Politechnika Krakowska
im. Tadeusza Kościuszki
Wydział Fizyki, Matematyki i Informatyki

Marcin Szewczyk
Numer albumu: 098117

Alternatywne metody poruszania kursorem

Alternative methods of moving graphic pointer

Praca magisterska na kierunku Informatyka

Praca wykonana pod kierunkiem:
dra Radosława Kycii

Uzgodniona ocena:.....

.....
podpisy promotora i recenzenta

Kraków 2017

Spis Treści

1. Wstęp

Obecny poziom zaawansowania technologicznego niesie ze sobą wiele możliwości. Produkowane są coraz mniejsze komputery, które umożliwiają interpretację otaczającego nas świata na wiele sposobów przy pomocy różnego rodzaju sensorów, czujników oraz kamer.

Jednym z podstawowych zagadnień, a zarazem bardzo interesujących, w których powyższe urządzenia mają zastosowanie jest realizacja interakcji człowiek-maszyna. Od czasów pierwszych układów scalonych projektanci tego typu urządzeń musieli podjąć trud przygotowania interfejsu, który umożliwiłby wymianę poleceń wykonywanych przez użytkownika oraz przetworzenie ich na konkretne akcje.

Myszka komputerowa, obok klawiatury, stanowi elementarne wyposażenie służące, jako urządzenie wejściowe od połowy XX wieku po dzisiejsze czasy [1]. Pierwsze jej realizacje były oparte o analizę zmiany położenia kulki w niej umiejscowionej, względem ruchów wykonanych myszką przy pomocy dłoni. Przez lata idea myszki komputerowej nie ulegała wielkim modyfikacjom, a jedynie wraz z rozwojem technologicznym dokonywano wymiany czujników na bardziej zaawansowane – najpierw diodowe, następnie laserowe.

Wraz ze wzrostem miniaturyzacji układów komputerowych oraz rozwojem komputerów przenośnych narodziła się idea touchpada. W przeciwieństwie do myszki komputerowej, użytkownik wykonuje ruchy palcem na plastikowym prostokącie, które są bezpośrednio przetwarzane i konwertowane na odpowiednie ruchy kursora myszki.

Innymi sposobami wykorzystywanymi do sterowania urządzeniami elektronicznymi to systemy oparte o interpretację ruchów wykonywanych przy pomocy różnych części ciała. Istnieją systemy, które analizują ruchy gałki ocznej, bądź interpretują położenie palców poprzez interpretację odbicia światła podczerwonego, czy gotowe rozwiązania oparte o system kamer umożliwiających interpretacje gestów wykonywanych przez użytkownika.

W ramach pracy dyplomowej zrealizowano aplikację dla systemów Unixowych służącą do sterowania kursorem myszki przy pomocy ruchów wykonywanych dłonią ręki. Główną motywacją przyczyniająca się do utworzenia niniejszego programu jest stworzenie kompletnego rozwiązania uproszczającego na płaszczyźnie komputer-człowiek.

2. Projekt

2.1 Cel pracy

Głównym celem niniejszej pracy magisterskiej jest zaprojektowanie oraz implementacja alternatywnej metody poruszania kursorem z wykorzystaniem rozpoznawania ruchów dłoni przy pomocy kamery w formie aplikacji desktopowej.

2.2 Zakres pracy

W trakcie pisania programu sprecyzowano dodatkowe założenia wynikające ze złożoności problemu rozpoznawanie gestów dłoni.

Pierwszym ograniczeniem jest rozpoznawanie koloru skóry, a konkretnie wyznaczenie podzbioru składowych przestrzeni kolorów, które mogłoby jednoznacznie wydobyć dłoń ręki z przetwarzanego obrazu. Z jednej strony kolor skóry każdego człowieka jest różny, z drugiej strony inne przedmioty niebędące skórą również mogłoby zostać zidentyfikowane, jako skóra człowieka. Kolejnym aspektem są możliwości techniczne oraz warunki, w jakich program miałby działać. Chodzi tutaj głównie o zależność, pomiędzy jakością obrazu, a poziomem naświetlenia pomieszczenia, w którym dany użytkownik miałby z niniejszej aplikacji korzystać. W związku z powyższym, postanowiono, że użytkownik będzie miał możliwości manualnej kalibracji podzbioru składowych przestrzeni kolorów odpowiadającej aktualnym warunkom. Co więcej zalecane jest, aby użytkownik wykorzystał rękawiczkę w kolorze znacznie różnym od otoczenia będącego w polu widzenia kamery, przy pomocy, której będzie chciał sterować kursorem myszki.

Rozpoznawanie kształtów na obrazie jest najbardziej skuteczne w pobliżu centralnego punktu przetwarzanej ramki obrazu. Ma to związek z faktem, iż kontury na obrzeżach ramki mogą zostać zniekształcone w wyniku słabej, jakości sprzętu, bądź stosowanych różnego rodzaju obiektywów, zmieniających geometrię elementów obrazu.

Ostatnim problemem, z którym należało się zmierzyć, jest fakt, iż użytkownik może wykonywać gesty na skraju spektrum kamery, co mogłoby wprowadzać znaczne zakłócenia. Poczyniono założenie, że gesty dłoni użytkownika będą analizowane jedynie w centralnym obszarze wyznaczonym przez prostokąt o bokach dwa razy mniejszych niż przetwarzana ramka.

3. Wstęp teoretyczny

W niniejszym rozdziale zawarte zostały wszystkie zagadnienia umożliwiające zrozumienie istoty działania programu zrealizowanego w ramach niniejszej pracy. Przedstawione są w nim wykorzystane technologie, algorytmy oraz rozwiązania umożliwiające poprawna prace programu.

3.1 Technologie

Program został zaimplementowany, jako aplikacja dla systemów opartych o architekturę UNIXową.

Najpopularniejszą biblioteką do przetwarzania obrazu jest open-sourcowa biblioteka OpenCV. Udostępniona jest na takie języki programowania jak: java, python, c++ i c oraz na wszystkie obecnie dostępne systemy operacyjne mobilne oraz desktopowe. Niniejszy pakiet udostępnia cały zestaw funkcji służących do obróbki obrazu od podstawowych filtrów po złożone operacje morfologiczne.

Do realizacji niniejszej pracy dyplomowej wykorzystano język programowania c++, zaliczany do wysokopoziomowych języków programowania. Umożliwia on programiście zarządzać pełnym cyklem życia obiektów: tworzenia, funkcjonowania oraz ich usuwania, (w odróżnieniu od np. Javy). W związku z powyższym tylko od umiejętności programisty zależy czy dostępne zasoby sprzętowe, systemowe zostaną w pełni wykorzystane. Dlatego też język C++ jest chętnie wykorzystywany do zastosowań, w których najważniejszy jest czas dostępu do zasobu – do których możemy zaliczyć np. obróbkę wideo.

Dodatkowo wykorzystano bibliotekę suinput, która służy do poruszania kursorem z poziomu kodu programu. W przypadku systemu Linux działa ona na zasadzie wysyłania komunikatów to urządzenia dostępnego pod ścieżką */dev/suinput*.

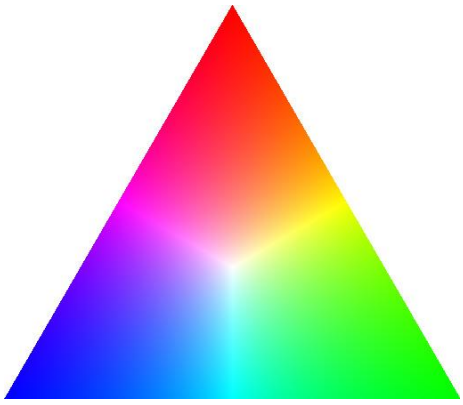
3.2 Algorytmy i rozwiązania

W tej sekcji zostały opisane podstawy teoretyczne operacji graficznych, które zostaną wykorzystane w implementacji programu opisanego w części praktycznej. Wówczas te operacje będą realizowane, w większości, przez metody zawarte w bibliotece OpenCv.

3.2.1 Przestrzeń kolorów

Sposób utrwalenia obrazu w postaci cyfrowej może zostać zrealizowany przy pomocy tzw. przestrzeni barw, cechujących się odmiennymi właściwościami. Jako najbardziej podstawową i najczęściej używaną przestrzeń barw uznaje się model RGB, którego nazwa pochodzi od pierwszych liter angielskich nazw kolorów: czerwony (ang. *red*), zielony (ang. *green*) i niebieski (ang. *blue*) – a jego reprezentacją jest sześcian RGB. Oznacza to, że każdy kolor w tej przestrzeni można przedstawić, jako kombinację trzech kolorów podstawowych, przy czym brak barw oznacza kolor czarny, a maksymalna mieszanka kolorów odpowiada kolorowi białemu.

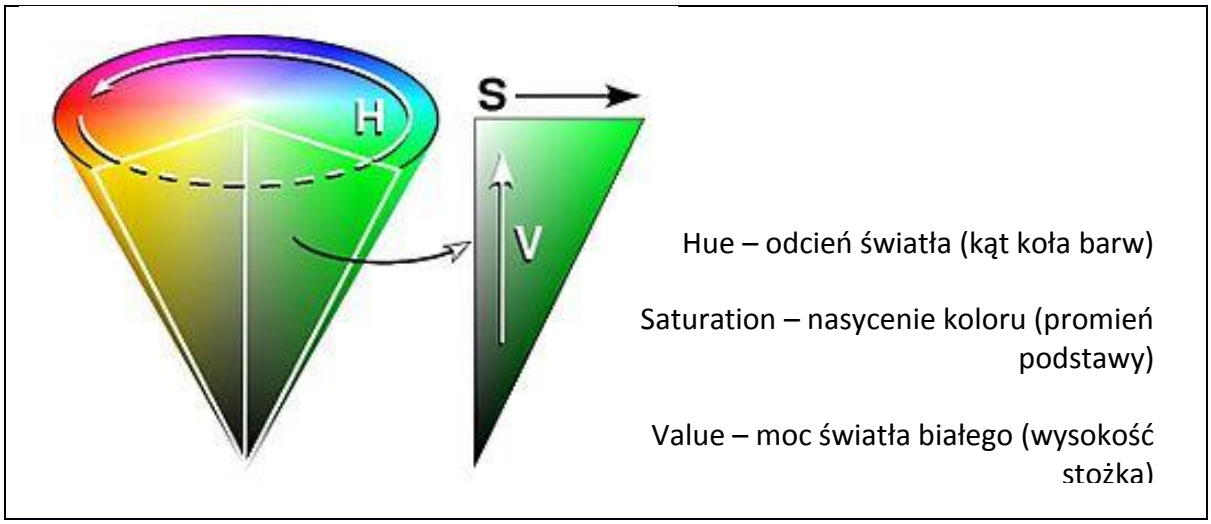
Model ten jest chętnie używany ze względu na prostotę użycia oraz małe zapotrzebowanie obliczeniowe, a co więcej dokładnie w taki sposób generowane są barwy na urządzeniach typu monitor i telewizor. Warto wspomnieć, że komórki czopkowe zlokalizowane na siatkówce ludzkiego oka, również analizują kolory pod względem złożenia trzech podstawowych kolorów: czerwonego, zielonego i niebieskiego [2]. Z drugiej strony prostota modelu niesie ze sobą wiele ograniczeń. Do wad modelu RGB wymienia się: wrażliwości składowych na zmiany oświetlenia, brak intuicyjności podczas wizualizacji barw na podstawie znajomości jej składowych R, G i B, czy percepcyjna niejednorodność (słaba zależność pomiędzy postrzeganą różnicą barw a ich odległością na trójkącie RGB – zobacz Rys. 1).



Rys. 1 Trójkąt przestrzeni kolorów RGB

Naprzeciw ograniczeniom związanym z modelem RGB wychodzi model HSV, którego główną zaletą jest nawiązanie do naturalnego sposobu interpretacji opisu barw. Każdy kolor

przedstawia się w nim przy pomocy trzech składowych: odcienia (z ang. *Hue*), nasycenia (z ang. *Saturation*) oraz jasności (z ang. *Value)* – jest on reprezentowany przy pomocy stożka HSV – zobacz Rys. 2. Odcień i nasycenie stanowią cechę jakościową, natomiast jasność stanowi cechę ilościową światła.



Rys. 2 Stożek przestrzeni kolorów HSV

Charakterystyka składowych:

- ➔ Oś V - wartości odzwierciedlają barwy achromatyczne, tzn. reprezentują poziomy szarości od czerni do bieli, zawierają się w zakresie od 0 do 1;
- ➔ Nasycenie S – identyfikowana, jako odległość punktu barwy od osi V, wartości z przedziału 0 do 1;
- ➔ Odcień H – mierzony, jako kąt obrotu wokół osi V, wartości z zakresu 0 do 360;

Do jego głównych zalet zaliczamy: naturalny opis barw oraz separacja składowych chromatycznych od achromatycznych, dzięki czemu barwa może być opisana przy pomocy tylko dwóch składowych H oraz S niezależnie od delty jasności V. Należy jednak liczyć się z występowaniem osobliwości: w przypadku H dla wszystkich barw achromatycznych, a w przypadku S dla czerni.

3.2.2 Progowanie obrazu

Przetwarzanie obrazów związane jest z koniecznością redukcji informacji nieistotnych z punktu widzenia przedmiotu analizy [3]. Procesem konwersji obrazu kolorowego, bądź w

skali szarości, na obraz dwuwartościowy jest binaryzacja. Efektem jej działania jest reprezentacja obrazu przy pomocy jedynie dwóch wartości 0 lub 1. Kluczowa rolę pełni dobór odpowiedniej metody binaryzacji, dzięki której obraz binarny będzie wartościowy w dalszym jego przetwarzaniu. Podstawowym sposobów realizacji binaryzacji jest tzw. progowanie (ang. *thresholding*). Polega ono na stosowaniu pewnej wartości progowej, powyżej której dany piksel klasyfikowany jest, jako 1, a w przeciwnym wypadku, jako 0. Dzięki takiemu podejściu progowanie jest bardzo szybkie, ale również mniej precyzyjnie w porównaniu do innych metod binaryzacji, które na przykład podejmują decyzje na podstawie sąsiedztwa każdego z pikseli.

3.2.3 Filtry cyfrowe

Jednym z podstawowych narzędzi służących przetwarzaniu obrazów są filtry cyfrowe. Są bardzo chętnie wykorzystywane ze względu na smukłe połączenie złożoności oraz prostoty użycia. Operacje takie opierają się na wyznaczaniu wartości danego punktu na podstawie wielu punktów otaczających dany punkt wejściowy, – dlatego w literaturze używa się często terminu operacji kontekstowych [3]. Zadaniem filtru jest wykonanie matematycznych operacji, będących odpowiednikiem funkcji matematycznych, mających na celu zmodyfikowanie obrazu wejściowego piksel po pikselu. Cechują się szerokim wachlarzem zastosowań, dzięki którym istnieje możliwość zniwelowania niepożądanych składowych obrazu oraz wypuklenie mało widocznych elementów.

Wyróżniamy dwie główne grupy filtrów [3]:

a) liniowe – filtracja obrazu na podstawie liniowej kombinacji pikseli obrazu;
b) nieliniowe – filtracja obrazu na podstawie nieliniowej kombinacji pikseli obrazu;

W następnym podrozdziale opiszę działanie filtru medianowego, zaliczanego do jednych z podstawowych filtrów nieliniowych, który jest podstawą implementacji programu zrealizowanego w ramach niniejszej pracy.

Filtr medianowy

Jak przy każdym cyfrowym przetwarzaniu sygnałów, jednym z głównych problemów, z którymi należy się zmierzyć są zakłócenia wywołane konwersją sygnału analogowego na cyfrowy. Mamy wtedy do czynienia z zakłóceniami różnego rodzaju, wśród których znajduje się zakłócenie typu „sól i pieprz" (ang. *salt and pepper*), czyli zagubione wartości pikseli, odstające znacząco od pikseli znajdujących się w najbliższym sąsiedztwie. Jedną z możliwości usunięcia tych zakłóceń jest zastosowanie filtru medianowego, który jak sama nazwa wskazuje, opiera się na obliczeniu dla każdego piksela mediany wartości pixeli otaczających dany punkt – nazywanej maską. Rozmiar maski definiuje się, jako wymiary kwadratu otaczającego dany piksel, np. 3x3, 5x5. Przy czym im większa maska tym redukcja szumów będzie powodowała większe zniekształcenie obrazu w postaci jego rozmycia.

Na Rys.3 możemy zobaczyć działanie filtra medianowego, dla przykładu została zastosowana maska o wymiarach 3x3. Danymi wejściowymi jest 9 pikseli wchodzących w skład w najbliższe otoczenie badanego piksela. W następnym kroku zbiór wartości powyższych pikseli jest sortowany. Wynikiem działania filtru medianowego jest mediana ze zbioru wartości wyznaczonego w poprzednim kroku – w tym przykładzie jest to liczba 126.

124	126	127
120	150	125
115	119	133

Mediana z ciągu liczb: 115,119,120,124,126,125,127,133,150 to 126

Rys. 3 Przykład działania filtru medianowego

Główną zaletę tego filtru stanowi odporność na zakłócenia impulsowe, czyli występowanie w zbiorze pikseli wartości ekstremalnych, silnie różnych od pozostałych. Wynika to z własności funkcji mediana, która wybiera wartość środkową ze zbioru uszeregowanych liczb. W związku z powyższym granicznym rozmiarem zakłócenia, który może zostać usunięty przy pomocy filtra medianowego jest liczba zakłóconych pikseli: liczba pikseli zakłóconych nie może być większa niż połowa liczby pikseli w bloku. Jak łatwo zauważyć największym ograniczeniem tego typu filtru jest złożoność obliczeniowa, związana z koniecznością sortowania zbioru wartości pikseli dla każdego przetwarzanego bloku.

3.2.4 Operacje morfologiczne

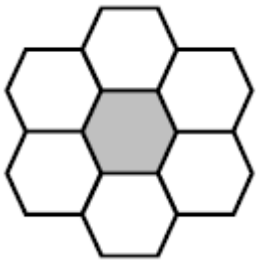
Przekształcenia morfologiczne są uznawane za jedne z najważniejszych operacji wykorzystywanych do analizy obrazu [3]. Opierają się o analizę każdego piksela na obrazie względem pikseli znajdujących się w sąsiedztwie. Kształt i rozmiar sąsiedztwa poddanych analizie definiowana jest przy pomocy elementu strukturalnego [4].

Ich największą zaletą jest modularność, która umożliwia dokonywanie bardzo złożonych operacji mających na celu między innymi: rozpoznanie kształtów, znalezienie położenia względem siebie, a nawet symulowania interakcji, jakie mogą zajść na danym zbiorze kształtów. Należy mieć na uwadze, że wachlarz możliwości, jakie oferują przekształcenia morfologiczne związany jest z dużą złożonością obliczeniową, co stanowi ich największą wadę. W kolejnej części przybliżę tematykę operacji morfologicznych umożliwiająca zapoznanie się z dokładnymi możliwościami i ograniczeniami ich stosowania.

Główną składową przekształceń morfologicznych jest element strukturalny, identyfikowany, jako cześć obrazu posiadająca punkt centralny. Podczas wykonywania operacji morfologicznych, element strukturalny służy do wyznaczenia fragmentu obrazu, na którym są one wykonywane.

.

Dobór kształtu elementu strukturalnego zależy od przetwarzanego obrazu, a w szczególności od poszukiwanych obiektów, pomimo tego literatura podaje [3], że najbardziej uniwersalnym kształtem jest ten oparty na kole o promieniu jednostkowym na siatce heksagonalnej – zobacz Rys. 4.

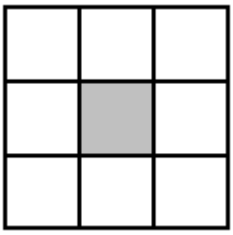


Rys. 4 Element strukturalny oparty o siatkę heksagonalną [3]

Do głównych zalet powyższego kształtu zalicza się:

- rezultat operacji morfologicznych jest bardziej spójny z intuicją;
- względnie mała złożoność obliczeniowa ze względów na 7 elementów;
- zbliżony do koła kształt, który jest bardzo często poddany analizie w znacznej ilości przekształceń morfologicznych;

Należy mieć na uwadze, że z punktu widzenia implementacji największym minusem takiej siatki jest jej „nienaturalny", jak na warunki programistyczne kształt. Dlatego też bardzo często zastępuję się optymalną siatkę heksagonalną siatką kwadratową – zobacz Rys. 5.



Rys. 5 Element strukturalny oparty o siatkę kwadratową [3]

Schemat działania przekształceń morfologicznych:

1. Sprawdzenie zależności pomiędzy elementem strukturalnym a każdym pojedynczym punktem obrazu.
2. Ustalenie, czy dany punkt oraz jego otoczenie spełnia warunki zadane w elemencie strukturalnym.
3. Dokonanie zadanych operacji na punktach, które spełniają zależność pomiędzy wzorcem pikseli obrazu i szablonu elementu strukturalnego.

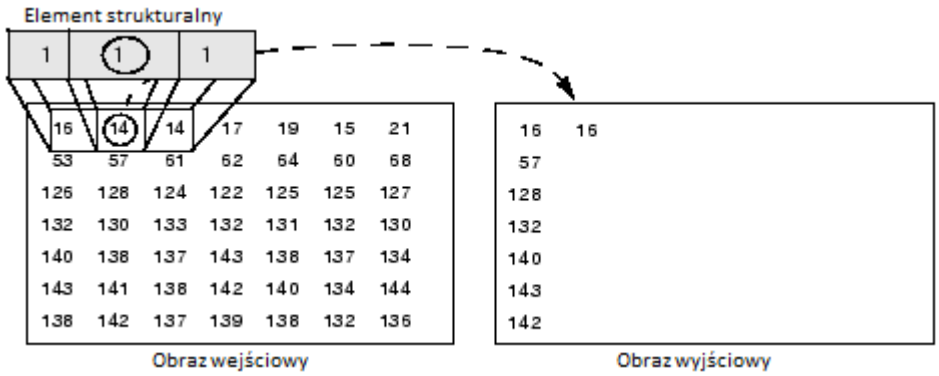
Literatura wyróżnia następujące operacje morfologiczne [3]:

- erozja;
- dylatacja;
- otwarcie;
- zamknięcie;

Najprostszymi z nich są erozja i dylatacja, które cechują się addytywnością, czyli trwałym przekształceniem obrazu. Złożeniem tych dwóch podstawowych operacji są operacje otwarcia i zamknięcia, które natomiast są niezmienne względem siebie. Zestawienie podstawowych typów operacji morfologicznych przedstawiono poniżej.

A. Erozja

Polega na ustawieniami na wyjściu wartości minimalnej dla każdego punktu obrazu należącego do zadanego elementu strukturalnego. Na Rys. 6 przedstawiono przykład przeprowadzenia operacji morfologicznej erozji. Obraz wejściowy jest obrazem w skali szarości, zastosowano element strukturalny o wymiarach 1x3. Dla każdego piksela wybierana jest wartość maksymalna ze zbioru wartości wejściowych piksela należących do bieżącej pozycji elementu strukturalnego. Dla niniejszego przykładu wartością wyjściową jest 16, która stanowi maksymalna wartość ze zbioru 3 liczb należących do pierwszego fragmentu obrazu wyznaczonego przez element strukturalny.



Rys. 6 Działanie operacji morfologicznej erozji [4]

Odpowiada to operacji wykonywanej przez filtr minimalny. Skutkiem ubocznym jest zmniejszenie pola powierzchni kształtu znajdującego się na przetwarzanym obrazie [4]

B. Dylatacja

Polega na ustawieniami na wyjściu wartości maksymalnej dla każdego punktu obrazu należącego do zadanego elementu strukturalnego. Odpowiada to operacji wykonywanej przez filtr maksymalny. Skutkiem ubocznym jest zwiększenie pola powierzchni przekształcanego kształtu znajdującego się na przetwarzanym obrazie.

C. Otwarcie

Złożenie operacji erozji oraz dylatacji. Jej celem jest usunięcie małych szczegółów z kształtu takich jak: półwyspy, wypustki.

D. Zamknięcie

Złożenie operacji dylatacji oraz erozji. Jej celem jest przyłączenie małych szczegółów do kształtu w szczególności wysp i otworów wewnątrz przetwarzanego kształtu.

E. Otwarcie i zamknięcie właściwe, automediana, detekcja ekstremów

Pozostałe przekształcenia stanowią złożenie przekształceń otwarcia i zamknięcia z obrazem pierwotnym (niepoddanym przekształceniom).

3.2.5 Momenty geometryczne obrazu

Narzędziami obróbki obrazu dostarczającymi dużą ilość informacji na temat figur płaskich są momenty geometryczne obrazu. Momenty są powszechnie wykorzystywane w różnych dziedzinach – od dziedzin nauk mechanicznych przez obszary związane ze statystyka, aż po przetwarzanie obrazów.

Definiowane są, jako skalarne wielkości, służące do opisu funkcji oraz w celu podkreślenia jej szczególnych cech. służąca do opisywania rozkładu przestrzennego zbioru punktów – zobacz [3]. Szczegółowość informacji na temat danej figury wzrasta wraz z przechodzeniem na wyższe rzędy momentów.

$$m_{ij} = \sum_x \sum_y \left(I(x,y) * x^j * y^i\right)$$

$$, \quad i,j - rzędy\ momentów,$$

$$I - obraz$$

Rys. 7 Momenty geometryczne - wzory

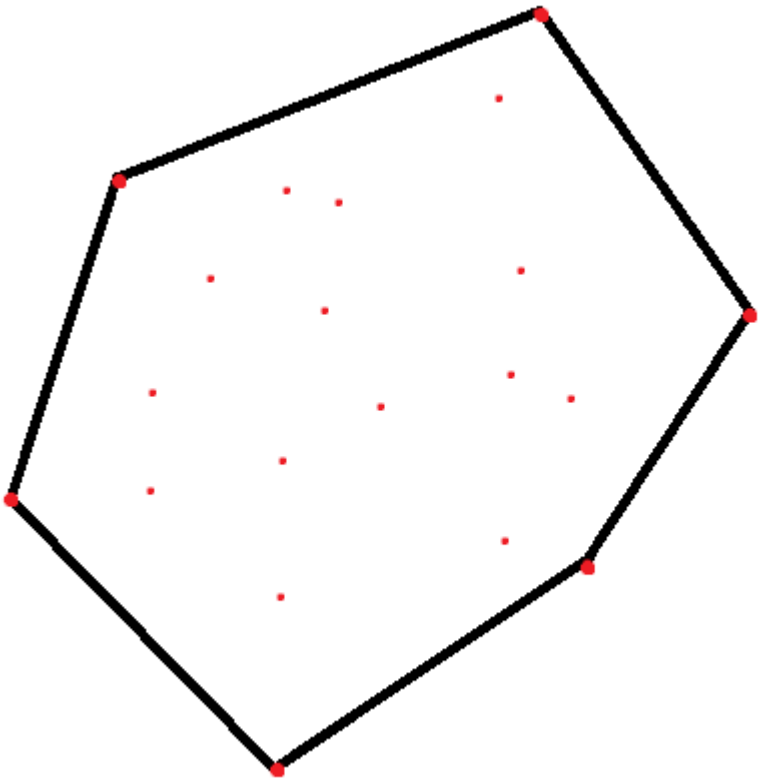
Momenty geometryczne niższych rzędów maja znaczenie intuicyjne - m_{00} stanowi „masa” obrazu, która w przypadku obrazów binarnych identyfikowane jest, jako powierzchnia obiektu. Natomiast posiadając informacje na temat dwóch pierwszych momentów geometrycznych możemy wyznaczyć środek masy(definiowany również jako centroid [3], bądź środek grawitacji [5]) zobacz .

$$x = \frac{m_{10}}{m_{00}}, \qquad y = \frac{m_{01}}{m_{00}}\ ,\ \ m - momenty\ centralne$$

Rys. 8 Wzór na środek masy na podstawie momentów geometrycznych

3.2.6 Algorytm otoczki wypukłej

Podstawowym problemem geometrii obliczeniowej jest znajdywanie otoczki wypukłej (z ang. Convex hull) zadanego zbioru punktów. Zgodnie z definicją [6], otoczka wypukła **P** jest to najmniejszy wielokąt wypukły taki, że każdy punkt ze zbioru **P** leży albo na brzegu wielokąta albo w jego wnętrzu – zobacz Rys. 9.



Rys. 9 Zbiór punktów wraz z punktami stanowiącymi otoczkę wypukłą

Obecnie istnieje kilka implementacji rozwiązania tego problemu: algorytm dziel i zwyciężaj, algorytm grahama, czy algorytm Jarvisa [7]. W bibliotece opencv udostępniona implementacja została oparta o algorytm Slansky'ego [8]

Terminem ściśle związanym z otoczką wypukłą są tzw. defekty [9], które stanowią wklęsłe punkty tworzące linię brzegową – w niniejszej pracy na podstawie ich analizy rozpoznawane są palce użytkownika.

3.2.7 Algorytm Douglas-Peuckera

Celem stosowania algorytmu Douglas-Peuckera jest uproszczenie opisu danej krzywej, przy pomocy mniejszej ilości punktów jej tworzących. Algorytm interpretuje wszystkie punkty reprezentujące krzywą i usuwa z niej te, których odległość od aproksymowanej krzywej jest większa niż zadana wartość.

W pierwszym kroku zostaje wybrany punkt początkowy i końcowy łamanej opisanej przy pomocy zbioru punktów. Następnie z pozostałego zbioru punktów znajdywany jest punkt posiadający największa wartość odległości prostopadłej do odcinka wyznaczonego na podstawie punktów z pierwszego kroku. Jeśli punkt najbardziej oddalony od krzywej ma wartość większa niż zadany epsilon dokonywany jest uproszczenie poprzez podział zbioru punktów i wyznaczenie dwóch podzbiorów punktów – dla każdego z nich wykonywane są ponownie wszystkie operacje od pierwszego kroku.

3.2.8 Algorytm znajdywania największego konturu

W większości przypadków przetwarzania obrazów konieczne jest znalezienie konturów obiektów będących przedmiotem danej analizy. W przypadku niniejszej pracy wykorzystano algorytm wykrywania krawędzi Border Following opisany przez Satoshi Suzuki [10]. Mając na uwadze fakt, iż w niniejszym projekcie mają być rozpoznawane gesty wykonywane jedną dłonią, dlatego wystarczy znaleźć największy konturu znajdujący się w aktualnie przetwarzanej ramce obrazu.

Posiadając informację na temat konturu możemy wyznaczyć środki okręgów opartych o krawędź konturu, które wraz z punktem centrum masy stanowią podstawę do dalszej analizy konturu.

3.2.9 Algorytm znajdywania środka maksymalnego okręgu wpisanego w kontur

Wyznaczenie środka okręgu o maksymalnym promieniu wpisanego w kontur polega na iteracyjnym przejściu po każdym pikselu i sprawdzeniu, czy należy on do zadanego, jako zbiór punktów wyznaczających krawędź konturu obszaru. W każdym kolejnym kroku dokonywane jest sprawdzenie, czy odległość pomiędzy danym punkcie a najbliższą krawędzią jest maksymalna.

Jak można zauważyć algorytm jest mało wydajny, ale z punktu widzenia dalszej analizy stanowi kluczową rolę.

3.2.10 Algorytm znajdywania środka minimalnego okręgu opisanego na konturze

Ostatnim bardzo ważnym punktem znajdującym się wewnątrz konturu jest środek okręgu o minimalnym promieniu opisanym na badanym konturze. Jest on wyznaczany przy pomocy metody z biblioteki OpenCV minEnclosingCircle, która implementuje algorytm zapropowany w pracy Emo Welzl, *Smallest enclosing disks (balls and ellipsoids)* [11]. Istotą działania powyższego algorytmu jest przejście przez punkty środkowe odcinków wyznaczonych przez kolejne pary punktów należące do zbioru wyznaczającego dany kontur. W przypadku, gdy większość punktów zawiera się na okręgu wyznaczonym w tymczasowym punkcie środowym okręgu działanie algorytmu zostaję przerwane w celu zwrócenia wyniku. W innym przypadku, algorytm wskazuje kolejne punkty potencjalnie będące punktem środkowym okręgu opartego na konturze, między innymi punkty przecięcia prostopadłych do wyznaczonych wcześniej odcinków. Powyższy algorytm zaliczany jest do algorytmów liniowych *O(n)* zarówno pod względem złożoności czasowej jak i obliczeniowej.

Rozdział 3. w sposób wyczerpujący opisuje zagadnienia teoretyczne związane z implementacja niniejszego projektu. W kolejnym rozdziale zostaną przedstawione szczegóły na temat zrealizowanego w ramach niniejszej pracy magisterskiej programu.

4. Część Praktyczna

W tej części zostaną opisane rezultaty implementacji programu, który realizuje założenia postawione przed tematem niniejszej pracy.

W pierwszym podrozdziale zostaną opisane wymagania wdrożeniowe oraz interfejs graficzny.

4.1 Program

Aplikacja umożliwia sterowanie kursorem myszki przy pomocy analizy gestów wykonywanych dłonią.

4.1.1 Wymagania projektowe

Program jest skierowany do użytkowników systemów opartych o architekturę Unix. Wymaga posiadania kamery podłączonej do laptopa oraz środowiska programistycznego *C++* wraz z bibliotekami: opencv oraz suinput. Użytkownik przy pomocy ruchów dłonią ma możliwość sterowania kursorem myszki.

4.1.2 Wymagania wdrożeniowe

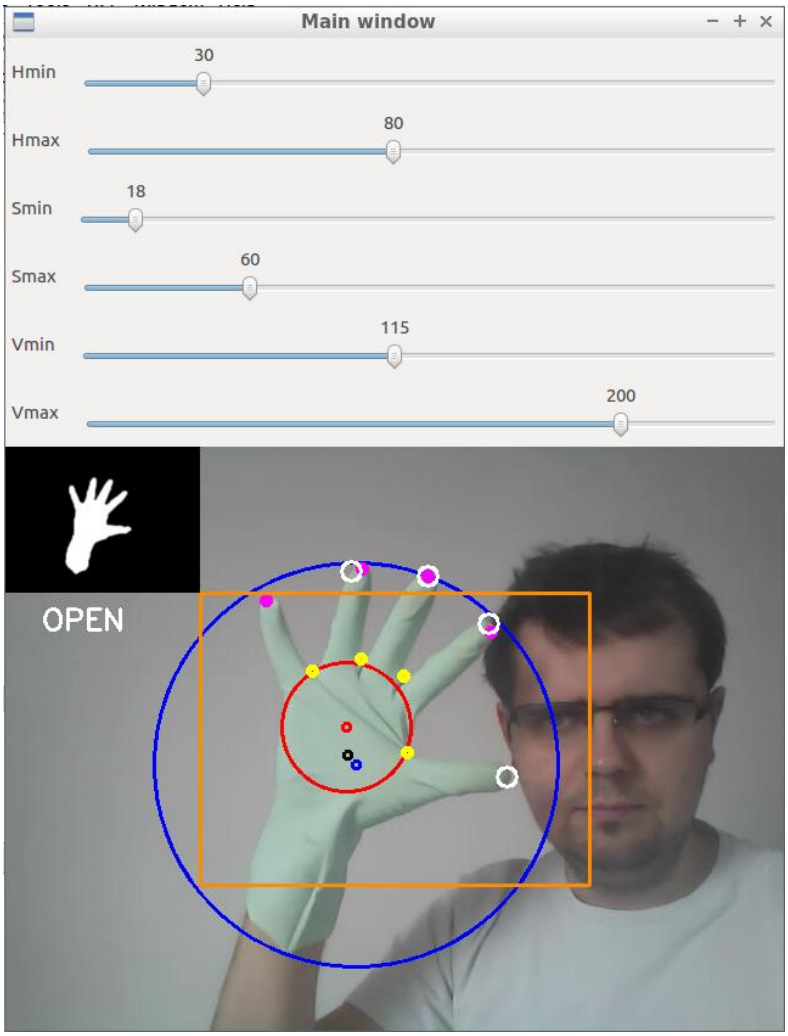
Podstawową kwestią jest posiadanie komputera wyposażonego w kamerę wideo, dzięki której będą rejestrowane gesty wykonywane przez użytkownika. Do wymagań wdrożeniowych odnośnie wspieranego oprogramowania zaliczamy: system operacyjny *Unix*, środowisko programistyczne *C++*, biblioteki O*penCV* oraz *suinput*. Ostatnim wymaganiem, które jest opcjonalne, ale zalecane ze względu na zwiększenie precyzji rozpoznania gestów jest wykorzystywanie przez użytkownika rękawiczki w kolorze znacznie różnym od otoczenia.

Tab. 1 Wymagania wdrożeniowe aplikacji

WYMAGANIA WDROŻENIOWE	
Sprzęt	Komputer oraz kamera wideo
System operacyjny	*Unix*
Środowisko programistyczne	C++
Biblioteki	*OpenCV* oraz s*uinput*
Akcesoria	Rękawiczka w kolorze znacznie różnym od otoczenia

4.1.3 Interfejs graficzny

Okno programu składa się z podglądu widoku kamery oraz ustawień – zobacz Rys. 10 Interfejs graficzny programu Rys. 10. Okno ustawień umożliwia dobranie przedziałów wartości składowych przestrzeni kolorów HSV wykorzystywanych do binaryzacji, czyli dokonania konwersji na obraz czarno biały.



Rys. 10 Interfejs graficzny programu

Powyższe parametry mają kluczowe znaczenie w kontekście działania całego programu, ponieważ tak uzyskany obraz stanowi podstawę do dalszej obróbki oraz analizy wykonywanych przez użytkownika gestów. Dlatego należy mieć na uwadze, że wartości tych parametrów zależą od wielu czynników: jakości używanego sprzętu do nagrywania wideo, kompozycji elementów widocznych w oku kamery, czy rodzaju oświetlenia oraz jego intensywności. W związku z powyższym nieuniknionym jest dokonywanie aktualizacji wspomnianych parametrów.

W widoku kamery użytkownik ma możliwość zobaczenia podglądu wykonywanych ruchów uzupełniony o informacje zgromadzone w trakcie analizy przetwarzanego obrazu. W lewym górnym rogu znajduje się rezultat binaryzacji obrazu po wykonaniu operacji morfologicznych oraz zastosowaniu filtru medianowego wraz z nazwaniem rozpoznanego gestu.

Na centralnym obszarze znajduje się podgląd widoku kamery z naniesionymi następującymi informacjami:

a) Kolor pomarańczowy – obszar rozpoznawania gestów,
b) Kolor czerwony – okrąg wpisany w kontur,
c) Kolor niebieski – okrąg opisany na konturze,
d) Kolor czarny – centroid konturu,
e) Kolor biały – punkt początkowy defektu,
f) Kolor różowy – punkt końcowy defektu,
g) Kolor żółty – punkt najdalszy defektu.

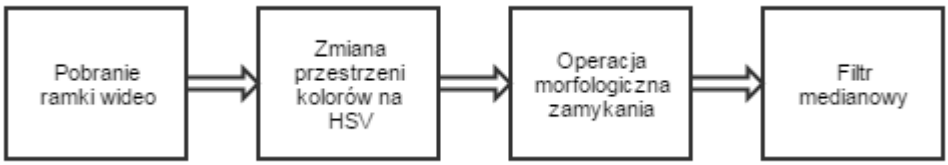
W kolejnym podrozdziale zostaną przedstawione poszczególne fazy rozpoznawania gestów wykonywanych przy pomocy dłoni.

4.2 Etapy rozpoznawania dłoni

Poniżej przedstawione zostały etapy rozpoznawania gestów wykonywanych przez użytkownika przy pomocy dłoni:

1. Pobranie ramki wideo
2. Zmiana przestrzeni kolorów na *HSV*
3. Operacja morfologiczna zamykania
4. Filtr medianowy
5. Znalezienie największego konturu
6. Aproksymacja Douglas-Peucker konturu
7. Znalezienie punktów centralnych
8. Algorytm otoczki wypukłej
9. Rozpoznawanie palców
10. Rozpoznawanie układów dłoni
11. Poruszanie kursorem myszki

Pierwszym etapem procesu rozpoznawania gestów jest modyfikacja danej ramki obrazu w sposób ułatwiający rozpoznanie cech poszczególnych dłoni. Mają one na celu wyostrzenie obrazu oraz wygładzenie krawędzi konturu – zobacz Rys. 11.

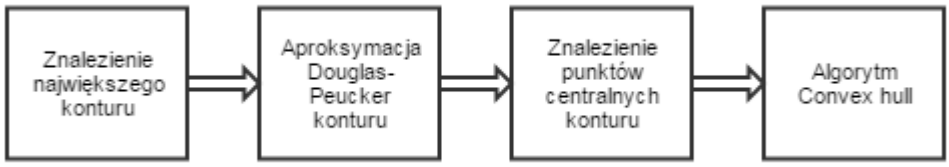


Rys. 11 Etapy korygujące ramkę wideo

Zaliczamy do nich:

1. Zmiana przestrzeni kolorów z RGB na HSV.
2. Operacja morfologiczna zamykania.
3. Filtr medianowy.

Na podstawie tak przetworzonej ramki obrazu wideo dokonuje się wyznaczenia grupy parametrów stanowiących cechy szczególne konturu dłoni – zobacz Rys. 12.



Rys. 12 Etapy wyznaczające parametry przetwarzanego obrazu

Zaliczamy do nich:

1. Znalezienie największego konturu – zobacz 3.2.8.
2. Aproksymacja Douglas-Peucker konturu – zobacz 3.2.7.
3. Znalezienie punktów środkowych konturu (środki koła opisanego i wpisanego w kontur oraz środka masy) – zobacz 3.2.5, 3.2.9 i 3.2.10.
4. Algorytm otoczki wypukłej – zobacz 3.2.6.

Ostatnie czynności wykonane przez program mają na celu interpretację zbioru parametrów otrzymanych podczas dotychczasowego przetwarzania oraz podjęcia akcji przesuwania kursora myszki – zobacz Rys. 13.

Rys. 13 Etapy wykonujące konkretne polecenia

Zaliczamy do nich:

1. Rozpoznanie palców.
2. Rozpoznanie układu dłoni.
3. Poruszanie kursorem myszki.

4.2.1 Pobranie ramki wideo

Pierwszym krokiem jest pobranie ramki wideo, z obrazu przetwarzanego przez kamerę – zobacz Rys. 14.

Rys. 14 Pobrana ramka wideo

4.2.2 Zmiana przestrzeni kolorów

Kolejnym krokiem jest przejście z przestrzeni kolorów RGB na HSV, opisującej w bardziej usystematyzowany sposób kolory występujące w rzeczywistości – zobacz 3.2.1. Dzięki takiemu podejściu łatwiejsze jest zidentyfikowanie m.in.: cieni związanych z brakiem właściwego oświetlenia.

Wykorzystano funkcję pakietu OpenCV cvtColor z argumentem code równym CV_BGR2HSV [12] – zobacz Rys. 15 oraz Rys. 16.

```
cvtColor(imgInput, imgOutput, CV_BGR2HSV);
```

Rys. 15 Wywołanie funkcji cvtColor

Rys. 16 Obraz wyjściowy funkcji cvtColor

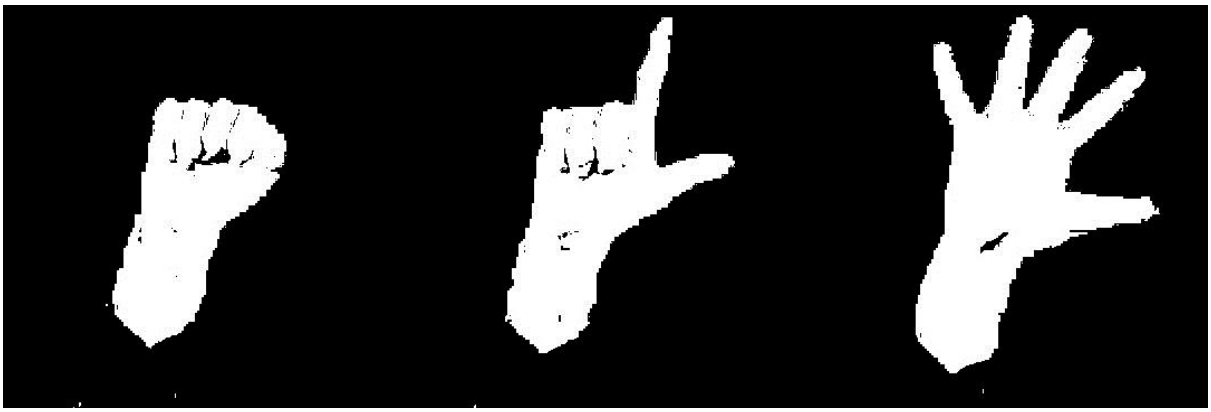
4.2.3 Progowanie obrazu

W celu identyfikacji dłoni z ramki wideo, dokonywana jest binaryzacja względem zadanego przedziału wartości HSV – zobacz 3.2.2. Aby binaryzacja jak najpełniej dokonała odróżnienia dłoni od otoczenia, użytkownik programu powinien skalibrować przedziały wartości HSV. W celu zapewnienia zwiększonej skuteczności rozpoznania dłoni oraz gestów przez nią wykonanych zalecane jest użycie rękawiczki w kolorze znacznie odróżniającym się od reszty przedmiotów znajdujących się w kadrze.

Wykorzystano funkcję pakietu OpenCV inRange z tablicami wartości minimalnych i maksymalnych pobranych z panelu ustawień – zobacz Rys. 17 oraz Rys. 18.

```
inRange(imgInput, bounds[0], bounds[1], imgOutput);
```

Rys. 17 Wywołanie funkcji inRange



Rys. 18 Obraz wyjściowy funkcji inRange

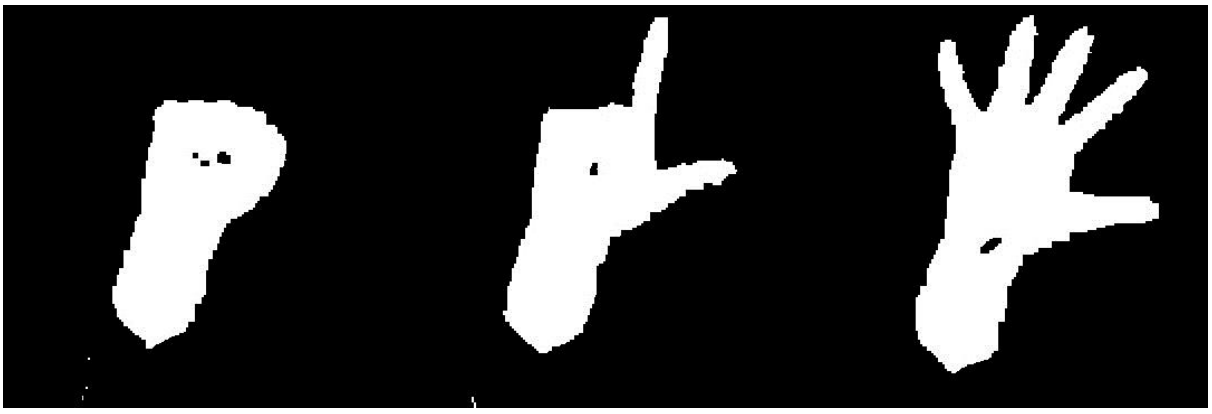
4.2.4 Usunięcie szumów

W celu wyeliminowania szumów, powstałych w wyniku procesu binaryzacji, w programie wykorzystano operację morfologiczną zamknięcia i operację morfologiczną otwarcia oraz filtr medianowy – zobacz 3.2.3 oraz 3.2.4.

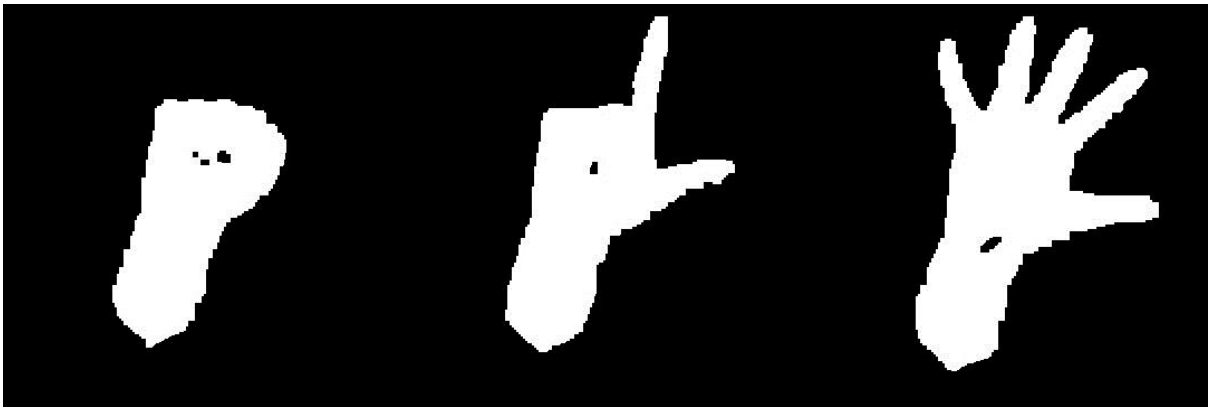
Wykorzystano funkcję pakietu OpenCV morphologyEx, w trybie MORPH_CLOSE i MORPH_OPEN, z elementem strukturalnym o wymiarach 5x5 [14] – zobacz Rys. 19, Rys. 20 oraz Rys. 21.

```
Mat element = getStructuringElement( MORPH_RECT, Size( 5,5 ), Point( -1, -1 ) );
morphologyEx(imgInput,imgOutput,MORPH_CLOSE,element,Point(-1,-1),1);
morphologyEx(imgCtx->output,imgCtx->output,MORPH_OPEN,element,Point(-1,-1),1);
```

Rys. 19 Wywołanie funkcji morphologyEx



Rys. 20 Obraz wyjściowy funkcji morphologyEx w opcji MORPH_CLOSE

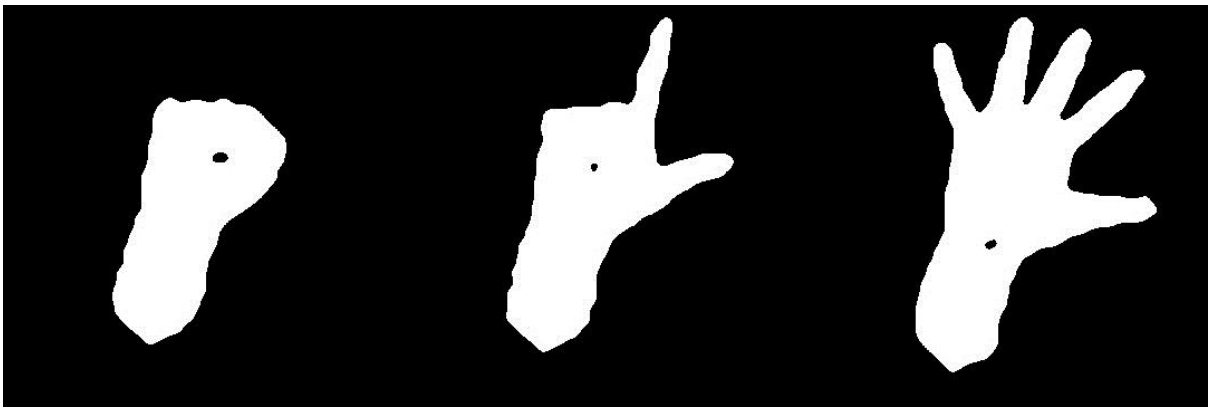


Rys. 21 Obraz wyjściowy funkcji morphologyEx w opcji MORPH_OPEN

Wykorzystano funkcję pakietu OpenCV medianBlur z parametrem ksize wynoszącym 9 [15] – zobacz Rys. 22 oraz Rys. 23.

```
medianBlur(imgInput,imgOutput,9);
```

Rys. 22 Wywołanie funkcji medianBlur



Rys. 23 Obraz wyjściowy funkcji medianBlur

4.2.5 Znalezienie konturu dłoni

Kolejnym krokiem jest wskazanie konturów kształtów znajdujących się na przetwarzanym obrazie – oznacza to w praktyce znalezienie białych konturów kształtów na czarnym tle.

Wykorzystano funkcję pakietu OpenCv findContours [16] w trybie pobrania konturów CV_RETR_TREE oraz przy pomocy przybliżania konturów w trybie CV_CLOCKWISE. Mając na uwadze fakt, że metoda findContours zwraca kontury wszystkich obiektów, a celem programu jest identyfikacja dłoni, dokonano filtracji kształtów posiadających małe powierzchnie (mniejsze niż 1000 pikseli) – zobacz Rys. 24 oraz Rys. 25.

```
findContours(imgInput, contours, hierarchy, CV_RETR_TREE, CV_CLOCKWISE, Point(0, 0));
```

Rys. 24 Wywołanie funkcji findContours

Rys. 25 Obraz wyjściowy funkcji findContours

4.2.6 Optymalizacja liczby punktów wyznaczających kontur dłoni

Znajdowanie konturów przy pomocy metody findContours [16] jest bardzo precyzyjne, a zarazem uciążliwe do dalszego procesowania. W przypadku rozpoznawania dłoni funkcja zwraca około 3000 punktów, których dalsze procesowanie mogłoby spowodować spowolnienie

pracy całego programu. Dlatego konieczne jest dokonanie selekcji i wybranie najbardziej znaczącego zbioru punktów opisujących kształt dłoni.

Wykorzystano funkcje pakietu OpenCV approxPolyPD [17], z parametrem *epsilon* o wartości 2, oraz parametrem closed ustawionym na true. Powyższa metoda dokonuje redukcji liczby punktów o około 100 razy, zachowujący przy tym odpowiedni balans pomiędzy ilością punktów, a dokładnością opisu – zobacz Rys. 26 oraz Rys. 27.

```
approxPolyDP(contourInput, contourOutput, 2, true);
```

Rys. 26 Wywołanie funkcji approxPolyDP

Rys. 27 Obraz wyjściowy funkcji approxPolyDP

4.2.7 Punkty środkowe dłoni

Kolejnym krokiem jest wyznaczanie punktów środkowych dłoni: środek masy (centroidu), środka okręgu o minimalnym promieniu opisanego na zbiorze punktów opisujących kontur dłoni oraz środek koła o maksymalnym promieniu wpisanym w zbiór punktów opisujących kontur dłoni – zobacz 3.2.9,3.2.10 oraz 3.2.5.

A. Środek masy

Wykorzystano funkcję pakietu OpenCV moments, a następnie dokonano konwersji tak otrzymanych momentów na współrzędne punktu centrum grawitacji.

```
moment = moments(imgCtx->biggestContour, false);
int xCenter = (int) round(moment.m10 / moment.m00);
int yCenter = (int) round(moment.m01 / moment.m00);
```

Rys. 28 Wywołanie funkcji moments oraz obliczenie punktów grawitacji konturu

Rys. 29 Obraz wyjściowy z zaznaczonym punktem grawitacji

B. Środek koła o minimalnym promieniu wpisanym w kontur

Wykorzystano funkcję pakietu OpenCV pointPolygonTest [18], która dla każdego piksela dokonuje sprawdzenia, czy dany punkt należy do konturu. Wartością zwrotną jest odległość pomiędzy punktem, a najbliższym punktem należącym do krawędzi konturu.

```
double distance = pointPolygonTest(polygonCurve, Point(i, j), true);
```

Rys. 30 Wywołanie funkcji pointPolygonTest

Rys. 31 Obraz wyjściowy z zaznaczonym maksymalnym okręgiem wpisanym w kontur dłoni

C. Środek koła o maksymalnym promieniu opisanym na konturze

Wykorzystano funkcję pakietu OpenCV minEnclosingCircle [19], która dla zadanych punktów wyznacza środek oraz promieniu okręgu okalającego punkty wejściowe.

```
minEnclosingCircle(inputContour, outputCenter, outputRadius);
```

Rys. 32 Wywołanie funkcji minEnclosingCircle

Rys. 33 Obraz wyjściowy z zaznaczonym minimalnym okręgiem oparty o punkty konturu

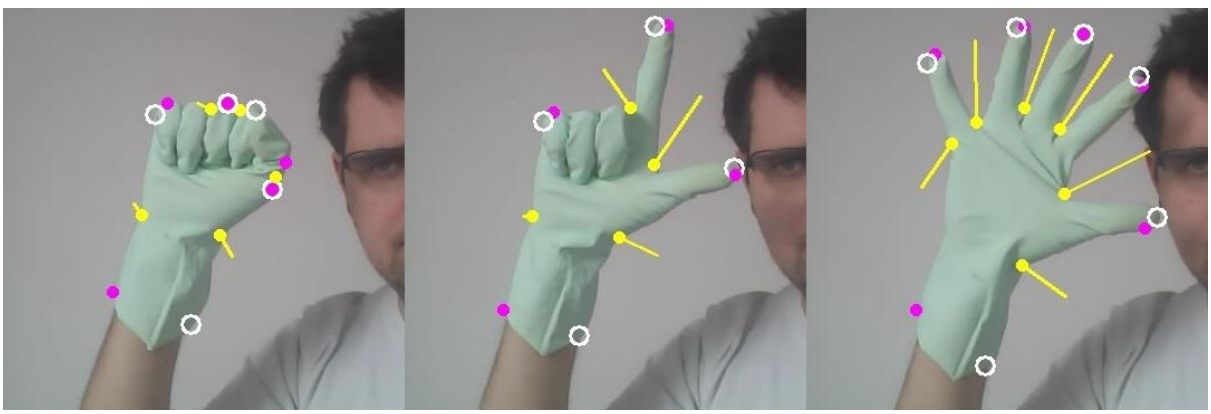
4.2.8 Algorytm otoczki wypukłej oraz jego defekty

Kluczowym etapem jest wykorzystanie algorytmu otoczki wypukłej (ang. *convex hull*), który znajduje minimalny wielokąt zawierający dany zbiór punktów – opisane w rozdziale 3.2.6. Z perspektywy rozpoznawania dłoni, najbardziej istotnym elementem jest znalezienie tzw. defektów, czyli obszarów nienależących do potencjalnego konturu dłoni, ale zawierających się w wielokącie wyznaczonym, jako rezultat algorytmu convex hull.

Wykorzystano funkcję pakietu OpenCv convexHull [20] oraz convexityDefects [21] Pierwsza z nich służy do wyznaczenia punktów stanowiących o istotnych zmianach w kształcie konturu. Natomiast convexityDefects umożliwia otrzymanie informacji na temat defektów, czyli parametrów zagłębień znajdujących się na konturze – punkt początkowy, końcowy, najdalszy oraz głębokość defektu.

```
convexHull(Mat(inputContour), outputHullInt, false);
convexityDefects(inputContour, outputHullInt, outputDefects);
```

Rys. 34 Wywołanie funkcji convexHull oraz convexityDefects



Rys. 35 Obraz wyjściowy z zaznaczonymi defektami algorytmu convex hull

4.2.9 Identyfikacja palców dłoni

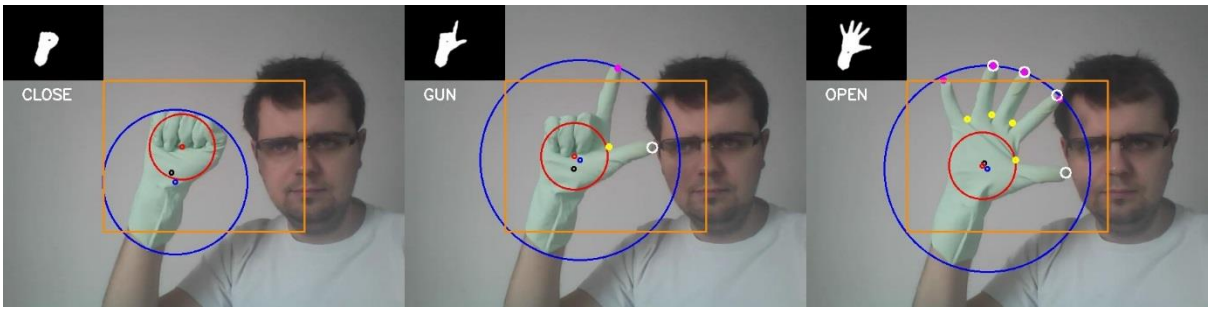
Posiadając informacje uzyskane w wyniku przetwarzania obrazu przy pomocy metody convexHull, przeprowadzana jest analiza konturu pod względem znalezienia kształtu odpowiadającego dłoni – zobacz Załącznik 2. Poniżej przedstawiono warunki, jakie dany kontur musi spełnić, aby został zbiór parametrów go opisujących został zakwalifikowany, jako palec – zostały one wyznaczone na podstawie prób i błędów.

1. Głębokość defektu mniejsza niż promień koła opisanego na konturze dłoni, oraz większa niż promień koła wpisanego w kontur dłoni.
2. Odległość pomiędzy początkiem defektu, a środkiem koła wpisanego w kontur powinna być większa niż 55% oraz mniejsza niż 120% długości promienia koła opisanego na konturze.
3. Kąt pomiędzy punktami wyznaczającymi defekt, tj. początkowym, a końcowym w punkcie najdalszym, powinien być mniejszy niż 90^o.
4. Kąt pomiędzy punktem początkowym defektu a punktem wyznaczającym środek koła opisanego na konturze dłoni powinien być większy niż 200^o.

4.2.10 Rozpoznawanie gestów

W niniejszej pracy zdefiniowano następujące gesty:

a) OPEN, otwarta dłoń - wszystkie palce wyprostowane dłoń frontalnie do kamer.
b) CLOSE, zamknięta dłoń - wszystkie palce zgięte.
c) GUN, pistolet - wszystkie palce zgięte oprócz kciuka i palca wskazującego.



Rys. 36 Obraz końcowy przetwarzania obrazu

Rozpoznanie wykonanego gestu odbywa się na podstawie analizy parametrów uzyskanych w wyniku wcześniejszego procesowania kształtów znajdujących się na obrazie -

ilości znalezionych defektów posiadających cechy palca oraz odległości pomiędzy środkiem grawitacji a środkiem koła opisanego na konturze – zobacz Załącznik 1.

Aby kontur został rozpoznany, jako kontur OPEN liczba defektów spełniających warunki przedstawione w punkcie 4.2.9 Powinna wynosić 3 lub 4, oraz dodatkowo odległość pomiędzy środkiem grawitacji konturu a środkiem koła opisanego na konturze powinna wynosić mniej niż 20.

Tab. 2 Zestawienie gestów oraz cech charakterystycznych

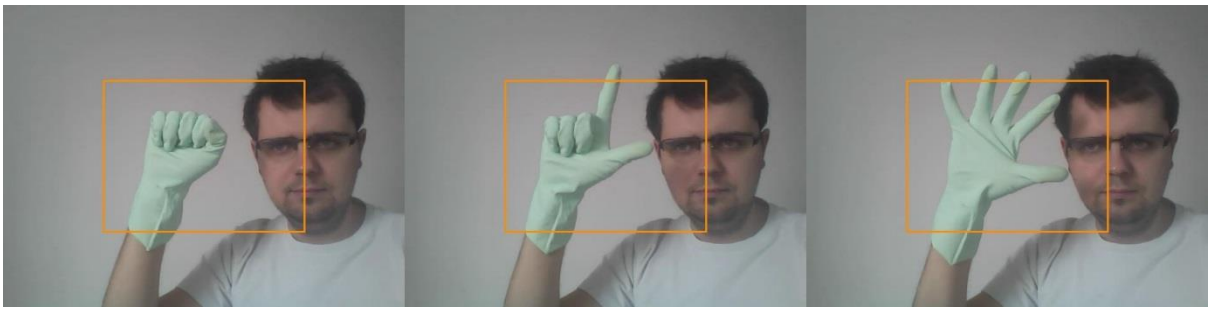
Gest	Liczba defektów	Odległość pomiędzy środkiem grawitacji a środkiem koła opisanego na konturze
OPEN	3 lub 4	Mniejsza niż 20
CLOSE	0	Mniejsza niż 60
GUN	1 lub 2	Mniejsza niż 45

W przypadku gestu CLOSE, czyli zamknięta dłoń, liczba defektów powinna wynosić 0, a odległość między punktami środkowymi powinna wynosić mniej niż 60. Natomiast, aby dany kontur został rozpoznany, jako gest GUN, liczba defektów powinna wynosić 1 lub 2, a odległość między punktami środkowymi nie powinna przekraczać 45.

4.2.11 Poruszanie myszką

Ostatnim elementem programu jest zamiana informacji zgromadzonych w poprzednich krokach na temat konturów, na akcje wykonywane przy pomocy kursora myszki. W celu poprawnego interpretowania ruchów dłoni zastosowano kilka warunków, jakie muszą zostać spełnione.

Pierwszym z nich jest fakt, iż ruchy wykonywane przy pomocy dłoni interpretowane są tylko w pewnym prostokątnym obszarze – zobacz Rys. 37. Jest on identyfikowany, jako prostokąt posiadający punkt centralny w punkcie centralnym przetwarzanego obrazu, a którego długość boków stanowi połowę długości boków wymiarów analizowanej ramki obrazu. Wynika to z faktu, iż rozpoznawanie kształtów jest najbardziej skuteczne w centralnej części obrazu.



Rys. 37 Ramka obrazu wideo z zaznaczonym prostokątnym aktywnym obszarem

Kolejnym aspektem, z którym należy się zmierzyć są problemy związane z tak zwanymi drganiami – nieodłącznym elementem składowym analizy zjawisk przy pomocy układów elektronicznych. W przypadku niniejszej pracy drganiami możemy nazwać wszystkie błędnie rozpoznane pojedyncze gesty – najczęściej, jako stany przejściowe pomiędzy jednym gestem a drugim. W celu zniwelowania tego typu zjawisk, dany gest musi zostać rozpoznany co najmniej 3 razy – oznaczać to będzie, że użytkownik faktycznie pokazuje dany gest.

Podstawowymi akcjami realizowanymi przez programu są: wykonanie symulacji operacji kliknięcia lewego oraz prawego przycisku myszki. Aby dokonać kliknięcia lewego/prawego przycisku myszy użytkownik powinien wykonać gest OPEN/CLOSE wewnątrz prostokąta opisanego w poprzednim akapicie. Wykonanie akcji naciśnięcia przycisku prawego/lewego myszki zostało zrealizowane przy funkcji suinput_emit_click z biblioteki suinput – zobacz Rys. 38.

```
suinput_emit_click(uinput_fd,BTN_LEFT/BTN_RIGHT);
```

Rys. 38 Wywołanie funkcji suinput_emit_click

O wiele bardziej złożoną operacją jest wykonanie przesunięcia kursora myszki. Poruszanie kursorem myszki zostało zrealizowane na zasadzie joysticka – analizowane są wychylenia punktu środkowego dłoni względem punktu centralnego obrazu. Posiadając informację na temat wektora przemieszczenia punktu środkowego, w szczególności kąt oraz jego moduł wykonywane są metody przemieszczające wskaźnik myszki na monitorze. W celu przesunięcia kursora myszki wykonuje się operacje przesunięcia myszki przy pomocy funkcji suinput_emit z pakietu suinput – zobacz Rys. 39.

```
suinput_emit(uinput_fd, EV_REL, REL_X, x);
suinput_emit(uinput_fd, EV_REL, REL_Y, y);
```

Rys. 39 Wywołanie funkcji suinput_emit

5. Podsumowanie

W ramach niniejszej pracy magisterskiej została zaprojektowana oraz zaimplementowana alternatywna metoda poruszania kursorem z wykorzystaniem rozpoznawania ruchów dłoni przy pomocy kamery, w formie aplikacji desktopowej.

Możliwe jest korzystanie z niej przy pomocy zwykłego laptopa z wbudowaną kamerą, posiadającego system Unix, wraz zainstalowanymi bibliotekami opencv oraz suinput.

Można wyróżnić następujące ścieżki rozwoju niniejszego systemu:

- rozpoznawanie większej ilości gestów,
- rozpoznawanie gestów wykonywanych przez dwie ręce,
- wykorzystanie sztucznej inteligencji w procesie rozpoznawania gestów,
- pełniejsza integracja z systemem Unix – działanie w tle, notyfikacje.

6. Bibliografia

[1] „Computing history - Computer mouse,” [Online]. Available: http://www.computinghistory.org.uk/det/613/the-history-of-the-computer-mouse/. [Data uzyskania dostępu: 18 06 2017].

[2] „Jak widzi człowiek,” [Online]. Available: http://analizaobrazu.x25.pl/articles/5. [Data uzyskania dostępu: 18 06 2017].

[3] Komputerowa analiza i przetwarzanie obrazów, Kraków: Wydawnictwo Fundacji Postępu Telekomunikacji, 1997.

[4] „Mathworks - Morphological dialtion and erosion,” [Online]. Available: http://www.mathworks.com/help/images/morphological-dilation-and-erosion.html. [Data uzyskania dostępu: 11 06 2017].

[5] J. Flusser, T. Suk i B. Zitova, Moments and Moment Invariants in Pattern Recognition, Chippenham: John Wiley & Sons Ltd, 2009.

[6] M. d. Berg, Geometria obliczeniowa. Algorytmy i zastosowania, Warszawa: Wydawnictwo Naukowo-Techniczne, 2007.

[7] „Otoczka wypukła - Wrocławski portal informatyczny,” [Online]. Available: http://informatyka.wroc.pl/node/910. [Data uzyskania dostępu: 11 06 2017].

[8] "ConvexHull Documentation," 14 Maj 2017. [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=convexhull#convexhull.

[9] „ConvexDef,” [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=convexhull#convexitydefects. [Data uzyskania dostępu: 11 06 2017].

[10] S. Suzuki, „Topological Structural Analysis of Digitized Binary Images by Border Following,” *Computer Vision, Graphics and Image Processing,* pp. 32-46, 1983.

[11] E. Welzl, „Smallest enclosing disks (balls and ellipsoids),” *New Results and New Trends in Computer Science,* 1991.

[12] „Dokumentacja Opencv - metoda cvtColor,” [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html#cvtcolor. [Data uzyskania dostępu: 11 06 2017].

[13] „Dokumentacja Opencv - metoda inRange,” [Online]. Available: http://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html#inrange.

[14] „Dokumentacja Opencv - metoda getStructuringElement,” [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#getstructuringelement. [Data uzyskania dostępu: 11 06 2017].

[15] „Dokumentacja Opencv - metoda medianBlur,” [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=medianblur#medianblur.

[16] „Dokumentacja OpenCV - metoda findContours,” [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=findcontours#findcontours. [Data uzyskania dostępu: 11 06 2017].

[17] „Dokumentacja OpenCV - metoda approxPolyDP,” [Online]. Available: http://docs.opencv.org/2.4.13.2/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#approxpolydp. [Data uzyskania dostępu: 11 06 2017].

[18] „Dokumentacja OpenCV - metoda pointPolygonTest,” [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=pointpolygontest#pointpolygontest. [Data uzyskania dostępu: 11 06 2017].

[19] „Dokumentacja OpenCV - metoda minEnclosingCircle,” [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=minenclosingcircle#minenclosingcircle. [Data uzyskania dostępu: 11 06 2017].

[20] „Dokumentacja OpenCV - metoda convexHull,” [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=convexhull#convexhull. [Data uzyskania dostępu: 11 06 2017].

[21] „Dokumentacja OpenCV - metoda convexityDefects,” [Online]. Available: http://docs.opencv.org/2.4.13.2/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#convexitydefects. [Data uzyskania dostępu: 11 06 2017].

[22] „Algorytmy graficzne - metody binaryzacji obrazu,” [Online]. Available: http://www.mif.pg.gda.pl/homepages/marcin/Wyklad3.pdf. [Data uzyskania dostępu: 11 06 2017].

7. Spis Rysunków

8. Załączniki

Załącznik 1 Kod źródłowy processor/DetectFingersProcessor.cpp

```
#include "headers/DetectFingersProcessor.h"

void DetectFingersProcessor::process(ImageContext *imgCtx) {
    imgCtx->fingers.clear();
    for (Vec4i defect : imgCtx->defects) {
        Point ptStart = imgCtx->biggestContour[defect[0]];
        Point ptEnd = imgCtx->biggestContour[defect[1]];
        Point ptFar = imgCtx->biggestContour[defect[2]];
        double depth = defect[3] / 256;

        bool notValidDepth = isNotValidDepth(imgCtx, depth);
        bool notValidDistanceToCOG = isNotValidDistanceToCircleCenter(imgCtx,
ptStart, imgCtx->maxInscribedCircleCenter);
        bool notValidAngle = isNotValidAngle(imgCtx, ptStart, ptEnd, ptFar);
        bool notValidAngleToCOG = isNotValidAngleToCOG(imgCtx, ptStart);
        if (notValidDepth ||
            notValidDistanceToCOG ||
            notValidAngle ||
            notValidAngleToCOG)
            continue;
        imgCtx->fingers.push_back(defect);
    }
};

bool DetectFingersProcessor::isNotValidAngle(ImageContext *imgCtx, Point start,
Point far, Point end) {
    return angleBetween(start, far, end) >= imgCtx->MAX_FINGER_ANGLE;
}


bool DetectFingersProcessor::isNotValidDistanceToCircleCenter(ImageContext
*imgCtx, Point ptStart, Point circleCenter) {
    double distance = norm(ptStart - circleCenter);
    return distance < imgCtx->MIN_FINGER_DISTANCE || distance > imgCtx-
>MAX_FINGER_DISTANCE;
}

bool DetectFingersProcessor::isNotValidDepth(ImageContext *imgCtx, double depth)
{
    return depth < imgCtx->MIN_FINGER_DEPTH || depth > imgCtx->MAX_FINGER_DEPTH;
}

bool DetectFingersProcessor::isNotValidAngleToCOG(ImageContext *imgCtx, Point
ptStart) {
    return !(angleToCOG(ptStart, imgCtx->minEnclosingCircleCenter) /*,
contourAxisAngle)*/ <= MAX_INDEX);
}

int DetectFingersProcessor::angleBetween(Point start, Point far, Point end) {
    return
            abs((radiansToDegrees(atan2(start.y - far.y, start.x - far.x) -
                                  atan2(end.y - far.y, end.x - far.x))));
}

int DetectFingersProcessor::angleToCOG(Point tipPt, Point cogPt) {
    int angleTip = (int) round(radiansToDegrees(atan2(tipPt.x - cogPt.x, cogPt.y
- tipPt.y)));
    int offsetAngleTip = angleTip + (90);// - contourAxisAngle);
    return abs(offsetAngleTip);
}

int DetectFingersProcessor::radiansToDegrees(double theta) { return theta * 180 /
M_PI; }
```

Załącznik 2 Kod źródłowy processor/RecognizeGestureProcessor

```
#include "headers/RecognizeGestureProcessor.h"

void RecognizeGestureProcessor::process(ImageContext *imgCtx) {
    char handState[50];
    sprintf(handState, "%s", "NONE");

    double distance = norm(imgCtx->gravityCenter - imgCtx-
>minEnclosingCircleCenter);
    int fingerCount = imgCtx->fingers.size();
    if(imgCtx->biggestContour.size()>0) {
        imgCtx->prevPointerPosition = imgCtx->pointerPosition;
        imgCtx->pointerPosition = imgCtx->gravityCenter;
        if ((fingerCount == 3  ||fingerCount == 4 ) && distance < 20) {
            recognizeGesture(imgCtx,1);
            sprintf(handState, "%s", "OPEN");
        }
        if (fingerCount == 0 && distance < 60) {
            recognizeGesture(imgCtx,2);
            sprintf(handState, "%s", "CLOSE");
        }

        if ((fingerCount == 1  ||fingerCount == 2 ) && distance < 45) {
            recognizeGesture(imgCtx,3);
            sprintf(handState, "%s", "GUN");
        }
    }else{
        recognizeGesture(imgCtx,0);

        sprintf(handState, "%s", "NONE");
        imgCtx->pointerPosition = Point(-1,-1);
        imgCtx->prevPointerPosition = Point(-1,-1);

    }
    cout<<"Recognized gesture: "<<handState<<endl;
    for (Vec4i finger : imgCtx->fingers) {
        Point ptStart = imgCtx->biggestContour[finger[0]];
        Point ptEnd = imgCtx->biggestContour[finger[1]];
        Point ptFar = imgCtx->biggestContour[finger[2]];
        double depth = finger[3] / 256;
        circle(imgCtx->original, ptStart, 8, COLORS_white, 2);
        circle(imgCtx->original, ptEnd, 3, COLORS_magenta, 3);
        circle(imgCtx->original, ptFar, 3, COLORS_yellow, 3);
    }
    putText(imgCtx->original, handState, Point(30, 150), CV_FONT_HERSHEY_SIMPLEX,
0.8, COLORS_white, 2,
            LINE_4);
}

void RecognizeGestureProcessor::recognizeGesture(ImageContext *imgCtx, int
gestureType) const {
    if(imgCtx->gestureType == gestureType){
        imgCtx->gestureCounter++;
    }else{
        imgCtx->gestureType = gestureType;
        imgCtx->gestureCounter = 0;
    }
};
int RecognizeGestureProcessor::angleBetween(Point start, Point far, Point end) {
    return abs((radiansToDegrees(atan2(start.y - far.y, start.x - far.x) -
                                 atan2(end.y - far.y, end.x - far.x))));
}
int RecognizeGestureProcessor::radiansToDegrees(double theta) { return theta *
180 / M_PI; }
```