Sprawozdanie z laboratorium:
Komunikacja człowiek-komputer

Część IV: Obrazy II

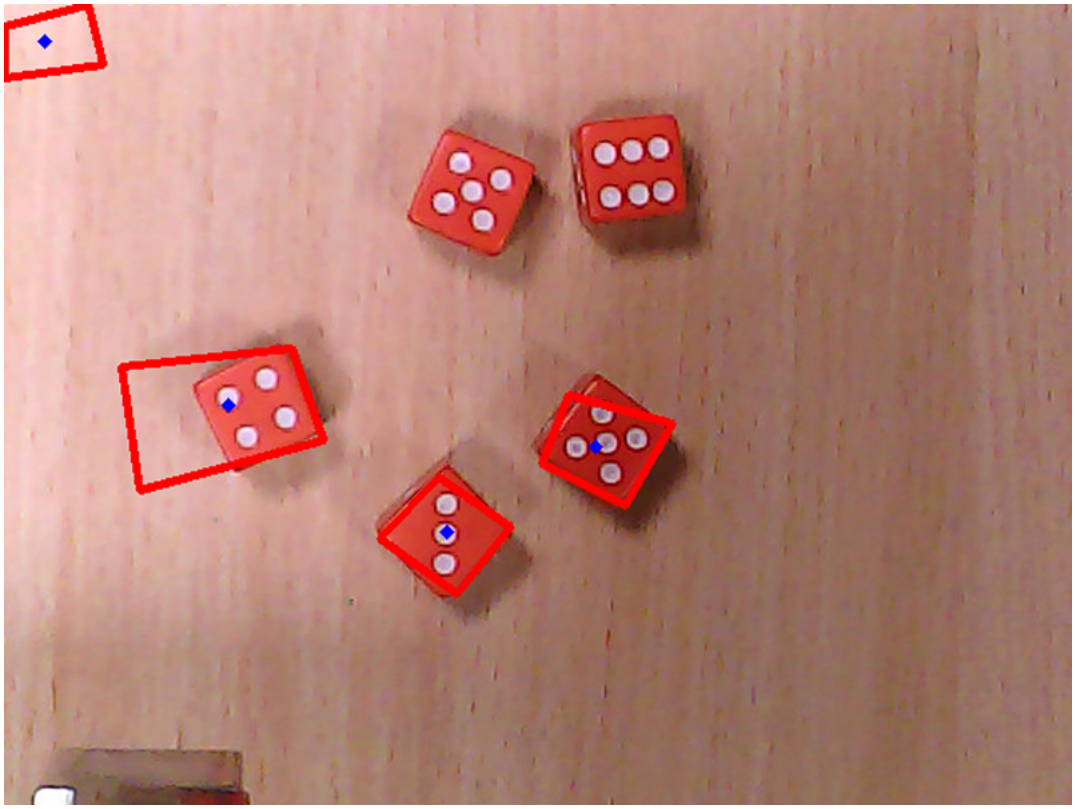
7 grudnia 2016

Prowadzący: dr hab. inż. Maciej Komosiński

Autorzy:	**Kamil Piotrowski**	inf122491	WI	kamil.piotrowski@student.put.poznan.pl
	Michał Lewiński	inf122505	WI	michal.lewinski@student.put.poznan.pl

Zajęcia środowe, 16:50.

Oświadczam/y, że niniejsze sprawozdanie zostało przygotowane wyłącznie przez powyższych autora/ów, a wszystkie elementy pochodzące z innych źródeł zostały odpowiednio zaznaczone i są cytowane w bibliografii.



Rysunek 1: Przykładowy wynik wyszukiwania kostek.

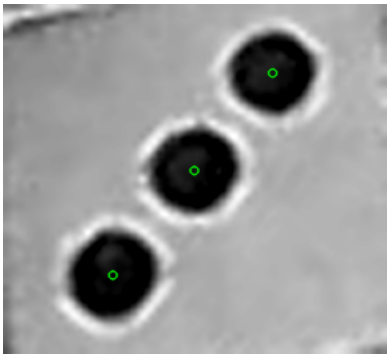
Rysunek 2: Widok perspektywiczny kostki.

1 Wstęp

Celem projektu było zaimplementowanie wykrywania liczby oczek na sześciennych kostkach przy użyciu obrazu z podłączonej kamerki. Znajdowanie oczek zostało następnie użyte do implementacji gry "kościany poker". Do uzyskiwania obrazu z kamerki i jego przekształcania użyliśmy biblioteki *OpenCV*.

2 Metoda wykrywania sześciennych kostek

Tworząc metodą wykrywania opieraliśmy się na paru ważnych założeniach. Uznaliśmy, że poszukiwane kostki są czerwone oraz zdjęcia wykonywane są pod niewielkim kątem w stosunku do powierzchni, na której znajdują się kostki. Największą napotkaną trudnością było oddzielenie kostek od tła. Początkowo chcieliśmy wykorzystać do tego funkcję `BackgroundSubtractorGMG` ale uzyskane wyniki nie były zadowalające. Następnie chcieliśmy wykorzystać algorytm *Watershed*, który na podstawie znalezionych centrów obiektów wyznaczał ich granicę. Niestety oczka w kostkach pokrywają większość powierzchni co przyczyniło się do powstania błędnych



Rysunek 3: Wynik wyszukiwania okręgów na kostce.

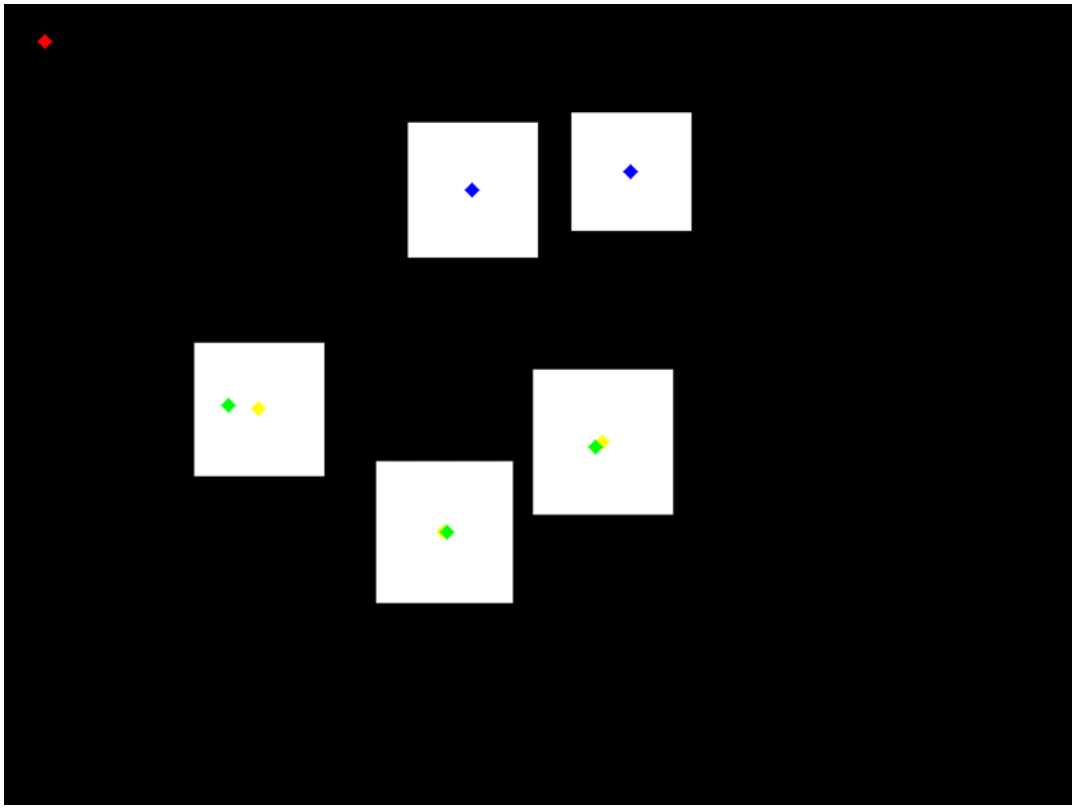
wyników (uniemożliwiały wykrycie prawidłowych środków obiektów), dlatego odrzuciliśmy to rozwiązanie. Ostatecznie utworzyliśmy algorytm, którego najważniejsze operacje przedstawiają się następująco:

1. Wczytanie obrazu.
2. Konwersja obrazu z **BGR** do **HSV** (w *OpenCV*, obraz wczytywany jest domyślnie do przestrzeni **BGR**).
3. Skalowanie kontrastu wszystkich 3 składowych
4. Pozbycie się wszystkich elementów, które nie są czerwone na podstawie wartości **HSV**).
5. Konwersja obrazu z **HSV** do **Grayscale**
6. Pozbycie się szumu za pomocą odpowiednich filtrowań.
7. Wykrycie czworokątów za pomocą funkcji `findContours`.
8. Transformacja perspektywiczna oryginalnego obrazu na podstawie uzyskanych czworokątów
9. Znalezienie na wyciętych fragmentach obrazu, obiektów zbliżonych kształtem do oczek.
10. Zwrócenie liczby zliczonych oczek.

W naszym algorytmie rozdzieliliśmy wykrywanie kostek oraz oczek, gdyż poszukiwania zniekształcały obraz. Wykrycie konturów w pierwszej kolejności oraz zastosowane przekształcenie perspektywiczne pozwoliło w łatwy sposób znaleźć oczka na fragmentach oryginalnego obrazu. Na rysunku1 czerwone czworokąty to znalezione obszary, natomiast niebieskie punkty wskazują środek figur. Na rysunku 3 zielone okręgi wskazują środki znalezionych oczek na kostce.

3 Eksperymenty

Eksperymenty przeprowadzono na podstawie 65 zdjęć o rozdzielczości 640x480 i większej, które zostały wykonane przy użyciu kamerki i telefonu komórkowego. Słaba jakość zdjęć spowodowana jest możliwościami kamerki. Dla każdego zdjęcia utworzyliśmy dodatkowy obraz



Rysunek 4: Wynik funkcji sprawdzającej poprawność działania algorytmu

testowy, na którym zaznaczone były pozycje kostek. Porównując środek znalezionych obszarów z testowymi, mogliśmy ustalić skuteczność detekcji. Na rysunku4 przedstawiony został wynik funkcji testującej, gdzie białe obszary to pozycje kostek. Zielone punkty wskazują środki prawidłowych obszarów, żółte odpowiadają znalezionym kostką, czerwone przedstawiają błędne obszary, natomiast niebieskie to kostki, które nie zostały znalezione. Otrzymane wyniki dla każdego obraz były wpisywane do macierzy pokrycia:

Macierz pokrycia3 składa się z trzech kolumn, gdzie *Prawidłowe* to znalezione kostki, kolumna *Błędne* wskazuje błędnie wskazane obszary, natomiast ostania opisuje kostki, które nie zostały prawidłowo znalezione. Dla każdego wiersza wyznaczaliśmy wartość skuteczności detekcji za pomocą wzoru:

$$\text{Skuteczność} = \frac{\text{Prawidłowe - Błędne}}{\text{Prawidłowe + Pozostałe}}$$

Następnie obliczyliśmy wartość średnią z obliczonych wyników. Na podstawie uzyskanych wartości mogliśmy określić czy wykorzystane parametry pozwalają nam najefektywniej poszukiwać kostek. Początkowo skuteczność programu wynosiła 65%. Następnie poszukując najlepszych parametrów dla poszczególnych funkcji, uzyskaliśmy skuteczność rzędu $83,54\%$.

lp	Prawidłowe	Błędne	Pozostałe
1.	0	0	1
2.	1	0	0
3.	1	0	0
4.	5	0	0
5.	1	0	0
6.	1	0	0
7.	1	0	0
8.	1	0	0
9.	1	0	0
10.	1	0	0
11.	0	0	1
12.	1	0	0
13.	1	0	0
14.	1	0	0
15.	1	0	0
16.	1	0	0
17.	1	0	0
18.	1	0	0
19.	1	0	0
20.	1	0	0
21.	0	0	1
22.	3	4	2
23.	4	2	1
24.	4	0	1
25.	5	0	0
...	...	...	...
44.	4	0	1
45.	4	0	1
46.	5	0	0
47.	2	0	3
48.	5	0	0
49.	2	0	3
50.	5	0	0
51.	5	0	0
52.	5	0	0
53.	3	1	2
54.	1	0	0
55.	5	0	0
56.	5	0	0
57.	4	0	1
58.	4	1	1
59.	5	1	0
60.	3	11	2
61.	5	0	0
62.	5	0	0
sum	188.0	24.0	34.0

Tabela 1: Przykładowa macierz pokrycia