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. Niniejsza praca dotyczy próby oceny przydatności tego urządzenia w rozpoznawaniu gestów statycznych. Zaprezentowana została przykładowa aplikacja, w pełni sterowalna przy użyciu Microsoft Kinect oraz pozwalająca na rozpoznanie gestu i ocenę dokładności tego rozpoznania.

Abstract

bla bla bla

Spis treści

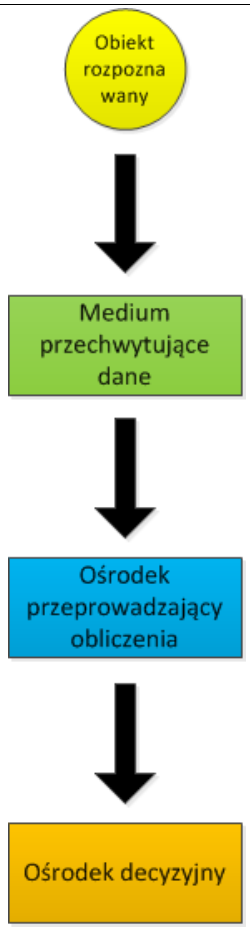
1. Wprowadzenie

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 [10],
- rozpoznawanie obrazów, w tym:
  - rozpoznawanie pisma [12],
  - rozpoznawanie kodów (kody kreskowe [13], kody QR [14], kody pocztowe [15]),
  - rozpoznawanie znaków drogowych [16],
  - rozpoznawanie twarzy [17],
  - rozpoznawanie gestów [18],
- rozpoznawanie stylu pisania [11].

Rozpoznawanie gestów w najprostszym scenariuszu wymaga obiektu rozpoznawanego (np. człowiek), medium przechwytującego dane (np. kamera), systemu przeprowadzającego operacje na tych danych, oraz ośrodka decyzyjnego (Rysunek 1).



Rysunek 1: Podstawowa budowa systemu rozpoznawania gestów

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 [19]

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 [20]

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 [21]

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, która wyświetlasię 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 [22]

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 [23].

Zagadnienie rozpoznawania gestów przy wykorzystaniu możliwości sensora Kinect, a szczególnie wykrywania odległości od obiektów wydaje się być niezmiernie interesujące i właśnie z tego powodu autor zdecydował się na napisanie tej pracy.

2. Cel i zakres

Celem projektu 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 dokumentacją urządzenia Kinect oraz wybór technologii do wykonania aplikacji badawczych,
2. Zapoznanie się z dostępnymi technikami rozpoznawania gestów i wybór odpowiedniego rozwiązania,
3. Przygotowanie aplikacji umożliwiającej rozpoznawanie gestów oraz zbadanie ich jakości,
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 mają być demonstracją możliwości Kinekta w zakresie:

- sterowania aplikacjami przy użyciu dłoni,
- przygotowania uniwersalnych kontrolek do obsługi przy pomocy ruchów dłońmi,
- 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.

3. Stan wiedzy (SOA)

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

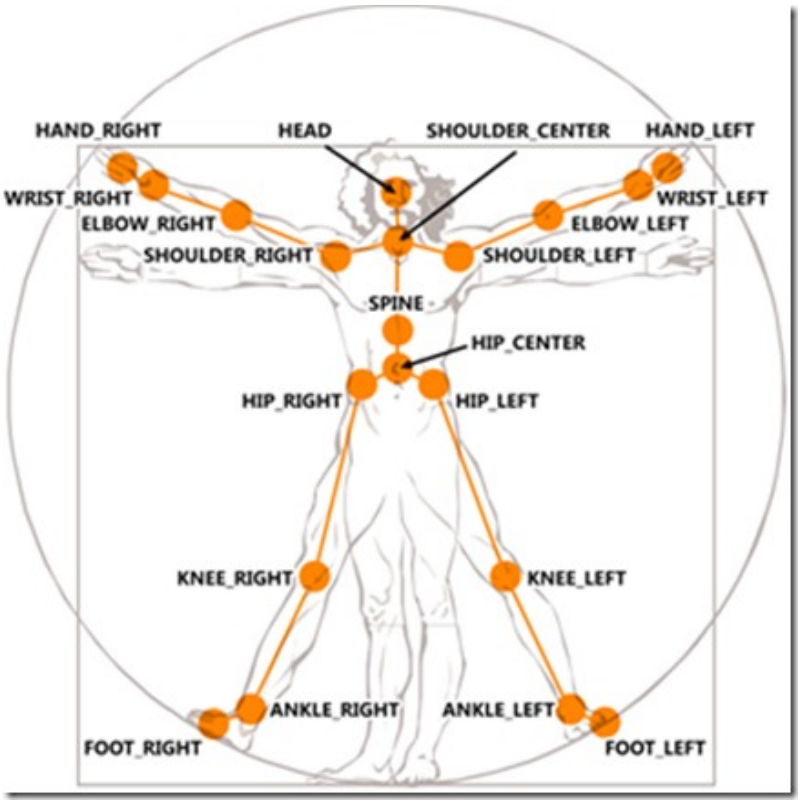
Microsoft Kinect SDK to zestaw narzędzi i bibliotek pozwalający na wykorzystanie urządzenia Kinect w aplikacji. Microsoft wypuścił SDK 16 czerwca 2011 roku, wspierając języki C#, C++ oraz Visual Basic [4].

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: 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 głosu[1].

3.3. Rozwiązania oparte na Microsoft Kinect

3.3.1. Początki Microsoft Kinect

Pierwsze próby przejęcia kontroli nad świeżo wypuszczonym 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. 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 wypuścił 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 do obsługi wymaga bardzo mocnej maszyny – dwurdzeniowy procesor o taktowaniu 2,66 GHz i 2 GB pamięci RAM. Microsoft Kinect SDK nie dostarcza też żadnego 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.

TODO: napisać coś mądrego na bazie artykułów... np. to http://www.creativedistraction.com/demos/gesture-recognition-kinect-with-hidden-markov-models-

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

hmms/

4. Środowisko badawcze

4.1. Opis projektu

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

4.1.1. Gestures Editor

Opis

Aplikacja służąca do przygotowania metadanych na temat zbiorów gestów.

Cel programu

Celem programu Gestures Editor jest przygotowanie zbioru gestów do nauki oraz rozpoznawania gestów, które są przeprowadzane w programie Gestures Recognizer. 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 (Gestures Recognizer aktualnie obsługuje jedynie 4 konkretne punkty na ciele – obie dłonie oraz łokcie. Powód takiego stanu rzeczy opisany został w rozdziale [TODO: Podać rozdział]),
- Przyporządkowanie nazwy do konkretnego gestu,
- Zapisanie zbioru gestów do bazy danych,
- Modyfikacji 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 Gestures Recognizer.

Dane wyjściowe

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

Przykładowe dane wyjściowe

Przykładowe dane wyjściowe wyabstrahowane z formatu danych w jakim zostały zapisane prezentować mogą się następująco:

- Zbiór gestów: „Ręce wysoko”,
- Liczba gestów: 3,
- Liczba wymiarów: 3,
- Liczba punktów: 4,
- Nazwa gestu 1: Lewa ręka wysoko,
- Nazwa gestu 2: Prawa ręka wysoko,
- Nazwa gestu 3: Obie ręce wysoko.

Wynikiem będzie zbiór gestów o nazwie „Ręce wysoko” wykorzystujący 4 punkty na ciele (lewa dłoń, lewy łokieć, prawa dłoń, prawy łokieć), w 3 wymiarach (X, Y, Z) i składająca się z 3 gestów.

4.1.2. Gestures Recognizer

Opis

Aplikacja pozwala na naukę sieci neuronowej gestów na bazie metadanych podanych w aplikacji Gestures Editor, 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 kontrolek 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.

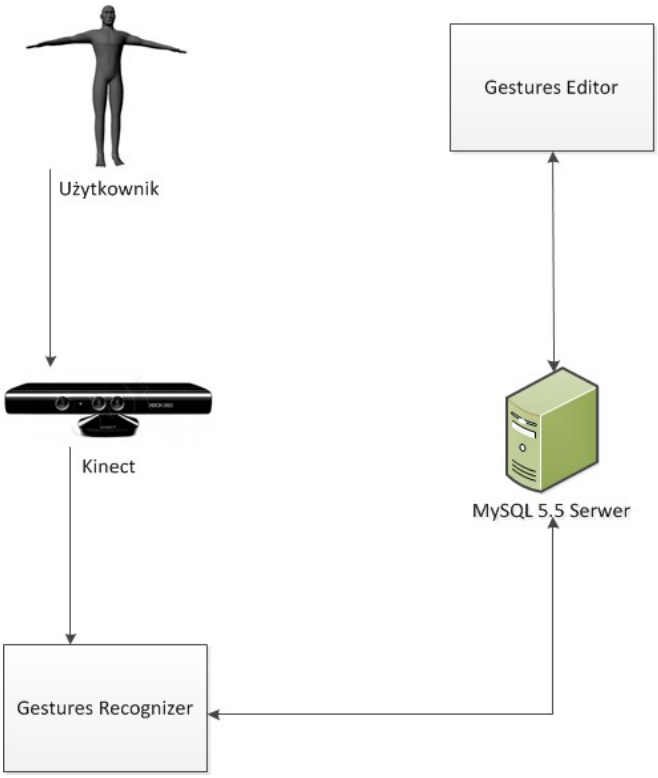
Dane wyjściowe

Sieci neuronowe nauczone rozpoznawania gestów.

4.1.3. Budowa systemu

System rozpoznawania gestów złożony jest z 2 aplikacji – Gestures Editor oraz Gestures Recognizer. Aplikacje te komunikują się ze sobą przy użyciu serwera MySQL 5.5. Gestures Editor dostarcza metadanych na temat zbiorów gestów, natomiast Gestures Recognizer pobiera informację

o gestach użytkownika poprzez sensor Kinect (Rysunek 8).



Rysunek 8: Budowa systemu

4.2. Wykorzystane technologie

Do wykonania aplikacji wykorzystano następujące technologie:

System operacyjny: Microsoft Windows 7 64-bit

Środowisko programistyczne: Microsoft Visual Studio 2010,

Język programowania: C#,

Biblioteka do komunikacji z Microsoft Kinect: Microsoft Kinect SDK [24],

Biblioteka sieci neuronowych: HSynapse [25],

Interfejs użytkownika: Windows Presentation Foundation (WPF),

Baza danych: MySQL Server 5.5,

System kontroli wersji: Git,

Serwer kontroli wersji: Github [26].

Sprzęt wykorzystany w projekcie prezentuje się następująco:

Microsoft Kinect,

Przejściówka USB,

Laptop Compal:

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

5. System w użyciu

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

5.1. Gestures Editor

5.1.1. Wygląd

5.1.2. Budowa

5.1.3. Sterowanie

5.1.4. Przykładowy scenariusz

5.2. Gestures Recognizer

5.2.1. Wygląd

5.2.2. Budowa

5.2.3. Sterowanie

5.2.4. Przykładowy scenariusz

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 (**XXX – podać rozdział w którym opisano strumienie danych z kinecta**) 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 [9]:

- 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. Koncepcja wykorzystania tej sieci do rozwiązania problemu rozpoznawania gestów prezentuje się następująco:

Warstwa wejściowa

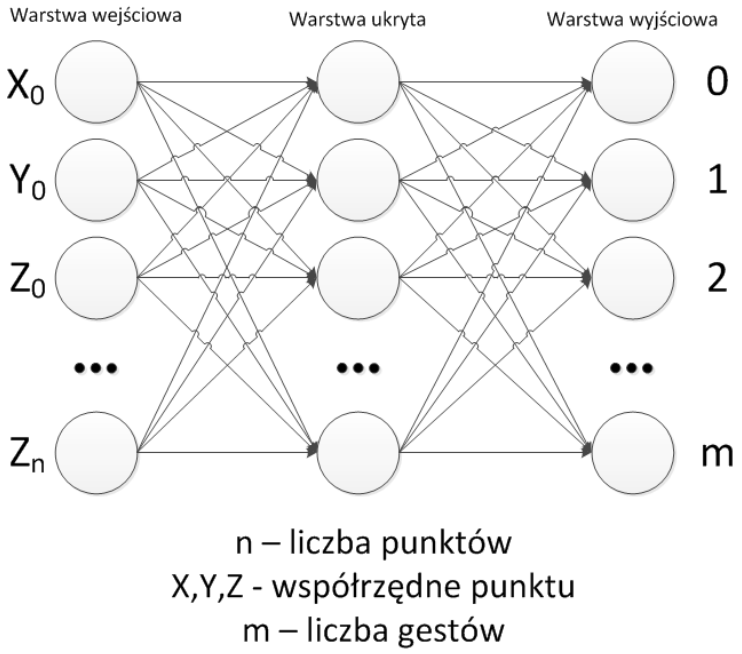
Warstwa wejściowa to posortowane współrzędne punktów na ciele użytkownika. Jeden neuron to jedna współrzędna. W przypadku uwzględnienia 3 wymiarów w przetwarzaniu układ wejściowy przydzielany jest według wzorca $X_0, Y_0, Z_0, X_1, Y_1, Z_1, \ldots, X_n, Y_n, Z_n$, gdzie n to indeks posortowanej listy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



Ilustracja 1: 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 **(XXX- opisać i określić te parametry)**. 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 **(XXX – napisać o algorytmie losowości)**. Następnie przez zadaną liczbę epok (domyślnie 2000 **XXX – czy na pewno?**) 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.

5.4.2. Struktury danych

5.4.3. Magazynowanie danych

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

Wszystkie dane przechowywane są w pojedynczej tabeli neural_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.

- **XXX – sprawdzić w domu czy na pewno dobrze**

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 następnym podpunkcie, 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 rozpoznawany jako inny, pomimo podobnego położenia rąk. Problem ten przedstawiono na rysunku XXX(**ZROBIĆ 2 ZRZUTY EKRANU PRZY ROZŁOŻONYCH RĘKACH STOJĄC W RÓŻNYCH MIEJSCACH I DODAĆ MOŻE JAKIŚ RYSUNEK PREZENTUJĄCY ZRZUT Z GÓRY).** Pomimo że użytkownik pokazywał, w swoim mniemaniu, ten sam gest, to położenie jego ciała w stosunku do kamery było zmienne. 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 proste z uwagi na dostarczanie danych o jego położeniu przez sam sensor Kinect. Położenie tego punktu przedstawiono na rysunku 3, 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

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 przydatny, że nie było potrzeby sprawdzania innych punktów.

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.

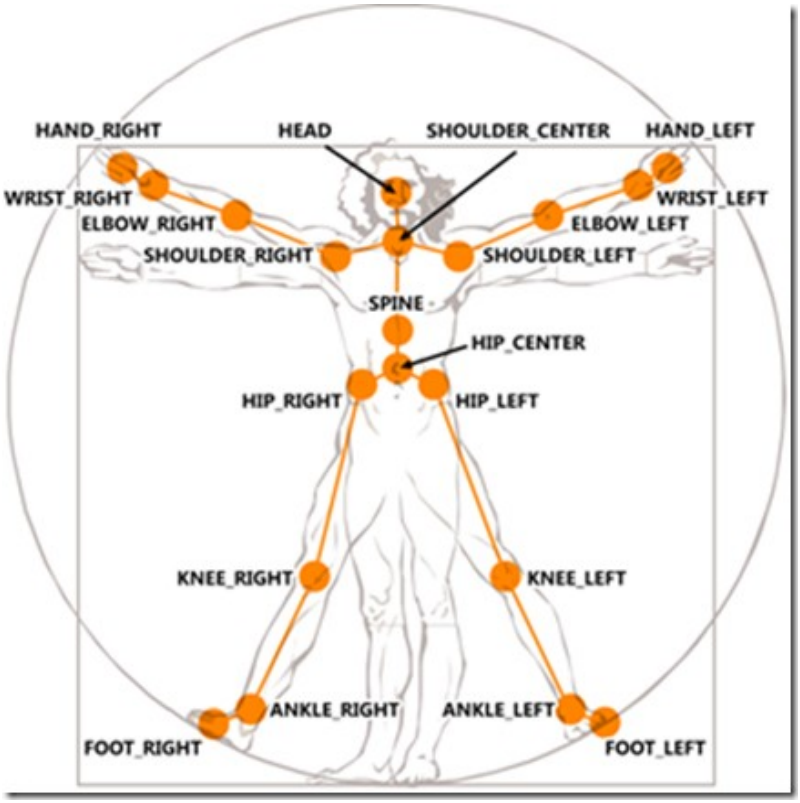
Z uwagi na możliwości sieci neuronowej do tolerancji błędów oraz aspekt eksperymentalny rozwiązania ewentualną potrzebę kalibracji aplikacji pozostawiono do wykonania przyszłości. Okazało się bowiem, że wartości korekcyjne ustawione w aplikacji w celu jak najwierniejszego uzyskania długości rąk okazały 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. W przypadku wyczerpującej kalibracji problem ten sam by się rozwiązał, jednak taka kalibracja mogłaby być niewygodna dla użytkownika i wymagałaby skomplikowanego procesu zarządzania. Zamiast tego wprowadzono oznaczenia dla użytkownika. Określono odpowiednią odległość 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. **XXX WRZUCIĆ TUTAJ OBRAZKI Z RÓŻNYMI ODLEGŁOŚCIAMI OD SENSORA I RÓŻNYMI KOLORAMI**.

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 tego rozwiązania nie wpłynął zauważalnie na jakość rozpoznania gestów.

Wprowadzenie do rozpoznania 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 9: 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 na 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 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

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,

3 W zaprezentowanym rozumowaniu pojęcie „klatki” należy rozumieć jako zebranie w danym momencie czasu danych o położeniu podanych punktów na ciele użytkownika w przestrzeni o podanej liczbie wymiarów.

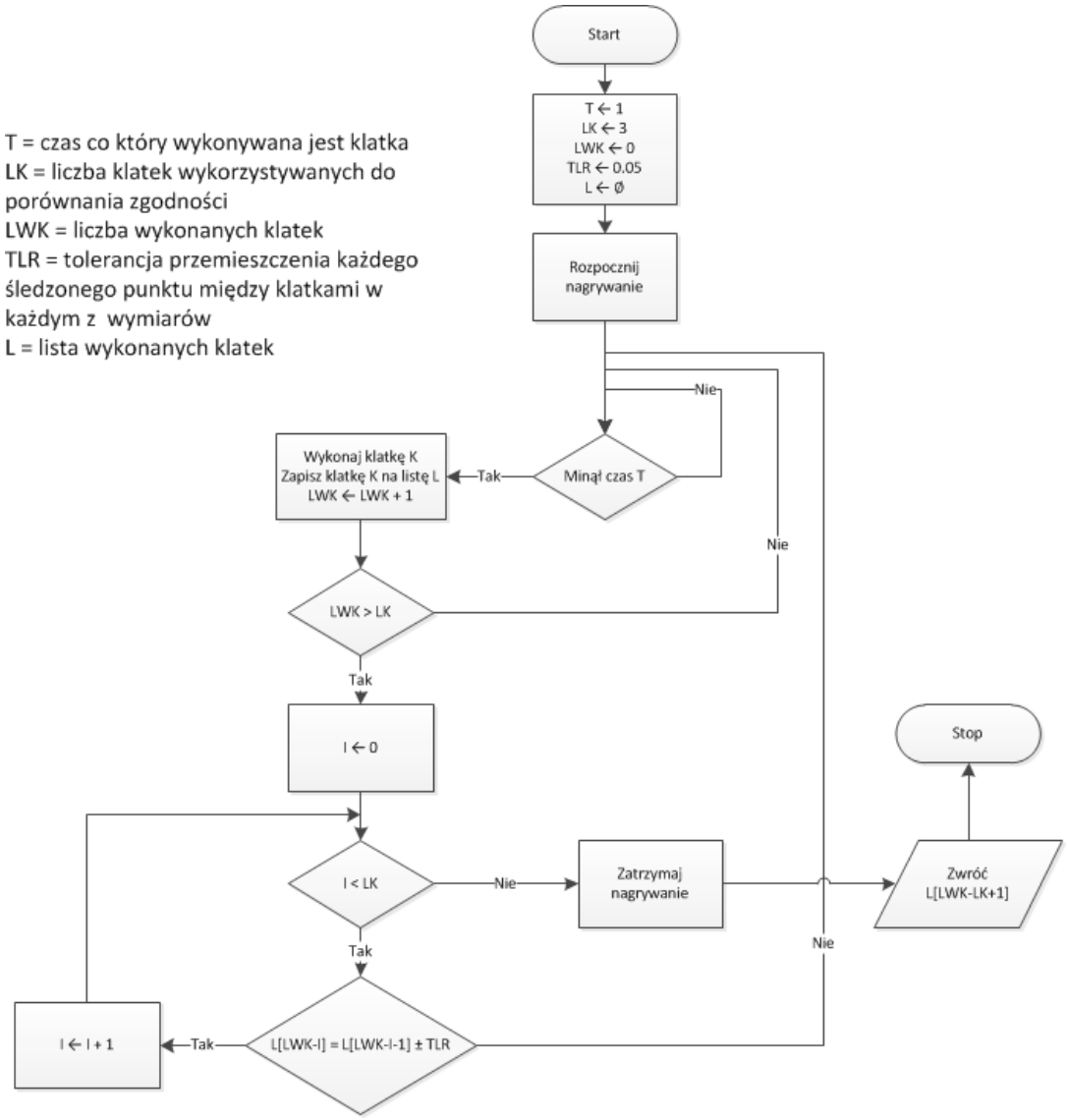
(c) jeżeli LWK większe od LK:

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 10: 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 zaplanowane zostały eksperymenty.

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 kilkadziesiąt centymetrów 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 wykalibrować poprzez mechanizmy zawarte w aplikacji.

6.2. Dane do eksperymentów

W celu zapewnienia poprawności wprowadzanych danych dane będące odpowiednikiem wskazywania gestów przy użyciu ciała zostały wprowadzone z pliku. Każdorazowe prezentowanie gestu przy użyciu ciała może wprowadzić niechciane zaburzenia.

Dodatkowo należy uściślić definicję gestu, który ma zostać rozpoznany:

6.2.1. Gest

Gest w ramach aplikacji Gesture Recognizer 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 Gesture Editor. Gest jest spójny w zakresie odległości od centralnej części ciała będącą reprezentowaną przez mostek. Gest jest niezależny od przesunięcia ciała w dowolnej z osi trójwymiarowego układu współrzędnych w ramach akceptowalnych przez aplikację (wszystkie części ciała muszą być widoczne przez kamerę, a odległość 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,

Dodatkowo ze względu na charakterystykę sieci neuronowych należy określić minimalną ilość danych uczących oraz minimalną liczbę epok nauki potrzebnych do osiągnięcia powyższych wyników.

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 zamodelowaniu 3 gestów w przestrzeni dwu- oraz trójwymiarowej. Są to fizycznie dwie różne sieci neuronowe. (**XXX – opisać przebieg eksperyementu, gesty które będą rozpoznawane, itp**)

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. (**XXX – pokazać nowe gesty**)

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 kilki osiach układu współrzędnych, w jednym, lub wielu punktach.

Przebieg

Zostaną utworzone nowe sieci neuronowe w celu zbadania możliwości rozpoznania gestów podobnych. Zbadane zostanie minimalne przesunięcie punktu, które skutkuje poprawnym rozpoznaniem.

7. Podsumowanie

8. Literatura

1. ProgrammingGuide_KinectSDK
   http://wenku.baidu.com/view/1e17d3a3b0717fd5360cdce9.html
2. What's inside a Kinect? http://kotaku.com/5682075/whats-inside-a-kinect
3. Macierz sensorów urządzenia Kinect http://www.generationrobots.com/microsoft-kinect-sensor,us,4,Kinect-Microsoft-Sensor.cfm
4. Official Kinect SDK released http://hackaday.com/2011/06/16/official-kinect-sdk-released/
5. Kinect for Silverlight 5 – Part 2: Skeletal Tracking
   http://www.fdesimoni.ch/post/2011/10/03/Kinect-for-Silverlight-5-Part-2-Skeletal-Tracking.aspx
6. Microsoft Kinect hacked
   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
   http://blog.makezine.com/2011/06/17/things-you-cant-do-with-the-microsoft-kinect-sdk/
9. Classification Techniques in Pattern Recognition
   http://wscg.zcu.cz/wscg2005/papers_2005/poster/k43-full.pdf
10. Speech Recognition with Dynamic Bayesian Networks
    http://www.aaai.org/Papers/AAAI/1998/AAAI98-024.pdf
11. Shorthand Writing on Stylus Keyboard
    http://www.almaden.ibm.com/u/zhai/papers/SharkFinal.pdf

12. Handwriting Recognition Systems: An Overview
http://www.drissman.com/avi/school/HandwritingRecognition.pdf

13. 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

14. Recognition of QR Code with Mobile Phones
http://blog.cs.nhcue.edu.tw/wpmu/klou/files/2009/09/rec_qrcode.pdf

15. Decision fusion for postal address recognition using belief functions
https://www.hds.utc.fr/~massomar/publis/eswa08.pdf

16. Road Sign Detection from Edge Orientation Histograms
http://www.vision.caltech.edu/VisionWiki/images/4/4b/Alefs07road.pdf

17. Face Recognition Technology White Paper
http://www.fingertec.com/download/tips/whitepaper-02.pdf

18. Posture and Gesture Recognition using 3D Body Shapes Decomposition
http://iris.usc.edu/outlines/papers/2005/cwchu-v4hci05.pdf

19. Playstation EyeToy http://upload.wikimedia.org/wikipedia/commons/b/bf/EyeToy.JPG

20. Nintendo Wii http://www.tweaks.pl/images/articles/nintendo-kontroler.gif

21. Playstation Move http://s.cdaction.pl/obrazki/playstation-move-trailer_173ed.jpg

22. Microsoft Kinect http://www.mixmedia.pl/kupic_img/51/600x600_microsoft_kinect---kinect-adventures_1549149586.jpg

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

24. Microsoft Kinect SDK http://www.microsoft.com/en-us/kinectforwindows/

25. Hsynapse
http://4programmers.net/C_sharp/Gotowce/Sieci_neuronowe_aproksymacja_i_rozpoznawan

ie_pisma

26. Github https://github.com/

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 Beta,

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.

3. Instalacja bazy danych MySQL 5.5,
4. Przekopiowanie aplikacji Gestures Editor i Gestures Recognizer na dysk,
5. Importowanie schemy na serwerze bazy danych MySQL 5.5,
6. Podłączenie sensora Kinect do komputera i instalacja sterowników.