**Akademia Górniczo-Hutnicza**

**im. Stanisława Staszica w Krakowie**


Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

Katedra Automatyki

# PRACA DYPLOMOWA

Magisterska

# Rejestracja przebiegu gier planszowych na podstawie obrazu

## (Image-based registration of board games)

Imię i nazwisko: Daniel Ważydrąg

Kierunek studiów: Automatyka i Robotyka

Opiekun pracy: dr inż. Paweł Rotter

Kraków 2012

**OŚWIADCZENIE AUTORA PRACY**

OŚWIADCZAM, ŚWIADOMY ODPOWIEDZIALNOŚCI KARNEJ ZA POŚWIADCZENIE NIEPRAWDY, ŻE NINIEJSZĄ PRACĘ DYPLOMOWĄ WYKONAŁEM OSOBIŚCIE I SAMODZIELNIE I ŻE NIE KORZYSTAŁEM ZE ŹRÓDEŁ INNYCH NIŻ WYMIENIONE W PRACY.

……..

PODPIS

**PODZIĘKOWANIA**

SKŁADAM SERDECZNE PODZIĘKOWANIA PROMOTOROWI PRACY, DR INŻ. PAWŁOWI ROTTEROWI ZA POMOC I WSKAZÓWKI MERYTORYCZNE UDZIELANE PODCZAS PISANIA NINIEJSZEJ PRACY. PRAGNĘ RÓWNIEŻ PODZIĘKOWAĆ RODZINIE ORAZ PRZYJACIOŁOM ZA OKAZANĄ ŻYCZLIWOŚĆ I POMOC.

# 1. Wstęp

Współcześnie coraz częściej w wielu dziedzinach nauki i życia codziennego stosuje się techniki rejestracji, przetwarzania i analizy obrazu. Pomagają zautomatyzować pewne procesy w których jeszcze do niedawna potrzebne było działanie człowieka lub zwiększać wydajność w systemach które były już zautomatyzowane (np.: kontrola elementów na taśmie produkcyjnej, system sterowania światłami). Pomimo zaawansowania i szybkiego rozwoju techniki przetwarzania obrazu, odwzorowanie przestrzeni trójwymiarowej jest problemem, który nie posiada uniwersalnego rozwiązania działającego w sposób niezawodny i jednoznaczny. Z tego powodu konieczne jest tworzenie systemów dostosowanych do danego zagadnienia, które działają w pewnych określonych warunkach i zastosowaniach.

## 1.1. Cel pracy

Celem pracy było stworzenie systemu zapisującego przebieg gry planszowej na przykładzie szachów [1], wykorzystując informację jaką dostarcza obraz wideo. Program ma za zadanie rozpoznać poszczególne figury na planszy wraz z ich ruchami oraz zapisać je w odpowiedniej notacji szachowej. Podczas klasycznej rozgrywki szachowej każdy ruch jest zapisywany przez graczy na specjalnych formularzu. Metoda zapisu partii szachowej nazywana „szachową notacją algebraiczną", powszechnie stosowana jest w zawodach szachowych, literaturze oraz w szachach korespondencyjnych. Właśnie takiego zapisu ma automatycznie dokonywać program zrealizowany w ramach tematu pracy. Szachy są dosyć rozbudowaną grą planszową, wymagającą poza rozróżnieniem koloru figur również jakiego rodzaju to figura (Pion, Wieża, Skoczek, Goniec, Hetman, Król). Informacje te są istotne i już na wstępnych etapach planowania musiały być wzięte pod uwagę. Zestawy szachów posiadają cechy wspólne jak plansza na bazie kwadratu, niezmienna ilość pól szachowych (64 pola ułożone 8x8 z na przemian leżącymi kolorami) taka sama ilość i rodzaj figur oraz położenie początkowe. Jednak występują istotne różnice uniemożliwiające stworzenie uniwersalnego programu działającego z wszystkimi zestawami szachów. Poszczególne zestawy różnią się między sobą kolorami pól (pola ciemne mogą być koloru czarnego lub w jednym z odcieni brązu), rodzajem materiału z jakiego są

wykonane, kształtem figur oraz wzorami zdobiącymi umieszczonymi na polach, które w procesie analizy mogą być wykryte jako figura. Należało więc zastosować takie mechanizmy przetwarzania obrazu aby w jak największym stopniu uwzględnić wpływ tych czynników na działanie programu dając jak największą uniwersalność.

## 1.2. Zawartość pracy

Rozdział 1. Rozdział ten wprowadza z tematykę projektu, nakreślając ogólny cel pracy oraz wstępne zagadnienia związane z grą w szachy które będą poruszane i opisywane w dalszej części, a które musiały być uwzględnione w pierwszym etapie realizacji pracy.

Rozdział 2. Zawiera informacje na temat środowiska oraz narzędzi jakie były wykorzystywane przy tworzeniu projektu. Przedstawiona jest część softwarowa oraz hardwarowa służąca do akwizycji obrazu i przetwarzania obrazu, oraz wyjaśnione jest dlaczego i w jaki sposób zostały wykorzystane. Opisana został sposób konfiguracji biblioteki OpenCV z kompilatorem.

Rozdział 3. W rozdziale tym przedstawiona jest część teoretyczna opisująca wykorzystywane operacje i zagadnienia z dziedziny wizji komputerowej. Wyjaśnione jest w jaki sposób reprezentowany jest obraz cyfrowy, kodowanie kolorów oraz ich konwersja. Opisywane są podstawowe zagadnienia z dziedziny przetwarzania i analizy obrazu t.j. histogram, filtracja, morfologia obrazu, sposób przeprowadzenia czteropunktowej transformacji płaskiej mającej na celu kalibrację płaszczyzny obrazu z płaszczyzną szachownicy, działania Transformaty Hougha wykorzystywanej przy wykrywaniu figur i szachownicy. Przedstawione są również funkcje OpenCV dzięki którym realizowane są przedstawione zagadnienia. Rozdział ma on na celu przybliżenie i lepsze zrozumienie tematyki związanej z wizją komputerową.

Rozdział 4. Przedstawione są poszczególne etapy implementacji programu rozpoznającego przebieg gry w szachy. W rozdziale opisano metody użyte do osiągnięcia zamierzonych celów, pojawiające się problemy oraz sposoby ich rozwiązania. Są nimi m.in. kalibracja współrzędnych szachownicy, wykrycie ruchu ręki, wykrycie figur i wyznaczenie ich przesunięć z właściwą interpretacją. Umieszczone są również zdjęcia obrazujące wynik działania poszczególnych funkcji.

Rozdział 5. Przedstawione są w nim sposoby przeprowadzania testów i otrzymane wyniki. Przedstawione są funkcje które były brane pod uwagę w badaniach lecz nie dały oczekiwanych rezultatów i zostały odrzucone z projektu.

Rozdział 6. Jest to podsumowanie prac, tego co udało się osiągnąć, a czego nie udało się zrealizować. Wypisane zostały wnioski jakie nasunęły się przy poruszanych w pracy zagadnieniach oraz pozycje możliwości rozwoju programu.

# 2. Wprowadzenie

Przystępując do realizacji projektu wybrano odpowiednie narzędzia (t.j. język programowania, urządzenia rejestrujące obraz) dzięki którym możliwe było łatwe i szybkie przeprowadzenie procesu rejestracji gry w szachy, analizy i rozpoznania ruchów. Obecnie na rynku istnieje mnogość języków programowania pozwalających programiście na wybór według własnych preferencji, np. ze względu na złożoność języka, ilość kodu czy oferowanych możliwości. Do tworzenia projektu został wybrany C++, ponieważ aplikacje napisane w nim nie wymagają dodatkowych programów aby mogły działać, a dzięki prostocie i możliwościom nadaje się do realizacji zagadnień związanych z przetwarzaniem obrazu. Istotne również w pierwszej fazie planowania projektu było wybranie rodzaju obrazu który będzie analizowany. Coraz większym powodzeniem cieszą się rozwiązania z wykorzystaniem obrazu stereowizyjnego dostarczającego lepszą informację o obserwowanym obszarze i obiektach na nim się znajdujących. Jednak złożoność tej techniki i problemy z nią związane mogą znacząco utrudnić realizację celu pracy. Podczas gry w szachy mogą występować sytuacje zasłaniania się figur szachowych przy ustawieniu perspektywy kamer w taki sposób aby możliwe było rozpoznawanie kształtów figur przy pomocy stereowizji . Obrazem z którego będzie pochodzić informacja o grze jest obraz 2D, ponieważ zwykle w grach planszowych nie potrzebna jest informacja o głębi pola, a jedynie o położeniu na dwuwymiarowej przestrzeni planszy.

## 2.1. Środowisko programistyczne

Oprogramowanie zostało realizowane w oparciu o platformę Microsoft Windows 7 Professional (64bit) wraz z zintegrowanym środowiskiem programistycznym Microsoft Visual Studio 2010 Professional (64bit). Środowisko to dzięki zaawansowanym narzędziom umożliwia szybsze i prostsze tworzenie oprogramowania m.in. w C, C++, C# działającego na różnych urządzeniach i platformach dając większą swobodę programiście w procesie projektowania. Pakiety działają na licencji udostępnionej przez MSDN Academic Alliance [I4] dla studentów i pracowników naukowych. Ponadto możliwa jest łatwa integracja pakietu Visual Studio 2010 z biblioteką OpenCV firmy Intel.

## 2.2. Biblioteka OpenCV 2.3

OpenCV [I1] jest to zbiór funkcji i aplikacji połączonych w jedną bibliotekę, stworzonych na potrzeby grafiki komputerowej w celu obróbki obrazu. Jest doskonałym narzędziem wykorzystywanym w systemach wizyjnych czasu rzeczywistego dzięki swojej wydajności, a zarazem prostocie obsługi. Stworzona została przez firmę Intel. Wersje OpenCV wcześniejsze niż 2.0 były napisana w języku C, i stworzone tak aby miały charakter obiektowy. Osiągnięte zostało to poprzez utworzenie trzech statycznych struktur IplImage, CvMat, CvArr reprezentujących obraz wraz z osobnymi funkcji operującymi na tych strukturach.

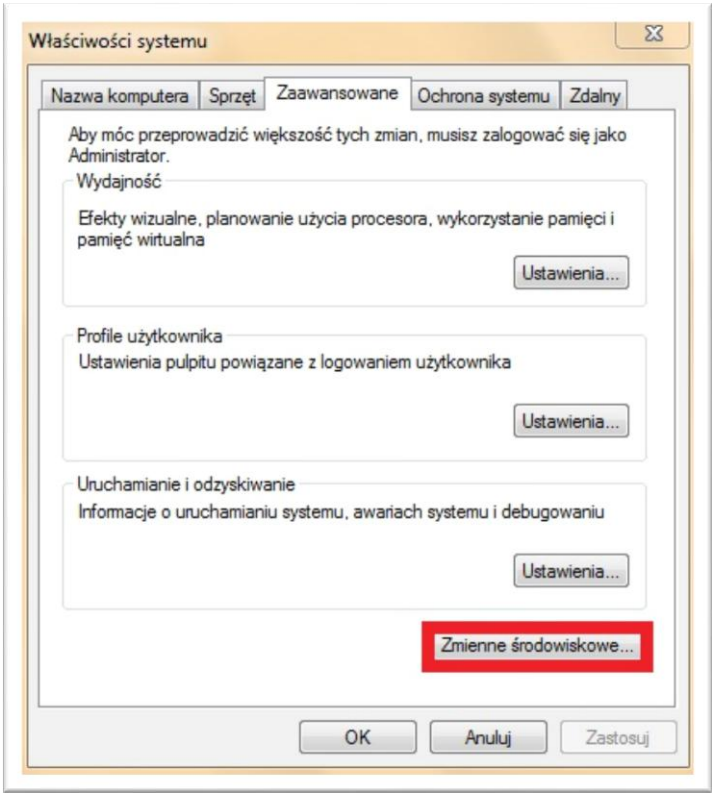
Od wersji OpenCV 2.0 dostępny jest interfejs C++, zawierający obiekty i metody dostępne z przestrzeni nazw *cv*, dostarczając m.in. mechanizmy dziedziczenia, szablony klas, konstruktory i destruktory pozwalając tworzyć w pełni obiektowo nie wymagając pamiętania o alokacji i czyszczeniu pamięci. Dzięki dodatkowym nakładkom tzw. wrapper'om (np. Emgu CV), istnieje możliwość programowania w innych językach t.j. C#, Python. Całość oparta jest na licencji BSD (*Berkeley Software Distribution License)*, przez co dostępna jest za darmo nawet w zastosowaniach komercyjnych, a dzięki specjalnej konstrukcji licencji biblioteka rozwijana jest poza ośrodkiem Intela przez jej użytkowników. OpenCV nie była stworzona do pracy z jednym narzędziem dzięki czemu dostępna jest na wielu platformach (Windows, Linux, MacOS, Solaris i innych), integruje się również z wieloma kompilatorami, m.in. Microsoft Visual Studio co było jednym z powodów jej wyboru.

Biblioteka daje możliwości:

- odczyt i zapis obrazu z kamery lub pliku
- tworzenie interfejsu użytkownika (okna, suwaki, obsługa myszy)
- operacje na macierzach i wektorach
- filtracja obrazu
- transformacja obrazu
- dostęp do pikseli lub obszarów obrazu
- wykrywanie ruchu
- wykrywanie obiektów
- rozpoznawanie kształtów
- rysowanie na obrazie

## Konfiguracja OpenCV

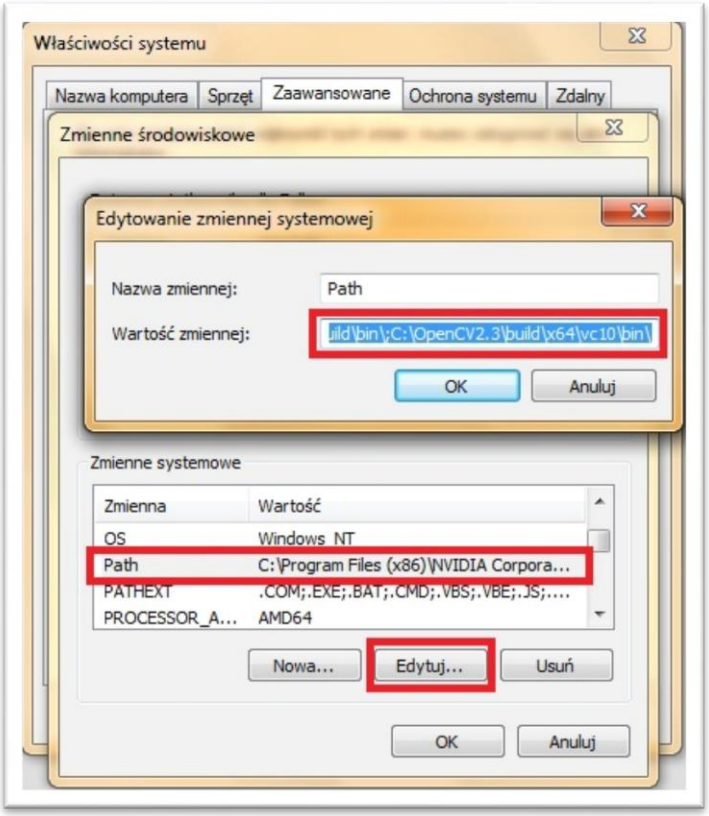
Tworzenie projektu rozpoczyna się od instalacji OpenCV 2.3, dostępnej pod adresem internetowym h*ttp://sourceforge.net/projects/opencvlibrary/files/opencv-win/2.3/,* następnie pobieramy plik *OpenCV-2.3.0-win-superpack.exe*. Po pobraniu i uruchomieniu instalacji należy wskazać ścieżkę dostępu (jej wybór ma wpływ na podawane adresy w dalszej części). Pozostawiając domyślne miejsce lokalizacji biblioteka zostanie zainstalowana w C:\OpenCV2.3. Kolejnym krokiem jest dodanie plików dynamicznych DLL do zmiennej systemowej PATH w zaawansowanych ustawieniach systemu. (*Komputer -> Właściwości -> Zaawansowane ustawienia systemu*, w nowo otwartym oknie: *Zaawansowane -> Zmienne środowiskowe…*)




**Rys. 2.1.** Zaawansowane ustawienia systemowe.

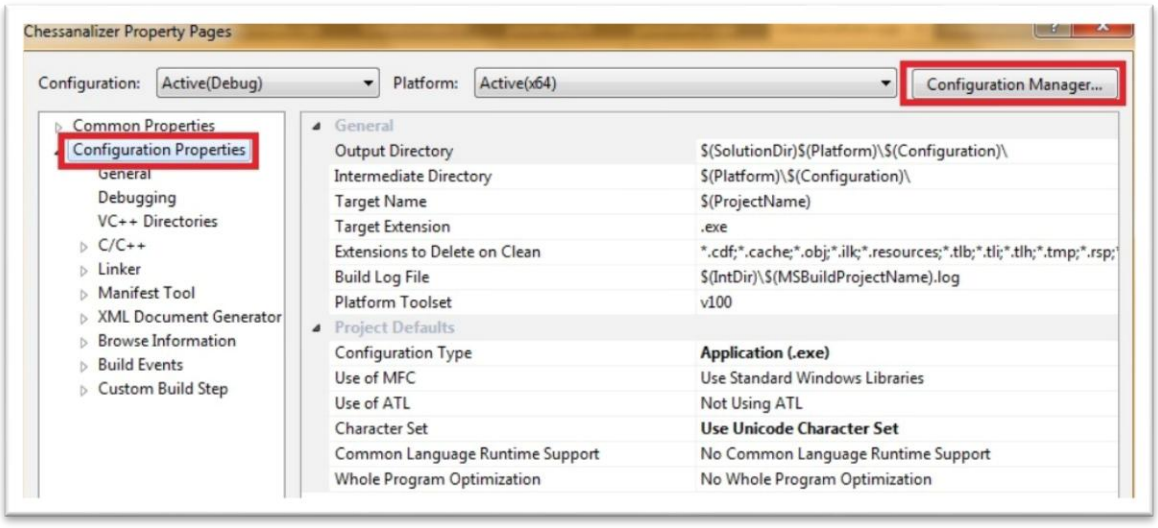
W oknie „zmienne środowiskowe” zaznaczamy **Path** i wybieramy przycisk Edytuj..., na końcu pola „wartość zmiennej” dodajemy:

***C:\OpenCV2.3\build\bin\; C:\OpenCV2.3\build\x64\vc10\bin***




**Rys. 2.2.** Okno ustawień zmiennych środowiskowych.

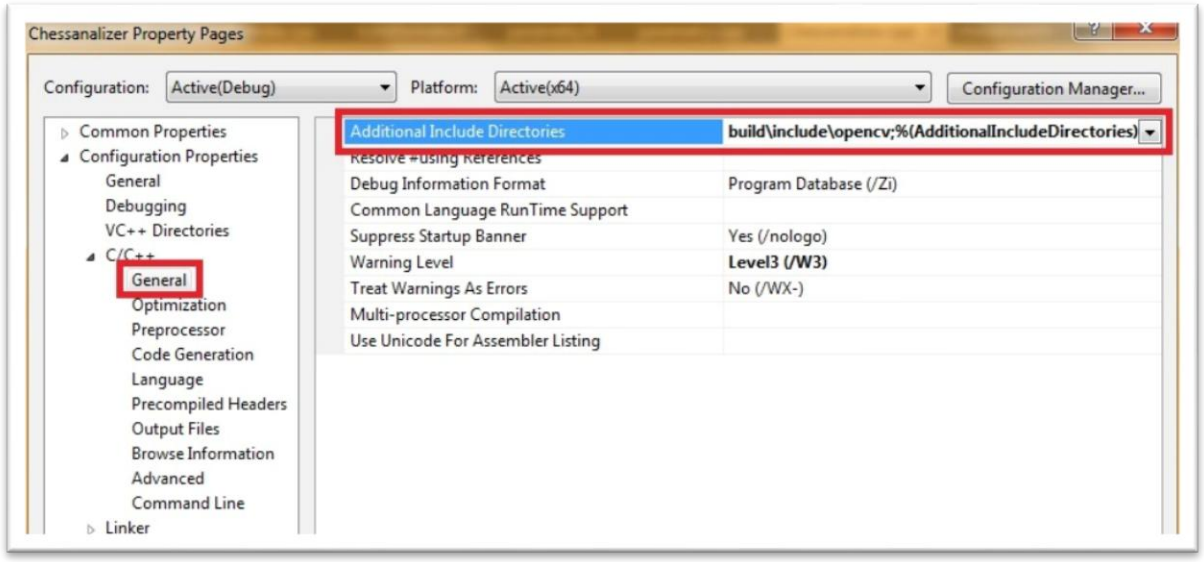
Otwieramy Visual Studio 2010 Professional i tworzymy nowy projekt wybierając Visual C++ -> Win32 Console Application. Po utworzeniu projektu należy skonfigurować Visuala wchodząc do menu *Project -> Properties* , zaznaczając *Configuration Properties* i klikając *Configuration Manager…* .




**Rys. 2.3.** Okno konfiguracyjne projektu.

W oknie Menagera z listy *Active solution platform* wybieramy <New…>, a następnie z listy *Type or select the new platform* wybieramy x64. Zatwierdzamy zmiany OK. Wracając do okna Properties, wybieramy *Configuration Properties -> C/C++ -> General* i do Additional Include Directories wybierając z listy <Edit…> wpisujemy:

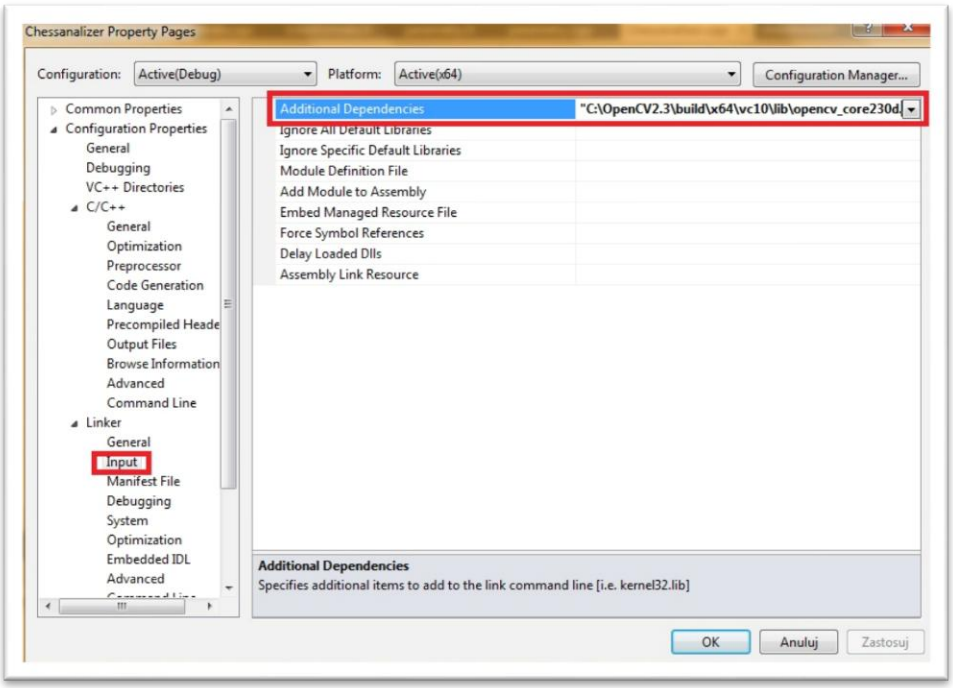
**C:\OpenCV2.3\build\include;**
**C:\OpenCV2.3\build\include\opencv;%(AdditionalIncludeDirectories)**




**Rys. 2.4.** Additional Include Directories.

Przechodząc do zakładki *Configuration Proporties -> Linker -> Input* następnie wybierając *Additional Dependencies* wpisujemy:

**"C:\OpenCV2.3\build\x64\vc10\lib\opencv_core230d.lib";**
**"C:\OpenCV2.3\build\x64\vc10\lib\opencv_highgui230d.lib";**
**"C:\OpenCV2.3\build\x64\vc10\lib\opencv_video230d.lib";**
**"C:\OpenCV2.3\build\x64\vc10\lib\opencv_ml230d.lib";**
**"C:\OpenCV2.3\build\x64\vc10\lib\opencv_legacy230d.lib";**
**"C:\OpenCV2.3\build\x64\vc10\lib\opencv_imgproc230d.lib";**
**%(AdditionalDependencies)**




**Rys. 2.5.** Dodanie plików bibliotecznych lib.

Zatwierdzamy zmiany w konfiguracji klikając OK.

W edytorze kodu do nagłówka pliku źródłowego projektu należy dodać:

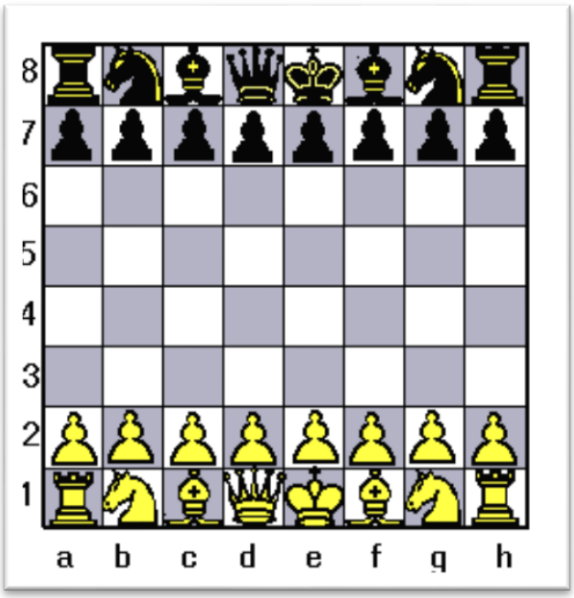
```
#include <cxcore.hpp>
#include <cv.h>
#include <highgui.h>
#include <opencv2/opencv.hpp>
#include <cvaux.h>
```

Należy zwrócić uwagę podczas konfiguracji na instalowaną wersję, ponieważ istnieją różnice w nazwach plików dodawanych do projektu zależnie od używanej wersji OpenCV. Dokumentacja biblioteki z opisem wszystkich funkcji dostępna jest w katalogu *OpenCV2.3/opencv/doc* lub pod adresem [I2].

## 2.3. Podstawy gry w szachy

Ze źródeł wynika, że w gry planszowe znane były już ludzkości sześć tysięcy lat temu. Natomiast, gra którą znamy obecnie jako szachy [1], najprawdopodobniej została stworzona około 600 roku n.e. w Indiach, lecz nie wiadomo czy została wymyślona jako niezależna gra, czy powstała na bazie kilku innych gier planszowych. Początkowo gra była znana tylko w krajach arabskich, jednak poprzez szlaki handlowe i w trakcie podbojów zaczęły być znane również w Europie. Szachy w przeciwieństwie do większości gier nie opierają się na przypadku, a wynik rozgrywki zależy wyłącznie od umiejętności graczy i ich decyzji. Szachy są jednymi z najbardziej złożonych gier i do tej pory, pomimo wielkiego rozwoju komputerów, nie udało się stworzyć algorytmu który zapewniałby zawsze wygrywanie partii. Szacuje się że ilość możliwych posunięć na szachownicy, jest większa, niż liczba atomów we Wszechświecie. Postać szachów jaką znamy dzisiaj, ustalono w XVI w. Podlegały one wcześniej zmianom tak samo jak struktury społeczne, które odzwierciedlane były przez poszczególne rodzaje figur. Hetman zyskał większe możliwości, stając się najpotężniejszą figurą, goniec zwiększył zasięg, a pionki mogą się poruszać o dwa pola z pozycji początkowej (wprowadzono wtedy ruch specjalny jakim jest bicie w przelocie). Obecnie za przepisy gry w szachy oraz organizację olimpiad i mistrzostw świata odpowiedzialna jest Międzynarodowa Federacja Szachowa zrzeszająca krajowe związki szachowe, utworzona 20 lipca 1924 w Paryżu.

W klasycznej grze w szachy biorą udział dwie osoby grając na kwadratowej planszy z 64 kwadratowymi polami. Pola te mają dwa na przemian leżące kolory jasny i ciemny nazywane białymi i czarnymi. Takiej samej planszy używa się do gry w warcaby. Szachy zawierają 32 figury również jasne i ciemne, występujące w sześciu rodzajach (inne ruch dla każdego rodzaju). Planszę należy ustawić tak aby w lewym bliższym gracza rogu znajdowało się czarne pole. Figury białe ustawia się w rzędzie 1-2, a figury czarne w rzędzie 7-8. Schemat ustawienia figur znajduje się na (rys.2.6).




**Rys. 2.6.** Ustawienie początkowe.

Jednoznacznym oznaczeniem pola są jego współrzędne składające się z litery i cyfry. Wiersze opisane są cyframi od 1 do 8, kolumny natomiast literami od *a* do *h.* Tak więc białe skoczki w ustawieniu początkowym mają pozycję *b1* i *g1*. Figury oznaczane są pojedynczą dużą literą alfabetu i są następujące: K – Król, H – Hetman, W – Wieża, G – Goniec, S – Skoczek. Pionki oznaczane są jako brak litery z samym oznaczeniem wykonanego ruchu. Rodzaje ruchów każdej z figur nie zostaną tutaj przedstawione gdyż nie są konieczne do rozpoznawania przebiegu gry dla wybranej koncepcji projektu. Program ma wskazywać która figura wykonała ruch oraz jej położenie początkowe i końcowe. Posunięcie figury można przedstawić przez podanie literowej nazwy figury i współrzędnych pola na które się przemieściła. Gdy występuje niejednoznaczność (ponieważ dwie identyczne figury mogą zająć do samo pole), wprowadza się również informację o polu z którego następuje ruch. Przykładowy zapis kilku ruchów:

| Pełny zapis: | Skrócony zapis: |
|---|---|
| 1. e2-e4 e7-e5 | 1. e4 e5 |
| 2. Sb1-c3 Sb8-c6 | 2. Sc3 Sc6 |
| 3. Sg1-f3 Sg8-f6 | 3. Sf3 Sf6 |
| 4. Gf1-c4 d7-d6 | 4. Gc4 d6 |

## 2.4. Stanowisko laboratoryjne

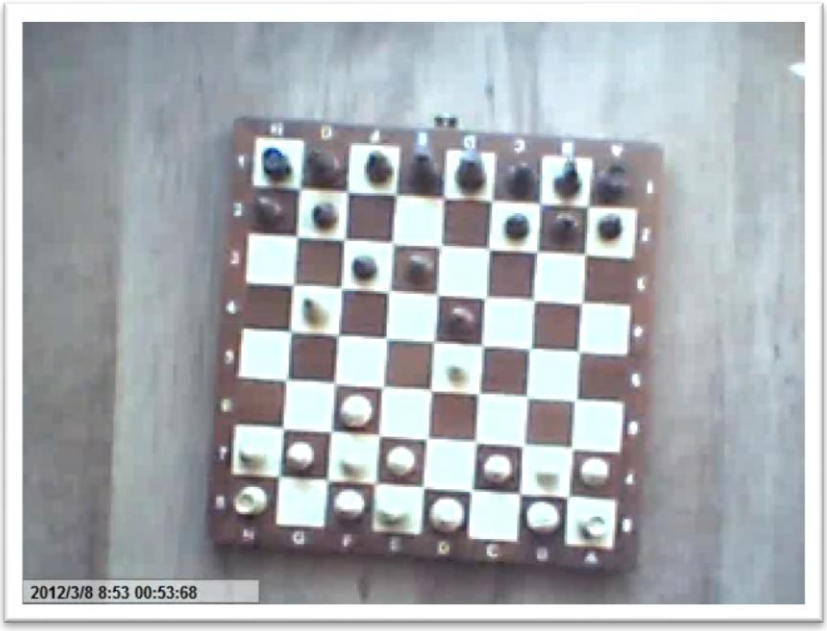
### 2.4.1. Użyty sprzęt

Do wykonania pracy posłużono się komputerem przenośnym typu notebook na którym został zaimplementowany program i odbywał się cały proces przetwarzania materiału wideo.

Na potrzeby projektu do akwizycji obrazu posłużono się urządzeniami:

- Kamera internetowa USB Logitech QuickCam Express 640x480px (30 fps).
- Kamera internetowea USB Creative LiveCam OPTIA z matrycą CMOS 640x480px (30 fps).
- Aparat cyfrowy Casio Exilim EX-F1 z opcją nagrywania filmów wideo FullHD. Przy ustawieniach STD rejestrujący 640x480 px i 30 fps. (udostępniony w pracowni katedry Automatyki AGH).

Bardzo ważnym czynnikiem jest nie tylko rozdzielczość matrycy podana w pikselach ale również rozdzielczość optyczna obiektywu podawana jako ilość linii na milimetr. Nawet matryce o dużej rozdzielczości nie będą w stanie dokładnie rejestrować obrazu jeśli wartość MFT50 (lpmm) [I5] będzie zbyt mała. Z tego powodu po sprawdzeniu jakości obrazu została odrzucona kamera Logitech QuickCam. Zbyt niska jakość obrazu oraz pojawiające się szumy uniemożliwiają rozróżnienie wykrywanych obiektów od tła.




**Rys. 2.7.** Przykład obrazu o niskiej rozdzielczości optycznej.


**Rys. 2.8.** Przykład obrazu o dobrej rozdzielczości optycznej.

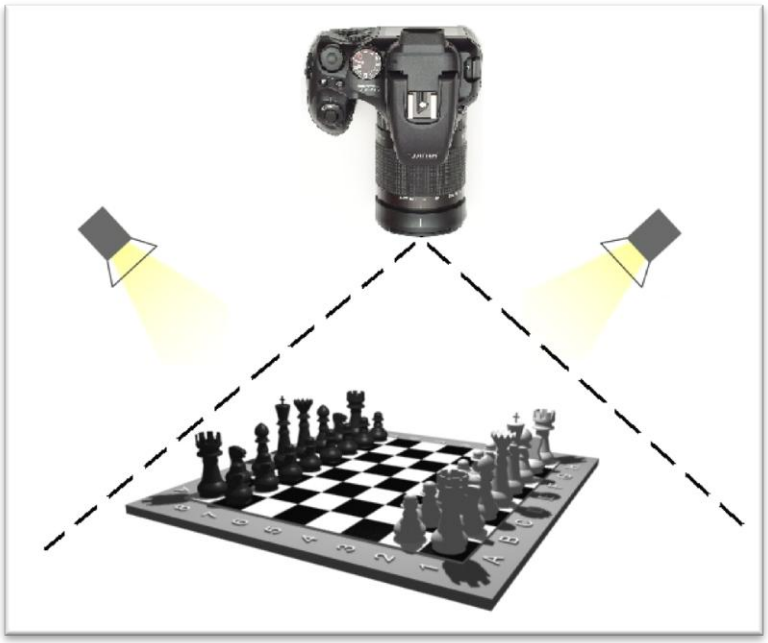
W pracy wykorzystano obrazy pochodzące z dwóch źródeł. Jednym jest obraz przechwytywany w czasie rzeczywistym z kamery, natomiast drugi pochodzi z nagranego pliku wideo. Algorytm programu powinien działać dostatecznie szybko aby pomiędzy kolejnymi pobieranymi klatkami nie następowało zbyt duże opóźnienie powodujące utratę informacji o wykonanym ruchu.

### 2.4.2. Perspektywa kamery

Stosując jedną kamerę dającą obraz dwuwymiarowy konieczne było ustawienie odpowiedniej perspektywy widzenia. Nagrywanie sceny pod różnymi kątami z boku pozwalałoby na przeprowadzenie procesu rozpoznawania kształtu figur. Jednak zjawisko zasłaniania się figur, całkowitego lub chociażby częściowego uniemożliwiło by ich rozróżnienie. Jedynie odpowiednio duży kąt w płaszczyźnie pionowej pozwolił by na wyeliminowanie zasłaniania się figur, równocześnie uniemożliwiając rozpoznanie ich kształtu.

**Rys. 2.9.** Perspektywa powodująca zasłanianie się figur.

W przypadku rozpoznawania kształtu konieczne byłoby stworzenie obszernej bazy kształtów figur jakie mogą przyjmować zestawy szachowe lub tworzenie bazy wiedzy o kształcie na początku rozgrywki. Do ominięcia tego problemu wykorzystano fakt iż przebieg gry rejestrujemy od samego początku rozgrywki, a ustawienie początkowe figur jest zawsze takie samo. Pozwoliło to na stworzenie programu który rozpoznaje tylko gdzie znajdują się figury i jaki ruch został wykonany aby wyznaczyć ustawienie figur po zmianie położenia. W tej sytuacji pozostał tylko problem z ustawieniem takiej perspektywy, aby możliwe było wykrycie na którym polu znajdują się figury, a na którym nie. Optymalnym ustawieniem kamery rozwiązującym ten problem jest rzut z góry, dzięki któremu figury nie nachodzą na sąsiednie pola. Eliminuje to problem z niejednoznacznością położenia figury.



**Rys. 2.10.** Schemat poglądowy stanowiska.

W przypadku gdy obiektyw znajduje się blisko szachownicy, konieczne jest stosowanie obiektywu szerokokątnego o mniejszej ogniskowej, co w konsekwencji prowadzi do znacznych zniekształceń radialnych. Przy zbyt małej ogniskowej i niewielkiej odległości obiektywu od szachownicy obraz zostaje tak zniekształcony, że wysokie figury na obrzeżach planszy mogą wychodzić poza pole. W rezultacie algorytm wykrywający może wyznaczyć ich położenie poza obrębem pól szachowych. Przez to wskazane jest stosowanie większej wysokości ustawienia kamery i dłuższej ogniskowej obiektywu. Kamerę należy tak ustawić aby bok szachownicy przy którym stoją białe figury znajdował się w dolnej części, ponieważ w ten sposób przedstawia się gry szachowe i schematy w literaturze. Dla działania programu nie miałoby to znaczenia ponieważ program mógłby sam wykrywać ustawienie planszy poprzez wykrycie po której stronie występuje jaki kolor figur. Ustalenie orientacji ustawień szachownicy jest to konieczne do późniejszego wykrywania współrzędnych pól. Dla uproszczenia działania przyjęto, że właściwe ustawienie powinno nastąpić już na etapie przygotowywania stanowiska i powinno przyjmować standardowy sposób prezentacji, czyli białe figury w dolnej części obrazu z umiejscowionym w lewym dolnym rogu ciemnym polem *a1*.

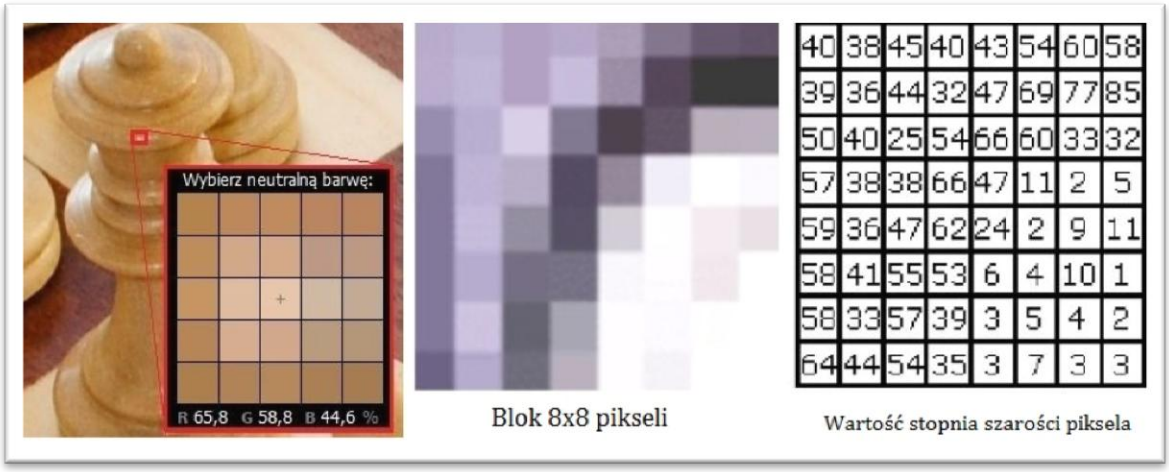
### 2.4.3. Oświetlenie

Prawidłowe działanie systemów wizyjnych zależy od równomierności i poziomu oświetlenia obserwowanej sceny. To właśnie światło w dużej mierze wpływa na jakość obrazów generowanych przez kamery. Nawet zaawansowane algorytmy analizy obrazu i najnowocześniejsze kamery nie będą w stanie rozpoznać tego co znajduje się na obrazie, jeśli scena nie będzie właściwie oświetlona. W systemach tych oprócz światła naturalnego stosuje się źródła światła sztucznego. Istotne jest jakiego rodzaju lampy używane są do oświetlenia sceny, gdyż ze względu na swoje właściwości mają różny zakres zastosowań. Często w budynkach stosuje się lampy wyładowcze, dające efekt migotania, który trudno wyeliminować. Zależnie od rodzajów materiałów użytych w lampach, ich światło posiada różną barwę, wpływając na kolory oświetlanej sceny. W przypadku większości źródeł światła, centralna cześć sceny jest intensywniej oświetlona. Istnieją systemy oświetleniowe pozwalające zminimalizować te różnice, umożliwiając regulację poziomu oświetlenia dla poszczególnych obszarów sceny. W zagadnieniach poruszanych w pracy ważny jest aspekt wpływu oświetlenia na proces rejestracji gry w szachy. Ponieważ pomiędzy ciemnymi i jasnymi polami występuje duży kontrast, zbyt mocne oświetlenie może spowodować prześwietlenie jasnych pól. Należy również zwrócić uwagę na ustawienie źródła światła, gdyż mogą tworzyć się odblaski i cienie. Podczas prac nad projektem wykorzystano rozproszone światło dzienne padające przez okno wraz z oświetleniem sztucznym lamp umieszczonych na suficie. Światło wtedy w miarę równomiernie pada na planszę szachową nie powodując zbyt głębokich cieni. Zastosowano również blendę odbijającą światło wypełniając cienie powstające przez światło padające z okna. Zaleca się unikania bezpośredniego światła słonecznego padającego na planszę lub w jej pobliżu. Przeważnie rozgrywki szachowe odbywają się w większych salach, w których światło pochodzi z dużych okien. Jednak zazwyczaj trudno w ten sposób uzyskać idealne warunki oświetleniowe w pomieszczeniu, ponieważ natężenie światła naturalnego w ciągu dnia ulega zmianie. Dlatego konieczne jest zastosowanie sztucznych źródeł światła, czyli np.: lamp umieszczonych pod sufitem, równomiernie oświetlających pomieszczenie.

# 3. Przetwarzanie obrazu

## 3.1. Reprezentacja obrazu

Obecnie systemy wizyjne zdominowane są przez obraz reprezentowany w postaci cyfrowej, dzięki czemu możliwa jest komputerowa akwizycja i przetwarzanie obrazu. Obraz składa się z pikseli rozmieszczonych w macierzy reprezentowanej w pamięci komputera jako wielowymiarowa tablica. W tablicy tej piksel opisany jest przy pomocy wartości liczbowej określającej jasność lub numer koloru, zależnie czy obraz jest w skali szarości czy kolorowy.




**Rys. 3.1.** Reprezentacja obrazu w postaci cyfrowej.

Macierz obrazu przechowywana jest w klasie *cv::Mat* [I2]. Ze względu dużą liczbę konstruktorów i metod, zostaną omówione tutaj tylko najważniejsze elementy klasy oraz mechanizm jej tworzenia, potrzebny do zrozumienia działania programu.

```
class CV_EXPORTS Mat
{
public:
    // ... metody klasy ...
    Mat (int _rows, int _cols, int _Typ); // konstrukcje macierzy
                                          określonego typu i rozmiaru
    //(_Typ jest CV_8UC1, CV_64FC3, CV_32SC (12) itp.)
     ...

    int flags
    int dims; //! wymiar, >= 2
    int rows, cols; //! numer wierszy i kolumn lub (-1, -1) kiedy wektor
ma więcej niż 2 wymiary
```

```
    uchar* data; //! wskaźnik do danych
    ... // inne parametry
};
```

Jednym ze sposobów utworzenia macierzy dla obrazu kolorowego jest użycie konstruktora z podaniem wymiarów macierzy i typem wartości w niej przechowywanej:

```
// utworzenie nowego obrazu 640x480
Mat img(Size(640,480),CV_8UC3);
Mat frame(640,480,CV_8UC3);
```

Określony typ jest zakodowany w postaci definiowanej wartości, a schemat definicji wygląda następująco:

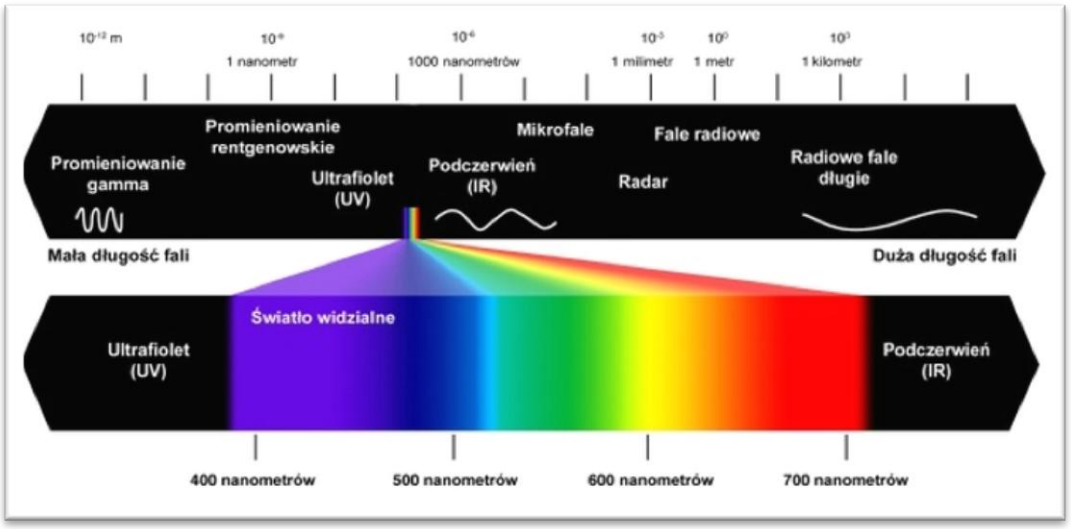
**CV_<bit_depth>(S/U/F)C<channel_numebers>**

*bit_depth* – ilość bitów na piksel
*(S/U/F)* – rodzaj liczby (całkowita, całkowita dodatnia, rzeczywista)
*channel_numbers* – ilość kanałów wykorzystywana w obrazach barwnych

Chcąc stworzyć kolorowe zdjęcie w RGB należało by utworzyć macierz używając typu **CV_8UC3**, oznaczającego trzy kanały po jednym na kolor przechowujących wartości 8 bitowe bez znaku.

## 3.2. Reprezentacja kolorów

Różnobarwne światło postrzegane przez wzrok człowieka to część widma promieniowania elektromagnetycznego w zakresie 380 – 780nm nazywane światłem widzialnym. Jest to zakres widoczny dla ludzkiego oka, jednak matryce aparatów i kamer potrafią rejestrować światło podczerwone nawet do 1200nm. Jednakże ze względów na zakłócenia jakie wprowadza światło podczerwone przy przetwarzaniu obrazu rejestrowanego na matrycy na zdjęcie, stosuje się filtry nie przepuszczające podczerwieni.




**Rys. 3.2.** Spektrum promieniowania elektromagnetycznego i zakres światła widzialnego.

W różnych dziedzinach stosuje się modele, które przedstawiają kolory w sposób zbliżony do percepcji ludzkiego oka. Modele tworzenia barwy :

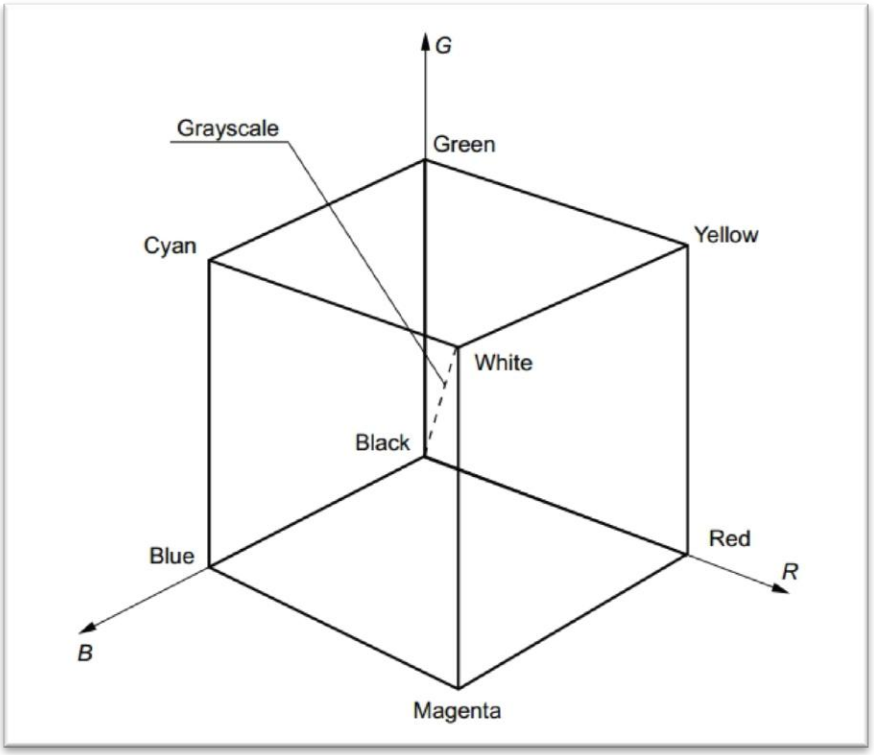
- Synteza addytywna – w modelu tym barwa powstaje poprzez dodanie do siebie strumieni światła trzech braw podstawowych (Red, Green, Blue).
- Synteza substraktywna – model bazujący na pochłanianiu niektórych długości światła białego (cyjan, magenta, yellow).

Modele substraktywne wykorzystuje się w procesach poligraficznych, gdzie na biały papier nanoszone są warstwy farby cyjan, magenta, yellow pochłaniające z padającego na nie światła białego odpowiednie zakresy promieniowania.

Urządzenia rejestrujące i wyświetlające obraz (monitory LCD, telewizory itp.) są oparte na addytywnym systemie tworzenia koloru. Modele te bazują na sposobie rozróżniania koloru przez pręciki i czopki. Istnieją trzy rodzaje czopków z których każdy reaguje silniej na dany rodzaj fali. Zdolność postrzegania otoczenia przez pręciki można porównać z obrazem w skali szarości informującym tylko o wartości jasności obrazu. Kolory w komputerowym przetwarzaniu obrazu opisuje się w przestrzeni barw będącej zestawem parametrów opisujących barwę jako punkt w przestrzeni trójwymiarowej (są nimi m.in. RGB, HSV, Lab).

### 3.2.1. Model RGB

System stosowany już w technologii analogowej i obecnie powszechnie stosowany w urządzeniach analizujących i wyświetlających obraz. Przedstawić go można w postaci sześciennej kostki reprezentującej w trzech osiach udział podstawowych kolorów R (red), G (green), B (blue) w tworzeniu wynikowej barwy.

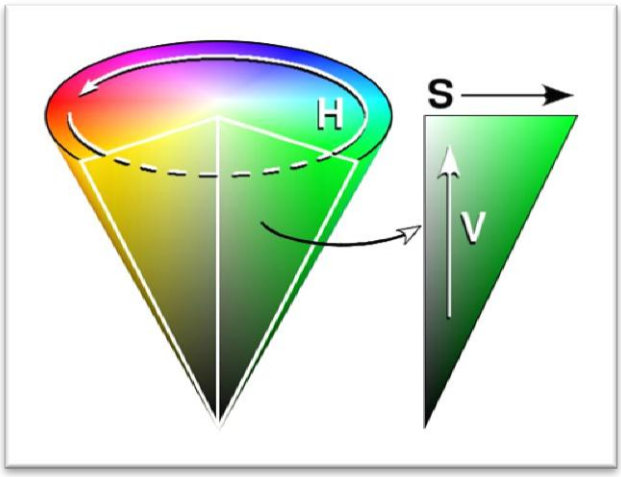



**Rys. 3.3.** Sześcian barw. Źródło obrazu [3].

Wartości R, G, B dla kolorów rzeczywistych przyjmują wartości dodatnie i ujemne. Jednak w komputerze piksel reprezentowany jest jako trójka nieujemnych liczb o wartościach z zakresu [0 – 255], czyli po 8 bitów na każdy kolor. Daje to 24 bitową głębię i możliwość zapisu ponad 16,5 miliona kolorów. Wartość (0,0,0) reprezentuje kolor czarny, a (255,255,255) kolor biały. Niestety w tej reprezentacji może pojawić się problem przy płynnym przejściu gradientu, ponieważ istnieje mały zakres możliwych odcieni danego koloru, jednak z sytuacją tą dość rzadko ma się do czynienia w obrazach rzeczywistych. W zastosowaniach cyfrowych istnieje jeszcze czwarty parametr określony jako kanał alpha będący współczynnikiem pochłaniania światła odpowiadając za przeźroczystość. Kanał alpha występuje w strukturze IplImage biblioteki OpenCV lecz nie jest wykorzystywany przez funkcję tej biblioteki.

### 3.2.2. Model HSV

Barwa w HSV przedstawiana jest za pomocą trzech wartości **H** (hue – barwa), **S** (saturation – nasycenie), **V** (value – wartość). Barwę H określa się przez podanie wartości kąta jaki ją określa na kole barw, natomiast nasycenie S i wartość V przy pomocy liczb rzeczywistych z zakresu 0 – 1 (w systemach cyfrowych używa się zakresu liczb całkowitych 0 – 255, do tego celu są stosowane funkcje normalizujące każdą składową). W celu lepszego zrozumienia modelu można go przedstawić jako ostrosłup foremny o podstawie koła lub sześciokąta. H jest interpretowane tak jak wcześniej pisano jako kąt. Nasycenie określa się jako promień podstawy stożka, poruszając się po nim od krawędzi do środka przechodzi się od czystej barwy do bieli. Wartość interpretuje się jako wysokość, gdzie oddalając się od podstawy zwiększa się ilość czerni biorącej udział w tworzeniu koloru.




**Rys. 3.4.** Reprezentacja geometryczna modelu HSV. Źródło [I7].

### 3.2.3. Skala szarości

Jest to reprezentacja obrazu przy udziale tylko koloru czarnego i białego w odpowiednich proporcjach. Ponieważ wiele funkcji przetwarzania wizyjnego wymaga obrazu źródłowego w skali szarości, konieczna jest konwersja kolorowego obrazu. W modelu RGB skala szarości powstaje poprzez taki sam udział składowych R, G, B, a ich wartość określa dany odcień. Natomiast w modelu HSV skala szarości odpowiada ustawieniu wartość nasycenia S na 0 i odpowiedniej zmianie składowej V. Dostępne są przez to kolor czarny, biały i 254 odcienie szarości.

## 3.3. Podstawowe operacje przetwarzania

### 3.3.1. Konwersja przestrzeni kolorów

**RGB -> GRAY**

Obraz w skali szarości z przestrzeni RGB otrzymuje się:

$$V = 0.299 * R_{kolor} + 0.587 * G_{kolor} + 0.114 * B_{kolor}; \quad (3.1)$$

**RGB ->HSV**

W przypadku 8-bitowych i 16-bitowych obrazów,, R G i B są konwertowane do formatu zmiennoprzecinkowych i skalowane do zakresu 0 do 1.

$$V = \max(R, G, B), \quad (3.2)$$

$$S = \begin{cases} \dfrac{V - \min(R,G,B)}{V} & jeśli\ V \neq 0 \\ 0 & pozostały\ przypadek \end{cases} \quad (3.3)$$

$$H = \begin{cases} 60(G-B)/(V - \min(R,G,B)) & dla\ V = R \\ 120 + 60(B-R)/(V - \min(R,G,B)) & dla\ V = G \\ 240 + 60(R-G)/(V - \min(R,G,B)) & dla\ V = B \end{cases} \quad (3.4)$$

Otrzymane wartości są następnie konwertowane na docelowy typ danych. Przy zapisie 8 bitowym wartość składowej $H_b$ wylicza się $H_b = H/2$, aby dopasować kąt z zakresu 0 - 360° do skali możliwych wartości 0 - 255. Zmniejsza to rozdzielczość kolorów do 2°. Przez to wartość H kodowana jest w zakresie 0 – 179.

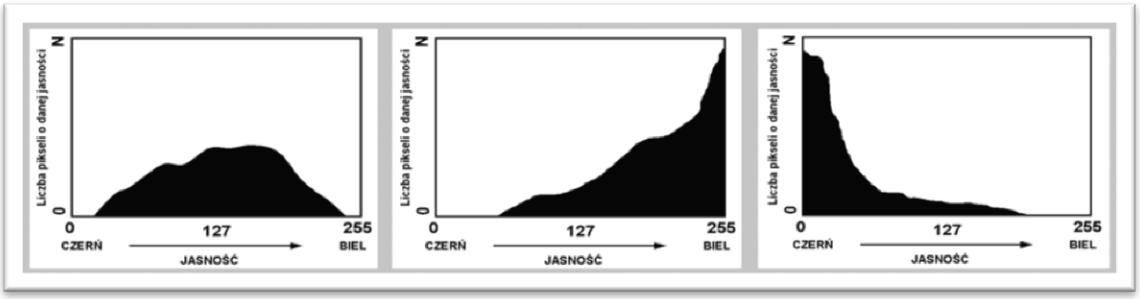
Konwersję pomiędzy przestrzeniami w OpenCV przeprowadza się przy użyciu funkcji:

*void* ***cvtColor****(InputArray* ***src****, OutputArray* ***dst****, int* ***code****, int* ***dstCn****=0 )*

- *src* – obraz źródłowy do konwersji.
- *dst* – obraz wynikowy po konwersji, taki sam rozmiar i głębia jak *src*.
- *code* – kod określający przestrzeń z której i do jakiej następuje konwersja.
- *dstCn* – ilość kanałów obrazu wynikowego. Wartość 0 lub jej pominięcie powoduje automatyczne wykrycie głębi.

### 3.3.2. Histogram

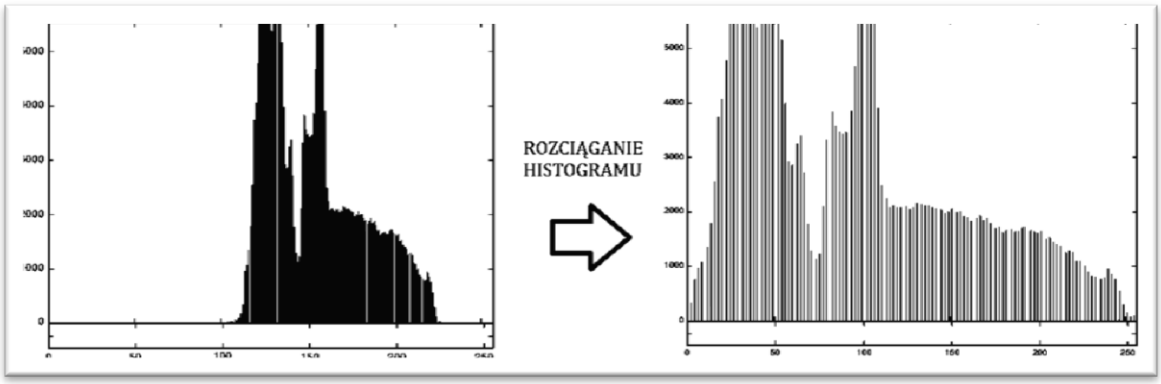
Histogram informuje o rozkładzie jasności pikseli (luminancji) oraz zależnościach pomiędzy obszarem świateł, półcieni i cieni jakie występują w obrazie. Graficznie jest reprezentowany przez wykres słupkowy. Oś pozioma przedstawia wartość jasności pikseli, natomiast oś pionowa pokazuje liczebność pikseli o danej jasności.




**Rys. 3.5.** Histogram: dobrze naświetlone zdjęcie (po lewej), prześwietlone obszary zdjęcia (*środkowy*), niedoświetlone obszary zdjęcia (po prawej). Źródło [I8].

Dzięki odpowiedniej interpretacji wykresu można dowiedzieć się czy zdjęcie jest prześwietlone, niedoświetlone albo naświetlone prawidłowo (zrównoważony rozkład wartości tonalnej, czyli wartości powinny zawierać się od obszaru głębokiego cienia do silnych świateł z wyraźnym maksimum w zakresie półcieni). Informuje również o widoczności szczegółów, kontraście lub jego braku.

W technice wizyjnej histogram jest bardzo użytecznym narzędziem ponieważ na jego podstawie można dokonać nie tylko analizy zawartości zdjęcia lecz również odpowiednio przetwarzać obraz. Poprawienie jakość zdjęcia zbyt ciemnego lub zbyt jasnego można uzyskać poprzez operację „wyrównania histogramu", powodującą zmianę jasności pikseli wg. jakiejś funkcji oraz operację „rozciągnięcia histogramu" nie zmieniającej wysokości słupków histogramu lecz jedynie ich rozsunięcie na cały dostępny zakres tonów. Operacja „rozciągnięcia" służy do uzyskania maksymalnie równomiernego rozkładu tonów w obrazie dla całego zakresu 0 - 255. Operacja podnosi kontrast na obrazie powodując uwydatnienie słabo rozróżnialnych elementów, wzmacnia jedynie różnicę pomiędzy pikselami, nie ukazując informacji, które zniknęły poprzez złą ekspozycję zdjęcia.




**Rys. 3.6.** Operacja „rozciągania histogramu".

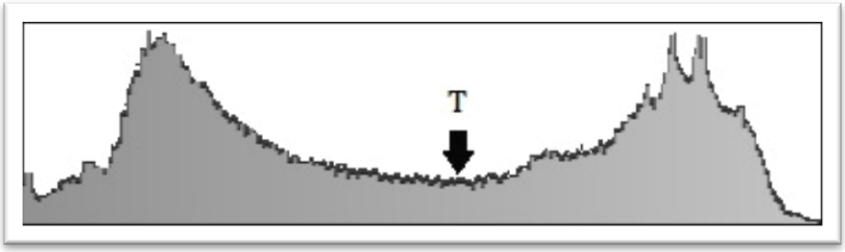
Tę oraz inne operacje na histogramie przeprowadza się przekształcenie obrazu przez odpowiednio dobraną „Look-up table"(LUT). Tablica ta zawiera wartości składowych obrazu oraz odpowiadające im nowe wartości wyznaczone według funkcji konwersji dla danego przekształcenia.

### 3.3.3. Progowanie

Progowanie jest metodą wydzielania obszarów obrazu poprzez klasyfikację przynależności pikseli do jednego z dostępnych zbiorów wartości. Przydzielenie do zbioru odbywa się poprzez porównanie wartości jasności piksela z ustalonym progiem T (threshold). Jednym z najczęstszych przypadków progowania jest binaryzacja. Polega ona na przydzieleniu danego piksela do jednego z dwóch zbiorów, reprezentującego punkty czarne (wartość 0) i punkty białe (wartość 255).

$$p(x,y) = \begin{cases} 255, & \text{dla } f(x,y) \geq T \\ 0, & \text{dla } f(x,y) < T \end{cases} \tag{3.5}$$

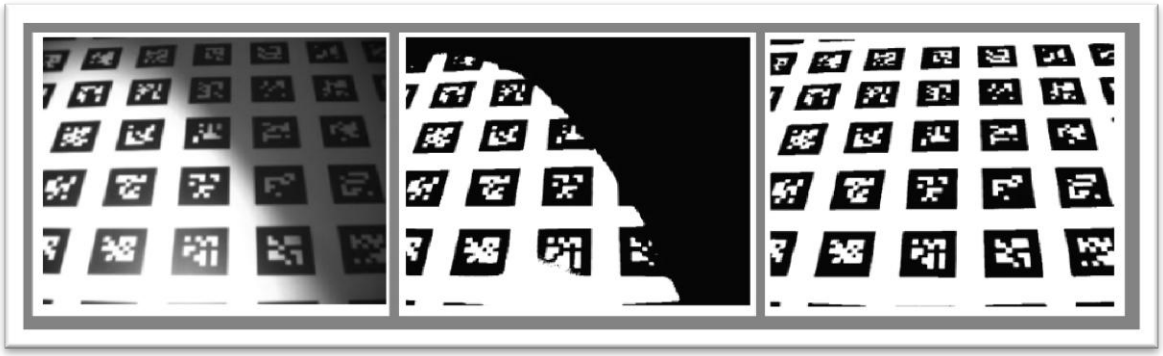
Wyznaczenie progu binaryzacji T nie jest zadaniem prostym. Do wyznaczenia jego wartości można posłużyć się histogramem, ustalając wartość progu pomiędzy dwoma szczytami jeśli takie występują.




**Rys. 3.7.** Histogram obrazu z wybranym progiem binaryzacji.

Gdy na histogramie brak jest wyraźnych maksimów, wybór progu często musi zostać dokonany przez użytkownika lub bardziej złożone algorytmy. Gdy próg dobierany jest na podstawie całego obrazu mamy do czynienia z globalnym progiem jasności, natomiast gdy dobór progu zależy od współrzędnych na obrazie jest to addytywny próg jasności.

W sytuacji gdy oświetlenie sceny jest nierównomierne i na jakąś część jednolitej powierzchni pada cień, algorytm ze stałym progiem nie będzie w stanie przetworzyć obrazu tak aby cała powierzchnia należała do jednego zbioru. Konieczne jest wtedy zastosowanie progowania adaptacyjnego dobierającego próg dla piksela zależnie od pikseli sąsiadujących wyznaczonych jako kwadratowy blok o wymiarach $n \times n$.



**Rys. 3.8.** Obraz wejściowy (po lewej), binaryzacja ze stałym progiem (środkowe), progowanie adaptacyjne (po prawej). Źródło [I6].

Operacja progowania z globalnym progiem wykonuje się przy użyciu funkcji:

*double* ***threshold****(InputArray* ***src****, OutputArray* ***dst****, double* ***thresh****, double* ***maxval****, int* ***type****)*

*src* – obraz źródłowy do konwersji.
*dst* – obraz wynikowy po konwersji, taki sam rozmiar i głębia jak *src*.
*thresh* – wartość progu.
*maxval* – maksymalna wartość używana w metodzie THRESH_BINARY i THRESH_BINARY_INV.
*type* – wybór metody progowania.

Natomiast progowanie adaptacyjne zrealizowane jest przy pomocy:

*void* ***adaptiveThreshold****(InputArray* ***src****, OutputArray* ***dst****, double* ***maxValue****, int* ***adaptiveMethod****, int* ***thresholdType****, int* ***blockSize****, double* ***C****)*

*maxValue* – niezerowa wartość przypisywana pikselom dla których został spełniony warunek.

*adaptiveMethod* – adaptacyjny algorytm używany w progowaniu.

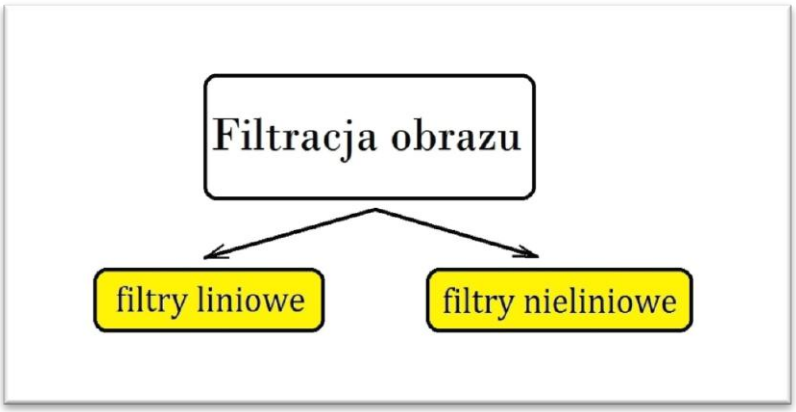
*thresholdType* – typ progowania którym musi być THRESH_BINARY lub THRESH_BINARY_INV .

*blockSize* – rozmiar bloku sąsiedztwa piksela, który jest używany do obliczania wartości progu. (Rozmiar musi być nieparzysty: 3, 5, 7, …)

*C* – stała odejmowana od średniej ważonej w algorytmie progowania. Normalnie ustawiane wartości dodatnie lecz może być równa zero lub ujemna.

### 3.3.4. Filtry

Filtracja obrazów jest jedną z metod cyfrowego przetwarzania obrazu. Wykorzystywana jest do wydobycia z obrazu szeregu informacji w celu dalszej obróbki i analizy. Polega na przeprowadzeniu operacji matematycznych na pikselach obrazu źródłowego z wykorzystaniem informacji z pikseli sąsiadujących, wyznaczając w ten sposób nowe wartości pikseli obrazu docelowego. Filtracja ma na celu np.: usunięcie szumów, wykrycie krawędzi, wygładzenie lub wyostrzenie obrazu. Ponieważ filtracja uwzględnia otocznie piksela, nazywana jest przekształceniem kontekstowym. Podział filtrów przedstawiony jest na poniższym rysunku (rys. 3.9.):




**Rys. 3.9**. Podstawowy podział filtracji.

**FILTRY LINIOWE**

Filtracja liniowa realizowana jest jako splot obrazu źródłowego z maską filtru. Maska filtru najczęściej jest macierzą kwadratową której wartości są wagami przypisywanymi poszczególnym pikselom. Wagi dla poszczególnych pikseli definiują rodzaj filtracji, natomiast wielkość maski oraz jej rozkład określa jak wielki wpływ na

obraz ma filtr. Maska powinna być dużo mniejsza niż wielkość obrazu. Typowe rozmiary macierzy są nieparzyste i wynoszą: $3 \times 3$, $5 \times 5$, $7 \times 7$ lub $9 \times 9$. Ułatwia to obliczenia, gdyż przetwarzany piksel znajduje się w środku. Przykładowa maska $3 \times 3$:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \text{(obraz źródłowy i wynikowy są tożsame)} \qquad (3.6)$$

Ze względu na sposób działania filtry liniowe dzieli się na:

- ***Dolnoprzepustowe*** – filtry te usuwają szumy i drobne detale z obrazu powodując jednoczesne wygładzenie i rozmycie obrazu poprzez uśrednienie wartości. Dzieje się to dlatego że filtr usuwa elementy o wysokiej częstotliwości, a właśnie do takich zaliczają się drobne zakłócenia w obrazie. Im większe zakłócenia chcemy usunąć tym następuje większe rozmycie obrazu. Przykładowe kombinacje filtrów rozmywających:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad (3.7)$$

- ***Górnoprzepustowe*** – filtry te wzmacniają elementy o dużej częstotliwości tłumiąc te o niskiej. Osiągane jest to poprzez zwiększenie różnicy pomiędzy sąsiednimi pikselami powodując wyostrzenie obrazu. Stosowany jest do wzmocnienia krawędzi i drobnych elementów jednak wzmacnia również zakłócenia w obrazie. Filtr tworzymy wprowadzając ujemne wagi przypisywane dla sąsiadujących pikseli. Zazwyczaj dobiera się tak współczynniki aby ich suma wynosiła 1, lecz odstępując od tej reguły można regulować wpływ filtra poprzez zmianę wartości wagi dla środkowego piksela.

Przykład filtrów wyostrzających:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} 0 & -2 & 0 \\ -2 & 11 & -2 \\ 0 & -2 & 0 \end{bmatrix} \qquad (3.8)$$

Do filtracji górnoprzepustowej zalicza się również filtry wykrywające krawędzie (gradienty, laplasjany). Różnią się one tym, że suma wartości współczynników wagowych równa się 0. Filtry krawędziowe ze względu na sposób wykrywania

dzielimy na gradientowe wykrywające krawędzie w danym kierunku lub Laplasjany umożliwiające wykrycie krawędzi w wielu kierunkach. Filtry gradientowe obliczają pierwszą, drugą i trzecią pochodną funkcji która określa zmianę jasności pikseli w obrazie. Przykładami są:

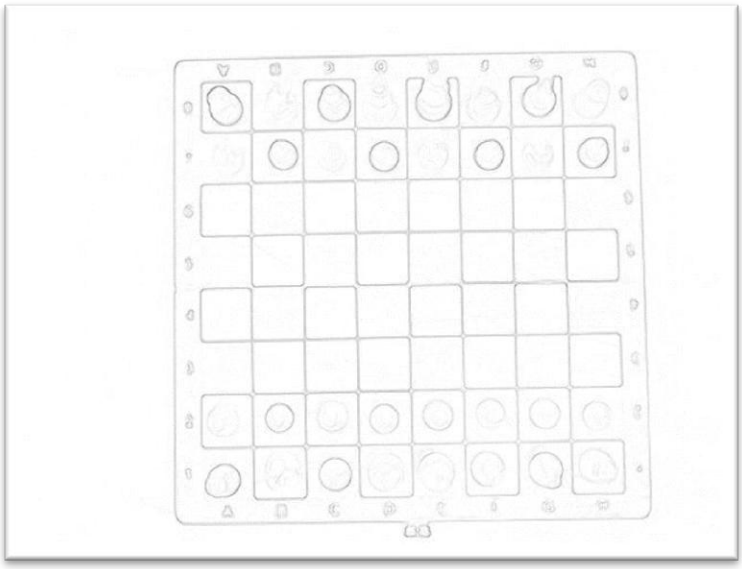
- operator Robertsa $\begin{bmatrix}0 & -1 & 0\\0 & 1 & 0\\0 & 0 & 0\end{bmatrix}$ $\begin{bmatrix}-1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 0\end{bmatrix}$ (3.9)

- operator Sobela $\begin{bmatrix}-1 & -2 & -1\\0 & 0 & 0\\1 & 2 & 1\end{bmatrix}$ $\begin{bmatrix}-2 & -1 & 0\\-1 & 0 & 1\\0 & 1 & 2\end{bmatrix}$ (3.10)

- wykrywający narożniki $\begin{bmatrix}1 & -1 & -1\\1 & -2 & -1\\1 & 1 & 1\end{bmatrix}$ $\begin{bmatrix}-1 & -1 & 1\\-1 & -2 & 1\\1 & 1 & 1\end{bmatrix}$ (3.11)

Jak można zaobserwować po wartościach maski, dany filtr wykrywa krawędzie tylko w jednym kierunku, natomiast dzięki połączeniu filtrów oraz ich obrót o 45° możliwe jest utworzenie filtrów wielokierunkowych nazywanych laplasjanami. Działanie laplasjanu polega na obliczeniu sumy drugich pochodnych cząstkowych w prostopadłych kierunkach x, y. Przykłady:

$$\begin{bmatrix}0 & -1 & 0\\-1 & 4 & -1\\0 & -1 & 0\end{bmatrix}\quad\begin{bmatrix}1 & -2 & 1\\-2 & 4 & -2\\1 & -2 & 1\end{bmatrix}\quad\begin{bmatrix}-1 & 0 & -1\\0 & 4 & 0\\-1 & 0 & -1\end{bmatrix} \tag{3.12}$$



**Rys. 3.10.** Efekt działania filtru Sobela w kierunku x, y. Obraz po inwersji.

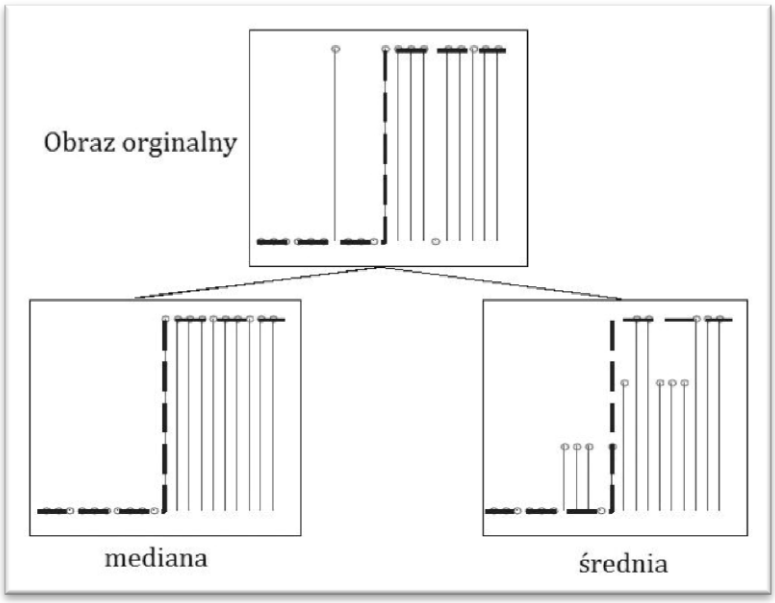
Nowa wartość wyznaczana dla piksela może przekroczyć zakres 8 bitowy dlatego wynik musi zostać przeskalowany do dopuszczalnego zakresu poprzez podzielenie przez wagę maski. Dla wartości ujemnej wagi maski, ujemny wynik splotu dla punktu można przyciąć do zera, wyznaczyć wartość bezwzględną lub przeskalować wartości wynikowe obrazu do zakresu 0 – 255.

**FILTRY NIELINIOWE**

Do filtrów nieliniowych należą filtry statystyczne i adaptacyjne . W pracy wykorzystane zostały jedynie filtry statystyczne. Do grupy tej zalicza się filtr medianowy, maksymalny oraz minimalny. Dla filtrów statystycznych podobnie jak filtrów liniowych wykorzystywana jest maska filtru, jednak w tym przypadku nie posiada ona wartości współczynników wagowych, służy jedynie do określenia otoczenia branego do obliczeń. Otoczenie piksela uwzględniane jest nie tylko przez wielkość maski lecz także poprzez wartości logiczne 0 i 1 zawarte w masce. Brane są do obliczeń jedynie te piksele na pozycji której w masce znajduje się wartość 1. Wartość wynikowa powstaje poprzez sortowanie pikseli obrazu określonych przez maskę, a następnie wybranie odpowiedniej wartości zależnie od rodzaju filtracji (wartość środkowa, minimalna, maksymalna).

- ***Filtr maksymalny i minimalny*** – działanie tych filtrów polega na wybraniu wartości min lub max piksela po ówczesnym posortowaniu całego zbioru. Filtr maksymalny nazywany jest dekompresującym lub dylatacyjnym ponieważ zwiększa jasność obrazu poprzez powiększanie obiektów jasnych. Natomiast minimalny określany jest jako kompresujący albo erozyjny, powodując zmniejszenie jasności powiększając obiekty ciemne.

- ***Filtr medianowy*** – wartość środkowa z uszeregowanego zbioru punktów otoczenia. Jeden z najczęściej stosowanych filtrów w systemach wizyjnych. Stosowany do usuwania zakłóceń punktowych w obrazie powodując nieznaczne rozmycie obrazu lub nie powodując żadnej straty jakości. Działanie jest zbliżone do filtrów dolnoprzepustowych z tą istotną różnicą, że nie powoduje utraty ostrości krawędzi. Jednak nie jest wolny od wad, powodując nadgryzanie naroży obiektów oraz wymaga większego nakładu obliczeniowego związanego z koniecznością sortowania punktów.




**Rys. 3.11.** Porównanie działania filtru medianowego i dolnoprzepustowego.

Ponieważ obliczane wartości piksela pochodzą z obrazu nie ma konieczności skalowania nowych wartości jak w przypadku filtracji liniowej.

Biblioteka OpenCV dostarcza gotowe funkcje, osobne dla każdego filtru. Rozmycie Gaussowskie przeprowadza funkcja ***GaussianBlur()***, jako parametry przyjmuje standardowo obraz źródłowy i docelowy. Następnie ***ksize*** będący dwu argumentowy określający wysokość i szerokość maski, wymiary te mogą być różne byle były całkowite i nieparzyste. Kolejno ***sigmaX, sigmaY*** oznaczające odchylenie standardowe jądra Gaussa dla kierunku x i y.

Filtr medianowy zrealizowany został przez funkcję ***medianBlur()***, wymagającą tylko obrazu źródłowego, docelowego oraz rozmiaru maski w postaci pojedynczej wartości gdyż maska jest kwadratowa.

Jednymi z funkcji wykrywających krawędzie są:

- ***Sobel()***, funkcja wymaga określenia jak większość funkcji obrazu źródłowego i docelowego, ***ddepth*** jako głębi wynikowego obrazu, ***xorder*** oraz ***yorder*** czyli wybranie kierunku wykrywania, ***ksize*** – ponieważ maska w tym przypadku jest również kwadratowa podawana jest pojedyncza wartość określająca rozmiar, ***scale*** jest opcjonalnym współczynnikiem skali dla wartości obliczanych pochodnych.

- ***Canny()***, jest to filtr wykorzystujący do wykrycia krawędzi w kierunku pionowym, poziomym i przekątnym, operatory Prewitta, Robertsa lub Sobela, jednak jego algorytm działania jest bardziej rozbudowany umożliwiając lepsze wykrycie krawędzi. Funkcja ta w OpenCV wykorzystuje operator Sobela, a parametrem ***apertureSize*** określa się jego wielkość. Algorytm wykorzystuje progowanie z histerezą mające na celu odrzucenie nieistotnych krawędzi i niedopuszczenie do podziału krawędzi w tych miejscach gdzie występuje słaby kontrast. Pierwszy i drugi próg histerezy określany jest przy pomocy ***threshold1*** i ***threshold2***. Flaga ***L2gradient*** określa czy ma być stosowana dokładniejsza norma do obliczania wielkości gradientu czy szybsza mniej dokładna.

### 3.3.5. Operacje morfologiczne

Przekształcenie morfologiczne obrazu wykorzystuje dział morfologii matematycznej oparty na teorii zbiorów. Jest to przekształcenie w wyniku którego ulega zmianie struktura lub forma obiektu. Operacje wykonywane przeważnie na obrazie binarnym jednak po pewnych modyfikacjach również na obrazie w odcieniach szarości. W wizji komputerowej najczęściej używanymi operacjami są: erozja, dylatacja zbioru oraz ich połącznie dające operację otwarcia i zamknięcia.

Operacje morfologiczne uwzględniają otoczenie analizowanego punktu, jednak nie poprzez maskę jak w przypadku filtrów lecz poprzez element strukturalny. Centralny punkt elementu strukturalnego przykładany jest do wszystkich pikseli obrazu i następuje sprawdzenie czy konfiguracja pikseli pod elementem strukturalnym odpowiada wzorcowi w tym elemencie. W sytuacji stwierdzenia zgodności zostaje wykonana operacja dla danego przekształcenia.

- ***Erozja*** – wynikiem erozji jest część wspólna wszystkich translacji obiektu A o elementy zbioru B będącego elementem strukturalnym. Erozję obszaru A przy pomocy elementu B przeprowadzić można na dwa sposoby: wyznaczana jest jako zbiór punktów centralnych elementu strukturalnego, zawartych w całości wewnątrz obszaru A lub przesuwając element strukturalny B po wewnętrznej stronie obszaru A, a kolejne środki tego elementu określają brzeg zerodowanej figury. Operacja obcina brzegi figur jednocześnie usuwając drobne wypustki oraz usuwa odizolowane drobne szczegóły mniejsze od elementu strukturalnego. Powoduje rozłącznie lekko

stykających się ze sobą figur. Działanie erozji odpowiada działaniu filtru minimalnego rozszerzając obszary jasne. Erozja matematycznie zdefiniowana jest jako:

$$A \otimes B = \bigcap_{b \in B} (A + b) \tag{3.13}$$

- ***Dylatacja*** - jest przekształceniem odwrotnym do erozji czyli wyznaczana jest jako zbiór centralnych punktów elementów strukturalnych B, których przynajmniej jeden punkt znajduje się wewnątrz obszaru figury A. Drugim sposobem zdefiniowania operacji jest przesuwanie elementu B po zewnętrznej stronie obszaru A, a środki B określają brzeg obszaru po dylatacji. Operacja zamyka drobne otwory oraz uzupełnia braki wewnątrz figur. Powoduje rozrost i połączenie blisko leżących obszarów. Dylatacja daje taki sam rezultat jak filtr maksymalny powodującego rozrost obszarów ciemnych. Definicja dylatacji:

$$A \oplus B = \bigcup_{b \in B} (A + b) \tag{3.14}$$

- ***Otwarcie i zamknięcie*** – operacje te polegają na połączeniu erozji i dylatacji umożliwiając wykorzystanie pozytywnych ich cech, nie powodując niekorzystnych zmian powierzchni figur. Otwarcie następuje przez wykonanie erozji, a następnie dylatacji na obrazie. Operacja usuwa drobne szczegóły i obiekty, powodując rozłączenie niektórych elementów bez zmiany ich wielkości. Nie wypełnia ona ubytków wewnątrz obszaru figury. Operacja zamknięcia dokonywana jest przez dylatację, a następnie erozję. Zamknięcie doprowadza do łączenia figur znajdujących się w niedużej odległości, wypełniając niewielki otwory i wcięcia pozostawiając wypustki na zewnątrz figury. Również ta operacja nie powoduje zmian wielkości figury. Operacje te matematycznie zapisuje się jako:

- *otwarcie*: $A \circ B = (A \otimes B) \oplus B$ (3.15)

- *zamknięcie:* $A \bullet B = (A \oplus B) \otimes B$ (3.16)

## 3.4. Wykrywanie konturów

Kontur jest zamkniętą krzywą reprezentowaną przez listę punktów i sposób ich łączenia. Operacja wykrywania konturów jest szczególnym rodzajem wykrywania krawędzi opisanym w rozdziale filtry liniowe. Kontury są to jedynie krawędzie zewnętrzne obiektów należące do tego samego poziomu hierarchii i tworzące zamknięty obszar. Wyznaczenie konturu można również zrealizować posługując się algorytmami morfologicznymi. Jedną z takich technik jest ekstrakcja konturu. Polega na przeprowadzeniu erozji obszaru figury A elementem strukturalnym B i odjęcie wyniku od zbioru początkowego. Kontur w ten sposób powstały jest konturem wewnętrznym zawierającym się w oryginalnym zbiorze A.

$$\kappa(A) = A - (A \otimes B) \tag{3.17}$$

Wykrycia konturu zewnętrznego wykonując operację dylatacji i odjęcia, a kontur wtedy nie zawiera się w zbiorze A.

$$\kappa(A) = (A \oplus B) - A \tag{3.18}$$

Oczywiście operacje te działają na obrazie binarnym więc konieczne jest zastosowanie metod progowania dla obrazu rzeczywistego lub poddanie go funkcji wykrywania krawędzi. Algorytm musi posiadać mechanizm uzupełniania bądź aproksymowania brakujących fragmentów krawędzi, ponieważ z założenia powinien wykrywać tylko kontury pełnych zamkniętych obiektów.

Kontur w OpenCV wykrywany jest na podstawie obrazu binarnego poprzez funkcję: *void **findContours**(InputOutputArray **image**, OutputArrayOfArrays **contours**, OutputArray **hierarchy**, int **mode**, int **method**).*

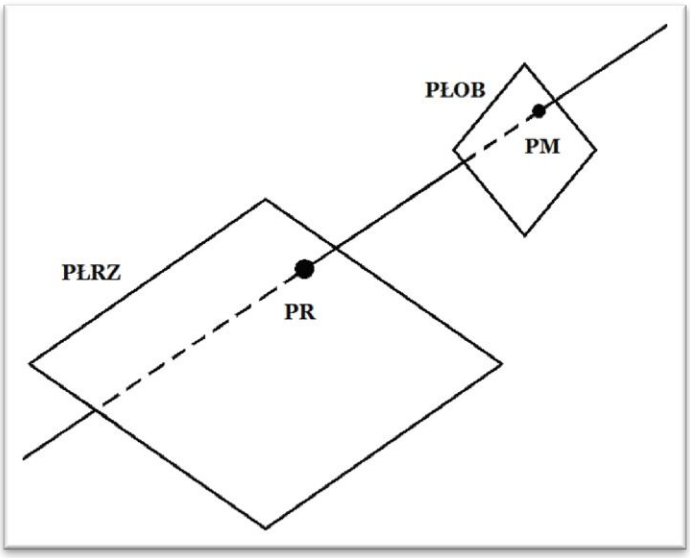
*image* – 8 bitowy obraz źródłowy traktowany jako binarny. Każda niezerowy piksel traktowany jako wartość 1.
*contours* – wektor punktów przechowujący wykryte kontury.
*hierarchy* – opcjonalny wektor informujący o topologii obrazu. Określa poziomy zagłębienia oraz hierarchię konturów.
*mode* – określa w jaki sposób mają być zwracane znalezione kontury.
*method* – metoda aproksymacji konturu.

Szczegółowe informacje na temat realizacji metody wykrywania konturów w OpenCV dostępne są w publikacji [10].

## 3.5. Transformata płaska

Transformacja jest to przekształcenie przyporządkowujące jednoznacznie elementowi pewnego zbioru funkcji inny zbiór funkcji, także przekształcenie pewnej przestrzeni w inną.

Metoda czteropunktowej identyfikacji transformaty płaskiej[6] określa położenie obiektu w przestrzeni dwuwymiarowej przy pomocy sceny zarejestrowanej przez kamerę. Metoda nie wymaga stosowania jednakowej skali osi współrzędnych i wykorzystuje się w niej proste operacje algebry liniowej. Działanie transformacji polega na wyznaczeniu parametrów określających relację między płaszczyzną rzeczywistą PŁRZ, a płaszczyzną PŁOB obrazu zarejestrowanego na matrycy kamery, dzięki czemu będzie możliwe przyporządkowanie punktu z PŁOB, punktowi znajdującemu się na PŁRZ.




**Rys. 3.12.** Graficzna interpretacja transformacji płaskiej.

Oznaczając transformację płaską jako funkcję TP, punkt na płaszczyźnie matrycy prze $PM = \begin{bmatrix} xm \\ ym \end{bmatrix}$, a punkt w przestrzenie rzeczywistej $PR = \begin{bmatrix} xr \\ yr \end{bmatrix}$, otrzymuje się postać:

$$PR = TP(PM) = \frac{1}{\alpha \cdot xm + \beta \cdot ym + 1} \cdot \begin{bmatrix} a_1 \cdot xm + b_1 \cdot ym + c_1 \\ a_2 \cdot xm + b_2 \cdot ym + c_2 \end{bmatrix} \qquad (3.19)$$

Przed wyznaczeniem transformaty konieczna jest kalibracja kamery czyli przeprowadzenie identyfikacji transformaty, która wyznacza wartości współczynników $WSP = [a_1, b_1, c_1, a_2, b_2, c_2, \alpha, \beta]$, jednoznacznie określa parametry wewnętrzne kamery oraz jej geometryczne ustawienie w stosunku do płaszczyzny rejestrowanej sceny. Nie jest wymagana ortogonalność układu współrzędnych, ani współosiowość płaszczyzn. Do przeprowadzenia identyfikacji 8 parametrów konieczne jest podanie czterech punktów (żadne trzy z nich nie mogą leżeć na jednej prostej) wzorcowych na płaszczyźnie rzeczywistej i odpowiadających im punktów na płaszczyźnie obrazu. Po odpowiednich przekształceniach równań dla czterech punktów otrzymuje się układ równań liniowych w postaci macierzy kwadratowej:

$$C = \begin{bmatrix} xm_1 & ym_1 & 1 & 0 & 0 & 0 & -xm_1 * xr_1 & -ym_1 * xr_1 \\ 0 & 0 & 0 & xm_1 & ym_1 & 1 & -xm_1 * yr_1 & -ym_1 * yr_1 \\ xm_2 & ym_2 & 1 & 0 & 0 & 0 & -xm_2 * xr_2 & -ym_2 * xr_2 \\ 0 & 0 & 0 & xm_2 & ym_2 & 1 & -xm_2 * yr_2 & -ym_2 * yr_2 \\ xm_3 & ym_3 & 1 & 0 & 0 & 0 & -xm_3 * xr_3 & -ym_3 * xr_3 \\ 0 & 0 & 0 & xm_3 & ym_3 & 1 & -xm_3 * yr_3 & -ym_3 * yr_3 \\ xm_4 & ym_4 & 1 & 0 & 0 & 0 & -xm_4 * xr_4 & -ym_4 * xr_4 \\ 0 & 0 & 0 & xm_4 & ym_4 & 1 & -xm_4 * yr_4 & -ym_4 * yr_4 \end{bmatrix}; \quad (3.20)$$

$$H = [xr_1 \quad yr_1 \quad xr_2 \quad yr_2 \quad xr_3 \quad yr_3 \quad xr_4 \quad yr_4]^T; \quad (3.21)$$

$$WSP = [a_1 \quad b_1 \quad c_1 \quad a_2 \quad b_2 \quad c_2 \quad \alpha \quad \beta]^T; \quad (3.22)$$

$$H = C \cdot WSP; \quad (3.23)$$

Równanie na WSP wygląda następująco:

$$WSP = inv(C) \cdot H; \quad (3.24)$$

Metodę można zastosować dla większej ilości punktów pomiarowych, jednak metoda czteropunktowa w zupełności wystarcza na potrzeby pracy.

## 3.6. Transformata Hougha

Transformata Hougha [7] (HT – Hough Transform), służy jako metoda do wykrywania wzorca na obrazach binarnych poprzez wyznaczenie parametrów opisujących wzorce. Sprowadza się do poszukiwania lokalnych maksimów w przestrzeni parametrów.

HT odwzorowuje przestrzeń kartezjańską opisaną parametrami (*x,y*) w przestrzeń Hougha reprezentowaną zależnie od metody np. wykrywania linii, przez (*r, theta*). Transformata przeprowadzana jest niezależnie dla każdego z punktów co daje możliwość wykonywania równoległych obliczeń w systemach wielowątkowych, zwiększając przez to jej wydajność. Dzięki niezależnym obliczeniom dla każdego fragmentu możliwe jest rozpoznanie kształtów obiektów które zostały częściowo zniekształcone, np.: brak ciągłości krawędzi, niespójne punkty należące do krawędzi. Jednakże transformata nie jest wolna od wad. Jedną z nich jest konieczność stosowania dużej przestrzeni akumulacyjnej będącą tablicą wielowymiarową (wymiar zależny od ilości parametrów) przechowujących parametry transformaty dla każdego z punktów.

Transformata stała się bardzo popularną metodą wykorzystywaną do zagadnienia wykrywania kształtów opisanych równaniami:

- linii prostych
- okręgów

Umożliwia również wykrywanie bardziej skomplikowanych i nieregularnych kształtów jak elipsy czy odciski palców.

Dostępnych jest wiele wersji transformaty t.j. klasyczna transformata Hougha (SHT), szybka transformata (FHT), zmodyfikowana szybka transformata (MFHT), metoda Geriga i Kleina (GKHT), transformata Geriga z gradientem (GHTG), 2-1 Hough Transform (21HT), probabilistyczna transformata (PPHT). W OpenCV dostępne są tylko niektóre z nich. Wykrywanie linii realizowane jest przy wykorzystaniu SHT wyznaczającej orientację linii oraz PPHT wyznaczającej poza orientacją również ich zakres. Natomiast jedyną zaimplementowaną metodą wykrywania okręgów jest 21HT, zmniejsza ona zapotrzebowanie pamięci przez podział wyszukiwania okręgów na dwa etapy.

Etapy przetwarzania obrazu dla HT:

- wstępna analiza – usunięcie zakłóceń, uwydatnieniu szczegółów obiektów
- progowanie – transformata przeprowadzana jest na obrazie binarnym
- filtry krawędziowe – wykrycie krawędzi obiektów

Aby było możliwe właściwe wykrycie linii i okręgów istotny jest właściwy dobór progu binaryzacji i algorytmu wykrywania krawędzi.

### 3.6.1.Hough Lines

Linię prostą na płaszczyźnie można przedstawić przy pomocy równania kierunkowego $\boldsymbol{y = ax + b}$ (lub $\boldsymbol{x = x_0}$ gdy prosta jest równoległa do osi OY), oraz w postaci ogólnej $\boldsymbol{Ax + By + C = 0}$. Równanie po normalizacji ma postać:

$$x\cos\theta + y\sin\theta = r \tag{3.25}$$

gdzie: $r - promień\ wodzący$
$\theta - kąt\ między\ r, a\ osią\ OX$

Znając współrzędne dwóch punktów $A = (x_1, y_1), B = (x_2, y_2)$ przez które przechodzi prosta, możliwe jest wyznaczenie równania ją opisującego:

$$(x_2 - x_1)(y - y_1) = (y_2 - y_1)(x - x_1) \tag{3.26}$$

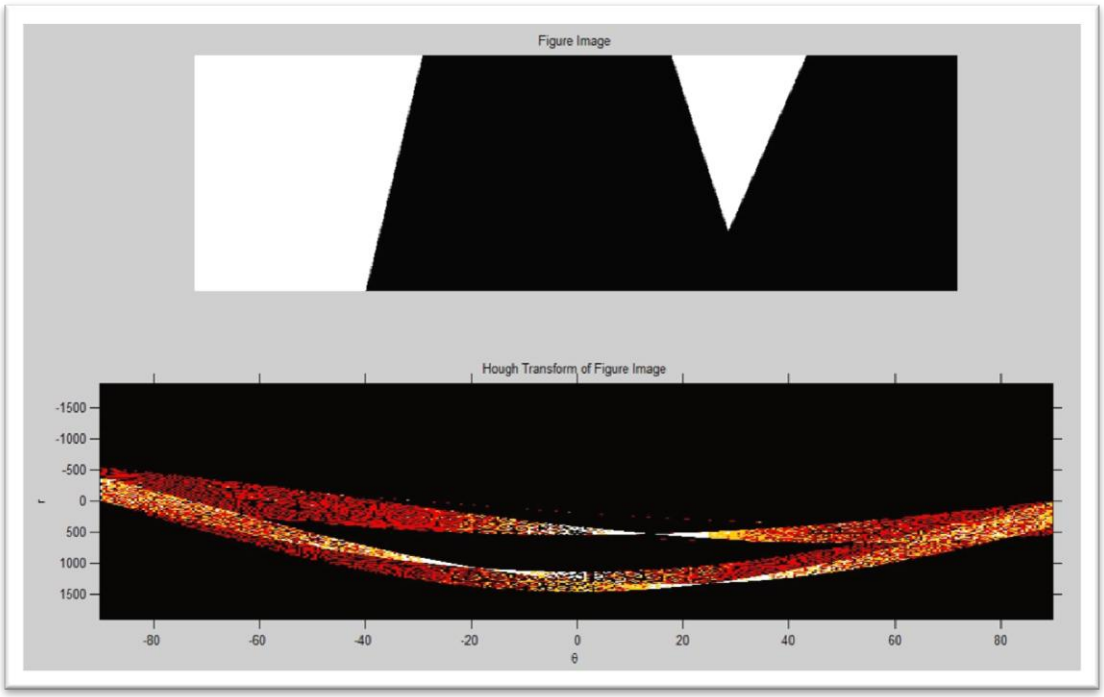
Problem pojawia się wtedy, gdy na obrazie znajduje się większa liczba punktów należących do różnych krawędzi, a proces rozpoznania ma automatycznie przeprowadzić komputer rozumiejący obraz inaczej niż ludzkie oko. W tym przypadku konieczne jest zastosowanie metod takich jak transformata Hougha, które dzięki swoim właściwościom wykrywają współliniowe punkty.

Właściwości HT wykorzystywane w wykrywaniu prostych:

- *Punkt* w układzie kartezjańskim reprezentowany jest jako *krzywa sinusoidalna* w przestrzeni Hougha.
- *Prosta* z przestrzeni kartezjańskiej reprezentowana jest jako *punkt* w przestrzeni Hougha opisany parametrami $(r, \theta)$.

Przez każdy analizowany punkt różny od zera w układzie kartezjańskim, przeprowadzany jest pęk prostych, które są transformowane do przestrzeni Hougha tworząc sinusoidę. Dla punktów leżących na jednej prostej, odpowiadające im sinusoidy przecinają się w jednym punkcie przestrzeni Hougha wyznaczając tą prostą i jej parametry $(r, \theta)$, podstawiane później do równania znormalizowanego. Im więcej punktów należy do prostej, tym większa liczba sinusoid przecina się tworząc lokalne

maksimum. Gdy na obrazie znajduje się więcej prostych, tworzą się osobne maksimum dla każdej z nich. Przeważnie dla obrazów rzeczywistych nie występuje jeden punkt przecięcia tylko skupisko punktów. Przy określaniu przynależności punktów do danej prostej wykorzystuje się histogram. Dobór rozdzielczości histogramu ma istotne znaczenie na dokładność wykrywania parametrów prostej.




**Rys. 3.13.** Transformata Hougha dla Linii. Przykład w Matlab'ie.

Jak już wcześniej wspomniano, przestrzeń Hougha jest to przestrzeń akumulacyjna będąca dwuwymiarową tablicą, której jeden wymiar reprezentuje wartość promienia $r$, a drugi kąt $\theta$. Dokładność i czas obliczeń zależy od poziomu kwantyzacji skali wartości w tablicy.

Po wyznaczeniu maksimów w przestrzeni Hougha można wyznaczyć parametry $a, b$ równania kierunkowego stosując zależność:

$$a = -\frac{\cos\theta}{sin\theta}, \qquad b = \frac{r}{\sin\theta}, \tag{3.27}$$

W bibliotece OpenCV transformatę Hougha realizuje funkcja:

*void* ***HoughLines****(InputArray* ***image****, OutputArray* ***lines****, double* ***rho****, double* ***theta****, int* ***threshold****, double* ***srn****=0, double* ***stn****=0)*

*image* – jednokanałowy, 8 bitowy obraz binarny.
*lines* – wyjściowy wektor przechowujący parametry $(r, \theta)$ wykrytych linii.

*rho* – rozdzielczość akumulatora w pikselach dla $r$.
*theta* – rozdzielczość akumulatora w radianach dla $\theta$.
*threshold* – próg, powyżej którego wartość komórki akumulatora zakwalifikowana jest jako linia.
*srn, stn* – dzielniki rozdzielczości akumulatora dla $r$ i $\theta$ w wielozakresowej transformacji Hougha. Dla klasycznej transformaty Hougha wartości są równe zero.

Dostępna jest również wersja Probabilistycznej Transformaty Hougha, która dodatkowo oblicza zakres linii oraz nie bada wszystkich punktów w przestrzeni kartezjańskiej, zwiększając przez to wydajność obliczeniową.

*void* ***HoughLinesP****(InputArray* ***image****, OutputArray* ***lines****, double* ***rho****, double* ***theta****, int* ***threshold****, double* ***minLineLength****=0, double* ***maxLineGap****=0 )*

*minLineLength* – minimalna długość linii, odcinki krótsze są odrzucane.
*maxLineGap* – maksymalna dopuszczalna różnica między punktami na tej samej linii do połączenia ich.

### 3.6.2. Hough Circles

Okrąg można przedstawić w postaci kanonicznej:

$$(x - x_S)^2 + (y + y_S)^2 = r^2 \tag{3.28}$$

gdzie:

$r -$ promień okręgu
$S = (x_S, y_S) -$ współrzędne środka okręgu

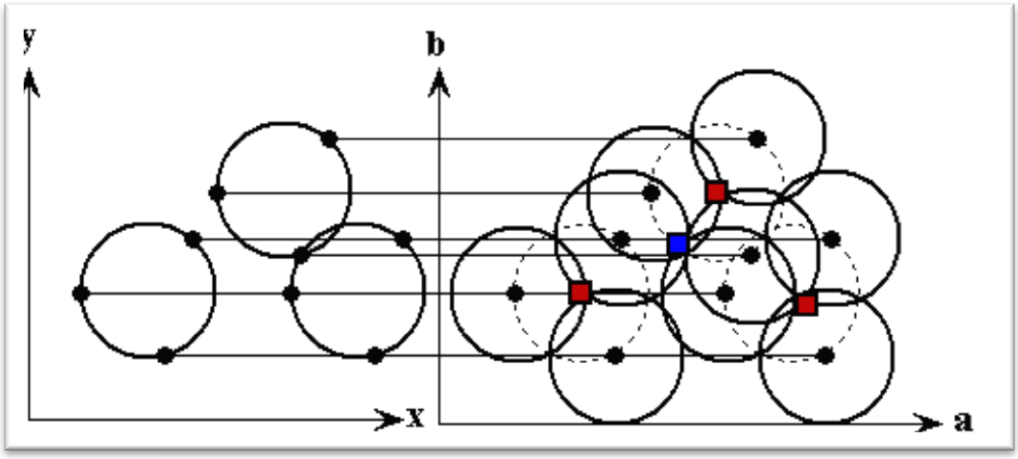
lub w postaci parametrycznej:

$$\begin{aligned} x &= x_S + r\cos\theta \\ y &= y_S + r\sin\theta \end{aligned} \tag{3.29}$$

Do wykrycia okręgów na obrazie można posłużyć się dwoma metodami. Pierwsza, podobna do transformaty dla linii, wymagająca trójwymiarowego akumulatora, ponieważ operuje na trzech parametrach $(x, y, r)$. Transformata dla okręgów również operuje na obrazie binarnym po detekcji krawędzi, przez co analizowane punkty stanowią fragment okręgu i stąd znane są wartości $(x, y)$. Natomiast nie jest znane położenie środka okręgu, dlatego obliczenia przeprowadzane są dla różnych wartości promienia $r$.

Algorytm działania metody:

1) dla każdego punktu z przestrzeni $(x, y)$ zostaje wyrysowany okrąg w przestrzeni $(a, b)$ o środku w tym punkcie i ustalonym promieniu $r$.
2) zostaje zmieniona wartość promienia $r$, po czym powtarzana jest operacja z punktu (1).
3) współrzędne szukanego okręgu oraz jego promień wyznaczają maksima w przestrzeni H wyznaczone na przecięciach wyrysowanych okręgów.

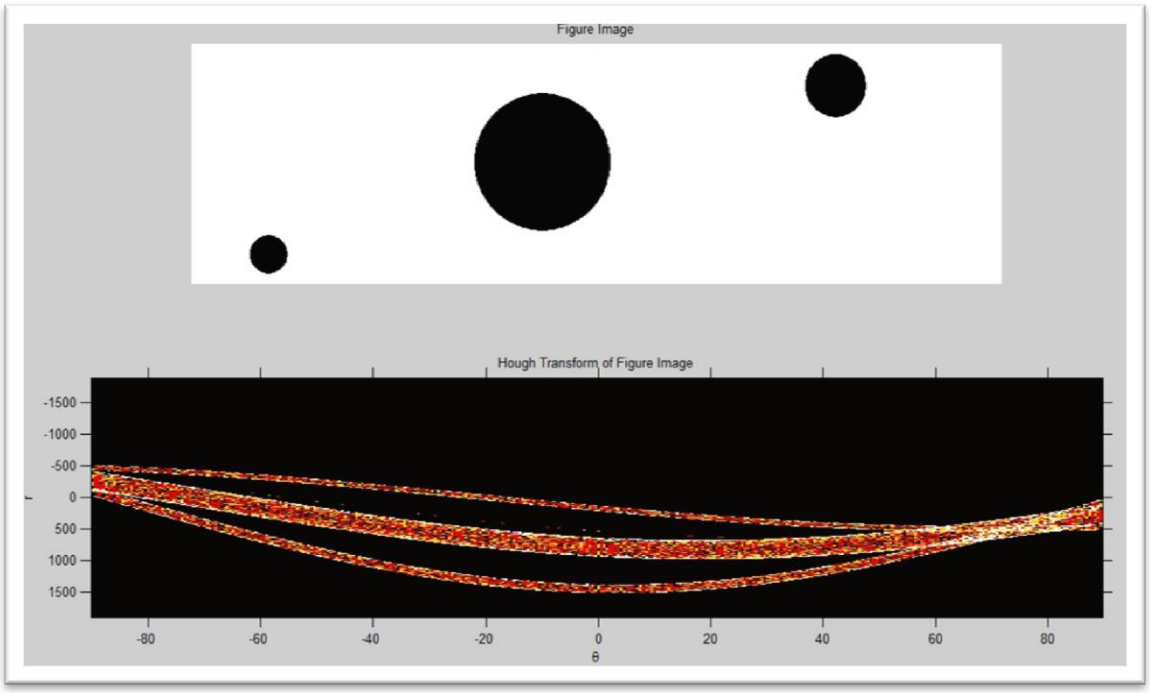



**Rys. 3.14.** Wykrywanie okręgów transformatą Hougha dla stałego parametru $r$.

Metoda ze względu na trzy wymiary jest dość wymagająca pamięciowo i obliczeniowo. Możliwe jest jej przyspieszenie poprzez przyjęcie stałego promienia $r$ lub jego ograniczenie jeśli spodziewamy się tylko okręgów o określonej wielkości.

Drugą metodą jest wykorzystanie klasycznej transformaty Hougha dla prostych i odpowiednią interpretację przestrzeni $(r, \theta)$. Jak w przypadku linii prostych dla każdego punktu okręgu wyrysowywany jest pęk prostych, a następnie transformowany do postaci krzywych sinusoidalnych w przestrzeni Hougha. Krzywe te przyjmują charakterystyczną postać wstęgi, której brzegi określają parametry okręgu w układzie kartezjańskim. Promień okręgu wyznaczony jest jako odległość pomiędzy brzegami wstęgi, natomiast środek okręgu wymaga bardziej skomplikowanej interpretacji:

$$\begin{aligned} d &= abs(rho) - r; \\ x_S &= \cos(theta) \cdot d; \\ y_S &= \sin(theta) \cdot d; \end{aligned} \tag{3.30}$$




**Rys. 3.15.** Transformata Hougha dla okręgów. Przykład w Matlab'ie.

W OpenCV został zaimplementowany tylko jeden algorytm [**21HT**] wykrywania okręgów który jest dość odporny na zakłócenia, a jego zasada działania bazuje na pierwszej metodzie używającej trójwymiarowej przestrzeni akumulacyjnej. Zrealizowany jest poprzez funkcję:

*void* ***HoughCircles****(InputArray* ***image****, OutputArray* ***circles****, int* ***method****, double* ***dp****, double* ***minDist****, double* ***param1****=100, double* ***param2****=100, int* ***minRadius****=0, int* ***maxRadius****=0 )*

*image* – jednokanałowy, 8 bitowy obraz binarny.

*circles* – wyjściowy wektor przechowujący parametry $(x, y, r)$ wykrytych okręgów zakodowanych jako 3 elementowy wektor zmiennoprzecinkowych punktów.

*method* – użyta metoda wykrywania. Obecnie dostępna tylko metoda CV_HOUGH_GRADIENT bazująca na 21HT.

*dp* – odwrotny stosunek rozdzielczości akumulatora do rozdzielczości obrazu.

*minDist* – minimalna odległość pomiędzy środkami wykrytych okręgów.

*param1* – parametr określający próg w funkcji wykrywania krawędzi $Canny()$ wewnątrz algorytmu transformaty.

*param2* – próg akumulatora na etapie wykrywania środków okręgów. Im mniejszy tym większa możliwość wykrycia fałszywych okręgów.

*minRadius* – minimalny promień okręgu.

*maxRadius* – maksymalny promień okręgu.

# 4. Realizacja pracy

## 4.1. Interfejs użytkownika

Pomimo iż aplikacja z założenia ma działać w pełni automatycznie, konieczne jest umożliwienie interakcji użytkownika z programem. Funkcje odpowiedzialne za tworzenie interfejsu znajdują się w bibliotece HighGUI będącej częścią OpenCV i dostępnej po dodaniu pliku nagłówkowego *highgui.h*. Wyszczególnić można dwie części interfejsu. Pierwsza odpowiedzialna jest za obsługę kamery, plików wideo i obrazu, poprzez ich odczyt i zapis w wielu dostępnych formatach. Druga część odpowiada za tworzenie okien służących do wyświetlenia wcześniej pobranego obrazu i oddziaływanie na parametry programu poprzez suwaki oraz obsługę myszy i klawiatury.

### 4.1.1. Odczyt i zapis

Obraz poddawany analizie może pochodzić z dwóch opcjonalnych źródeł jakimi są kamera internetowa lub plik video. Do jego pobrania służy klasa *VideoCapture* w której jako parametry konstruktora podaje się nazwę i typ pliku video lub numer określający urządzenie z którego będzie pochodzić obraz, w tym przypadku numer kamery.

```
VideoCapture cap(0);                  // połączenie z kamerą
VideoCapture cap("newchess.mov");     // połączenie z plikiem wideo,
```

Utworzony w ten sposób obiekt klasy przechowuje wskaźnik do pierwszej klatki obrazu. Odczytanie kolejnej klatki odbywa się przy pomocy operatora '>>':

```
cap >> frame;
```

gdzie *frame* jest obiektem klasy *cv::Mat* do którego przekazany zostaje obraz z klatki wideo.

Odczyt i zapis obrazu oraz zapis pliku wideo nie jest potrzebny w głównym algorytmie działania aplikacji. Zostanie jednak w tym miejscu przedstawiony ponieważ wykorzystywany był w celach testowania i prezentacji wyników poszczególnych metod przetwarzania i rozpoznawania.

Zapis do pliku wideo wymaga utworzenia obiektu służącego do zapisu przed użyciem funkcji zapisu.

Obiekt tworzony jest przez:

```
VideoWriter record("HoughCircle.avi", CV_FOURCC('D','I','V','X'), 10, frame.size(), true);
```

Jako parametry podaje się nazwę i typ tworzonego pliku, rodzaj kodeka używanego do kodowania (zależne od formatu pliku), liczba klatek na sekundę, rozdzielczość obrazu i zmienna logiczna określające czy obraz ma być kolorowy. Zapis kolejnej klatki do pliku wykonuje się przy pomocy operatora '<<':

```
record << writeImage;
```

*writeImage* jest klatką obrazu która zostanie zapisana do pliku wideo.

Do odczytu i zapisu obrazu z pliku używa się funkcji:

```
Image = imread("chess1.jpg");
imwrite(„newChess.jpg”, Image);
```

***imread***() zwraca obraz do obiektu klasy ***cv::Mat***. Funkcja wymaga podania nazwy pliku, z ewentualną ścieżką lokalizacji, jeśli znajduje się w innym folderze niż plik projektu oraz opcjonalną flagę określającą typ koloru ładowanego obrazu (domyślnie *flags = 1*, kolorowy obraz trzykanałowy). Natomiast ***imwrite***() poza nazwą pliku do którego zostanie zapisany obraz, wymaga podania nazwy obiektu zawierającego obraz do zapisu. ***VideoCapture*** i ***VideoWriter*** posiadają metody pozwalające sprawdzić czy zostało utworzone połączenie z wczytywanym lub zapisywanym plikiem, natomiast ***imwrite()*** zwraca wartość *bool* informującą o poprawności zapisu.

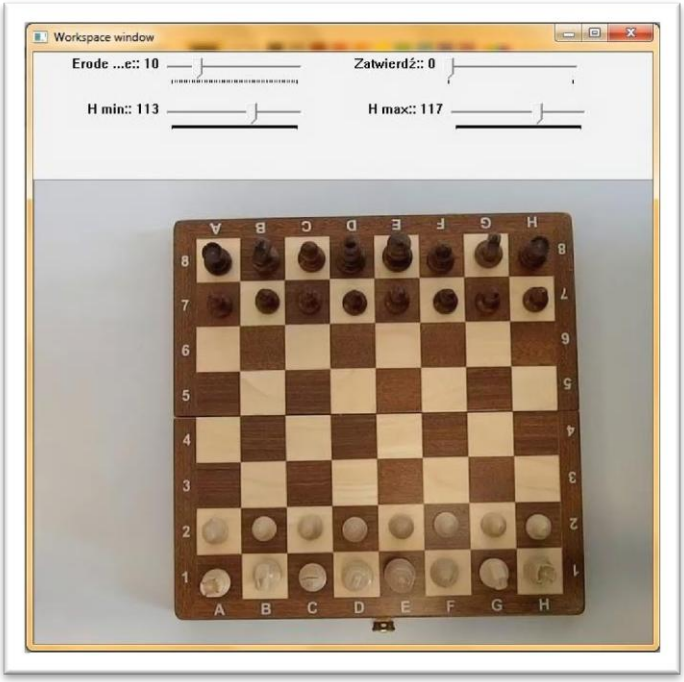
Notacja algebraiczna przebiegu gry zapisywana jest do pliku tekstowego. Odbywa się to przy pomocy biblioteki **iostream.** Udostępnia ona cały interfejs umożliwiający obsługę operacji we/wy na plikach i obiektach klasy *string*. Aby mieć dostęp do obsługi plików znajdujących się na dysku lub innych nośnikach należy włączyć plik **fstream**. Zapis i odczyt z pliku umożliwia klasa o tej samej nazwie co plik. Utworzenie strumienia umożliwiającego tylko zapis do pliku, odbywa się przez klasę **ofstream.** natomiast odczyt przy pomocy **ifstream**.

```
ofstream note_file;                  //deklaracja strumienia do plików
note_file.open("algebchess.txt");    //utworzenie pliku i otworzenie połączenia
        if(!note_file.is_open())     //sprawdzenie czy połączono
                return -1;
note_file << config.note_move <<"\n"; //zapis zawartości stringu do pliku
```

### 4.1.2. Tworzenie okien i suwaków

Najistotniejszą częścią interfejsu dla użytkownika są okna, umożliwiające podgląd obrazu oraz zmian jakie w nim następują po działaniu różnego rodzaju funkcji. Okno tworzone jest przy użyciu funkcji ***namedWindow****("Main window", 1).* Należy przy tym podać nazwę okna która będzie później używana w celu jego jednoznacznego określenia oraz flagę definiującą zachowanie się wymiarów okna. W wersji OpenCV 2.3 dostępny jest tylko schemat *CV_WINDOW_AUTOSIZE* (zdefiniowane jako wartość *1*, można używać zamiennie) automatycznie dopasowujący wielkość okna do wymiarów obrazu. Użycie każdej innej flagi powoduje ustawienie skalowania okna bez zachowania proporcji. Ustawienie okna w wybranym położeniu wykonuje się poprzez: ***cvMoveWindow****("Main window", 0, 0).* Wersja użyta do projektu korzysta jeszcze ze starej funkcji używanej w języku C. W parametrach funkcji posługuje się nazwą wcześniej utworzonego okna, które jest pozycjonowane wraz ze współrzędnymi pikselowymi monitora, w których to będzie znajdować się lewy górny róg okna. Do wyświetlenia obrazu służy polecenie ***imshow****("Main window", currFrame);* którego standardowymi parametrami są nazwa okna w którym będzie wyświetlany oraz zmienna przechowująca obraz. Funkcja automatycznie wykrywa jego głębię i inne parametry, aby prawidłowo go wyświetlić.

Aby kontrolować ustawienia programu użytkownik ma do dyspozycji możliwość tworzenia suwaków. Pozwalają one wpływać na parametry różnych funkcji w trakcie działania aplikacji. Tworzone są one przez ***createTrackbar****("Erode value:", "Workspace window", &param.erode_number, 50, NULL).* Pierwszym parametrem jest opis suwaka przy nim wyświetlany, następnie definiuje się okno do którego suwak ma być przypięty i podaje się zmienną do której wprowadzana będzie wartość ustawiona przy jego pomocy. Kolejno podawana jest maksymalna wartość jaką można ustawić suwakiem, przy czym wartością początkową zawsze jest zero. Suwaki operują na liczbach całkowitych typu *int*. Jako parametr funkcji można ustawić wskaźnik do funkcji typu *void* która będzie wywoływana po każdej zmianie pozycji suwaka.




**Rys. 4.1.** Widok utworzonego okna z suwakami.

Biblioteka nie dostarcza takich elementów jak przyciski, a projekt konsolowy również nie daje możliwości dodania ich w prosty sposób. Do utworzenia przycisku można posłużyć się suwakiem ustawiając jego maksymalny zakres na jeden otrzymując suwak z dwoma położeniami 0 – 1. Aby suwak po przełączeniu w pozycję jeden i wykonaniu przypisanej temu zdarzeniu czynności powrócił do pozycji zero zastosować należy funkcję obsługującą zdarzenie zmiany pozycji suwaka wykorzystując ***setTrackbarPos****("Zatwierdź:", "Workspace window", 0);* Funkcje tworzące okna i suwaki umieszczone są w funkcji ***createWindowsAndTrackbar****( frame, config );* Parametrami jakie przyjmuje funkcja jest obiekt *config* klasy *parametry* którego zmienne będą przechowywać wartości ustawiane przez suwaki oraz macierz z obrazem, na podstawie którego ustawiana jest pozycja okien.

### 4.1.3. Parametry konfiguracyjne programu

Klasa *parametry* służy do przechowywania wartości zmiennych ustawianych suwakami, wczytywanych z klawiatury lub wyliczanych w programie. Zebrane w niej są zmienne które odpowiadają za sterowanie wykonaniem programu oraz parametry używane w różnych klasach i funkcjach.

W klasie tej przechowywane są parametry których wartości przekazywane są do funkcji określając ich działanie (t.j. ilość operacji erozji, zakres dla składowej H,

wybór źródła obrazu, ile okręgów zostało znalezionych), zmienne bool'owskie służące jako flagi sterujące wykonaniem poszczególnych algorytmów programu. Flagi te określają m.in. który gracz wykonuje aktualnie ruch, czy wykryto ruch na planszy, kiedy uruchomić funkcję wykrywającą figury, informacje o błędnym działaniu, czy wykryto awans pionka. Obiekt tej klasy zawiera również obiekty typu *Mat* będące macierzami z obrazem przetwarzanym i przekazywanym do innych funkcji w celu dalszego przetwarzania i analizy oraz struktury przechowujące współrzędne wykrytych okręgów dla których sprawdzany jest wykonany ruch. Inicjalizacja wartości początkowych zmiennych następuje w konstruktorze i określa parametry początkowe rozgrywki.

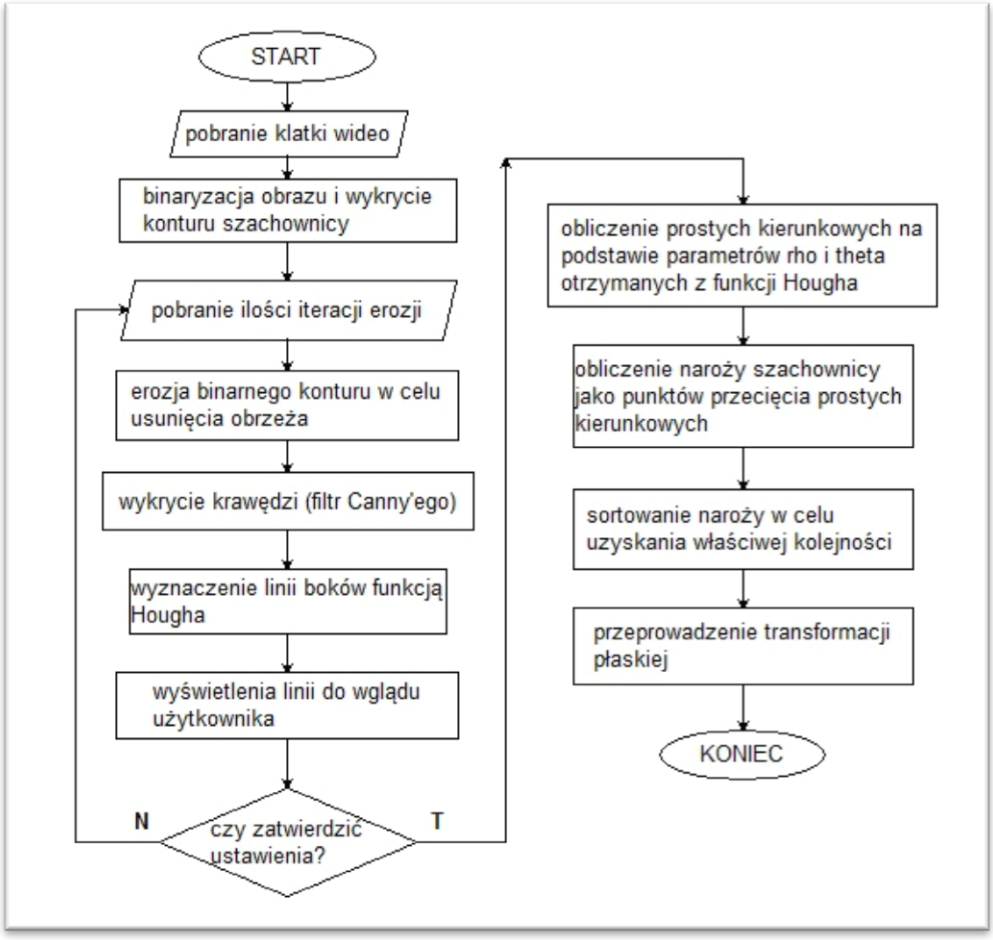
## 4.2. Wyznaczenie współrzędnych szachownicy

Rzeczywista pozycja figury na szachownicy wyznaczana jest przez funkcję ***transformata::tp*** *(int xm, int ym);* będącej funkcją składową klasy *transformata,* realizującej czteropunktową transformatę płaską. Do prawidłowego działania tej funkcji potrzebne jest podanie współrzędnych *xm*, *ym* na obrazie poszukiwanego punktu oraz wcześniejsza identyfikacja współczynników WSP. Obliczenie tych współczynników jest czasochłonne, lecz wykonywane jest jednokrotnie na początku działania aplikacji. Po wyznaczeniu współczynników transformaty, nie można zmieniać ustawienia kamery i szachownicy. Natomiast sama funkcja ***tp****(xm,ym)* może być uruchamiana wielokrotnie podczas działania programu, gdyż obliczenia w niej wykonywane są szybko, nie powodując opóźnień w wykonaniu algorytmu programu. Do przeprowadzenia inwersji macierzy przy identyfikacji transformaty wykorzystano gotową bibliotekę wykonującą operacje macierzowe zrealizowaną w *class macierz.* Biblioteka nie jest własnością autora pracy, została zrealizowana w ramach projektu studenckiego na kierunku Informatyka – Instytutu Politechnicznego PWSZ w Tarnowie i udostępniona w Internecie [I11].

Do przeprowadzenia identyfikacji transformaty płaskiej konieczne jest przekazanie wartości współrzędnych rzeczywistych naroży szachownicy przy pomocy funkcji ***transformata::setReal*** *(double [], int),* oraz współrzędnych monitorowych naroży przez funkcję ***transformata::setMon*** *(int []),* w której następuje identyfikacja współczynników WSP. Przekazując współrzędne narożników do funkcji identyfikującej należy ustawić je w takiej kolejności, aby punkty naroży na płaszczyźnie obrazu odpowiadały punktom naroży płaszczyzny szachownicy.

Ustawienie boków planszy równoległe do boków obrazu i ustawienie rogów zawsze w tych samych punktach na obrazie jest trudne do osiągnięcia. Pociąga to za sobą konieczność oprogramowania takich funkcji, które niezależnie od ułożenia boków i wielkości szachownicy pozwolą wyznaczyć położenie figur według współrzędnych szachowej notacji algebraicznej. Ograniczeniem jest kąt między górnym brzegiem szachownicy i górnym bokiem obrazu, przyjęto że nie powinien przekroczyć 45 stopni, gdyż może zostać wyznaczona niewłaściwa kolejność naroży. Plansza szachowa musi się odróżniać się od powierzchni na której się znajduje, a sama powierzchnia powinna być możliwie jednolita. Pojawienie się w kadrze innych

krawędzi lub linii może zakłócić proces analizy obrazu. Przed wyznaczeniem współczynników WSP konieczne jest wykonanie operacji dzięki którym obliczone zostaną współrzędne naroży na płaszczyźnie monitorowej. Operacje związane z szachownicą, realizowane są w stworzonej w tym celu klasie *szachownica*.

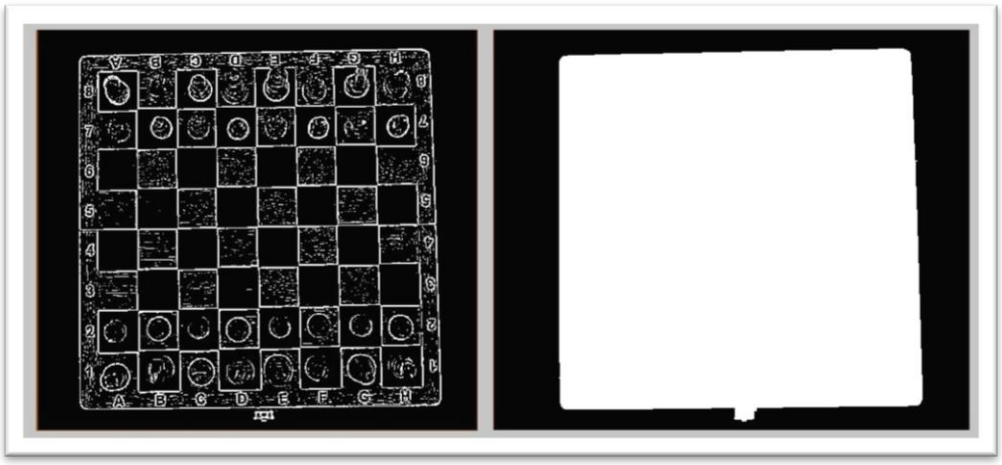



**Rys. 4.2.** Algorytm wykrywania współrzędnych naroży i kalibracji płaszczyzn.

### 4.2.1. Wykrycie konturu

Obraz wczytany przy pomocy OpenCV jest w przestrzeni BGR, a nie jak przeważnie RGB, na co należało zwrócić uwagę, ponieważ podczas konwersji trzeba użyć właściwego schematu *CV_BGR2GRAY.* Obraz skonwertowany został do skali szarości i poddany działaniu progowania adaptacyjnego, gdyż funkcja wykrywająca kontury operuje na obrazie binarnym, a zastosowane progowanie radzi sobie bardzo dobrze nawet z obrazami nierównomiernie oświetlonymi. Obraz binarny przed wykryciem konturów poddawany jest pięciokrotnej operacji dylatacji, a następnie erozji w celu domknięcia obszaru szachownicy i jego wygładzenia. Jedną z opcji

funkcji ***findContours***() jest **CV_RETR_EXTERNAL** powodująca wyszukiwanie tylko najbardziej zewnętrznych konturów w hierarchii. Jako metoda przybliżenia konturu jest ustawiona **CV_CHAIN_APPROX_SIMPLE,** kompresująca punkty leżące w jednej linii konturu do punktu początkowego i końcowego takiego odcinka, przez co idealny kwadrat miałby zapis konturu w postaci czterech punktów. Następnie kontur jest rysowany na obrazie funkcją ***drawContours***() z opcją wypełnienia środka otrzymując jednolitą powierzchnię planszy. Przykład na rysunku niżej:



**Rys. 4.3.** Obraz po wykonaniu progowania adaptacyjnego – po lewej, oraz wykryciu konturu i wyrysowaniu go – po prawej.

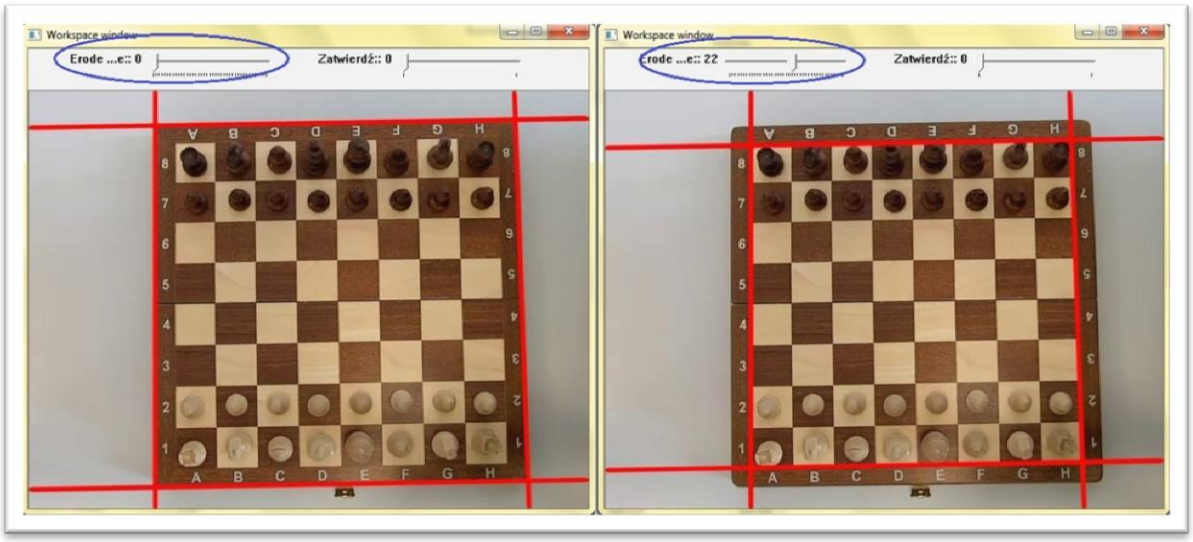
Uzyskany obraz zapisany jest jako składnik klasy *szachownica* i używany w funkcji wyznaczającej linie boków szachownicy. Posłużono się operacją wykrywania konturu dlatego, że dzięki opcji znajdowania tylko zewnętrznego konturu i wypełnieniu wnętrza, linie znajdujące się na szachownicy nie utrudniają dalszego przetwarzania i analizy.

### 4.2.2. Wyznaczenie boków szachownicy

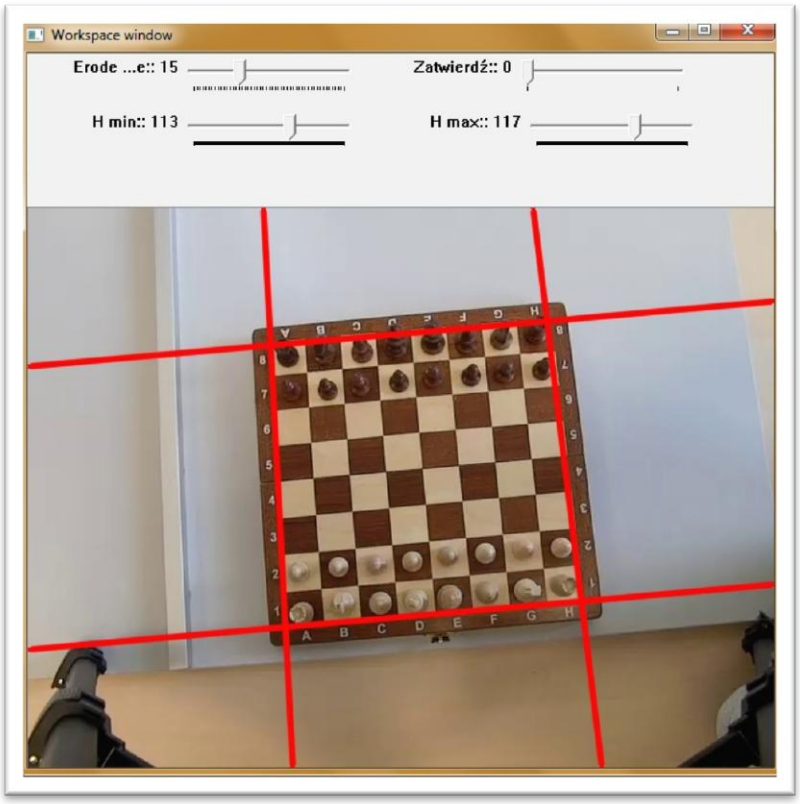
Pozycja rzeczywista położenia figur będzie podawana w notacji algebraicznej, która dla każdej planszy jest taka sama. Aby ją wyznaczyć nie jest konieczne podanie rzeczywistego wymiaru szachownicy, a jedynie uwzględnienie faktu, że plansza jest kwadratowa. Z tego powodu dla uproszczenia obliczeń przyjęto długość boku szachownicy jako 32cm. Na podstawie tej długości wyznaczane są współrzędne naroży płaszczyzny rzeczywistej przekazywane do transformaty. Uproszczenie to pozwoliło również pominąć konieczność mierzenia i podania przez użytkownika wymiaru szachownicy.

Współrzędne monitorowe naroży wyznaczane są przez funkcje klasy *szachownica* na podstawie obrazu. Ustawiona długość boku 32cm jest tylko sumą długości boków pól szachowych z pominięciem obrzeża, a do wyznaczania położenia figur uwzględniana jest tylko płaszczyzna pól szachowych. Z tego powodu istnieje konieczność obcięcia obrzeża z konturu na obrazie. Możliwe byłoby odejmowanie wymiaru obrzeża w obliczeniach, lecz wprowadzało by to pewne utrudnienia w obliczeniach, a użytkownik musiałby podawać za każdym razem jego wymiar. Innym rozwiązaniem było by napisanie funkcji wykrywającej tylko pola, lecz często obrzeże bywa tak zdobione, że utrudniałoby to właściwy proces rozpoznania. Dlatego zdecydowano się zastosować prostszy sposób lecz wymagający udziału użytkownika. Poprzez zmianę wartości parametru *erode_number* użytkownik wyznacza o ile ma być obcięta płaszczyzna szachownicy (wyznaczona przez kontur we wcześniej omawianej funkcji) przy pomocy funkcji ***erode()***. Operacja ta wykonywana jest w pętli w funkcji ***chessboard::findSideBoard****(Mat& Frame, parametry &param)*. Czynność ta zawsze musi być wykonana, gdyż każda szachownica posiada obrzeże. Usuwane są na tym etapie elementy, które wcześniej wykryto jako kontury o tym samym poziomie w hierarchii co plansza, a będące tylko niepotrzebnymi obiektami w tle.

Następnie w tej samej pętli za pomocą funkcji ***Canny()*** zostaje wykryta sama krawędź, po czym przeprowadzona jest transformata Hougha obliczająca linie będące bokami szachownicy. Linie te są rysowane na nieprzekształconym obrazie wejściowym, aby użytkownik mógł wizualnie ustalić gdzie obrzeże zostało odcięte, a linie wyznaczają brzegi powierzchni pól szachowych. Po zatwierdzeniu przez użytkownika wyniku przetwarzania obrazu następuje wyjście z pętli, a parametry linii jakie zwraca funkcja Hougha są konwertowane do postaci parametrów reprezentujących prostą kierunkową. Zapis ten znacznie ułatwi w dalszej części wyznaczenie przecięcia tych linii określając naroża na obrazie. Prosta kierunkowa reprezentowana jest jako struktura i zawiera zmienne typu *double* będące parametrami *a* i *b* oraz zmienną logiczną informującą czy prosta nie jest prostopadła do osi OX. Te wszystkie operacje mają na celu wyznaczenie linii, z których każda jest określona tylko dwoma parametrami ułatwiającymi ich matematyczny zapis.




**Rys. 4.4.** Ustawienie położenia linii wyznaczających boki obszaru uwzględnianego w późniejszych obliczeniach. Wartości erode_number równe 0 i 22.




**Rys. 4.5.** Prawidłowe zaznaczenie boków szachownicy pomimo występowania w tle innych krawędzi i obiektów.

### 4.2.3. Wyznaczenie naroży na podstawie prostych

Ostateczne współrzędne monitorowe naroży szachownicy zostały wyznaczone na podstawie równań kierunkowych w funkcji ***chessboard::designateCorner()***. Wyznaczenie przecięcia prostych przebiega następująco:

1) Ze zbioru czterech prostych wybrana zostaje para i sprawdzony zostaje warunek czy któraś z nich nie jest prostopadła do osi OX.
2) Jeśli żadna nie jest prostopadła, sprawdza się czy nie są one równoległe. Odbywa się to wykorzystując zależność, że współczynnik ***a*** linii biegnących poziomo jest bliski zera, a linie zbliżone do pionu mają duże wartości współczynnika większe od jeden.
3) W przypadku gdy jednak któraś z prostych jest prostopadła do OX, stosowany jest inny wzór obliczający współrzędną x, a przecięcie wyliczane jest tylko gdy druga prosta ma współczynnik bliski 0, ponieważ tylko wtedy prosta określa bok przyległy do pierwszej prostej.
4) Operacje są powtarzane dla pozostałych par prostych.

### 4.2.4. Ustawienie naroży

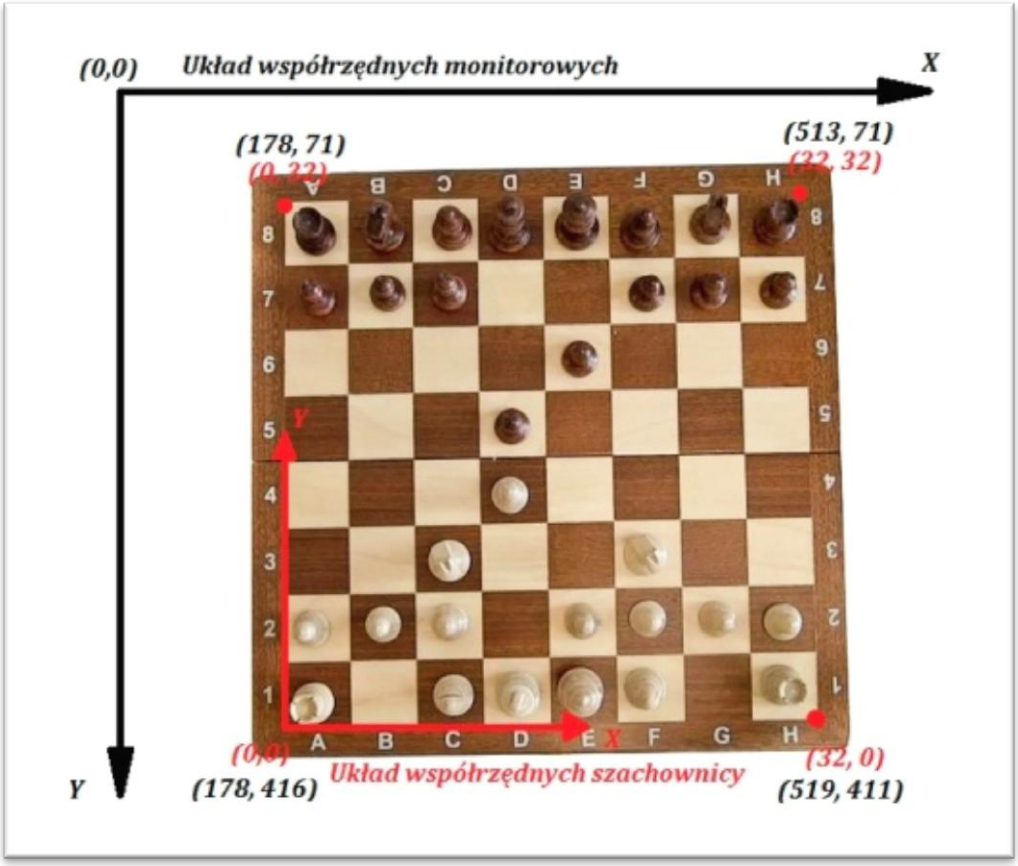
Ponieważ do wykonania identyfikacji transformaty płaskiej konieczne jest podanie czterech punktów rzeczywistych i odpowiadających im w tej samej kolejności punktów monitorowych, należało ułożyć naroża we właściwej kolejności poprzez ich posortowanie. Punkty są ustawiane w kolejności przeciwnej do ruchu wskazówek zegara, zaczynając od lewego górnego rogu, ponieważ w nim znajduje się początek układu współrzędnych płaszczyzny obrazu wyświetlanego przez OpenCV.

Ustawienie naroży monitorowych odbywa się następująco:

1) Narożniki sortowane są według średniej odległości od początku układu wyznaczonej jako średnia wartość sumy współrzędnych $xm_i$ i $ym_i$.
2) Najmniejsza z nich określa najbliższy narożnik początku układu i po sortowaniu ustawiony jest jako pierwszy.
3) Jeśli narożnik na pozycji drugiej ma mniejszą wartość współrzędnej $ym_i$ wtedy zamieniany jest kolejnością z tym na pozycji trzeciej.
4) Zamieniane są miejscami narożniki trzeci i czwarty.

Układ monitorowy jest odbiciem lustrzanym układu kartezjańskiego względem osi OX. Fakt ten wpływa na właściwe wyznaczenie pozycji figury. Przyjmując ustawienie układu rzeczywistego zgodnie z układem wyznaczonym przez

notację na szachownicy (ustawienie odpowiada układowi kartezjańskiemu) i dla takiej konfiguracji podając odpowiadające sobie współrzędne naroży, transformata płaska niewłaściwie oblicza pozycję rzeczywistą.




**Rys. 4.6.** Układy współrzędnych płaszczyzny rzeczywistej (szachownicy) i płaszczyzny monitorowej (obrazu).

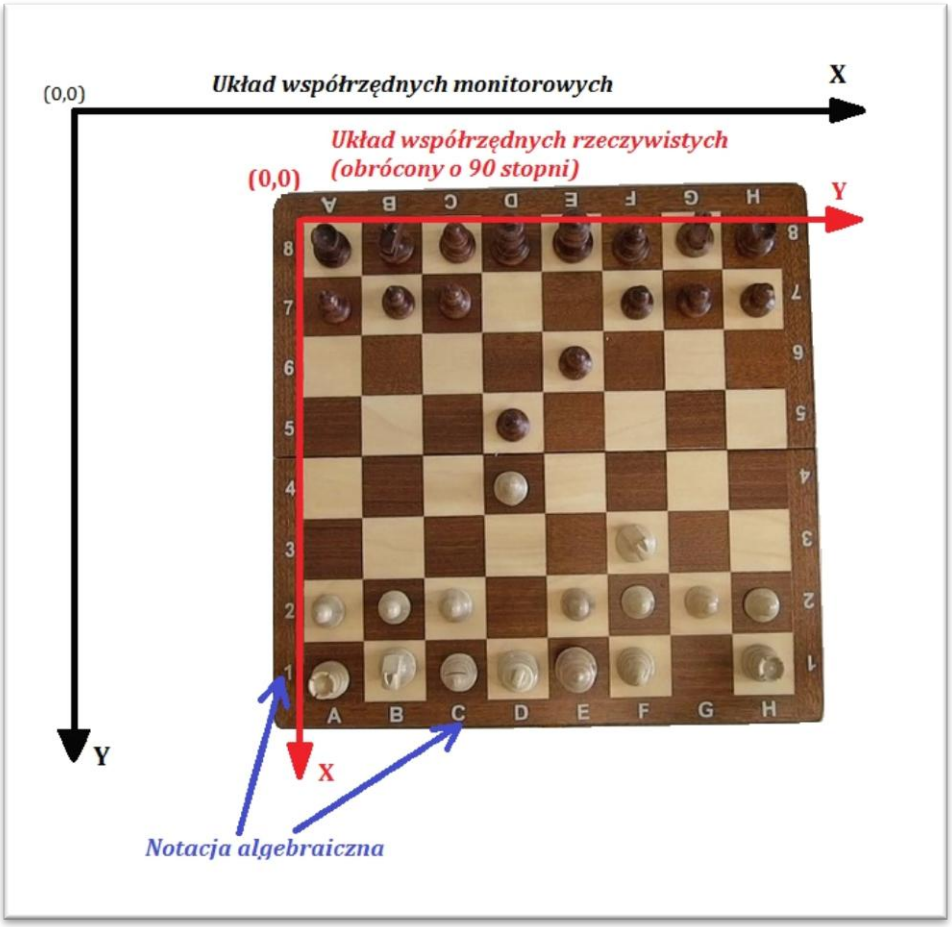
Przyjęto zatem, że punkt (0,0) układu rzeczywistego będzie znajdować się w lewym górnym narożniku tak jak początek układu monitorowego i zostanie obrócony o 90 stopni, aby osie miały ten sam zwrot. Po takiej zmianie ustawienia przeprowadzenie transformaty daje poprawne wartości położenia lecz należy wynik x traktować jako y, a wartość y jako x ponieważ układy mają w stosunku do siebie zamienione osie. Aby było możliwe przedstawienie współrzędnych rzeczywistych w notacji algebraicznej zastosowano następującą funkcję konwertującą:

```
//Wyznaczenie położenia na szachownicy miejsc w których wykryto okręgi
void chessboard::determinantPosition(transformata &tf, parametry &param)
{
        double size_field = 0;
```

```
        size_field = real_size/8; //wymiar pojedynczego pola

        for(unsigned i=0; i<param.found; i++)
        {
           tf.tp(param.circles_Im[i].x, param.circles_Im[i].y);
           param.circles_Re[i].x = (int)ceil(tf.yr/size_field);  //numer kolumny
           param.circles_Re[i].y = 9 - (int)ceil(tf.xr/size_field); //numer wiersza
           cout<<"\nWspolrzedne wg. numeru pola :"<<param.circles_Re[i].x
           <<" "<<param.circles_Re[i].y;
        }
}
```

W powyższej funkcji rozwiązany jest również problem odwrotnej kolejności wartości na osi Y, a numeracji wierszy notacji szachowej. Ponadto wartości liczbowe osi X zamieniane są na odpowiadające im symbole literowe dopiero przed ich wyświetleniem, a wszelkie obliczenia wykonywane są na liczbach określających numery kolumny.




**Rys. 4.7**. Zmiana ustawienia przyjętych układów współrzędnych.

## 4.3. Metody wykrywania figur

Jednym z najważniejszych zadań realizowanych w pracy jest wykrywanie na obrazie obiektów jakimi są figury szachowe, aby było możliwe określenie ich rodzaju i wyznaczenie położenia. Do tego konieczne jest ich wstępne rozpoznanie. Ponieważ procesy rozpoznawania obrazu przez komputer przebiegają inaczej niż procesy zachodzące w ludzkim mózgu, należy brać pod uwagę fakt, że obiekty, kształty elementów, krawędzie i faktura doskonale rozpoznawane przez człowieka nie są tak samo widziane przez systemy wizyjne. Dlatego należało wziąć pod uwagę, że figura która jest widoczna dla człowieka może nie być znaleziona przez zastosowane mechanizmy przetwarzania i rozpoznawania obrazu. Zastosowanie perspektywy kamery dającej widok z góry pozwoliło zmniejszyć złożoność procesów, przez co również stosowanych algorytmów jakie były by potrzebne do rozpoznawania figur na podstawie ich kształtu. Wykrywane figury na obrazie cechują się prostotą kształtu, przyjmując formę okręgów i znacznie upraszczając stosowane mechanizmy rozpoznawania. Jednak również w tym przypadku nie dało się uniknąć problemów z ich wykrywaniem. Aby wykryć obiekt na obrazie należy w pierwszym etapie przetworzyć obraz usuwając elementy nieistotne, wykryć fragmenty które mogą należeć do poszukiwanego obiektu, a następnie posługując się pewnymi kryteriami które spełnia ten obiekt, sprawdzić czy na obrazie są fragmenty posiadające wybrane cechy t.j. linie, teksturę, kolor.

We wstępnej fazie pracy pod uwagę było brane wykrywanie figur na podstawie koloru, jednak po analizie zagadnienia zostało ostatecznie odrzucone. Wykorzystanie barwy figur okazało się dość trudne, gdyż przy każdej rozgrywce konieczne było by definiowanie jej zakresu, ponieważ każdy zestaw posiada inne barwy. Zmienna charakterystyka oświetlenia wpływa znacznie na kolory figur wprowadzając do obrazu wyraźną dominantę barwną. Rozpoznawanie figur, nawet przy użyciu innych metod, utrudnia bardzo zbliżona barwa figur i pól szachowych. W pracy wykorzystano poza cechą związaną z wyglądem figur, ich zmianę położenia na planszy.

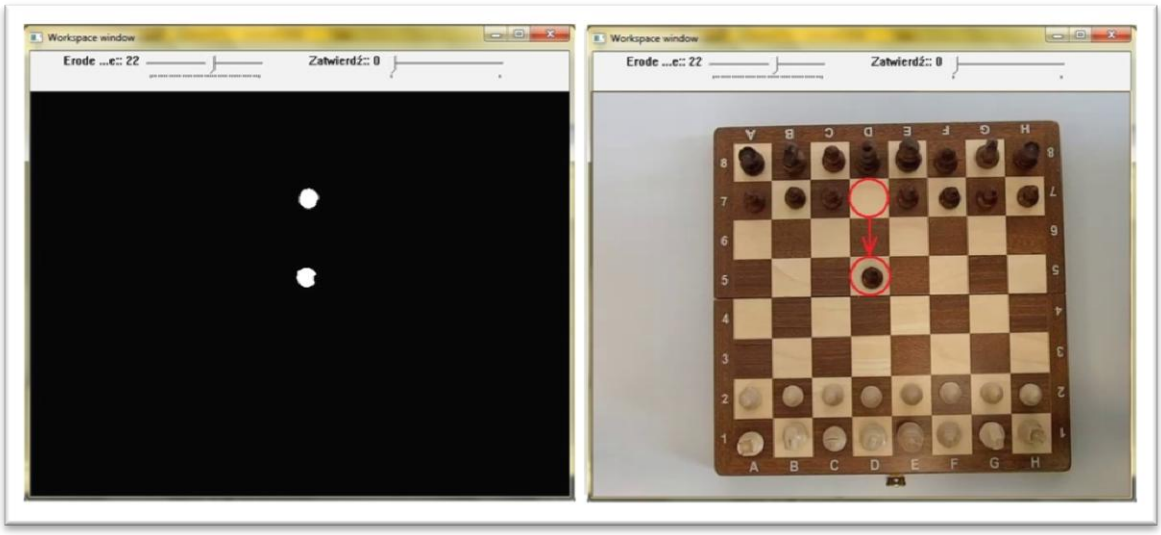
### 4.3.1. Metoda różnicowa

W przypadku obrazu pochodzącego z nieruchomej kamery, ma się do czynienia z sekwencjami obrazów dającymi możliwość wykrywania obiektów poruszających się. Jest to możliwe dzięki metodzie różnicowej, która polega na odjęciu od siebie dwóch kolejnych obrazów. Otrzymujemy w ten sposób obraz, na którym są wyszczególnione tylko te obszary, w których nastąpiła zmiana. Realizacja tej metody wymaga niewielkiej mocy obliczeniowej, ale również trudnej do zrealizowania kontroli, czy wykryte zmiany spowodowane są rzeczywistym ruchem, czy tylko lokalnymi zmianami oświetlenia.

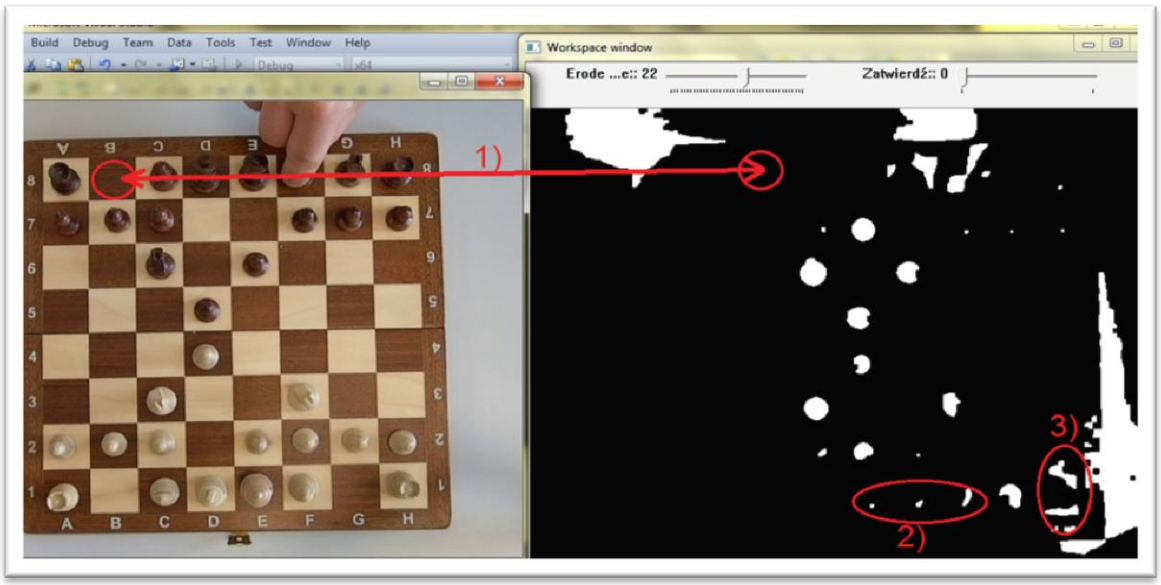
W przypadku rejestracji przebiegu gry użycie tej metody polega na odjęciu dwóch klatek z przed i po wykonaniu ruchu, a następnie interpretacji wykrytych zmian i ich położenia na obrazie. Przed odjęciem obrazy poddano konwersji do skali szarości, ponieważ jak wspomniano we wcześniejszych rozdziałach, większość operacji wymaga obrazu źródłowego w skali szarości lub binarnego. W kolejnych metodach obraz również będzie poddawany konwersji, jednak nie będzie to już zaznaczane, chyba że stosowana będzie inna konwersja np. do HSV. Następnie obrazy poddawano działaniu filtru rozmywającego usuwającego szumy ***GaussianBlur()*** lub tylko wygładzającego obraz ***medianBlur()***. Kolejno przeprowadzono odjęcie bezwzględne ***absdiff()*** i binaryzację, której dobór progu pozwala ustalić poziom istotności zmian, kwalifikowanych jako pozycje przesuniętych figur. Przeprowadza się również operację erozji usuwając nieistotne elementy, po czym operacją dylatacji powiększa się obszar figur pomniejszony przez erozję. W metodzie różnicowej wyznaczane są tylko miejsca, w których znajdowała się figura oraz gdzie znajduje się obecnie. Wyznaczenie pozycji można przeprowadzać poprzez etykietowanie obszarów i wyznaczenie ich położenia lub stosując szkieletyzację, która określa punkt będący w przybliżeniu środkiem figury wyznaczający jej położenie na obrazie. Następnie korzystając z funkcji analizującej współrzędne w których wykryto zmiany i znajomości wcześniejszego położenia figur, wykrycie jaki ruch został wykonany. Metoda ta dostarcza informacji jedynie czy i jaki ruch nastąpił. Wystarcza to jednak do rejestracji przebiegu gry w szachy, ponieważ reszta figur nie zmieniła swojej pozycji, więc nie jest konieczne wykrywanie położenia wszystkich figur,.

Niestety, przy użyciu tej metody występują błędy w sytuacjach, gdy pomiędzy odejmowanymi klatkami nastąpi znaczna zmiana oświetlenia spowodowana

zacienieniem fragmentu obrazu. Również wtedy, gdy kolor figury jest zbliżony do koloru pola, a zastosowanie mniejszego progu binaryzacji spowodowało by wykrycie innych obszarów na których nie przesuwano figur np.: możliwe było by wtedy wykrycie figury w miejscu, gdzie przed ruchem znajdował się na polu jej cień i zniknął gdy figurę przestawiono.




**Rys. 4.8.** Obraz różnicowy po binaryzacji (po lewej).
Wykonany ruch pionkiem(po prawej).




**Rys. 4.9.** Błędy pojawiające się w metodzie różnicowej. 1) Nierozróżnienie figury od tła i nie wykrycie jej przesunięcia. 2) - 3) wykrywanie elementów na szachownicy w miejscach gdzie nie nastąpiło przesunięcie. Elementy wykryte poza obrębem szachownicy można nie uwzględniać w obliczeniach.

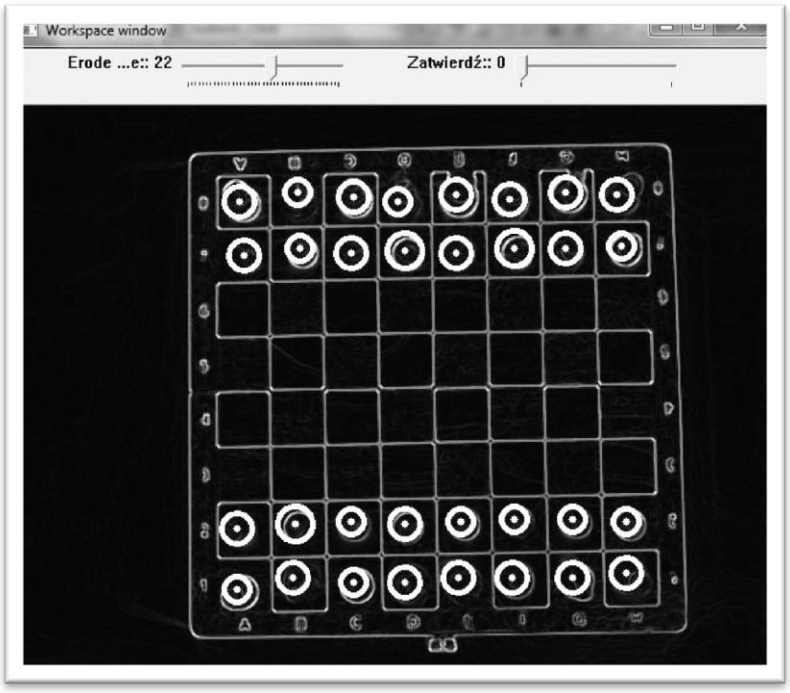
### 4.3.2. Metoda krawędziowa

Celem metody jest wykrywanie pikseli należących do krzywej będącej krawędzią. Dostarcza ona nadmiarowej informacji o obiektach znajdujących się na obrazie dlatego wymaga użycia funkcji interpretujących otrzymane krawędzie. Niestety poza krawędziami należącymi do figur wykryte zostają linie pól szachownicy, indeksy wierszy i kolumn, wyróżniające się słoje drewna lub kamienia oraz zdobienia na polach (jeśli takie występują), szczególnie utrudniające proces rozpoznawania. Metoda ta jest znacznie bardziej złożona obliczeniowo i wymaga bardziej złożonych operacji przetwarzania. Można w niej wyróżnić następujące etapy:

- Redukcja zakłóceń i wyostrzenie obrazu
- Użycie filtrów do detekcji krawędzi
- Poszukiwanie kształtów opisanych równaniami okręgów

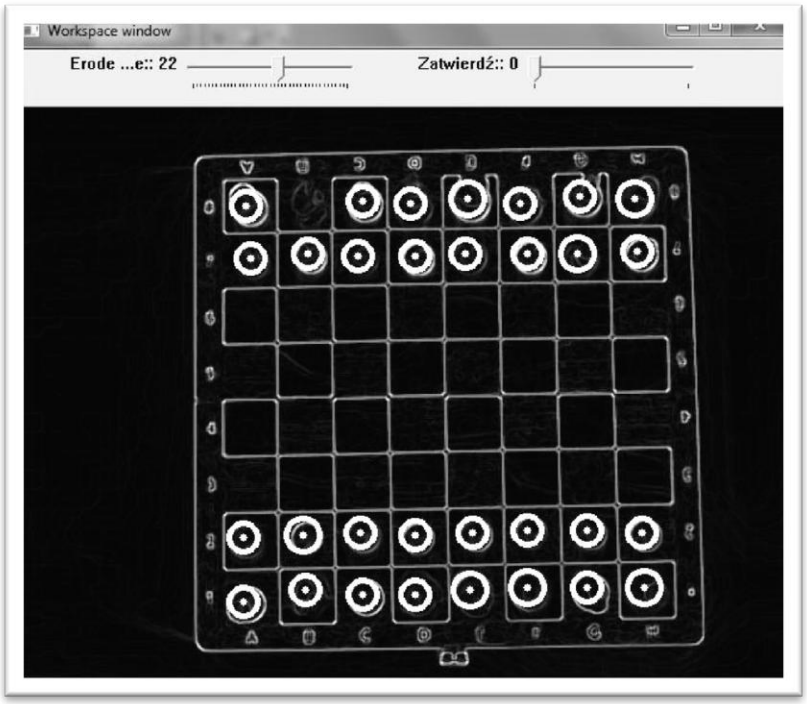
Do redukcji zakłóceń użyto filtra medianowego ***medianBlur()*** wygładzającego obraz w niewielkim stopniu go rozmywając. Wykrycie krawędzi następuje dwoma alternatywnymi filtrami. Jednym jest filtr Canny'ego (funkcja: ***Canny()***) odpowiednio z progami 0 i 30 oraz wielkością 3x3. Jako drugi filtr używano operatora Sobela (funkcja: ***Sobel()***), który wyznacza gradient w kierunku x, a następnie y, po czym ***convertScaleAbs()*** wylicza wartości bezwzględne i konwertuje je na wynik 8 bitowy. Całkowity gradient w dwóch kierunkach obliczany jest jako suma ważona dwóch pochodnych x i y przy użyciu ***addWeighted()*** z takimi samymi wagami 0.5. Filtr krawędziowy powinien wykrywać krawędzie możliwie we wszystkich kierunkach aby wyznaczone zostały krawędzie tworzące okręgi. Do wykrycia krawędzi figur będących okręgami posłużono się transformatą Hougha ***HoughCircles(),*** która zwraca współrzędne środka znalezionego okręgu oraz jego promień *r.* Funkcja ta zawiera już wbudowaną metodę wykrywania krawędzi, jednak zastosowanie jednego z wyżej wymienionych filtrów krawędziowych przed transformatą, poprawiało wyniki wykrywania. Należało ustawić kilka istotnych parametrów: wielkość akumulatora do obliczeń jako połowa wielkości obrazu wejściowego, parametr będący jego progiem dla wykrywania środków okręgów na 60 ( im mniejsza jego wartość tym więcej fałszywych okręgów może zostać wykrytych). Minimalna odległość pomiędzy wykrywanymi okręgami to 20 pikseli, jako próg binaryzacji wewnętrznego detektora krawędzi ustawiono wartość 30, a minimalny i maksymalny poszukiwany promień

odpowiednio 10 i 18, ponieważ w tym przedziale zawierają się okręgi tworzone przez wykryte krawędzie.

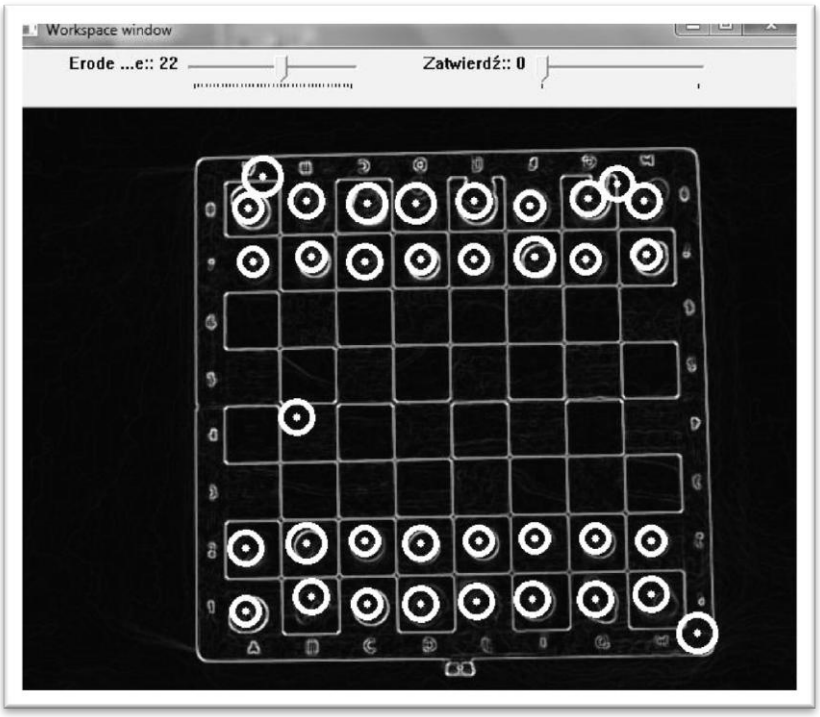



**Rys. 4.10**. Przykład działania metody dla pojedynczej klatki.
Wyrysowane okręgi w miejscach gdzie zostały wykryte.

Jednakże w przypadku obrazu rzeczywistego, gdzie na obrazie występują cienie, zmiana oświetlenia, niepełne i zniekształcone krawędzie okręgów, metoda wykrywa również okręgi w miejscach w których nie występują lub nie rozpoznaje figur. W jednym z użytych zestawów szachów najczęściej niewykrywaną figurą był czarny skoczek na polu b8 gdyż warunki oświetlenia powodujące zlewanie się jego części z ciemnym polem oraz nieregularny kształt, spowodował, że w początkowych etapach przetwarzania został wykryty jako zniekształcona krawędź. Ustawienie parametrów w taki sposób, aby było możliwe wykrycie okręgów po części zniekształconych lub niekompletnych, spowodowało wykrywanie okręgów w miejscach, w których nie występowały np. w zaokrąglonych narożach szachownicy.




**Rys. 4.11.** Problemy z wykryciem skoczka na polu b8.



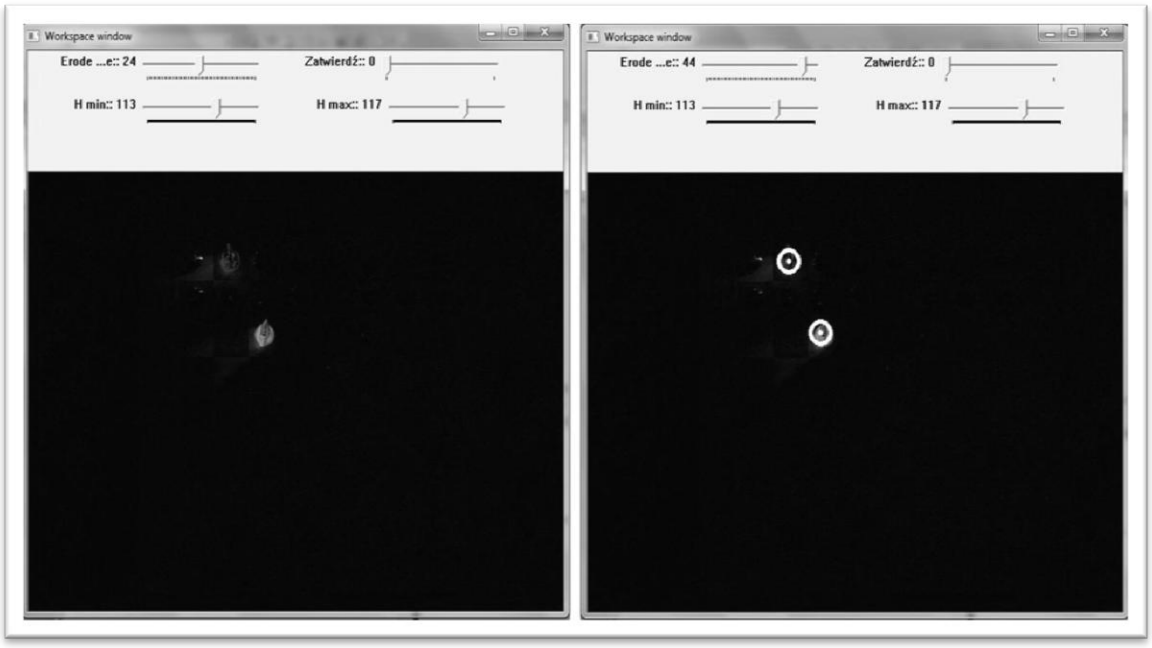

**Rys. 4.12.** Wykrycie okręgów w miejscach gdzie nie znajdują się figury.

### 4.3.3. Metoda różnicowo krawędziowa

Ponieważ żadna z powyższych metod sama w sobie nie umożliwiła prawidłowego wyznaczenie położenia figur oraz ich ruchu, postanowiono połączyć obydwie metody (wykorzystując ich zalety) w jedną metodę. W metodzie różnicowo krawędziowej, dzięki wykorzystaniu odejmowania obrazu, ogranicza się informację na obrazie jedynie do miejsc gdzie następuje ruch, a detekcja i rozpoznanie krawędzi przeprowadzane jest tylko dla tych obszarów, zmniejszając możliwość wykrycia obiektów nie będących pozycjami figur.

Po konwersji do skali szarości obrazy są od razu od siebie odejmowane przez ***absdiff()***. O ile w poprzednich metodach redukcja szumu i wygładzenie obrazu poprawiało efekty wykrywania, to w tym przypadku powodowało pogorszenie wykrywalności. Istotne było, aby wykryć jak największe różnice w miejscach, gdzie występowało przesunięcie figur. Również nie wykonano binaryzacji, która spowodowała by odcięcie obszarów na których następował ruch, a różnice były by zbyt małe. Obraz w skali szarości powstały po odjęciu, bezpośrednio przekazywany jest do funkcji ***HoughCircles****(img_diff, circles, CV_HOUGH_GRADIENT, 2, 20, 30, 60, 9, 18).* Zastosowanie tylko odjęcia i transformaty Hougha dla okręgów dało znacznie lepsze efekty wykrywania miejsc w których następował ruch. Otrzymano w ten sposób od razu informację o współrzędnych wykonanych ruchów, bez konieczności rozpoznawania położeń okręgów wszystkich figur, które nie zmieniły położenia, co miało miejsce w metodzie krawędziowej. Współrzędne środków okręgów zapisane są w tablicy struktur *circles_Im[]* znajdującej się w obiekcie klasy *parametry* i w nim przekazywane do dalszych obliczeń.

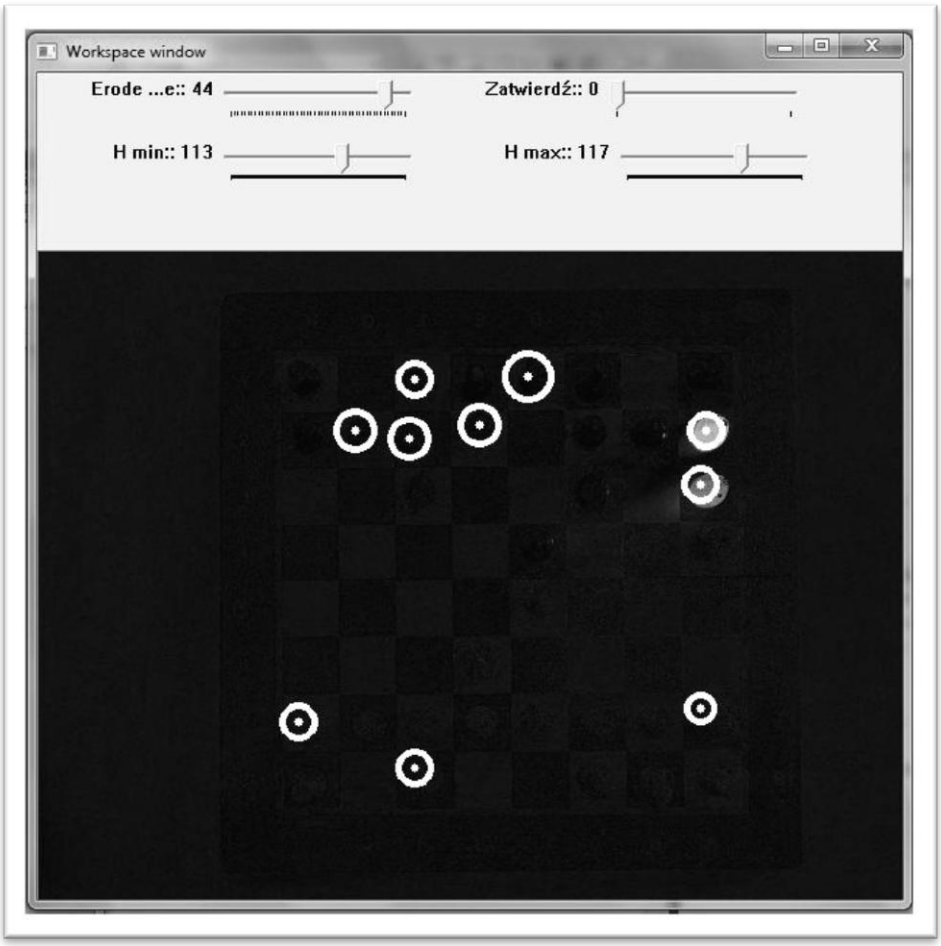
Metoda zrealizowana jest wewnątrz funkcji *Mat* ***fcnSubCircles****(Mat Frame1, Mat Frame2, parametry &param)*, gdzie Frame1 i Frame2 to odejmowane klatki, obiekt *param* zawiera zmienne do których zapisane zostaną współrzędne wykrytych środków okręgów oraz stan flagi informujący, czy wykryto okręgi i ile ich znaleziono, w celu kontroli poprawności wykonania programu. Współrzędne monitorowe wykrytych okręgów jako tablica struktur zawierająca dane x, y, przekazywane są do prezentowanej wcześniej funkcji ***chessboard****::****determinantPosition****(transformata &tf, parametry &param)* gdzie są wyznaczane ich pozycje na szachownicy.




**Rys. 4.13**. Etapy przetwarzania w metodzie różnicowo – krawędziowej. Wynik odjęcia bezwzględnego (po lewej). Wykryte okręgi funkcją ***HoughCircles()*** (po prawej).

Pomimo znacznej poprawy, nie udało się uzyskać stuprocentowo pewnej metody zapewniającej ciągłe i nieprzerwane działania podczas rozgrywki partii szachowej. Głównymi powodami, które powodowały błędne rozpoznawanie, były zmiany warunków oświetleniowych pomiędzy odejmowanymi klatkami oraz zlewanie się figur z polami o tych samych kolorach. Zmiany oświetlenia powodują pojawienie się w obrazie różnicowym zarysów szachownicy i wszystkich figur mniej widocznych niż miejsca, gdzie następował ruch, jednak wykrywanych przez detektor krawędzi. Znaczne problemy występowały przy wykrywaniu figury skoczka, dlatego zdecydowano się użyć innego zestawu szachów, gdzie różnica pomiędzy ciemnymi figurami, a ciemnymi polami jest większa. Pozwoliło to na wykrywanie problematycznej figury w większej liczbie przypadków, a efekt można zaobserwować na rysunku 4.13, gdzie nawet pomimo słabszych zarysów algorytm wykrywania był w stanie sobie z nim poradzić. Na obrazie po odjęciu można zaobserwować niekorzystne zjawisko wykrywania również różnic spowodowanych przez cień figury rzucany na sąsiednie pola. Podczas kręcenia filmików testowych do pracy oświetlenie było dobrane tak, aby światło padało w miarę równomiernie z każdej strony nie tworząc głębokich cieni. Dzięki temu nie zaobserwowano problemów z wykrywaniem ruchów na polach, gdzie znajdował się cień figury.

Jednak należy założyć, że podczas partii taki cień może spowodować błędne wykrycie okręgów, jeśli zostanie wyróżniona znaczna część figury stojącej obok na polu.




**Rys. 4.14.** Zmiana warunków oświetlenia spowodowała wykrycie figur, które nie zmieniły pozycji.

Ważną kwestią jest dobór parametrów transformaty, aby możliwe było prawidłowe działanie programu w różnych warunkach i ustawieniach. Istotne jest dobranie minimalnej odległości pomiędzy poszukiwanymi okręgami oraz ich promienia. Jeśli szachownica na obrazie będzie zbyt mała lub zbyt duża, ich promienie mogą znajdować się poza ustawionym przedziałem. Ustawienie szerszego zakresu rozwiązało by problem, jednak wymaga to większego nakładu obliczeniowego, zwiększając zarazem ryzyko wykrywania fałszywych okręgów.

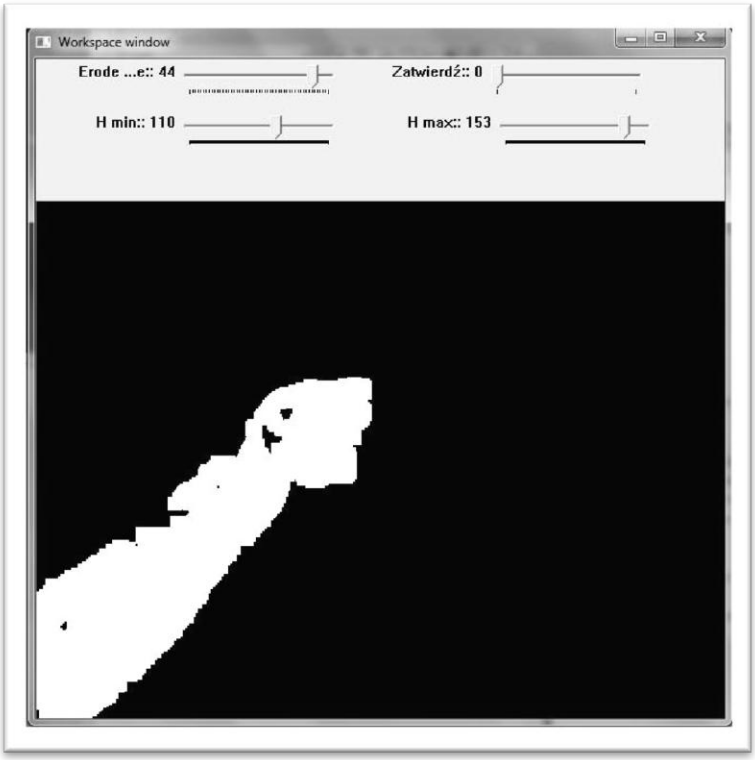
## 4.4. Wykrywanie ruchu ręki

Zastosowana w pracy metoda różnicowo-krawędziowa wymaga podania obrazu z przed ruchu i po jego wykonaniu. Aby było to możliwe, konieczne jest wykrycie, kiedy ten ruch nastąpił. Celem tego etapu pracy było stworzenie mechanizmu rozpoznającego, kiedy na obrazie znajduje się ręka, a kiedy jej nie ma. Pozwoliło to na wyznaczenie klatki od której będzie odejmowany obraz, oraz klatki która będzie odejmowana. W trakcie pracy okazało się, że zadanie to nie jest proste, pomimo faktu, że zmiany na obrazie teoretycznie występują tylko wtedy, gdy jest na nim widoczna ręka.

Z pośród kilku testowanych sposobów do realizacji pracy posłużono się wykrywaniem koloru skóry. Wyszukiwanie dłoni odbywało się na obrazie w przestrzeni HSV. Obraz pochodzący z kamery lub pliku w formacie BGR konwertowany jest przez funkcję ***cvtColor()*** używając zdefiniowanego kodu konwersji CV_BGR2HSV. Detekcja pikseli należących do ręki odbywa się poprzez wybranie zakresu wartości składowej H, który odpowiada barwie ludzkiej skóry, natomiast składowe S i V mogą być ustawione w dużym zakresie. Wartość składowej H dla skóry zawiera się w przedziale 0 – 30, jednak barwa jest w znacznym stopniu zależna od źródła światła dlatego dla różnych źródeł oświetlenia ten zakres może ulec zmianie. W jednym z testowanych plików wideo wartość H dla ręki wynosiła 110 – 120. Z tego powodu użytkownik ma wpływ na ustawienie wartości minimalnej i maksymalnej składowej H. Dokonuje tego na początku programu suwakami, po wybraniu źródła obrazu. Jeśli obraz pochodzi z kamery, użytkownik powinien umieścić rękę w polu obserwacji kamery i przesunąć suwanki w taką pozycję, aż będzie na obrazie zaznaczony tylko obszar należący do ręki. Gdy zostanie jako źródło wybrany plik wideo, ustawiana jest przez użytkownika wartość, aby była widoczna tylko powierzchnia ręki znajdującej się na filmie. Po zatwierdzeniu ustawień, plik wideo jest odtwarzany od nowa.

Zakres wykrywania dla składowej S ustawiony jest na 10 – 255. Ponieważ ciemne punkty są niedokładnie odwzorowywane w składowych H i S, za pomocą dolnej wartości V można je wykluczyć, przez co nie zostaną zakwalifikowane jako punkty skóry. Wartości progów dla kanału V zostały dobrane eksperymentalnie i wynoszą 40 – 255. Wyznaczenie obszaru, którego piksele mają barwę z podanego zakresu, wykonywane jest w funkcji ***inRange()*** . Piksele spełniające warunek przynależności oznaczane są jako białe (255), nie spełniające tego warunku, jako czarne (0). Na otrzymanym obrazie binarnym przeprowadzono operację erozji, a następnie dylatacji, ponieważ szumy

na obrazie mogą przyjmować barwę taką jak skóra i należy je usunąć, otrzymując tylko maskę ręki. Kolejnym krokiem było przeprowadzenie wykrywania konturu ***findContours***() oraz sprawdzenie czy został on wykryty na obrazie. Odbywa się to przez sprawdzenie wektora zawierającego wykryte kontury, czy nie jest on pusty.

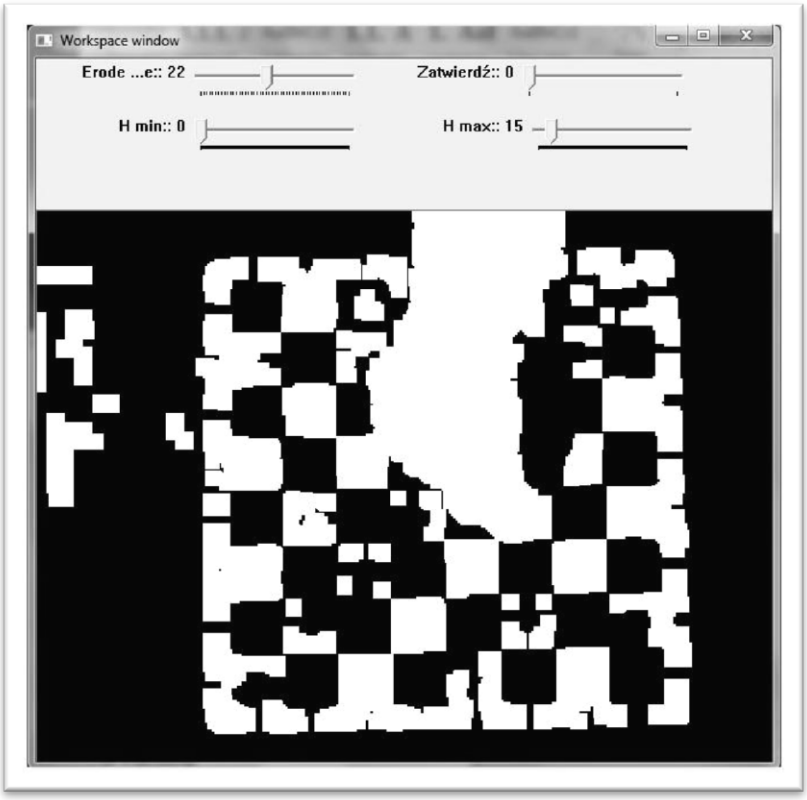



**Rys. 4. 15.** Obraz binarny przedstawiający działanie wykrywania ręki na podstawie składowej H.

Operacje te zostały zawarte w funkcji ***fcnDetectHand****(Mat Frame, parametry &param),* do której przekazany zostaje obraz do detekcji oraz obiekt klasy *parametry,* zawierający zmienne określające zakres H. Obecności ręki na obrazie komunikowana jest przez ustawienie flagi *param.check_hand* . Algorytm stworzony w pracy zapamiętuje trzecią klatkę przed wykryciem ręki i przechowuje ją aż do momentu, gdy ruch się nie zakończy i ręka nie zniknie z obrazu. Wtedy zapisywana jest kolejna klatka po wykonaniu ruchu i obydwie przekazywane są do funkcji ***fcnSubCircles****()*, wykrywającej zmianę położenia figur. Ponieważ z powodów technicznych nie jest wykrywana cała powierzchnia ręki, a jedynie jej fragmenty, może się zdarzyć, że na odejmowanych obrazach będzie znajdować się jej fragment. Dlatego po wykryciu końca ruchu, następuje opóźnienie zapisu obrazu do porównania o 4 klatki. Obraz przed ruchem zapisywany jest co trzecią klatkę, przez co może być on oddalony

od 1 do 3 klatek. Opóźnienie pomiędzy wykryciem ruchu, a zapisami klatek pozwoliło na wyeliminowanie problemów, jakie związane były z pojawieniem się fragmentów ręki lub jego cienia na odejmowanych obrazach, powodujących wykrycie fałszywych przesunięć. Cień ręki może powodować efekt widoczny na rysunku 4.14 w poprzednim podpunkcie.

Innym problemem związanym z metodą wykrywania na podstawie koloru jest występowanie przypadków, gdy plansza szachowa ma podobny zakres barw, co skóra dłoni. Jeden z zestawów szachowych używanych w projekcie posiadał kremowe i brązowe pola zbliżone do koloru skóry, przez co ustawienie wartości H jako 0 – 30 powodowało wykrycie obszarów szachownicy. Dopiero obniżenie zakresu do 4 pozwoliło na oznaczenie tylko obszaru skóry.




**Rys. 4.16**. Pokrywanie się zakresu barwy skóry ręki z barwą szachownicy.

Konieczność zastosowania tak małego zakresu może uniemożliwić działanie programu, ponieważ występują różnice w barwie skóry dwóch graczy. Rozwiązaniem tego problemu mogłoby być zastosowanie algorytmów generacji tła lub metod śledzenia ruchu typu Optical Flow.

## 4.5. Reprezentacja figur i interpretacja wykonanych ruchów

Metody wykrywania figur dostarczają jedynie informacji, gdzie na obrazie nastąpił ruch, po czym wyznaczane są pozycje tych zmian na szachownicy. Jednak konieczne było określenie, jaki rodzaj ruchu został wykonany i przez jaką figurę lub figury. Na początek określono sposób reprezentacji figur i ich położenia na szachownicy. Została w tym celu utworzona klasa *class figures* reprezentująca pojedynczą figurę oraz jej atrybuty t.j.: położenie, nazwę, skrót mnemoniczny używany w zapisie, flagę zbicia i flagę pierwszego ruchu. Klasa posiada dwa konstruktory. Jeden domyślny, inicjalizujący wartości difoltowe dla obiektu. Drugi konstruktor posiada listę argumentów, przez którą inicjalizowane są własności figury. W szachach występuję jednak po szesnaście figur dla każdego koloru, zatem utworzone zostały dwie tablice obiektów:

*figures biale[NUMBERFIGURE] = {…}*
*figures czarne[NUMBERFIGURE] = {…}*

Stała NUMBERFIGURE została zdefiniowana dyrektywą #define i podmieniana jest w miejscu jej wystąpienia wartością 16. Ustawienie początkowe figur, takie jak na początku partii, następuje na liście inicjalizacyjnej. Klasa zawiera również funkcje składowe: *void* ***changePosition****(int, int),* którą wykonywana będzie zmiana pozycji figury, *void* ***insertFigure****(int, int),* wstawiająca figurę na pole po awansie pionka oraz *void* ***bitUp****(),* ustawiająca status figury jako zbita. Ponieważ łatwiej operować, gdy kolumny i wiersze określane są tymi samymi typami wartości, współrzędne x i y są określane liczbami całkowitymi, a nie tak jak w notacji algebraicznej wiersze liczbami, a kolumny literami. Jak już wcześniej zostało wspomniane, poprawny zapis szachowy potrzebny jest tylko do wyświetlenia informacji o ruchu i dopiero przed wyświetleniem, numery kolumn są konwertowane funkcją *string* ***checkLetter****(int nr)* na odpowiadające im litery.

Mając wyznaczone wcześniej przypuszczalne pozycje gdzie nastąpił ruch, następuje wykrycie, jaki ruch został wykonany. W sytuacji gdyby nastąpił błąd wykrycia ruchu i na odjętym obrazie nie wykryto okręgów, została by ustawiona flaga *bool ferr* i etap rozpoznawania jaki ruch nastąpił został by pominięty. Ponieważ zadaniem programu jest tylko rejestracja przebiegu gry w szachy nie ma konieczności

rozpoznawania poprawności ruchu zależnego od rodzaju figury. Rozpoznawane jest natomiast jaka figura wykonała ruch i na jakich polach, co nie stanowi większego problemu. Występuje natomiast konieczność rozpoznawania ruchów takich jak: bicie figury, bicie w przelocie, awans pionka oraz krótka i długa roszada. Do wykrycia sytuacji, kiedy dany ruch nastąpił, posłużono się sprawdzaniem ilości wykrytych okręgów. Dla przesunięcia figury, bicia i awansu, wykrywane są dwa okręgi (w przypadku awansu następuje również wykrycie jednej pozycji). W przypadku bicia w przelocie, ruch następuje na trzech polach, a dla roszady na czterech polach. Wykrycie danej ilości figur nie gwarantuje, że występuje na pewno ten rodzaj ruchu, dlatego sprawdzane są dodatkowe warunki związane z danym ruchem. Wybór rodzaju ruchu następuje w instrukcji *switch*, a ponieważ najczęściej wykonywanymi ruchami są przemieszczenia figur i bicia, dlatego ich sprawdzenie jest umieszczone w pierwszym warunku *case*.

**STANDARDOWA ZMIANA POZYCJI, BICIE I AWANS**

Standardowa zmiana pozycji figury polega na przemieszczeniu jej z jednego pola na kolejne. Bicie i awans pionka jest realizowany w ramach takiego ruchu, ponieważ tylko podczas takiego przesunięcia mogą zaistnieć. Wykonywane jest to przez funkcję ***fcnFindFigure****(parametry &param, figures curr_player[], figures sec_player[]).* Mając wyznaczone dwa pola na których zaszedł ruch, sprawdzone zostaje na którym polu znajdowała się wcześniej figura i jaka, należąca do gracza który ten ruch wykonuje. Wyznaczony zostaje w ten sposób symbol figury oraz współrzędne, z których nastąpiło przesunięcie, i dodawane są ich wartości do zmiennej typu string *seq_move* w której zapisywana jest i składana notacja ruchu. Drugie pole natomiast automatycznie zostaje uznane za pole, na które nastąpił ruch i dodany do zapisu. Jednak wcześniej sprawdzane jest, czy na tym polu nie znajduje się figura gracza przeciwnego oraz czy nie nastąpił awans figury gracza wykonującego ruch. Jeśli ma miejsce sytuacja bicia, zostaje dodany do opisu symbol ‘:’ oznaczający bicie, status figury zostaje ustawiony jako zbity, a pozycja wyzerowana przez ***bitUp****()*.

Awans dotyczy tylko pionów i oznaczany jest przez dodanie na końcu zapisu wykonanego ruchu symbolu figury na którą jest promowany pionek. Jego wykrycie polega na sprawdzeniu, czy pion wykonujący ruch przesunął się na pole znajdujące

się w linii 8 (numer wiersza odpowiednio dla gracza białego) lub 1 (dla gracza czarnego). Gdy taka sytuacja nastąpi, uruchamiana jest ***fcnPromotionPawn****(parametry &param, figures curr_player[], int i, int j)* gdzie *i* określa indeks wektora współrzędnych pola, a *j* indeks awansowanej figury. Wyświetlona zostaje lista figur (poza Królem) z pośród których, poprzez podanie numeru, gracz ma możliwość wybrania figury, na którą zostanie zamieniony pionek. Po wybraniu figury, w tablicy obiektów reprezentującej figury zostają w miejsce pionka ustawiane są wartości nowej figury. Występować wtedy mogą na planszy np. trzy wieże. Gracz podczas rozgrywki może wymienić figurę od razu, gdy wykonuje ruch pionka na pole awansu lub dopiero po jego ustawieniu na polu i zabraniu ręki z kadru. Gdyby gracz podczas wykonania ruchu zabrał pionka bez postawienia jego lub figury na którą zamienia, na polu awansu, nie rozpoznany zostałby ruch. Wykryta byłaby w takiej sytuacji jedna figura bez ustawienia flagi informującej o awansie, co doprowadziłoby do niepoprawnego rozpoznawania w dalszej części partii. Prawidłowe działanie algorytmu przy wykryciu jednej figury następuje tylko wtedy, gdy wcześniej został wykryty awans, a pojawienie się zmiany na jednej pozycji związane jest z ustawieniem nowej figury na planszy. Każdy inny przypadek komunikowany jest jako błąd, spowodowany nie wykryciem jednej z pozycji przemieszczającej się figury.

W przypadku bicia figury poprawność wykonania tego ruchu polega na sprawdzeniu drugiego pola pod kątem obecności figury przeciwnika, dopiero po wykryciu pola, na którym znajdowała się wcześniej figura gracza aktualnie wykonującego ruch. Dla sytuacji awansu sprawdzane są warunki jego zaistnienia, czyli przemieszczenie na właściwą linię oraz czy ruch został wykonany przez pionka. Istnieje jednak problem ze sprawdzaniem standardowego przesunięcia figury. Wynika to z tego, że ruch wykonywać może każda figura i na dowolne pole. Jedynie sprawdzeniu może podlegać, czy znaleziona pozycja nie znajduje się poza obszarem gry i czy nie wykryto dwóch okręgów na jednym polu.

**BICIE W PRZELOCIE**

Bicie w przelocie, tak jak awans, związane jest tylko z figurami pionów i występuje wtedy, gdy pionek gracza wykonując pierwszy swój ruch o dwa pola, przechodzi przez pole zagrożone biciem przez pionka przeciwnika. Podczas takiego bicia wykrywane są trzy okręgi, zatem gdy spełniony jest warunek dla wystąpienia

takiego rodzaju ruchu, wywoływana jest w nim funkcja ***fcnBeatInPassing****(parametry &param, figures curr_player[], figures sec_player[])*. Sprawdza ona jednocześnie, na którym polu znajduje się figura aktualnego gracza oraz gracza przeciwnego. Jest to jedyny przypadek, w którym figura bijąca nie staje na polu figury zbijanej. Pole, na które przemieszcza się figura bijąca, wyznaczane jest poprzez sprawdzenie, na której z trzech wykrytych pozycji nie znajduje się żadna figura. Wykrycie figury bitej służy jedynie ustawieniu jej statusu przez ***bitUp()*** oraz kontroli poprawności wykonania funkcji. Zapis bicia w przelocie wygląda tak samo jak w przypadku standardowym, zatem w celu rozróżnienia, do zapisu dodawane jest *„(e.p.)”* (od francuskiego: en passant).

Sprawdzenie czy nie nastąpiło przypadkowe wykrycie trzech okręgów, jest kontrolowane przez sprawdzenie, czy obydwie figury są pionkami i znajdują się w tej samej linii numer 4 lub 5. Sprawdzeniu podlegać może to, czy wszystkie pola są oddalone od siebie o jedną pozycję oraz czy pozycja figury bitej jest ostatnią pozycją poprzedniego ruchu.

**ROSZADA**

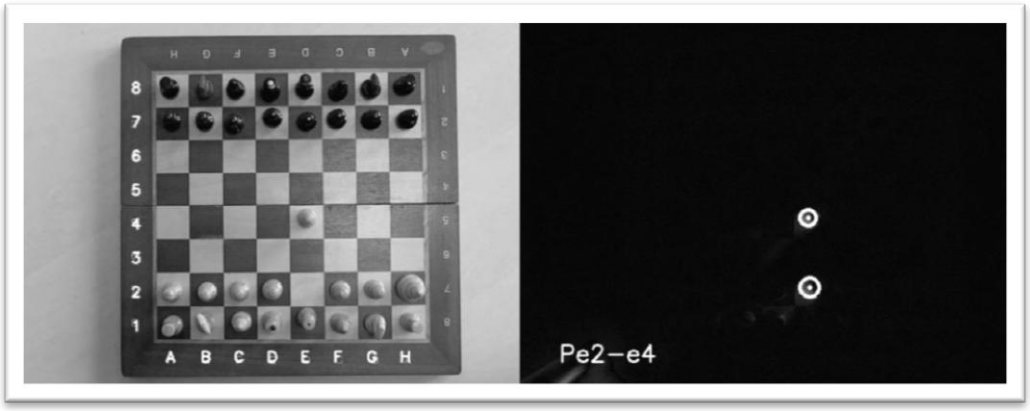
„Roszada polega na przesunięciu Króla o dwa pola w stronę Wieży i jednocześnie, w tym samym ruchu, przesunięciu Wieży w taki sposób, aby stanęła za Królem, obok niego.”[I9] Roszada jest jedynym przypadkiem, gdy podczas ruchu przemieszczają się dwie figury. Jeśli roszadę wykonuje Wieża ze skrzydła Hetmańskiego, mówimy o roszadzie długiej, natomiast jeśli Wieża z kolumny *h,* jest to roszada krótka. Każdy gracz podczas partii może wykonać ją tylko jeden raz. Wynika to z faktu że w roszadzie mogą brać tylko udział figury które wykonują ruch po raz pierwszy. Pomiędzy Królem, a Wieżą wykonującymi roszadę nie mogą stać żadne figury. Ten warunek pomimo możliwości sprawdzenia, nie jest kontrolowany, ponieważ sprawdzanie poprawności wykonania ruchów jest opcjonalne. Innym warunkiem, lecz niemożliwym do kontroli bez sprawdzania poprawności ruchów, jest to, aby Król wykonujący roszadę nie był szachowany.

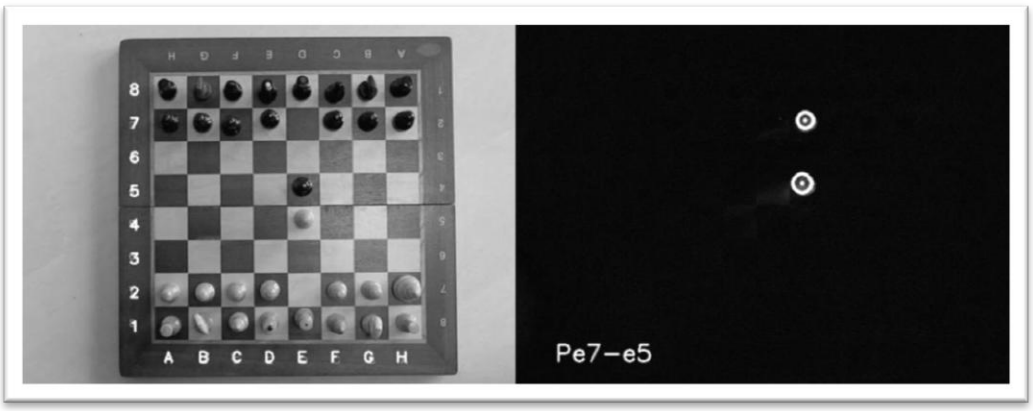
Po spełnieniu warunku *case* dla roszady, czyli wykryciu czterech pozycji wystąpienia zmian, następuje sprawdzenie, czy wszystkie pozycje leżą w jednej linii odpowiedniej dla koloru gracza mającego aktualnie ruch. Jeśli i ten warunek jest spełniony, wywoływana jest funkcja ***fcnFindCastling****(parametry &param,*

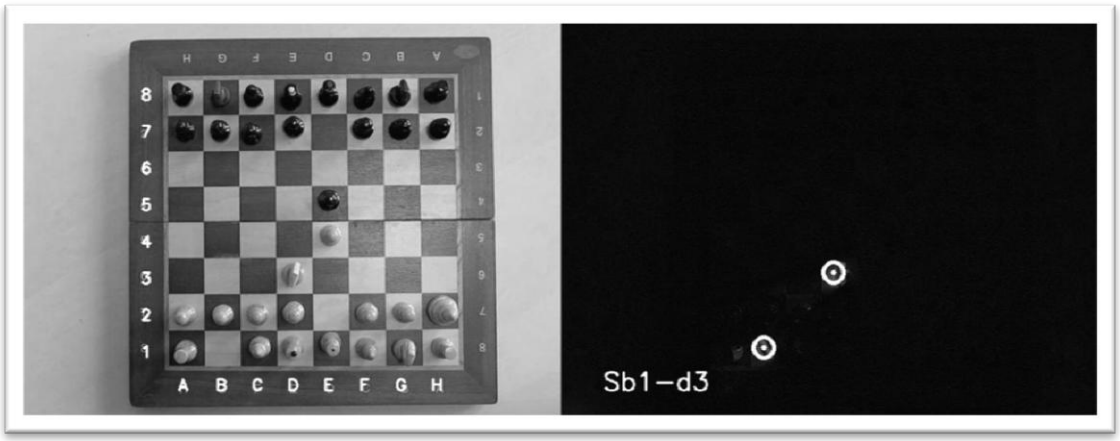
*figures curr_player[]).* Do zapisu roszady nie używane są współrzędne pól, a jedynie zapis określający roszadę krótką czy długą. Funkcja kontroluje tylko pewne warunki konieczne do zaistnienia takiego rodzaju ruchu. W niej określany jest rodzaj wykonywanej roszady, sprawdzając czy wykonuje ją Wieża z kolumny *a* czy *h*. Następnie Wieża ustawiana jest automatycznie na nowe pole znane z góry. Sprawdzane jest czy ruch został wykonany również przez Króla z kolumny *e*, po czym jego pozycja zostaje zmieniona również automatycznie na określone pole. Współrzędne pól na które zostają przemieszczone Wieża i Król zależne są od aktualnego koloru gracza i rodzaju roszady.

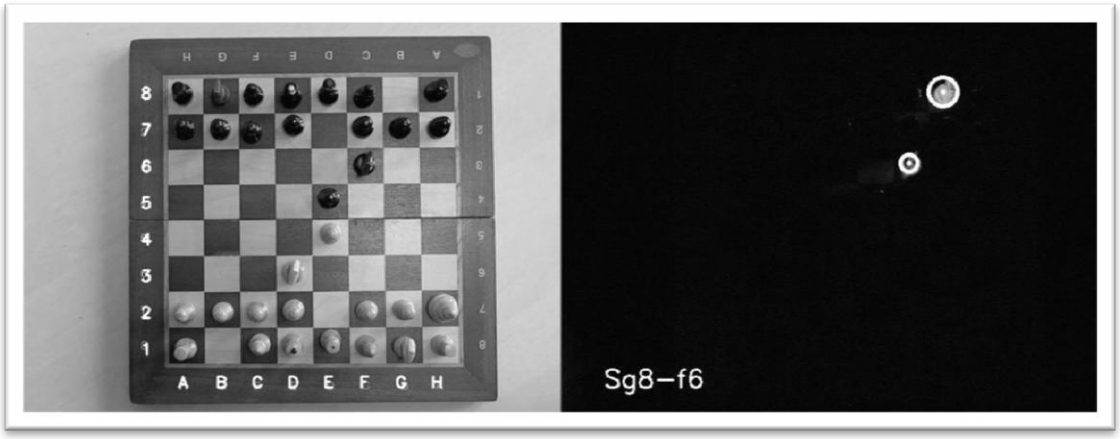
## 4.6. Przykład działania programu

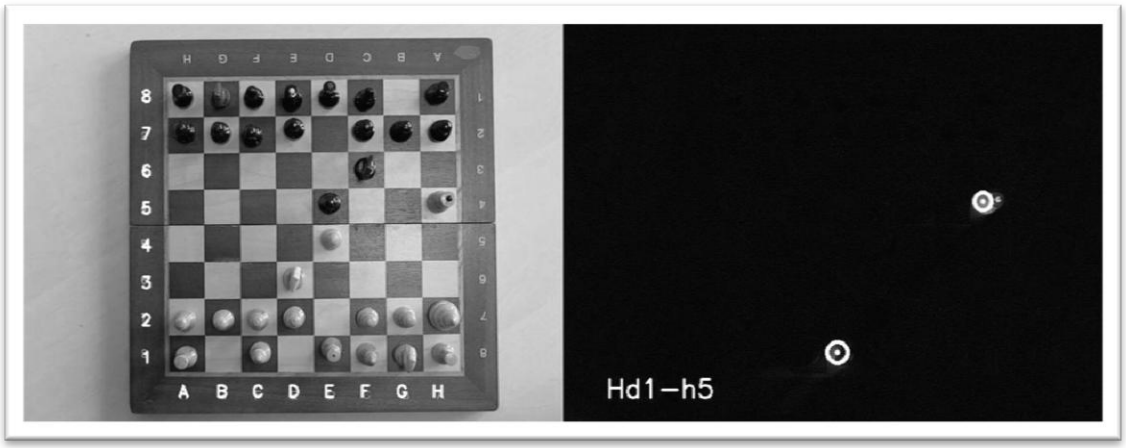
Efekt działania programu przygotowany jest na podstawie nakręconego krótkiego filmu zawierającego przebieg 10 ruchów i prezentowany na zdjęciach poniżej. Niektóre z tych ruchów są wykonywane przez figury które najczęściej sprawiały problemy przez niewłaściwe ich wykrywanie podczas realizacji pracy. Zarejestrowane ruchy z przebiegu całej partii zapisywane są w pliku tekstowym dając możliwość wglądu użytkownikowi i archiwizacji.

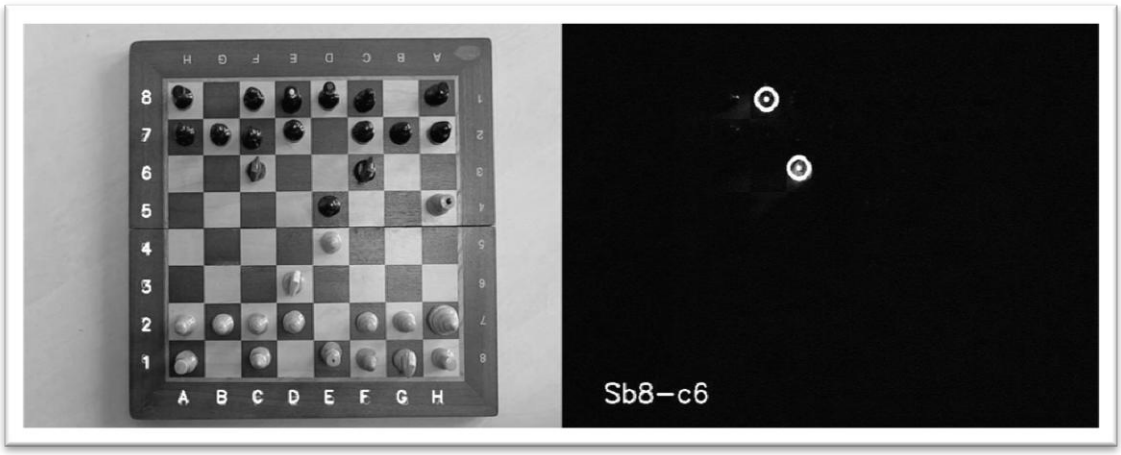


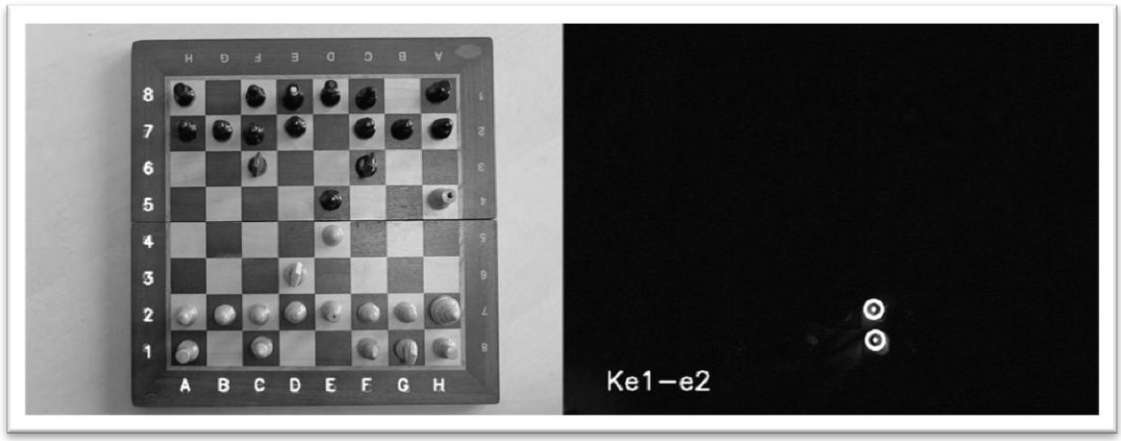



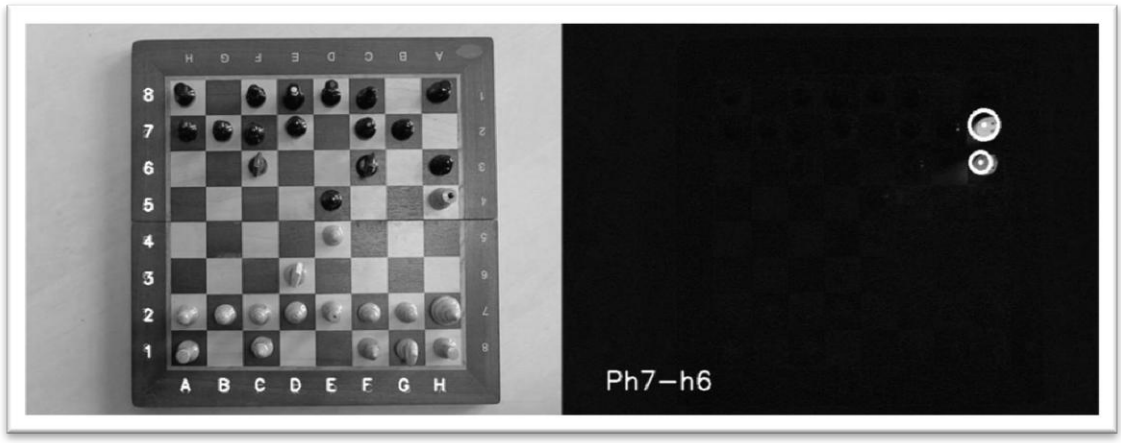

Sb1–d3

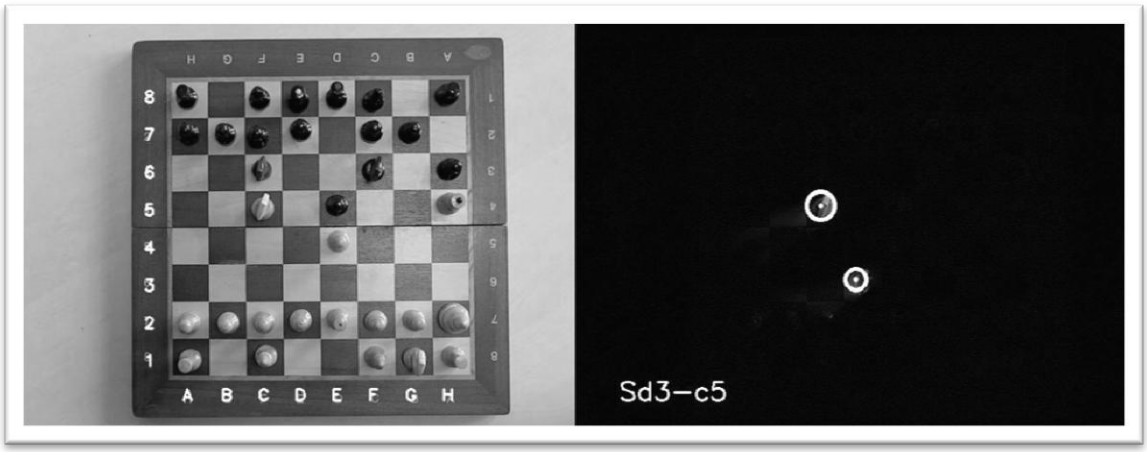

Sg8–f6

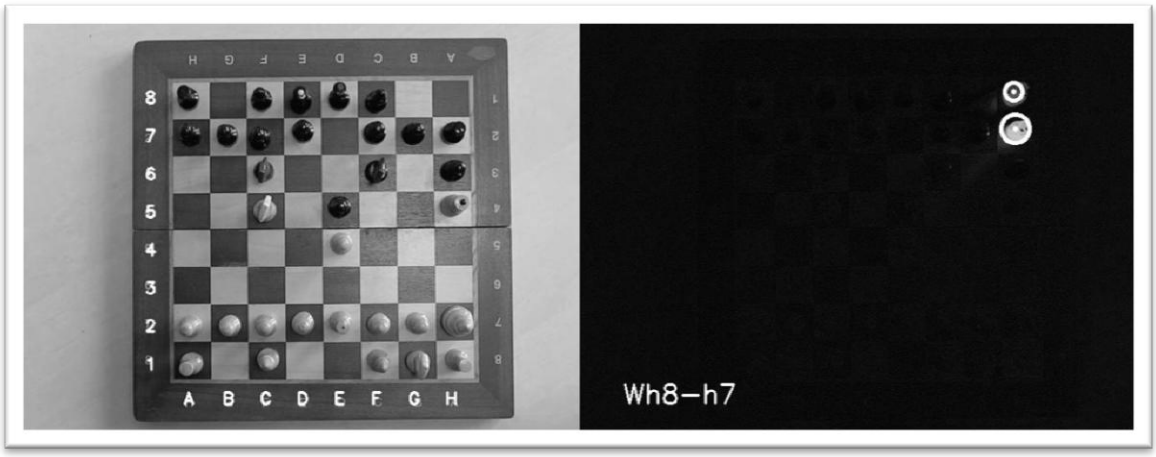

Hd1–h5


Sb8–c6

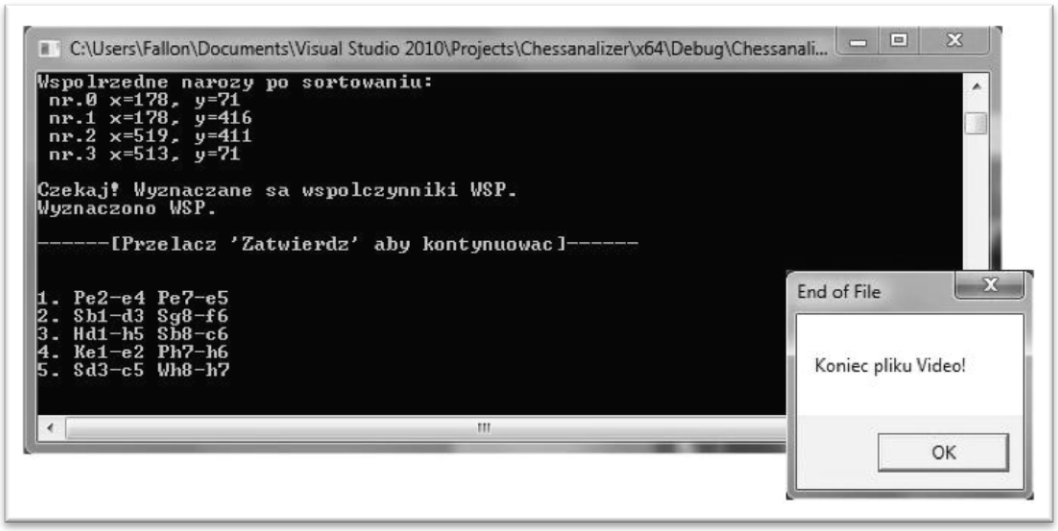

Ke1–e2


Ph7–h6




**Rys. 4.17 – 4.26.** Wynik rozpoznawania przebiegu gry na przykładzie dziesięciu kolejnych ruchów.




**Rys. 4.27.** Zapis dziesięciu wykonanych ruchów w notacji algebraicznej.

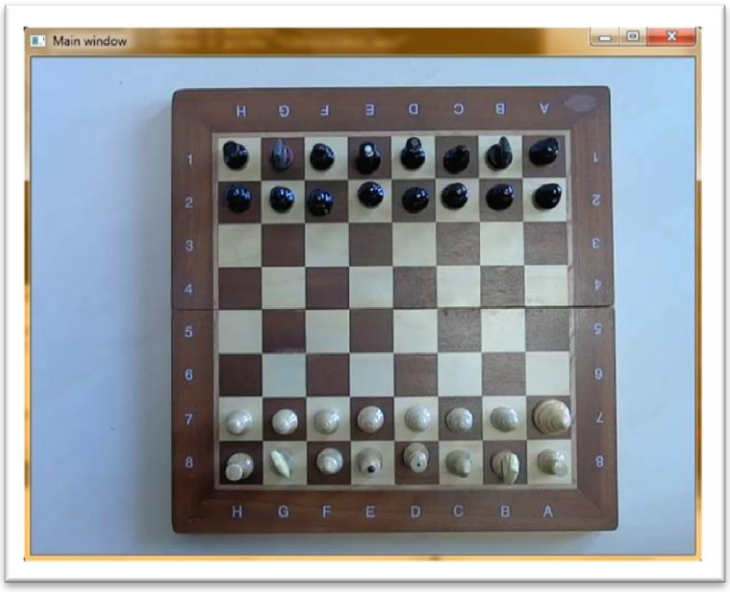
# 5. Testy

Czas przebiegu partii szachowej może wynosić nawet 2 godziny, choć często w turniejach szachowych jest on ograniczany np.; do 45min. Z tego powodu do pracy zostały nakręcone krótkie filmiki, zawierające jedynie kilka ruchów, pozwalających testować ich wykrywanie i analizę. Wykorzystywano również obraz nagrywany w czasie rzeczywistym pochodzący z kamery internetowej. Testom podlegały różne metody wykrywania figur opisane w podrozdziale 4.3, działanie metody wykrywania naroży szachownicy dla różnych warunków, działanie programu dla różnych zestawów szachów ( w skład zestawu wchodzi plansza szachowa oraz komplet figur ciemnych i jasnych) oraz różne metody wykrywania ręki lub jej ruchu.

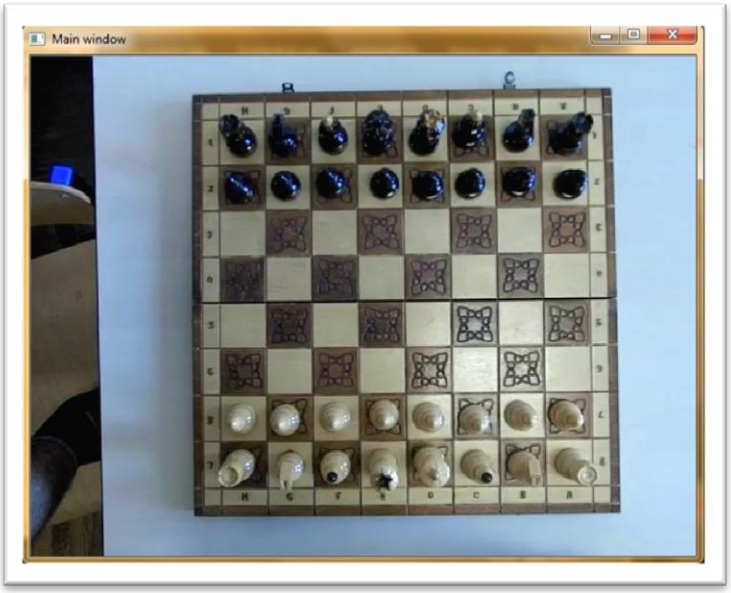
**ZESTAWY SZACHÓW**

Do realizacji pracy wykorzystywano trzy zestawy szachowe różniące się między sobą. Miało to na celu przetestowanie programu pod kątem uniwersalności działania. Poniżej zamieszczono zdjęcia tych zestawów:


**Rys. 5.1.** Zestaw szachów nr.1.




**Rys. 5.2.** Zestaw szachów nr.2.



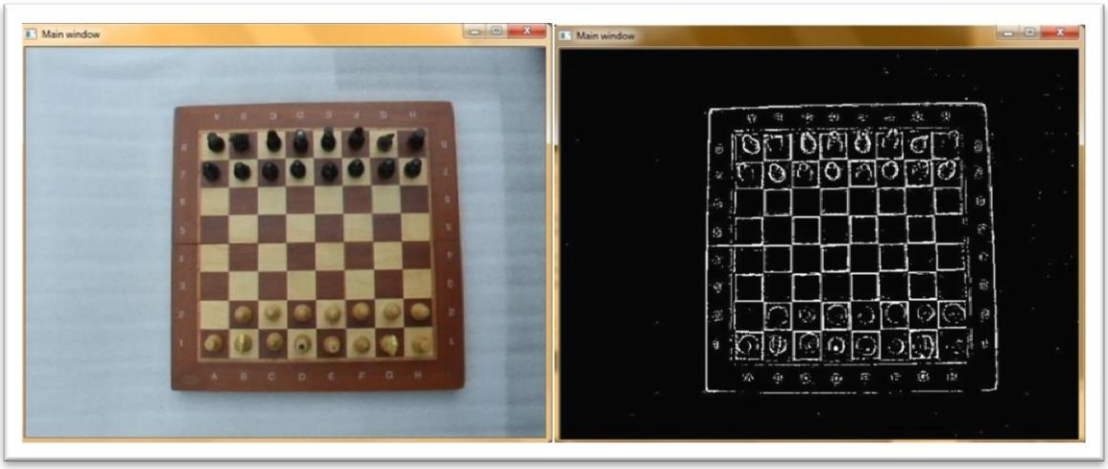

**Rys. 5.3.** Zestaw szachów nr.3.

Jak widać, zestaw 1 i 2 mają proste wzory, natomiast zestaw 3 posiada zdobienia na polach, które powodowały niewykrycie figury jako okręgu, pomimo, że po odjęciu obrazów wykryty obszar był dobrze widoczny. Z powodu obecności widocznego wzoru w wykrytym obszarze ruchu, niemożliwe było wykrycie krawędzi tworzącej okrąg. Jednak również w przypadku szachownic o gładkich polach, nie zawsze następowało właściwe działanie programu. W przypadku zestawu pierwszego, którego figury mają taki sam kolor jak pola, figury zlewały się z polami

i dlatego program niekiedy ich nie wykrywał. Bardzo zbliżona do koloru skóry barwa szachownicy, przeszkadzała również we właściwym wykryciu kiedy nastąpił ruch. Jedynym zestawem, przy którym udało się zapewnić poprawne działanie programu, był zestaw drugi.

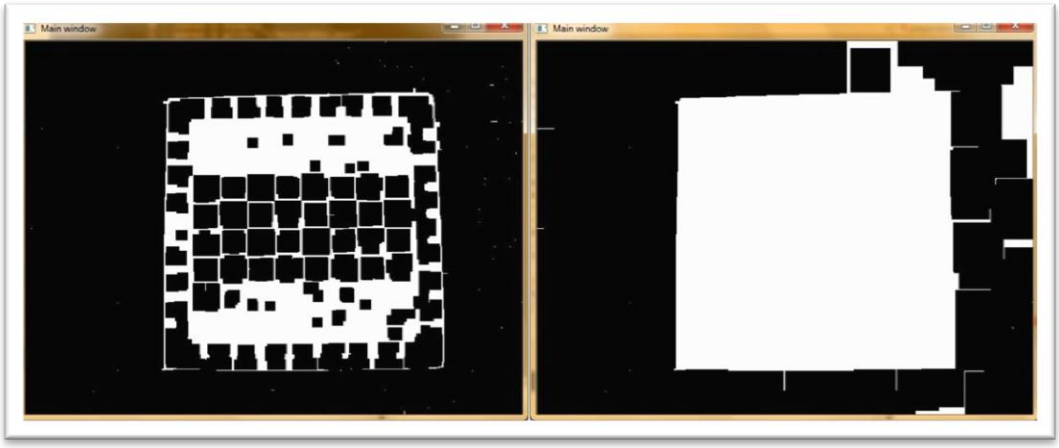
**METODA WYKRYWANIA NAROŻY**

Do wykrycia naroży szachownicy potrzebnych do identyfikacji transformaty płaskiej, wykorzystywano metodę, która nie działa w pełni automatycznie, ponieważ wymaga interakcji z użytkownikiem. Metoda była testowana pod kątem działania przy różnych ustawieniach planszy, różnych rodzajach szachownic, odporności na występowanie innych elementów w kadrze oraz jakości obrazu kamery. Metoda ta dawała dobre wyniki przy różnych rodzajach szachownic i ich kącie ustawienia oraz przy występowaniu innych elementów i krawędzi na obrazie. Jednak w testach pojawiały się problemy wynikające z niewłaściwego ustawienia kamery połączonego z niską jakością obrazu (np. obraz nieostry). Przy stosowaniu zbyt krótkiej ogniskowej i małej odległości kamery brzegi szachownicy ulegały zjawisku dystorsji beczkowej. Z tego powodu linia boku szachownicy nie jest prosta tylko składa się z kilku krótszych prostoliniowych odcinków. Ponieważ transformata Hougha zwraca w pierwszej kolejności najdłuższe linie, może się zdarzyć, że zwróci najdłuższe odcinki należące do jednego boku. Dodatkowo sytuację pogorsza słaba jakość obrazu, powodując nieciągłość krawędzi boku po operacji progowania adaptacyjnego.

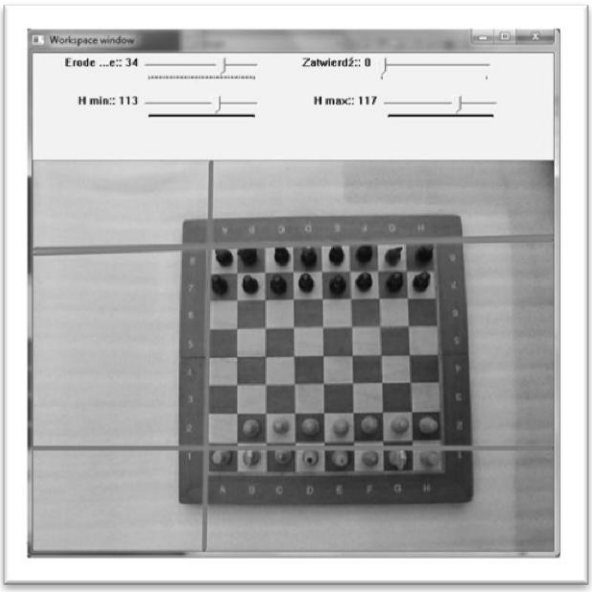



**Rys. 5.4.** Nieciągłość krawędzi dla obrazu o niskiej jakości.

Stosowanie sześciokrotnej operacji dylatacji, a następnie takiej samej ilości iteracji erozji uzupełniającej i wygładzającej obszar szachownicy w przypadku innych filmów testowych, nie rozwiązuje problemu nieciągłości. Dopiero zastosowanie morfologicznego zamknięcia dla dwudziestoczterokrotnej iteracji umożliwia prawidłowe wykrywanie linii dla tego przypadku. Jednak operacja ta powoduje zbytni i nierównomierny rozrost obszaru szachownicy w pozostałych testowanych przypadkach, gdy opisywane w podrozdziale 4.2 operacje doskonale się sprawdzały.

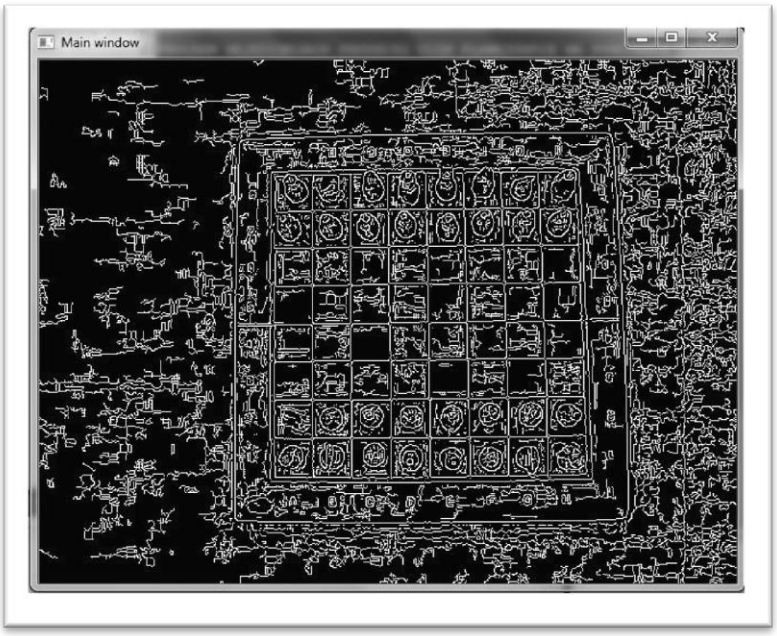



**Rys. 5.5.** Obszar szachownicy po wykonaniu sześciokrotnej operacji dylatacji i erozji(po lewej). Efekt zastosowania operacji zamknięcia dla 24 iteracji (po prawej), dopiero dla konturu takiego obszaru udało się właściwie wykryć linie będące krawędziami.



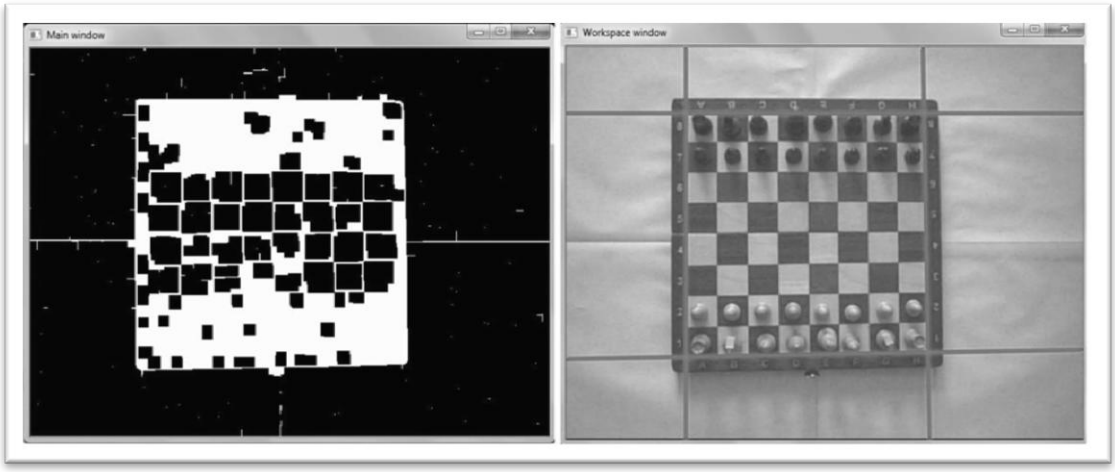

**Rys. 5.6**. Przykład błędnego wykrywania krawędzi boków szachownicy.

Zastąpienie operacji progowania adaptacyjnego filtrem krawędziowym Canny'ego w celu wykrycia krawędzi szachownicy, nie poprawia sytuacji. Funkcja ta wykrywa bowiem znaczną ilość krawędzi nie należących do planszy, znacząco utrudniając dalszy proces przetwarzania i analizy.




Rys. 5.7. Efekt wykrywania krawędzi szachownicy funkcją ***Canny()***.

Występowały również przypadki, gdzie pomimo niskiej rozdzielczości optycznej kamery i złych warunków oświetlenia krawędzie boków szachownicy były poprawnie wykrywane.



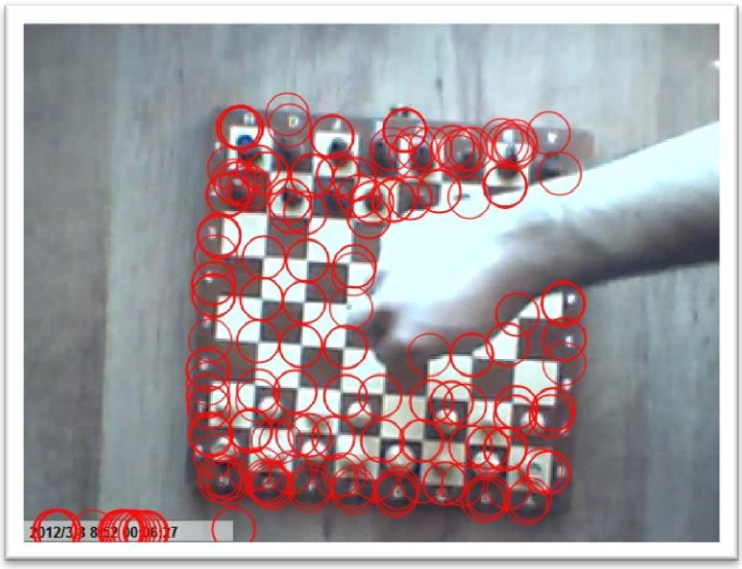

**Rys. 5.8.** Przykład prawidłowego wykrycia krawędzi pomimo niskiej jakości obrazu rejestrowanego przez kamerę internetową. Obraz po operacji progowania adaptacyjnego, dylatacji oraz erozji (po lewej). Linie wykryte transformatą Hougha (po prawej).

**METODY WYKRYWANIA RĘKI**

Prawidłowe działanie metody różnicowo – krawędziowej wykrywającej ruchy figur, zależy od właściwego wykrywania ruchu ręki, wyznaczającego klatki wideo, które należy odejmować. Na chwilę obecną w programie zaimplementowana jest metoda wykrywania ręki na podstawie składowych barw w przestrzeni HSV. Metoda pomimo sprawdzenia się w niektórych przypadkach, nie daje pewności działania i niezawodności przy posługiwaniu się różnymi zestawami szachów, nawet w tych samych warunkach oświetlenia. Częstymi problemami związanymi z tym sposobem wykrywania, jest pojawienie się fragmentów ręki na odejmowanych obrazach, co powoduje wykrywanie fałszywych okręgów. Wynika to z faktu, że nie cała powierzchnia ręki ma barwę w zakresie wyszukiwanym przez metodę. Innym poważniejszym problemem jest zanikanie poszukiwanej barwy, gdy ręka znajdowała się nad szachownicą, co również powoduje wykrywanie fałszywych okręgów, lecz tym razem w obrębie pól szachowych. Sytuacja taka powoduje, że mogą zostać wykrywane ruchy nie wykonane (a prawdopodobne do wystąpienia).

Do realizacji tego zagadnienia, branych było pod uwagę kilka alternatywnych metod wykrywania ruchu ręki na podstawie przemieszczenie, poszukania cech lub różnic w obrazie.

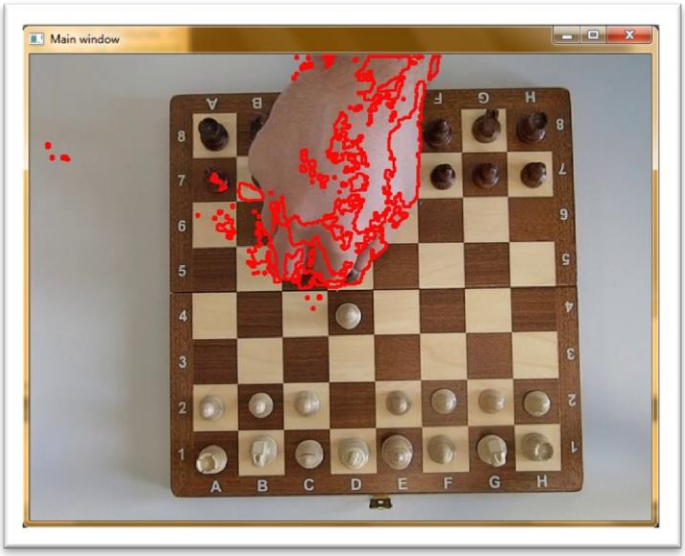
Pierwsza metoda wykrywania ruchu ręki polega na wyznaczeniu pola przepływu optycznego. Do jego wyznaczenia użyto metody gradientowej LK (Lukas–Kandake). Algorytm ten pozwala wykrywać większy ruch dzięki analizie obrazu od poziomu o mniejszej szczegółowości do poziomu o większej liczbie szczegółów, oraz analizie wcześniejszych jego iteracji. W metodzie tej najpierw poszukiwane są punkty, które nadają się do śledzenia. Takimi punktami są naroża, gdyż wykazują wyraźne zmiany jasności w dwóch różnych kierunkach. Wykrywanie naroży w analizowanych obrazach jest realizowane przez funkcję ***goodFeaturesToTrack()***. Okazało się jednak na tym etapie, że funkcja nie znajduje punktów na obszarze ręki, gdyż głównie punkty szachownicy spełniają podane warunki.



**Rys. 5.9**. Punkty nadające się do śledzenia wyznaczone funkcją ***goodFeaturesToTrack()***.

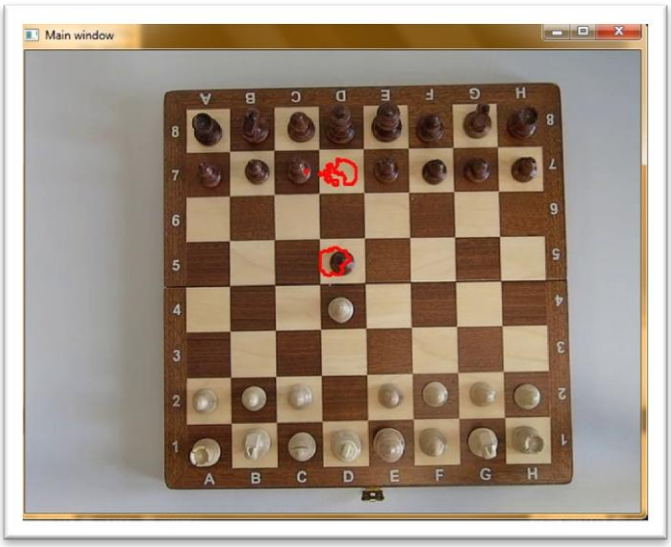
W metodzie tej występuje również inny problem w sytuacji, gdy ręka nad szachownicą przestaje się ruszać i wektory przepływu są równe zero. Aby określić czy ręka została przesunięta nad planszę czy zabrana z kadru, należy zastosować obliczenia uwzględniające kierunek wykonywania ruchu.

Kolejną testowaną metodą jest estymacja tła przy użyciu gotowej funkcji z OpenCV. Funkcja estymująca wykonywana jest przez klasę, zatem należało utworzyć jej obiekt ***BackgroundSubtractorMOG2 bg***. Funkcja ta jest detektorem wykorzystującym mieszaniny Gaussowskie (informacje na ten temat dostępne są w [I10]). Parametry tej funkcji ustawione są domyślnie, wystarczy tylko ustawić liczbę rozkładów, ewentualnie wyłączyć opcję wykrywania cieni. Całą detekcję obiektów nie będących tłem wykonuje funkcja ***bg.operator*** *()(ramka,fore);* do której przekazany jest obraz źródłowy i docelowy. Tworzy on model tła wraz z binarną mapą obiektów pierwszoplanowych. Zapis modelu tła w macierzy wykonać można przy użyciu funkcji ***bg.getBackgroundImage****(back)*. Usunięcie szumów z maski wykonane zostaje operacją otwarcia morfologicznego. Następnie na podstawie mapy obiektów wyrywane są ich kontury. Zdecydowaną zaletą funkcji jest fakt, że wykrywa rękę od samego początku jak tylko pojawi się w kadrze i przez cały czas jej obecności nad planszą. Rozwiązuje to problem wykonywania operacji odjęcia i wykrycia przesunięć figur w trakcie jego wykonywania. Umożliwia również rozpoznawanie ręki w przypadku różnych kolorów skóry graczy.




**Rys. 5.10.** Wykrycie ruchu ręki przy użyciu metody estymacji tła *BackgroundSubtractorMOG2*.

W stosowaniu tej metody pojawia się problem wykrywania miejsc, gdzie nastąpiła zmiana pozycji figur. Po zabraniu ręki z kadru, przesunięte figury wykrywane są jeszcze przez jakiś czas jako obiekty pierwszoplanowe nie będące tłem. Obiekty pozostające bez ruchu, dopiero po pewnym czasie uznawane są przez funkcję jako tło. Spowodować to może utrzymywanie stanu flagi informującej o obecności ręki nawet do czasu pojawienia się ręki przeciwnika na obrazie. Aby wyeliminować ten problem, należało by sprawdzać jak wielkie jest pole powierzchni konturów.




**Rys. 5.11.** Wykrywanie pozycji na których nastąpił ruch figur.

Najważniejszą przeszkodą okazało się jednak wykrywanie miejsc lokalnych zmian oświetlenia jako obiektów.

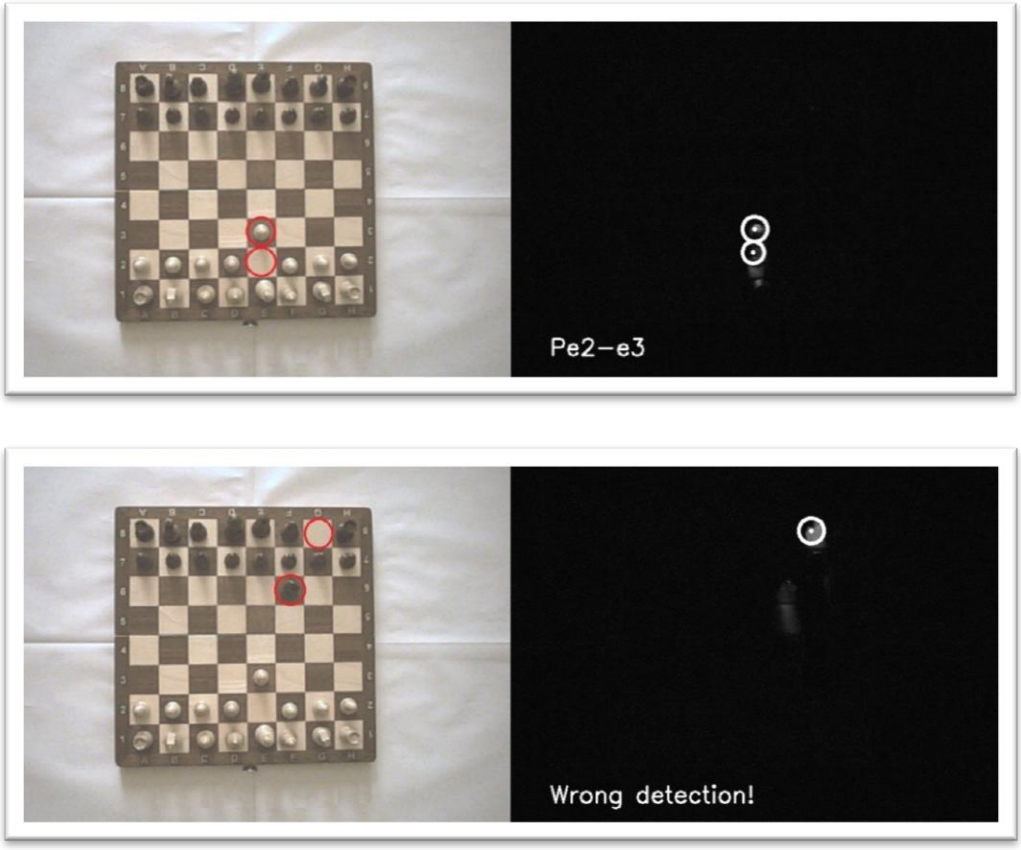

**Rys. 5.12**. Wykrywanie konturów lokalnych zmian oświetlenia.

Następna testowana metoda wykorzystuje różnicę w obrazie poprzez odejmowanie kolejno po sobie występujących klatek. Jednak podobnie jak w przypadku wcześniej opisywanej metody, problem pojawia się w sytuacji, gdy ręka zatrzymuje się nad planszą. Może on zostać rozwiązany przez zastosowanie systemu odejmowania obrazów z metody różnicowo – krawędziowej, wykorzystującego obraz bazowy bez ręki w kadrze. Określenie, kiedy nastąpił koniec ruchu, odbywa się poprzez obliczenie, czy zmiany w obrazie stanowią np. mniej niż 5% powierzchni całkowitej. Wynika to z faktu, że po wykonanym ruchu wykryte byłyby zmiany spowodowane przemieszczeniem figur, stanowiące niewielki procent całej powierzchni. Metoda ta nie została zaimplementowała, jednak stanowi dobrą alternatywę dla metody zawartej w programie.

**DZIAŁANIE PROGRAMU**

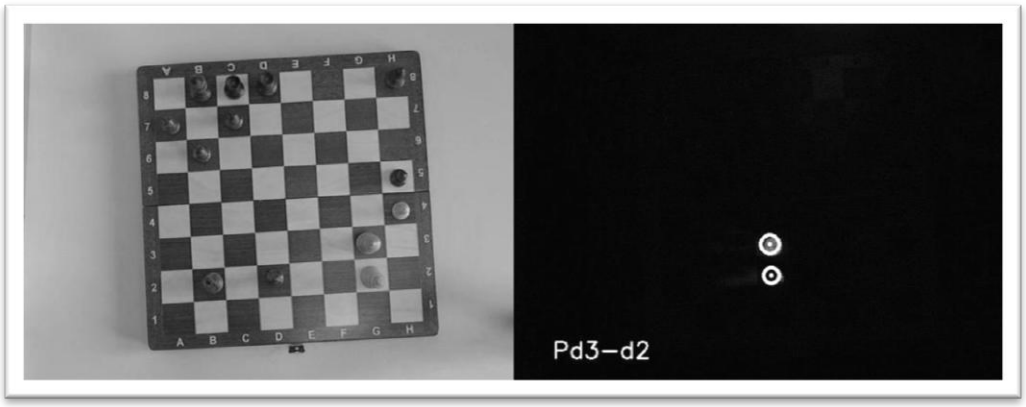
Sprawdzenie poprawności działania algorytmu rozpoznawania rodzaju wykonanego ruchu, odbywało się na podstawie obrazu pobieranego w czasie rzeczywistym z kamery i kilkunastu krótkich plików wideo. Gdy źródłem obrazu była kamera internetowa, nie udało się uzyskać prawidłowego działania programu dłużej niż dla kilku ruchów, z powodu nieprawidłowego rozpoznania miejsc, gdzie nastąpiło przemieszczenie, lub problemów z algorytmem wykrywania ręki. Zwiększenie

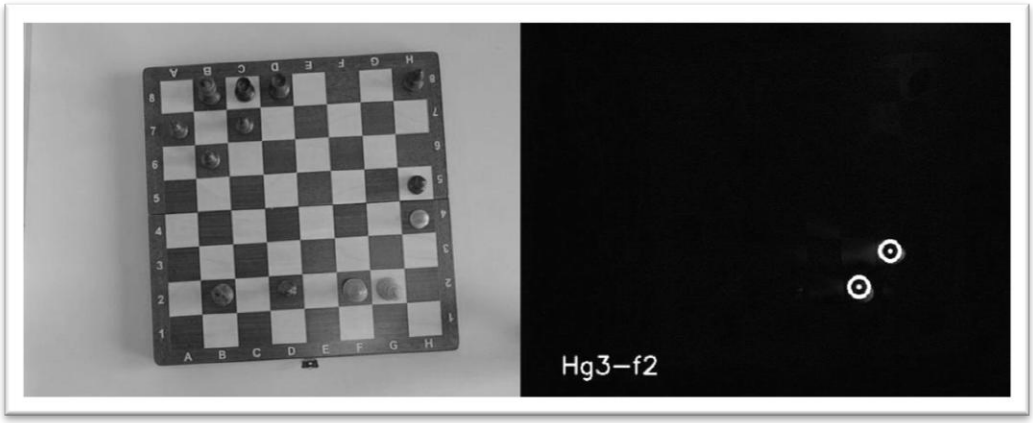
poprawności wykrywania można otrzymać, stosując kamerę posiadającą wyższą rozdzielczość optyczną.

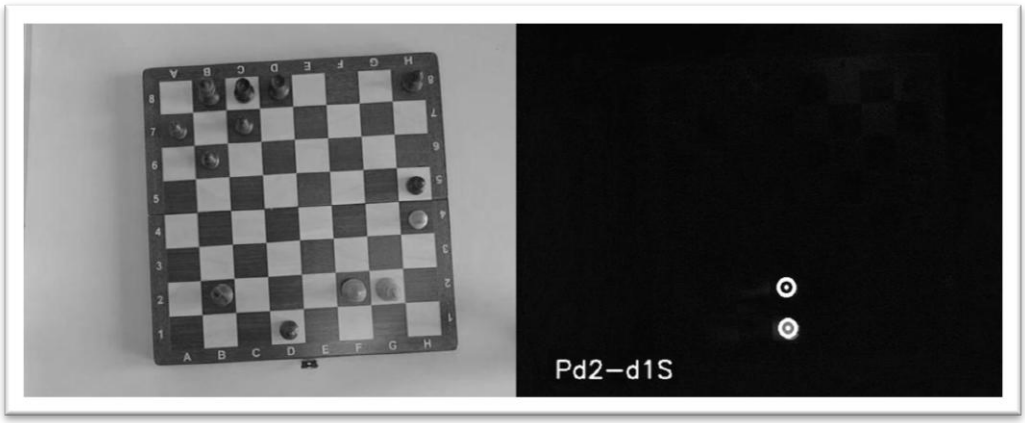


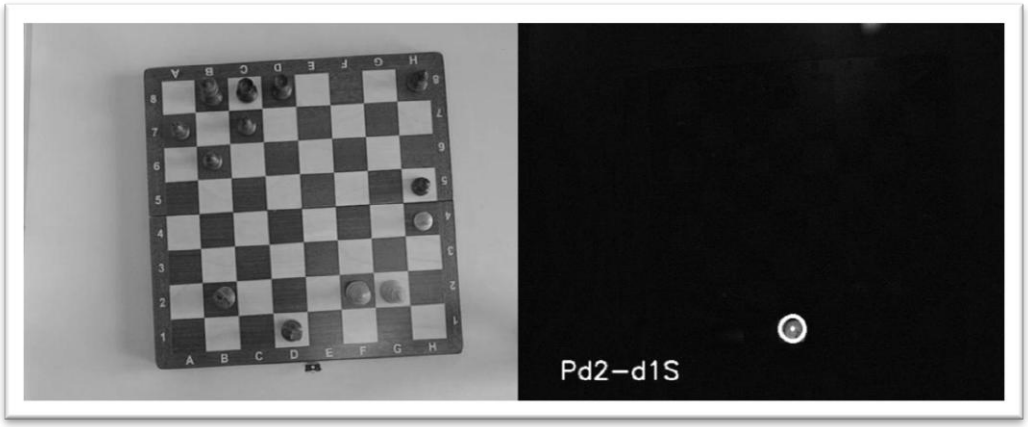

**Rys. 5.10 – 5.11.** Niewykrycie pozycji figury przy drugim ruchu na obrazie pochodzącym z kamery internetowej.

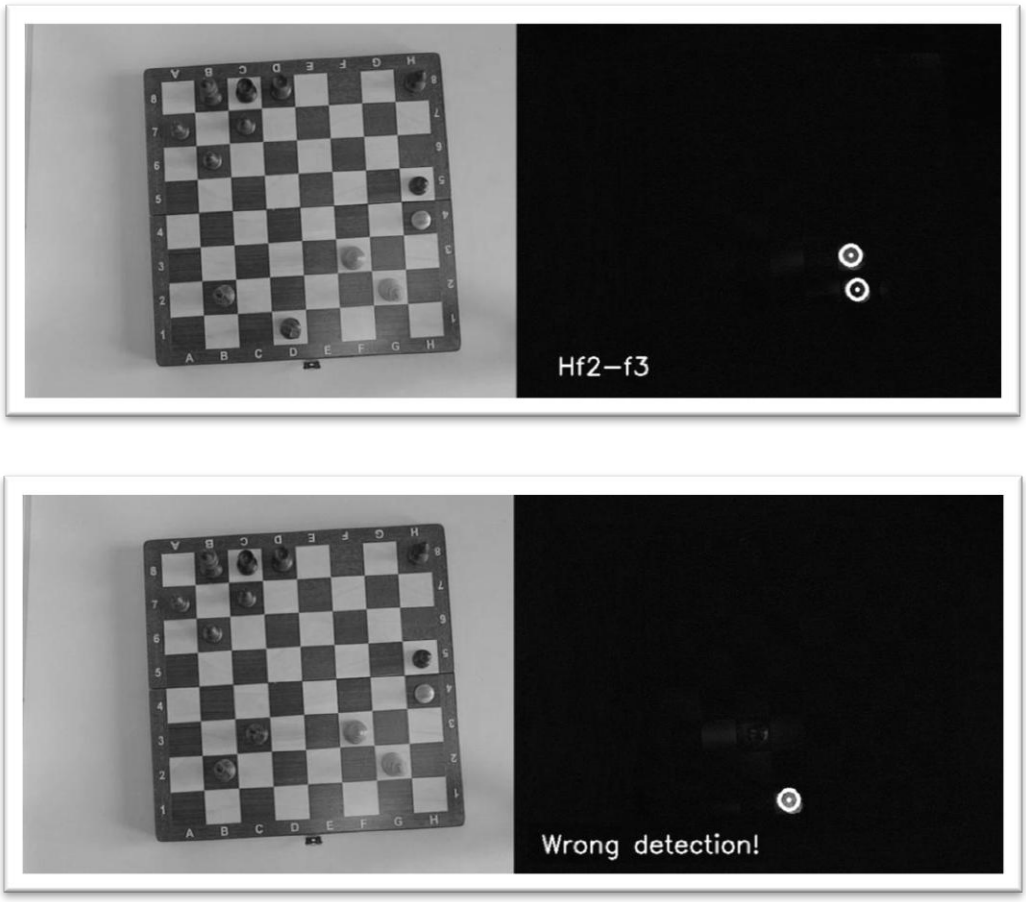
Filmy zawierają przykładowe posunięcia figur: zmianę pozycji, bicie, awans, czy roszadę, na których podstawie sprawdzana była poprawność ich rozpoznawania i odporność na błędne wykrycie okręgów. Większość z tych ruchów można wykonać w pierwszych kilku posunięciach. Dla przypadku awansu pionka nakręcono film, w którym figury posiadają inne ustawienia początkowe. Podczas testów w programie ustawiono wartości dla poszczególnych figur, takie jak widoczne były na filmie. Wykonany awans został rozpoznany prawidłowo.


Pd3–d2


Hg3–f2

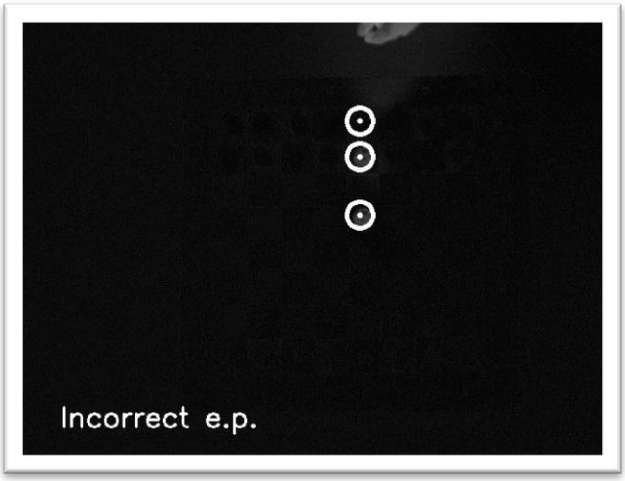

Pd2–d1S


Pd2–d1S




**Rys. 5.12 – 5.17**. Przykład wykrywania ruchu awansu pionka.

Działanie pozostałych algorytmów rozpoznawania rodzajów ruchu również zostało przetestowane i działa właściwie. W sytuacji wykrycia liczby figur odpowiadającej danemu ruchowi pomimo jego nie wystąpienia, zostaje prawidłowo wykryty i zakomunikowany błąd.




**Rys. 5.18.** Komunikacja o nierozpoznaniu bicia w przelocie, w sytuacji błędnego wykrycia trzech okręgów.

# 6. Podsumowanie

Tematem niniejszej pracy była rejestracja przebiegu gier planszowych na podstawie obrazu, realizowana na przykładzie gry w szachy. Do tego celu został napisany program ***Chessanalizer.exe***. Temat wymagał wiedzy z zakresu przetwarzania i analizy informacji zawartej w obrazie oraz podstawowych zasad gry w szachy. Z powodu braku uniwersalnych metod działających w każdych warunkach, niemożliwe okazało się przewidzenie wszystkich problemów na etapie planowania i projektowania programu.

Pierwsza część pracy wymagała zapoznania się z informacjami teoretycznymi na temat metod i mechanizmów wykorzystywanych do przetwarzania i rozpoznawania obrazów, które były wykorzystane w praktycznej realizacji pracy. Część praktyczna polegała na implementacji algorytmów, które okazały się najbardziej właściwe do osiągnięcia zamierzonego celu pracy. Obejmowała ona napisanie funkcji: interfejsu użytkownika, wyznaczających współrzędne szachownicy na podstawie współrzędnych obrazu, wykrywających figury i ruch ręki oraz rozpoznających wykonane ruchy i ich zapis. Po napisaniu programu i przeprowadzeniu testów, stwierdzono, że stosowane metody i warunki w jaki zostały testowane, nie dają możliwości stworzenia programu działającego bezbłędnie i niezależnie od warunków oświetleniowych. Oprócz problemów, jakie wynikały w metodach wykrywania ruchu ręki, najistotniejszymi błędami związanymi z sednem zagadnienia wykrywania było częste niewykrywanie pozycji figur, które zostały przemieszczone podczas wykonywania ruchu. Trudności wynikały również w sytuacjach zmian warunków oświetleniowych i zestawów szachów. Zmuszało by to do zwiększenia złożoności działania programu i interakcji użytkownika, spowodowanej koniecznością zmian parametrów funkcji użytych w programie dla różnych konfiguracji warunków. W testowanych przykładach, udało się poprawnie przeprowadzić tylko jeden proces rozpoznawania przebiegu gry, dla filmu zawierającego dziesięć posunięć trwających 1min i 50s. W testowanym przykładzie obraz pochodził z aparatu cyfrowego umożliwiającego nagrywanie wideo w wysokiej rozdzielczości optycznej, a zestaw szachów miał odróżniające się kolory figury i pól szachowych. Zakres stosowalności programu jest ograniczony do ściśle określonych, dobrych warunków oświetleniowych i wysokiej jakości obrazu.

W dalszych pracach nad tematem można byłoby rozważyć inne metody cyfrowego przetwarzania obrazu do realizacji zagadnienia pracy, oraz dopracować funkcje wykonujące poszczególne części programu t.j. wykrywanie ruchu ręki. Możliwościami rozwoju programu, jest wprowadzenie mechanizmów rozpoznawania wykonywanych ruchów zależnie od rodzaju figury, a także dzięki pewnym modyfikacjom, rozszerzenie programu o rozpoznawanie przebiegu innych gier bazujących na planszy szachowej.

# Bibliografia


[1] D. King „Szachy. Od pierwszego ruchu do mata" wyd. Atlas, Wrocław 2000r.

[2] G. Bradski, A. Kaehler „Learning OpenCV: Computer Vision with the OpenCV Library" O'Reilly Media 2008r.

[3] T. Hebisz „Multimedia i Grafika Komputerowa: Modele Barw" Instytut Sterowania i Systemów Informatycznych 2003r.

[4] R. Tadeusiewicz, P. Korohoda „Komputerowa analiza i przetwarzanie obrazów" Wyd. Fundacja Postępu Telekomunikacji Kraków 1997r.

[5] M. Denkowski, P. Mikołajczak „Przetwarzanie obrazów cyfrowych – laboratorium" Uniwersytet Marii Curie-Skłodowskiej, Lublin 2011r.

[6] S. Fuksa, W. Byrski „Czteropunktowa metoda identyfikacji transformacji stereowizyjnej", Automatyka AGH w Krakowie 2005, Tom9, zeszyt 3, s.533–541.

[7] H.K. Yuen, J. Princen, J. Illingworth and J. Kittler „A comparative study of hough transform methods for circle finding" Department of Electronics and Electricial Engineering University of Surrey, Guildford U.K.

[8] Josep Llados „The Hough Transform as a Tool for Image Analysis" Computer Vision Center – Dept. Informatica Universitat Autonoma de Barcelona 2003r.

[9] S. Wojas „Metody Przetwarzania Obrazów z Wykorzystaniem Biblioteko OpenCV" praca magisterska AGH Kraków 2010r.

[10] Suzuki, S. and Abe, K., *Topological Structural Analysis of Digitized Binary Images by Border Following*. CVGIP 30 1, pp 32-46 (1985)

# Źródła Internetowe i Artykuły


[I1] Strona projektu OpenCV: http://sourceforge.net/projects/opencvlibrary/

[I2] Dokumentacja OpenCV v2.4.2: http://docs.opencv.org/

[I3] OpenCV komputerowa wizja w praktyce „Software Developer's Journal" 02/2009

[I4] Strona programu MSDN Academic Alliance: http://www.microsoft.com/poland/edukacja/dreamsparkpremium_dla_studenta.aspx

[I5] http://www.optyczne.pl/38-s%C5%82ownik-MTF.html

[I6] Ch. Baadsgaard, S. Eskildsen, K. Rodil, S. Sakthicel „ Adaptive Threshold: Computer Vision".

[I7] http://pl.wikipedia.org/wiki/HSV_(grafika)

[I8] http://www.fotoporadnik.pl/histogram-szczegolowo-en.html

[I9] http://pl.wikibooks.org/wiki/Szachy/Roszada

[I10] http://personal.ee.surrey.ac.uk/Personal/R.Bowden/publications/avbs01/avbs01.pdf

[I11] http://pwsz-tarnow.edu.pl/Informatyka-f, data odczytu VI.2009.

## Spis załączników:

Załącznik A – Opis plików programu *Chessanalizer*;

Załącznik B – Płyta CD;

# Załącznik A – opis plików programu *Chessanalizer*.

Pliki:

| | |
|---|---|
| Chessanalizer.cpp | – program główny, zawiera algorytm wykrywania ruchów. |
| chessboard.h | – plik nagłówkowy zawierający deklarację klasy *chessboard* służącej do wykrywania naroży szachownicy oraz wyznaczenia położenia na szachownicy w notacji algebraicznej. |
| chessboard.cpp | – zawiera definicje funkcji znajdujących się w pliku chessboard.h. |
| figury.h | – plik nagłówkowy zawierający deklarację klasy *figures* reprezentującej figury szachowe oraz metody które są na nich wykonywane. |
| figury.cpp | – zawiera definicje funkcji znajdujących się w pliku figury.h. |
| parametry.h | – plik nagłówkowy zawierający deklarację klasy *parametry* zawierający zmienne odpowiedzialne za działanie algorytmu programu oraz przekazywane jako argumenty funkcji przetwarzających obraz. |
| parametry.cpp | – zawiera definicje funkcji znajdujących się w pliku parametry.h. |
| transformata.h | – plik nagłówkowy zawierający deklarację klasy *transformata* realizującej czteropunktową transformatę płaską w celu wyznaczenia rzeczywistych współrzędnych szachownicy. |
| transformata.cpp | – zawiera definicje funkcji znajdujących się w pliku transformata.h. |
| FcnMovement.cpp | – zawiera definicje funkcji wykrywających zmianę położenia figur, identyfikujących wykonany ruch oraz wykrywającej rękę gracza. |
| FcnUsefull.cpp | – plik zawiera definicję funkcji konwertujących, tworzących elementy interfejsu użytkownika (t.j. suwaki, okna), funkcje wyboru źródła obrazu. |
| mat.h | – zawiera deklarację struktur reprezentujących współrzędne pozycji, punkt i prostą. |
| macierz.h | – plik nagłówkowy zawierający deklarację klasy *transformata* zawierającej funkcje operacji macierzowych. |
| macierz.cpp | – zawiera definicje funkcji znajdujących się w pliku macierz.h. |

## Załącznik B – Płyta CD.

Zawartość płyty CD dołączonej do pracy:

- Elektroniczna wersja pracy magisterskiej w formacie PDF.
- Skompilowana wersja programu *Chessanalizer*.
- Pliki źródłowe programu (projekt MS Visual Studio 2010)
- Plik instalacyjny użytej w pracy  biblioteki OpenCV 2.3.0