebie. Pozycja czubków uzyskana z algorytmu wykrywania punktu kluczowego openCV jest wykorzystywana do znalezienia pozycji każdej kostki, biorąc średnią ze wszystkich lokalizacji oczek, ponieważ oczka na kostce zawsze byłyby rozproszone wokół środka górnej powierzchni. W ten sposób każda grupa jest uważana za kość, a pozycja matrycy jest brana z położenia oczek w tej grupie.

**Wyniki**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nazwa pliku** | **Liczba kostek** | **Kostki wykryte** | **Liczba oczek** | **Oczka wykryte** |
| test6big | **5** | **5** | **16** | **14-15** |
| test5black | **3** | **3** | **11** | **11** |
| test10es | **3** | **3** | **9** | **7-9** |
| test11ess | **3** | **3** | **12** | **10** |
| test12koldra | **3** | **3** | **13** | **9 (bardzo trudny przypadek, stół zawierał czarne plamy)** |
| test13slabajakosc | **3** | **3** | **13** | **10** |

Trudnościami napotkanymi podczas testowania programu były:

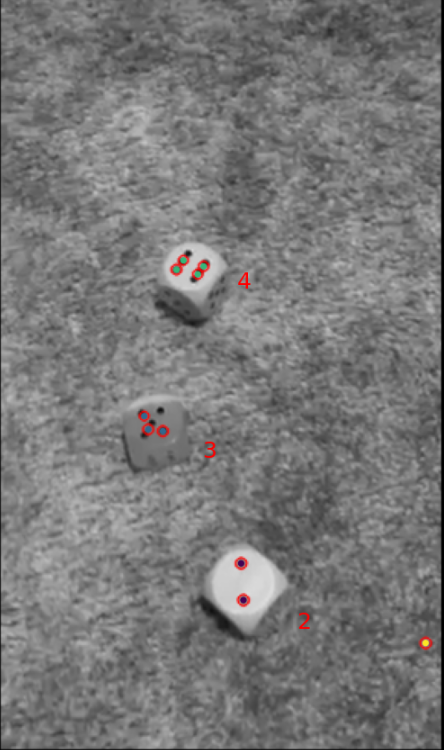
- manualne ustawianie parametrów (opisanych wyżej) w zależności od filmiku, częstą sytuacją było to, że algorytm wykrywał, np. szwy na pościeli, na której rzucaliśmy kostkami jako kropki, nierówności w stole/biurku, czarne kropki na stole

-jakość filmików, przy słabej jakości i małej liczbie pixeli program często nie był w stanie poprawnie wykryć wszystkie oczka nawet przy idealnym doborze parametrów

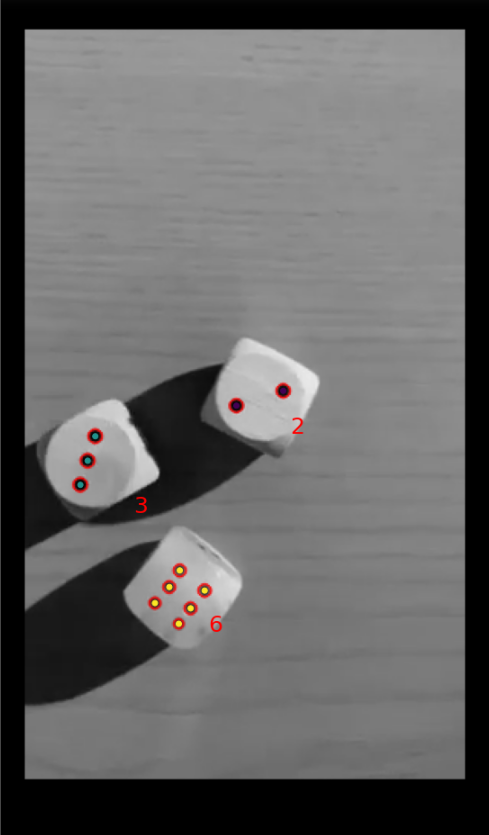
-błędny dobór klatki filmu- czasem algorytm wyłapywał klatkę z filmu w której jedna z kostek była ‘ostra’ a reszta jeszcze była w tzw. ‘locie’

Aby poprawić wyniki należałoby podać na początku zadania ile kostek i ile oczek powinno być wykryte, po czym napisać algorytm który dostosowuje parametry do otrzymania pożądanego wyniku.

**Przykładowe zrzuty ekranu**



Zrzut ekranu pochodzi z wyników po badaniu filmu „test13slabajakosc” który był w słabej jakości, przez co algorytm wykrył oczko kostki na stole.



Zrzut ekranu pochodzi z filmiku „test5black” w którym są przyjęte najlepsze warunki badawcze- nagranie jest ostre, nie ma problemów z otoczeniem.



Zrzut ekranu z filmiku „test6big”, kostki oryginalnie były różnokolorowe, nagranie po przybliżeniu nie jest do końca ostre.