# Spis treści

# 1. Wprowadzenie

Rozpoznawanie gestów dłoni to jedno z najbardziej obiecujących metod służących do interakcji pomiędzy człowiekiem a maszyną. Gest ten można zdefiniować jako ciągły w czasie zbiór sekwencji ludzkiej dłoni. Rozpoznawanie gestów jest szczególnie wykorzystywane w komunikacji pomiędzy ludźmi mającymi problemy ze słuchem. Dodatkowo znajduje zastosowanie w przypadku obserwacji oraz monitoringu wideo, jak również w zdalnym sterowaniu urządzeń (roboty, telewizory, konsole do gier).

W początkowej fazie rozwoju tej techniki, rozpoznawanie gestów było realizowane przy pomocy specjalnych rękawic lub znaczników umieszczonych na opuszkach palców. Jednakże z powodu, że użytkowanie takiego interfejsu jest niewygodne i nienaturalne, większość obecnych prac koncentruje się na podejściu opartym o systemy wideo rejestrujące gołą dłoń.

Wydajny interfejs człowiek-maszyna (z ang. *Human Computer Interface*, w skrócie HCI) to taki system, który jest w stanie rozpoznać gest ręki działając w czasie rzeczywistym. Jednakże śledzenie oraz rozpoznawanie gestów oparte o system wideo musi sprostać wielu wymagającym problemom, których źródłem jest złożoność gestów wynikająca z dużej ilości punktów swobody ludzkiej dłoni. Dodatkowo system czasu rzeczywistego musi poprawnie odseparować dłoń od reszty tła, spełniać wymagania odnośnie czasu obliczeń oraz poprawności klasyfikacji.

Ludzka dłoń może zostać odseparowana od innych elementów obrazu na podstawie informacji o kolorze skóry. Ta technika jest szeroko stosowana w kontekście detekcji dłoni ([6] oraz [9]). Twórcy pracy [5] zaproponowali sposób rozpoznawania gestów dłoni działający w czasie rzeczywistym. Pierwszy etap projektu zakładał wykrycie koloru skóry poprzez transformację obrazu gestu z przestrzeni barw RGB do modelu HSL (skrót od ang. *Hue, Saturation, Luminance*). Przestrzeń ta jest bardziej odporna na zmianę natężenia światła dla obserwowanej dłoni. Dodatkowo zastosowali metodę wykrywania twarzy opisaną w pracy [8], aby można było ją usunąć w momencie wyliczania wektora cech dla dłoni. Pozostający obraz został poddany działaniu algorytmu SIFT (skrót od ang. *Scale Invariance Feature Transform*) [3], który wyliczał wektor cech opisujący pojedynczy obraz. Kolejno każdy rezultat działania algorytmu SIFT został poddany grupowaniu, aby wszystkie elementy były takiej samej długości. Ostatni etap to proces tworzenia modelu SVM służący do klasyfikacji próbek testowych. Algorytm rozpoznawania gestów dłoni zaproponowany w niniejszej pracy został w dużym stopniu oparty o zasadę działania opisaną w pracy [5].

Na rynku istnieje kilka firm zajmujących się tworzeniem oprogramowania pomagającego osobom głuchym bądź słabo słyszącym. Jedną z najciekawszych aplikacji do rozpoznawania języka migowego

dostępnej na rynku jest produkt o nazwie *UNI* autorstwa firmy *Motionsavvy* z siedzibą w Rochester, Stany Zjednoczone [1]. Jest to organizacja zajmująca się rozwiązywaniem problemów z komunikacją dla ludzi mających problemy ze słuchem. Od 2011 roku *Motionsavvy* rozbudowuje bazę danych dla gestów Amerykańskiego Języka Migowego (z ang. *American Sign Language*, w skrócie ASL). Rozwijana przez nich aplikacja umożliwia komunikację osób niesłyszących poprzez rozpoznawanie gestów obu rąk oraz palców. Dodatkowo daje możliwość tłumaczenia mowy na język migowy. Aplikacja zawiera funkcjonalności umożliwiające nagrywanie, etykietowanie oraz edycję gestów wprowadzonych przez użytkownika. Zawiera również słownik, w którym każdy użytkownik aplikacji ma możliwość dodawania własnych gestów. Oprogramowanie zostało przetestowane oraz zainstalowane na lotnisku w Rochester, które co roku obsługuje największy odsetek ludzi słabo słyszących w całych Stanach Zjednoczonych. Problematyczny jest fakt, ze na chwilę obecną (Czerwiec 2018) aplikacja nie jest dostępna dla użytkowników komercyjnych. Niemożliwe zatem jest przetestowanie aplikacji pod względem poprawności klasyfikacji gestów.

## 1.1. Cele pracy

Celem pracy było stworzenie aplikacji do rozpoznawania gestów dłoni działające w czasie rzeczywistym. W tym celu opracowano mechanizm do definiowania własnych gestów, edytowania oraz usuwania istniejących rekordów z bazy danych. Kolejno należało zapoznać się z bibliotekami dla języka C# zawierającymi implementację potrzebnych metod pomocnych podczas tworzenia aplikacji. Dodatkowo zaproponowano schemat architektury dla tworzonego oprogramowania, jak również opracowano algorytm służący do rozpoznawania gestów oraz ich klasyfikacji. Następnie aplikacja została poddana testom, których celem było sprawdzenie poprawności implementacji oraz jej szybkości działania. Testy wykonano ze względu na metodę wyboru algorytmu oraz dla zmieniających się wartościach parametrów.

## 1.2. Zawartość pracy

Niniejsza praca składa się z 5 rozdziałów:

1. Wstęp - wprowadzenie do rozpoznawania gestów, ukazanie istoty oraz podkreślenie możliwości, jakie może przynieść zaimplementowana aplikacja. Dodatkowo opisano przykład działania algorytmu służącego do rozpoznawania gestów dłoni działającego w czasie rzeczywistym. Oprócz tego wspomniano o przykładzie firmy, która na swoim koncie ma udany projekt aplikacji pomagającej osobom głuchym lub słabosłyszącym w komunikacji.

2. Pojęcia związane z analizowanym problemem, przedstawienie jednej z najbardziej popularnych metod służących do opisu cech obrazu. Dodatkowo w rozdziale znajduje się opis wykorzystanych technologii oraz narzędzi, które okazały się pomocne w trakcie implementacji oprogramowania.

3. W rozdziale 3 przedstawiono architekturę aplikacji wraz z dokładnym opisem poszczególnych składowych programu. Opisano najważniejsze wzorce projektowe wykorzystane w projekcie.

4. Testy porównujące działanie aplikacji dla rożnych wartości parametrów.

5. Ostatni rozdział zawiera podsumowanie całej pracy oraz wskazuje kierunki dalszego rozwoju aplikacji.

# 2. Technologie i pojęcia wykorzystane w projekcie

W poniższym rozdziale przedstawiono zagadnienia, które są pomocne w zrozumieniu kontekstu całej pracy. W pierwszej części rozdziału opisano pojęcia związane z przetwarzaniem, detekcją oraz opisem cech dla podanego obrazu. Dodatkowo opisano jedną z najbardziej popularnych metod klasyfikacji. Druga część to opis technologii oraz narzędzi wykorzystanych podczas tworzenia aplikacji.

## 2.1. Ciągła przestrzeń skali dla obrazu

W cyfrowym przetwarzaniu obrazów model ciągłej przestrzeni skali może zostać użyty do reprezentacji obrazu jako rodziny stopniowo rozmywających się obrazów. Wykorzystanie ciągłej przestrzeni skali umożliwia znalezienie punktów na obrazie, które są niewrażliwe na zmiany skali (z ang. *scale invariant*).

To zagadnienie jest sformułowane bardzo ogólnie i istnieje wiele reprezentacji przestrzeni skali. Typowym podejściem do zdefiniowania szczególnej reprezentacji przestrzeni skali jest podanie zbioru aksjomatów opisujących podstawowe własności szukanej przestrzeni. Najbardziej powszechnym zbiorem aksjomatów jest zbiór definiujący liniową przestrzeń skali powiązaną z funkcją Gaussa. Wybór takiej funkcji jest motywowany faktem, że jądro rozkładu Gaussa nie potęguje wpływu lokalnych ekstremów na jakość otrzymanej przestrzeni skali. Dodatkowo użycie funkcji Gaussa zapewnia liniowość, odporność na przesunięcia, zmianę skali czy rotacji.

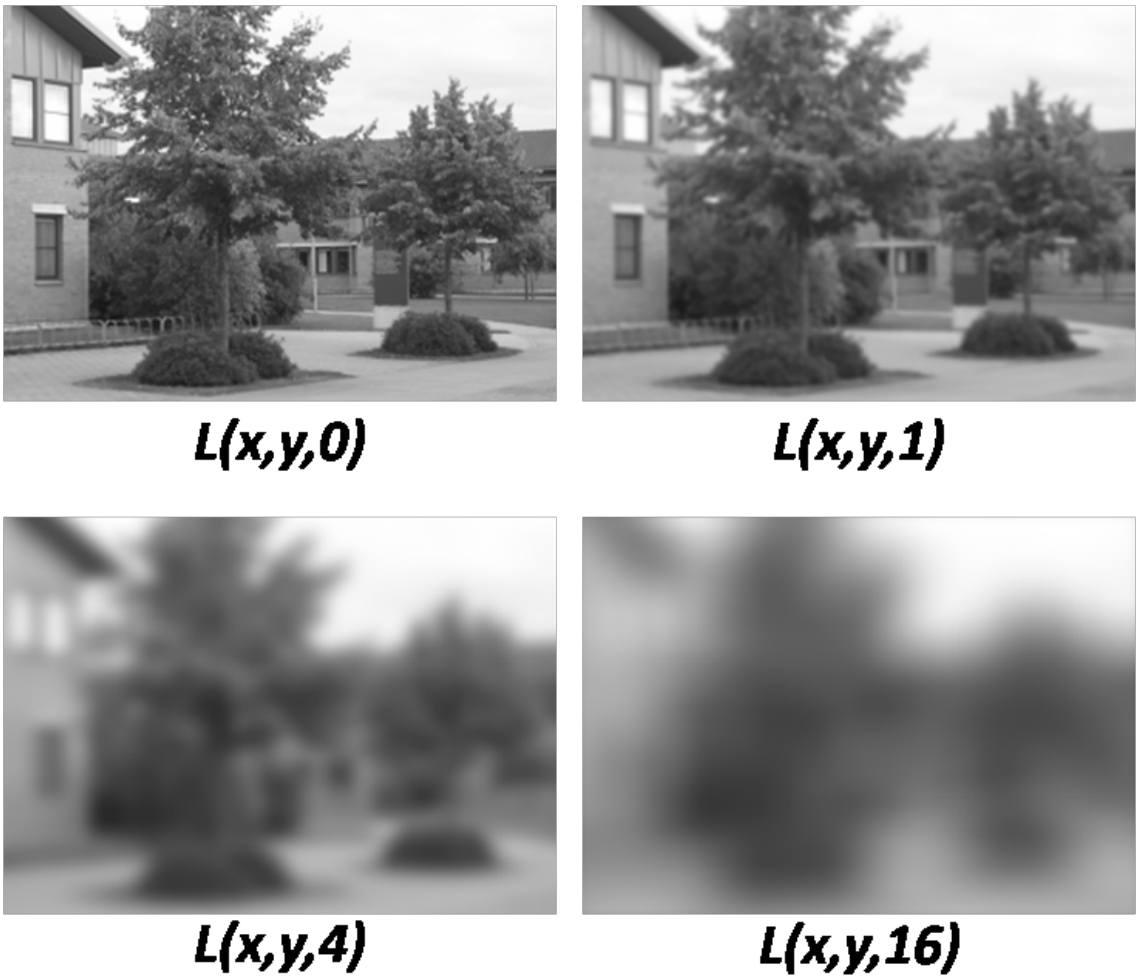
Gaussowska przestrzeń skali (dla obrazu dwuwymiarowego) zdefiniowana jest jako splot obrazu *I(x,y)* z dwuwymiarową funkcją Gaussa *g(x,y,σ)*:

$$L(x, y, \sigma) = g(x, y, \sigma) * I(x, y) \tag{2.1}$$

gdzie:

$$g(x, y, \sigma) = \frac{1}{2\pi\sigma} e^{-(x^2+y^2)/2\sigma} \tag{2.2}$$

Dla $\sigma = 0$ filtr Gaussa staje się funkcją impulsową, zatem *L(x,y,0) = f(x,y)*. Wraz ze zwiększaniem parametru $\sigma$ przestrzeń skali $L$ staje się coraz bardziej gładka, czyli coraz mniej szczegółów przestaje być widoczne. Na rysunku 2.1 przedstawiono przykład tworzenia przestrzennej reprezentacji skali.



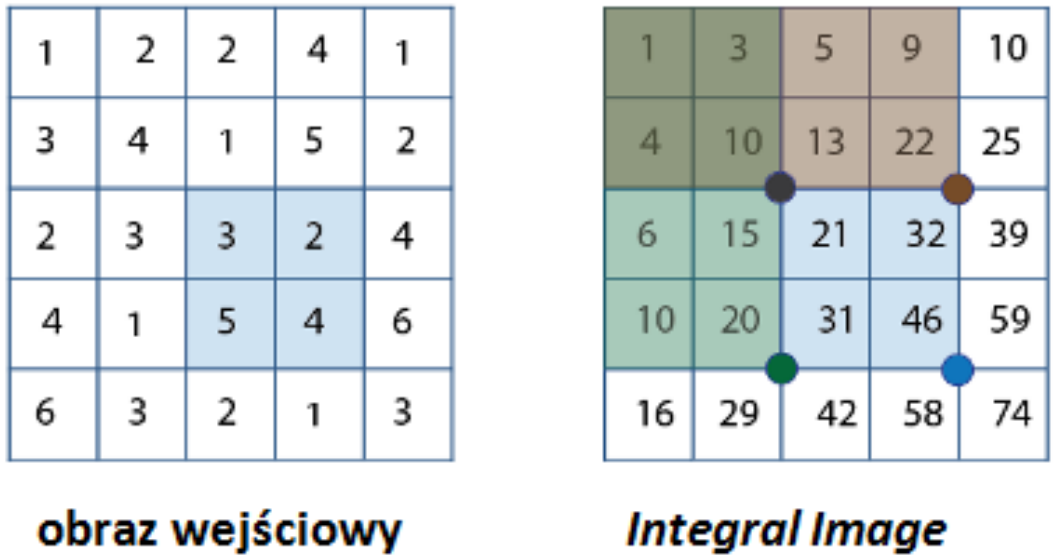

**Rys. 2.1.** Reprezentacja przestrzeni skali dla różnych wartości $\sigma$. Jak można zauważyć, wraz ze wzrostem parametru $\sigma$ korespondujące obrazy są coraz mocniej rozmyte.

## 2.2. Obraz całkowy

Obraz całkowy (z ang. *Integral Image* [7]) to struktura danych wykorzystywana w celu efektywnej i szybkiej generacji sum pikseli dla podanego regionu obrazu. Znalazła zastosowanie wszędzie tam, gdzie konieczna jest wysoka wydajność obliczeń dla dowolnie dużego regionu w obszarze obrazu. Dowolny piksel *(x, y)* wchodzący w skład obrazu *I* może zostać przedstawiony jako suma wszystkich pikseli na lewo oraz powyżej *(x, y)*:

$$\textit{Obraz całkowy}(x^{'}, y^{'}) = \sum_{x<x^{'}, y<y^{'}} I(x, y). \tag{2.3}$$

Użycie takiej reprezentacji umożliwia uzyskanie sumy pikseli dowolnego obszaru obrazu w stałym czasie, bez względu na jego rozmiar. Dodatkowo wyliczenie obrazu całkowego następuje w pojedynczym przejściu po pikselach. Wynika to z faktu, że kolejne elementy struktury są tworzone na podstawie już istniejących. Przykład wykorzystania obrazu całkowego przedstawiono na rysunku 2.2




**Rys. 2.2.** Zasada wyliczania obrazu całkowego

Wyliczenie sumy wyróżnionego regionu na obrazie wejściowym można zastąpić operacjami na obrazie całkowym. Sumę obszaru można uzyskać korzystając z czterech wartości powyżej oraz na lewo od zaznaczonych kropek: *46 - 22 - 20 + 10 = 14*. Jak nietrudno obliczyć, wynik ten jest równy sumie zaznaczonych elementów obrazu wejściowego.

## 2.3. Algorytm SURF

Algorytm SURF (skrót od ang. *Speeded Up Robust Features*) został opatentowany przez grupę naukowców w 2008 roku [4]. Należy do rodziny algorytmów bazujących na punktach kluczowych i służy do porównywania dwóch obrazów operując w odcieniach szarości. W celu znalezienia cech obrazu niezależnych od zmiany skali wykorzystuje opisaną w podrozdziale 2.1 technikę utworzenia ciągłej przestrzeni skali opartej o funkcję Gaussa. Działanie algorytmu można podzielić na 3 etapy:

– Detekcja (z ang. *Detection*) – faza automatycznej identyfikacji punktów kluczowych (z ang. *interest points*). Te same punkty powinny zostać wykryte niezależnie od zmian w położeniu, naświetleniu oraz orientacji obrazu (również w pewnym stopniu od zmiany skali oraz punktu widzenia).

– Opis (z ang. *Description*) – każdy punkt kluczowy powinien zostać opisany w unikatowy sposób, aby był niezależny od rotacji oraz przeskalowania obrazu.

– Zestawienie (z ang. *Matching*) – faza, podczas której określa się (na podstawie podanych punktów kluczowych) jakie obiekty znajdują się na obrazie.

Na potrzeby niniejszej pracy w dalszej części rozdziału przedstawiono bardziej dokładną analizę dwóch pierwszych etapów.

### 2.3.1. Detekcja

Algorytm SURF do wykrycia punktów kluczowych wykorzystuje wyznacznik Hesjanu. Dokładniej rzecz ujmując, metoda ta wyszukuje na obrazie regionów, w których wyznacznik macierzy Hessego jest lokalnie maksymalny. Mając do dyspozycji punkt $\boldsymbol{x}=(x,y)$ z obrazu całkowego, macierz Hessego $H(\boldsymbol{x},\sigma)$ dla skali $\sigma$ jest zdefiniowana następująco:

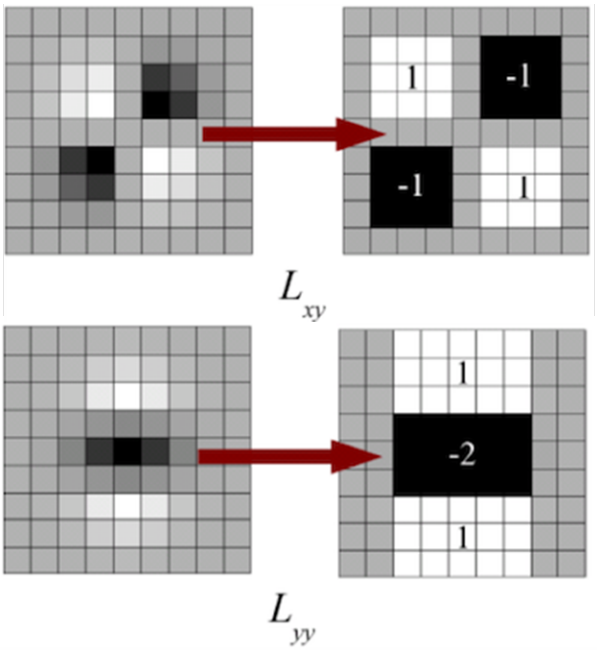
$$H(\mathbf{x},\sigma) = \begin{bmatrix} L_{xx}(\mathbf{x},\sigma) & L_{xy}(\mathbf{x},\sigma) \\ L_{xy}(\mathbf{x},\sigma) & L_{yy}(\mathbf{x},\sigma) \end{bmatrix} \tag{2.4}$$

gdzie

$$L_{xx}(\mathbf{x},\sigma) = I(\mathbf{x}) * \frac{\delta^2}{\delta x^2} g(\sigma) \tag{2.5}$$

$$L_{xy}(\mathbf{x},\sigma) = I(\mathbf{x}) * \frac{\delta^2}{\delta xy} g(\sigma) \tag{2.6}$$

Udowodniono, że przestrzeń skali oparta o funkcję Gaussa jest rozwiązaniem optymalnym ([10]), jednakże w zastosowaniach praktycznych wyliczanie splotu jest niezwykle kosztowne obliczeniowo. W celu przyspieszenia obliczeń dokonano aproksymacji drugich pochodnych cząstkowych filtrami przedstawionymi na rysunku 2.3. Dodatkowo wykorzystanie obrazu całkowego powoduje, że czas wyliczania splotów nie zależy od wielkości filtra.




**Rys. 2.3.** Dwa rysunki po lewej to sploty $L_{xy}$ oraz $L_{yy}$ poddane dyskretyzacji oraz przycięciu. Po prawej stronie przedstawiono aproksymacje wyżej wymienionych splotów (odpowiednio $D_{xy}$ oraz $D_{yy}$). Szare regiony są równe zero. Źródło: [4].

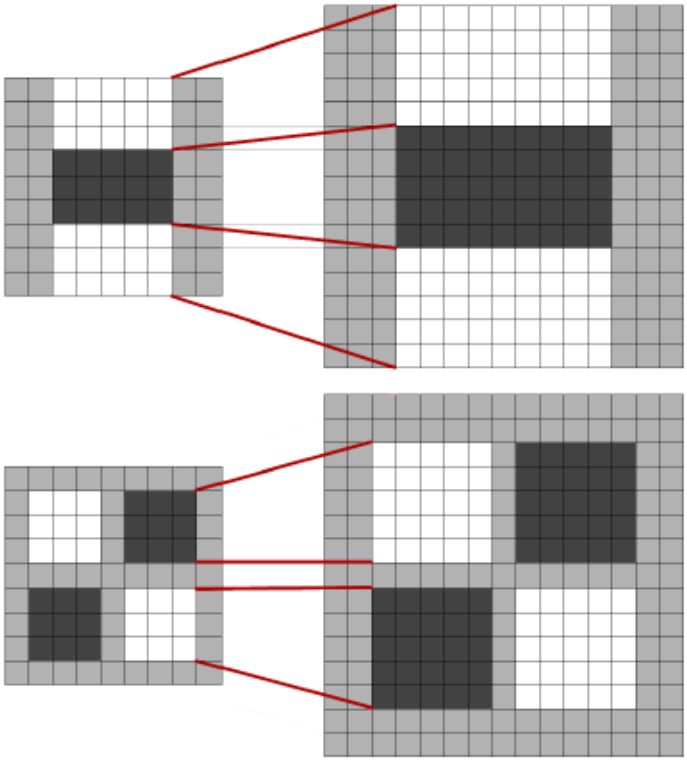
Przedstawione filtry o rozmiarze *9 x 9* odpowiadają splotom, dla których parametr $\sigma$ jest równy 1.2. Jest to najmniejsza wartość skali, dla której algorytm SURF może dawać zadowalające rezultaty.

Biorąc pod uwagę powyższe założenia, wyznacznik aproksymowanej macierzy Hessego wynosi:

$$det(H_{aproks}) = D_{xx}D_{yy} - (wD_{xy})^2. \tag{2.7}$$

Aby uczynić aproksymację Hesjanu bardziej dokładną wprowadzono parametr $w$. Teoretycznie jest on zależny od skali, jednakże badania wykazały ([4]), że można uczynić go stałą równą *0.9*. Wynikiem powyższych działań jest uzyskanie aproksymowanego wyznacznika Hesjanu dla każdego punktu obrazu $\boldsymbol{x}$ przy różnych wartościach parametru $\sigma$.

Algorytm SURF dzieli przestrzeń skali na oktawy. Oktawa reprezentuje zbiór odpowiedzi filtrów otrzymanych przez splot obrazu z filtrami coraz większych rozmiarów. Każda oktawa odpowiada fragmentowi przestrzeni skali, w którym nastąpiło podwojenie parametru $\sigma$. Dodatkowo podzielona jest na stałą liczbę poziomów. Wraz ze wzrostem wielkości filtrów muszą zostać spełnione dwa założenia: o istnieniu piksela centralnego oraz o zachowaniu proporcji poszczególnych obszarów maski. W pracy [4] opisano szczegółowo, w jaki sposób definiować oktawy oraz liczbę poziomów dla nich. Przykład poprawnego skalowania filtra przedstawiono na rysunku 2.4.

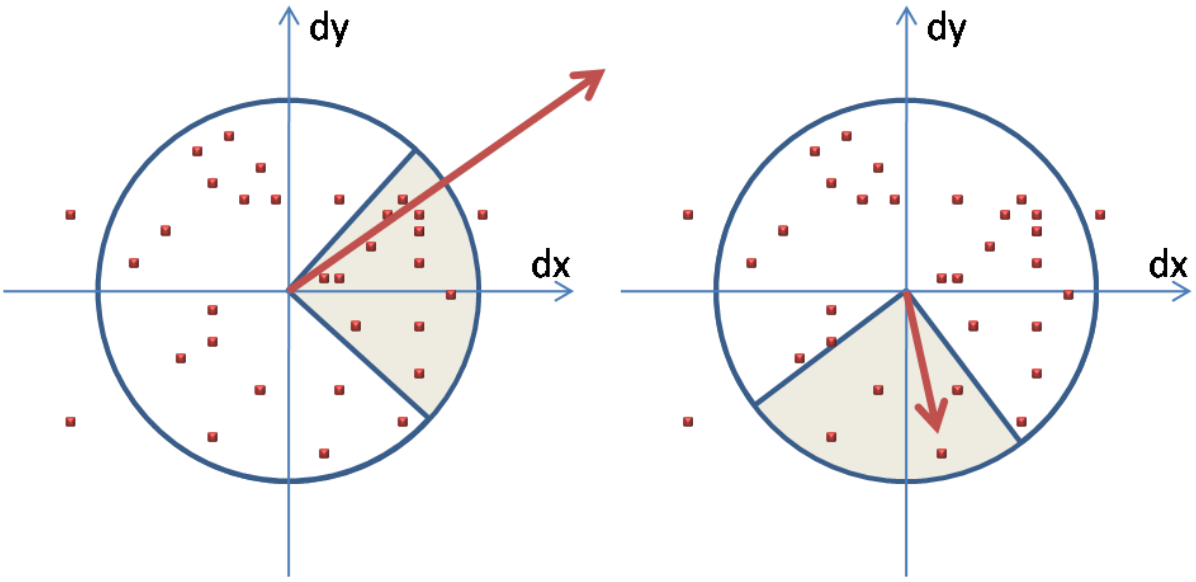


**Rys. 2.4.** Filtr $D_{yy}$ oraz $D_{xy}$ dla dwóch kolejnych poziomów w oktawie (*9 x 9* oraz *15 x 15*). Długość czarnego regionu dla górnego filtra może zostać zwiększona tylko o parzystą liczbę pikseli w celu zagwarantowania istnienia piksela centralnego.

W celu zlokalizowania punktów kluczowych na obrazie we wszystkich skalach, algorytm SURF wykorzystuje ograniczanie lokalnych wartości niemaksymalnych (z ang. *non-maximal suppression*) dla obszaru wielkości *3 x 3 x 3* piksele. Zasada działania została opisana w pracy [4]. Następnie maksima wyznacznika Hessjanu dla poszczególnych skal są interpolowane na obraz oryginalny.

### 2.3.2. Deskryptor

Aby punkt kluczowy był niewrażliwy na zmiany orientacji, punktowi przypisywana zostaje orientacja. W tym celu algorytm SURF wylicza odpowiedź falki Haara dla kolistego otoczenia punktu orientacji. Falka jest wyliczana w kierunku poziomym (*dx*) oraz pionowym (*dy*) dla każdego elementu z otoczenia punktu kluczowego. Główna orientacja jest wyliczana następująco: mając odpowiedzi falki Haara dla każdego punktu z otoczenia, skonstruowano przesuwne okno o kącie rozwarcia równym 60 stopni. Dla danego okna wyliczano sumę wszystkich elementów, po czym okno zostaje przesunięte. Najdłuższy znaleziony wektor stanowi główną orientację podanego punktu kluczowego. Szczegóły na rysunku 2.5.

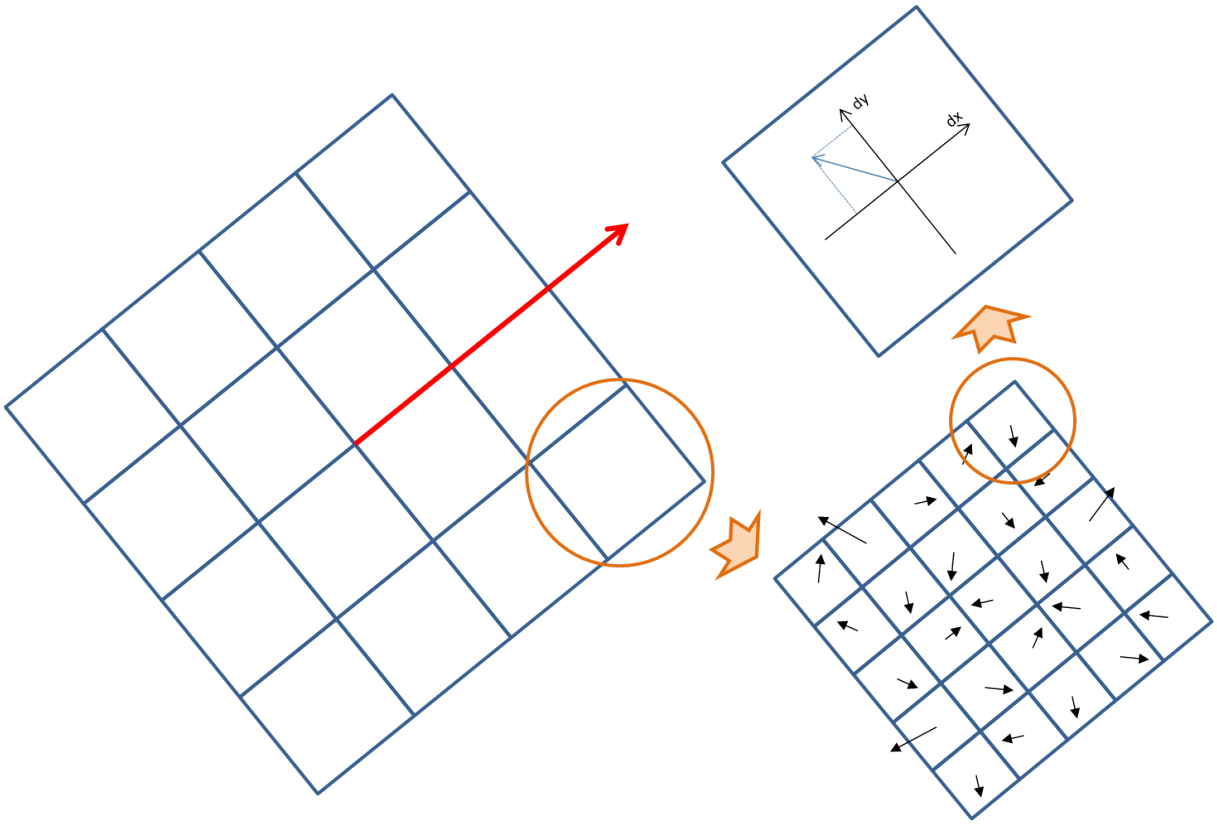



**Rys. 2.5.** Przypisanie orientacji. Okno o kącie rozwarcia 60 stopni obraca się wokół początku układu współrzędnych, a wyliczone odpowiedzi falki Haara zostają sumowane tworząc wektory oznaczone kolorem czerwonym. Najdłuższy wektor determinuje główną orientację punku kluczowego.

Kolejnym etapem tworzenia deskryptora jest podzielenie obszaru wokół punktu kluczowego na *4 x 4* kwadratowe obszary. Taki podział zachowuje istotne przestrzenne informacje. Każdy z podregionów zawiera *5 x 5* punktów rozmieszczonych regularnie w wierzchołkach siatki. Dla każdego punktu wyliczone zostają odpowiedzi falki Haara w kierunku poziomym oraz pionowym. Odpowiedzi te uwzględniają rotację całego obszaru zgodnie z główną orientacją badanego punktu kluczowego. Schemat przedstawiono na rysunku 2.6. Następnie rezultaty *dx* oraz *dy* są sumowane dla każdego z podregionów. Stanowią one

pierwszą część deskryptora cechy. Dodatkowo w celu uwzględnienia informacji o zmianach intensywności wyliczane są sumy modułów odpowiedzi falki Haara.

Zatem, dla każdego z podregionów otrzymano czterowymiarowy wektor opisujacy o strukturze $\boldsymbol{v} = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$. Uwzględniając wszystkie podregiony, punkt kluczowy opisany jest 64-elementowym wektorem.




**Rys. 2.6.** Tworzenie deskryptora. Otoczenie punktu kluczowego obrócono zgodnie z orientacją cechy. Dla wszystkich elementów podregionu wyliczono odpowiedź falki Haara w dwóch kierunkach.

## 2.4. Algorytm centroidów

Algorytm K-średnich (z ang. *K-means*) jest jedną z najprostszych metod klasyfikacji bez nadzoru (z ang. *unsupervised learning*). Grupowanie elementów oparte jest o wstępne podzielenie zbioru danych na z góry założoną liczbę klastrów. W kolejnych krokach niektóre z elementów skupień są przenoszone do innych klastrów w taki sposób, aby wariancja wewnątrz każdej z grup była jak najmniejsza. Minimalizacja wartości wariancji powoduje, że elementy wewnątrz poszczególnych klastrów są do siebie maksymalnie podobne. Schemat działania algorytmu K-średnich może zostać opisany w następujący sposób:

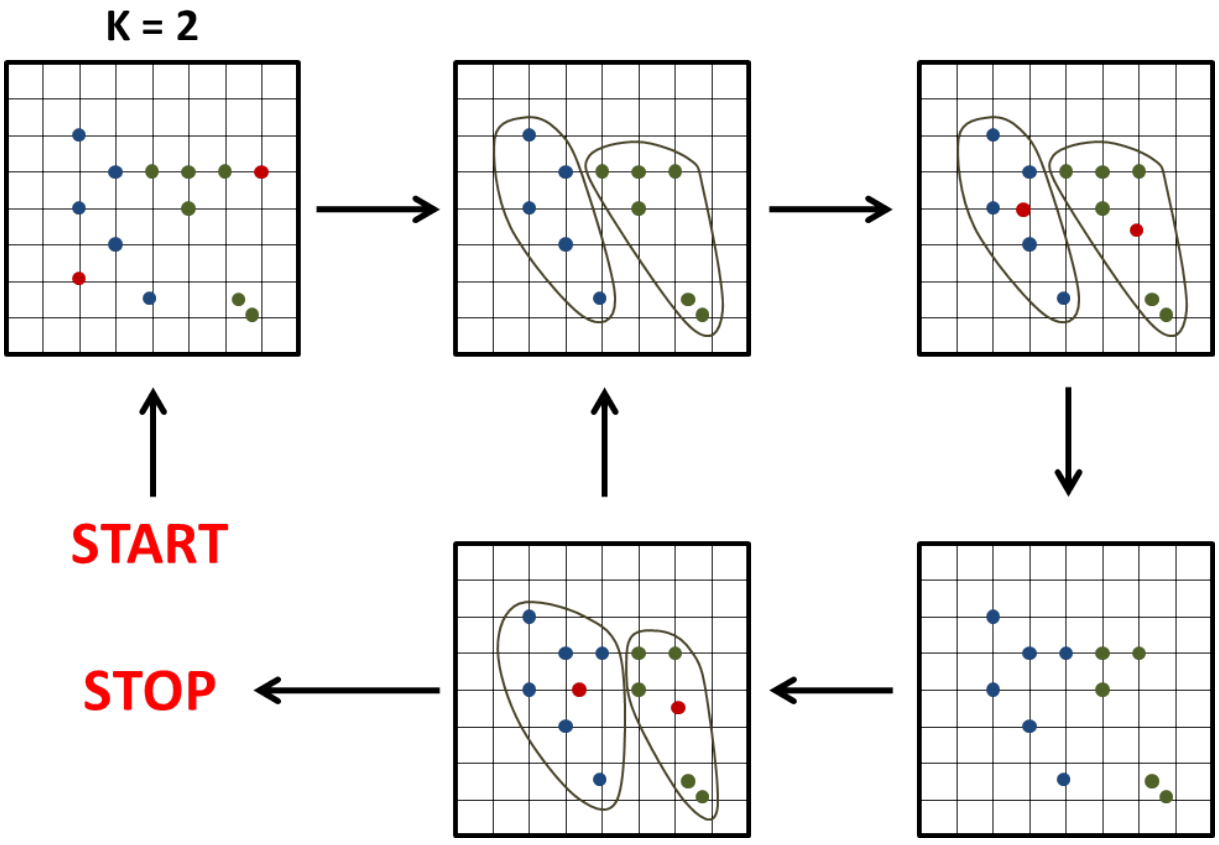
1. Wybór centroidów klastrów: dla podanej z góry liczby klas (skupień) środki klastrów powinny zostać dobrane w taki sposób, aby były możliwie jak najdalej od siebie.

2. Przyporządkowanie każdego elementu ze zbioru danych do najbliższego środka klastra, normę może stanowić odległość euklidesowa lub Czebyszewa.

3. Ponowne wyliczenie środków skupień: w większości zastosowań nowym środkiem klastra jest punkt o współrzędnych stanowiących średnią arytmetyczną elementów wewnątrz skupienia.

4. Powtarzanie algorytmu do momentu osiągnięcia kryterium zbieżności: sytuacja, podczas której środki klastrów pozostają bez zmian bądź nie zmienia się przynależność elementów do klas.

Na rysunku 2.7 przedstawiono graf przepływu wyjaśniający zasadę działania algorytmu K-średnich.




**Rys. 2.7.** Algorytm K-średnich. Pierwszy etap to wybór środków dla klastrów. Następnie dokonywane jest przypisanie punktów do odpowiedniej klasy. Kolejny etap to ponowne wyliczenie centroidów oraz reorganizacja klastrów. Algorytm działa dopóki środki klas ulegają zmianie.

## 2.5. Metoda Wektorów Nośnych

Metoda wektorów nośnych (z ang. *Support Vector Machine*, SVM) to jeden z modeli uczenia maszynowego służący do klasyfikacji danych. Model SVM po raz pierwszy został opublikowany przez dwóch naukowców: Vladimira N. Vapnika oraz Alexey'a Chervonenkisa w 1963 roku. Może zostać użyty do rozwiązywania takich problemów jak rozpoznawanie tekstu pisanego czy klasyfikacja obrazów.

Mając do dyspozycji zbiór punktów uczących, w którym każdy z elementów należy do jednej z dwóch klas, SVM tworzy model, który przypisuje próbki testowe do jednej z dwóch kategorii. Model SVM jest reprezentowany jako zbiór punktów na przestrzeni skonstruowany w taki sposób, aby istniała wyraźna przerwa oddzielająca elementy rożnych klas. W zależności od tego, po której stronie przerwy próbka testowa została umiejscowiona, do tej kategorii zostanie przypisana. Bardziej formalnie SVM konstruuje hiperpłaszczyznę rozdzielającą zbiór próbek uczących. Najlepsza separacja zachodzi wtedy, gdy odległość hiperpłaszczyzny od najbliższego elementu dla każdej z klas jest maksymalna.

Zbiór $n$-punktów uczących opisano następująco:

$$(\vec{x}_1, y_1), ..., (\vec{x}_n, y_n) \tag{2.8}$$

gdzie $\vec{x}_i$ oznacza $\boldsymbol{p}$-wymiarowy wektor. Zmienna $y_i$ przyjmuje wartości -1 bądź 1 w zależności od tego, do której kategorii należy $\vec{x}_i$. Równianie płaszczyzny może zostać opisane następująco:

$$\vec{w} \cdot \vec{x} - b = 0 \tag{2.9}$$

gdzie $\vec{w}$ oznacza wektor normalny do szukanej hiperpłaszczyzny. W przypadku, gdy zbiór danych może być separowany liniowo, istnieje możliwość wyboru dwóch równoległych hiperpłaszczyzn, których odległość względem siebie jest maksymalna. Region leżący pomiędzy tymi hiperpłaszczyznami jest nazywany *marginesem*, a optymalna hiperpłaszczyzna leży w jego połowie. Równania dla dwóch równoległych hiperpłaszczyzn są opisane następująco:

$$\vec{w} \cdot \vec{x}_i - b = 1 \tag{2.10}$$

$$\vec{w} \cdot \vec{x}_i - b = -1 \tag{2.11}$$

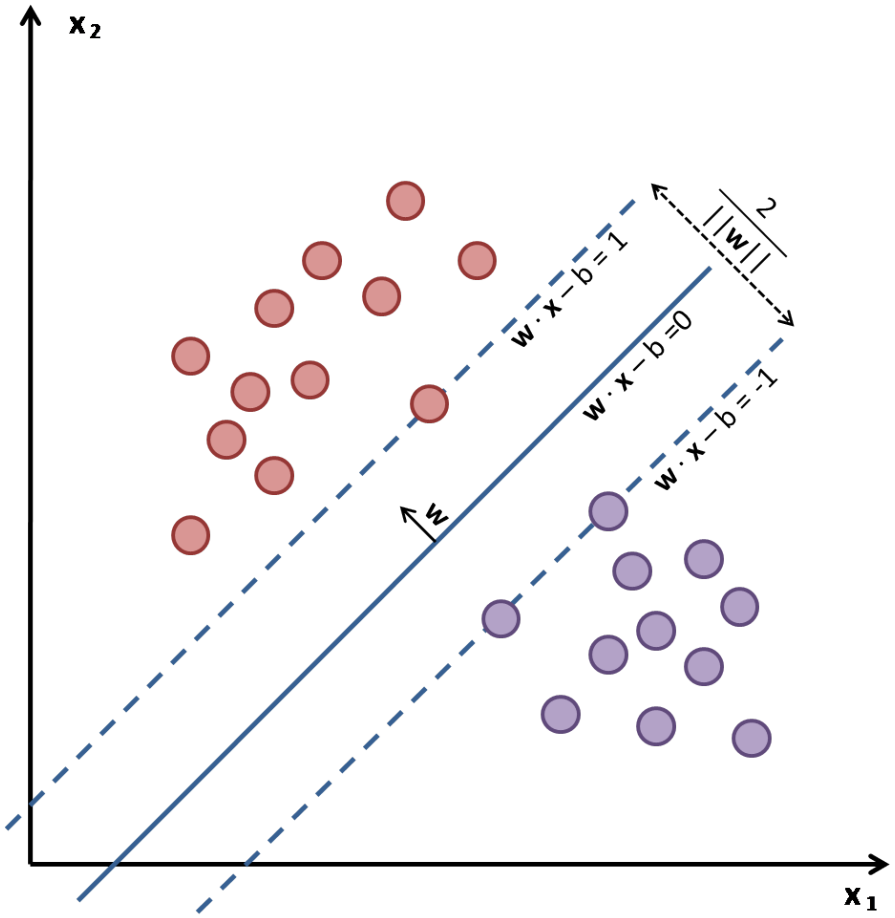
Dystans pomiędzy dwoma hiperpłaszczyznami jest równy $2/||\vec{w}||$. Zatem w celu maksymalizacji dystansu pomiędzy płaszczyznami należy dokonać minimalizacji $||\vec{w}||$. Dodatkowo wymuszono obecność dowolnej próbki po właściwej stronie marginesu używając następujących zależności:

$$\vec{w} \cdot \vec{x}_i - b \geq 1 \quad dla \quad y_i = 1 \tag{2.12}$$

$$\vec{w} \cdot \vec{x}_i - b \leq -1 \quad dla \quad y_i = -1 \tag{2.13}$$

Powyższe nierówności mogą zostać przekształcone do:

$$y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1 \quad dla \ \ wszystkich \ \ 1 \leq i \leq n \tag{2.14}$$




**Rys. 2.8.** Sposób wyznaczania optymalnej hiperpłaszczyzny.

Zadanie minimalizacji może zostać postawione w następujący sposób: "Minimalizacja normy wektora $\vec{w}$ dla zależności $y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1$ dla $i = 1 \ldots n$". Szczegóły tworzenia hiperprzestrzeni w dwóch wymiarach przedstawiono na rysunku 2.8

Oryginalny algorytm do szukania hiperpłaszczyzny rozdzielającej z maksymalnym marginesem stworzony w 1963 roku był liniowym klasyfikatorem. Jednakże w latach 90-tych Vapnik przy wsparciu innych naukowców zaproponował sposób umożliwiający konstrukcję nieliniowego klasyfikatora przez zastosowanie tzw. *"kernel trick"*. Algorytm w swoim działaniu jest bardzo podobny do oryginału z tą różnicą, że każdy iloczyn skalarny zostaje zastąpiony przez nieliniową funkcję jądra. Do najbardziej znanych funkcji jądra należą:

– Wielomianowa jednorodna:

$$k(\vec{x}_i, \vec{x}_j) = (\vec{x}_i \cdot \vec{x}_j)^d \tag{2.15}$$

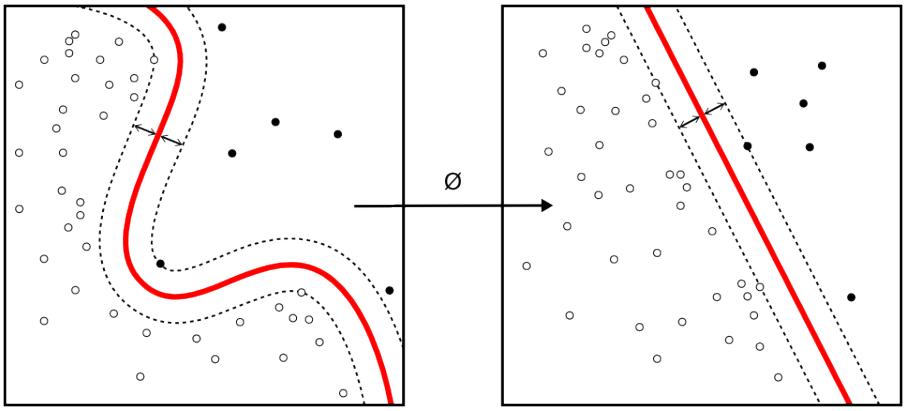
– Wielomianowa niejednorodna:

$$k(\vec{x}_i, \vec{x}_j) = (\vec{x}_i \cdot \vec{x}_j + 1)^d \tag{2.16}$$

– Jądro RBF (z ang. Radial Basis Funcion):

$$k(\vec{x}_i, \vec{x}_j) = \exp\left(-\frac{||\vec{x}_i - \vec{x}_j||^2}{2\sigma^2}\right) \tag{2.17}$$

Przykład zastosowania funkcji jądra przedstawiono na rysunku 2.9.




**Rys. 2.9.** Sposób wyznaczania optymalnej hiperpłaszczyzny.

Istnieje kilka strategii służących do redukcji problemu klasyfikacji dla wielu klas. Do najpopularniejszej metody należy redukcja takiego problemu do wielu przypadków dwuklasowych. Metoda występuje w dwóch wariantach: *"jeden-na-resztę"* oraz *"jeden-na-jednego"*.

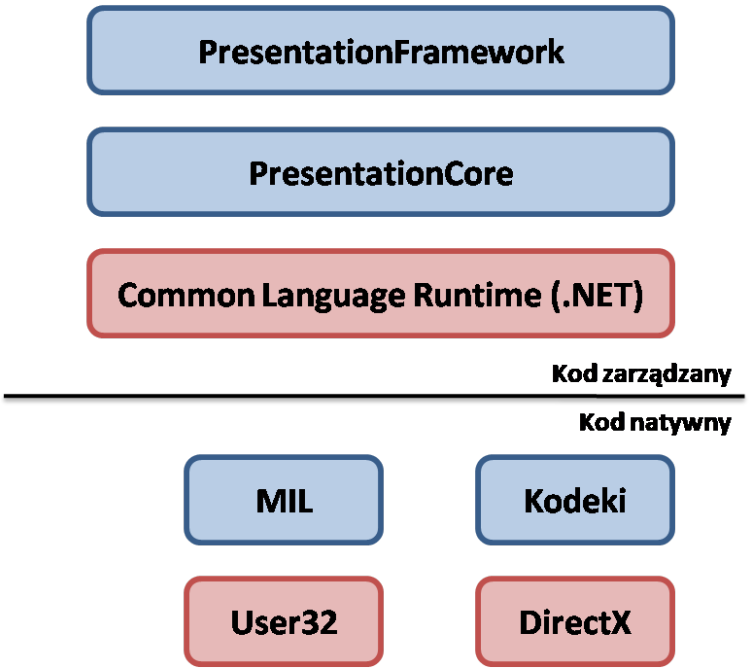
## 2.6. Windows Presentation Foundation (WPF)

Windows Presentation Foundation ([2]) jest silnikiem graficznym dostarczanym przez firmę *Microsoft*. Jego premiera nastąpiła w 2006 roku, gdy stał się częścią platformy programistycznej .NET w wersji 3.0. Jest wykorzystywany głównie do budowania aplikacji okienkowych nowej generacji dla systemu operacyjnego Windows. WPF zbudowany został całkowicie niezależnie do dotychczasowego silnika renderującego GDI. Dostarcza model programistyczny umożliwiający budowanie aplikacji oraz pozwalający na bezwzględną separację logiki biznesowej od interfejsu użytkownika.

Architektura silnika WPF została oparta zarówno o kod zarządzany, jak i o kod natywny. Większość elementów składowych znajduje się w kodzie zarządzanym, tak jak publiczne API dostępne dla deweloperów. Na rysunku 2.10 przedstawiono architekturę silnika, w skład którego wchodzą:

– PresentationFramework – biblioteka implementująca elementy do prezentacji dla końcowego uzytkownika tj. rozkład kontrolek, wyświetlanie animacji, skalowanie aplikacji.

– PresentationCore – podstawowa biblioteka w technologii WPF. Dostarcza wraper dla MIL z poziomu kodu zarządzanego oraz implementuje bazowe usługi dla każdej aplikacji WPF. W skład tych usług wchodzi przede wszystkim system zarządzania wiadomościami , którego implementację stanowi obiekt typu Dispacher.




**Rys. 2.10.** Architektura WPF. Czerwone elementy to komponenty bibliotek Windows. Składowe WPF oznaczono kolorem niebieskim.

– Media Integration Layer, MIL – komponent działający w kodzie niezarządzanym w celu zapewnienia wydajnej współpracy z DirectX. Zawiera silnik kompozycji, który odpowiada za podstawową obsługę renderowania powierzchni 2D oraz 3D.

– Kodeki – zbiór programów służących do przekształcania strumienia danych do postaci multimedialnej.

– DirectX – kolekcja zawierająca interfejsy programistyczne aplikacji (z ang. Application Programming Interfaces, APIs). Zestaw ten wspomaga generację grafiki, dźwięku oraz innych elementów związanych z aplikacjami multimedialnymi

– User32 – komponent Microsoft Windows dostarczający bazowe funkcjonalności do tworzenia prostych interfejsów użytkownika. Aplikacje WPF zawierają obiekt typu Dispacher, który używa systemu zarządzania wiadomościami dostępnymi w User32.

– Common Language Runtime, CLR – wspólne środowisko uruchomieniowe. Podstawowy komponent .NET. Pełni wiele kluczowych roli, a do najważniejszych z nich należy uruchomienie aplikacji czy zarządzanie pamięcią. Dodatkowo zajmuje się również konwersją języka IL do kodu maszynowego. Elementem bazowym środowiska CLR jest standardowy zestaw typów danych, który jest wykorzystywany przez wszystkie języki programowania oparte o CLR.

Silnik WPF udostępnia system własności dla obiektów, które dziedziczą z DependencyObject. Obiekt ten monitoruje wszystkie zależności pomiędzy własnościami i jest w stanie wykonywać odpowiednie akcje bazując na ich zmianach. Własności implementują mechanizm informujący o zmianach

(z ang. Change notifications), który wywołuje wbudowane zachowania (z ang. Behaviors) w przypadku wykrycia jakiejkolwiek zmiany. Dodatkowo istnieje możliwość definiowania własnych zachowań w celu propagowania informacji o zmianie własności do innych składowych . System zarządzania rozkładem elementów w obszarze interfejsu użytkownika wykorzystuje powyższy zbiór zachowań do przeliczania nowego rozkładu w przypadku zmiany własności. Dzięki temu architektura systemu WPF spełnia deklaratywny paradygmat programowania, w którym praktycznie wszystko, począwszy od ustawania wielkości kontrolek do tworzenia animacji może zostać osiągnięte poprzez zmianę własności. Takie zachowanie umożliwia tworzenie aplikacji WPF w XAML (z ang. Extensible Application Markup Language) – deklaratywnym języku znaczników, gdzie przy pomocy atrybutów oraz słów kluczowych tworzone jest bezpośrednie połączenie z własnościami oraz klasami technologii WPF.

Każdy element interfejsu aplikacji WPF dziedziczy z abstrakcyjnej klasy Visual. Obiekty tej klasy dostarczają interfejs do drzewa kompozycji zarządzanego przez MIL. Każdy element WPF tworzy oraz dodaje przynajmniej jeden węzeł kompozycji do drzewa. Węzły te zawierają przede wszystkim instrukcje renderowania takie jak przycinanie elementu bądź transformacja wizualna. Zatem cała aplikacja może być traktowana jako kolekcja węzłów kompozycji, które są przechowywane w buforze pamięci. Okresowo MIL przechodzi po strukturze drzewa i wykonuje instrukcje renderowania dla każdego węzła. Powoduje to tworzenie kompozytu na powierzchni DirectX, która następnie jest wyświetlana na ekranie. MIL wykorzystuje algorytm malarza, w którym wyświetlanie elementów na monitorze rozpoczyna się od tych najbardziej odległych (tło). Takie zachowanie umożliwia renderowanie złożonych efektów takich jak rozmycie czy transparentność. Dodatkowo proces rysowania jest sprzętowo wspomagany przy pomocy GPU.

Każda z aplikacji WPF staruje z dwoma wątkami: pierwszy służy do obsługi interfejsu użytkownika, a drugi, działający w tle, obsługuje renderowanie oraz przerysowywanie – jego działanie jest automatyczne, więc nie wymaga żadnej interwencji dewelopera. Wątek powiązany z UI przechowuje obiekt Dispacher'a (poprzez instancję klasy DispacherObject), który zajmuje się kolejkowaniem operacji koniecznych do wykonania na interfejsie użytkownika.

Etap tworzenia układu interfejsu użytkownika podzielony jest na dwie fazy: mierzenie (z ang. measure) oraz porządkowanie (z ang. arrange). Faza mierzenia rekursywnie wywołuje wszystkie elementy oraz określa rozmiar, z jakim one będą wyświetlane. Porządkowanie to faza, podczas której następuje rekursywne układanie wszystkich elementów w stosunku do ich rodziców w drzewie kompozycji.

## 2.7. C#

C# jest językiem programowania spełniającym wiele paradygmatów takich jak programowanie funkcyjne, obiektowe, imperatywne czy generyczne. Został utworzony przez firmę *Microsoft* wewnątrz platformy .NET i zatwierdzony jako standard przez ISO (ISO/IEC 23270:2006) oraz Ecma (ECMA-334). Jest jednym z języków wchodzącym w skład Architektury Wspólnego Języka (z ang. *Common Language*

*Infrastructure, w skrócie CLR*). Najnowsza wersja języka to C# 7.3, która ukazała się w 2018 roku wraz z środowiskiem programistycznym *Visual Studio 2017* w wersji 15.7.2. Przykładowe cechy języka C#:

- C# z założenia ma być językiem prostym, nowoczesnym, obiektowym
- Język posiada hierarchię klas, a wszystkie elementy (nawet najprostsze typy) dziedziczą z klasy *System.Object*.
- Język ma zapewniać wsparcie podczas tworzenia oprogramowania, w którego skład wchodzą: sprawdzanie silnego typowania, kontrola zakresu tablic, detekcja prób użycia niezainicjowanych zmiennych.
- Automatyczne odśmiecanie pamięci (usuwanie nieużywanych elementów) - wykorzystanie mechanizmu *Garbage Collector*.
- Wielodziedziczenie, czyli dziedziczenie od więcej niż jednej klasy jest niedozwolone. Wielokrotne dziedziczenie jest możliwe jedynie po interfejsach.
- Wsparcie dla internacjonalizacji.
- Przenośność oprogramowania oraz łatwość wdrożenia w rozproszonych środowiskach.
- Język C# jest bezpieczny w kontekście konwersji typów. Automatyczna konwersja jest dokonywana tylko w przypadku, gdy dane rzutowanie jest uznawane za bezpieczne.
- Mechanizm refleksji oraz dynamicznego tworzenia kodu. Takie wsparcie umożliwia tworzenie oprogramowania, którego części nie są w całości znane podczas kompilacji. Takie działanie jest szeroko wykorzystywane w procesie mapowania obiektowo-relacyjnego (z ang. Object-Relational Mapping, w skrócie ORM).

## 2.8. .NET

Platforma programistyczna .NET została zaprojektowana przez firmę *Microsoft*. Zawiera w sobie obszerną bibliotekę klas FCL (z ang. *Framework Class Library*) oraz zapewnia kompatybilność dla kilku języków programowania. Programy napisane z wykorzystaniem .NET są wykonywane w środowisku CLR (z ang. *Common Language Runtime*) - jest to maszyna wirtualna, która dostarcza usługi takie jak bezpieczeństwo, zarządzanie pamięcią czy obsługę wyjątków. Cechy platformy .NET:

- Głównym elementem platformy .NET jest Środowisko Uruchomieniowe Wspólnego Języka (z ang. *Common Language Runtime*, CLR). Stanowi ono implementację CLI gwarantując wiele właściwości oraz zachowań w obszarach zarządzania pamięcią bądź bezpieczeństwa. Głównym zadaniem komponentu CLR jest zamiana skompilowanego kodu CIL (z ang. Common Intermediate Language) na kod maszynowy, który jest dostosowany do maszyny, na jakiej został uruchomiony.

– Kompatybilność wsteczna: ponieważ systemy komputerowe bardzo często wymagają interakcji pomiędzy nowszymi i starszymi komponentami, .NET daje możliwość wykonywania funkcji poza platformą. Dostęp do komponentów COM jest możliwy dzięki wykorzystaniu przestrzeni nazw *System.Runtime.InteropServices* oraz *System.EnterpriseServices*.

– W skład platformy .NET wchodzi komponent CTS (z ang. *Common Type System*), który definiuje wszystkie możliwe typy danych wspierane przez CLR oraz w jaki sposób mogą one ze sobą ingerować. Dzięki temu jest możliwa wymiana typów bądz instancji obiektów pomiędzy bibliotekami oraz aplikacjami napisanymi w różnych językach opartych o .NET.

– Przenośność: platforma została zaprojektowana w taki sposób, aby jej implementacja była możliwa dla różnych systemów operacyjnych.

– Bezpieczeństwo: platforma .NET dostarcza wspólny model bezpieczeństwa dla programów tworzonych w jej ramach. Została zaprojektowana w taki sposób, aby uniknąć wielu problemów z bezpieczeństwem aplikacji, do których należy m.in. przepełnienie bufora, które jest bardzo często używane przez złośliwe oprogramowanie (z ang. *malicious software*, w skrócie *malware*).

## 2.9. Accord .NET

Accord .NET jest to szkielet aplikacyjny oparty o środowisko .NET. Zawiera biblioteki implementujące bardzo wiele algorytmów z szerokiej listy dziedzin nauki takich jak:

– **klasyfikacja**: sieci neuronowe, metody wektorów nośnych (z ang. *Support Vector Machine*, w skrócie SVM), algorytm Levenberga-Marquardta, model Markova, tworzenie drzew decyzyjnych

– **regresja**: regularyzacja, regresja liniowa, wielomianowa,

– **analiza skupień** (z ang. *clustering*): algorytm k-średnich, podział binarny.

– **rozkład prawdopodobieństwa**: rozkład normalny, Poissona, Cauchy'ego

– **Przetwarzanie obrazów cyfrowych** (z ang. *digital image processing, DIP*): deskryptory punktów kluczowych - SURF, FREAK, FAST; deskryptory gęstości - HOG, LBP

– **Rozpoznawanie obrazów** (z ang. computer vision): metody do detekcji, śledzenia oraz transformacji obiektów w strumieniu wideo.

Szkielet ten został zaimplementowany, aby rozszerzyć możliwości istniejącego rozwiązania - *AForge .NET*, jednak z czasem oba podmioty zostały ze sobą połączone pod jedną nazwą *Accord .NET*. Framework jest dostępny do pobrania z poziomu kodu źródłowego jak również za pomocą systemu zarządzania pakietami - *NuGet*.

## 2.10. Microsoft Visual Studio

Microsoft Visual Studio to zintegrowane środowisko programistyczne (z ang. *Integrated Deveopment Environment, w skrócie IDE*) dostarczane prze firmę *Microsoft*. Służy do budowania aplikacji konsolowych, jak również stron internetowych, aplikacji webowych oraz mobilnych. Visual Studio używa dostarczanych przez firmę Microsoft platform do tworzenia oprogramowania, do których należą m.in Windows Forms oraz Windows Presentation Foundation. Umożliwia tworzenie oprogramowania w 36 różnych językach programowania, do których należą m.in. C#, C, C++, Visual Basic .NET, JavaScript czy CSS. Obecnie najnowsza wersja to Visual Studio 2017.

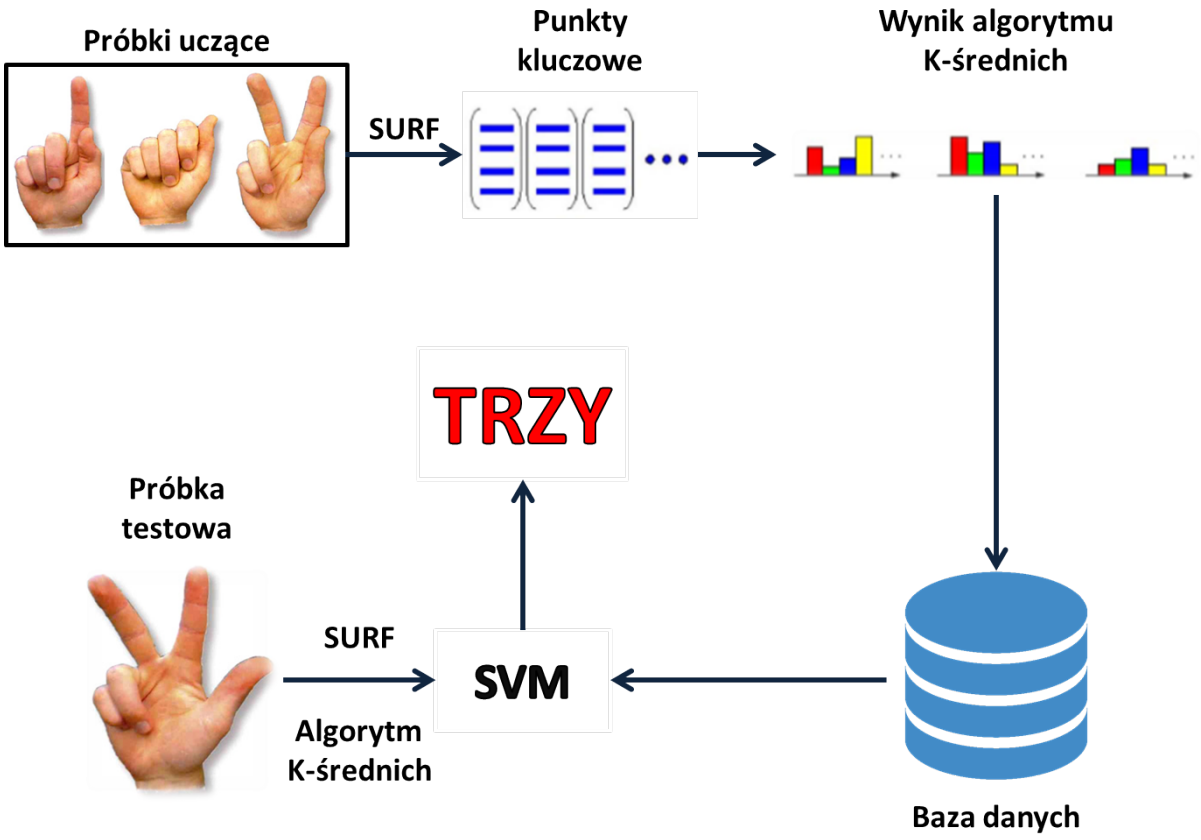
Jednym z najważniejszych elementów wchodzących w skład środowiska jest edytor kodu. Jak każde nowoczesne IDE, Visual Studio zawiera edytor umożliwiający podświetlanie składni oraz autouzupełnianie brakujących fraz wykorzystując mechanizm *IntelliSense*. Środowisko programistyczne wspiera tzw. kompilację w tle. W trakcie pisania kodu, Visual Studio kompiluje kod w tle w celu dostarczenia informacji o składni oraz ewentualnych błędach kompilacji.

Visual Studio zawiera debugger umożliwiający redukcję błędów w oprogramowaniu zarówno dla kodu zarządzanego, jak i natywnego. Dodatkowo istnieje możliwość dopięcia debuggera do procesu w celu monitorowania zachowania. Visual Studio umożliwia dokonywania zrzutów pamięci (z ang. *memory dumps*) jak również wczytywania ich w dalszych momentach debugowania. Debugger umożliwia tworzenie punktu wstrzymania (z ang. *breakpiont*) - miejsca celowego zatrzymania wykonywania programu w celu analizy jego zachowania. Dodatkowo punkty wstrzymania mogą być warunkowe, czyli zatrzymanie w punkcie następuje w momencie spełnienia warunku. Debugger wspiera tzw. *edytuj i kontynuuj* - jak nazwa sugeruje, istnieje możliwość zmiany wartosci zmiennej podczas procesu debugowania.

Visual Studio zawiera narzędzia do tworzenia interfejsu użytkownika. Jeden z przykładów może stanowić narzędzie *Cider* pomocne w budowaniu aplikacji WPF. Umożliwia tworzenie aplikacji poprzez przeciąganie istniejących elementów z przybornika oraz ustawianiu ich własciwości w dedykowanym oknie. Ponadto istnieje możliwość edycji widoku z poziomu języka znaczników XAML. Szczegółowy opis technologii WPF został opisany w podrozdziale 2.6.

# 3. Aplikacja do rozpoznawania gestów

Projekt zakładał utworzenie aplikacji do rozpoznawania gestów w czasie rzeczywistym. Aplikacja została zaimplementowana za pomocą środowiska Visual Studio 2015 tworząc projekt WPF. Jak opisano w podrozdziale 2.6, WPF to silnik graficzny do tworzenia aplikacji okienkowych, gdzie odpowiednie graficzne elementy oraz widoki są konstruowane za pomocą języka znaczników XAML. Język C# posłużył do implementacji *back-end'u* aplikacji, wykorzystano również elementy biblioteki Accord .NET, w skład których należały m.in. funkcje do obsługi kamery oraz filtracji obrazu, algorytm SURF jak również klasyfikator SVM. Dla części widocznej ze strony użytkownika zastosowano zestaw narzędzi o nazwie *MahApps.Metro*, który nadpisuje domyślny styl graficznych elementów w silniku WPF.




**Rys. 3.1.** Schemat działania aplikacji.

Zasada działania aplikacji została przedstawiona na rysunku 3.1. Próbki treningowe wraz z odpowiednimi etykietami są poddawane działaniu algorytmu SURF. Wynik operacji to zbiór punktów kluczowych dla każdego z obrazów. Następnie dokonywane jest grupowanie odpowiednich punktów w znaną z góry liczbę klastrów. Wyniki grupowania lądują w bazie danych, która jest wykorzystywana do utworzenia modelu SVM. Próbka testowa poddawana jest takim samym operacjom co wszystkie próbki uczące - jest ona przekształcana za pomocą algorytmów SURF oraz K-średnich w wektor o znanej liczbie klastrów. Model SVM klasyfikuje próbkę do jednej z podanych kategorii.

W dalszej części rozdziału przedstawiono szczegółową implementację aplikacji. Pierwszy podrozdział zawiera opis wykorzystanych wzorców projektowych. W kolejnych podrozdziałach opisano każdy z widoków dostępnych w aplikacji wraz ze wzajemnymi zależnościami pomiędzy nimi.

## 3.1. Wzorzec MVVM

Wzorzec MVVM (skrót od ang. Model-View-ViewModel) jest jednym z wzorców architektonicznych służący do tworzenia oprogramowania. Daje możliwość pełnej separacji interfejsu użytkownika od modelu aplikacji. Twórcami wzorca są architekci firmy *Microsoft* Ken Cooper oraz Ted Peters, którzy pracowali nad sposobem usprawnienia programowania sterowanego zdarzeniami. Wzorzec ten stał się jednym z komponentów nowo powstałego silnika graficznego WPF. W skład wzorca MVVM wchodzą następujące elementy:

- Model: Reprezentuje prawdziwą zawartość aplikacji lub stanowi warstwę dostępu do danych.
- View (widok): jest to struktura, którą użytkownik widzi na ekranie swojego monitora.
- ViewModel: abstrakcyjna warstwa dla widoku udostępniająca publiczne własności oraz komendy. Za pomocą techniki *data binding* widok wymienia informację z powiązanym z nim ViewModelem. ViewModel odpytuje model w celu pobrania lub zmiany danych. Zastosowanie struktury ViewModelu eliminuje istnienie bezpośredniego połączania pomiędzy widokiem a modelem.

## 3.2. Wzorzec Mediator

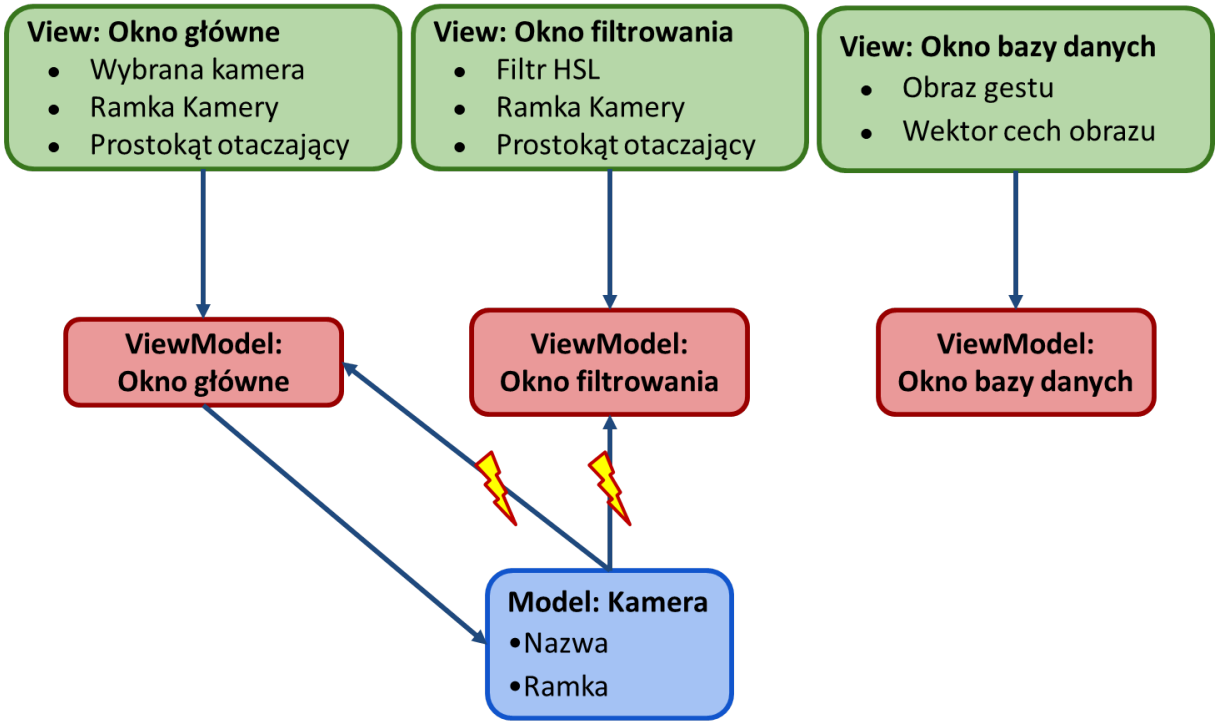
W inżynierii oprogramowania mediator jest wzorcem projektowym należącym do grupy wzorców czynnościowych. Jest jednym z 23 wzorców projektowych opisanych przez tzw. *"Bandę Czworga"* (z ang. *Gang of Four*, GoF) w książce *"Design Patterns: Elements of Reusable Object-Oriented Software"* stanowiącej jeden z kanonów tworzenia oprogramowania. Zastosowanie wzorca powoduje, że komunikacja pomiędzy obiektami jest zawarta w obiekcie mediatora. Obiekty nie komunikują się bezpośrednio ze sobą, lecz wykorzystują mediatora do pośredniej komunikacji, który koordynuje wzajemną interakcję. Dzięki temu następuje redukcja wzajemnych zależności pomiędzy komunikującymi się obiektami. Klasa klienta może wykorzystać mediatora w celu wysłania wiadomości do innych klientów oraz może otrzymać powiadomienie zwrotne poprzez zdarzenie pochodzące od klasy mediatora. Obiekty sa w ten

sposób luźno powiązane pomiędzy sobą, dzięki temu komunikacja jest łatwa w implementacji. Testowanie oraz ponowne wykorzystanie kodu jest możliwe, ponieważ obiekty muszą jedynie odnosić się do mediatora oraz nie posiadają żadnej informacji o innych elementach.

## 3.3. Architektura aplikacji

Architektura aplikacji została przedstawiona na rysunku 3.2. Składa się ona z dwóch klas stanowiących model, trzech widoków oraz z powiązanymi z nimi modelami widoków. W skład modelu wchodzi klasa *Camera* posiadająca flagę informującą o stanie urządzenia (czy jest włączone) oraz właściwości opisujące nazwę kamery oraz bieżącą ramkę. Model ten jest powiązany z dwoma modelami widoków: z oknem głównym oraz z oknem służącym do filtrowania obrazu w przestrzeni barw HSL. Model kamery informuje widoki modelu o zmianie swoich właściwości za pomocą zdarzenia. Zmiana kamery lub jej właściwości jest możliwa jedynie za pomocą głównego okna aplikacji. Kolejną częścią modelu jest klasa powiązana z gestem. Posiada właściwości opisujące nazwę gestu, etykietę oraz zawiera listę wektorów opisujących każdy obraz wchodzący w skład gestu. Model ten jest powiązany z widokiem modelu dla bazy danych, jak również z pomocniczymi widokami do dodawania oraz edycji gestów w bazie.

Wymiana informacji pomiędzy modelami widoków zachodzi z wykorzystaniem opisanego w sekcji 3.2 mediatora. Dzięki niemu elementy aplikacji są ze sobą luźno powiązane i ewentualna zmiana w architekturze aplikacji jest łatwo wykonywalna.




**Rys. 3.2.** Architektura aplikacji. DO ZMIANY

Powiązanie pomiędzy modelem widoku a widokiem wykorzystuje wzorzec projektowy o nazwie *View Model Locator*. Wykorzystanie tego wzorca stanowi standardowy, spójny oraz deklaratywny sposób powiązania widoku z modelem widoku w podejściu *view first*. Zasada łączenia widoku modelu z widokiem można opisać następujacymi krokami:

1. *View Model Locator* sprawdza jaki widok ma zostać stworzony.
2. Wzorzec identyfikuje widok modelu dla tworzonego widoku.
3. *View Model Locator* konstruuje widok modelu.
4. Wysłanie danych ustawionych w widoku do *View Model'u*.

Baza danych jest w postaci pliku XML (skrót on ang. *Extensible Markup Language*). Zawiera ona informacje o wszystkich gestach dostępnych w bazie. Każdy węzeł występujący w pliku posiada dwa atrybuty: nazwę gestu oraz numer etykiety. Wewnątrz każdego węzła znajdują się rekordy, które są mapowane na poszczególne obrazy w obrębie gestu. Pojedynczy rekord posiada dwa atrybuty: nazwę pliku oraz wektor cech opisujący dany gest.

### 3.3.1. Główny widok aplikacji

Na rysunku RYSUNEK DODAĆ przedstawiono widok główny aplikacji do rozpoznawania gestów. W jego skład wchodzą następujące elementy:

– Bieżąca ramka kamery. Jest to element stanowiący największą część głównego okna. Odświeżanie obrazu zachodzi po każdym zdarzeniu wynikającym z pojawienia się nowej ramki w modelu kamery.

– Lista rozwijana, której elementy reprezentują urządzenia wideo podłączone do komputera. Zmiana kamery możliwa jedynie w przypadku, gdy obecnie wybrana kamera jest rozłączona.

– Element umożliwiający zmianę rozdzielczości dla wybranej kamery. Zmiana jest możliwa jedynie w przypadku, gdy wybrana kamera nie rejestruje strumienia wideo.

– Przyciski *CONNECT* oraz *DISCONNECT*. Odpowiadają za start oraz zatrzymanie wybranej kamery. W przypadku, gdy kamera jest uruchomiona, niemożliwe jest użycie przycisku związanego ze startem kamery. Analogiczne zachowanie występuje dla przycisku *DISCONNECT*.

– Przycisk *TAKE SNAPSHOT*. Naciśnięcie przycisku powoduje pobranie prostokąta otaczającego stanowiącego część bieżącej ramki ze strumienia wideo. Prostokąt zostaje wysłany do widoku modelu powiązanego z bazą danych, gdzie zachodzi jego klasyfikacja oraz przypisanie do istniejącego gestu. Działanie jest możliwe wyłącznie wtedy, gdy wybrana kamera jest włączona oraz gdy w bazie danych jest więcej niż jeden gest.

– Przycisk *SET HSL FILTER*. Otwarcie okna, w którym możliwa jest edycja składowych filtra przestrzeni barw HSL. Szczegóły w podrozdziale 3.3.2.

– Przycisk *OPEN DATABASE*. Naciśnięcie przycisku otwiera widok bazy gestów. Szczegółowe informacje o zachowaniu widoku bazy są dostępne w podrozdziale 3.3.3.

### 3.3.2. Okno filtrowania

Okno filtrowania zostało przedstawione na rysunku RYSUNEK. Podobnie jak w przypadku głównego okna, najważniejszy element stanowi obraz z bieżącą ramką kamery. Powyżej ramki znajdują się elementy związane z filtrowaniem przestrzeni barw HSL. Istnieje możliwość zmian zakresu wartości poszczególnej składowej, a wyniki tych operacji sa widoczne na ramce kamery. Wyliczanie obrazu z nałożonym filtrem następuje po każdym zdarzeniu powiązanym z pojawieniem się nowej ramki w modelu kamery. Obraz jest tworzony w taki sposób, że jeżeli dany piksel z oryginalnego obrazu spełnia kryteria nałożonego filtru, to taki piksel zostaje wyświetlony. W przeciwnym wypadku przypisuje mu się wartość koloru czarnego. Dodatkowo na obszarze obrazu zastosowano prostokąt otaczający, który obejmuje największy element znaleziony po etapie filtrowania. Następnie prostokąt zostaje wysłany do widoku głównego i tam wyświetlony na obrazie oryginalnym.

### 3.3.3. Okno bazy danych

Na rysunku RYSUNEK przedstawiono okno bazy danych dla aplikacji rozpoznawania gestów. W obrębie okna można wyodrębnić dwa główne elementy: widok zakładek oraz panel umożliwiający konfigurację bazy danych.

Wszystkie gesty dostępne w bazie są widoczne w widoku zakładek. Nazwy gestów stanowią nagłówki, które użytkownik może przełączać w celu zaciągnięcia informacji o danym geście. Pojedyncza zakładka zawiera listę elementów, a każdy rekord składa się ze zdjęcia gestu, nazwy pliku oraz deskryptora opisującego pojedynczy gest.

Po prawej stronie okna umieszczono panel służący do konfiguracji bazy gestów. Pierwszy element stanowi przycisk umożliwiający dodanie nowego gestu. Po jego kliknięciu zostaje otworzony prosty widok, w którym użytkownik może wczytać pliki zawierające gesty. W przypadku, gdy nazwa wczytywanego gestu istnieje w bazie, obrazy zostaną dodane właśnie do tego elementu bazy danych.

Pierwsza grupa elementów umieszczona poniżej przycisku do dodawania gestów powiązana jest z ekstrakcją cech. W jej skład wchodzi edytowalne pole numeryczne, w którym użytkownik podaje długość wektora cech. Drugi element to przycisk służący do wyliczania deskryptora. Po jego kliknięciu wszystkie obrazy z bazy danych są poddawane działaniu algorytmu SURF, który wylicza ekstrema dla każdego z nich. Aby wszystkie obrazy były opisane przez deskryptor o takiej samej długości, konieczne jest zastosowanie algorytmu k-średnich. Algorytm ten tworzy klastry, których liczba jest podana przez użytkownika.

Kolejna grupa właściwości jest powiązana z ustawieniami klasyfikatora. W aplikacji wykorzystano dostępny w bibliotece Accord.NET klasyfikator rozwiązujący problem klasyfikacji dla wielu kategorii. Użytkownik ma możliwość wyboru pomiędzy metodami tzw. *kernel tricku'u* opisanego w podrozdziale 2.5. W aplikacji zaimplementowano wielomianową niejednorodną funkcję jądra oraz metodę opartą o jądro RBF. Dla metody wielomianowej użytkownik ma możliwość podania stopnia wielomianu oraz stałej, a dla metody opartej o rozkład Gaussa jest możliwa zmiana parametru $\sigma$. Dla obu metod istnieje możliwość konfiguracja dwóch parametrów: parametru dla funkcji kary, która zostaje nałożona w przypadku złej klasyfikacji próbek treningowych oraz tolerancji, której wartość stanowi kryterium stopu podczas uczenia klasyfikatora SVM.

Kolejne dwa przyciski są powiązane z uczeniem modelu SVM oraz do klasyfikacji próbek treningowych. Dla każdego obrazu gestu z widoku zakładek po procesie klasyfikacji zostaje przypisany kolor tła: zielony w przypadku poprawnego rezultatu, czerwony dla złego. Dodatkowo, gdy oba procesy zostaną wykonane, na ekranie wyświetlane są informacje dotyczące czasu trwania klasyfikacji oraz jej dokładności. Są one wykorzystane podczas porównywania działania klasyfikatora dla różnych wartości parametrów.

Ostatni przycisk służy do otwarcia widoku służącego do testowania zbioru próbek. Użytkownik podaje zbiór obrazów oraz specyfikuje nazwę gestu, dla którego zostaje przeprowadzona klasyfikacja. Dodatkowo w widoku dostępny jest przycisk *TEST*, po którym następuje klasyfikacja każdego z obrazu. Możliwość testowania istnieje jedynie w przypadku, gdy przynajmniej jeden obraz znajduje się w widoku. Kolor tła dla każdej próbki wskazuje na poprawność klasyfikatora. Pomimo dobrej klasyfikacji obrazu, nie zostanie on dodany do istniejącego gestu. Konieczne jest użycie przycisku *ADD NEW GESTURE*.

# 4. Testy oraz prezentacja wyników

Testy sprawdzające działanie aplikacji zostały przeprowadzone za pomocą komputera wyposażonego w czterordzeniowy procesor Intel®CORE$^{TM}$i7 oraz pamięć RAM 16GB DDR3. W ramach testów wykorzystano bazę gestów opisaną w pracy BIBLIOGRAFIA. Zawiera ona 900 sekwencji obrazów tworząc 9 różnych klas. Klasy są utworzone na podstawie 3 prostych gestów dłoni, która jest poddawana rotacji oraz zaciskaniu w pięść. Każda z klas zawiera 100 sekwencji obrazów (5 odmiennych warunków oświetleniowych, 10 różnych sekwencji ruchu dłoni w dwóch kierunkach). Każdy z gestów jest umieszczony na ciemnym tle bez dodatkowych elementów.

Na potrzeby pracy z bazy wybrano 4 różne gesty, do których należały: dłoń ze złączonymi oraz rozłączonymi palcami (kolejno *Palm* oraz *Five Fingers*), pięść (*Fist*) oraz dwa złączone palce (*Two Fingers*). Stworzono łącznie 33 sekwencje testowe. Różnią się one względem siebie wartościami poszczególnych parametrów. Pierwszy podział dotyczy ilości klastrów, na jakie podzielona zostanie przestrzeń zmiennych opisujących cechy obrazu. Wybrano 3 wartości: 10, 36 oraz 50. Dla każdej z nich przeprowadzono następujące sekwencje testowe:

1. Wartość współczynnika funkcji kary: 100; Funkcja jądra: wielomianowa; Stopień wielomianu: 1; Stała wielomianu: 1;
   (a) Tolerancja: 0.00001
   (b) Tolerancja: 0.01
   (c) Tolerancja: 10
2. Wartość współczynnika funkcji kary: 100; Tolerancja: 0.00001; Funkcja jądra: wielomianowa;
   (a) Stopień wielomianu: 1; Stała wielomianu: 1;
   (b) Stopień wielomianu: 1; Stała wielomianu: 50;
   (c) Stopień wielomianu: 1; Stała wielomianu: 1000;
   (d) Stopień wielomianu: 3; Stała wielomianu: 1;
   (e) Stopień wielomianu: 10; Stała wielomianu: 1;
3. Wartość współczynnika funkcji kary: 100; Tolerancja: 0.00001; Funkcja jądra: metoda RBF;
   (a) Sigma: 10

(b) Sigma: 20

(c) Sigma: 5

Dla każdego scenariusza zastosowano współczynnik określający skuteczność metody identyfikacji:

$$POPRAWNOŚĆ = \frac{LICZBA\ PRÓBEK\ POPRAWNIE\ SKLASYFIKOWANYCH}{LICZBA\ WSZYSTKICH\ PRÓBEK} * 100\% \quad (4.1)$$

## 4.1. Wyniki testów dla próbek treningowych

Z podanej bazy wybrano próbki stanowiące zestaw treningowy dla zaproponowanej metody identyfikacji gestów dłoni. Reprezentację każdego gestu stanowiło 360 obrazów zawierającą odmienne warunki oświetleniowe oraz różne pozycje podanego gestu. Całą bazę danych stanowiło 1440 obiektów. Wyniki testów dla deskryptora złożonego z 36 elementów przedstawiono w tabeli 4.1.

| Sekwencja | Palm | Fist | Five Fingers | Two Fingers | Razem |
|---|---|---|---|---|---|
| 1a | 88.6% | 94.7% | 87.2% | 98.3% | 92.2% |
| 1b | 88.6% | 94.7% | 87.2% | 98.3% | 92.2% |
| 1c | 100% | 0% | 0% | 0% | 25% |
| 2a | 88.6% | 94.7% | 87.2% | 98.3% | 92.2% |
| 2b | 88.6% | 94.4% | 86.9% | 99.2% | 92.3% |
| 2c | 86.9% | 87.2% | 70.8% | 98.1% | 89.4% |
| 2d | 86.9% | 98.1% | 86.7% | 96.7% | 93.5% |
| 2e | 100% | 0% | 0% | 0% | 25% |
| 3a | 98.9% | 98.9% | 98.6% | 99.7% | 99.1% |
| 3b | 95.3% | 95.6% | 92.8% | 99.4% | 95.8% |
| 3c | 100% | 99.7% | 100% | 100% | 99.9% |

**Tabela 4.1.** Wyniki testów próbek treningowych dla deskryptora złożonego z 36 elementów.

Pierwsza sekcja to testy, w których zmieniano wartość tolerancji. Dla zbyt dużej wartości tego parametru klasyfikator był w stanie zaklasyfikować poprawnie tylko jeden gest. Zmiana parametru pomiędzy *0.00001* a *0.01* nie powodowała żadnych zmian w etapie klasyfikacji.

Druga grupa testów to testy powiązane ze zmianą wartości współczynników dla wielomianowej funkcji jądra. Najlepszy rezultat osiągnięto dla wielomianu stopnia 3 oraz stałej równej 1. W przypadku, gdy stopień wynosił 10, klasyfikacja się nie powiodła. Zmiana wartości stałej wielomianu wnosiła nieznaczne wahania w kontekście poprawności klasyfikacji.

Najlepsze rezultaty osiągnięto w przypadku wykorzystania metody RBF jako funkcji jądra. Dokładność klasyfikacji wynosiła powyżej 95% dla wszystkich przypadków. Scenariusz, w którym wartość

parametru $\sigma$ wynosiła 5, skuteczność klasyfikacji była bliska 100% - tylko jedna próbka została błędnie sklasyfikowana.

Kolejny etap testów obejmował badanie poprawności klasyfikacji dla deskryptora złożonego z 10 elementów. Wyniki przedstawiono w tabeli 4.2.

| **Sekwencja** | **Palm** | **Fist** | **Five Fingers** | **Two Fingers** | **Razem** |
|---|---|---|---|---|---|
| 1a | 81.9% | 87.2% | 85% | 90.3% | 86.1% |
| 1b | 86.9% | 86.4% | 84.4% | 90.1% | 86% |
| 1c | 100% | 0% | 0% | 0% | 25% |
| 2a | 81.9% | 87.2% | 85% | 90.3% | 86.1% |
| 2b | 80.3% | 86.4% | 82.2% | 92.5% | 85.4% |
| 2c | 80.8% | 86.9% | 80.8% | 92.2% | 85.2% |
| 2d | 81.1% | 88.9% | 80.3% | 85.6% | 84% |
| 2e | 100% | 0% | 0% | 0% | 25% |
| 3a | 92.8% | 90.6% | 94.7% | 99.2% | 94.3% |
| 3b | 87.8% | 87.8% | 86.7% | 96.4% | 89.7% |
| 3c | 98.1% | 98.1% | 100% | 99.2% | 98.8% |

**Tabela 4.2.** Wyniki testów próbek treningowych dla deskryptora złożonego z 10 elementów.

Podobnie jak w przypadku deskryptora złożonego z 36 elementów najlepsze wyniki otrzymano dla scenariuszy wykorzystujących metodę RBF jako funkcję jądra. Jednakże każdy rezultat sprawdzający poprawność klasyfikacji był mniej dokładny niż dla przypadku opisanego za pomocą tabeli 4.1.

Ostatni zbiór scenariuszy testowych wykorzystujący próbki treningowe został przeprowadzony dla deskryptora złożonego z 50 elementów. Wyniki zawarto w tabeli 4.3. Podobnie jak dla dwóch powyższych przypadków, metoda RBF zwracała najbardziej zadowalające rezultaty. Dla sytuacji, w której wartość parametru $\sigma$ wynosiła 5 otrzymano 100% poprawność klasyfikacji.

W tabeli 4.4 zestawiono wynikowe rezultaty klasyfikacji dla każdego scenariusza testowego ze względu na liczbę klastrów. Najlepsze rezultaty otrzymano w przypadku deskryptora złożonego z 50 elementów. Wszystkie wyniki są lepsze od dwóch pozostałych przypadków (z wyłączeniem dwóch scenariuszy, w których klasyfikacja nie powiodła się). Najgorsze wyniki to rezultaty otrzymane dla deskryptora zbudowanego z 10 elementów. Wynika to z faktu, że wektor opisujący zbyt mocno generalizuje punkty kluczowe otrzymane z ekstrakcji cech za pomocą algorytmu SURF.

| Sekwencja | Palm | Fist | Five Fingers | Two Fingers | Razem |
|---|---|---|---|---|---|
| 1a | 88.1% | 95.6% | 90% | 99.4% | 93.2% |
| 1b | 87.5% | 95.3% | 90.1% | 90.1% | 99.7% |
| 1c | 100% | 0% | 0% | 0% | 25% |
| 2a | 88.1% | 95.6% | 90% | 99.4% | 93.2% |
| 2b | 88.1% | 95% | 89.7% | 100% | 93.2% |
| 2c | 87.2% | 90% | 84.2% | 95.3% | 89.2% |
| 2d | 91.1% | 99.2% | 90% | 97.8% | 94.5% |
| 2e | 100% | 0% | 0% | 0% | 25% |
| 3a | 99.4% | 99.7% | 99.7% | 100% | 99.7% |
| 3b | 93.3% | 97.5% | 97.8% | 100% | 97.2% |
| 3c | 100% | 100% | 100% | 100% | 100% |

**Tabela 4.3.** Wyniki testów próbek treningowych dla deskryptora złożonego z 50 elementów.

| Sekwencja | 36 klastrów | 10 klastrów | 50 klastrów |
|---|---|---|---|
| 1a | 92.2% | 86.1% | 93.2% |
| 1b | 92.2% | 86% | 99.7% |
| 1c | 25% | 25% | 25% |
| 2a | 92.2% | 86.1% | 93.2% |
| 2b | 92.3% | 85.4% | 93.2% |
| 2c | 89.4% | 85.2% | 89.2% |
| 2d | 93.5% | 84% | 94.5% |
| 2e | 25% | 25% | 25% |
| 3a | 99.1% | 94.3% | 99.7% |
| 3b | 95.8% | 89.7% | 97.2% |
| 3c | 99.9% | 98.8% | 100% |

**Tabela 4.4.** Zestawienie wyników testów próbek treningowych ze względu na liczbę klastrów.

## 4.2. Wyniki dla próbek testowych

Ostatni etap polegał na sprawdzaniu poprawności klasyfikacji próbek testowych. W tym celu wybrano nowe elementy z dostępnej bazy danych. Dla każdego gestu utworzono zestaw próbek składający się z 50 elementów. Wyniki dla odmiennych wartości parametrów umieszczono w tabelach.

Pierwszy przypadek testowy dotyczył sytuacji, w której deskryptor był zbudowany z 36 elementów. Szczegóły w tabeli 4.5. Dla wszystkich scenariuszy testowych skuteczność klasyfikacji była wyższa niż

| Sekwencja | Palm | Fist | Five Fingers | Two Fingers | Razem |
|---|---|---|---|---|---|
| 1a | 61.4% | 99.7% | 74.2% | 99.1% | 83.6% |
| 1b | 61.4% | 99.4% | 74.2% | 99.4% | 83.6% |
| 1c | - | - | - | - | - |
| 2a | 61.4% | 98.9% | 74.2% | 100% | 83.6% |
| 2b | 62.3% | 98.5% | 73.8% | 98.1% | 83.2% |
| 2c | 59.1% | 98.1% | 73.8% | 98.1% | 82.3% |
| 2d | 55.7% | 97.4% | 75.4% | 96.2% | 81.2% |
| 2e | - | - | - | - | - |
| 3a | 77.1% | 98.5% | 76.9% | 98.3% | 87.7% |
| 3b | 63.9% | 97.7% | 70.7% | 98.8% | 82.8% |
| 3c | 67.2% | 96.6% | 84.6% | 86.5% | 83.7% |

**Tabela 4.5.** Wyniki próbek testowych dla deskryptora złożonego z 36 elementów.

80% Najlepszy rezultat otrzymano dla metody RBF jako funkcji jądra z parametrem $\sigma$ równym 10. Najwyższą skuteczność otrzymano dla gestu pięści - poprawność wyższa niż 90%. Najgorzej wypadł gest dłoni - żaden rezultat nie przekroczył 80-cio procentowej poprawności klasyfikacji.

Kolejna zestaw scenariuszy testowych wykonano dla deskryptora złożonego z 10 elementów. Wyniki przedstawiono w tabeli 4.6. Dla większości sytuacji poprawność klasyfikacja mieściła się pomiędzy 60 a 70%. Dla przypadku, w którym wykorzystano metodę wielomianową stopnia 3 oraz dla stałej równej 1 otrzymano poprawność bliską 90%. Najgorsze wyniki klasyfikacji wystąpiły dla gestu pięciu palców. Dla wielu przypadków poprawność wynosiła poniżej 30%.

| Sekwencja | Palm | Fist | Five Fingers | Two Fingers | Razem |
|---|---|---|---|---|---|
| 1a | 88.5% | 99.5% | 27.7% | 50% | 66.4% |
| 1b | 86.9% | 99.1% | 27.7% | 50% | 65.9% |
| 1c | - | - | - | - | - |
| 2a | 88.5% | 99.5% | 27.7% | 50% | 66.4% |
| 2b | 86.9% | 98.5% | 21.5% | 51.2% | 64.5% |
| 2c | 86.9% | 98.5% | 21.5% | 55.8% | 65.7% |
| 2d | 91.8% | 95.6% | 80.3% | 85.6% | 88.3% |
| 2e | - | - | - | - | - |
| 3a | 80.3% | 99.1% | 40.2% | 50% | 67.4% |
| 3b | 77.1% | 97.9% | 32.3% | 51.9% | 64.8% |
| 3c | 83.6% | 93.2% | 53.9% | 46.2% | 69.2% |

**Tabela 4.6.** Wyniki próbek testowych dla deskryptora złożonego z 10 elementów.

Ostatni etap to testy dla deskryptora zbudowanego z 50-ciu elementów. Wyniki umieszczono w tabeli 4.7. Dla każdego scenariusza testowego rezultaty były wyższe niż 85%. Najlepszą poprawność klasyfikacji otrzymano dla metody wykorzystującą funkcję Gaussa z parametrem $\sigma$ równym 20. Skuteczność wynosiła 94% - gesty pięści oraz dwóch palców zostały sklasyfikowane ze 100% poprawnością.

| Sekwencja | Palm | Fist | Five Fingers | Two Fingers | Razem |
|---|---|---|---|---|---|
| 1a | 88.5% | 98.3% | 78.5% | 99.2% | 91.1% |
| 1b | 88.5% | 98.3% | 78.5% | 99.2% | 91.1% |
| 1c | - | - | - | - | - |
| 2a | 88.5% | 98.3% | 78.5% | 99.2% | 91.1% |
| 2b | 88.5% | 98.5% | 78.5% | 97.3% | 90.1% |
| 2c | 88.3% | 97.5% | 64.6% | 98.2% | 87.2% |
| 2c | 80.3% | 97.5% | 84.6% | 94.2% | 89.6% |
| 2e | - | - | - | - | - |
| 3a | 75.4% | 98% | 95.4% | 100% | 92.2% |
| 3b | 83.6% | 100% | 92.3% | 100% | 94% |
| 3c | 75.4% | 94.9% | 96.2% | 71.2% | 84.4% |

**Tabela 4.7.** Wyniki próbek testowych dla deskryptora złożonego z 50 elementów.

## 4.3. Podsumowanie testów

W niniejszej sekcji zestawiono końcowe rezultaty klasyfikacji uwzględniając badany scenariusz, rodzaj próbek oraz długość wektora opisującego obraz. Szczegóły w tabeli 4.8. Najlepsza klasyfikacja uwzględniająca próbki treningowe oraz testowe wystąpiła dla metody wykorzystującej funkcję Gaussa przy parametrze $\sigma$ wynoszącym 20. Poprawność klasyfikacji dla obrazów treningowych wynosiła 97.2%, natomiast dla próbek testowych 94%.

| | Próbki Treningowe | | | Próbki Testowe | | |
|---|---|---|---|---|---|---|
| | **36 klastrów** | **10 klastrów** | **50 klastrów** | **36 klastrów** | **10 klastrów** | **50 klastrów** |
| 1a | 92.2% | 86.1% | 93.2% | 83.6% | 66.4% | 91.1% |
| 1b | 92.2% | 86% | 99.7% | 83.6% | 65.9% | 91.1% |
| 1c | 25% | 25% | 25% | - | - | - |
| 2a | 92.2% | 86.1% | 93.2% | 83.6% | 66.4% | 91.1% |
| 2b | 92.3% | 85.4% | 93.2% | 83.2% | 64.5% | 90.1% |
| 2c | 89.4% | 85.2% | 89.2% | 82.3% | 65.7% | 87.2% |
| 2d | 93.5% | 84% | 94.5% | 81.2% | 88.3% | 89.6% |
| 2e | 25% | 25% | 25% | - | - | - |
| 3a | 99.1% | 94.3% | 99.7% | 87.7% | 67.4% | 92.2% |
| 3b | 95.8% | 89.7% | 97.2% | 82.8% | 64.8% | 94% |
| 3c | 99.9% | 98.8% | 100% | 83.7% | 69.2% | 84.4% |

**Tabela 4.8.** Zestawienie końcowe próbek ze względu na liczbę klastrów.

# 5. Podsumowanie oraz kierunki dalszych prac

W ramach niniejszej pracy magisterskiej zaimplementowano aplikację służącą do rozpoznawania gestów dłoni działającą w czasie rzeczywistym. Oprogramowanie miało na celu rozpoznawanie kilku prostych gestów otrzymanych za pomocą urządzenia do rejestracji strumienia wideo. W pierwszym etapie pracy dokonano analiza istniejących rozwiązań dostępnych na rynku komercyjnym. Okazało się, że nie znaleziono aplikacji, która spełniałaby cele postawione w niniejszej pracy. Kolejny etap to przegląd istniejących metod do rozpoznawania oraz klasyfikacji gestów. Dzięki takiej analizie dokonano wyboru metody służącej do znajdowania fragmentów obrazu o kolorystyce ludzkiej skóry. Zdecydowano się na ekstrakcję takich regionów za pomocą filtracji obrazu w przestrzeni barw HSL. Model ten jest zdecydowanie bardziej odporny na zmiany oświetlenia niż klasyczna przestrzeń RGB. Następnie zaproponowano wykorzystanie algorytmu SURF do detekcji oraz opisu punktów kluczowych obrazu. Metoda ta została wykorzystana w wielu projektach starających się rozwiązać problem opisu cech obrazu za pomocą wektora cech. W niniejszej pracy wykorzystano model SVM jako klasyfikator dla próbek testowych. Sukces klasyfikacji w ogromnej mierze zależał od parametrów wybranych dla modelu SVM.

Działanie zaimplementowanej metody rozpoznawania gestów zaprezentowano w rozdziale 4.

Badano wpływ zmian poprawności działania metody ze względu na różne wartości parametrów, do których należały: długość wektora opisującego obraz, rodzaj metody jądra modelu SVM, jego tolerancję oraz wartość współczynnika funkcji kary. Testy przeprowadzono również ze względu na różne zbiory próbek uczących. Dla bazy danych, w której zdjęcia obejmowały jedynie dłoń, najlepsze rezultaty otrzymano dla ..., gdzie poprawność klasyfikacji wynosiła ... W przypadku zbioru gestów, dla których

Testowanie zaproponowanej metody rozpoznawania gestów przedstawiono w rozdziale 4.

# Bibliografia


[1] *http://www.motionsavvy.com/uni.html*.

[2] *https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/wpf-architecture*.

[3] Lowe D.G. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, wolumen 60, strony 91–110, 2004.

[4] Bay H., Ess A., Tuytelaars T., Van Gool L. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, strony 346–359, 2008.

[5] Dardas N. H., Georganas D. Real-time hand gesture detection and recognition using bag-of-features and support vector machine techniques. IEEE, 2011.

[6] Bretzner L., Laptev I., Lindeberg T. Hand gesture recognition using multi-scale colour features, hierarchical models and particle filtering. *5th IEEE International Conference on Automatic Face and Gesture Recognition*, strony 405–410, 2002.

[7] Viola P., Jones M. Rapid object detection using a boosted cascade of simple features. *Computer Vision and Pattern Recognition*. IEEE, 2001.

[8] Viola P., Jones M. Robust real-time object detection. *Computer Vision and Pattern Recognition*, wolumen 2, strony 137–154, 2004.

[9] McKenna S., Morrison K. A comparison of skin history and trajectory-based representation schemes for the recognition of user-specific gestures. *Pattern Recognition*, wolumen 37, strony 999–1009, 2004.

[10] Lindeberg T. Scale-space for discrete signals. *IEEE Transactions of Pattern Analysis and Machine Intelligence*, wolumen 12, strony 234–254, 1990.