Rozpoznawanie i estymacja pozycji obiektu z użyciem systemu wizyjnego w zadaniu chwytania realizowanego przez manipulator

Praca inżynierska

Adam Wołkowycki

Promotor: Dr Adam Wolniakowski

18 stycznia 2022

Abstract

In the recent years, a rising trend in application of autonomous robots can be observed. Nowadays, many solutions intended to this part of market are developed and each of these is based on solutions that are hard to algorithmizable. The examples of this kind of tasks are image processing, object position estimation and configuration of kinematics. I am going to focus on them in my thesis.

The purpose of this thesis is to develop a solution resposible for recognition a given object and perform grasping it by a manipulator. The further development of the problem known as *bin picking* aims at increasing a scope of industrial robots' operation in a context of their work.

Because the tasks I mentioned are often performed by a particular type of robots, in my work I am going to use UR5 manipulator by Danish manufacturer Universal Robots. There will be implemented algorithm on it that will be able to define a location of seen objects based on the data from the computer vision sensors. In my work I will be using code examples written in Python. Thanks to this, it will be possible to deeply verified in practice the working of a robot.

Streszczenie

W ostatnich latach można zaobserwować wzrastający trend na zastosowanie robotów o działaniu autonomicznym. Powstaje obecnie wiele rozwiązań przeznaczonych dla tego segmentu rynku, a każdy z nich opiera swoje działanie o rozwiązania, które są przyjęte jako trudno algorytmizowalne. Przykładami takich zadań są przetwarzanie obrazu, estymacja położenia obiektów i konfiguracja kinematyki robota. Właśnie na nich zamieram skupić się w mojej pracy.

Celem pracy jest opracowanie rozwiązania odpowiedzialnego za rozpoznawanie wskazanego obiektu i realizację chwycenia go przy wykorzystaniu manipulatora. Rozwinięcie problemu znanego pod nazwą *bin picking* ma przede wszystkim na celu zwiększenie zakresu działania robotów przemysłowych w kontekście ich pracy.

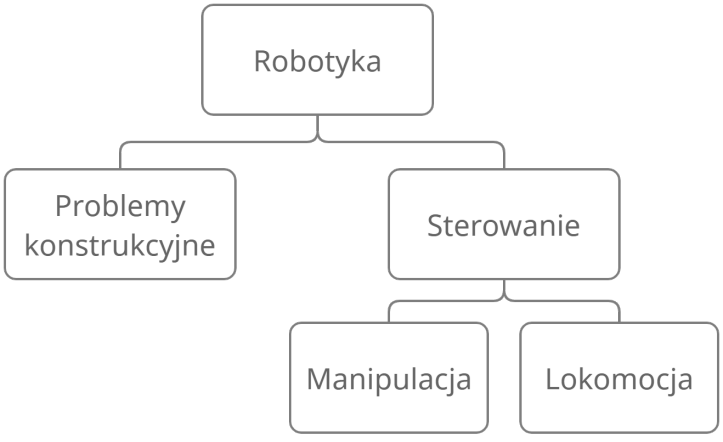
Ponieważ zadania, o których wspomniałem są często realizowane przez określony rodzaj robotów, w swojej pracy zamierzam wykorzytać manipulator UR5 duńskiego producenta Universal Robots. Na nim zostanie zaimplementowany algorytm, który w oparciu o dane z systemu wizyjnego będzie w stanie określić położenie widzianych przedmiotów. W pracy posłużę się programami stworzonymi w języku Python. Dzięki temu działanie robota będzie można gruntownie zweryfikować w praktyce.

Spis treści

1 Wprowadzenie

Definiując współczesną robotykę jako dziedzinę wiedzy mamy na uwadze przede wszystkim dwa problemy: natury konstrukcyjnej i sterowania. Ponieważ wielu producentów z branży robotyki oferuje szeroką gamę rozwiązań spełniających potrzeby niniejszej pracy, posłużymy się robotem industrialnym, w którego konstrukcję nie będziemy ingerować. Dlatego, aby zrozumieć koncepcje zawarte w tej pracy nie jest wymagana dogłębna znajomość budowy robota. Natomiast, jeśli chodzi o sterowanie to możemy ponownie wyodrębnić dwie podrzędne kategorie: manipulację i lokomocję.

- **Manipulację** definiujemy jako zdolność do wykonywania precyzyjnych ruchów przez efektor końcowy robota w celu osiągnięcia wyznaczonego celu, np. przeniesienia przedmiotu procesowanego - tzw. zadanie *pick and place*. Roboty posiadające tą cechę nazywamy mianem manipulatorów.
- **Lokomocja** z kolei, jak sama nazwa wskazuje, opisuje zdolność do przemieszczania się platformy mobilnej. Jest kluczowa m.in. dla pojazdów AVG *(Automated Guided Vehicles)*, łazików czy robotów kroczących. Ponieważ w swojej pracy zamierzam skupić się na robotach stacjonarnych, nie będę wracał więcej do tego tematu.



Rysunek 1: Podział robotyki na odrębne zagadnienia.

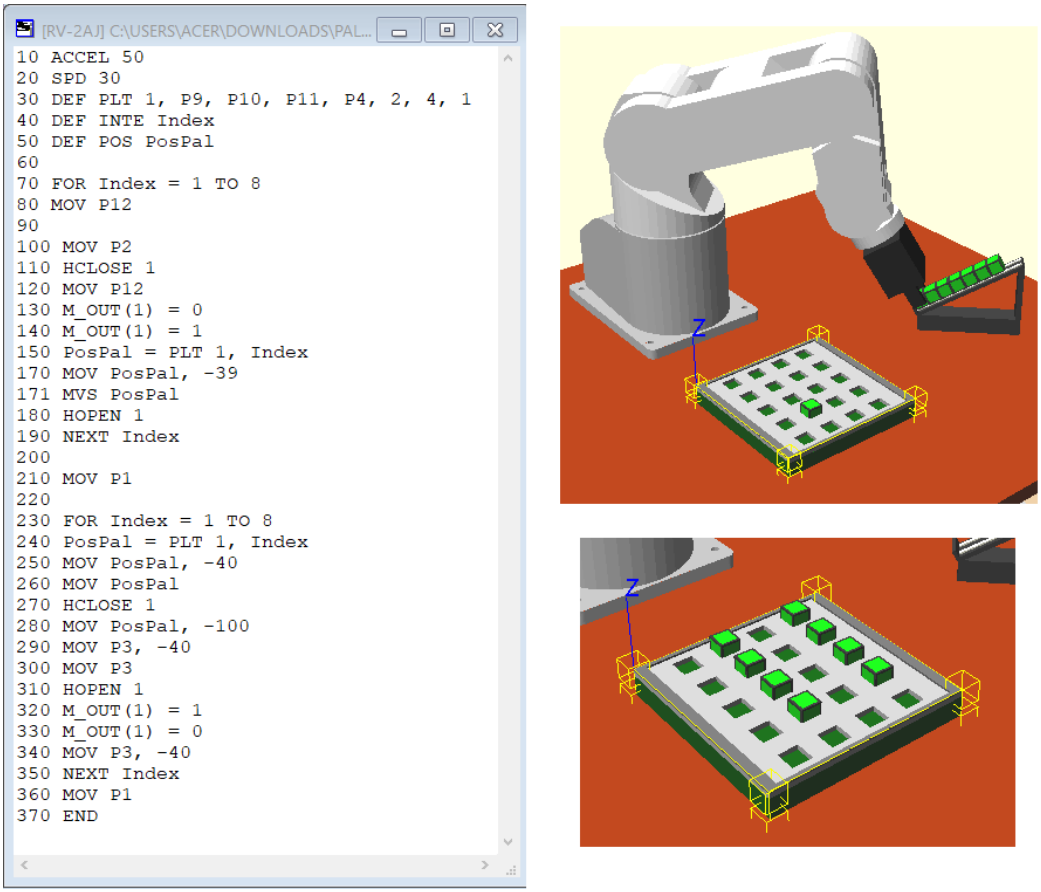
Musimy jednak mieć na uwadze, że jest to dość ogólna i luźna klasyfikacja. Pierwotnie, podobny podział zaproponował Marc H. Raibert kończąc książkę *Legged Robots That Balance* w ostatnim rozdziale w pytaniu *Do Locomotion and Manipulation Have a Common Ground?* Dalej problem został rozwinięty w kontekście badań nad lokomocją i manipulacją oraz tego jak te dwie dziedziny wpływają na siebie nawzajem.

1.1 Manipulacja a percepcja

Gdy mamy już wyjaśnione ogólne zagadnienia możemy skupić się na danym zadaniu manipulatora. Załóżmy, że zadaniem robota jest pobranie kilku kostek z podajnika w miejscu A i przeniesienie ich w odpowiednie miejsca na palecie B, tzw. paletyzacja. Aby wykonać to zadanie możemy podejść do niego na dwa sposoby: zapewniając maszynie odpowiednią percepcję lub nie.

- **Przypadek 1. Brak percepcji**
  Brak percepcji oznacza brak danych wejściowych. Operujemy jedynie na wielkościach, które muszą zostać założone z góry. W tym przykładzie będą to: pozycje bazy i kiści manipulatora, podajnika, palety oraz rozmiar kostki. Z tego powodu, robot pozbawiony percepcji zawsze musi otrzymać od nas dokładne informacje odnośnie trajektorii ruchu.

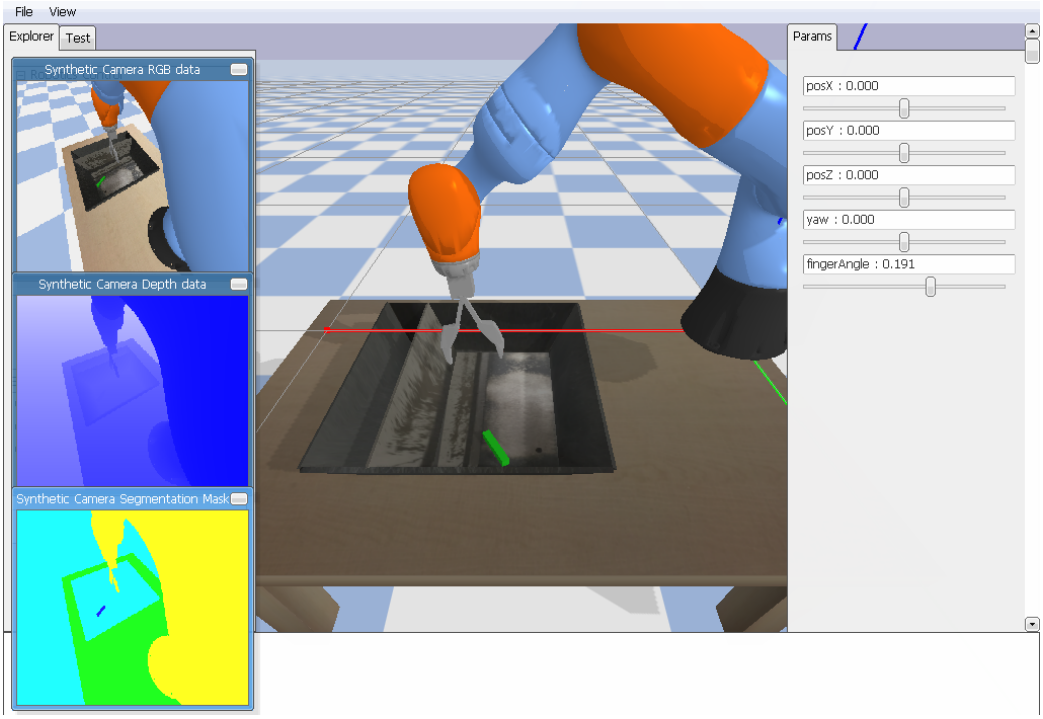
- **Przypadek 2. Percepcja z użyciem systemu wizyjnego**
  Przez system wizyjny możemy rozumieć dowolny rodzaj kamery, jak również projektory chmur punktów i urządzenia nakładające na siebie kolory z danymi o głębokości obrazu, zwane kamerami RGBD. W takim przypadku operujemy znaczną ilością danych, dzięki czemu znajomość wymienionych wyżej wartości nie musi być konieczna - możemy oszacować te wartości używając metod widzenia maszynowego *(Computer Vision).* Zastosowanie systemu wizyjnego możemy ponownie podzielić na dwa typy: z kamerą zamontowaną nieruchomo względem bazy robota oraz na jego narzędziu. Różnica między nimi polega głównie na tym, że w pierwszym przypadku widziane obiekty procesowane, jeśli robot ich bezpośrednio nie dotyka, pozostają w bezruchu względem kamery, a przez to nie zmienia się ich obraz. Przypadek z kamerą zamontowaną na kiści jest w stanie zapewnić więcej użytecznych ujęć, ale przez to jest też trudniejszy do zaimplementowania.



Rysunek 2: Przykład zadania paletyzacji jako manipulacji przy braku percepcji. Po lewej widoczny jest kod programu stworzony w języku MELFA-BASIC IV. Po prawej na górze - ujęcie wykonane podczas pracy robota Kawasaki, na dole - widok palety do ukończeniu zadania. Symulacja została wykonana w środowisku Cosimir.

1.2 Cele manipulacji

Chociaż pierwsze w pełni sprawne manipulatory istnieją w sektorze industrialnym od dekad, wykorzystanie ich w zadaniu manipulacji stanowi wyzwanie, podczas gdy często stosowane rozwiązania nie wykorzystują w pełni ich potencjału. Dlatego chcąc określić potrzebę implementacji omawianych tu rozwiązań warto jest przyjrzeć się celom przed, którymi staje współczesna



Rysunek 3: Przykład zadania *bin picking* wykonywanego przez robota KUKA w oparciu o dane z systemu wizyjnego. Widoczne po lewej obrazy zostały wygenerowane sztucznie jako hipotetyczne dane z kamery umieszczonej nieruchomo względem bazy robota. Od góry - ujęcie powstałe jako bezpośrednia projekcja, obraz niosący dane o głębii i obraz poddany segmentacji. Symulacja została stworzona w języku Python z wykorzystaniem biblioteki *pybullet*.

robotyka. Przede wszystkim chcemy, aby dane rozwiązanie było możliwie jak najbardziej **uniwersalne**, to znaczy działało poprawnie w różnych środowiskach, było odporne na zaszumienia i efektywne. Aby te cechy mogły zostać spełnione należy pewne kwestie muszą zostać rozwiązane, a należą do nich:

- **Zdolność do operowania wieloma różnymi obiektami.** Uniwersalność może zostać osiągnięta przez adaptację, czyli trwający w czasie proces, którego celem jest przystosowanie danego podmiotu do obserwowanego środowiska. Algorytmy działające zgodnie z tą zasadą stają się dość popularne szczególnie w ostatnich latach, a techniki wiodące prym na tym polu określane są mianem *Reinforcement Learning* (w ję-

zyku polskim przyjęły się nazwy uczenia ze wspomaganiem i uczenia z krytykiem). Nie zawsze stanowią jednak wydajne rozwiązanie zadania, głownie ze względu na wspomniany wyżej czas potrzebny do przetrenowania algorytmu. Problematyczne mogą być również dość chaotyczne, a przez to nieoptymalne i trudne do przewidzenia trajektorie jakie algorytmy tego typu generują.

- **Sterowanie we wszystkich stopniach swobody**. Manipulator przemysłowy wykorzystany w tej pracy charakteryzuje się sześcioma stopniami swobody (bez chwytaka), co sprawia, że dla niektórych pozycji zadanie kinematyki odwrotnej będzie miało więcej niż jedno rozwiązanie i wiele możliwych trajektorii, z których nie wszystkie są optymalne.

- **Odporność na zaszumione lub zniekształcone dane wejściowe**. Ograniczona rozdzielczość kamer RGBD, zaszumienie spowodowane światłem zewnętrznym i różne możliwe rozrzucenie obiektów komplikuje działanie algorytmu i aby proces przebiegał płynnie - musi zostać obsłużone.

  Do powyższej listy moglibyśmy także dopisać zdolność do przenoszenia zachowań z symulacji do rzeczywistego świata, ale ponieważ nie we wszytkich problemach stosuje się metody symulacji - nie dopisywałem ich do tej listy.

1.3 Afordancje

Zdolność oddziaływania na jakiś obiekt lub środowisko została określona mianem **afordancji** przez psychologa Jamesa J. Gibsona w 1966 roku w książce *The Senses Considered as Perceptual Systems* i opisana w artykule *The Theory of Affordance. In: Perceiving, Acting and Knowing Toward an Ecological Psychology.* Definicja ta przyjęła szerokie znaczenie w dziedzinach tj. psychologia, kognitywistyka, sztuczna inteligencja czy robotyka.

Możemy zdefiniować afordancje na przykładzie ludzi, ale człowiek ze względu na swoją złożoność często pozostaje niewdzięcznym modelem. Dlatego dla naszych rozważań posłużymy się morskim bezkręgowcem - krabem *Limulus polyphemus*. Przykład ten został opisany m.in. w *Biocybernetyce* Ryszarda Tadeusiewicza. Korzystnymi cechami tego stawonoga z naszego punktu widzenia są posiadanie ośrodkowego układu nerwowego oraz oczu, z których

impulsy jesteśmy w stanie śledzić. Tak więc, jesteśmy w stanie w przybliżeniu zobrazować jego pole widzenia.

Co wobec tego widzi krab? Przede wszystkim **obiekty** stanowiące najczęściej zagrożenie lub będące przedmiotem łowów niezależnie od tła. W warunkach laboratoryjnych możemy także osiągnąć pobudzenie neuronalne za pomocą punktowego oświetlenia lub gwałtownych zmian światła. Krab nie widzi natomiast równomiernych gradientów oświetlenia czy monotonnego pastelowego tła. Jest to biologicznie uzasadnione, ponieważ informacje te nie są dla niego w żaden sposób niezbędne do życia.

Przykład kraba, mimo że może wydawać się surowy, niesie ze sobą istotne wskazówki na temat tego co powinniśmy rozumieć przez afordancje i jaka jest ich rola. Przechodząc do chwytania, które w środowisku naturalnym jest domeną naczelnych, aby mogło ono zostać zrealizowane - wykorzystywane są informacje zawarte w obrazie. Pozwala to tłumaczyć dlaczego niektóre gatunki, przykładem tu mogą być wczesne homidy, wykształciły bardziej rozwinięty wzrok konieczny do wytypowania elementów otoczenia. Podobnie w robotyce, robot musi nauczyć się jak chwytać i operować uchwyconymi obiektami, tak aby osiągnąć wyznaczony cel. Różne przedmioty, np. młotek mogą być uchwycone na wiele różnych sposobów natomiast liczba optymalnych chwytów jest ograniczona w kontekście danego zadania.

2 Przegląd dotychczasowych rozwiązań

Problem sterowania robotami z wykorzystaniem informacji zwrotnej z otoczenia sięga początków współczesnej robotyki. Chcąc zapewnić możliwość pracy robota w środowiskach, w których nie mamy wystarczającej wiedzy na temat położenia obiektów z jakimi ma wchodzić w interakcje, konieczne jest zaimplementowanie podstawowej percepcji. Najprostszy przypadek może stanowić robot wyposażony w czujnik koloru w zadaniu sortowania kolorowych kulek. Wiele rozwiązań jednak opiera się na wykorzystaniu danych i przetworzeniu obrazu z kamery i czujników w taki sposób, aby robot był w stanie rozpoznać lub zlokalizować wskazane obiekty. Taką analizę obrazu nazywany widzeniem maszynowym lub CV od angielskich słów *Computer Vision*.

2.1 Widzenie maszynowe

W dziedzinie widzenia maszynowego wyróżniamy kilka typów problemów związanych z obrazami, takich jak: detekcja, lokalizacja i segmentacja.

- **Detekcja** dostarcza nam informację czy wskazany obiekt znajduje się na obrazie, a także w niektórych przypadkach, otrzymujemy prawdopodobieństwo jego wystąpienia. Nie zapewnia nam jednak informacji, gdzie dokładnie ten obiekt się znajduje.
- **Lokalizacja** często pojawia się w parze z detekcją. Wynika to z faktu, że gdy mamy rozpoznany dany obiekt, chcemy pozyskać także informację o jego położeniu, co otrzymujemy w postaci współrzędnych lokalizacji, a czasem także wielkości.
- **Semantyczna segmentacja** niesie nam odpowiedź o położeniu oraz kształcie obiektów. Załóżmy, że mamy trzy różne obiekty w puli do rozpoznania i wszystkie znajdują się na jednym obrazie, z czego dwa z nich są na nim po jednej sztuce, a ostatni – w dwóch. Semantyczna segmentacja umożliwi nam znalezienie położenia ich wszystkich, natomiast nadal nie wiemy czy jakiś z nich nie wystąpił w więcej niż jednej sztuce, przez co niemożliwe jest określenie ilości.
- **Segmentacja z uwzględnieniem instancji** powstała jako rozwiązanie problemu opisanego powyżej. Niesie nam odpowiedź, która oprócz podania dokładnego położenia i kształtu obiektów także uwzględnia ich ilość.

Wymieniowe wyżej problemy należą do dość złożonych zadań wymagających sporej mocy obliczeniowej. Z pomocą przychodzą algorytmy takie jak SIFT (w celu detekcji i lokalizacji danych obiektów na podstawie podobieństwa cech) czy metody uczenia nienadzorowanego (pomocne przy segmentacji). Często jednak z pomocą przychodzą sztuczne sieci neuronowe, a szczególnie popularne są sieci zawierające operację splotu, czyli sieci splotowe.

2.2 Sztuczne sieci neuronowe

Jedną z najbardziej popularnych technik w uczeniu maszynowych stanowią sieci neuronowe *(ang. ANN - Artificial Neural Network)*. Taką sieć można w uproszczony sposób przedstawić w postaci grafu, którego wierzchołki reprezentują neurony a krawędzie - połączenia między nimi, tzw. synapsy.

Sieci splotowe

Rodzaj sieci, na który szczególnie warto jest zwrócić uwagę stanowią sieci splotowe zwane częściej sieciami konwolucyjnymi, w skrócie CNN (ang. *Convolutional Neural Network*). Te sieci zawdzięczają swoją popularność przede wszystkim dzięki wykorzystaniu do problemów związanych z analizą obrazów, przy których ich możliwości przewyższają zwykłe sieci jednokierunkowe. Ich nazwa wzięła się od operacji splotu (konwolucji), która jest wykorzystywana do ekstrakcji cech przez splot obrazu z filtrem. Sieci splotowe są z powodzeniem stosowane w rozpoznawaniu, klasyfikacji i segmentacji obrazów. Ponieważ możliwe jest przeprowadzenie operacji splotu na danych o zarówno jednym, dwóch oraz trzech wymiarach, użycie sieci CNN da się poszerzyć do analizy danych takich jak dźwięk, obrazy RGBD czy chmury punktów, a to z kolei implikuje ich przydatność w robotyce.

GraspNet

Daleko idącym przykładem wykorzystania sieci neuronowych w robotyce jest projekt *GraspNet*. Celem przedsięwzięcia było przetrenowanie modelu sztucznej inteligencji na zbiorze obrazów pochodzących z kamer RGBD przedstawiających przedmioty z możliwymi chwytami wyznaczonymi analitycznie. Twórcom projektu udało się zgromadzić 97280 obrazów RGBD zawierających ponad miliard możliwych pozycji chwytu. Zbiór ten został ujawniony i opublikowany pod nazwą *GraspNet-1Billion*. W rzeczywistości *GraspNet* jest

modelem składającym się z kilku mniejszych sieci, który oprócz nich wykorzystuje funkcje tj. grupowanie, wyśrodkowywanie i filtrowanie danych. Tak więc, dla danych wejściowych w postaci chmury punktów przyporządkowuje pewną ilość chwytów jakie mogą zostać zastosowane podczas próby uchwycenia obiektów w niej przedstawionych. Możliwe chwyty zostają wyznaczone w sposób, na który mogą zostać przeprowadzone z użyciem chwytaka dwupalczastego.

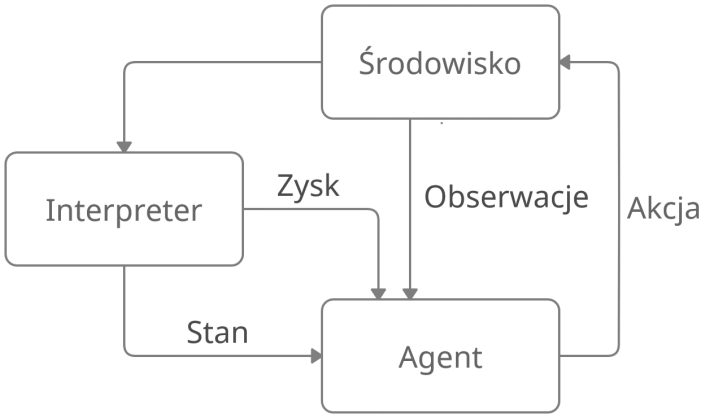
Rysunek 4: Wizualizacja możliwych chwytów wykrytych przez sieć *GraspNet*. Na niebiesko oznaczone jest położenie szczęk chwytaka.

2.3 Uczenie przez wzmacnianie

Uczenie przez wzmacnianie zwane też uczeniem z krytykiem lub *Reinforcement Learning* opisuje metody uczenia maszynowego, w którym optymalne rozwiązanie danego zadania powstaje na skutek interakcji agenta ze środowiskiem. W zastosowaniach tego podejścia w robotyce, środowiskiem jest symulacja lub rzeczywiste otoczenie, z którym agent (robot) wchodzi w interakcję. Celem jest maksymalizacja zysku ściśle związanego z pożądanym rezultatem. W ten sposób robot próbując osiągnąć jak najwyższy wynik uczy się rozwiązywać zadany problem.

Istnieje wiele wariantów algorytmów wykorzystujących *Reinforcement Learning*, które możemy podzielić ze względu na kryterium optymalizacji na:

- **Oparte na polityce**
  W tym podejściu przyjmowana jest polityka, którą należy zoptymali-



Rysunek 5: Przedstawienie zasady działania metod *Reinforcement Learning* w postaci grafu: agent podejmuje działanie w danym środowisku co jest interpretowane jako zysk i reprezentacja stanu, które trafiają z powrotem do agenta. W dalej idących implementacjach algorytmu występują także obserwacje, czyli dostarczane agentowi informacje na temat środowiska.

zować. Politykę możemy definiować jako odpowiedzialną za zachowanie agenta funkcję, która na wejściu przyjmuje obserwację, a na wyjściu zwraca akcję.

- **Oparte na funkcji wartości stanu**
  Funkcja wartości stanu $V_\pi(s)$ określa jak wysoki będzie zysk, gdy zaczniemy ze stanu s i będziemy opierać się o politykę π.

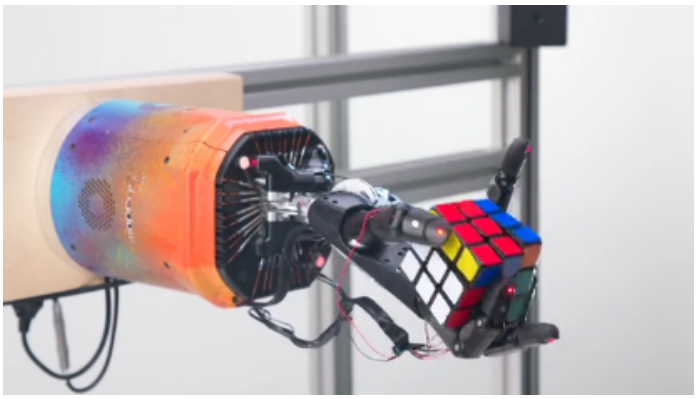
$$V_\pi(s) = \mathbb{E}[R|s_0 = s] = \mathbb{E}\left[\sum_{t=0}^{n} \gamma^t r_t | s_0 = s\right]$$

  W tym kryterium zostaje wprowadzona wartość R jako całkowity **zysk**, który jest zdefiniowany jako suma poszczególnych przyszłych zysków

$$R = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots + \gamma^n r_n = \sum_{t=0}^{n} \gamma^t r_t \quad gdzie \quad \gamma \in \langle 0, 1)$$

  Ponieważ γ jest mniejsza niż 1, odległe przyszłe zdarzenia są mniej

istotne od zdarzeń w bliskiej i niedalekiej przyszłości. Jest to istotne, ponieważ zależy nam na njaszybszym możliwym uzyskaniu zysku.



Rysunek 6: Robot wykorzystujący techniki uczenia ze wzmacnianiem podczas układania kostki Rubika. Eksperyment został opisany w publikacji *Learning Dexterous In-Hand Manipulation* i miał na celu przetrenowanie sztucznej inteligencji pod kątem umiejętności motorycznych - praca zbiorowa OpenAI.

Zalety uczenia przez wzmacnianie

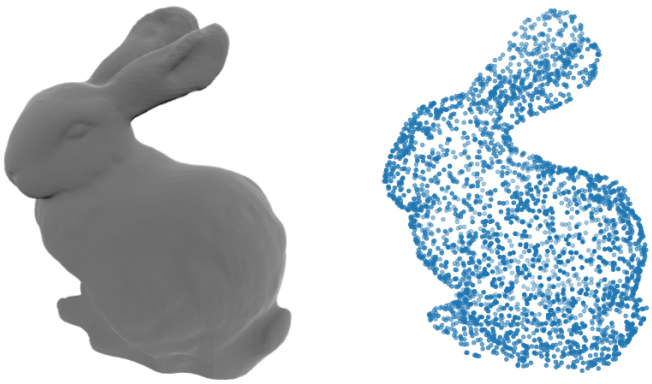
Do niewątpliwych zalet uczenia przez wzmacnianie należy zdolność do tworzenia zachowań **emergentnych**. Przez emergencję rozumiemy powstawanie zachowań, do których instrukcje nie zostały bezpośrednio udzielone, a znalezione metodą prób i błędów i następnie z powodzeniem stosowane. Drugą ważną zaletą jest jego uniwersalność - metody oparte na *Reinforcement Learning* mogą zostać z lepszym lub gorszym skutkiem użyte w większości problemów związanych z motoryką w robotyce, tj. nauka chodzenia, chwytania itp.

Wady uczenia przez wzmacnianie

Jak już zostało wspomniane we wprowadzeniu, do wad podejścia wykorzystującego uczenie przez wzmacnianie w robotyce zaliczamy czas potrzebny do przetrenowania algorytmu oraz niewydajne i często chaotyczne ruchy, które są jego rezultatem.

3 Estymacja położenia obiektu

Algorytm Iterative Closest Point (ICP) został szerzej opisany w artykule *A Method for Registration of 3-D Shapes* przez Paula Besla i Neila McKaya w 1992 roku. Wcześniej została opublikowana praca *Least-Squares Fitting of Two 3-D Point Sets* K.S Aryn at el.
Aby przejść do zadania estymacji położenia obiektu najpierw należy określić czy dysponujemy jego kształtem. Ponieważ w wielu zastosowaniach industrialnych przedstawionego algorytmu cecha ta jest wiadoma z góry, dla potrzeby tej pracy możemy przyjąć, że dysponujemy takim modelem. Mamy więc podane dwa kształty: rzeczywisty model obiektu pobrany z kamery RGBD jako chmura punktów oraz wyidealizowany model, który posłuży jako szablon.
Mamy więc dwa zbiory punktów, z których każdy opisany jest przez współrzędne x, y, z.



Rysunek 7: Model obiektu i utworzona sztucznie chmura punktów.

3.1 Wyznaczenie najbliższego punktu

Przypadek 1: dwa punkty
Odległość $d(p_1, p_2)$ między dwoma punktami $p_1 = (x_1, y_1, z_1)$ i $p_2 = (x_2, y_2, z_2)$ jest dana w metryce euklidesowej wzorem:

$$d(p_1, p_2) = ||p_1 - p_2|| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Przypadek 2: punkt i zbiór punktów
Załóżmy, że A jest zbiorem n punktów oznaczonych jako a_i. Odległość między punktem p a zbiorem punktów A jest równa:

$$d(p, A) = \min_{i \in 1,\ldots,n} d(p, a_i)$$

Wówczas najbliższy punkt spełnia warunek $d(p, a_i) = d(p, A)$

Przypadek 3: punkt i odcinek
Załóżmy, że l jest odcinkiem łączącym punkty p_1 i p_2 Odległość między punktem p a odcinkiem l wynosi:

$$d(p, l) = \min_{u+v=1} ||up_1 + vp_2 - p||$$

gdzie $u \in [0, 1]$ i $v \in [0, 1]$

Przypadek 4: punkt i figura płaska
Załóżmy, że t jest trójkątem opisanym przez trzy punkty leżące na jego wierzchołkach: $p_1 = (x_1, y_1, z_1)$, $p_2 = (x_2, y_2, z_2)$ oraz $p_3 = (x_3, y_3, z_3)$ Odległość między punktem p a trójkątem t jest dana:

$$d(p, t) = \min_{u+v+w=1} ||up_1 + vp_2 - p + wp_3 - p||$$

gdzie $u \in [0, 1]$ i $v \in [0, 1]$

3.2 Transformacje

Wspomniane algorytmy wyznaczenia najbliższego punktu mogą być rozwinięte do przestrzeni n-wymiarowych. Aby dokonać takiej generalizacji do przestrzeni trójwymiarowej konieczne jest zastosowanie macierzy obrotu - rotacji **R** i wektora przesunięcia liniowego - translacji **T**.

$$R = \begin{bmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{bmatrix} \qquad T = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}$$

3.2.1 Macierz rotacji

W zależności od tego jakim sposobem chcemy wykonywać działania mamy do wyboru trzy równoważne opcje obrotu danego kształtu w przestrzeni. Możemy w tym celu posłużyć się macierzą opartą o kąty obrotu wokół wszystkich trzech osi w układzie kartezjańskim, macierzą opartą o oś obrotu i kąt o jaki dany obiekt ma zostać obrócony lub kwaternionem.

3.2.2 Kąty Eulera

Odnosząc się do rysunku, w pierwszym przypadku po lewej stronie mamy dane trzy osie układu kartezjańskiego i wykonujemy składowe obroty wokół każdej z nich.

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

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

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

Mamy więc trzy obroty, które składają się na wynikową rotację.

$$R = R_z(\gamma)R_y(\beta)R_x(\alpha)$$

Macierz rotacji przyjmuje wtedy postać

$$R = \begin{bmatrix} \cos\gamma\cos\beta & \cos\gamma\sin\beta\sin\alpha - \sin\gamma\cos\alpha & \cos\gamma\sin\beta\cos\alpha + \sin\gamma\sin\alpha \\ \sin\gamma\cos\beta & \sin\gamma\sin\beta\sin\alpha + \cos\gamma\cos\alpha & \sin\gamma\sin\beta\cos\alpha - \cos\gamma\sin\alpha \\ -\sin\beta & \cos\beta\sin\alpha & \cos\beta\cos\alpha \end{bmatrix}$$

3.2.3 Oś obrotu i kąt

W drugim - dysponujemy osią pożądanej rotacji, więc wykonujemy już tylko jeden obrót. Oś obrotu możemy przedstawić w postaci wektora jedenostkowego $\mathbf{u} = [u_x, u_y, u_z]$ takiego, że $u_x^2 + u_y^2 + u_z^2 = 1$ Dla ułatwienia oznaczmy

$c = \cos\theta$ i $s = \sin\theta$

$$R = \begin{bmatrix} c + u_x^2(1-c) & u_x u_y(1-c) - u_z s & u_x u_z(1-c) + u_y s \\ u_y u_x(1-c) - u_z s & c + u_y^2(1-c) & u_y u_z(1-c) + u_x s \\ u_z u_x(1-c) - u_z s & u_z u_y(1-c) - u_x s & c + u_z^2(1-c) \end{bmatrix}$$

3.2.4 Kwaternion

W trzecim przypadku posługujemy się kwaternionem, który możemy zapisać w postaci sumy algebraicznej jako

$$q = a \cdot \mathbf{e} + b \cdot \mathbf{i} + c \cdot \mathbf{j} + d \cdot \mathbf{k}$$

gdzie $a, b, c, d \in \mathbb{R}$ zaś **e, i, j, k** to pewne jednostki urojone, między którymi zachodzi zależność $i^2 = j^2 = k^2 = -1$.

Wówczas transformację tą możemy zapisać równoważnie w postaci macierzy:

$$R = \begin{bmatrix} a^2 + b^2 - c^2 - d^2 & 2(bc - ad) & 2(bd + ac) \\ 2(bc + ad) & a^2 + c^2 - b^2 - d^2 & 2(cd - ab) \\ 2(bd - ac) & 2(cd * ab) & a^2 + d^2 - b^2 - c^2 \end{bmatrix}$$

3.3 Rozkład macierzy według wartości osobliwych

Celem rozkładu według wartości osobliwych, w skrócie rozkładu SVD (ang. *Singular Value Decomposition*) w proponowanym algorytmie jest znalezienie macierzy rotacji, która stanowi transformację między chmurą punktów a szablonem, do którego próbujemy ją dopasować. Umożliwi to późniejsze uchwycenie przedmiotu przez chwytak manipulatora, ponieważ szablon jest obiektem, który jest znany, a więc możemy zamodelować poprawny proces chwytania.

Opisany w ten sposób problem stanowi jedynie pewien szczególny przypadek zastosowania rozkładu SVD. Ogólnie rzecz biorąc, SVD jest metodą pozwalającą na znalezienie rzędu macierzy, co teoretycznie może zostać wyznaczone metodą eliminacji Gaussa. To podejście jest jednak niepraktyczne, gdy ma zostać zrealizowane za pomocą metod numerycznych. Błędy biorą się często z niedoszacowań z powodu zaokrąglania wartości. Na przykład gdy rząd macierzy A jest niedoszacowany a U stanowi obliczoną numerycznie postać schodkową, wówczas możliwe jest, że U będzie miało niepoprawną liczbę niezerowych wierszy.

Najprostszym sposobem na zrozumienie rozkładu według wartości osobliwych jest zapisanie macierzy rzeczywistej A w postaci iloczynu trzech czynników, z których środkowy to macierz z pewnymi wartościami rosnącymi wzdłuż przekątnej. Taki rozkład jest zawsze możliwy, co można zapisać jako

$$A = UDV^T$$

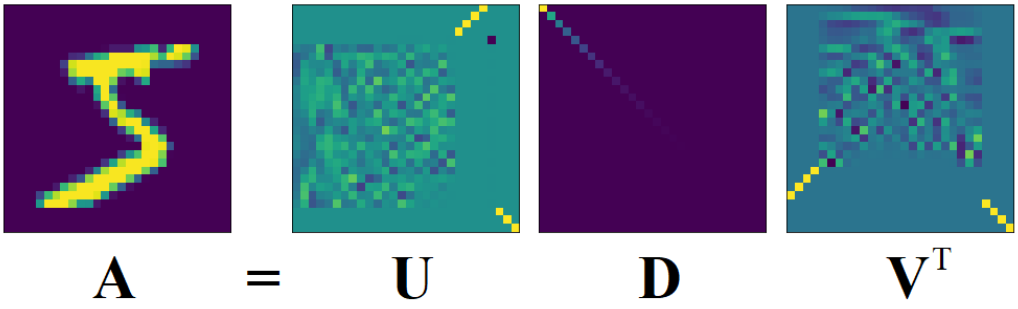
Zakładając, że A jest macierzą $m \times n$, wówczas U stanie się macierzą ortogonalną o wymiarach $m \times m$, V - macierzą ortogonalną o wymiarach $n \times n$, a D macierzą $m \times n$, której wartości nieleżące na przekątnej są równe 0, a wartości na przekątnej spełniają zależność

$$\sigma_1 \geqslant \sigma_2 \geqslant \ldots \geqslant \sigma_n \geqslant 0$$

$$D = \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \\ & & & \end{bmatrix}$$

Wartości σ_i określone przez rozkład są różne i są nazywane **wartościami osobliwymi** macierzy A. Liczba niezerowych wartości osobliwych jest równa

rzędowi macierzy A, a ich wielkość stanowi miarę jak bardzo ta macierz różni się od macierzy niższego rzędu. Z kolei kolumny macierzy U stanowią **lewostronne wektory osobliwe** a kolumny V - **prawostronne wektory osobliwe**.



Rysunek 8: Graficzne przedstawienie zasady działania SVD na przykładzie obrazu ręcznie pisanej cyfry pięć o wymiarach 28×28 pikseli. Kolory zbliżone do fioletu reprezentują niskie wartości, a jaskrawe zbliżone ku żółci - wysokie. Możemy zauważyć, że sposób w jakim na obrazie macierzy D układają się coraz ciemniejsze kolory jest zgodny z założonym wzorem $\sigma_i \geqslant \sigma_{i+1} \geqslant 0$.

Twierdzenie związane z SVD zakłada, że jeśli A jest macierzą o wymiarach $m \times n$ możliwy jest jej rozkład według wartości osobliwych. Aby to wykazać, posłużmy się macierzą A^TA. Jest to macierz symetryczna, toteż wszystkie jej wartości własne są liczbami rzeczywistymi i istnieje taka macierz ortogonalna V, która ją diagonalizuje. Co więcej, jej wartości własne muszą być nieujemne. Aby to zobaczyć, niech λ będzie wartością własną A^TA a $\mathbf{x}$ wektorem własnym związanym z λ. Wynika z tego, że

$$||A\mathbf{x}||^2 = \mathbf{x}^T A^T A\mathbf{x} = \lambda \mathbf{x}^T\mathbf{x} = \lambda||\mathbf{x}||^2$$

Aby wyznaczyć λ z powyższego równania wykonujemy dzielenie

$$\lambda = \frac{||A\mathbf{x}||^2}{||\mathbf{x}||^2} \geqslant 0$$

Podniesienie do kwadratu dowodzi, że λ jest dodatnia. Załóżmy, że kolumny macierzy V zostały uporządkowane tak, że odpowiednie wartości własne spełniają zależność

$$\lambda_1 \geqslant \lambda_2 \geqslant \ldots \geqslant \lambda_n \geqslant 0$$

Wartości własne macierzy A są dane

$$\sigma_i = \sqrt{\lambda_i} \quad i = 1, 2, ..., n$$

Niech r oznacza rząd macierzy A. Należy zauważyć, że macierz $A^T A$ też będzie rzędu r. Ponieważ $A^T A$ jest symetryczna jej rząd będzie równy liczbie niezerowych wartości własnych.

$$\lambda_1 \geqslant \lambda_2 \geqslant \ldots \geqslant \lambda_r \geqslant 0 \qquad \lambda_{r+1} = \lambda_{r+2} = \ldots = \lambda_n \geqslant 0$$

Ta sama zależność zachodzi dla wartości osobliwych

$$\sigma_1 \geqslant \sigma_2 \geqslant \ldots \geqslant \sigma_r \geqslant 0 \qquad \sigma_{r+1} = \sigma_{r+2} = \ldots = \sigma_n \geqslant 0$$

Dla naszych dalszych rozważań konieczne jest przedstawienie poszczególnych czynników U, D i V w rozbiciu na mniejsze macierze, z których są złożone. Zaczynając od macierzy V mamy

$$V_1 = (\mathbf{v}_1, \mathbf{v}_2, ...\mathbf{v}_r), \quad V_1 = (\mathbf{v}_{r+1}, \mathbf{v}_{r+2}, ...\mathbf{v}_n)$$

$$D_1 = \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_r \end{bmatrix} \qquad D = \begin{bmatrix} D_1 & 0 \\ 0 & 0 \end{bmatrix}$$

Wektory V_2 stanowią wektory własne macierzy $A^T A$ związane z $\lambda = 0$

$$A_T A \mathbf{x}_i = 0 \quad i = r+1, r+2, ..., n$$

$$AV_2 = 0$$

Ponieważ V jest macierzą ortogonalną możemy zapisać

$$I = VV^T$$

$$A = AI = AVV^T$$

Zostało nam jeszcze skonstruować macierz ortogonalną U o wymiarach $m \times m$ taką, że

$$A = UDV^T$$

Co po przeniesieniu V na lewą stronę możemy równoważnie zapisać jako

$$AV = UD$$

Porównując pierwsze r kolumn po obu stronach możemy zauważyć

$$A\mathbf{v}_i = \sigma_i \mathbf{u}_i \quad i = 1, 2, ..., n$$

Zatem jeśli, ustalimy

$$\mathbf{u}_i = \frac{1}{\sigma_i} A\mathbf{v}_i \quad i = 1, 2, ..., n$$

$$U_1 = (\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_r)$$

To będzie wynikać z tego

$$AV_1 = U_1 D_1$$

$$\mathbf{u}_i^T \mathbf{u}_j = \left(\frac{1}{\sigma_i}\mathbf{v}_i^T A^T\right)\left(\frac{1}{\sigma_j} A\mathbf{v}_j\right) = \frac{1}{\sigma_i \sigma_j}\mathbf{v}_i^T \left(A^T A \mathbf{v}_j\right) = \frac{\sigma_j}{\sigma_i}\mathbf{v}_i^T \mathbf{v}_j = \delta_i j$$

Jeśli $\mathbf{u}_1, ..., \mathbf{u}_m$ tworzy bazę ortonormalną dla $\mathbb{R}^m$ to U jest macierzą ortogonalną. Wówczas ostatnim krokiem jest wykazanie, że A rzeczywiście jet równe UDV^T

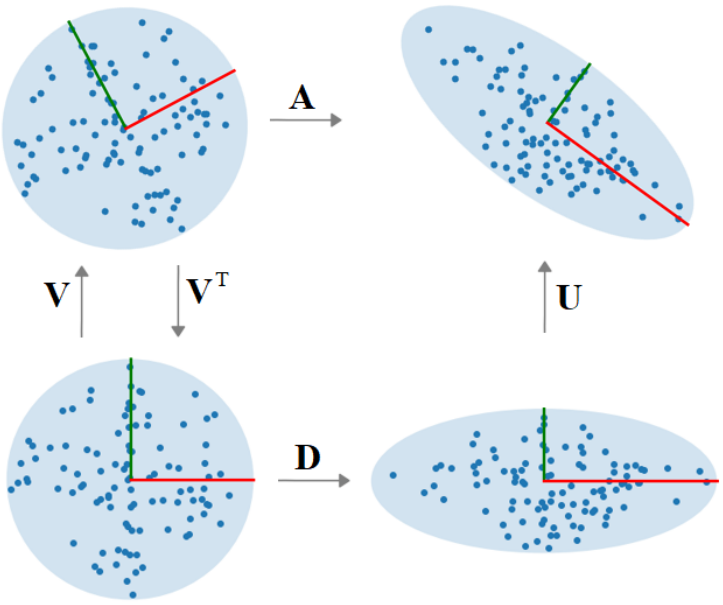
$$UDV^T = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} D_1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} = U_1 D_1 V_1^T = A V_1 V_1^T = A$$

Interpretacja geometryczna rozkładu SVD

W interpretacji geometrycznej wartości osobliwe mogą być rozumiane jako długości półosi elipsy na płaszczyźnie, co pokazano na rysunku. Wówczas zakładając, że A stanowi przekształcenie liniowe przestrzeni - macierze U i V reprezentują rotacje i odbicie tej przestrzeni, a D odpowiada za skalowanie. Innymi słowy, rozkład SVD rozkłada dowolne przekształcenie liniowe na złożenie funkcji trzech przekształceń: obrotu lub odbicia (V), skalowania (D) i kolejnego obrotu lub odbicia (U). W szczególności, jeśli wyznacznik poddawanej przekształcaniu macierzy jest dodatni, wówczas macierze U i V mogą

odpowiadać zarówno za rotację jak i odbicie. Jeśli wyznacznik jest ujemny tylko jedna z nich odpowiada za odbicie, jeśli jest równy zero - odbicia nie ma wcale.

Taka interpretacja może również zostać uogólniona do n-wymiarowych przestrzeni euklidesowych. Wówczas wartości osobliwe dowolnej macierzy kwadratowej n × n staną się długościami półosi n-wymiarowej elipsoidy. Wartości osobliwe zawierają w sobie informację o długościach a wektory osobliwe - o kierunku półosi. Nie jest to jednak zbyt praktyczne rozwiązanie, dlatego mówiąc o rozkładzie SVD w proponowanym algorytmie będziemy posługiwać się uproszczonym SVD. Przede wszystkim, zakładamy, że dysponujemy szablonem, który ma dokładnie te same wymiary co szukany obiekt oraz szukany przedmiot jest bryłą sztywną (więc również jego szablon traktujemy w ten sam sposób). Te założenia sporo upraszczają. Przede wszystkim, odnosząc się do praktycznego zastosowania możemy pominąć operację skalowania.



Rysunek 9: Interpretacja geometryczna SVD macierzy A o wymiarach 2×2. Celem jest przekształcenie przestrzeni pokazanej w lewym górnym rogu, tak aby uzyskać obraz widoczny w prawym górnym. Transformacja zostaje rozłożona przez SVD na trzy etapy: rotację, skalowanie i ponowną rotację.

3.4 Algorytm Iterative Closest Point

Mając teoretyczne podstawy za sobą możemy przejść do implementacji końcowego programu stworzonego w oparciu o algorytm ICP (ang. *Iterative Closest Point*). Dla danej chmury punktów jego celem jest znalezienie położenia i orientacji obiektu, który przedstawia. W tej sytuacji dane wejściowe stanowią: chmura punktów i szablon szukanego obiektu, do którego chcemy dopasować widoczny przedmiot. Przez dopasowanie rozumiemy znalezienie transformacji (obrotu i przesunięcia) wykrytego obiektu względem początkowego położenia szablonu. Tak więc, mamy dwa zbiory punktów: p i p' w postaci macierzy o wymiarach 3 $\times$ n. Równanie opisujące przejście punktu z jednego zbioru do drugiego wygląda następująco

$$p' = Rp + T + N$$

Gdzie R jest macierzą rotacji o wymiarach 3 $\times$ 3, T - wektorem przesunięcia o wymiarach 3 $\times$ 1 a N - wektorem szumu. Zakładamy też, że obrót odbywa się wokół początku układu. Chcemy dobrać takie R i T, aby zminimalizować wyrażenie

$$d^2(p, p') = \sum_{i=1}^{n} ||p'_i - (Rp_i + T)||^2$$

Niech

$$p = \frac{1}{n} \sum_{i=1}^{n} p_i$$

$$p' = \frac{1}{n} \sum_{i=1}^{n} p'_i$$

Wówczas q i q' będą zbiorami odległości poszczególnych punktów w zbiorach p i p' od ich centroidów

$$q_i = p_i - p$$

$$q'_i = p'_i - p'$$

Mamy

$$d^2(q, q') = \sum_{i=1}^{n} ||q'_i - Rq_i||^2$$

Następnym krokiem jest wyznaczenie macierzy 3×3

$$H = \sum_{i=1}^{n} q_i q_i'^T$$

I rozłożenie jej na wartości osobliwe

$$H = UDV^T$$

Ponieważ, zgodnie z tym co zostało wspomniane podczas interpretacji geometrycznej rozkładu SVD, macierz D jest odpowiedzialna za skalowanie - podczas wyznaczenia wynikowej rotacji możemy ją odrzucić. W ten sposób otrzymujemy wyrażenie

$$X = VU^T$$

Jeśli wyznacznik $det\ X = 1$, X stanowi macierz obrotu. W przeciwnym wypadku, gdy mamy do czynienia z wyznacznikiem $det\ X$= -1 algorytm nie zwraca rozwiązań, ale ten przypadek z reguły nie występuje.

3.5 Implementacja algorytmu

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import MinMaxScaler
import random
from math import *
import trimesh
import os
import icp
import test

mesh = trimesh.load_mesh('bunny.stl')

density = 3000
dim = 3
translation = 0.5
rotation = 0.5
```

```
A = mesh.sample(density)
scaler = MinMaxScaler(feature_range=(0, 1))
A = scaler.fit_transform(A)

B = mesh.sample(density)
B = scaler.fit_transform(B)

t = np.random.rand(dim) * translation
B += t

R = test.rotation_matrix(np.random.rand(dim),
np.random.rand() * rotation)
B = np.dot(R, B.T).T

iterations = 15

for i in range(iterations):
  T, distances, iters = icp.icp(B, A, max_iterations=(i+1))

  C = np.ones((density, 4))
  C[:, 0:3] = B
  C = np.dot(T, C.T).T

  fig = plt.figure(figsize=(9, 9))
  ax = Axes3D(fig)
  ax.scatter(A[:, 0], A[:, 1], A[:, 2], c='b')
  ax.scatter(C[:, 0], C[:, 1], C[:, 2], c='r')

  fig.savefig('icp{}.png'.format(i))
```

4 Bibliografia

1. *Legged Robots That Balance* - Marc H. Raibert
2. *Learning Dexterous In-Hand Manipulation* - OpenAI
3. *The Senses Considered as Perceptual Systems* - James J. Ginson

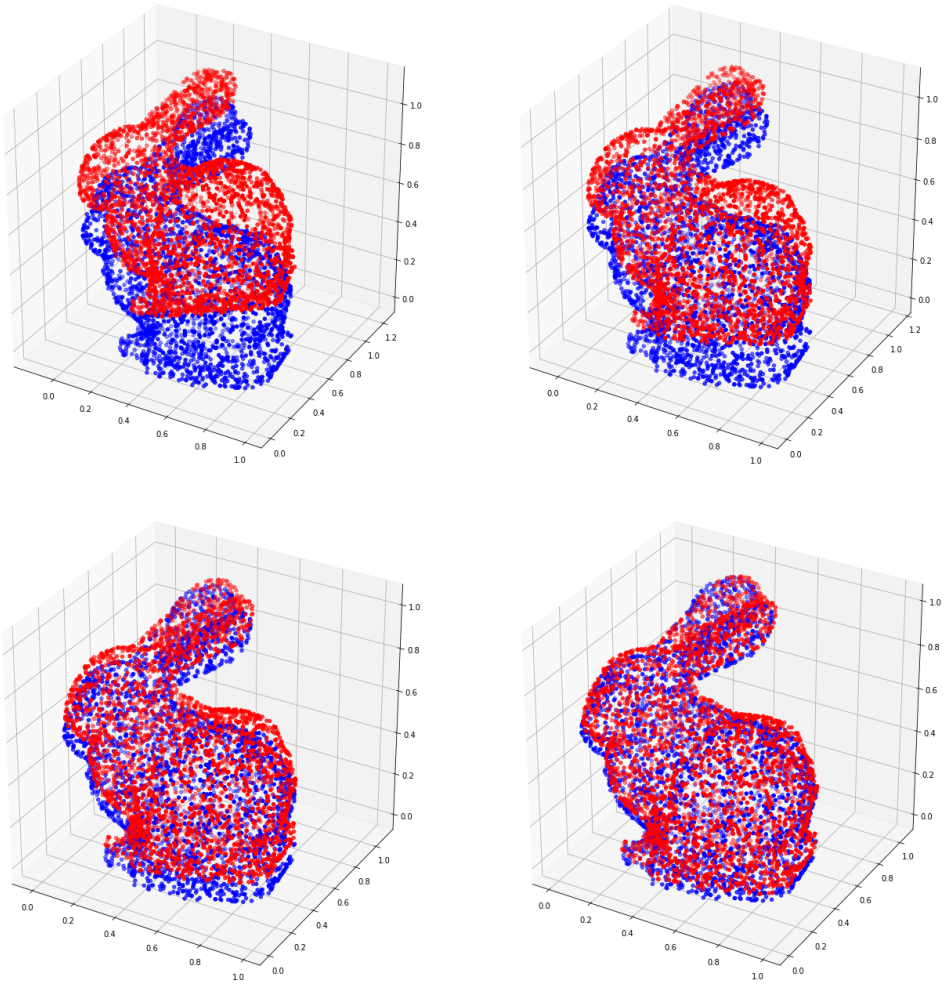
4. *Biocybernetyka* - Ryszard Tadeusiewicz

5. *Linear Algebra with Applications* - Steven J. Leon

6. *A Singularly Valuable Decomposition* - Dan Kalman

7. *A Method for Registration of 3-D Shapes* - Paul J. Besl, Neil D. McKay

8. *Least-Squares Fitting of Two 3-D Point Sets* - K.S. Arun, T.S. Huang, S.D. Blostein



Rysunek 10: Działanie algorytmu ICP na przykładzie obiektu w kształcie królika.