

# Kółko i krzyżyk

## Rozpoznawanie stanu gry

Natalia Szymczyk      Jan Świątek  
145250                  145390

07.12.2021

## 1 Wstęp

Projekt został zrealizowany w ramach przedmiotu Komunikacja Człowiek-Komputer. Jego tematyka jest związana z przetwarzaniem obrazu. Aplikacja ma na celu rozpoznawanie stanu gry w kółko i krzyżyk. Została zaimplementowana w języku Python z wykorzystaniem biblioteki OpenCV.

## 2 Działanie programu

Za pomocą odpowiedniej aplikacji przesyłamy obraz nagrywany aparatem telefonu do naszego programu. Robimy to, aby mieć możliwość pracy w trybie rzeczywistym. Telefon pełni funkcję kamery sieciowej.

Automatyczne rozpoznawanie stanu gry następuje po naciśnięciu klawisza Enter dla obecnie udostępnianego obrazu. Na początku wykonujemy odpowiednie przetwarzanie obrazu, aby uzyskać jak najdokładniejsze kontury rozgrywki. Zrzuty ekranów po poszczególnych funkcjach zostały umieszczone w dalszej części sprawozdania.

Następnie dla tak przetworzonego obrazu wywołujemy funkcję „find\_boards” w celu znalezienia pojedynczych planszy gry. Na tym etapie szukamy konturów. Rozgrywka powinna mieć miejsce na planszy, która jest w przybliżeniu czworokątem. Testowaliśmy przypadki, gdzie na zdjęciu znajdują się również inne elementy, które także mogą zostać wyłapane, dlatego staramy się ograniczyć kontury jedynie do plansz.

Na znalezionych planszach program wykonuje dalsze działania. Znajdowany jest kąt, pod którym uwieczniona została gra i następnie cała plansza jest obracana tak, aby jej linie były równoległe (lub prostopadłe).

Dla tak przygotowanej planszy wywoływana jest funkcja „find\_tiles”, której celem jest rozpoznanie kształtów na 9 możliwych pozycjach. Za pomocą centroidów rozpoznajemy pozycje komórek wraz ze znakiem, a na koniec ustalamy i wypisujemy wynik rozgrywki.

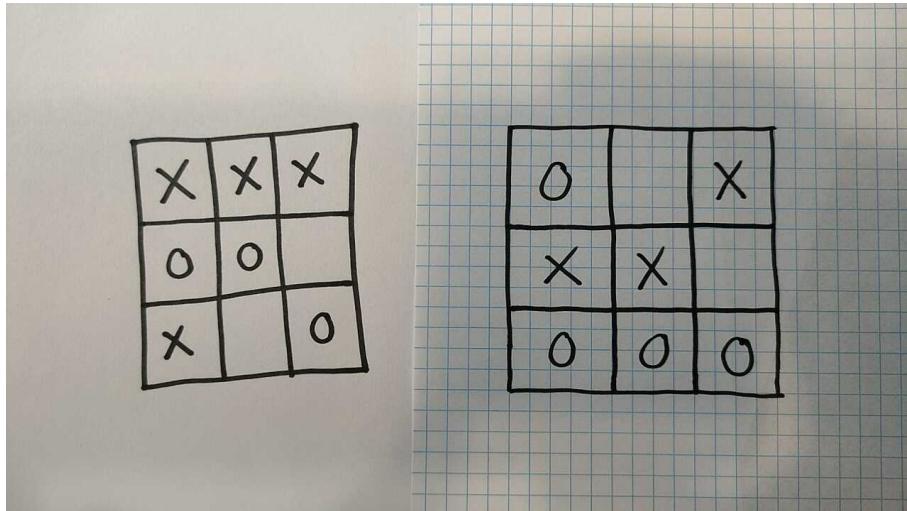
### 3 Szczegółowy opis poszczególnych kroków

#### 3.1 Wczytywanie obrazu

Obraz jest udostępniany przez telefon pod konkretnym adresem url, który przekazujemy do naszej aplikacji i odczytujemy z wykorzystaniem wbudowanej funkcji z biblioteki OpenCV.

```
url = 'http://192.168.1.77:8080/video'  
...  
response, img = cv2.VideoCapture(url).read()
```

Adres IP oczywiście zmienia się w zależności od połączenia z siecią.

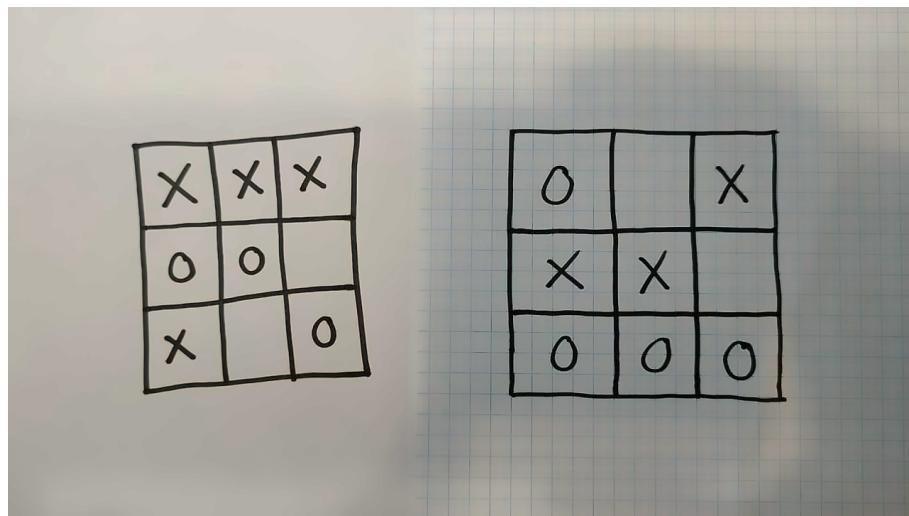


#### 3.2 Przetwarzanie obrazu

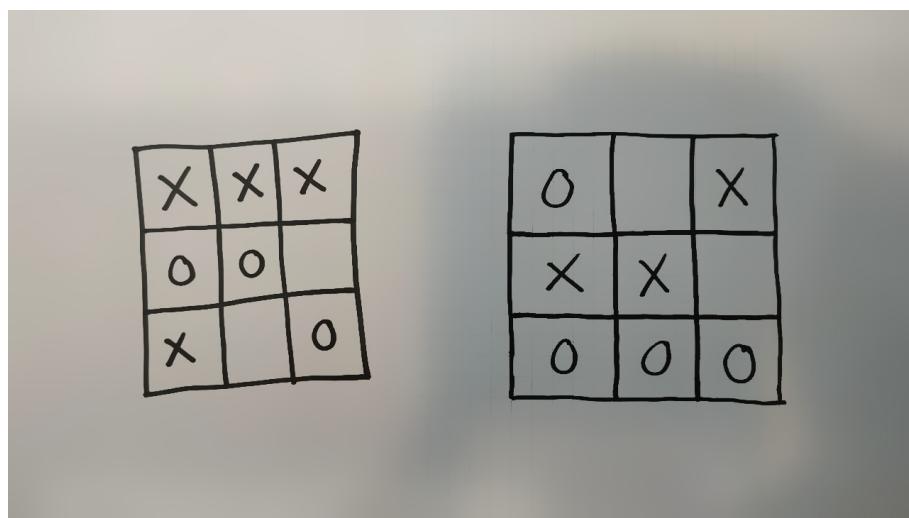
Przetestowaliśmy wiele możliwości i najlepsze wyniki daje poniższe przetwarzanie obrazu:

```
img = cv2.fastNlMeansDenoisingColored(img, None, 10, 10, 7, 21)  
img = cv2.bilateralFilter(img, 75, 75, 75)  
img = channel_selection(img)  
img = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_MEAN_C,  
                           cv2.THRESH_BINARY_INV, 33, 3)  
img = cv2.morphologyEx(img, cv2.MORPH_CLOSE, np.ones((2, 2), np.uint8))
```

Na początku wykonujemy odszumianie obrazu w celu pozbycia się kratek oraz drobnych niedoskonałości.



Następnie rozmywamy obraz pozostawiając krawędzie ostrymi.

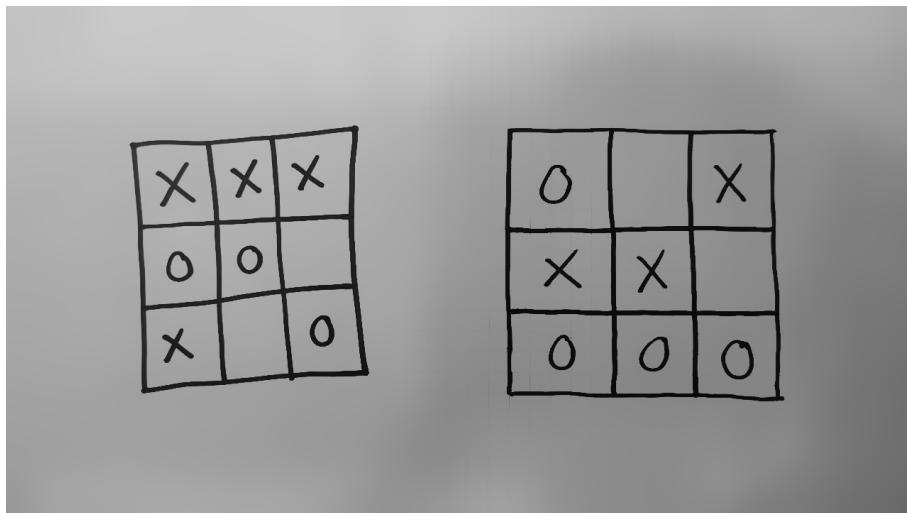


Kolejnym krokiem jest wybór kanału, na którym obraz jest najciemniejszy. Do tego celu stworzyliśmy własną funkcję:

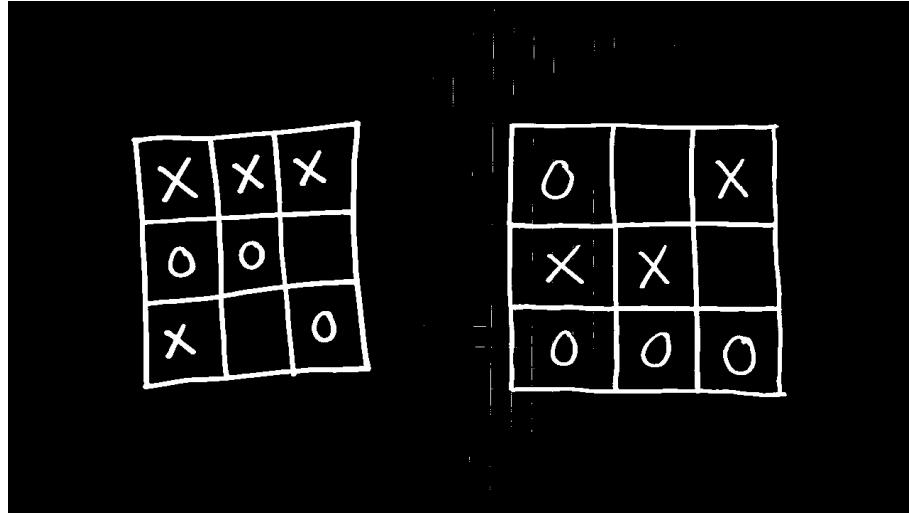
```
def channel_selection(img):
    channel_r = img[:, :, 0]
    channel_g = img[:, :, 1]
    channel_b = img[:, :, 2]
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    means = [
        np.mean(channel_r),
        np.mean(channel_g),
        np.mean(channel_b),
        np.mean(gray)
    ]

    if min(means) == means[0]:
        return channel_r
    elif min(means) == means[1]:
        return channel_g
    elif min(means) == means[2]:
        return channel_b
    else:
        return gray
```



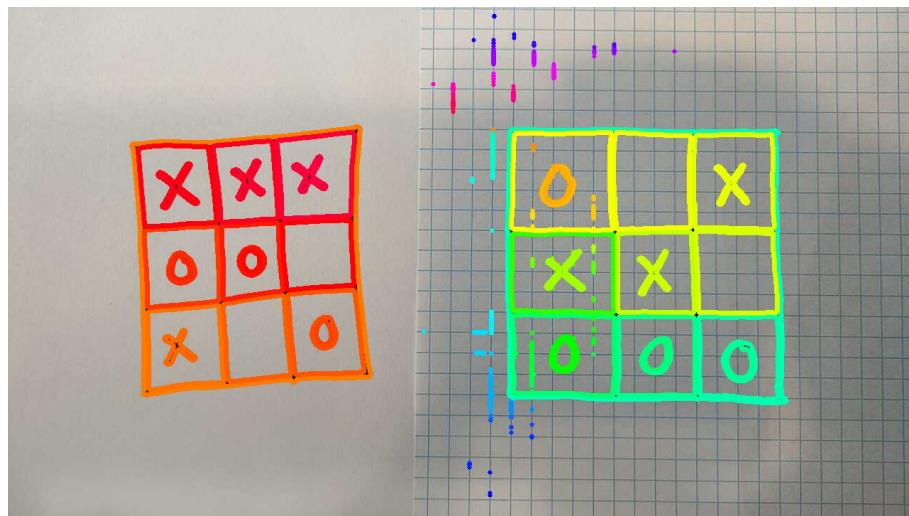
Ostatnim krokiem w procesie przygotowania obrazu do dalszej pracy jest progowanie i morfologia. Najbardziej satysfakcyjne wyniki uzyskaliśmy wykorzystując progowanie adaptacyjne, które określa indywidualny próg dla piksela na podstawie małego regionu wokół niego. Otrzymujemy więc różne progi dla różnych obszarów tego samego obrazu, co daje lepsze wyniki dla obrazów o różnym oświetleniu.



Na planszy, która została narysowana na kartce w kratkę widać parę niedoskonałości, których nie jesteśmy w stanie wyeliminować. Jednak nie powinny one wpływać na końcowe wyniki. Status gry i tak został odczytany poprawnie.

### 3.3 Znajdowanie planszy

Po odpowiednim przetworzeniu obrazu wyszukujemy kontury na całym zdjęciu. Wszystkie rozpatrywane kontury widoczne są na poniższym zdjęciu:

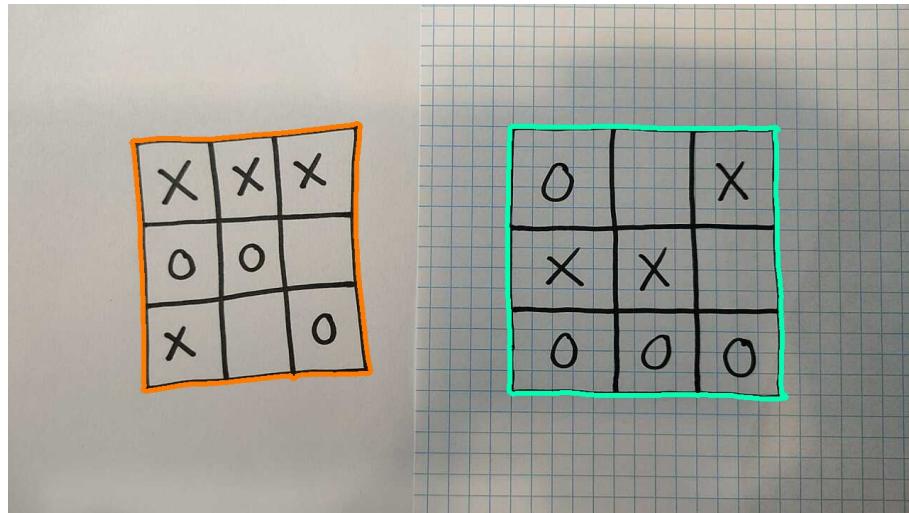


Każdy kontur został pomalowany innym kolorem.

Kontur zostaje sklasyfikowany jako plansza, gdy spełni następujące warunki:

- Obszar musi być większy od określonej stałej,
- Aproxymowany kształt konturu musi być czworokątem,
- Kontur musi posiadać co najmniej kilku potomków.

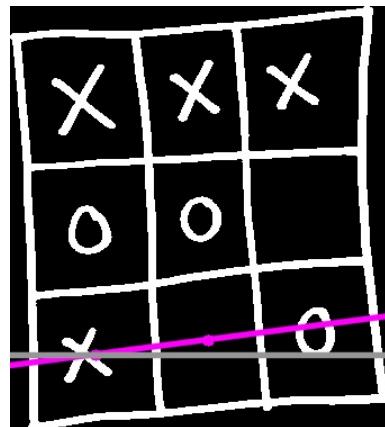
Program będzie więc rozpatrywał jedynie następujące kontury:



Dalsze funkcje pracują jedynie na wycinkach zdjęcia z odpowiednim konturem.

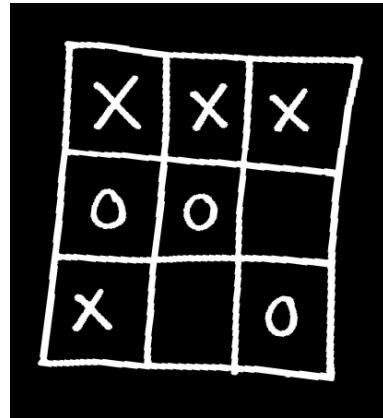
### 3.4 Przygotowywanie planszy

Na wycinku planszy szukamy 9 „kafelków” i wyznaczamy ich centroidy. Następnie znajdujemy współczynnik kierunkowy funkcji liniowej przechodzącej przez centroidy dwóch pierwszych pozycji. Na podstawie współczynnika jesteśmy w stanie wyznaczyć kąt, o który pochylona jest funkcja w stosunku do osi OX.



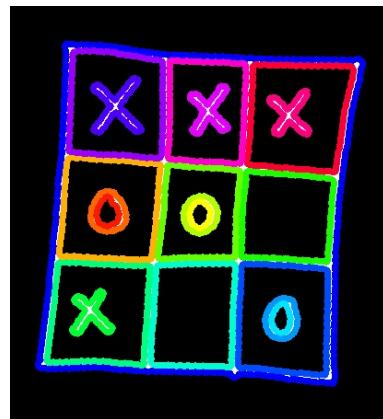
Różowa linia przedstawia aproksymację pochylenia planszy. Całą planszę będziemy obracać o kąt między wyznaczonymi prostymi.

Następnie powiększamy zdjęcie (dodajemy obramowanie do wyciętej planszy), aby nie stracić żadnych danych po obróceniu zdjęcia.



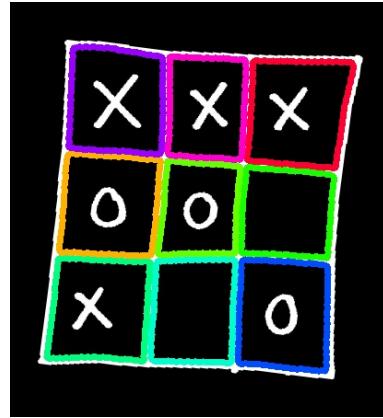
### 3.5 Rozpoznawanie kafelków

Na początku szukamy konturów dla odpowiednio przygotowanej planszy.



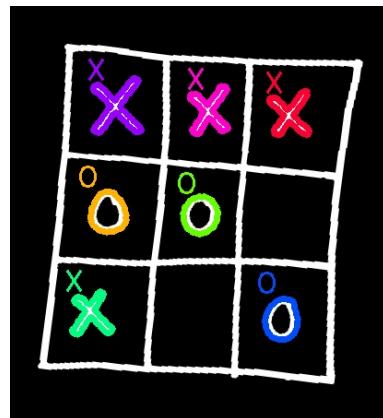
Jak widać na zdjęciu, największym konturem jest cała plansza. Jako kontury zostały także zaznaczone oddzielnie kafelki oraz opcjonalne w nich znaki. Naszym celem jest teraz przefiltrowanie otrzymanych konturów, aby uzyskać jedynie kafelki.

Obliczamy i sortujemy pola wszystkich konturów oraz na ich podstawie wybieramy 9, które będą analizowanymi przez nas kafelkami. Dla każdego kafelka wywołujemy funkcję „approxPolyDP”, która aproksymuje kształt konturu. Jeśli figura okaże się czworokątem oraz jej obszar pasuje do wcześniej wybranych obszarów, zaczynamy go analizować.



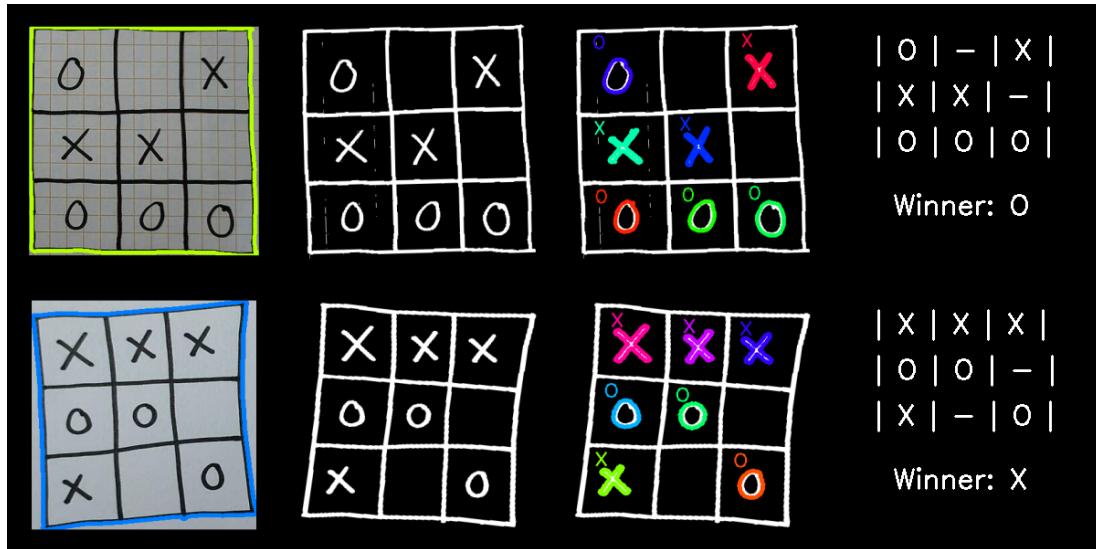
### 3.6 Rozpoznawanie kształtów

Najpierw na podstawie hierarchii konturów sprawdzamy, czy dany kontur ma jakiegoś potomka. Jeśli nie, to wiadomo, że na danej pozycji nie ma żadnego znaku. Natomiast jeśli kontur posiada potomka, musimy zweryfikować czy jest nim kółko czy krzyżyk. Do określenia tego używamy współczynnika „solidity”. Jest to stosunek obszaru konturu do obszaru jego powłoki wypukłej. Dla kółka (O) - będzie one zbliżone do 1. Dla krzyżka (X) - powinno być mniejsze niż 0,5 (w naszej aplikacji daliśmy inną wartość, wyznaczoną empirycznie na podstawie naszych przykładów).



### 3.7 Wypisywanie wyników

Plansza zostaje odwzorowana na listę dwuwymiarową, gdzie poszczególne kafelki zostają umieszczone zgodnie z ich pozycją w posortowanej liście centroidów. Wyniki i ostateczny stan dla każdej planszy wypisujemy w konsoli oraz umieszczamy na podsumowaniu, które wyświetlamy w postaci graficznej.



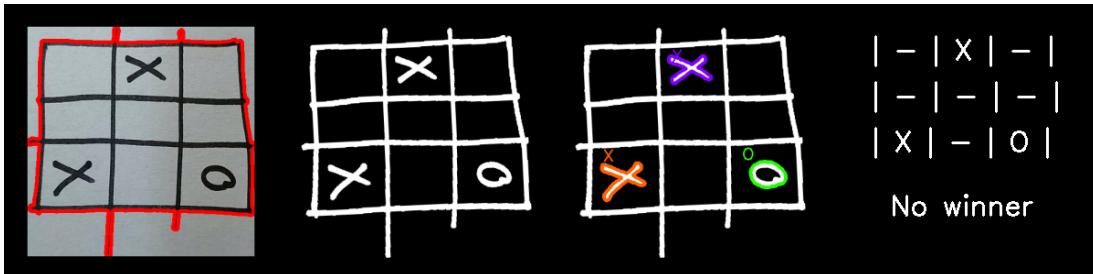
## 4 Przykłady

Wykonaliśmy wiele testów, żeby sprawdzić działanie naszego algorytmu. Do sprawozdania rozdzieliliśmy je na różne kategorie. Na początku pokazane jest działanie programu na markerze. Jest to narzędzie do pisania, które nasz algorytm wyłapuje najsukuteczniej. Dalej pokazane są plansze pisane cienkopisem. Skuteczność algorytmu w tym przypadku jest trochę słabsza, jednak dalej radzi sobie bardzo dobrze. Dla każdego rodzaju pisadła wykonaliśmy testy zarówno na białej kartce, jak i kartce w kratkę. Na końcu przygotowaliśmy także kategorię z utrudnieniami. Pokazuje ona słabsze strony naszego algorytmu, które oczywiście omówimy.

## 4.1 Marker

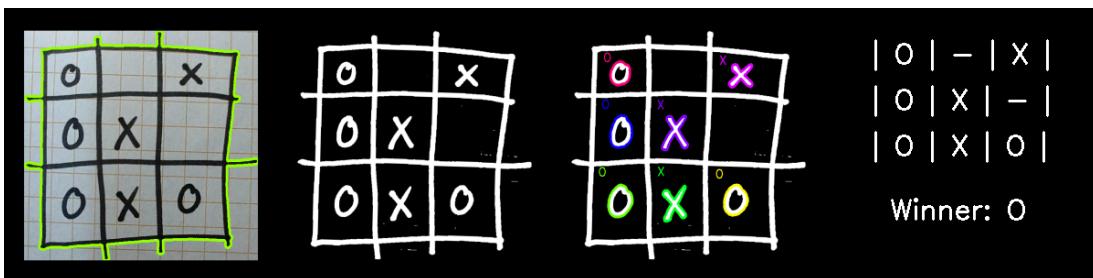
Na początku wykonaliśmy testy na pojedynczych planszach.

### 4.1.1 Biała kartka



Z pojedynczą planszą narysowaną markerem algorytm radzi sobie bez problemu.

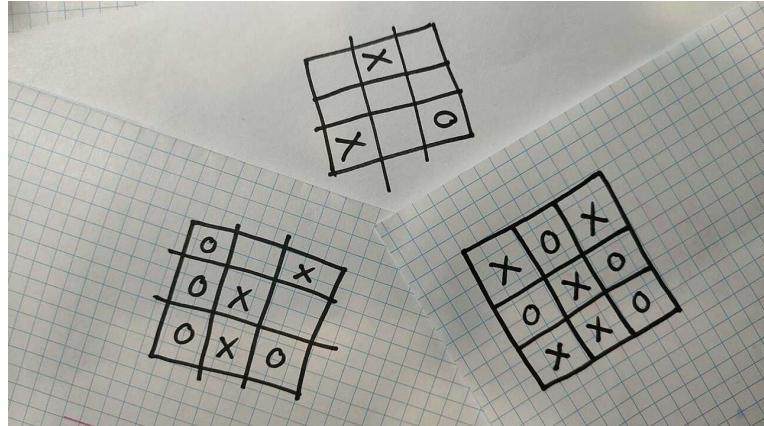
### 4.1.2 Kartka w kratkę



Na kartce w kratkę marker również jest bardzo dobrze widoczny, program radzi sobie z praktycznie każdym wyraźnym przykładem.

#### 4.1.3 Kilka plansz

Oryginalny kadr wyglądał następująco:

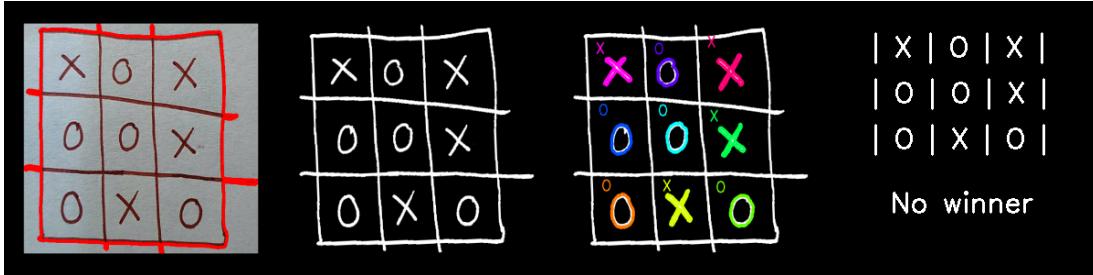


			$\begin{array}{ c c c } \hline 0 & - & X \\ \hline 0 & X & - \\ \hline 0 & X & O \\ \hline \end{array}$ <p>Winner: O</p>
			$\begin{array}{ c c c } \hline X & O & X \\ \hline 0 & X & 0 \\ \hline X & X & O \\ \hline \end{array}$ <p>Winner: X</p>
			$\begin{array}{ c c c } \hline - & X & - \\ \hline - & - & - \\ \hline X & - & O \\ \hline \end{array}$ <p>No winner</p>

Jak widać, plansze zostały odczytane idealnie. Algorytm prawidłowo zaznaczył kontury każdej planszy, indywidualnie obrócił o odpowiedni kąt oraz rozpoznał stan gry.

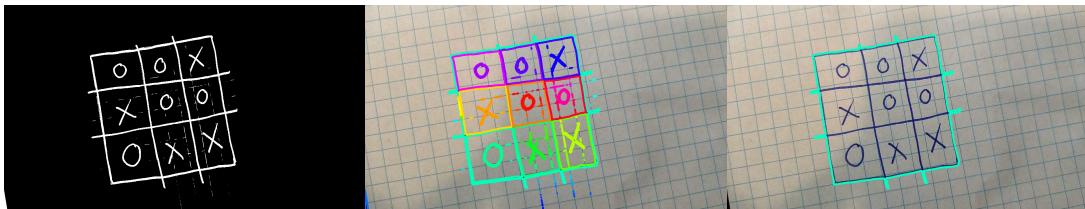
## 4.2 Cienkopis

### 4.2.1 Biała kartka

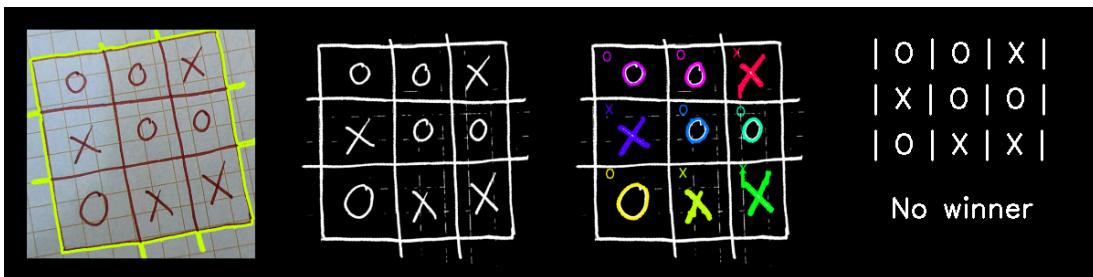


Na gładkiej kartce cienkopis jest bardzo dobrze odczytywany.

### 4.2.2 Kartka w kratkę



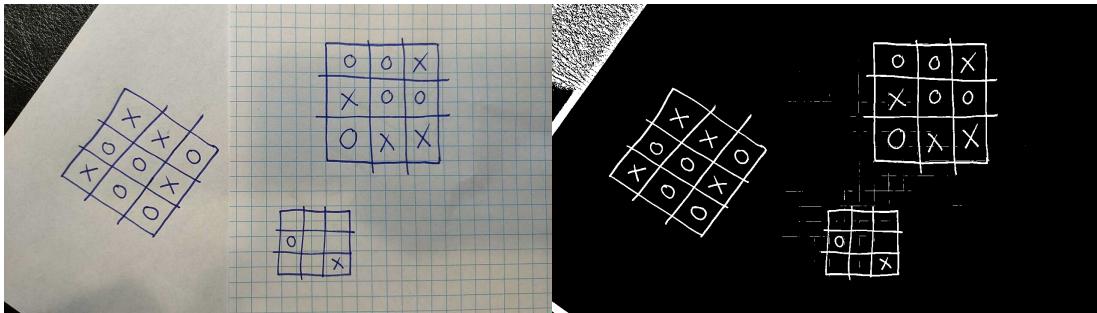
Na zdjęciu widać, że nawet po przetwarzaniu obrazu, kratki z kartki są lekko widoczne. Zostają one także wyłapane jako kontury. Jednak dzięki ograniczeniom na klasyfikację plansz, kontury planszy zostają wyłapane prawidłowo.



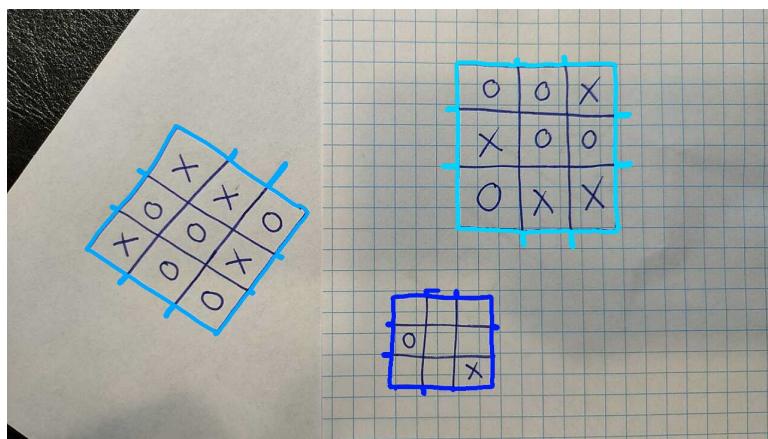
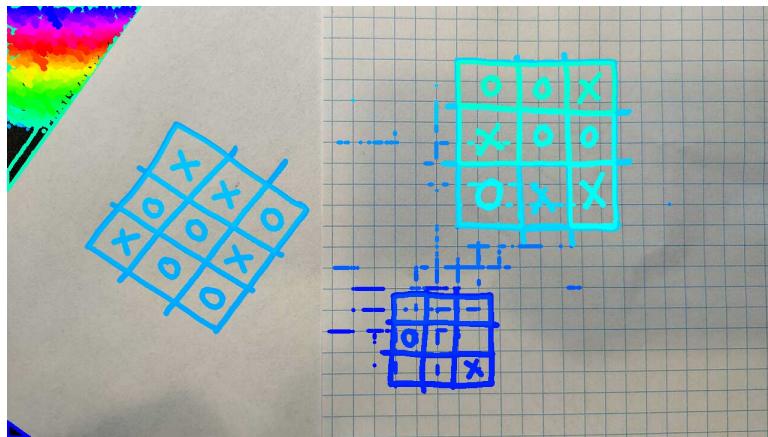
Algorytm, mimo drobnych niedoskonałości w preprocessingu, prawidłowo odczytał stan gry.

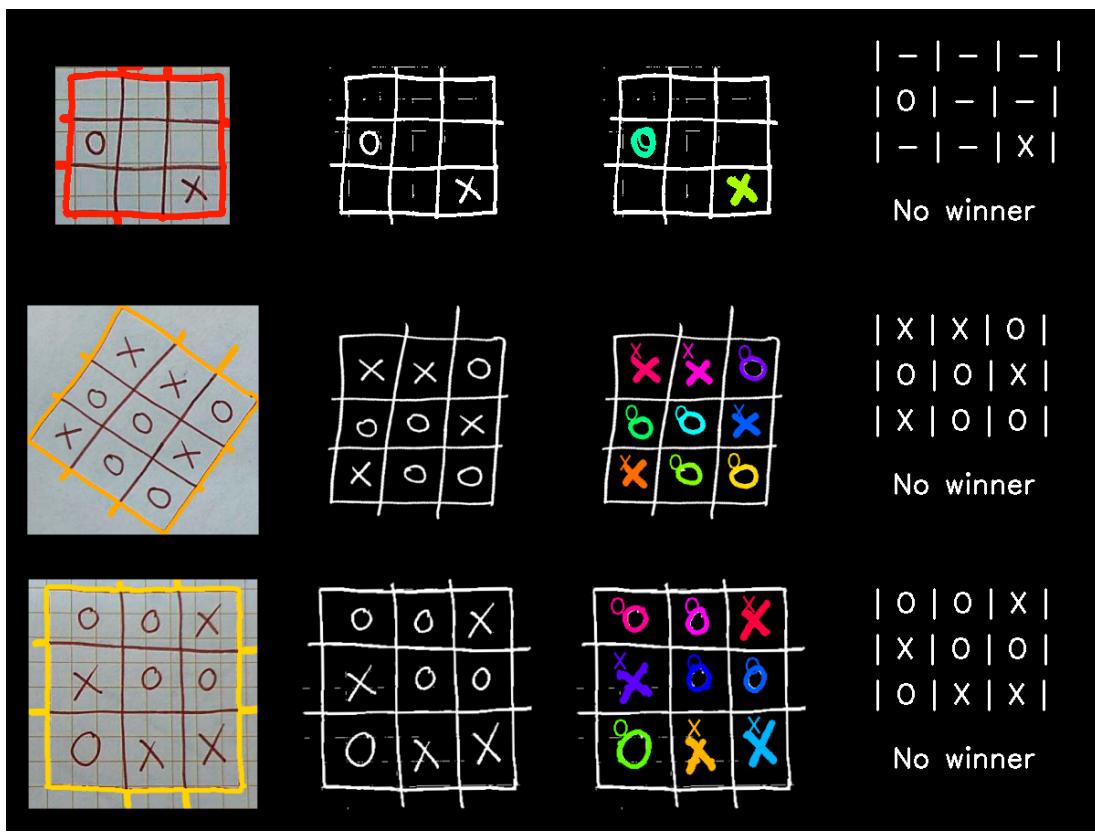
#### 4.2.3 Kilka plansz

Przygotowaliśmy przykład z 3 planszami oraz różnymi podłożami. Oryginalny kadr oraz screeny z przetwarzania wyglądają następująco:



Na zdjęciu z lewej strony został ukazany także kawałek teczki. Widać to mocno na zdjęciu po progowaniu oraz na zdjęciu z wszystkimi konturami. Jednak znowu algorytm poprawnie zaznaczył same plansze, co widać na ostatnim przykładzie.

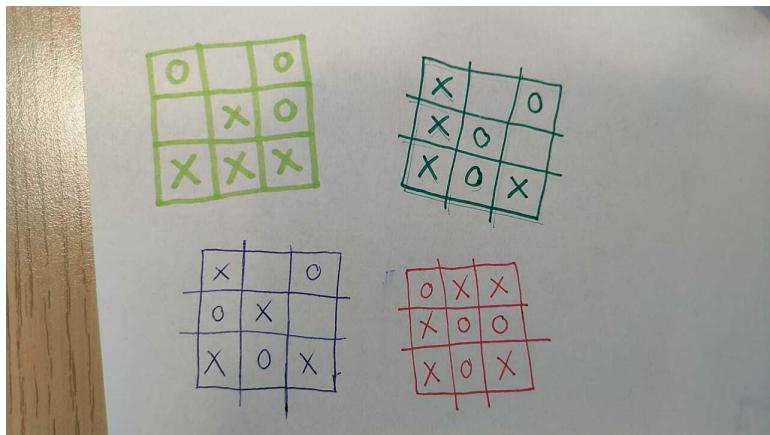


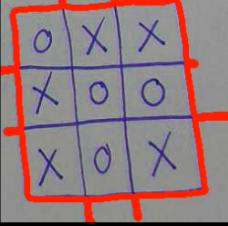
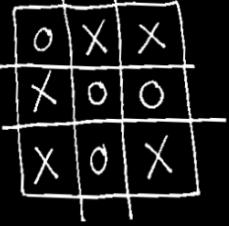
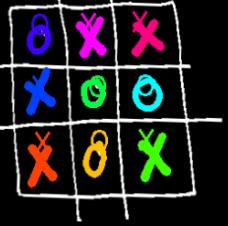
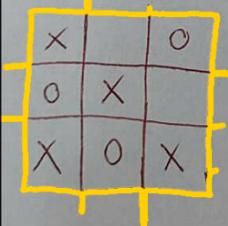
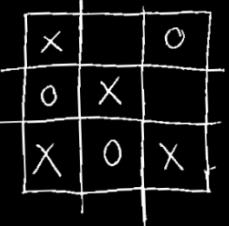
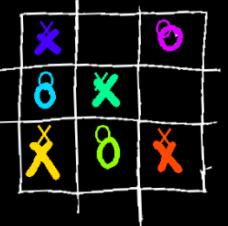
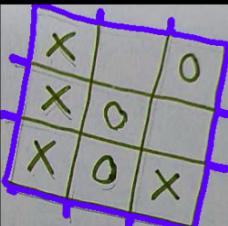
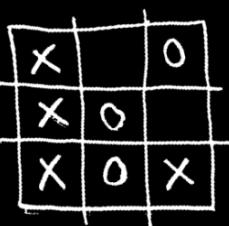
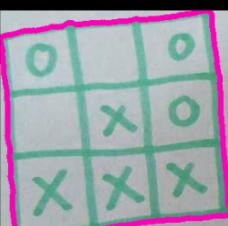
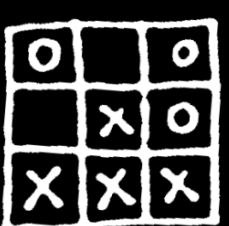


### 4.3 Różne narzędzia do pisania

Do tych testów wykorzystaliśmy długopis, kolorowe cienkopisy oraz jasny zielony marker.

#### 4.3.1 Biała kartka

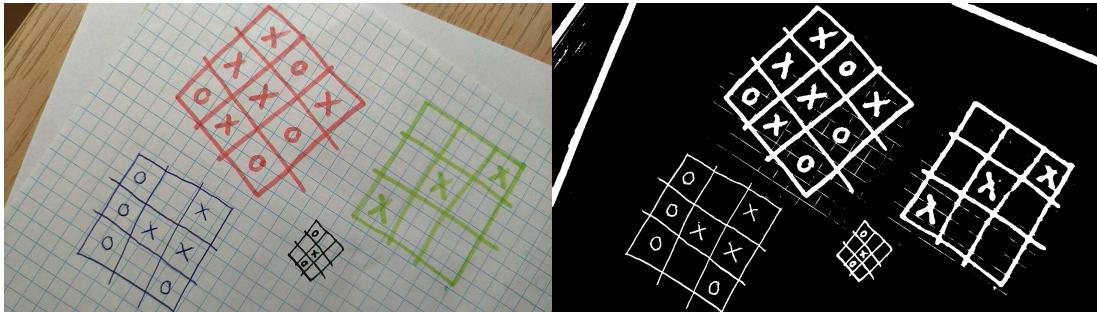


			O   X   X     X   O   O     X   O   X   No winner
			X   -   O     O   X   -     X   O   X   Winner: X
			X   -   O     X   O   -     X   O   X   Winner: X
			O   -   O     -   X   O     X   X   X   Winner: X

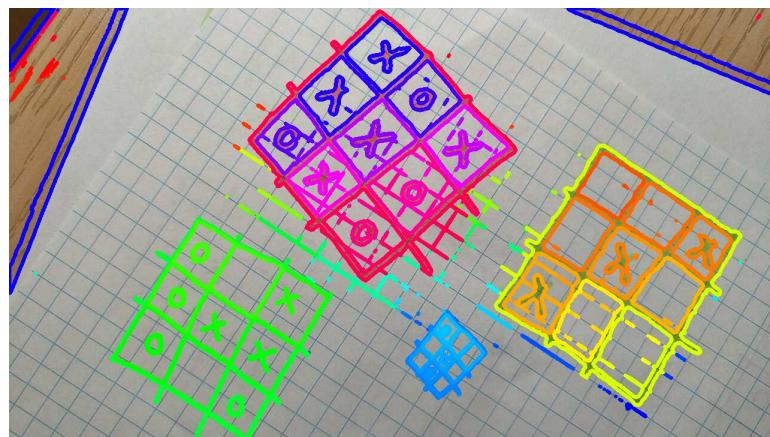
Na białej kartce nawet jasny marker został prawidłowo odczytany. Jednak zdarzyło się, że przy gorszym świetle program zwracał dla niego nie do końca prawidłowe wyniki.

#### 4.3.2 Kartka w kratkę

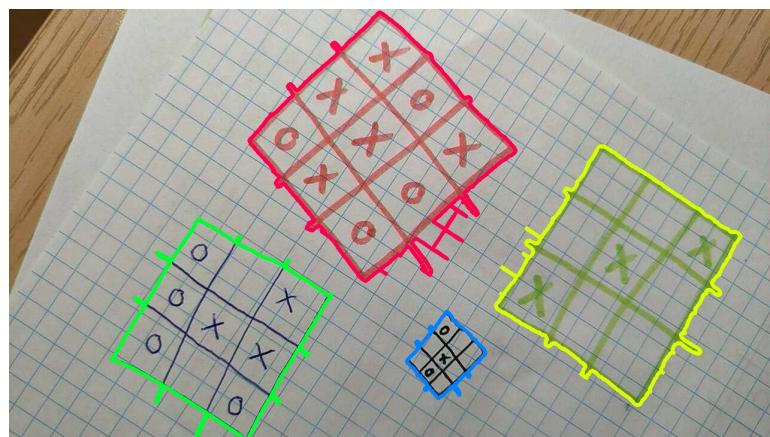
Oryginalny kadr oraz obraz po przetworzeniu prezentują się następująco:

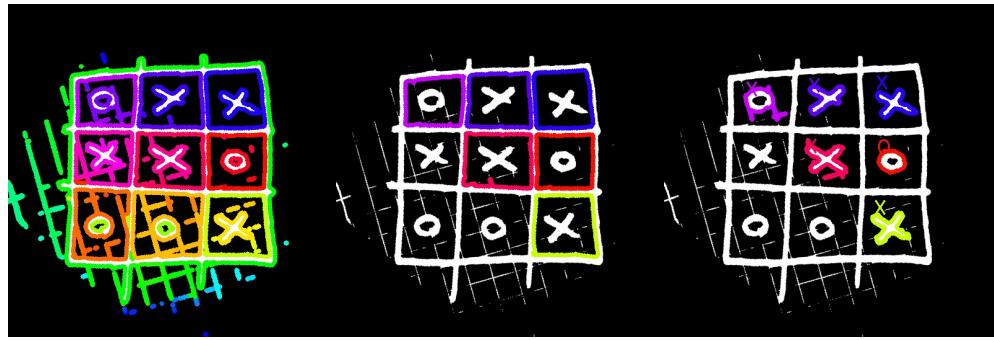


Checieliśmy umieścić te 4 plansze w kadrze, dlatego musieliśmy wyżej trzymać telefon, z czym wiąże się słabsza rozdzielcość. Z tego względu kontury zostały wyłapane z pewnymi problemami, zwłaszcza dla planszy narysowanej jasnymi pisakami.

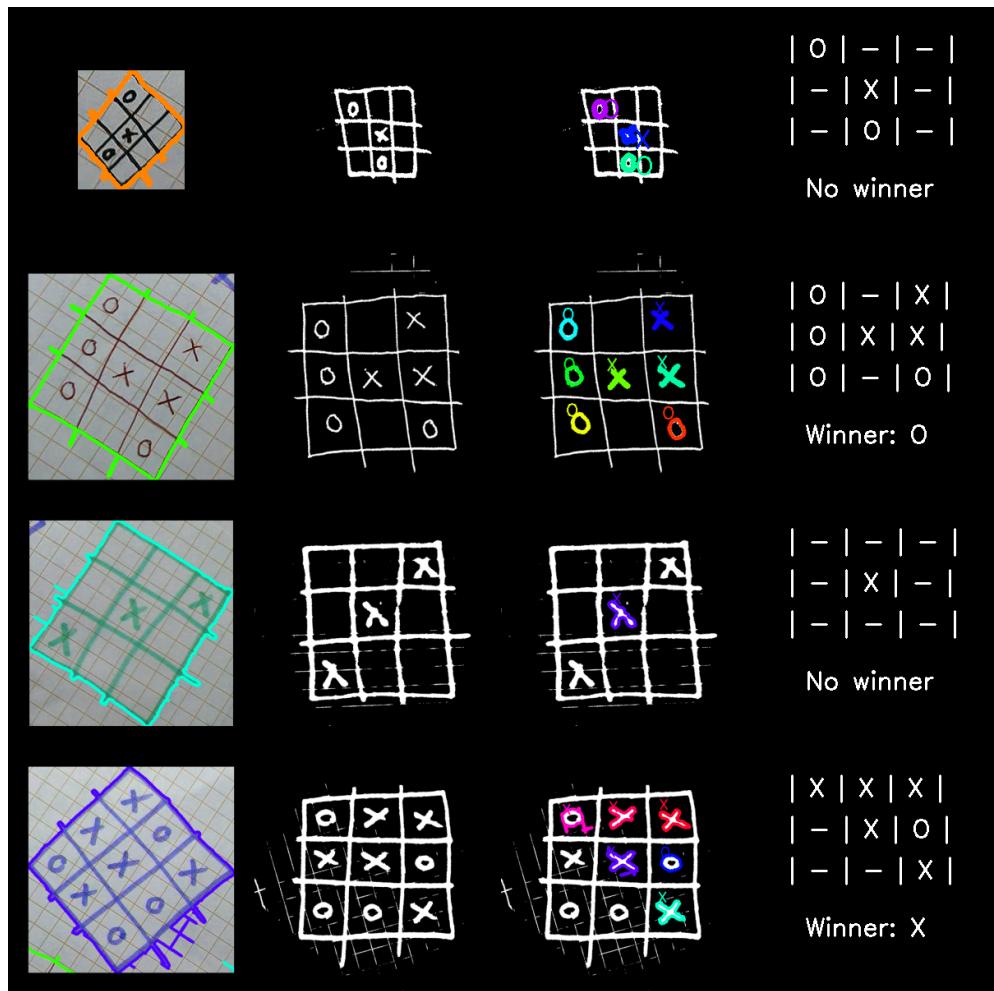


Mimo wszystko, kontury samych planszy zostały prawidłowo zaznaczone.





Powyżej zostały ukazane poszczególne operacje algorytmu na planszy narysowanej rózowym pisakiem. Ze względu na duży szum, który spowodowały kratki, kafelki nie zostały prawidłowo odczytane. Wiąże się to z tym, że algorytm nie rozpoznał znaków na tych konkretnych pozycjach. Zwrócony wynik jest więc niepełny. Bardzo podobnie wygląda sytuacja dla planszy narysowanej zielonym markerem. Także nie wszystkie znaki zostały rozpoznane. Co ciekawe, dla najmniejszej planszy algorytm zwrócił prawidłowy wynik, czego się nie spodziewaliśmy.

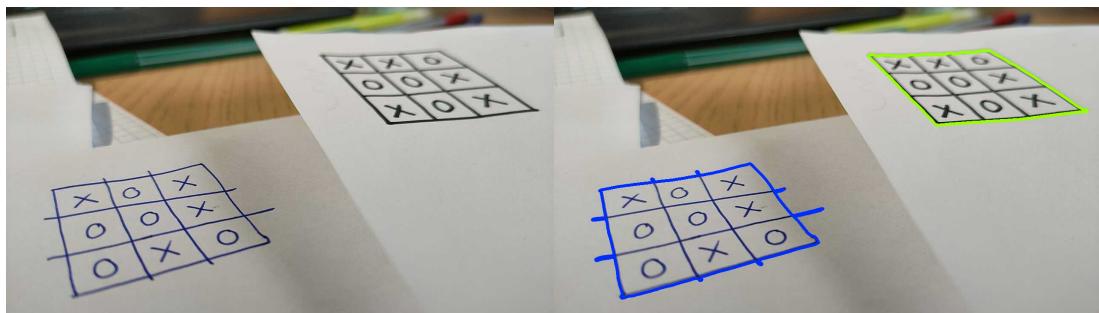


## 4.4 Utrudnienia

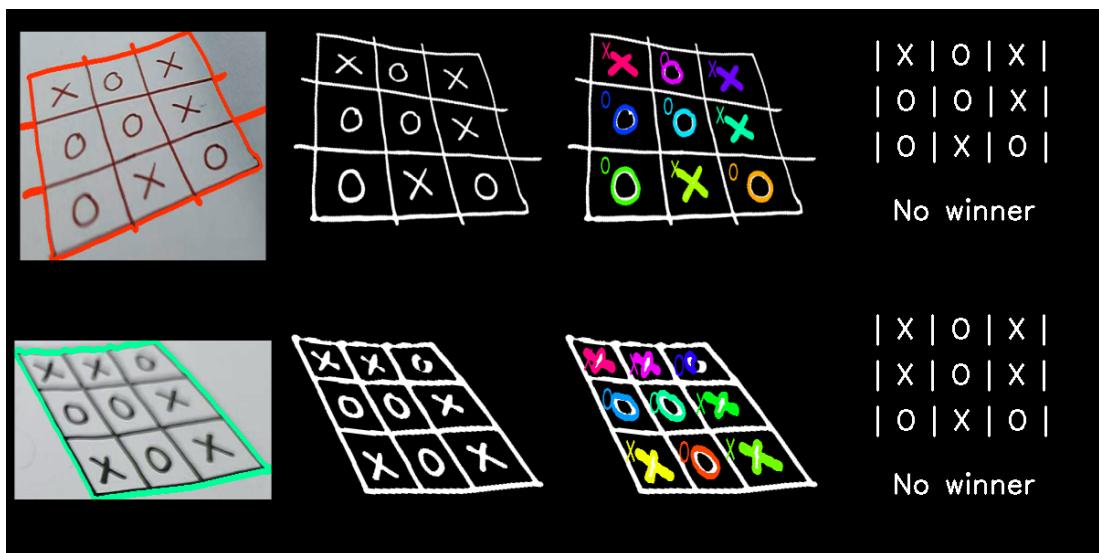
### 4.4.1 Zdjęcie pod kątem

#### 4.4.1.1 Biała kartka

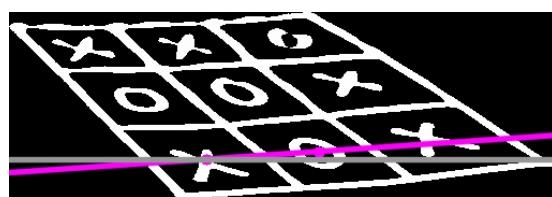
Jako pierwsze utrudnienie wykonaliśmy zdjęcie pod kątem, dodatkowo jedna kartka jest kilka centymetrów ponad drugą, co jeszcze bardziej pogarsza widoczność planszy.



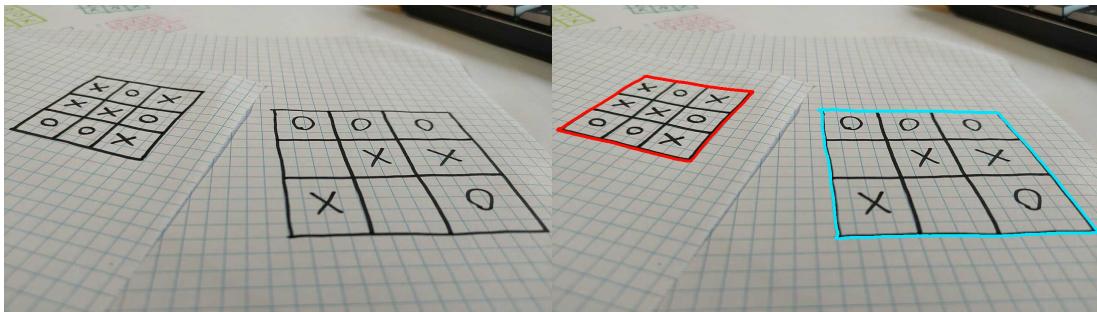
Kontury planszy zostały odczytane prawidłowo. Nathirdem zdjęciu w podsumowaniu widać, że znaki także zostały dobrze rozpoznane. Jednak wynik wypisany jest nieprawidłowy.



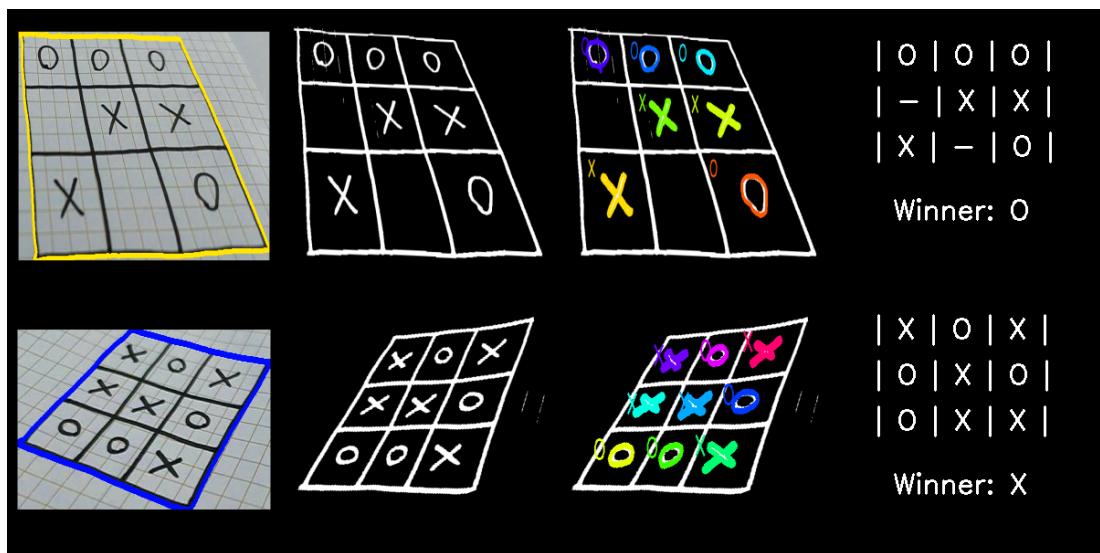
Jest to związane z niestandardowym położeniem centroidów kafelków. Symbole w kafelkach musiały się nadpisać, ponieważ funkcja „find\_index” zwróciła nieprawidłową pozycję kafelka.



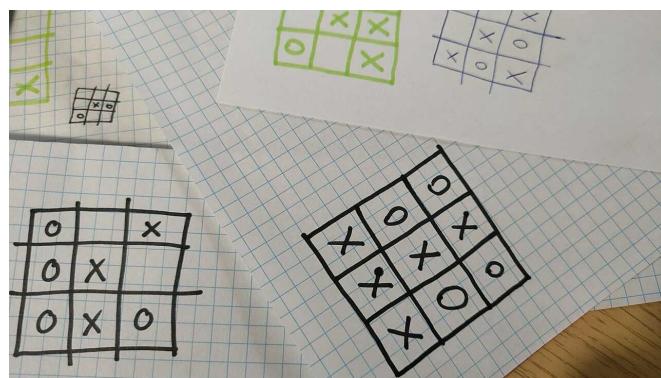
#### 4.4.1.2 Kartka w kratkę

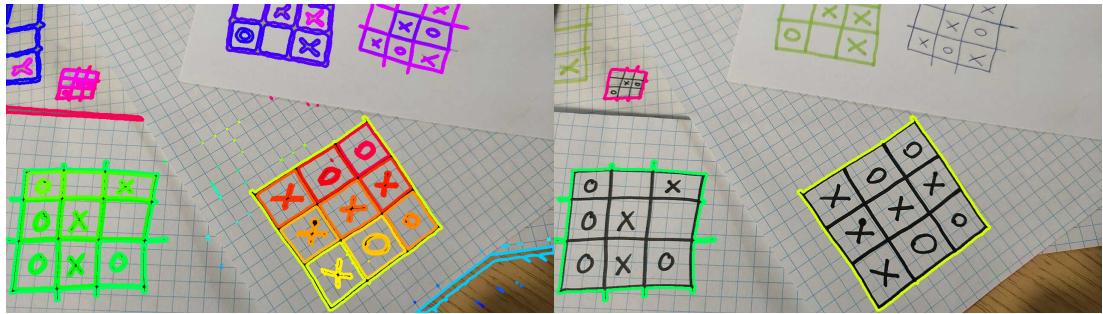


Tutaj również jedna z plansz nie została prawidłowo rozpoznana.

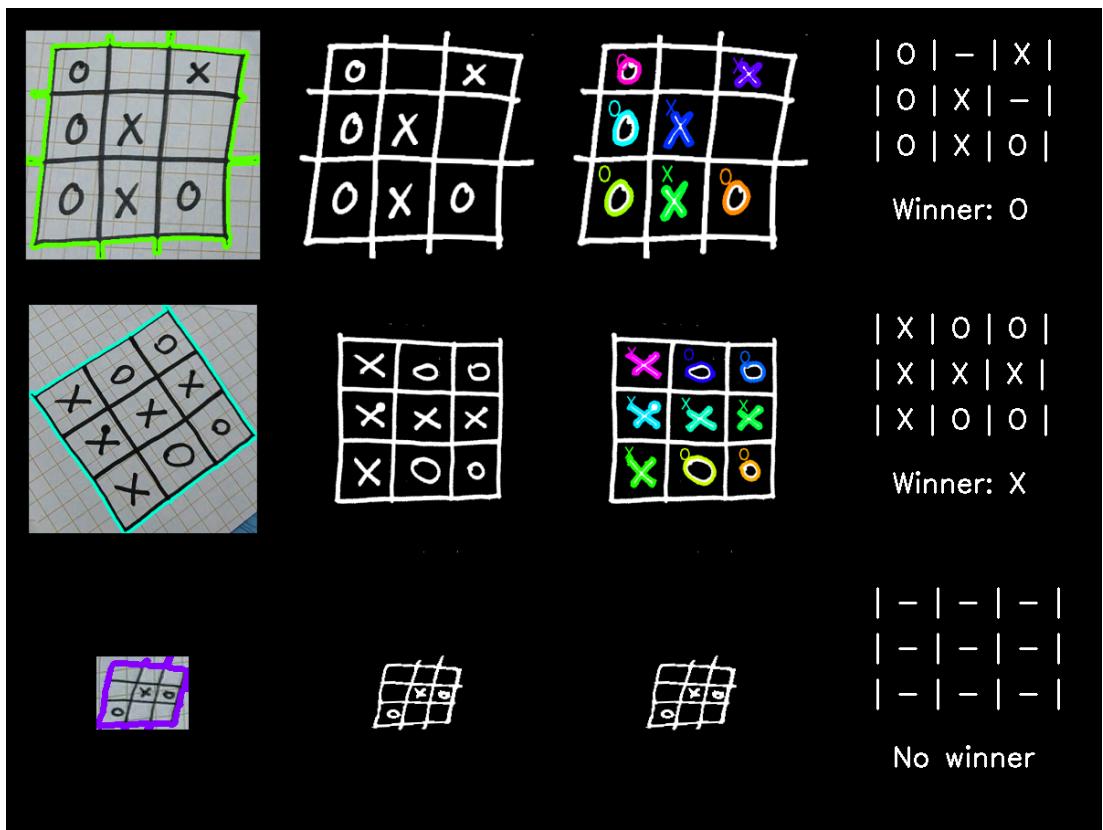


#### 4.4.2 Ucięte plansze





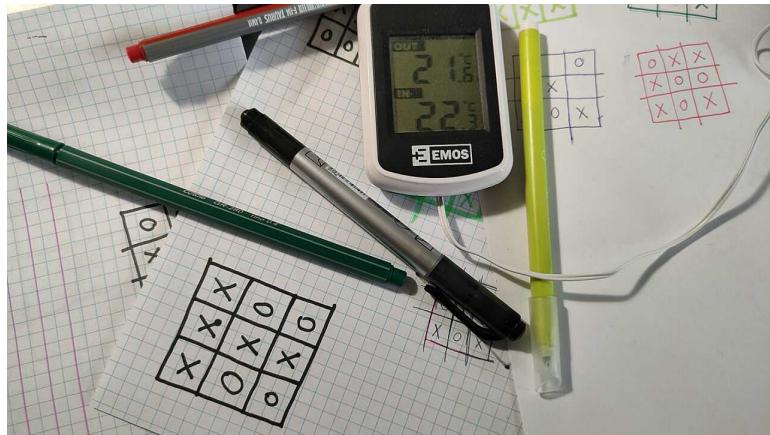
Na powyższym zdjęciu widać, że algorytm wykrył bardzo dużo konturów, także tych uciętych plansz. Jednak jako ostateczne plansze zwrócił jedynie te, które były w 100% widoczne.



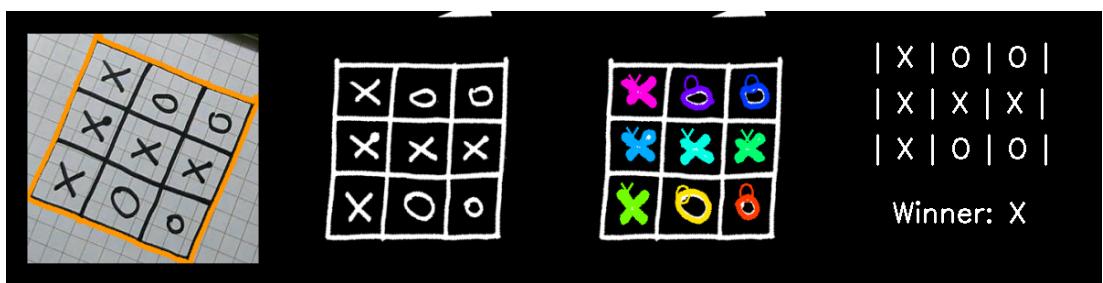
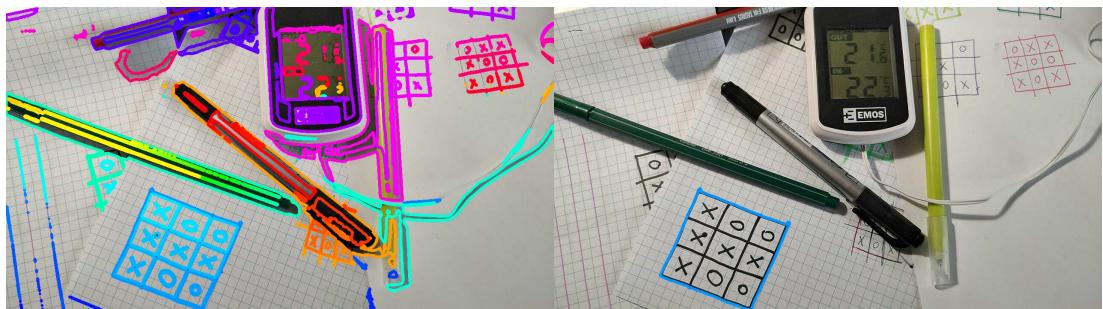
Algorytm zwrócił nieprawidłowy wynik dla ostatniej planszy. Wiąże się to z jej wielkością, jakość zdjęcia jest za niska, żeby znaki w kafelkach były dla programu czytelne.

#### 4.4.3 Inne elementy na zdjęciu

Teraz chcieliśmy utrudnić zadanie programowi stawiając w kadrze zbędne obiekty. Wybrany przez nas kadr prezentuje się następująco:

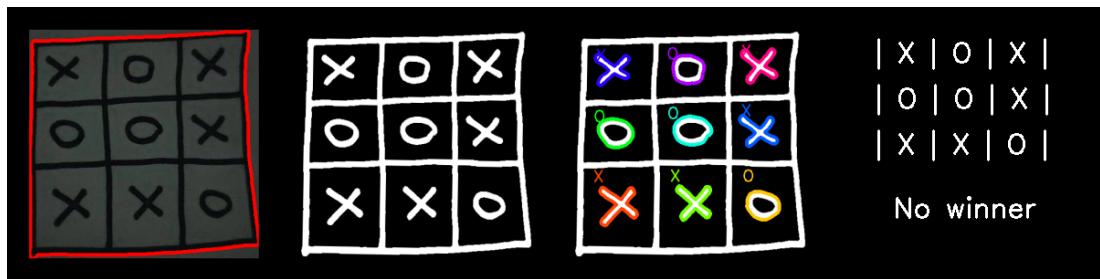
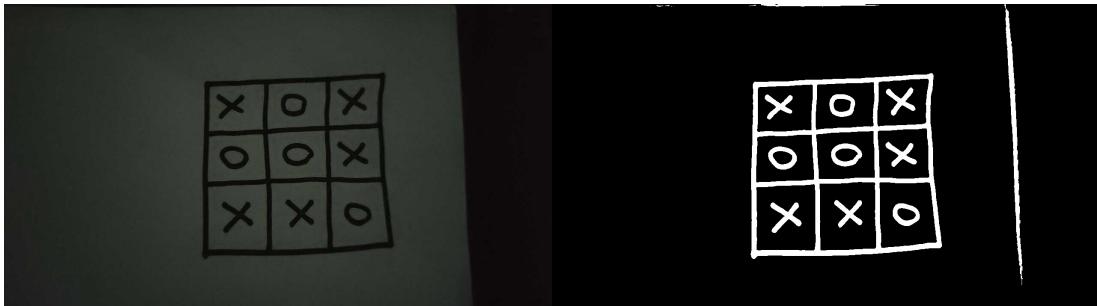


Na oryginalnym zdjęciu dosyć dobrze widoczne są dwie plansze. Pierwsza z przodu narysowana markerem, druga z tyłu narysowana różowym pisakiem. Po przetworzeniu zdjęcia, kontury tylnej planszy mocno się rozmyły. Algorytm nie potrafił przez to wyłapać tej planszy.

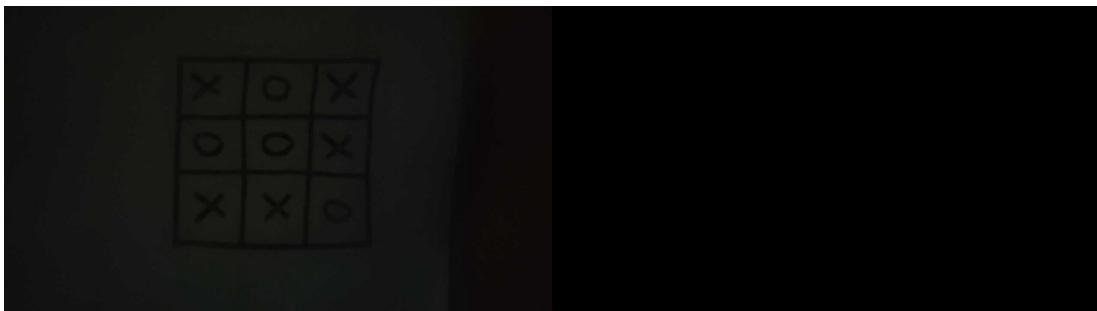


#### 4.4.4 Słabe warunki oświetleniowe

Zdjęcie wykonaliśmy w pomieszczeniu, gdzie jedynym źródłem światła był ekran laptopa. Wiadać, że zdjęcie jest bardzo słabo oświetlone, jednak algorytm poradził sobie z tym zadaniem i prawidłowo rozpoznał planszę.



Po pozytywnym wyniku testu, postanowiliśmy zrobić jeszcze jedno zdjęcie, w pomieszczeniu bez żadnego oświetlenia. Lewe zdjęcie przedstawia oryginalny kadr. Prawe zaś przedstawia obraz po obróbce. Zdjęcie jest praktycznie czarne, nie widać żadnych konturów, dlatego algorytm nic nie zwrócił.

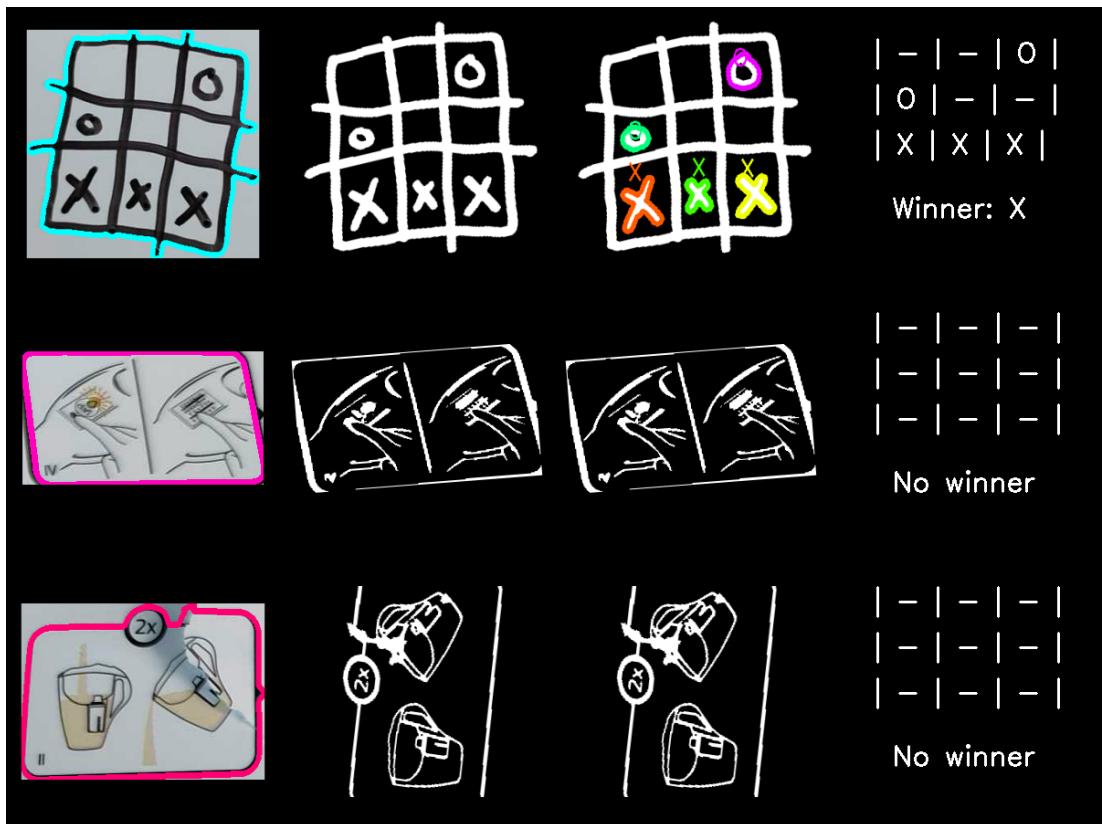


#### 4.4.5 Niestandardowe tła

##### Opakowanie filtrów do wody

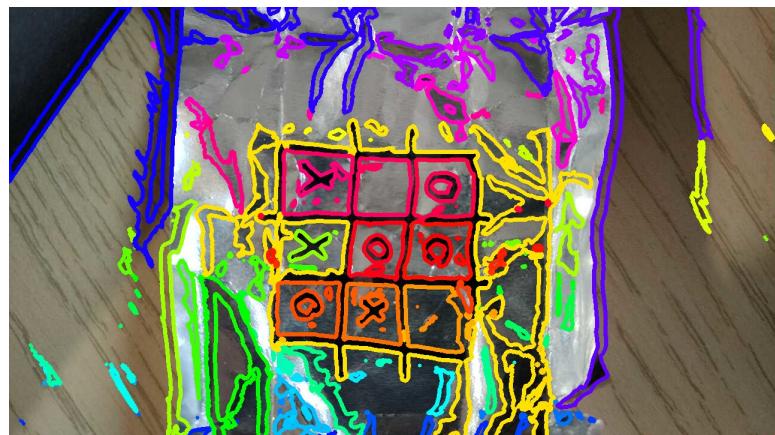


Zdecydowaliśmy, że jeden test wykonamy na opakowaniu filtrów do wody z widocznym kodem QR oraz kreskowym. Byliśmy ciekawi, czy algorytm wyłapie któryś z tych kodów jako plansze. Okazało się, że program rzeczywiście odnalazł dodatkowe kontury „plansz”. Stało się tak, ponieważ te fragmenty spełniają nasze warunki: mają pole większe od minimalnego, są w przybliżeniu prostokątami oraz posiadają co najmniej 9 potomków. Jednak na szczęście algorytm nie wykrył żadnego kafelka.



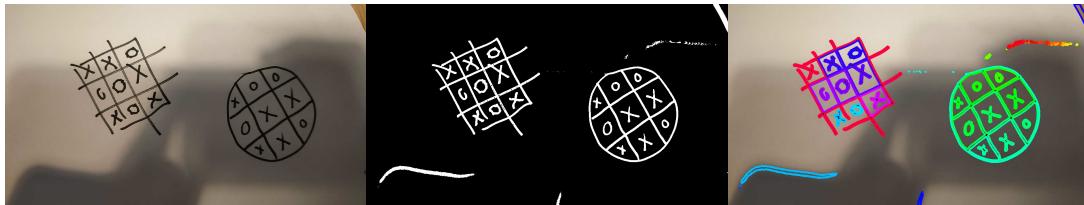
### Papierek aluminiowy

Ze względu na mocne odbicie światła od tego rodzaju podłoża, plansza jest bardzo słabo widoczna. Przetwarzanie obrazu wykryło bardzo dużo konturów, które nie były związanego z planszą. Samej planszy algorytm również nie był w stanie wykryć.



#### 4.5 Pesymistyczne przypadki

Niestety są przypadki, kiedy nasz algorytm zawodzi. Dzieje się tak, kiedy plansza nie jest domknięta, bądź nie jest czworokątem. Poza tym nie zawsze sobie radzi w sytuacji, gdy znaczek jest przecięty z planszą. Wtedy w niektórych przypadkach błędnie rozpoznaje kształt znaku.



Na tym zdjęciu, mimo bardzo dokładnej obróbki zdjęcia, nie została znaleziona żadna plansza.

### 5 Podsumowanie wyników - wnioski

Algorytm jest najbardziej skuteczny w stosunkowo prostych przypadkach. Doskonale rozpoznaje plansze narysowane markerem we większości sytuacji. Jeśli chodzi o inne narzędzia do pisania, musi mieć sprzyjające warunki (np. biała kartka, dobre światło, wysoka rozdzielcość zdjęcia, stabilna pozycja w trakcie robienia zdjęcia).

Wyniki, które wykonaliśmy w sprzyjających warunkach, wychodzą doskonale. Wyliczyliśmy skuteczność około 95%. Sprawdzaliśmy 20 prostych plansz i tylko w jednym przypadku zwrócony został nieprawidłowy wynik.

Jeśli zaś chodzi o skuteczność w cięższych warunkach (słabsze oświetlenie, jasne pisaki, gorsza powierzchnia), algorytm osiąga wyniki około 60%. Również sprawdzaliśmy 20 przypadków w różnych, najmniej korzystnych środowiskach.