Politechnika Poznańska

Wydział Informatyki


Tomasz Merda

Praca magisterska

# Ocena jakości rozpoznawania gestów statycznych przy użyciu technologii Microsoft Kinect

Promotor: dr inż. Mikołaj Sobczak

Poznań, 2012

# Streszczenie

*Komputerowe rozpoznawanie gestów dzięki błyskawicznemu postępowi technologicznemu staje się realnym podejściem do wykorzystywania ludzkiego ciała do sterowania aplikacjami. Urządzenie Microsoft Kinect dzięki nowatorskiemu podejściu do przechwytywania obrazu ciała użytkownika pozwala wprowadzić rozpoznawanie gestów na nowy, wyższy poziom. Niestety w dotychczasowym oprogramowaniu dla tego urządzenia brakuje bibliotek dla rozpoznawania gestów. Niniejsza praca proponuje rozwiązanie problemu rozpoznawania gestów statycznych przy użyciu sieci neuronowych. W ramach pracy powstała aplikacja korzystająca z nowego rozwiązania, w pełni sterowalna przy użyciu ludzkiego ciała, oparta na nowych komponentach interfejsu użytkownika. Ostatecznie zostały zaprezentowane wyniki oceny dokładności rozpoznania poparte przeprowadzonymi badaniami.*

# Abstract

*Computer recognition of gestures becomes a growing perspective for application control. Microsoft with his Kinect is an innovation leader in this field of science, offering a standalone device for tracking user's body in an affordable price. Unfortunately Microsoft Kinect SDK lacks of the default gesture recognition facility. The thesis proposes a solution for static gesture recognition problem based on neural networks. A concrete application based on the new solution is presented, fully steerable with the human body, along with new GUI controlls for aplaction control. Finally, the results assessing the accuracy of recognition supported by conducted experiments are presented.*

# Spis treści

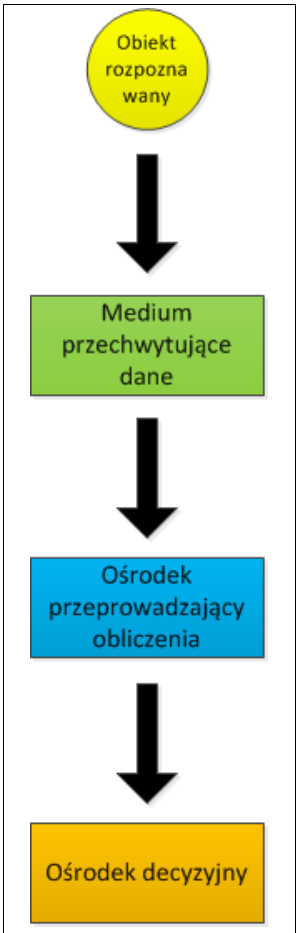
# 1. Wprowadzenie

## 1.1. Rozpoznawanie gestów

Rozpoznawanie gestów jest podzbiorem komputerowego rozpoznawania wzorców, które może być definiowane jako wychwytywanie określonych schematów ze zbioru surowych danych. Do najbardziej znanych aplikacji rozpoznawania wzorców można zaliczyć:

- rozpoznawanie mowy [B],
- rozpoznawanie obrazów, w tym:
  - rozpoznawanie pisma [D],
  - rozpoznawanie kodów (kody kreskowe [E], kody QR [F], kody pocztowe [G]),
  - rozpoznawanie znaków drogowych [H],
  - rozpoznawanie twarzy [I],
  - rozpoznawanie gestów [J],
- rozpoznawanie stylu pisania [C].

Rozpoznawanie gestów w najprostszym scenariuszu wymaga obiektu rozpoznawanego (np. człowiek), medium przechwytującego dane (np. kamera), systemu przeprowadzającego operacje na tych danych, oraz




*Rysunek 1: Podstawowa budowa systemu rozpoznawania gestów*

ośrodka decyzyjnego (Rysunek 1).

## 1.2. Rzeczywiste aplikacje rozpoznawania gestów

Na początku XXI wieku rozpoznawanie gestów nabrało rozpędu dzięki systematycznie wzrastającej mocy obliczeniowej komputerów osobistych oraz zastosowaniu rozpoznawania gestów jako nowego kontrolera. Kierunek ten rozwijał się szczególnie szybko w przemyśle gier wideo. Najbardziej popularne podejścia do rozpoznawania gestów to (w porządku chronologicznym):

- Playstation EyeToy,
- Nintendo Wii,
- Playstation Move,
- Microsoft Kinect.

Playstation EyeToy było pierwszym podejściem do zastosowania rozpoznawania gestów w grach komputerowych przy użyciu kamery wideo wprowadzonym na szeroką skalę. Był to moduł (kamera z mikrofonem) dołączany do konsoli Sony Playstation 2 (Rysunek 2). Problemem była jednak wrażliwość tego systemu, ponieważ operował w całości w spektrum światła widzialnego i zaburzenia tego ośrodka (np. mocne światło od słońca, ciemność) odbijały się na skuteczności rozpoznawania.

*Rysunek 2: Playstation EyeToy [9]*

Nintendo Wii zrewolucjonizowało rozpoznawanie gestów poprzez całkowite pozbycie się kamery wideo i wykorzystanie urządzeń, wyposażonych w akcelerometry, trzymane w dłoniach (Rysunek 3) później wyposażone także w inne sensory (np. nacisku w Nintendo Wii Fit)). Dzięki temu podejściu Nintendo Wii zdecydowanie prowadziło w liczbie sprzedawanych konsol nad

swoimi bardziej zaawansowanymi technologicznie konkurentami – Microsoft Xbox 360 oraz Playstation 3 – nie posiadającymi jednak kontrolerów ruchu.


*Rysunek 3: Nintendo Wii [10]*

Po kilku latach Playstation 3 wypuściło swoje kontrolery „Move" (Rysunek 4), będące podobnym rozwiązaniem do Wii, z tą różnicą, że wykorzystywały jeszcze dodatkowe świecące kule przy kontrolerach, których poszukiwała kamera wideo. Można zaryzykować stwierdzenie, że doszło do technologicznego połączenia Playstation EyeToy oraz Nintendo Wii w Playstation Move.


*Rysunek 4: Playstation Move [11]*

Kolejna rewolucja rozpoczęła się jednak, gdy Microsoft wypuścił swój kontroler nazwany „Kinect”, pozbywając się „zbędnych” kontrolerów trzymanych w dłoniach (Rysunek 5). Kinect wyświetla siatkę w podczerwieni na trójwymiarowej przestrzeni. Obraz tej przestrzeni jest następnie zbierany przez specjalną kamerę potrafiącą zmierzyć odległość od obiektu na którym wyświetlana jest siatka.


*Rysunek 5: Microsoft Kinect [12]*

Wprowadzenie na rynek Microsoft Kinect okazało się wielkim sukcesem, a Kinect zaczął być wykorzystywany w badaniach nie mających wiele wspólnego z przemysłem gier wideo. Ekstremalnym przykładem jest plan wykorzystania sensora Kinect w kosmosie do dokowania małych satelitów [13].

## 1.3. Motywacja pracy

Zagadnienie rozpoznawania gestów przy wykorzystaniu możliwości sensora Kinect, a szczególnie wykrywania odległości od obiektów wydaje się być niezmiernie interesujące. Brak domyślnych mechanizmów rozpoznawania gestów wprowadza pole do popisu dla programistów – eksperymentatorów, a nowa metoda sterowania aplikacjami, rodem z filmów science-fiction, daje ogromne możliwości rozwoju interfejsów użytkownika. Wszystko to składa się na motywację autora do przeprowadzenia badań nad możliwościami sensora Kinect oraz nad możliwymi implementacjami rozpoznawania gestów.

# 2. Cel i zakres

Celem niniejszej pracy jest określenie przydatności urządzenia Microsoft Kinect w procesie rozpoznawania gestów oraz dokładności jaką można uzyskać dzięki temu urządzeniu.

W zakres projektu wpisują się następujące punkty:

1. Zapoznanie się z dostępnymi technikami rozpoznawania gestów i wybór odpowiedniego rozwiązania,
2. Przygotowanie aplikacji umożliwiającej rozpoznawanie gestów oraz zbadanie ich jakości,
3. Zapoznanie się z dokumentacją urządzenia Kinect oraz wybór technologii do wykonania aplikacji badawczych,
4. Przeprowadzenie badań określających przydatność i skuteczność urządzenia Kinect w rozpoznawaniu gestów statycznych,
5. Sporządzenie pracy dokumentującej wykonane badania.

Dodatkowo programy rozwijane w ramach projektu będą demonstracją możliwości Kinekta w zakresie:

- sterowania aplikacjami przy użyciu dłoni,
- rozpoznawania gestów statycznych opartych na ruchach rąk,
- możliwości manipulowania parametrami rozpoznawania gestów w celu jego kalibracji,
- przygotowania przepływu sterowania w aplikacji, tak aby zminimalizować wykorzystanie standardowych urządzeń wejścia/wyjścia w całym procesie przygotowania i rozpoznania gestów,
- przygotowania uniwersalnych komponentów do obsługi przy pomocy ruchów dłońmi.

# 3. Stan wiedzy

Niniejsza praca prezentuje możliwości sensora Kinect oraz ocenia przydatność tego urządzenia w gestii rozpoznawania gestów, na przykładzie gestów statycznych. Niniejszy rozdział przedstawia specyfikację sensora Kinect, proces jego powstawania oraz przeprowadzone prace nad wykorzystaniem tego urządzenia w praktyce.

## 3.1. Microsoft Kinect

Macierz sensorów w urządzeniu Microsoft Kinect zawiera[1, 2]:

- kamerę VGA o rozdzielczości 1280x1024 pikseli i szybkości 30 klatek na sekundę,
- kamerę QVGA o rozdzielczości 320x240 i szybkości 30 klatek na sekundę do pomiaru głębokości,
- promiennik podczerwieni,
- macierz 4 mikrofonów kierunkowych,
- napęd pozwalający na uchylanie głowicy w promieniu ± 28°,
- akcelerometr pracujący w 3 wymiarach.

*Rysunek 6: Macierz sensorów w urządzeniu Kinect [3]*

## 3.2. Microsoft Kinect SDK

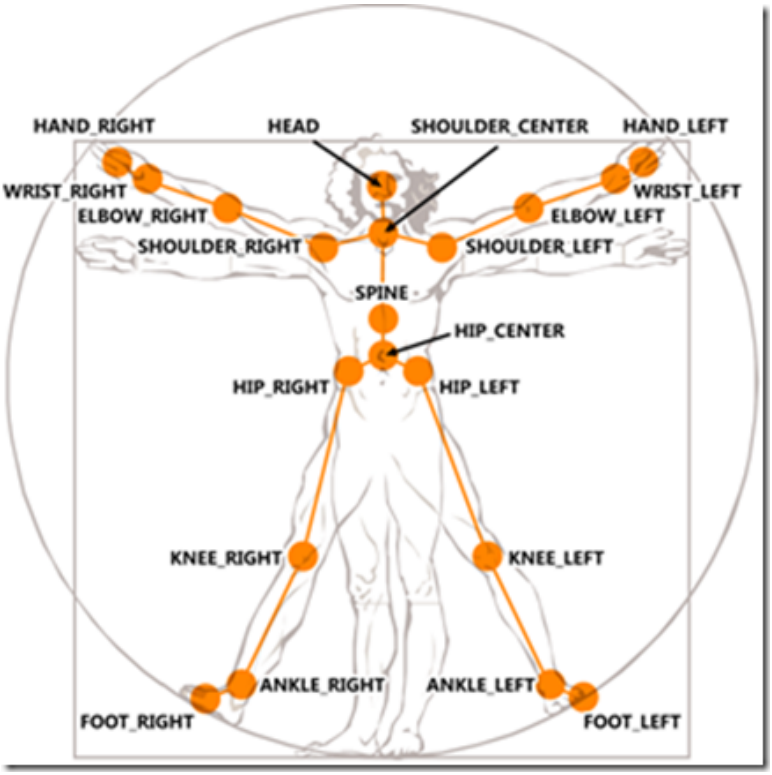
Microsoft Kinect SDK to zestaw narzędzi i bibliotek pozwalający na wykorzystanie urządzenia Kinect w aplikacji. Microsoft wypuścił SDK 16 czerwca 2011 roku, wspierając języki C#, C++ oraz Visual Basic [4].

Najważniejszą częścią SDK są strumienie danych zwracane z macierzy sensorów.

Podstawowy strumień danych pobierany jest z kamery VGA z szybkością 30 FPS. Obraz przekazywany może być w formacie kolorów sRGB, lub YUV i w jednym z kilku wielkości ramek (maksymalnie 1280x1024 pikseli).

Nowością wprowadzoną przez Microsoft jest „strumień głębokości" pobierany z kamery QVGA w ramkach wielkości do 640x480 pikseli i z szybkością 30 FPS. Kamera ta działa w spektrum podczerwieni. Specjalne źródło światła w podczerwieni oświetla przestrzeń przed Kinektem, a procesor przy pomocy danych z kamery oblicza odległość w każdym z odczytanych pikseli. Odległość jaką może odczytać sensor wynosi od 0,7 do 6 metrów od urządzenia.

Kolejnym strumieniem jest „strumień szkieletowy". Kinect na podstawie pierwszych dwóch strumieni rozpoznaje ludzką sylwetkę, nadaje jej identyfikator oraz zwraca pozycję 20 części ciała (Rysunek 7).




*Rysunek 7: Punkty na ciele użytkownika zwracane jako strumień danych przez Kinecta[5]*

Współrzędne punktów zwracane są w 3 wymiarach dzięki czemu bez żadnych dodatkowych operacji można uzyskać umiejscowienie sylwetki użytkownika w przestrzeni. Kinect nie pozwala jednak na rozpoznawanie pojedynczych palców, a jedynie dłoni jako całości.

Ostatni strumień danych to strumień dźwięku pochodzący z macierzy mikrofonów. Wbudowane możliwości sensora pozwalają na poprawę jakości dźwięku oraz określenia kierunku jego pochodzenia. Microsoft dostarcza także biblioteki pozwalające na rozpoznawanie komend głosowych[1].

## 3.3. Rozwiązania oparte na Microsoft Kinect

### 3.3.1. Początki Microsoft Kinect

Pierwsze próby przejęcia kontroli nad oddanym do rąk użytkowników urządzeniem w listopadzie 2010 roku zostały podjęte natychmiast po premierze. W przeciągu tygodnia powstały nieoficjalne sterowniki do kontrolowania podstawowych możliwości Kinekta [6]. Następnie zaczęła rozwijać się społeczność wspierająca otwarte oprogramowanie OpenKinect oparte na języku Java.

1 W czasie pisania pracy przez autora dostępny był jedynie język angielski.

Nie było ono jednak oficjalnie wspierane przez Microsoft przez co nie można było liczyć na reklamację w przypadku uszkodzenia sprzętu. 16 czerwca 2011 roku Microsoft udostępnił własne oprogramowanie – Microsoft Kinect SDK Beta [4]. Jest to jedyne licencjonowane oprogramowanie dla Kinekta i co za tym idzie – nie wiąże się z utratą gwarancji. Od tego czasu lawinowo rośnie liczba projektów opartych na Kinekcie. Użytkownicy mogą chwalić się i udostępniać swoje aplikacje na portalach takich jak np. Coding4Fun [7] wpierany przez Microsoft. Licencja dla wersji beta nie pozwala jednak na wykorzystanie Kinekta w celach komercyjnych [8].

### 3.3.2. Prace oparte na Microsoft Kinect

Microsoft zachęca ośrodki badawcze do rozwijania możliwości Kinekta, gdyż oprogramowanie posiada jeszcze wiele niedociągnięć. Jednym z problemów jest optymalizacja wykorzystania zasobów, gdyż Kinect ma wysokie wymagania sprzętowe – dwurdzeniowy procesor o taktowaniu 2,66 GHz i 2 GB pamięci RAM. Microsoft Kinect SDK nie dostarcza też żadnego standardowego mechanizmu do rozpoznawania gestów – jedynymi informacjami są strumienie danych opisane w rozdziale 3.2. Właśnie w tym miejscu wpisuje się niniejsza praca, próbując opracować metodę rozpoznawania gestów.

Mimo że powstaje co raz więcej aplikacji prezentujących rozpoznawanie gestów, liczba artykułów na ten temat jest jeszcze bardzo skromna. Praca [K] dotyczy rozpoznawania gestów dłoni, rozszerzając standardowe umiejętności rozpoznawcze Kinekta o rozpoznawanie gestów opartych na palcach. Brak możliwości rozpoznawania poszczególnych palców jest problemem przy rozpoznawaniu np. języka migowego. Mimo obiecujących wyników osiągniętych przez autorów są jednak marne szanse na osiągnięcie ww. celów bez zwiększenia rozdzielczości kamery mierzącej głębokość.

Z kolei artykuł [L] demonstruje rozpoznawanie gestów przy wykorzystaniu ukrytych modeli Markova. Opisane przez autora rozwiązanie problemu rozpoznawania gestów dynamicznych może okazać się użyteczne w dalszych badaniach nad możliwościami Kinekta. W tej pracy zostaną jednak bezpośrednio wykorzystane sieci neuronowe, aby gruntownie poznać ich możliwości oraz przetestować je w praktyce.

Problematyka zastąpienia standardowych interfejsów przy pomocy gestów jest omawiana w artykułach [M] oraz [N]. W pracy [M] zaproponowano gesty podobne do gestów używanych na ekranach dotykowych. Autor spotkał się z problemem zastąpienia interfejsu dotykowego gestami w obszarach:

- rozpoznania rozpoczęcia gestu,
- rozpoznania „kliknięcia”,
- intuicyjnością zaproponowanych rozwiązań.

Hongli Lai jako główny element graficzny gestu zaproponował rysunek reprezentujący dłoń, będący wirtualnym kursorem sterowanym przez dłoń użytkownika stojącego przed Kinektem. Jako alternatywa „kliknięcia” zaproponowano przytrzymanie dłoni w punkcie, kiedy to dłoń-kursor zmieniał swą przezroczystość. Zgodnie z wynikami przeprowadzonych przez Hongli Lai badań zaproponowany przez niego interfejs oparty na gestach okazał się wielce nieintuicyjny (użytkownicy nie mogli odgadnąć gestów) oraz niewygodny (długie, przez co męczące, oczekiwanie na rozpoczęcie gestu i jego zakończenie). Przy projektowaniu nowego systemu należy zaproponować bardziej intuicyjne sterowanie aplikacją.

Autorzy [N] podjęli się problematyki uczenia użytkowników sterowania aplikacją przy użyciu gestów oraz analizą ich zachowania. Aplikacja powinna brać pod uwagę zróżnicowanie użytkowników i ich możliwości w aspekcie prezentowania gestów.

# 4. Środowisko badawcze

## 4.1. Opis projektu

W zgodzie założeniami podanymi w rozdziale 2. wykonano dwie aplikacje: GesturesEditor oraz GesturesRecognizer. Aplikacje te uzupełniają się w procesie tworzenia nowej bazy gestów. GesturesEditor służy do przygotowania bazy pod naukę, natomiast GesturesRecognizer wspomaga naukę sieci oraz rozpoznawanie gestów. Zaimplementowano 2 oddzielne aplikacje, ponieważ GesturesRecognizer w założeniu miał być sterowanych wyłącznie przy użyciu własnego ciała, natomiast przygotowanie metadanych gestów bez użycia klawiatury byłoby nieefektywne i męczące fizycznie dla użytkownika (np. wprowadzanie nazw gestów).

### 4.1.1. GesturesEditor

**Opis**

Aplikacja służąca do przygotowania metadanych na temat zbiorów gestów.

**Cel programu**

Celem programu GesturesEditor jest przygotowanie zbioru gestów do nauki oraz rozpoznawania gestów, które są przeprowadzane w programie GesturesRecognizer. Szczegółowa funkcjonalność prezentuje się następująco:

- Utworzenie nowego zbioru gestów,
- Nadanie nazwy zbiorowi gestów,
- Wybór liczby gestów w ramach zbioru,
- Wybór liczby wymiarów (2 – X, Y, lub 3 – X, Y, Z) wykorzystywanych do rozpoznania gestu,

- Wybór liczby punktów na ciele zawartych w geście (GesturesRecognizer aktualnie obsługuje jedynie 4 konkretne punkty na ciele – obie dłonie oraz łokcie. Powód takiego stanu rzeczy opisany został w rozdziale 5.4.4),
- Przyporządkowanie nazwy do konkretnego gestu,
- Zapisanie zbioru gestów do bazy danych,
- Modyfikacja nazw zbiorów gestów oraz gestów w ramach tych zbiorów zapisanych w bazie danych.

**Dane wejściowe**

Koncepcja na temat zbioru gestów, które mają być wynikiem wykonania aplikacji GesturesRecognizer.

**Dane wyjściowe**

Wyjściem aplikacji jest zbiór metadanych na temat gestów, do których w programie GesturesRecognizer przyłączone zostaną informacje identyfikujące same gesty.

### 4.1.2. GesturesRecognizer

**Opis**

Aplikacja pozwala na naukę sieci neuronowej gestów na bazie metadanych podanych w aplikacji GesturesEditor, na rozpoznawanie nauczonych gestów oraz ustawienia parametrów pozwalających na ocenę dokładności rozpoznania.

**Cel programu**

Celem aplikacji jest:

- demonstracja możliwości wykorzystania sensora Kinect do sterowania aplikacją,
- zaproponowanie uniwersalnych komponentów do sterowania aplikacją przy użyciu Kinekta,
- możliwość nauczenia sieci neuronowej konkretnych gestów,
- możliwość rozpoznania gestu przy pomocy nauczonej sieci neuronowej,
- możliwość manipulowania parametrami określającymi dokładność rozpoznania gestu,
- możliwość określenia dokładności rozpoznania gestu.

**Dodatkowe możliwości**

- możliwość wyeksportowania nauczonej sieci neuronowej,
- możliwość manipulacji parametrami sieci neuronowej,
- możliwość zmiany języka aplikacji (polski, lub angielski),
- możliwość zapisania zmian w zbiorze gestów w bazie danych,

**Dane wejściowe**

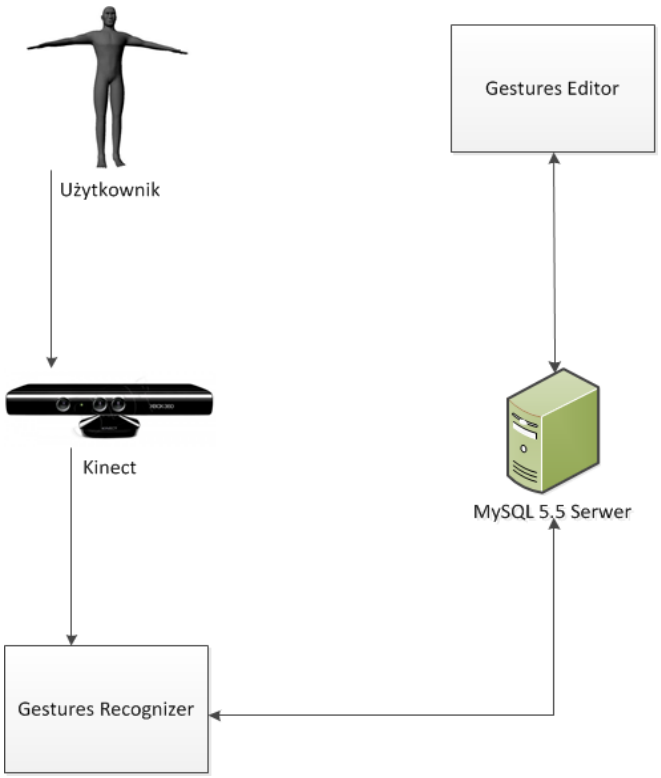
Metadane dotyczące zbiorów gestów, pobierane z bazy danych.

**Dane wyjściowe**

Sieci neuronowe nauczone rozpoznawania gestów.

### 4.1.3. Architektura systemu

System rozpoznawania gestów złożony jest z 2 aplikacji – GesturesEditor oraz GesturesRecognizer. Aplikacje te komunikują się ze sobą przy użyciu serwera baz danych MySQL 5.5. GesturesEditor dostarcza metadanych na temat zbiorów gestów, natomiast GesturesRecognizer pobiera informację o gestach użytkownika poprzez sensor Kinect (Rysunek 8).




*Rysunek 8: Architektura systemu*

## 4.2. Wykorzystane technologie

### 4.2.1. Aplikacje GesturesEditor i GesturesRecognizer

Do wykonania aplikacji wykorzystano następujące technologie:

**Środowisko programistyczne:** Microsoft Visual Studio 2010,

**Język programowania:** C#,

**Biblioteka do komunikacji z Microsoft Kinect:** Microsoft Kinect SDK [14],

**Biblioteka sieci neuronowych:** HSynapse [15],

**Interfejs użytkownika:** Windows Presentation Foundation (WPF),

**Baza danych:** MySQL Server 5.5,

**System kontroli wersji:** Git,

**Serwer kontroli wersji:** Github [16].

Sprzęt wykorzystany w projekcie prezentuje się następująco:

Microsoft Kinect,

Adapter USB,

Laptop Compal:

- Procesor: Intel DualCore T4200 2.0 GHz 64bit,
- Pamięć RAM: 4 GB.

### 4.2.2. Praca magisterska

Do wykonania niniejszej pracy wykorzystano następujące technologie:

**System operacyjny**: Microsoft Windows 7 64-bit,

**System kontroli wersji:** Git,

**Serwer kontroli wersji:** Github [16],

**Edytor tekstu:** OpenOffice Writer 3.3.0,

**Arkusz kalkulacyjny:** OpenOffice Calc 3.3.0,

**Narzędzia graficzne:**

- Microsoft Visio,
- Microsoft Paint.

**Narzędzie do przechwytywania zrzutu z ekranu:** CamStudio.

# 5. System w użyciu

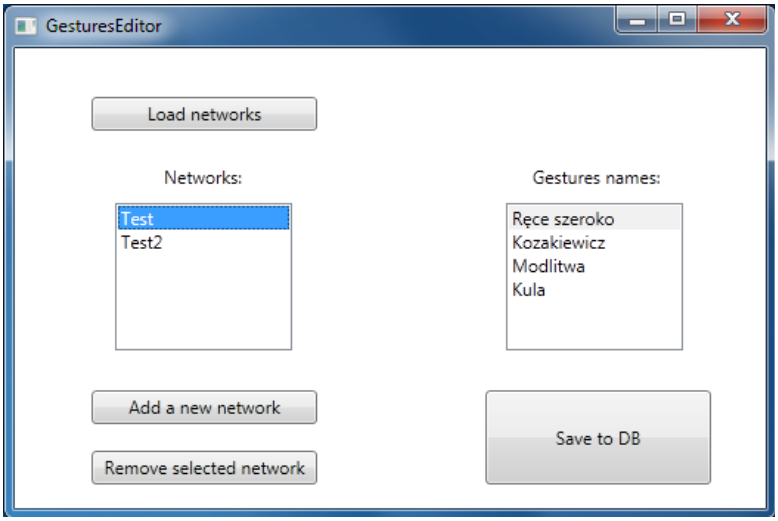
Niniejszy rozdział przedstawia aplikacje przygotowane w celu przeprowadzenia eksperymentów. Proces instalacyjny wymienionych aplikacji przedstawiono w Dodatku A.

## 5.1. GesturesEditor

### 5.1.1. Budowa

Aplikacja zbudowana została przy użyciu standardowych elementów technologii WPF, znanych wcześniej z Windows Forms. Postawiono na prostotę i intuicyjność projektu, tak aby jak najszybciej użytkownik mógł przejść do aplikacji GesturesRecognizer i rozpocząć naukę utworzonej właśnie sieci.

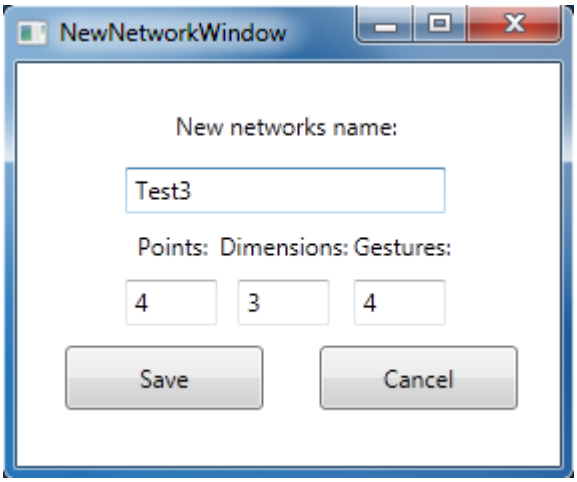
### 5.1.2. Wygląd



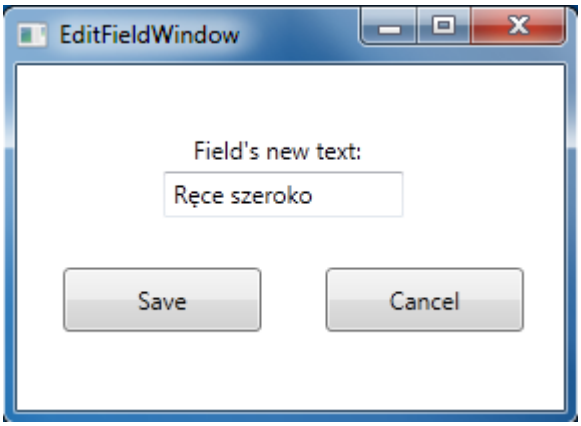

*Rysunek 9: Główne okno programu GesturesEditor*

Aplikacja GesturesEditor składa się z jednego głównego okna programu (Rysunek 9). Po lewej stronie okna dostępne są przyciski pozwalające na załadowanie sieci neuronowych z bazy danych (**Load Networks**), utworzyć nową sieć (**Add a new network**) oraz usunąć sieć (**Remove selected networ**k). Dodatkowo w mniejszym oknie opisanym jako **Networks:** wyświetlane są załadowane oraz utworzone sieci neuronowe. Po prawej stronie okna znaleźć można przycisk zapisujący sieci neuronowe do bazy danych (**Save to DB**) oraz mniejsze okno opisane jako **Gestures names:** wyświetlające gesty zawarte w danej sieci neuronowej.

Program podczas wykonywaniu akcji dodawania nowej sieci neuronowej wyświetla okno z metadanymi, które użytkownik musi uzupełnić (Rysunek 10). Można w nim określić nazwę sieci **New networks name:**, liczbę punktów na ciele do analizy (**Points**; obecnie obsługiwana jest jedynie domyślna wartość „4", która oznacza obie dłonie oraz łokcie), liczbę wymiarów (**Dimensions**; można podać 2 wymiary – X,Y, lub 3 wymiary – X, Y, Z) oraz liczbę gestów, których sieć ma się docelowo nauczyć (**Gestures:**). Zmiany można następnie zatwierdzić przyciskiem **Save**, lub odrzucić przyciskiem **Cancel**.

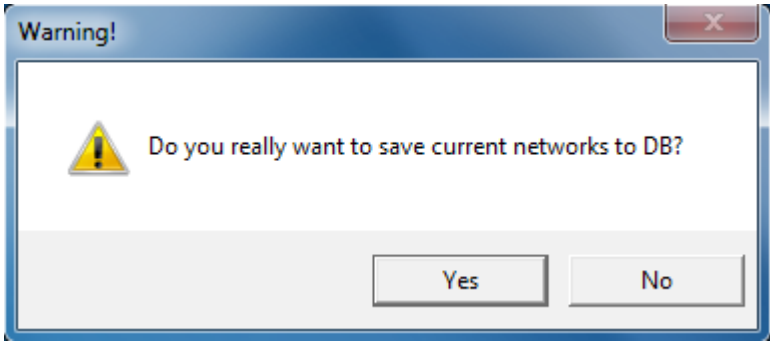



*Rysunek 10: Okno dodawania nowej sieci neuronowej*

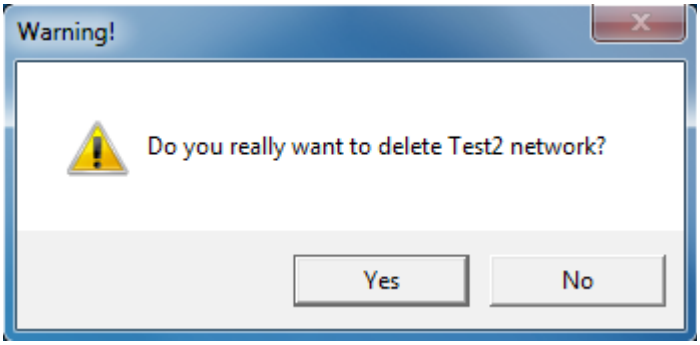



*Rysunek 11: Okno edycji pola*

Podwójne kliknięcie w nazwę sieci neuronowej, lub w nazwę gestu w głównym oknie programu powoduje pojawienie się okna edycji pola (Rysunek 11). W jedynym dostępnym w nim polu tekstowym można podać nową treść edytowanego pola tekstowego. Zmiany można następnie zatwierdzić przyciskiem **Save**, lub odrzucić przyciskiem **Cancel**.




*Rysunek 12: Monit pytający o zapisanie zmian do bazy danych*




*Rysunek 13: Monit pytający o usunięcie sieci neuronowej*

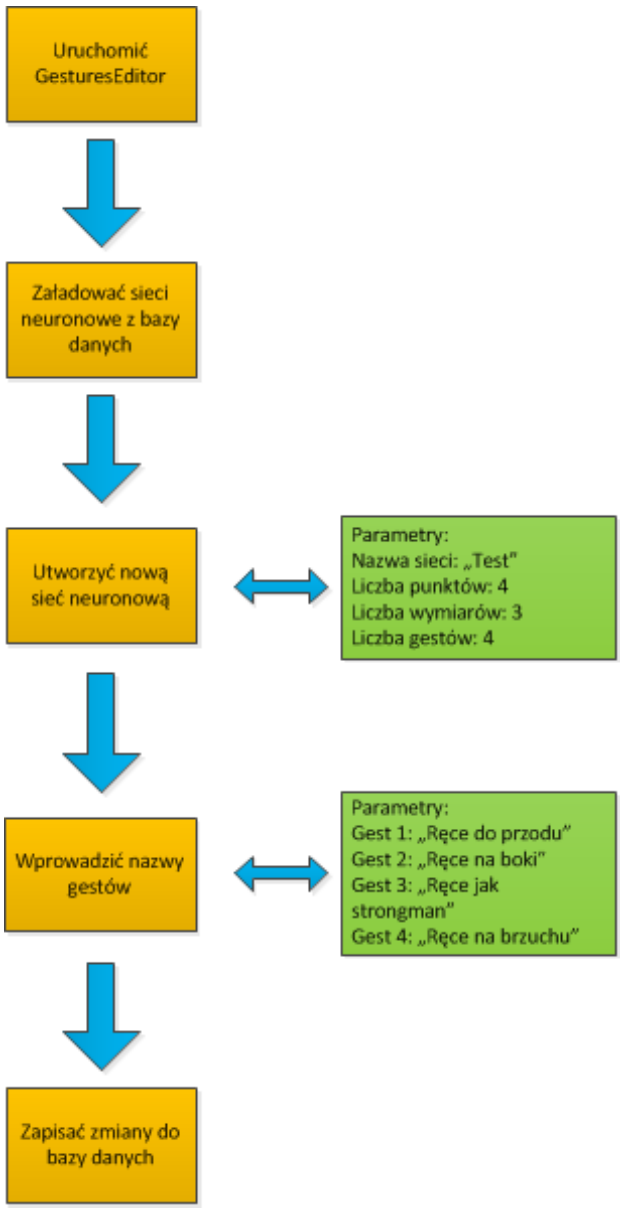
Zapisywanie zmian do bazy danych oraz usuwanie sieci neuronowej skutkują odpowiednimi monitami (Rysunki 12 i 13).

### 5.1.3. Sterowanie

Sterowanie programem GesturesEditor przebiega przy użyciu standardowej klawiatury i myszki. Aplikacja ta została wyodrębniona z aplikacji GesturesRecognizer właśnie z powodu problemów we wprowadzeniu sterowania przy użyciu ciała. Rozważany był scenariusz wprowadzenia wirtualnej klawiatury sterowanej ruchami dłoni, ale podejście to wydawało się zbyt nieintuicyjne i niewygodne, aby można było z niego swobodnie korzystać. Wprowadzenie niewielkich metadanych na temat gestów i sieci neuronowych z klawiatury nie powinno zaburzać opinii o sterowaniu aplikacją jako całości.

### 5.1.4. Przykładowy scenariusz

Przykładowy scenariusz użycia programu GesturesEditor wraz z kolejnymi krokami przedstawiony został na poniższym diagramie (Rysunek 14).




*Rysunek 14: Przykładowy scenariusz dla aplikacji GesturesEditor*

## 5.2. GesturesRecognizer

### 5.2.1. Budowa

Aplikacja GesturesRecognizer zaprojektowana została, aby obsługiwać w pełni sterowanie przy użyciu dłoni. W całym programie nie ma miejsca w którym potrzebna by było ingerencja przy użyciu standardowych urządzeń sterujących takich jak klawiatura czy myszka.

## CircularMinuteTimer

Podstawowym komponentem interakcji z użytkownikiem jest CircularMinuteTimer będący formą przycisku (Rysunek 15). Przycisk został zaprojektowany biorąc pod uwagę wyniki testów użyteczności aplikacji opisanej w artykule [M], gdzie zachowanie komponentu reprezentującego kursor okazało się być bardzo nieintuicyjne dla użytkowników. Zmianę parametru przezroczystości kursora zastąpiono paskiem ładowania połączonym z podświetleniem zaznaczonego przycisku. Położenie dłoni na CMT powoduje rozpoczęcie procesu zatwierdzania indykowanego przy użyciu paska okrążającego przycisk (Rysunek 16). Zatwierdzanie trwa 1 sekundę. W dowolnym momencie dłoń można wysunąć spod przycisku co skutkować będzie przerwaniem ładowania. Jeżeli ładowanie zakończy się pomyślnie wykonana zostaje przypisana do niego akcja. Dodatkowo CMT opisane jest przy użyciu klasy GraphicalEffects, która pozwala na zarządzanie pozycją CMT na ekranie. Pozwala też na zastosowanie specyficznych układów pojawiania i znikania CMT, jak na przykład „ucieczka" z ekranu przy użyciu losowych kierunków.


*Rysunek 15: Przycisk CMT w stanie spoczynku*


*Rysunek 16: Przycisk CMT podczas zatwierdzania*

## ExtendedCircularMinuteTimer

Przycisk CMT może być także używany w trybie rozszerzonym („Extended"). W tym przypadku na przycisku pojawia się dodatkowa wartość mogąca być wartością liczbową, ciągiem znaków, lub wartościami wyliczeniowymi. Tryb rozszerzony ma funkcje zmiany parametrów programu. Przykładem ECMT może być zmiana języka aplikacji. W tym przypadku przygotowane zostały 2 wartości będące wartościami wyliczeniowymi – PL i EN. Lista tych wartości

przekazywana jest do przycisku, a załadowanie paska skutkować będzie zmianą wartości na kolejną na liście, lub na pierwszą w przypadku, gdy poprzednia wartość była ostatnia na liście (Rysunek 17).


*Rysunek 17: Zmiana wartości na przycisku ECMT*

## InfoBoard

Projekt aplikacji GesturesRecognizer zakładał maksymalizację przejrzystości i intuicyjności aplikacji przy minimalnym stopniu dezorientacji użytkownika. Należało wprowadzić system dostarczania komunikatów do użytkownika nie zaburzający kompozycji aplikacji. W tym celu zaimplementowano komponent InfoBoard (inaczej IB). Jest to ramka zawierająca zbiór komunikatów, która w zależności od zastosowania może pojawiać się, lub znikać. Przykładem wykorzystania takiego komunikatu jest powiadomienie użytkownika o zakończeniu procesu nauki sieci neuronowej (Rysunek 18). IB pojawia się na okres 5 sekund prezentując informację, po czym powoli znika. Inne zastosowanie komponentu InfoBoard to przedstawianie informacji na temat rozpoznanego właśnie gestu. W tym przypadku komponent jest wyświetlany przez cały okres przebywania użytkownika na ekranie **Gestures** (Rysunek 19).


*Rysunek 18: Komponent InfoBoard informujący użytkownika o fakcie zakończenia procesu nauki sieci neuronowej*


*Rysunek 19: Komponent InfoBoard prezentujący nazwę rozpoznanego gestu (w tym przypadku gestu nie rozpoznano)*

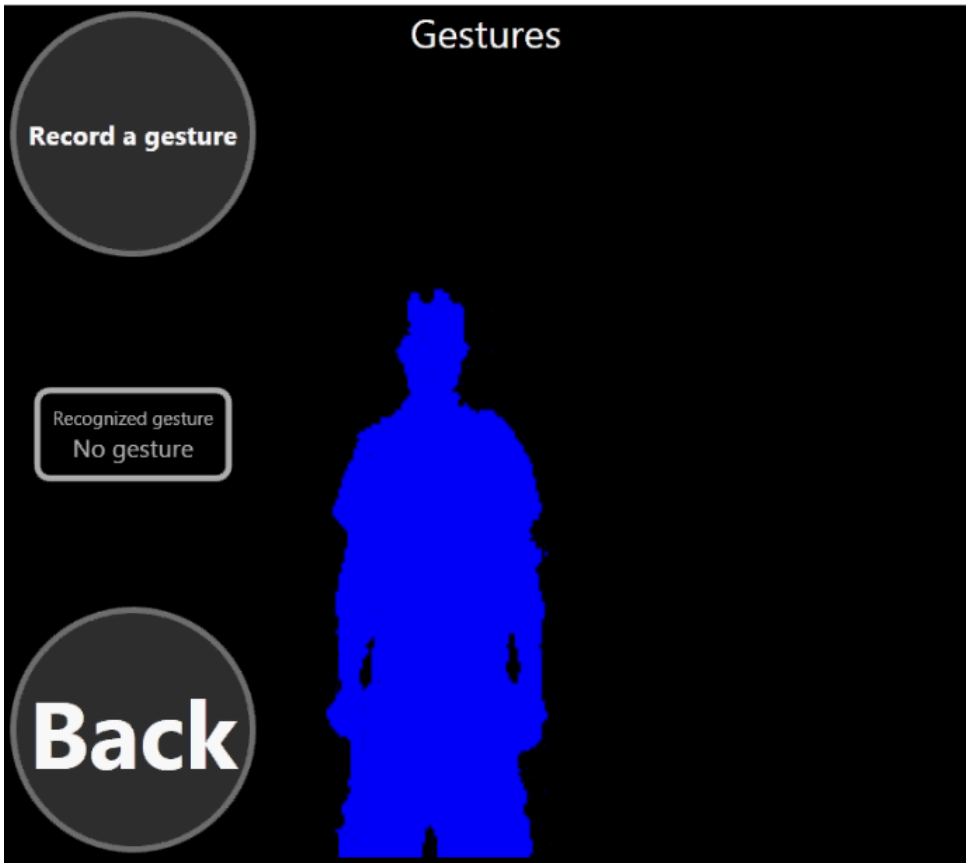
## Logika aplikacji

Aplikacja podzielona została na 3 sekcje – **Gestures**, **Configuration** oraz **Learning**. Do każdej z tych sekcji można dostać się przechodząc przez ekran główny **Main screen**. Funkcje poszczególnych sekcji wyglądają następująco:

- **Gestures** – nagrywanie oraz rozpoznawanie gestów,
- **Learning** – nauka oraz zmiana parametrów nauki sieci neuronowych,
- **Configuration** – ustawienia aplikacji.

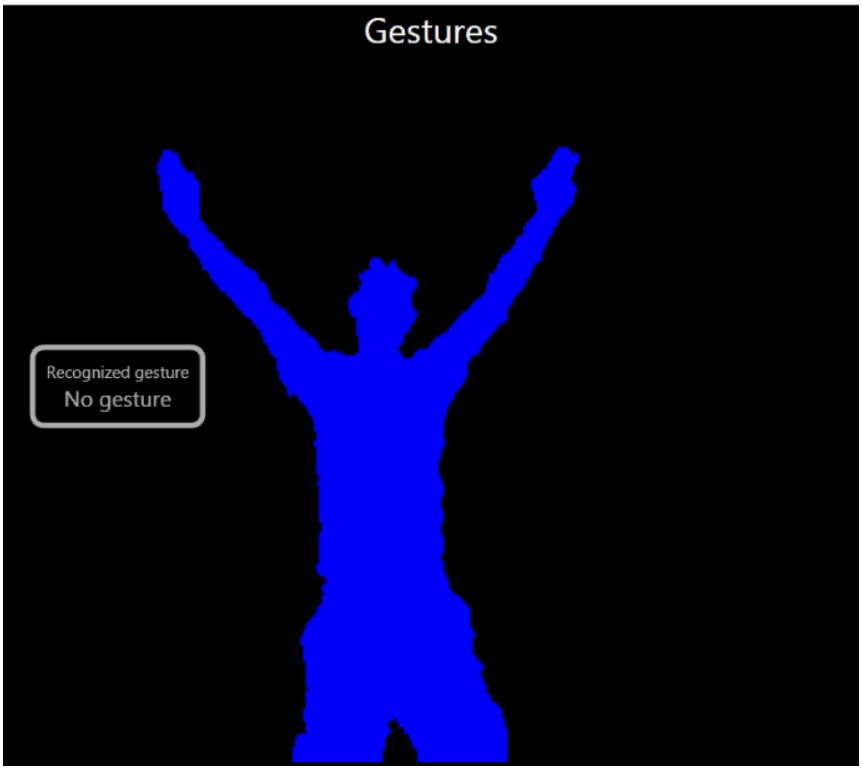
## Sekcja Gestures

Sekcja Gestures to serce całej aplikacji, pozwala na nagrywanie oraz rozpoznawanie gestów oraz przydzielanie przykładów do nauki sieci neuronowej. Po zatwierdzeniu przycisku CMT Gestures pojawiają się przyciski **Record a gesture** służący do rozpoczęcia nagrywania gestu oraz **Back** służący do powrotu do ekranu głównego (**Main screen**). Pojawia się także komponent InfoBoard z informacją o rozpoznanym uprzednio geście (Rysunek 20).



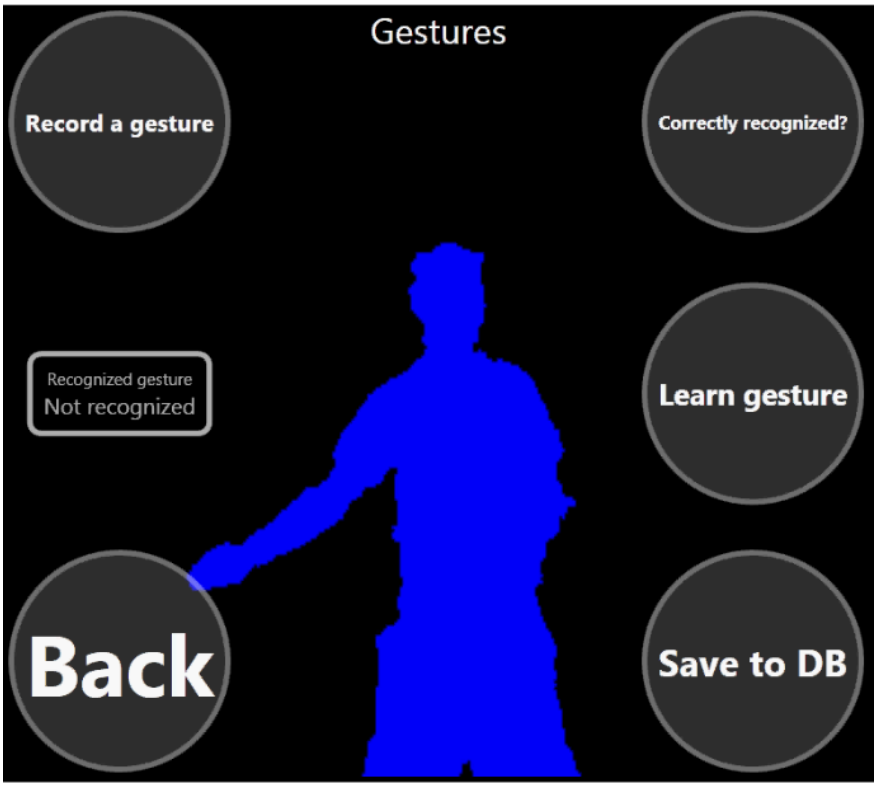
*Rysunek 20: Pierwszy ekran sekcji Gestures*

Po zatwierdzeniu przycisku **Record a gesture** wszystkie przyciski chowają się, a użytkownik ma czas na zaprezentowanie gestu (Rysunek 21). Po ustabilizowaniu gestu (użytkownik musi pozostać w nieruchomej pozycji z pewną tolerancją obliczoną na lekkie drganie rąk) i około 4-5 sekundach pojawiają się przyciski związane z decyzją podjętą przez sieć neuronową (Rysunek 22).



*Rysunek 21: Ekran nagrywania gestu*

Na ekranie odpowiedzi sieci neuronowej należy zwrócić uwagę na odpowiedź sieci neuronowej w komponencie IB. Jeżeli efekt jest zgodny z oczekiwaniem można zasygnalizować to aplikacji używając przycisku **Correctly recognized**. Jeżeli odpowiedź była błędna można określić prawidłowy gest przy użyciu ekranu wyboru gestu do którego można się dostać przy użyciu przycisku **Learn gesture**. Zatwierdzenie aktualnego gestu nie jest wymagane - może się zdarzyć, że zademonstrowano gest który w danej bazie gestów nie został zaprojektowany. W tym wypadku można wrócić do głównego ekranu (przycisk **Back**), lub nagrać kolejny gest (przycisk **Record a gesture**). Ostatnim przyciskiem ekranu odpowiedzi sieci neuronowej jest **Save to DB** powodujący zapisanie wszystkich zatwierdzonych do tej pory gestów do bazy danych. Można następnie udać się do sekcji **Learning** i nauczyć sieć neuronową zapisanych w bazie danych gestów.



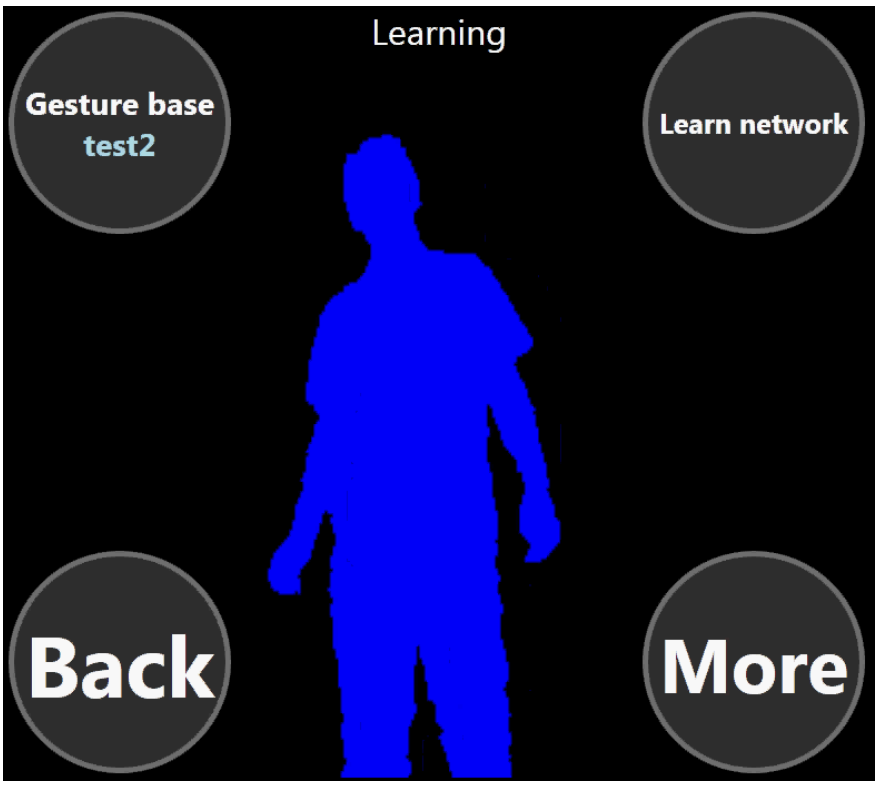

*Rysunek 22: Ekran odpowiedzi sieci neuronowej*

## Sekcja Learning

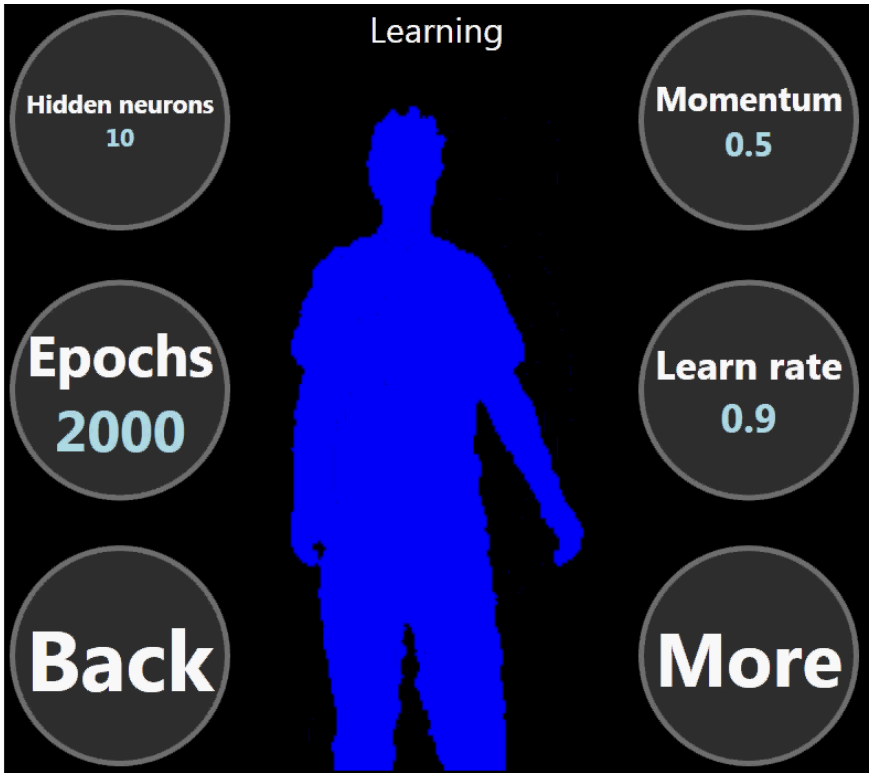
Sekcja Learning pozwala na wybór, naukę oraz modyfikację parametrów nauki sieci neuronowej. Na pierwszym ekranie sekcji Learning pojawiają się przyciski CMT i ECMT odpowiadające za wybór bazy gestów (**Gesture base**), naukę wybranej sieci neuronowej (**Learn network**), powrót do ekranu głównego (**Back**) oraz przejście do kolejnego ekranu ustawień (**More**) (Rysunek 23).

Zatwierdzenie przycisku **More** skutkuje pojawieniem się przycisków ECMT odpowiedzialnych za ustawianie parametrów sieci neuronowych (Rysunek 24).

Kolejne zatwierdzenie przycisku **More** powoduje powrót przycisków z pierwszego ekranu sekcji Learning.



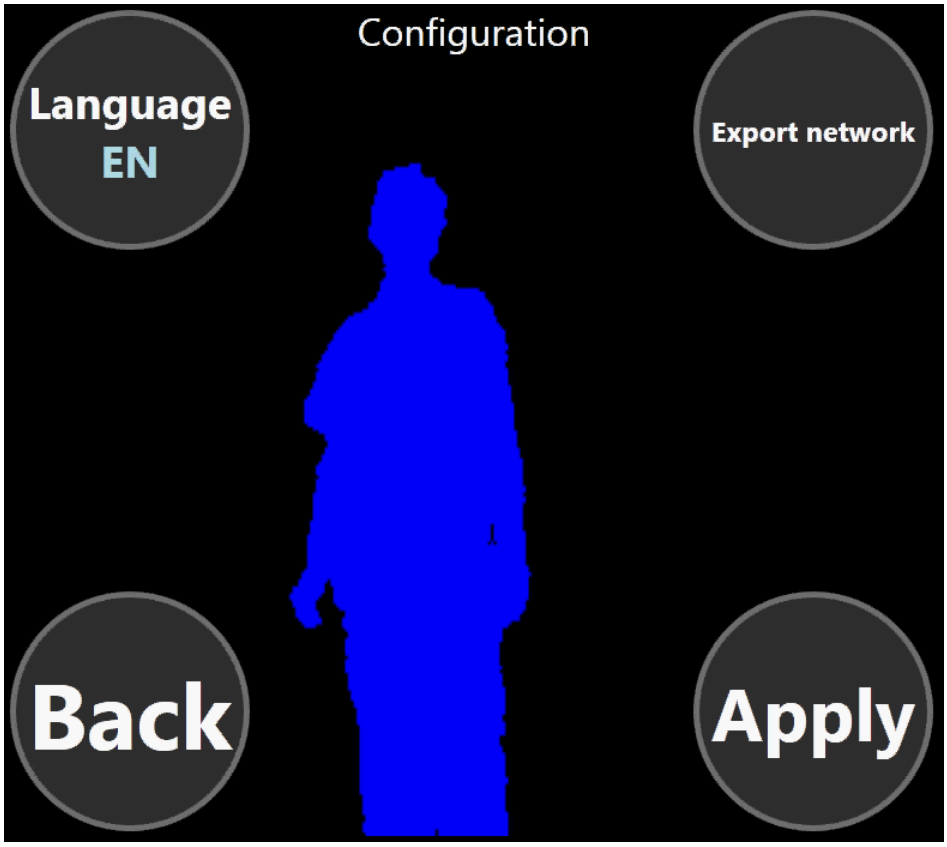
*Rysunek 23: Pierwszy ekran sekcji Learning*



*Rysunek 24: Ekran ustawień sieci neuronowych w sekcji Learning*

### Sekcja Configuration

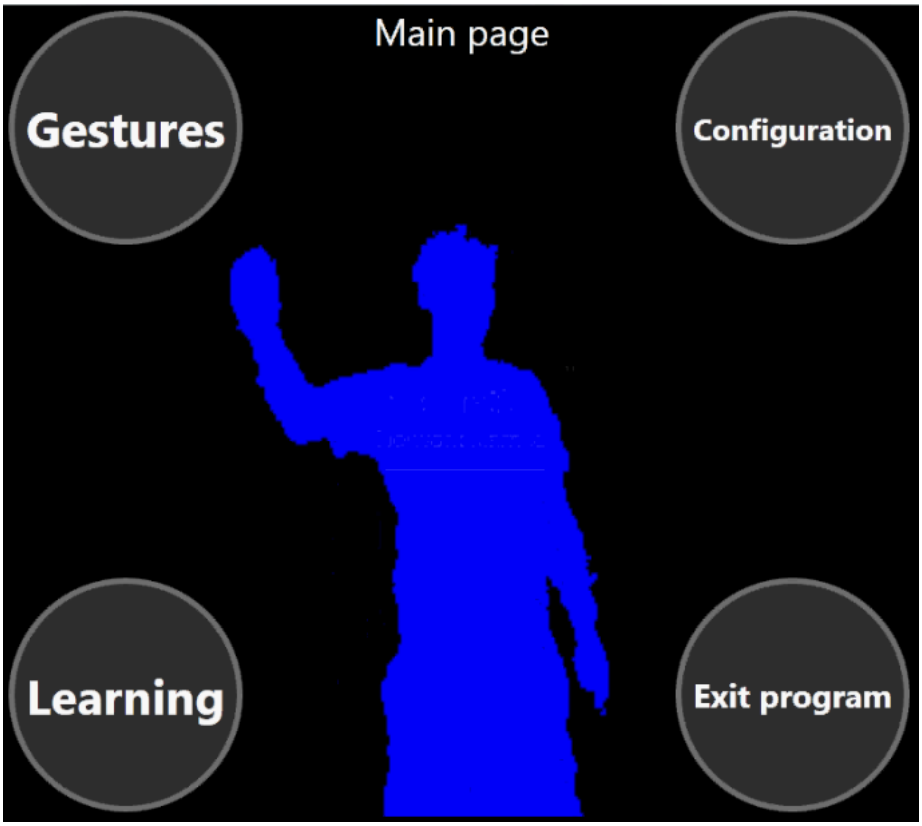
Sekcja Configuration zawiera ustawienia własne aplikacji GesturesRecognizer. W sekcji tej odnaleźć można ustawienia języka aplikacji (**Language**) oraz możliwość wyeksportowania sieci neuronowej do pliku (**Export network**) (Rysunek 25). Zserializowaną do formatu XML sieć można w przyszłości umieścić w innej aplikacji użytkowej, gdzie wyuczone gesty będą służyły do komunikacji z programem.




*Rysunek 25: Ustawienia aplikacji w sekcji Configuration*

## 5.2.2. Wygląd

Wygląd aplikacji utrzymywany jest w przejrzystym, ascetycznym klimacie. Większość ekranu przeznaczona jest dla wirtualnej reprezentacji użytkownika na czarnym tle. Poza tym pojawiają się jeszcze przyciski CMT, komunikaty IB oraz napis określający ekran na którym użytkownik się właśnie znajduje (Rysunek 26). Pod poszczególnymi przyciskami CMT kryją się kolejne ekrany aplikacji.



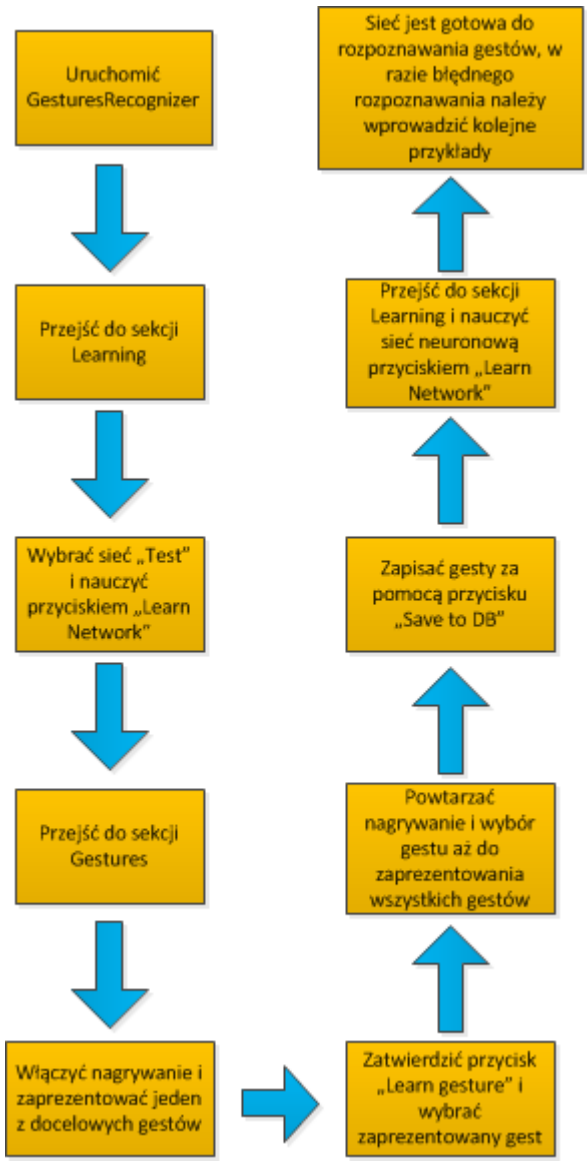
*Rysunek 26: Główny ekran aplikacji GesturesRecognizer*

### 5.2.3. Sterowanie

Sterowanie w aplikacji GesturesRecognizer jest w 100% realizowane przy użyciu ciała użytkownika. Najechanie dłońmi wirtualnej reprezentacji użytkownika na przyciski CMT i ECMT powoduje wykonanie określonej akcji. Podobnie gesty mogą być wykorzystane jako element generujący zdarzenia.

### 5.2.4. Przykładowy scenariusz

Przykładowy scenariusz jest kontynuacją przykładowego scenariusza dla aplikacji GesturesEditor. Zademonstrowany został na rysunku 27.




*Rysunek 27: Przykładowy scenariusz dla aplikacji GesturesRecognizer*

## 5.3. Problem rozpoznawania gestów

### 5.3.1. Definicja problemu

Rozpoznawanie gestów to rozpoznanie konkretnych wzorców na podstawie ułożenia ciała użytkownika. Rozpoznany wzorzec powinien zostać zwrócony użytkownikowi w zrozumiałej dla niego formie jako reakcja na akcję, będącą gestem. Może to być treść odpowiadająca gestowi (np. „Ręce w górze"), zdarzenie (np. zmiana koloru użytkownika, komunikat dźwiękowy), itp.

### 5.3.2. Szczegóły problemu

Do wykrycia układu ciała użytkownika wykorzystany został Microsoft Kinect. Tak jak opisano w rozdziale 3.2 Kinect zwraca strumień danych szkieletowych. Gest składać się powinien z określonej liczby części ciała (do ruchu ręką nie powinno być brane pod uwagę położenie nóg), w określonej liczbie wymiarów (3 wymiary dają większą przestrzeń i mogą skutkować większą różnorodnością gestów, ale 2 wymiary to mniej danych do analizy i potencjalnie szybsza odpowiedź programu). Kinect dostarcza 20 punktów na ciele w 3 wymiarach co oznacza 60 punktów do analizy. Jest to znacznie mniej niż np. w przypadku rozpoznawania obrazów, lub pisma i powinno pozwolić na szybką analizę danych. Dodatkowo projekt systemu rozpoznawania gestów musi brać pod uwagę następujące czynniki:

- Rozpoznawanie musi być realizowane w czasie rzeczywistym. Użytkownik nie może czekać dłużej niż kilka sekund na odpowiedź programu,
- Rozpoznawanie musi charakteryzować się odpowiednio dużą tolerancją na niedokładność gestu. Najlepiej jeżeli tolerancją taką dałoby się sterować w stosunku do potrzeb.

### 5.3.3. Metody rozpoznawania wzorców

Rozpoznawanie gestów wpisuje się w ogólną dziedzinę rozpoznawania wzorców. Do najczęściej używanych metod rozpoznawania wzorców zaliczają się **[A]:**

- klasyfikator K-najbliższych sąsiadów,
- estymator gęstości Parzena,
- liniowa i kwadratowa analiza dyskryminacyjna,
- sieci neuronowe,
- dopasowywanie wzorców.

W charakterystykę rozpoznawania gestów opisaną w poprzednim rozdziale najlepiej wpisuje się rozwiązanie oparte na sieciach neuronowych. Sieci neuronowe charakteryzują się dużą tolerancją na błędy – w zależności od metody i parametrów nauki oraz szybką odpowiedzią sieci, gdy ta jest już nauczona.

## 5.4. Szczegóły zaproponowanych rozwiązań

### 5.4.1. Metoda rozpoznawania gestów – sieć neuronowa

Najważniejszym elementem aplikacji rozpoznającej gesty jest, oczywiście, sam mechanizm rozpoznawania. Do realizacji tego celu wybrano sieci neuronowe. Z powodu niewielkiej ilości danych biorących udział w rozpoznaniu (maksymalnie 20 punktów na ciele w 3 wymiarach – 60 punktów) stosunkowo niewielka i prosta sieć powinna być w stanie podołać temu zadaniu. Zaproponowano sieć wielowarstwową (MLP - Multilayer Perceptron). MLP składa się z warstwy wejściowej, warstw ukrytych i warstwy wyjściowej i schematu połączenia każdy z każdym (Rysunek 28). Koncepcja wykorzystania tej sieci do rozwiązania problemu rozpoznawania gestów prezentuje się następująco:

**Warstwa wejściowa**

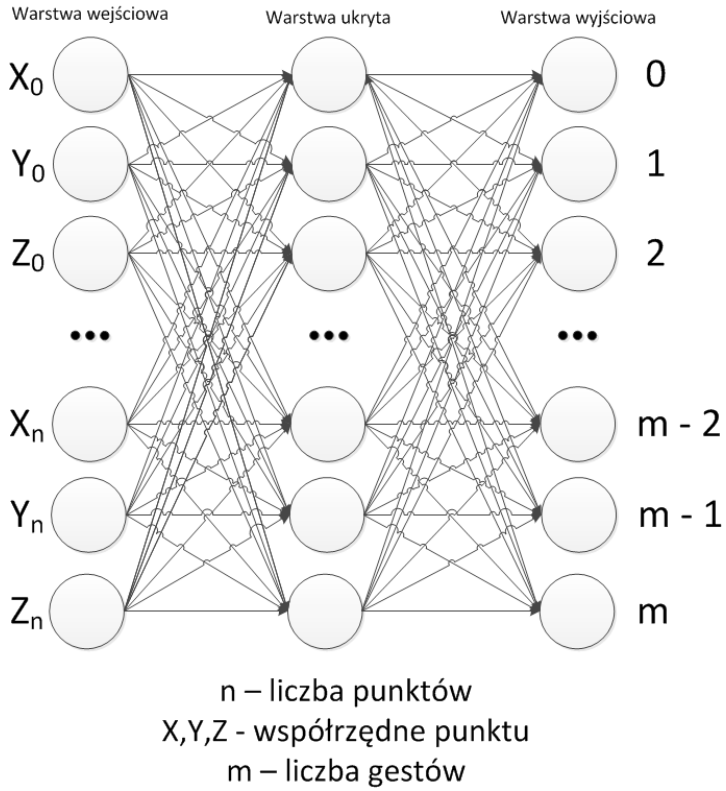
Warstwa wejściowa to posortowane współrzędne punktów na ciele użytkownika. Jeden neuron to jedna współrzędna. W przypadku uwzględnienia 3 wymiarów w przetwarzaniu układ wejściowy przydzielany jest według wzorca $X_0, Y_0, Z_0, X_1, Y_1, Z_1, \ldots, X_n, Y_n, Z_n$, gdzie n to indeks posortowanej listy punktów na ciele użytkownika. Dla 2 wymiarów kolejność będzie analogiczna, jednak bez współrzędnej Z.

**Warstwa ukryta**

W tej warstwie przebiegać będzie nauka sieci neuronowej. Po krótkich testach okazało się, że do rozwiązania tego problemu w zupełności wystarcza 10 neuronów w pojedynczej warstwie ukrytej.

**Warstwa wyjściowa**

Warstwa wyjściowa to posortowana lista identyfikatorów gestów. Każdy neuron odpowiada konkretnemu gestowi.




*Rysunek 28: Architektura wielowarstwowej sieci neuronowej zastosowanej w rozpoznawaniu gestów*

**Konstruowanie sieci**

Sieć neuronowa konstruowana jest na bazie metadanych określających liczbę punktów na ciele i liczbę wymiarów (warstwa wejściowa), liczbę gestów (warstwa wyjściowa) oraz pozostałych parametrów (liczba epok, liczba neuronów w warstwie ukrytej, współczynnik uczenia, bezwładność, funkcja wejścia, itp). Każdy gest ma nazwę i identyfikator który równy jest identyfikatorowi neuronu, aby można je było wzajemnie odwzorowywać.

**Nauka sieci**

Użytkownik prezentuje gest. Po przechwyceniu gestu użytkownik określa który z dostępnych gestów właśnie zaprezentował. Użytkownik prezentuje kolejny gest, aż do wyczerpania listy gestów do nauki. Użytkownik może podać tyle wersji danego gestu ile zechce, jednak musi brać pod uwagę, że gest źle oznaczony będzie zaburzał proces rozpoznawania w sieci. Po zakończeniu wprowadzania danych aplikacja wprowadza dane do sieci neuronowej w losowej kolejności. Aby zapewnić jak największą entropię zbioru gestów do sortowania zastosowano algorytm Fisher'a-Yates'a [17]. Posiada on złożoność liniową, co gwarantuje szybkość sortowania, a dodatkowo wszystkie warianty rozłożenia wartości są równo prawdopodobne [18]. Następnie przez

zadaną liczbę epok (domyślnie 2000) sieć uczy się zadanego zbioru danych. Czas trwania nauki zależy od wielkości zbioru danych i liczby epok, jednak przy podstawowym zbiorze danych i domyślnych ustawieniach trwa to zaledwie kilka sekund. Nauczona sieć gotowa jest do rozpoznawania gestu. Samo rozpoznanie trwa zaledwie kilkaset milisekund.

**Decyzja sieci**

Wynikiem wykonania przetwarzania w sieci neuronowej jest wektor odpowiedzi. Jego liczność to liczba możliwych gestów do rozpoznania. Wartości wektora to liczby zmiennoprzecinkowe z przedziału (0,1). Czym wyższa wartość tym pewniejsza odpowiedź sieci dotycząca danego gestu. Aplikacja przegląda wektor w poszukiwaniu najwyższej wartości odpowiedzi, po czym ostatecznie określa który z gestów był rozpoznawany. W wektorze musi być jednak co najmniej jedna odpowiedź przekraczająca wartość 0,85. Jeżeli nie ma takiej odpowiedzi w wektorze rozumiane jest to jako niezdecydowanie sieci, a użytkownikowi zwracana jest odpowiedź „Nie rozpoznano”.

### 5.4.2. Struktury danych

Zaraz po zatwierdzeniu gestu w programie GesturesRecognizer gest przechowywany jest na liście w postaci obiektu klasy LearnElement, która zawiera tablice **Input** oraz **Output** wartości zmiennoprzecinkowych. Tablica **Input** zawiera informacje o pozycjach kolejnych współrzędnych kolejnych części ciała. Tablica **Output** to wektor określający który z gestów odpowiada wartościom z tablicy **Input**. W przypadku zapisu gestów do bazy danych lista obiektów LearnElement zostaje zserializowana i zapisana w bazie danych.

### 5.4.3. Magazynowanie danych

Wszystkie dane dotyczące gestów przechowywane są w bazie danych MySQL 5.5. Ułatwia to wymianę informacji między aplikacjami oraz gwarantuje przechowanie danych po zamknięciu aplikacji.

Wszystkie dane przechowywane są w pojedynczej tabeli n**eural_gestures**. Tabela ta zawiera pola:

- **idneural_gestures** – unikatowy identyfikator,
- **gesture_base_name** – nazwa zbioru gestów,
- **gestures_map_xml** – zserializowana lista metadanych gestów zawierających informację o

nazwie gestu, identyfikatorze gestu, liczbie wejść i liczbie wyjść sieci neuronowej,

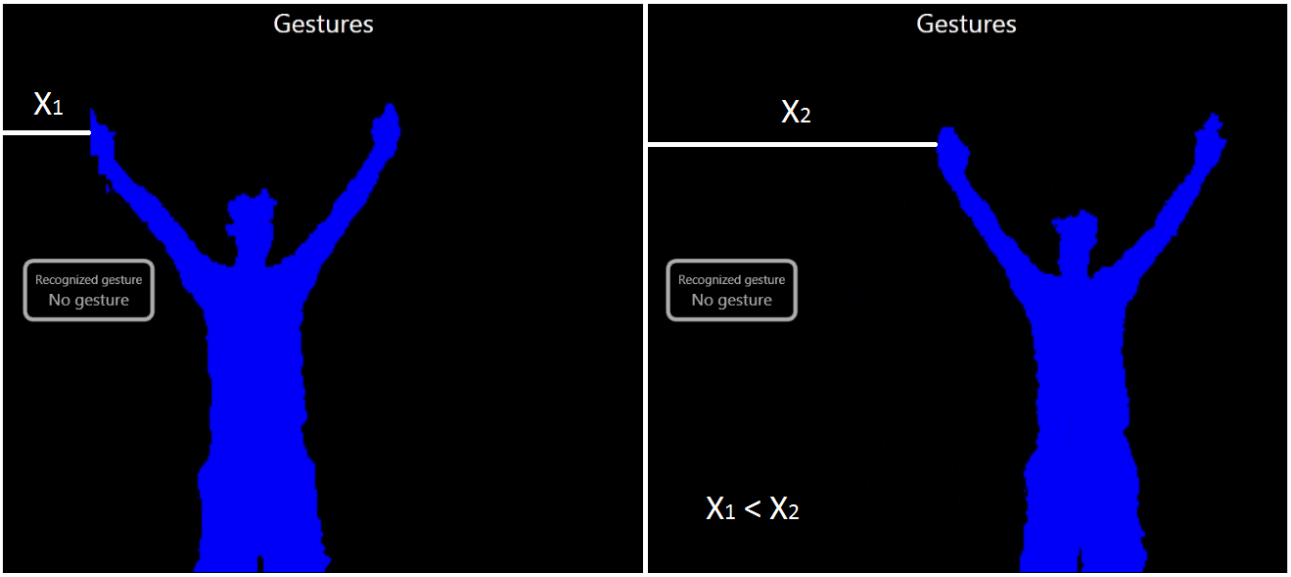
- **learn_xml** – zserializowana lista elementów uczących sieć neuronową.

Baza danych przechowuje jedynie informacje służące do nauki sieci. Użytkownik ma możliwość wyeksportować zserializowaną, nauczoną sieć neuronową do pliku w celu dalszych zastosowań. Aplikację przygotowaną w ramach tej pracy można wykorzystać jako aplikację przygotowującą sieć neuronową do wykorzystania w innych aplikacjach.

### 5.4.4. Metoda zbierania danych

Specyficznym problemem okazało się zbieranie danych identyfikujących gest. Nie chodzi tu o samą metodę przechwytywania gestu, która opisana została w rozdziale 5.4.5, ale o sposób zbierania danych tak, aby niezależnie od warunków fizycznych użytkownika oraz jego położenia w przestrzeni gest był identyfikowany tak samo.

Początkowo dane zbierane były w dwóch wymiarach poprzez odczyt wartości położenia pikseli w ramach klatki przechwytywanej przez Kinekta. Okazało się, że w zależności od tego, w którym miejscu w stosunku do kamery stanął użytkownik gest był nie był identyfikowany jednoznacznie, pomimo podobnego położenia rąk. Problem ten przedstawiono na rysunku 29**.**




*Rysunek 29: Problem przesunięcia ciała użytkownika*

Pomimo, że użytkownik pokazywał, w swoim mniemaniu, ten sam gest, to położenie jego ciała w stosunku do kamery zmieniło się. Aby wyeliminować ten problem należało zmienić punkt odniesienia z kamery na samego użytkownika. W tym celu wprowadzono punkt odniesienia oparty

na centralnym punkcie ciała – splocie słonecznym. Jego wprowadzenie było stosunkowo proste z uwagi na dostarczanie danych o jego położeniu przez sam sensor Kinect. Położenie tego punktu przedstawiono na rysunku 31, gdzie oznaczony jest jako SPINE[2]. Następnie należało określić położenie pozostałych punktów na ciele w porównaniu do punktu centralnego. Ponieważ zaproponowana sieć neuronowa przyjmuje na wejściu wartości (0,1) położenie punktów na ciele w stosunku do punktu centralnego musiało mieć wartości znormalizowane do tego przedziału. Jeżeli założymy, że wyprostowane ręce rozłożone wzdłuż jednej osi są średnicą okręgu o długości 1 oraz, że potrafią nakreślić okrąg wokół punktu centralnego to zbiór takich osi w 3 wymiarach pozwoli na nakreślenie sfery wokół tego punktu. Jeżeli wpiszemy taki okrąg w sześcian możemy otrzymać figurę o bokach o długości 1. Takie rozwiązanie oznacza, że punkt centralny ma wartość 0,5 w wymiarach X,Y,Z, a dłonie wartości z przedziału (0,1) w tych wymiarach. Oczywistym jest, że człowiek z powodu ograniczeń ruchowych w stawach nie potrafi wykręcać rąk o 360 stopni w każdej osi oraz że splot słoneczny nie jest idealnym środkiem okręgu kreślonego przez dłonie. Zastosowanie kalibracji aplikacji w celu przechwycenia dokładnej długości rąk dla danego użytkownika pozwoliłoby, aby to uogólnienie okazało się wystarczające do wykorzystania go w dalszym przetwarzaniu.

Z uwagi na możliwości sieci neuronowej do tolerancji błędów oraz aspekt eksperymentalny rozwiązania ewentualną potrzebę kalibracji aplikacji pozostawiono do wykonania w przyszłości. Okazało się bowiem, że wartości korekcyjne ustawione w aplikacji w celu jak najwierniejszego uzyskania długości rąk okazały się na tyle skuteczne, że tymczasowo porzucono problem kalibracji.

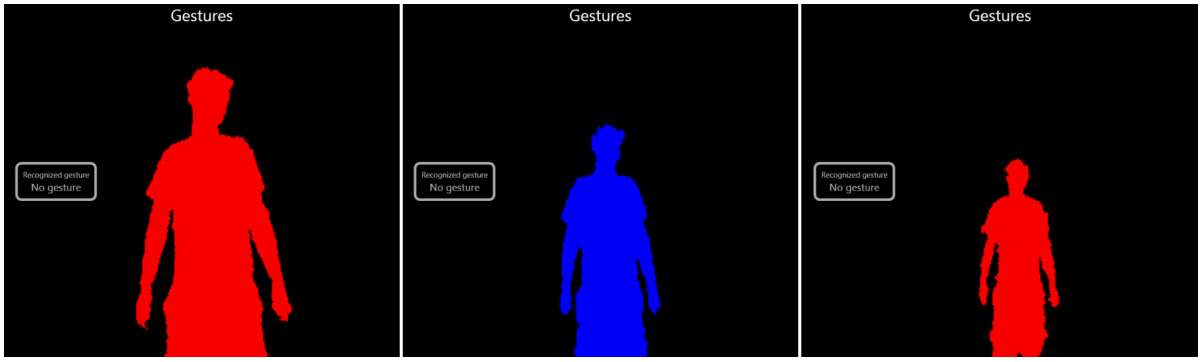
Sposób ten opiera się na przydzieleniu 500 pikseli jako maksymalnej odległości między dłońmi w ramach osi X oraz osi Y. Odległość w osi Z mierzona jest w metrach, więc w tej osi wprowadzono maksymalną odległość równą 2 metry. Następnie mierzona jest rzeczywista odległość między dłońmi w 3 osiach, a następnie dzielona przez wartość maksymalną. Jeżeli wartość ta nie znajduję się w przedziale (0,1) jest ona ustawiana na wartość minimalną, lub maksymalną z tego przedziału.

Dodatkowym kłopotem pozostawała odległość użytkownika od kamery. W zależności od odległości jego obraz maleje lub rośnie. Wyczerpująca kalibracją mogłaby być rozwiązanie problemu, jednak taka kalibracja mogłaby być niewygodna dla użytkownika i wymagałaby skomplikowanego procesu zarządzania. Zamiast tego wprowadzono oznaczenia dla użytkownika.

2 Kwestią do dyskusji jest czy lepszym punktem odniesienia byłby punkt oznaczony na rysunku jako SHOULDER_CENTER. Nie jest to jednak problem implementacyjny, a jedynie kwestia który punkt lepiej spełnia swoją rolę, a punkt SPINE okazał się na tyle użyteczny, że nie było potrzeby sprawdzania innych punktów.

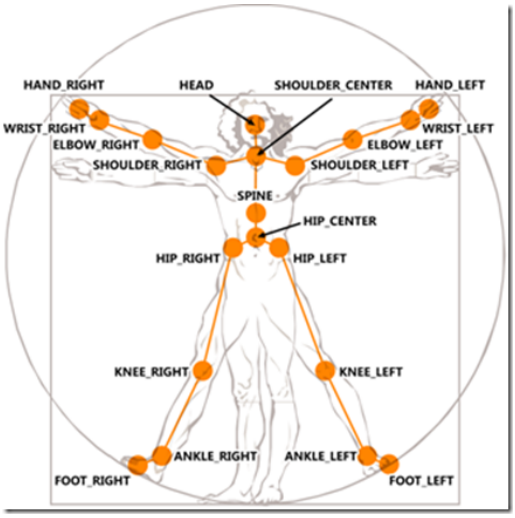
Określono odległość optymalną w przedziale od 1,8 do 2,5 metra. Jeżeli użytkownik znajduje się w tej odległości od sensora – jego kolor jest niebieski. Jeżeli wykracza poza ten przedział – jest czerwony. Oczywiście obraz użytkownika w ramach tego przedziału będzie się zmieniał, jednak po raz kolejny tolerancja sieci neuronowych na niewielkie zmiany okazała się zbawienna i stwierdzono, że nie wpływa to znacząco na jakość rozpoznania. Należy zauważyć, że kolor ma jedynie aspekt informacyjny, a aplikacja zezwala na nagrywanie gestów poza optymalnym przedziałem odległości.

Z powyższego opisu wynika, że w rozpoznawaniu biorą udział jedynie punkty na dłoniach. W celu urozmaicenia gestów wprowadzono także punkty na łokciach. Nie są one jednak tak samo normalizowane jak dłonie a jedynie wpisują się w nakreśloną dłońmi sferę. Oznacza to, że nie jest możliwe osiągnięcie łokciami wartości progowych z przedziału (0,1). W dalszych pracach można spróbować wprowadzić taką normalizację dla łokci, jednak w zaproponowanym rozwiązaniu brak tej metody nie wpłynął zauważalnie na jakość rozpoznawania gestów.




*Rysunek 30: Identyfikator optymalnej odległości w postaci koloru użytkownika (kolor niebieski oznacza poprawną odległość od sensora Kinect)*

Wprowadzenie do procesu rozpoznawania pozostałych punktów dla których mostek jest punktem centralnym nie powinno stanowić problemu, jednak problemem może być wykorzystanie nóg. Nogi w najlepszym wypadku mogłyby wykorzystać punkt oznaczony na rysunku 3. jako HIP_CENTER, jednak mogłoby się to okazać niewystarczające, aby wykryć subtelne zmiany ruchu nóg przy rozpoznawaniu biegania, chodzenia, kopania itp. Jest to ciekawy aspekt do dalszych badań.




*Rysunek 31: Punkty na ciele użytkownika zwracane jako strumień danych przez Kinecta[5]*

### 5.4.5. Metoda przechwytywania gestu

Projektując system rozpoznawania gestów należało zastanowić się w jaki sposób informować system o zakończeniu prezentacji gestu. Ponieważ cały system sterowany jest przy użyciu ciała nie można było wykonać żadnego ruchu w celu identyfikacji zakończenia gestu ponieważ zaburzyłoby to sam gest i wymagałoby skomplikowanego przetwarzania nagranego gestu w celu eliminacji niepożądanych ruchów. Potencjalnym rozwiązaniem problemu mogłoby być wykorzystanie macierzy mikrofonów umieszczonych w Kinekcie oraz dostarczonej wraz z oficjalnymi bibliotekami funkcjonalności rozpoznawania mowy. Wypowiedzenie określonego słowa oznaczałoby zakończenie gestu. Rozwiązanie to wydaje się możliwe do zastosowania jednak mogłoby powodować dodatkowe niedogodności takie jak konieczność zachowania ciszy w pomieszczeniu, lub dokładnej synchronizacji wymawianego słowa z prezentowanym gestem. Z uwagi na to, że aplikacja projektowana była z perspektywą dalszego zastosowania zaimplementowanych w niej rozwiązań, takie podejście do problemu okazało się nazbyt problematyczne.

Ostatecznie zaproponowano rozwiązanie oparte na przechwytywaniu klatek[3] i

3 W zaprezentowanym rozumowaniu pojęcie „klatki” należy rozumieć jako zebranie w danym momencie czasu danych o położeniu podanych punktów na ciele użytkownika, względem punktu

porównywaniu ich do klatek poprzednich w celu określenia zakończenia gestu. Oznacza to, że zakończenie gestu rozpoznawane jest jako zakończenie ruchu użytkownika. Metoda zaimplementowana w programie pozwala jedynie na rozpoznawanie gestu statycznego jednak potencjalnie możliwe jest wykorzystanie jej do nagrania gestu dynamicznego.

Poniżej przedstawiono algorytm przechwytywania gestu statycznego (algorytm przedstawiono także na rysunku 32).

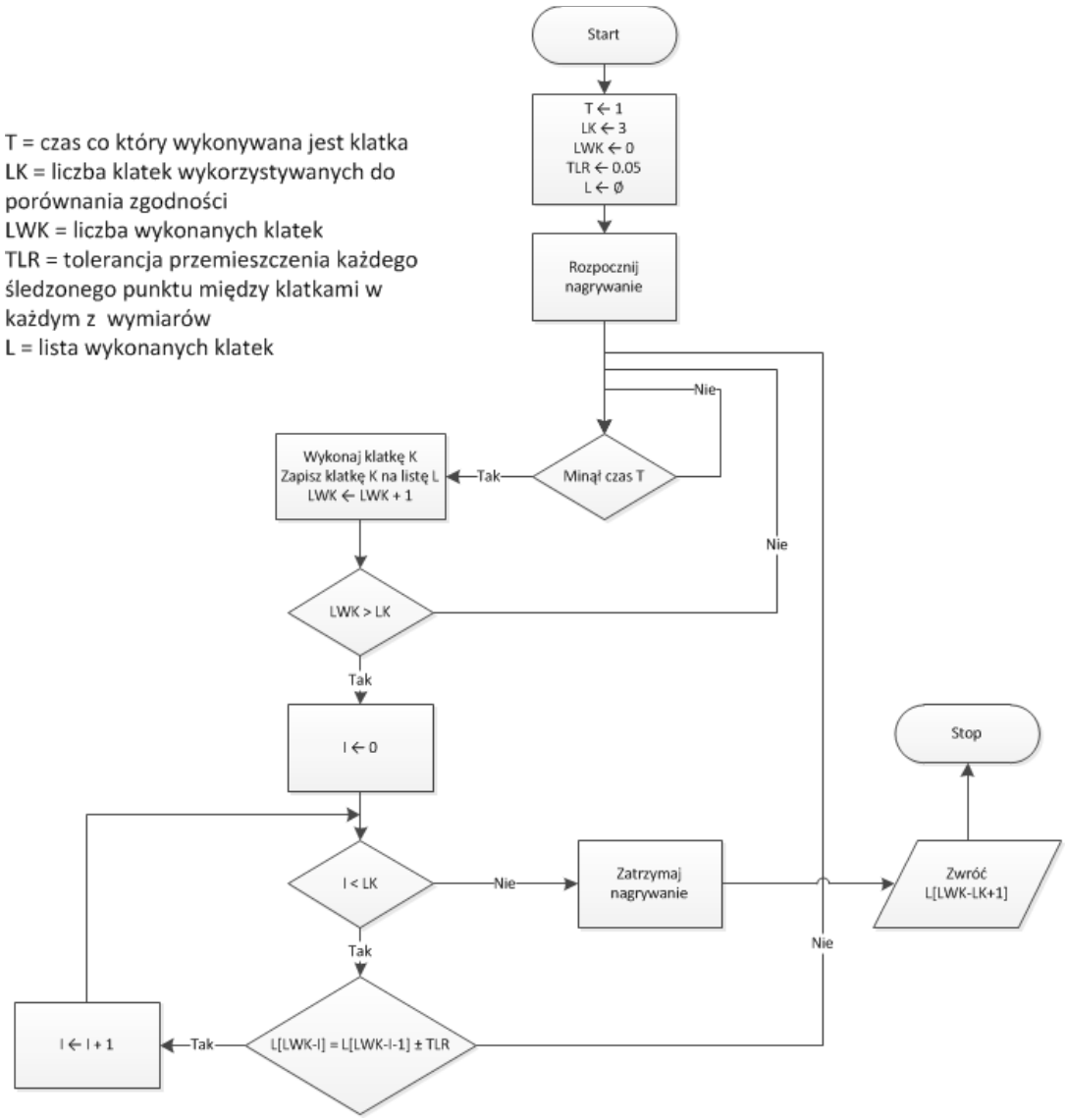
centralnego SPINE, w przestrzeni o podanej liczbie wymiarów.

**ALGORYTM**

**Dane wejściowe:** klatki K przechwytywane przez aplikację.

**Dane wyjściowe:** klatka K będąca reprezentacją gestu.

1. Niech T oznacza czas co który wykonywana jest klatka, niech LK oznacza liczbę klatek wykorzystanych do porównania zgodności, niech LWK oznacza liczbę wykonanych klatek, niech TLR oznacza tolerancję przemieszczenia każdego śledzonego punktu między klatkami w każdym z wymiarów, niech L oznacza listę wykonanych klatek.
2. Ustaw początkowe wartości zmiennych: T = 1 sekunda, LK = 3, LWK = 0, TLR = 0,05 metra, L = pusta lista.
3. Uruchom nagrywanie gestu i wykonuj klatki K co czas T.
4. W każdym momencie wykonania klatki K wykonuj:
   (a) dodaj klatkę K do listy L,
   (b) inkrementuj LWK o 1,
   (c) jeżeli LWK większe od LK:
      i. dla każdego I od 0 do LK:
         A. porównaj L[LWK – I] do L[LWK – I – 1] z tolerancją TLR i zwróć wartość TRUE jeżeli porównanie wskazuje na identyczność, lub FALSE w przeciwnym przypadku,
         B. jeżeli FALSE przerwij pętlę.
      ii. jeżeli TRUE przechwyć klatkę K z listy L, taką że K = L[LWK – LK + 1], uznaj ją jako klatkę reprezentującą gest i zakończ przetwarzanie.




*Rysunek 32: Schemat blokowy algorytmu przechwytywania statycznego gestu*

Algorytm ten jest ograniczony kilkoma zmiennymi dostrojonymi w sposób empiryczny:

C**zas (T) co który zbierane są klatki wynosi 1 sekundę**, co w przypadku gestów statycznych nie jest czasem zbyt długim ponieważ nie występuje w nim ruch, ani zbyt krótkim ponieważ można się spodziewać, że użytkownik wykonał już w tym czasie odpowiedni gest.

**Liczba klatek (LK) wykorzystywanych do porównania zgodności wynosi 3**. Zmienna ta wiąże się nierozłącznie z czasem co który zbierane są klatki. Przy obecnych ustawieniach: czas – 1 sekunda, liczba klatek – 3, oznacza to co najmniej 3 sekundy podczas których użytkownik musi stać nieruchomo (z dokładnością do zmiennej tolerancji opisanej poniżej), prezentując dany gest w celu jego zatwierdzenia.

**Tolerowane (TLR) przemieszczenie każdego śledzonego punktu między klatkami w każdym wymiarze wynosi 5 centymetrów.** Zmienną tą wprowadzono, aby uwzględnić czynnik

ludzki podczas nagrywania gestu. Żaden człowiek nie stoi idealnie nieruchomo przez kilka sekund i może zdarzyć się drobne poruszenie danej części ciała, które nie powinno mieć wpływu na odczytanie gestu.

W ostatnim kroku algorytmu klatką reprezentującą gest została klatka będąca pierwszą klatką biorąca udział w ostatnim przetwarzaniu w celu rozpoznania braku ruchu. Wybrano ją z tego powodu, iż wraz z oczekiwaniem kilku sekund pozycja użytkownika mogła się lekko zmienić i wydaje się, że pierwsza klatka z klatek identyfikujących gest będzie go najwierniej odzwierciedlać.

# 6. Eksperymenty

W celu zbadania efektywności sensora Kinect w parze z sieciami neuronowymi w rozpoznawaniu gestów statycznych zaplanowane zostały eksperymenty badawcze.

## 6.1. Założenia do eksperymentów

Aby rozpoznawanie gestów było użyteczne w aplikacjach musi spełniać poniższe założenia.

### 6.1.1. Dokładność

Aby gest można było uznać za dokładnie rozpoznany odpowiedź programu musi być dostrojona do oczekiwań użytkownika. Przesunięcie dłoni o kilka centymetrów w dowolnym wymiarze nie powinno mieć wpływu na rozpoznanie gestu polegającego na przykład na rozłożeniu rąk wzdłuż jednej osi. Z drugiej strony ręce złożone jak do modlitwy i rozsunięte o kilkanaście centymetrów, co mogłoby sugerować trzymanie jakiegoś przedmiotu, powinny oznaczać odpowiednio zróżnicowaną odpowiedź programu.

### 6.1.2. Szybkość

Czas odpowiedzi programu w dowolnym momencie nie może wynosić więcej niż kilka sekund. Dotyczy to zarówno nauki sieci neuronowej, wprowadzania nowego gestu jak i rozpoznania prezentowanego gestu.

### 6.1.3. Pewność

Odpowiedź programu po rozpoznaniu gestu powinna być powtarzalna. Ten sam gest rozpoznawany n razy powinien prowadzić do niemal n poprawnych odpowiedzi. Ewentualną nieskuteczność powinna dać się skalibrować poprzez mechanizmy zawarte w aplikacji.

## 6.2. Charakterystyka danych wejściowych

Charakterystyka informacji przekazywanych do programu poprzez układ ciała użytkownika ma pewne specyficzne własności. Podczas gdy przy standardowej obsłudze aplikacji przy użyciu klawiatury, myszki, czy panelu dotykowego można dokładnie określić i zautomatyzować wprowadzanie danych wejściowych, przy wprowadzaniu gestów użytkownik sam wprowadza pewną nieścisłość do danych, patrząc z perspektywy maszyny. Przede wszystkim żadne dwa gesty nie są identyczne. Dane wpływające do aplikacji po odczytaniu położenia części ciała użytkownika z dokładnością do milimetrów w wielu wymiarach raczej nie zostaną nigdy powtórzone. W tym przypadku należy mówić o pewnym zbiorze możliwych pozycji punktów, które powinny przełożyć się na konkretną odpowiedź systemu. Niestety nie można z góry określić z zadowalającą dokładnością jakie będą przedziały tego zbioru. Użytkownik sam je wyznacza, świadomie, lub nieświadomie, i oczekuje konkretnej odpowiedzi. Z tego powodu dane do eksperymentów wprowadzane będą w ten sam sposób w jaki użytkownik będzie korzystał z aplikacji – poprzez prezentowanie gestów, które z perspektywy użytkownika są identyczne, natomiast z perspektywy maszyny – niekoniecznie.

Dodatkowo należy uściślić definicję gestu, który ma zostać rozpoznany:

### 6.2.1. Gest

Gest w ramach aplikacji GesturesRecognizer należy rozumieć jako zbiór punktów 4 części ciała (lewa dłoń, lewy łokieć, prawa dłoń, prawy łokieć) podanych w 2, lub 3 wymiarach, który posiada swoją reprezentację tekstową ustawioną wcześniej w aplikacji GesturesEditor. Gest jest spójny w zakresie odległości od centralnej części ciała będącą reprezentowaną przez mostek. Gest jest niezależny od przesunięcia ciała jako całości w dowolnej z osi trójwymiarowego układu współrzędnych w ramach akceptowalnych przez aplikację (wszystkie części ciała muszą być widoczne przez kamerę, a odległość użytkownika od kamery może wahać się między 1,8 a 2,5 metra, co jest sygnalizowane przez aplikację kolorem użytkownika na ekranie).

## 6.3. Cele eksperymentów

Eksperymenty mają wykazać użyteczność sensora Microsoft Kinect w połączeniu z sieciami neuronowymi w rozpoznawaniu gestów. Należy udowodnić, że:

- rozpoznawane gesty są zgodne z definicją gestu opisaną w rozdziale 6.2,
- skuteczność rozpoznawania gestów w nauczonej sieci jest większa niż 90%,

- czas oczekiwania użytkownika w dowolnym momencie przetwarzania danych nie jest większy niż 3 sekundy.

Wszystkie scenariusze zostaną przetestowane dla przestrzeni dwu- oraz trójwymiarowej, aby wykazać zalety i wady obu podejść oraz ich komplementarność w zależności od wymagań.

## 6.4. Scenariusze eksperymentów

Scenariusze eksperymentów muszą spełniać założenia podane w rozdziale 6.1.

### 6.4.1. Scenariusz 1 – Rozróżnianie wielu zróżnicowanych gestów

#### Cel

Celem scenariusza jest wykazanie podstawowej funkcjonalności aplikacji w zakresie rozróżniania gestów, których składowe części ciała są od siebie znacząco wektorowo oddalone w przestrzeni dwu-, lub trójwymiarowej. Zademonstrowane zostaną różnice, lub ich brak w zakresie rozpoznawania gestów, czasu nauki sieci oraz czasu odpowiedzi sieci.

#### Przebieg

Eksperyment polega na wymodelowaniu 3 gestów w przestrzeni dwu- oraz trójwymiarowej. Są to fizycznie dwie różne sieci neuronowe. Gesty, które zostaną zaprezentowane w tym eksperymencie prezentują się następująco:

1. Ręce szeroko,
2. Ręce do modlitwy,
3. Ręce jak siłacz,
4. Ręce do przodu,
5. Ręka na czole,
6. Ręce na biodrach.

Dodatkowo zaprezentowane zostaną dane dotyczące czasu rozpoznawania gestu przez aplikację.

### 6.4.2. Scenariusz 2 – Rozróżnianie gestów przy przesunięciu ciała względem sensora Kinect

#### Cel

Celem scenariusza jest zademonstrowanie niezależności położenia ciała względem kamery od osiąganych wyników.

#### Przebieg

Eksperyment przeprowadzany będzie przy wykorzystaniu sieci neuronowych nauczonych w scenariuszu 1. Następnie wprowadzone zostaną dane odpowiadające tym samym gestom, ale przesuniętym w osiach X, Y oraz Z (w przypadku gestów trójwymiarowych).

### 6.4.3. Scenariusz 3 – Rozróżnianie gestów w osi Z

#### Cel

Celem eksperymentu jest zademonstrowanie rzeczywistych różnic w rozpoznaniu gestów zróżnicowanych jedynie pod względem pozycji w osi Z. Sieć nauczona gestów dwuwymiarowych powinna błędnie odpowiadać na poszczególne gesty z uwagi na zbyt małe przesunięcia w osiach X oraz Y. Sieć nauczona gestów trójwymiarowych powinna poradzić sobie z problemem bezbłędnie.

#### Przebieg

Zostanie utworzona nowa sieć neuronowa o gestach niemal identycznych w wymiarach X oraz Y i znacznie zróżnicowanych w osi Z. Gesty prezentowane w ramach tego scenariusza prezentują się następująco:

1. Ręka z przodu,
2. Ręka pośrodku,
3. Ręka z tyłu.

### 6.4.4. Scenariusz 4 – Rozróżnianie gestów podobnych

#### Cel

Celem eksperymentu jest demonstracja możliwości aplikacji przy rozróżnianiu gestów podobnych. Przez podobieństwo należy rozumieć przesunięcie rzędu kilkunastu centymetrów w jednej, lub kilku osiach układu współrzędnych, w jednym, lub wielu punktach.

#### Przebieg

Zostaną utworzone nowe sieci neuronowe w celu zbadania możliwości rozpoznania gestów podobnych. Gesty, które zostaną zaprezentowane w tym eksperymencie prezentują się następująco:

1. Ręka na prawej piersi,
2. Ręka na lewej piersi,
3. Ręka na brzuchu.

### 6.4.5. Scenariusz 5 – Rozpoznawanie gestów uniwersalnych

#### Cel

Celem eksperymentu jest nauczenie sieci rozpoznawania gestów, które mogą wydawać się na pierwszy rzut oka różne, jednak użytkownik wymaga by były rozpoznawane tak samo. Przykładem takiego gestu może być podniesienie ręki do góry – lewej, lub prawej – rozpoznawane jako ten sam gest.

#### Przebieg

Utworzona zostanie nowy zbiór gestów. Następnie sieć pozna pojedyncze przykłady gestów, które nie wyczerpują możliwości związanych z prezentacją gestu. Po nauce sieć będzie odpowiadała na prezentowane gesty, zarówno w formie już nauczonej, jak i drugiej, nieznanej dla sieci. W przypadku błędnego rozpoznania sieć pozna kolejne przykłady i zostanie przebadana po raz kolejny. W niniejszym scenariuszu prezentowane będą następujące gesty:

1. Dowolna ręka w bok,
2. Dowolna ręka na brzuchu,
3. Skręcający samolot.

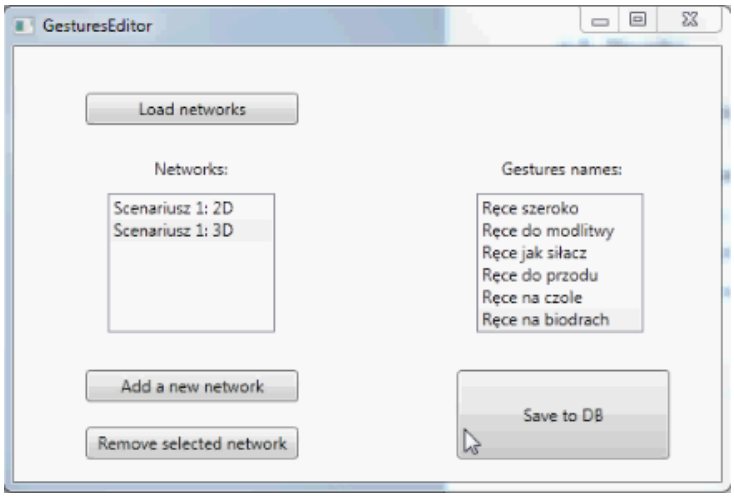
### 6.4.6. Zestawienie czasów przetwarzania podczas przeprowadzania eksperymentów

Zgodnie z założeniami w dowolnym momencie użytkowania aplikacji użytkownik nie może czekać na odpowiedź systemu dłużej niż 3 sekundy. W tym zestawieniu podane zostaną czasy przetwarzania dla najbardziej czasochłonnych operacji – nauki sieci neuronowej oraz zapisu gestów do bazy danych – zmierzone podczas przeprowadzania eksperymentów.

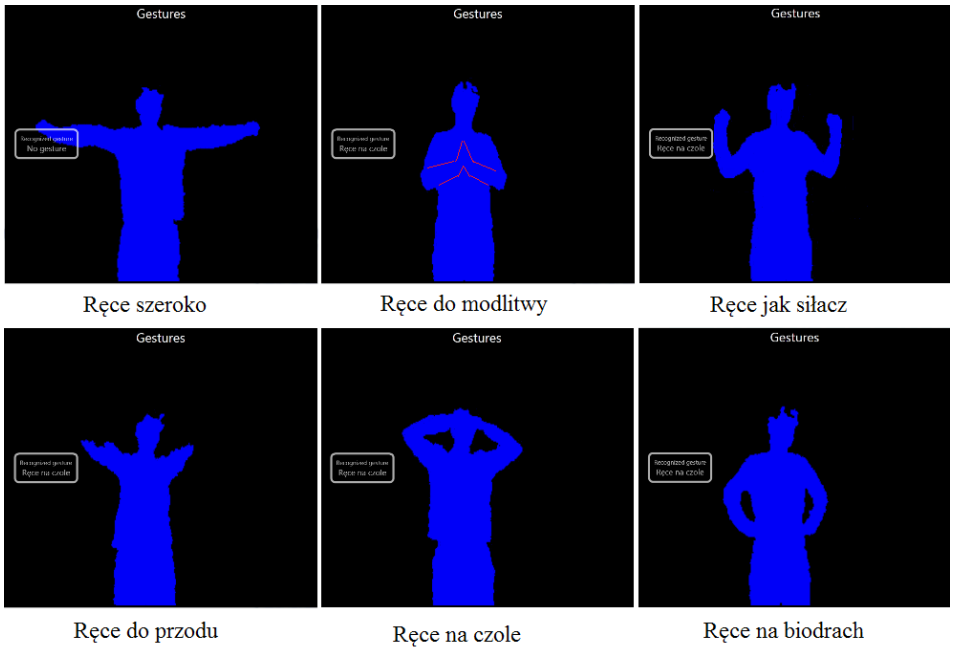
## 6.5. Wyniki

### 6.5.1. Scenariusz 1 – Rozróżnianie wielu zróżnicowanych gestów

Projekt bazy gestów utworzony w programie GesturesEditor zaprezentowano na rysunku 33. Następnie rozpoczęto naukę sieci neuronowej w programie GesturesRecognizer. Gesty prezentowane w scenariuszu pokazane zostały na rysunku 34. Ostatecznie osiągnięto wyniki zaprezentowane w tabeli 6.1.




*Rysunek 33: Projekt baz gestów w Scenariuszu 1.*




*Rysunek 34: Zaprezentowane gesty w Scenariuszu 1. w przestrzeniach dwu- i trójwymiarowej*

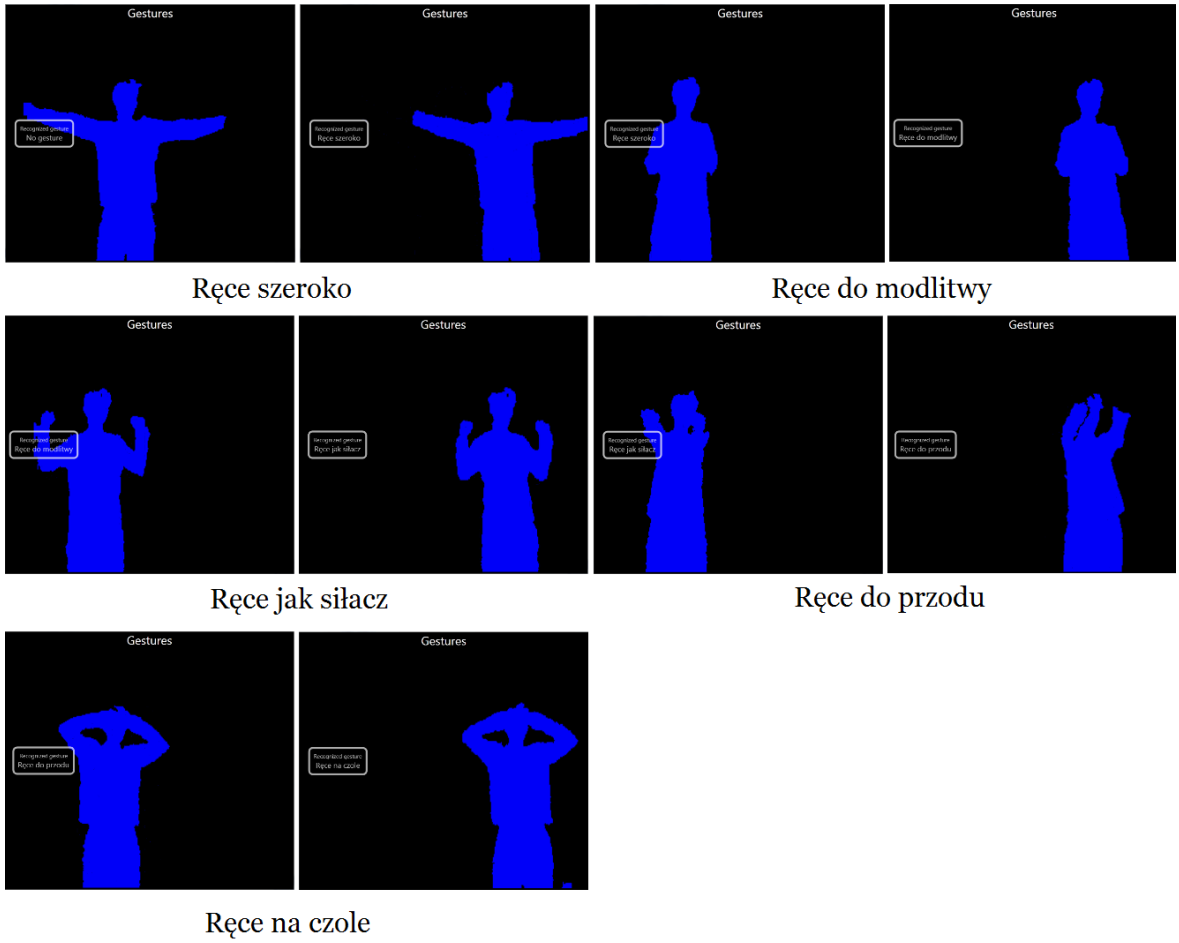
*Tabela 6.1: Wyniki dla Scenariusza 1.*

| Rozpoznawany gest | Przestrzeń 2D | | Przestrzeń 3D | |
|---|---|---|---|---|
| | Czas rozpoznawania [ms] | Odpowiedź poprawna? | Czas rozpoznawania [ms] | Odpowiedź poprawna? |
| Ręce szeroko | 112 | Tak | 66 | Tak |
| Ręce do modlitwy | 59 | Tak | 59 | Tak |
| Ręce jak siłacz | 123 | Tak | 72 | Tak |
| Ręce do przodu | 92 | Tak | 96 | Tak |
| Ręce na czole | 52 | Tak | 96 | Tak |
| Ręce na biodrach | 81 | Tak | 118 | Tak |
| Skuteczność | | 100% | | 100% |

Czas rozpoznawania gestu dla sieci dwuwymiarowej nie przekroczył 123 ms natomiast dla sieci trójwymiarowej 118 ms co jest wynikiem więcej niż zadowalającym. Widać też, że wprowadzenie trzeciego wymiaru nie wpływa negatywnie na czas rozpoznawania gestu. Wszystkie odpowiedzi sieci okazały się zgodne z oczekiwanymi (sieć zwracała wynik oczekiwany przez użytkownika).

### 6.5.2. Scenariusz 2 – Rozróżnianie gestów przy przesunięciu ciała względem sensora Kinect

Bazy gestów zostały zaczerpnięte ze scenariusza 1. i przestawione są na rysunku 33. Gesty prezentowane w tym scenariuszu przedstawiono na rysunku 35. Wyniki eksperymentu przedstawiono w tabeli 6.2.




*Rysunek 35: Gesty prezentowane w Scenariuszu 2.*

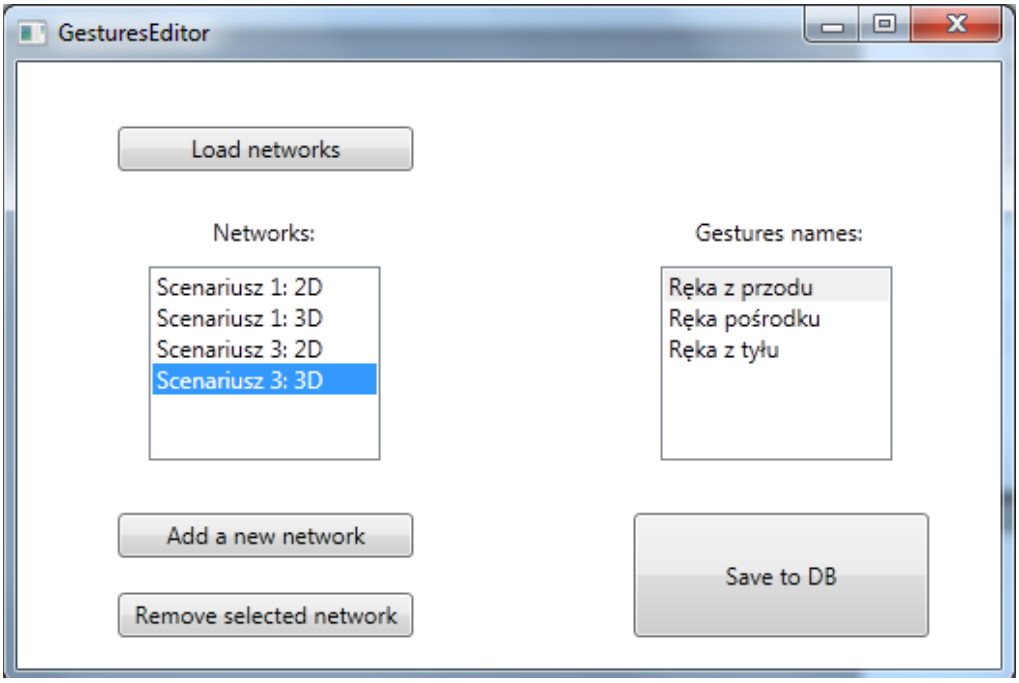
*Tabela 6.2: Wyniki dla scenariusza 2.*

| Rozpoznawany gest | Przestrzeń 2D | | Przestrzeń 3D | |
|---|---|---|---|---|
| | Lewa strona. Odpowiedź poprawna? | Prawa strona. Odpowiedź poprawna? | Lewa strona. Odpowiedź poprawna? | Prawa strona. Odpowiedź poprawna? |
| Ręce szeroko | Tak | Tak | Tak | Tak |
| Ręce do modlitwy | Tak | Tak | Tak | Tak |
| Ręce jak siłacz | Tak | Tak | Tak | Tak |
| Ręce do przodu | Tak | Tak | Tak | Tak |
| Ręce na czole | Tak | Tak | Tak | Tak |
| Ręce na biodrach | Tak | Tak | Tak | Tak |
| Skuteczność | 100% | 100% | 100% | 100% |

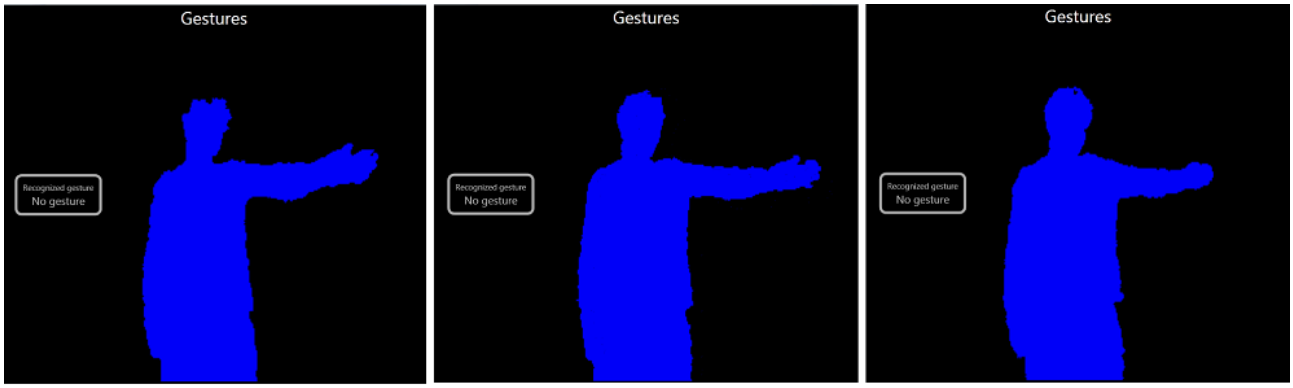
Zgodnie z oczekiwaniami przesunięcie postaci użytkownika względem sensora Kinect nie miało negatywnego wpływu na rozpoznawanie gestów. Odpowiedź sieci neuronowej była w stu procentach zgodna z oczekiwaniami dla wszystkich zaprezentowanych gestów.

### 6.5.3. Scenariusz 3 – Rozróżnianie gestów w osi Z

Baza gestów została zaprezentowana na rysunku 36. Gesty prezentowane w tym scenariuszu przedstawione zostały na rysunku 37. Wyniki zaprezentowano w tabeli 6.3.




*Rysunek 36: Projekt baz gestów w Scenariuszu 3.*




Ręka z przodu Ręka pośrodku Ręka z tyłu

*Rysunek 37: Gesty prezentowane w scenariuszu 3.*

*Tabela 6.3: Wyniki dla Scenariusza 3.*

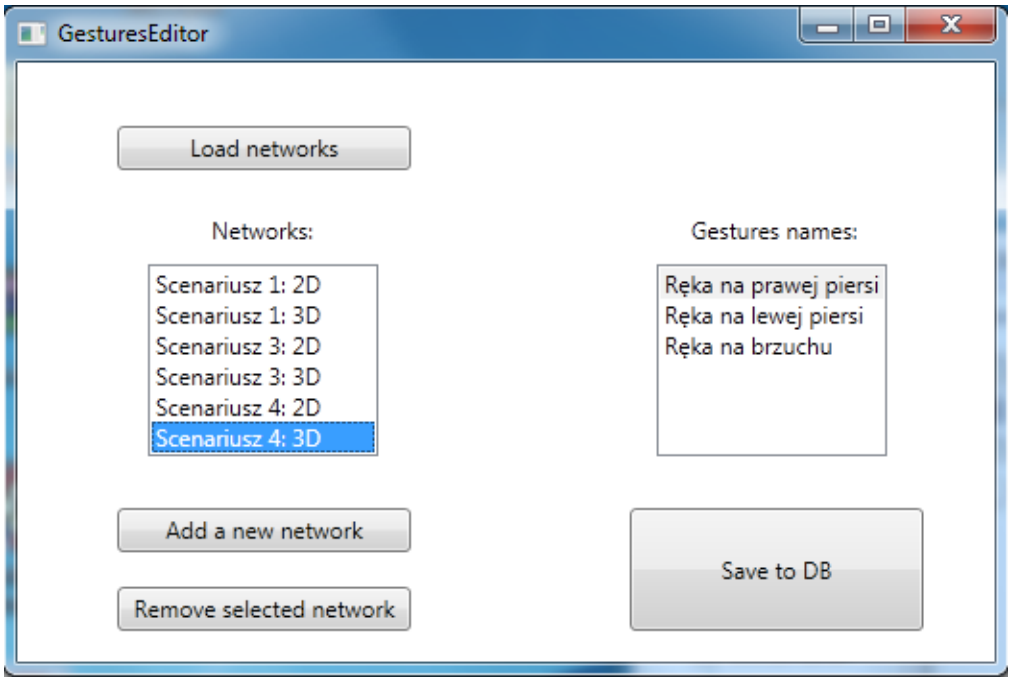
| Seria | Rozpoznawany gest | Przestrzeń 2D | | Przestrzeń 3D | |
|---|---|---|---|---|---|
| | | Odpowiedź sieci | Odpowiedź poprawna? | Odpowiedź sieci | Odpowiedź poprawna? |
| 1 | Ręka z przodu | Ręka z przodu | Tak | Ręka z przodu | Tak |
| | Ręka pośrodku | Ręka z przodu | Nie | Ręka pośrodku | Tak |
| | Ręka z tyłu | Nie rozpoznano | Nie | Ręka z tyłu | Tak |
| 2 | Ręka z przodu | Ręka z przodu | Tak | Ręka z przodu | Tak |
| | Ręka pośrodku | Ręka z przodu | Nie | Ręka pośrodku | Tak |
| | Ręka z tyłu | Nie rozpoznano | Nie | Ręka z tyłu | Tak |
| Skuteczność | | | 33% | | 100% |

Wyniki eksperymentu są zgodne z oczekiwanymi. Rozpoznawanie gestów, zróżnicowanych jedynie w osi Z uniemożliwia poprawne rozpoznanie w przestrzeni dwuwymiarowej – osiągnięto zaledwie skuteczność 33%, a w dwóch przypadkach wynik sieci nie był na tyle zadowalający, aby zdecydować się na którykolwiek z rezultatów i zwrócona została odpowiedź „Nie rozpoznano” (dzieje się tak, gdy wynik sieci dla wszystkich gestów wynosi poniżej 0,85 w skali (0,1)). Z drugiej

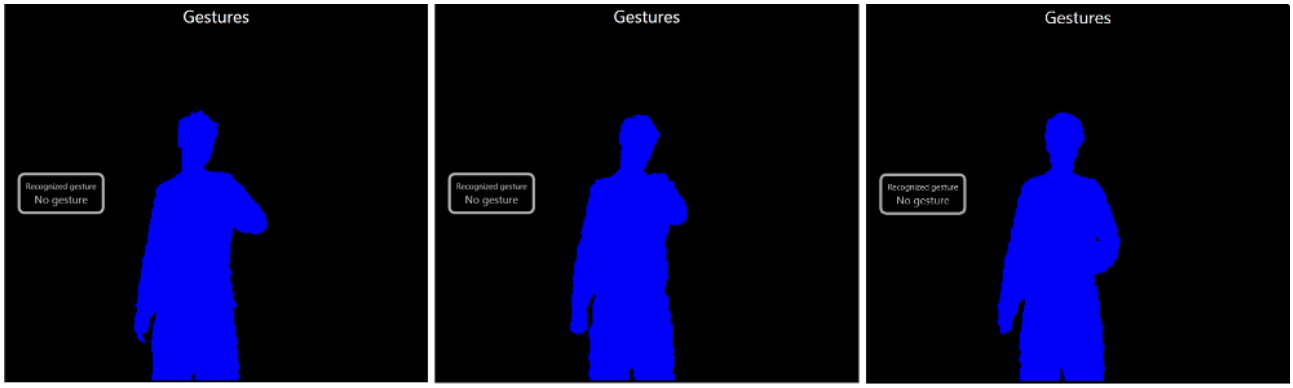
strony sieć trójwymiarowa osiągnęła wynik 100% co oznacza, że wszystkie odpowiedzi sieci były zgodne z oczekiwaniem użytkownika.

### 6.5.4. Scenariusz 4 – Rozróżnianie gestów podobnych

Baza gestów została zaprezentowana na rysunku 38. Gesty prezentowane w tym scenariuszu przedstawione zostały na rysunku 39. Wyniki zaprezentowano w tabeli 6.4.




*Rysunek 38: Projekt baz gestów w Scenariuszu 4.*




*Rysunek 39: Gesty prezentowane w scenariuszu 4.*

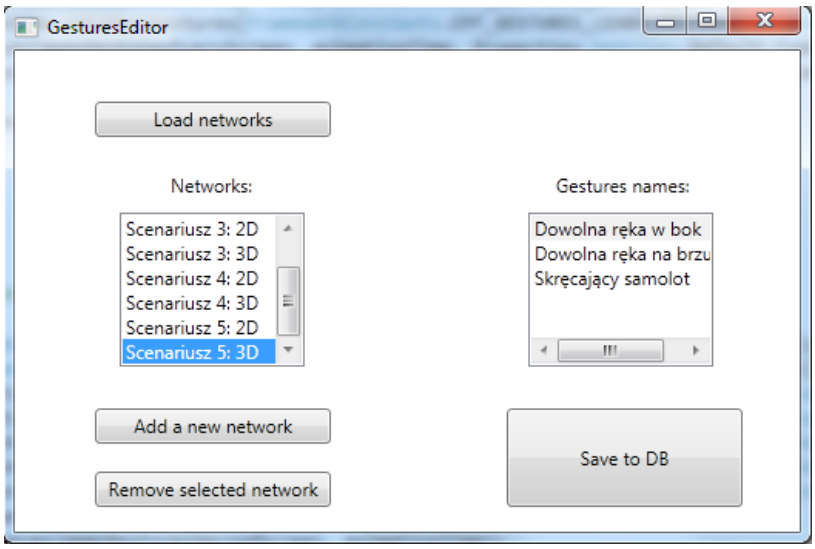
*Tabela 6.4: Wyniki dla Scenariusza 4.*

| Rozpoznawany gest | Przestrzeń 2D | Przestrzeń 3D |
|---|---|---|
| | Odpowiedź poprawna? | Odpowiedź poprawna? |
| Ręka na prawej piersi | Tak | Tak |
| Ręka na lewej piersi | Tak | Tak |
| Ręka na brzuchu | Tak | Tak |
| Skuteczność | 100% | 100% |

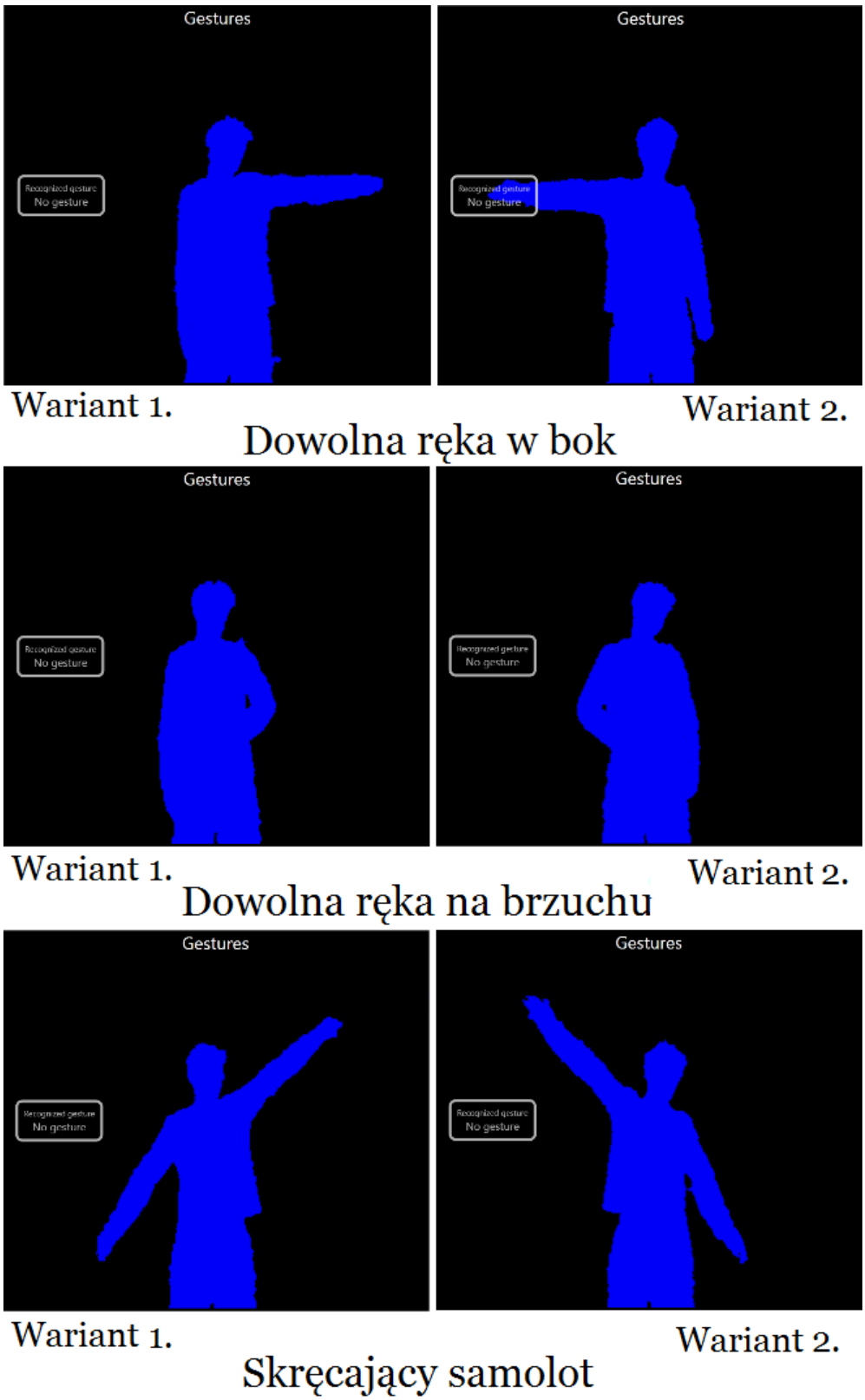
Gesty zademonstrowane w niniejszym eksperymencie były podobne, co oznacza że wszystkie rozpoznawane punkty na ciele, w porównaniu z pozostałymi gestami, były niemal w tej samej pozycji, z dokładnością do 10 centymetrów. Należy zwrócić uwagę, że lewa dłoń, lewy łokieć oraz prawy łokieć znajdowały się w tym samym punkcie dla wszystkich gestów. Natomiast Jedynym punktem zmieniającym pozycję była prawa dłoń, która przesuwała się o nie więcej niż 10 cm względem każdego z gestów. W związku z tak postawionym problemem wyniki rozpoznawania są więcej niż zadowalające. Stuprocentowa skuteczność rozpoznania zarówno dla przestrzeni dwu-, jak i trójwymiarowej prezentuje wielkie możliwości jakie stoją przed Kinektem i sieciami neuronowymi w dziedzinie rozpoznawania gestów.

### 6.5.5. Scenariusz 5 – Rozpoznawanie gestów uniwersalnych

Baza gestów została zaprezentowana na rysunku 40. Gesty prezentowane w tym scenariuszu przedstawione zostały na rysunku 41. Wyniki zaprezentowano w tabeli 6.5.




*Rysunek 40: Projekt baz gestów w Scenariuszu 5.*




*Rysunek 41: Gesty prezentowane w scenariuszu 5.*

*Tabela 6.5: Wyniki dla Scenariusza 5.*

| Seria | Rozpoznawany gest | Przestrzeń 2D | | Przestrzeń 3D | |
|---|---|---|---|---|---|
| | | Wariant 1. | Wariant 2. | Wariant 1. | Wariant 2. |
| | | Odpowiedź poprawna? | Odpowiedź poprawna? | Odpowiedź poprawna? | Odpowiedź poprawna? |
| Po nauce pierwszego wariantu | Dowolna ręka w bok | Tak | Nie | Tak | Nie |
| | Dowolna ręka na brzuchu | Tak | Tak | Tak | Tak |
| | Skręcający samolot | Tak | Nie | Tak | Nie |
| Skuteczność | | 100% | 33% | 100% | 33% |
| Po nauce pierwszego i drugiego wariantu | Dowolna ręka w bok | Tak | Tak | Tak | Tak |
| | Dowolna ręka na brzuchu | Tak | Tak | Tak | Tak |
| | Skręcający samolot | Tak | Tak | Tak | Tak |
| Skuteczność | | 100% | 100% | 100% | 100% |

Początkowo nauczono sieci neuronowe jedynie pierwszego wariantu gestów. Następnie przetestowano oba warianty gestów na sieciach. Ich odpowiedź była podobna – w wariancie pierwszym skuteczność rozpoznawania wyniosła 100%, natomiast w wariancie drugim 33%. Następnie sieci douczono drugiego wariantu gestów. Po tej nauce ich odpowiedź była już bezbłędna i wyniosła 100% dla wszystkich gestów.

W niniejszym eksperymencie wykazano, iż mimo że sieć neuronowa nie wie wszystkiego po pierwszej fazie nauki to kolejne etapy doszkalania sieci prowadzą do lepszych rezultatów.

### 6.5.6. Zestawienie czasów przetwarzania podczas przeprowadzania eksperymentów

Zestawienie wyników zaprezentowane zostało w tabeli 6.6.

*Tabela 6.6: Zestawienie czasów przetwarzania danych w czasie przeprowadzania eksperymentów*

| Scenariusz | Liczba przykładów | Czas zapisu do bazy danych [ms] | | Czas nauki sieci neuronowej [ms] | |
|---|---|---|---|---|---|
| | | Przestrzeń 2D | Przestrzeń 3D | Przestrzeń 2D | Przestrzeń 3D |
| Scenariusz 1 | 6 | 432 | 512 | 194 | 660 |
| Scenariusz 2 | 6 | 553 | 635 | 442 | 767 |
| Scenariusz 3 | 3 | 290 | 393 | 198 | 809 |
| Scenariusz 4 | 3 | 488 | 516 | 746 | 31 |
| Scenariusz 5 Wariant 1. | 3 | 570 | 598 | 891 | 807 |
| Scenariusz 5 Wariant 1. i 2. | 6 | 567 | 535 | 278 | 154 |
| Średnia | --- | 483 | 532 | 458 | 532 |
| Odchylenie standardowe | --- | 108,99 | 83,61 | 296,78 | 351,46 |

Maksymalny czas przetwarzania podczas przeprowadzania eksperymentów wyniósł 891 milisekund i jest to czas nauki sieci neuronowej dla Scenariusza 5. w przestrzeni dwuwymiarowej. Wynik ten mieści się dużo poniżej granicy określonej w założeniach i wynoszącej 3 sekundy. Uwagę zwraca znaczące odchylenie standardowe co wskazuje na dużą zależność przeprowadzania operacji od aktualnej dostępności zasobów komputerowych (procesor, pamięć). W sytuacji skrajnego wyczerpania zasobów komputerowych czasy wykonywania operacji należy określić jako zadowalające.

## 6.6. Podsumowanie wyników

Przeprowadzone eksperymenty dowiodły możliwości stojących przed połączeniem sensora Microsoft Kinect z sieciami neuronowymi. Biorąc po uwagę założenia do eksperymentów

określone w rozdziale 6.1 oraz cele eksperymentów w rozdziale 6.3 należy zwrócić uwagę na następujące rezultaty:

- warunek szybkości został zachowany, ponieważ czas przetwarzania w najgorszym wypadku osiągnął pułap 891 milisekund, co jest wartością znacząco niższą od akceptowalnych 3 sekund,
- warunek pewności został zachowany, ponieważ skuteczność odpowiedzi sieci neuronowych po odpowiedniej nauce wyniosła 100%,
- warunek dokładności została zachowana, ponieważ najrozmaitsze interpretacje gestów przez użytkownika po odpowiedniej nauce były zgodne z odpowiedziami sieci neuronowych,
- eksperymenty dowiodły większych możliwości sieci neuronowych opartych o dane w trzech wymiarach niż w dwóch wymiarach,
- czas nauki i odpowiedzi sieci neuronowych nauczonych gestów trójwymiarowych nie odbiegały znacząco od czasów dla sieci z gestami dwuwymiarowymi,

Z drugiej strony należy zwrócić uwagę na problemy zaproponowanych rozwiązań:

- rozpoznawanie gestów wymaga znaczących zasobów komputerowych,
- w obecnym rozwiązaniu osoby trzecie zaburzają proces rozpoznawania,
- sensorowi Kinect zdarza się rozpoznawać inne obiekty jako ludzi (np. krzesła),
- mocny kontrast w ubiorze użytkownika może powodować problemy z rozpoznaniem,
- nie zauważono problemów związanych z temperaturą (ze względu na spektrum działania kamery w podczerwieni) podczas testów przeprowadzanych w temperaturze pokojowej (20-30 stopni Celsjusza).

# 7. Podsumowanie

Rozpoznawanie gestów jest jedną z wiodących dziedzin nowoczesnej informatyki zajmującej się interfejsami użytkownika. Połączenie sensora Microsoft Kinect – narzędzia do przechwytywania obrazu użytkownika – oraz sieci neuronowych – narzędzia do rozpoznawania gestów – okazały się rozwiązaniem skutecznym i szybkim. W kolejnych rozdziałach przedstawiono rezultaty niniejszej pracy.

## 7.1. Główne rezultaty

Główne rezultaty pracy prezentują się następująco:

- Dokonano przeglądu najpopularniejszych narzędzi do rozpoznawania gestów,
- Zaprojektowano sieć neuronową zdolną do rozpoznawania gestów na podstawie informacji z sensora Kinect,
- Zaimplementowano aplikację zdolną do rozpoznawania gestów przy użyciu sensora Kinect oraz sieci neuronowych,
- Zaproponowano nowe komponenty kontrolne w środowisku WPF odpowiadające potrzebom sterowania aplikacją przy użyciu ciała,
- Wykonano eksperymenty, które dowiodły skuteczności połączenia sensora Kinect i sieci neuronowych w rozpoznawaniu gestów statycznych.

## 7.2. Najważniejsze wnioski

Najważniejsze wnioski z przeprowadzonych prac prezentują się następująco:

- Połączenie Microsoft Kinect oraz sieci neuronowych jest rozwiązaniem skutecznym w problemie rozpoznawania gestów statycznych,
- Środowisko WPF okazało się na tyle elastyczne, że stosunkowo łatwo udało się w nim

zaprojektować komponenty sterujące aplikacją przy użyciu ciała użytkownika,

- Do obsługi sensora Kinect, sieci neuronowych oraz aplikacji sterowanej ciałem użytkownika potrzebna jest duża moc obliczeniowa,
- Rozłożenie aplikacji na dwie osobne, sterowane przy użyciu innego medium okazało się rozwiązaniem skutecznym. Wcześniejsze zaprojektowanie gestu przy użyciu klawiatury i myszki nie wpływa negatywnie na wrażenia związane z obsługa aplikacji jedynie przy użyciu ciała. Natomiast rozdzielność aplikacji pozwala zachować spójność w zakresie przewidzianych dla aplikacji zadań,
- Wykorzystywanie własnego ciała do sterowania aplikacjami jest fizycznie meczące. Należy optymalizować czas ekspozycji części ciała pod użytkownika, tak aby jego wysiłek bym jak najmniejszy (dotyczy to zarówno prezentacji gestów, jak i sterowania aplikacją),
- Zastosowanie sieci neuronowej do rozpoznawania gestów statycznych okazało się lepszym pomysłem niż można było sądzić. Sieci neuronowe wydają się być skomplikowanym narzędziem, któremu należy poświęcić dużo czasu i pracy, aby je odpowiednio skonfigurować do wyznaczonego zadania. Okazało się jednak, że standardowa konfiguracja sieci neuronowej proponowana przez autora biblioteki, plus odpowiednio zaprojektowane warstwy wejściowa i wyjściowa skutkowały stuprocentową poprawnością odpowiedzi po odpowiedniej nauce.
- Problem rozpoznawania gestów dla sieci neuronowej okazał się na tyle prosty, że w ogromnej większości przypadków wystarczył jeden przykład dla każdego wariantu danego gestu, aby sieć nauczyła się gestu wystarczająco, by go rozpoznać,
- Baza danych MySQL jest skutecznym metodą komunikacji pomiędzy aplikacjami w zakresie projektowania baz gestów jako danych dla sieci neuronowych.

## 7.3. Dalsze prace rozwojowe

Praca nad niniejszym rozwiązaniem pozostawiła wiele obszarów do zagospodarowania na przyszłość:

- Zaproponowane rozwiązanie okazało się wystarczająco szybkie i skuteczne, aby wdrożyć opartą na nim funkcjonalność w aplikacjach biznesowych. Przykładowe zastosowania rozpoznawania gestów statycznych prezentują się następująco:
    - gry wideo,

- inteligentne środowisko,
- sterowanie prezentacjami multimedialnymi,
- rozpoznawanie gestów funkcjonariuszy naprowadzających samolot do lądowania,
- analiza zachowania mówców na seminariach.

- Należy zaimplementować rozpoznawanie gestów dynamicznych, w których badany jest ruch użytkownika, a nie tylko jego położenie. Wstępna analiza wskazuje możliwość oparcia takiego rozwiązania o ukryte modele Markowa opisane w [L],
- Ze względu na aspekt eksperymentalny aplikacji do rozpoznawania gestów wykorzystywano jedynie 5 punktów na ciele (dłonie, łokcie i splot słoneczny) z dostępnych 20. Należy rozważyć rozszerzenie możliwości aplikacji o kolejne punkty. Problemem jest jednak znalezienie punktu centralnego np. dla nóg (w obecnej implementacji splot słoneczny służy jako punkt odniesienia dla pozycji dłoni i łokci, co pozwala uniezależnić rozpoznawany gest od położenia użytkownika względem sensora),
- Należy rozważyć możliwość rozpoznawania języka migowego. Problemem jest jednak brak możliwości rozpoznawania palców u dłoni przy obecnych możliwościach sensora Kinect. Autorzy [K] zaprezentowali metodę rozpoznawania prostych gestów dłoni, jednak także oni borykali się z niską jakością danych,
- Należy rozważyć rozwijanie profili , w których program mógłby się uczyć zachowania użytkownika, podczas gdy użytkownik uczyłby się aplikacji. To oznacza, iż jeżeli użytkownik sterując aplikacją nauczył się szybciej prezentować bezbłędnie dany gest, to należy skrócić czas ekspozycji gestu przed rozpoznaniem. Podobnie badając liczbę pomyłek użytkownika przy przełączaniu się między ekranami (np. poprzez badanie liczby powrotów do poprzedniego ekranu bezpośrednio po przejściu do danego ekranu) można wydłużać, lub skracać czas potrzebny do zatwierdzania przycisku.

# 8. Literatura

## 8.1. Publikacje


[A] Lihong Zheng and Xiangjian He; Classification Techniques in Pattern Recognition http://wscg.zcu.cz/wscg2005/papers_2005/poster/k43-full.pdf

[B] Geoffrey Zweig and Stuart Russell; Speech Recognition with Dynamic Bayesian Networks http://www.aaai.org/Papers/AAAI/1998/AAAI98-024.pdf

[C] Shumin Zhai and Per-Ola Kristensson; Shorthand Writing on Stylus Keyboard; http://www.almaden.ibm.com/u/zhai/papers/SharkFinal.pdf

[D] Avi Drissman; Handwriting Recognition Systems: An Overview http://www.drissman.com/avi/school/HandwritingRecognition.pdf

[E] Eisaku Ohbuchi, Hiroshi Hanaizumi and Lim Ah Hock; Barcode Readers using the Camera Device in Mobile Phones; http://www.google.pl/url?sa=t&rct=j&q=&esrc=s&source=web&cd=7&ved=0CHAQFjAG&url=http%3A%2F%2Facademicfamily.com%2Fpapers%2Fdownloadpresentation%2F12118ed440bfabffd3eae68a55658571.pdf&ei=w8X6T-2FCsnE8QP7x4SHBw&usg=AFQjCNGWzJbkWfZ71oBpuKqx1O4EpqKmrA&sig2=yEXN-lGhxNbCOqr4XrUopA

[F] Yue Liu, Ju Yang, Mingjun Liu; Recognition of QR Code with Mobile Phones; http://blog.cs.nhcue.edu.tw/wpmu/klou/files/2009/09/rec_qrcode.pdf

[G] David Mercier, Genevieve Cron, Thierry Denoeux and Marie-Helene Masson; Decision fusion for postal address recognition using belief functions; https://www.hds.utc.fr/~massomar/publis/eswa08.pdf

[H] Bram Alefs, Guy Eschemann, Herbert Ramoser, Csaba Beleznai; Road Sign Detection

from Edge Orientation Histograms;
http://www.vision.caltech.edu/VisionWiki/images/4/4b/Alefs07road.pdf

[I] Fingertec; Face Recognition Technology White Paper;
http://www.fingertec.com/download/tips/whitepaper-02.pdf

[J] Chi-Wei Chu, Isaac COHEN; Posture and Gesture Recognition using 3D Body Shapes; Decomposition http://iris.usc.edu/outlines/papers/2005/cwchu-v4hci05.pdf

[K] Matthew Tang; Recognizing Hand Gestures with Microsoft's Kinect;
http://www.stanford.edu/class/ee368/Project_11/Reports/Tang_Hand_Gesture_Recognition.pdf

[L] Jonathan C. Hall; How to do gesture recognition with Kinect using hidden Markov models;
http://www.creativedistraction.com/demos/gesture-recognition-kinect-with-hidden-markov-models-hmms/

[M] Hongli Lai; Using Commodity Visual Gesture Recognition Technology to Replace or to Augment Touch Interfaces; http://www.google.pl/url?sa=t&rct=j&q=&esrc=s&source=web&cd=7&ved=0CGoQFjAG&url=http%3A%2F%2Freferaat.cs.utwente.nl%2FTSConIT%2Fdownload.php%3Fid%3D959&ei=61gNUJrNJqSM0wXPoqW3Cg&usg=AFQjCNEjdtD15piyvepnectbahWFWhoI0Q&sig2=7yFGbo32LqR692Ge0fWaqg

[N] Simon Fothergill, Helena M. Mentis, Pushmeet Kohli, Sebastian Nowozin; Instructing People for Training Gestural Interactive Systems; http://research.microsoft.com/en-us/um/people/pkohli/papers/fmkn_chi2012.pdf

## 8.2. Zasoby internetowe

[1] ProgrammingGuide_KinectSDK; 2011;
http://wenku.baidu.com/view/1e17d3a3b0717fd5360cdce9.html

[2] What's inside a Kinect?; 2011; http://kotaku.com/5682075/whats-inside-a-kinect

[3] Macierz sensorów urządzenia Kinect; 2011; http://www.generationrobots.com/microsoft-kinect-sensor,us,4,Kinect-Microsoft-Sensor.cfm

[4] Official Kinect SDK released; 2011; http://hackaday.com/2011/06/16/official-kinect-sdk-released/

[5] Kinect for Silverlight 5 – Part 2: Skeletal Tracking; 2011; http://www.fdesimoni.ch/post/2011/10/03/Kinect-for-Silverlight-5-Part-2-Skeletal-Tracking.aspx

[6] Microsoft Kinect hacked; 2010; http://www.telegraph.co.uk/technology/microsoft/8129616/Microsoft-Kinect-hacked.html

[7] Coding4Fun; http://channel9.msdn.com/coding4fun

[8] Things you can't do with the Microsoft Kinect SDK; 2011; http://blog.makezine.com/2011/06/17/things-you-cant-do-with-the-microsoft-kinect-sdk/

[9] Playstation EyeToy; 2011; http://gta.wikia.com/File:Playstation_2_Eyetoy.jpg

[10] Nintendo Wii; 2011; http://www.gamekool.com/images/GKWA-00001.jpg

[11] Playstation Move; 2011; http://cdn4.digitaltrends.com/wp-content/uploads/2010/12/Move-Controller.jpg

[12] Microsoft Kinect; http://www.hea7en.info/wp-content/uploads/2011/12/kinnn.jpg

[13] Satellites dock with Microsoft's Kinect; 2012; http://www.vision-systems.com/articles/2012/05/satellites-dock-with-microsofts-kinect.html

[14] Microsoft Kinect SDK; 2011; http://www.microsoft.com/en-us/kinectforwindows/

[15] Hsynapse; http://4programmers.net/C_sharp/Gotowce/Sieci_neuronowe_aproksymacja_i_rozpoznawanie_pisma

[16] Github; https://github.com/

[17] Fisher-Yates shuffle; http://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle

[18] Statistical analysis of the Fisher-Yates shuffle; 2011; http://mvngu.wordpress.com/2011/05/08/statistical-analysis-of-the-fisher-yates-shuffle/

# 9. DODATEK A

INSTALACJA APLIKACJI

## 9.1. Wymagania sprzętowe

Wymagania sprzętowe[4]:

System operacyjny: Windows 7 32-bit, lub 64-bit,

Procesor: dwa rdzenie taktowane zegarem 2.0 GHz,

Pamięć RAM: 4 GB,

Miejsce na dysku: 10 MB na samą aplikację plus miejsce wymagane przez dodatkowe oprogramowanie,

Urządzenie Microsoft Kinect, przejściówka do komputera PC.

## 9.2. Instalacja aplikacji

1. Instalacja Microsoft .NET 4.0,
2. Instalacja Kinect for Windows SDK Beta2,
3. Instalacja bazy danych MySQL 5.5,
4. Przekopiowanie aplikacji GesturesEditor i GesturesRecognizer na dysk,
5. Importowanie schemy na serwerze bazy danych MySQL 5.5,
6. Podłączenie sensora Kinect do komputera i instalacja sterowników.

4 Wymagania zawarte w tej sekcji mogą różnić się od tych podanych w specyfikacji Microsoft Kinect SDK. Spowodowane jest to faktem, iż wymagania zostały zweryfikowane przez autora i jest on pewien, że aplikacja będzie działać płynnie na podanym sprzęcie.