# Spis treści

# 1. Wprowadzenie

## 1.1. Cele pracy

## 1.2. Zawartość pracy

# 2. Technologie i pojęcia wykorzystane w projekcie

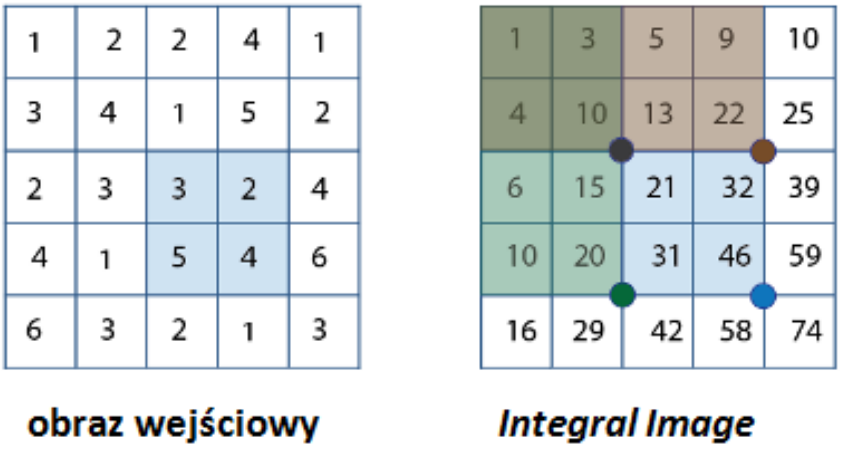
W poniższym rozdziale przedstawiono zagadnienia

## 2.1. Obraz całkowy

Obraz całkowy (z ang. *Integral Image* [8]) to struktura danych wykorzystywana w celu efektywnej i szybkiej generacji sum pikseli dla podanego regionu obrazu. Dowolny piksel *(x, y)* obrazu *I* może zostać przedstawiony jako suma wszystkich pikseli na lewo oraz powyżej *(x, y)*:

$$\textit{Obraz całkowy}(x^{'}, y^{'}) = \sum_{x<x^{'}, y<y^{'}} I(x, y). \tag{2.1}$$

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 2.1




**Rys. 2.1.** Zasada wyliczania obrazu całkowego

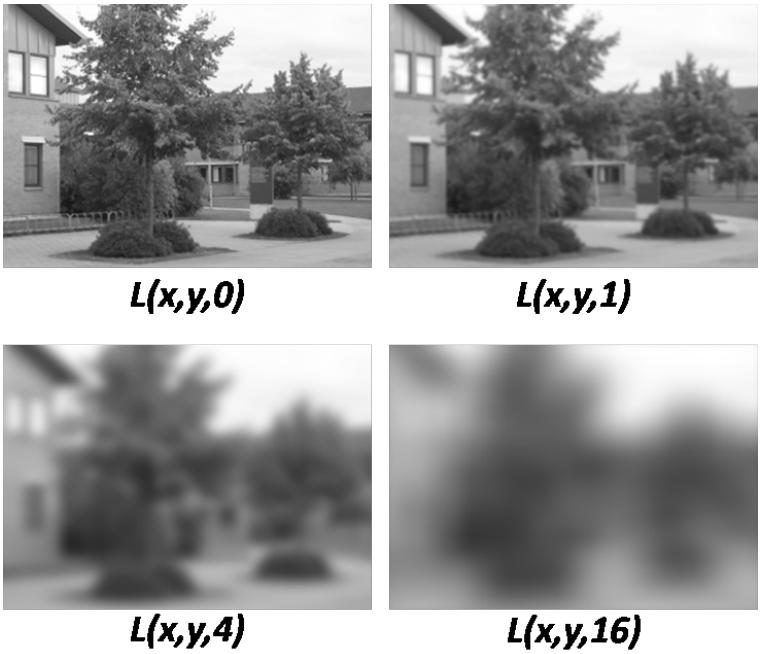
Wyliczenie sumy wyróżnionego regionu na obrazie wejściowym można zastąpić operacjami na obrazie całkowym. Sumę obszaru można uzyskać korzystając z czterech wartości powyżej oraz na lewo

od zaznaczonych kropek: *46 - 22 - 20 + 10 = 14*. Jak nietrudno obliczyć, wynik ten jest równy sumie zaznaczonych elementów obrazu wejściowego.

## 2.2. 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 bardzo ogólne i istnieje wiele reprezentacji przestrzeni skali. Typowym podejściem do zdefiniowania szczególnej reprezentacji przestrzeni skali jest zdefiniowanie zbioru aksjomatów opisujących podstawowe własności szukanej przestrzeni. Najbardziej powszechnym zbiorem aksjomatów jest zbiór definiujący liniową przestrzeń skali powiązaną z funkcją Gaussa.




**Rys. 2.2.** Reprezentacja przestrzeni skali dla różnych wartości $\sigma$

Problem sprowadza się do znalezienia takiego zbioru operatorów $\tau_s$, który operując na obrazie oryginalnym zdefiniuje zbiór obrazów rozmytych:

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

$$L(x, y, \sigma) = g(x, y, \sigma) * I(x, y) \tag{2.2}$$

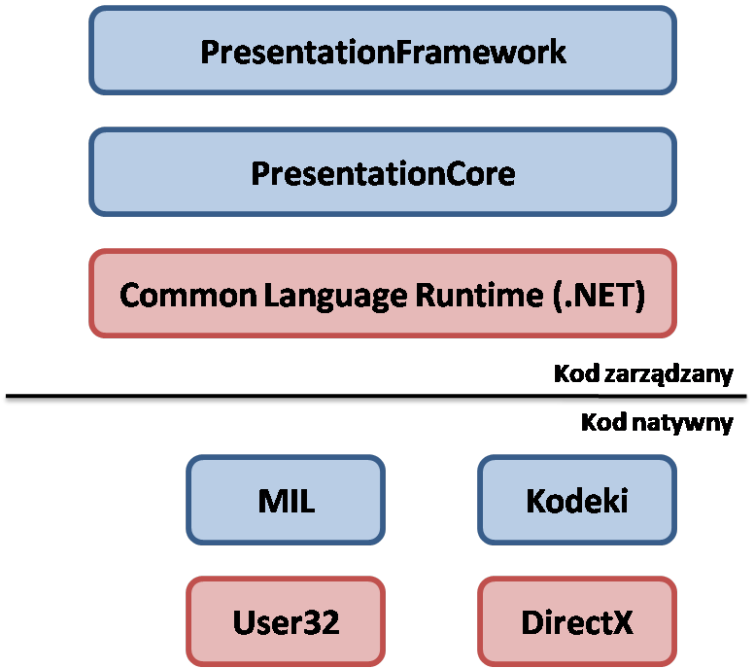
gdzie:

$$g(x, y, \sigma) = \frac{1}{2\pi\sigma} e^{-(x^2+y^2)/2\sigma} \tag{2.3}$$

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 rozmyta, czyli coraz mniej szczegółów przestaje być widoczne. Na rysunku 2.2 przedstawiono przykład tworzenia przestrzennej reprezentacji skali.

## 2.3. Windows Presentation Foundation (WPF)

Windows Presentation Foundation 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 opracyjnego Windows. WPF zbudowany został całkowicie niezależnie do dotychczasowego silnika renderujacego GDI. Dostarcza model programistyczny umożliwiajacy budowanie aplikacji oraz pozwalający na bezwzględną separację logiki biznesowej od interfejsu użytkownika.




**Rys. 2.3.** Architektura WPF. Czerwone elementy to komponenty bibliotek Windows. SKładowe WPF oznaczono kolorem niebieskim.

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 2.3 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 impementuje bazowe usługi dla każdej aplikacji WPF. W skład

tych usług wchodzi przede wszystkim system zarządzania wiadomościami , którego implementację stanowi obiekt typu Dispacher.

– 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 odpowiedzialnych 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. Pelni wiele kluczowych roli tj. uruchomienie aplikacji, zarządzanie pamięcią. Dodatkowo zajmuje się również konwersja języka IL do kodu maszynowego. Elementem bazowym środowiska CLR jest standardowy zestaw typów danych, który jest wykorzystywany przez wszyskie języki programowania oparte o CLR.

Silnik WPF udostępnia system własności dla obiektów, które dziedziczą z DependencyObject. Obiekt ten monitoruje wszytkie zależności pomiędzy własnościami i jest w stanie wykonywać odpowiednie akcje bazujac 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 isniej możliwość definiowania własnych zachowań w celu propagowania informacji o zmianie własności do innych elementów . System zarządzania rozkładem elementów w obrzarze interfejsu użytkownika wykorzystuje powyższy zbior 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 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.

## 2.4. Algorytm SURF

Algorytm SURF (skrót od ang. *Speeded Up Robust Features*) został opatentowany przez grupę naukowców w 2007 roku [BIBLIOGRAFIA]. 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 2.2 technikę utworzenia ciągłej przestrzeni skali opartej na rozkładzie Gaussa.

Działanie algorytmu można podzielić na 3 etapy:

– Detekcja (z ang. *Detection*) – faza automatycznej identyfikcji 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 przeskalowaniu obrazu.

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

W dalszej części rozdziału przedstawiono bardziej dokładną analizę dwóch pierwszych etapów.

### 2.4.1. 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 maksymalny.

Mając do dyspozycji punkt ***x***=*(x,y)* z obrazu całkowego, macierz Hessego *H(**x**,σ)* dla skali $\sigma$ jest zdefiniowana następująco:

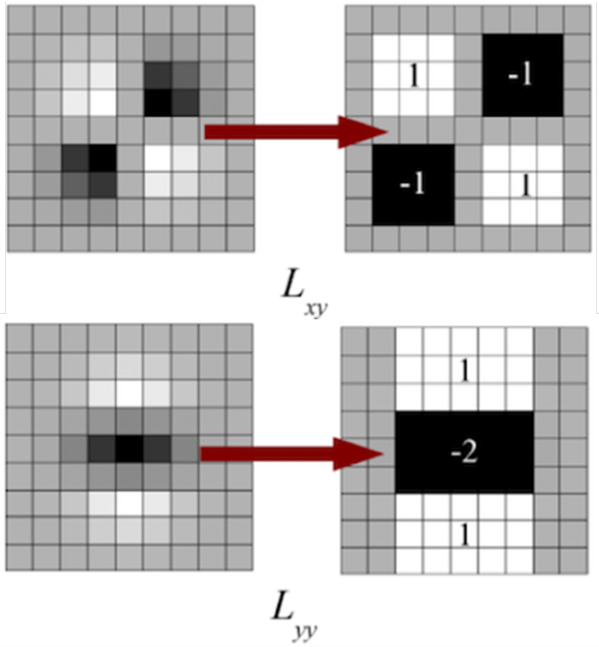
$$H(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix} \tag{2.4}$$

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 BIBLIOGRAFIA, jednakże w zastosowaniach praktycznych wyliczanie splotu jest niezwykle kosztowne obliczeniowo. W celu przyspieszenia obliczeń dokonano aproksymacji drugich pochodnych cząstkowych filtrami przedstawionymi na rysunku 2.4. Dodatkowo wykorzystanie obrazu całkowego powoduje, że czas wyliczania splotów nie zależy od wielkości filtra.




**Rys. 2.4.** 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 [BIBLIOGRAFIA]

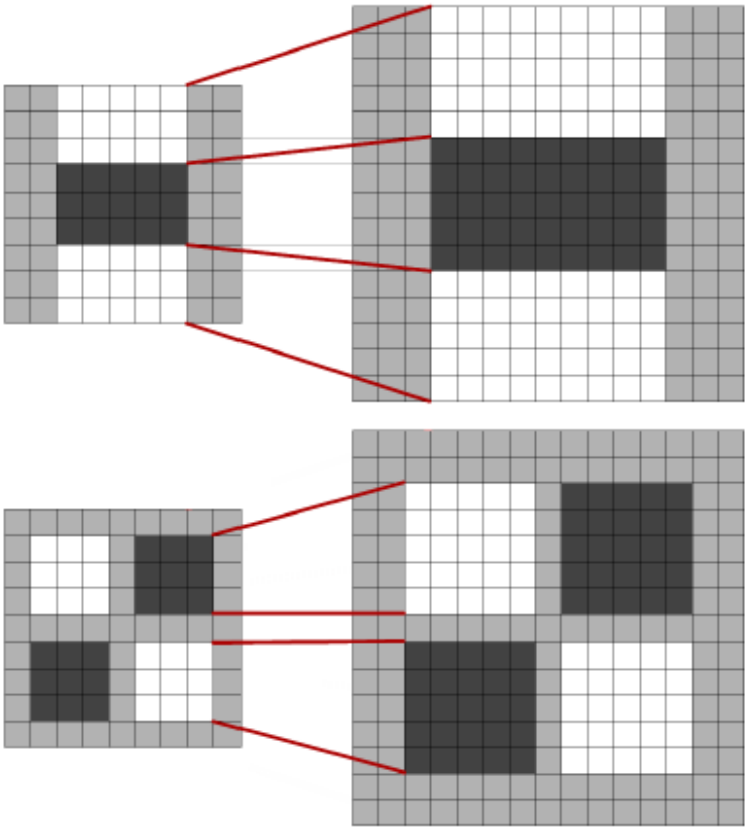
Przedstawione filtry o rozmiarze *9 x 9* odpowiadają splotom, dla których parametr $\sigma$ jest równy 1.2. Jest to najmniejsza wartość skali, dla której algorytm SURF może dawać zadowalające rezultaty.

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 [BIBLIOGRAFIA], 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 podojenie parametru $\sigma$ oraz jest podzielona na stałą liczbę poziomów. Wraz ze wzrostem wielkości filtrów musi zostać spełnione dwa założenia: o istnieniu piksela centralnego oraz o zachowaniu proporcji poszczególnych obszarów maski. W pracy [BIBLIOGRAFIA] opisano szczegółowo, w jaki sposób definiować oktawy oraz liczbę poziomów dla nich. Przykład poprawnego skalowania filtra przedstawiono na rysunku 2.5.

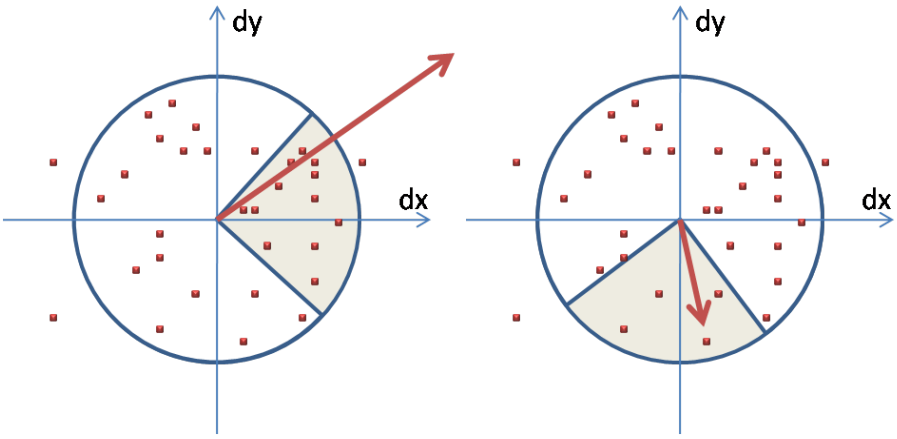


**Rys. 2.5.** Filtr $D_{yy}$ oraz $D_{xy}$ dla dwóch kolejnych poziomów w oktawie (*9 x 9* oraz *15 x 15*). Długość czarnego regionu dla górnego filtra może zostać zwiększona tylko o parzystą liczbę pikseli w celu zagwarantowania istnienia piksela centralnego.

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 [BIBLIOGRAFIA]. Następnie maksima wyznacznika Hessjanu dla poszczególnych skal sa interpolowane na obraz oryginalny.

### 2.4.2. Deskryptor

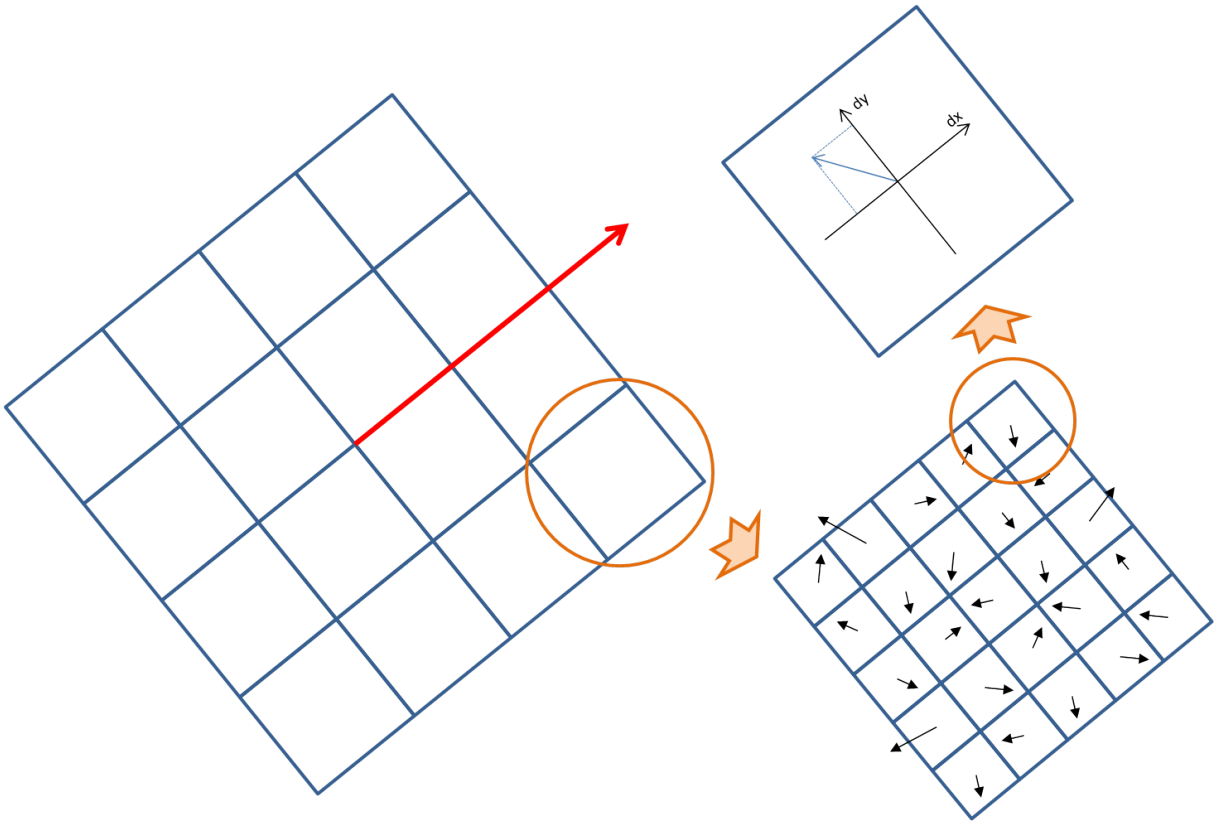
Aby punkt kluczowy był niewrażliwy na zmiany orientacji, punktowi przypisywana zostaje orientacja. W tym celu algorytm SURF wylicza odpowiedź falki Haara dla kolistego otoczenia punktu orientacji. Falka jest wyliczana w kierunku poziomym (*dx*) oraz pionowym (*dy*) dla każdego elementu z otoczenia punktu 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 każdego okna wyliczano sumę wszystkich elementów, a maska zostaje przesunięta. Najdłuższy znaleziony wektor stanowi główną orientację znalezionego punktu kluczowego. Szczegóły na rysunku 2.6.




**Rys. 2.6.** 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 orientacją badanego punktu kluczowego. Schemat przedstawiono na rysunku 2.7. Następnie odpowiedzi *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. 2.7.** Tworzenie deskryptora. Otoczenie punktu kluczowego obrócono zgodnie z orientacją cechy. Dla wszystkich elementów podregionu wyliczono odpowiedź falki Haara w dwóch kierunkach.

MOŻNA DOPISAC O ZNAKU LAPLASJANU I O ROZRZERZENIU DO 128 ELEMENTOW.

## 2.5. 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, 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 zródłowego jak również za pomocą systemu zarządzania pakietami - *NuGet*.

## 2.6. 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,* ). 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 w tworzeniu oprogramowania, w skład którego 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 jest w całości znane podczas kompilacji. Takie działanie jest szeroko wykorzystywane w procesie mapowania obiektowo-relacyjnego (z ang. Object-Relational Mapping, ORM).

## 2.7. .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. W skład architektury wchodzą: MOŻNA COS DOPISAĆ

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. MOŻE RYSUNEK?

– Kompatybilność wsteczna: ponieważ systemu komputerowe bardzo często wymagają interakcji pomiędzy nowszymi i starszymi komponentami, platforma .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*).

## 2.8. Microsoft Visual Studio

Microsoft Visual Studio to zintegrowane środowisko programistyczne (z ang. *Integrated Deveopment Environment, 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 tj: 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 IDE, Visual Studio zawiera edytor umożliwiający podświetlanie składni oraz auto-uzupełnianie 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. Ędytuj 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 znajduje się w podrozdziale 2.3.

## 2.9. 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 wartosci wariancji powoduje, że elementy wewnątrz poszczególnych klastrów są do siebie maksymalnie podobne.

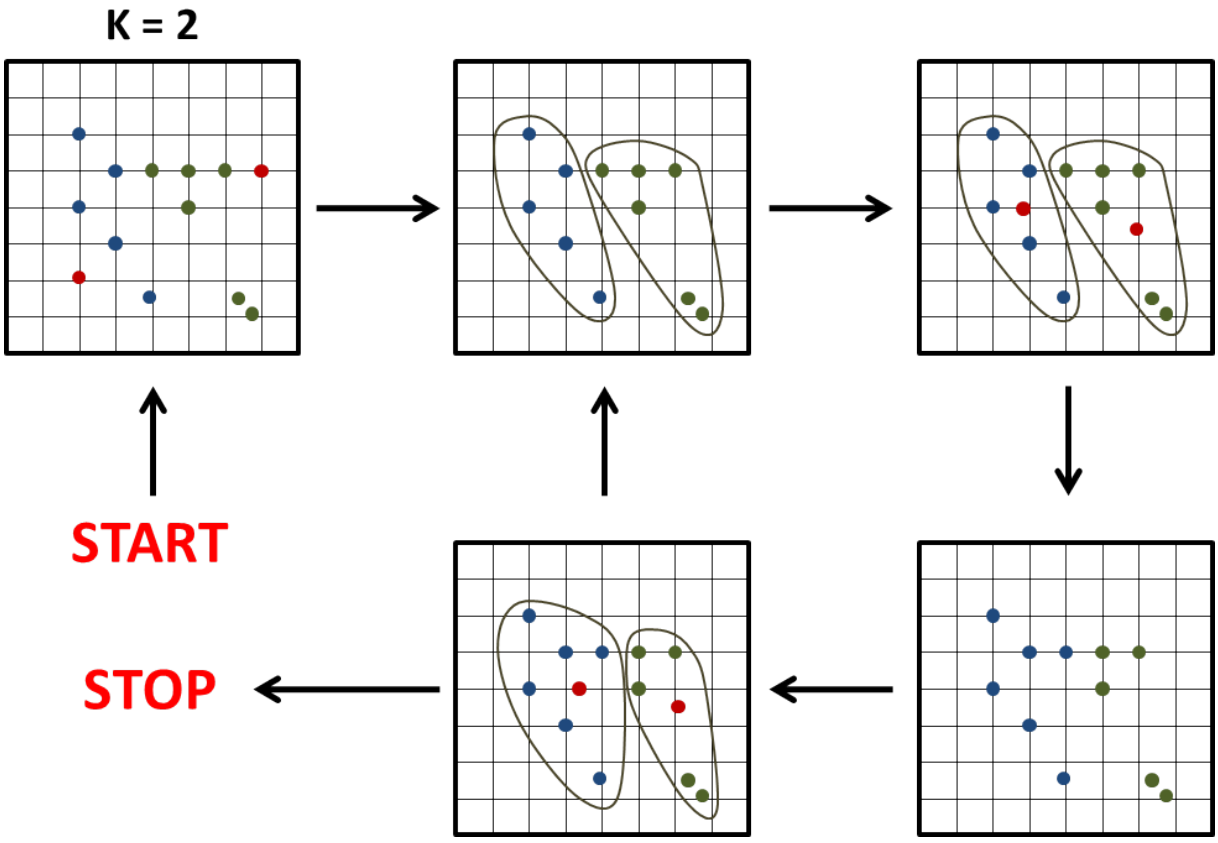
1. Wybór centroidów klastrów: dla podanej z góry liczby klas (skupień) środki klastrów powinny zostać dobrane w taki sposób, aby były możliwie jak najdalej od siebie.

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 2.8 przedstawiono graf przepływu wyjaśniający zasadę działania algorytmu K-średnich.




**Rys. 2.8.** 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.

## 2.10. 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 ***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:

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

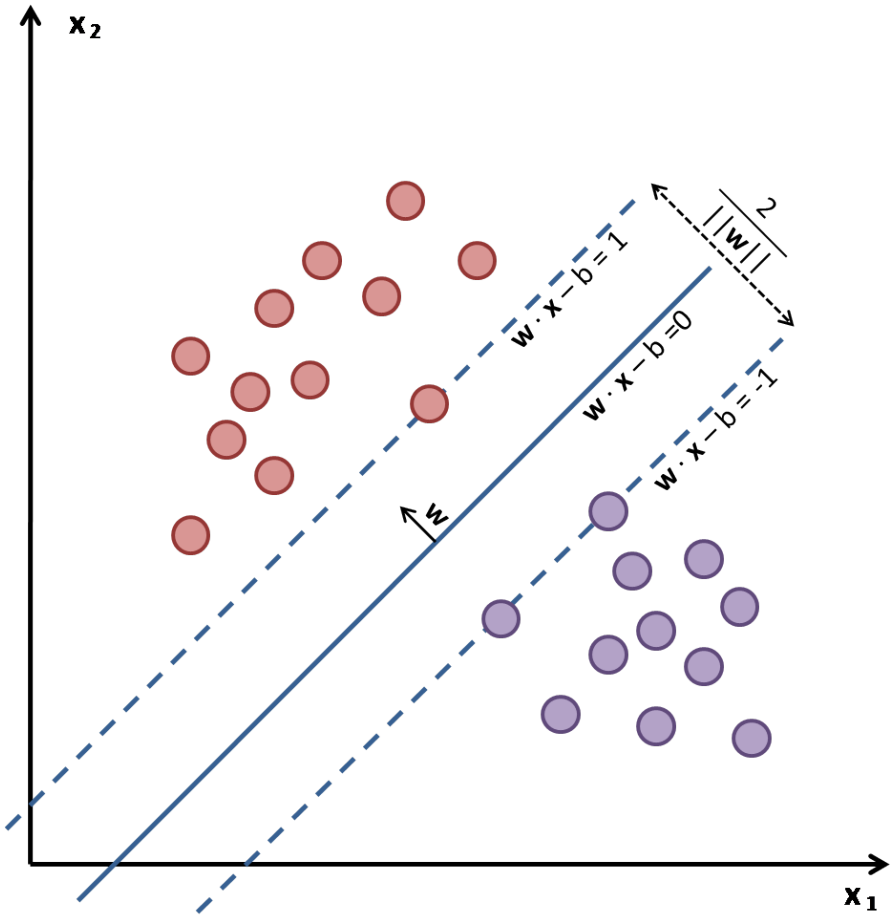
$$\vec{w} \cdot \vec{x}_i - b \leq -1 \quad dla \quad y_i = -1 \tag{2.13}$$

Powyższe nierówności mogą zostać przekształcone do:

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

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 2.9

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:




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

– Wielomianowa jednorodna:

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

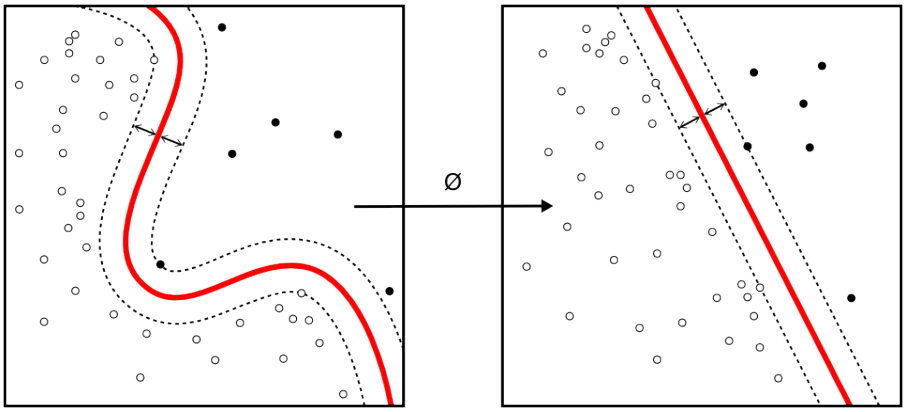
– Wielomianowa niejednorodna:

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

– 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 2.10.




**Rys. 2.10.** Sposób wyznaczania optymalnej hiperpłaszczyzny.

# 3. Opis realizacji aplikacji

Projekt zakładał utworzenie aplikacji SCADA do zarządzania i monitorowania celą spawalniczą. Aplikacja ta została zaimplementowana z użyciem platformy systemowej firmy Wonderware. Z kolei komunikacja pomiędzy aplikacją, a obiektem rzeczywistym została zrealizowana z użyciem protokołu Modbus TCP z udziałem driver-a ... .

## 3.1. Aplikacja SCADA

Aplikacja SCADA pełni główną rolę w monitorowaniu i zarządzaniu celą spawalniczą. Dzięki niej użytkownik ma możliwość zdalnego sterowania całym obiektem, a przede wszystkim robotem spawającym umieszczonym w centralnym punkcie celi. Dodatkowo użytkownik ma dostęp do danych w czasie rzeczywistym, co pozwala mu na dokładne śledzenie pracy robota, szybką reakcję na alarmy, bądź sygnały ostrzegawcze.

Aplikacja SCADA, dedykowana celi spawalniczej, została utworzona w środowisku *Wonderware Application Server* jako projekt o nazwie *Cela*. Komunikację, tworzenie obiektu oraz elementów graficznych opisują poniższe podrozdziały.

### 3.1.1. Komunikacja z robotem

Jednym z kluczowych etapów tworzenia aplikacji było nawiązanie połączenia z robotem Kawasaki BA006Do tego zadania został wykorzystany driver *OI Modbus*, skonfigurowany w programie *System Management Console*, gdzie konfigurację można podzielić na cztery główne kroki.

Pierwszym krokiem było zdefiniowanie typu połączenia, poprzez wybranie z listy dostępnych połączeń, modułu OPC Connection oraz określenie jego nazwy. W projekcie przyjęto nazwę OPC. Rysunek 14 prezentuje parametry konfiguracyjne, które zostały określone dla wybranego modułu.

### 3.1.2. Utworzenie obiektu

Na podstawie szablonu *UserDefined* z biblioteki *Wonderware*, w projekcie został utworzony obiekt o nazwie *Kawasaki*, w którym została zawarta logika zarówno dla samego robota jak i pozostałych elementów celi spawalniczej. Logika ta została zaimplementowana przy użyciu skryptów, które z kolei korzystały z poszczególnych zmiennych umożliwiających komunikację.

#### 3.1.2.1. Definicje atrybutów

Definicja atrybutów polegała na dodaniu sygnałów, które umożliwiły komunikację z obiektem. Atrybuty zostały zdefiniowane pod kątem dwóch grup:

– wejściowe – odczyt wartości sygnałów z robota,

– wyjściowe – zapis wartości sygnałów.

Do sygnałów wejściowych należały:

– I_BasePosition – położenie robota w pozycji bazowej,

– I_Clean – robot w drodze do stacji czyszczącej,

– I_Cycle – praca w cyklu robota,

– I_DoorService – status drzwi serwisowych, OFF – drzwi zamknięte, ON – drzwi otwarte,

– I_ErrorEthIP – błąd komunikacji ze źródłem spawalniczym,

– I_ErrorRobot – błąd wystąpił po stronie robota,

– I_Estop – stan przycisków bezpieczeństwa,

– I_GateClosed – brama zamknięta,

– I_GateOpened – brama otwarta,

– I_HoldRun – status robota $hold$ lub $run$,

– I_jt1, I_jt2, I_jt3, I_jt4, I_jt5, I_jt6 – położenie poszczególnej osi robota,

– I_jt7, I_jt8 – położenie poszczególnej osi manipulatora,

– I_LC_Orange, I_LC_Green, I_LC_Red – odpowiednie kolory wieży sygnalizacyjnej,

– I_Motor – stan silników robota,

– I_OdsRequire – wymagane potwierdzenie zamknięcia drzwi serwisowych,

– I_Ready – robot gotowy do pracy,

– I_Start – rozpoczęcie pracy robota,

– I_Stop – praca robota została przerwana,

– I_TeachLock – zablokowanie trybu uczenia,

– I_TeachMode – tryb uczenia jest aktywny,

– I_WeldingActivation – robot w trybie spawania,

– I_WeldingCurrent – wartość prądu spawania, generowanego przez źródło spawalnicze,

– I_WeldingVoltage – wartość napięcia spawania, generowanego przez źródło spawalnicze,

– I_WeldingWFS – prędkość podawania drutu.

W grupie wyjściowej znalazły się sygnały:

– O_Stop – zatrzymanie pracy robota w cyklu,

– O_OpenServiceDoor – otwarcie drzwi serwisowych,

– O_Motor – uruchomienie motorów robota,

– O_StartCycle – start pracy robota,

– O_gate_open – otwarcie bramy,

– O_gate_close – zamknięcie bramy,

– O_CloseServiceDoor – zamknięcie drzwi serwisowych,

– O_NumberCycle – liczba cykli, po których robot ma pojechać do stacji czyszczącej,

– O_ProgramSet – potwierdzenie wybrania numeru programu,

– O_T_Splash – czas sprysku,

– O_F_Clean – częstość czyszczenia,

– O_L_Cut – długość obcięcia drutu w milimetrach,

– O_ProgramNumber – numer programu jaki ma wykonać robot,

– O_SpeedMonit – prędkość pracy robota,

– O_T_Mill – czas frezowania.

Niektóre z atrybutów, zarówno z grupy wejściowych, jak i wyjściowych zostały powołane do logowania historycznego, poprzez zaznaczenia opcji *History*, podczas ich definiowania **??**.

#### 3.1.2.2. Definicje skryptów

Skrypt jest to zapis instrukcji, które powinien wykonać procesor aby zrealizować pewne określone zadanie. W projekcie można wyróżnić trzy/cztery rodzaje skryptów:

– Skrypty inicjalizacyjne - to skrypty wywoływane jednorazowo podczas uruchomienia aplikacji. W projekcie koniecznym było napisanie skryptu inicjalizacyjnego w celu skojarzenia wcześniej utworzonych atrybutów obiektu z odpowiednimi zmiennymi robota Kawasaki BA006N. Zadanie to wykonane zostało z użyciem formuły: Me.<Nazwa atrybutu>.InputSource = <Nazwa OPC Clienta>.<Nazwa scan group>.<Nazwa OPC Connection>.<Nazwa OPC-Group Connection >.<Nazwa zmiennej> Np. Me.I_BasePosition.InputSource = "OPCClient_001_001.OPC_DeviceGroup.OPC.DeviceGroup.I_BasePosition"

– Skrypty sterujące impulsowe - to skrypty umożliwiające sterowanie robotem poprzez wysyłanie do niego odpowiednich impulsów. Robot, będąc w trybie nasłuchiwania, oczekiwał od aplikacji SCADA instrukcji w postaci impulsów, a gdy tylko wykrył narastające zbocze danego sygnału, od razu uruchamiał odpowiednie procedury. Rysunek **??** przedstawia skrypt z instrukcjami oraz okno ustawień dla atrybutu O_gate_close.

  Instrukcje, które wystąpiły w powyższym skrypcie: *System.Threading.Thread.Sleep(1500)* – odczekuje 1500 ms i przechodzi do następnej instrukcji, *Me.O_gate_close = false* – ustawia sygnał zamknięcia bramy na stan niski. Sygnał w postaci impulsów został również zrealizowany dla atrybutów *O_Stop*, *O_OpenServiceDoor*, *O_Motor*, *O_StartCycle*, *O_gate_open*, *O_CloseServiceDoor* i *O_ProgramSet*.

– Skrypty konwertujące liczbę dziesiętną na ZU2 - to skrypty wysyłające dane całkowite, lecz rozbite na poszczególne bity. Utworzenie tego rodzaju skryptów było konieczne ze względu na narzucone ograniczenia przesyłu danych ze strony robota Kawasaki. Robot ten miał ograniczoną przestrzeń zmiennych, dlatego też, niektóre zmienne zostały zapisane na mniej niż 8 bitach. Rysunek **??** obrazuje przykładowy skrypt wywoływany przy każdorazowej zmianie wartości T_Splash.

  Część kodu znajdująca się w pętli warunkowej *if else* sprawdza zakres wprowadzanej danej, aby nie przekraczała założonych wartości. W dalszym fragmencie kodu następuję konwersja liczby dziesiętnej na liczbę w systemie dwójkowym z dopełnieniem do dwóch. Wprowadzone liczby nie mogły być liczbami ujemnymi zatem wszelkie konwersje dokonywane były jak na rysunku **??**.

  Pozostałe atrybuty, dla których został napisany skrypt sterujący bitowy:

  O_F_Clean - 5 bitów, zakres 1-30,

  O_L_Cut - 5 bitów, zakres 5-15,

  O_ProgramNumber - 8 bitów, zakres 1-100,

  O_SpeedMonit - 8 bitów, zakres 1-100,

  O_T_Mill – 5 bitów, zakres 0-10,

  O_NumberCycle – 5 bitów, zakres 1-10.

– Skrypty konwertujące liczbę w ZU2 na system dziesiętny

#### 3.1.2.3. Obiekty graficzne

Tworzenie obiektów graficznych zrealizowano w środowisku *Application Server*. W celu ich powstania wykorzystano podstawowe elementy z przybornika graficznego oraz animację, które mają na celu „ożywienie" grafiki. W powstałych obiektach zastosowano następujące animacje:

- Visibility – wraz ze zmianą wartości zmiennej, element graficzny znika, bądź się pojawia,
- Value Display - umożliwia zmianę wyświetlanej wartości (informacji) w zależności od stanu atrybutu, od którego wartość jest uzależniona,
- Fill Style – wraz ze zmianą wartości sygnału obiekt zmienia swój kolor,
- Pushbutton - po kliknięciu w komponent graficzny, który zawiera tą animację, wartość powiązanego atrybutu ulega zmianie,
- User Input - z użyciem tej animacji użytkownik jest w stanie wprowadzić wartość dla analogowej zmiennej. Dodatkowo w oknie konfiguracji można ograniczyć zakres w jakim powinna się znajdować wprowadzona wartość,
- Action Scripts - powoduje wywołanie zdefiniowanego skryptu.

##### 3.1.2.3.1 Strona główna

Jednym z elementów graficznych, widocznych podczas uruchomienia aplikacji, był obraz celi spawalniczej z otwartą, bądź zamkniętą bramą. Grafiki z różnym stanem bramy celi spawalniczej, zostały na siebie nałożone, stąd koniecznym było użycie animacji *Visibility*. Animacja ta, w zależności od atrybutu *I_GateOpened* ukazywała użytkownikowi bramę otwartą, bądź zamkniętą. Dodatkowo do obrazu celi dodano opis poszczególnych elementów, takich jak: pozycjoner, robot, źródło spawalnicze oraz ogólne informacje o celi.

##### 3.1.2.3.2 Status

Jednym z podstawowych obiektów graficznych, informujący o statusie pracy robota, aplikacji był komponent złożony z elementów: *Textbox* oraz *Rectangle*. Miał on na celu odwzorowywanie stanu danego sygnału. Wspomniany element został skonfigurowany przez dodanie animacji *Value Display* dla *Textbox*-a oraz animacji *Fill Style* dla *Rectangle*. Poniższe rysunki przedstawiają przykładową wizualizację obiektu oraz jego konfigurację dla atrybutu *I_ErrorRobot*.

##### 3.1.2.3.3 Statusy celi

Na podstawie obiektu graficznego*Status* utworzono grupę, która odzwierciedlała wartości sygnałów płynących od celi spawalniczej **??**.

#### 3.1.2.3.4 Sterowanie bramą i drzwiami serwisowymi

Implementacja elementów graficznych, które odpowiadały za sterowanie obiektem rzeczywistym, było kolejnym etapem do powstania aplikacji. W grupie obiektów sterujących bramą oraz drzwiami serwisowymi wykorzystano *Status* (do którego podpięto odpowiednio sygnał *I_gate_closed* i *I_DoorService*) oraz element *Rectangle*, dla którego zdefiniowano animacje *Pushbutton* **??**.

#### 3.1.2.3.5 Sterowanie stacją czyszczącą

Kolejną grupą, która decydowała o przebiegu procesu spawania, był blok zatytułowany „Stacja czyszcząca", która determinowała jak powinien odbywać się etap czyszczenia palnika. W tym obiekcie graficznym użyto elementów typu *Textbox*, do których dodano animacje *User Input*.

#### 3.1.2.3.6 Sterowanie robotem

Ostatnią grupą, która decydowała o przebiegu spawania, dotyczyła bezpośrednio pracy robota Kawasaki. Można było z niej wybrać numer programu, który robot miał wykonać, a także z jaką prędkością powinien się poruszać. Dodatkowo, znalazły się tam przyciski, które decydowały o starcie i przerwaniu pracy robota.

Wpisana tam instrukcja zmienia wartość atrybutu *InTouch:$Language*, która decyduje w jakim języku powinny pokazać się informację w aplikacji. Dla języka polskiego jest to numer 1045, angielskiego – 1033, a niemieckiego – 1031.

#### 3.1.2.3.7 Symulacja ruchu robota

Kolejny obiekt graficzny odwzorowywał pozycję robota w trzech różnych układach odniesieniach. Odwzorowanie to było możliwe, dzięki wykorzystaniu zadania prostej kinematyki dla odczytanych kątów poszczególnych osi. Dodatkowo poniższy element graficzny wyświetlał pozycję kątową sześciu osi robota oraz dwie osie pozycjonera.

### 3.1.2.4. Wizualizacja aplikacji

Z szablonu *InTouchViewApp* został utworzony obiekt o nazwie *CelaApp*, w którym wykreowano wizualizacje do powstałej aplikacji. Zdefiniowano w nim 8 okien:

- Menu – położenie: 920, 180, wymiary: 1000, 810,
- Header – położenie: 0, 0, wymiary: 1920, 180,
- Footer - położenie: 0, 990, wymiary: 1920, 90,
- Home – położenie: 220, 180, wymiary: 1700, 810,

– Control – położenie: 920, 180, wymiary: 1000, 810,

– Status - położenie: 920, 180, wymiary: 1000, 810,

– Kawasaki – położenie: 920, 180, wymiary: 700, 810,

– WeldingCharts - położenie: 220, 180, wymiary: 1700, 810,

– Web - położenie: 220, 180, wymiary: 1700, 810.

# Bibliografia


[1] Pauwels E., van Gool L., Fiddelaers P., Moons T. : An extended class of scale-invariant and recursive scale space filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, wolumen 17, strony 691–701.

[2] Wang H., Qi H., Xu M., Tang Y., Yao J., Yan X., Li M. Research on the relationship between classic denavit-hartenberg and modified denavit-hartenberg. 2014 Seventh International Symposium on Computational Intelligence and Design, 2014.

[3] Lee J., Bagheri B., Kao H. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. ScienceDirect, 2015.

[4] Qina J., Liua Y., Grosvenora R. A categorical framework of manufacturing for industry 4.0 and beyond. ScienceDirect, 2016.

[5] Tabakow M., Korczak J., Franczyk B. Big data – definicje, wyzwania i technologie informatyczne. *Informatyka ekonomiczna*, strony 138–157, Wrocław, 2014. Uniwersytet Ekonomiczny we Wrocławiu.

[6] Zaczyk M. Kinematyka prosta. materiały wykładowe.

[7] Sishi M. N., Telukdarie A. Implementation of industry 4.0 technologies in the mining industry: A case study. IEEE, 2017.

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

[9] Wittbrodt P., Łapuńka I. Przemysł 4.0 - wyzwanie dla współczesnych przedsiębiorstw produkcyjnych. *Innowacje w zarządzaniu i inżynierii produkcji*, strony 793–799, Opole, 2017. Oficyna Wydawnicza Polskiego Towarzystwa Zarządzania Produkcją.

[10] Jakuszewski R. *Podstawy programowania systemów SCADA*. Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice, 2010.

[11] Iarovyi S., Mohammed W., Lobov A., Ferrer B., Lastra J. Cyber–physical systems for open-knowledge-driven manufacturing execution systems. wolumen 104, strony 1142–1152, 2016.

[12] Finley T. Two's complement., Kwiecień 2000.

[13] Rutkowski T. *Protokół modbus*. Politechnika Gdańska, Gdańsk, 2011.

[14] Stock T., Seliger G. Opportunities of sustainable manufacturing in industry 4.0. *Procedia CIRP*, wolumen 20, strony 536–541, 2016.