POLITECHNIKA POZNAŃSKA

WYDZIAŁ INFORMATYKI

MAGISTERSKA PRACA DYPLOMOWA

**APLIKACJA KOMPUTEROWA DO BEZDOTYKOWEJ OBSŁUGI DOKUMENTACJI MEDYCZNEJ Z WYKORZYSTANIEM PLATFORMY MICROSOFT KINECT**

**inż. Alicja Szulta**

### 

**Promotor**

**dr inż. Janusz Pochmara**

Poznań, 2013

Spis treści

[1. Wstęp 4](#_Toc396286871)

[2. Platforma Microsoft Kinect 7](#_Toc396286872)

[2.1. Budowa sprzętowa 7](#_Toc396286873)

[2.2. Software Development Kit 11](#_Toc396286874)

[2.2.1. Naturalny Interfejs Użytkownika (NUI) 13](#_Toc396286875)

[2.2.2. Przechwytywanie strumieni danych 14](#_Toc396286876)

[2.2.3. Color Stream 15](#_Toc396286877)

[2.2.4. Depth Stream 16](#_Toc396286878)

[2.2.5. Skeleton Stream 17](#_Toc396286879)

[2.2.6. Interaction Stream 21](#_Toc396286880)

[2.3. Windows Developer Toolkit 22](#_Toc396286881)

[3. Implementacja systemu 24](#_Toc396286882)

[3.1. Wybór zastosowanych technologii 24](#_Toc396286883)

[3.2. Gesty 25](#_Toc396286884)

[3.2.1. Implementacja rozpoznawania gestów z użyciem wzorca 26](#_Toc396286885)

[3.2.2. Rozpoznawanie gestów dwuręcznych z użyciem wzorca 34](#_Toc396286886)

[3.2.3. Algorytmiczna implementacja rozpoznawania gestów 40](#_Toc396286887)

[3.2.4. Podsumowanie implementacji gestów 46](#_Toc396286888)

[3.3. Interfejs programu 47](#_Toc396286889)

[3.3.1. KinectRegion 48](#_Toc396286890)

[3.3.2. ZoomBorder 50](#_Toc396286891)

[3.3.3. UcImageSelection 53](#_Toc396286892)

[4. Podsumowanie 55](#_Toc396286893)

[5. Spis ilustracji 56](#_Toc396286894)

[5.1. Schematy 56](#_Toc396286895)

[5.2. Rysunki 57](#_Toc396286896)

[6. Bibliografia 58](#_Toc396286897)

# Wstęp

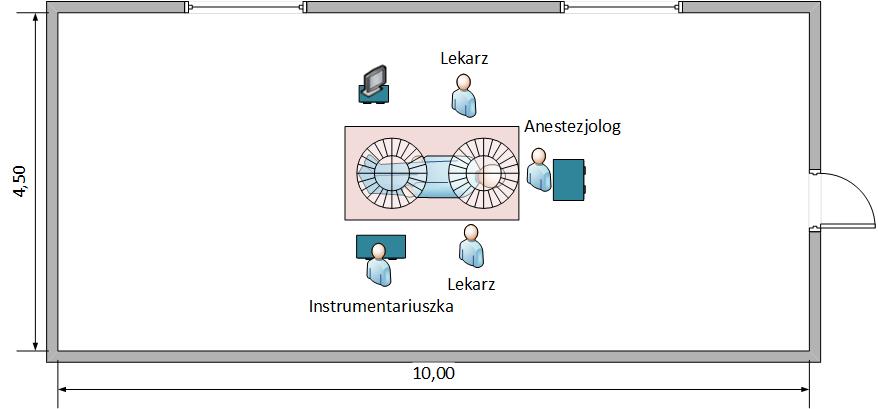
Wraz z rozwojem nauki i technologii, dostępne dla zwykłych użytkowników stają się coraz bardziej skomplikowane urządzenia, których obsługa staje się coraz bardziej intuicyjna, przyjazna i łatwa do nauki. Zmianie ulegają metody interakcji z urządzeniami - sterowanie dotykiem wypiera dotychczasowe klawiatury, sterowanie głosem staje się coraz powszechniej stosowane, komunikacja bezdotykowa (oparta na ruchach, gestach, mimice) już nikogo nie dziwi.

Ze względu na wymaganą sterylność, technologie bezdotykowe wkraczają w przestrzeń medyczną. Urządzeniem, które pozwala na bezdotykowe sterowanie aplikacjami jest Microsoft Kinect. Wśród projektów wykorzystujących Kinect'a w medycynie znajdują się: zastosowanie platformy Kinect do systemu rehabilitacji pozawałowej , Reflexion Rehabilitation Measurement Tool - system rehabilitacji po operacjach mięśni szkieletowych oraz systemu do przetwarzania dokumentacji medycznej podczas trwania operacji .

W ramach pracy magisterskiej zrealizowano projekt mający na celu usprawnienie korzystania z dokumentacji medycznej podczas zabiegów dokonywanych na sali operacyjnej jednego z poznańskich szpitali. W toku prac wstępnych nad projektem, odbyto konsultacje z lekarzami oraz zapoznano się z rzeczywistymi warunkami panującymi na sali operacyjnej.

Obecnie, podczas wykonywania procedur medycznych, dokumentacja medyczna dostępna jest w formie papierowej. W związku z wymaganiem sterylności podczas operacji, konieczna jest obecność na sali osoby, która może prezentować lekarzom dokumentację (wyniki badań, zdjęcia, prześwietlenia). W ramach projektu postanowiono zrealizować system, który pozwoli na wyświetlanie i prostą obróbkę (przybliżanie, oddalanie, obrót) dokumentacji medycznej w postaci elektronicznej. Co istotne, system musiał spełniać kryterium bezdotykowego sterowania. W związku z tymi wymaganiami zdecydowano się na wykorzystanie urządzenia MIcrosoft Kinect do stworzenia aplikacji komputerowej, która pozwoli na swobodne przeglądanie dokumentacji podczas zabiegów.

Poniższy schemat przedstawia rzut poziomy sali operacyjnej.



Schemat 1.1 Sala operacyjna - widok z góry (opracowanie własne)

Na środku sali znajduje się stół operacyjny, na którym leży operowany pacjent. Zlokalizowane bezpośrednio nad stołem lampy są źródłami bardzo intensywnego światła. Przy zabiegu/operacji może brać udział do dwóch lekarzy, anestezjolog oraz instrumentariuszka. Lekarze stoją po przeciwnych stronach stołu operacyjnego, anestezjolog znajduje się przy końcu stołu od strony głowy pacjenta, a instrumentariuszka obok jednego z lekarzy, od strony nóg pacjenta. Instrumentariuszka ma przed sobą stolik z narzędziami potrzebnymi przy wykonywanych procedurach medycznych (skalpele, nici etc.), natomiast anestezjolog przez cały czas operacji monitoruje stan pacjenta za pomocą specjalistycznej aparatury (na schemacie zaznaczona prostokątem za anestezjologiem). Ponadto, na sali znajduje się sprzęt medyczny, z którego korzysta się podczas zabiegu (na schemacie zaznaczone jako ekran po lewej stronie lekarza powyżej stołu operacyjnego). Warto zauważyć, że pacjent podłączony jest do wielu urządzeń medycznych, dlatego między stołem a aparaturą anestezjologa oraz pozostałą aparaturą przebiegają rozmaite połączenia, rurki, ssaki itp.

Na schemacie uwzględniono wymiary sali. Są to wymiary orientacyjne otrzymane od obsługi szpitala. Są one istotne ze względu na zasięg prawidłowej pracy urządzenia Kinect. Przedstawiono także umieszczenie okien. Okna są koloru mlecznego (nieprzezroczyste), przez co zapobiegają wpadaniu ostrego światła do sali operacyjnej. Ten fakt również jest bardzo ważnym czynnikiem, który należy wziąć pod uwagę rozważając umieszczenie urządzenia Kinect tak, żeby było to funkcjonalne dla obsługi medycznej oraz zapewniało prawidłowe działanie projektowanego systemu.

Podczas wykonywania operacji lekarze muszą zachować sterylność. Oznacza to, że nie mają możliwości sterowania żadnymi urządzeniami przy użyciu rąk, ponieważ są one w bezpośrednim kontakcie z organizmem pacjenta. Wśród obecnych na sali osób jest tylko jedna, która może dotykać różnych sprzętów, ale ma ona obowiązki podczas wykonywania procedur medycznych, stąd nie można zaangażować jej do sterowania systemem komputerowym do wyświetlania dokumentacji medycznej. Jest to przyczyna, dla której używanie myszki czy klawiatury - standardowych urządzeń do sterowania komputerem - nie jest możliwe podczas zabiegu. Jednakże, podczas przygotowania do wykonywania zabiegu odbywa się inicjalizacja sprzętu medycznego, kiedy to kontakt bezpośredni jest możliwy. Dokumentacja medyczna znajduje się na elektronicznych nośnikach danych (płyty CD).

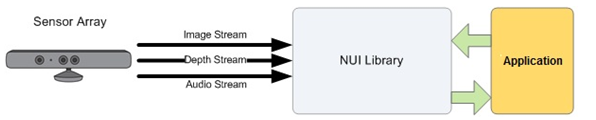
Odbywając konsultacje z lekarzami zapytano o ich potrzeby względem projektowanego systemu. Na podstawie zebranych informacji sformułowano następujące wymagania projektowe:

* system musi być sterowany bez bezpośredniego kontaktu (bezdotykowo),
* podczas inicjalizacji systemu przed zabiegiem dopuszczalny jest bezpośredni kontakt ze sprzętem (a więc sterowanie myszką, klawiaturą),
* system ma zapewniać funkcjonalność powiększania, pomniejszania dokumentacji medycznej (zdjęć) oraz ich obracanie,
* ustawienie urządzenia Microsoft Kinect na sali operacyjnej musi zapewniać funkcjonalność systemu dla użytkowników (nie może kolidować z aparaturą medyczną i wymagać zbytniego przemieszczania się po sali),
* ustawienie urządzenia Microsoft Kinect na sali operacyjnej musi zapewniać prawidłowe działanie systemu (problem wielu źródeł intensywnego światła oraz wielu osób przebywających na sali i biorących aktywny udział w przeprowadzanej operacji).

Sposoby rozwiązania oraz argumentacja przyjętych metod realizacji powyższych wymagań opisana jest w zasadniczej części niniejszej pracy.

# Platforma Microsoft Kinect

Urządzenie Kinect jest platformą wykrywania ruchu. Na stronie internetowej producenta można przeczytać, że Kinect for Windows "jest zestawem technologii, które umożliwiają ludziom naturalną komunikację z komputerami; czujnik oraz darmowy zestaw bibliotek i narzędzi programistycznych (SDK) Kinect for Windows dostarczają deweloperom fundament wymagany do tworzenia i rozwijania interaktywnych aplikacji, które reagują na ludzki naturalny ruch, gesty i polecenia głosowe" (tłumaczenie własne). Można zatem stwierdzić, że zestaw Kinect składa się z dwóch zasadniczych elementów - platformy sprzętowej oraz platformy programistycznej. Dane, otrzymane w wyniku przetwarzania otoczenia przez czujniki platformy Kinect, przekazywane są do aplikacji. Dzięki wykorzystaniu naturalnego interfejsu użytkownika (NUI - ang. *Natural User Interface*), sygnały wejściowe urządzenia Kinect oraz komendy sterujące programem są przyjazne i intuicyjne. Te zasady współdziałania platformy programistycznej i sprzętu przedstawia poniższy schemat.



Schemat . Interakcja sprzętu i oprogramowania z aplikacją (6)

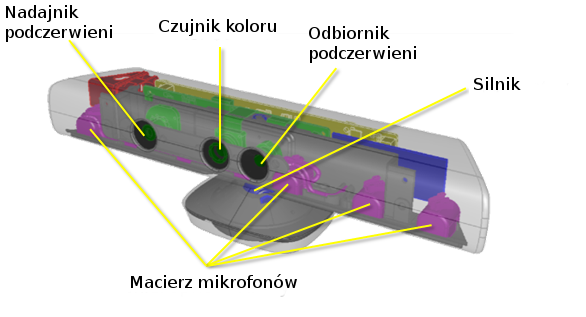
Ponadto, przydatnym narzędziem w pracy nad programami wykorzystującymi urządzenie Kinect for Windows jest Windows Developer Toolkit. Wszystkie te komponenty omówione są w podrozdziałach niniejszego rozdziału.

## Budowa sprzętowa

Urządzenie Kinect składa się z następujących podzespołów:

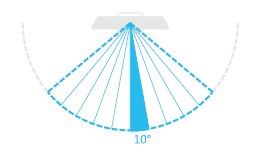
* macierz mikrofonów kierunkowych,
* promiennik podczerwieni,
* odbiornik podczerwieni,
* kamera RGB,
* akcelerometr,
* silnik

Ich rozmieszczenie obrazuje poniższy schemat:



Schemat 2.2 Podzespoły urządzenia Kinect for Windows - opracowanie własne na podstawie (7)

Macierz mikrofonów składa się z czterech mikrofonów, posiada 24-bitowy przetwornik analogowo-cyfrowy. Układ wykrywa dźwięki z obszaru w zakresie pięćdziesięciu stopni na prawo i pięćdziesięciu stopni na lewo od środka czujnika. Obszar ten można podzielić na dziesięć dziesięciostopniowych fragmentów, co pozwala określić które dźwięki pochodzą z precyzyjnie określonego kierunku. Jest to rozwiązanie szczególnie przydatne do wybierania przetwarzanych dźwięków na podstawie położenia ich źródła.

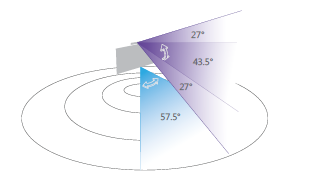


Schemat 2.3 Zakres wykrywania dźwięków (8)

Ponadto, dopuszczalne jest wprowadzenie progu natężenia dźwięku - Kinect umożliwia odfiltrowanie dźwięków poniżej 20 decybeli. Co więcej, dźwięki pochodzące zza czujnika są tłumione o 6dB. Zapewniona jest obsługa dźwięku w formacie PCM z częstotliwością próbkowania 16 kHz, z rozdzielczością 24 bitów na próbkę reprezentującym dźwięk monofoniczny (1 kanał). Z poziomu programowego dostępne są funkcje śledzenia głosu jednego ze śledzonych graczy oraz śledzenia najgłośniejszego źródła dźwięku.

Promiennik i odbiornik podczerwieni razem stanowią podsystem sensora głębokości (ang. *depth camera*), który pozwala na wykrywanie odległości pomiędzy przedmiotami i osobami znajdującymi się w zasięgu ich działania a Kinect'em. Zmierzona wartość wyrażona jest w milimetrach. Układ sensora głębokości przekazuje obraz z prędkością 30 FPS w jednej z dostępnych rozdzielczości - 320x420, 640x480 oraz 80x60. Zasięg pracy czujnika stanowi przestrzeń zakreślona między 43,5 stopniami w pionie (+/- 27 stopni względem poziomu czujnika) i 57,5 stopniami w poziomie.

Kamera RGB dostarcza obraz z taką samą prędkością jak sensor głębokości (30 FPS). Zakres pracy czujnika jest taki sam jak dla sensora głębokości. Schematycznie obrazuje to poniższy rysunek.



Schemat . Zasięg kątowy pracy czujnika głębokości i RGB

W związku z fizycznymi ograniczeniami czujników obrazu oraz głębokości, efektywne korzystanie z Kinect'a wymaga przestrzegania pewnych wymagań dotyczących obszaru działania użytkownika. Dostępne są dwa tryby działania sensora głębokości - domyślny (ang. *default*) oraz bliski (ang. *near*). Tryb działania sensora wpływa na to jakie odległości użytkownika od urządzenia Kinect są dopuszczalne i prawidłowo rozpoznawalne. W trybie domyślnym Kinect ma zakres pracy w odległości od 0.8 do 4 metrów, jednak dla najlepszej efektywności otrzymywanych sygnałów zalecane jest użytkowanie Kinect'a w odległości 1.2 do 3.5 metra od czujnika. Czujnik głębokości pracuje poprawnie także dla dystansów większych niż zalecane, ale informacje o śledzeniu i rozpoznawaniu graczy mogą być już niewiarygodne. W trybie pracy bliskiej (ang. *near mode*) zalecana odległość od czujnika zapewniająca stabilną pracę wynosi od 0.8 do 3 metrów, zaś fizycznie dopuszczalny obszar zamyka się między 0.4 a 3 metry od Kinect'a.

Podczas prac nad niniejszym projektem, gdy próbowano ustawić w programie tryb *near mode*, okazało się, że model urządzenia Kinect, którym dysponowała autorka nie jest egzemplarzem dedykowanym do użytku z systemem Windows (a co za tym idzie z pakietem bibliotek Kinect SDK) ale do użytku z konsolą Xbox. W związku z tym, tryb pracy bliskiej jest dla niego niedostępny. Poniżej przytoczono komunikat, który pojawił się podczas debuggowania błędu:

The Kinect sensor plugged into your computer is for use on the Xbox 360. You may continue using your Xbox 360 Kinect sensor on your computer for development purposes. Microsoft does not guarantee full compatibility for Kinect for Windows applications and the Xbox 360 Kinect Sensor.

Powyższa informacja oznacza, że urządzenie Kinect podłączone do komputera jest dedykowane do użytku z konsolą Xbox 360; dopuszczalne jest używanie go w celu rozwoju oprogramowania, jednak Microsoft nie gwarantuje pełnej kompatybilności aplikacji dedykowanych dla Kinect for Windows z czujnikiem Kinect dedykowanym dla Xbox 360 (tłumaczenie własne). W związku z tym, dla pełnej sprawności programu, zaleca się korzystanie z czujnika Kinect for Windows.

Kinect wyposażony jest także w trójosiowy akcelerometr, który może pracować w trybach 2G, 4G oraz 8G (domyślnie skonfigurowany na 2G). Obecność akcelerometru na platformie Kinect pozwala na ustalenie aktualnej orientacji układu względem kierunku grawitacji. Zaimplementowane w SDK metody pozwalają ponadto ustalić orientację względem poziomu. Daje to możliwość manipulacji kątem pochylenia zestawu Kinect for Windows za pomocą silniczka, który wmontowany jest w podstawkę urządzenia. Układ pozycjonowania Kinect'a działa z dokładnością jednego stopnia, jednak jak przestrzega producent, akcelerometr jest czuły na temperaturę pracy. W związku z tym, przekroczenie normalnej temperatury pracy (5-35o Celsjusza ) może skutkować w maksymalnie trzystopniowym odchyleniu od deklarowanej dokładności pomiaru akcelerometru (1o). Na szczęście, dla konkretnego urządzenia odchylenie to ma zawsze stałą wartość.

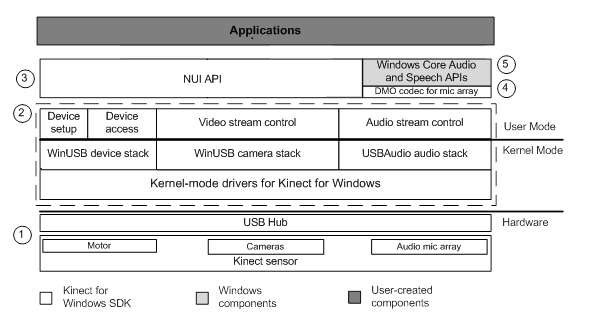
Podczas korzystania z Kinect'a w aplikacjach systemu Windows trzeba pamiętać, że ponieważ czujnik jest wyposażony jedynie w procesor sygnałowy (ang. *Digital Signal Processor*) poddający obróbce sygnał dźwiękowy z mikrofonów, przetwarzanie danych odbywa się po stronie sterownika Kinect'a na komputerze.

## Software Development Kit

Kinect for Windows Software Development Kit to zestaw pomocy do obsługi programowej czujnika na komputerach z systemem operacyjnym Windows. Zestaw ten zawiera:

* sterowniki i dokumentację techniczną,
* dokumentację i zestaw źródłowych API w języku C++, C# oraz VisualBasic,
* przykładowe projekty przedstawiające wykorzystanie dostarczonych API.

Strukturę dostarczonego SDK przedstawia niniejszy schemat:



Schemat 2.5 Architektura SDK (6)

Bloki z białym tłem oznaczają komponenty Kinect for Windows SDK, bloki z jasnoszarym tłem symbolizują komponenty systemu Windows, a bloki z ciemnoszarym tłem obrazują komponenty użytkownika (tutaj - aplikację wykorzystującą urządzenie Kinect).

Obszar oznaczony numerem jeden odnosi się do sprzętowej budowy platformy Kinect omówionej szerzej w poprzednim podrozdziale. Moduły obszaru numer dwa współpracują ze sobą w ramach sterowników urządzenia Kinect. Dzięki driverom macierz mikrofonów może być używana przez użytkownika systemu /Windows jako standardowe wejście audio. Obszar trzeci to obszar NUI (ang. *Natural User Interface*), na który składają się komponenty audio i wideo Kinect'a. Blok czwarty składa się z kodeku DMO[[1]](#footnote-1) (ang. *DirectX Media Object*) dla macierzy mikrofonów. Blok piąty to blok standardowych API Windows 7 dostępnych także dla aplikacji desktopowych Windows 8. Z powyższego schematu wynika, że czujnik Kinect wraz z dostarczonymi bibliotekami SDK tworzy całościową platformę programistyczną zintegrowaną na poziomie sprzętowym.

W ramach pracy magisterskiej korzystano z Kinect for Windows SDK w wersji 1.8. Dla tej wersji zestawu producent deklaruje wsparcie dla następujących systemów operacyjnych:

* Windows 7
* Windows 8
* Windows Embedded Standard 7
* Windows Embedded Standard 8

W przypadku gdy posiadany przez użytkownika system Windows 7 występuje w wersji Windows 7N albo Windows 7 KN, wymagane jest doinstalowanie zestawu Media Feature Pack . Ponadto, występują też wymagania sprzętowe dla prawidłowego działania dostarczonych narzędzi:

* procesor 32-bit (x86) lub 64-bit (x64)
* procesor Dual-core, 2.66-GHz lub szybszy
* szyna USB 2.0 dedykowana dla urządzenia Kinect
* 2 GB pamięci RAM
* karta graficzna wspierająca DirectX 9.0c

Zestaw SDK dostępny jest na zasadach wolnej licencji dla użytkowników niekomercyjnych. Szczegółowe zasady określone są w dokumencie EULA dostępnym na stronie Microsoftu. Warto jednak wspomnieć, iż na mocy licencji dopuszczalne jest projektowanie, tworzenie i testowanie oprogramowania Kinect for Windows Application przy użyciu urządzenia Kinect for Xbox 360, jednakże użytkownik końcowy, aby legalnie korzystać ze stworzonego programu, musi posiadać sensor Kinect dedykowany dla systemu Windows (12). Jest to także związane z faktem, iż podczas tworzenia i testowania programu obsługującego urządzenie Kinect, na komputerze dewelopera czujnik Kinect for Xbox 360 jest warunkowo obsługiwany. Na innych maszynach, próba włączenia tego samego programu poprzez podłączenie Kinect'a dedykowanego do konsoli Xbox zostanie uniemożliwiona - urządzenie nie zostanie wykryte jako obsługiwane (13).

### Naturalny Interfejs Użytkownika (NUI)

Naturalny interfejs użytkownika jest to interfejs użytkownika, który zapewnia łatwe i szybkie nauczenie się obsługi dzięki temu, że nie jest konieczne używanie sztucznych metod i urządzeń do interakcji z aplikacją, ale wykorzystywane są naturalne metody komunikacji (np. mowa, gesty) . "Naturalność" NUI polegać powinna na tym, że przyjęte sposoby komunikacji oparte są na wrodzonych umiejętnościach, niegdyś nauczonych metodach. Nie oznacza to, że w ogóle nie wymagają one nauczania, choć pochodzą z życia codziennego, przez co są zrozumiałe dla użytkownika . Drugą cechą charakterystyczną dla naturalnego interfejsu użytkownika jest jego bezpośredniość. Innymi słowy, medium uczestniczące w interakcji użytkownika z programem/urządzeniem nie jest widoczne ani angażujące w żaden sposób. Tak na przykład, przy komunikacji głosowej nie występuje żadne narzędzie, którego obsługą obarczony byłby użytkownik. Co prawda, skomplikowane algorytmy przetwarzają sygnały dźwiękowe na komendy interfejsu, niemniej jednak użytkownik nie ponosi z tego tytułu żadnego obciążenia, co więcej nawet nie widzi, że takie procesy zachodzą .

Programista korzystający z Kinect for Windows API buduje interfejs programu przy użyciu danych przechwytywanych przez NUI. Dane te zawierają się w strumieniu danych audio oraz w strumieniu obrazu i strumieniu głębokości. Te trzy typy sygnałów wejściowych dają możliwość stworzenia interfejsu programu przyjaznego użytkownikowi. Na podstawie pomiarów sensora głębokości oprogramowanie obsługujące urządzenie Kinect rozpoznaje i śledzi ludzkie ciało. System jest w stanie rozpoznać nie więcej niż sześciu użytkowników jednocześnie, z czego dwóch może być aktywnie śledzonych (rozpoznawanie i śledzenie użytkowników jest szerzej omówione w kolejnych częściach pracy). Kompatybilność z Microsoft Speech API umożliwia implementację komend słownych, za pomocą których dostępne jest sterowanie aplikacją korzystającą z Kinect'a. Ponadto, dzięki dostępnym strumieniom danych zrealizowano śledzenie twarzy przy pomocy omawianego urządzenia. Wszystkie te rozwiązania dostępne są dla deweloperów w Kinect for Windows SDK.

W kolejnych sekcjach opisane są strumienie danych, z których korzystano na potrzeby pracy. Jako że nie używano przetwarzania dźwięków, nie omawiano strumienia audio. W celu głębszego poznania specyfiki tego strumienia odsyłam do dokumentacji dostępnej na stronie Microsoftu: http://msdn.microsoft.com/en-us/library/jj131026.aspx.

### Przechwytywanie strumieni danych

Strumienie danych dostępnych z czujników platformy Kinect można programowo przechwytywać dzięki API dostarczonemu wraz z Kinect for Windows SDK. Zestaw bibliotek daje możliwość uzyskania dostępu do danych w dwojaki sposób - poprzez zdarzenia wywoływane przy otrzymaniu każdej klatki z odpowiednimi danymi oraz poprzez mechanizm odpytywania o kolejne klatki.

Metoda odpytywania polega na tym, aby po inicjalizacji danego strumienia odpytywać go ręcznie o nową ramkę danych. Zapytanie trwa tak długo jak zdefiniowany okres czasu. Jeżeli okres ten minie, a nowa ramka danych nie nadejdzie, zwracana jest wartość *null* (w języku C#). Dzieje się to za pomocą metody *OpenNextFrame* wywołanej dla żądanego typu strumienia danych.

Aby przechwytywać ramki z danymi metodą zdarzeń (ang. *events*), należy zapisać się na wywoływane przez strumień zdarzenie. W języku C# dzieje się to przez uchwyt *ColorFrameReady, DepthFrameReady, SkeletonFrameReady* albo *AllFramesReady* obiektu klasy *KinectSensor*. Trzy pierwsze wymienione uchwyty pozwalają na przechwycenie klatek danego typu (odpowiednio - kolorowego/podczerwonego, głębokości, szkieletu). Uchwyt *AllFramesReady* zawiera wszystkie trzy typy klatek. Zdarzenia typu *ColorFrameReady*, *DepthFrameReady* oraz *SkeletonFrameReady* wywoływane są w momencie, gdy dostępna jest nowa klatka danego typu - wówczas następuje zapytanie o tę nową ramkę danych (*OpenNextFrame*). Natomiast zdarzenie *AllFramesReady* wywoływane jest, gdy dostępne są nowe klatki dla każdego typu strumienia . Zdarzeniowy mechanizm przechwytywania danych z czujników urządzenia Kinect jest charakterystyczny dla platformy Windows Presentation Foundation, której użyto do realizacji projektu.

Warto zauważyć, iż dla każdy strumień można obsługiwać za pomocą dowolnie wybranej metody. W związku z tym, dopuszczalny jest stan kiedy strumień obrazu kolorowego obsługiwany będzie mechanizmem zdarzeń, a strumień głębokości zapytaniami. Jednakże, w przypadku zapisania się na zdarzenie *AllFramesReady*, manualne odpytywanie któregokolwiek ze strumieni powoduje wystąpienie wyjątku typu *InvalidOperationException* .

### Color Stream

Strumień obrazu kolorowego (RGB) umożliwia przechwycenie zwykłego obrazu z kamery Kinect'a z prędkością od 12 do 30 klatek na sekundę. Obraz dostępny jest w jednym z trzech zdefiniowanych formatów - RGB, YUV oraz Bayer. Niezależnie od wybranego formatu obraz zawsze reprezentuje te same dane, gdyż przetwarzanie do wybranego formatu odbywa się z tych samych danych źródłowych pochodzących z kamery . Podczas inicjalizacji strumienia w kodzie programu należy wybrać rozdzielczość (dostępne są 80x60, 320x240, 640x480 oraz 1280x960 pikeli) i format przechwytywanego obrazu. Tak zdefiniowanych parametrów nie można zmieniać podczas przechwytywania danych - aby je przeformułować należy wyłączyć i ponownie włączyć strumień danych obrazu kolorowego (z nowymi wartościami parametrów).

Klatka obrazu pochodzącego z kamery RGB urządzenia Kinect składa się z pikseli. Ich ilość zależy od wybranej rozdzielczości, zatem układ współrzędnych (x,y) obrazu związany jest z jego rozmiarem. Każdy piksel posiada informację o wartościach kolorów czerwonego, zielonego i niebieskiego.



Rysunek 2.1 Obraz kolorowy (źródło własne)

Strumień RGB posiada także w sobie strumień pochodzący z czujnika podczerwieni. Podczas przechwytywania strumienia można korzystać tylko z jednego obrazu na raz - albo z czujnika RGB, albo z czujnika podczerwieni .

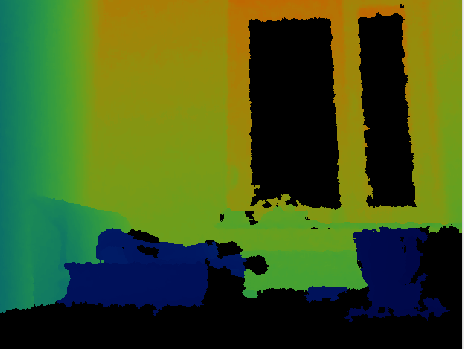


Rysunek 2.2 Obraz podczerwony (źródło własne)

### Depth Stream

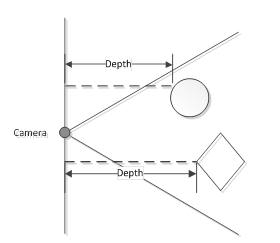
Strumień głębokości obrazu dostępny jest pod nazwą *DepthStream*. Każdy piksel ramki typu *DepthImage* zawiera informację o odległości wykrytego obiektu (najbliższego w danym punkcie obszaru widzenia sensora głębokości) od płaszczyzny czujnika wyrażoną w milimetrach. Ponadto, każdy piksel posiada informację o tym, który użytkownik na nim występuje. Jest to określone za pomocą indeksu w zakresie od 0 do 6, ponieważ Kinect zapewnia wykrywanie do sześciu użytkowników (ang. *player segmentation data*). Indeksowanie użytkowników dostępne jest tylko wtedy, kiedy oprócz strumienia głębokości zainicjalizowany jest strumień informacji o szkielecie (ang. *skeleton stream*) . Wartości indeksu w zakresie 1-6 odpowiadają indeksom użytkowników w tablicy danych szkieletu pomniejszonym o jeden, wartość 0 oznacza, że w danym miejscu nie ma żadnego z użytkowników Kinect'a.

Obraz przechwytywany ze strumienia głębokości może przyjąć jedną z dostępnych rozdzielczości - 640x480, 320x240 albo 80x60 pikseli. Zakres pracy bliskiej (ang. *near mode*) oraz domyślnej (ang. *default mode*) omówione są w sekcji dotyczącej sprzętowych uwarunkowań platformy Kinect.



Rysunek 2.3 Obraz głębokości (źródło własne)

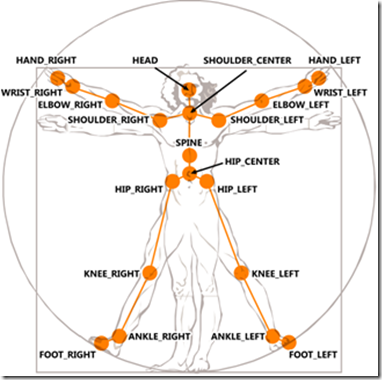
Dane otrzymywane ze strumienia głębokości zawierają odległość do najbliższego wykrytego obiektu w konkretnym punkcie. Zależność tę ilustruje schemat:



Schemat 2.6 Układ współrzędnych dla strumienia głębokości (19)

### Skeleton Stream

Dane strumienia szkieletu obliczane są na podstawie informacji ze strumienia RGB z czujnika podczerwieni. Do danych strumienia szkieletu należy położenie stawów, ich orientacja oraz rodzaj. Urządzenie umożliwia wykrywanie do sześciu osób (szkieletów), przy czym zapewnia śledzenie maksymalnie dwóch. Śledzenie oznacza wykrywanie i przetwarzanie informacji o wszystkich dostępnych ("widzialnych" dla czujnika) stawach użytkownika. Poniższy rysunek przedstawia wszystkie obsługiwane przez SDK stawy oraz ich umiejscowienie na ciele człowieka.



Rysunek 2.4 Układ stawów - Skeleton Stream (20)

Praca strumienia szkieletu może odbywać się w jednym z dwóch dostępnych trybów - w trybie pełnego szkieletu (ang. *full skeleton mode*) oraz w trybie siedzącym (ang. *seated mode*). Śledzenie w trybie pełnego szkieletu rozpoznaje wszystkie 20 stawów, zaś w trybie siedzący skupia się tylko na dziesięciu górnych stawach.



Rysunek . Tryb siedzący

Interesującym zagadnieniem jest filtrowanie i wygładzanie położenia stawów przy użyciu parametrów typu *TransformSmoothParameters*. Konieczność skorzystania z funkcji wygładzania położenia stawów wynika z błędów pomiaru sensorów oraz występujących w otoczeniu szumów. Wymienione niedokładności są konsekwencją wielu czynników determinujących wiarygodność pomiarów - od dynamicznie zmieniającego się oświetlenia po kształty i rozmiary użytkownika . Inicjalizując w kodzie programu wykorzystującego platformę Kinect strumień szkieletu, można uwzględnić parametry filtrowania. Struktura typu *TransformSmoothParameters* składa się z następujących pól:

* *Smoothing* - parametr wygładzania, wzrost parametru wpływa na wzrost opóźnień, zakres wartości 0-1,
* *Correction* - korekcja, niższe wartości korygują wolniej, ale płynniej, wyższe wartości korygują szybko (i skokowo), zakres wartości 0-1,
* *Prediction* - predykcja, ilość klatek do przewidzenia z wyprzedzeniem, zakres wartości - od zera w górę,
* *JitterRadius* - kąt fluktuacji, kąt wyrażony w metrach do redukcji fluktuacji,
* *MaxDeviationRadius* - maksymalny kąt wyrażony w metrach, o który mogą się odchylać wartości przefiltrowane od wartości surowych (wejściowych) .

W ramach pracy wartości parametrów dobrano eksperymentalnie. Ostateczna postać omawianej struktury przyjęła następujące współczynniki:

Smoothing = 0.75f,

Correction = 0.07f,

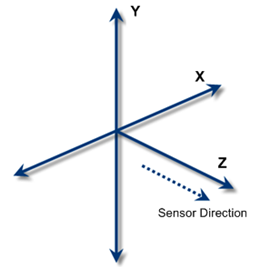
Prediction = 0.08f,

JitterRadius = 0.08f,

MaxDeviationRadius = 0.07f.

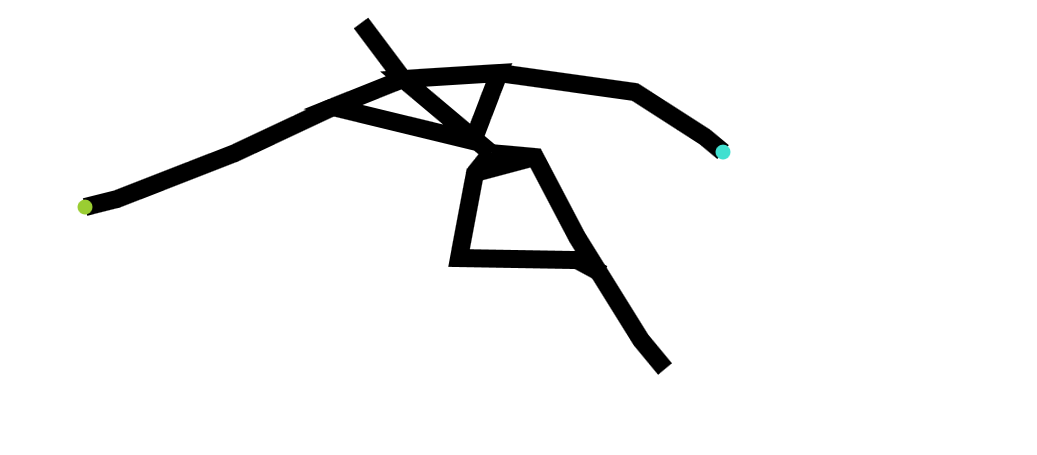
Każdy z rozpoznanych szkieletów posiada właściwość stanu śledzenia (ang. *tracking state*) określoną jednym z trzech stanów - śledzony (ang. *tracked*), tylko pozycja (ang. *position only*) albo nieśledzony (ang. *not tracked*). Śledzenie użytkownika (szkieletu) oznacza pozycjonowanie jego stawów w przestrzeni, tylko pozycja oddaje położenie gracza w przestrzeni, jednak bez szczegółowych danych na temat jego poszczególnych części ciała. Podobnie, każdy ze stawów śledzonych szkieletów posiada określony stan - śledzony, nieśledzony oraz implikowany (ang. *inferred*). Stan implikowany (wnioskowany, dedukowany) oznacza, że dany staw nie jest bezpośrednio "widoczny" dla sensora Kinect, dlatego jego pozycja jest przypuszczana na podstawie położenia i kątów skręcenia pozostałych stawów. Jest to stan obarczony dużą niepewnością, dlatego wszelkie przetwarzanie położeń i kątów między stawami powinno się odbywać dla stawów w stanie śledzenia.

Strumień danych szkieletu związany jest przestrzenią trójwymiarową. Każdy śledzony punkt typu szkieletowego (ang. *SkeletonPoint*) posiada pozycję wyrażoną w układzie (x,y,z). Jednostką miary w omawianym układzie jest metr (należy o tym pamiętać, ponieważ w układzie strumienia głębokości wartości wyrażone są w milimetrach). Orientację tego układu przedstawia schemat.



Schemat . Układ współrzędnych strumienia szkieletu

Jest to prawoskrętny układ współrzędnych kartezjańskich. Oś z skierowana jest w stronę, w którą skierowany jest czujnik. Stąd, współrzędna z określa odległość użytkowników od urządzenia Kinect we współrzędnych szkieletowych. Wartości na osi y rosną w górę względem położenia czujnika, natomiast wartości dla osi odciętych x wzrastają wraz z przesuwaniem się obiektów bardziej w lewą stronę Kinect'a. .

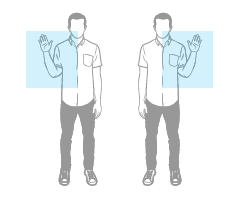


Rysunek . Śledzony szkielet (źródło własne)

### Interaction Stream

Kolejnym interesującym strumieniem, którym dysponuje urządzenie Kinect jest strumień interakcji (ang. *interaction stream*). Jest to o tyle ciekawy zestaw danych, że powstaje na podstawie danych ze strumienia szkieletu oraz strumienia głębokości przetworzonych przez złożone algorytmy, co pozwala uzyskać funkcję śledzenia dłoni i rozpoznawania gestów . Dostęp do strumienia interakcji w aplikacji można uzyskać, jeżeli użyje się dostarczonej z zestawem SDK biblioteki KinectInteraction. Ponadto, w bibliotece kontrolek obsługujących urządzenie Kinect (występuje w SDK) Microsoft.Kinect.Toolkit.Controls.dll występują komponenty wykorzystujące strumień interakcji do obsługi kursora dłoni. Do najważniejszych z nich należy zaliczyć wskaźnik dłoni (ang. *hand pointer*), który wykorzystywany jest jako odpowiednik kursora myszki w sterowaniu aplikacją windowsową wykorzystującą platformę Kinect. Bardzo korzystnym z deweloperskiego punktu widzenia jest fakt, iż biblioteki zawierające kontrolki korzystające ze strumienia interakcji dostępne są nie tylko z poziomu skompilowanej biblioteki (ang. *dynamic-link library*), ale dostarczone są także ich kody źródłowe. Dzięki temu, można dostosować kontrolki do własnych potrzeb - z tego rozwiązania skorzystano na potrzeby niniejszej pracy, co zostanie omówione w dalszych frgamentach.

Strumień interakcji generuje strefę fizycznej interakcji (ang. *Physical Interaction Zone, PhIZ*). Jest to dynamicznie obliczany obszar, osobny dla każdej z rozpoznanych dłoni, związany z położeniem ciała użytkownika. Położenie i rozmiar stref interakcji dla poszczególnych dłoni użytkownika przedstawia schemat.



Schemat 2.8 Fizyczna strefa interakcji (ang. *Physical Interaction Zone*) (8)

Wraz z przemieszczaniem się użytkownika aplikacji, jego strefa interakcji przemieszcza się razem z nim. Dzięki temu uzyskana jest ergonomia ruchów - ruchy ręki wewnątrz PhIZ rzutowane są na całą kontrolkę, dzięki czemu, aby kursorem dłoni osiągnąć skrajną krawędź ekranu nie trzeba sięgać ręką daleko w bok.

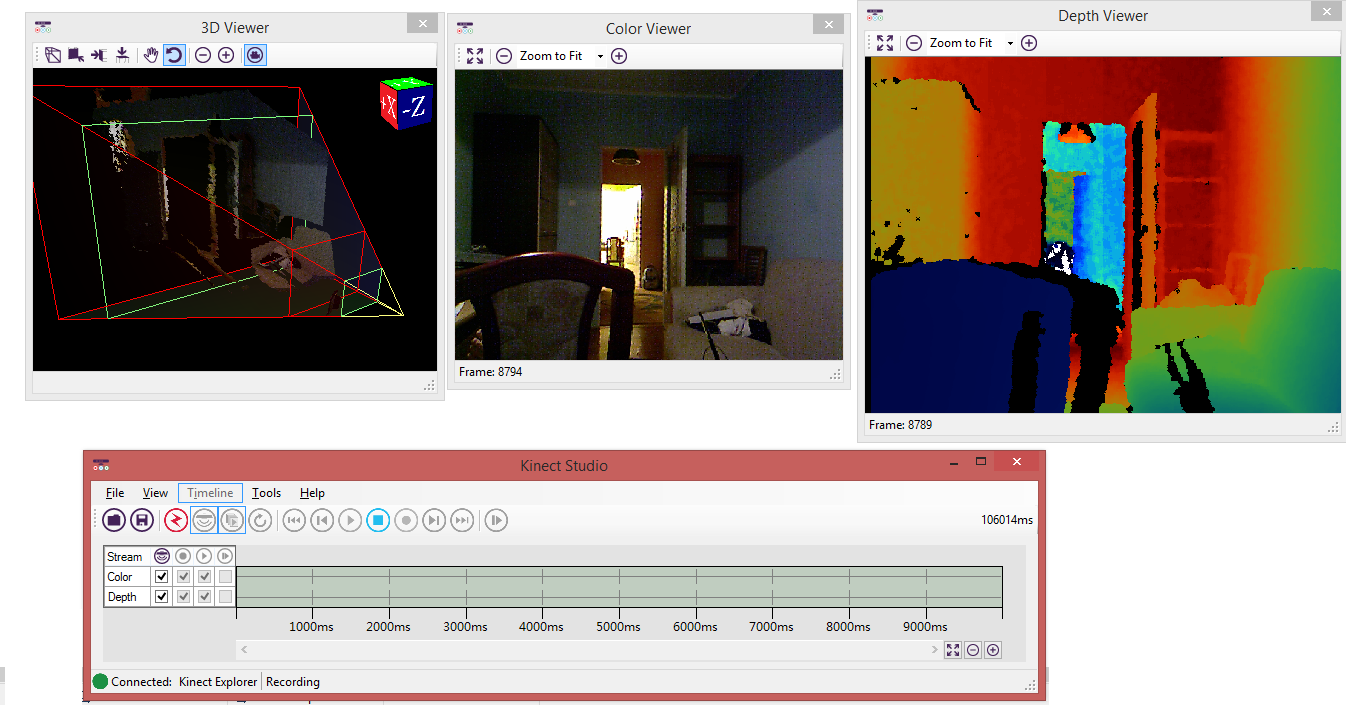
Poprzez zastosowane w strumieniu interakcji złożonych algorytmów, wybierana jest dłoń główna, za pomocą której odbywa się sterowanie aplikacją. Zasadniczo sprowadza się to do wyboru tej dłoni, która jako pierwsza została uniesiona. Podczas użytkowania aplikacji, dłoń główna (ang. *primary hand*) jest zmieniana dynamicznie - w zależności od ruchów i postaw przyjmowanych przez użytkownika programu. Domyślnie wspierane jest operowanie jedną dłonią na raz.

Strumień i bibliotek interakcji implementują także trzy rodzaje interakcji dłoni (kursora) z aplikacją. Są to - uchwycenie i puszczenie (ang. *grip and release*), wciśnięcie (ang. *press*) oraz przewijanie (ang. *scroll*). Dzięki tym narzędziom zaimplementowano naturalny interfejs użytkownika (NUI), który opiera się na komunikatach znanych z życia codziennego, przez co bardzo intuicyjnych dla użytkownika programu.

## Windows Developer Toolkit

Kinect for Windows Developer Toolkit (w ramach pracy używano w wersji 1.8.0) jest środowiskiem, w którym zebrano wszystkie komponenty zestawu SDK. Dzięki temu, programista korzystający z Kinect for Windows SDK może z jednego miejsca mieć dostęp do dokumentacji, aktualizacji, przykładowych projektów i niezbędnych bibliotek.

Bardzo ciekawym i wszechstronnym narzędziem, do którego można uzyskać dostęp m.in. z poziomu Windows Developer Toolkit jest Kinect Studio. Kinect Studio jest aplikacją, którą podłącza się do działającego programu wykorzystującego sygnały płynące z czujnika Kinect. W ten sposób Kinect Studio uzyskuje dostęp do strumieni danych - strumienia głębokości oraz koloru, a także umożliwia podgląd trójwymiarowej przestrzeni w zasięgu pracy czujników. Występuje także funkcjonalność nagrywania sesji uruchomionego programu, dzięki czemu później można odtwarzać i analizować odbierane i przetwarzane sygnały i obrazy. Zrzut ekranu pokazujący działanie Kinect Studio zamieszczony jest poniżej.



Rysunek 2.7 Kinect Studio (źródło własne)

# Implementacja systemu

Faza tworzenia systemu przeglądania dokumentacji medycznej podczas wykonywania procedur medycznych składała się z dwóch głównych elementów - projekt i implementacja wymaganych funkcjonalności oraz projekt i implementacja układu (ang. *layout*) interfejsu użytkownika programu.

## Wybór zastosowanych technologii

Skorzystano z technologii Windows Presentation Foundation, ponieważ umożliwia ona budowanie aplikacji komputerowych dedykowanych do obsługi materiałów multimedialnych. Co jednak było najistotniejsze przy wyborze tego rozwiązania, to fakt, iż WPF pozwalało na swobodną implementację interfejsu bezdotykowego. Często wykorzystywana technologia Windows Forms posiada interfejs przystosowany typowo do obsługi standardowymi urządzeniami wejściowymi - myszką i klawiaturą. Mnogość przycisków, rozwijane menu kontekstowe, układ okienek zdawały się być zbyt nieprzystępne do obsługi bezdotykowej. WPF z kolei udostępnia narzędzia do tworzenia oprogramowania także na urządzenia dotykowe (tablety, telefony). Co prawda możliwym jest zaimplementowanie obsługi kursora myszy za pomocą Kinect'a, jednak przy użyciu technologii Windows Forms byłoby to bardzo nieergonomiczne, nieintuicyjne i sprzeczne z naturalnym interfejsem użytkownika (NUI).

Mimo wspomnianych wyżej czynników, w jednym miejscu aplikacji wykorzystano standardową obsługę myszką - jest to pierwsze okno, które pokazuje się po uruchomieniu programu. Pozwala ono na wybór obrazów, które następnie użytkownik chce przeglądać za pomocą stworzonego systemu. W trakcie konsultacji z lekarzami ustalono, że inicjalizacja systemu przed samą procedurą medyczną, bez względu na późniejszy bezdotykowy interfejs, zachodzi przy pomocy standardowego sterowania myszką. Wynika to z faktu, iż dokumentacja danego pacjenta zapisana jest na nośniku CD i występuje konieczność jej skopiowania w celu przetwarzania. Inicjalizacja oprogramowania zostaje wykonana przez osobę, która przed rozpoczęciem wykonywania zabiegu lub operacji nie musi jeszcze zachować sterylności. Co więcej, wydaje się, że przeglądanie plików i folderów na komputerze obsługiwane bezdotykowo jest bardzo nieintuicyjnym i niezgrabnym rozwiązaniem.

Zdecydowano się na użycie języka programowania C# z trzech powodów:

* technologia WPF jest dostępna m.in. dla języka C#,
* biblioteki, przykładowe programy, kontrolki z zestawu SDK są dostępne w językach C++, C# oraz w technologii XNA,
* osobiste preferencje autorki.

W przypadku, gdyby przyjęte rozwiązanie okazało się zbyt wolne w działaniu, rozważano skorzystanie z języka C++, co jednak nie miało miejsca. W związku z tym, aplikację napisano przy użyciu technologii WPF w języku C#.

## Gesty

Wymaganiami co do funkcjonalności tworzonej aplikacji były: bezdotykowość oraz możliwości obrotu, przybliżania oraz oddalania wybranych zdjęć. Ponadto, dodano możliwość dodania kilku obrazów do programu i wyboru aktualnie przetwarzanego w trakcie korzystania z aplikacji. Aby zrealizować wymienione zagadnienia konieczne było zaimplementowanie rozpoznawania gestów. Czym zatem jest gest?

*"Gest jest ruchem ciała, który zawiera informacje. Machanie na pożegnanie jest gestem. Wciskanie klawisza na klawiaturze nie jest gestem, ponieważ ruch palca wykonywany w celu wciśnięcia przycisku nie jest ani obserwowany, ani istotny. Tym co jest ważne jest fakt czy klawisz został wciśnięty."* (15)(tłumaczenie własne)

Z przytoczonej definicji wynika, że tym co wyróżnia gest od innych ruchów i zachowań jest komunikat. Innymi słowy, aby mówić o wykonaniu gestu należy rozpoznać jego intencjonalność, uprzednią chęć wykonania go. Spośród wszystkich wykonywanych ruchów tylko te, które niosą ze sobą pewną informację można skategoryzować jako gest.

Aby podczas interakcji aplikacji z użytkownikiem skutecznie wychwytywać gesty spośród mnóstwa wykonywanych ruchów, trzeba przyjąć pewien klucz, według którego będą one rozpoznawane. Wyróżnia się trzy podstawowe metody rozpoznawania gestów. Są to:

* metody algorytmiczne,
* metody wykorzystujące sieci neuronowe,
* metody porównywania z wzorcem.

Metody algorytmiczne, jak sama nazwa wskazuje, polegają na algorytmicznym zapisie kryteriów, które należy spełnić, aby dany ruch rozpoznać jako gest. Jest to rozpoznawanie dychotomiczne - dany ruch zostaje skategoryzowany jako gest, albo nie. W związku z tym, najmniejsze odstępstwo wykonanego ruchu w stosunku do zdefiniowanego wzorca algorytmu skutkuje negatywnym wynikiem rozpoznawanie. W ramach niniejszej pracy zaimplementowano gesty przy pomocy metod algorytmicznych, co zostanie omówione szerzej w dalszej części rozdziału.

Kolejną grupą metod są metody implementujące sieci neuronowe do rozpoznawania gestów. Sieci neuronowe jako można zdefiniować następująco:

*"Zbiór połączonych ze sobą jednostek wejściowo-wyjściowych. Z każdym połączeniem skojarzona jest waga, która może zostać zmieniona w trakcie uczenia."* (24)

albo

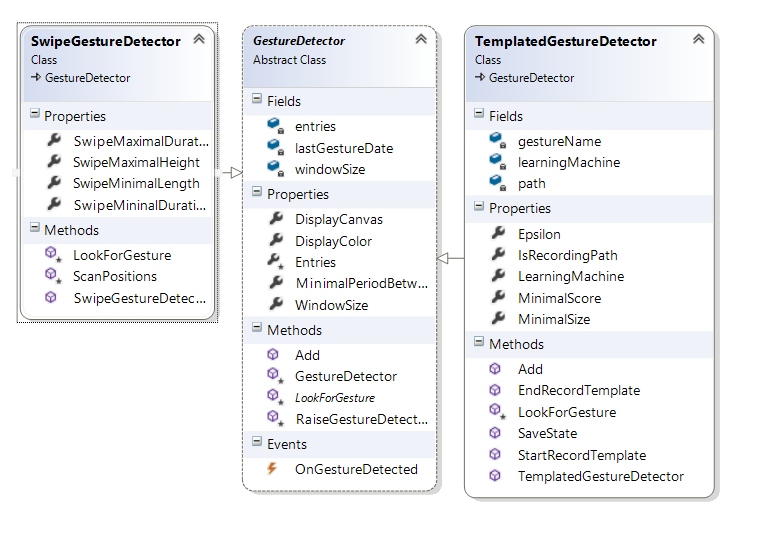
*"Sztuczna sieć neuronowa to wysoce równoległy rozproszony procesor złożony z prostych elementów obliczeniowych mający naturalną skłonność zapamiętywania podawanych informacji"* (25)

Ze względu na to, że sieci neuronowe stworzono po to, by naśladowały możliwości ludzkiego mózgu (gromadzenie, przetwarzanie i odzyskiwanie informacji), jest to zagadnienie z dziedziny sztucznej inteligencji . Korzystanie z sieci neuronowych jest dwuetapowe - na początku występuje etap uczenia, a następnie rozpoznawania przy pomocy wytrenowanego klasyfikatora. Z uwagi na złożoność zagadnienia i oraz osobiste preferencje zrezygnowano z użycia tej metody w niniejszej pracy.

Ostatnią z wymienionych technologii jest porównywanie z wzorcem. Metody z tej grupy działają na następującej zasadzie - najpierw budowana jest baza wzorcowych gestów, a następnie wykonywane przez użytkownika aplikacji ruchy są porównywane z zapisanymi wzorami. Rozwiązanie to daje większą swobodę w projektowaniu gestów - trudności z ich opisaniem na podstawie przemieszczeń stawów i ich wzajemnego położenia przestają być istotne. Porównywanie z wzorcem testowano w czasie przygotowywania pracy magisterskiej, jednak nie wykorzystano go w finalnej wersji programu. Analiza przyczyn takiego stanu rzeczy znajduje się w dalszej części pracy.

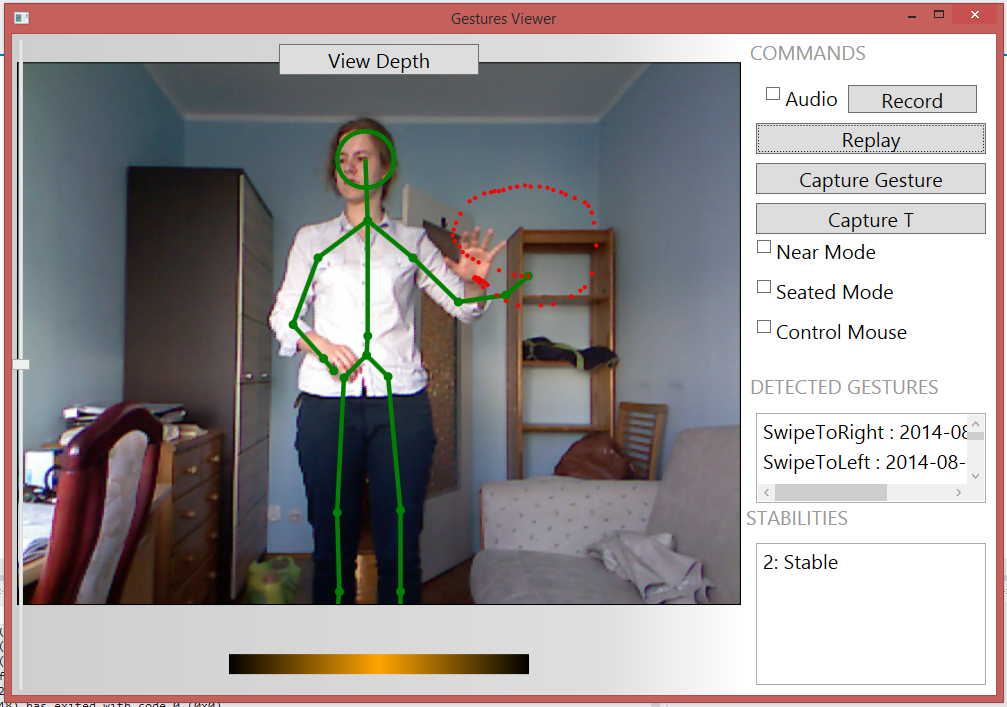
### Implementacja rozpoznawania gestów z użyciem wzorca

Do implementacji gestów bazowano na rozwiązaniu Kinect.Toolbox, które można pobrać ze strony http://go.microsoft.com/FWLink/?Linkid=258661. Ponadto, rozwiązanie to posiada obszerną dokumentację . Podstawą obsługi gestów są tutaj klasy *GestureDetector* oraz *TemplatedGestureDetector*. Poniższy schemat UML pokazuje relację między wymienionymi klasami:



Schemat 3.1 Schemat UML klas gestów biblioteki Kinect.Toolbox (opracowanie własne)

Abstrakcyjna klasa *GestureDetector* stanowi klasę bazową dla gestów rozpoznawanych poprzez porównywanie z szablonem (*TemplatedGestureDetector*) oraz dla konkretnych implementacji algorytmicznych gestów (dla przykładu *SwipeGestureDetector* modeluje gest przesuwania dłoni w prawo lub w lewo). *GestureDetector* obsługuje tylko gesty wykonywane jedną ręką, posiada możliwości wizualizacji wykonywanych ruchów oraz śledzenia szkieletu, co jest przydatne podczas tworzenia i debugowania oprogramowania. Ponadto, posiada definicję abstrakcyjnej metody *LookForGesture*, której implementacja zależy od przyjętej metody rozpoznawania gestów oraz od konkretnego rodzaju gestu, który ma być obsłużony w danej klasie potomnej po *GestureDetector*. Ponadto, obsługiwane jest wywołanie zdarzenia wykrycia gestu. W domyślnych ustawieniach programu minimalnym odstępem czasu między gestami jest zero milisekund, a więc dopuszczone jest płynne rozpoznawanie gestów następujących jeden po drugim. Warto dodać, że w momencie rozpoznania poszukiwanego gestu i wywołania zdarzenia, które informuje o tym fakcie inne obiekty, czyszczona jest lista dotychczasowych punktów składających się na gest. Jeżeli jednak w wyniku przetwarzania otrzymano wynik, iż dana trajektoria nie odpowiada trajektorii zdefiniowanego gestu, usuwany jest najstarszy punkt z listy i dodawane jest bieżące położenie stawu.



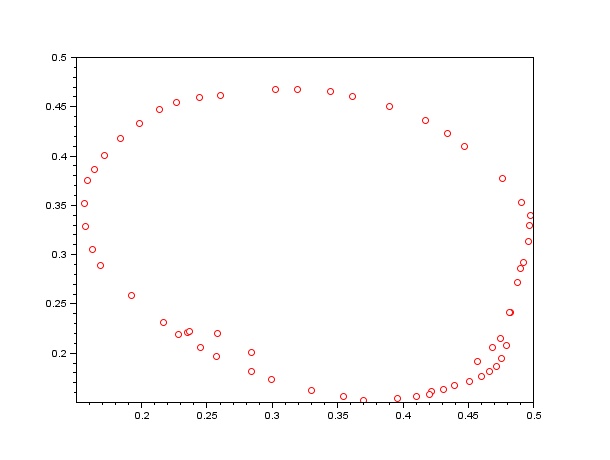
Rysunek 3.1 Śledzenie szkieletu i ruchów ręki w Kinect.Toolbox (źródło własne)

Rozpoznawanie gestów poprzez porównywanie z wzorcem odbywa się z użyciem algorytmu złotego podziału. Zanim jednak to nastąpi wymagane jest sprowadzenie wzorców i próbek porównywanych ruchów do jednakowej (porównywalnej) postaci. Podczas ruchu zapisywane są kolejne punkty (współrzędne) śledzonej dłoni (na Rysunku 2.4 jest to dłoń prawa) na liście punktów. Następnie, dla każdego punktu rysowana jest kropka na ekranie, a jej współrzędne służą do zamodelowania trajektorii odbytego ruchu. Odbywa się to według następującego algorytmu:



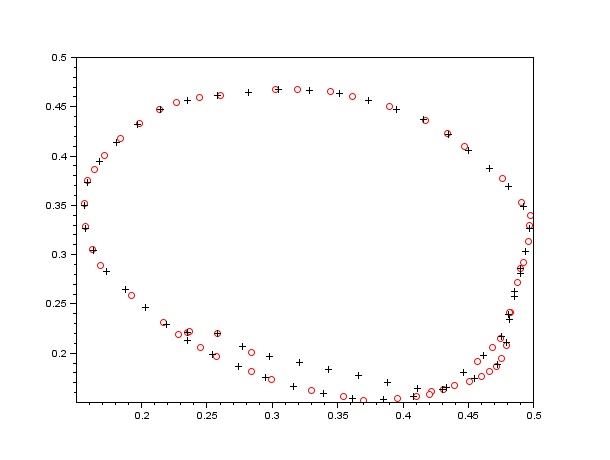
Schemat 3.2 Algorytm przygotowania wzorców (opracowanie własne)

Pierwszym etapem obróbki zamodelowanego ruchu jest dodawanie kolejnych próbek zawierających współrzędne śledzonego stawu do listy. Dopiero po zapełnieniu listy przez określoną przez programistę liczbę próbek - w tym wypadku 60 - rozpoczyna się obróbka ruchu w celu zweryfikowania czy wykonano badany gest. Poniższy rysunek ukazuje przykładową trajektorię:



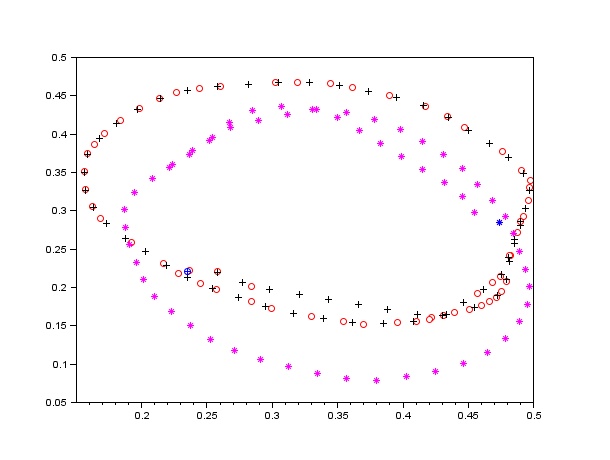
Schemat 3.3 Trajektoria ruchu ręki (źródło własne)

Posiadając początkowy wzorzec, należy przystąpić do jego normalizacji tzn. na całej jego długości rozmieścić punkty w równych odległościach, co pozwoli nadać mu płynny kształt i zniwelować duże przerwy między punktami. Znormalizowaną trajektorię ze schematu 4.3 przedstawiono poniżej. Czerwonymi kółkami oznaczone jest początkowe (rzeczywiste) rozmieszczenie punktów, natomiast czarnymi krzyżykami zaznaczono trajektorię znormalizowaną.



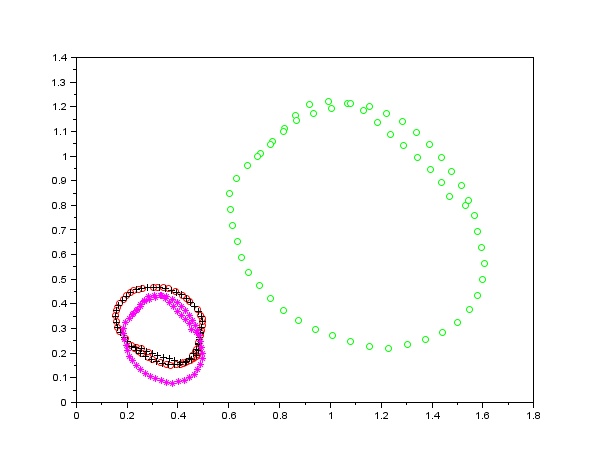
Schemat 3.4 Trajektoria znormalizowana (źródło własne)

Kolejnym etapem przygotowania próbek do porównania z wzorcem jest obrót. Obrót wykonywany jest względem środka ciężkości wyznaczonego dla trajektorii. Zakładając, że lokalny układ współrzędnych ma swój początek właśnie w punkcie środka ciężkości, obrót jest wykonywany w taki sposób, aby punkt początkowy trajektorii znajdował się na osi odciętych tego układu. Cała figura obracana jest wokół środka ciężkości o ten sam kąt. Na zamieszczonym schemacie próbki koloru niebieskiego oznaczają pierwsze próbki danej trajektorii - a zatem ich punkty początkowe.



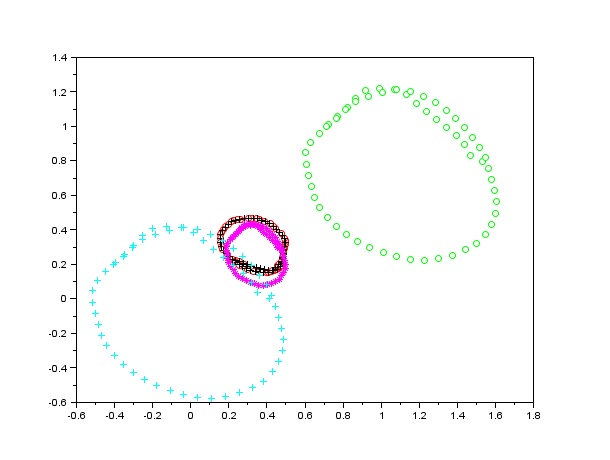
Schemat 3.5 Obrócona trajektoria (źródło własne)

Następnie przeprowadzane jest skalowanie, dzięki czemu każdorazowo figura mieści się w obrysie kwadratu o wymiarach 1 x 1.



Schemat 3.6 Przeskalowana trajektoria (źródło własne)

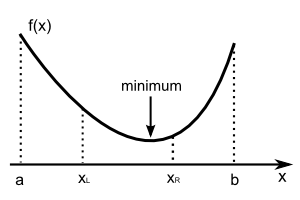
Ostatnim etapem wstępnej obróbki jest przesunięcie ustandaryzowanej trajektorii w taki sposób, aby jej środek ciężkości pokrył się ze środkiem układu współrzędnych.



Schemat 3.7 Wycentrowana trajektoria (źródło własne)

W ten sposób otrzymywana jest ustandaryzowana figura, która może być porównywana ze zdefiniowanym wzorcem. Podczas zapisywania bazy wzorców w programie, wykorzystywany jest ten sam algorytm. Dzięki temu, ruchy (a dokładniej ich matematyczne modele) sprowadzane są do ustandaryzowanej postaci, co umożliwia ich porównywanie.

Porównywanie aktualnej trajektorii ruchu z wzorcami zapisanymi w bazie odbywa się poprzez zastosowanie algorytmu złotego podziału. Metoda ta pozwala znaleźć lokalne minimum jednowymiarowej funkcji w przedziale <a, b>. Zakłada się funkcję ciągłą w danym przedziale oraz posiadającą dokładnie jedno minimum w tym przedziale - są to wymogi konieczne dla prawidłowego działania algorytmu. Dzieląc badany przedział na coraz to mniejsze fragmenty, znajdujemy poszukiwane minimum stosunkowo niewielkim kosztem obliczeniowym.



Schemat 3.8 Algorytm złotego podziału (27)

Podział badanego odcinka wprowadzany jest poprzez dobór nowych punktów xL oraz xR, w taki sposób, aby xL było bliżej lewej krawędzi przedziału, a xR bliżej prawej krawędzi. Sprawdzając wartość funkcji w tak obranych punktach xL i xR, weryfikujemy dla którego z nich wartość funkcji jest mniejsza według następującej prawidłowości:

wówczas stosuje się następujące wzory do przeliczeń:

*b = xR;*

*xR = xL;*

*xL = b - k \* ( b - a ).*

Natomiast w sytuacji odwrotnej:

,

wzory przyjmują postać:

*a = xL;*

*xL = xR;*

*xR = a + k \* ( b - a );*

gdzie

k = \frac{\sqrt{5} - 1}{2} \approx 0,61803398

czyli wartość złotego podziału.

Po zawężeniu przedziałów, należy dobrać nowe wartości parametrów xL oraz xR i ponownie wykonać procedurę. Przetwarzanie kończy się w momencie osiągnięcia warunku

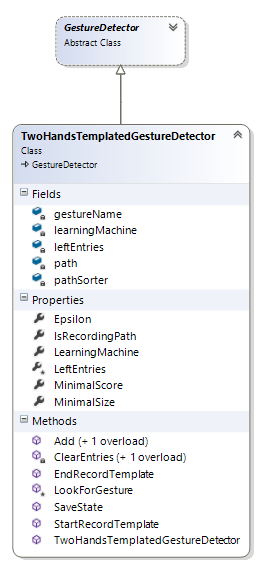
(b - a) \leqslant \epsilon

gdzie epsilon jest wartością zadaną przez użytkownika.

W przypadku porównywania wzorców gestów, przedziałem są wartości kątów, o które należy obrócić wzorzec wykonanego ruchu. Następuje porównanie próbek szablonów, jeśli nie ma wystarczającej spójności, wzorzec odwzorowujący ruch zostaje obrócony i porównywanie rozpoczyna się od nowa. Algorytm złotego podziału zwraca współczynnik pewności w zakresie 0-1. Do programisty należy dobór współczynnika w zależności od potrzeb. Uzyskanie wartości współczynnika powyżej ustalonego progu jest równoznaczne z wykryciem gestu.

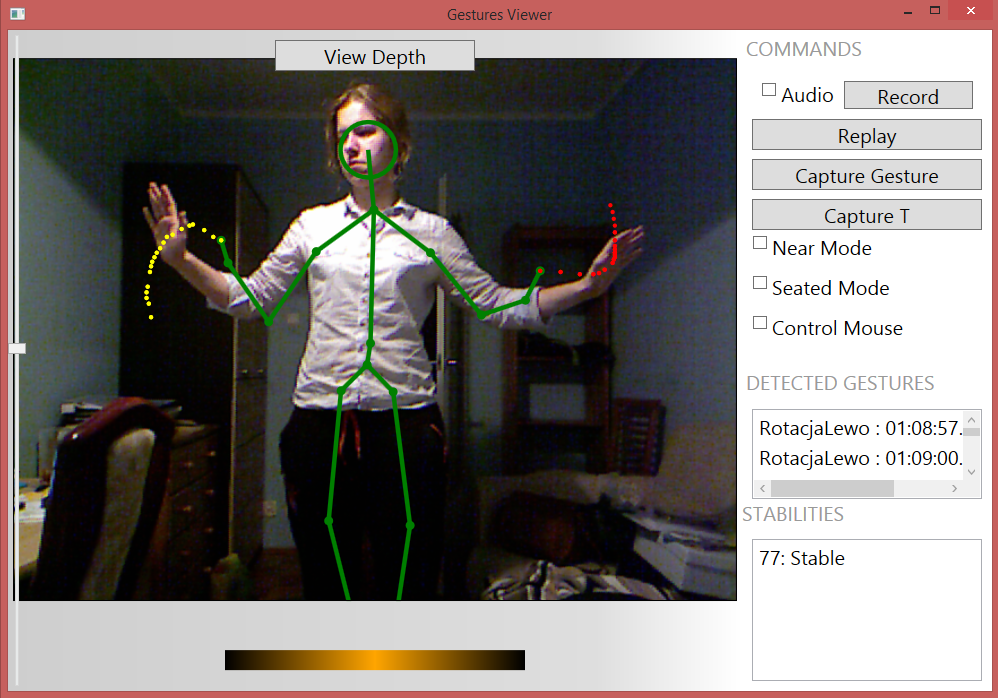
### Rozpoznawanie gestów dwuręcznych z użyciem wzorca

W ramach pracy magisterskiej należało zbudować aplikację komputerową, która obsługiwana będzie przy pomocy gestów. Zdecydowano się na implementację gestów obsługiwanych dwiema rękoma, co powinno zapewnić ergonomię i przestrzeganie zasad projektowania naturalnego interfejsu użytkownika. Aby omówione w poprzednim podrozdziale struktury programowe obsługiwały gesty wykonywane przy pomocy dwóch rąk, należało poszerzyć ich funkcjonalność. Dodane funkcjonalności prezentuje schemat UML:



Schemat 3.9 Dwuręczny gest rozpoznawany metodą porównywania z wzorcem (opracowanie własne)

Tym, co odróżnia gest dwuręczny od gestu jednoręcznego jest przede wszystkim możliwość śledzenia dwóch dłoni równocześnie. Podczas przetwarzania strumienia szkieletu w programie głównym, dodawane są kolejne próbki położeń stawów dłoni do obiektu klasy *TwoHandsTemplatedGestureDetector*. Ich śledzenie pokazane jest na poniższym obrazku. Żółte kropki pokazują trajektorię lewej dłoni, zaś kropki czerwone - prawej.



Rysunek 3.2 Śledzenie dwóch dłoni (źródło własne)

Pierwszą modyfikacją, w stosunku do gestów wykonywanych jedną ręką, jest wprowadzenie drugiej listy, na której zapisywane są współrzędne drugiej dłoni. Także sam proces zapisu punkt ów wymagał zmian, ponieważ w przypadku równoczesnego śledzenia dwóch rąk za każdym razem należy zweryfikować z której dłoni jest sczytywana pozycja oraz czy dłoń ta jest stawem śledzonym (ang. *joint tracked*). Oczywiście należało także uwzględnić metody odpowiedzialne za wizualizację ruchów wykonywanych przez obie ręce.

Kolejnym istotnym zagadnieniem było przystosowanie metod zapisywania bazy gestów. Ze względu na fakt, iż wewnątrz klasy *TwoHandsTemplatedGestureDetector* występuje dodatkowa lista punktów (*LeftEntries*), należało zadbać o scalanie list (składujących informacje o prawej i lewej dłoni) w miejscach, w których przetwarzano szablon jako pojedynczą listę gestów.

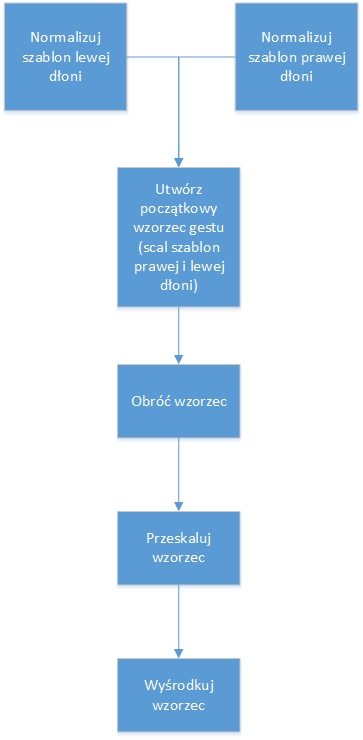
Ciekawym problemem, który pojawił się zarówno podczas nagrywania nowych gestów dwuręcznych dla zapełnienia bazy szablonów, jak i podczas prób rozpoznania tychże gestów podczas korzystania z aplikacji, było scalanie osobnych trajektorii dłoni prawej i lewej do jednej wspólnej ścieżki. Przedstawia to poniższy obraz:



Rysunek 3.3 Łączenie trajektorii dłoni w geście dwuręcznym

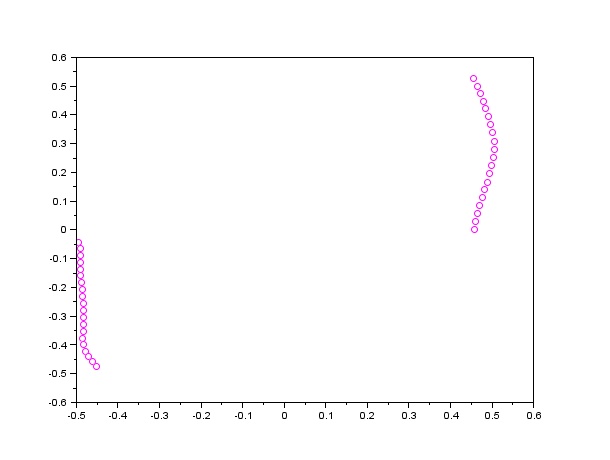
W przypadku gestów wykonywanych przy pomocy obu dłoni, stanem pożądanym jest powstanie dwóch trajektorii - dla prawej i dla lewej dłoni i modelowanie oraz zapisywanie szablonów z zachowaniem tego rozróżnienia (przynajmniej na pewnym etapie przetwarzania). Sztuczne łączenie punktów symbolizujących współrzędne dłoni uniemożliwia rozróżnienie gestów nieciągłych (gdy prawa i lewa dłoń "nie spotykają się", ale wykonują odrębne ruchy) od gestów ciągłych (dla których charakterystyczne jest uwspólnienie trajektorii obu dłoni).

Aby pozbyć się efektu "uciąglenia" gestu, należało zaingerować w algorytm przedstawiony na schemacie 4.2 w następujący sposób:



Schemat 3.10 Algorytm przygotowania wzorców gestu dwuręcznego (opracowanie własne)

Dzięki powyższej modyfikacji proces przygotowania wzorców przebiega następująco - najpierw zebrane próbki dla ręki prawej są normalizowane (czyli punkty rozmieszczane są w równych odstępach na całej długości trajektorii ruchu, bez względu na jej kształt), to samo dzieje się dla próbek zebranych dla lewej dłoni. Następnym krokiem jest scalenie tak znormalizowanych dwóch trajektorii do jednej wspólnej (w tym - nieciągłej jeśli taki ruch miał miejsce). Kolejne kroki algorytmu są takie same jak dla rozpoznawania gestu jednoręcznego, ale w przypadku gestu dwuręcznego zachodzą dla wspólnej listy pozycji obu dłoni. W efekcie, otrzymujemy następujący model trajektorii:



Schemat . Gest dwuręczny nieciągły

Taka reprezentacja ruchu (nieciągła) pozwala dużo lepiej interpretować i przetwarzać potencjalne gesty w celu ich wykrywania. Przy użyciu w ten sposób rozbudowanego narzędzia, zbudowano bazę siedemnastu szablonów dla gestu obracania. Ich matematyczna reprezentacja wygląda tak, jak przedstawiono to na

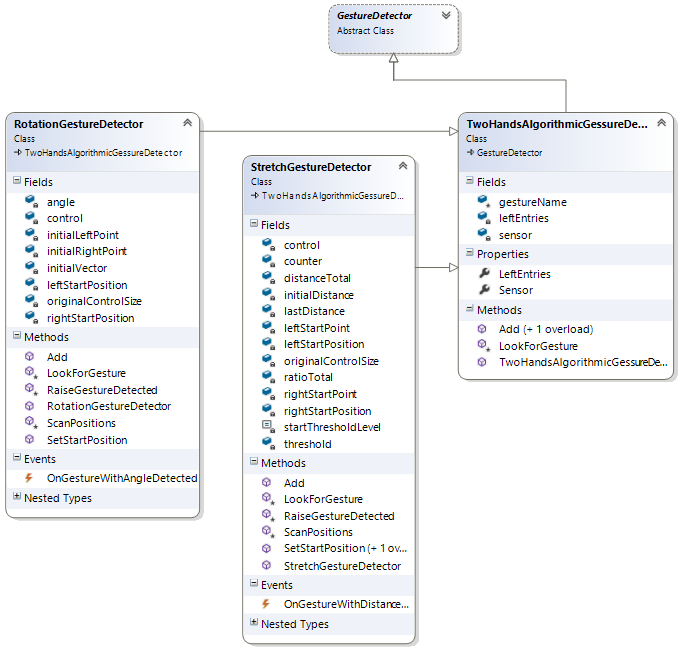


Rysunek 3.4 Szablony gestu rotacji (opracowanie własne)

Celność rozpoznań tak zamodelowanego gestu była niesatysfakcjonująca. Próbowano zwiększyć liczbę prawidłowych kategoryzacji gestów poprzez manipulowanie przy parametrach algorytmu i gestu (wartość epsilon czy próg współczynnika pewności w algorytmie złotego podziału). Jednak tym, co okazało się najbardziej dotkliwym było utracenie wszelkich informacji o orientacji dłoni, kącie wykonanego obrotu itp., a są to dane kluczowe do późniejszego przetwarzania interfejsu użytkownika. Rotacje, normalizacje, skalowanie - wszystkie te procesy konieczne do ustandaryzowania szablonów nadpisywały istotne informacje. Z tego powodu należało znaleźć inny sposób implementacji gestów - zdecydowano się na implementację algorytmiczną.

### Algorytmiczna implementacja rozpoznawania gestów

W implementacji algorytmicznej także posłużono się bazową klasą *GestureDetector* z biblioteki Kinect.Toolbox, ale dopisano jeszcze klasę *TwoHandsAlgorythmicDetector*, która jest bezpośrednim przodkiem dla klas poszczególnych obiektów. Zamodelowano dwa gesty - gest obracania (*RotationGestureDetection*) oraz przybliżania i oddalania obrazu poprzez rozciąganie (*StretchGestureDector*). Oba gesty są wykrywane tylko w przypadku zaciśniętych pięści użytkownika. Jako użytkownik sterujący aplikacją wybierana jest osoba stojąca najbliżej czujnika Kinect. Dobór osób odbywa się na zasadzie mierzenia odległości od sensora, czyli współrzędnej Z w układzie współrzędnych związanych z Kinectem. Strukturę wymienionych klas przedstawia schemat UML:



Schemat 3.12 9 Gest rozciągania i gest obracania w implementacji algorytmicznej (opracowanie własne)

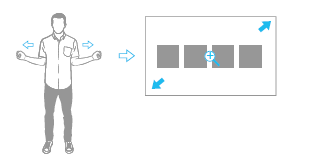
#### Gest rozciągania

Gest rozciągania używany jest w celu powiększenia oraz pomniejszenia oglądanego obrazu. Gdy użytkownik stoi z rozpostartymi ramionami i zaczyna zbliżać dłonie do siebie, następuje pomniejszanie obrazu - jakby użytkownik ściskał go. Z kolei gdy użytkownik rozpoczyna wykonywanie gestu mając dłonie w bliskim sąsiedztwie, a następnie oddala je od siebie następuje zjawisko "rozciągania" obrazu, a więc w efekcie - jego przybliżanie. Wykrywanie gestu trwa podczas gdy użytkownik ma zaciśnięte pięści.



Schemat 3.13 Gest rozciągania - obsługa (opracowanie własne)

Podczas inicjalizacji gestu ładowane są wszystkie domyślne wartości i dane podstawowe. Dodanie pierwszych próbek ruchu e geście rozciągania (współrzędnych pobranych ze współrzędnych szkieletowych stawu dłoni prawej i lewej) przypisuje startowe położenie dłoni. Następnie, podczas dodawania każdej kolejnej próbki z informacją o położeniu dłoni, następuje przeliczenie odległości między dłońmi na ekranie. Otrzymana wartość porównywana jest z minimalnym progiem zmiany odległości między rękoma - w przypadku, gdy próg zostanie osiągnięty wykonywane są kolejne polecenia mające na celu obliczenie czy wykonano gest pomniejszania czy powiększania.

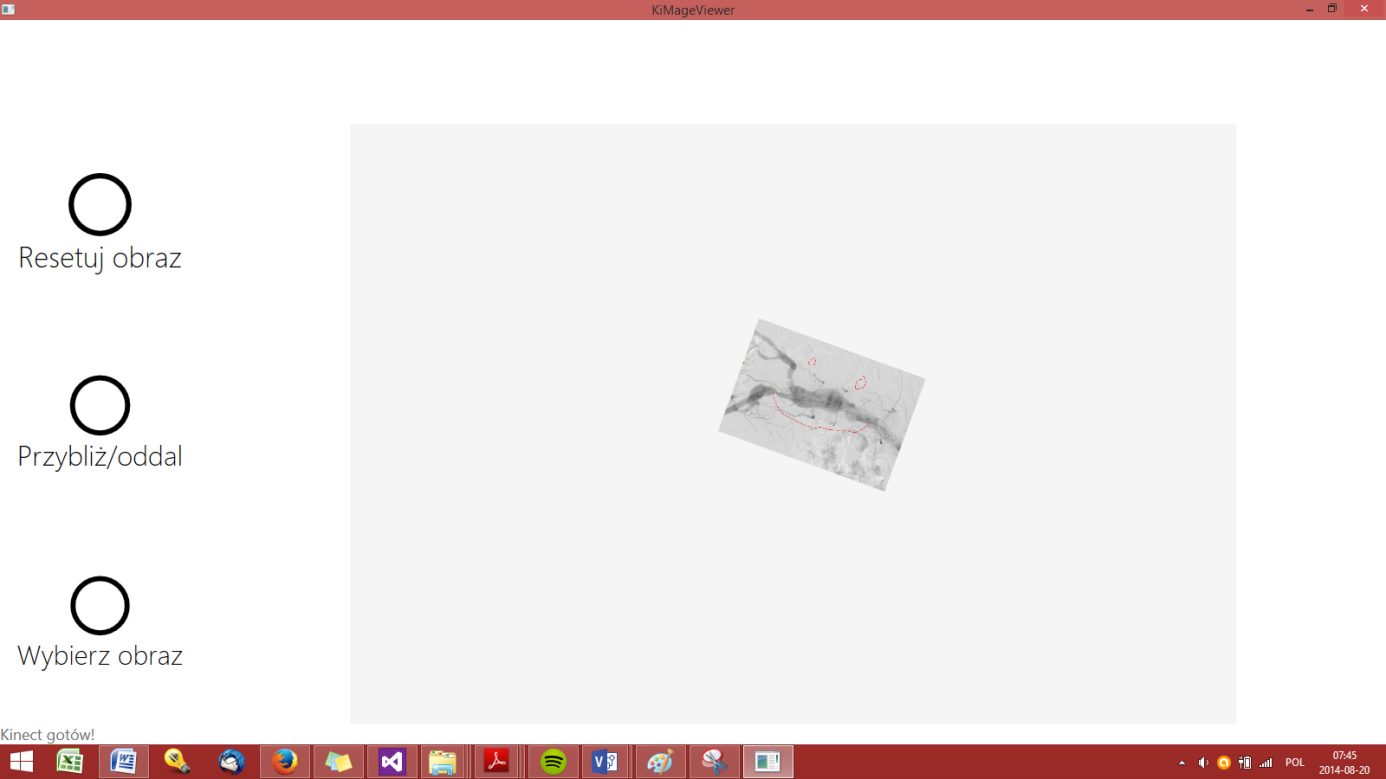


Rysunek 3.5 Gest rozciągania - powiększanie obrazu (8)



Rysunek 3.6 Gest rozciągania - pomniejszanie obrazu (8)

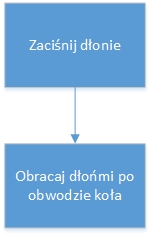
Następną procedurą w procesie algorytmicznego wykrywania gestu jest przeliczenie zmiany odległości między dłońmi w rzeczywistości (na podstawie danych ze strumienia szkieletu) do współrzędnych kontrolek na ekranie. Takie postępowanie zapewnia skalowalność aplikacji w zależności od wielkości ekranu i jego rozdzielczości. Jeżeli dystans między dłońmi rośnie w porównaniu z położeniem początkowym, to następuje wykrycie gestu powiększania obrazu. Jeżeli odległość między dłońmi maleje, wykryto komendę do pomniejszania obrazu. Każde rozpoznanie gestu sygnalizowane jest wywołaniem zdarzenia *OnGestureWithDistanceDetected*.



Rysunek 3.7 Obraz po wykonaniu gestu pomniejszania (źródło własne)

#### Gest obracania

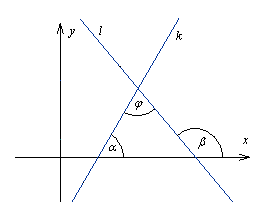
Gest obracania jest obsługiwany, podobnie jak gest rozciągania, tylko w czasie, kiedy użytkownik ma zaciśnięte pięści. Obsługa gestu polega na zaciśnięciu dłoni, a następnie przemieszczanie się nimi jakby po okręgu, którego środek jest w połowie drogi między rękoma.



Schemat 3.14 Gest obracania - obsługa (opracowanie własne)

Podczas inicjalizacji gestu rotacji, analogicznie jak w przypadku gestu rozciągania, następuje ustawienie domyślnych parametrów. W momencie zaciśnięcia obu pięści, gdy jest to wykryte przez sensor Kinect, zapisywane są początkowe współrzędne położenia rąk. Z każdą kolejną próbką obliczany jest kąt między linią poziomą, a linią wykreśloną przez punkty aktualnego położenia dłoni.

W zapisie matematycznym, mamy do dyspozycji 2 proste k - prostą obliczaną na podstawie bieżących próbek oraz l - prostą początkową. Można opisać je następującymi równaniami:

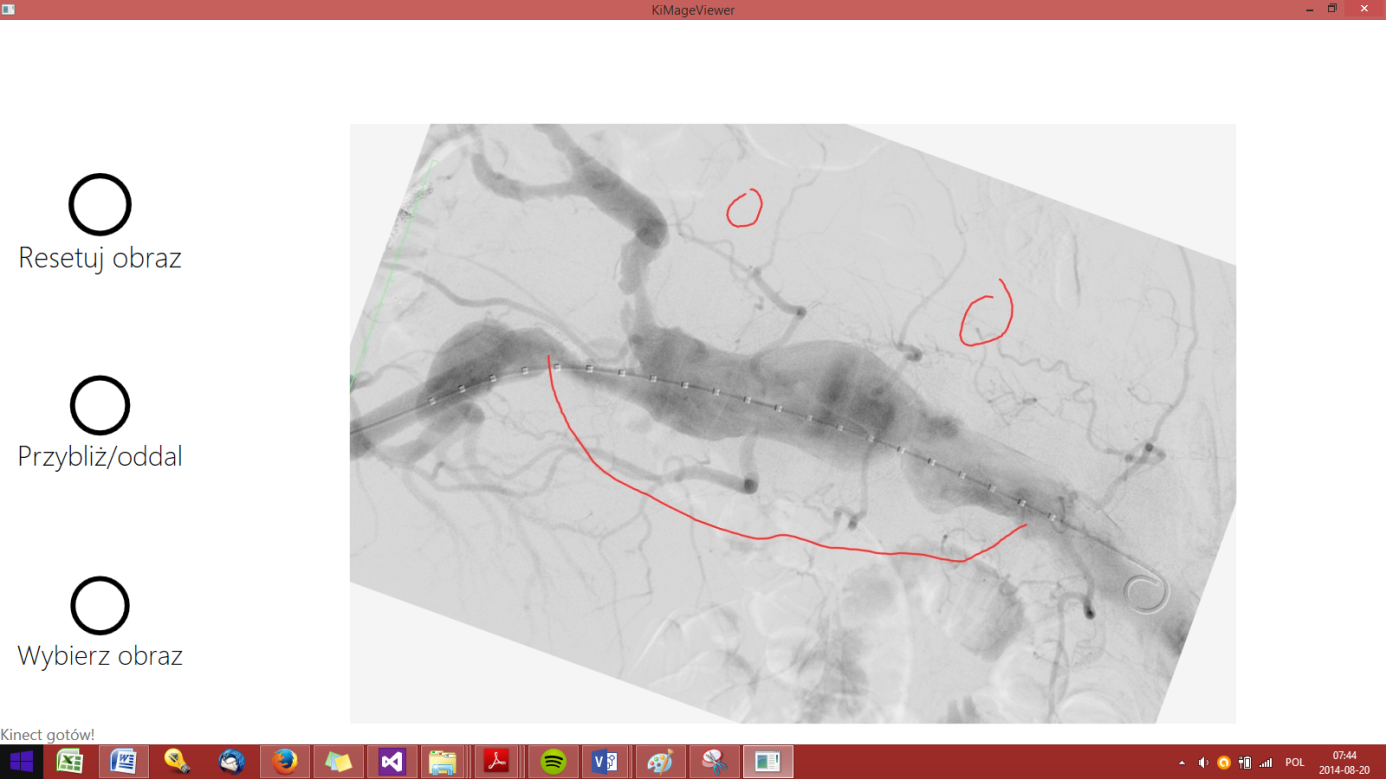


Rysunek 3.8 Kąty między dwiema prostymi (29)

Z prostych przekształceń matematycznych wiadomo, że

gdzie kąt α jest kątem między osią odciętych a prostą k (wyznaczoną na podstawie bieżących próbek). Z powyższego równania przechodzimy do wyznaczenia kąta α

gdzie



Rysunek 3.9 Obraz po wykonaniu gestu rotacji (źródło własne)

### Podsumowanie implementacji gestów

Przetestowano dwie alternatywne metody implementacji gestów. Metoda porównywania z wzorcem wymaga więcej nakładu pracy, a w miarę rozrastania się bazy gestów wzorcowych wykazuje coraz większe zapotrzebowanie na zasoby sprzętowe (każdy wykonany ruch jest porównywany z każdym wzorcem z bazy). Z drugiej strony, metoda ta daje większą dowolność w projektowaniu gestów - zupełnie likwiduje problem matematycznego modelowania gestu. Niestety, w ramach przygotowań niniejszej pracy magisterskiej zrezygnowano z wykorzystania tej metody rozpoznawania gestów.

W związku z niewykorzystaniem zaimplementowanej metody porównywania gestów z wzorcem, postanowiono opisać proste gesty używane w aplikacji przy pomocy narzędzi matematycznych. Jako że stworzenie takich modeli okazało się łatwe, a wyniki rozpoznawania tak zaprojektowanych i opisanych gestów były zadowalające, pozostano przy algorytmicznej metodzie rozpoznawania gestów.

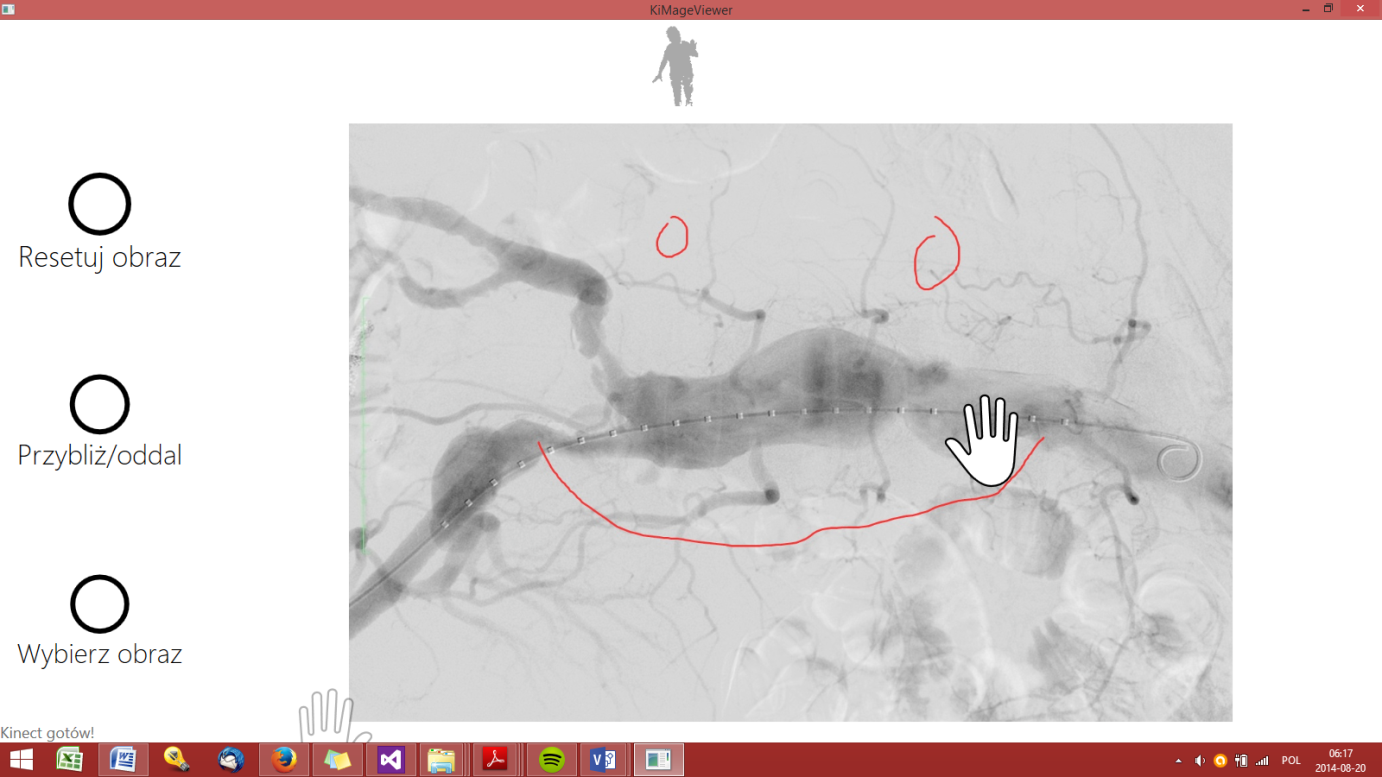
## Interfejs programu

W interfejsie użytkownika stworzonej aplikacji wykorzystano zestaw kontrolek obsługujących strumień interakcji oraz inne kontrolki pozwalające na zrealizowanie założeń projektowych. Do najważniejszych z nich należą:

* KinectRegion,
* KinectCircleButton,
* KinectTileButton,
* KinectScrollViewer,
* KinectUserViewer,
* ZoomBorder,
* UcImageSelection.

Pierwsze cztery wymienione elementy pochodzą z bibliotek Kinect for Windows SDK, ZoomBorder stanowi znalezione przez autorkę pracy rozwiązanie do powiększania i przesuwania obrazów przy użyciu myszy, które zaadoptowano do obsługi za pomocą Kinect'a. Modyfikacje kontrolek wprowadzone w ramach niniejszej pracy opisane są w kolejnych podrozdziałach.

UcImageSelection jest kontrolką powstałą na bazie okna głównego z przykładowego projektu ControlsBasics-WPF dostępnego w ramach Kinect for Windows SDK. UcImageSelection zapewnia funkcjonalność wyboru obrazu do aktualnej obróbki.



Rysunek 3.10 Interfejs aplikacji (źródło własne)

### KinectRegion

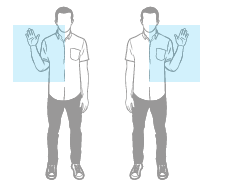
KinectRegion jest kontrolką, która dzięki danym pochodzącym ze strumienia interakcji zapewnia podstawową kontrolę nad kursorem oraz innymi potomnymi jej kontrolkami. Widoczne na rysunku 3.8 dłonie są odzwierciedleniem dłoni użytkownika. W ten sposób zapewniona jest informacja zwrotna, co umożliwia zorientowanie użytkownika w tym, jak aplikacja reaguje na jego ruchy i gesty. Domyślnie KinectRegion obsługuje tylko jedną dłoń - tę, która w danym momencie jest dłonią *główną*. Jako główną rękę przyjmuje się tę, która była uniesiona jako pierwsza przez użytkownika aplikacji . Wybór ten odbywa się podczas obsługi strumienia interakcji. Podobnie jak w przypadku obsługi gestów, tak i tutaj zaimplementowano interakcję tylko z najbliższym użytkownikiem aplikacji.

Włączenie dwóch kursorów dłoni w KinectRegion wymagało nadpisania jednej linijki kodu w pliku KinectRegion.cs (w projekcie Microsoft.Kinect.Toolkit.Controls dostarczonym wraz z SDK):

//IsPrimaryHandOfUser = handPointer.IsPrimaryForUser,

IsPrimaryHandOfUser = true,

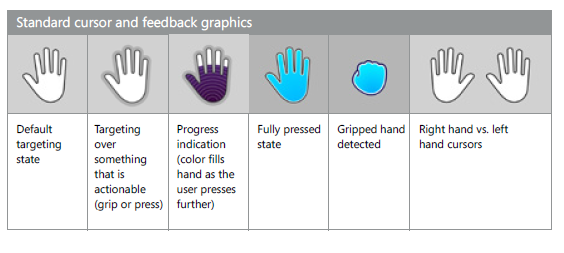
Niestety, implementacja samego strumienia interakcji jest niedostępna (Microsoft zapewnia dostęp jedynie do skompilowanej biblioteki .dll), przez co trudno pozbyć się pewnych niechcianych efektów. Takim efektem jest krzyżowanie się wskaźników dłoni na ekranie mimo tego, że użytkownik swoich rąk w rzeczywistości nie krzyżuje. Wynika to z faktu, iż każda dłoń ma osobną strefę fizycznej interakcji (PhIZ) i każda z tych stref rzutowana jest na całą kontrolkę KinectRegion.



Rysunek 3.11 Strefa fizycznej interakcji (PhIZ) (8)

W związku z mapowaniem PhIZ na KinectRegion, po włączeniu dwóch kursorów dla dwóch dłoni, zachodzi następujące zjawisko - gdy prawa dłoń jest w lewej części swojej PhIZ (np. na wysokości pępka), a lewa dłoń jest w prawej części swojej PhIZ (np. również w okolicach pępka), to na ekranie zobaczymy lewą dłoń w okolicach prawej krawędzi KinectRegion, a prawą dłoń w okolicach lewej krawędzi KinectRegion. Jest to zachowanie bardzo nieintuicyjne, jednak ze względu na fakt, że algorytmy wyznaczające strefę fizycznej interakcji są niedostępne, trudno rozwiązać ten problem. Próbowano zrezygnować z korzystania z danych pochodzących ze strumienia interakcji i pozycjonować kursory tylko na podstawie strumienia szkieletu, jednakże jest to rozwiązanie bardzo chwiejne. Korzystając tylko i wyłącznie ze strumienia danych szkieletu, bardzo łatwo jest samemu użytkownikowi zaniżyć wiarygodność danych. Otóż, wystarczy schować jedną dłoń za drugą, aby absolutnie stracić pewność co do położenia dłoni, co może skutkować niekontrolowanymi skokami kursorów na ekranie. Niewiarygodność danych pochodzących tylko i wyłącznie ze strumienia szkieletu przesądziła o zaniechaniu takiego alternatywnego rozwiązania. W obecnej chwili, pozostawiono korzystanie z KinectRegion bez interwencji w strumień interakcji. W związku z tym, wskazane jest obsługiwanie aplikacji w szerszym rozstawie rąk.

Dzięki możliwości podpinania dowolnych kontrolek - niekoniecznie dedykowanych do interakcji z sygnałami pochodzącymi z Kinect'a - pod KinectRegion, także z ich poziomu (o ile są zakotwiczone w KinectRegion) można uzyskać dostęp do kursorów oraz ich stanów. Kursor reprezentując dłoń oddaje swoim wyglądem także zachowanie śledzonej dłoni oraz jej niektóre parametry. Dostępne są następujące stany kursora: domyślny, nad kontrolką sterowaną kursorem Kinect, postęp nacisku, wciśnięcie, uchwyt. Wizualizacja prawej i lewej ręki są różne.



Rysunek 3.12 Stany kursora KinectRegion (8)

Za pomocą publicznych metod statycznych KinectRegion, w łatwy sposób uzyskano dostęp do stanów kursora z poziomu kontrolki ZoomBorder. Obsługa jej za przy użyciu urządzenia Kinect opisana jest w kolejnym podrozdziale.

### ZoomBorder

ZoomBorder jest kontrolką, która dziedziczy po klasie Border. Obiekty klasy Border służą do obrysowania i/lub rysowania tła wokół innych kontrolek. Ciekawą implementację ZoomBorder przystosowaną do przesuwania oraz przybliżania obrazów znaleziono na forum internetowym stackoverflow.com . Pierwotnie kontrolka pozawalała na przybliżanie, oddalanie oraz przesuwanie obrazu przy pomocy myszki. Dodano funkcjonalność obrotu używając myszki oraz umożliwiono wykonanie wszystkich wymienionych transformacji za pomocą gestów omówionych we wcześniejszych częściach pracy.

Obsługa kursorów KinectRegion wywoływana jest za pomocą następujących zdarzeń:

* QueryInteractionStatus - wywoływane gdy zmieni się status interakcji kursora (dostępne statusy to true - kursor w stanie interakcji i false - kursor nie jest interaktywny),
* HandPointerGrip - oznacza zaciśnięcie dłoni,
* HandPointerMove - wywoływane podczas poruszania się kursora po KinectRegion,
* HandPointerGripRelease - zwolnienie uścisku dłoni,
* HandPointerLeave - opuszczenie przez kursor danej kontrolki lub KinectRegion (definiowane przez programistę).

Transformacje przesunięcia, obrotu oraz skalowania wykonywane są przy użyciu komponentu TransformGroup. Jest to zestaw przekształceń, które są aplikowane do przetwarzanego obrazu. Dostępnymi w jej ramach transformacjami są: rotacja, skalowanie, pochylenie, translacja . Bardzo istotna jest kolejność dołączania danych transformacji do obiektu TransformGroup, ponieważ waży ona na kolejność ich wykonywania. Co ciekawe, pierwsza z dołączonych transformacji zostanie przeprowadzona jako ostatnia. Ma to zasadniczy wpływ na implementację zachowań transformowanego obiektu, ponieważ np. wykonanie najpierw rotacji, a potem translacji przyniesie inny efekt niż wykonanie najpierw translacji, a potem rotacji (mimo podania tych samych wartości argumentów). Wynika to z zasad przekształceń geometrycznych, z którymi wiąże się zmiana położenia, orientacji lokalnego układu współrzędnych.

#### Przesunięcie

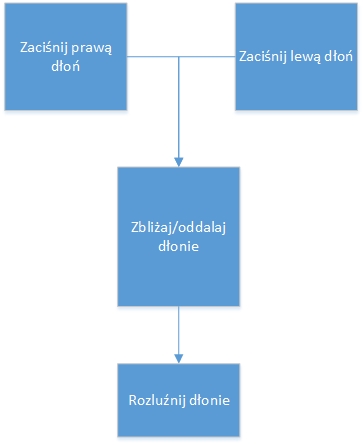
Możliwość przesuwania obrazu na ekranie zaimplementowano według poniższego algorytmu. Obraz przesuwa się tak samo jak kursor dłoni na ekranie. Gest jest obsługiwany, jeśli podczas przesuwania kursora jest ona zaciśnięta i została zaciśnięta nad obrazem - wówczas obraz jest jakby chwycony w dłoń.



Schemat 3.15 Algorytm przesuwania obrazu (opracowanie własne)

#### Pomniejszanie/Powiększanie (Zoom)

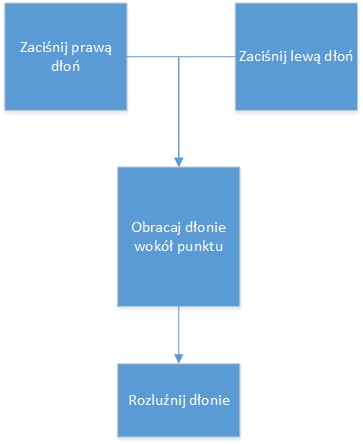
Operacja pomniejszania i powiększania obrazu przebiega zgodnie z algorytmem zamieszczonym na schemacie 3.16. Przetwarzanie jest tutaj dość złożone. Kinect Region wyzwala zdarzenie kursora (zaciśnięcie dłoni) - wówczas odbywa się sprawdzenie czy obie dłonie są zaciśnięte. Jeśli nie, to nic sygnał przekazywany jest do sprawdzenia przez inne gesty. Jeśli tak, to wywołane jest zdarzenie sygnalizujące, że rozpoczęto wykonywanie gestu rozciągania - w tym momencie rozpoczyna się inicjalizacja gestu rozciągania, przypisanie wartości początkowych i rozpoznawanie gestów z każdą nadchodzącą klatką strumienia szkieletu. Rozpoznawanie gestu odbywa się w klasie za to odpowiedzialnej (omówionej w poprzednich sekcjach pracy). W przypadku wykrycia gestu rozszerzania wywołane zostaje zdarzenie, które informuje główny program o tym fakcie. Jednym z argumentów tego zdarzenia jest wartość, o jaką należy pomniejszyć bądź powiększyć obraz, co zostaje wykonane z poziomu kontrolki ZoomBorder.



Schemat 3.16 Algorytm gestu pomniejszania i powiększania (opracowanie własne)

#### Rotacja

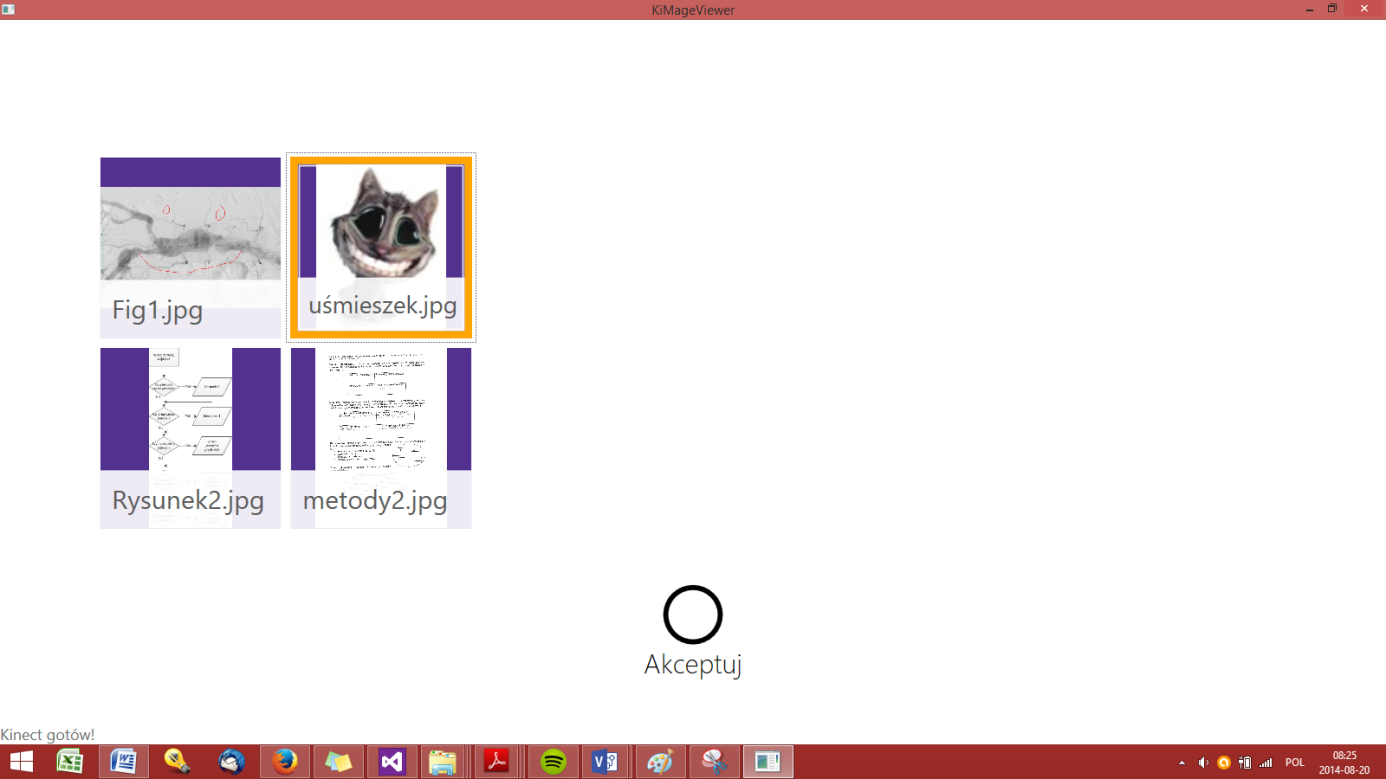
Obracanie obrazu jest procedurą wykonywaną w powiązaniu z rozpoznawaniem gestu rotacji. Algorytm postępowania zobrazowano na schemacie. Zasada współdziałania kontrolki ZoomBorder z obiektem klasy rozpoznawania gestów oraz z oknem głównym jako trzonem aplikacji jest taka sama jak w przypadku operacji przybliżania i oddalania.



Schemat 3.17 Alogrytm operacji obracania (opracowanie własne)

### UcImageSelection

Konrolka UcImageSelection jest widokiem aplikacji, w którym można zmienić obraz, który poddawany jest aktualnie obróbce. Dostępne są obrazy, które zostały dodane przy uruchamianiu programu. Zapewniona jest obsługa zarówno gestami, jak i myszką. Pomarańczową ramką zaznaczony jest wybrany obraz. Kliknięcie przycisku *Akceptuj* powoduje przejście do okna głównego, gdzie zostaje wyświetlony wybrany obraz.



Rysunek 3.13 UcImageSelection (opracowanie własne)

# Podsumowanie

W ramach pracy magisterskiej należało stworzyć aplikację komputerową, która umożliwi dokonywanie prostych przekształceń obrazów (oddalanie, zbliżanie, obracanie) przy użyciu technologii bezdotykowej. Wskazane było rozwiązanie z wykorzystaniem urządzenia Microsoft Kinect. Dużo czasu poświęcono na zapoznanie się z platformą Microsoft Kinect, co później zaowocowało skutecznymi rozwiązaniami natury programistycznej.

Udało się zrealizować wszystkie z wymagań projektowych określonych we wstępie pracy. Obróbka obrazów zawierających dane medyczne została zapewniona. W trakcie realizacji pracy przeanalizowano, zaimplementowano, przetestowano i porównano różne metody rozwiązań niektórych problemów (dobór metody rozpoznawania gestów, próba przypisania nowych współrzędnych kursorom kontrolki KinectRegion). Wynikało to z potrzeby zapewnienia jak najlepszej jakości obsługi systemu. Bardzo ważnym czynnikiem, na który mocno zwracano uwagę podczas tworzenia pracy, była ergonomia ruchów i intuicyjność sterowania.

Pracując nad aplikacją komputerową okazało się, że wersja urządzenia Microsoft Kinect, którym dysponowano nie jest wersją dedykowaną do współpracy z komputerem z systemem operacyjnym Windows i do wykorzystywania za pomocą Kinect for Windows SDK. Co więcej, z postanowień licencyjnych wynika, iż czujnik Microsoft Kinect for Xbox 360 może być używany podczas prac programistycznych, niemniej niedozwolone jest korzystanie z niego we współpracy z tak stworzoną aplikacją finalną. Jest to niestety problem, którego autorka pracy nie jest w stanie rozwiązać.

Istnieją perspektywy rozwoju aplikacji powstałej w ramach niniejszej pracy magisterskiej. Poza dopracowaniem rozwiązań związanych z nawigacją przy użyciu dwóch rąk i kursorów kontrolki KinectRegion, ciekawym i przede wszystkim funkcjonalnym rozwiązaniem byłoby powiązanie stworzonego programu z bazą danych, w której umieszczano by dokumentację medyczną pacjentów oraz istotne dane, które mogłyby być przydatne podczas przeprowadzania procedur medycznych (operacji i zabiegów).

W tym miejscu pragnę wyrazić moje podziękowania dla współpracujących lekarzy, z którymi odbyto konsultacje medyczne. Bez ich pomocy niemożliwe byłoby dostosowanie aplikacji to realiów panujących w szpitalach.

# Spis ilustracji

## Schematy

[Schemat 1.1 Sala operacyjna - widok z góry (opracowanie własne) 5](#_Toc396286825)

[Schemat 2.1 Interakcja sprzętu i oprogramowania z aplikacją (6) 7](#_Toc396286826)

[Schemat 2.2 Podzespoły urządzenia Kinect for Windows - opracowanie własne na podstawie (7) 8](#_Toc396286827)

[Schemat 2.3 Zakres wykrywania dźwięków (8) 9](#_Toc396286828)

[Schemat 2.4 Zasięg kątowy pracy czujnika głębokości i RGB (8) 9](#_Toc396286829)

[Schemat 2.5 Architektura SDK (6) 11](#_Toc396286830)

[Schemat 2.6 Układ współrzędnych dla strumienia głębokości (19) 17](#_Toc396286831)

[Schemat 2.7 Układ współrzędnych strumienia szkieletu (19) 20](#_Toc396286832)

[Schemat 2.8 Fizyczna strefa interakcji (ang. *Physical Interaction Zone*) (8) 21](#_Toc396286833)

[Schemat 3.1 Schemat UML klas gestów biblioteki Kinect.Toolbox (opracowanie własne) 27](#_Toc396286834)

[Schemat 3.2 Algorytm przygotowania wzorców (opracowanie własne) 29](#_Toc396286835)

[Schemat 3.3 Trajektoria ruchu ręki (źródło własne) 30](#_Toc396286836)

[Schemat 3.4 Trajektoria znormalizowana (źródło własne) 30](#_Toc396286837)

[Schemat 3.5 Obrócona trajektoria (źródło własne) 31](#_Toc396286838)

[Schemat 3.6 Przeskalowana trajektoria (źródło własne) 32](#_Toc396286839)

[Schemat 3.7 Wycentrowana trajektoria (źródło własne) 32](#_Toc396286840)

[Schemat 3.8 Algorytm złotego podziału (27) 33](#_Toc396286841)

[Schemat 3.9 Dwuręczny gest rozpoznawany metodą porównywania z wzorcem (opracowanie własne) 35](#_Toc396286842)

[Schemat 3.10 Algorytm przygotowania wzorców gestu dwuręcznego (opracowanie własne) 38](#_Toc396286843)

[Schemat 3.11 Gest dwuręczny nieciągły 39](#_Toc396286844)

[Schemat 3.12 9 Gest rozciągania i gest obracania w implementacji algorytmicznej (opracowanie własne) 41](#_Toc396286845)

[Schemat 3.13 Gest rozciągania - obsługa (opracowanie własne) 42](#_Toc396286846)

[Schemat 3.14 Gest obracania - obsługa (opracowanie własne) 44](#_Toc396286847)

[Schemat 3.15 Algorytm przesuwania obrazu (opracowanie własne) 51](#_Toc396286848)

[Schemat 3.16 Algorytm gestu pomniejszania i powiększania (opracowanie własne) 52](#_Toc396286849)

[Schemat 3.17 Alogrytm operacji obracania (opracowanie własne) 53](#_Toc396286850)

## Rysunki

[Rysunek 2.1 Obraz kolorowy (źródło własne) 15](#_Toc396286851)

[Rysunek 2.2 Obraz podczerwony (źródło własne) 16](#_Toc396286852)

[Rysunek 2.3 Obraz głębokości (źródło własne) 17](#_Toc396286853)

[Rysunek 2.4 Układ stawów - Skeleton Stream (20) 18](#_Toc396286854)

[Rysunek 2.5 Tryb siedzący (8) 18](#_Toc396286855)

[Rysunek 2.6 Śledzony szkielet (źródło własne) 20](#_Toc396286856)

[Rysunek 2.7 Kinect Studio (źródło własne) 23](#_Toc396286857)

[Rysunek 3.1 Śledzenie szkieletu i ruchów ręki w Kinect.Toolbox (źródło własne) 28](#_Toc396286858)

[Rysunek 3.2 Śledzenie dwóch dłoni (źródło własne) 36](#_Toc396286859)

[Rysunek 3.3 Łączenie trajektorii dłoni w geście dwuręcznym 37](#_Toc396286860)

[Rysunek 3.4 Szablony gestu rotacji (opracowanie własne) 40](#_Toc396286861)

[Rysunek 3.5 Gest rozciągania - powiększanie obrazu (8) 42](#_Toc396286862)

[Rysunek 3.6 Gest rozciągania - pomniejszanie obrazu (8) 43](#_Toc396286863)

[Rysunek 3.7 Obraz po wykonaniu gestu pomniejszania (źródło własne) 43](#_Toc396286864)

[Rysunek 3.8 Kąty między dwiema prostymi (29) 45](#_Toc396286865)

[Rysunek 3.9 Obraz po wykonaniu gestu rotacji (źródło własne) 46](#_Toc396286866)

[Rysunek 3.10 Interfejs aplikacji (źródło własne) 48](#_Toc396286867)

[Rysunek 3.11 Strefa fizycznej interakcji (PhIZ) (8) 49](#_Toc396286868)

[Rysunek 3.12 Stany kursora KinectRegion (8) 50](#_Toc396286869)

[Rysunek 3.13 UcImageSelection (opracowanie własne) 54](#_Toc396286870)

# Bibliografia

1. **The Kinect for Windows Team.** Jintronix makes rehabilitation more convenient, fun, and affordable with Kinect for Windows. [Online] 2014. [Zacytowano: 17 sierpień 2014.] http://blogs.msdn.com/b/kinectforwindows/archive/2014/01/22/jintronix-makes-rehabilitation-more-convenient-fun-and-affordable-with-kinect-for-windows.aspx.

2. **Stango, Nicholas.** Kinect Can Help Doctors Monitor Your Physical Rehab From Home. *Gizmodo.* [Online] 2012. [Zacytowano: 17 sierpień 2014.] http://gizmodo.com/5946737/kinect-can-doctors-monitor-your-physical-rehab-from-home.

3. **Corish Bob, Criminisi Antonio , Gonzalez Gerardo , O'Hara Kenton, Sellen Abigail.** Touchless Interaction in Medical Imaging. *Microsoft Research .* [Online] 2014. [Zacytowano: 17 sieprpień 2014.] http://research.microsoft.com/en-us/projects/touchlessinteractionmedical/.

4. **Microsoft Corporation.** Kinect for Windows - Meet Kinect. *Microsoft.* [Online] Microsoft, 2014. [Zacytowano: 17 sierpień 2014.] www.microsoft.com/en-us/kinectforwindows/meetkinect/default.aspx.

5. —. Accelerometer. *Developer Network.* [Online] 2014. [Zacytowano: 17 sierpień 2014.] http://msdn.microsoft.com/en-us/library/jj663790.aspx.

6. —. Kinect for Windows Architecture. [Online] 2014. [Zacytowano: 17 sierpień 2014.] http://msdn.microsoft.com/en-us/library/jj131023.aspx.

7. —. Kinect for Windows Sensor Components and Specifications. *Developer Network.* [Online] Microsoft, 2014. [Zacytowano: 17 sierpień 2014.] http://msdn.microsoft.com/en-us/library/jj131033.aspx.

8. —. *Human Interface Guidelines v1.8.* 2013. str. 39.

9. —. Setting Up a Kinect Sensor. *Developer Network.* [Online] Microsoft, 2014. [Zacytowano: 17 sierpień 2014.] http://msdn.microsoft.com/en-us/library/hh855356.aspx.

10. **Catuhe, David.** *Programming with the Kinect for Windows Software Development Kit.* Redmond, Washington : Microsoft Press, 2012. strony 4, 75-127.

11. **Microsoft Corporation.** System Requirements. *Developer Network.* [Online] 2014. [Zacytowano: 17 sierpień 2014.] http://msdn.microsoft.com/en-us/library/hh855359.aspx.

12. **Microsft Corporation.** Microsoft Kinect for Windows Software Development Kit (SDK) End User License Agreement.

13. **Microsoft Corporation.** Running a Kinect-enabled Application on a Developer Machine. *Developer Network.* [Online] 2014. [Zacytowano: 17 sierpień 2014.] http://msdn.microsoft.com/en-us/library/hh855358.aspx.

14. Natural user interface - Wikipedia. [Online] 2014. [Zacytowano: 17 sierpień 2014.] http://en.wikipedia.org/wiki/Natural\_user\_interface.

15. **James Ashley, Jarett Webb.** *Beginning Kinect Programming with the Microsoft Kinect SDK.* New York : Apress, 2012. strony 39-40, 168-176.

16. **Microsoft Corporation.** Getting the Next Frame of Data by Polling or Using Events. *Developer Network.* [Online] 2014. [Zacytowano: 17 sierpień 2014.] http://msdn.microsoft.com/en-us/library/hh973076.aspx.

17. —. Color Stream. *Developer Network.* [Online] 2014. [Zacytowano: 17 sierpień 2014.] http://msdn.microsoft.com/en-us/library/jj131027.aspx.

18. —. Depth Stream. *Developer Network.* [Online] 2014. [Zacytowano: 17 sierpień 2014.] http://msdn.microsoft.com/en-us/library/jj131028.aspx.

19. **Microsoft Kinect.** Coordinate Spaces. *Developer Network.* [Online] 2014. [Zacytowano: 17 sierpień 2014.] http://msdn.microsoft.com/en-us/library/hh973078.aspx.

20. **Raiten, Shai.** Kinect – Getting Started – Become The Incredible Hulk. [Online] 2014. [Zacytowano: 17 sierpień 2014.] http://blogs.microsoft.co.il/blogs/shair/image\_48A9C26C.png.

21. **Azimi, Mehran.** *Skeletal Joint Smoothing White Paper.* 2014.

22. **Microsoft Corporation.** Joint Filtering. *Developer Network.* [Online] 2014. [Zacytowano: 17 sierpień 2014.] http://msdn.microsoft.com/en-us/library/jj131024.aspx.

23. —. KinectInteraction Architecture. *Developer Network.* [Online] 2014. [Zacytowano: 17 sierpień 2014.] http://msdn.microsoft.com/en-us/library/dn188672.aspx.

24. **Stefanowski, Jerzy.** *Sztuczne sieci neuronowe.* [http://www.cs.put.poznan.pl/jstefanowski/aed/TPDANN.pdf] Poznań : brak nazwiska, 2006.

25. **Urbanek, Piotr.** *Sztuczne Sieci Neuronowe.* [ftp://zly.iis.p.lodz.pl/pub/people/P.Urbanek/PSI\_2013/SZTUCZNE%20SIECI%20NEURONOWE-wyklad\_2007.pdf] 2013.

26. **Macukow, Bohdan i Grzenda, Maciej.** *Sieci neuronowe w różnych dziedzinach nauki.* [http://www.mini.pw.edu.pl/MiNIwyklady/sieci/neuronowe.html] Warszawa : Wydział Matematyki i Nauk Informacyjnych Politechniki Warszawskiej , 2000.

27. http://pl.wikipedia.org/wiki/Metoda\_z%C5%82otego\_podzia%C5%82u. *Wikipedia.pl.* [Online] [Zacytowano: 17 sierpień 2014.]

28. Metoda złotego podziału. *Wikipedia.* [Online] [Zacytowano: 17 sieprpień 2014.] http://pl.wikipedia.org/wiki/Metoda\_z%C5%82otego\_podzia%C5%82u.

29. **Microsoft Corporation.** DirectX Media Objects. *Windows.* [Online] 2014. [Zacytowano: 17 sierpień 2014.] http://msdn.microsoft.com/en-us/library/windows/desktop/dd375474%28v=vs.85%29.aspx.

1. DMO - komponent strumieniowania danych oparty o technologię COM (tłumaczenie własne) [↑](#footnote-ref-1)