

**Uniwersytet Warszawski**  
Wydział Matematyki, Informatyki i Mechaniki

**Paweł Tomaszewski**

Nr albumu: 292647

**Interaktywna eksploracja i  
dopasowanie lokalnie optymalnych  
struktur biopolimerów z  
wykorzystaniem metod wirtualnej  
rzeczywistości**

**Praca licencjacka  
na kierunku BIOINFORMATYKA**

Praca wykonana pod kierunkiem  
**dr. Pawła Daniluka**  
Instytut Fizyki Doświadczalnej  
Zakład Biofizyki

Maj 2017

## **Oświadczenie kierującego pracą**

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data

Podpis kierującego pracą

## **Oświadczenie autora (autorów) pracy**

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data

Podpis autora pracy

## **Streszczenie**

W pracy przedstawiono implementację programu narzędziowego służącego do interaktywnej eksploracji cząsteczek biopolimerów (polinukleotydów i polipeptydów) w poszukiwaniu miejsc optymalnych ze względu na lokalne podobieństwo do zadanych struktur. W pracy szeroko zastosowano metody wirtualnej rzeczywistości wraz z obsługą urządzenia haptycznego Sensable Phantom Omni.

## **Słowa kluczowe**

bioinformatyka, wirtualna rzeczywistość, podobieństwo strukturalne, RNA, DNA, białka, biopolimery, dopasowanie strukturalne

## **Dziedzina pracy (kody wg programu Socrates-Erasmus)**

11.3 Informatyka

13.1 Biologia

13.2 Fizyka

## **Klasyfikacja tematyczna**

D. Software

D.127. Blabalgorithms

D.127.6. Numerical blabalalysis



# Spis treści

<b>Wprowadzenie</b>	5
<b>Cel i zakres pracy</b>	7
<b>1. Podstawy teoretyczne</b>	9
1.1. Metody dopasowania struktur chemicznych	9
1.1.1. Globalne dopasowanie struktur	10
1.1.2. Lokalne dopasowanie struktur	11
1.2. RMSD i algorytm Kabsch’a	11
1.3. Przekształcenia geometryczne i ich reprezentacje	15
1.3.1. Skalowanie	15
1.3.2. Translacje	17
1.3.3. Rotacje i kwaterniony	17
1.4. Pole siłowe	20
<b>2. Urządzenie haptyczne Sensable Phantom Omni</b>	21
2.1. Opis urządzenia	22
2.2. Wymagania sprzętowe	22
2.3. OpenHaptics Toolkit v3.0	23
2.3.1. Phantom Device Drivers	23
2.3.2. Haptic Device API	23
2.3.3. Haptic Library API	24
2.3.4. QuickHaptics Micro API	25
2.4. Servo Loop	26
<b>3. Virtual Reality Peripheral Network</b>	27
3.1. Opis pakietu	27
3.2. Opis użycia VRPN	28
<b>4. Implementacja i uruchomienie oprogramowania</b>	29
4.1. Opis stanowiska laboratoryjnego	29
4.2. Rozszerzanie funkcjonalności pakietu PyMOL	30
4.3. Dane wejściowe	30
4.4. Opis oprogramowania	31
<b>5. Podsumowanie</b>	37
<b>Bibliografia</b>	39

Kod źródłowy . . . . .	41
------------------------	----

# Wprowadzenie

Biopolimery, a w szczególności kwasy nukleinowe i polipeptydy stanowią fundamenty życia. W każdej komórce oraz w całym organizmie są one odpowiedzialne za najważniejsze funkcje związane z odżywianiem, rozmnażaniem czy obroną przed patogenami. Dzięki kwasom nukleinowym możliwe jest przenoszenie informacji genetycznej, biorą one udział w syntezie białek, a także mają funkcję enzymatyczne. Polipeptydy zaś są składnikami budulcowymi wielu organów, pełnią funkcje hormonalne czy regulatorowe.

Pomimo odmiennej budowy tych dwóch klas cząsteczek w obu przypadkach ich funkcja wynika bezpośrednio ze struktury przestrzennej, która to za hipotezą Anfinsena [1] jest ściśle zdeterminowana przez sekwencję nukleotydów czy aminokwasów.

Struktura przestrzenna biopolimerów stanowi obecnie przedmiot niezwykle intensywnych badań na całym świecie. Jest to bardzo ważne zagadnienie integrujące ze sobą grupy naukowców z dziedzin, które jeszcze na przełomie wieków miały ze sobą niewiele wspólnego. Do grona biologów i chemików dołączyli matematycy, fizycy oraz informatycy wspólnie tworząc bardzo nowatorskie narzędzia ułatwiające modelowanie i tworzenie symulacji zachodzących w skali molekularnej. W badania nad strukturami przestrzennymi biopolimerów zaangażowane są największe na świecie ośrodki naukowe, korporacje farmaceutyczne czy agencje rządowe. Badanie interakcji receptorów z ligandami, projektowanie nowych leków czy terapie celowane to tylko wąski wycinek zagadnień związanych z tymi pracami.

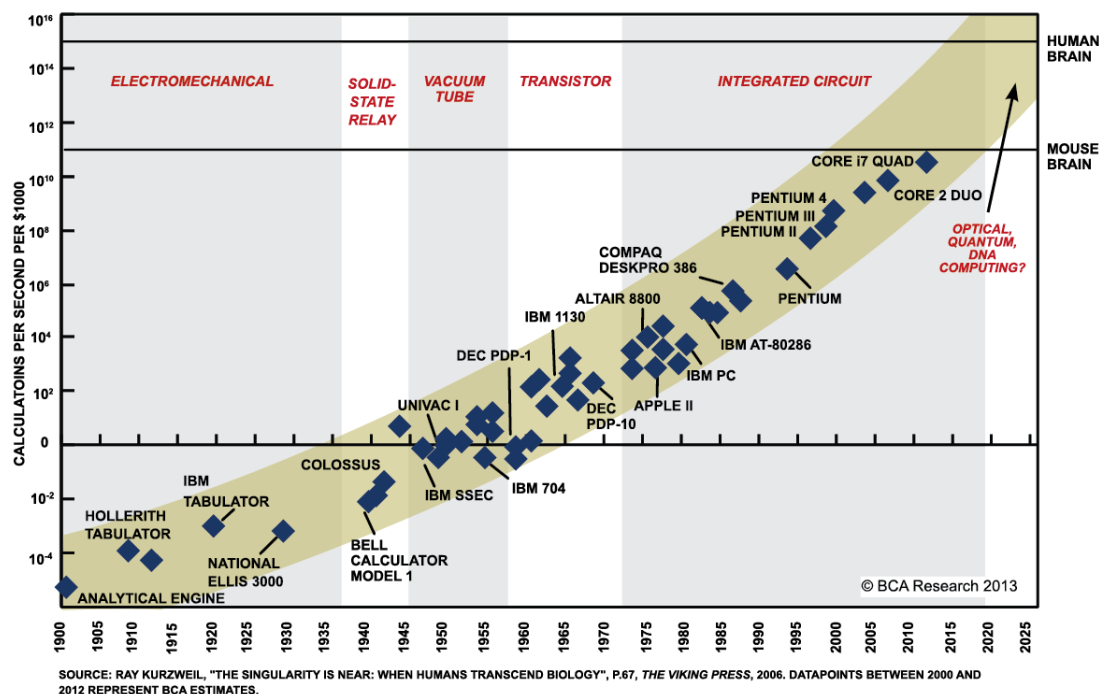
Intensywny rozwój narzędzi bioinformatycznych znacznie ułatwił i przyspieszył te badania. Obecnie najwydajniejsze komputery bez przerwy prowadzą obliczenia optymalnie pofałdowanych białek, kwasów nukleinowych czy prowadzą symulacje dynamiki molekularnej. Jest to dziedzina w której niemalże każdego roku dokonuje się znaczących odkryć i prawdopodobnie jeszcze przez długi czas to się nie zmieni. Wystarczy przytoczyć wyniki odbywającego się co dwa lata międzynarodowego konkursu CASP na najwierniejsze przewidywanie struktury białka, który za każdym razem przynosi wyniki coraz bardziej zbieżne z konformacjami natywnymi [2].

Ważnym aspektem tych badań jest prezentacja wyników szerokiemu gronu odbiorców. Same efektowne wizualizacje nie zawsze są wystarczające, coraz częściej chcemy wchodzić w bezpośrednią interakcję ze światem do którego dostępu nie mieliśmy nigdy wcześniej. W związku z tym w ostatnim czasie coraz częściej sięga się do rozwiązań z zakresu wirtualnej rzeczywistości.

Mianem *wirtualnej rzeczywistości* (*ang. virtual reality, VR*) określamy sztuczne, wykreowane przy pomocy technologii informatycznych, multimedialne projekcje przestrzeni, przedmiotów czy zdarzeń. Na obecnym poziomie rozwoju technologii wirtualna rzeczywistość umożliwia człowiekowi wchodzenie w interakcję z tym środowiskiem przede wszystkim za pośrednictwem zmysłów wzroku, słuchu czy dotyku wykorzystując urządzenia klasy HCI (*ang. human computer interface*).

Wirtualna rzeczywistość w dzisiejszym świecie zyskuje coraz to większą popularność w wielu dziedzinach życia, od zastosowań czysto rozrywkowych po zaawansowane projekty na-

ukowe, przemysłowe, a także wojskowe. Postępująca od wielu lat miniaturyzacja, rozwój nowych algorytmów czy drastyczne zwiększenie wydajności obliczeniowej sprzętu komputerowego jedynie przyspiesza ten proces.



Rysunek 1: Prawo Moora

Elementami niezbędnymi do prawidłowego wykreowania wirtualnego środowiska jest zarówno dedykowane oprogramowanie jak i sprzęt konieczny do przekazywania informacji zwrotnych do użytkownika. Rzeczywistość wirtualna aby zostać możliwie najlepiej zinterpretowana przez ludzki mózg musi jak najbardziej przypominać rzeczywistość w której żyjemy na co dzień. Aby sprostać temu zadaniu najczęściej reprezentuje się ją w postaci trójwymiarowych scen. Już tylko ten jeden czynnik powoduje, że do poprawnej symulacji niezbędne są nowoczesne, wysokowydajne procesory i karty graficzne będące w stanie przeprowadzić niezbędne obliczenia.

Jak już wspomniano wirtualna rzeczywistość może znajdować zastosowanie także w nauce w szczególności w dziedzinach w których obiekty zainteresowań są zbyt małe aby być widoczne gołym okiem, takie jak cząsteczki chemiczne lub pojedyncze atomy. Istnieje cały szereg programów przeprowadzających np. symulacje oddziaływań międzycząsteczkowych, zwijania białek (*ang. protein folding*) czy przeprowadzających obliczenia dynamiki molekularnej (*ang. molecular dynamics*), a także umożliwiających wizualizację tych symulacji. Stosunkowo niewiele jednak jest rozwiązań dedykowanych wirtualnej rzeczywistości, które mogłyby umożliwić interakcję z użytkownikiem za pośrednictwem zmysłu dotyku.

Pracownie Laboratorium Biofizyki na Wydziale Fizyki Uniwersytetu Warszawskiego dysponują sprzętem niezbędnym do realizacji takich zadań. Urządzenie haptyczne Sensable Phantom Omni jest przykładem trójwymiarowego wskaźnika ze zwrotną projekcją momentów sił, którego wykorzystanie otwiera całe spektrum nowych możliwości związanych z realizacją projektów wirtualnej rzeczywistości w biofizyce, biologii molekularnej czy chemii [3].



# Cel i zakres pracy

W ramach niniejszej pracy została przedstawiona implementacja aplikacji narzędziowej z pogranicza biofizyki molekularnej i wirtualnej rzeczywistości. Program służący do interaktywnej eksploracji struktur biopolimerów w poszukiwaniu regionów optymalnych ze względu na jakość lokalnego dopasowania strukturalnego pomiędzy wzorcem (*ang. template structure*) i wybranymi regionami cząsteczki celu (*ang. target structure*). Jakość znalezionej dopasowania odwzorowuje zwrotna projekcja momentów sił w urządzeniu haptycznym.

Program został zrealizowany w formie wtyczki rozszerzającej możliwości pakietu PyMOL. Do swojego działania wykorzystuje urządzenia i metody wirtualnej rzeczywistości - urządzenie haptyczne Phantom Omni oraz bibliotekę Virtual Reality Peripheral Network - a także bioinformatyczne algorytmy wyszukujące optymalne lokalne dopasowania strukturalne (*ang. local structure alignment*), algorytm maksymalizujący stopień dopasowania struktur i minimalizujący wartość RMSD (algorytm Kabsch'a).

W pracy zostały także przedstawione podstawy teoretyczne leżące u jej podstaw, szczególne opisy wykorzystanego urządzenia i metod wirtualnej rzeczywistości, a także perspektywy dalszego rozwoju i wykorzystania praktycznego opracowanego oprogramowania.

Kod źródłowy stworzonego oprogramowania stanowi integralny załącznik do niniejszej pracy.



# Rozdział 1

## Podstawy teoretyczne

W tym rozdziale zaprezentowano teorię leżącą u podstaw niniejszej pracy. Skupiono się tutaj przede wszystkim na metodach dopasowania (uliniawiania) struktur chemicznych, sposobach optymalnego nakładania struktur i oceny jego jakości (algorytm Kabsch’a i RMSD) oraz zagadnieniach związanych z przekształceniami geometrycznymi (skalowanie, translacje i rotacje).

### 1.1. Metody dopasowania struktur chemicznych

Celem poszukiwania optymalnych metod dopasowania strukturalnego (*ang. structural alignment*) jest znalezienie homologii pomiędzy cząsteczkami polimerów lub ich fragmentami jedynie na podstawie kształtu, bez znajomości sekwencji.

Metody dopasowań strukturalnych pierwotnie odnosiły się do cząsteczek polipeptydów i białek jako podstawowych biopolimerów. Szybko jednak ich zastosowanie zostało rozszerzone także na kwasy nukleinowe, w szczególności niekodujący RNA gdyż posiada on ważne funkcje biologiczne (np. enzymatyczne) oraz analogiczne do polipeptydów formy drugo- i trzeciorzędowe.

Z uwagi na znacznie wyższą ewolucyjną trwałość struktury przestrzennej w porównaniu do sekwencji (zarówno aminokwasowej jak i nukleotydowej) poszukiwanie dopasowań strukturalnych często bywa dużo skuteczniejszą metodą znajdowania związków ewolucyjnych pomiędzy organizmami niż klasyczne uliniowanie sekwencji.

Istotnym aspektem tych metod jest efektywna ocena jakości dopasowań. Niestety nie ma jednej uniwersalnej miary podobieństwa struktur, istnieje jednak kilka dobrze opisanych algorytmów służących do ich szacowania. Najczęściej wykorzystywaną do tego celu metryką jest RMSD (*ang. root-mean-squared deviation*), szczegółowo opisana w dalszej części pracy.

Obliczenie dopasowania strukturalnego ponadto implikuje powstanie dopasowania sekwencyjnego pomiędzy przyrównywanymi merami w każdym z łańcuchów. Ocena takiego jednowymiarowego uliniowania sekwencji również może dać nam wiedzę o bliskości ewolucyjnej występującej pomiędzy strukturami.

Z powodu dużej złożoności biopolimerów zostało opracowanych wiele metod upraszczających ich reprezentacje do celów obliczeniowych. Przede wszystkim dąży się do rezygnacji z bezpośredniego rozpatrywania lokalizacji wszystkich atomów pochodzących z każdego meru na rzecz jedynie tych należących do szkieletu (*ang. backbone*) cząsteczki (np.  $\alpha$ -węgle aminokwasów czy pentozy kwasów nukleinowych), z pominięciem lub daleko idącym ograniczeniem roli łańcuchów bocznych.

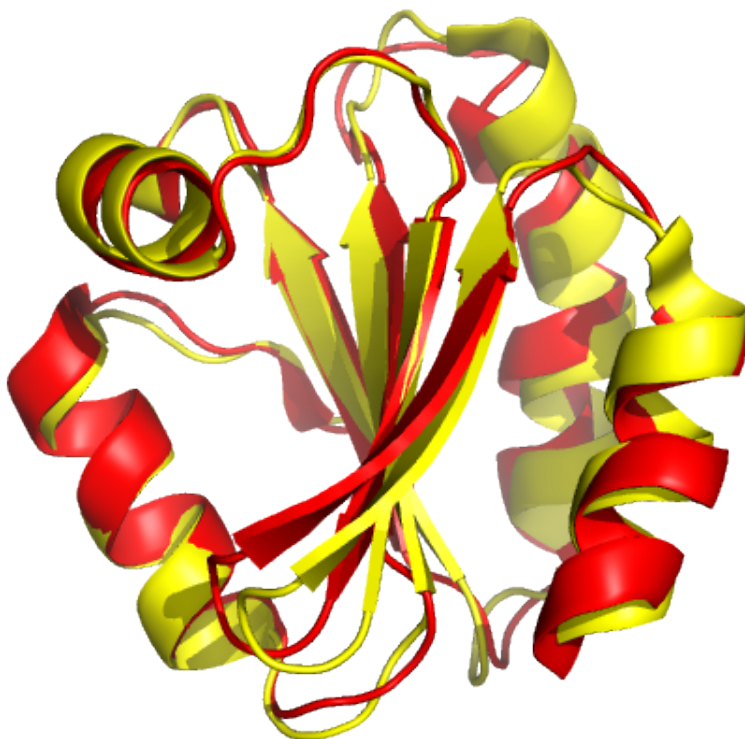
Metody dopasowania możemy podzielić na globalne, których celem jest porównywanie

całych struktur trzeciorzędowych oraz lokalne polegające na poszukiwaniu najlepszego dopasowania fragmentu cząsteczki wzorca (np. struktury drugorzędowej) do wybranego regionu (miejsca w obrębie cząsteczki celu) i ewentualna późniejsza ich kompozycja do dopasowania globalnego.

#### 1.1.1. Globalne dopasowanie struktur

*Globalne dopasowanie* polega na obliczaniu najlepszego nałożenia na siebie dwóch lub więcej struktur i ich superpozycji. Jest to iteracyjne podejście, które dąży do minimalizowania odległości pomiędzy atomami (lub wybranymi punktami charakteryzującymi dany mer cząsteczki) w każdym kolejnym kroku poprzez dokonywanie odpowiednich przesunięć i obrotów struktur względem siebie.

Globalne dopasowanie jest najskuteczniejsze gdy podobieństwo porównywanych struktur jest małe, aby obliczenia dążyły do optymalnego globalnie rezultatu. Jest to często stosowana metoda służąca do porównywania różnych konformacji tego samego polimeru, na przykład do oceny jakości algorytmów przewidujących strukturę białek lub porównywanie struktur analogicznych białek pochodzących z różnych organizmów.



Rysunek 1.1: Superpozycja dwóch struktur tioredoksyny (białko o długości 105 aminokwasów) ludzkiej (kolor czerwony) i pochodzącej z muszki owocowej *Drosophila melanogaster* (kolor żółty), RMSD=1.244Å

Wadą podejścia globalnego jest to, że jego wykorzystanie dla zróżnicowanych cząsteczek może być bezcelowe z powodu niemożności obliczenia miarodajnej wartości podobieństwa pomiędzy takimi strukturami. Innymi słowy obliczona wartość RMSD pomiędzy takimi cząsteczkami może być pozbawiona sensownej interpretacji. Wówczas należy rozważyć podejście lokalne.

### 1.1.2. Lokalne dopasowanie struktur

Jak już wspomniano, lokalne dopasowanie polega na poszukiwaniu w części lub całości cząsteczki celu (*ang. target*) występowania takich regionów, które „są podobne” do wybranych struktur zwanych wzorcami lub szablonami (*ang. template*). Przez podobieństwo należy tutaj rozumieć taką odpowiedniość pomiędzy ww. strukturami, że wyznaczona dla ich superpozycji wartość RMSD nie przekracza zadanego progu. Wzorcami mogą być na przykład struktury drugorzędowe, miejsca wiążące, całe domeny białkowe lub inne wybrane fragmenty cząsteczek których występowania poszukujemy.

Istnieje wiele opracowanych metod i algorytmów realizujących takie obliczenia. Do najpopularniejszych należy zaliczyć DALI[4], SSAP[5], CE[6], VAST[7], MATRAS[8], GANGSTA[9] i inne. Każda z nich ma inne podejście do dekompozycji struktury i sposobów poszukiwania lokalnych podobieństw.

Szczególną uwagę musimy jednak zwrócić na metodę lokalnych deskryptorów, zaproponowaną przez Krzysztofa Fidelisa [10][11] (jednego z twórców i organizatorów eksperymentu CASP). Stanowi ona podstawowy algorytm generujący dane wejściowe dla oprogramowania będącego przedmiotem niniejszej pracy. Metoda lokalnych deskryptorów w przeciwieństwie do metod liniowych, bazuje na lokalnym przestrzennym otoczeniu każdego aminokwasu. Metoda szczegółowo została opisana w cytowanych wyżej publikacjach.

Wynikiem przeprowadzonych uliniowień i superpozycji struktur jest indeks miejsc (wraz z ich miarą dopasowania) analizowanej struktury do których wzorzec jest podobny. Należy pamiętać, że jeden wzorzec może pasować do wielu regionów w obrębie tej samej cząsteczki, przez co może występować w wielu miejscach indeksu mapującego. Innymi słowy dzięki takiej operacji uzyskujemy kompletną mapę struktury celu z wyróżnionymi regionami, których superpozycja z wzorcem daje stopień podobieństwa nie gorszy od zadanego.

Indeks miejsc jest esportowany do formatu akceptowanego przez opisane dalej oprogramowanie w formie pliku mapującego (*ang. mapping file*).

## 1.2. RMSD i algorytm Kabsch’a

Efektywna ocena podobieństwa strukturalnego jest jednym z kluczowych elementów procesu dopasowania. W bioinformatyce istnieje kilka sposobów oszacowania tej wartości: obok GDT (*ang. global distance test*) i TM-score (*ang. template modeling score*) [13] najpopularniejsza i stosunkowo prosta w zastosowaniu jest miara odchylenia średniokwadratowego - RMSD (*ang. root-mean-squared deviation*) [12].

Ocena podobieństwa metodą RMSD polega na obliczeniu średniokwadratowej odległości pomiędzy współrzędnymi odpowiadających sobie atomów (lub innych punktów charakterystycznych) zawartych w strukturze wzorca (*ang. template*) i cząsteczce celu (*ang. target*). W tym przypadku jednostką najczęściej jest Å(angstrom).

$$\text{RMSD}(p, q) = \sqrt{\frac{1}{N} \sum_{i=1}^N \|p_i - q_i\|^2}$$

gdzie:

$p$  - wektor punktów struktury wzorca

$q$  - wektor punktów wybranego regionu w strukturze celu

$N$  - długość wektorów współrzędnych  $p$  i  $q$

O ile same obliczenia są trywialne, to dobór danych wejściowych do algorytmu RMSD może stanowić poważne wyzwanie. Tutaj pochodzą one z zewnętrznej aplikacji i są wynikiem przeprowadzonej wcześniej procedury lokalnego dopasowania wzorca do struktury celu.

RMSD jest jedną podstawowych miar podobieństwa strukturalnego wykorzystywanych w bioinformatyce. Sama w sobie ma dla nas jednak jedynie wartość informacyjną. Wiadomym jest, że dwie struktury są tym bardziej do siebie podobne im bliższa zeru jest wartość RMSD. Nie oznacza to jednak, że wysoka wartość RMSD implikuje brak lub niskie podobieństwo pomiędzy strukturami. Może być dokładnie odwrotnie: dwie identyczne struktury są względem siebie przesunięte i obrócone, co znacznie zwiększa wartość metryki.

Jednym ze znanych sposobów minimalizacji jej wartości jest optymalna translacja i rotacja struktury wzorca nad wybranym regionem ze struktury celu w taki sposób aby zostały one jak najlepiej na siebie nałożone.

Istnieje kilka metod wyznaczania optymalnych transformacji, jedną z popularniejszych opisał opisał w 1976 roku w swojej pracy Wolfgang Kabsch [14][15]. Celem działania algorytmu Kabsch’a jest wyznaczenie macierzy translacji i rotacji takich, aby możliwie najlepiej nałożyć na siebie dwa obiekty przestrzenne, tutaj dwa fragmenty struktur chemicznych.

Algorytm Kabsch’a startuje z dwoma wektorami współrzędnych  $p$  i  $q$  o długości  $N$ :

$$p = \begin{pmatrix} x_{p1} & y_{p1} & z_{p1} \\ x_{p2} & y_{p2} & z_{p2} \\ \vdots & \vdots & \vdots \\ x_{pN} & y_{pN} & z_{pN} \end{pmatrix}$$

$$q = \begin{pmatrix} x_{q1} & y_{q1} & z_{q1} \\ x_{q2} & y_{q2} & z_{q2} \\ \vdots & \vdots & \vdots \\ x_{qN} & y_{qN} & z_{qN} \end{pmatrix}$$

gdzie:

$p$  wektor punktów struktury wzorcowej o długości  $N$

$q$  wektor punktów wybranego regionu ze struktury celu o długości  $N$

Algorytm Kabsch’a składa się z 3 kroków (<http://cnx.org/contents/HV-RsdwL@23/Molecular-Distance-Measures>):

### 1. Translacja

W pierwszym kroku należy dokonać obliczenia centroidów ( $Cp$  i  $Cq$ ) obu struktur i dokonać przesunięcia struktury wzorca o wektor  $\vec{T}$  rozpięty pomiędzy tymi punktami tak, aby nałożyły się one na siebie. Do obliczenia centroidu można posłużyć się wzorem na średnią arytmetyczną wartości poszczególnych współrzędnych:

$$Cp = (Cp_x, Cp_y, Cp_z)$$

gdzie:

$$Cp_x = \frac{1}{N} \sum_{i=1}^N x_{pi}$$

$$Cp_y = \frac{1}{N} \sum_{i=1}^N y_{pi}$$

$$Cp_z = \frac{1}{N} \sum_{i=1}^N z_{pi}$$

oraz

$$Cq = (Cq_x, Cq_y, Cq_z)$$

gdzie:

$$Cq_x = \frac{1}{N} \sum_{i=1}^N x_{qi}$$

$$Cq_y = \frac{1}{N} \sum_{i=1}^N y_{qi}$$

$$Cq_z = \frac{1}{N} \sum_{i=1}^N z_{qi}$$

zatem wektor translacji  $\vec{T}$  taki, że:

$$\vec{T} = |Cp - Cq| = (|Cp_x - Cq_x|, |Cp_y - Cq_y|, |Cp_z - Cq_z|)$$

możemy użyć do wykonania przesunięcia wszystkich punktów w  $p$  i  $q$ :

$$p' = p + \vec{T}$$

## 2. Macierz kowariancji

Po wykonanym przesunięciu należy obliczyć zależność liniową pomiędzy współrzędnymi wektorów  $p'$  i  $q$ . Poprawne wyliczenie macierzy kowariancji  $A$  będzie także stanowiło podstawę do ustalenia optymalnej macierzy rotacji (w kolejnym kroku):

$$A = cov(p', q)$$

lub równoważnie w zapisie macierzowym:

$$A = p' q^T$$

## 3. Optymalna macierz rotacji

Optymalna macierz rotacji powstaje z *dekompozycji według wartości szczególnych* macierzy  $A$ . SVD (*ang. singular value decomposition*) to taki rozkład zadanej macierzy na trzy specyficzne macierze  $U$ ,  $\Sigma$  oraz  $V$ , że zachodzi zależność:

$$A = U \Sigma V^T$$

gdzie:

$U$  i  $V$  to macierze ortogonalne (takie, że  $U^{-1} = U^T$  oraz  $V^{-1} = V^T$ )

$\Sigma$  macierz diagonalna (taka, że na przekątnej mamy nieujemne liczby rzeczywiste, będące wartościami szczególnymi macierzy  $A$ )

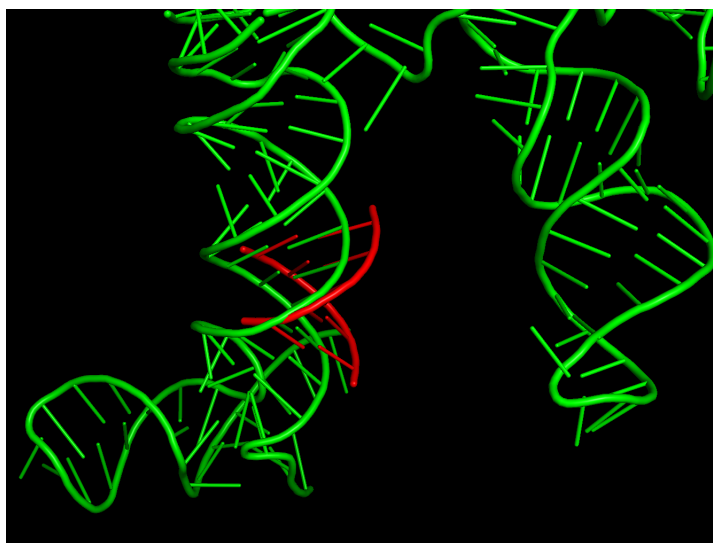
Sprawdzając znak wartość wyznacznika:

$$s = sign(det(VW^T))$$

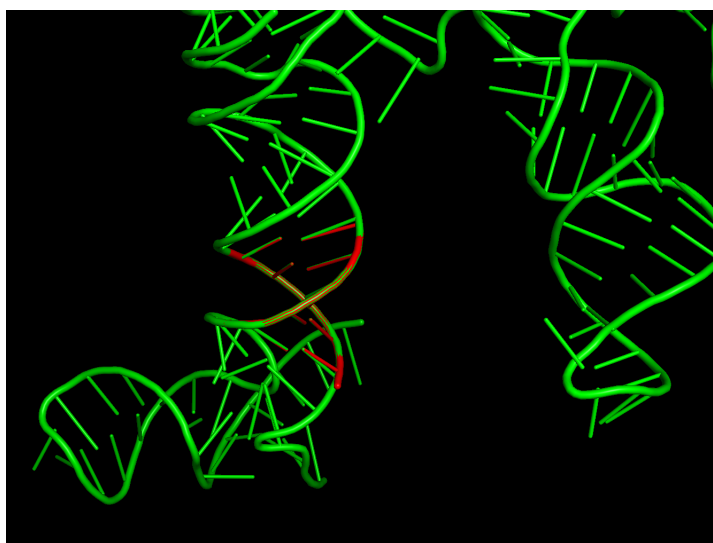
upewniamy się, że operujemy w ramach prawoskrętnego układu współrzędnych. Ostatecznie uzyskujemy optymalną macierz rotacji  $R$ :

$$R = V \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{bmatrix} U^T$$

Zastosowanie translacji i rotacji wyznaczonych zgodnie z algorytmem Kabsch'a prezentują poniższe wizualizacje. Prezentują one różne wartości RMSD dla superpozycji tych samych struktur przed i po zastosowaniu algorytmu:



Rysunek 1.2: Przykładowe dopasowanie struktury wzorca (czerwona) do podobnego regionu struktury celu (zielona), RMSD=3.985



Rysunek 1.3: Przykładowe dopasowanie struktury wzorca (czerwona) do podobnego regionu struktury celu (zielona), RMSD=0



### 1.3. Przekształcenia geometryczne i ich reprezentacje

Przekształcenia geometryczne w przestrzeni trójwymiarowej są elementarnymi operacjami używanymi w niniejszej pracy [16]. Zaliczamy do nich przede wszystkim skalowanie, translacje i rotacje. Wszystkie te operacje można praktycznie zrealizować na kilka sposobów. Ze względu na to, że mogą operować na milionach punktów w przestrzeni trójwymiarowej (pikseli), optymalne metody numeryczne które je realizują są niezmiernie ważne i muszą w maksymalnym stopniu wykorzystywać możliwości oferowane przez komputerowe jednostki obliczeniowe i procesory graficzne.

Współpraca bibliotek programistycznych pochodzących od różnych dostawców może wymagać wielu przekształceń pomiędzy różnymi formatami opisu transformacji. Szczególną uwagę należy zwrócić na konwersję orientacji wskaźnika urządzenia haptycznego z kwaternionów - postaci dostarczanej przez bibliotekę VRPN - na oś i kąt obrotu (*ang. axis-angle rotation*) - akceptowany przez PyMOL - i odwrotnie.

Z uwagi na konieczność dostosowania transformacji do wykonywania na współczesnych komputerach niezbędne było opracowanie efektywnych obliczeniowo metod numerycznych i reprezentacji tych operacji. Bezpośrednia ich implementacja byłaby dość kłopotliwa gdyż nie dość, że każda transformacja musiałaby zostać wykonana oddzielnie (sekwencyjnie), to przede wszystkim nie byłoby to optymalne obliczeniowo rozwiązanie. Do tego celu wybrano ujednolicone narzędzia rozwiązujące powyższe problemy: elementarne macierze transformacji i współrzędne jednorodne.

*Elementarne macierze transformacji* stanowią łatwe narzędzie matematyczne do operowania przekształceniami geometrycznymi. Każda z transformacji może zostać opisana jako prosta operacja macierzowa (dodawanie albo mnożenie) modyfikująca zbiór punktów w przestrzeni.

Często pojawia się sytuacja w której kilka przekształceń geometrycznych chcemy wykonać jednocześnie. Można sobie wyobrazić przypadek w którym obiekt chcemy jednocześnie przesunąć i dokonać jego obrotu. Niestety stosowanie „surowych” elementarnych macierzy transformacji nie umożliwia wykonania takich operacji w tym samym czasie, dopiero współrzędne jednorodne pozwoliły w pełni wyeliminować ten problem.

*Współrzędne jednorodne* są sposobem reprezentacji punktów przestrzeni  $n$ -wymiarowej za pomocą układu  $n + 1$  współrzędnych. Zostały one wprowadzone przez niemieckiego matematyka Augusta Möbiusa w 1827 roku i opisane w jego pracy [17].

Współrzędne jednorodne to narzędzie matematyczne stanowiące pewne udoskonalenie elementarnych macierzy transformacji. Umożliwiają wykonywanie wielu przekształceń jednocześnie i za pomocą jednej operacji macierzowej - mnożenia. Współrzędne jednorodne dają możliwość skumulowania wszystkich transformacji jakie chcemy wykonać na strukturze przestrzennej w jednej odpowiednio zbudowanej macierzy. W dzisiejszych czasach zostały one docenione w wielu dziedzinach nie tylko bezpośrednio związanych z grafiką komputerową ale także w robotyce czy biofizyce.

Poniżej szczegółowo opisane zostały wykorzystane w pracy transformacje: skalowanie, translacja oraz rotacja. Dla każdej z nich podane zostały ich reprezentacje za pomocą elementarnych macierzy transformacji oraz współrzędnych jednorodnych. Dodatkowo dla rotacji została przedstawiona reprezentacja w kwaternionach.

#### 1.3.1. Skalowanie

Skalowanie to operacja polegająca na mnożeniu współrzędnych obiektu określonego w przestrzeni trójwymiarowej przez współczynniki skalowania. Współczynniki skalowania są dodat-

nią liczbą rzeczywistą  $S$  albo dodatnio określonym wektorem liczb rzeczywistych  $\vec{S}$ . W ogólności możemy skalować wszystkie składowe wektora współrzędnych niezależnie przez stosowanie wektora o różnych współczynnikach  $\vec{S}$ , jednak dla potrzeb niniejszej pracy ograniczymy się do mnożenia wektora współrzędnych wyłącznie przez skalar  $S$ .

Geometryczna intuicja stojąca za operacją skalowania polega na takim przekształceniu współrzędnych danego obiektu, że w zależności od wartości współczynnika skalującego  $S$  może on zostać:

- niezmienny, gdy  $S = 1$
- pomniejszony proporcjonalnie do  $S$ , gdy  $0 < S < 1$
- powiększony proporcjonalnie do  $S$ , gdy  $S > 1$

Współrzędne dowolnego punktu  $P = (x, y, z)$  lub  $\vec{P} = [x, y, z]^T$  po operacji skalowania o współczynnik  $S$  są równe  $P' = (x', y', z')$ , gdzie:

$$\begin{aligned}x' &= Sx \\y' &= Sy \\z' &= Sz\end{aligned}$$

Reprezentacja skalowania za pomocą elementarnych macierzy transformacji polega na stworzeniu takiej macierzy  $\hat{S}$ , że wartość współczynnika  $S$  zostanie wpisane do niej w następujący sposób:

$$\hat{S} = \begin{bmatrix} S & 0 & 0 \\ 0 & S & 0 \\ 0 & 0 & S \end{bmatrix}$$

zatem w zapisie macierzowym otrzymujemy:

$$\vec{P}' = \hat{S}\vec{P}$$

$$\vec{P}' = \begin{bmatrix} S & 0 & 0 \\ 0 & S & 0 \\ 0 & 0 & S \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} Sx \\ Sy \\ Sz \end{bmatrix}$$

Aby przedstawić operację skalowania we współrzędnych jednorodnych należy stworzyć nową macierz  $\bar{S}$  o rozmiarze o 1 większym niż macierz elementarna  $\hat{S}$  oraz nowy wektor  $\vec{P}$  o długości o 1 większej niż  $\vec{P}$ , taki że:

$$\bar{S} = \begin{bmatrix} S & 0 & 0 & 0 \\ 0 & S & 0 & 0 \\ 0 & 0 & S & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \vec{P} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

zatem:

$$\vec{P}' = \bar{S}\vec{P}$$

$$\vec{P}' = \begin{bmatrix} S & 0 & 0 & 0 \\ 0 & S & 0 & 0 \\ 0 & 0 & S & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} Sx \\ Sy \\ Sz \\ 1 \end{bmatrix}$$

### 1.3.2. Translacje

Translacje to kolejna elementarna operacja przekształcenia geometrycznego. Translacją nazywamy takie przekształcenie, które przesuwa każdy punkt zbioru określony w przestrzeni o dowolny wektor  $\vec{T}$ . W odróżnieniu od skalowania, w przypadku przesunięć w przestrzeni do wektora współrzędnych dodajemy wektor współczynników (liczb rzeczywistych) przesunięcia  $\vec{T}$ . W niniejszej pracy operacja przesunięcia wykonywana jest przy każdej zmianie położenia wskaźnika urządzenia haptycznego oraz podczas wykonywania operacji nałożenia i dopasowania struktur.

Dla zadanego punktu  $P = (x, y, z)$  lub  $\vec{P} = [x, y, z]^T$  efektem operacji przesunięcia o wektor  $\vec{T} = [t_x, t_y, t_z]^T$  są współrzędne  $\vec{P}' = (x', y', z')$ :

$$\begin{aligned}x' &= x + t_x \\y' &= y + t_y \\z' &= z + t_z\end{aligned}$$

Z powyższego jasno wynika, że translacje można reprezentować jako operację sumy dwóch wektorów:

$$\vec{P}' = \vec{P} + \vec{T}$$

$$\vec{P}' = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ z + t_z \end{bmatrix}$$

Translacje można przedstawić także za pomocą współrzędnych jednorodnych. Wymaga to stworzenia macierzy  $\bar{T}$  oraz wektora  $\dot{\vec{P}}$ , takich że:

$$\bar{T} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \dot{\vec{P}} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

zatem:

$$\vec{P}' = \bar{T} \dot{\vec{P}}$$

$$\vec{P}' = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ z + t_z \\ 1 \end{bmatrix}$$

### 1.3.3. Rotacje i kwaterniony

Trzecią elementarną transformacją jest rotacja, która w odróżnieniu od skalowania czy translacji jest operacją bardziej złożoną. Należy zwrócić uwagę, że kolejność wykonywania rotacji ma znaczenie ( $R_x R_y \neq R_y R_x$ ), po drugie samych reprezentacji rotacji jest co najmniej kilka. Najpopularniejszym z nich są obroty o zadany kąt wokół jednej z osi układu współrzędnych lub arbitralnej osi obrotu (*ang. axis-angle*).

Współrzędne punktu  $P' = (x', y', z')$  będącego wynikiem rotacji punktu  $P = (x, y, z)$  wokół poszczególnych osi ( $OX$ ,  $OY$ ,  $OZ$ ) układu współrzędnych o zadany kąt (odpowiednio  $\alpha$ ,  $\beta$  i  $\gamma$ ) wyrażają się następująco:

rotacja  $P$  wokół osi  $OX$  o kąt  $\alpha$ :

$$\begin{aligned}x' &= x \\y' &= y \cos \alpha - z \sin \alpha \\z' &= y \sin \alpha + z \cos \alpha\end{aligned}$$

rotacja  $P$  wokół osi  $OY$  o kąt  $\beta$ :

$$\begin{aligned}x' &= z \sin \beta + x \cos \beta \\y' &= y \\z' &= x \cos \beta - z \cos \beta\end{aligned}$$

rotacja  $P$  wokół osi  $OZ$  o kąt  $\gamma$ :

$$\begin{aligned}x' &= x \cos \gamma - y \sin \gamma \\y' &= x \sin \gamma + y \cos \gamma \\z' &= z\end{aligned}$$

Elementarne macierze przekształceń w tym przypadku wyglądają następująco:

$$\hat{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix}$$

$$\hat{R}_y(\beta) = \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix}$$

$$\hat{R}_z(\gamma) = \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

gdzie:

$\hat{R}_x(\alpha)$  obrót wokół osi  $OX$  o kąt  $\alpha$

$\hat{R}_y(\beta)$  obrót wokół osi  $OY$  o kąt  $\beta$

$\hat{R}_z(\gamma)$  obrót wokół osi  $OZ$  o kąt  $\gamma$

Jak wszystkie inne transformacje, rotacje również można przedstawić we współrzędnych jednorodnych. Odpowiednie macierze mają wówczas następującą postać:

$$\bar{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha & 0 \\ 0 & \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\bar{R}_y(\beta) = \begin{bmatrix} \cos \beta & 0 & \sin \beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \beta & 0 & \cos \beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\bar{R}_z(\gamma) = \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 & 0 \\ \sin \gamma & \cos \gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotacje reprezentowane przez współrzędne jednorodnie można dowolnie ze sobą łączyć poprzez mnożenie odpowiednich macierzy, należy jednak mieć na uwadze fakt (jak już wcześniej wspomniano), że rotacje nie są przemienne - kolejność wykonywanych operacji ma znaczenie.

Rotacje posiadają jeszcze jedną być może najważniejszą z punktu widzenia programisty reprezentację: kwaterniony [18], które są one strukturą algebraiczną stanowiącą rozszerzenie liczb zespolonych. Kwaterniony zostały wprowadzone przez Williama Hamiltona [19], który poszukiwał wygodnego sposobu opisu mechaniki w przestrzeni trójwymiarowej.

*Kwaternion*  $q$  to taka czwórka liczb rzeczywistych  $x, y, z$  i  $w$  spełniająca równanie:

$$q = w + xi + yj + zk$$

gdzie:

$i, j, k$  - współczynniki urojone takie, że  $i^2 = j^2 = k^2 = ijk = -1$

$w$  - część skalarna kwaterniona

$x, y, z$  - część wektorowa kwaterniona

Kwaterniony mogą stanowić alternatywną wobec współrzędnych jednorodnych lub macierzy transformacji formę opisu rotacji. Możemy z powodzeniem przeprowadzać konwersję pomiędzy różnymi reprezentacjami rotacji.

Dzięki kwaternionom można przedstawić rotację wokół arbitralnie wybranej osi (wektora) obrotu. Załóżmy, że chcemy stworzyć kwaternion  $q$  reprezentujący obrót o kąt  $\alpha$  wokół wektora  $\vec{v} = [v_x, v_y, v_z]$ . Kwaternion taki tworzymy w następujący sposób:

$$\begin{aligned} w &= \cos \frac{\alpha}{2} \\ x &= v_x \sin \frac{\alpha}{2} \\ y &= v_y \sin \frac{\alpha}{2} \\ z &= v_z \sin \frac{\alpha}{2} \end{aligned}$$

wówczas:

$$q = \cos \frac{\alpha}{2} + v_x i \sin \frac{\alpha}{2} + v_y j \sin \frac{\alpha}{2} + v_z k \sin \frac{\alpha}{2}$$

Kwaterniony mogą reprezentować nie tylko rotację (operację obrotu bryły) ale także orientację (stan bryły). Biblioteka VRPN (wykorzystana w niniejszej pracy) wydaje informację o bieżącej orientacji (stanie) wskaźnika, natomiast pakiet PyMOL akceptuje jedynie rotacje (zmiany orientacji).

Prawidłowe wykonanie rotacji pomiędzy dwoma kolejnymi kwaternionami  $q_n$  i  $q_{n+1}$  niosącymi informację o orientacji (stanie) polega na zastosowaniu kwaterniona odwrotnego  $q_n^{-1}$  (powracającego strukturę do orientacji początkowej, zerowej), a następnie kwaterniona  $q_{n+1}$ . Kwaternion odwrotny jest zdefiniowany następująco:

$$q^{-1} = \frac{w - xi - yj - zk}{w^2 + x^2 + y^2 + z^2}$$

Taki ciąg operacji należy wykonywać cyklicznie w miarę odbierania kolejnych informacji o orientacji wskaźnika.

## 1.4. Pole siłowe

Biblioteka VRPN dostarcza kilku metod sterowania zwrotną projekcją sił (szerszy opis w dalszej części). W tej pracy wykorzystano tzw. metodę pola siłowego (*ang. force field*), która podobnie jak w pozostałych przypadkach (np. symulacja powierzchni lub brył) realizowana jest przez lokalną aproksymację. Jej wywołanie polega na dostarczeniu funkcji trzech parametrów: punktu zaczepienia (*ang. origin*), wektora siły oraz macierzy Jacobiego.

Punktem zaczepienia wektora siły jest w tym przypadku środek geometryczny wzorcowej struktury chemicznej obliczany jako średnia arytmetyczna wartości poszczególnych współrzędnych.

Wektor siły  $\vec{F}$  to wektor wskazujący kierunek i zwrot siły działającej na wskaźnik. W tym przypadku jest on rozpięty pomiędzy środkami geometrycznymi struktury wzorcowej, a najbliższym „podobnym” regionem - wyznaczonym przez algorytm lokalnego dopasowania strukturalnego. Zwrot siły jest skierowany w stronę owego regionu. Do dalszych rozważań możemy traktować wektor  $\vec{F}$  jako iloczyn wektora jednostkowego (wersora)  $\dot{F}$  oraz skalarnej wartości siły  $f$ :

$$\vec{F} = \dot{F}f = [\vec{F}_x f, \vec{F}_y f, \vec{F}_z f]$$

Macierz Jacobiego  $J_{\vec{F}}$  jest macierzą zbudowaną z pochodnych cząstkowych pierwszego rzędu funkcji definiującej pole siłowe:

$$J_{\vec{F}} = \begin{bmatrix} \frac{\delta \vec{F}_x}{\delta x} & \frac{\delta \vec{F}_y}{\delta x} & \frac{\delta \vec{F}_z}{\delta x} \\ \frac{\delta \vec{F}_x}{\delta y} & \frac{\delta \vec{F}_y}{\delta y} & \frac{\delta \vec{F}_z}{\delta y} \\ \frac{\delta \vec{F}_x}{\delta z} & \frac{\delta \vec{F}_y}{\delta z} & \frac{\delta \vec{F}_z}{\delta z} \end{bmatrix}$$

Tutaj różniczkowaniu poddawane są poszczególne składowe wektora siły  $\vec{F}$ , w wyniku czego macierz upraszcza się do postaci:

$$J_{\vec{F}} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & f \end{bmatrix}$$

Wyznaczenie powyższych wartości jest niezbędne do prawidłowego wysterowania momentów sił wskaźnika haptycznego za pośrednictwem biblioteki VRPN.

## Rozdział 2

# Urządzenie haptyczne Sensable Phantom Omni

Nadrzędnym celem wirtualnej rzeczywistości jest możliwie wierne odtworzenie świata rzeczywistego lub fikcyjnego w formie programu komputerowego czy multimedialnej prezentacji. Ponadto metody i narzędzia wirtualnej rzeczywistości umożliwiają użytkownikowi wchodzenie w czynną interakcję z tak wykreowanymi scenami. Obecny stan rozwoju techniki pozwala na tworzenie wydajnych urządzeń klasy HCI (*ang. human computer interface*) będących interfejsami pomiędzy światem realnym a wirtualnym. Przykładem takiego urządzenia jest Phantom Omni, opracowane przez firmę SensAble (obecnie Geomagic).



Rysunek 2.1: Urządzenie haptyczne SensAble Phantom Omni

Niniejsza praca w znacznym stopniu opiera się na wykorzystaniu tych urządzeń do odwzorowywania rzeczywistych ruchów ręki użytkownika w świecie wirtualnym. Pracownie Zakładu Biofizyki Wydziału Fizyki Uniwersytetu Warszawskiego są wyposażone w tego typu urządzenia, zostały one udostępnione autorowi celem realizacji niniejszej pracy.

## 2.1. Opis urządzenia

Urządzenie haptyczne (*ang. haptic device*) Sensable Phantom Omni jest trójwymiarowym wskaźnikiem o 6 stopniach swobody wskazywania pozycji i orientacji. Urządzenie ma także możliwość programowego sterowania zwrotną projekcją (*ang. force feedback*) momentów sił w trzech stopniach swobody (sterowanie pozycją wskaźnika). Ponadto w urządzeniach dostępne są dwa przyciski monostabilne do dowolnego wykorzystania.

Phantom Omni jest urządzeniem znajdującym zastosowanie w różnego rodzaju aplikacjach działających na styku świata wirtualnego z rzeczywistym. Umożliwia ono użytkownikowi wchodzenie w interakcję z cyfrowym światem za pomocą zmysłu dotyku. Dzięki dostarczonym przez producenta sterownikom i bogatemu zestawowi bibliotek OpenHaptics i narzędzi programistycznych możliwe jest tworzenie oprogramowania w pełni wykorzystującego możliwości oferowane przez urządzenie.

Specyfikację techniczną urządzenia prezentuje poniższa tabela [20]:

Wymiary pola roboczego	szerkość: 160 mm głębokość 70 mm wysokość 120 mm
Rozdzielczość	450 dpi ( $\sim 0.055$ mm)
Maksymalny moment obrotowy	3.3 Nm
Szttywność	oś X: 1.26 N/mm oś Y: 2.31 N/mm oś Z: 1.02 N/mm
Dane o pozycji i orientacji	osi X, Y i Z kąty $\alpha$ , $\beta$ i $\gamma$ (6 stopni swobody)
Zwrotna projekcja sił	osi X, Y i Z (3 stopnie swobody)
Interfejs	IEEE-1394a Ethernet

## 2.2. Wymagania sprzętowe

Urządzenia Sensable Phantom Omni w które wyposażone są Pracownie Biofizyki F UW posiadają dwa rodzaje interfejsów: Ethernet oraz FireWire (IEEE-1394a).

Interfejs Ethernet jest powszechnie spotykany w większości komputerów i systemy operacyjne bezproblemowo radzą sobie z jego obsługą. Użycie urządzenia Phantom Omni z interfejsem sieciowym wiąże się jednak z instalacją dedykowanych sterowników oraz pakietu OpenHaptics v3.0, które jak podaje producent są wspierają jedynie systemy do Windows 8 włącznie, co uniemożliwia korzystanie z nich na nowszych platformach.

FireWire jest standardem łącza szeregowego opracowanym w 1995 roku, który umożliwia szybką transmisję danych. Został on zaprojektowany przede wszystkim do szybkiego przesyłania danych o dużym rozmiarze, zatem jest często wykorzystywany przez producentów sprzętu multimedialnego. Jednak w związku z systematycznym wycofywaniem się (od 2011 roku) producentów sprzętu i oprogramowania ze wspierania interfejsu FireWire, korzystanie z urządzeń w niego wyposażonych rodzi wiele problemów. Użytkownik jest zmuszony do poszukiwania dedykowanych (spełniających specyficzne wymagania producenta) kart rozszerzeń do stacji roboczych mających obsługiwać urządzenia - na dodatek firma SenseAble zaleca



korzystanie wyłącznie z kontrolerów IEEE-1394a opartych o chipset firmy NEC lub VIA co może dodatkowo utrudnić proces wdrażania urządzeń przy stanowisku pracy.

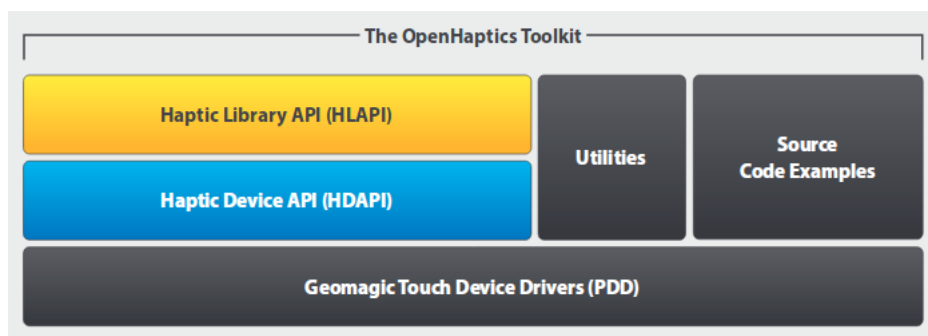
Powyższe trudności powodują, że sama instalacja i uruchomienie urządzenia staje się dość kłopotliwa. Stacje robocze korzystające z urządzeń Phantom Omni w Pracowniach Biofizyki F UW działają na systemach operacyjnych CentOS w wersji 6.0 ze zmodyfikowanym jądrem GNU/Linux, który pozwalał na stosunkowo stabilne działanie urządzenia.

Wraz ze sterownikami i pakietem OpenHaptics v3.0 w pracowniach uruchomiony jest framework Virtual Reality Peripheral Network (opisany w dalszej części pracy), dostarczający uniwersalną i łatwą w użyciu warstwę API nad wieloma rodzajami urządzeń klasy HCI.

Proces instalacji i konfiguracji urządzenia SensAble Phantom Omni nie był przedmiotem niniejszej pracy. Autor pracy miał do dyspozycji przygotowane stanowisko pracy, wyposażone w uruchomione urządzenie haptyczne, zainstalowany zestaw sterowników, bibliotek i pakiet VRPN.

## 2.3. OpenHaptics Toolkit v3.0

OpenHaptics Toolkit v3.0 jest bogatym pakietem oprogramowania dostarczanego przez producenta urządzenia. Zawiera on zestaw sterowników, bibliotek, narzędzi i przykładowych kodów źródłowych ułatwiających programiście wdrożenie rozwiązań haptycznych w dowolnym programie komputerowym. Biblioteki programistyczne dają dostęp do szerokiego zakresu niskopoziomowych funkcji urządzenia Phantom Omni tworząc przyjazną dla programisty warstwę abstrakcji [21][22].



Rysunek 2.2: Architektura pakietu OpenHaptics

Składnia bibliotek programistycznych pakietu OpenHaptics Toolkit jest wzorowana na składni biblioteki OpenGL i umożliwia tworzenie programów w językach C oraz C++.

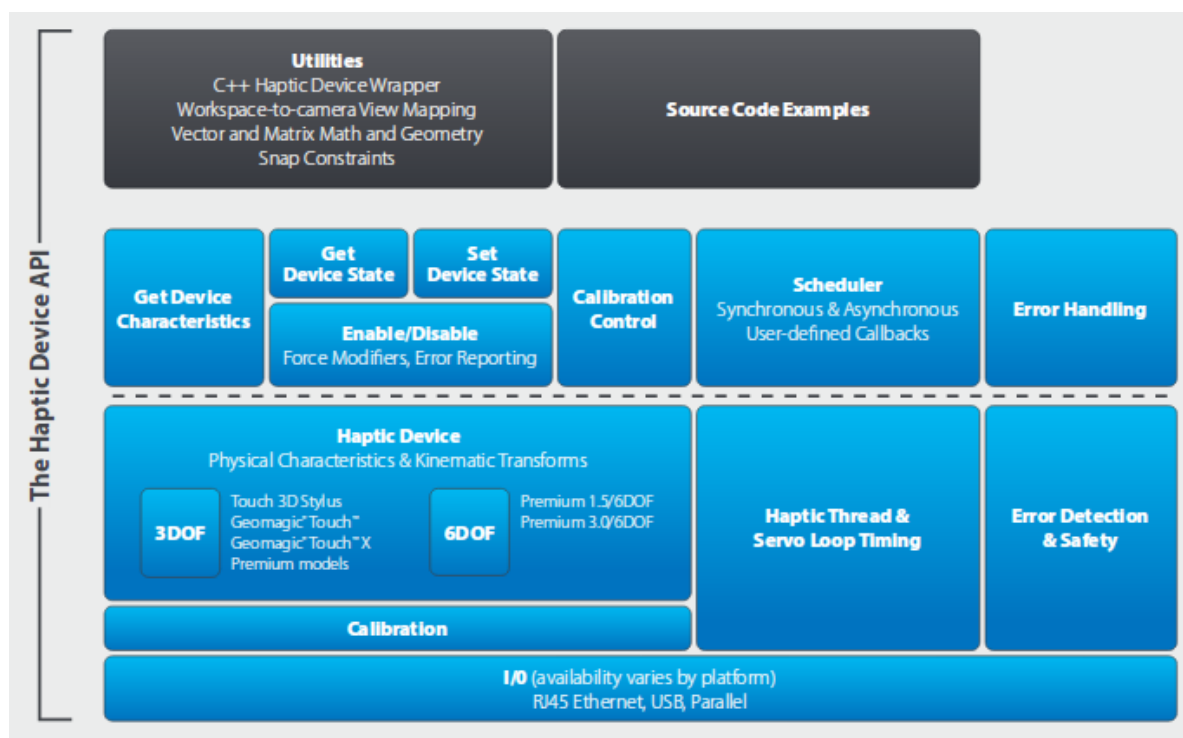
### 2.3.1. Phantom Device Drivers

Producent oprócz Phantom Omni dostarcza także innego rodzaju urządzenia haptyczne, skierowane do innych grup odbiorców. Phantom Device Drivers (PDD) to zestaw sterowników do wszystkich urządzeń haptycznych oferowanych przez producenta.

### 2.3.2. Haptic Device API

Haptic Device API (HDAPI) to niskopoziomowy interfejs programistyczny umożliwiający bezpośredni dostęp do wszystkich funkcji związanych z urządzeniami haptycznymi.

Poniższy diagram przedstawia kompletną architekturę modułu HDAPI:



Rysunek 2.3: Architektura HDAPI

Poza bardziej zaawansowanymi funkcjami związanymi z takimi aspektami jak wielowątkowość czy wywołania synchroniczne i asynchroniczne, HDAPI zwraca dane o pozycji i orientacji, umożliwia sterowanie sprzężeniem zwrotnym czy wyzwalanie funkcji (*ang. callback*) w odpowiedzi na wciskanie wbudowanych przycisków. HDAPI dostarcza także rozbudowany interfejs programistyczny systemu detekcji i przechwytywania błędów, kalibracji urządzenia oraz zestaw przykładowych kodów źródłowych demonstrujących wykorzystanie poszczególnych jego funkcji.

### 2.3.3. Haptic Library API

Haptic Library API (HLAPI) to biblioteki wysokopoziomowego interfejsu programistycznego zaprojektowane głównie pod kątem zgodności z OpenGL API. HLAPI umożliwia daleko idącą integrację z istniejącym kodem i bytami (strukturami, funkcjami, itp.) pochodzącymi z OpenGL. Ponadto ułatwia współpracę z zewnętrznymi bibliotekami dostarczającymi funkcjonalności obliczeń fizycznych (dynamika, detekcja kolizji, itp.).

W odróżnieniu od prostego sterowania momentami sił działającymi na urządzenie (jak w przypadku HDAPI), tutaj możemy definiować bardziej złożone obiekty przestrzenne (bryły, powierzchnie, pola siłowe) na które urządzenie haptyczne może oddziaływać. Również sposób reprezentacji sił dostarczony przez HLAPI jest bardziej rozbudowany, do dyspozycji programisty jest symulacja lepkości, sprężyny, grawitacji czy tarcia.

Podobnie jak w przypadku HDAPI programista dostaje bogaty zestaw przykładowych kodów źródłowych, umożliwiających natychmiastowe przetestowanie interesujących rozwiązań.

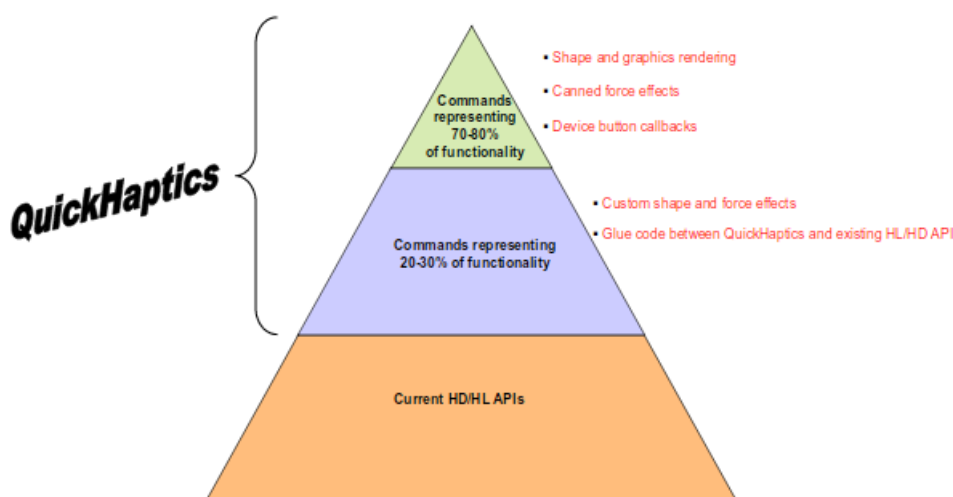
Poniższy diagram przedstawia kompletną architekturę modułu HLAPI:



Rysunek 2.4: Architektura HLAPI

### 2.3.4. QuickHaptics Micro API

QuickHaptics Micro API jest rozwiązaniem działającym „na szczycie piramidy”, OpenHaptics Toolkit.



Rysunek 2.5: QuickHaptics Micro API

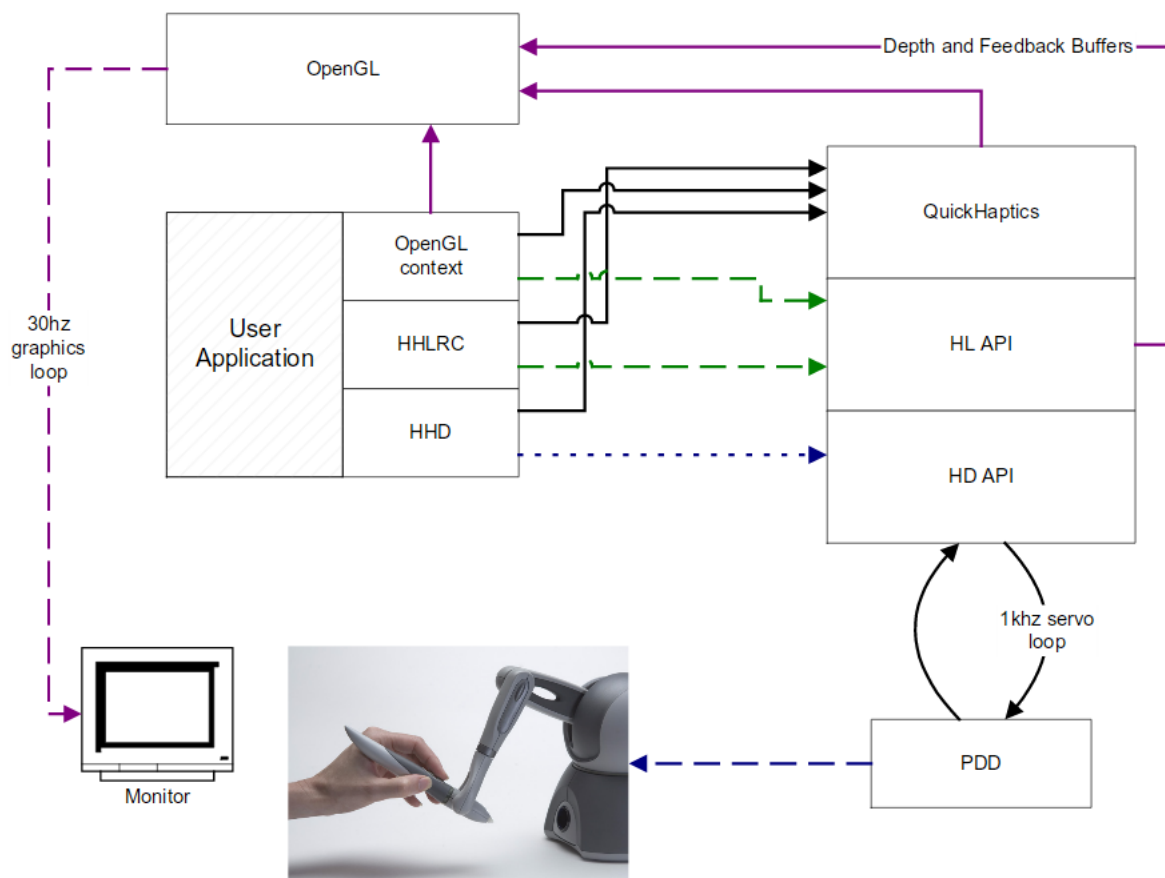
Zgodnie z zapewnieniami producenta QuickHaptics dostarcza jeszcze szybszych i łatwiejszych rozwiązań do tworzenia aplikacji wykorzystujących możliwości jego urządzeń. QuickHaptics Micro API stanowi kolejny, jeszcze wyższy poziom abstrakcji ponad HDAPI oraz

HLAPI, który integrując je dostarcza interfejsy w postaci klas, metod i funkcji języka C++. Umożliwia tworzenie w pełni funkcjonalnych programów przez osoby nie posiadające dużego doświadczenia w dziedzinie wirtualnej rzeczywistości, programowania grafiki komputerowej czy obsługi urządzeń haptycznych.

Wszystkie elementy składające się na OpenHaptics Toolkit mogą być ze sobą łączone i używane zamiennie.

## 2.4. Servo Loop

*Servo Loop* to specjalna, działająca w osobnym wątku z wysokim priorytetem pętla, którą musi posiadać każdy program współpracujący z urządzeniami haptycznymi za pośrednictwem bibliotek OpenHaptics.



Rysunek 2.6: Servo Loop

Zapewnienie stabilnego i realistycznego odwzorowania efektów sprzężenia zwrotnego jest tutaj najwyższym priorytetem. Wymagane jest aby pętla Servo Loop była wykonywana z częstotliwością co najmniej 1kHz.

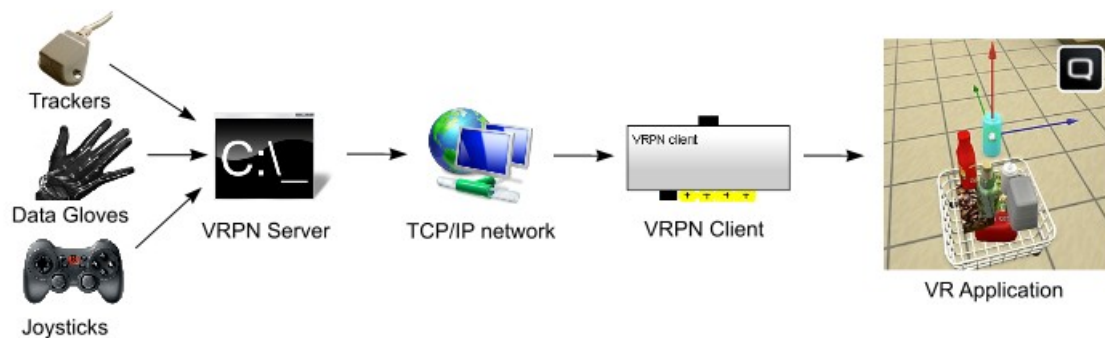
## Rozdział 3

# Virtual Reality Peripheral Network

*Virtual Reality Peripheral Network* (w skrócie VRPN) jest rozbudowaną biblioteką, szkieletem aplikacji (*ang. application framework*) i zbiorem narzędzi ułatwiających tworzenie programów komputerowych wykorzystujących urządzenia używane w systemach wirtualnej rzeczywistości. VRPN dostarcza abstrakcyjnych interfejsów programistycznych i serwerów uniezależniających programistę konkretnych rozwiązań sprzętowych. Umożliwia stosunkowo proste budowanie aplikacji obsługujących urządzenia HCI w wielu popularnych językach programowania (np. C++, Python czy Java) [23].

### 3.1. Opis pakietu

Pakiet VRPN jest od podstaw zaprojektowany jako oprogramowanie sieciocentryczne, tzn. takie w którym sieć komputerowa odgrywa kluczową rolę.



Rysunek 3.1: Architektura VRPN

VRPN definiuje kilka abstrakcyjnych klas wspieranych urządzeń: Analog, Button, Dial, ForceDevice, Imager, Sound, Text oraz Tracker. Przy czym konkretne urządzenie może należeć jednocześnie do jednej lub kilku klas (np. Phantom Omni to Tracker, Button oraz ForceDevice). Istnienie ww. klas powoduje, że wszystkie urządzenia danego typu zgłaszają zmiany stanów jako asynchroniczne wywołania funkcji (*ang. callback*) przekazując w ten sposób obiekty lub wartości parametrów swoich klas. Zadaniem programisty jest odbiór i interpretacja przychodzących danych. Pozwala to na wygodne i szybkie wdrażanie rozwiązań opartych o urządzenia HCI nawet przez osoby bez wcześniejszego doświadczenia z systemami wirtualnej rzeczywistości.

Domyślnie VRPN jest napisany w języku C++ jednak posiada wbudowane mechanizmy generujące kod dla języków Python i Java. W trakcie kompilacji biblioteki z zamiarem jej użycia po stronie klienckiej należy zwrócić szczególną uwagę na docelowy język którego będziemy używać i podać odpowiednie parametry kompilacji zgodnie z wymogami dokumentacji VRPN.

### 3.2. Opis użycia VRPN

W niniejszej pracy wykorzystuje się integrację pakietu VRPN z urządzeniem Phantom Omni do otrzymywania danych statusowych o pozycji, orientacji i wciśniętych przyciskach oraz sterowania sprzężeniem zwrotnym [24].

Stacja robocza obsługująca urządzenie haptyczne posiada skonfigurowany do jego obsługi proces `vrpn_server` będący serwerem TCP/IP oczekującym na przychodzące połączenia od zainteresowanych aplikacji klienckich. Konfiguracja serwera jest przechowywana w tekstowym pliku konfiguracyjnym `vrpn.cfg`, który jest ładowany każdorazowo podczas uruchamiania.

Stworzona aplikacja, w tym przypadku wtyczka do programu PyMOL, jest klientem TCP/IP próbującym nawiązać połączenie z ww. serwerem. W trakcie jej uruchamiania tworzone są obiekty klas `vrpn_Tracker`, `vrpn_Button` i `vrpn_ForceDevice` posiadające wskaźniki do funkcji-uchwytów (*ang. handler*) odpowiednio przetwarzających odebrane dane.

Zarejestrowanie funkcji-uchwyty w obiekcie klasy `vrpn_Button` i obsługa przychodzących zdarzeń jest trywialna i sprowadza się do rozpoznania zmiany stanu jednego z dwóch przycisków. W ramach niniejszej pracy przyciski zostały oprogramowane tak, że wciśnięcie pierwszego z nich powoduje wykonanie superpozycji (nałożenia) struktury wzorca z najbliższym możliwym do dopasowania regionem - zgodnie z danymi pochodzącymi z pliku mapującego. Wciśnięcie drugiego przycisku powoduje wykonanie zbliżenia na linię łączącą środki ciężkości ww. struktur.

Dane udostępniane przez obiekt klasy `vrpn_Tracker` to pozycja i orientacja wskaźnika. Odebrane współrzędne należy przeliczyć na jednostki i wielkości obsługiwane przez pakiet PyMOL. W przypadku orientacji należy przejść z reprezentacji w kwaternionach (wydawana przez VRPN) na postać (akceptowaną przez PyMOL) oś-kąt (*ang. axis-angle*), ta operacja została opisana w części teoretycznej pracy. Współrzędne pozycji wskaźnika należy przemnożyć przez eksperymentalnie wyznaczony współczynnik skalujący, który spowoduje realistyczne odwzorowanie przesunięć z przestrzeni rzeczywistej do wirtualnej.

Klasa `vrpn_ForceDevice` umożliwia sterowanie serwomechanizmami wbudowanymi w urządzenie Phantom Omni. Biblioteka VRPN dostarcza szereg metod umożliwiających wykonanie takich operacji na kilka sposobów: począwszy od najprostszego podania wektora pola siłowego, poprzez zdefiniowanie wirtualnej sprężyny zaczepionej w zadanym punkcie, aż do tworzenia wirtualnych powierzchni czy brył.

Integracja oraz przetwarzanie danych dostarczanych przez pakiet VRPN były znaczącą częścią pracy. Użycie pakietu VRPN zamiast OpenHaptics dało znaczący wzrost niezależności projektu od konkretnego urządzenia haptycznego i otworzyło wiele możliwości dalszego rozwoju i wykorzystania stworzonego w ramach niniejszej pracy oprogramowania.

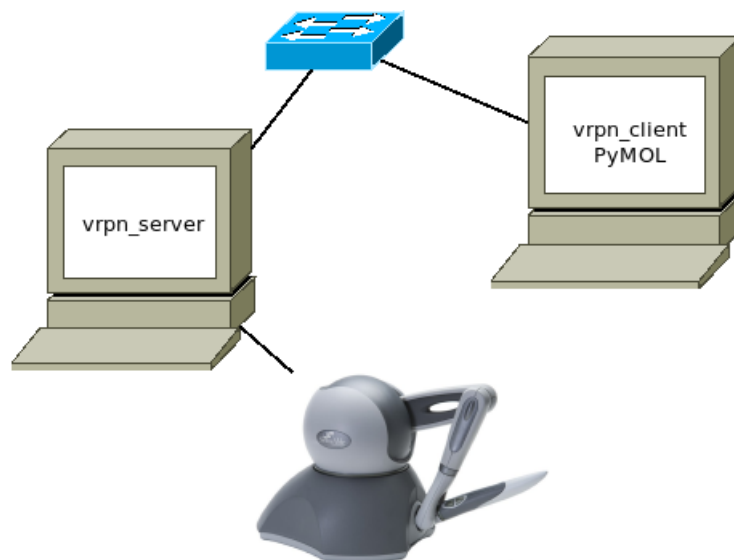
## Rozdział 4

# Implementacja i uruchomienie oprogramowania

Oprogramowanie będące przedmiotem niniejszej pracy zostało stworzone jako rozszerzenie (*ang. plugin*) popularnego pakietu PyMOL. Wykorzystano w nim możliwości wizualizacyjne pakietu w połączeniu z danymi o pozycji i orientacji urządzenia Phantom Omni odbieranymi dzięki bibliotece VRPN. Dane wejściowe do programu - plik mapujący struktury biopolimerów - został dostarczony przez zewnętrzne oprogramowanie (opisane w części teoretycznej pracy).

### 4.1. Opis stanowiska laboratoryjnego

Jak wspomniano we wstępie do pracy prawidłowe uruchomienie oprogramowania będącego jej przedmiotem wymaga wcześniejszego zestawienia odpowiednio skonfigurowanego stanowiska laboratoryjnego.



Rysunek 4.1: Schemat stanowiska laboratoryjnego

Stanowisko składa się z połączonych siecią komputerową stacji roboczych, gdzie jedna stanowi host urządzenia Phantom Omni, druga natomiast to łączący się z nią klient ze stworzonym oprogramowaniem.

Komputer-host musi posiadać kompatybilną z urządzeniem Phantom Omni kartę rozszerzeń z portem IEEE-1394 (FireWire), sterowniki wraz z pakietem OpenHaptics oraz VRPN ze skonfigurowanym procesem `vrpn_server` nasłuchującym połączeń przychodzących na wybranym porcie TCP.

Druga stacja robocza stanowiska laboratoryjnego posiada zainstalowany pakiet PyMOL z oprogramowaniem rozszerzającym jego funkcjonalność - przedmiotem niniejszej pracy. Po jego uruchomieniu, załadowaniu danych wejściowych oraz wprowadzeniu adresu sieciowego hosta urządzenia Phantom Omni, program próbuje nawiązać z nim połączenie i rozpocząć pracę.

W szczególności host urządzenia Phantom Omni oraz pakiet PyMOL mogą być uruchomione na jednej maszynie i wykorzystywać interfejs `localhost`.

## 4.2. Rozszerzanie funkcjonalności pakietu PyMOL

PyMOL jest oprogramowaniem służącym głównie do wizualizacji struktur chemicznych oraz przeprowadzania na nich prostych obliczeń. Jego historia sięga roku 2000, kiedy to Warren Lyford DeLano zainicjował powstanie projektu. Przez lata intensywnych prac nad rozwojem program stał się standardowym narzędziem wykorzystywanym przez wiele wiodących ośrodków naukowo badawczych. Od roku 2010 nad jego rozwojem czuwa firma Schrödinger Inc.

Funkcjonalność pakietu PyMOL może być w łatwy sposób rozszerzalna poprzez wbudowany system obsługi wtyczek (*ang. plugin*) tworzonych w języku Python. Programista ma dostęp do rozbudowanego API (*ang. application programming interface*) zawierającego szeroki wachlarz funkcji umożliwiających tworzenie wizualizacji oraz obliczeń na załadowanych reprezentacjach struktur chemicznych. Ponadto programista tworząc wtyczki może z powodzeniem korzystać z biblioteki standardowej języka Python oraz dowolnych innych bibliotek pochodzących od zewnętrznych dostawców.

Podczas tworzenia niniejszego oprogramowania wykorzystano zarówno API wbudowane w PyMOL jak również pochodzące z zewnętrznych bibliotek takich jak VRPN do efektywnej obsługi urządzenia haptycznego.

Proces tworzenia wtyczek do pakietu PyMOL został szczegółowo opisany w jego dokumentacji.

## 4.3. Dane wejściowe

Oprogramowanie do poprawnego uruchomienia wymaga podania dwóch plików w formacie PDB (*ang. Protein Data Bank*) oraz pliku mapującego.

Pierwszy plik wejściowy definiuje *strukturę celu*. To najczęściej cała, duża cząsteczka białka lub kwasu nukleinowego. W programie zostaje ona umieszczona w początku układu współrzędnych i jest nieruchoma względem wskaźnika urządzenia haptycznego. Stanowi bazową strukturę na której mapowane są regiony podobne do struktury wzorcowej.

*Wzorcowy fragment* może stanowić krótki (najczęściej kilkanaście lub kilkadziesiąt merów) wycinek struktury celu lub dowolną inną podjednostkę czy fragment struktury drugorzędowej (np.  $\alpha$ -helisa). Zostaje on na stałe związany z ruchem wskaźnika urządzenia haptycznego. Każde jego przesunięcie czy zmiana orientacji zostaje w czasie rzeczywistym odwzorowane na ekranie.

*Plikiem mapującym* nazywamy dane uzyskane z obliczeń przeprowadzonych przez zewnętrzne oprogramowanie, zawierających informacje o takich regionach struktury celu, których stopień podobieństwa (dopasowania) do struktury wzorcowej nie przekracza zadanego

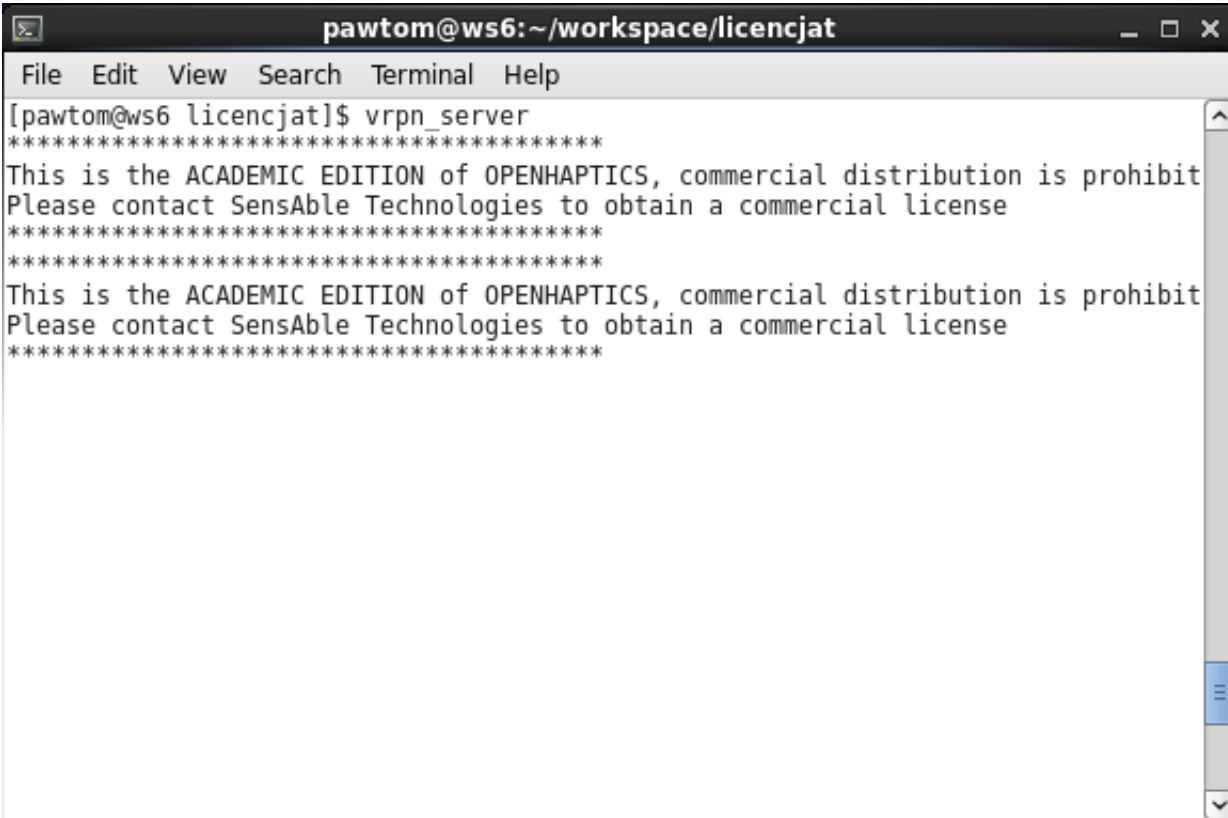


progu. Dane te są kluczowe do działania niniejszego oprogramowania, gdyż na ich podstawie wyliczane są odległości i momenty sił projektowane do urządzenia Phantom Omni. Szczegóły generowania pliku mapującego zostały opisane we wcześniejszym rozdziale.

Po ozwierciedleniu odległości w urządzeniu haptycznym, na ekranie PyMOL wyświetlany jest wektor łączący wzorcową strukturę z najbliższym optymalnie pasującym regionem struktury głównej.

## 4.4. Opis oprogramowania

Po zestawieniu i skonfigurowaniu stanowiska laboratoryjnego zgodnie z wcześniejszym opisem na komputerze będącym hostem urządzenia Phantom Omni należy uruchomić proces `vrpn_server`:

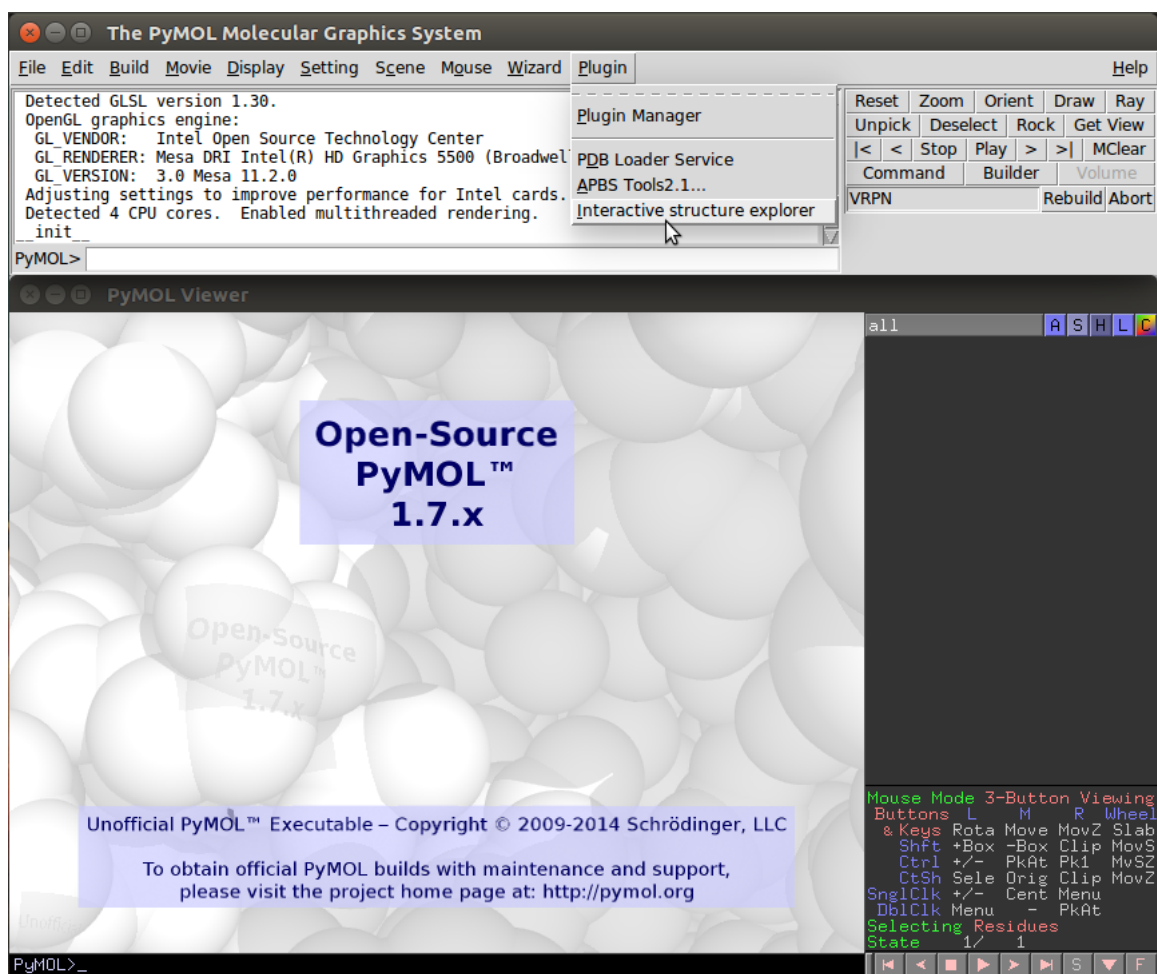
A screenshot of a terminal window titled 'pawtom@ws6:~/workspace/licencjat'. The terminal shows the command '[pawtom@ws6 licencjat]\$ vrpn\_server' being executed. The output consists of two identical blocks of text, each preceded and followed by lines of asterisks. The text in each block reads: 'This is the ACADEMIC EDITION of OPENHAPTICS, commercial distribution is prohibit Please contact SensAble Technologies to obtain a commercial license'. The terminal window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The right side of the window shows standard window controls and a scrollbar.

```
pawtom@ws6:~/workspace/licencjat
File Edit View Search Terminal Help
[pawtom@ws6 licencjat]$ vrpn_server
*****
This is the ACADEMIC EDITION of OPENHAPTICS, commercial distribution is prohibit
Please contact SensAble Technologies to obtain a commercial license
*****
*****
This is the ACADEMIC EDITION of OPENHAPTICS, commercial distribution is prohibit
Please contact SensAble Technologies to obtain a commercial license
*****
```

Rysunek 4.2: Poprawne uruchomienie procesu `vrpn_server`

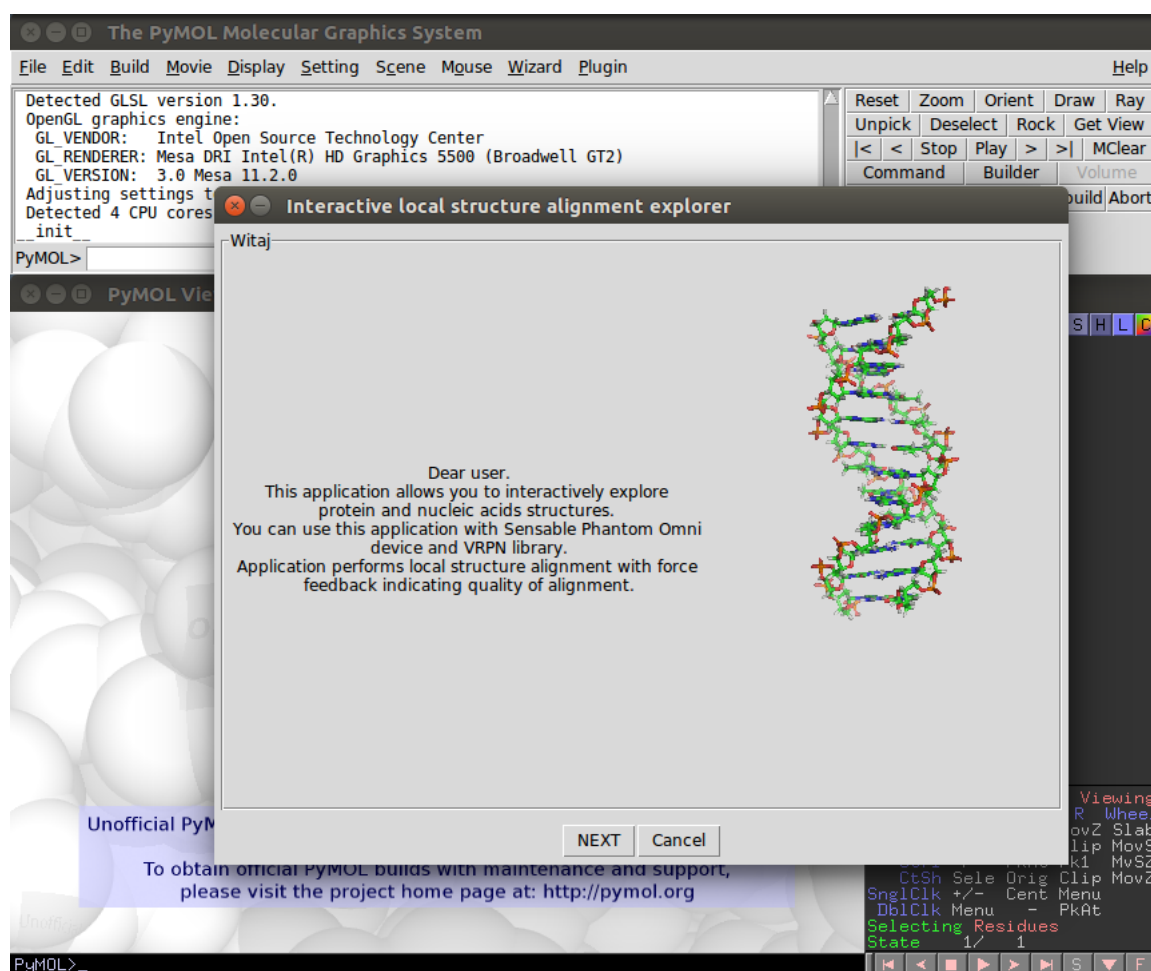
Powyższy ekran pokazuje komunikat z informacją licencyjną. Gdy proces `vrpn_server` uruchomi się pomyślnie, nie powinno być żadnych innych komunikatów.

Na komputerze z oprogramowaniem PyMOL należy uruchomić zainstalowaną wtyczkę przez kliknięcie odpowiedniej pozycji w menu **Plugin**.



Rysunek 4.3: Uruchamianie oprogramowania

W kolejnym kroku pojawi się ekran powitalny z krótkim opisem możliwości oferowanych przez oprogramowanie.



Rysunek 4.4: Ekran powitalny

Po kliknięciu na przycisk *Dalej* przechodzimy do kolejnego okna, które daje nam możliwość przeglądania dysku w poszukiwaniu odpowiednich plików wejściowych. Ponadto należy podać adres komputera, na którym uruchomiony jest proces `vrpn_server`. W przypadku pracy na lokalnej maszynie w oknie należy podać adres `127.0.0.1` lub `phantom@localhost`.

**Interactive local structure alignment explorer**

**Wzorzec**  
 Wzorzec jest struktura, która będziemy próbowali dopasować do cząsteczki bazowej.  
 Wzorzec może stanowić wycinek cząsteczki bazowej, np. jakaś struktura drugorzędowa

**Plik mapowania**  
 Tutaj wybierz plik mapowania

**Identyfikator PDB**  
 Tutaj wybierz plik

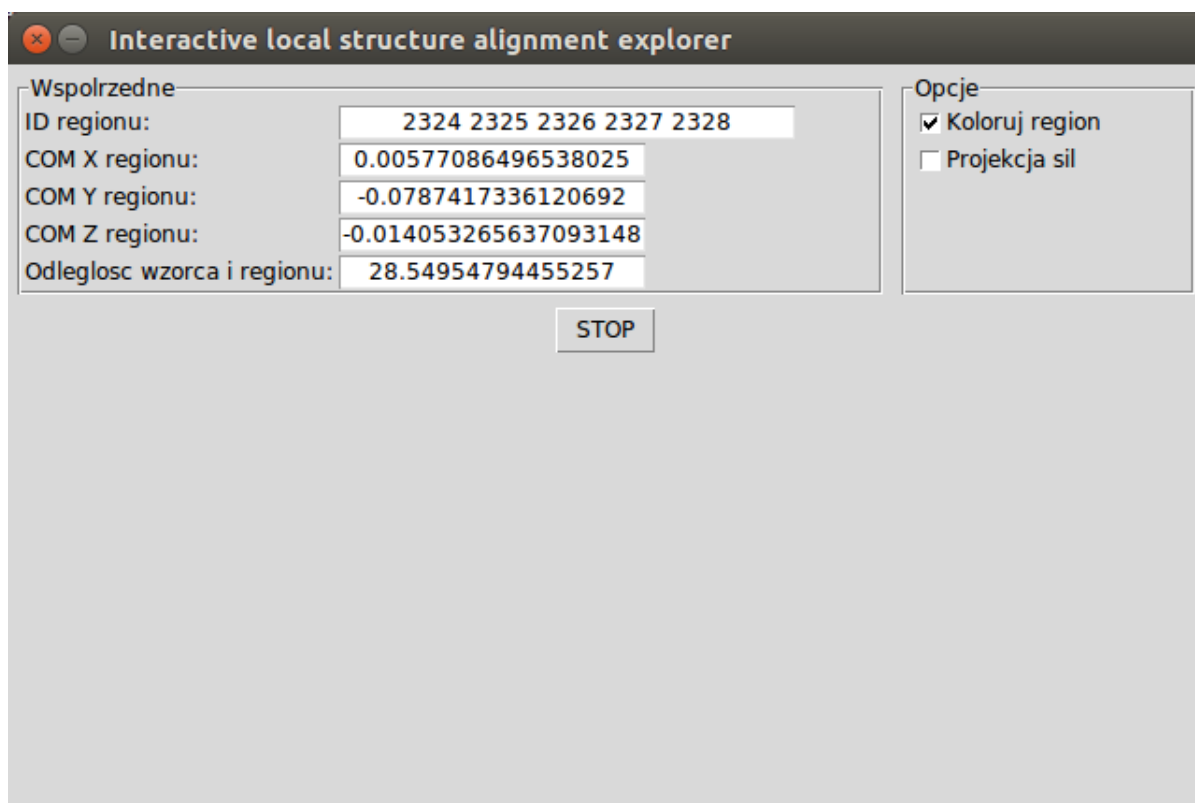
**Adres IP serwera VRPN**

Rysunek 4.5: Wprowadzanie danych wejściowych

Program zakłada poprawność wprowadzonych danych, tzn. nie jest przeprowadzane sprawdzenie, czy załadowane pliki są zgodne z formatem PDB czy prawidłowym formatem pliku mapującego.

W kolejnym kroku program próbuje nawiązać połączenie TCP z hostem i urządzeniem haptycznym. Prawidłowe nawiązanie połączenia rozpoczyna wymianę danych pomiędzy aplikacjami. Proces `vrpn_server` wysyła komunikaty o pozycji oraz orientacji wskaźnika, natomiast PyMOL za pośrednictwem sieci wysyła informację o sprzężeniu zwrotnym do urządzenia Phantom Omni.

Na ekranie widać wzorcową strukturę poruszającą się zgodnie z ruchami urządzenia haptycznego, natomiast urządzenie haptyczne dostaje polecenia sterujące momentami sił, przyciągając wskaźnik do najbliższego (wskazywanego na ekranie przez biały odcinek) podobnego regionu w ramach struktury celu.



Rysunek 4.6: Ekran główny 1

W trakcie normalnej pracy programu, na jego ekranie głównym wyświetlane jest w czasie rzeczywistym szereg informacji...todo - co się wyświetla takiego ciekawego tutaj?

TODO

Rysunek 4.7: Ekran główny 2

# TODO

Rysunek 4.8: Ekran główny 3

## Rozdział 5

# Podsumowanie

W ramach niniejszej pracy została przedstawiona implementacja oprogramowania, którego funkcjonalność obejmuje interaktywne przeglądanie struktur biopolimerów w poszukiwaniu regionów podobnych do wybranej struktury wzorcowej oraz zwrotną projekcję momentów sił wskazującą na jakość dopasowania. Wykorzystuje do tego dane wejściowe pochodzące z zewnętrznych źródeł - plik mapujący oraz pliki z opisem struktur w formacie PDB.

Oprogramowanie wykorzystuje metody i urządzenia wirtualnej rzeczywistości w szczególności pakiet VRPN, urządzenie haptyczne Phantom Omni. Powstało ono w formie wtyczki rozszerzającej funkcjonalność pakietu PyMOL, wykorzystując jego możliwości wizualizacji struktur chemicznych.

Ostateczna forma oprogramowania może mieć dość ograniczone wykorzystanie praktyczne, dość dobrze może jednak nadawać się do celów naukowo-edukacyjnych np. demonstrujących możliwości zastosowania metod wirtualnej rzeczywistości w takich dziedzinach nauki jak biofizyka, biologia molekularna czy chemia.

Wykorzystanie integracji urządzenia Phantom Omni z pakietem VRPN i PyMOL w połączeniu z różnymi metodami bioinformatycznymi tworzy cały szereg nowych i być może nieznanych dotąd (lub niemających praktycznego wykorzystania) możliwości. Łatwość wchodzenia w interakcję z wykreowanym, wirtualnym, molekularnym światem i odzwierciedlanie sił obecnych na tym poziomie może znacznie ułatwić i rozszerzyć proces edukacji przyszłych adeptów różnych nauk przyrodniczych.

Dalszy rozwój niniejszego oprogramowania mógłby polegać na dodaniu funkcjonalności, które w chwili obecnej realizuje oprogramowanie zewnętrzne. Wymagałoby to rozbudowy obecnego programu o część realizującą obliczenia dopasowań strukturalnych implementując przykładowe metody opisane we wcześniejszych rozdziałach.

Kolejnym krokiem mogłoby być dodanie bardziej zaawansowanych metod bioinformatycznych związanych z dokowaniem molekularnym czy projektowaniem leków.

W przypadku dalszego znacznego rozwoju funkcjonalności oprogramowania zgodnie z powyższymi lub innymi propozycjami możliwe jest wyobrażenie sobie wykorzystania go na szerszą skalę w ośrodkach naukowo-badawczych uczelni lub przemysłu.

Poza oprogramowaniem w pracy zostały także przedstawione podstawy teoretyczne na które składały się globalne i lokalne metody dopasowania (uliniowienia) strukturalnego oraz algorytm Kabsch'a maksymalizujący stopień tego dopasowania. Opisane także zostały elementarne przekształcenia geometryczne wykorzystywane w pracy oraz sposób konwersji kwaternionów na macierze rotacji (wykorzystywane przez pakiety VRPN i PyMOL).

Szczegółowo zostało opisane urządzenie haptyczne Phantom Omni i jego integracja z pa-

kietem Virtual Reality Peripheral Network.

W końcowej części została podana bibliografia, z której korzystano w trakcie tworzenia niniejszej pracy.

Kod źródłowy stworzonego oprogramowania wraz z obszernymi komentarzami stanowi załącznik do niniejszego opracowania.



# Bibliografia

- [1] Christian B. Anfinsen, *Principles that govern the folding of protein chains*, Science, 1973
- [2] *Critical Assessment of protein Structure Prediction*, <http://predictioncenter.org/casproll/results.cgi>
- [3] Paweł Daniluk, *Metody wirtualnej rzeczywistości. Urządzenia haptyczne.*, Uniwersytet Warszawski, 2011
- [4] L. Holm, C. Sander, *Protein structure comparison by alignment of distance matrices*, J Mol Biol, 233(1):123–38, 1993
- [5] C. A. Orengo, W. R. Taylor, *SSAP: sequential structure alignment program for protein structure comparison*, Methods Enzymol, 266:617–35, 1996
- [6] I. N. Shindyalov, P. E. Bourne, *Protein structure alignment by incremental combinatorial extension (CE) of the optimal path*, Protein Eng, 11(9):739–47, 1998
- [7] T. Madej, J. F. Gibrat, S. H. Bryant, *Threading a database of protein cores*, Proteins, 23(3):356–69, 1995
- [8] T. Kawabata, K. Nishikawa, *Protein structure comparison using the markov transition model of evolution*, Proteins, 41(1):108–22, 2000
- [9] A. Guerler, E. W. Knapp, *Novel protein folds and their nonsequential structural analogs*, Protein Sci, 17(8):1374–82, 2008
- [10] Paweł Daniluk, *Analiza podobieństwa struktur przestrzennych białek przy użyciu dekrptorów lokalnej struktury*, Uniwersytet Warszawski, 2011
- [11] Krzysztof Fidelis, *A novel approach to fold recognition using sequence-derived properties from sets of structurally similar local fragments of proteins*, Bioinformatics, 2003
- [12] Irina Kufareva, Ruben Abagyan, *Methods of protein structure comparison*, Methods in Molecular Biology, 2012
- [13] Yang Zhang, Jeffrey Skolnick, *TM-align: a protein structure alignment algorithm based on the TM-score*, Nucleic Acids Research, 2005
- [14] Wolfgang Kabsch, *A solution for the best rotation to relate two sets of vectors*, Acta Crystallographica, 1976
- [15] Wolfgang Kabsch, *A discussion of the solution for the best rotation to relate two sets of vectors*, Acta Crystallographica, 1978

- [16] Andries van Dam, *How Are Geometric Transformations (T,R,S) Used in Computer Graphics?*, Introduction to computer graphics, 2000
- [17] A. F. Mobius, *Der Barycentrische Calcul : ein neues Hülfsmittel zur analytischen Behandlung der Geometrie*, 1827
- [18] Y. Magarshak, *Quaternion representation of RNA sequences and tertiary structures*, Bio-Systems, 1993
- [19] William Hamilton, *On Quaternions; or on a new System of Imaginaries in Algebra*, list do Johna T. Graves'a, 1843
- [20] 3D Systems, *Haptic Devices. Haptic devices that add the sense of Touch to your digital world*
- [21] 3D Systems, *OpenHaptics Developer Edition*, Broszura informacyjna producenta
- [22] 3D Systems, *OpenHaptics Toolkit version 3.0 Programmers Guide*, przewodnik programisty, 2015
- [23] Russell M. Taylor II, Thomas C. Hudson, Adam Seeger, Hans Weber, Jeffrey Juliano, Aron T. Helser, *VRPN: A Device-Independent, Network-Transparent VR Peripheral System*, University of North Carolina, 2001
- [24] M. Cuevas-Rodriguez, D. Gonzalez-Toledo, L. Molina-Tanco, A. Reyes-Lecuona, *Contributing to VRPN with a new server for haptic devices*, University of Malaga, 2015

# Kod źródłowy

```
# -*- coding: utf-8 -*-

"""
    This program was created in 2015-16 by Pawel Tomaszewski
    In cooperation with Biophysics Laboratory of Warsaw University
"""

import os
import sys
sys.path.append("/home/crooveck/workspace/LICENCJAT/python_vrpn")
sys.path.append(".")
from transformations import *
from Tkinter import *
from tkinterFileDialog import *
#from ttk import *
from pymol import *
from pymol.cgo import *
from time import *
import vrpn_Tracker
import vrpn_Button
import vrpn_ForceDevice
from math import *

"""
    Mainloop running flag. This indicates if threads and mainloops are
                                running.
"""
IS_RUNNING = False
AUTO_ZOOMING = False
REGION_COLORING=True
FORCES_ENABLED=False

"""
    Global variables for translations.
    trackerX,trackerY,trackerZ - Phantom native coordinates
    x,y,z - Pymol coordinates
    scale - ratio between PYMOL and PHANTOM coordinate
"""
trackerX = trackerY = trackerZ = 0
x = y = z = 0
scale=750

# srodki ciezkosci wzorca i regionu
templateCOM=(0,0,0)
regionCOM=(0,0,0)

"""
```

```

    Global variables for rotations
    previous_orientation stores a quaternion that represent previous
                                orientation
"""
previous_orientation=0

"""
    center of mass of a loaded molecule
"""
molecule_com=[0,0,0]
mapping=[]
regions={}

currentWindow=0
phantomIp=0

# input data files (target,template,mapping):
targetStructureFile=0
templateStructureFile=0
structureMappingFile=0

# zmienne do wyswietlania danych w UI
regionId=regionX=regionY=regionZ=regionTemplateDistance=rmsdEntry=0

def simple_com(region_coordinates):
    # pobiera jako parametr liste z krotkami ze wspolrzednymi i zwraca srodek
    # masy
    x, y, z = 0,0,0
    size = len(region_coordinates)

    for coordinate in region_coordinates:
        x += coordinate[0]
        y += coordinate[1]
        z += coordinate[2]

    return (x/size, y/size, z/size)

def tracker_handler(u, tracker):
    global trackerX, trackerY, trackerZ
    trackerX = tracker[1]
    trackerY = tracker[2]
    trackerZ = tracker[3]

#     print trackerX,trackerY,trackerZ

#     TRANSLACJE:
x0 = trackerX*scale
y0 = trackerY*scale
z0 = trackerZ*scale
#     funkcja dokonujaca przekształcenia - transjacji
global x, y, z
cmd.translate(vector=[(x0-x), (y0-y), (z0-z)], object="template", camera=
1)

x = x0
y = y0
z = z0

#     ROTACJE
global previous_orientation
# bierzacy stan - orientacja

```

```

orientation=(tracker[7],tracker[4],tracker[5],tracker[6]) # inny format
                                                           kwateriona do transformations.py
                                                           niz dostaje z VRPN

# przy pierwszym uruchomieniu
# gdy nie ma poprzedniej orientacji
if(previous_orientation == 0):
    previous_orientation=orientation

rotation_quaternion=quaternion_multiply(quaternion_inverse(
    previous_orientation),orientation)
previous_orientation=orientation

rotation_matrix = quaternion_matrix(rotation_quaternion) # Return
                                                           homogeneous rotation matrix from
                                                           quaternion.
(rotation_angle,rotation_axis,point) = rotation_from_matrix(
    rotation_matrix)

cmd.rotate(axis=[rotation_axis[0],rotation_axis[1],rotation_axis[2]],
    angle=(rotation_angle*180/math.pi), origin=[templateCOM[0],
    templateCOM[1],templateCOM
    [2]], object="template",
    camera=1)

def button_handler(u, button):
    # button[0] - numer przycisku (0-gorny,1-dolny)
    # button[1] - status przycisku (0-puszczony,1-wcisniety)

    global AUTO_ZOOMING
    if(button[0]==0 and button[1]==0):
        AUTO_ZOOMING = False
    elif(button[0]==0 and button[1]==1):
        AUTO_ZOOMING = True

    # wykonuje dopasowanie i translacje wzorca nad regionem
    if(button[0]==1 and button[1]==1):

        cmd.align("template","region")

        reg_com=cmd.centerofmass("region")
        temp_com=cmd.centerofmass("template")

        cmd.translate(object="template", vector=[reg_com[0]-temp_com[0],
            reg_com[1]-temp_com[1],reg_com[2]-temp_com[2]], camera=0)

def force_handler(u, force):
    # print "force",force
    abc='test'

def draw_xyz_axes(x0, y0, z0):
    w = 0.5 # cylinder width
    l = 10 # cylinder length
    h = 2 # cone height
    d = w * 1.618 # cone base diameter

    axes = [
        CYLINDER, x0, y0, z0, l, 0.0, 0.0, w, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0,
        CYLINDER, x0, y0, z0, 0.0, 1, 0.0, w, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0,

```

```

CYLINDER, x0, y0, z0, 0.0, 0.0, 1, w, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0,
CONE, 1, 0.0, 0.0, (h+1), 0.0, 0.0, d, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0,
                                0.0, 1.0, 1.0,
CONE, 0.0, 1, 0.0, 0.0, (h+1), 0.0, d, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0,
                                0.0, 1.0, 1.0,
CONE, 0.0, 0.0, 1, 0.0, 0.0, (h+1), d, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0,
                                1.0, 1.0, 1.0]

cmd.load_cgo(axes, "axes")

def draw_template_structure(template_pdb_file):
    cmd.load(template_pdb_file, "template")
    cmd.hide("lines", "template")
    cmd.show("cartoon", "template")
    cmd.color("green", "template")

    template_com=cmd.centerofmass("template")
    # centruje wskaxnik (przenosze go do zera)
    cmd.translate(vector=[-template_com[0], -template_com[1], -template_com[2]
                        ], object="template", camera=0)

def draw_target_structure(target_pdb_file):
    # laduje czasteczke
    cmd.load(target_pdb_file, "target")
    cmd.hide("lines", "target")
    cmd.show("cartoon", "target")
    cmd.color("red", "target")

    # licze jej centrum masy
    global molecule_com
    molecule_com=cmd.centerofmass("target")
    # przesuwam ja na srodek ekranu, srodek ciezkosci na (x,y,z)=(0,0,0)
    cmd.translate(vector=[-molecule_com[0], -molecule_com[1], -molecule_com[2]
                        ], object="target", camera=0)

    # pobieram liste pozycji 3D atomow w czasteczce
    stored.pos = []
    cmd.iterate_state(1, "target", "stored.pos.append((x,y,z,elem,chain,resi)
    )")

def load_mapping_file(mapping_file):
    file=open(mapping_file, "r")

    global mapping
    mapping=[]

    for line in file:
        mapping_symbol=line.split()[0]
        chain=line.split()[1][0]
        resi=line.split()[1][1:]
        mapping.append([mapping_symbol, chain, resi, 0])
    file.close()

    # na ostatnie pole w mapping wstawiam COM obliczony dla kazdego
    # nukleotydu
    for nucleotide in mapping: # iteruje po wszystkich liniach z pliku
    # mapujacego
    # pobieram atomy nalezace do poszczegolnych nukleotydow

```

```

nucl_coords = [[atom[0],atom[1],atom[2]] for atom in stored.pos if (
    atom[5]==nucleotide[2])]
nucleotide[3]=simple_com(nucl_coords) # srodek masy dla nukleotydu

print "SKONCZYLEM LADOWANIE MAPOWANIA"

def calculate_regions_com():
    # zliczam ilosc nukleotydow w helisie wzorcowej
    # UWAGA: tutaj zakladam, ze wzorcowa helisa (czy struktura) sklada sie z
    #         dwuch lancuchow o
    # tej samej dlugosci (ilosci nukleotydow). To oznacza, ze szukam regionow
    # o dlugosci jednego lancucha helisy, czyli polowy wszystkich nukleotydow
    # z tej helisy.

    unique_nucl=set()
    cmd.iterate_state(1,"template","unique_nucl.add((chain,resi))",space={'
        unique_nucl':unique_nucl})
    M=len(mapping)      # ilosc nukleotydow w BADANEJ czasteczce
    N=len(unique_nucl)/2 # polowa wszystkich nukleotydow z WZORCOWEJ
                        # czasteczki

    # wyszukiwanie regionow
    for m in range(0,M-N+1): # -1 jesli blad
        region=() # inicjuje pusty tuple
        region_coords=[]
        for n in range(0,N): # -1 jesli blad
            if mapping[m+n][0]=='0':
                break
            region=region+(mapping[m+n][2],)
            region_coords.append(mapping[m+n][3])
        if n==(N-1):
            #znaleziono region w badanej czasteczke pasujacy do
            #                             czasteczki wzorcowej
        #
        print "JEST REGION: ", m, region, region_coords
        # dodaje znaleziony region do tablicy regions
        regions[region]=simple_com(region_coords)

    #     print regions

def new_calculate_regions_com():
    unique_nucl=set()
    cmd.iterate_state(1,"template","unique_nucl.add((chain,resi))",space={'
        unique_nucl':unique_nucl})
    M=len(mapping)      #ilosc nukleotydow z czasteczki celu
    N=len(unique_nucl)   # wszystkie nukleotydy z WZORCOWEJ czasteczki

    # wyszukiwanie regionow
    for m in range(0,M-N+1): # -1 jesli blad
        region=() # inicjuje pusty tuple
        region_coords=[]
        for n in range(0,N): # -1 jesli blad
            if mapping[m+n][0]=='0':
                break
            region=region+(mapping[m+n][2],)
            region_coords.append(mapping[m+n][3])
        if n==(N-1):
            # znaleziono region w badanej czasteczke pasujacy do
            #                             czasteczki wzorcowej
        #
        print "JEST REGION: ", m, region, region_coords
        # dodaje znaleziony region do tablicy regions

```

```

        regions[region]=simple_com(region_coords)

#     print regions

def find_closest_atom(x0,y0,z0):
    if(len(stored.pos)==0):
        return [0,0,0]

    minimumDistance=sys.float_info.max      # poczatkowa wartosc minimalna
                                            # powinna byc duza
    closestAtomDistance=0                    # index najblizszego atomu
    # liczymy najblizszy atom
    for atomNumber in xrange(0,len(stored.pos)):
        xDistance = (stored.pos[atomNumber][0]-molecule_com[0])-x0
        yDistance = (stored.pos[atomNumber][1]-molecule_com[1])-y0
        zDistance = (stored.pos[atomNumber][2]-molecule_com[2])-z0
        distance=sqrt(math.pow(xDistance,2)+math.pow(yDistance,2)+math.pow(
            zDistance,2))

        if(distance<minimumDistance):
            minimumDistance=distance
            closestAtomDistance=atomNumber

    # zapamietuje wsp. nablizszego atomu
    closestAtomX=stored.pos[closestAtomDistance][0]-molecule_com[0]
    closestAtomY=stored.pos[closestAtomDistance][1]-molecule_com[1]
    closestAtomZ=stored.pos[closestAtomDistance][2]-molecule_com[2]

    return (closestAtomX/scale, closestAtomY/scale, closestAtomZ/scale)

def find_closest_region(x0,y0,z0):
    # jesli nie ma regionow, to zwroc najblizszy atom
    if len(regions)==0:
        return find_closest_atom(x0, y0, z0)

    min_distance=sys.float_info.max # startujemy od najwiekszej mozliwej
                                    # wielkosci
    closestX,closestY,closestZ = 0,0,0
    closestRegionId=0

    for region in regions:          # iteruje po regionach
        coords=regions[region]      # pobieram z mapy regionow wspolrzedne
        x_dist = (coords[0]-molecule_com[0])-x0
        y_dist = (coords[1]-molecule_com[1])-y0
        z_dist = (coords[2]-molecule_com[2])-z0
        distance=sqrt(math.pow(x_dist,2)+math.pow(y_dist,2)+math.pow(z_dist,2
        ))

        if(distance<min_distance):
            min_distance=distance
            closestRegionId=region
            closestX = coords[0]-molecule_com[0]
            closestY = coords[1]-molecule_com[1]
            closestZ = coords[2]-molecule_com[2]

    # tworze nowy selection do najblizszego regionu
    selectCommand="resi "
    for resi in closestRegionId:
        selectCommand+=resi+", "

```



```

if REGION_COLORING:
    cmd.color("red","target")
    cmd.select("region",selectCommand)
    cmd.color("yellow","region")
else:
    cmd.color("red","target")
    cmd.select("region",selectCommand)

# robie alter dla funkcji rms_cur
#   resiList=set()
#   cmd.iterate("template","resiList.add(resi)",space={'resiList':resiList})
#   resiList=sorted(resiList)
#
#   for i in range(0,len(closestRegionId)):
#       selection="template and resi "+str(resiList[i])
#       expression="resi="+str(closestRegionId[i])
#       cmd.alter(selection,expression)
##       print "sel= "+ selection + "exp= "+ expression

return [closestRegionId,(closestX/scale),(closestY/scale),(closestZ/scale
),min_distance]

def vrpn_client():
    global structureMappingFile,templateStructureFile,phantomIp

    tracker = vrpn_Tracker.vrpn_Tracker_Remote(phantomIp.get())
    vrpn_Tracker.register_tracker_change_handler(tracker_handler)
    vrpn_Tracker.vrpn_Tracker_Remote.register_change_handler(tracker, None,
                                                             vrpn_Tracker.
                                                             get_tracker_change_handler())

    button = vrpn_Button.vrpn_Button_Remote(phantomIp.get())
    vrpn_Button.register_button_change_handler(button_handler)
    vrpn_Button.vrpn_Button_Remote.register_change_handler(button, None,
                                                            vrpn_Button.
                                                            get_button_change_handler())

    forceDevice = vrpn_ForceDevice.vrpn_ForceDevice_Remote(phantomIp.get())
    vrpn_ForceDevice.register_force_change_handler(force_handler)
    vrpn_ForceDevice.vrpn_ForceDevice_Remote.register_force_change_handler(
        forceDevice, None,
        vrpn_ForceDevice.
        get_force_change_handler())

    draw_xyz_axes(0,0,0)

    draw_template_structure(templateStructureFile.get())
    draw_target_structure(targetStructureFile.get())
    load_mapping_file(structureMappingFile.get())

    new_calculate_regions_com()

    global regionId, regionX, regionY, regionZ
    global x,y,z
    global templateCOM,regionCOM

    sleep(1)      # czekam az sie wszystko polaczy i narysuje

```

```

while IS_RUNNING:
    if(not AUTO_ZOOMING):
        cmd.zoom('all')

    tracker.mainloop()
    forceDevice.mainloop()
    button.mainloop()

    # obliczam aktualny srodek ciezkosci wzorca
    templateCOM=cmd.centerofmass("template")

    # znajduje nablizszy dla wzorca region w czasteczce celu
    # do ktorego bede przyciagac wzorzec/wskaznik
    region=find_closest_region(templateCOM[0],templateCOM[1],templateCOM[2])

    regionCOM=cmd.centerofmass("region")

    print "RMSD: ", cmd.rms_cur("template","region")

    # aktualizacja interfejsu
    regionId.delete(0,'end') #todo: zmienic na zmienna textvariable
    regionId.insert(0,region[0])
    regionX.delete(0,'end')
    regionX.insert(0,region[1])
    regionY.delete(0,'end')
    regionY.insert(0,region[2])
    regionZ.delete(0,'end')
    regionZ.insert(0,region[3])
    regionTemplateDistance.delete(0,'end')
    regionTemplateDistance.insert(0,region[4])

    if FORCES_ENABLED:
        force=100 # skalarna wartosc sily |F|
        forceX = (region[1]-trackerX) # wektor X sily
        forceY = (region[2]-trackerY) # wektor Y sily
        forceZ = (region[3]-trackerZ) # wektor Z sily
        forceDevice.setFF_Origin(trackerX, trackerY, trackerZ)
        forceDevice.setFF_Force(force*forceX, force*forceY, force*forceZ)
        forceDevice.setFF_Jacobian(force,0,0, 0,force,0, 0,0,force)
        forceDevice.setFF_Radius(0.1)
        forceDevice.sendForceField()
    else:
        forceDevice.setFF_Origin(0,0,0)
        forceDevice.setFF_Force(0,0,0)
        forceDevice.setFF_Jacobian(0,0,0, 0,0,0, 0,0,0)
        forceDevice.setFF_Radius(0.0)
        forceDevice.sendForceField()

    # rysuje linie laczac wzorzec/wskaznik z najblizszym regionem/atomem
    cmd.delete('link')
    cmd.load_cgo([CYLINDER, templateCOM[0],templateCOM[1],templateCOM[2],
                    (regionCOM[0]), (regionCOM[1]), (regionCOM[2]),
                    (regionCOM[2]), 0.1, 255,
                    255, 255, 255, 255, 255], 'link')

#    cmd.load_cgo([CYLINDER, templateCOM[0],templateCOM[1],templateCOM[2],
                    (region[1]*scale), (region[2]*scale),
                    (region[3]*scale), 0.1, 255, 255,
                    255, 255, 255, 255], 'link')

```

```

cmd.delete("*")
x=y=z=0

def stop():
    global IS_RUNNING
    IS_RUNNING = False
    sleep(1)
    configWindow()

def doColorRegion():
    global REGION_COLORING

    if REGION_COLORING:
        REGION_COLORING=False
    else:
        REGION_COLORING=True

def doEnableForces():
    global FORCES_ENABLED

    if FORCES_ENABLED:
        FORCES_ENABLED=False
    else:
        FORCES_ENABLED=True

def statsWindow():
    global currentWindow, IS_RUNNING
    IS_RUNNING = True
    thread.start_new_thread(vrpn_client, ())

    currentWindow.destroy()

    w=640
    h=400
    currentWindow=Tk()
    currentWindow.title("Interactive local structure alignment explorer")
    x=currentWindow.winfo_screenwidth()/2 - w/2
    y=currentWindow.winfo_screenheight()/2 - h/2
    currentWindow.geometry("%dx%d+%d+%d" % (w,h,x,y))
    currentWindow.attributes('-topmost', 1)
    currentWindow.resizable(False, False)
    currentWindow.grid_columnconfigure(0, weight=1)
    currentWindow.grid_columnconfigure(1, weight=1)

    global regionId, regionX, regionY, regionZ, regionTemplateDistance,
        rmsdEntry

    group=LabelFrame(currentWindow, text="Wspolrzedne")
    group.grid(column=0,padx=5,pady=5,sticky='WE')
    Label(group, text="ID regionu:",anchor=W).grid(row=0,column=0,sticky='WE',
        )
    regionId=Entry(group, width=30, justify=CENTER)
    regionId.grid(row=0,column=1,sticky='W')
    Label(group, text="COM X regionu:",anchor=W).grid(row=1,column=0,sticky='
        WE')
    regionX=Entry(group, width=20, justify=CENTER) # todo: odswiezac te pola
        po textvariable zamiast tego co
        jest
    regionX.grid(row=1,column=1,sticky='W')

```

```

Label(group, text="COM Y regionu:", anchor=W).grid(row=2, column=0, sticky='
WE')
regionY=Entry(group, width=20, justify=CENTER) # todo: jw.
regionY.grid(row=2, column=1, sticky='W')
Label(group, text="COM Z regionu:", anchor=W).grid(row=3, column=0, sticky='
WE')
regionZ=Entry(group, width=20, justify=CENTER) # todo: jw.
regionZ.grid(row=3, column=1, sticky='W')
Label(group, text="Odleglosc wzorca i regionu:", anchor=W).grid(row=4,
column=0, sticky="WE")
regionTemplateDistance=Entry(group, width=20, justify=CENTER) # todo: jw
regionTemplateDistance.grid(row=4, column=1, sticky='W')

group=LabelFrame(currentWindow, text="Opcje")
group.grid(column=1, row=0, sticky='NSWE', pady=5, padx=5)
colors=Checkbutton(group, text="Koloruj region", command=doColorRegion)
colors.select()
colors.grid(row=0, sticky="W")
Checkbutton(group, text="Projekcja sil", command=doEnableForces).grid(row=1
, sticky="W")

# RMSD
# group=LabelFrame(currentWindow, text="Wykres RMSD (Root-mean-square
diviation)")
# group.grid(row=1, columnspan=2, sticky="NSWE", padx=5, pady=5)
# rmsdEntry=Entry(group, width=20, justify=CENTER)
# Label(group, text="\n\ntu bedzie wykres...\n\n").grid(sticky="WE")

# spacer
Frame(currentWindow).grid(sticky="NSWE")

# przyciski
group=Frame(currentWindow)
group.grid(row=3, columnspan=2)
stopButton=Button(group, text="STOP", command=stop)
stopButton.grid()

currentWindow.mainloop()

def chooseTemplateStructureFile():
    global templateStructureFile
    templateStructureFile.set(askopenfilename( filetypes=(("PDB", "*.pdb"), (
"All files", "*.*")) ))
    print templateStructureFile.get()

def chooseStructureMappingFile():
    global structureMappingFile
    structureMappingFile.set(askopenfilename( filetypes=(("MAP", "*.map"), (
All files", "*.*")) ))
    print structureMappingFile.get()

def chooseTargetStructureFile():
    global targetStructureFile
    targetStructureFile.set(askopenfilename( filetypes=(("PDB", "*.pdb"), (
All files", "*.*")) ))
    print targetStructureFile.get()

def configWindow():

```

```

global currentWindow, templateStructureFile, structureMappingFile, phantomIp
                                , targetStructureFile

currentWindow.destroy()

w=640
h=480
currentWindow=Toplevel()
currentWindow.title("Interactive local structure alignment explorer")
x=currentWindow.winfo_screenwidth()/2 - w/2
y=currentWindow.winfo_screenheight()/2 - h/2
currentWindow.geometry("%dx%d+%d+%d" % (w,h,x,y))
currentWindow.attributes('-topmost', 1)
currentWindow.resizable(False, False)

#   Wybor wzorca
message="Wzorzec jest struktura, ktora bedziemy probowali dopasowac do
                                czasteczki bazowej.\
\nWzorzec moze stanowic wycinek czasteczki bazowej, np. jakas struktura
                                drugorzędowa"

group=LabelFrame(currentWindow, text="Wzorzec", padx=5, pady=5)
group.pack(fill=BOTH, padx=5, pady=5)
Label(group, text=message, anchor=W).pack(fill=BOTH)
Entry(group, textvariable=templateStructureFile, width=50, state="readonly").
                                .pack(pady=5, side=LEFT)
Button(group, text="Wybierz plik", command=chooseTemplateStructureFile).
                                pack(side=LEFT)

#   Wybor pliku mapowania
group=LabelFrame(currentWindow, text="Plik mapowania", padx=5, pady=5)
group.pack(fill=BOTH, padx=5, pady=5)
Label(group, text="Tutaj wybierz plik mapowania", anchor=W).pack(fill=BOTH)
Entry(group, textvariable=structureMappingFile, width=50, state="readonly").
                                pack(pady=5, side=LEFT)
Button(group, text="Wybierz plik", command=chooseStructureMappingFile).pack
                                (side=LEFT)

#   Wybor identyfikatora PDB
group=LabelFrame(currentWindow, text="Identyfikator PDB", padx=5, pady=5)
group.pack(fill=BOTH, padx=5, pady=5)
Label(group, text="Tutaj wybierz plik ", anchor=W).pack(fill=BOTH)
Entry(group, textvariable=targetStructureFile, width=50, state="readonly").
                                pack(pady=5, side=LEFT)
Button(group, text="Wybierz plik", command=chooseTargetStructureFile).pack(
                                side=LEFT)

#   ustawianie adresu IP serwera VRPN
group=LabelFrame(currentWindow, text="Adres IP serwera VRPN", padx=5, pady=5
                                )
group.pack(fill=BOTH, padx=5, pady=5)
Entry(group, justify=CENTER, textvariable=phantomIp, width=30).pack(pady=5,
                                side=LEFT)

#   spacer
Frame(currentWindow).pack(padx=5, expand=TRUE)

#   przyciski
group=Frame(currentWindow)
group.pack(padx=5, pady=5)

```

```

        Button(group, text="Anuluj", command=currentWindow.destroy).pack(side=RIGHT
        )
        Button(group, text="Dalej", command=statsWindow).pack()

        currentWindow.mainloop();

def helloWindow():
    global currentWindow, templateStructureFile, structureMappingFile, phantomIp
        , targetStructureFile

    w=640
    h=480
    currentWindow=Toplevel()
    currentWindow.title("Interactive local structure alignment explorer")
    x=currentWindow.winfo_screenwidth()/2 - w/2
    y=currentWindow.winfo_screenheight()/2 - h/2
    currentWindow.geometry("%dx%d+%d+%d" % (w,h,x,y))
    currentWindow.attributes('-topmost',1)
    currentWindow.resizable(False, False)

    helloMsg="\
Dear user.\
\nThis application allows you to interactively explore \
\nprotein and nucleic acids structures. \
\nYou can use this application with Sensable Phantom Omni \
\ndevice and VRPN library.\
\nApplication performs local structure alignment with force \
\nfeedback indicating quality of alignment.\
"

    group=LabelFrame(currentWindow, text="Witaj", padx=5, pady=5)
    group.pack(fill=BOTH, padx=5, pady=5, expand=True)
    Label(group, text=helloMsg).pack(fill=BOTH, side=LEFT)
    dnaImage=PhotoImage(file="dna.gif")
    Label(group, image=dnaImage).pack(fill=BOTH)

    group=Frame(currentWindow)
    group.pack(padx=5, pady=5)
    Button(group, text="Cancel", command=currentWindow.destroy).pack(side=RIGHT
    )
    Button(group, text="NEXT", command=configWindow).pack()

#    inicjalizacja zmiennych globalnych
    templateStructureFile=StringVar(value=os.getcwd()+"/helix_chain_a.pdb")
    structureMappingFile=StringVar(value=os.getcwd()+"/1fg0_helix.map")
    targetStructureFile=StringVar(value=os.getcwd()+"/1fg0.pdb")
    phantomIp=StringVar(value="phantom0@10.21.2.136")

    currentWindow.mainloop();

def __init__(self):
    print "__init__"
    self.menuBar.addmenuitem('Plugin', 'command', 'VRPN',
    label = 'Interactive structure explorer', command = lambda s=self:
        helloWindow())

```