Politechnika Krakowska

Wydział Fizyki, Matematyki i Informatyki

Praca magisterska

Środowisko do symulacji systemów ewolucyjnych w wirtualnym świecie 2D.

Autor:
Jarosław Sporysz

Promotor:
dr inż. Radosław Kycia

9 września 2015

Spis treści

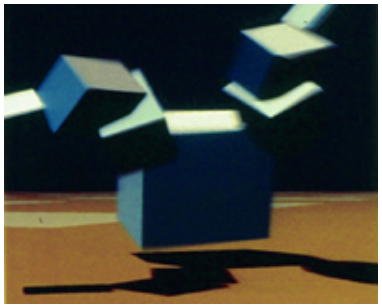
Wprowadzenie

Rozdział 1

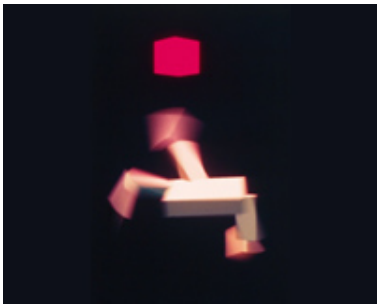
Inspiracja i rys historyczny

Bezpośrednią inspiracją do stworzenia środowiska symulacji robotyki ewolucyjnej były prace Karla Simsa [7][8][9] sprzed 20 lat, w których badał on możliwość użycia symulacji opartej na odwzorowaniu procesów fizycznych zachodzących w rzeczywistym świecie do przeprowadzenia sztucznej ewolucji wirtualnych układów mechanicznych, imitujących pod kątem budowy i zachowania prawdziwe organizmy biologiczne.
Opisał on w pracach na ile nienadzorowany przez człowieka system komputerowy zaprogramowany do łączenia ze sobą na różne sposoby mechanicznych bloków w postaci elementów konstrukcyjnych, silników, czujników i układów logicznych jest w stanie znaleźć najbardziej optymalny kształt, budowę, metodę ruchu i konfigurację coraz to nowszych i bardziej skomplikowanych mechanicznie układów - „wirtualnych stworzeń" - do efektywnego wykonywania konkretnych zadań, jakie Sims przed nimi postawił.
Rezultatem był otrzymany przez niego szereg sztucznych organizmów umieszczonych na rysunku 1.1, w których poprzez proces ewolucji udało się wytworzyć ich kształt, sposób zachowania i reakcję na otaczające je środowisko tak, by nauczyć je pływać, podążać za ruchomych celem, skakać, a nawet konkurować w walce o dostęp do wybranego obiektu.

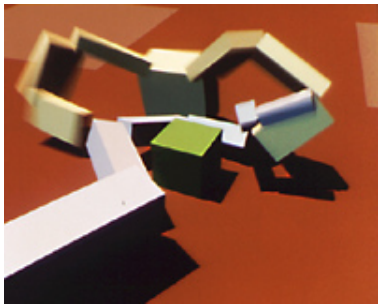
(a) Układ pływający



(b) Układ podskakujący



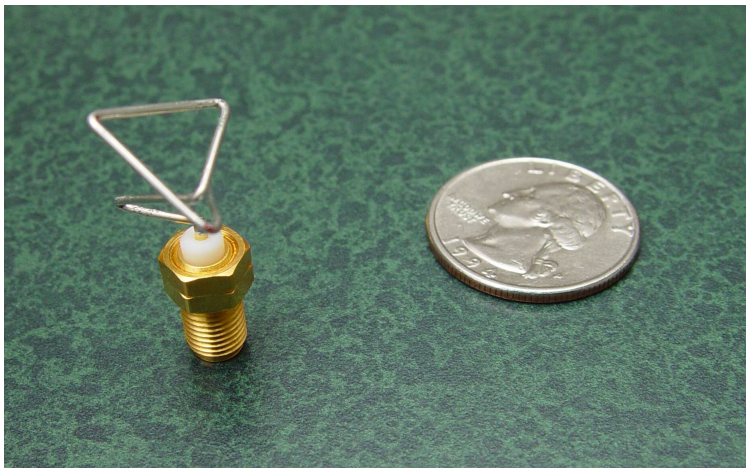
(c) Układ podążający za celem



(d) Układy konkurujące o piłkę

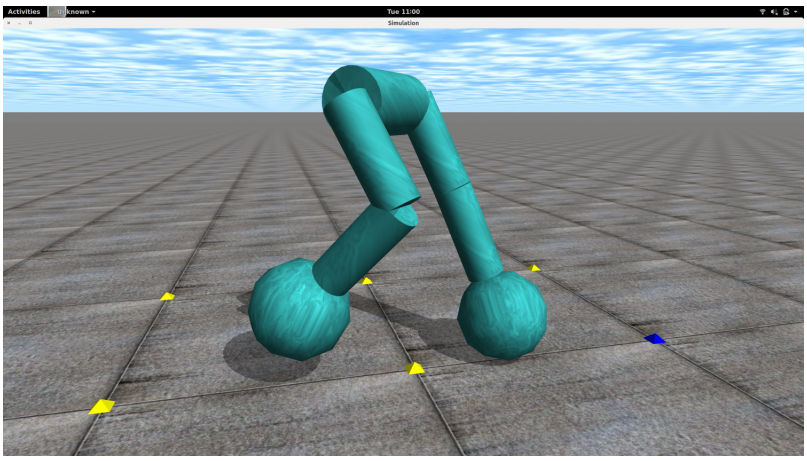
Rysunek 1.1: Sztuczne organizmy otrzymane przez Karla Simsa w procesie ewolucji wspomaganej komputerowo w 1994 roku. Rysunki pochodzą z [7]

Kolejnym przykładem inspirującym do badań nad systemami ewolucyjnymi była wynaleziona w 2006 roku przez naukowców z NASA niekonwencjonalnych kształtów antena realizująca trudny dla człowieka z punktu widzenia projektowego problem inżynierski. Naukowcy szukali optymalnego kształtu anteny [10], która byłaby w stanie pracować w wyjątkowych warunkach radiacyjnych, mając cały czas na względzie miniaturyzację anteny. W tym celu postanowili zaprząc do pracy algorytmy ewolucyjne.
Efektem tego było znalezienie kształtu anteny (zob. rysunek 1.2) przypominającej spinacz biurowy, która sprawdziła się w warunkach w których została umieszczona bez zarzutu. Co ciekawe, antena po zaprezentowaniu inżynierom specjalizującym się w projektowaniu tego typu urządzeń wydała się w swoim kształcie wręcz niedorzeczna. Po przeprowadzeniu testów i wprowadzeniu do użycia okazała się jednak spełniać swoje zadanie dużo lepiej niż wcześniejsze tego typu urządzenia, zaprojektowane bezpośrednio przez człowieka.



Rysunek 1.2: Niekonwencjonalna antena zaprojektowana przez NASA, otrzymana dzięki zastosowaniu algorytmów genetycznych. Rysunek pochodzi z [10]

Bardzo inspirujące okazały się również wyniki badań w dziedzinie robotyki ewolucyjnej, prowadzonych przez naukowców z Uniwersytetu w Teksasie [11], które pozwoliły na znalezienie zależności między występowaniem naturalnych katastrof a prędkością postępu ewolucji. Naukowcy przeprowadzili symulację dwunożnego układu mechanicznego sprzężonego ze sztuczną siecią neuronową (zob. rysunek 1.3), któremu postawili dwa kolejne cele: utrzymanie postawy stojącej przez minimum 15 sekund, a następnie naukę poruszania się.

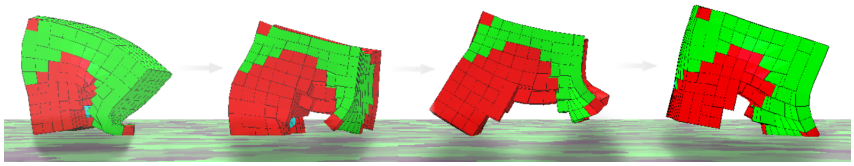


Rysunek 1.3: Dwunogi robot wyposażony w sieć neuronową poddany optymalizacji z użyciem algortmów ewolucyjnych w celu nauki utrzymania pozycji stojącej i chodzenia. Rysunek pochodzi z [11]

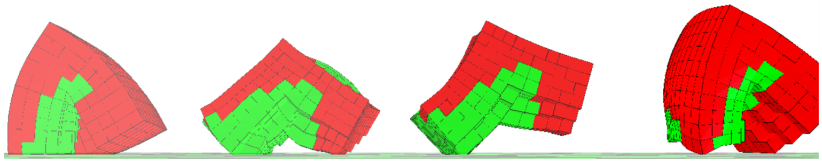
Symulowane unicestwienie 90% populacji po kilkuset generacjach ewolucji, umożliwi-

ło zwiększenie prędkości nauki robotów, co spowodowane było eliminacją puli genetycznej osobników słabiej przystosowanych do realizacji postawionego przed nimi zadania.

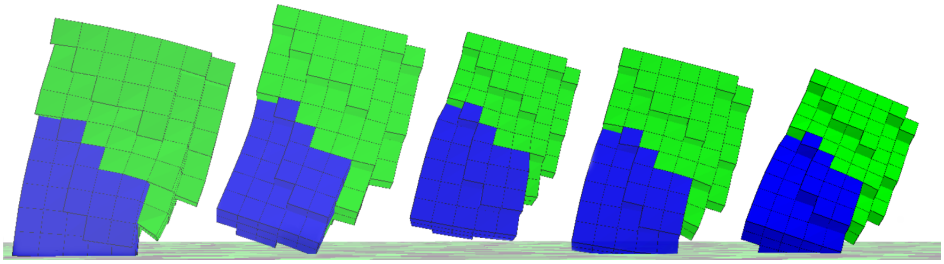
Wynikiem ostatnich badań w tej dziedzinie są stworzone przez naukowców z Uniwersytetu Cornella symulacje ewolucji robotów których budowę oparto nie jak dotychczas na sztywnych elementach konstrukcyjnych lecz na strukturze miękkiej, imitującej tkankę mięśniową[35]. Ewolucji poddano nie tylko proces kontroli elementów napędowych robotów (kurczenia i rozciągania „mięśni") ale również ich strukturę morfologiczną, pozwalając na ewolucję kształtu robotów składającego się z sześciennych bloków kilku typów, imitujących kurczliwo-rozciągliwą tkankę mięśniową oraz podtrzymującą ją strukturę sztywną imitującą kości. W efekcie otrzymano szereg wirtualnych robotów które w wielu przypadkach wytworzyły sposób ruchu do złudzenia przypominający organizmy biologiczne (rysunki 1.4 - 1.7).



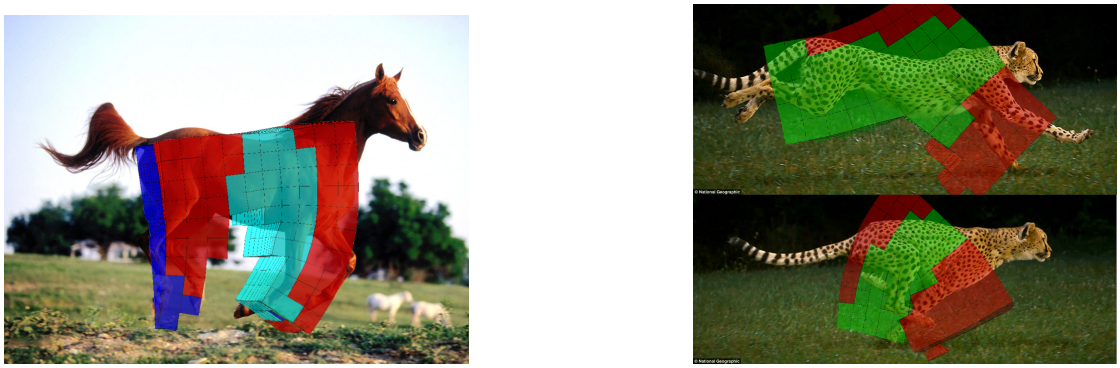
Rysunek 1.4: Robot biegnący. Rysunek pochodzi z [36]



Rysunek 1.5: Robot chodzący. Rysunek pochodzi z [36]



Rysunek 1.6: Robot skaczący. Rysunek pochodzi z [36]



Rysunek 1.7: Roboty które poprzez ewolucję osiągnęły sposób biegu przypominający ten obserwowany u biegnących koni i lampartów. Rysunki pochodzą z [36]

Czytelnik chcący poznać szerzej stan badań dotyczących robotyki ewolucyjnej i darwinistycznego podejścia do sztucznej inteligencji może w tym celu sięgnąć do pracy [18] zawierającej ogólny przegląd dziedziny.

Rozdział 2

Cel i zakres pracy

Poniższa praca jest multidyscyplinarna i łączy w sobie elementy rzeczywistych symulacji fizycznych, robotyki ewolucyjnej oraz zastosowania sztucznych sieci neuronowych. Te zagadnienia zostaną opisane szczegółowo w kolejnych rozdziałach pracy.
Do celów pracy należą:

- Stworzenie silnika symulacji dynamiki bryły sztywnej 2D umożliwiającego symulację złożonych układów mechanicznych.
- Stworzenie środowiska umożliwiającego budowę i symulację ewolucji autonomicznych układów mechanicznych oraz optymalizację ich budowy i sposobu zachowania, w celu realizacji przez nie określonych celów, łatwo definiowalnych przez twórcę środowiska ewolucji.
- Przeprowadzenie serii symulowanych ewolucji mających na celu zbadanie możliwości zdobycia i optymalizacji przez sztuczne układy robotyczne umiejętności chodzenia w środowisku z przeszkodami, pełzania, bezkolizyjnego latania i śledzenia ruchomego celu.

Rozdział 3

Metodyka

Do realizacji tak postawionego celu użyto symulacji komputerowej, z zastosowaniem specjalnie do tego celu stworzonego silnika symulacji dynamiki bryły sztywnej 2D, algorytmów ewolucyjnych i sieci neuronowych.
Algorytmów ewolucyjnych użyto do optymalizacji morfologii (budowy szkieletu) maszyn oraz parametrów sterujących przebiegiem pracy programu sterującego a także doborem parametrów urządzeń napędowych. Optymalizacji poddano również parametry (wagi) sztucznych sieci neuronowych, które zastosowano w pracy jako element łączący część sensoryczną maszyn z ich napędami, wprowadzając w ten sposób element odpowiedzialny za decyzyjność maszyny inspirowany mechanizmem używanym do tego celu przez organizmy biologiczne [24].

Powodem dla którego postanowiono stworzyć silnik symulacji fizycznej od podstaw była chęć lepszego zrozumienia jego budowy od strony zarówno teoretyczno-fizycznej jak i od strony zastosowanych w nim mechanizmów numerycznych, co dałoby większą kontrolę nad ścisłą integracją silnika symulacji z częścią odpowiedzialną za robotykę i pozwoliło w miarę potrzeb na łatwiejsze wprowadzanie zmian w silniku symulacji, np. w celu wprowadzenia nowych lub częściowo zmodyfikowanych typów układów napędowych maszyn czy nowego rodzaju przegubów.

Kolejnym z powodów była chęć przeanalizowania poziomu dyssypacji jako miary poprawności implementacji stworzonego silnika symulacji i porównania wyników z najpopularniejszym tego typu silnikiem symulacji o otwartym kodzie: Box2D, którego model numeryczny przyjęto jako bazowy przy tworzeniu silnika symulacji do poniższej pracy (więcej na temat porównania obu silników w rozdziale 5.3).
Z kolei w celu wprowadzenia możliwości programistycznej kontroli i sterowania elementami maszyn stworzono i zaimplementowano prosty język programowania typu assembler.
Poniższą pracę podzielono na 3 główne części:

- „Część fizyczną", w której opisano podstawy teoretyczne i mechanizmy numeryczne zastosowane podczas budowy silnika symulacji fizycznej 2D.
- „Wirtualne środowisko do konstrukcji i symulacji robotów", opisującą stworzone na bazie silnika symulacji fizycznej środowisko do konstrukcji i optymalizacji au-

tonomicznych układów mechanicznych z użyciem algorytmów ewolucyjnych.

- „Część ewolucyjną", opisującą budowę systemu symulacji robotyki ewolucyjnej, zawierającą analizę wyników przeprowadzonych symulacji ewolucji kilku układów mechanicznych przeznaczonych do wykonywania 3 wybranych zadań.

Część fizyczna

Rozdział 4

Model teoretyczny bryły sztywnej 2D

W tym rozdziale wprowadzimy podstawowe pojęcia dotyczące bryły sztywnej, więzów, kinematyki i dynamiki ruchu, które będą nam potrzebne w dalszej części do budowy silnika symulacji fizycznej, podczas implementacji numerycznego rozwiązania problemu kolidujących brył sztywnych z uwzględnieniem tarcia oraz problemu symulacji ruchu układów mechanicznych zbudowanych z brył sztywnych połączonych przegubowo.

4.1 Więzy i kinematyka bryły sztywnej

Liczbą **stopni swobody**[2] układu fizycznego określamy liczbę zmiennych których potrzeba do opisania aktualnego stanu układu. Dla przykładu pozycję układu 3 płaskich brył sztywnych poruszających się swobodnie po płaszczyźnie opiszemy przy pomocy 9 zmiennych:

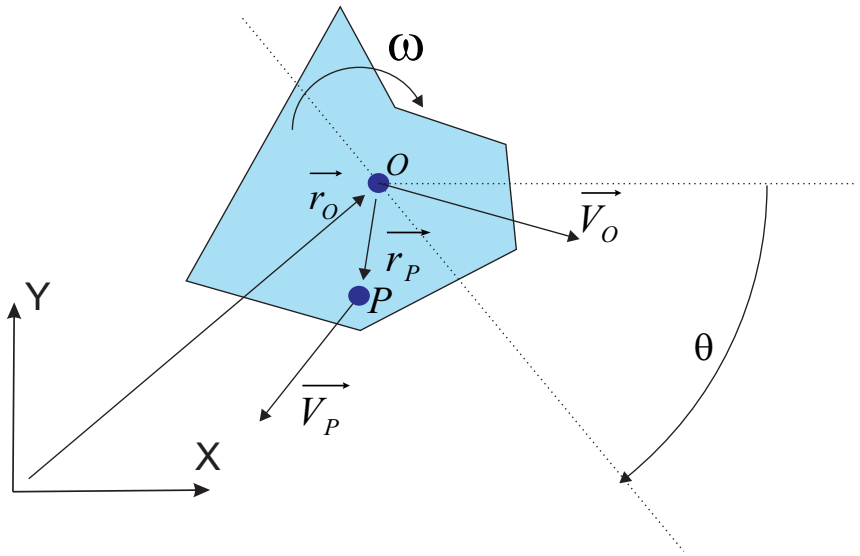
$$r(x_1, y_1, \theta_1, x_2, y_2, \theta_2, x_3, y_3, \theta_3), \tag{4.1}$$

potrzebując dla każdej z brył dwóch zmiennych (x_n, y_n) wskazujących pozycję bryły i zmiennej θ_n wskazującej wychylenie bryły od osi współrzędnych. Możemy stan takiego układu ciał zapisać w postaci tzw. **współrzędnych uogólnionych**, potrzebując znać jedynie liczbę stopni swobody, co upraszcza opis układu:

$$q(q_1, q_2, .., q_9). \tag{4.2}$$

Dwuwymiarowa bryła sztywna (rys. 4.1) posiada 3 stopnie swobody, gdyż jej orientację $\vec{r}$ w przestrzeni w danej chwili możemy opisać przy pomocy wektora położenia środka masy $\vec{r}_O$ i kąta obrotu θ względem wybranej osi wsp.:

$$\begin{aligned} \vec{r} &= [\vec{r}_O, \theta], \\ \vec{r}_O &= [r_{O_x}, r_{O_y}], \end{aligned} \tag{4.3}$$



Rysunek 4.1: Bryła sztywna 2D o środku masy w punkcie O.

Prędkość liniową ciała oraz prędkość kątową ciała możemy wprowadzić jako pierwsze pochodne wektora położenia $\vec{r}$ oraz wychylenia kątowego θ po czasie:

$$\begin{aligned} \vec{v} &= \frac{d\vec{r}}{dt}, \\ \vec{\omega} &= \frac{d\theta}{dt}, \end{aligned} \tag{4.4}$$

czyli dla przedstawionej bryły sztywnej:

$$\begin{aligned} \vec{V} &= \frac{d\vec{r}}{dt} = [\frac{d\vec{r_O}}{dt}, \frac{d\theta}{dt}] = [\vec{V_O}, \omega], \\ \vec{\omega} &= [0, 0, \omega], \end{aligned} \tag{4.5}$$

gdzie $\vec{V_O}$ to prędkość liniowa środka masy, ω to skalarna prędkość kątową będąca co do wartości równą długości wektora prędkości kątowej $\vec{\omega}$ skierowanego prostopadle do płaszczyzny XY.

Natomiast prędkość dowolnego punktu P na poruszającej się ze stałą prędkością bryle sztywnej, który łączy ze środkiem masy bryły wektor $\vec{r}_P$ (Rys. 4.1), zapiszemy jako:

$$\vec{V}_P = \vec{V}_O + \vec{\omega} \times \vec{r}_P, \tag{4.6}$$

pamiętając cały czas, że iloczyn wektorowy wyraża się poprzez:

$$\vec{\mathbf{a}} \times \vec{\mathbf{b}} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} = \begin{vmatrix} a_2 & a_3 \\ b_2 & b_3 \end{vmatrix} \mathbf{i} - \begin{vmatrix} a_1 & a_3 \\ b_1 & b_3 \end{vmatrix} \mathbf{j} + \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} \mathbf{k}, \tag{4.7}$$

i dzięki temu możemy zapisać iloczyn wektora prędkości kątowej ω i wektora r_P jako:

$$\vec{\omega} \times \vec{r}_P = [0, 0, \omega] \times [r_x, r_y, 0] = -\omega \cdot r_y \cdot \mathbf{i} + \omega \cdot r_x \cdot \mathbf{i} + 0 \cdot \mathbf{k}, \tag{4.8}$$

co oznacza, że wektor $[-\omega \cdot r_y, \omega \cdot r_x]$ leży w płaszczyźnie XY trójwymiarowego układu kartezjańskiego (w przestrzeni 2D).
Łącząc ze sobą w różny sposób swobodnie poruszające się ciała możemy stworzyć bardziej złożony układ mechaniczny, ograniczając w wybrany sposób ruch ciał. Takie ograniczenie nałożone na układ ciał, wymuszające ich specyficzny rodzaj ruchu nazwiemy **więzem**. Wprowadzenie więzu do układu ciał zmniejsza liczbę stopni swobody. Ogólnie, więz nałożony na układ ciał możemy zapisać jako pewną funkcję, której argumentami są współrzędne i prędkości uogólnione należących do układu ciał:

$$f(\vec{x}) = f(x_1, x_2, .., x_n, \dot{x_1}, \dot{x_2}, .., \dot{x_n}), \tag{4.9}$$

a mówiąc o **spełnieniu więzu**, rozumieć będziemy spełnienie warunku:

$$f(\vec{x}) = 0. \tag{4.10}$$

Więzy możemy podzielić na **holonomiczne** i **nieholonomiczne** [4]. Więzy holonomiczne to takie, dla których opisująca je funkcją nie zależy od prędkości a jedynie od położenia, a jeśli nawet zależy od prędkości to da się ją zcałkować do postaci nie zawierającej prędkości. Więz holonomiczny znacznie ogranicza swobodę układu ciał, umożliwiając ich ruch jedynie wzdłuż poziomic zadanych przez więz. Przykładem więzu holonomicznego może być ograniczenie nałożone na ruch koralika poruszającego się bez tarcia wzdłuż drutu, lub ograniczenie ruchu ciała zawieszonego na końcu wahadła. Więzem nieholonomicznym nazwiemy więz którego nie da się wycałkować do postaci nie zawierającej prędkości. Przykładem więzu nieholonomicznego (więzu prędkości) może być tarcie. Więzy mogą również zależeć od czasu, jednak w tej pracy nie będziemy się tego typu więzami zajmować.
Różniczkując $f(\vec{x})$ po czasie, otrzymamy:

$$\frac{df(\vec{x})}{dt} = \frac{df}{d\vec{x}} \frac{d\vec{x}}{dt} = \frac{df}{d\vec{x}} \vec{V}. \tag{4.11}$$

$\vec{V}$ to wektor prędkości uogólnionych, natomiast pochodna $\frac{df}{d\vec{x}}$ jest pochodną funkcji wektorowej względem wektora, dającą w rezultacie macierz J będącą gradientem funkcji f, zwaną **macierzą Jakobiego**. Macierz tę będziemy w dalszej części pracy nazywać **Jakobianem**[1]. Równanie warunku spełnienia ograniczenia nałożonego przez więz można więc zapisać ostatecznie:

$$\dot{f} = J\vec{V} = 0. \tag{4.12}$$

[1] Jakobianem w matematyce nazywamy wyznacznik z macierzy Jakobiego przy założeniu, ze ta macierz jest kwadratowa. W j. angielskim Jakobian może oznaczać zarówno macierz Jakobiego jak i jej wyznacznik.

Równanie to zmodyfikujemy dodając człon b (z j. angielskiego: *bias*) odpowiedzialny za błąd numeryczny / dryft numeryczny więzu:

$$\dot{f} = J\vec{V} + b = 0. \tag{4.13}$$

Człon ten będzie nam potrzebny w dalszej części pracy przy wprowadzaniu numerycznego modelu silnika obrotowego oraz przy rozwiązaniu kolizji bryły sztywnej. Dalej, zgodnie z **drugą zasadą dynamiki Newtona**[1] zapiszemy:

$$F = \frac{dP}{dt} = m\frac{d^2x}{dt}, \tag{4.14}$$

$$T = \frac{dL}{dt} = I\frac{d^2\theta}{dt}, \tag{4.15}$$

gdzie F to siła, T moment siły, P to pęd ciała, L moment pędu (kręt), m masa ciała, a I moment bezwładności ciała. Korzystając z (4.10) i (4.12) nałożymy na układ ciał pewien więz C ($f \to C$), siły reakcji F_c pochodzące od więzu możemy zapisać następująco:

$$F_c = J^T\lambda, \tag{4.16}$$

gdzie współczynnik λ jest tzw. mnożnikiem Lagrange'a, zawierającym długość wektora siły F_c wymuszającej ruch po torze opisanym więzem, a J^T wektorem wyznaczającym jej kierunek, czyli wektorem normalnym do toru ruchu ciała.
W tym miejscu spróbujemy zapisać równania (4.14) i (4.15) w postaci pojedynczego równania opisującego zarówno siły jak i momenty sił działających na i-te ciało w pewnym układzie N ciał. W tym celu wprowadzimy tzw. **macierz masy**[6]. Dla pojedynczej dwuwymiarowej bryły sztywnej, o masie m i momencie bezwładności I macierz masy zapiszemy jako:

$$M_i = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I \end{bmatrix}, \tag{4.17}$$

a równanie dynamiki opisujące siły i momenty sił działające na i-te ciało układu zapiszemy:

$$M_i\ddot{q_i} = F_i, \tag{4.18}$$

gdzie q_i to wektor współrzędnych uogólnionych dwuwymiarowej bryły sztywnej:

$$\vec{q_i} = [x_1^i, x_2^i, x_3^i], \tag{4.19}$$

a F_i to wektor sił uogólnionych działających na i-te ciało, zawierający siły i momenty sił. W celu opisu dynamiki całego układu, zapiszemy uogólnioną postać macierzy masy

układu N ciał o masach m_i i momentach bezwładności I_i, gdzie $i = 1..N$:

$$M = \begin{bmatrix} m_1 & 0 & 0 & .. & 0 & 0 & 0 \\ 0 & m_1 & 0 & .. & 0 & 0 & 0 \\ 0 & 0 & I_1 & .. & 0 & 0 & 0 \\ . & & & & & & \\ . & & & & & & \\ 0 & 0 & 0 & .. & m_N & 0 & 0 \\ 0 & 0 & 0 & .. & 0 & m_N & 0 \\ 0 & 0 & 0 & .. & 0 & 0 & I_N \end{bmatrix}. \tag{4.20}$$

W tym miejscu możemy zapisać równanie dynamiki układu ciał, uwzględniając rozróżnienie między siłami F_c pochodzącymi od więzów (równanie 4.16) oraz siłami zewnętrznymi F_e działającymi na układ:

$$M\ddot{q} = F_c + F_e. \tag{4.21}$$

Naszym celem będzie rozwiązanie układu równań opisujących dynamikę systemu ciał (4.21) oraz warunki spełnienia nałożonych na system więzów (4.12). Jednym z możliwych podejść do tego problemu jest próba rozwiązania analitycznego z zastosowaniem metod rozwiązywania równań algebraiczno-różniczkowych[26] (DAE). My zastosujemy podejście równorzędne jak chodzi o rezultaty jednak prostsze i bazujące na fizycznej intuicji. Polega ono na rozbiciu problemu na dwa osobne kroki: aplikację sił zewnętrznych działających na układ, ignorując na chwilę istnienie więzów, a potem zastosowanie metody tzw. **impulsów korekcyjnych**[27] w celu korekcji naruszonych warunków więzów. Metoda ta została spopularyzowana i rozwinięta dzięki stworzonemu przez Erina Catto silnikowi symulacji fizycznej Box2D open-source, którego model numeryczny[15] przyjęto jako wzorcowy przy projektowaniu silnika symulacji fizycznej w poniższej pracy. Stosując przybliżenie numeryczne pochodnej $\ddot{q}$:

$$\frac{d^2q}{dt^2} = \frac{\Delta V}{\Delta t}, \tag{4.22}$$

zapiszemy równanie 4.21:

$$M\frac{\Delta V}{\Delta t} = F_c + F_e, \tag{4.23}$$

$$\Delta V = (F_c + F_e)M^{-1}\Delta t, \tag{4.24}$$

gdzie ΔV to różnica między prędkością początkową (aktualną) V_0 a prędkością końcową (docelową) V_1 (otrzymaną po scałkowaniu numerycznym działających na ciał sił i momentów sił), co zapiszemy:

$$\vec{V_1} = \vec{V_0} + F_cM^{-1}\Delta t + F_eM^{-1}\Delta t. \tag{4.25}$$

Równanie (4.25) rozwiążemy numerycznie w sposób 2 krokowy: najpierw aplikując jedynie siły zewnętrzne, przyjmując brak jakichkolwiek więzów nałożonych na układ, a

potem aplikując siły pochodzące od więzów przy założeniu braku sił zewnętrznych. Ponieważ siły zewnętrzne działające na układ są nam znane w każdym kroku numerycznym, potrzebujemy w tym momencie jedynie skupić się nad znalezieniem sił reakcji więzu. Przyjmując siły zewnętrzne równe zero, zapiszemy:

$$\vec{V_1} = \vec{V_0} + M^{-1}F_c\Delta t. \tag{4.26}$$

W tym momencie wstawiając do warunku spełnienia więzu (4.13) równanie (4.26), otrzymamy:

$$\dot{f} = J\vec{V_1} + b = J(\vec{V_0} + M^{-1}F_c\Delta t) + b = 0, \tag{4.27}$$

$$J\vec{V_0} + JM^{-1}F_c\Delta t + b = 0. \tag{4.28}$$

Korzystając z (4.16) zapiszemy:

$$J\vec{V_0} + JM^{-1}J^T\lambda\Delta t + b = 0. \tag{4.29}$$

Tutaj pamiętając że λ jest wielkością siły reakcji więzów, oraz że siła o wielkości F działająca w czasie Δt powoduje zmianę pędu o wielkości równej ΔP, czyli że:

$$F\Delta t = \Delta P, \tag{4.30}$$

$$F = \lambda, \tag{4.31}$$

z (4.29) otrzymamy:

$$J\vec{V_0} + JM^{-1}J^T\Delta P + b = 0, \tag{4.32}$$

$$\Delta P = -\frac{J\vec{V_0} + b}{JM^{-1}J^T}, \tag{4.33}$$

gdzie ΔP to szukana przez nas wielkość zmiany pędu zwana dalej **impulsem korekcyjnym**, której aplikacja umożliwi spełnienie naruszonego warunku więzu. W tym miejscu możemy wprowadzić wielkość K:

$$K = JM^{-1}J^T, \tag{4.34}$$

która opisuje układ mas i momentów bezwładności ciał na które nałożono więz, w przestrzeni więzu. Odwrotność tej wielkośći:

$$M_{eff} = K^{-1}, \tag{4.35}$$

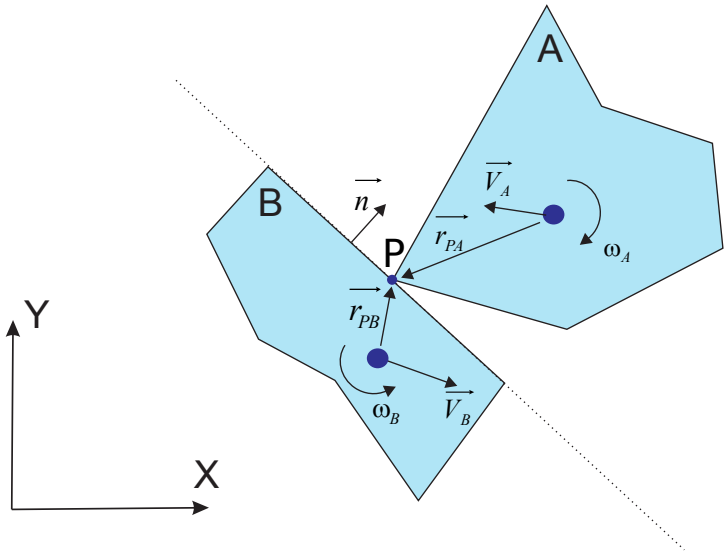
opisującą efektywną miarę bezwładności w oddziaływaniu ciał nazwiemy dalej **masą efektywną**.
Równanie (4.33) opisujące wartość impulsu korekcyjnego reakcji więzu możemy zapisać ostatecznie jako:

$$\Delta P = -M_{eff}(J\vec{V_0} + b). \tag{4.36}$$

4.2 Dynamika bryły sztywnej 2D

4.2.1 Zderzenie brył sztywnych

Na początku zajmiemy się podstawowym problemem jaki staje przed nami gdy opisujemy dynamikę bryły sztywnej[3], a mianowicie zderzeniami brył. To co interesuje nas najbardziej to znalezienie poprawnego matematycznie zapisu procesu wymiany pędu między ciałami, wskutek którego po zderzeniu ciała zmieniają swoje prędkości liniowe i kątowe i co za tym idzie kierunek ruchu.



Rysunek 4.2: Kolizja brył sztywnych A i B bez tarcia. W rezultacie reakcja w postaci wymiany pędu w kierunku normalnej $\vec{n}$ do ściany incydentu.

Aktualną (w momencie kolizji) prędkość w punkcie incydentu P (zob. rysunek 4.2) czyli punkcie w którym nastąpił kontakt brył, zapiszemy jako:

$$\vec{V}_P = \vec{V}_{PA} - \vec{V}_{PB}, \tag{4.37}$$

$$\vec{V}_{PA} = \vec{V}_A + \vec{\omega}_A \times \vec{r}_{PA}, \tag{4.38}$$

$$\vec{V}_{PB} = \vec{V}_B + \vec{\omega}_B \times \vec{r}_{PB}. \tag{4.39}$$

W celu znalezienia nowych prędkości dla ciał po kolizji nałożymy na układ dwóch ciał restrykcję w postaci więzu, którym ograniczymy ruch kolidujących ciał w kierunku normalnej do ściany incydentu.
Ogólnie postać różniczkową więzu, nakładającego ograniczenie na ruch dwóch ciał A i B, zgodnie z równaniem (4.13) zapiszemy jako:

$$\dot{C} = J\vec{V}_{AB} + b, \tag{4.40}$$

gdzie wektor prędkości $\vec{V}_{AB}$ układu ciał zapiszemy jako:

$$\vec{V}_{AB} = [\vec{V}_A, \vec{\omega}_A, \vec{V}_B, \vec{\omega}_B]. \tag{4.41}$$

Natomiast w tym konkretnym przypadku, chcemy nałożyć na układ ciał ograniczenie, uniemożliwiające w momencie kolizji dalszą penetrację ciał. Zrobimy to wprowadzając następujący warunek więzu:

$$\dot{C} = \vec{V}_P \cdot \vec{n} = 0. \tag{4.42}$$

Tak zdefiniowany warunek więzu stanowi iż składowa prędkości kolidujących ciał w kierunku normalnej do ściany kolizji ma zostać wytracona w trakcie kolizji, co byłoby właściwe gdybyśmy rozpatrywali wyłącznie przypadki kolizji idealnie niesprężystych (plastycznych). My jednak chcielibyśmy rozwiązania bardziej uniwersalnego, w którym możemy kontrolować poziom elastyczności kolizji, przy pomocy znormalizowanego współczynnika sprężystości. W tym celu zapiszemy zmodyfikowany warunek więzu:

$$\dot{C} = \vec{V}_P \cdot \vec{n} + V_r = 0, \tag{4.43}$$

$$V_r = e\vec{V}_P \cdot \vec{n}, \tag{4.44}$$

gdzie V_r to wielkość prędkości odbitej w kierunku normalnej do ściany kolizji, a e to współczynnik sprężystości przyjmujący wartości od 0 (kolizje idealnie plastyczne) do 1 (kolizje idealnie sprężyste). Warunek rozpiszemy dalej:

$$\dot{C} = (\vec{V}_A + \vec{\omega}_A \times \vec{r}_{PA} - \vec{V}_B - \vec{\omega}_B \times \vec{r}_{PB}) \cdot \vec{n} + V_r = 0. \tag{4.45}$$

W tym miejscu, korzystając z tożsamości:

$$(\vec{A} \times \vec{B}) \cdot \vec{C} = (\vec{C} \times \vec{A}) \cdot \vec{B} = (\vec{B} \times \vec{C}) \cdot \vec{A}, \tag{4.46}$$

otrzymamy:

$$\dot{C} = \vec{V_A} \cdot \vec{n} + (r\vec{_{PA}} \times \vec{n}) \cdot \omega_A - \vec{V_B} \cdot \vec{n} - (r\vec{_{PB}} \times \vec{n}) \cdot \omega_B + V_r, \tag{4.47}$$

z czego przez tożsamość, korzystając z równania (4.13) odczytamy Jakobian więzu J, oraz współczynnik b jako:

$$\vec{J} = \left[\vec{n}, \vec{r_{PA}} \times \vec{n}, -\vec{n}, -\vec{r_{PB}} \times \vec{n}\right] \tag{4.48}$$

$$b = V_r. \tag{4.49}$$

Następnie, znając Jakobian, możemy obliczyć szukaną masę efektywną:

$$K = JM^{-1}J^T =$$

$$[\vec{n}, r\vec{_{PA}} \times \vec{n}, -\vec{n}, -r\vec{_{PB}} \times \vec{n}] \begin{bmatrix} m_A & 0 & 0 & 0 & 0 & 0 \\ 0 & m_A & 0 & 0 & 0 & 0 \\ 0 & 0 & I_A & 0 & 0 & 0 \\ 0 & 0 & 0 & m_B & 0 & 0 \\ 0 & 0 & 0 & 0 & m_B & 0 \\ 0 & 0 & 0 & 0 & 0 & I_B \end{bmatrix}^{-1} \begin{bmatrix} \vec{n} \\ \vec{r_{PA}} \times \vec{n} \\ -\vec{n} \\ -\vec{r_{PB}} \times \vec{n} \end{bmatrix}$$

Po wymnożeniu otrzymujemy:

$$M_{eff} = K^{-1} = (\frac{1}{m_A} + \frac{1}{m_B} + \frac{(r\vec{AP} \times \vec{n})^2}{I_A} + \frac{(r\vec{BP} \times \vec{n})^2}{I_B})^{-1}. \tag{4.50}$$

Mając masę efektywną, korzystając z równania (4.36) oraz warunku więzu (4.43) możemy przejść do numerycznego rozwiązania kolizji zapisując wartość impulsu korekcyjnego, działającego na oba ciała w reakcji na kolizję wzdłuż normalnej do ściany kolizji:

$$\Delta P_{AB} = M_{eff}(\vec{V_P} \cdot \vec{n} + V_r). \tag{4.51}$$

ΔP_{AB} jest skalarną wartością opisującą ilość wymienionego pędu miedzy kolidującymi ciałami w kierunku normalnej do ściany incydentu (ściany kolizji).
Mając impuls korekcyjny, nowe prędkości ciał po kolizji w procedurze numerycznej obliczymy następująco:

$$\vec{V}_{A_2} = \vec{V}_{A_1} + m_A^{-1} \cdot \vec{n} \cdot \Delta P_{AB}, \tag{4.52}$$

$$\vec{V}_{B_2} = \vec{V}_{B_1} - m_B^{-1} \cdot \vec{n} \cdot \Delta P_{AB}, \tag{4.53}$$

$$\omega_{A_2} = \omega_{A_1} + I_A^{-1} \cdot \vec{r}_{AP} \times (\vec{n} \cdot \Delta P_{AB}), \tag{4.54}$$

$$\omega_{B_2} = \omega_{B_1} - I_B^{-1} \cdot \vec{r}_{BP} \times (\vec{n} \cdot \Delta P_{AB}). \tag{4.55}$$

4.2.2 Więz tarcia

Używając zapisu różniczkowego więzów, nakładające na ruch ograniczenie podobnie jak to zrobiliśmy w przypadku równania (4.42) możemy wprowadzić tarcie do naszego modelu. Więz pochodzący od tarcia zapiszemy jako:

$$\dot{C} = \vec{V_P} \cdot \vec{t} = 0, \tag{4.56}$$

gdzie $\vec{t}$ to jednostkowy wektor prostopadły do $\vec{n}$, czyli wektor równoległy do powierzchni incydentu.
Więz ten stanowi iż docelowym stanem działania tarcia jest zatrzymanie ruchu ciała w kierunku równoległym do powierzchni kontaktu. Nie chcemy jednak ograniczać całkowicie ruchu w tym kierunku, a jedynie krokowo (w każdej klatce w której istnieje kontakt miedzy ciałami) aplikować impulsy korekcyjnie odpowiadające ilościowo sile tarcia jaka powstrzymywałaby ruch wzdłuż powierzchni. Ilościowo, impuls pochodzący od siły tarcia, odpowiadający sile działającej w jednostce czasu zapiszemy jako:

$$P_f^{max} = \mu \cdot P_n, \tag{4.57}$$

czyli że maksymalny impuls P_f^{max} pochodzący od siły tarcia będzie proporcjonalny do impulsu P_n pochodzącego od więzu kontaktu w kierunku normalnym.
Potrzebujemy znaleźć transwersalną masę efektywną M_{eff}^t dla więzu tarcia, czyli masę efektywną dla oddziaływania bezwładności ciał wzdłuż linii kontaktu. Zastosujemy tutaj identyczną metodę jak w przypadku wcześniejszych więzów kontaktu, chcąc doprowadzić

definicję więzu tarcia do postaci (4.13).
Zapiszemy:

$$\dot{C} = (\vec{V}_A + \vec{\omega_A} \times \vec{r}_{PA} - \vec{V}_B - \vec{\omega_B} \times \vec{r}_{PB}) \cdot \vec{t} = 0, \tag{4.58}$$

$$\dot{C} = \vec{V}_A \cdot \vec{t} + (\vec{r}_{PA} \times \vec{t}) \cdot \omega_A - \vec{V}_B \cdot \vec{t} - (\vec{r}_{PB} \times \vec{t}) \cdot \omega_B. \tag{4.59}$$

Podobnie jak poprzednio, odczytujemy Jakobian J oraz czynnik b przez tożsamość z równaniem (4.13):

$$J = [\vec{t}, \vec{r}_{PA} \times \vec{t}, -\vec{t}, -\vec{r}_{PB}], \tag{4.60}$$

$$b = 0, \tag{4.61}$$

po czym wstawiamy do równania:

$$K = JM^{-1}J^T =$$

$$[\vec{t}, \vec{r}_{PA} \times \vec{t}, -\vec{t}, -\vec{r}_{PB} \times \vec{t}] \begin{bmatrix} m_A & 0 & 0 & 0 & 0 & 0 \\ 0 & m_A & 0 & 0 & 0 & 0 \\ 0 & 0 & I_A & 0 & 0 & 0 \\ 0 & 0 & 0 & m_B & 0 & 0 \\ 0 & 0 & 0 & 0 & m_B & 0 \\ 0 & 0 & 0 & 0 & 0 & I_B \end{bmatrix}^{-1} \begin{bmatrix} \vec{t} \\ \vec{r}_{PA} \times \vec{t} \\ -\vec{t} \\ -\vec{r}_{PB} \times \vec{t} \end{bmatrix},$$

znajdując w rezultacie transwersalna masę efektywną dla więzu tarcia:

$$M^t_{eff} = \frac{1}{m_A} + \frac{1}{m_B} + \frac{(\vec{r}_{AP} \times \vec{t})^2}{I_A} + \frac{(\vec{r}_{BP} \times \vec{t})^2}{I_B}. \tag{4.62}$$

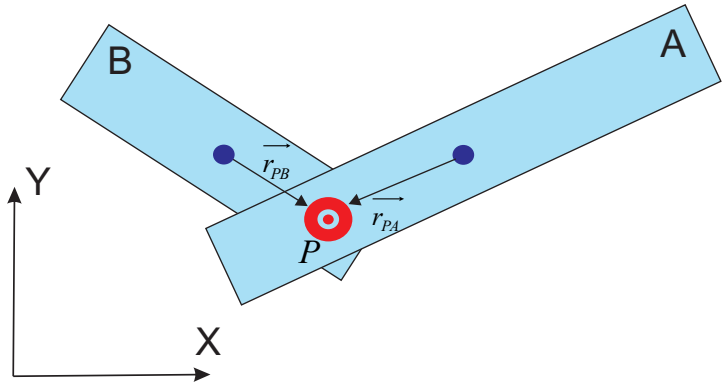
W tym momencie możemy już obliczyć wartość impulsu korekcyjnego dla więzu tarcia:

$$\Delta P^t = M^t_{eff} \vec{V}_P \cdot \vec{t}. \tag{4.63}$$

4.3 Układ brył sztywnych połączonych przegubowo

Używając tego samego podejścia co w przypadku rozwiązania kolizji, do naszego modelu wprowadzimy przeguby łączące bryły sztywne w bardziej złożone układy mechaniczne. Do wymuszenia na ciałach ograniczenia ruchu, charakterystycznego dla danego typu przegubu, użyjemy również zapisu więzów i metody impulsów korekcyjnych. To umożliwi nam przejrzysty zapis modelu teoretycznego potrzebny przy implementacji silnika symulacji, jak również da nam możliwość łatwego połączenia mechaniki przegubów z rozwiązaniem kolizji brył na poziomie numerycznym.

4.3.1 Przegub obrotowy



Rysunek 4.3: Przegub obrotowy.

Ciała połączone przegubem obrotowym mogą wykonywać swobodny ruch w przestrzeni, pod warunkiem, że dwa ustalone punkty na obu ciałach, zwane dalej **punktami zaczepu** przegubu będą w każdej chwili pokrywać się ze sobą. Pozycję obu punktów zaczepu w danym momencie w dwuwymiarowym układzie współrzędnych opiszemy wektorami wodzącymi $\vec{r}_A$ i $\vec{r}_B$, natomiast wektory łączące środki mas obu ciał z punktami zaczepu przegubu oznaczymy wektorami $\vec{r}_{PA}$ i $\vec{r}_{PB}$ (rysunek 4.3). Więz nałożony na ciała A i B w celu uniemożliwienia separacji punktów wskazanych przez wektory $\vec{r}_A$ i $\vec{r}_B$ zapiszemy jako:

$$C = \vec{r}_A - \vec{r}_B = 0. \tag{4.64}$$

Różniczkujemy więz C po czasie:

$$\dot{C} = \vec{V}_{PA} - \vec{V}_{PB}, \tag{4.65}$$

gdzie prędkości zdefiniowane są w następujący sposób:

$$\vec{V}_{PA} = \vec{V}_A + \vec{\omega}_A \times \vec{r}_{PA}, \tag{4.66}$$

$$\vec{V}_{PB} = \vec{V}_B + \vec{\omega}_B \times \vec{r}_{PB}. \tag{4.67}$$

Równanie więzu (4.65) podobnie jak w przypadku więzu (4.42) zapobiegającego dalszej penetracji ciał, doprowadzimy do postaci (4.40). Zapiszemy:

$$\dot{C} = \vec{V}_A + \vec{\omega}_A \times \vec{r}_{PA} - \vec{V}_B - \vec{\omega}_B \times \vec{r}_{PB}. \tag{4.68}$$

Dalej, zapisując otrzymane równanie w postaci macierzowej:

$$\dot{C} = \begin{bmatrix} 1 & 0 & r_{PA_x} & -1 & 0 & -r_{PB_x} \\ 0 & 1 & r_{PA_y} & 0 & -1 & -r_{PB_y} \end{bmatrix} \begin{bmatrix} \vec{V}_A \\ \omega_A \\ \vec{V}_B \\ \omega_B \end{bmatrix}, \tag{4.69}$$

możemy ostatecznie odczytać przez porównanie tożsamościowe z równaniem (4.40) szukany jakobian J więzu oraz współczynnik dryftu więzu b:

$$J = \begin{bmatrix} 1 & 0 & r_{PA_x} & -1 & 0 & -r_{PB_x} \\ 0 & 1 & r_{PA_y} & 0 & -1 & -r_{PB_y} \end{bmatrix} \tag{4.70}$$

$$b = 0. \tag{4.71}$$

Jak widać otrzymany jakobian ma rząd 2. Ponieważ na każde z ciał przypadają 3 stopnie swobody: dwuwymiarowy wektor prędkości środka masy i skalarna prędkość kątowa, całkowita liczba stopni swobody opisanego układu wyniesie 6 - 2 = 4 (liczba stopni układu bez ograniczeń, pomniejszona o rząd jakobianu).
W tym momencie znając jakobian więzu, obliczymy masę efektywną więzu, będącą miarą bezwładności ciał jaką oddziałują one na siebie w punkcie zaczepu przegubu:

$$K = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ r_{PA_x} & r_{PA_y} \\ -1 & 0 \\ 0 & -1 \\ -r_{PB_x} & -r_{PB_y} \end{bmatrix}^T \begin{bmatrix} m_A & 0 & 0 & 0 & 0 & 0 \\ 0 & m_A & 0 & 0 & 0 & 0 \\ 0 & 0 & I_A & 0 & 0 & 0 \\ 0 & 0 & 0 & m_B & 0 & 0 \\ 0 & 0 & 0 & 0 & m_B & 0 \\ 0 & 0 & 0 & 0 & 0 & I_B \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ r_{PA_x} & r_{PA_y} \\ -1 & 0 \\ 0 & -1 \\ -r_{PB_x} & -r_{PB_y} \end{bmatrix}.$$

Ostatecznie po wymnożeniu, otrzymany czynnik K będzie macierzą 2×2, którą zapiszemy jako sumę 3 macierzy, zawierających masy ciał, moment bezwładności pierwszego ciała i moment bezwładności drugiego ciała:

$$\begin{aligned} K = &\begin{bmatrix} m_a^{-1} + m_b^{-1} & 0 \\ 0 & m_a^{-1} + m_b^{-1} \end{bmatrix} + \\ &\begin{bmatrix} I_a^{-1} \cdot r_{PA_y} \cdot r_{PA_y} & -I_a^{-1} \cdot r_{PA_x} \cdot r_{PA_y} \\ -I_a^{-1} \cdot r_{PA_x} \cdot r_{PA_y} & I_a^{-1} \cdot r_{PA_x} \cdot r_{PA_x} \end{bmatrix} + \\ &\begin{bmatrix} I_b^{-1} \cdot r_{PB_y} \cdot r_{PB_y} & -I_b^{-1} \cdot r_{PB_x} \cdot r_{PB_y} \\ -I_b^{-1} \cdot r_{PB_x} \cdot r_{PB_y} & I_b^{-1} \cdot r_{PB_x} \cdot r_{PB_x} \end{bmatrix}. \end{aligned} \tag{4.72}$$

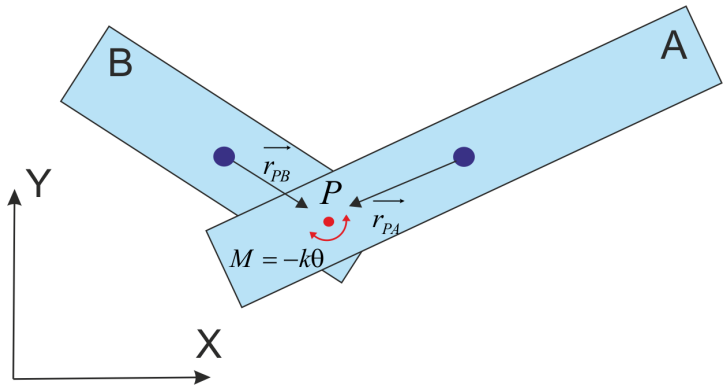
Natomiast szukaną masę efektywną znajdziemy obliczając macierz odwrotną do macierzy K:

$$M_{eff} = K^{-1}. \tag{4.73}$$

Jak więc widać, w tym przypadku szukana masa efektywna M_{eff} dla przegubu jest macierzą 2×2, a nie skalarem, jak w przypadku więzu (4.42) zapobiegającego penetracji ciał po kolizji. Ostatecznie szukany impuls korekcyjny dla przegubu obrotowego zapiszemy więc jako:

$$\Delta P = M_{eff}(\vec{V}_{PA} - \vec{V}_{PB}). \tag{4.74}$$

4.3.2 Przegub obrotowo-sprężynowy



Rysunek 4.4: Przegub obrotowo-sprężynowy.

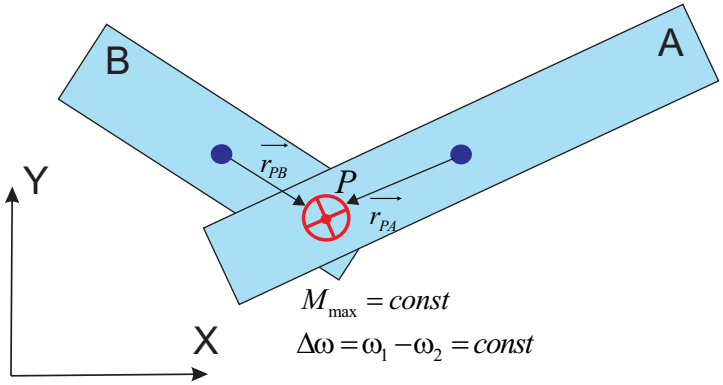
Przegub obrotowo-sprężynowy (rysunek 4.4) można sobie wyobrazić jako zwykły przegub obrotowy z zamocowaną sprężyną generującą moment siły przy próbie wychylenia ramion przegubu ze stanu równowagi. Moment siły podobnie jak przy zwykłej liniowej sprężynie będzie wprost proporcjonalny do wychylenia θ (kątowego) od poziomu równowagi oraz współczynnika k charakteryzującego sprężynę.
W procedurze numerycznej, obliczenie i aplikację impulsu podzielimy na dwie części, pierwszą zapewniającą połączenie ciał w punkcie mocowania przegubu (używając do tego impulsu korekcyjnego (4.74) dla przegubu obrotowego), i drugą generującą przyspieszenie kątowe proporcjonalne do działającego w danym momencie momentu siły. Ponieważ moment siły pochodzący od sprężyny o współczynniku sprężystości k działający na oba ramiona przegubu wynosi:

$$T_{spr} = -k\theta, \tag{4.75}$$

więc przybliżenie numeryczne zmiany krętu (momentu pędu) po czasie Δt zapiszemy jako:

$$\Delta L_{spr} = -k\theta\Delta t. \tag{4.76}$$

4.3.3 Silnik



Rysunek 4.5: Przegub silnikowy.

Podobnie jak przegub obrotowo-sprężynowy także i silnik możemy w prosty sposób zamodelować wprowadzając do przegubu obrotowego moment siły. W tym przypadku będziemy chcieli uzyskać efekt pracy silnika o stałej prędkości, dążyć więc będziemy do utrzymania stałej wartości różnicy $\Delta\omega$ między prędkościami obrotowymi ω_1 i ω_2 ramion przegubu. Pamiętać musimy że każdy prawdziwy silnik dysponuje pewną graniczną charakteryzującą go mocą, bezpośrednio związaną z maksymalnym momentem siły jakim dysponuje silnik. Powstanie momentu siły, czy to na skutek spalania paliwa w cylindrach czy działającego pola elektromagnetycznego, powoduje powstanie przyspieszenia kątowego przymocowanych ramion, a ono zmianę prędkości kątowej, aż do momentu zrównoważenia sił oporu, takich jak tarcie. Ograniczymy maksymalny moment silnika do pewnej stałej wartości M_{max}, dzięki czemu możliwy będzie do uzyskania efekt, w którym siły zewnętrzne (takie jak kolidujące z układem mechanicznym bryły, tarcie i grawitacja) uniemożliwią uzyskanie docelowej prędkości obrotowej, a nawet unieruchomią przegub z powodu niewystarczającej mocy silnika (zbyt małego momentu obrotowego).
Podobnie jak w przypadku przegubu obrotowo-sprężynowego, wykorzystamy to samo ograniczenie w postaci więzu (4.65) w celu utrzymania punktów zaczepu przegubu w jednej pozycji, natomiast wprowadzimy dodatkowo drugi więz, którym nałożymy ograniczenie na względną prędkość kątową ramion przegubu:

$$\dot{C} = \omega_A - \omega_B = \Delta\omega, \tag{4.77}$$

$$\dot{C} = \omega_A - \omega_B - \Delta\omega = 0, \tag{4.78}$$

gdzie $\Delta\omega$ to docelowa prędkość kątowa z jaką pracować ma silnik. Doprowadzają równanie (4.78) do postaci (4.40), zapiszemy je w postaci:

$$\dot{C} = \left[0, 0, 1, 0, 0, -1\right] \vec{V}_{AB} + \Delta\omega = 0. \tag{4.79}$$

Stąd możemy natychmiast odczytać przez tożsamość wartość jakobianu J oraz czynnik dryftu numerycznego b dla więzu:

$$J = \big[0, 0, 1, 0, 0, -1\big] \tag{4.80}$$

$$b = \Delta\omega. \tag{4.81}$$

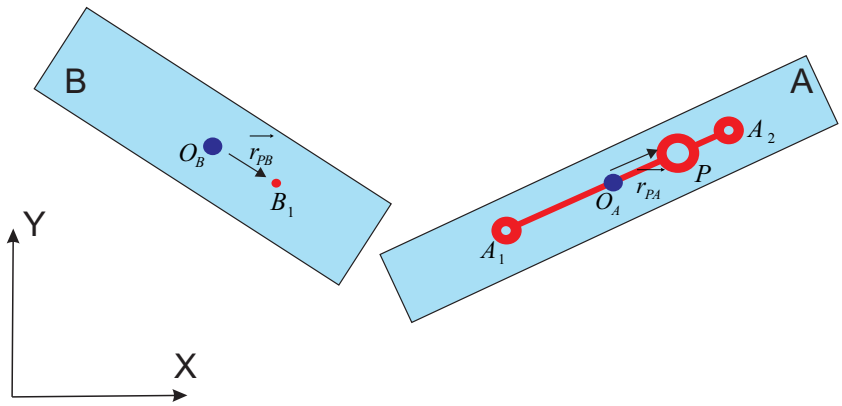
Następnie, korzystając z (4.34) oraz (4.35) obliczymy ostatecznie masę efektywną dla więzu ograniczającego ruch w kierunku obrotowym, która wynosi:

$$M_{eff}^{ang} = \frac{1}{I_A^{-1} + I_B^{-1}}. \tag{4.82}$$

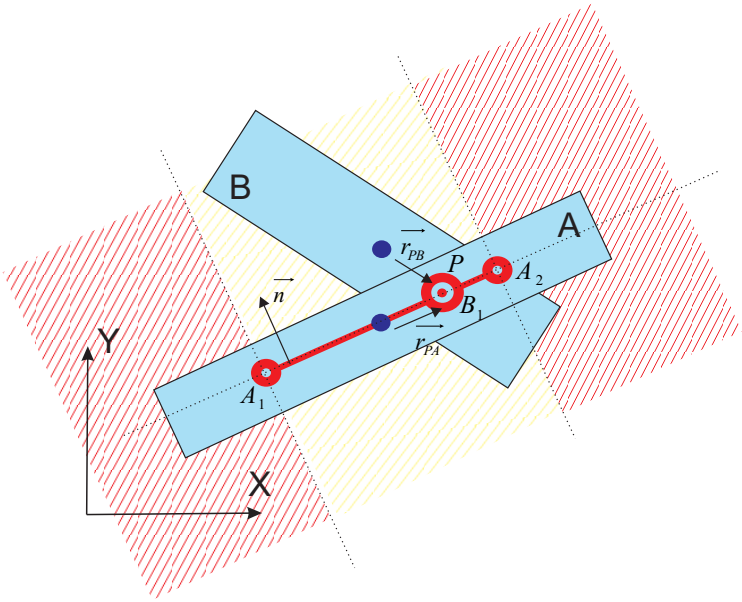
Szukaną zmiana krętu ΔL dla tego więzu zapiszemy więc ostatnie jako:

$$\Delta L = M_{eff}^{ang}(\omega_A - \omega_B + \Delta\omega). \tag{4.83}$$

4.3.4 Przegub liniowy



Rysunek 4.6: Przegub liniowy. Elementy konstrukcyjne przed zaczepieniem punktu zaczepu B_1 na osi przegubu A_1A_2.



Rysunek 4.7: Przegub liniowy. Punkt zaczepu B_1 spoczywa na osi przegubu A_1A_2.

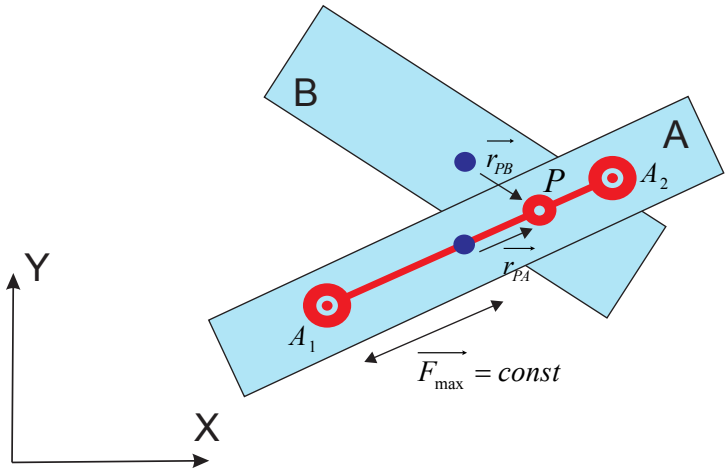
Dwuwymiarowy przegub liniowy (zob. rysunki 4.6 i 4.7) posiada dwa stopnie swobody, umożliwia połączonym bryłom swobodny obrót wokół punktu mocowania oraz ruch punktu zamocowania ciała B (punkt B_1) wzdłuż ustalonego odcinka na ciele A (A_1A_2).

W celu implementacji takiego modelu przegubu odseparowano mechanizmy numeryczne odpowiadające dwóm ograniczeniom nałożonym na ruch przegubu:
1) ograniczeniu ruchu zaczepu B_1 w kierunku normalnej do osi A_1A_2 przegubu,
2) ograniczeniu ruchu zaczepu B_1 poza punkty graniczne A_1, A_2 na osi przegubu.

Pierwsze z ograniczeń ruchu przegubu liniowego uzyskamy stosując ten sam więz (4.42), którym ograniczyliśmy ruch ciał w kierunku normalnej w trakcie zderzenia. Ograniczenie to możemy sobie wyobrazić jako idealnie nieelastyczną kolizję ciała B w punkcie B_1 z ciałem A, podczas gdy powierzchnią incydentu będzie oś przegubu A_1A_2. Taka „kolizja" będzie miała za zadanie naprowadzić punkt B_1 ciała B z powrotem na oś A_1A_2 natychmiast po utracie z nią kontaktu w kierunku normalnym. W celu korekcji numerycznej utraty kontaktu punktu zaczepu B_1 przegubu B z osią A_1A_2 zastosujemy więc impuls korekcyjny (4.51) i odpowiadającą mu masę efektywną działającą w kierunku normalnym do osi A_1A_2.
Drugie z ograniczeń - zamocowanie punktu B_1 na odcinku A_1A_2 - wprowadzimy natomiast z równania więzu tarcia (4.56). Chcemy, by po przekroczeniu przez mocowanie B_1 któregoś z punktów granicznych przegubu A_1 lub A_2 (obszar czerwony na rys. 4.7), aplikowany został impuls korekcyjny (4.63) w kierunku osi przegubu, który zatrzyma ruch mocowania przegubu B_1 poza obszar żółty.
Należy również pamiętać iż impuls korekcyjny pochodzący od reakcji w kierunku normalnym do osi przegubu aplikować będziemy zawsze, natomiast impuls w kierunku zgodnym

z osią - jedynie po przekroczeniu granic A_1 A_2 przegubu przez mocowanie B_1.

4.3.5 Siłownik



Rysunek 4.8: Przegub siłownik.

Siłownik jest wersją przegubu liniowego w której wzdłuż osi przegubu działa stała, ograniczona z góry siła, mająca na celu utrzymanie stałej prędkości przesuwu zaczepu B_1 wzdłuż osi A_1A_2, aż do osiągnięcia jednego z punktów granicznych osi przegubu. Jest to więc mechanizm analogiczny do zaprezentowanego wcześniej modelu przegubu-silnika. Utrzymanie zaczepu przegubu siłownika na osi siłownika zapewnimy stosując ten sam więz (4.42) i pochodzący od niego impuls korekcyjny (4.51) co w przypadku zwykłego przegubu liniowego. Podobnie więc wprowadzimy docelową prędkość V_t z jaką miałby się poruszać zaczep siłownika do równania więzu tarcia (4.56), zapisując więz pracy siłownika (więz działający w kierunku osi A_1A_2 siłownika):

$$\dot{C} = \vec{V}_P \cdot \vec{t} + V_t = 0. \tag{4.84}$$

W rezultacie otrzymamy ten sam jakobian (4.60) i masę efektywną M^t_{eff} (4.62) co w przypadku więzu tarcia, natomiast różnicą miedzy więzem tarcia a więzem pracy siłownika będzie wartość czynnika b (4.61) która tutaj będzie niezerowa i równa docelowej prędkości pracy siłownika V_t. Ostatecznie więc szukany impuls korekcyjny działający wzdłuż osi przegubu w przypadku siłownika zapiszemy jako:

$$\Delta P = M^t_{eff}(\Delta\vec{V}_P \cdot \vec{t} + V_t). \tag{4.85}$$

Rozdział 5

Silnik symulacji

5.1 Architektura silnika symulacji fizycznej opartego na dynamice bryły sztywnej

Wyrażenie „silnik symulacji" oznacza tak naprawdę bibliotekę programistyczną stworzoną do przeprowadzania symulacji uproszczonego modelu otaczającego nas świata z zachowaniem praw fizycznych. Silnik ten umożliwia całkowanie równań ruchu opisujących dany model układu fizycznego, przy użyciu małego (kontrolowanego przez programistę) kroku czasowego.
W każdym silniku symulacji dynamiki bryły sztywnej możemy wyróżnić 3 zasadnicze elementy:

- Mechanizm detekcji kolizji.
- Mechanizm rozwiązania kolizji.
- Mechanizm aplikacji sił zewnętrznych i całkowania równań ruchu.

Dwa ostatnie mechanizmy są ze sobą mocno powiązane, a w poniższej pracy główną uwagę skupimy na numerycznym mechanizmie rozwiązania kolizji, w którym do całkowania równań ruchu użyjemy symplektycznego schematu Eulera [29].

5.2 Iteracyjny solver impulsowy

Moduł silnika symulacji odpowiedzialny za aplikację sił zewnętrznych, rozwiązanie równań ruchu, rozwiązywanie kolizji miedzy ciałami oraz zadbanie o poprawne uwzględnienie więzów zwany jest **solverem** (z j. angielskiego: **solver**). Istnieją różne typy solverów[39][40][41] i tak naprawdę to zastosowane algorytmy detekcji kolizji, rozwiązania kolizji i spełnienia więzów nałożonych na układy połączone przegubowo są tym co rozróżnia miedzy sobą dostępne na rynku otwarte i zamknięte silniki symulacji fizycznej oparte o dynamikę bryły sztywnej. Kierując się wyborem solvera stosującego metodę impulsową, wzięto pod uwagę iż stosują go najpopularniejsze silniki symulacji fizycznej

stosowane w grach komputerowych, (takie jak Box2D[14], Chipmunk[31], Bullet[38] czy PhysX[37]), w których krytyczna jest wydajność obliczeniowa, a realizm fizyczny rozwiązania jest kwestią drugoplanową[5].
Użyty w poniższej pracy **iteracyjny solver impulsowy** (z j. angielskiego: impulse based iterative solver) stanowi rdzeń budowanego silnika symulacji. Moduł ten działa nieprzerwanie, a jego efektywna praca (najbardziej kosztowna obliczeniowo) przypada na to co dzieje z symulowanym światem od momentu wystąpienia kolizji aż do momentu jej „rozwiązania" i separacji kolidujących obiektów, oraz na obliczenia związane z utrzymaniem więzów przegubowych i więzów układów napędowych nałożonych na ciała.

W pracy do zapisu użytych algorytmów użyjemy pseudokodu, czyli uproszczonego zapisu nie wymuszającego na czytelniku stosowania żadnego konkretnego typu języka programowania. Algorytm ewoluujący świat w silniku symulacji o pojedynczy krok czasowy Δt w pętli głównej symulatora, zapiszemy jako:

Algorytm 1 Kroki iterujące symulowany świat o pojedynczy krok czasowy

1: Aplikacja sił zewnętrznych działających na układ (np. grawitacja)
2: Detekcja kolizji ciał
3: Detekcja zachowania warunków nałożonych przez więzy przegubów
4: Rozwiązanie kolizji i niezachowanych więzów przegubów
5: Aplikacja impulsów korekcyjnych dla kolidujących ciał i przegubów
6: Aplikacja ruchu ciał (całkowanie pozycji)

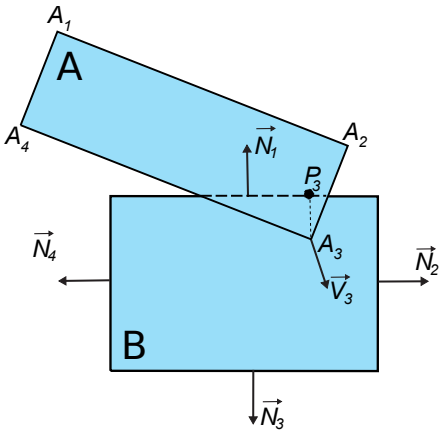
Przedstawiony algorytm jest bardzo ogólny, i ma na celu nakreślić ogólny sposób działania solvera impulsowego. Poniżej omówimy dokładniej kroki detekcji kolizji, rozwiązania kolizji, numerycznej aplikacji tarcia oraz zachowania więzów, przedstawiając na końcu szczegółowy algorytm uwzględniający wszystkie typy nałożonych na układ ograniczeń.

5.2.1 Detekcja kolizji

Detekcja kolizji brył w silniku Box2D została oparta na algorytmie SAT[30] (Separating Axis Theorem). W poniższej pracy użyto jednak zmodyfikowanej wersji tego algorytmu, którą opracowano na potrzeby pracy. Podejście takie było podyktowane faktem iż cały świat symulacji na potrzeby pracy uproszczono, ograniczając się wyłącznie do jednego typu brył sztywnych - prostokątów, dla których detekcja kolizji jest zadaniem stosunkowo prostym od strony geometrycznej.
Algorytm którego użyjemy do identyfikacji kolizji pomiędzy dwoma bryłami sztywnymi A i B o kształcie prostokąta (rysunek 5.1) zapiszemy następująco:

Algorytm 2 Detekcja kolizji ciał

1: Dla danej pary brył A i B sprawdź czy któreś z wierzchołków bryły A znajdują się wewnątrz bryły B.
2: Dla każdego z wierzchołków bryły A znajdujących się wewnątrz bryły B oblicz iloczyn: $R_{ij} = \vec{V_i} \cdot \vec{N_j}$, gdzie $\vec{V_i}$ to i-ty znormalizowany wektor wierzchołka wewnętrznego A_i bryły A (wierzchołka incydentu), a $\vec{N_j}$ to j-ta normalna (zorientowana na zewnątrz) do ściany bryły B w której znajduje się wierzchołek incydentu.
3: Ściana incydentu odpowiadająca danemu wierzchołkowi incydentu to ściana dla której R_{ij} jest najmniejsze, gdzie $j = 1, 2, 3, 4$
4: Współrzędna incydentu P_i (punkt incydentu) to rzut wierzchołka incydentu A_i na odpowiadającą jej ścianę incydentu.



Rysunek 5.1: Kolidujące bryły sztywne A i B. Na rysunku zaznaczono znormalizowany wektor $\vec{V_3}$ wierzchołka incydentu A_3 bryły A, odpowiadający mu punkt incydentu P_3 na ścianie bryły B, oraz wektory normalne $\vec{N_1}..\vec{N_4}$ ścian bryły B.

Efektem wykonania algorytmu jest zbiór zawierający listę punktów kontaktu kolidujących ciał A i B, ścianę incydentu ciała B, punkt incydentu na ścianie incydentu i wierzchołek incydentu ciała A.

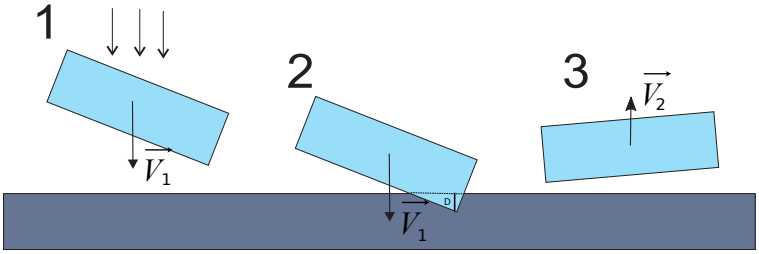
5.2.2 Rozwiązanie kolizji

W naszym przypadku „rozwiązanie" kolizji polega na znalezieniu impulsów korekcyjnych dwóch typów, których aplikacja do kolidujących ciał ma na celu rozwiązanie dwóch osobnych problemów:
- korekcji geometrycznej (korekcji pozycji) ciał aż do usunięcia penetracji,
- znalezienia prędkości separacji, czyli takich nowych prędkości kolidujących ciał, które zapobiegną dalszej penetracji ciał i umożliwią w kolejnym kroku numerycznym $t + \Delta t$ ich separację, zgodnie z ustalonym współczynnikiem sprężystości.

Oba problemy rozwiążemy opisując je dwoma typami więzów, odpowiednio: więzem uniemożliwiającym dalszą penetrację ciał (4.43) oraz więzem wymuszającym korekcję pozycji w celu usunięcia penetracji ciał (4.43), który opiszemy w następnym punkcie. Ponieważ musimy brać pod uwagę sytuacje w których naraz kolidują miedzy sobą więcej niż tylko dwa ciała, o różnych współczynnikach tarcia, różnych współczynnikach sprężystości, a nieraz połączonych ze sobą również przegubami, znalezienie impulsów korekcyjnych dla tak złożonych układów mechanicznych nie jest możliwe na drodze analitycznej. Przedstawiona tutaj metoda opisana mechaniką więzów i iteracyjną aplikacją impulsów korekcyjnych umożliwi nam jednak efektywnie rozwiązać ten problem na drodze numerycznej.

5.2.3 Więz korekcji pozycji dla kolizji

Ponieważ w momencie wykrycia kolizji ciała z reguły przenikają się wzajemnie (rysunek 5.2) np. jedno z ciał wierzchołkiem przenika ścianę drugiego, poza zmianą prędkości ciał (więz (4.43)) wynikającą z zasady zachowania pędu dla kolizji, musimy również zadbać o usunięcie geometrycznej penetracji ciał. W tym celu zastosujemy więz ograniczający pozycję kolidujących ciał.



Rysunek 5.2: Kolizja ruchomej bryły sztywnej z nieruchomym podłożem. Rysunek przedstawia pozycję bryły w kolejnych trzech krokach numerycznych: t, $t + \Delta t$ i $t + 2\Delta t$. Dla kroku 2 (w którym rozpoznano kolizję) widać penetrację bryły podłoża na głębokość D przez poruszającą się bryłę.

Zapis różniczkowy więzu jest identyczny jak w przypadku więzu zapobiegającego dalszej penetracji ciał (4.43), gdyż tak samo ograniczamy ruch ciał wzdłuż normalnej kolizji:

$$\dot{C} = \vec{V}_P \cdot \vec{n} + b = 0. \tag{5.1}$$

Tak więc również masa efektywna w dla więzu korekcji pozycji będzie identyczna jak w przypadku więzu (4.43). Natomiast wartość współczynnika dryftu b w tym przypadku zapiszemy jako:

$$b = \frac{D}{\Delta t}\vec{n}, \tag{5.2}$$

gdzie D to głębokość penetracji ściany incydentu przez wierzchołek incydentu, a Δt to krok czasowy symulacji. Tak zdefiniowany warunek więzu stanowi więc iż pod koniec

obecnego kroku numerycznym ciało powinno uzyskać prędkość, która po scałkowaniu w fazie całkowania pozycji powinna usunąć penetrację ciał.

5.2.4 Rozwiązanie numeryczne jednoczesnych kolizji całego układu.

Wprowadzenie tarcia do symulacji wieloobiektowej kolizji brył sztywnych sprowadza rozwiązanie układu do tzw. problemu LCP (Linear Complementarity Problem)[12] znanego z mechaniki teoretycznej, którego rozwiązanie na drodze analitycznej jest trudne i możliwe jedynie dla wybranych przypadków. W celu rozwiązania problemu kolidujących ze sobą jednocześnie trzech i więcej ciał, z uwzględnieniem tarcia użyjemy iteracyjnej metody Gaussa-Seidla[13], z której korzystają silniki Box2D[16][14] i Chipmunk2D[31].
Metoda ta pozwala małym nakładem obliczeniowym rozwiązać skomplikowany układ równań opisujących impulsy korekcyjne odpowiadające wszystkim istniejącym w danym kroku numerycznym kontaktom (punktom kolizji ciał) i połączeniom przegubowym ciał. Jest to metoda iteracyjna, umożliwiająca stopniowe zmniejszanie błędu numerycznego rozwiązania układu w miarę kolejnych iteracji metody.

Algorytm 3 Metoda rozwiązania kolizji bez uwzględnienia tarcia

1: Dane wejściowe: ilość N iteracji solvera, lista zawierająca n kontaktów, z których każdy zawiera wierzchołek kolizji oraz ścianę kolizji
2: **for** Powtarzaj dla kontaktów $i = 1..n$ **do**
3: Znajdź wartość masy efektywnej M_{eff} dla ciał A i B (zgodnie z równaniem (4.50))
4: Znajdź prędkość $\vec{V}_P$ w punkcie kolizji w momencie incydentu
5: **end for**
6: **for** Powtarzaj N razy iterację solvera **do**
7: **for** Powtarzaj dla kontaktów $i = 1..n$ **do**
8: Dla danego kontaktu aplikuj impuls od więzu korekcji prędkości (4.43)
9: Dla danego kontaktu aplikuj impuls od więzu korekcji pozycji (5.1)
10: **end for**
11: **end for**

Procedura aplikacji impulsu dla więzu korekcji prędkości wygląda następująco:

Algorytm 4 Algorytm aplikacji impulsu dla więzu korekcji prędkości

1: Dane wejściowe: współczynnik elastyczności kolizji e, prędkości kolidujących ciał A i B w punkcie kolizji, wartość akumulatora impulsu korekcji prędkości P^a_{vel}.
2: Oblicz docelową prędkość normalną w punkcie incydentu $V^i_n = (1+e)\vec{V}^i_P \cdot \vec{n}$
3: Oblicz cząstkowy impuls korekcyjny w aktualnym kroku iteracji $\Delta P^i_{vel} = M_{eff} V^i_n$
4: Dodaj cząstkowy impuls do akumulatora impulsu $P^a_{vel} = Max(0, \Delta P^i_{vel} + P^a_{vel})$
5: Aplikuj impuls cząstkowy do aktualnych wartości prędkości ciał A i B:
6: $\vec{V}_{A_2} = \vec{V}_{A_1} + m_A^{-1} \Delta P^i_{vel} \cdot \vec{n}$
7: $\vec{V}_{B_2} = \vec{V}_{B_1} - m_B^{-1} \Delta P^i_{vel} \cdot \vec{n}$
8: $\omega_{A_2} = \omega_{A_1} + I_A^{-1} \vec{r}_{AP} \times (\vec{n} \Delta P^i_{vel})$
9: $\omega_{B_2} = \omega_{B_1} - I_B^{-1} \vec{r}_{BP} \times (\vec{n} \Delta P^i_{vel})$

Natomiast aplikacja impulsu dla wiezu korekcji pozycji:

Algorytm 5 Algorytm aplikacji impulsu dla więzu pozycji

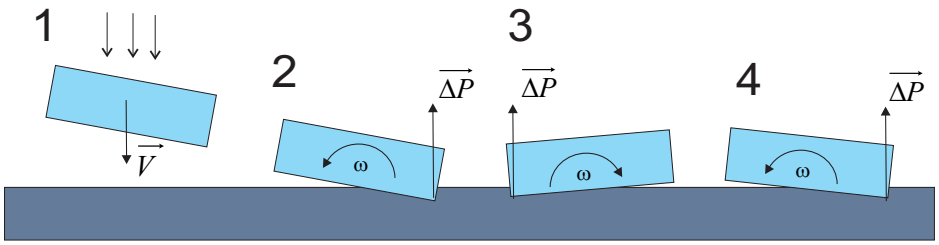
1: Dane wejściowe: głębokość penetracji D, prędkości kolidujących ciał A i B w punkcie kolizji, wartość akumulatora impulsu korekcji pozycji P^a_{pos}.
2: Oblicz docelową prędkość normalną w punkcie incydentu $V^i_n = V^i_P \cdot \vec{n} + \frac{D}{\Delta t}$
3: Oblicz cząstkowy impuls korekcyjny w aktualnym kroku iteracji $\Delta P^i_{pos} = M \cdot V^i_n$
4: Dodaj cząstkowy impuls do akumulatora impulsu $P^a_{pos} = Max(0, \Delta P^i_{pos} + P^a_{pos})$
5: Aplikuj impuls cząstkowy do tymczasowych wartości prędkości ciał A i B:
6: $\vec{V}^{temp}_{A_2} = \vec{V}^{temp}_{A_1} + m_A^{-1} \Delta P^i_{pos} \cdot \vec{n}$
7: $\vec{V}^{temp}_{B_2} = \vec{V}^{temp}_{B_1} - m_B^{-1} \Delta P^i_{pos} \cdot \vec{n}$
8: $\omega^{temp}_{A_2} = \omega^{temp}_{A_1} + I_A^{-1} \vec{r}_{AP} \times (\vec{n} \Delta P^i_{pos})$
9: $\omega^{temp}_{B_2} = \omega^{temp}_{B_1} - I_B^{-1} \vec{r}_{BP} \times (\vec{n} \Delta P^i_{pos})$

Jak można zauważyć impulsu pochodzącego od naruszenia więzów pozycji nie aplikujemy bezpośrednio do aktualnych prędkości ciał lecz zachowujemy w osobnych zmiennych tymczasowych. Robimy tak by rozseparować procesy aplikacji impulsów pochodzących od więzów korekcji prędkości i więzów korekcji pozycji w celu uniknięcia tzw. mikrowibracji, o których wspomnimy w dalszej części oraz w celu polepszenia zbieżności numerycznej do szukanego rozwiązania. Obliczone na tym etapie prędkości pochodzące od więzów pozycji zaaplikowane zostaną bezpośrednio w ostatnim kroku pętli iteracyjnej, w procedurze całkowania pozycji.

5.2.4.1 Stabilizacja Baumgarte dla więzu korekcji pozycji

Podczas stosowania metod impulsowej korekcji pozycji w celu rozseparowania kolidujących brył częstym artefaktem jaki można zauważyć w symulacjach jest problem charakterystycznych mikrowibracji. Problem pojawia się np. gdy w polu grawitacyjnym umieścimy spadające ciało i przyjmiemy warunki kolizji nieelastycznych. Po krótkim czasie ciało straci swoją energię kinetyczną uzyskaną w trakcie spadania, lecz zamiast spocząć

nieruchomo na ziemi zacznie lekko wibrować w kierunku prostopadłym do powierzchni. Dzieje się tak gdy ciało którego energia bliska jest zeru i prawie „spoczywa" na ziemi, spenetruje powierzchnię jednym tylko wierzchołkiem, podczas gdy drugi spoczywa jeszcze cały czas bardzo blisko jej powierzchni ale wciąż ponad nią (zob. rysunek 5.3).



Rysunek 5.3: Proces powstania mikrowibracji w warunkach zderzeń nieelastycznych.

Impuls korekcyjny, który spowoduje przesunięcie, obrót ciała i powrót wierzchołka ponad powierzchnię ziemi, spowoduje również zmianę prędkości obrotowej ciała i penetrację powierzchni podłoża przez drugi z wierzchołków bryły w kolejnej klatce symulacji świata. Proces sukcesywnej aplikacji impulsów korekcyjnych nie ustabilizuje ciał w pozycji leżącej lecz spowoduje charakterystyczne mikro-wibracje. Poza negatywnym efektem wizualnym, dodatkowo zauważymy pojawienie się niegasnącej oscylacji wokół niezerowej energii układu, który w warunkach kolizji nieelastycznych powinien w skończonym czasie energię wytracić.
W celu rozwiązania tego problemu zastosujemy bardzo prostą ale afektywną metodę - **stabilizację Baumgarte**[25]. Polega ona na „zezwoleniu" mechanizmowi rozwiązania kolizji na pozostawienie ciał w stanie penetracji przez więcej niż jeden krok numeryczny symulacji i stopniowym korygowaniu naruszonego więzu korekcji pozycji. W tym celu wprowadza się współczynnik β_k określający względną prędkość powrotu do stanu bez kolizji - współczynnik stabilizacji Baumgarte dla kolizji.
Zmodyfikujemy część opisującą dryft b więzu korekcji pozycji (5.1) zapisując go w nowej postaci:

$$b = \beta_k \cdot \frac{D}{\Delta t}\vec{n}, \tag{5.3}$$

gdzie:

$$\beta_k \in (0, 1). \tag{5.4}$$

Taki zabieg spowoduje że jedynie pewna część penetracji o głębokości D, na którą koliduje ciało A z ciałem B zostanie na końcu kroku numerycznego Δt usunięta. Pozostała penetracja, na którą zezwalamy, zostanie rozłożona na kolejne kroki numeryczne symulacji, stabilizując położenie kolidujących ciał i usuwając mikro-wibracje. Jak można zauważyć działanie stabilizacji przypomina działanie sprężyny przymocowanej do wierzchołka i ściany incydentu, ze względu na liniową zależność siły stabilizacji w danym kroku symulacji od odległości od powierzchni incydentu.

5.2.5 Rozwiązanie numeryczne tarcia

Działanie zaprezentowanego modelu więzu tarcia można porównać do silnika hamującego ruch w kierunku ściany incydentu. Choć więz tarcia stanowi iż prędkość w tym kierunku powinna być zerowa, co prowadzi do uzyskania impulsu korekcyjnego ΔP_{AB} którego bezpośrednie zaaplikowanie unieruchomiłoby natychmiast obiekt poruszający się po ścianie incydentu - to zamiast całego impulsu korekcyjnego zaaplikujemy tylko jego część - odpowiadającą maksymalnemu impulsowi pochodzącemu od tarcia, który obliczyliśmy powyżej. Tak więc ostateczny impuls pochodzący od tarcia zapiszemy jako:

$$P^t = Clamp(\Delta P^t_{AB}, -P^{max}_f, P^{max}_f), \tag{5.5}$$

gdzie $Clamp$ to funkcja przycięcia, ograniczająca wartość ΔP^t_{AB} z góry wartością P^{max}_f i z dołu wartością $-P^{max}_f$. Możemy teraz zapisać zmodyfikowaną procedurę rozwiązania kolizji, uwzględniającą tarcie:

Algorytm 6 Algorytm rozwiązania kolizji z uwzględnieniem tarcia

1: **for** Powtarzaj dla każdego kontaktu $i = 1..n$ **do**
2: Znajdź wartość mas efektywnych M^i_{eff} (4.50) i $M^i_{eff_t}$ (4.62)
3: Znajdź prędkość V_P w punkcie kolizji w momencie incydentu
4: **end for**
5: **for** Powtarzaj iterację solvera N razy **do**
6: **for** Powtarzaj dla każdego kontaktu $i = 1..n$ **do**
7: Oblicz i aplikuj impuls więzu korekcji prędkości dla kolizji
8: Oblicz i aplikuj impuls więzu tarcia dla kolizji
9: Oblicz i aplikuj impuls więzu korekcji pozycji dla kolizji
10: **end for**
11: **end for**

Procedura aplikacji impulsu pochodzącego od tarcia będzie wyglądać następująco:

Algorytm 7 Algorytm aplikacji impulsu pochodzącego od tarcia

1: Oblicz aktualną prędkość transwersalną w punkcie incydentu $V^i_t = V^i_P \cdot \vec{t}$
2: Oblicz cząstkowy impuls korekcyjny w aktualnym kroku iteracji $\Delta P^i_t = M^i_{eff_t} \cdot V^i_t$
3: Dodaj cząstkowy impuls do akumulatora impulsu transwersalnego, przycięty do maksymalnej wartości impulsu tarcia $P^a_t = Clamp(P^a_t + \Delta P^i_t, -P^{max}_f, P^{max}_f)$
4: Nowa prędkość ciała A: $\vec{V}_{A_2} = \vec{V}_{A_1} + m^{-1}_A \Delta P^i_t \cdot \vec{t}$
5: Nowa prędkość ciała B: $\vec{V}_{B_2} = \vec{V}_{B_1} - m^{-1}_B \Delta P^i_t \cdot \vec{t}$
6: Nowa prędkość kątowa ciała A: $\omega_{A_2} = \omega_{A_1} + I^{-1}_A \vec{r}_{AP} \times (\vec{t} \Delta P^i_t)$
7: Nowa prędkość kątowa ciała B: $\omega_{B_2} = \omega_{B_1} - I^{-1}_B \vec{r}_{BP} \times (\vec{t} \Delta P^i_t)$

5.2.6 Rozwiązanie numeryczne przegubów

Mając już wszystkie elementy potrzebne do opisu numerycznego rozwiązania kolizji możemy w końcu zintegrować numerycznie z pętlą krokowania symulacji mechanizm przegubów. Podobnie jak w przypadku rozwiązania zderzeń, proces obliczania i aplikacji impulsów korekcyjnych dla więzów opisujących przeguby możemy również rozdzielić na dwa etapy.
Pierwszy z nich, będzie wykonywany tylko raz dla każdego z przegubów w danym kroku iteracji świata, przed wykonaniem pętli iterującej rozwiązania impulsów korekcyjnych metodą Gaussa-Seidela[16]. W tym inicjacyjnym etapie obliczymy wszystko to co w fazie iteracyjnej pozostanie stałe, by przyspieszyć obliczenia. Będą to więc: obliczenie skalarnych i macierzowych mas efektywnych dla wszystkich składowych więzów każdego z przegubów, znalezienie prędkości aktualnych w punktach zaczepu przegubów oraz obliczenia geometryczne związane z pozycją przegubu (która w fazie iteracyjnej nie zmieni się, ponieważ całkowanie pozycji umieszczone zostanie na końcu procedury numerycznej solvera)
W drugiej fazie będziemy starali się znaleźć iteracyjnie impulsy korekcyjne dla wszystkich więzów układu łącznie (pochodzących od zderzeń oraz ograniczeń przegubów). W tym miejscu szukać będziemy rozwiązania układu równań opisującego działanie złożonego układu mechanicznego: posiadającego wielokrotne kolizje z uwzględnieniem tarcia i połączonego przegubami pasywnymi i aktywnymi (silnikami i siłownikami) wprowadzającymi do układu energię kinetyczną. Rozwiązanie analityczne w większości przypadków tego typu układów nie jest możliwe, procedura iteracyjnego rozwiązania problemu metodą Gaussa-Seidela ma na celu znalezienie kompromisu pomiędzy dokładnością a szybkością obliczeń. Jako że jednym z głównych założeń budowy silnika symulacji czasu rzeczywistego jest jego wydajność obliczeniowa, będziemy dbać o to by jak najmniejszy błąd numeryczny osiągnąć w jak najmniejszej ilości kroków iteracji Gaussa-Seidela, czyli by osiągnąć jak najlepszą zbieżność numeryczną szukanego rozwiązania. W poniższej pracy przyjęto stałą liczbę iteracji algorytmu Gaussa-Seidela, równą 10, która pozwoliła osiągnąć zadowalającą do poniższych badań zbieżność.
Ostateczną postać procedury iteracji symulowanego świata o krok czasowy Δt zapiszemy następująco:

Algorytm 8 Procedura iteracji świata o krok numeryczny Δt w silniku symulacji

1: Aplikuj siły zewnętrzne
2: Zidentyfikuj kolizje miedzy ciałami
3: **for** Powtarzaj dla każdego kontaktu $i = 1..n$ **do**
4: Znajdź wartość mas efektywnych M^i_{eff} (4.50) i $M^i_{eff_t}$ (4.62)
5: Znajdź prędkość V^i_P w punkcie kolizji w momencie incydentu
6: **end for**
7: **for** Powtarzaj dla każdego przegubu $i = 1..m$ **do**
8: Znajdź wartości mas efektywnych składowych więzów przegubu

```
    Oblicz prędkości aktualne w danym kroku czasowym dla przegubu
end for
for Powtarzaj iterację solvera N razy do
    for Powtarzaj dla każdego kontaktu i = 1..n do
        Oblicz i aplikuj impuls więzu korekcji prędkości dla kolizji
        Oblicz i aplikuj impuls więzu tarcia dla kolizji
        Oblicz i aplikuj impuls więzu korekcji pozycji dla kolizji
    end for
    for Powtarzaj dla każdego przegubu i = 1..m do
        Oblicz i aplikuj impulsy korekcyjne wszystkich składowych więzów przegubu
    end for
end for
Wykonaj całkowanie pozycji brył z użyciem nowo znalezionych prędkości.
```

5.3 Analiza dyssypacji energii

W celu sprawdzenia poprawności implementacji stworzonego silnika symulacji postanowiono zbadać poziom dyssypacji energii i porównać wyniki z silnikiem Box2D. W tym celu umieszczono 100 kwadratowych brył o wymiarach 1×1 każda, w zamkniętej kwadratowej komorze symulacji o wymiarach 40×40. Każdej z brył nadano wektor prędkości początkowej równy $(0, -5)$. Współczynnik sprężystości kolizji ustalono na 1.0 (kolizje idealnie sprężyste). Poziom dyssypacji energii zdefiniowano jako:

$$Ke_{err} = \frac{|Ke - Ke_0|}{Ke_0} \cdot 100\%, \tag{5.6}$$

gdzie Ke to całkowita energia kinetyczna układu pod koniec symulacji, a Ke_0 to początkowa energia kinetyczna układu. Liczbę iteracji solvera ustalono równą 10. Tak zdefiniowany system poddano symulacji której długośc trwania ustalono na 10 milionów kolizji. W trakcie symulacji poziom dyssypacji układu rósł liniowo, a po jej zakończeniu, wyniósł 4.38%. Taki wynik uznano za zadowalający jak chodzi o docelowe przeznaczenie silnika i uznano za dowód poprawności implementacji.
Następnie dla porównania przeprowadzono symulację systemu o tej samej konfiguracji uzywając silnika Box2D w wersji 2.3.0. Ewolucję poziomu dyssypacji w trakcie symulacji umieszczono w tabelce 5.1.

Liczba kolizji	Ke_{err}
100K	16.23%
150K	25.85%
200K	34.27%
300K	57.19%
500K	102.50%
1M	759.51%
1.5	31849.52%

Tabela 5.1: Postęp dyssypacji w układzie 100 brył sztywnych symulowanym z użyciem silnika Box2D. Założono warunki kolizji idealnie sprężystych.

Po około $1.5M$ kolizji dyssypacja ustabilizowała się na poziomie ok. 32000%. Zbadano dokładniej przyczynę tak znacznego przyrostu energii w trakcie symulacji w silniku Box2D, co doprowadziło do znalezienia błędu w warunku logicznym opisującym poziom elastyczności kolizji dla kolizji ciał o bardzo niskiej prędkości. Dokładniejszą analizę błędu, wraz z sugestią poprawki usuwającą opisany problem wysłano do autora silnika symulacji i umieszczono na oficjalnym forum biblioteki fizycznej[42]. Po wprowadzeniu poprawki w kodzie silnika Box2D zasugerowanej w przedstawionej na stronie forum analizie, test powtórzono dla tej samej konfiguracji układu, otrzymano poziom dyssypacji równy 2.17%, czyli lepszy (niższy) niż otrzymany dla symulacji przeprowadzonej z użyciem silnika zbudowanego na potrzeby pracy.

Wirtualne środowisko do konstrukcji i symulacji robotów

Rozdział 6

Mobilny układ mechaniczny

Skonstruowanego silnika symulacji możemy teraz użyć do budowy świata w którym umieścimy działające bez nadzoru człowieka, autonomiczne układy mechaniczne. Naszym celem jest budowa i analiza procesu ewolucji wirtualnych robotów wchodzących w interakcję z otaczającym ich światem. Takie autonomiczne, inteligentne układy mechaniczne, zwane dalej **agentami**, będą składać się z 4 podstawowych typów elementów:

- Elementów nośnych (konstrukcyjnych).
- Czujników.
- Układów napędowych.
- Urządzeń sieci neuronowych.

W dalszej części przedstawimy przykłady i dokładniejszy opis elementów każdego typu.

6.1 Elementy nośne

W opracowanym systemie symulacji ograniczymy się do najprostszego typu elementu konstrukcyjnego do budowy układów mechanicznych - dwuwymiarowej bryły sztywnej w kształcie prostokąta, posiadającej stały rozkład masy. Maszyny tworzyć będziemy łącząc bryły między sobą w bardziej złożone układy mechaniczne z użyciem zaimplementowanych w silniku symulacji przegubów. Mówiąc o elementach konstrukcyjnych mamy na myśli bryły tworzące szkielet maszyny oraz wszystkie przeguby pasywne - czyli te które nie są kontrolowane przez logikę maszyny i nie wprowadzają do układu energii mechanicznej (przeguby obrotowe, przeguby liniowe, przeguby sztywne).

6.2 Czujniki

W celu umożliwienia maszynie odbioru bodźców zewnętrznych, pewnego rodzaju prymitywnego „czucia” otaczającego ją środowiska i reakcji na te bodźce, wyposażymy

użytkownika projektującego maszynę w zestaw czujników. Ich zadaniem będzie próbkowanie informacji o otaczającym maszynę świecie i przekazywanie ich do kodu sterującego zachowaniem maszyny w sposób **pasywny** (przez przerwania) oraz **aktywny** (na żądanie).
Zaimplementowano następujące typy czujników:

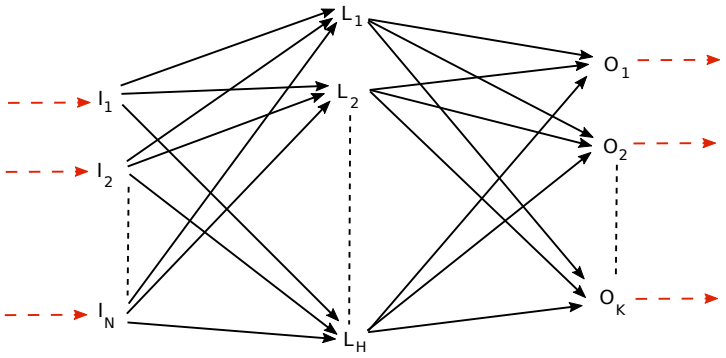
- Czujnik kontaktu: czujnik pasywny, reagujący na zderzenie z przeszkodą (bryłą sztywną) wywołaniem przerwania przekazując do kodu sterującego informacje o typie przeszkody (element konstrukcyjny innej maszyny / przeszkoda ruchoma / przeszkoda nieruchoma) i jej unikatowym identyfikatorze.
- Czujnik odległości: czujnik aktywny, umożliwia odczyt informacji o najbliższej przeszkodzie (odległość, typ, identyfikator) znajdującej się w linii prostej, w kierunku zależnym od mocowania czujnika do elementu konstrukcyjnego.
- Czujnik żyroskopowo-pozycyjny: czujnik aktywny, umożliwia odczyt aktualnej prędkości, przyspieszenia, wychylenie od kąta równowagi oraz pozycji względem środka świata symulacji.
- Radar: czujnik aktywny, daje możliwość identyfikacji pozycji i identyfikatorów innych maszyn w pobliżu, w zadanym promieniu. Zwraca listę N najbliższych maszyn posortowaną według odległości.

Wszystkie zaimplementowane czujniki mają swój odpowiednik w świecie rzeczywistym, co umożliwia wierniejsze odwzorowanie budowy i działania prawdziwych robotów.

6.3 Układy napędowe

Układy napędowe umożliwiające ruch maszyny wprowadzimy do wirtualnego świata używając zaimplementowanych przegubów aktywnych: silników i siłowników.
Oprócz zwykłych silników obrotowych i siłowników liniowych wprowadzimy również trzeci rodzaj napędu - silnik impulsowy (odrzutowy), którego użyjemy do stworzenia mechanicznych układów latających. Jego działanie opiera się na idei generowania impulsu siły przyłożonego w punkcie zaczepienia silnika do bryły konstrukcyjnej, o częstości generowania impulsu kontrolowanej przez program sterujący logiką maszyny. Takie urządzenie napędowe nie wymaga integracji z wewnętrzną częścią silnika symulacji odpowiedzialną za rozwiązanie więzów (przeguby i kolizje), tak więc zostało zrealizowane przez krokową aplikacje siły do bryły sztywnej w wybranym punkcie, jako zwykłej siły zewnętrznej.
Do wszystkich układów napędowych dodamy możliwość nadzoru poziomu zużytej energii (paliwa), tak by móc w procesie optymalizacji układów mechanicznych o którym będzie mowa w dalszej części pracy użyć tego parametru jako decydującego o poziomie przystosowania danej maszyny do realizacji postawionego przed nią zadania. Dla przegubów zużytą energię zliczać będziemy sumując ilość pędu (impulsów korekcyjnych) wygenerowanego przez przegub w każdym kroku symulacji w celu utrzymania wymaganej prędkości pracy silnika / siłownika. Dla silnika impulsowego (odrzutowego) będzie to sumaryczna ilość impulsów (popędu siły) wygenerowanych w trakcie pracy silnika.

6.4 Urządzenie sieci neuronowej



Rysunek 6.1: Schemat jednokierunkowej sieci neuronowej zawierającej N wejść, jedną warstwę ukrytą zawierającą H neuronów oraz K wyjść.

Sztuczna sieć neuronowa[32][33] jest inspirowanym biologicznie modelem matematycznym, stosowanym do przetwarzania sygnałów. Sieci te starają się imitować przepływ sygnałów elektrycznych w tkance nerwowej (np. mózgu), w której sygnał rozchodzi się poprzez system **neuronów** i **synaps**. Na rysunku 6.1 przedstawiono jeden z najprostszych typów sieci neuronowej - jednowarstwową sieć jednokierunkową - której implementację wykorzystamy do stworzenia wirtualnego urządzenia sieci neuronowej. Urządzenie to posłuży nam przy budowie autonomicznych maszyn, jako element pełniący rolę ich „układu nerwowego".
Każda jednokierunkowa sieć neuronowa zawiera N wejść pobierających znormalizowane sygnały w postaci liczb rzeczywistych, jedną lub więcej tzw. warstw ukrytych zawierających stałą lub jednakową liczbę neuronów oraz K wyjść, na których odczytywane są znormalizowane sygnały w postaci liczb rzeczywistych. Na rysunku 6.1 czarnymi strzałkami zaznaczono połączenia miedzy wejściem, neuronami warstw wewnętrznych i wyjściem, które imitują synapsy. Strzałki symbolizujące synapsy pokazują kierunek przepływu sygnału, a każda z nich symbolizuje mnożenie sygnału przez charakterystyczny dla danej synapsy współczynnik, zwany dalej **wagą**. Na końcu każdej z czarnych strzałek znajduje się sztuczny neuron (warstwy ukryta L i wyjściowa O), który ma naśladować działanie prawdziwego neuronu: sumować sygnały wchodzące z synaps, a po przekroczeniu przez sumaryczny sygnał pewnego progu zwanego progiem aktywacji neuronu, przekazać dalej sygnał wyjściowy połączonymi synapsami wyjściowymi. Sygnał wyjściowy sztucznego neuronu obliczany jest w następujący sposób:

$$N_{out} = Sigm(\sum_{i=1}^{n} w_{ij} \cdot s_i - N_{act}), \tag{6.1}$$

gdzie N_{out} to wartość sygnału na wyjściu neuronu, w_{ij} to waga odpowiadająca i-temu sygnałowi wejściowemu i j-tej synapsie, s_i to wartość i-tego sygnału, a $Sigm$ to tzw.

funkcja sigmoidalna[34] określona równaniem:

$$Sigm(x) = \frac{1}{1 - e^x}, \tag{6.2}$$

której celem jest aktywacja sygnału wyjściowego neuronu, po przekroczeniu pewnego minimalnego progu sumarycznego sygnału wejściowego.
Jak łatwo policzyć, całkowita liczba wag w tak zdefiniowanym modelu sieci neuronowej (rysunek 6.1), wliczając w to współczynniki aktywacji neuronów wyniesie:

$$W = (N + 1) \cdot H + (H + 1) \cdot K. \tag{6.3}$$

Tak zdefiniowanego modelu urządzenia sieci neuronowej przetwarzającego pewne abstrakcyjne sygnały wejściowe w postaci liczb rzeczywistych, użyjemy w implementowanym środowisku symulacji do sprzężenia układów sensorycznych maszyn z ich układami napędowymi. Widać tutaj bezpośrednią analogię do organizmów biologicznych, których interakcja z otaczającym je środowiskiem realizowana jest przez sprzężenie ich układów sensorycznych i motorycznych przy pomocy ich układu nerwowego.
Choć zaimplementowane urządzenie sieci neuronowej umożliwia konfigurację i użycie wielowarstwowej sieci jednokierunkowej, w poniższej pracy wykorzystano wyłącznie sieci jednowarstwowe jako wystarczające do realizacji postawionych przed szkolonymi maszynami zadań. Regułą której trzymano się podczas wyposażania danego typu maszyny w sieć neuronową było stosowanie sieci o warstwie ukrytej wielkości (jak chodzi o liczbę neuronów) mniej więcej dwukrotnie większej niż liczba sygnałów wejściowych.
Ponieważ to właśnie wagi i współczynniki aktywacji neuronów stanowią o tym na ile różnić będą się sygnały wyjściowe dwóch sieci tego samego typu które otrzymały na wejściu ten sam sygnał wejściowy - można się domyślić że to właśnie ich właściwy dobór będzie stanowił o tym w jaki sposób sygnały wejściowe przekazane z czujników maszyny na wejście sieci neuronowej wpłyną na sterowanie układami napędowymi maszyny sprzężonymi z wyjściem sieci neuronowej. To właśnie doborem tych wag w na drodze ewolucji zajmiemy się w dalszej części pracy, definiując zbiór wag sieci neuronowej danej maszyny jako jedyny ulegający zmianom w procesie ewolucji element maszyny.

Rozdział 7

Wewnętrzny język asembler i architektura procesora wirtualnego

Sama maszyna, posiadająca jedynie szkielet zbudowany z elementów konstrukcyjnych połączonych przegubowo i układów napędowych dających jej bezwiedny ruch, nie posiada własnej logiki zachowania. Jest jedynie pasywnym układem mechanicznym podlegającym wyłącznie nałożonym na symulowany świat prawom fizyki i mechaniki, a maszynę taką można przyrównać do nakręcanej mechanicznie zabawki, która będzie pracować bez złożonej interakcji ze środowiskiem, aż do wyczerpania energii.
By umożliwić wyposażenie maszyny w logikę decyzyjną i wprowadzić do jej zachowania element inteligencji, musimy wyposażyć ją w układ podejmowania decyzji. Układ ten połączy w jedną całość dwa pozostałe układy: układ szkieletowo-napędowy oraz układ sensoryczny. Maszyna będzie mogła aktywnie analizować i odpowiadać na bodźce zgodnie z algorytmami wykonywanymi przez jej jednostkę centralną.
W tym celu wprowadzimy do systemu symulacji nowy typ urządzenia: wirtualną jednostkę centralną (**VCPU**), dla której zaprojektujemy prosty skryptowy język programowania typu asembler. Języka tego użyjemy nie tylko do programowania inteligencji i zachowania maszyn, ale również w dalszej części pracy do opisu reguł stanowiących o doborze i przeżyciu osobników podczas opartej o algorytmy ewolucyjne optymalizacji maszyn oraz do budowy reguł sterujących inteligentnym środowiskiem w którym umieścimy maszyny.

7.1 Model i implementacja interpretera języka typu asembler

Powodem dla którego wybrano zaprojektowanie własnego języka do sterowania logiką maszyn, zamiast użycia jednego z wielu dostępnych języków zarządzanych (Python, Java Script, Java) jest potrzeba osiągnięcia atomowości instrukcji, lepsza kontrola nad synchronizacją czasu wykonania kodu dla wszystkich maszyn i przede wszystkim element

edukacyjny, jakim jest niskopoziomowość architektury języka, dająca lepiej „poczuć" użytkownikowi budującemu mobilny układ mechaniczny sposób działania maszyny. Zaletą atomowości instrukcji jest również możliwość łatwiejszej wizualizacji przebiegu wykonania programu w trakcie jego działania, co również stanowi element edukacyjny.
Język wyposażono w mechanizmy [22] najpotrzebniejsze do implementacji prostej logiki zachowania maszyn, dające jednocześnie jak największy komfort programiście tworzącemu kod, a więc:

- 3 typy adresowania: natychmiastowy, względny i bezwzględny
- instrukcje skoku warunkowego i bezwarunkowego
- instrukcje operacji arytmetycznych i geometrycznych dla 3 typów danych: całkowitego, rzeczywistego i wektorowego
- instrukcje sterowania i odczytu urządzeń (czujników, napędów i sieci neuronowej)
- instrukcje specjalne: NOP (nierób nic), DAT (dane), WAIT (odczekaj wybraną ilość cykli procesora), ZERO (instrukcja niedozwolona, powodująca wyjątek i zatrzymanie pracy procesora)
- instrukcje dostępu do wewnętrznych parametrów symulowanego świata służące do opisu reguł środowiska i doboru osobników w trakcie ewolucji (niedostępne z poziomu sterowania logiką układu mobilnego)

Wirtualny procesor odpowiedzialny za wykonanie kodu języka, zawiera 9 rejestrów, po 3 na każdy z typów danych: rejestry całkowite: REG_IA, REG_IB, REG_IC, rejestry rzeczywiste: REG_FA, REG_FB, REG_FC i rejestry wektorowe: REG_VA, REG_VB, REG_VC. Każda instrukcja wykonywana jest dokładnie w jednym cyklu zegara, taktowanego z częstotliwością jednakową dla wszystkich maszyn w symulowanym świecie, konfigurowaną przez użytkownika definiującego zasady działania tworzonego świata symulacji. Zapewnienie synchronizacji między taktami zegarów poszczególnych procesorów kolejnych maszyn umieszczonych w świecie symulacji nie wymaga żadnych dodatkowych mechanizmów ze względu na jednowątkowość obliczeń w obrębie danego świata.
Dokładny spis instrukcji języka i odpowiadających im mnemoników umieszczono na listingu 7.2.

7.2 Wirtualna pamięć współdzielona i prywatna

Wirtualna jednostka centralna VCPU każdej z maszyn wykonuje kod w wydzielonym buforze umieszczonym w uwspólnionej pamięci wirtualnej, w której znajdują się zarówno dane każdego programu sterującego jak i kod. Daje to możliwość dostępu do własnego kodu i danych nawzajem, pomiędzy maszynami, np. w celu zasymulowania kanału komunikacyjnego pomiędzy współpracującymi maszynami, potrzebnego do wspólnego działania i kooperacji. Z drugiej strony wprowadza to również bardzo ciekawą możliwość konkurencji pomiędzy maszynami, na odmiennej niż mechaniczno-fizyczna płaszczyźnie,

pozwalając na wzajemną wrogą ingerencję w kod (i dane). Daje to możliwość przeprowadzania np. symulacji współczesnego pola walki w którym autonomiczne maszyny konkurując na dwu osobnych płaszczyznach: fizycznej(mechanicznej) i cyfrowej.
Każda komórka pamięci zawiera dokładnie jedną instrukcję kodu języka lub dokładnie jedną wartość jednego z 3 typów danych. Odczyt i zapis do pamięci wirtualnej może być zrealizowany zarówno przez instrukcję o mnemoniku MOV (kopiuj), jak i każdą z instrukcji arytmetycznych, z zastosowaniem adresowania przez referencję do pamięci.

7.3 Sterowanie pasywne i aktywne

Komunikacja i sterowanie czujnikami zostało zrealizowane na 2 sposoby: aktywny i pasywny. W poniższej sekcji omówiono przykłady i zasadę działania czujników obu typów. W trakcie analizy porównania zamieszczonego poniżej warto zauważyć analogię do podobnego rozdziału: na autonomiczny i somatyczny układ nerwowy, w organizmach zwierząt[24].

7.3.1 Sterowanie aktywne: komendy sterujące dla urządzeń

Sterowanie aktywne to takie w którym maszyna w sposób bezpośredni żąda odczytu z czujnika, realizując to żądanie poprzez wykonanie instrukcji o mnemoniku DEVCTL (zobacz listing 7.1 i 7.2). Instrukcja ta wysyła do urządzenia o identyfikatorze przekazanym w argumencie kod sterujący, przekazany w drugim argumencie i natychmiast (w kolejnym takcie zegara wirtualnego procesora) otrzymuje zestaw danych zwrotnych z sensora, pod adresem wskazanym w trzecim argumencie. Ten typ sterowania jest odpowiedni dla czujników odległości, żyroskopowo-pozycyjnych oraz radarowych. Używając tej metody możemy również sterować pracą urządzeń napędowych takich jak silniki i siłowniki, zmieniając ich prędkość, zatrzymując je i uruchamiając oraz zmieniając kierunek ich pracy.

7.3.2 Sterowanie pasywne: przerwania od urządzeń

Sterowanie pasywne zrealizowane natomiast zostało poprzez przerwania i jest charakterystyczne dla urządzeń które sygnalizację zmiany stanu muszą zrealizować natychmiast, nie czekając na odczyt stanu czujnika z programu sterującego. Przykładem urządzenia sterowanego pasywnie jest czujnik kontaktu. Podłączając takie urządzenie do maszyny wskazujemy kod przerwania odpowiadający za obsługę sygnału nadchodzącego z urządzenia. W przypadku czujnika kontaktu, będzie to sygnał wywołany kolizją z dowolnym obiektem. Po zajściu zdarzenia generującego sygnał, w następnym takcie zegara procesora danej maszyny sterowanie (zmiana rejestru IP wskaźnika instrukcji procesora) zostanie przekazane do kodu obsługi przerwania i zwrócone po wykonaniu instrukcji powrotu z przerwania (mnemonik IRET).
Na poniższym listingu (7.1) przedstawiono kod sterujący prostej maszyny kroczącej poruszającej się przy pomocy dwu obrotowych kończyn połączonych z tułowiem dwoma silnikami obrotowymi. Na końcu każdej z kończyn znajdują się czujniki kontaktu, po

jednym na każdą kończynę. Sygnał kolizji z przeszkodą każdego z czujników generuje przerwanie, obsługiwane przez procedury: *'sen1proc'* oraz *'sen2proc'*, odpowiednio dla czujnika pierwszej kończyny i drugiej kończyny.

Listing 7.1: Kod sterujący robota kroczącego typu *'walker'*. Punkt startu procesu głównego oznaczono etykietą *'start'*. Punkty startu procedur obsługi przerwań dla czujników kontaktu oznaczono etykietami *'sen1proc'* oraz *'sen2proc'*.

```
// blok danych:
S1IO1:      DAT
SENDAT1:    DAT
SENDAT2:    DAT
SENDAT3:    DAT
SENDAT4:    DAT
WALLCNT:    DAT

// początek wykonania programu:
START:      MOV     "S1IO1", 1
P1:         NOP
            NOP
            NOP
            NOP
            NOP

// poczekaj 50sek i przełącz kierunek pracy silników:
            WAIT    50000
            DEVCTL  "MT0", DEV_CTL_MOTOR_POLAR_TOGGLE, "S1IO1"
            DEVCTL  "MT1", DEV_CTL_MOTOR_POLAR_TOGGLE, "S1IO1"

// zwiększ licznik iteracji pętli głównej i wykonaj zapętlenie:
            INC     "S1IO1"
            JMP     "P1"

// punkt startu procedury obsługi przerwania czujnika kontaktu #1:
SEN1PROC:   DEVCTL  "SEN1", DEV_CTL_CONTACT_SENSOR_GET_DATA, "SENDAT1"
            JMP     "SEN"

// punkt startu procedury obsługi przerwania czujnika kontaktu #2:
SEN2PROC:   DEVCTL  "SEN2", DEV_CTL_CONTACT_SENSOR_GET_DATA, "SENDAT1"
            JMP     "SEN"

// Sprawdź czy kontakt czujnika nastąpił z jedną z przeciwległych ścian.
// Jeśli kontakt nastąpił z obiektem ruchomym (jeśli typ obiektu
// przekazany do przerwania jest równy 4) opuść przerwanie nie robiąc nic:
SEN:        NOP
            JEQ     "SENDAT2", 4, "P2"

// jeśli kontakt nastąpił z przeszkodą nieruchomą której pozycja $x$ jest
// równa zero (podłoże), opuść przerwanie nie robiąc nic:
            VGETX   REG_FA, "SENDAT3"
            JNE     REG_FA, 0.0, "SWITCH"
            JMP     "P2"

// przełącz kierunek pracy silników i zwiększ licznik kolizji ze ścianą:
SWITCH:     DEVCTL  "MT0", DEV_CTL_MOTOR_POLAR_TOGGLE, "S1IO1"
            DEVCTL  "MT1", DEV_CTL_MOTOR_POLAR_TOGGLE, "S1IO1"
            INC     "WALLCNT"

// powrót z przerwania:
P2:         NOP
            IRET
```

Jedyną wykonywaną przez obie procedury obsługi przerwań operacją jest zmiana obecnego kierunku pracy obu silników na przeciwny, a tym samym zmiana kierunku ruchu robota, pod warunkiem że kontakt nastąpił z jedną z dwu przeciwległych ścian: prawą lub lewą. Celem takiego robota będzie więc wykonywanie ciągłego nawracającego ruchu od prawej do lewej ściany komory symulacji w której został umieszczony. Dodatkowo program główny (którego wykonanie rozpoczyna się od etykiety *'start'*) zawiera pętlę nieskończoną której zadaniem będzie „manualne" (niezależnie od przerwań) okresowe przełączanie kierunku pracy silników co 50 sekund.

Listing 7.2: Lista dostępnych operacji języka wraz z odpowiadającymi im mnemonikami.

```
ZERO                  - instrukcja niedozwolona, zatrzymanie procesora
DAT                   - komórka pamieci zawierajaca dane
MOV a0, a1            - kopiuj wartosc argumentu a1 pod adres a0

IADD a0, a1, a2       - dodawanie calkowite a0+a1, wynik w a2
ISUB a0, a1, a2       - odejmowanie calkowite a0-a1, wynik w a2
IMUL a0, a1, a2       - mnozenie calkowite a0*a1, wynik w a2
IDIV a0, a1, a2       - dzielenie calkowite a0/a1, wynik w a2
IMOD a0, a1, a2       - reszta z dzielenia calkowitego a0/a1, wynik w a2
IRND a0, a1           - generuj calkowita liczbe losowa z
                        przedzialu (0, a0), wynik w a1
INC a0                - inkrementuj a0
DEC a0                - dekrementuj a0

FADD a0, a1, a2       - dodawanie rzeczywiste a0+a1, wynik w a2
FSUB a0, a1, a2       - odejmowanie calkowite a0-a1, wynik w a2
FMUL a0, a1, a2       - mnozenie rzeczywiste a0*a1, wynik w a2
FDIV a0, a1, a2       - dzielenie rzeczywiste a0/a1, wynik w a2
FMOD a0, a1, a2       - reszta z dzielenia rzeczywistego a0/a1, wynik w a2
FABS a0, a1           - wartosc bezwzgledna a0, wynik w a1
FSQRT a0, a1          - pierwiastek kwadratowy z a0, wynik w a1
FRND a0, a1           - generuj rzeczywista liczbe losowa
                        z przedzialu (0, a0), wynik w a1

VADD a0, a1, a2       - dodaj wektory a0+a1, wynik w a2
VSUB a0, a1, a2       - odejmij wektory a0-a1, wynik w a2
VDOT a0, a1, a2       - pomnoz skalarnie wektory a0*a1, wynik w a2
VMUL a0, a1, a2       - pomnoz wektor a0 przez wartosc
                        rzeczywista a1, wynik w a2
VDIV a0, a1, a2       - podziel wektor a0 przez wartosc
                        rzeczywista a1, wynik w a2
VPROJ a0, a1, a2      - znajdz wektor rzutu wektora a0 na
                        wektor a1, wynik w a2
VSETX a0, a1          - ustaw wartosc skladowej x wektora a0 rowna
                        wartosci rzeczywistej a1
VSETY a0, a1          - ustaw wartosc skladowej y wektora a0 rowna
                        wartosci rzeczywistej a1
VGETX a0, a1          - pobierz wartosc skladowej x wektora a0, wynik w a1
VGETY a0, a1          - pobierz wartosc skladowej y wektora a0, wynik w a1
VANG a0, a1, a2       - znajdz kat pomiedzy wektorami a0 i a1, wynik w a2
VLEN a0, a1           - znajdz dlugosc wektora a0, wynik w a1
VNORM a0, a1          - normalizuj wektor a0, wynik w a1

JMP a0                - skocz po adres a0
JEQ a0,a1,a2          - jesli a0 rowne a1 to skocz pod adres a2
JNE a0,a1,a2          - jesli a0 rozne od a1 to skocz pod adres a2
```

```
JGT a0,a1,a2          - jesli a0 wieksze od a1 to skocz pod adres a2
JLT a0,a1,a2          - jesli a0 mniejsze od a1 to skocz pod adres a2
IRET                  - powrót z przerwania

NOP                   - nie rob nic
WAIT a0               - wstrzymaj wykonywanie kodu na
                        a0 milisekund (czasu symulacji)

DEVCTL a0,a1,a2       - przekaz do urzadzenie o identyfikatorze a0 rozkaz o
                        kodzie sterujacym a1, wynik odczytaj pod adresem a2
SYS a0,a1,a2          - wykonaj instrukcje systemowa o kodzie a0, dane
                        wejsciowe pod adresem a1, dane wyjsciowe
                        pod adresem a2
```

Część ewolucyjna

Rozdział 8

Optymalizacja autonomicznych maszyn z użyciem algorytmów ewolucyjnych

Mając zdefiniowane narzędzia do budowy i symulacji fizycznej inteligentnych układów mechanicznych, możemy ostatecznie przystąpić do stworzenia docelowego systemu umożliwiającego optymalizację maszyn i ich „uczenie" się. Wykorzystamy do tego mechanizm analogiczny do tego, który od początku jest motorem rozwoju świata biologicznego - ewolucję.

8.1 Genotyp i fenotyp maszyny. Agent.

Do symulacji procesu ewolucji w środowisku wirtualnym użyjemy mechanizmu **algorytmów ewolucyjnych**[28]. Umożliwia on optymalizację sparametryzowanego abstrakcyjnego układu (w naszym przypadku ruchomego układu mechanicznego), poprzez sukcesywne wprowadzanie losowych zmian (mutacji) w wartościach parametrów opisujących osobniki kolejnych populacji danego typu układu. Zestaw zmiennych parametrów którymi taki typ układu (abstrakcyjny i bezosobowy) został opisany to **genotyp**.
Natomiast zestaw konkretnych wartości parametrów (genów maszyny) wchodzących w skład genotypu, opisujący i wyróżniający konkretnego osobnika (przedstawiciela populacji danego typu układu) wśród innych osobników określimy jako **fenotyp**.
Tak jak w świecie rzeczywistym - organizmy biologiczne kodujące informację o swojej budowie i mechanice działania w kodzie genetycznym ulegają mutacjom, które są dla nich destruktywne lub tworzą nowy gatunek mający przewagę nad przodkami, tak i w przypadku maszyn - ich budowę, zachowanie, parametry pracy układów napędowych i czujników możemy sparametryzować, umożliwiając zapisanie w pakiecie informacji o stałej długości kod genetyczny danej maszyny - jej fenotyp.
Genotypem maszyny będzie to co niezmienne dla danego gatunku - czyli np.: liczba kończyn, sposób połączenia elementów konstrukcyjnych szkieletu, liczba i metoda mo-

cowania przegubów, rodzaje i ilość napędów, kod sterujący, typ sieci neuronowej.
Fenotypem maszyny natomiast będą cechy charakterystyczne danego osobnika, czyli wartości parametrów genotypu. Mogą to być więc np.: długość kończyn, długość elementów szkieletu, modyfikacje współrzędnych zaczepu przegubów lub zakresy pracy układów napędowych (maksymalna moc, częstości pracy silników impulsowych, maksymalna prędkość silników i siłowników, wagi danej sieci neuronowej, itd.).
W tym momencie możemy wprowadzić pojęcie **agenta**. Nazwiemy tak przedstawiciela danego typu maszyny (osobnika pewnej populacji maszyn danego typu), posiadającego zestaw jednoznacznie definiujących jego wygląd i zachowanie wartości genów (fenotyp agenta), który umieszczony w specjalnie zaprojektowanym środowisku ma w nim realizować jak najlepiej pewien określony, mierzalny cel, zdefiniowany przez twórcę danego świata symulacji. Cel ten opiszemy w sposób jednoznaczny przy pomocy tzw. **funkcji przystosowania** - funkcji która dla osobnika o danym fenotypie umieszczonego w danym środowisku zwracać będzie wartość rzeczywistą, opisując poziom najlepszego dopasowania danej jednostki do warunków środowiska i postawionego przed nią zadania, a co za tym idzie umożliwiać będzie (bądź nie) przekazanie dalszym generacjom jej informacji genetycznej.

8.2 Parametryzacja agenta dla potrzeb mutacji

W celu zdefiniowania genotypu agenta, po zbudowaniu szkieletu maszyny, jej układu napędowego i zdefiniowaniu kodu sterującego, musimy określić które z elementów jego budowy i logiki zachowania opisanej kodem sterującym mogą podlegać zmianom w procesie ewolucji. Dla każdego z elementów który parametryzujemy, musimy podać maksymalny zakres wartości (amplitudę zmian genu) jaki dany parametr może przyjmować ulegając mutacji, oraz domyślną wartość genu, która zostanie użyta jako wartość genu w fenotypie agenta zerowej generacji na początku procesu ewolucji.
Należy pamiętać o jednej istotnej kwestii, a mianowicie parametryzacja której dokonujemy przed uruchomieniem ewolucji, dotyczy jedynie pewnej lokalnej podprzestrzeni, wyszczególnionej z globalnej przestrzeni wszystkich możliwych parametrów, ich typów i zakresów zmian, dla danej maszyny. Dobór odpowiednich parametrów w poniższej pracy został przeprowadzony intuicyjnie, metodą prób i błędów, a otrzymane wyniki dla poszczególnych badanych maszyn nie muszą stanowić globalnych maksimów ich funkcji przystosowania. Badanie metod poszukiwania globalnego maksimum funkcji przystosowania i analitycznych metod parametryzacji nie stanowiło zakresu tej pracy, lecz może stanowić jeden z kierunków dalszych badań.
Przy omawianiu poszczególnych typów agentów zostanie podany dokładny zestaw parametrów użytych do ewolucji, a dla niektórych typów agentów zostaną również podane wskazówki na temat trafności i efektywności wyboru tych parametrów.

8.2.1 Parametryzacja morfologii agenta

Budowę (morfologię) układu mechanicznego stanowiącego ciało agenta możemy parametryzować sterując zarówno pozycją i rozmiarem elementów konstrukcyjnych jak i punktami zaczepu przegubów. Dostępnymi parametrami morfologicznymi do budowy genotypu agenta są:

- Wymiary elementu nośnego (szerokość i wysokość).
- Pozycja elementu nośnego (pozycja lokalna - początkowa pozycja elementu w lokalnym układzie współrzędnych maszyny, zadana w fazie konstrukcji robota).
- Kąt elementu nośnego.
- Gęstość materiału elementu nośnego.
- Pozycja lokalna zaczepu przegubu (przegub obrotowy, przegub liniowy) względem elementu nośnego.
- Ograniczenie kątowe pracy przegubu obrotowego.
- Współczynnik sprężystości przegubu obrotowo-sprężynowego.

8.2.2 Parametryzacja układów napędowych i czujników agenta

Dostępnymi parametrami pracy układów napędowych oraz czujników przy budowie genotypu agenta są:

- Punkty zaczepu napędów i czujników względem elementu konstrukcyjnego.
- Wymiary czujnika (czujnik kontaktu).
- Maksymalny moment silnika obrotowego.
- Ograniczenie kątowe pracy silnika obrotowego.
- Wartość impulsu silnika impulsowego.
- Częstość pracy silnika impulsowego.
- Kierunek generowania impulsu w silniku impulsowym.
- Docelowa prędkość pracy siłownika.
- Moc siłownika.

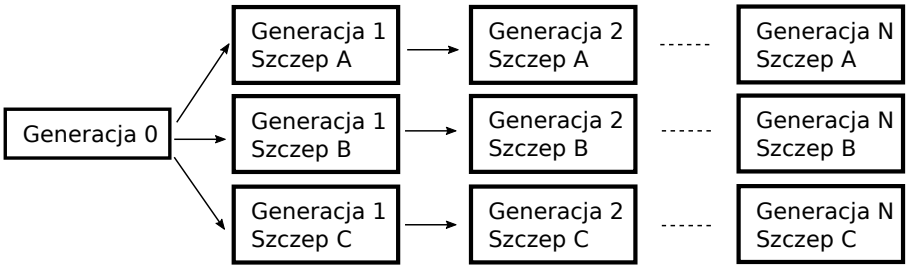
8.2.3 Parametryzacja programu sterującego agenta

Podobnie jak układ mechaniczny maszyny, sparametryzować możemy również kod sterujący agenta, a dokładniej wartości wybranych komórek zawierających parametry sterujące przebiegiem wykonania kodu agenta. Możemy w ten sposób uzależnić logikę działania maszyny i jej poszczególnych funkcji motorycznych od wartości parametru odpowiadającego ściśle danemu osobnikowi. Mogą to być np. parametry zawierające wartości graniczne wychylenia maszyny latającej, po przekroczeniu których silniki zmieniają tryb pracy umożliwiając balansowanie w powietrzu.

8.2.4 Parametryzacja urządzeń sieci neuronowych

Parametryzując wybrane urządzenie sieci neuronowej tak naprawdę parametryzujemy zestaw wag i współczynników aktywacji neuronów sieci. Liczba warstw sieci neuronowej jak i liczba neuronów poszczególnych warstw pozostaje niezmienna w trakcie ewolucji. Po sparametryzowaniu sieci neuronowej i połączeniu jej wejść i wyjść z wybranymi urządzeniami napędowymi i czujnikami, do genotypu maszyny dodany zostanie zestaw genów których liczbę określa równanie (6.3). Celem ewolucji maszyn których wagi urządzeń sieci neuronowych stanowią część genotypu maszyny, będzie więc dobór wartości wag w procesie ewolucji w taki sposób, by zachowanie maszyny wynikające z przetwarzanego przez sieć neuronową sygnału dało jej przewagę nad innymi osobnikami.

8.3 Algorytm ewolucyjny



Rysunek 8.1: Proces równoległej ewolucji trwającej N generacji dla 3 szczepów A, B i C.

Rysunek 8.1 przedstawia proces ewolucji trwającej N generacji. Symulacja prowadzona jest równolegle dla 3 szczepów, które łączy między sobą jedynie podstawowy zestaw wartości genów - fenotyp agentów generacji 0.
Wartości genów wszystkich agentów generacji 0 są ustalane przez użytkownika jako domyślne wartości parametrów maszyny wybrane w procesie parametryzacji. Każda z generacji może zawierać wielu agentów, z których każdy posiada unikatowy zestaw wartości parametrów maszyny (fenotyp) których liczba, typ i przykładowe wartości tworzą genotyp danej maszyny.

Strzałkami na rysunku 8.1 przedstawiono proces tworzenia nowej populacji maszyn i-tej generacji. W procesie tym celem jest utworzenie stałej dla każdej generacji liczby N nowych agentów, korzystając przy tym z puli genetycznej osobników (agentów) generacji poprzedniej danego szczepu. Do nowej generacji przechodzi automatycznie zdefiniowana przez użytkownika liczba L najlepszych agentów zwanych **osobnikami elitarnymi**, których całe fenotypy są kopiowane.
Proces tworzenia pozostałych $N - L$ agentów nowej populacji składa się z dwóch faz: **mutacji genów** i **wymiany genów** (z j. angielskiego: crossover), obie fazy zostaną omówione poniżej.
Proces tworzenia fenotypu nowego agenta generacji G_n na bazie dostępnej puli genetycznej agentów generacji G_{n-1} przebiega następująco. Najpierw wybierane są 2 osobniki A i B z generacji G_{n-1}. Ich wybór, zwany **selekcją**, polega na wylosowaniu dwu osobników z puli wszystkich osobników generacji G_{n-1} z niejednorodnym rozkładem prawdopodobieństwa, promującym osobniki lepsze (osobniki o większej wartości funkcji przystosowania).
Następnie rozpoczyna się faza tworzenia fenotypu nowego osobnika C na bazie fenotypów osobników A i B. Najpierw wykonywana jest procedura wymiany genów (z j. angielskiego: crossover), której mechanizm przedstawia poniższy algorytm:

Algorytm 9 Algorytm wymiany genów wybranych osobników A i B

1: Dane wejściowe: zbiór zawierający J wartości g_j^A genów osobnika A (fenotyp A) i J wartości g_j^B genów osobnika B (fenotyp B), parametr C_{rate} określający prawdopodobieństwo zajścia wymiany genów miedzy osobnikami.
2: Znajdź liczbę losową $r = RND(0, 1)$
3: **if** $r > C_{rate}$ **then**
4: **for** Dla każdej pary genów g_j^A i g_j^B **do**
5: Znajdź liczbę losową $r = RND(0, 1)$
6: **if** $r > 0.5$ **then**
7: $g_j^C = g_j^A$
8: **else**
9: $g_j^C = g_j^B$
10: **end if**
11: **end for**
12: **else**
13: Znajdź liczbę losową $r = RND(0, 1)$
14: **if** $r > 0.5$ **then**
15: Kopiuj fenotyp A do wyjściowego fenotypu C
16: **else**
17: Kopiuj fenotyp B do wyjściowego fenotypu C
18: **end if**
19: **end if**

Następnie na tak otrzymanym fenotypie osobnika C wykonywany jest proces mutacji

genów:

Algorytm 10 Algorytm mutacji genów wybranego osobnika

1: Dane wejściowe: zbiór zawierający J wartości g_j genów osobnika (fenotyp) i odpowiadających im N amplitud wartości g_j^{amp} genów osobnika, prawdopodobieństwo mutacji M_{chance}, amplituda mutacji M_{amp}.
2: **for** Dla każdego genu o wartości g_j **do**
3: Znajdź liczbę losową $r = RND(0, 1)$
4: **if** $r > M_{chance}$ **then**
5: $d = g_j^{amp} \cdot M_{amp}$
6: $g_j = -0.5 \cdot d + RND(d)$
7: **end if**
8: **end for**

W tym miejscu wyjściowy fenotyp jest kopiowany do nowo utworzonego agenta i osobnik jest gotowy do umieszczenia w świecie symulacji, po czym uruchamiana jest symulacja o długości czasu skonfigurowanej przez użytkownika. Po zakończeniu symulacji fenotyp osobnika i osiągnięta przez niego wartość funkcji przystosowania zostaje dodana do listy ukończonych osobników danej generacji. Gdy lista ukończonych symulacji osobników generacji G_n osiągnie rozmiar N, może rozpocząć się proces symulacji osobników generacji G_{n+1}.

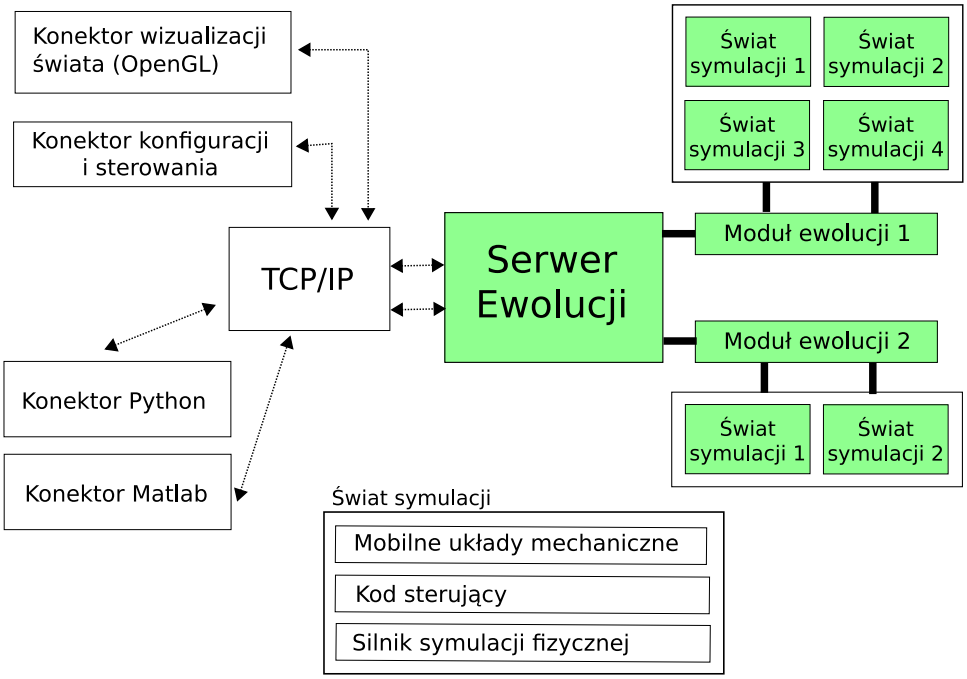
Rozdział 9

Modelowanie i budowa rozproszonego systemu ewolucji

Jednym z głównych celów pracy jest przebadanie optymalnego sposobu realizacji zróżnicowanych zadań w zależności od budowy morfologicznej i konfiguracji urządzeń tworzących mobilne roboty. W tym celu zaprojektowano i stworzono oprogramowanie - system obliczeń rozproszonych, umożliwiające projektowanie ruchomych układów mechanicznych, ich parametryzację, definiowanie zróżnicowanych środowisk symulacji i kryteriów przystosowawczych (funkcji przystosowania) oraz przeprowadzanie symulowanej ewolucji robotów.

9.1 Architektura rozproszonego systemu ewolucji

Na rysunku 9.1 przedstawiono schemat budowy oprogramowania do przeprowadzania symulacji ewolucji mobilnych układów mechanicznych. Można w nim wyróżnić 2 zasadnicze elementy, stanowiące odrębne funkcjonalnie części: serwer ewolucji, i konektory służące do komunikacji, zarządzania i wizualizacji wyników ewolucji.
Serwer ewolucji jest pojedynczym procesem, każdy świat symulacji jest uruchamiany w pojedynczym wątku, co pozwala na równoległe[20][21] symulowanie wielu światów jednocześnie, na poziomie jednej generacji maszyn.
Moduł ewolucji jest częścią realizującą pojedyncze zadanie przeprowadzenia symulacji ewolucji trwającej określoną ilość generacji dla wybranego typu maszyny w określonym środowisku. W jego kontekście, w miarę możliwości sprzętowych, symulacje kolejnych światów w następujących po sobie kolejnych generacjach maszyn są rozdzielane pomiędzy wątki procesora.



Rysunek 9.1: Architektura systemu ewolucji. Kolorem zielonym zaznaczono elementy wchodzące w skład procesu serwera ewolucji.

Linią przerywaną i strzałkami zaznaczono komunikację miedzy modułami zewnętrznymi a serwerem ewolucji, które stanowią osobne procesy. Komunikację zrealizowano z użyciem zaprojektowanego do tego celu protokołu opartego na TCP/IP. Takie podejście umożliwia rozdzielenie kontroli, konfiguracji i wizualizacji od silnika obliczeniowego (umieszczonego w serwerze ewolucji). Celem separacji części obliczeniowej było stworzenie multiplatformowego (używając standardu POSIX [19]) modułu obliczeniowego, zdolnego do wielowątkowej, równoległej symulacji wielu światów symulacji jednocześnie, umożliwiającego również uruchamianie i zarządzanie niezależnymi od siebie obliczeniami osobnych środowisk ewolucyjnych na rozproszonych fizycznie maszynach.
Choć w założeniach tej pracy była symulacja środowisk jednoagentowych i rozwiązanie w postaci jednowątkowych obliczeń w kontekście pojedynczego świata symulacji w zupełności wystarcza, kolejnym krokiem w kontynuacji rozpoczętych badań może być zrównoleglenie obliczeń na poziomie pojedynczego świata symulacji, co umożliwiłoby w przyszłości wsparcie obliczeniowe bardziej złożonych środowisk symulacji.

9.2 Protokół komunikacji zdalnej

Separacja modułów kontroli i wizualizacji z użyciem komunikacji sieciowej umożliwia zdalną konfigurację i zarządzanie częścią obliczeniową, która może być realizowana na osobnej, wyspecjalizowanej do tego maszynie, np. w centrum obliczeń wysokiej wydajno-

ści. Opracowany protokół, zbudowany na warstwie TCP/IP, zaprojektowano w postaci niewielkiej biblioteki C++ (interfejsu programistycznego API), na bazie której można dalej projektować konektory - dowolnego przeznaczenia aplikacje klienckie, służące do zarządzania, konfiguracji i podglądu wyników symulacji. Celem budowy interfejsu API było umożliwienie następującej funkcjonalności zdalnej:

- Budowę i oprogramowanie mobilnych układów mechanicznych
- Budowę i konfigurację inteligentnych środowisk symulacji
- Definiowanie zróżnicowanych schematów testowych ewolucji i odpowiadających im reguł: funkcji przystosowania, parametrów ewolucji
- Uruchamianie i zdalne monitorowanie procesu ewolucji

9.3 Serwer kreowania i jednoczesnego zarządzania wieloma światami ewolucji

Budowa modułu obliczeniowego - serwera ewolucji - umożliwia jednoczesne definiowanie różnych środowisk i schematów testowych oraz przeprowadzanie jednoczesnych równoległych obliczeń ich ewolucji, z użyciem architektury wielowątkowej. Zamysłem było umożliwienie wykonania obliczeń w centrach obliczeń wysokiej wydajności. W zależności od możliwości sprzętowych jednostki obliczeniowej (liczby dostępnych rdzeni i procesorów), użytkownik może skonfigurować maksymalną ilość wątków na którą rozdzielone zostaną obliczenia. Pojedynczy wątek odpowiedzialny jest za obliczenia związane z symulacją fizyczną pojedynczego świata symulacji zarządzanego przez jego moduł ewolucji (zob. rysunek 9.1). Świat symulacji zawiera określoną podczas konfiguracji ilość agentów umieszczonych w wybranym środowisku fizycznym (pojedynczym świecie symulacji) o wcześniej zdefiniowanych parametrach fizycznych i budowie.

Rozdział 10

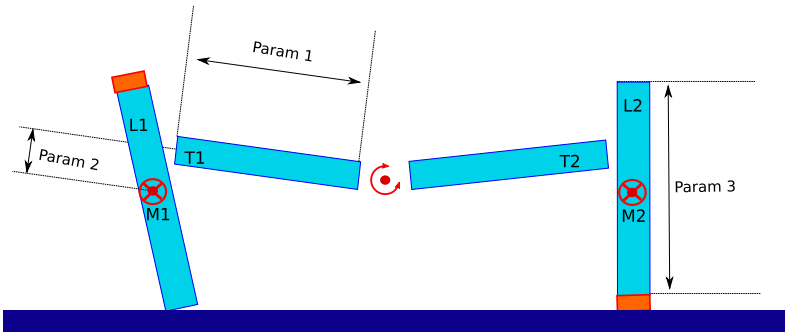
Wyniki symulacji

W tym rozdziale przedstawiono wyniki symulacji ewolucji i analizę kilku wybranych typów autonomicznych układów mechanicznych. Celem przeprowadzonych symulacji było zbadanie możliwości wyuczenia poprzez proces ewolucji kilku typów zachowań:

- efektywnego poruszania się w środowiskach bez przeszkód i z przeszkodami przez układy kroczące,
- nauki ruchu pełzającego przez układy o budowie segmentowej,
- bezkolizyjnego unoszenia się w powietrzu i podążania za ruchomym celem przez układy latające napędzane silnikami impulsowymi.

10.1 Układ kroczący

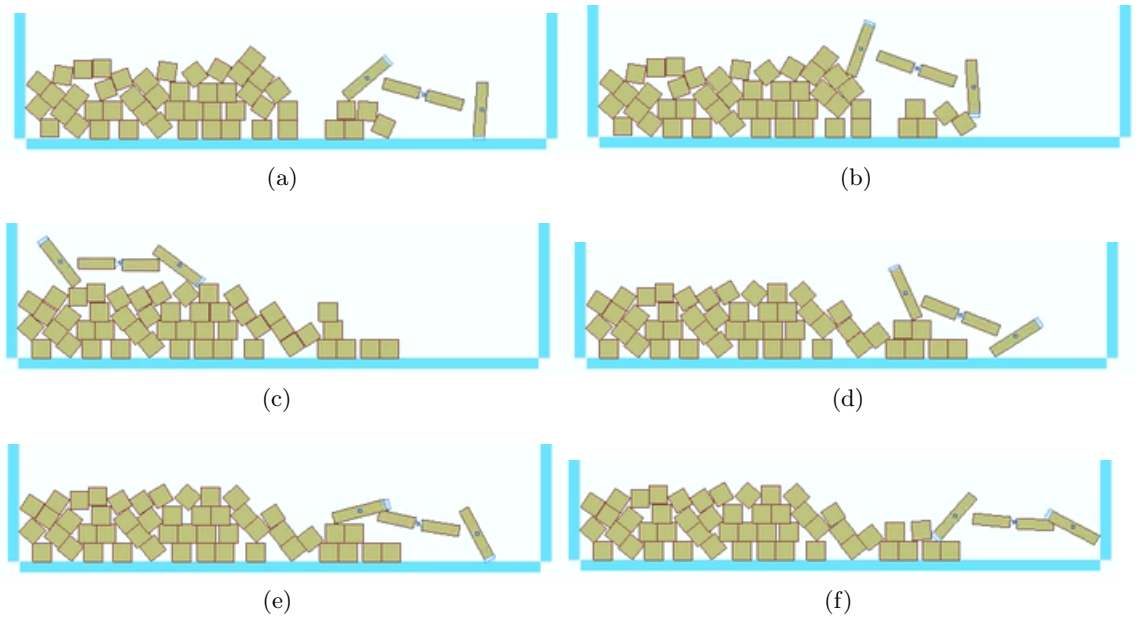
Na rysunku 10.1 przedstawiono model pierwszego z układów mechanicznych (nazwany $'Walker'$), który poddano optymalizacji z użyciem stworzonego systemu ewolucji.



Rysunek 10.1: Schemat budowy i parametryzacja mechanicznego układu kroczącego.

Maszyna składa się z dwóch elementów tworzących tułów ($T1$, $T2$) połączonych przegubem sprężynowym. Elementy konstrukcyjne tułowia połączone są z odnóżami ($L1$, $L2$)

przy pomocy dwóch silników obrotowych ($M1$, $M2$) o identycznej mocy i stałej prędkości docelowej pracy. Na końcu każdego z odnóż umieszczono po jednym czujniku kontaktu, którego reakcja na kolizję z jedną z przeciwległych ścian komory symulacji (prawą bądź lewą) została zaprogramowana by przełączyć kierunek pracy obu silników, powodując tym samym zmianę kierunku ruchu maszyny.
W celu przetestowania poprawności działania maszyny, przed poddaniem jej budowy parametryzacji i rozpoczęciem procesu ewolucji, maszynę umieszczono w komorze symulacji wypełnionej dynamicznymi (ruchomymi) przeszkodami imitującymi gruzowisko. Wizualizację pokonywania przez nią gruzowiska i przełączania kierunku pracy silników po dotarciu do przeciwległych ścian umieszczono na rysunku 10.2.

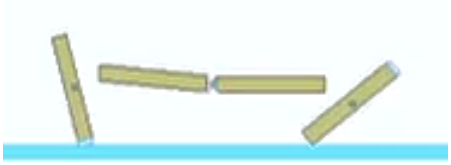


Rysunek 10.2: Układ mechaniczny typu $'Walker'$ poruszający się po ruchomym gruzowisku.

10.1.1 Wyniki ewolucji maszyny typu $'Walker'$ w środowisku bez przeszkód

Układ mechaniczny $'walker'$ poddano parametryzacji w celu znalezienia optymalnych wymiarów i punktu zaczepienia elementów konstrukcyjnych, które dadzą maszynie pokonywać drogę od ściany do ściany w jak jak najkrótszym czasie.
Jako parametry poddane mutacji w procesie ewolucji wybrano: długość elementu konstrukcyjnego tułowia ($'Param1'$), wysokość zaczepu silnika obrotowego łączącego kończyny z tułowiem ($'Param2'$) oraz długość kończyny ($'Param3'$).
Układ umieszczono w prostokątnej komorze symulacji, pozbawionej przeszkód, czas życia pojedynczego agenta ustalono na 20 sekund. Funkcję przystosowania zdefiniowano jako sumaryczną odległość przebytą w trakcie życia agenta. Przyjęto prawdopodobień-

(a) Szczep-A

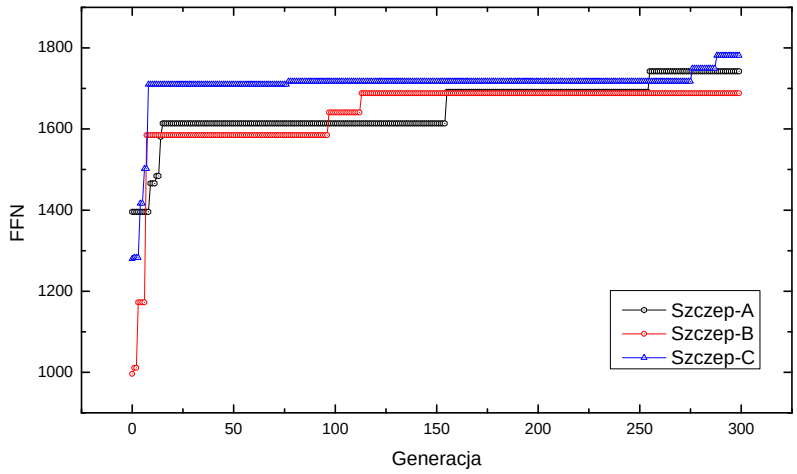


(b) Szczep-B

(c) Szczep-C

Rysunek 10.3: Najlepiej przystosowani przedstawiciele 3 wyewoluowanych szczepów maszyn po 300 generacjach. Wartości funkcji przystosowania: $FFN_A = 1781$, $FFN_B = 1688$, $FFN_C = 1793$

stwo mutacji równe 0.05 oraz amplitudę mutacji 0.8. Układ poddano ewolucji trwającej 300 generacji, dla 3 szczepów jednocześnie. Proces ewolucji układów dla wszystkich 3 szczepów przedstawiono na wykresie 10.4.



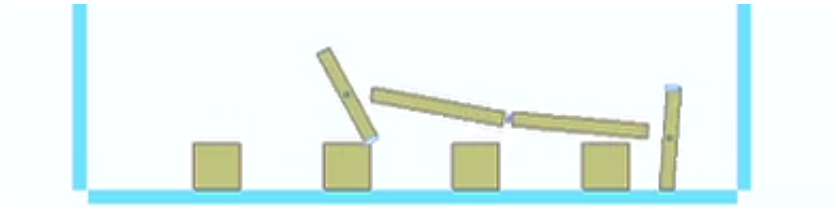
Rysunek 10.4: Przebieg ewolucji układu mechanicznego typu *'walker'*. Wykres funkcji przystosowania (FFN) najlepszego osobnika populacji w danej generacji, dla 3 odrębnych szczepów.

Jak widać wszystkie 3 szczepy osiągnęły po 300 generacjach stopień ewolucji pozwalający im na poruszanie się z dosyć podobną prędkością, czego oznaką jest niewielka różnica w wartości funkcji przystosowania. Długość odnóży dla wszystkich 3 szczepów

pozostała niezmienna. Dobra zbieżność wszystkich 3 wykresów funkcji przystosowania może oznaczać iż udało się dobrać parametry na tyle dobrze, że udało się znaleźć globalne maksimum funkcji przystosowania. Można zauważyć natomiast cechę wspólną (zob. rysunek 10.3) 2 najlepszych szczepów (A i C), czyli krótki element konstrukcyjny budujący tułów, dający obu odmianom mniejszą masę przy stosunkowo długich odnóżach i stałym dla wszystkich szczepów momencie obrotowym silnika, co skutkuje większą prędkość liniową ruchu maszyn. Z drugiej strony można było również zauważyć sporadycznie charakterystyczne ślizganie się po powierzchni maszyn o krótkich tułowiach, co było spowodowane rozwinięciem się tarcia dynamicznego na skutek zbyt małej masy maszyny i zbyt dużej mocy silnika.

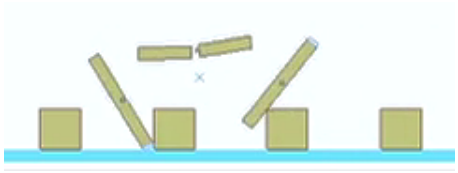
10.1.2 Wyniki ewolucji maszyny typu $'Walker'$ w środowisku z przeszkodami

Kolejnym badanym przypadkiem była analiza ewolucji robota $'Walker'$ w środowisku wypełnionym statycznymi (nieruchomymi, przymocowanymi do ścian pojemnika symulacji) przeszkodami (zob. rysunek 10.5).

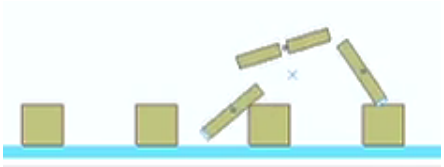


Rysunek 10.5: Środowisko testowe robota $'Walker'$ z przymocowanymi do podłoża przeszkodami.

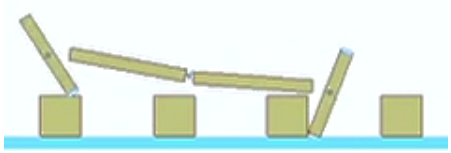
Maszyna została sparametryzowana w identyczny sposób co w przypadku poprzednim. Podobnie przyjęto prawdopodobieństwo mutacji 0.05, amplitudę mutacji 0.8. Układ poddano ewolucji trwającej 300 generacji, dla 3 szczepów jednocześnie, czas życia pojedynczego agenta ustalono na 20 sekund. Otrzymanych najlepiej przystosowanych przedstawicieli po zakończeniu procesu ewolucji umieszczono na rysunku 10.6. Jak widać dwa najlepsze szczepy (A i B) posiadają stosunkowo wysoko położony tułów, co umożliwia im unikanie kolizji z przeszkodami i utratę prędkości na skutek tarcia, która jest przyczyną słabego wyniku przedstawicieli szczepu C. Z przedstawionego na wykresie 10.7 procesu adaptacji osobników w kolejnych generacjach można również wyczytać iż istotnym dla końcowego wyniku procesu optymalizacji jest to jaką pulę genetyczną rozlosowano w procesie mutacji w pierwszej iteracji ewolucji (w pierwszej generacji) dla danego szczepu.



(a) Szczep-A



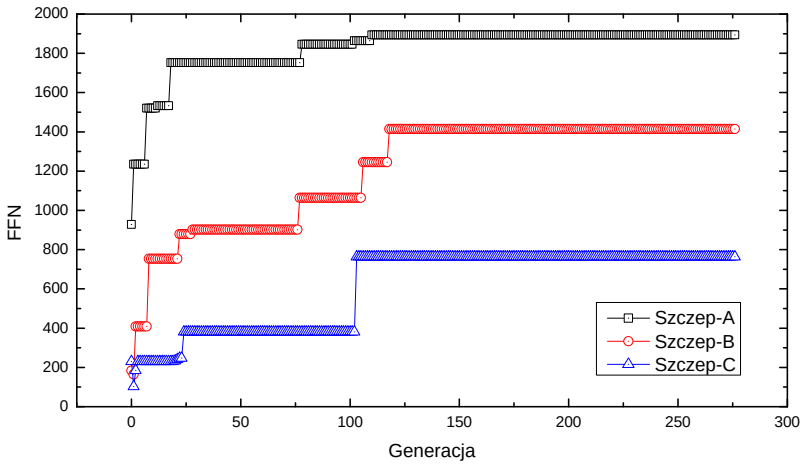
(b) Szczep-B



(c) Szczep-C

Rysunek 10.6: Najlepiej przystosowani przedstawiciele 3 szczepów maszyn $'Walker'$ po 300 generacjach ewolucji w środowisku z nieruchomymi przeszkodami. Wartości funkcji przystosowania: $FFN_A = 1893$, $FFN_B = 1414$, $FFN_C = 843$

Pomimo iż bazowa pula genetyczna wchodząca do pierwszej generacji dla wszystkich szczepów jest ta sama - jest to osobnik o początkowej konfiguracji wartości genów zadanej przez użytkownika - to jednak mutacje zachodzące w pierwszej generacji mogą zdeterminować wyższość danego szczepu nad innym. Możemy zasugerować dwie potencjalne przyczyny takiego stanu. Pierwszą z nich jest istnienie lokalnych maksimów, do których tak sparametryzowany układ zbiega, zdeterminowany mutacjami wczesnych generacji.



Rysunek 10.7: Przebieg ewolucji układu mechanicznego typu $'walker'$ w środowisku z nieruchomymi przeszkodami. Wykres funkcji przystosowania (FFN) najlepszego osobnika populacji w danej generacji, dla 3 odrębnych szczepów.

Druga przyczyną może być błędny dobór parametrów tworzących genotyp maszyny, który gdyby był optymalny, pozwalałby na dotarcie do globalnego maksimum wybranej podprzestrzeni parametryzacji, pomimo istnienia dużych różnic między fenotypami agentów pierwszych generacji a fenotypem agenta najlepszego w danej parametryzacji (globalnego maksimum).
Należy również zauważyć że w przeciwieństwie do poprzedniego badanego przypadku - ewolucji w środowisku pozbawionym przeszkód - tutaj rozbieżność miedzy końcowymi wynikami dla poszczególnych szczepów opisanymi funkcją przystosowania jest stosunkowo duża (ponad dwukrotnie większa dla szczepów A i C). Przyczynę można spróbować wyjaśnić analizując zachowanie dominujących osobników po 300 generacjach dla każdego ze szczepów. Najsłabszy szczep (C) traci dużą ilość energii kinetycznej przy tarciu długim niskozawieszonym tułowiem o przeszkody. Mógłby ten problem wyeliminować wydłużając kończyny lub podnosząc punkt zaczepu napędów w procesie mutacji, jednak przy tak długim tułowiu, stały, niewielki moment siły silników dla wszystkich szczepów nie pozwoliłby na uniesienie tułowia i obrót takiej kończyny. Drugi najlepiej przystosowany szczep (B) nie ma natomiast problemu związanego z tarciem gdyż posiada wysokopołożony tułów, na tyle krótki, by móc pomagać pokonywać przeszkody na odpowiednio długich kończynach. Najciekawszym przypadkiem jest jednak zdecydowanie najlepszy ze szczepów (A). Jak można zauważyć ten sam typ układu mechanicznego dla tych samych parametrów ewolucji osiągnął wynik lepszy w środowisku z przeszkodami ($FFN = 1893$) niż w środowisku bez przeszkód ($FFN = 1781$). Znajdując odpowiednią długość kończyn, tułowia i punkt zaczepu napędów szczep ten zdołał nie tylko unikać rozlokowanych przeszkód, ale wręcz użyć ich do tego by poruszać się szybciej. Umożliwił mu to ruch składający się z jednej strony z odpychania od przeszkód przy pomocy odpowiednio dobranej do odległości miedzy przeszkodami długości kończyn, a z drugiej strony z ześlizgiwania się długimi kończynami z przeszkody po wejściu na nią. Szczep ten dopasował się kształtem do środowiska w którym został umieszczony by zdobyć przewagę.

10.2 Układ pełzający

Na rysunku 10.8 przedstawiono schemat budowy mechanicznego układu nazwanego *'serpent'*, który zaprojektowano w celu zbadania możliwych sposobów ruchu maszyn imitujących zwierzęta o budowie segmentowej, poruszających się po ziemi ruchem pełzającym.

Rysunek 10.8: Schemat budowy układu mechanicznego typu *'serpent'*.

Maszyna składa się z segmentowego szkieletu: głowy i 4 połączonych silnikami obrotowymi prostokątnych elementów konstrukcyjnych. Na końcu głowy umieszczono czujnik żyroskopowo-pozycyjny do odczytu kierunku, w którym skierowana jest aktualnie gło-

wa maszyny. W odróżnieniu od opisanego w poprzednim rozdziale robota kroczącego, którego sposób zachowania opisano warunkami logicznymi w programie sterującym, tym co odpowiada za zachowanie i logikę tej maszyny jest połączona z układem napędowo - sensorycznym sieć neuronowa. Sieć neuronowa została wybrana jako jedyny element podlegający parametryzacji w maszynie, tworząc w całości jej genotyp.
Sieć posiada 6 wejść pod które podłączono odpowiednio: odczyt czujnika żyroskopowo-pozycyjnego na głowie maszyny zawierający aktualny wektor kierunkowy (2 rzeczywiste składowe), oraz odczyty kąta wychylenia kolejnych silników w radianach (4 wartości rzeczywiste). Pojedyncza warstwa wewnętrzna (ukryta) sieci składa się z 15 węzłów. Wyjściem sieci są 4 węzły.
Program sterujący maszyną *'serpent'* ma za zadanie bez wprowadzania jakiejkolwiek zbędnej logiki warunkowej do zachowania maszyny, połączyć część sensoryczną z napędem przy pomocy sieci neuronowej symulującej „mózg" maszyny i pozwolić zadecydować o takim a nie innym ruchu maszyny poprzez proces przetwarzania sygnału wejściowego z sieci z użyciem dobranych w procesie ewolucji współczynników sieci neuronowej (wag i progów aktywacji neuronów).
Na początku pętli głównej programu sterującego wykonywany jest odczyt wektora kierunkowego z czujnika na głowie oraz kątów wychylenia silników normalizując je wartością graniczną wychylenia silnika ($MOT\_ANG\_LIMIT = 0.3\pi$):

Listing 10.1: Odczyt czujników

```
DEVCTL   "GPS1", DEV_CTL_GYROPOS_SENSOR_GET_ANGLE_VECTOR, "NN.HEADANG"

DEVCTL   "MOT1", DEV_CTL_MOTOR_GET_ANGLE, "NN.MOTANG1"
FDIV     "NN.MOTANG1", MOT_ANG_LIMIT, "NN.MOTANG1"

DEVCTL   "MOT2", DEV_CTL_MOTOR_GET_ANGLE, "NN.MOTANG2"
FDIV     "NN.MOTANG2", MOT_ANG_LIMIT, "NN.MOTANG2"
```

Następnie wartości przekazane są na wejścia sieci neuronowej, i odczytane są 4 wartości wyjścia sieci (znormalizowane wartości rzeczywiste), które są mnożone przez stałą maksymalną prędkość obrotową ($MOT\_MAX\_SPEED = 3\pi$) i z ich użyciem wykonane są kolejno rozkazy zmiany prędkości obrotowej 4 silników:

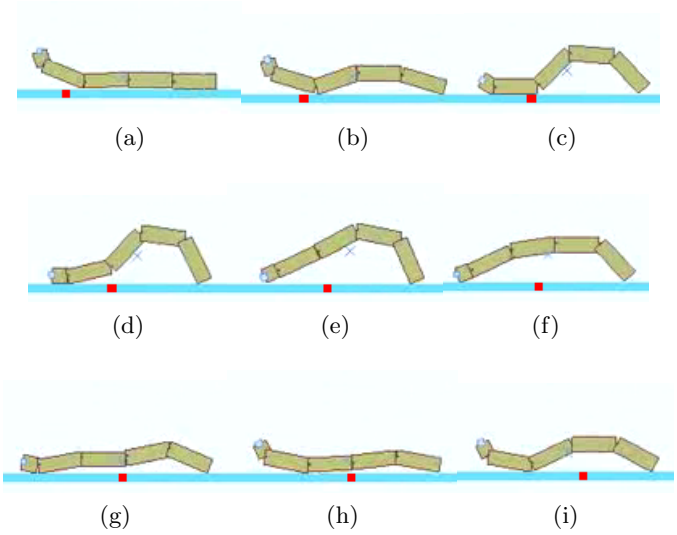
Listing 10.2: Wykonanie sieci neuronowej dla danych wejściowych z czujników i przekazanie wyjścia do układów napędowych.

```
DEVCTL   "NN1", DEV_CTL_NEURAL_NET_SET_INPUT_SIGNALS, "NN.HEADANGX"
DEVCTL   "NN1", DEV_CTL_NEURAL_NET_EXECUTE_INDIRECT, "NN.OUT1"

FMUL     "NN.OUT1", MOT_MAX_SPEED, "NN.OUT1"
FSUB     "NN.OUT1", MOT_MAX_SPEED/2, "NN.OUT1"
DEVCTL   "MOT1", DEV_CTL_MOTOR_SET_TARGET_SPEED, "NN.OUT1"

FMUL     "NN.OUT2", MOT_MAX_SPEED, "NN.OUT2"
FSUB     "NN.OUT2", MOT_MAX_SPEED/2, "NN.OUT2"
DEVCTL   "MOT2", DEV_CTL_MOTOR_SET_TARGET_SPEED, "NN.OUT2"
```

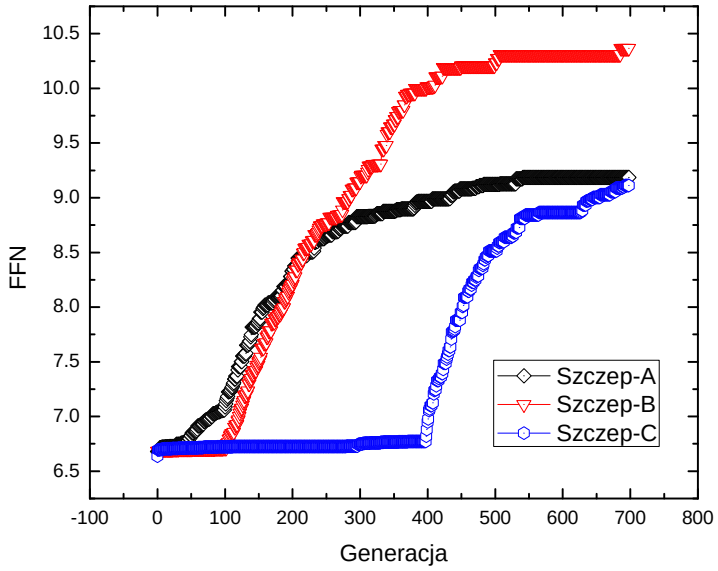
Funkcję przystosowania zdefiniowano jako:

$$FFN(n) = \sum_{i=1}^{n} d_i, \tag{10.1}$$

gdzie d_i to odległość środka ciężkości maszyny od lewej ściany komory symulacji, w i-tym kroku iteracyjnym symulacji świata, a n to liczba wszystkich kroków. Ponieważ w miarę zbliżania się osobnika do lewej ściany, przyczynek do wartości sumy wyrażonej funkcją FFN będzie również wzrastać, można spodziewać się, iż w miarę liniowego wzrostu prędkości maszyny w kolejnych generacjach (czyli w miarę postępu w przystosowaniu do postawionego przed nią zadania), wartość funkcji przystosowania najlepszych osobników kolejnych generacji będzie rosła nieliniowo.
Ewolucję trwającą 700 generacji przeprowadzono dla 6 szczepów w zamkniętej prostokątnej komorze symulacji o wymiarach 140×80, z współczynnikiem tarcia 1.0 oraz współczynnikiem sprężystości 0.0. Czas życia agenta ustalono na 10 sekund (czyli dwa razy krócej niż w przypadku maszyny kroczącej), ze względu na przewidywaną małą początkową ruchliwość agenta (spowodowaną brakiem wyuczenia sieci), a co za tym idzie krótszy potrzebny czas do zarejestrowania widocznego postępu w nauce poruszania się. Stały dla wszystkich maszyn moment obrotowy wszystkich silników przyjęto równy $3 \cdot 10^6$. Po zakończeniu procesu ewolucji, wyodrębniono 3 najlepsze szczepy. Każdy z nich cechowała identyczna metoda poruszania się, przypominająca pełzanie gąsienicy owada. Metodę ruchu przedstawiciela najlepszego ze szczepów oraz zestawienie wartości funkcji przystosowania dla 3 najlepszych szczepów przedstawiono na rysunkach 10.9 i 10.10.

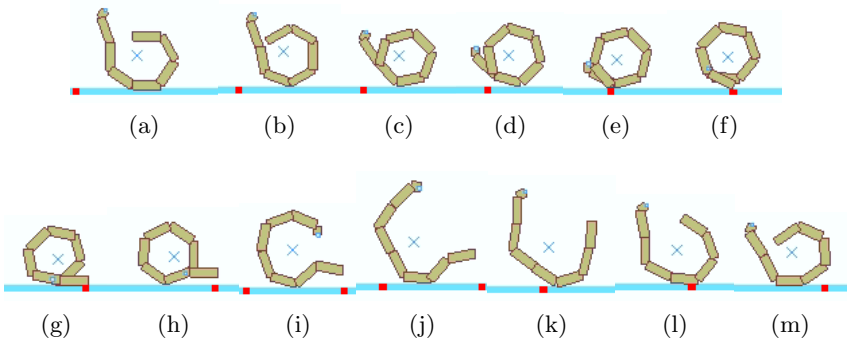


Rysunek 10.9: Sposób ruchu najlepszego osobnika populacji po 700 generacjach ewolucji 5-segmentowego robota typu *'serpent'*. Czerwona kropka na podłożu została umieszczona jako punkt referencyjny do zobrazowania przemieszczenia w kierunku poziomym.

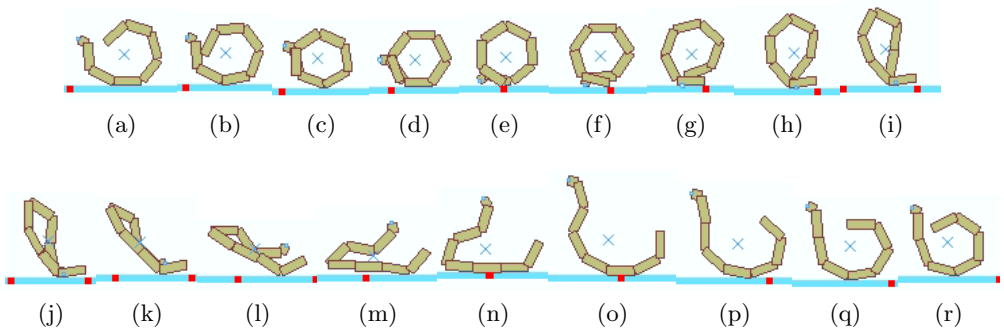


Rysunek 10.10: Przebieg ewolucji 5-segmentowego układu mechanicznego typu *'serpent'* o momencie obrotowym silnika $3 \cdot 10^6$. Wykres przedstawia wartość funkcji przystosowania (FFN) najlepszego osobnika populacji w danej generacji, dla 3 najlepszych z 6 symulowanych szczepów.

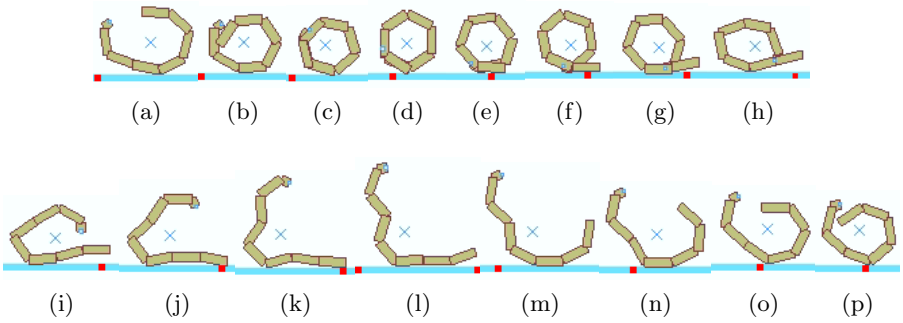
Kolejnym etapem było przeprowadzenie ewolucji układu *'serpent'* o większej liczbie segmentów szkieletu i zwiększonej mocy silników. Zmodyfikowano maszynę dodając jej 2 segmenty (całkowita liczba segmentów: 7) i zmieniono moment silników z $3 \cdot 10^6$ na $5 \cdot 10^6$. Maszynę umieszczono w świecie symulacji o identycznych parametrach co poprzednio, zwiększono jedynie wymiary komory symulacji na 700×100, pozostawiając punkt startowy maszyny przy prawej ścianie a punkt docelowy na lewej ścianie. System poddano ewolucji trwającej 300 generacji dla 6 szczepów.
Wszystkie otrzymane osobniki charakteryzował w tym przypadku inny sposób poruszania się, polegający na ruchu zamachowym i wykorzystaniu bezwładności. Co ciekawe, ta tendencja do promowania ruchu zamachowego w procesie ewolucji charakteryzowała wszystkie otrzymane szczepy tej wersji robota *'serpent'* i wszystkie najlepsze osobniki poszczególnych szczepów po 300 generacjach poruszały się w ten właśnie sposób.
Dało się natomiast wyróżnić różnice w sposobie ruchu pomiędzy poszczególnymi szczepami, które stanowiły o również o lepszej (lub gorszej) adaptacji do nałożonych warunków środowiska wyrażonej w wartości funkcji przystosowania.
Różnice w sposobie poruszania się dla przedstawicieli 3 najlepszych szczepów ukazano na rysunkach 10.11 - 10.13.



Rysunek 10.11: Sposób ruchu najlepszego osobnika szczepu B po 700 generacjach.

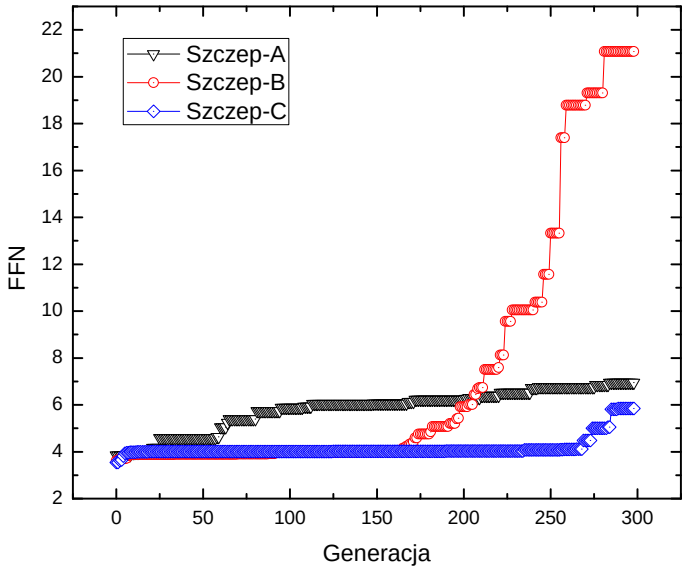


Rysunek 10.12: Sposób ruchu najlepszego osobnika szczepu C po 700 generacjach.



Rysunek 10.13: Sposób ruchu najlepszego osobnika szczepu A po 700 generacjach.

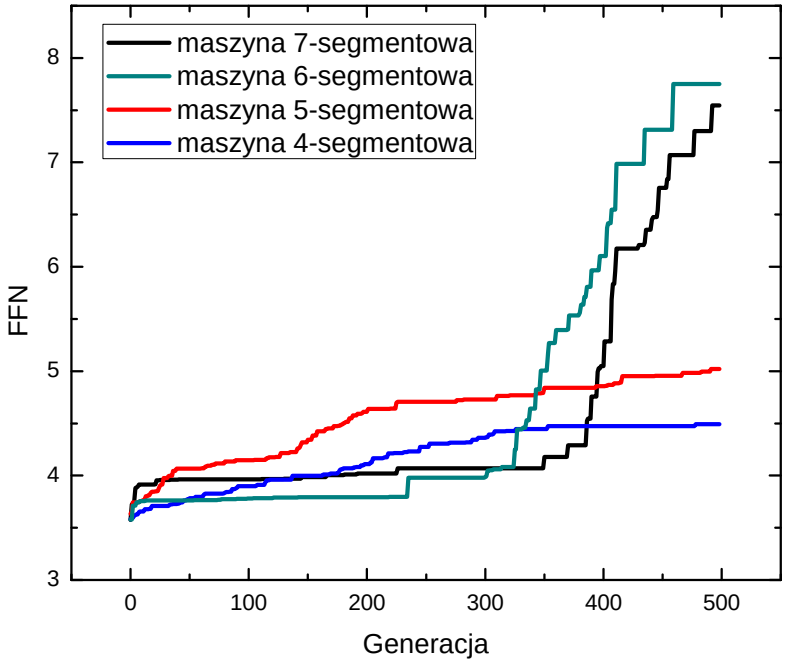
Z przedstawionych na rysunku 10.14 wykresów funkcji przystosowania (FFN) 3 najlepszych szczepów należy zauważyć dużą przewagę jaką uzyskali przedstawiciele szczepu najlepszego (A) nad szczepami B i C, pomimo iż ogólna zasada poruszania się osobników wszystkich tych szczepów pozostaje ta sama.



Rysunek 10.14: Przebieg ewolucji 7-segmentowego układu mechanicznego typu *'serpent'* z momentem silnika $5 \cdot 10^6$. Wykres przedstawia wartość funkcji przystosowania (FFN) najlepszego osobnika populacji w danej generacji, dla 3 z 6 otrzymanych szczepów.

Poza bezpośrednią przyczyną takiego stanu - czyli tym że szczep A zdołał „nauczyć się" poruszać przyjmując kształt najbardziej przypominający okrąg i tocząc się tylko przez krótką chwilę formować inny kształt w celu wzięcia zamachu - tak duży skok wartości funkcji FFN jest spowodowany sugerowaną wcześniej nieliniową zależnością miedzy średnią prędkością ruchu maszyny (tym jak daleko zajdzie w trakcie życia) a wartością funkcji przystosowania opisanej wzorem (10.1).

Postanowiono zbadać dla jakiej ilości segmentów szkieletu następuje przejście w pomiędzy pełzającym a toczącym sposobem poruszania się, oraz porównać jak efektywność ruchu (wartości funkcji przystosowania wyuczonego osobnika) zmienia się w miarę wzrostu liczby segmentów szkieletu. W tym celu przeprowadzono kolejno cztery symulacje ewolucji maszyny typu *'serpent'* dla czterech różnych długości szkieletu: 4, 5, 6 i 7-elementowego, w komorze symulacji o wymiarach 700×100, przy stałym momencie silnika dla wszystkich maszyn równym $5 \cdot 10^6$. Czas życia agenta ustalono na 20 sekund. Porównanie wyników efektywności maszyn, w postaci zestawienia wykresów funkcji przystosowania po ewolucji trwającej 500 generacji umieszczono na rys. 10.15.



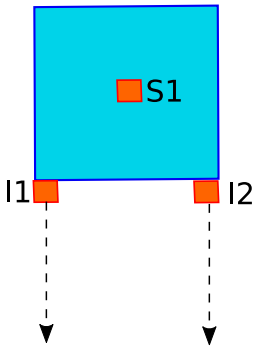
Rysunek 10.15: Zestawienie przebiegu ewolucji trwającej 500 generacji dla maszyn typu $'serpent'$ o 4 różnych długościach szkieletu.

Można zauważyć skok w wartości funkcji przystosowania najlepiej wyuczonych osobników, pomiędzy maszynami z cztero- i pięcioelementowym szkieletem, a maszynami z 6 i 7-elementowym szkieletem. Przyczyną jest zapewne pojawienie się szukanego przez nas przejścia w sposobie ruchu: u maszyn 6-cio elementowych, podobnie jak u maszyn 7-elementowych, wśród najlepszych osobników zaobserwowano ruch toczący, natomiast najlepsze maszyny 4 i 5-elementowe poruszały się pełzając.
Końcowym wnioskiem podsumowującym analizę ewolucji maszyn typu $'serpent'$ jest fakt, iż znaleziona, alternatywna forma ruchu poprzez toczenie jest bardziej efektywna (szybsza) niż ruch pełzający, który przyjęto podczas projektowania maszyny typu $'serpent'$ jako główną (najbardziej oczywistą) formę ruchu tego typu układów mechanicznych.

10.3 Układ latający

Ostatnim typem zachowania którego naukę postanowiono zbadać była umiejętność utrzymywania się w powietrzu. Najprostszy model maszyny przedstawiono na rysunku 10.16. Maszyna składa się z pojedynczej bryły o wymiarach 20×20, umocowanych

u spodu dwóch silników impulsowych o stałej wielkości impulsu $600K[\frac{N}{s}]$ i zmiennej kontrolowanej siecią neuronową częstości pracy oraz czujnika żyro-pozycyjnego zamocowanego na środku. Na środku maszyny umieszczono czujnik żyropozycyjny służący do odczytu wychylenia maszyny od poziomu równowagi.



Rysunek 10.16: Schemat budowy 2-silnikowego układu mechanicznego typu $'drone'$. Symbolem $S1$ oznaczono czujnik żyropozycyjny, natomiast symbolami $I1$, $I2$ silniki impulsowe. Linią przerywaną oznaczono kierunek generowania impulsu przez silniki.

Maszynę wyposażono w 1 warstwową sieć neuronową o 2 wejściach, 2 wyjściach i 15 neuronach warstwy ukrytej. Podobnie jak w przypadku maszyn $'serpent'$, sieć neuronowa została wybrana jako jedyny element parametryzujący maszynę, a jej wagi i współczynniki aktywacji neuronów jako jedyny składnik genotypu. Do urządzenia sieci neuronowej podłączono na wejściu 2 sygnały zawierające znormalizowany wektor wskazujący wychylenie maszyny o poziomu równowagi. Wyjście sieci neuronowej połączono z sygnałem kontrolującym częstotliwość pracy silników impulsowych (mnożąc sygnał wyjściowy sieci przez stałą maksymalną wartość pracy silników).

10.3.1 Unikanie kolizji

Jako pierwszy cel do osiągnięcia na drodze optymalizacji z użyciem algorytmów genetycznych postawiono nauczenie maszyny utrzymania się w powietrzu w sposób całkowicie pozbawiony kontaktu ze ścianami komory symulacji. Adaptację maszyny do tak

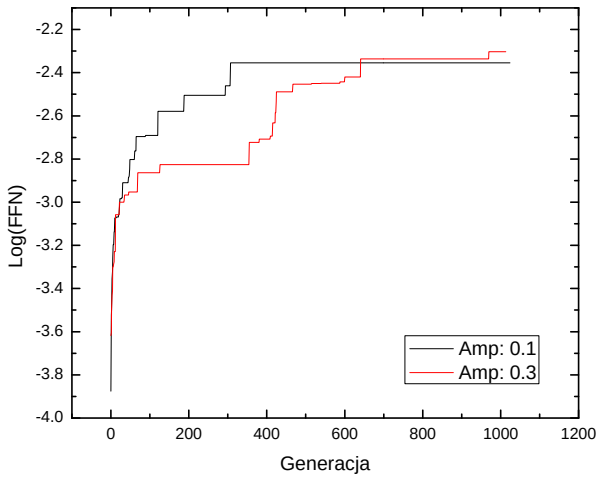
postawionego zadania opisano następująco zdefiniowaną funkcją przystosowania:

$$FFN(n) = \sum_{i=1,j=1}^{n,m} -a_j \cdot \delta_{ij}$$

$$\delta_i = \begin{cases} 1, & \text{gdy w i-tym kroku symulacji maszyna koliduje z j-tą ścianą komory symulacji} \\ 0, & \text{gdy w i-tym kroku symulacji maszyna nie koliduje j-tą ścianą komory symulacji} \end{cases} \quad (10.2)$$

$$a_j = \begin{cases} 1, & \text{gdy nastąpiła kolizja z prawą(j=1) lub lewą(j=3) ścianą komory symulacji} \\ 2, & \text{gdy nastąpiła kolizja z górną(j=0) lub dolną(j=2) ścianą komory symulacji} \end{cases}$$

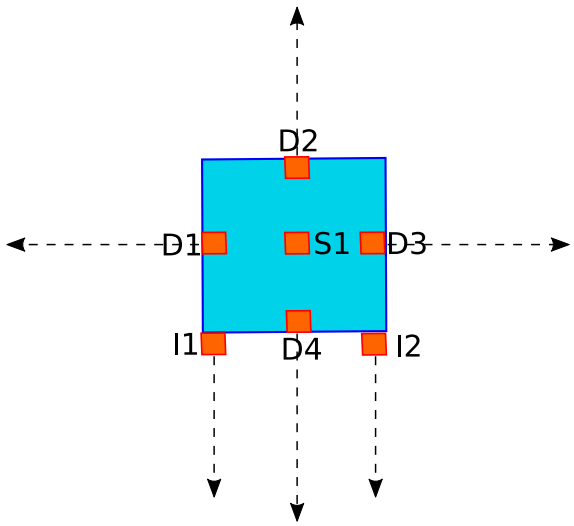
Tak zdefiniowany model poddano ewolucji trwającej 1000 generacji, dla 2 różnych wartości amplitudy mutacji: 0.1 i 0.3, po 6 symulowanych szczepów dla każdej z wartości amplitudy mutacji.
Wynik ewolucji przedstawiono na wykresie 10.17.



Rysunek 10.17: Wykresy funkcji przystosowania (FFN) w skali logarytmicznej, dla ewolucji 2-silnikowego układu mechanicznego typu *'drone'* dla 2 najlepszych szczepów każdej z amplitud mutacji: 0.1 i 0.3.

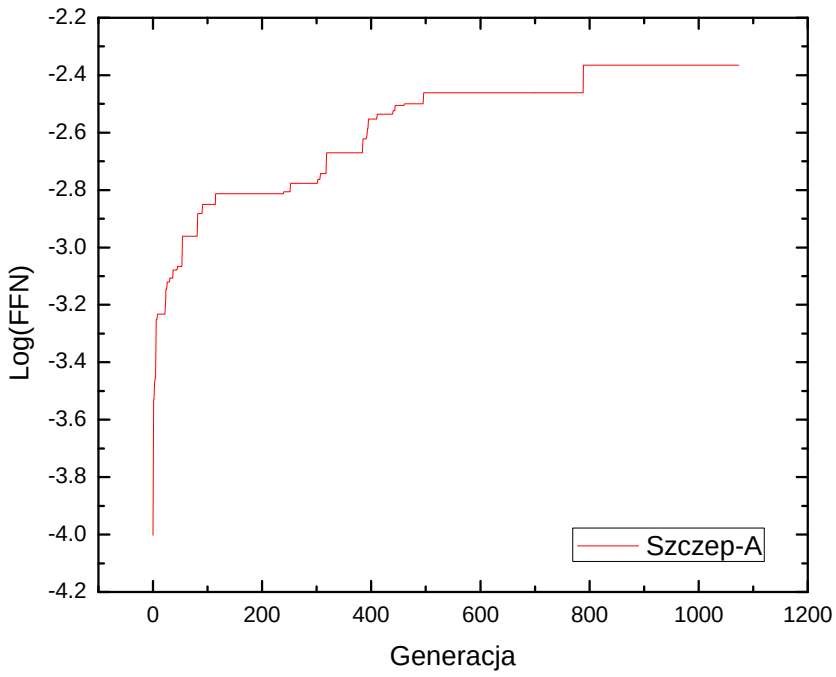
Dla obu badanych amplitud mutacji otrzymano podobny wynik funkcji przystosowania po ukończeniu procesu ewolucji, dla współczynnika 0.3 najlepszy ze szczepów

osiągnął wartość funkcji 200, dla współczynnika 0.1 osiągnął wartość 225. Sposób ruchu najlepszych osobników w obu badanych przypadkach po 1000 generacji charakteryzował się zdolnością do utrzymania braku kontaktu z podłożem i sufitem komory symulacji, oraz systematycznym uderzaniem kolejno o prawą i lewą ścianę komory. Mogło to być spowodowane faktem, iż jedynym sposobem postrzegania świata przez maszynę był wgląd w jej wychylenie kątowe - sieć neuronowa połączona na wyjściu z sygnałami częstości pracy silników posiadała na wejściu jedynie dwa sygnały w postaci współrzędnych znormalizowanego wektora wskazującego wychylenie od osi OX, odczytane z czujnika żyro-pozycyjnego. Dlatego też postanowiono wprowadzić do jej układów sensorycznych element który umożliwi detekcję zbliżających się przeszkód. W tym celu podłączono 4 czujniki odległości, po jednym na każdej ze ścian maszyny (zob. rysunek 10.18).



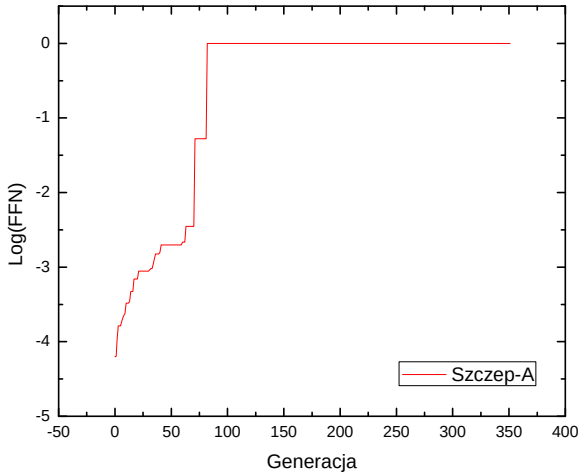
Rysunek 10.18: Schemat budowy 2-silnikowego układu mechanicznego typu *'drone'* zawierającego 4 czujniki odległości do celu $D1$ - $D4$ umieszczone na środku każdej ze ścian maszyny. Linią przerywaną oznaczono kierunek detekcji odległości każdego z czujników.

Czujniki sprzężono z wejściem sieci neuronowej, normalizując odczyt z czujników przed przekazaniem sygnału na wejście sieci.
Układ poddano ewolucji trwającej 1000 generacji, dla wartości amplitudy mutacji 0.3. Wynik ewolucji przedstawia wykres 10.19.



Rysunek 10.19: Wykres funkcji przystosowania (w skali logarytmicznej) dla ewolucji 2-silnikowego układu mechanicznego typu $'drone'$ wyposażonego w czujniki odległości.

Osiągnięto nieznacznie lepszą wartość funkcji przystosowania dla najlepszego z otrzymanych szczepów, równą 180. Cechą charakterystyczną ruchu pozostało uderzanie naprzemian w boczne ściany komory symulacji, wytracając w ten sposób całą prędkość w kierunku poziomym, zauważyć natomiast można było próbę balansowania wychylenia silnikami w celu uniknięcia kolizji ze ścianą, na krótko przed kolizją. Nie była ona jednak odpowiednio zsynchronizowana z prędkością ruchu maszyny, przez co próba balansowania była zbyt późna lub niewystarczająco silna by uniknąć kolizji.
W tym celu wyposażono maszynę w wejściowe sygnały sensoryczne umożliwiając jej „odczucie" pędu z jakim się porusza i precyzyjniejsze dostosowanie częstości pracy silników impulsowych w celu uniknięcia kolizji ze ścianą. Do sieci neuronowej podłączono 3 wejściowe sygnały rzeczywiste: wektor prędkości postępowej oraz prędkość kątową, oba odczytane z czujnika żyro-pozycyjnego.
Przeprowadzona ewolucja tej wersji modelu, wstępnie skonfigurowana na 1000 generacji, dała już w 81 generacji maksymalną wartość funkcji przystosowania, równą zero (zob. rysunek 10.20).



Rysunek 10.20: Wykres funkcji przystosowania (w skali logarytmicznej) dla ewolucji 2-silnikowego układu mechanicznego typu *'drone'* wyposażonego w czujniki odległości, z sprzężonym z siecią neuronową odczytem prędkości kątowej i postępowej.

Osiągnięty więc został założony dla tego typu maszyny cel, model maszyny uzyskał umiejętność utrzymywania się w powietrzu bez jakiejkolwiek kolizji ze ścianami komory symulacji przez cały okres życia.

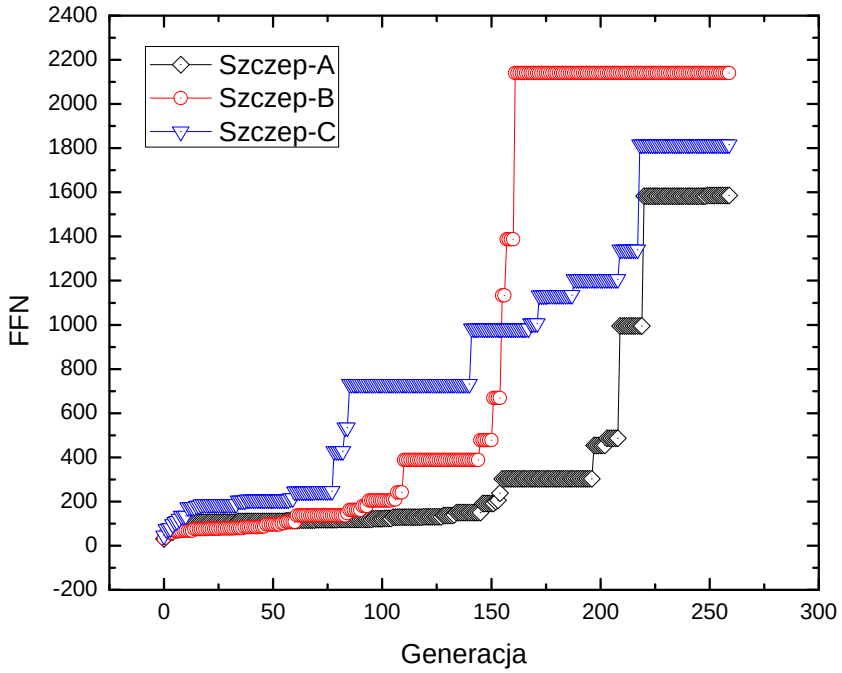
10.3.2 Śledzenie nieruchomego celu

Następnie zbadano możliwość wykorzystania zbudowanego systemu do nauki utrzymywania się maszyn w powietrzu opartego na podążaniu za celem umieszczonym z dala od podłoża komory symulacji. W tym celu użyto bazowej maszyny typu *'drone'* (zob. rysunek 10.16) pozbawionej czujników odległości. Wyposażono ją natomiast w czujnik typu radar, umożliwiający odczyt pozycji obiektów zadanego typu znajdujących się w komorze symulacji. Do wejść sieci neuronowej podłączono 5 znormalizowanych sygnałów: wektor kierunku do celu odczytany z czujnika radarowego (2 sygnały), wektor prędkości postępowej maszyny (2 sygnały) oraz prędkość kątową maszyny odczytane z czujnika żyro-pozycyjnego.
Funkcję przystosowania opisującą adaptację do postawionego celu zdefiniowano następująco:

$$FFN(n) = \sum_{i=1}^{n} d_i, \tag{10.3}$$

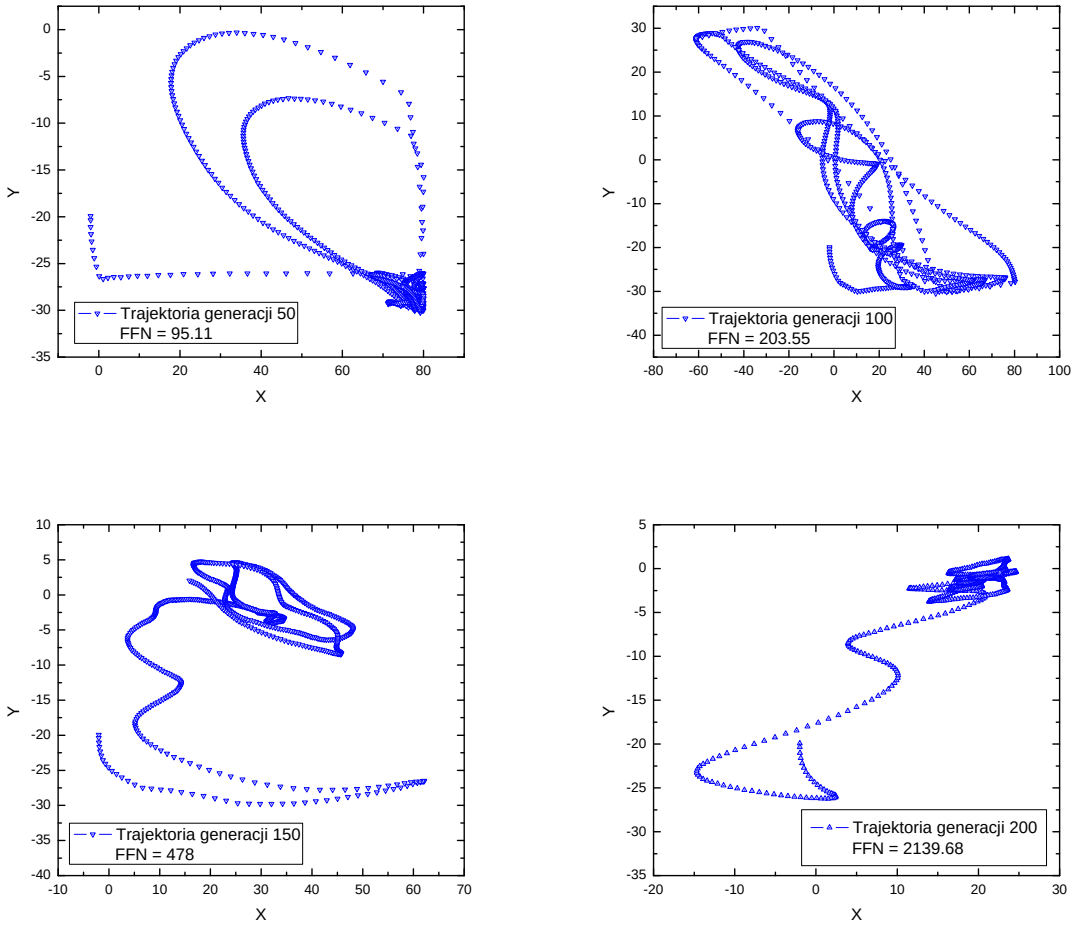
gdzie d_i to odległość środka masy maszyny od celu, w i-tym kroku iteracji symulowanego świata.
Punkt startowy maszyny umieszczono w powietrzu, w pozycji $(0, 20)$. Nieruchomy cel w

postaci markera umożliwiającego jego identyfikację przez urządzenie radarowe, umieszczono w pozycji $(20, 0)$. Maszynę umieszczono w komorze symulacji o wymiarach 180×80. Czas życia agenta ustalono na 20 sekund, krok numeryczny silnika symulacji na 5 milisekund.
Tak zdefiniowany model poddano ewolucji trwającej 250 generacji dla 6 szczepów. Wynik ewolucji dla otrzymanych 3 najlepszych szczepów przedstawiono na rysunku 10.21.



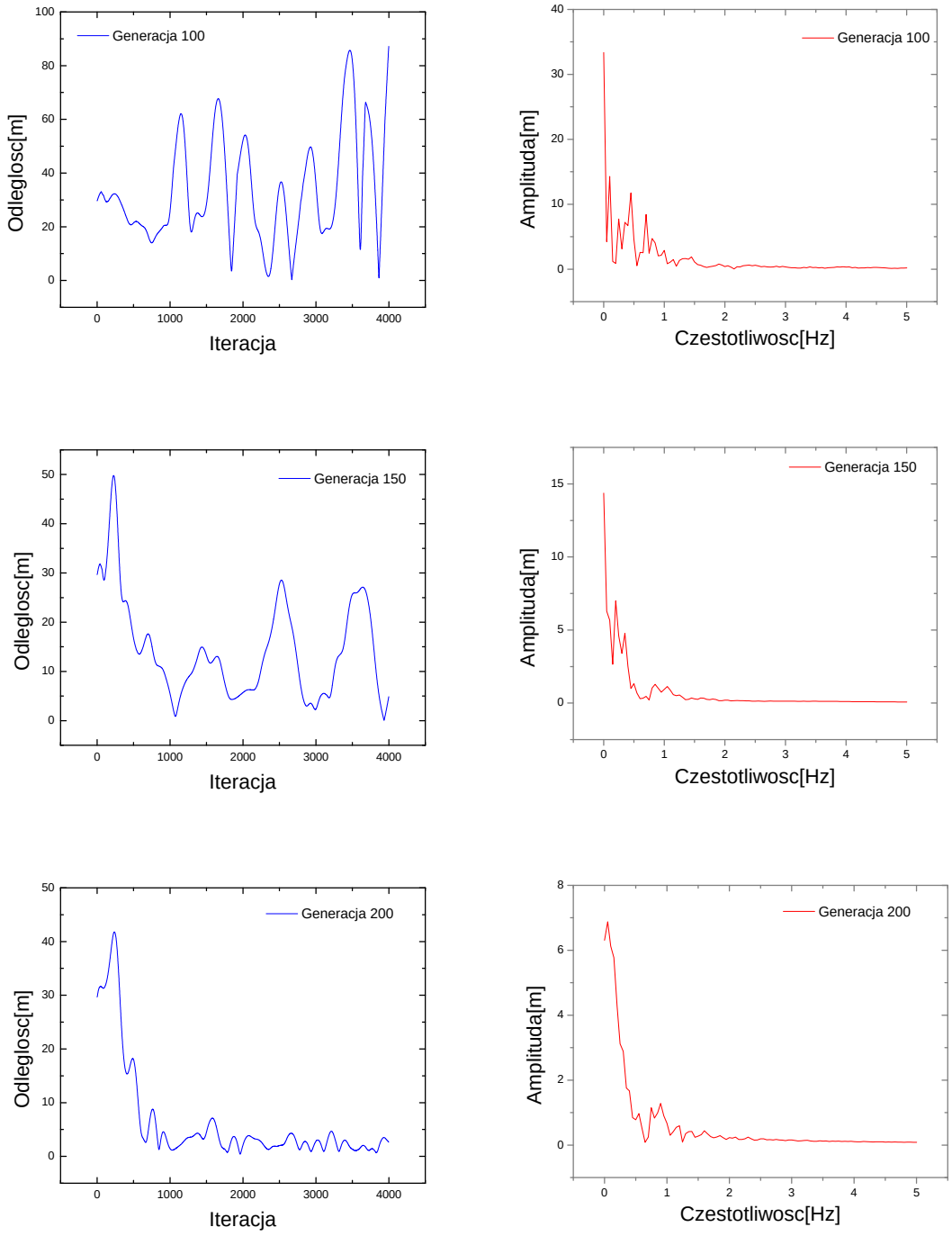
Rysunek 10.21: Postęp ewolucji 3 najlepszych szczepów dla układu mechanicznego typu *'drone'* w środowisku promującym podążanie za nieruchomym celem.

Na rysunku 10.22 umieszczono porównanie trajektorii ruchu osobników 4 kolejnych generacji w miarę postępu ewolucji, dla najlepszego z otrzymanych szczepów.



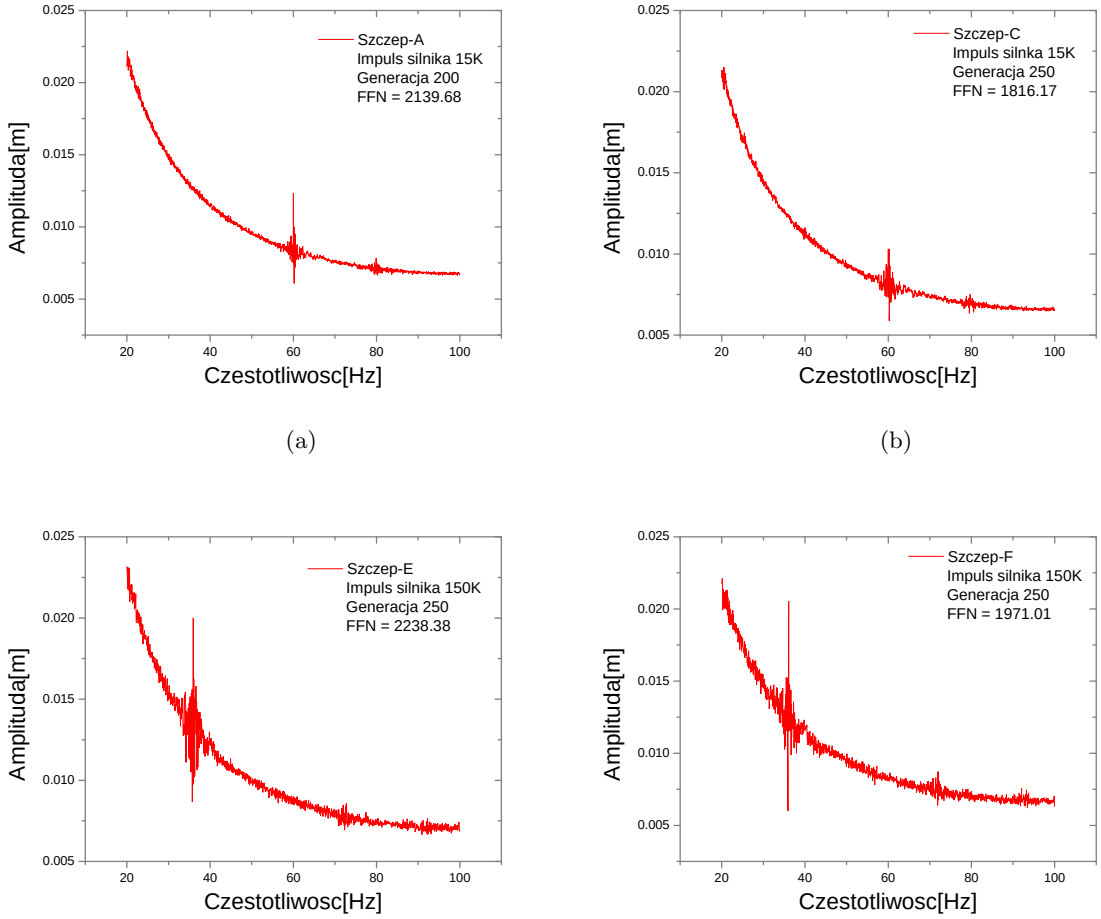
Rysunek 10.22: Trajektorie czterech kolejnych generacji najlepszego szczepu (B) maszyn latających typu $'drone'$ ewoluujących w środowisku promującym podążanie za nieruchomym celem. Cel umieszczono w punkcie $(20, 0)$

Przeprowadzono analizę zależności poziomu wyuczenia maszyny od częstości jej oscylacji wokół celu. Na rysunku 10.23 zestawiono wykresy funkcji odległości od celu i ich transformat Fouriera, dla trzech generacji najlepszego z otrzymanych szczepów (B). Można zauważyć wraz ze wzrostem przystosowania osobników spadek dominującej amplitudy oscylacji i przesunięcie pików FFT o względnie wysokich amplitudach (powyżej 1) w kierunku zerowej częstości widma. Można ten fakt traktować jako alternatywną metodę analizy poziomu wyuczenia maszyn latających, wyspecjalizowanych do śledzenia celu.



Rysunek 10.23: Funkcje odległości maszyny od celu w czasie (iteracje z krokiem numerycznym $\Delta t = 5ms$) oraz ich transformaty Fouriera, dla najlepszych osobników 3 generacji: 100, 150 i 200.

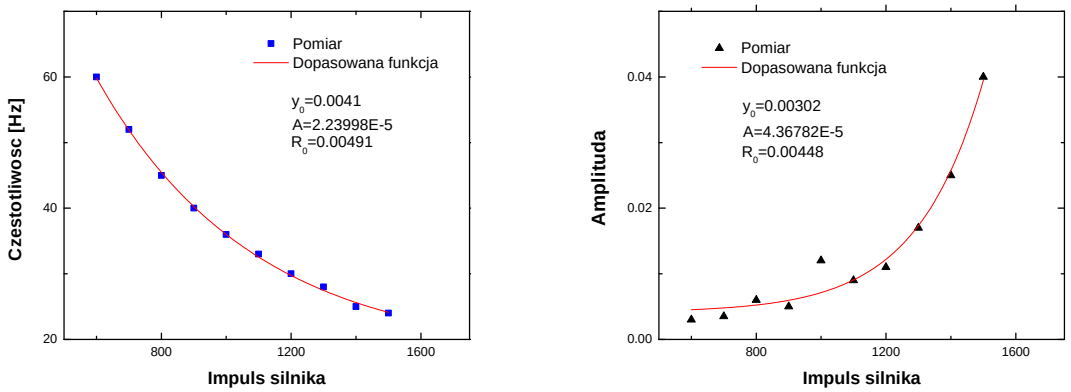
W trakcie analizy transformat funkcji odległości zauważono, iż dla najlepiej wyuczonego szczepu, poza pikiem w zerze istnieje również pojedynczy niewielki skok amplitudy rzędu 0.01, dla częstości ok $60Hz$. Zgadzałoby się to z faktem iż dla najlepszych osobników ostatnich generacji maszyn $'drone'$ tropiących cel, poza widoczną oscylacją agenta wokół celu o względnie dużej amplitudzie (zauważalnej gołym okiem w animacji ruchu), można zauważyć iż maszyna będąca blisko celu lub pokrywająca się z nim potrafi niemal w bezruchu „zawisnąć" w powietrzu, wykonując oscylacje o bardzo małej amplitudzie.
W celu sprawdzenia czy pik ten nie jest tylko wynikiem błędu numerycznego pojedynczego szczepu, porównano transformaty funkcji odległości do celu dla najlepszego otrzymanego szczepu (B) oraz szczepu drugiego w kolejności (C). Mając na uwadze fakt iż ewolucja obu szczepów przebiegała całkiem niezależnie od siebie, stwierdzono iż nie tylko dla obu szczepów występuje taki właśnie pojedynczy pik w wysokich częstościach, ale występuje on również dla tej samej częstości $60Hz$ (rysunki 10.24 (a) i (b)).
Może to oznaczać istnienie pewnego rodzaju determinizmu w ewolucji badanego typu maszyny: osobniki różnych szczepów, ewoluujących na odseparowanych od siebie „wyspach", nie komunikując się ze sobą i nie wymieniając miedzy sobą informacji genetycznej, dochodzą niezależnie od siebie do tego samego numerycznie rodzaju zachowania.



Rysunek 10.24: Pik wysokich częstości transformaty Fouriera funkcji odległości od nieruchomego celu dla najlepiej wyuczonych osobników maszyn typu $'drone'$, dla szczepów o zwykłej mocy silników (szczepy B i C) oraz zwiekszonej mocy silników (szczepy E i F).

Kontynuując analizę piku FFT w wyższych częstotliwościach przeprowadzono kolejną serię ewolucji maszyny typu $'drone'$, wyposażonej w większą mocą silników ($1M\frac{N}{s}$) niż badana dotychczas ($600K\frac{N}{s}$). Dwa najlepsze z 6 wyewoluowanych szczepów po 250 generacjach również posiadały pik w wysokich częstościach, oba dla częstości $36Hz$ (zob. rysunek 10.24 (c) i (d)). To może dowodzić istnienia determinizmu w modelu ewolucji tego typu maszyny, jak również wskazywać na istnienie zależności między mocą silników i wartością docelowej częstości oscylacji o niskiej amplitudzie, do której będą dążyć ewoluujące osobniki wszystkich szczepów, niezależnie od siebie.
W celu dokładniejszego zbadania tej zależności przeprowadzono serię 9 ewolucji układów

typu $'drone'$ dla różnych mocy silnika, zaczynając od silnika o impulsie $600K$ a kończąc na silniku o impulsie $1500K$. Dla każdej konfiguracji (mocy silnika) ewolucję kontynuowano tak długo aż wyłonił się szczep o wartości funkcji przystosowania powyżej 2000, którego funkcję odległości do celu poddano następnie analizie FFT. Wyniki zależności pozycji piku FFT i wielkości (amplitudy) piku przedstawiono na rysunku 10.25. Jak widać obie wielkości można opisać zależnością wykładniczą w funkcji mocy silnika.
Fakt malenia badanych częstości oscylacji o niskich amplitudach w miarę wzrostu mocy silników może być związany ze spadkiem precyzji kontroli nad maszyną w miarę wzrostu impulsu silnika, co w oczywisty sposób wiąże się również z jednoczesnym wzrostem amplitudy oscylacji odległości do celu.

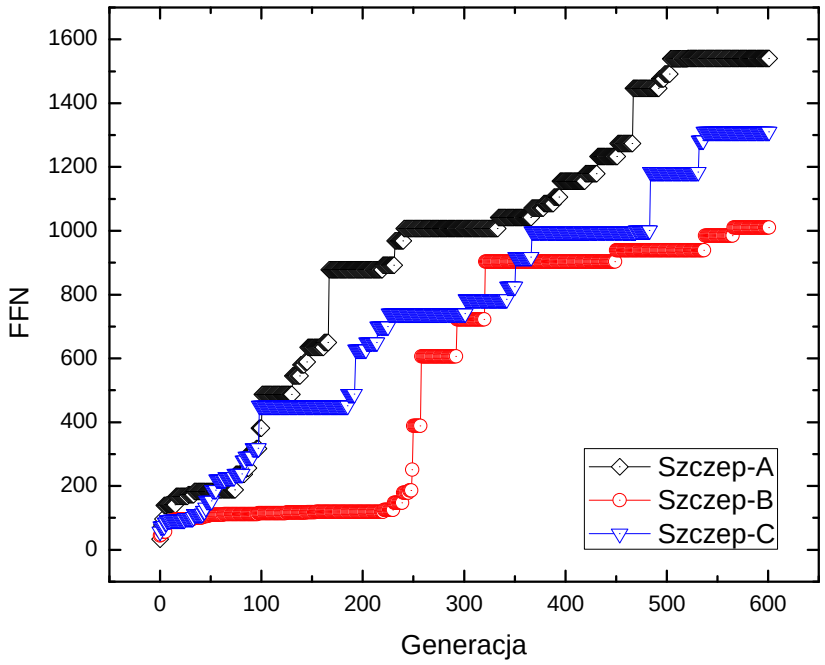


(a) (b)

Rysunek 10.25: Zbadana zależność między mocą silników impulsowych, a pozycją (a) i amplitudą (b) piku FFT w wyższych częstościach, dla funkcji odległości od celu maszyn typu $'drone'$ o impulsach silników od $600K$ do $1500K$. W obu przypadkach dopasowano funkcję eksponecjalną $y = y_0 + e^{R_0 \cdot x}$.

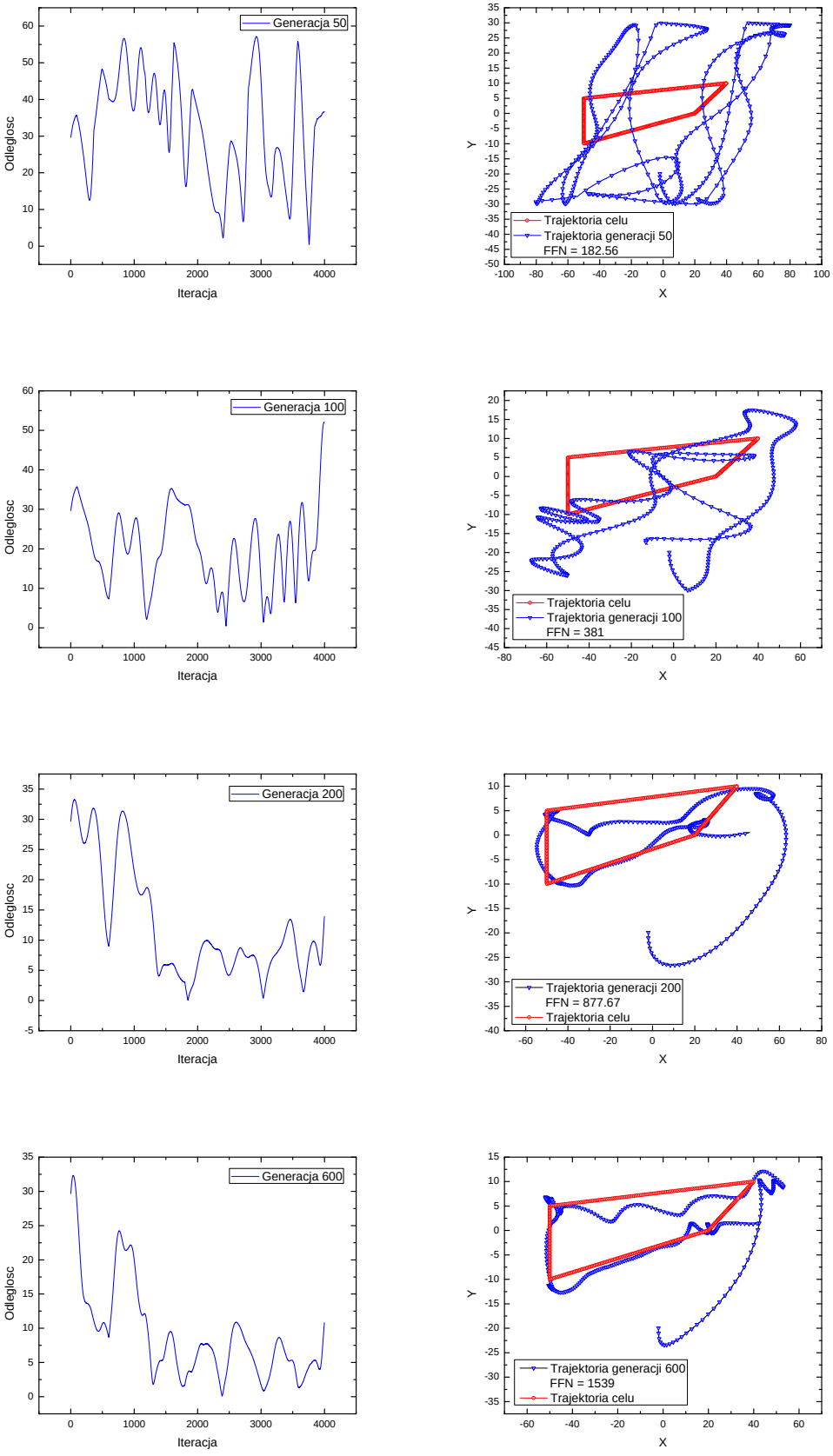
10.3.3 Śledzenie ruchomego celu

Ostatnim krokiem było zbadanie możliwości uczenia maszyn podążania za ruchomym celem. W tym celu umieszczono ruchomy marker na stałej zamkniętej ścieżce zadanej 4 punktami: $(20, 0), (40, -10), (-50, -5), (-50, 10)$ i zadano jego ruch ze stałą prędkością równą 22.
Wynik ewolucji trwającej 600 generacji dla 3 najlepszych szczepów przedstawiono na rysunku 10.26.



Rysunek 10.26: Postęp ewolucji 3 najlepszych szczepów dla układu mechanicznego typu *'drone'* w środowisku promującym podążanie za ruchomym celem.

Porównanie adaptacji trajektorii lotu maszyny do toru celu, w miarę uczenia, dla najlepszego szczepu (A), zestawione wraz z wykresem zależności odległości od celu w czasie (iteracje) przedstawiono na rysunku 10.27.



Rysunek 10.27: Trajektorie czterech kolejnych generacji najlepszego szczepu (A) maszyn latających typu $'drone'$ ewoluujących w środowisku promującym podążanie za ruchomym celem.

Jak widać zaprojektowany układ mechaniczny typu $'drone'$ zdobył umiejętność efektywnego śledzenia ruchomego celu już dla dwusetnej generacji. Jest to o tyle ciekawe, że oba procesy: nauki unoszenia się w powietrzu i śledzenia celu przebiegały równolegle i stawia pytanie na ile pierwsza z umiejętności jest wymogiem rozpoczęcia nauki drugiej, oraz czy można wyróżnić miedzy tymi fazami wyraźny punkt przejścia. Dokładniejsza analiza tego typu zachowania wymagać będzie jednak bardziej szczegółowego podejścia, ze względu na dużą ilość parametrów takiego systemu (kształt trajektorii celu, moc silników, rodzaj przyspieszenia celu), co czyni ten przykład dobrym kandydatem do dalszych badań.

Podsumowanie

Rozdział 11

Wnioski

Podsumowując, w pracy opisano:

- Zasadę działania i implementację silnika symulacji fizycznej 2D, którego poprawność działania przetestowano analizując poziom dyssypacji energii.
- Architekturę i proces implementacji systemu służącego do symulacji robotyki ewolucyjnej w świecie 2D.
- Analizę wyników ewolucji przykładowych autonomicznych układów mechanicznych, które nauczono efektywnego: poruszania się w środowisku z przeszkodami, pełzania, bezkolizyjnego latania oraz podążania za ruchomym celem.

Pozytywnym rezultatem pracy było znalezienie i usunięcie błędu dyssypacji w najpopularniejszym otwartym silniku symulacji opartym na dynamice bryły sztywnej - Box2D - na którego budowie i zastosowanych mechanizmach numerycznych wzorowano się przy tworzeniu silnika symulacji zastosowanego w tej pracy. Może to być również sygnał iż silniki symulacji fizycznej nie zawsze dokładnie zachowują energię i mogą wymagać wcześniejszych testów przed zastosowaniem do realizacji wyznaczonego zadania.

Jedną z kluczowych informacji na jaką udało się natrafić podczas analizy wyników ewolucji przykładowych robotów, jest fakt iż otrzymane w procesie optymalizacji układy mechaniczne sugerują istnienie pewnego rodzaju determinizmu w procesie ewolucji. Może na to wskazywać kierunek ewolucji otrzymanych organizmów segmentowych i dążenie do specyficznych zachowań w zależności od liczby segmentów tworzących szkielet tych maszyn - maszyny o krótszych kręgosłupach dążyły zawsze do ruchu pełzającego, natomiast te o dłuższych kręgosłupach - do dużo efektywniejszego ruchu zamachowego z wykorzystaniem bezwładności. Dowodem tego może być również istnienie docelowej dla procesu ewolucji częstości mikro-oscylacji pozycji maszyn latających, o wartościach częstości i amplitudy zdeterminowanych przez przez moc silników.

Należy również zauważyć iż efektywne prowadzenie badań w dziedzinie robotyki ewolucyjnej może wymagać rozbudowanego środowiska do symulacji równoległych, którego przykładem jest zaimplementowany w tej pracy serwer ewolucji. Tego typu badania mogą być dobrym celem wykorzystania zarówno rozproszonych obliczeń wieloprocesorowych

(np. do jednoczesnej symulacji ewolucji wielu szczepów) jak i obliczeń wielowątkowych wspomaganych układami GPU (np. do wielowątkowych obliczeń w ramach pojedynczego świata symulacji o stosunkowo dużej ilości obiektów i skomplikowanej budowie).

Jak przekonano się podczas nauki maszyn przeznaczonych do bezkolizyjnego lotu, zastosowanie sieci neuronowych może być konieczne gdy ręczny dobór parametrów granicznych sterujących balansowaniem czy inną formą precyzyjnego ruchu wydaje się mało dokładny.

Analizując wyniki ewolucji robota $'walker'$ można wyciągnąć ciekawy wniosek iż to co dla nas wydaje się oznaczać przeszkodę, w istocie wcale nie musi przeszkadzać w realizacji wyznaczonego celu, a nawet może zostać przez proces ewolucji wykorzystane jako element służący do osiągnięcia wyniku lepszego niż u osobników żyjących w środowisku pozbawionym tej przeszkody. Identycznie sparametryzowane maszyny kroczące (posiadające identyczny genotyp) nauczyły się poruszać z większą prędkością niż po idealnie płaskim podłożu, używając do tego przeszkód w postaci przytwierdzonych do podłoża progów.

Ciekawym obserwacją podczas analizy ewolucji układów latających typu $'drone'$ jest fakt iż udało połączyć się w jednym procesie ewolucyjnym z pozoru dwustopniowy proces nauki, w którym najpierw powinno uczyć się umiejętności prymitywnych (takich jak umiejętność latania), a dopiero potem z ich użyciem umiejętności wyższego poziomu (czyli w tym przypadku śledzenia ruchomego celu w powietrzu). W przeprowadzonej ewolucji robota $'drone'$ dla którego zdefiniowano cel wyższego poziomu, stopniowa ewolucja wartości wag sieci neuronowej sterującej zachowaniem robota umożliwiła jednoczesną naukę zarówno unoszenia w powietrzu jak i tropienia celu.

Wyniki przeprowadzonych w trakcie badań nad powyższą pracą symulacji ewolucji w postaci filmów obrazujących zachowanie najciekawszych przedstawicieli poszczególnych wyewoluowanych szczepów podczas realizacji wyznaczonych im celów zebrano i umieszczono na płycie CD dołączonej do pracy oraz w internecie pod adresem [43].

Rozdział 12

Kierunki dalszego rozwoju

Robotyka ewolucyjna jest prężnie rozwijającą się dziedziną nauki, a urywek tego co zostało przedstawione w powyższej pracy może być zaczątkiem wielu badań, których autor pracy chciałby podjąć się w przyszłości.
Choć zbadano i przedstawiono wyniki analizy wszystkich typów założonych na początku robotów, stworzony system pozwala na dużo więcej. Przykłady bardziej złożonych układów mechanicznych: platformowych układów latających czy maszyn kroczących napędzanych siłownikami, których analizy nie podjęto w tej pracy, a które wybrano jako punkt wyjścia do dalszych badań, zostały umieszczone w postaci filmów pod wcześniej wspomnianym adresem [43].

Ciekawym przykładem morfologicznym do dalszej analizy byłoby zasymulowanie maszyn wieloodnóżowych, imitujących zachowaniem pajęczaki i optymalizacja pracy i synchronizacji odnóży napędzanych siłownikami połączonymi z siecią neuronową, stawiając na celu naukę poruszania się po terenie z nieruchomymi przeszkodami i ruchomym gruzowisku.

Choć zaimplementowany serwer symulacji robotyki ewolucyjnej umożliwia symulacje układów wieloagentowych, ze względu na obszerność tematu nie podjęto go w tej pracy. Jest to jednak tak naprawdę docelowy kierunek badań którym autor tej pracy się kierował podczas prowadzenia badań i jeden z głównych kierunków planowanego dalszego rozwoju. Wydaje się że symulacja systemów wieloagentowych może wymagać podejścia krokowego przy zastosowaniu algorytmów ewolucyjnych - sekwencyjnej uczenia maszyn realizacji zachowań coraz wyższego poziomu - począwszy od nauki efektywnego poruszania się a skończywszy na ewolucji zachowań takich jak komunikacja i koordynacja wspólnych zadań czy konkurencja i walka między osobnikami o wyszczególniony zasób. Nawiązując do wyników powyższej pracy można jednak spekulować czy optymalizacji opartej na ewolucji nie można efektywnie przeprowadzać jednocześnie dla wyuczenia wysoko- i nisko poziomowych docelowych zachowań, tak jak udało się to osiągnąć w przypadku badanych w pracy układów latających, które w tym samym czasie poza unoszeniem się w powietrzu i balansowaniem przy pomocy regulacji częstości pracy silników impulsowych nauczyły się również efektywnego tropienia ruchomego celu.

Badania ewolucyjnych systemów wieloagentowych można rozdzielić na dwa zasadni-

cze kierunki: kooperację(współpracę) i konkurencję.

Interesującymi przypadkami do dalszych badań wieloagentowych systemów współpracujących będzie z pewnością nauka synchronicznego ruchu układów wielosegmentowych naśladujących ławicę ryb, lub optymalizacja układów latających mających na celu imitację synchronicznego ruchu stada ptaków.

Dużo ciekawszym kierunkiem badań wydaje się jednak ewolucja układów wieloagentowych stawiających na celu konkurencję. Przykładem kierunku badań w tym temacie może być symulacja robotów konkurujących o zdobycie zasobu znajdującego się w środowisku, korzystając np. z modelu tzw. *pojazdow Breiteberga* [17], idąc jednak o krok dalej i tworząc agenta o bardziej złożonej budowie mechanicznej, zróżnicowanych układach napędowych i umieszczonego w bardziej realistycznym środowisku fizycznym uwzględniającym grawitację, siły tarcia i bardziej zróżnicowane sposoby ruchu pojazdów mechanicznych, niż w przypadku klasycznych pojazdów Breitenberga.

Kierunkiem badań konkurencyjnych układów mechanicznych który nasunął się jeszcze przed rozpoczęciem pisania pracy było stworzenie środowiska symulacji wirtualnego pola walki, w którym autonomiczne układy mechaniczne konkurowałyby ze sobą na dwóch osobnych płaszczyznach:
- płaszczyźnie mechaniczno-mobilnej, wykorzystując nabyte umiejętności efektywnego poruszania się, wytrzymałość wytworzonego w trakcie ewolucji szkieletu i siłę rażenia,
- oraz na płaszczyźnie cyfrowej, będącej symulacją walki elektronicznej, z wykorzystaniem nabytych umiejętności zakłócania komunikacji oraz wrogiej ingerencji w kod sterujący konkurencyjnych maszyn (agentów).
Taki system mógłby znaleźć zastosowanie militarne do symulacji współczesnego pola walki autonomicznych maszyn bojowych nie sterowanych bezpośrednio przez człowieka.

Kolejnym kierunkiem rozwoju wartym poświęcenia uwagi może być badanie efektywności maszyn pod kątem zużycia energii i szukania układów mechanicznych o parametrach optymalnych zarówno pod kątem realizacji celu podstawowego - szybkiego przemieszczania się, bezkolizyjnego lotu czy tropienia ruchomego celu - jak również pod kątem ograniczonej, przeznaczonej na realizację tego celu energii.

Oczywistym i koniecznym krokiem w prowadzeniu dalszych badań w tym kierunku będzie zmiana silnika symulacji 2D na 3D, co można zrealizować efektywnie wykorzystując istniejącą już warstwę abstrakcji między modułem robotyki i silnikiem symulacji oraz stosując jeden wielu obecnie dostępnych silników symulacji o otwartym kodzie.

Bardzo przydatne byłoby również stworzenie interaktywnej bazy danych otrzymywanych sukcesywnie genotypów i fenotypów maszyn, z możliwością łatwego dostępu i podglądu przez użytkownika. Takiej bazy można by użyć dalej jako zaplecza przy sekwencyjnym nauczaniu organizmów coraz to nowych, wyżejpoziomowych zachować (np. synchronicznego ruchu w grupie lub śledzenia celu), korzystając z wyuczonych wcześniej modułów sieci neuronowych lub otrzymanych optymalnych konfiguracji układów mechanicznych, zdolnych do realizacji zachowań bardziej prymitywnych, takich jak np. efektywne poruszanie się w wybranym terenie.

Jednym z najbardziej fascynujących i inspirujących kierunków dalszych badań jest ewolucja morfologii maszyny czyli mutacja genów odpowiedzialnych nie tylko za para-

metry elementów konstrukcyjnych maszyny ale i za strukturę jej szkieletu.

Kontynuując prace Karla Simsa sprzed 20 lat temu, z wykorzystaniem dzisiejszej mocy obliczeniowej centrów superkomputerowych można by w ten sposób pozwolić na ewolucję całkiem nowych kształtów maszyn i zacząć poszukiwania nowych metod ruchu i zachowań sztucznych autonomicznych organizmów, odmiennych od tych osiągniętych w trakcie trwania 5 miliardów lat ewolucji świata jaki znamy.

Bibliografia

[1] Januszajtis A., "Fizyka dla politechnik", PWN, tom 1, 1977,

[2] Awrejcewicz J., "Mechanika techniczna i teoretyczna: statyka i kinematyka", Politechnika Łódzka, 1952,

[3] Awrejcewicz J., "Mechanika techniczna i teoretyczna: dynamika", Politechnika Łódzka, 1952,

[4] Nejmark J.I., Fufajew N.A., "Dynamika układów nieholonomicznych", PWN,

[5] Millington I., "Game Physics Engine Development", Morgan Kaufman, 2007,

[6] Shabana A. A., "Computational Dynamics", John Wiley & Sons, 2001,

[7] Sims K., http://www.karlsims.com/evolved-virtual-creatures.html (18.08.2015),

[8] Sims K., "Evolving Virtual Creatures", Siggraph '94 Proceedings, (July 1994),

[9] Sims K., "Evolving 3D Morphology and Behavior by Competition", Artificial Life IV Proceedings, ed.by Brooks & Maes, MIT Press, 1994.

[10] NASA, "Evolutionary Antenna", http://www.nasa.gov/centers/ames/news/releases/2004/antenna/antenna.html (18.08.2015),

[11] Miikkulainen R., Lehman J., "Computer scientists find mass extinctions can accelerate evolution", http://phys.org/news/2015-08-scientists-mass-extinctions-evolution.html, (18.08.2015),

[12] Cottle, Richard W., Pang, Jong-Shi, Stone, Richard E, "The linear complementarity problem", Computer Science and Scientific Computing, Boston, 1992,

[13] Golub, Gene H., Van Loan, Charles F, "Matrix Computations", 3rd edition, 1996,

[14] Catto E., http://box2d.org/, (03.09.2015),

[15] Catto E., "Modeling and solving constraints", (GDC 2009),

[16] Catto E., "Iterative dynamics with temporal coherence", (GDC 2005),

[17] Braitenberg V., "Vehicles: Experiments in synthetic psychology", Cambridge, MA: MIT Press, 1984.

[18] Nolfi S., Floreano D., "Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines", MIT Press, 2000,

[19] Stevens W. R., "Programowanie w środowisku systemu UNIX", WNT, 2002,

[20] Ben-Air M., "Podstawy programowania współbieżnego i rozproszonego", WNT,

[21] Helihy N., Shavit N., "Sztuka programowania wieloprocesorowego", PWN,

[22] Salomon D., "Assemblers and Loaders", 1993,

[23] Korn J., "Ewolucja Kodu", Chip, (Marzec 2015),

[24] "Autonomic nervous system", https://en.wikipedia.org/wiki/Autonomic_nervous_system, (29.08.2015),

[25] Baumgarte J., "Stabilization of constraints and integrals of motion in dynamical systems", Computer Methods in Applied Mechanics and Engineering, Volume 1, Issue 1, Pages 1–16, 1972,

[26] Ascher Uri M., Petzold Linda R., "Computer Methods for Ordinary Differential equations and Differential-Algebraic equations", Philadelphia: SIAM, (1998),

[27] Eichner H., "Equality Constraints", http://myselph.de/gamePhysics/equalityConstraints.html, (30.08.2015),

[28] Michalewicz Z., "Algorytmy genetyczne + struktury danych = programy ewolucyjne", 2003,

[29] Giordano, Nicholas J., Hisao Nakanishi, "Computational Physics", 2nd edition, July 2005,

[30] Bittle W., "SAT: Separating Axis Theorem", http://www.dyn4j.org/2010/01/sat/, (03.09.2015)

[31] Howling Moon Software, https://chipmunk-physics.net/, (03.09.2015)

[32] Tadeusiewicz R., "Sieci neuronowe", Akademicka Oficyna Wydawnicza RM, 1993,

[33] Kosinski R., "Sztuczne sieci neuronowe", WNT

[34] Mitchell, Tom M. "Machine Learning", Chapter 4: "Artificial Neural Networks", pp. 96–97, WCB–McGraw–Hill, 1997,

[35] Cheney, N. MacCurdy, R., Clune, J., Lipson, H. "Unshackling Evolution: Evolving Soft Robots with Multiple Materials and a Powerful Generative Encoding", (2013),

[36] Cheney, N. MacCurdy, R., Clune, J., Lipson, H., "Unshackling Evolution", http://creativemachines.cornell.edu/soft-robots, (06.09.2015),

[37] NVIDIA, "NVIDIA PhysX SDK", https://developer.nvidia.com/sites/default/files/akamai/physx/Index.html, (08.09.2015),

[38] Real-Time Physics Simulation, "Bullet Physics", http://bulletphysics.org, (08.09.2015),

[39] Couman E., "Exploring MLCP solvers and Featherstone", (2014),

[40] Lu Y., http://www.cs.rpi.edu/~trink/sim_packages.html, (08.09.2015),

[41] Erez T., Tassa Y., Todorov E., "Simulation Tools for Model-Based Robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX", (2015),

[42] Sporysz J., "About energy conservation", http://box2d.org/forum/viewtopic.php?f=4&t=9830, (04.07.2014),

[43] Sporysz J., http://fizyk.ifpk.pk.edu.pl/~rkycia/master/JSAdvanced/EvolRob.zip (18.08.2015).