Spis treści

1. Wprowadzenie

1.1. Cele pracy

1.2. Zawartość pracy

2. Technologie i pojęcia wykorzystane w projekcie

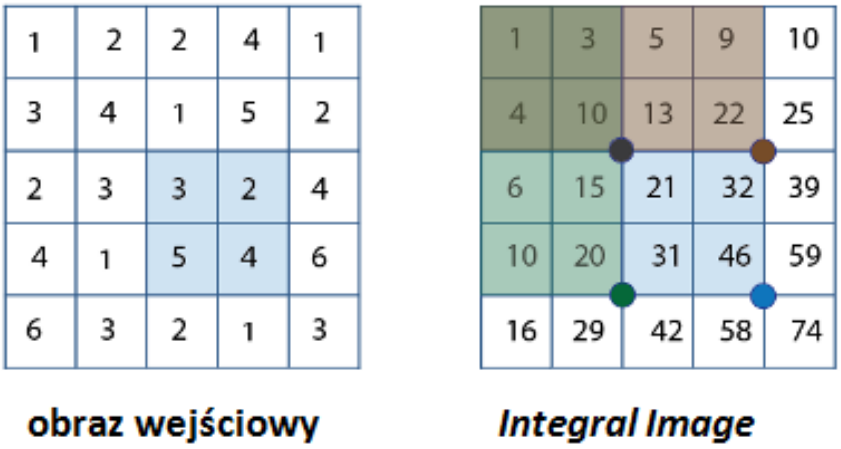
W poniższym rozdziale przedstawiono zagadnienia

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

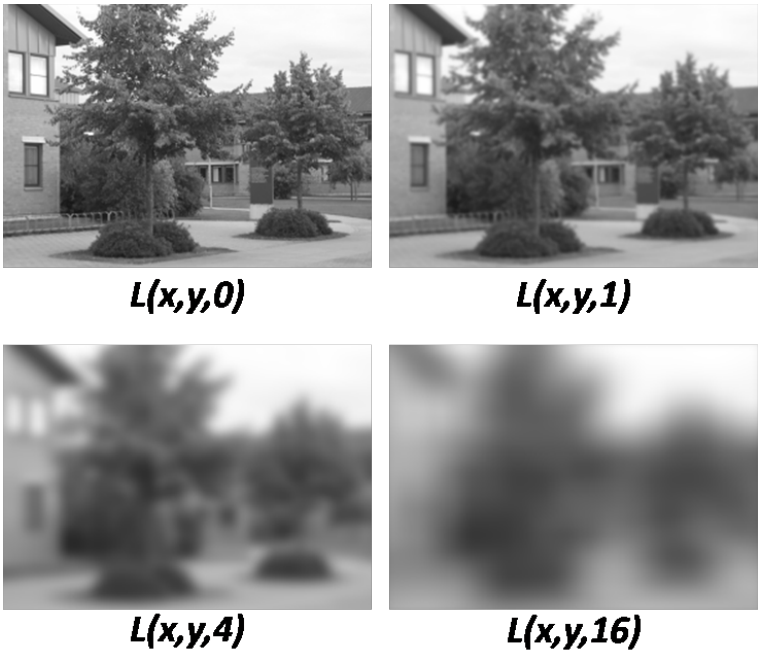
Wyliczenie sumy wyróżnionego regionu na obrazie wejściowym można zastąpić operacjami na obrazie całkowym. Sumę obszaru można uzyskać korzystając z czterech wartości powyżej oraz na lewo

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 σ

Problem sprowadza się do znalezienia takiego zbioru operatorów τ_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)*:

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

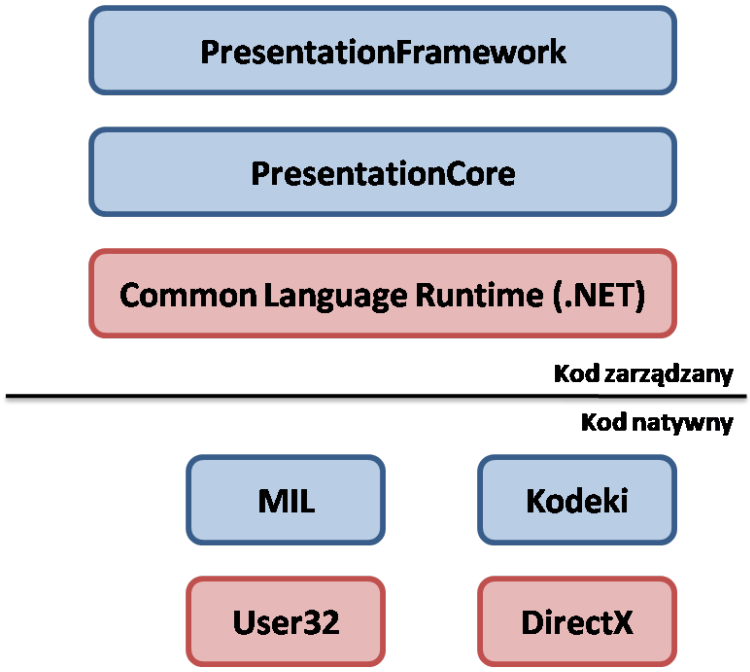
gdzie:

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

Dla $\sigma = 0$ filtr Gaussa staje się funkcją impulsową, zatem $L(x,y,0) = f(x,y)$. Wraz ze zwiększaniem parametru σ 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 σ jest zdefiniowana następująco:

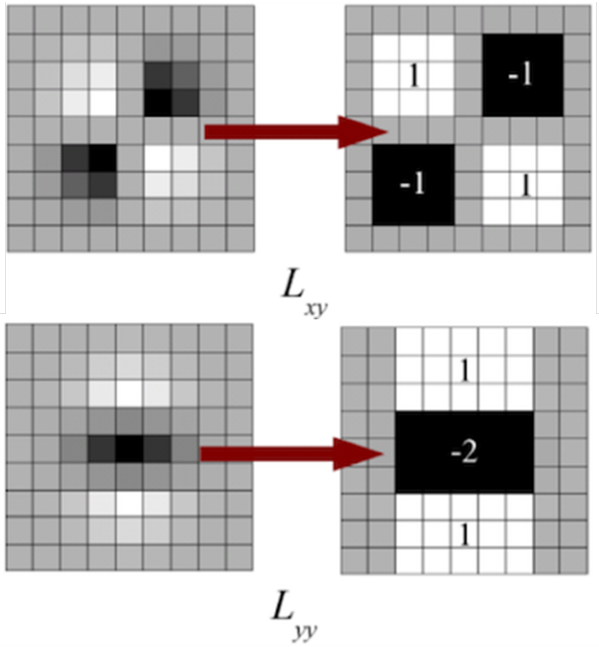
$$H(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix} \tag{2.4}$$

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]

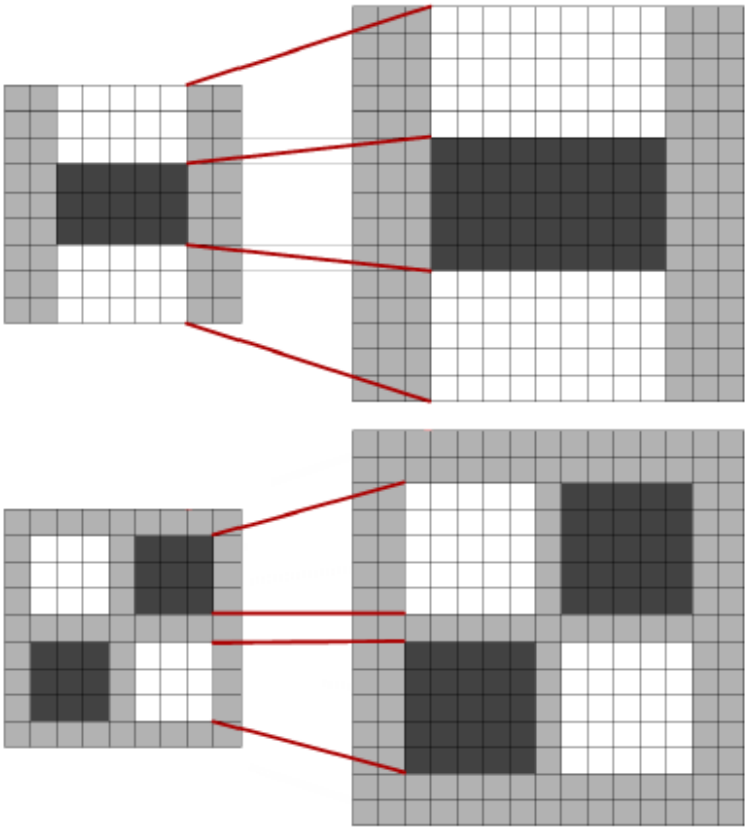
Przedstawione filtry o rozmiarze *9 x 9* odpowiadają splotom, dla których parametr σ jest równy 1.2. Jest to najmniejsza wartość skali, dla której algorytm SURF może dawać zadowalające rezultaty.

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 σ.

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 σ 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.

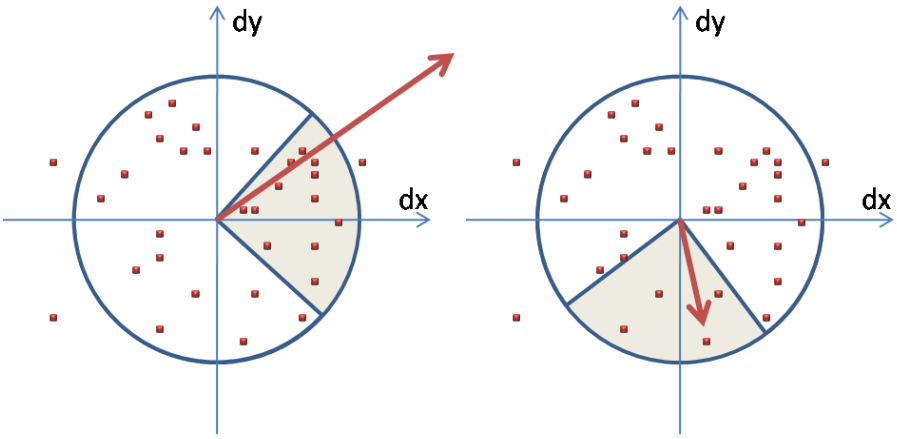


Rys. 2.5. Filtr D_{yy} oraz D_{xy} dla dwóch kolejnych poziomów w oktawie (*9 x 9* oraz *15 x 15*). Długość czarnego regionu dla górnego filtra może zostać zwiększona tylko o parzystą liczbę pikseli w celu zagwarantowania istnienia piksela centralnego.

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

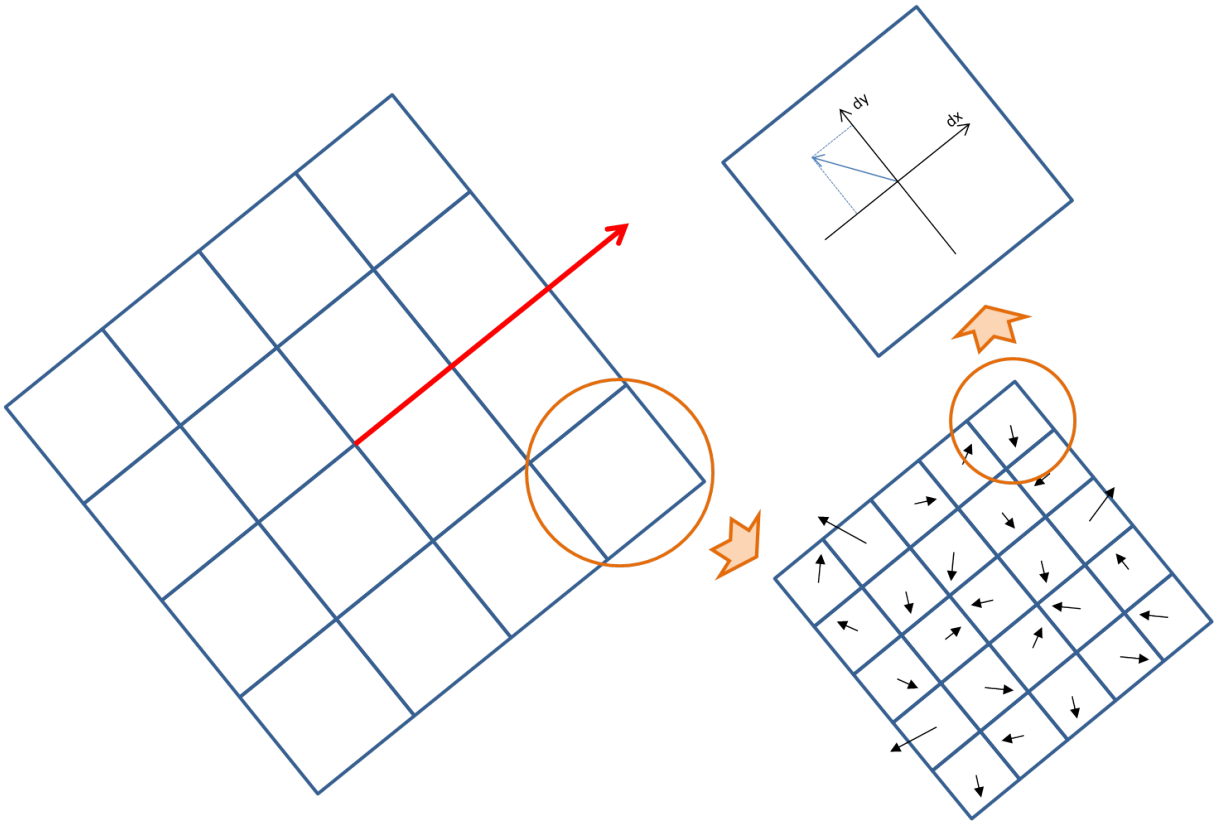
Aby punkt kluczowy był niewrażliwy na zmiany orientacji, punktowi przypisywana zostaje orientacja. W tym celu algorytm SURF wylicza odpowiedź falki Haara dla kolistego otoczenia punktu orientacji. Falka jest wyliczana w kierunku poziomym (*dx*) oraz pionowym (*dy*) dla każdego elementu z otoczenia punktu 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:

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. Systemy SCADA

SCADA (ang. Supervisory Control and Data Acquisition) jest to system, składający się zarówno z oprogramowania jak i części sprzętowej, który pozwala przemysłowym organizacjom na usprawnienie procesu produkcyjnego. Usprawnienie to polega na możliwości kontrolowania procesu lokalnie oraz ze zdalnych lokalizacji, monitorowania i zbierania danych procesu w czasie rzeczywistym, bezpośredniej

interakcji pomiędzy urządzeniami a człowiekiem, a także rejestrowanie zdarzeń w formie plików tekstowych.

Systemy te obejmują swoim działaniem większą część produkcji, od kilku stanowisk, aż po kompletny proces. Umożliwiają pełny monitoring procesu w postaci wizualizacji poszczególnych etapów produkcji. Integracja systemów SCADA z systemami sterowania (np. ze sterownikami PLC lub urządzeniami RTU) oraz urządzeniami pomiarowymi i wykonawczymi pozwala na sterowanie elementami procesu. Przyczynia się to do minimalizacji czasu przestoju występującego na produkcji, poprawnego eksploatowania maszyn, przez co maleje prawdopodobieństwo awarii.

Wykorzystanie systemów SCADA pozwala na szybki i przejrzysty wgląd w rzeczywisty stan urządzeń produkcyjnych i wykonawczych. Umożliwia także nie tylko zmianę języka maszyn na język ludzki, ale również na podstawowe rejestrowanie danych, szybką lokalizację awarii, czy też automatyczną reakcję na określone zdarzenie.

W opisywanych systemach można zdefiniować również algorytmy postępowania oraz tzw. receptury, które przyśpieszają i wspomagają pracę operatora. Dodatkowym mechanizmem jest logowanie historyczne, które gromadzi dane na serwerze, następnie umożliwia szybką i intuicyjną analizę procesów za pomocą przygotowanych raportów. Całość tych narzędzi przyczynia się do optymalizacji procesu produkcyjnego.

Systemy SCADA znajdują zastosowanie w organizacjach przemysłowych i w firmach obejmujących sektory publiczne i prywatne. Celem zastosowania systemów tego typu jest kontrolowanie i utrzymywanie wydajnego procesu, a także podejmowanie lepszych decyzji w celu uniknięcia awarii lub skrócenia czasu przestoju. Systemy SCADA obejmują zarówno proste konfiguracje, jak również te złożone, dzięki czemu sprawdzają się w różnych przedsiębiorstwach, np:

– energetyka,

– wod-kan (wodociągi i kanalizacja),

– branża spożywcza,

– wydobywanie oleju i gazu.

Współczesne systemy SCADA umożliwiają dostęp do danych w czasie rzeczywistym z dowolnego miejsca na świecie. Taki dostęp pozwala na podejmowanie decyzji o dalszym przebiegu procesu z dala od hali produkcyjnej. Dodatkowo nowoczesne aplikacje projektowe pozwalają na szybkie programowanie (ang. RAD - rapid application development), które umożliwia użytkownikowi stosunkowo łatwą implementacje aplikacji.

W ostatnich latach wprowadzono w systemach SCADA nowoczesne standardy i praktyki IT (ang. information technology), w myśl koncepcji Przemysłu 4.0, które poprawiły wydajność, bezpieczeństwo i niezawodność opisywanego systemu. Jednym z unowocześnień jest wykorzystanie Big Data, które pozwoliło na usprawnienie podstawowego zadania systemów SCADA, tzn. archiwizacje danych. Narzędzie, które jest odpowiedzialne za tą funkcjonalność to tzw. Historian. Większość systemów SCADA

jest wyposażone w to narzędzie, które wraz z rozwojem technologii także się zmieniło. Do kluczowych zmian zalicza się: przetwarzanie danych w czasie rzeczywistym, predykcję, wspieranie systemów rozproszonych. Dodatkowo użycie relacyjnych baz danych w Historian-ach ułatwiło integrację SCADA z systemami MES i ERP.

	Tradycyjny system SCADA	Nowoczesny system SCADA
Zastosowanie	Zdalne monitorowanie i zbieranie danych	Poprawa ogólnej efektywności wyposażania (OEE - ang. Overall Equipment Effectiveness) oraz przeglądu całego procesu, w celu zarządzania procesem długoterminowo
Sterowanie	Sterowanie nadrzędne	Zoptymalizowane i proaktywne sterowanie
Raporty	Wykresy trendów historycznych	Analityka dla nowych modeli biznesowych
Alarmy	Alarmowanie centrali operacyjnej	Alarmowanie wielu urządzeń (wliczając urządzenia mobilne)

Tabela 2.1. Porównianie tradycjnych i nowoczesnych systemów SCADA [**?**].

W tabeli 2.1 przedstawiono różnice pomiędzy tradycyjnymi, a nowoczesnymi systemami SCADA.

3. Technologie i pojęcia wykorzystane w projekcie

Protokół Modbus, opisany poniżej, został wykorzystany do wymiany danych między robotem Kawasaki BA006 i aplikacją SCADA. Dodatkowo w poniższym rozdziale przedstawiono sposób kodowania robota oraz wyznaczenie jego położenia.

3.1. Protokół Modbus

Protokół Modbus został stworzony przez firmę Modicon i opublikowany w 1979 roku. Do dnia dzisiejszego jest on wykorzystywany, szczególnie w aplikacjach przemysłowych do komunikacji pomiędzy urządzeniami elektronicznymi. Standard Modbus definiuje protokół siódmej warstwy modelu OSI in. warstwy aplikacji, który zapewnia komunikację typu klient - serwer pomiędzy urządzeniami mogącymi znajdować się w różnych sieciach. Modbus określa również specjalny protokół dla łącza szeregowego, który umożliwia wymianę żądań wysyłanych pomiędzy urządzeniem typu master, a jednym lub kilkoma urządzeniami typu slave. Specyfikacja ta powoduje, że Modbus klasyfikowany jest również do drugiej warstwy modelu OSI, czyli warstwy łącza danych (rys. **??**).

3.1.1. Komunikacja typu Master-Slave

Modbus jest protokołem typu Master-Slave. Oznacza to, że w danym czasie do jednej magistrali szeregowej może być podłączony tylko jedno urządzenie typu master oraz klika (maksymalnie 247) urządzeń typu slave. Komunikację zawsze rozpoczyna węzeł typu master. Urządzenia typu slave nigdy nie wysyłają danych bez uprzedniej prośby od węzła typu master, a także nie komunikują się bezpośrednio z innymi urządzeniami typu slave.

Węzeł typu master może wysyłać żądania do węzłów typu slave w dwóch trybach:

- pojedynczej emisji (ang. unicast mode) - master wysyła żądanie do jednego węzła typu slave. Ten z kolei po odebraniu i przetworzeniu żądania wysyła wiadomość zwrotną do mastera. Do tego typu transmisji danych koniecznym jest, aby każde z urządzeń typu slave posiadał swój unikalny adres.

- zbiorowej emisji - (ang. broadcast mode) - master może wysłać żądanie do wszystkich urządzeń typu slave równocześnie. W tym trybie przepływ informacji jest w jedną stronę, to znaczy, że master nie oczekuje na odpowiedzi od węzłów, do których wysłał wiadomość.

3.1.2. Zasady adresacji

Przestrzeń adresowa protokołu Modbus składa się z 256 różnych adresów (tab. 3.1). Adres 0 jest zarezerwowany do przesyłu wiadomości w trybie broadcast. Wymagane jest, aby wszystkie urządzenia typu slave rozpoznawały adres 0. Każde z urządzeń slave posiada swój unikalny adres, natomiast master nie ma przypisanego żadnego adresu.

3.1.3. Opis pojedynczej ramki

Dla warstwy aplikacji modelu OSI protokół Modbus definiuje przesyłaną wiadomość in. ramkę, jako pojedynczą jednostkę danych (ang. Protocol Data Unit PDU). Ramka ta jest niezależna od niższych warstw komunikacyjnych i składa się z kodu funkcji (ang. Function code) oraz danych (ang. Data), co przedstawia rysunek **??**.

Jednakże chcąc użyć protokołu Modbus dla warstwy łącza danych należy zdefiniować dodatkowo pola służące do komunikacji czyli pole adresowe (ang. Address field) oraz pole sumy kontrolnej CRC (ang. Error checking), które przedstawia rysunek **??**.

Poszczególne pola zawierają informację o:

- pole adresowe (Address field) - przechowuje adres węzła typu slave,
- pole kodu funkcji (Function code) - niesie informację o typie akcji, która jest aktualnie wykonywana,
- pole danych (Data) - zawiera parametry żądania lub odpowiedzi,
- pole sumy kontrolnej (CRC) - zawiera informację o błędach, które ewentualnie wystąpiły.

3.1.4. Modbus TCP/IP

Modbus TCP/IP jest to po prostu protokół Modbus używający interfejsu TCP oraz łącza Ethernet do przesyłu danych. Struktura komunikacyjna Modbus jest protokołem aplikacji, który definiuje zasady organizowania i interpretowania danych niezależnie od medium transmisyjnego. Z kolei TCP/IP zapewnia medium transmisyjne do przesyłania bloków danych binarnych pomiędzy komputerami. Podstawową funkcją TCP jest zapewnienie, aby wszystkie pakiety danych zostały poprawnie otrzymane, podczas gdy IP gwarantuje, że wiadomości są poprawnie adresowane i przekierowywane. W skrócie, Modbus TCP/IP jest to komunikacja standardu Modbus opakowana w Ethernet TCP/IP, o czym świadczy budowa jej pojedynczej ramki (rys. **??**)

Ramka Modbus TCP/IP składa się z niezmienionych dwóch pól protokołu Modbus czyli kodu funkcji oraz danych, a także z pól pochodzących od standardu TCP/IP:

- pole identyfikatora transakcji (Transaction identifier) - zawiera identyfikator pozwalający na rozróżnienie od siebie wiadomości, podczas gdy są one wysłane w tym samym czasie poprzez jedno łącze TCP,

– pole identyfikatora protokołu (Protocol identifier) - w przypadku protokołu Modbus wynosi ono zawsze 0,

– pole długości (Length Field) - zawiera informację o pozostałych bajtach w następujących po nim polach,

– pole identyfikatora jednostki (Unit ID) - używane do rozróżniania zdalnych serwerów zlokalizowanych poza siecią TCP/IP.

3.2. Kod uzupełnień do dwóch

Kod uzupełnień do dwóch (w skrócie U2 lub ZU2) to system reprezentacji liczb całkowitych w dwójkowym systemie pozycyjnym. To najpopularniejszy obecnie sposób zapisu liczb całkowitych, ponieważ operacje dodawania i odejmowania są w nim wykonywane tak samo jak dla liczb binarnych bez znaku. Dzięki tej zależności, procesor jest obciążany mniejszą ilością rozkazów, niż przy użyciu innych kodowań.

Nazwa systemu pochodzi od sposobu obliczania liczb przeciwnych. Wartość przeciwna dla liczby jednobitowej obliczana jest poprzez odjęcie danej liczby od 2. Dla n-bitowych, liczb wartości przeciwne uzyskuje się w analogiczny sposób tzn. odejmując wartość liczby od dwukrotnej wagi najstarszego bitu ($2 \cdot 2^{n-1} = 2^n$).

W systemie U2 istnieje tylko jedno zero, co niewątpliwe jest zaletą, mimo niesymetryczności przedziału kodowanych liczb. W opisywanym systemie kodowania na n bitach można zapisać liczby z zakresu:

$$[-2^{n-1}, 2^{n-1} - 1]$$

Konwersje liczb (w przykładzie zamieniono liczbę 50 i -50) z systemu dziesiętnego na systemem U2 (zapisany na 8-bitach) należy przeprowadzić w następujący sposób:

– **Liczba dodatnia** - liczbę należy zamienić na system dwójkowy

$$50 = (110010)_2.$$

Następnie z lewej strony należy dopełnić liczbę zerami, aby w sumie otrzymać osiem bitów

$$50 = (00110010)_{U2}.$$

– **Liczba ujemna** - w pierwszym kroku należy wyznaczyć wartość bezwzględną konwertowanej liczby:

$$|-50| = 50$$

Następnie postąpić analogicznie jak z liczbą dodatnią otrzymując wartość bezwzględną liczby w systemie U2

$$50 = (00110010)_{U2}.$$

W kolejnym kroku należy zanegować wszystkie bity

$$(00110010)_{U2} \cong 11001101.$$

Ostatnim etapem konwersji liczby ujemnej jest zwiększenie otrzymanego wyniku o 1:

$$11001101 + 1 = 11001110$$
$$-50 = (11001110)_{U2}$$

Poniżej zaprezentowano konwersje w przeciwnym kierunku. W tym przypadku jest jeden algorytm postępowania dla liczb dodatnich i ujemnych.

$$(11001110)_{U2} = -1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = -128 + 78 = -50$$

3.3. Proste zadanie kinematyki

Proste zadanie kinematyki w robotyce jest definiowane jako przeliczanie współrzędnych złączowych na współrzędne kartezjańskie w celu znalezienia położenia i orientacji końcówki robota.

Do rozwiązania prostego zadania kinematyki w bazowym układzie współrzędnych $X_0Y_0Z_0$ stosuje się dwie składowe:

- pozycja - opisana jest przez wektor d_i, w którym znajdują się współrzędne x, y, z początku lokalnego układu $X_iY_iZ_i$ względem układu bazowego,
- orientacja - wyrażona przez macierz orientacji R_i, która opisuje obrót układu $X_iY_iZ_i$ względem układu bazowego. Opis ten wyrażony jest za pomocą rzutów wektorów jednostkowych osi układu lokalnego na wektory osi układu bazowego.

Na rysunku **??** zaprezentowano punkt P, którego położenie można wyrazić za pomocą wektora p_0 zapisanego jako sumę dwóch wektorów znajdujących się w tym samym układzie współrzędnych.

$$p_0 = R_0^1 \cdot p_1 + d_0^1, \tag{3.1}$$

gdzie:

R_0^1 – macierz, która przelicza współrzędne punktu P z układu „1” na układ „0”,

p_1 – wektor opisujący położenie punktu P w układzie „1”,

d_0^1 – wektor początku układu „1” w układzie „0”.

3.3.1. Macierz orientacji oraz macierz przekształcenia

Na podstawie wyprowadzenia [6] otrzymano ogólną postać macierzy orientacji, która składa się z iloczynów skalarnych wektorów.

$$R_0^1 = \begin{bmatrix} i_1 i_0 & j_1 i_0 & k_1 i_0 \\ i_1 j_0 & j_1 j_0 & k_1 j_0 \\ i_1 k_0 & j_1 k_0 & k_1 k_0 \end{bmatrix} \tag{3.2}$$

– i_0, j_0, k_0 - wektory jednostkowe osi w układzie „0”,

– i_1, j_1, k_1 - wektory jednostkowe osi w układzie „1”.

Wykorzystując (3.2) oraz tzw. kąty RPY (ang. Roll, Pitch, Yard), które opisują obrót bryły sztywnej w trzech elementarnych płaszczyznach: obrót (roll), nachylenie (pitch), odchylenie (yaw), wyznacza się pełną macierz orientacji.

$$R = R_{Z,\phi} R_{Y,\vartheta} R_{X,\psi} = \begin{bmatrix} \cos\phi\cos\vartheta & \cos\phi\sin\vartheta\sin\psi - \sin\phi\cos\psi & \cos\phi\sin\vartheta\cos\psi + \sin\phi\sin\psi \\ \sin\phi\cos\vartheta & \sin\phi\sin\vartheta\sin\psi + \cos\phi\cos\psi & \sin\phi\sin\vartheta\cos\psi - \cos\phi\sin\psi \\ -\sin\vartheta & \cos\vartheta\sin\psi & \cos\vartheta\sin\psi \end{bmatrix} \tag{3.3}$$

W celu ujednolicenia zapisu i ułatwienia operacji matematycznych wprowadzona została tzw. macierz przekształcenia T opisująca przekształcenie jednego układu współrzędnych w drugi. Macierz ta składa się z wektora pozycji d oraz macierzy orientacji R.

$$T = \begin{bmatrix} R & d \\ 0 & 1 \end{bmatrix} \tag{3.4}$$

3.3.2. Kinematyka prosta

Równanie kinematyki prostej otrzymuje się mnożąc macierze przekształceń.

$$T_0^n = T_0^1 \cdot T_1^2 \cdot \dots \cdot T_{i-1}^i \cdot \dots \cdot T_{n-1}^n, \tag{3.5}$$

gdzie:

T_0^n – macierz, zawierająca pozycję i orientację n-tego układu lokalnego w układzie bazowym,

T_{i-1}^{i} – macierz przekształcająca układ $i-1$ w i.

Rozwiązanie równania (3.5), czyli wynik prostego zadania kinematyki, zawsze daje jednoznaczne rozwiązanie.

3.3.3. Notacja Denavita-Hartenberga

Notacja Denavita-Hartenberga (D-H) jest to konwencja uproszczająca rozwiązywanie równań mechaniki klasycznej. Została zapoczątkowana przez Jacques Denavit-a i Richard Hartenberg-a w 1955 roku w celu standaryzacji układu współrzędnych dla mechanicznych łączeń. Do opisu każdego połączenia notacja D-H wykorzystuje cztery parametry, dwa z nich opisują swój układ współrzędnych, a dwa kolejne reprezentują sposób połączenia z sąsiednim układem. Znaczenie parametrów jest następujące:

θ_i – kąt przegubu pomiędzy X_i i X_{i-1} liczony względem Z_{i-1},

b_i – odległość członu między osiami X_i i X_{i-1} mierzona wzdłuż Z_{i-1},

a_i – odległość członu między osiami Z_i i Z_{i-1} wzdłuż X_i,

α_i – kąt przegubu pomiędzy Z_i i Z_{i-1} liczony wokół osi X_i.

Dla każdego rodzaju połączeń trzy z powyższych parametrów są stałe. Dla połączenia obrotowego zmienną jest θ_i, a dla złącza postępowego b_i.

Opisywana notacja wykorzystuje 4 podstawowe przekształcenia do wyznaczenia macierzy T_{i-1}^{i}.

$$T_{i-1}^{i} = Rot(Z_{i-1},\theta_i)Trans(Z_{i-1},b_i)Trans(X_i,a_i)Rot(X_i,\alpha_i) = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & b_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

4. 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 ... .

4.1. 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.

4.1.1. 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.

4.1.2. 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.

4.1.2.1. 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 **??**.

4.1.2.2. 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

4.1.2.3. 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.

4.1.2.3.1 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.

4.1.2.3.2 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*.

4.1.2.3.3 Statusy celi

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

4.1.2.3.4 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* **??**.

4.1.2.3.5 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*.

4.1.2.3.6 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.

4.1.2.3.7 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.

4.1.2.4. 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.