POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: AiR
SPECJALNOŚĆ: ARR

PROJEKT INŻYNIERSKI

Wykrywanie wybranych gestów z wykorzystaniem wizji trójwymiarowej

Selected gestures detection with 3D vision

AUTOR:
Mateusz Skiba

PROWADZĄCY PROJEKT:
dr inż. Marek Wnuk

OCENA PROJEKTU:

WROCŁAW 2013

Spis treści

Rozdział 1

Wstęp

Rozwojowi techniki nieodłącznie towarzyszy rozwój interfejsu użytkownika. Nie tylko ułatwia to komunikację pomiędzy człowiekiem a maszyną, lecz również zwiększa wydajność i komfort pracy. Na przestrzeni lat sposób takiej interakcji ewoluował od plików wsadowych przez linię poleceń, aż do najchętniej dziś używanych interfejsów graficznych. Coraz liczniejsze projekty dotyczące przetwarzania obrazów, coraz bardziej popularne gry konsolowe oparte na ruchu, jak również sceny przedstawiane w filmach gatunku science-fiction sugerują, że w tej dziedzinie kolejnym krokiem będzie ruchowy interfejs użytkownika *(**K**inetic **U**ser **I**nterface)*, gdzie sekwencje ruchu użytkownika, nagrane przez urządzenie rejestrujące obraz, będą interpretowane jako polecenia.

Interfejs tego rodzaj jest wysoce pożądany w ujęciu komunikacji człowiek — robot społeczny. Jednym z podstawowych bodźców odbieranych przez maszyny tej klasy jest wizja. „Oczy" robota pozwalają mu wykrywać użytkowników, rozpoznawać emocje zaistniałe na jego twarzy. Te informacje niezbędne są do odegrania zachowania robota zgodnie z zaimplementowanym scenariuszem. Jednak to możliwość identyfikowania gestów wykonywanych przez użytkownika stanowi podstawę dwukierunkowej interakcji, którą można uznać za naturalną.

Niniejszy projekt zawiera opis detektora gestów statycznych oraz dynamicznych gestów cyklicznych w oparciu o wizję trójwymiarową.

1.1 Cel projektu

Celem projektu jest stworzenie programu pozwalającego na wykrywanie wybranych gestów opisanych w dodatku A oraz ocena skuteczności i przydatności takiego oprogramowania w ujęciu robotyki społecznej.

1.2 Wymagania projektowe

Proces wykrywania gestów można podzielić na trzy etapy:

1. Akwizycję danych z urządzenia rejestrującego.
2. Zidentyfikowanie ludzkiej sylwetki oraz zinterpretowanie wybranych cech tejże sylwetki.
3. Wykrycie wystąpienia gestów na podstawie otrzymanych danych.

Z racji, że w projekcie szczególny nacisk zostanie położony na trzeci z etapów, wymaga się dobrania sensora, który w jak największym stopniu przy jak największej prostocie jest w stanie wykonać samodzielnie dwa pierwsze etapy detekcji gestu. Dodatkowo od samego czujnika wymaga się:

- Standardowego interfejsu komunikacyjnego dostępnego w komputerach oraz układach mikroprocesorowych z dużą mocą obliczeniową.
- Prędkości rejestracji obrazu pozwalającej wychwytywać płynne ruchy.
- Trójwymiarowej reprezentacji rejestrowanego obrazu.

Od części odpowiedzialnej za wykrywanie gestów, w celu łatwego oraz intuicyjnego sposobu ich definiowania, wymaga się:

- Niedużej wielkości wektora cech.
- Prostego modelu interpretowania cech.
- Możliwości definiowania gestów w sposób zewnętrzny, niezależny od modelu wykrywania gestów.

Rozdział 2

Wykorzystane narzędzia

Po uwzględnieniu wymagań projektowych wybrano poniżej przedstawiony zestaw narzędzi sprzętowych i programowych.

2.1 Sensor Kinect

Jako urządzenie rejestrujące obraz wybrano sensor Kinect[3] firmy Microsoft. Czujnik ten, początkowo dedykowany do gier ruchowych na konsolę Xbox 360, dzięki swym możliwością szybko stał się wykorzystywany w różnych projektach używających wizji trójwymiarowej[14][15][16][17].
Sensor składa się z następujących czujników:

- Kamery RGB o rozdzielczości 640x480 pikseli i 8-bitowym poziomie wartości.
- Czujnika głębi o rozdzielczości 640x480 pikseli i 8-bitowym poziomie wartości.
- Matrycy mikrofonów o częstotliwości próbkowania 16 kHz i 16-bitowym poziomie wartości.

Mapa głębi powstaje za pomocą projekcji podczerwonej oraz monochromatycznej matrycy CMOS. Zastosowana technologia pozwala zupełnie zniwelować wpływ oświetlenia sceny na wynik odczytu. Sprzętowa implementacja algorytmów filtrujących i rekonstruujących obraz trójwymiarowy sprawia, że sensor nie obciąża jednostki obliczeniowej ponad stopień zaangażowania wynikający z odczytu danych wyjściowych czujnika.

Główne cechy, który przyczyniły się do wyboru Kinecta jako uzywanego sensora to:

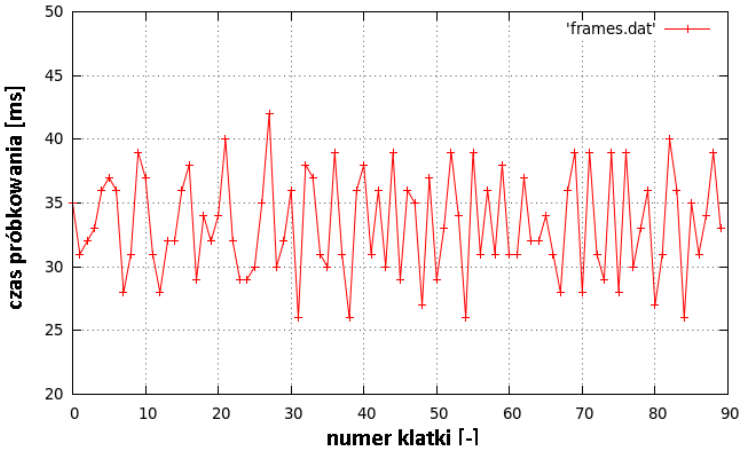
Rysunek 2.1 Wygląd sensora

- Relatywnie niewielka cena (około 500 zł) i dostępność na rynku.
- Komunikacja za pomocą interfejsu USB.
- Wystarczająca częstotliwość próbkowania (30 klatek na sekundę).
- Wykorzystywanie czujnika w licznych projektach.
- Dostępność bibliotek przetwarzających zarejestrowany obraz.

W celu weryfikacji danych przedstawionych przez producenta dokonano pomiarów częstotliwości próbkowania obrazu oraz zdolności płynnej rejestracji obrazu.

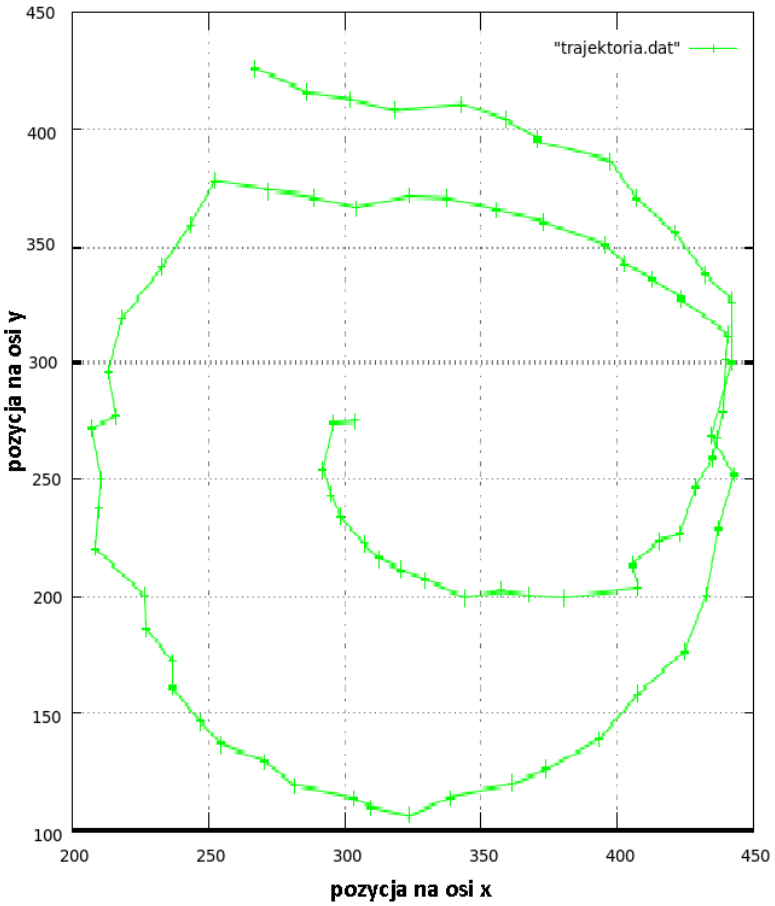
W pierwszym pomiarze zmierzono czas odświeżania danych wyjściowych Kinecta w pętli programowej dla kolejnych 90 klatek obrazu. Na poniższym wykresie przedstawiono wyniki.

Średni wynik pomiaru czasu próbkowania wyniósł $33,32ms$, co odpowiada poda-



Rysunek 2.2 Pomiar czasu próbkowania

wanej przez producenta częstotliwości 30 klatek na sekundę. Poszczególne klatki mogą być nieznacznie odchylone od wartości średniej, lecz w takim stopniu, że nie powinno to wprowadzać dysproporcji w zarejestrowanej sekwencji ruchu.
Dla potwierdzenia tej tezy postanowiono przedstawić na obrazie ścieżkę zakreślaną przez rękę wykonującą ruch spiralny w płaszczyźnie XY. Tę ścieżkę przedstawiono na rysunku 2.3. Jak można wywnioskować z wykresu ścieżka wykreślana przez dłoń jest rozpoznawalna, co potwierdza wyżej przedstawioną tezę.



Rysunek 2.3 Ścieżka zakreślona przez poruszającą się dłoń

2.2 Środowisko programowe

Projekt tworzony był na komputerze PC z wykorzystaniem systemu operacyjneg Linux w 32-bitowej dystrybucji Ubuntu 12.04. Programowano w języku C++. Poniżej opisano komponenty, które wymagały zainstalowania:

2.2.1 Oprogramowanie Kinecta

Sensor

Biblioteka Sensor autorstwa firmy PrimeSense (projektanta czujnika Kinect) przetworzona przez użytkownika *avin2*[4] na licencji wolnego oprogramowania GNU LGPL. Biblioteka służy do komunikacji komputera z Kinectem oraz pobierania danych z urządzenia zgodnych z formatem wejściowym biblioteki OpenNI[5]. Dostępna w środowiskach Windows, Linux, MacOS.

OpenNI

Biblioteka OpenNI(*Open Natural Interaction*)[5] powstała w celu standaryzowania kompatybilności czujników oraz wykorzystywania ich możliwości w kierunku rozwoju naturalnych interakcji. Aktualnie urządzeniami kompatybilnymi z tą biblioteką są sensor Kinect oraz Xtion[6] (bliźniaczy brat Kinecta wyprodukowany przez firmę Asus).
Biblioteka zapewnia:

- Detekcję użytkowników oraz podstawowych gestów dłoni.
- Śledzenie ruchów sylwetki.
- Detekcje głosu oraz poleceń głosowych.
- Warstwę śródprogramową odpowiedzialną za konfigurację czujnika, pobieranie danych oraz udostępnianie struktur przechowujących informacje o scenie, dane użytkowników oraz śledzone gesty dłoni.

Bibliotekę obejmuje licencja GNU LGPL, dostępna jest ona dla platform Windows, Linux, MacOS.

NITE

Biblioteka NITE[7] to zbiór algorytmów stworzonych przez firmę PrimeSense na rzecz organizacji OpenNI. Biblioteka ta jest nakładką na OpenNI zapewniającą następujące możliwości:

- Segmentacja sceny.
- Identyfikowanie użytkowników i wyszczególnianie części ciała.
- Śledzenie dłoni.
- Sterowanie aplikacją za pomocą śledzonej dłoni.

Biblioteka jest dostępna na licencji GNU LGPL na platformy Windows, Linux, MacOS.

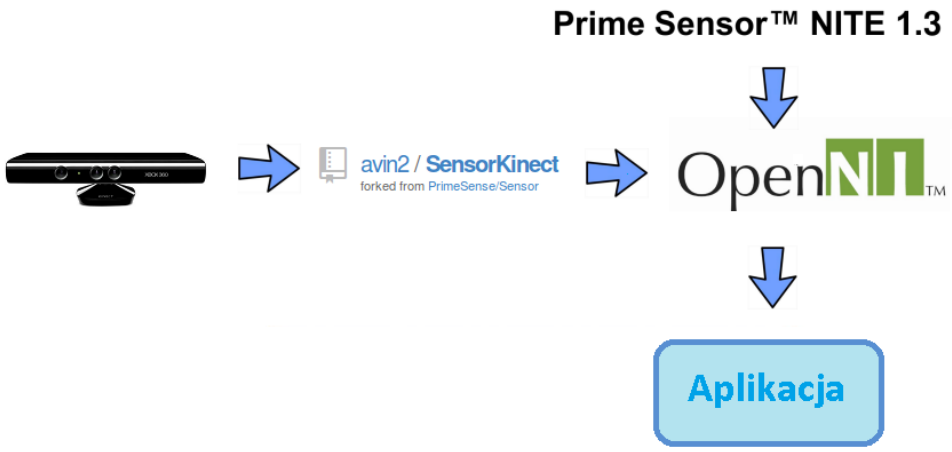
Powyższe biblioteki zostały w projekcie wykorzystane w celu wykrywania użytkowników, śledzenia ich oraz określania pozycji oraz orientacji zadanej w postaci macierzy RPY dla następujących części ciała:

- głowy (bez orientacji),
- środka linii barkowej,
- barków,
- łokci,
- dłoni (bez orientacji),
- środka torsu,
- bioder,
- kolan,
- stóp.

Powyższe pakiety oraz opis sposobu ich instalacji znajdują się na stronie OpenNI.

2.2.2 OpenGL

Biblioteka OpenGL[8] służy do generowania grafiki. W projekcie wykorzystywana jest do wizualizacji sceny oraz użytkowników. Kod odpowiedzialny za obsługę jest biblioteki został skopiowany z dostarczonych przez OpenNI przykładów. OpenGL dostępne jest na platformy Windows, Linux, MacOS. Biblioteka ta dostępna jest w standardzie wraz z wykorzystywaną dystrybucją systemu operacyjnego.



Rysunek 2.4 Diagram przepływu danych wykorzystywanych w aplikacji

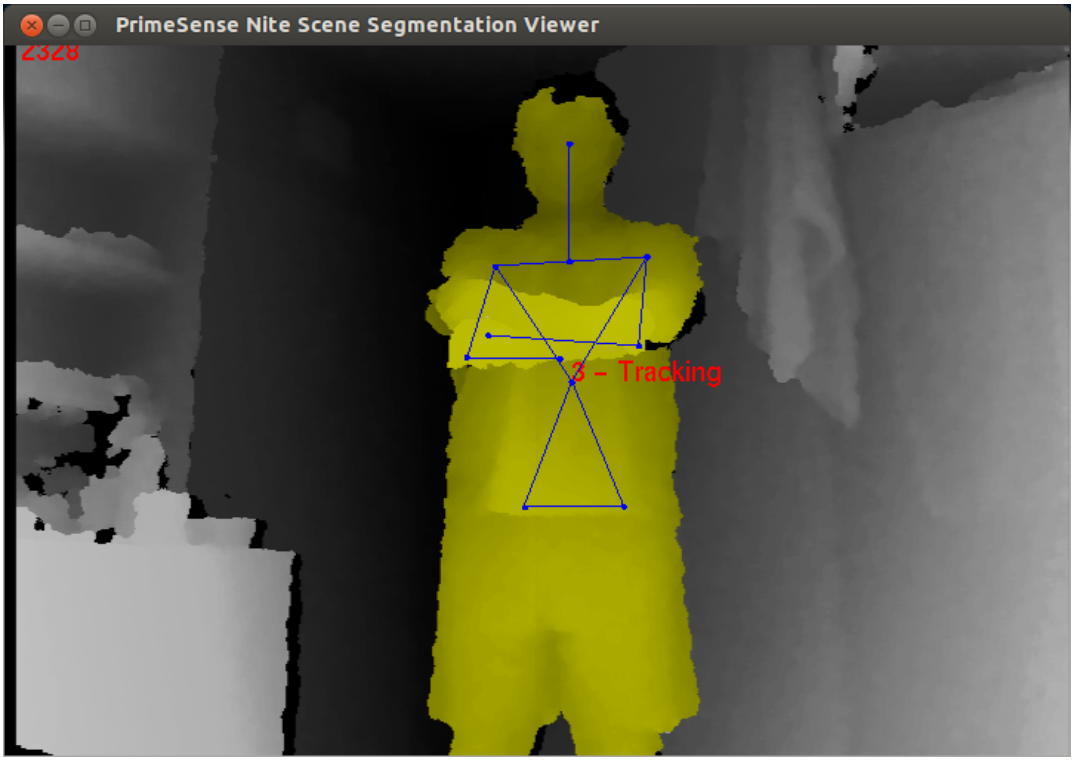
2.2.3 Xerces-C++

Biblioteka Xerces[9] służy do parsowania plików XML. Wykorzystywana jest do wczytywania zewnętrznie zdefiniowanych gestów. Dzięki temu każda zmiana dokonana w zestawie danych niezbędnych do rozpoznawania gestów nie wymaga ponownego kompilowania programu. Biblioteka nie tworzy drzewa *DOM* (ang. *Document Object Model*) dla znaczników, lecz wczytuje i analizuje kolejno każdą linijkę dokumentu. Zwiększa to prędkość wczytywania, szczególnie dla większych struktur danych. Biblioteka dostępna jest na licencji Apache Software License w wersji 2.0 na platformy Windows, Linux, MacOS. Sposób instalacji opisany jest na stronie projektu. Dla wykorzystywanej dystrybucji systemu operacyjnego biblioteka ta jest również dostępna w standardowym repozytorium.

2.2.4 Urbi

Urbi[12] jest platformą przeznaczoną do projektowania systemów złożonych. Wykorzystuje do tego źródła C++ lub Javy skompilowane do formy *UObject*[13]. One, w połączeniu z językiem skryptowym, pozwalają na wysokopoziomowe tworzenie aplikacji. Urbi dostarcza następujące mechanizmy:

- zrównoleglenie operacji,
- sterowanie zdarzeniowe,
- pracę w trybie klient—serwer,



Rysunek 2.5 Wizualizacja sceny i użytkownika z wyszczególnionymi górnymi częściami szkieletu

Urbi dostępne jest na licencji BSD na platformy Windows, Linux, MacOS. Sposób instalacji opisany jest na stronie projektu.

Rozdział 3

Implementacja

3.1 Lokalny układ współrzędnych

Podstawowymi danymi służącymi do detekcji wykonywanych gestów są współrzędne wykrywanych przez bibliotekę NITE części szkieletu użytkownika. Uzyskuje się pozycje XY oraz odległość Z. Początek układu współrzędnych zlokalizowany jest w centrum sensora. Dla użytkownika stojącego na wprost urządzenia oś X skierowana jest w prawą stronę, oś Y pionowo w górę, a oś Z skierowana jest od czujnika. Taka reprezentacja jest niewygodna. Sprawia ona, że skuteczność wykrywania gestów zależna jest od orientacji sylwetki w stosunku do Kinecta.

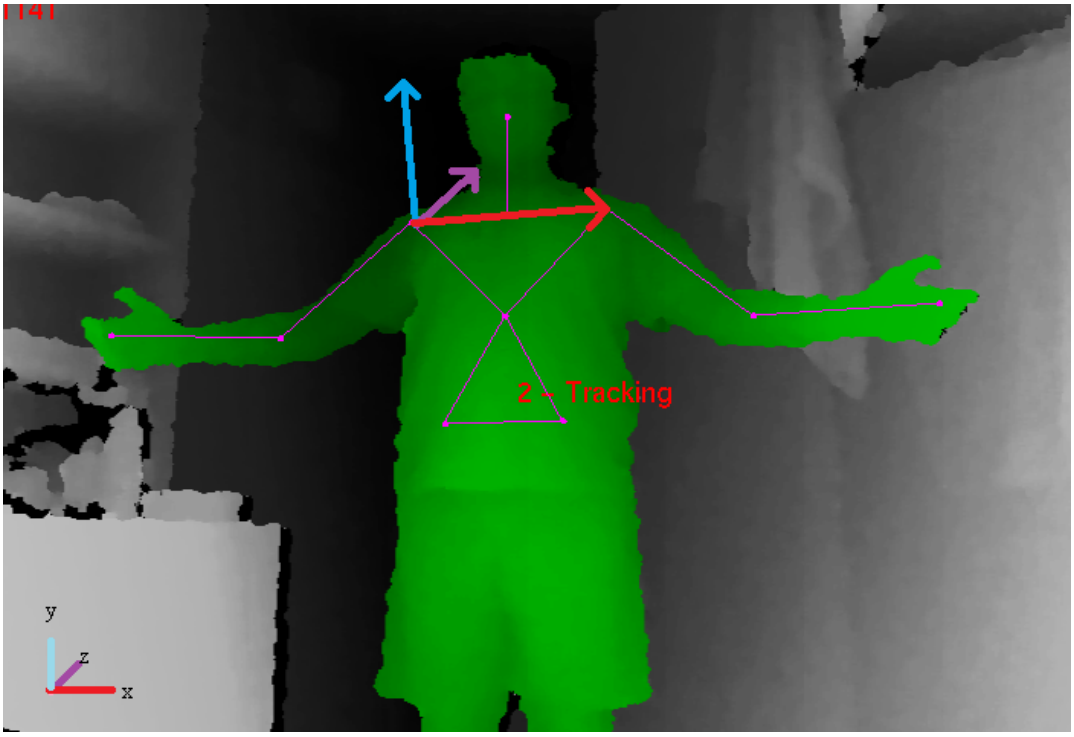
W celu uniezależnienia detekcji gestów od orientacji użytkownika na scenie wybrano następujący układ współrzędnych dla każdego użytkownika:

- Oś X skierowana jest od lewego do prawego barku.
- Oś Z jest wyznaczona przez iloczyn wektorowy linii łączącej lewy bark z torsem oraz linii barkowej (normalna do płaszczyzny torsu).
- Oś Y jest iloczynem wektorowym osi Z i X.

Dla każdej klatki obrazu obliczana jest macierz rotacji, a następnie wszystkie punkty obracane są do nowego układu współrzędnych. Dzięki takiej transformacji nie ma znaczenia jak użytkownik zlokalizowany jest na scenie, póki algorytm wykrywania szkieletu jest w stanie zidentyfikować elementy górnej sylwetki.

3.2 Reprezentacja sylwetki użytkownika

Detekcja gestów oparta jest na relacjach zachodzących między wektorami zdefiniowanymi w zewnętrznym pliku XML. Cała sekcja definiująca wektory mieści się



Rysunek 3.1 Nowy układ współrzędnych

między znacznikami<BodySet></BodySet>. Następnie określa się zbiory wektorów zawarte między elementami <Set Name=’’ ’’></Set>, gdzie atrybut Name określa unikalną nazwę zestawu, pozwalającą go identyfikować. Takie rozdzielenie zbiorów pozwala na definiowanie oddzielnych zestawów dla danych gestów w celu zwiększenia przejrzystości dokumentu. Zbiory zawierają wektory zdefiniowane w sposób następujący:

```
<Vector Name="">
  <Point Type="" />
  <Point Type="" />
</Vector>
```

Parametr Name zawiera identyfikator wektora unikalny w kontekście całego zbioru. Wektor składa się z dwóch elementów typu Point będącymi punktami początku i końca tego wektora. Atrubyty Type definiują rodzaj punktu. Ze względu na rodzaj składnia elementu Point przyjmuje następującą postać:

- body

  <Point Type="body" Joint="LEFT_ELBOW"/>,

gdzie atrybut `Joint` zawiera nazwę dostępnej za pomocą biblioteki NITE pozycji części ciała.

- `null`

  ```
  <Point Type="null" />,
  ```

  oznaczającej punkt o współrzędnych (0,0,0).

- `defined`

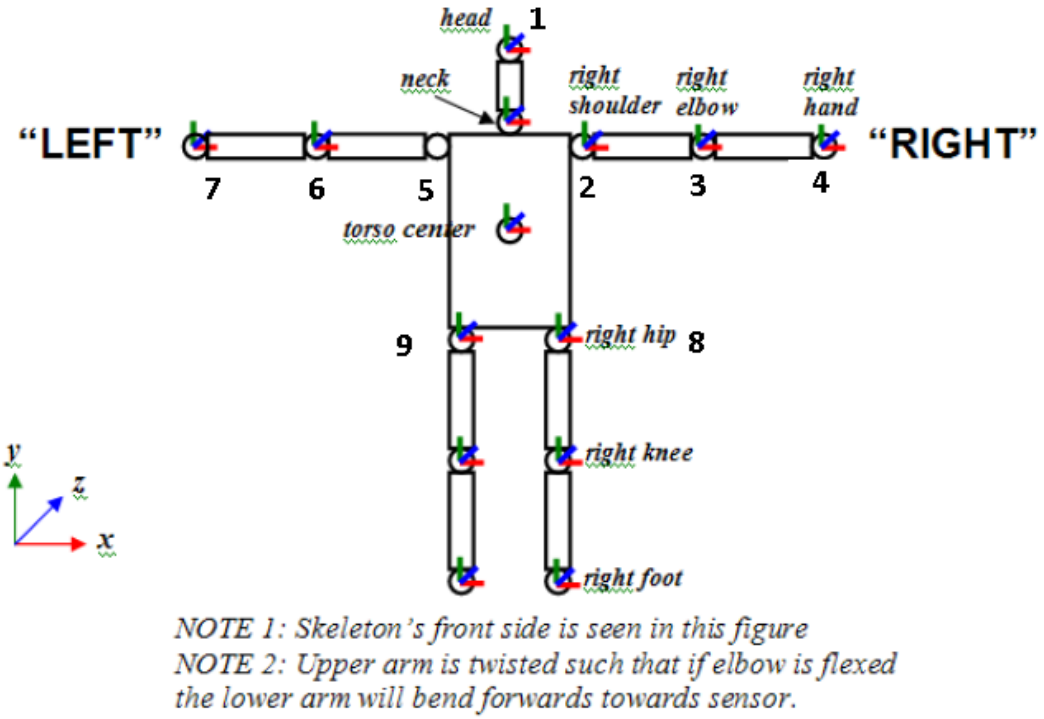
  ```
  <Point Type="defined" x=" " y=" " z=" "/>,
  ```

  oznaczającej punkt o współrzędnych zdefiniowanych za pomocą atrybutów `x`, `y`, `z`.

Jedynie punkty typu `body` są transformowane do układu związanego z sylwetką. Pozwala to na definiowanie własnych wektorów (np. wersorów), które pozwala na wygodne opisanie zachodzących relacji.
Dla punktów typu `body` zdefiniowane zostały następujące wartości atrybutu `Joint`:

1. `HEAD`,
2. `LEFT_HIP`,
3. `RIGHT_SHOULDER`,
4. `LEFT_SHOULDER`,
5. `RIGHT_ELBOW`,
6. `LEFT_ELBOW`,
7. `RIGHT_HAND`,
8. `RIGHT_HIP`,
9. `LEFT_HAND`.

Reszta dostępnych za pomocą biblioteki NITE elementów nie znajduje zastosowania dla omawianego problemu, dlatego w celu zmniejszenia generowanego obciążenia nie są one śledzone.



Rysunek 3.2 Dostępne za pomocą NITE punkty ciała (zaczerpnięto z dokumentacji biblioteki)

3.3 Rozpoznawanie gestów statycznych

Gesty statyczne są sekwencjami ruchu, które nie przekazują informacji poprzez dynamikę zjawiska, lecz pozę końcową sylwetki. Przykładem takich gestów może być wskazywanie, ręka podniesiona na znak powitania. Podczas prac nad projektem po udanej próbie wykrywania gestów statycznych zdefiniowanych w dodatku A okazało się, że dla całego zbioru można wytworzyć uniwersalną metodę w postaci:

1. Zdefiniować warunki ułożenia ciała, każdy za pomocą relacji dwóch wektorów, oraz czekać, aż wszystkie warunki zostaną spełnione.

2. Badać statyczność wybranych części ciała, gdy warunki z punktu 1. są spełnione.

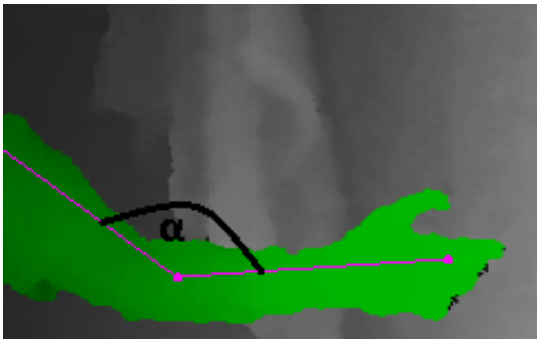
3. Zasygnalizować wystąpienie gestu jeśli bezruch trwał odpowiednio długo.

Uniwersalny opis pozwala na wprowadzanie warunków na wykrycie gestu z pliku zewnętrznego. Są one deklarowane są za pomocą następującej składni elementu Condition:

`<Condition Type=" " Subtype=" " Rel=" " Vector1=" " Vector2=" " Val=" " />,`

gdzie znaczenie występujących atrybutów jest następujące:

- Atrybut `Type` oznacza typ relacji zachodzący między podanymi wektorami. Dozwolone jest badanie kąta między wektorami (wtedy element przyjmuje wartość `angle`) lub stosunku ich długości (`length`). W celu wprowadzenia bardziej intuicyjnej, w kontekście ułożenia ciała, relacji przyjęto, że badany jest kąt wypukły między wektorem pierwszym oraz przyłożonym do jego końca wektorem drugim. Przykład przedstawiono na Rysunku 3.3.



Rysunek 3.3 Kąt α dla pary wektorów ramienia oraz przedramienia

- Podtyp `subtype` określony jest wyłącznie dla typu `length`. Wymaga się, by dla każdego warunku typu `angle` posiadał on wartość `none`. Dla warunków na długość wektora dostępne są wartości:
  - `x`, `y` lub `z` oznaczające badanie długości wybranej składowej wektora.
  - `mag` oznaczający normę euklidesową wektora.
- `Rel` określa rodzaj relacji badającej czy zadany warunek jest spełniony. Dostępne wartości to: `l, le, e, ge, g` reprezentujące odpowiednio następujące operatory porównania: $<$, $\leq$, $=$, $\geq$, $>$.
- Atrybuty `Vector1, Vector2` określają dwa wcześniej zdefiniowane w strukturze `BodySet` wektory, których stosunek długości lub kąt jest badany, zgodnie z wcześniej zdefiniowanym atrybutem `Type`. Nie jest możliwe zadanie wektora, który nie został wcześniej zadeklarowany.
- Pole `Val` definiuje wartość stojącą po prawej stronie operatora porównania.

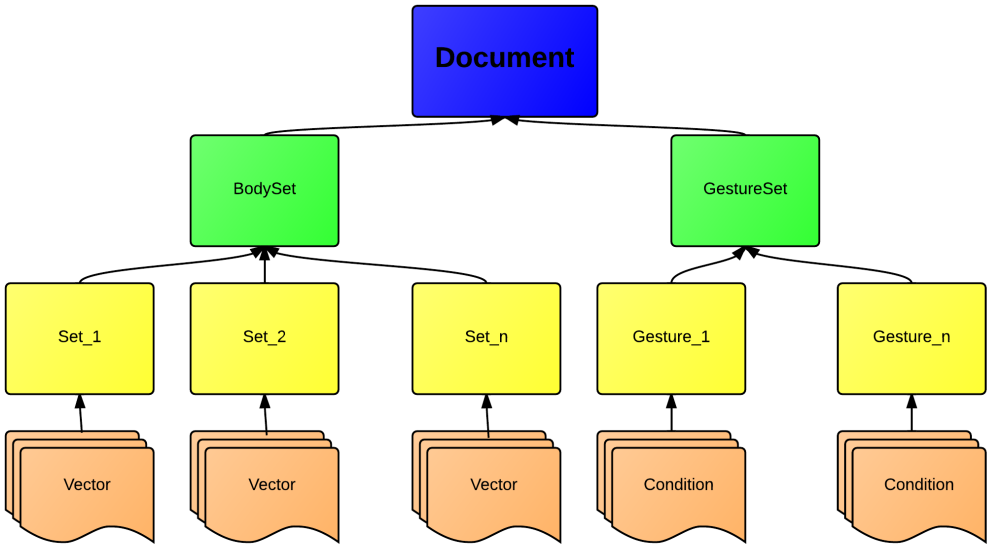
Poniżej przedstawiono przykładowy zestaw warunków zdefiniowanych w celu wykrywania gestu wskazywania lewą ręką.

```
  <Condition Type="angle" Subtype="none" Rel="g" \
Vector1="ShoulderElbow" Vector2="ElbowHand" Val="160"/>

  <Condition Type="angle" Subtype="none" Rel="g" \
Vector1="ShoulderElbow" Vector2="VersorJ" Val="45"/>
```

Zestaw ten bada kąty między lewym ramieniem i przedramieniem oraz lewym ramieniem i wersorem j (0,1,0). W pierwszym przypadku zdefiniowane wektory muszą być współliniowe z dopuszczalnym marginesem rzędu 20°. Drugie nałożone ograniczenie wymaga, by kąt między ramieniem a wersorem j był większy od kąta 45°. Powoduje to wyłączenie z takiego ułożenia sylwetki, dla którego ręka jest skierowana wzdłuż ciała.

Wszystkie elementy typu `Condition` osadzone są w znaczniku `Gesture` opisującym warunki wystarczające do wykrycia gestu. W celu utrzymania hierarchicznej struktury dokumentu XML znajdują się one w znaczniku `GestureSet`. Element



Rysunek 3.4 Struktura dokumentu XML

`Gesture` zdefiowany jest w sposób następujący:

```
<Gesture Name=" " Model=" " Set=" " Samples=" " \
Vector=" " Threshold=" " Message=" ">
  //Opcjonalna lista znaczników Condition
</Gesture>,
```

gdzie sens kolejnych atrybutów jest następujący:

- `Name` określa identyfikator gestu w postaci alfanumerycznej. Musi być on unikalny.
- `Model` determinuje czy gest traktowany jest jako statyczny czy dynamiczny. W celu wykrywania omawianych gestów atrybut przyjmuje wartość `static`.
- `Set` określa który z wcześniej zdefiniowanych zbiorów wektorów jest wykorzystywany do detekcji. Różne elementy typu `Gesture` mogą używać tych samych zbiorów, ale każdemu z gestów można przyporządkować tylko jeden zbiór.
- `Samples` definiuje liczbę klatek podczas których badane części ciała muszą pozostać w bezruchu w celu wykrycia gestu.
- `Vector` określa listę tychże członków. Zadaje się ją poprzez nazwy opisane w rozdziale 3.2 rozdzielone przecinkami. Wymaga się podania przynajmniej jednego elementu ciała.
- `Threshold` definiuję górną granicę średniej prędkości przemieszczania się zadanych przez `Vector` części ciała dla kolejnych klatek w liczbie zdefiniowanej w atrybucie `Samples`. Wprowadzenie tego parametru wynika z niemożliwości pozostania w dokładnie tej samej pozycji przez człowieka, a także z zakłóceń i niedokładności występujących podczas rejestracji obrazu.
- `Message` zawiera treść wiadomości komunikującej o wykryciu gestu.

Poniższy pseudokod prezentuje działanie algorytmu wykrywania gestów statycznych:

```
while(gest_wykrywany){
  if(warunki_sylwetki_spełnione){
    dodaj_do_zbioru_predkosci_wartosc;
    if(rozmiar_zbioru_prędkości == Samples){
      if(wartość_średnia_prędkości_każdego_elementu < Threshold)
        zakomunikuj_o_wystąpieniu_gestu;
      else
        wyczyść_zbiór_prędkości;
    }
  }else wyczyść_zbiór_prędkości;
}
```

3.4 Rozpoznawanie gestów dynamicznych

Rozpoznawanie gestów dynamicznych zachodzi w sposób analogiczny do przedstawionej metody detekcji gestów statycznych. Warunkami koniecznymi detekcji gestu jest spełnienie warunków nałożonych na pozę. Spełnienie tych warunków pozwala na zbieranie informacji na temat wektora cech zaistniałego ruchu. Jeśli wektor ten spełnia założenia, wiadomość o detekcji gestu jest wyświetlana. Elementy `Condition` mają taką samą strukturę jak w przypadku gestów dynamicznych. Znacznik `Gesture` określany jest w następujący sposób:

```
<Gesture Name=" " Model=" " Set=" " Samples=" " Vector=" " Threshold=" " \
Message=" " Type=" " Subtype=" " Rel=" " Val=" " Tolerance=" " Amplitude=" ">
</Gesture>,
```

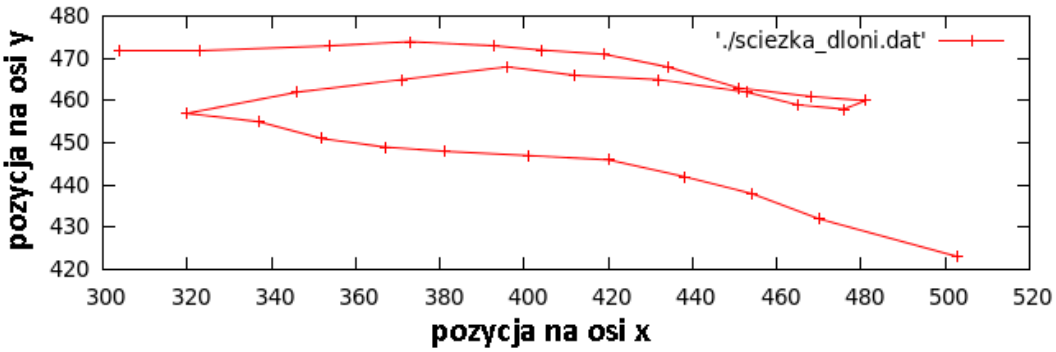
gdzie:

- Znaczenie znaczników `Name, Mode, Set, Samples, Message` jest analogiczne do statycznego rodzaju gestów.
- Atrybut `Mode` przyjmuje wartość `dynamic`.
- `Vector` określa część (części) ciała, dla której zbierany jest wektor cech.
- `Threshold` określa minimalną liczbę zmian kierunku prędkości badanego elementu, powyżej której może wystąpić detekcja gestu.
- `Type` definiuje czy badana jest prędkość kątowa między wektorami, czy liniowa pojedynczego wektora. Wartość `angle` wymaga zdefioniowania dwóch elementów w atrybucie `Vector`, a `length` tylko jednego.
- `Subtype` przyjmuje wartość `none` dla atrybutu `Type` równego `angle` oraz jedną z wartości `x, y, z, mag` dla parametru `Type` równego `length`.
- Oprócz liczby zmiany kierunku prędkości zliczana jest liczba klatek dla sekwencji pomiędzy kolejnymi zmianami kierunku (z odpowiednim znakiem zależnym od kierunku ruchu). Następnie suma zliczonych klatek porównywana jest za pomocą operatora określonego parametrem `Rel` z wartością zadaną atrybutem `Val`. `Rel` przyjmuje wartości analogiczne jak w elemencie `Condition` lub wartość `none`, co powoduje wykluczenie tego warunku z algorytmu detekcji gestów. Wprowadzenie tej cechy ruchu pozwala na rozróżnienie gestów nierozróżnialnych ze względu na ścieżkę, ale nie ze względu na trajektorię ruchu.

- Atrybut `Tolerance` określa minimalną liczbę klatek w jednym kierunku, dla której ruch ten ma zostać uwzględniony.

- `Amplitude` określa minimalną odległość (wychylenie) dla ruchu w jednym kierunku przy której zmiana kierunku ma zostać odnotowana. Dwa powyższe parametry stanowią swego rodzaju filtr dolnoprzepustowy niwelujący niedokładności wyznaczania położenia części ciała oraz drgania ludzkiej sylwetki. Przykładem sytuacji, która wymaga zastosowania takiej filtracji i dobrania odpowiednich parametrów jest rozróżnianie statycznego gestu powitania oraz gestu machania. W pierwszym przypadku ręka uniesiona jest pionowo, na wysokości głowy i pozostaje w bezruchu. Dla drugiego wariantu ręka spełnia te same wymagania co do pozy, jednak zamiast być bezczynna, porusza się ona cyklicznie w prawo i lewo. Wykonując gest powitania ręka drga, nawet nieznacznie, w płaszczyźnie torsu. Drganie to bez wprowadzenia tejże filtracji może zostać sklasyfikowane jako gest machania.

Opisane parametry zostaną zilustrowane za pomocą przykładowej ścieżki.



Rysunek 3.5 Zarejestrowana ścieżka ruchu dłoni podczas gestu machania

Kierunek ruchu dłoni dwukrotnie ulega zmianie, dlatego minimalna wartość parametru `Threshold` klasyfikująca gest powinna być równa 2. Dla kolejnych etapów ruchu podzielonych na części wynikające z kierunku przemieszczenia liczba zarejestrowanych klatek wynosi odpowiednio: 10, −9, 10. Po zsumowaniu wartości dla parzystej liczby elementów otrzymuje się wynik 1. Można z tego wnioskować, że ruch jest równomierny w obu kierunkach i nie ma potrzeby wprowadzenia dodatkowego warunku określanego parametrami `Val` oraz `Rel`. Parametry filtrujące należałoby dobrać jako połowę zliczonych klatek i połowę wychylenia dla ruchu w jednym kierunku. `Tolerance` wynosiłoby 5, a `Amplitude` 100. Z tak dobranymi parametrami powyższy ruch zostałby z sukcesem sklasyfikowany jako gest machania.

Rozdział 4

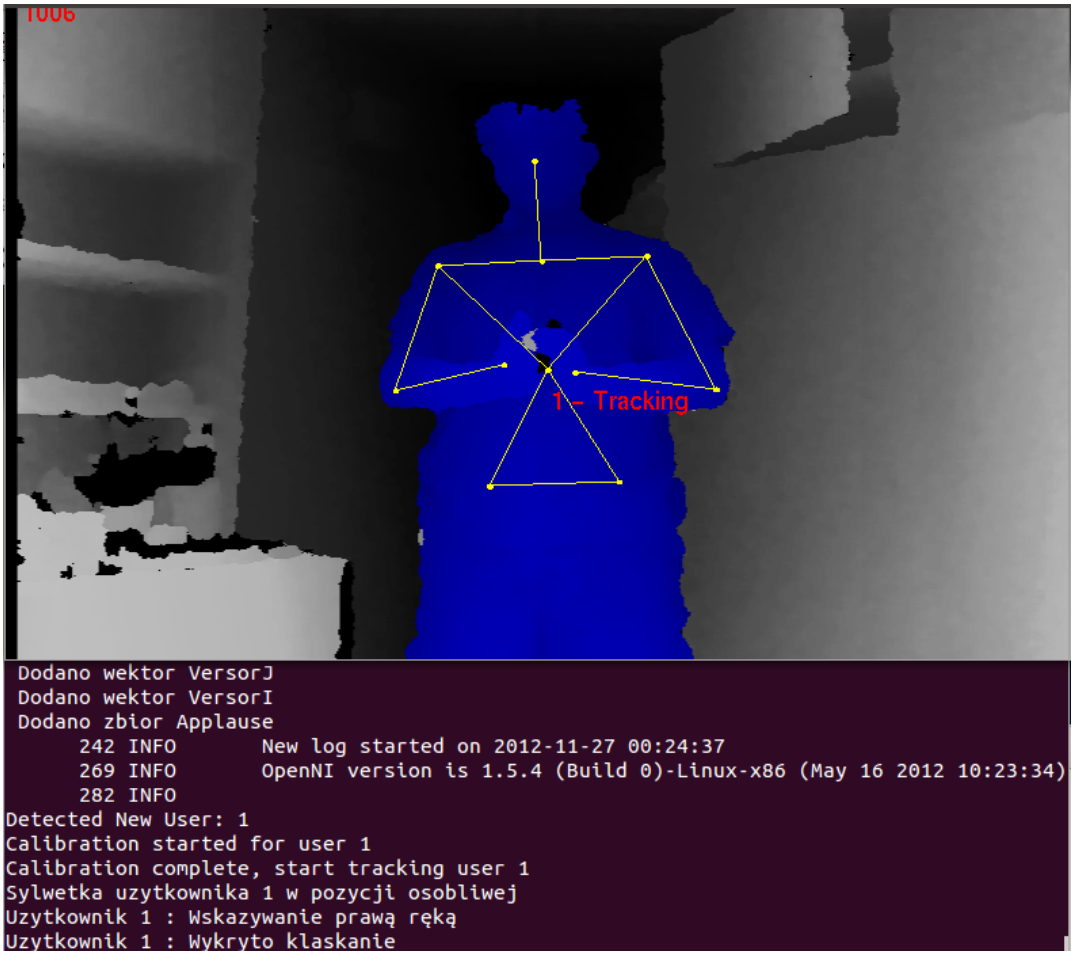
Uzyskane rezultaty

4.1 Aplikacja konsolowa

Głownym rezultatem osiągniętym podczas projektu jest aplikacja konsolowa realizująca wykrywanie gestów zgodne z przedstawionym modelem.
Właściwości aplikacji:

- Wykrywane gesty definiowane są zewnętrznie, bez konieczności każdorazowego kompilowania aplikacji.
- Podczas wykrycia gestu w oknie konsoli pojawia się komunikat o detekcji zdarzenia.
- Za pomocą biblioteki OpenGL, podczas działania programu, następuje wizualizacja mapy głębi z wyróżnieniem użytkowników.
- Liczba użytkowników jest ograniczona jedynie ze względu na rozmiar sceny.
- Sylwetka powinna znajdować się nie bliżej niż w odległości $1.8m$ dla jednego lub 2.5m dla większej liczby użytkowników.
- Aplikacja wstrzymuje proces detekcji gestów dla użytkownika w przypadku gdy biblioteka NITE nie jest w stanie jednoznacznie stwierdzić gdzie znajduje się którakolwiek z górnych części ciała (tzw. sylwetka osobliwa). Użytkownik zostaje o tym fakcie poinformowany komunikatem w oknie konsoli.
- Detekcja gestów działa w sposób niezależny dla każdego użytkownika.
- Aplikacja cechuje się przenośnością. Przetestowane zostały wersje systemu Ubuntu 12.04 oraz 10.04, jednak dostępność wykorzystywanych bibliotek na platformy Windows oraz MacOS wskazuje, że te systemu również będą w stanie obsłużyć aplikację.

- W aplikacji zamiennie można wykorzystywać sensor Xtion firmy Asus lub inne czujniki, które współpracują z OpenNI.
- Wraz z aplikacją dostarczony zostaje plik XML opisujący gesty zawarte w dodatku A.
- Aplikacja jest udokumentowana za pomocą programu Doxygen.



Rysunek 4.1 Wygląd aplikacji

Wady aplikacji:

- W przypadku częstego występowania sylwetek osobliwych aplikacja zgłasza błąd przekroczenia limitu czasu oczekiwania na dane z sensora. Wada ta jest związana z cechami biblioteki wykorzystanej do komunikacji z Kinectem.

- Użyta w projekcie biblioteka NITE nie wykrywa pozycji nadgarstków i koniuszków palców oraz orientacji dłoni, co sprawia, że ułożenie dłoni nie może być elementem decydującym o detekcji gestu.
- Skuteczność aplikacji zależy od parametrów zdefiniowanych w dokumencie XML.

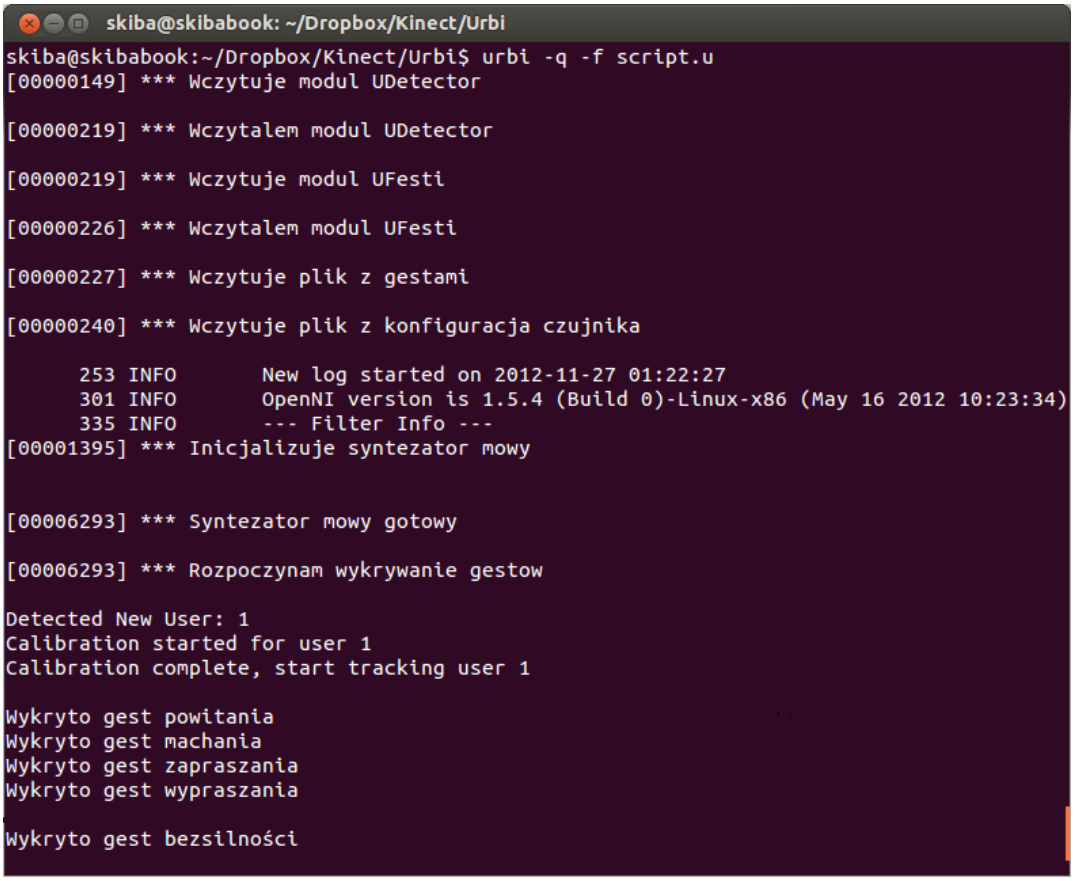
4.2 Moduł Urbi

Aplikacja została skompilowana również do modułu typu *UObject*, który można wykorzystywać w programach Urbi lub Gostai Studio. Dedykowane są do projektów związanych z robotyką.
W tym celu przetworzono strukturę programu do klasy statycznej, którą później za pomocą mechanizmów Urbi przygotowano do kompilacji do modułu *UObject*. Moduł zapewnia funkcję dokonującą jednorazowej detekcji gestu oraz funkcę zwracającą komunikat o wystąpieniu danego gestu.
Moduł kompilowany jest za pomocą komendy:

```
umake-shared UDetector.cpp UDetector.hh inc/detector.hh src/ \
-o UDetector -I/usr/include/ni/ -I../../Include -Iinc -lOpenNI  \
-lxerces-c -lglut -DUSE_GLUT
```

W celu zaprezentowania możliwości URBI załadowano również moduł do syntezy mowy za pomocą biblioteki Festival[10] przygotowany przez Natalię Czop w ramach kursu „Sterowniki robotów”[11]. Przykładowy skrypt wykonujący program zaprezentowany na rysunku 4.2 ma postać:

```
echo("Wczytuje modul UDetector\n");
loadModule("UDetector.so");
echo("Wczytalem modul UDetector\n");
echo("Wczytuje modul UFestivali\n");
loadModule("Festivali.so");
echo("Wczytalem modul UFestivali\n");
var Detector=UDetector.new()|;
echo("Wczytuje plik z gestami\n");
Detector.XmlInit("actions.xml")|;
echo("Wczytuje plik z konfiguracja czujnika\n");
Detector.KinectInit("Config/Config.xml");
echo("Inicjalizuje syntezator mowy\n");
var Festival=UFestivali.new()|;
Festival.Initialize();
echo("Syntezator mowy gotowy\n");
```



Rysunek 4.2 Skrypt wykonywany w Urbi

```
echo("Rozpoczynam wykrywanie gestow\n");
while(1){
  Detector.Run();
  Detector.GetMsg();
};
```

Rozdział 5

Podsumowanie

5.1 Wyniki projektu

Niniejszy projekt zakładał stworzenie aplikacji do detekcji gestów opisanych w dodatku A. Do tego celu wykorzystano sensor Kinect oraz biblioteki wspomagające. W ramach projektu rozwiązano liczne problemy, które stanęły na drodze do realizacji postawionych celów.

Postawione zadanie udało się zrealizować. Powstał program, który z sukcesem identyfikuje wybrane gesty. W procesie tworzenia wykształcono model, dzięki któremu w sposób uniwersalny można wykrywać różne gesty. Dodatkowo zapewniono możliwość zewnętrznego definiowania gestów za pomocą pliku XML o zadanej strukturze. Aplikacja oferuje również wizualizację mapy głębi. Dostarczono algorytm do detekcji gestów w postaci modułu *UObject* do programu URBI, który prężnie rozwija się na polach naukowych robotyki.

5.2 Dalszy rozwój projektu

Planowany jest dalszy rozwój projektu. Kolejne cele, które podczas rozwoju pracy zostały sformułowane to:

- Uaktualnienie biblioteki odpowiadającej za komunikację z sensorem do nowszej, bardziej stabilnej wersji.
- Wymiana biblioteki NITE na KINECT SDK, która zapewnia detekcję położenia nadgarstków, koniuszków palców oraz orientacje dłoni.
- Rozszerzenie zbioru wykrywanych gestów.

- Przygotowanie scenariusza zachowania robota społecznego w programie Gostai Studio z wykorzystaniem przygotowanego modułu URBI oraz przeprowadzenie doświadczeń na robocie EMYS.

Rozdział 6

Dodatek A

Wybrane gesty — plik XML

Definicja wektorów

```
<Document>
  <BodySet>
    <Set Name="LeftHandPointing">
      <Vector Name="ShoulderElbow">
        <Point Type="body" Joint="LEFT_SHOULDER"/>
        <Point Type="body" Joint="LEFT_ELBOW"/>
      </Vector>
      <Vector Name="ElbowHand">
        <Point Type="body" Joint="LEFT_ELBOW"/>
        <Point Type="body" Joint="LEFT_HAND"/>
      </Vector>
      <Vector Name="VersorJ">
        <Point Type="null" />
        <Point Type="defined" x="0" y="1" z="0"/>
      </Vector>
    </Set>
    <Set Name="RightHandPointing">
      <Vector Name="ShoulderElbow">
        <Point Type="body" Joint="RIGHT_SHOULDER"/>
        <Point Type="body" Joint="RIGHT_ELBOW"/>
      </Vector>
      <Vector Name="ElbowHand">
        <Point Type="body" Joint="RIGHT_ELBOW"/>
        <Point Type="body" Joint="RIGHT_HAND"/>
```

```
    </Vector>
    <Vector Name="VersorJ">
      <Point Type="null" />
      <Point Type="defined" x="0" y="1" z="0"/>
    </Vector>
  </Set>
  <Set Name="Bezsilnosc">
    <Vector Name="VersorI">
      <Point Type="null"/>
      <Point Type="defined" x="1" y="0" z="0"/>
    </Vector>
    <Vector Name="VectorRight">
      <Point Type="null" />
      <Point Type="defined" x="1" y="-1" z="0"/>
    </Vector>
    <Vector Name="ShoulderElbowRight">
      <Point Type="body" Joint="RIGHT_SHOULDER"/>
      <Point Type="body" Joint="RIGHT_ELBOW"/>
    </Vector>
    <Vector Name="ElbowHandRight">
      <Point Type="body" Joint="RIGHT_ELBOW"/>
      <Point Type="body" Joint="RIGHT_HAND"/>
    </Vector>
    <Vector Name="VectorLeft">
      <Point Type="null" />
      <Point Type="defined" x="-1" y="-1" z="0"/>
    </Vector>
    <Vector Name="ShoulderElbowLeft">
      <Point Type="body" Joint="LEFT_SHOULDER"/>
      <Point Type="body" Joint="LEFT_ELBOW"/>
    </Vector>
    <Vector Name="ElbowHandLeft">
      <Point Type="body" Joint="LEFT_ELBOW"/>
      <Point Type="body" Joint="LEFT_HAND"/>
    </Vector>
  </Set>
  <Set Name="Greeting">
    <Vector Name="VersorJ">
      <Point Type="null" />
      <Point Type="defined" x="0" y="1" z="0"/>
```

```
    </Vector>
    <Vector Name="ElbowHand">
      <Point Type="body" Joint="RIGHT_ELBOW"/>
      <Point Type="body" Joint="RIGHT_HAND"/>
    </Vector>
  </Set>
  <Set Name="HandOnHead">
    <Vector Name="BodyLine">
      <Point Type="body" Joint="HEAD" />
      <Point Type="body" Joint="TORSO" />
    </Vector>
    <Vector Name="HeadLeftHand">
      <Point Type="body" Joint="HEAD" />
      <Point Type="body" Joint="LEFT_HAND"/>
    </Vector>
    <Vector Name="HeadRightHand">
      <Point Type="body" Joint="HEAD" />
      <Point Type="body" Joint="RIGHT_HAND"/>
    </Vector>
    <Vector Name="Hands">
      <Point Type="body" Joint="LEFT_HAND" />
      <Point Type="body" Joint="RIGHT_HAND"/>
    </Vector>
    <Vector Name="Elbows">
      <Point Type="body" Joint="LEFT_ELBOW" />
      <Point Type="body" Joint="RIGHT_ELBOW"/>
    </Vector>
    <Vector Name="VersorI">
      <Point Type="null"/>
      <Point Type="defined" x="1" y="0" z="0"/>
    </Vector>
    <Vector Name="Shoulders">
      <Point Type="body" Joint="LEFT_SHOULDER" />
      <Point Type="body" Joint="RIGHT_SHOULDER"/>
    </Vector>
  </Set>
  <Set Name="HandsOnHips">
    <Vector Name="RightShoulderElbow">
      <Point Type="body" Joint="RIGHT_SHOULDER" />
      <Point Type="body" Joint="RIGHT_ELBOW"/>
```

```
    </Vector>
    <Vector Name="LeftShoulderElbow">
      <Point Type="body" Joint="LEFT_SHOULDER" />
      <Point Type="body" Joint="LEFT_ELBOW"/>
    </Vector>
    <Vector Name="LeftElbowHand">
      <Point Type="body" Joint="LEFT_ELBOW" />
      <Point Type="body" Joint="LEFT_HAND"/>
    </Vector>
    <Vector Name="RightElbowHand">
      <Point Type="body" Joint="RIGHT_ELBOW" />
      <Point Type="body" Joint="RIGHT_HAND"/>
    </Vector>
    <Vector Name="VersorJ">
      <Point Type="null" />
      <Point Type="defined" x="0" y="-1" z="0"/>
    </Vector>
    <Vector Name="RightShoulderHand">
      <Point Type="body" Joint="RIGHT_SHOULDER" />
      <Point Type="body" Joint="RIGHT_HAND"/>
    </Vector>
    <Vector Name="LeftShoulderHand">
      <Point Type="body" Joint="LEFT_SHOULDER" />
      <Point Type="body" Joint="LEFT_HAND"/>
    </Vector>
    <Vector Name="Shoulders">
      <Point Type="body" Joint="LEFT_SHOULDER" />
      <Point Type="body" Joint="RIGHT_SHOULDER"/>
    </Vector>
    <Vector Name="Hands">
      <Point Type="body" Joint="LEFT_HAND" />
      <Point Type="body" Joint="RIGHT_HAND"/>
    </Vector>
  </Set>
  <Set Name="Waving">
    <Vector Name="RightShoulderElbow">
      <Point Type="body" Joint="RIGHT_SHOULDER" />
      <Point Type="body" Joint="RIGHT_ELBOW"/>
    </Vector>
    <Vector Name="LeftShoulderElbow">
```

```
      <Point Type="body" Joint="LEFT_SHOULDER" />
      <Point Type="body" Joint="LEFT_ELBOW"/>
    </Vector>
    <Vector Name="VersorJ">
      <Point Type="null" />
      <Point Type="defined" x="0" y="1" z="0"/>
    </Vector>
    <Vector Name="RightElbowHand">
      <Point Type="body" Joint="RIGHT_ELBOW" />
      <Point Type="body" Joint="RIGHT_HAND"/>
    </Vector>
  </Set>
  <Set Name="Invite">
    <Vector Name="RightShoulderElbow">
      <Point Type="body" Joint="RIGHT_SHOULDER" />
      <Point Type="body" Joint="RIGHT_ELBOW"/>
    </Vector>
    <Vector Name="VersorJ">
      <Point Type="null" />
      <Point Type="defined" x="0" y="1" z="0"/>
    </Vector>
    <Vector Name="RightElbowHand">
      <Point Type="body" Joint="RIGHT_ELBOW" />
      <Point Type="body" Joint="RIGHT_HAND"/>
    </Vector>
  </Set>
  <Set Name="Applause">
    <Vector Name="RightShoulderElbow">
      <Point Type="body" Joint="RIGHT_SHOULDER" />
      <Point Type="body" Joint="RIGHT_ELBOW"/>
    </Vector>
    <Vector Name="RightElbowHand">
      <Point Type="body" Joint="RIGHT_ELBOW" />
      <Point Type="body" Joint="RIGHT_HAND"/>
    </Vector>
    <Vector Name="LeftShoulderElbow">
      <Point Type="body" Joint="LEFT_SHOULDER" />
      <Point Type="body" Joint="LEFT_ELBOW"/>
    </Vector>
    <Vector Name="LeftElbowHand">
```

```
      <Point Type="body" Joint="LEFT_ELBOW" />
      <Point Type="body" Joint="LEFT_HAND"/>
    </Vector>
    <Vector Name="Hands">
      <Point Type="body" Joint="LEFT_HAND" />
      <Point Type="body" Joint="RIGHT_HAND"/>
    </Vector>
    <Vector Name="Elbows">
      <Point Type="body" Joint="LEFT_ELBOW" />
      <Point Type="body" Joint="RIGHT_ELBOW"/>
    </Vector>
    <Vector Name="Shoulders">
      <Point Type="body" Joint="LEFT_SHOULDER" />
      <Point Type="body" Joint="RIGHT_SHOULDER"/>
    </Vector>
    <Vector Name="VersorJ">
      <Point Type="null" />
      <Point Type="defined" x="0" y="1" z="0"/>
    </Vector>
    <Vector Name="VersorI">
      <Point Type="null" />
      <Point Type="defined" x="1" y="0" z="0"/>
    </Vector>
  </Set>
</BodySet>
<GestureSet>
  <Gesture Name="Applause" Model="dynamic" Set="Applause" Samples="60"\
   Vector="Hands" Threshold="4" Message="Wykryto klaskanie" \
   Type="length" Subtype="mag" Rel="none" Val="30" Severity="3"\
    Density="30">
    <Condition Type="angle" Subtype="none" Rel="l" Vector1="VersorJ"\
     Vector2="RightShoulderElbow" Val="30"/>
    <Condition Type="angle" Subtype="none" Rel="l" Vector1="VersorJ"\
     Vector2="LeftShoulderElbow" Val="30"/>
    <Condition Type="angle" Subtype="none" Rel="g" Vector1="VersorJ"\
     Vector2="RightElbowHand" Val="90"/>
    <Condition Type="angle" Subtype="none" Rel="g" Vector1="VersorJ"\
     Vector2="LeftElbowHand" Val="90"/>
    <Condition Type="length" Subtype="mag" Rel="l" Vector1="Hands"\
     Vector2="Shoulders" Val="1" />
```

```
  <Condition Type="angle" Subtype="none" Rel="g" Vector1="VersorI"\
   Vector2="Elbows" Val="160"/>
</Gesture>
<Gesture Name="RightHandPointing" Model="static" Set=\
"RightHandPointing" Samples="60" Vector="RIGHT_HAND,RIGHT_ELBOW"\
 Threshold="10" Message="Wykryto wskazywanie prawą ręką">
  <Condition Type="angle" Subtype="none" Rel="g"\
   Vector1="ShoulderElbow" Vector2="ElbowHand" Val="160"/>
  <Condition Type="angle" Subtype="none" Rel="g"\
   Vector1="ShoulderElbow" Vector2="VersorJ" Val="45"/>
</Gesture>
<Gesture Name="HandsOnHips" Model="static" Set="HandsOnHips"\
 Samples="60" Vector="LEFT_HAND,RIGHT_HAND" Threshold="10"\
  Message="Wykryto zniecierpliwienie">
  <Condition Type="angle" Subtype="none" Rel="g" Vector1=\
  "VersorJ" Vector2="LeftShoulderHand" Val="130"/>
  <Condition Type="angle" Subtype="none" Rel="g" Vector1=\
  "VersorJ" Vector2="RightShoulderHand" Val="130"/>
  <Condition Type="angle" Subtype="none" Rel="g" Vector1=\
  "LeftShoulderElbow" Vector2="LeftElbowHand" Val="70"/>
  <Condition Type="angle" Subtype="none" Rel="l" Vector1=\
  "LeftShoulderElbow" Vector2="LeftElbowHand" Val="110"/>
  <Condition Type="angle" Subtype="none" Rel="g" Vector1=\
  "RightShoulderElbow" Vector2="RightElbowHand" Val="70"/>
  <Condition Type="angle" Subtype="none" Rel="l" Vector1=\
  "RightShoulderElbow" Vector2="RightElbowHand" Val="110"/>
  <Condition Type="length" Subtype="mag" Rel="g" Vector1=\
  "Hands" Vector2="Shoulders" Val="0.8"/>
</Gesture>
<Gesture Name="LeftHandPointing" Model="static" Set=\
"LeftHandPointing" Samples="60" Vector="LEFT_HAND" Threshold="10"\
  Message="Wykryto wskazywanie prawą ręką">
  <Condition Type="angle" Subtype="none" Rel="g"\
   Vector1="ShoulderElbow" Vector2="ElbowHand" Val="160"/>
  <Condition Type="angle" Subtype="none" Rel="g"\
   Vector1="ShoulderElbow" Vector2="VersorJ" Val="45"/>
</Gesture>
<Gesture Name="Bezsilnosc" Model="static" Set="Bezsilnosc"\
 Samples="30" Vector="LEFT_HAND,RIGHT_HAND" Threshold="20"\
  Message="Wykryto gest bezsilności">
```

```
  <Condition Type="angle" Subtype="none" Rel="g" Vector1=\
  "VectorRight" Vector2="ShoulderElbowRight" Val="150"/>
  <Condition Type="angle" Subtype="none" Rel="g" Vector1=\
  "VectorLeft" Vector2="ShoulderElbowLeft" Val="150"/>
  <Condition Type="angle" Subtype="none" Rel="l" Vector1=\
  "ShoulderElbowRight" Vector2="ElbowHandRight" Val="160"/>
  <Condition Type="angle" Subtype="none" Rel="l" Vector1=\
  "ShoulderElbowLeft" Vector2="ElbowHandLeft" Val="160"/>
  <Condition Type="angle" Subtype="none" Rel="g" Vector1=\
  "ShoulderElbowRight" Vector2="ElbowHandRight" Val="120"/>
  <Condition Type="angle" Subtype="none" Rel="g" Vector1=\
  "ShoulderElbowLeft" Vector2="ElbowHandLeft" Val="120"/>
  <Condition Type="angle" Subtype="none" Rel="l" Vector1=\
  "VersorI" Vector2="ElbowHandLeft" Val="90"/>
  <Condition Type="angle" Subtype="none" Rel="g" Vector1=\
  "VersorI" Vector2="ElbowHandRight" Val="90"/>
</Gesture>
<Gesture Name="HandOnHead" Model="static" Set="HandOnHead"\
 Samples="60" Vector="LEFT_HAND,RIGHT_HAND" Threshold="10"\
  Message="Wykryto gest zdziwienia">
  <Condition Type="angle" Subtype="none" Rel="g" Vector1=\
  "VersorI" Vector2="Hands" Val="160"/>
  <Condition Type="length" Subtype="y" Rel="l" Vector1=\
  "HeadRightHand" Vector2="BodyLine" Val="0.3"/>
  <Condition Type="length" Subtype="y" Rel="l" Vector1=\
  "HeadLeftHand" Vector2="BodyLine" Val="0.3"/>
  <Condition Type="length" Subtype="z" Rel="l" Vector1=\
  "HeadRightHand" Vector2="BodyLine" Val="1"/>
  <Condition Type="length" Subtype="z" Rel="l" Vector1=\
  "HeadLeftHand" Vector2="BodyLine" Val="1"/>
</Gesture>
<Gesture Name="Greeting" Model="static" Set="Greeting" Samples="60"\
 Vector="RIGHT_HAND" Threshold="10" Message="Wykryto gest powitania">
  <Condition Type="angle" Subtype="none" Rel="g" Vector1=\
  "VersorJ" Vector2="ElbowHand" Val="150"/>
</Gesture>
<Gesture Name="ThrowOut" Model="dynamic" Set="Waving" Samples="90"\
 Vector="RightElbowHand" Threshold="3" Message="Wykryto gest\
  wypraszania" Type="length" Subtype="z" Rel="l"\
   Val="-10" Severity="3" Density="30">
```

```
      <Condition Type="angle" Subtype="none" Rel="l" Vector1=\
      "VersorJ" Vector2="RightShoulderElbow" Val="90"/>
      <Condition Type="angle" Subtype="none" Rel="l" Vector1=\
      "RightShoulderElbow" Vector2="RightElbowHand" Val="120"/>
    </Gesture>
    <Gesture Name="Invite" Model="dynamic" Set="Waving" Samples="90"\
     Vector="RightElbowHand" Threshold="3" Message="Wykryto gest\
 zapraszania" Type="length" Subtype="z" Rel="g" Val="10"\
  Severity="3" Density="30">
      <Condition Type="angle" Subtype="none" Rel="l" Vector1=\
      "VersorJ" Vector2="RightShoulderElbow" Val="90"/>
      <Condition Type="angle" Subtype="none" Rel="l" Vector1=\
      "RightShoulderElbow" Vector2="RightElbowHand" Val="120"/>
    </Gesture>
    <Gesture Name="Waving" Model="dynamic" Set="Waving" Samples="60"\
     Vector="VersorJ,RightElbowHand" Threshold="4"\
      Message="Wykryto machanie" Type="angle" Subtype="none"\
       Rel="none" Val="0" Severity="2" Density="5">
      <Condition Type="angle" Subtype="none" Rel="g" Vector1=\
      "VersorJ" Vector2="RightElbowHand" Val="130"/>
      <Condition Type="angle" Subtype="none" Rel="l" Vector1=\
      "VersorJ" Vector2="RightShoulderElbow" Val="90"/>
    </Gesture>
  </GestureSet>
</Document>
```

Bibliografia

[1] Ying Wu, Thomas S. Huang, *Vision-Based Gesture Recognition: A Review*

[2] Sushmita Mitra, Tinku Acharya, *Gesture Recognition: A Survey* , IEEE Transactoons on systems, MAN, and Cybernetics—part C: Applications and reviews, VOL. 37, NO. 3, May 2007

[3] http://www.microsoft.com/en-us/kinectforwindows/develop/resources.aspx

[4] https://github.com/avin2/SensorKinect

[5] http://openni.org/

[6] http://asus.com/Multimedia/Motion_Sensor/Xtion_PRO/

[7] http://www.primesense.com/nite

[8] http://www.opengl.org/

[9] http://xerces.apache.org/xerces-c/

[10] http://www.cstr.ed.ac.uk/projects/festival/

[11] Natalia Czop, *Gostai Urbi — Syntezator mowy*, Wrocław 2011

[12] http://www.urbiforge.org/

[13] The Urbi Software Development Kit

[14] http://flash.lirec.ict.pwr.wroc.pl/

[15] http://www.informatik.uni-augsburg.de/de/lehrstuehle/hcm/projects/tools/fubi/

[16] http://channel9.msdn.com/coding4fun/kinect/Faceshift-a-Kinect-based-real-time-facial-movement-package

[17] http://channel9.msdn.com/coding4fun/kinect/Kinect-mousing-with-Kinecursor-and-a-plea-for-your-help