POLITECHNIKA GDAŃSKA
WYDZIAŁ ELEKTRONIKI, TELEKOMUNIKACJI I INFORMATYKI

Imię i nazwisko studenta: Norman Dryś
Nr albumu: 171887
Poziom kształcenia: Studia pierwszego stopnia
Forma studiów: stacjonarne
Kierunek studiów: Automatyka i Robotyka
Profil: Systemy automatyki

Imię i nazwisko studenta: Paweł Zieliński
Nr albumu: 172247
Poziom kształcenia: Studia pierwszego stopnia
Forma studiów: stacjonarne
Kierunek studiów: Automatyka i Robotyka
Profil: Systemy automatyki

PROJEKT DYPLOMOWY INŻYNIERSKI

Tytuł projektu w języku polskim: Sterowanie odwróconym wahadłem Furuta za pomocą sieci neuronowej.

Tytuł projektu w języku angielskim: Control of the Furut inverted pendulum using a neural network.

Opiekun pracy: dr inż. Piotr Fiertek

Data ostatecznego zatwierdzenia raportu podobieństw w JSA: 25.01.2021

POLITECHNIKA GDAŃSKA
WYDZIAŁ ELEKTRONIKI, TELEKOMUNIKACJI I INFORMATYKI

OŚWIADCZENIE dotyczące pracy dyplomowej zatytułowanej: Sterowanie odwróconym wahadłem Furuta za pomocą sieci neuronowej.

Imię i nazwisko studenta: Norman Dryś
Data i miejsce urodzenia: 29.03.1998, Tarnów
Nr albumu: 171887

Wydział: Wydział Elektroniki, Telekomunikacji i Informatyki
Kierunek: automatyka i robotyka

Poziom kształcenia: pierwszy
Forma studiów: stacjonarne

Typ pracy: projekt dyplomowy inżynierski

Świadomy(a) odpowiedzialności karnej z tytułu naruszenia przepisów ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz. U. z 2019 r. poz. 1231, z późn. zm.) i konsekwencji dyscyplinarnych określonych w ustawie z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (t.j. Dz. U. z 2020 r. poz. 85, z późn. zm.),[1] a także odpowiedzialności cywilnoprawnej oświadczam, że przedkładana praca dyplomowa została opracowana przeze mnie samodzielnie.

Niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadaniem tytułu zawodowego.

Wszystkie informacje umieszczone w ww. pracy dyplomowej, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami zgodnie z art. 34 ustawy o prawie autorskim i prawach pokrewnych.

25.01.2021, Norman Dryś

Data i podpis lub uwierzytelnienie w portalu uczelnianym Moja PG

**) Dokument został sporządzony w systemie teleinformatycznym, na podstawie §15 ust. 3b Rozporządzenia MNiSW z dnia 12 maja 2020 r. zmieniającego rozporządzenie w sprawie studiów (Dz.U. z 2020 r. poz. 853). Nie wymaga podpisu ani stempla.*

[1] Ustawa z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce:

Art. 312. ust. 3. W przypadku podejrzenia popełnienia przez studenta czynu, o którym mowa w art. 287 ust. 2 pkt 1–5, rektor niezwłocznie poleca przeprowadzenie postępowania wyjaśniającego.

Art. 312. ust. 4. Jeżeli w wyniku postępowania wyjaśniającego zebrany materiał potwierdza popełnienie czynu, o którym mowa w ust. 5, rektor wstrzymuje postępowanie o nadanie tytułu zawodowego do czasu wydania orzeczenia przez komisję dyscyplinarną oraz składa zawiadomienie o podejrzeniu popełnienia przestępstwa.

STRESZCZENIE

Systemy automatyki z biegiem czasu stają się coraz bardziej złożone, przez co trudniej jest nimi sterować. Obecnie stosowane metody z użyciem teorii sterowania wymagają od automatyka obszernej wiedzy oraz doświadczenia. Inżynierowie od lat starają się uprościć ten proces wyręczając się maszynami. Przykładem złożonego obiektu, którego kontrolowanie do dziś przysparza wielu problemów jest odwrócone wahadło. Dodatkowe utrudnienia wynikające z ruchu obrotowego posiada wahało będące obiektem zainteresowania tej pracy, zwane wahadłem Furuta. Celem projektu było opracowanie sterownika umożliwiającego stabilizację wahadła w pozycji odwróconej w pionie. Obszerny przegląd algorytmów z kategorii uczenia ze wzmocnieniem umożliwił wybranie metody pasującej do tej pracy. Praca od podstaw prezentuje zasady algorytmu DQN jak i wszystkie związane z nim zagadnienia. Algorytm zaimplementowano za pomocą oprogramowania Matlab. W celu sprawdzenia poprawności działania programu stworzono model symulacyjny wiernie odwzorowujący obiekt rzeczywisty oraz trójwymiarową wizualizację jego działania. Eksperymentalnie dobrane parametry uczenia oraz rozmiar głębokiej sieci neuronowej sprawiły, że regulator był w stanie skutecznie balansować wahadłem. Przebadane zostały zdolności adaptacyjne zaprogramowanego sterownika oraz jego odporność na zakłócenia. Zbudowane zostało rzeczywiste wahadło, obsługiwane przez zewnętrzny mikrokomputer. Ze względu na zbyt duże opóźnienia związane z pracą sterownika silnika, zadowalający efekt sterowania rzeczywistym obiektem nie został osiągnięty. Sterownik opanował problem rozbujania wahadła, jednakże poprawne balansowanie w pozycji pionowej (odwróconej) wymagałoby wprowadzenia poprawek technicznych. Komputerowa symulacja dowiodła osiągnięcie zakładanego celu projektu, czyli stworzenie poprawnie działającego, w pełni adaptacyjnego sterownika.

Słowa kluczowe: odwrócone wahadło Furuta, uczenie ze wzmocnieniem, DQN

Dziedzina nauki i techniki, zgodnie z wymogami OECD: Nauki inżynieryjne i techniczne**,** inżynieria informatyczna.

ABSTRACT

Automation systems are becoming more complex over time, making them more difficult to control. Current methods using control theory require extensive knowledge and experience on the part of the automation specialist. For years, engineers have tried to simplify this process by using machines. An example of a complex object that is still difficult to control today is an inverted pendulum. The pendulum of interest in this thesis, called the Furut pendulum, has the added complication of rotational motion. The goal of the project was to develop a controller, operating on a neural network, to stabilize a pendulum in a vertically inverted position. An extensive review of algorithms in the category of reinforcement learning enabled the selection of a method that fit this work. The paper presents the principles of the DQN algorithm, as well as all related issues. The algorithm was implemented using Matlab software. In order to check the correctness of the program, a simulation model faithfully reproducing the real object and a three-dimensional visualization of its operation were created. Experimentally selected learning parameters and the size of the deep neural network meant that the controller was able to balance the pendulum effectively. The adaptability of the programmed controller and its robustness to disturbances were tested. A real pendulum was built and operated by an external microcomputer. Due to excessive delays associated with the motor controller, a satisfactory control effect of the real object was not achieved. The controller mastered the swing-up problem however, correct vertical (inverted) balancing would require technical corrections. Computer simulation proved that the project's goal of creating a properly functioning, fully adaptive controller was achieved.

Keywords: Furut inverted pendulum, reinforcement learning, DQN

Field of science and technology, in accordance with the requirements of the OECD: Engineering and technology, Information engineering.

SPIS TREŚCI

Spis treści 5

Wykaz ważniejszych oznaczeń i skrótów 7

1. Wstęp i cel pracy (Paweł Zieliński, Norman Dryś) 8

2. Dynamika obrotowego, odwróconego wahadła (Norman Dryś) 9

2.1. Sformułowanie Lagranżjanów za pomocą tensorów 10

2.2. Równania dynamiki obiektu 11

3. Sterowanie za pomocą sztucznej inteligencji (Paweł Zieliński) 12

3.1. Głębokie sieci neuronowe 12

3.2. Uczenie ze wzmocnieniem 15

3.2.1. Proces decyzyjny Markova 15

3.2.2. Funkcja nagrody 15

3.2.3. Wybór algorytmu RL 16

3.2.4. Algorytm głębokiego uczenia funkcji Q 17

4. Implementacja algorytmu DQN i symulacji (Paweł Zieliński) 20

4.1. Simulink 20

4.1.1. Model matematyczny 21

4.1.2. Wizualizacja działania symulacji 23

4.2. Program Sterujący 23

4.2.1. Pamięć doświadczeń 24

4.2.2. Implementacja głębokiej sieci neuronowej 24

4.2.3. Nagroda 25

4.2.4. Aktualizacja funkcji celu 25

4.2.5. Aktualizacja parametrów sieci 26

4.3. Symulacja obiektu 26

4.4. Dobranie rozmiaru sieci 28

4.5. Symulacja z uwzględnieniem opóźnień 34

4.6. Implementacja priorytetowej pamięci doświadczeń 35

4.7. Adaptacyjność, odporność sterownika i wpływ zakłóceń ... 36
5. Budowa wahadła (Norman Dryś) ... 37
5.1. Konstrukcja mechaniczna ... 39
5.1.1. Stabilność konstrukcji ... 39
5.1.2. Minimalizacja zakłóceń i dodatkowych nieliniowości ... 39
5.2. Zastosowana elektronika i silnik ... 40
5.2.1. Silnik ... 40
5.2.2. Mikrokontroler / mikrokomputer ... 41
5.2.3. Sterownik silnika ... 41
5.2.4. Enkoder ... 42
5.2.5. Zasilanie ... 42
6. Sterowanie rzeczywistym obiektem (Norman Dryś) ... 42
6.1. Układ sterowania ... 43
6.2. Redukcja opóźnienia komunikacyjnego ... 44
6.2.1. Komunikacja USB ... 44
6.2.2. Komunikacja Wi-Fi ... 44
6.2.3. Bezpośredni wybór akcji przez sterownik ... 45
6.3. Potencjalne przyczyny niepowodzeń ... 46
7. Podsumowanie (Paweł Zieliński, Norman Dryś) ... 48
Wykaz literatury ... 50
Wykaz rysunków ... 51
Wykaz tabel ... 52
Załącznik nr 1: Materiały prezentujące działanie ... 53

WYKAZ WAŻNIEJSZYCH OZNACZEŃ I SKRÓTÓW

AI – Artificial Inteligence – sztuczna inteligencja

NN – Neural Network – sieć neuronowa

DNN – Deep Neural Network – głęboka sieć neuronowa

RL – Reinforcement Learning – uczenie ze wzmocnieniem

MDP – Markov Decision Process – proces decyzyjny Markova

DQN – Deep Q Network – głęboka sieć Q

DQL – Deep Q Learning – głębokie uczenie Q

ER – Experience Replay – pamięć doświadczeń

PER – Prioritized Experience Replay – priorytetowa pamięć doświadczeń

PC – Personal Computer – computer osobisty

USB – Universal Serial Bus – uniwersalny port szeregowy

3-D – 3 dimensions – trójwymiar

EEPROM – electrically erasable programmable read-only memory – elektronicznie kasowalna, programowalna pamięć tylko do odczytu

DC – Direct Current – prąd stały

BLDC – brushless DC electric motor – bezszczotkowy silnik elektryczny prądu stałego

BEMF – back electromotive force – wsteczna siła elektromotoryczna

MOSFET – Metal-Oxide Semiconductor Field-Effect Transistor – tranzystor polowy z izolowaną bramką typu MOS

IPD – Initial Position Detect – detekcja pozycji początkowej

I2C – Inter-Integrated Circuit – szeregowy interfejs przesyłu danych

SMT – surface-mount technology – montaż powierzchniowy

1. WSTĘP I CEL PRACY (PAWEŁ ZIELIŃSKI, NORMAN DRYŚ)

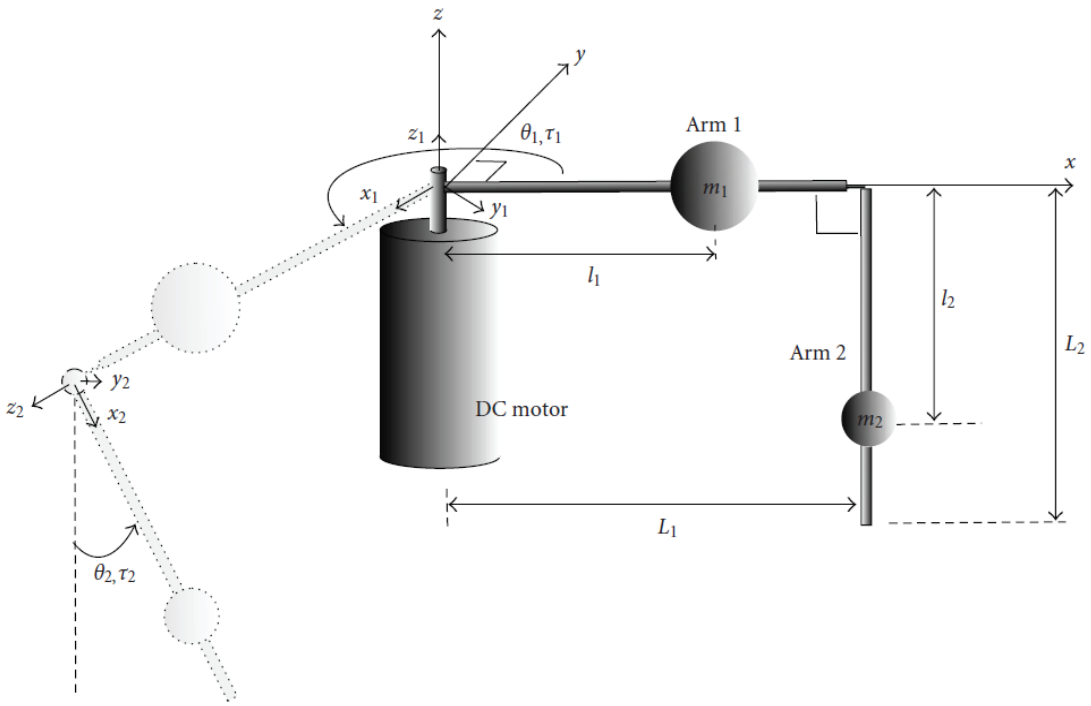
Od układów regulacji często wymaga się sterowania wymagającymi obiektami, charakteryzującymi się bardzo dużą nieliniowością. Coraz częściej zdarza się, że poznane metody sterowania albo zawodzą, albo stopień złożoności ich implementacji wykracza poza obecne możliwości technologiczne. Wciąż najlepszym narzędziem poznawczym, najsprawniej radzącym sobie w problemach uczących, jest ludzki mózg. Z tego powodu naukowcy coraz częściej opierają działanie nowych algorytmów na modelach odwzorowujących procesy zachodzące wewnątrz ludzkiego mózgu. Jedną z najbardziej rozwijanych dziedzin nauk technicznych ostatnich lat jest uczenie maszynowe.

Odwrócone wahadło od lat służy jako przykład silnie nieliniowego obiektu, omawianego podczas nauki teorii sterowania. Problem polega na rozbujaniu, a następnie balansowaniu wahadłem w taki sposób aby utrzymywał on pozycję odwróconą w pionie. Od lat 90-tych, stosując opracowane metody oparte o teorię sterowania, z powodzeniem rozwiązywany jest problem obrotowego, odwróconego wahadła. Głównymi operacjami klasycznego podejścia są: matematyczna identyfikacja oparta na modelu stanowym oraz zaprojektowanie sterowania od stanu.

Głównym celem pracy jest dobór oraz implementacja algorytmu uczenia maszynowego. Sterowanie miało być przetestowane w środowisku symulacyjnym. Jest to bardzo czasochłonne zadanie ze względu na dużą liczbę hiperparametrów sieci wymagających doboru. Jednocześnie należało zbudować fizyczne wahadło, za pomocą którego zweryfikowana zostałaby poprawność działania algorytmu w praktyce. W tej części należy wpierw dobrać wszystkie elementy wchodzące w skład konstrukcji a następnie mechanicznie je połączyć. W skład wahadła wchodzą również elementy elektroniczne wymagające odpowiedniej konfiguracji. Norman Dryś w głównej mierze odpowiadał za stworzenie rzeczywistego obiektu odwróconego wahadła Furuta. Paweł Zieliński w większym stopniu skupiał się na tworzeniu oprogramowania sterującego.

2. DYNAMIKA OBROTOWEGO, ODWRÓCONEGO WAHADŁA (NORMAN DRYŚ)

Wahadło to obiekt o bardzo dużej nieliniowości, spowodowanej istnieniem sił nieustannie utrudniających mu osiągnięcie zakładanego celu. Przede wszystkim oddziałuje na niego siła grawitacji. Dodatkowa praca musi zostać wykonana w celu przeciwdziałania siłom Coriolisa oraz siły dośrodkowej. Praca naukowa [10] została użyta jako źródło zrozumienia zjawisk zachodzących podczas pracy obiektu jakim jest odwrócone (obrotowe) wahadło.



Rys. 2.1 Schemat ruchu obrotowego, odwróconego wahadła [10]

Wahadło składa się z trzech połączonych ze sobą elementów fizycznych. Pierwszym jest silnik prądu stałego wprawiający cały układ w ruch obrotowy za pomocą momentu siły τ_1. Kolejno zamocowane są dwa ramiona podpisane na rysunku 2.1 jako Arm 1 oraz Arm 2. Wielkości fizyczne o indeksach dolnych równych 1 odnoszą się do ramienia 1. Analogiczne oznaczenia zastosowano w przypadku ramienia 2. Kąty obrotu θ_i wskazują przesunięcie kątowe ramienia wokół własnej osi obrotu. Analiza nie zakłada położenia środka masy ramion m_i w środku ich długości. Każdy punkt środka masy m_i jest oddalony od osi obrotu z_i o odległość równą l_i. Długości obu ramion podpisane zostały literą L_i. Każde z ramion posiada moment bezwładności wyznaczany względem osi obrotu z_i oraz oznaczany odpowiednio jako J_i. Opory ruchu każdego z obu ramion reprezentowane są odpowiednio przez współczynniki b_1 i b_2. Znaki momentów siły ustalone zostały za pomocą zasady prawej dłoni, czyli przyjmowana jest wartość dodatnia, gdy ruch odbywa się przeciwnie do ruchu wskazówek zegara.

W celu uzyskania prostszego modelu matematycznego wahadła kilka zjawisk fizycznych zostało pominiętych:

- ugięcie ramion wahadła,
- moment bezwładności silnika,
- prawo Coulomba jako siła tłumiąca.

2.1. Sformułowanie Lagranżjanów za pomocą tensorów

Opisanie właściwości mechanicznych obiektu za pomocą lagranżjanów składa się z wyznaczenia różnicy między wartościami energii kinetycznej oraz potencjalnej. Użycie tensorów uogólnia układy współrzędnych, co sprawia, że wyprowadzenie staje się bardziej przejrzyste.

Macierze obrotu opisujące przejścia między dwoma kolejnymi elementami, a w konsekwencji inaczej położonymi układami współrzędnych wyglądają następująco:

$$R_1 = \begin{bmatrix} cos\ (\theta_1) & sin\ (\theta_1) & 0 \\ -sin\ (\theta_1) & cos\ (\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix}, R_2 = \begin{bmatrix} 0 & sin\ (\theta_2) & -cos\ (\theta_2) \\ 0 & cos\ (\theta_2) & sin\ (\theta_2) \\ 1 & 0 & 0 \end{bmatrix} \quad (1)$$

gdzie R_1 jest macierzą obrotu ramienia 1, a R_2 jest macierzą obrotu ramienia 2. Prędkość obrotowa ω_1 oraz postępowa v_1 ramienia 1 wchodzi w skład jego prędkości całkowitej v_{1c} w punkcie środka obrotu:

$$\omega_1 = [0 \quad 0 \quad \dot{\theta}_1]^T, \qquad v_1 = [0 \quad 0 \quad 0]^T,$$

$$v_{1c} = v_1 + \omega_1 \times [l_1 \quad 0 \quad 0]^T = [0 \quad \dot{\theta}_1 l_1 \quad 0]^T.$$

W skład prędkości obrotowej ramienia 2 ω_2 wchodzi własna prędkość kątowa ramienia 2 $\dot{\theta}_2$ oraz przemnożona prędkość obrotowa ramienia 1 ω_1 z drugą macierzą obrotu R_2, wyznaczoną w równaniach (1). Prędkość postępowa ramienia 2 v_2 powstaje poprzez przekształcenie prędkości obrotowej ω_1 wzdłuż ramienia 1 oraz przemnożenie wyniku z drugą macierzą obrotu R_2. Obie prędkości ramienia 2 wyznaczane są za pomocą zależności:

$$\omega_2 = [-cos\ (\theta_2)\dot{\theta}_1 \quad sin\ (\theta_2)\dot{\theta}_1 \quad \dot{\theta}_2]^T,$$

$$v_2 = \begin{bmatrix} \dot{\theta}_1 L_1 sin\ (\theta_2) \\ \dot{\theta}_1 L_1 cos\ (\theta_2) \\ 0 \end{bmatrix}.$$

Prędkość całkowita ramienia 2 w punkcie środka obrotu to suma obu prędkości, z czego prędkość obrotowa musi zostać przemnożona przez odległość masy od osi obrotu:

$$v_{2c} = \begin{bmatrix} \dot{\theta}_1 L_1 \sin\ (\theta_2) \\ \dot{\theta}_1 L_1 \cos(\theta_2) + \dot{\theta}_2 l_2 \\ -\dot{\theta}_1 l_2 \sin\ (\theta_2) \end{bmatrix}.$$

Posiadając już wyznaczone prędkości całkowite każdego z ramion, można wyznaczyć energie potencjalne oraz kinetyczne. Energia potencjalna ramienia 1 E_{p1} jest zerowa, zaś energia kinetyczna E_{k1} jest średnią energii kinetycznej ruchu posuwistego oraz obrotowego:

$$E_{p1} = 0,$$

$$E_{k1} = \frac{1}{2}({v_{1c}}^T m_1 v_{1c} + {\omega_1}^T J_1 \omega_1) = \frac{1}{2}\dot{\theta}_1^2(m_1 l_1^2 + J_{1zz}).$$

Energia potencjalna ramienia 2 E_{p2} posiada wartość zależną od jego masy, przyspieszenia grawitacyjnego oraz odległości środka masy od osi obrotu. Energia kinetyczna ramienia 2 E_{k2} powstaje w sposób analogiczny do pierwszego. Obie wspomniane energie wyznaczane są dzięki zależnościom:

$$E_{p2} = g m_2 l_2 (1 - cos(\theta_2)),$$

$$E_{k2} = \frac{1}{2}({v_{2c}}^T m_2 v_{2c} + {\omega_2}^T J_2 \omega_2)$$
$$= \frac{1}{2}\dot{\theta}_1^2\left(m_2 L_2^2 + \left(m_2 l_2^2 + J_{2yy}\right)sin^2(\theta_2) + J_{2xx}cos^2(\theta_2)\right) + \frac{1}{2}\dot{\theta}_2^2(J_{2zz} + m_2 l_2^2)$$
$$+ m_2 L_1 l_2\, cos(\theta_2)\, \dot{\theta}_1 \dot{\theta}_2.$$

Energia całego obiektu jest sumą poszczególnych energii obu ramion. Lagranżjan opisuje różnice całkowitej energii kinetycznej oraz potencjalnej:

$$E_p = E_{p1} + E_{p2},$$

$$E_k = E_{k1} + E_{k2},$$

$$L = E_k - E_p.$$

2.2. Równania dynamiki obiektu

Równanie Euler'a-Lagrange'a (2) jest opisem ruchu obiektu z uogólnionymi układami współrzędnych. Daje ono możliwość przejścia na stanowy model układu.

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_\iota}\right) + b_i \dot{q}_\iota - \frac{\partial L}{\partial q_I} = Q_i, \tag{2}$$

gdzie w L jest lagranżjanem, $q_i = [\theta_1, \theta_2]^T$ są uogólnionymi współrzędnymi kątowymi ramion, $b_i = [\mathrm{b}_1, \mathrm{b}_2]^T$ są uogólnionymi współczynnikami tłumiącymi, a $Q_i = [\tau_1, \tau_2]^T$ są uogólnionymi momentami siły. W tym momencie należy policzyć kolejne pochodne lagranżjana po prędkościach, bądź położeniach kątowych odpowiednich ramion:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_1}\right) = \ddot{\theta}_1\left(J_{1zz} + m_1 l_1^2 + m_2 L_1^2 + \left(m_1 l_2^2 + J_{2yy}\right)sin^2(\theta_2) + J_{2xx}cos^2(\theta_2)\right) + m_2 L_1 l_2\, cos(\theta_2)\, \ddot{\theta}_2$$
$$- m_2 L_1 l_2\, sin(\theta_2)\, \dot{\theta}_2^2 + \dot{\theta}_1 \dot{\theta}_2\, sin(2\theta_2)\left(m_2 l_2^2 + J_{2yy} - J_{2xx}\right),$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_2}\right) = \ddot{\theta}_1 m_2 L_1 l_2\, cos(\theta_2) + \ddot{\theta}_2(J_{2zz} + m_2 l_2^2) - \dot{\theta}_1 \dot{\theta}_2 m_2 L_1 l_2\, sin(\theta_2),$$

$$-\frac{\partial L}{\partial \theta_1} = 0,$$

$$-\frac{\partial L}{\partial \theta_1} = -\frac{1}{2}\dot{\theta}_1^2\, sin(2\theta_2)\left(m_2 l_2^2 + J_{2yy} - J_{2xx}\right) + \dot{\theta}_1\dot{\theta}_2 m_2 L_1 l_2\, sin(\theta_2) + g m_2 l_2\, sin(\theta_2).$$

Dzięki policzonym pochodnym można podstawić powyższe zależności do równania 2, Euler'a-Lagrange'a.

$$\begin{bmatrix} ((\ddot{\theta}_1(J_{1zz} + m_1 l_1^2 + m_2 L_1^2 + \left(J_{2yy} + m_2 l_2^2 \times sin^2(\theta_2) + J_{2xx} cos^2(\theta_2)\right) + \ddot{\theta}_2 m_2 L_1 l_2 \cos(\theta_2)) \\ -m_2 L_1 l_2 \sin(\theta_2)\,\dot{\theta}_2^2 + \dot{\theta}_1\dot{\theta}_2 \sin(2\theta_2) \times \left(m_2 l_2^2 + J_{2yy} - J_{2xx}\right) + b_1\dot{\theta}_1)) \\ (\ddot{\theta}_1 m_2 L_1 l_2 \cos(\theta_2) + \ddot{\theta}_2(m_2 l_2^2 + J_{2zz}) + \frac{1}{2}\dot{\theta}_1^2 \sin(2\theta_2)\left(-m_2 l_2^2 - J_{2yy} + J_{2xx}\right) + b_2\dot{\theta}_2 + g m_2 l_2 \sin(\theta_2))) \end{bmatrix} =$$

$$= \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix}. \quad (3)$$

Stosując przekształcenia matematyczne można zamienić równanie (3) w równania macierzy stanu, gdzie stanami byłyby prędkości kątowe każdego ramienia oraz ich pochodne.

Żeby rozpocząć sterowanie obiektem, należy jeszcze zidentyfikować model silnika napędzającego cały układ oraz wybrać i dostroić jeden z dostępnych sterowników. Rozdział ten ma za zadanie przedstawić równania fizyczne kryjące się za problemem wahadła Furuta, dlatego kolejne kroki nie zostaną tutaj przedstawione. Sterownie wahadłem w oparciu o standardowe teorie sterowania wymaga biegłej znajomości dynamiki klasycznej jak i równań różniczkowych. Dodatkowo, żeby właściwie zaprojektować sterownik niezbędne jest używanie wielu metod zawartych w teorii sterowania.

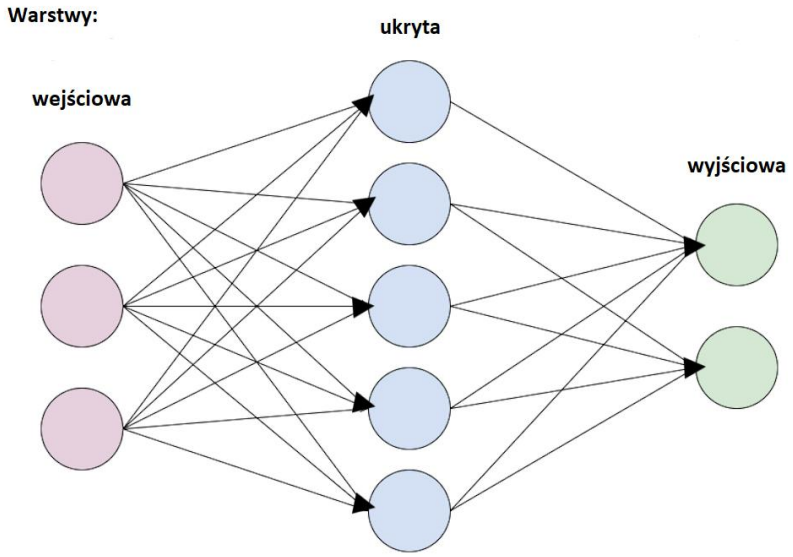
3. STEROWANIE ZA POMOCĄ SZTUCZNEJ INTELIGENCJI (PAWEŁ ZIELIŃSKI)

Klasyczne podejście do sterowania w głównej mierze polega na utworzeniu modelu stanowego poprzez identyfikację parametrów obiektu a następnie zaprojektowaniu sterownika. W tej części przedstawione zostaną metody oparte na sztucznej inteligencji, umożliwiające wyeliminowanie żmudnej pracy towarzyszącej wspomnianemu podejściu. Tym razem czas poświęcony na identyfikację obiektu zastąpiony będzie czasem jaki algorytm potrzebował będzie na naukę. Sterowanie przy wykorzystaniu uczącej się, głębokiej sieci neuronowej nie wymaga od automatyka obszernej wiedzy na temat teorii sterowania. Jako architekci budujemy oprogramowanie sterujące, gdzie dostarczaną informacją jest aktualny stan obiektu.

3.1. Głębokie sieci neuronowe

Wprowadzenie do głębokich sieci neuronowych nie jest możliwe bez wcześniejszego zaznajomienia się z pojęciem sieci neuronowej, gdyż jedno jest rozwinięciem drugiego. Okazuję się, że poprzez naukę procesów zachodzących wewnątrz naszego organizmu jesteśmy w stanie wykorzystać te wiedzę jako narzędzie do rozwiązywania skomplikowanych problemów. Już w 1943 roku Warren McCulloch i Walter Pitts zbudowali model komputera, który wykonywał

operacje logiczne oraz matematyczne zwane logiką progową. Funkcje te miały za zadane imitację działania ludzkiego procesu myślowego. Od tego czasu technologia stale się rozwijała, lecz poziom wysokiej użyteczności uzyskała dopiero przy końcu ubiegłego wieku. Sztuczna sieć neuronowa powstała jako imitacja działań zachodzących w ludzkim systemie nerwowym. Sieć neuronowa składa się z warstw, a każda warstwa zawiera określoną liczbę neuronów. Patrząc od strony architektonicznej, neuron traktowany jest jak węzeł sieci posiadający skończoną liczbę wejść oraz jedno, możliwe do powielenia wyjście. Każde połączenie jest wagą, której znaczenie zgłębione zostanie w dalszej części tekstu. Warstwy dzielone są na trzy rodzaje. Jest wejściowa, która przyjmuje oraz przetwarza dane wejściowe do sieci. Istnieje jedna, lub więcej ukrytych warstw, odpowiadających za interpretacje przetwarzanych danych. Ostatnia warstwa, zwana wyjściową, ma za zadanie aktywować wyjścia.



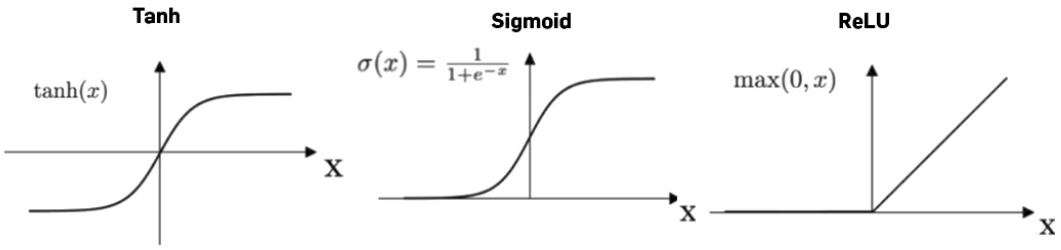
Rys. 3.1 Warstwowa budowa sieci

Działanie sieci neuronowej można przybliżyć do problemu parametrycznej aproksymacji funkcji nieliniowej. Przybliżenie jest tym trafniejsze, im bardziej budowa sieci odpowiada poziomowi skomplikowania aproksymowanej funkcji. Dokładność zależy również od zasobów czasowych oraz materiałowych użytych podczas treningu sieci. Waga jest to regularnie aktualizowany czynnik obciążający odpowiednie wejście neuronu. Z wolnego tłumaczenia - przesunięcie (ang. *Bias*) jest to stała wartość dodawana do wartości neuronu, przesuwająca funkcję aktywacji w osi x. Działanie pojedynczego neuronu, będące nieliniową transformacją, opisane jest za pomocą równania:

$$z = \mathrm{h}\left(\sum_{k=1}^{n}(w_k x_k + w_0)\right), \tag{4}$$

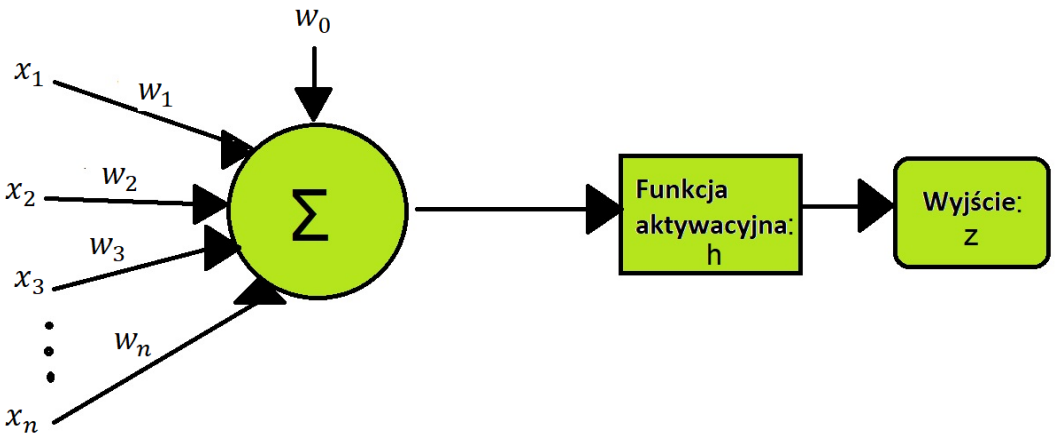
gdzie z jest wyjściem naszego neuronu, x_k jest *k*-tym wejściem, w_k jest wagą przypisaną pod *k*-te wejście, zaś w_0 reprezentuje przesunięcie. Sumowanie obejmuje wszystkie wejścia neuronu. Niezbędnym elementem działania neuronu jest funkcja aktywacji, reprezentowana

w powyższym równaniu jako h(). Dzięki niej następuje normalizacja wejść i wyjść neuronów. Funkcja aktywacji dodaje nieliniowość w pracy neuronu, co umożliwia całej sieci na aproksymacje złożonych, nieliniowych funkcji. Najpowszechniejsze z nich przedstawione zostały na rysunku 3.2.



Rys. 3.2 Wykresy funkcji aktywacji [4]

Równanie (4) można przedstawić w bardziej przystępny sposób tj. za pomocą schematu blokowego. Neuron przyjmuje przemnożone przez odpowiadające wagi wejścia oraz sumuje je z przesunięciem. Otrzymany wynik normalizuje funkcją aktywacji, co dostarcza gotowe wyjście, mogące albo wchodzić do kolejnego neuronu albo wychodzić na zewnątrz sieci.



Rys. 3.3 Schemat blokowy działania neuronu

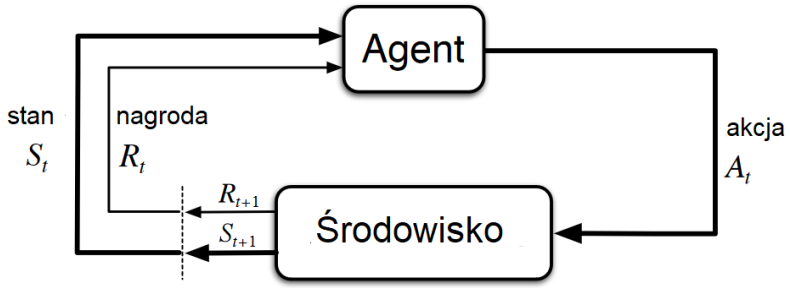
Należy podkreślić, że aby sztuczna sieć neuronowa mogła być nazwana głęboką, musi posiadać przynajmniej dwie warstwy ukryte. Na potrzeby tego projektu dobrana została sieć zwana głęboką siecią ze sprzężeniem w przód (ang. *Deep Feed Forward*). Wyróżnia ją brak sprzężeń zwrotnych. Użyta sieć nazywana jest całkowicie połączoną (ang. *Fully Connected*), co oznacza połączenie każdego neuronu ze wszystkimi neuronami w poprzedzającej go warstwie.

3.2. Uczenie ze wzmocnieniem

Wykorzystany i opisany w tym projekcie algorytm uczenia ze wzmocnieniem najpierw musi zostać skategoryzowany na mapie sztucznej inteligencji. Uczenie ze wzmocnieniem (ang. *Reinforcement Learinng*) jest jedną z trzech głównych metod uczenia maszynowego. Do pozostałych dwóch należy uczenie nadzorowane (ang. *Supervised*) oraz uczenie bez nadzoru (ang. *Unsupervised*). Tutaj jednak skupimy się na RL zawierającym gromadę algorytmów bardzo dobrze nadających się do wykorzystania w sterowaniu silnie nieliniowym obiektem. W przypadku tej pracy jest to wahadło Furuta.

3.2.1. Proces decyzyjny Markova

Procesem, o który opera się działanie wszystkich algorytmów uczenia wzmocnionego jest proces decyzyjny Markowa (ang. *Markov Decision Process*). Jednym z elementów MDP jest agent. Ma on za zadanie podejmować decyzje sterowania. Akcje realnie oddziałują na środowisko zmieniając jego stan. W miarę postępu działania algorytmu, agent uczy się podejmować trafniejsze decyzje, zbliżające go do poprawnego sterowania obiektem. Informacją zwrotną, określającą trafność podjętych akcji jest funkcja nagrody.



Rys. 3.4 Schemat działania MDP [1]

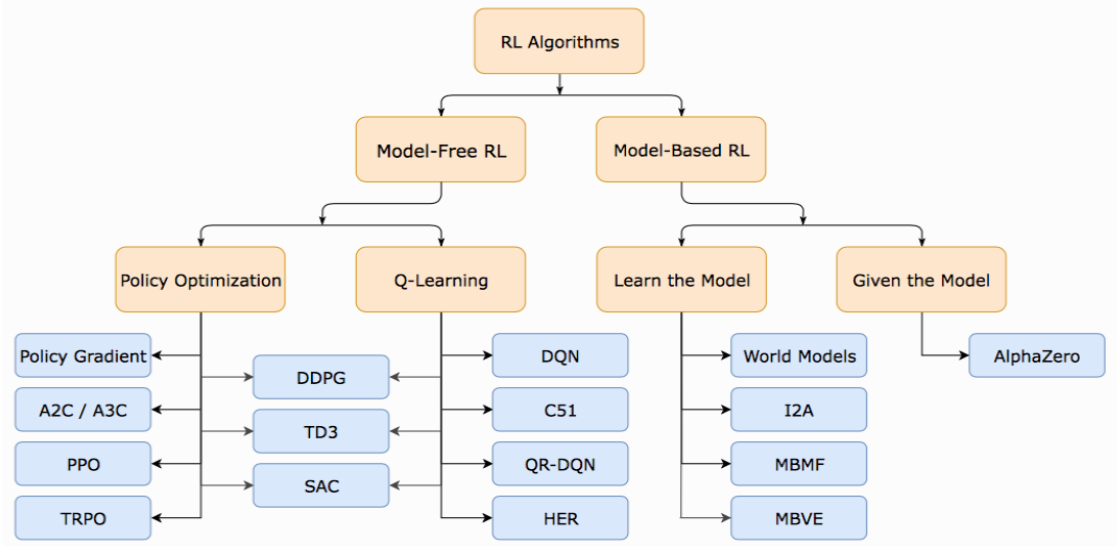
Z początku agent przyjmuje stan środowiska w momencie *t (ang. Prestate)*. Po odpowiedniej interpretacji podejmuje decyzję o wykonaniu akcji w czasie *t*, która jest jego wartością wyjściową. Działanie to wpływa na środowisko, zmieniając jego stan, który określony jest w chwili *t+1 (ang. Poststate)*. Za pomocą tych danych jesteśmy w stanie policzyć nagrodę dodaną za wykonaną akcję oraz zdefiniować ją jako otrzymaną w chwili *t+1*. Następnie w miejscu oznaczonym na rysunku 3.4 za pomocą przerywanej linii, następuje przesunięcie wszystkich danych w czasie oraz rozpoczęcie całego cyklu na nowo.

3.2.2. Funkcja nagrody

Informacją zwrotną otrzymywaną przez agenta, dostarczającą informację o poprawności wykonanej akcji jest wartość funkcji nagrody. W algorytmach uczenia wzmocnionego agent jest nagradzany albo karany, przy czym oba te pojęcia różni dodatnia

lub ujemna ich wartość. W przypadku nagrody sieć neuronowa dąży do jej maksymalizacji, zaś w przypadku kary do minimalizacji. Należy dodać, iż im precyzyjniej dobrane zostaną parametry oraz ich skalowanie (tworząc funkcję nagrody), tym trafniej agent będzie osiągał zakładane cele.

3.2.3. Wybór algorytmu RL



Rys. 3.5 Drzewo algorytmów RL [6]

Na rysunku 3.5 przedstawiono drzewo algorytmów RL. Początkowo dzielone są one na te działające w oparciu o identyfikacje obiektu za pomocą tworzenia modelu oraz te, w których działania takie nie są wymagane. Do sterowania dynamicznymi obiektami nieliniowymi, gdzie identyfikacja jest bardzo skomplikowana, częściej stosowane są algorytmy niedziałające w oparciu o matematyczny model obiektu.

Algorytmy niebazujące na modelu, dzielone są na metody optymalizujące taktykę (ang. *Policy Optimazation*) oraz takie, których działanie iteracyjnie aproksymuje funkcję Q. Istotnym wyróżnikiem przy implementacji obu metod jest postać wyjść sieci. Algorytmy optymalizujące taktykę posiadają wyjścia stochastyczne, mogące przyjąć wartości liczb mieszczących się w ograniczonym przedziale. Grupa metod aproksymujących funkcję Q posiada pojedynczą wartość liczbową przypisaną do każdego wyjścia, co nazywane jest podejściem deterministycznym. Na pierwszy rzut oka to algorytmy posiadające wyjścia stochastyczne powinny lepiej radzić sobie z problemem odwróconego wahadła. Pożądane jest, aby silnik rozbujał wahadło w jak najszybszym czasie, stosując duże wartości momentu siły. Gdy wahadło ustawione jest już w pozycji pionowej (odwróconej) wymagane jest sterowanie akcjami o odpowiednio mniejszej wartości. Praca naukowa [8], zestawiająca najpopularniejsze algorytmy RL, przedstawia osiągi każdego z nich w starciu z problemem sterowania

odwróconego wahadła. Metodą doświadczalną wykazano, że właśnie algorytm DQN, wraz ze swoimi ulepszeniami, osiągał najlepsze wyniki. Zadanie to przerosło metody doboru optymalnej polisy ze względu na zbyt złożone modele optymalizacyjne. Jak się okazuje, trzy bądź pięć możliwych do wykonania akcji w satysfakcjonujący sposób umożliwia osiągnięcie zakładanego celu, znacząco upraszając algorytm.

3.2.4. Algorytm głębokiego uczenia funkcji Q

Po każdorazowym upływie próbki czasu *t*, do każdej akcji wybieranej przez agenta, według równania (5), dobierana jest wartość funkcji Q – jakości (ang. *Quality*). Wartość ta interpretowana jest jako suma nagrody za obecną akcję wraz z maksymalną przewidywaną nagrodą otrzymaną po wybraniu akcji kolejnych. Stosując ten algorytm należy zdecydować się na kilka wartości wyjściowych (akcji). Wybierana jest ta z największą wartością funkcji Q.

$$Q_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T \tag{5}$$

Zadania kontynuowane w czasie bez podziału na epizody okazują się problematyczne, gdyż definicja funkcji Q (równanie 5) niezbyt dobrze określa wartość sumy dla $T = \infty$. W tym przypadku sam wynik skumulowanej nagrody może być równy nieskończoności.

Wprowadzono następującą modyfikację równania (5). Zamiast posługiwać się całkowitą przyszłą nagrodą, użyto funkcji obniżonej całkowitej przyszłej nagrody (ang. *Discounted Future Reward*), inaczej nazywanej funkcją celu. Występuje w niej dodatkowy współczynnik zwany obniżającym γ, który przyjmuje wartości od 0 do 1. Ma on na celu zmniejszenie znaczenia wartości nagrody otrzymanej w każdym kolejnym momencie. Im dalej sięgamy w przyszłość, tym mniej pewna jest nasza przewidywana nagroda. Algorytm bardziej skupia się na natychmiastowej nagrodzie, która jest najpewniejsza. Dodatkowo, posługując się własnościami zbieżnych ciągów arytmetycznych, pozwala na uproszczony zapis równania (5). Po zastosowaniu omówionych zmian w definicji funkcji Q otrzymujemy równanie Bellmana:

$$Q_t = R_{t+1} + \gamma \cdot max\,(Q_{t+1}). \tag{6}$$

3.2.4.1. Funkcja straty

W drodze do ciągłego poprawiania osiągów sieci estymowana jest funkcja straty (ang. *Loss Function*). Za jej pomocą otrzymujemy informację zwrotną, dotyczącą poprawności dokonywanych przewidywań wartości funkcji Q. Opiera się ona na obliczeniu błędu średniokwadratowego. Funkcję straty opisuję zależność:

$$L = (R_{t+1} + \gamma \cdot max\,(Q_{t+1}) - Q_t)^2. \tag{7}$$

Stratę zawartą w nazwie równania (7) można rozumieć jako trafność estymacji funkcji celu. Odjemną w działaniu różnicy błędu średniokwadratowego jest funkcja celu, zaś odjemnikiem aproksymowana wartość całkowitej nagrody w poprzedniej chwili czasu (Q_t). W równaniu (7) natrafiamy na tak zwany problem uciekającego celu. Powstaje on podczas korzystania z jednej sieci neuronowej do estymacji wartość funkcji celu oraz wartość funkcji Q. W miarę zbliżania się wartości funkcji Q do obecnej wartości funkcji celu, ta druga dokładnie w takim samym tempie się oddala. Żeby zniwelować skutki tego zjawiska zaczęto stosować dwie oddzielne sieci do estymowania dwóch wspomnianych wartości. Pierwsza zwana siecią Q (ang. *Q-network*) służy do wyliczeń estymowanych wartości Q_t. Druga, sieć celu (ang. Target Network) używana jest do aproksymacji wartości Q_{t+1}. Strukturalnie obie te sieci są identyczne. Każdorazowo, podczas wykonywania sesji, aktualizowane są tylko wagi sieci Q, podczas gdy sieć celu ma swoje wagi zamrożone. Żeby uczenie postępowało, każdorazowo po upływie stale założonej liczby kroków, wagi sieci celu zostają całkowicie zrównane z regularnie aktualizowanymi nastawami sieci Q.

3.2.4.2. Eksploracja kontra eksploatacja

Istotną kwestią przy tworzeniu algorytmu uczącego głębokiej sieci Q jest nadanie stosunku, w jakim agent poznaje nowe ścieżki prowadzące do osiągnięcia celu (eksploracja) lub poprawia te już poznane (eksploatacja). Z początku głęboka sieć posiada wagi przypisane w sposób losowy, co sprawia, że rozwiązania podsuwane za pomocą funkcji Q są dalekie od poprawności. Z tego powodu należy eksplorować różne ścieżki, a odbywa się to poprzez wybieranie mimowolnych akcji. W miarę postępu uczenia nabieramy zaufania do poprawności akcji podejmowanych przez sieć neuronową. Współczynnikiem określającym wybór akcji losowej, bądź tej dobranej za pomocą estymacji funkcji Q jest epsilon (ε). Początkowo jego wartość ustawiana jest jako bliska 1, co oznacza prawdopodobieństwo wybrania losowej akcji. Analogicznie, akcja którą wyznaczyła sieć neuronowa wybierana jest z prawdopodobieństwem (1- ε). W kolejnych iteracjach działania algorytmu epsilon podlega regresji, powoli niwelując szansę na wybór kolejnej akcji w sposób losowy.

3.2.4.3. Pamięć doświadczeń (ang. Experience replay)

Algorytm, który poprawia wagi swojej sieci na podstawie danych z pojedynczego doświadczenia jest bardzo niestabilny. Zachowanie to przejawia się destabilizacją dobrze dobranych wagi sieci, potocznie nazywaną zapominaniem. W tym celu algorytm DQL korzysta z pamięci do przechowywania doświadczeń.

$$e = (s_t, a_t, r_{t+1}, s_{t+1}). \qquad (8)$$

Dane pojedynczego przejścia e kolejno zawierają informacje o stanie w chwili t s_t akcji jaka została dla niego podjęta a_t, nagrodzie otrzymanej za wykonaną akcję r_{t+1}

oraz stanie w jakim kolejno obiekt się znalazł s_{t+1}. Tak przygotowana wartość, zwana próbką (ang. *Sample*), zapisywana jest w pamięci, przeważnie w przygotowanej to tego celu tablicy.

3.2.4.4. Aktualizacja wag

Aktualizacja wag odbywa się podczas uczenia sieci. Z całej zgromadzonej pamięci doświadczeń losowo wydobywa się partię (ang. *Batch*) doświadczeń. Stochastyczny wybór próbek niweluje korelację uczenia sieci, która występowałaby w przypadku nauki stanami kolejno zachodzącymi po sobie. Następnie partia dzielona jest na minipartie (ang. *Minibatch*), z których przeliczana jest suma błędu średniokwadratowego funkcji straty (punkt 3.2.4.1). Traktując funkcję straty jako funkcję kosztu, a wagi sieci neuronowej jako zmienne optymalizacyjne, przeprowadzamy proces minimalizacji. Wyliczany jest gradient dostarczający informację o kierunku najszybszego spadku wartości funkcji straty w przestrzeni wag. Krok minimalizacji kontrolowany za pomocą wskaźnika nazywanego tempem uczenia (α). Opisany proces zachodzi dzięki propagacji wstecznej (and. backpropagation), a następnie aktualizowane są wagi sieci Q. Metoda używana w tym schemacie nosi nazwę stochastycznego zejścia gradientowego (ang. *Stochastic Gradient Descent*) oraz powszechnie stosowana jest do aktualizacji wag sieci w algorytmach uczenia ze wzmocnieniem oraz z nadzorem.

3.2.4.5. Optymalizator ADAM

Poprawioną wersją optymalizatora stochastycznego zejścia gradientowego jest ADAM (ang. *Adaptive Moment Estimation*). Od zejścia gradientowego wyróżniają go ciągle wyznaczane estymaty średniej oraz wariancji momentu gradientów. Wyliczona za pomocą współczynnika β_1 średnia oraz za pomocą β_2 wariancja momentu gradientu kolejno uwzględniania jest przy aktualizacji każdej wagi. Zmiana uwzględnia jeszcze stałą η determinującą wielkość wprowadzonych zmian. Udowodnione jest, że algorytm ADAM trafniej odnajduje minimum szukanej funkcji, co powoduję że jest częściej wybieranym optymalizatorem w algorytmach uczenia ze wzmocnieniem.

3.2.4.6. Kompletny algorytm DQN

Kompletny algorytm DQN można opisać za pomocą następujących punktów [7]:

1. Wczytaj pamięć powtórki doświadczeń
2. Utwórz siec Q o losowych wartościach wag
3. Utwórz sieć celu i nadaj jej wagi sieci Q
4. **For** sesje wykonuj
5. Wczytaj stan obecny s_t
6. **For** czas sesji wykonuj
7. Wybierz akcje a_t:

- Losową z prawdopodobieństwem ε
- Proponowaną siecią Q z prawdopodobieństwem (1-ε)

8. Wykonaj akcję a_t oraz obserwuj nagrodę r_t
9. Wczytaj stan kolejny s_{t+1}
10. Zapisz doświadczenie e w ER
11. Wyciągnij losową partię doświadczeń z ER
12. **For** kolejnych e z partii wykonuj
13. Oblicz funkcję celu za pomocą równania Bellmana (6)
14. Aktualizuj parametry sieci Q przy użyciu równania straty (7)
15. Co określoną liczbę kroków przepisz wagi sieci celu do wag sieci Q.

4. IMPLEMENTACJA ALGORYTMU DQN I SYMULACJI (PAWEŁ ZIELIŃSKI)

Opisany w poprzednim rozdziale algorytm uczący zawiera wysoki stopień złożoności, z powodu którego dobór odpowiedniego środowiska programistycznego jest kluczowy. Środowisko powinno być zdolne do szybkiego przetwarzania dużej ilości danych w ograniczonym czasie. Wymagane jest również programowanie wysoko poziomowe, nieskupiające się na prostych operacjach, lecz na tych zdefiniowanych przez algorytm. Środowiskiem spełniającym przytoczone wymagania jest Matlab oferowany przez firmę Mathworks. Dodatkową zaletą tego środowiska jest możliwość prostej symulacji obiektu fizycznego za pomocą rozszerzenia Simulink.

Matlab zawiera gotowe narzędzie, pod nazwą ‘Deep Learning Toolbox’ umożliwiające implementację algorytmów uczenia ze wzmocnieniem w bardzo przystępny sposób. Skorzystano z tego narzędzia, gdyż jednym z celów pracy było utworzenie własnego programu sterującego. Jedyne gotowe biblioteki dotyczące uczenia głębokiego, używane przy tworzeniu programu, służyły do implementacji sieci neuronowej w postaci struktury, obliczania gradientów najszybszego spadku oraz aktualizacji wag sieci.

4.1. Simulink

Sprawdzenie poprawności działania algorytmu sterującego początkowo należy testować na wiernie odzwierciedlającej rzeczywisty obiekt symulacji. W projekcie, do tworzenia skomplikowanych symulacji użyto środowiska Simulink. Rozwiązanie problemu przy pomocy tego oprogramowania głównie opiera się na tworzeniu oraz łączeniu wyspecjalizowanych bloków. W przypadku opisywanego problemu, konieczne stało się spełnienie przez nasz model dwóch głównych założeń: zamodelowanie modelu fizycznego wahadła oraz sterowanie nim.

4.1.1. Model matematyczny

Symulacja bazuje na modelu matematycznym opisanym w rozdziale 2. Model wahadła podzielony jest na dwie części: ramię 1, przenoszące moment siły z silnika na wahadło oraz ramię 2 będące obiektem sterowania. Sterowanie takim obiektem odbywa się za pomocą momentu obrotowego generowanego przez silnik, wpływającego na przyspieszenia kątowe obu ramion w sposób opisany za pomocą zależności:

$$\begin{bmatrix}\ddot{\theta}_1\\ \dot{\theta}_2\end{bmatrix} = \begin{bmatrix}\widehat{J_0} + \widehat{J_2}sin^2(\theta_2) & m_2L_1l_2\,cos(\theta_2)\\ m_2L_1l_2\,cos(\theta_2) & \widehat{J_2}\end{bmatrix}^{-1} \times$$

$$\left(\begin{bmatrix} b_1 + \frac{1}{2}\dot{\theta}_2\widehat{J_2}sin\,(2\theta_2) & \frac{1}{2}\dot{\theta}_2\widehat{J_2}\,sin(2\theta_2) - m_2L_1l_2sin\,(\theta_2)\dot{\theta}_2\\ -\frac{1}{2}\dot{\theta}_1\widehat{J_2}sin\,(2\theta_2) & b_2\end{bmatrix} \times \begin{bmatrix}\dot{\theta}_1\\ \dot{\theta}_2\end{bmatrix} - \begin{bmatrix}0\\ gm_2l_2\,sin(\theta_2)\end{bmatrix} + \begin{bmatrix}\tau_1\\ \tau_2\end{bmatrix}\right),$$

gdzie:

θ_1 jest położeniem kątowym ramienia 1 $[RAD]$,

θ_2 jest położeniem kątowym ramienia 2 $[RAD]$,

$\dot{\theta}_1$ jest prędkością kątową ramienia 1 $[\frac{RAD}{s}]$,

$\dot{\theta}_2$ jest prędkością kątową ramienia 2 $[\frac{RAD}{s}]$,

$\ddot{\theta}_1$ jest przyspieszeniem kątowym ramienia 1 $[\frac{RAD}{s^2}]$,

$\ddot{\theta}_2$ jest przyspieszeniem kątowym ramienia 2 $[\frac{RAD}{s^2}]$,

m_1 jest masą ramienia 1 równą 90 g,

m_2 jest masą ramienia 2 równą 15 g,

L_1 jest długością ramienia 1 równą $0{,}05$ m,

L_2 jest długością ramienia 2 równą $0{,}1$ m,

l_1 jest odległością środka masy ramienia 1 od jego osi obrotu równą $22\ mm$,

l_2 jest odległością środka masy ramienia 2 od jego osi obrotu równą $50\ mm$,

$\widehat{J_1}$ jest momentem bezwładności ramienia 1 równym $43{,}6$ µ kgm^2,

$\widehat{J_2}$ jest momentem bezwładności ramienia 2 równym 50 µ kgm^2,

$\widehat{J_0}$ jest momentem bezwładności ramienia 2 w odniesieniu do osi obrotu silnika równym 81 µ $kg \cdot m^2$,

g jest przyspieszeniem grawitacyjnym przybliżonym do wartości $9{,}81\ \frac{m}{s^2}$,

b_1 jest współczynnikiem tłumienia dla ramienia 1 równym $-5 \cdot 10^{-3}\ Nms$,

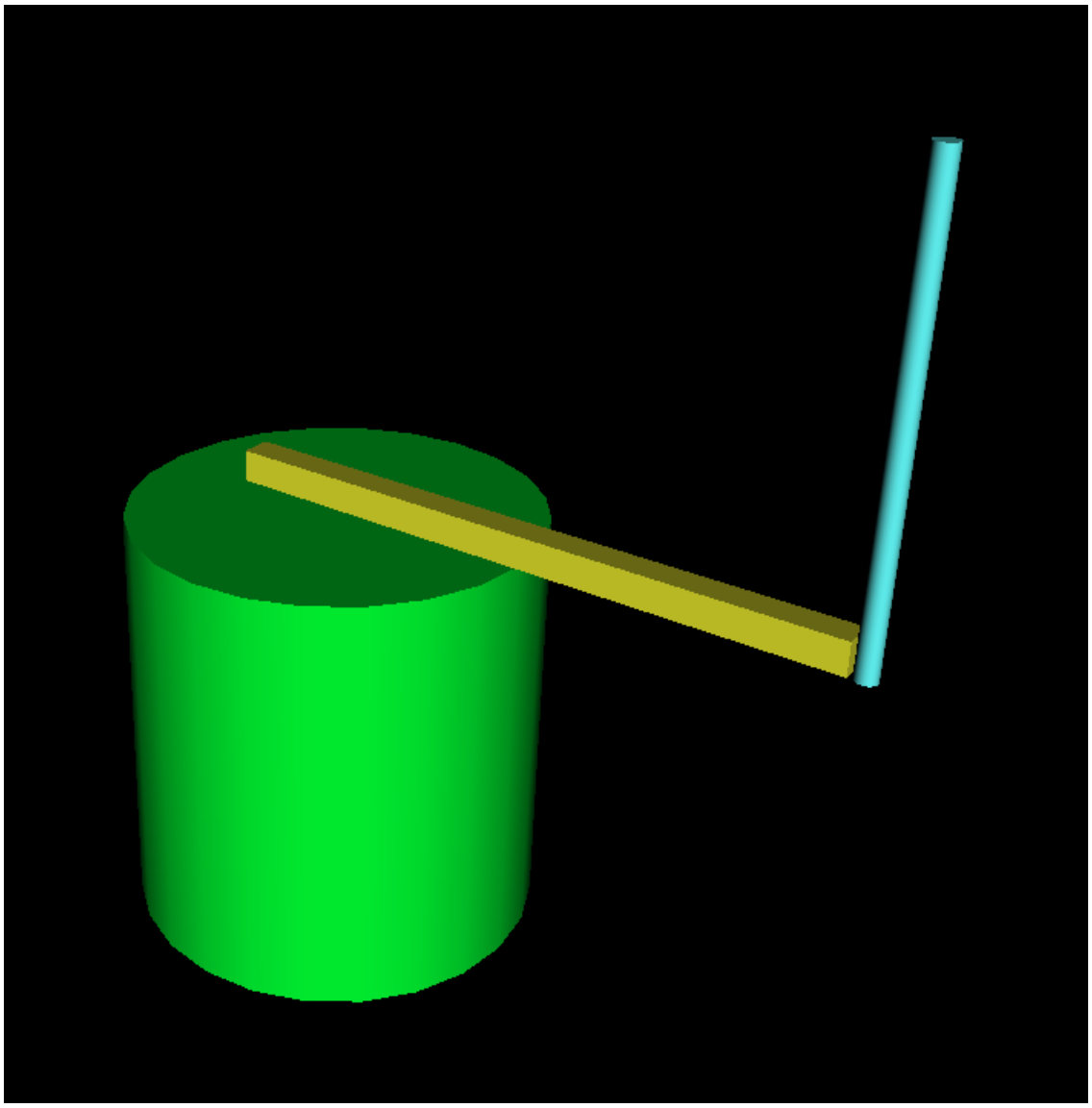
b_2 jest współczynnikiem tłumienia dla ramienia 2 równym $-5 \cdot 10^{-5}\ Nms$,

τ_1 jest momentem siły nadanym na ramię $[Nm]$,

τ_2 jest zakłócającym momentem siły nadanym na ramię 2$[Nm]$.

Obliczone wartości przyspieszeń modyfikowane są iteracyjnie na podstawie poprzednich wartości prędkości i położeń kątowych. W symulacji konieczne było nadanie zmiennym warunków początkowych, które określały pozycje swobodnego, nieruchomego zwisu wahadła. Żeby symulacja jak najrzetelniej odwzorowywała rzeczywisty obiekt, należało zidentyfikować wielkości fizyczne opisujące obiekt, którego budowę opisano rozdziale 0. Dokładność pomiarów masy oraz długości ramion wahadła określona na potrzeby tej pracy równa była trzem cyfrom znaczącym. Dla masy było to ±0,001 kg, a dla długości ±0,001 m. Przybliżone wartości momentów bezwładności wyznaczone zostały na podstawie dostępnych wielkości fizycznych oraz prostych wzorów. Problemem okazało się wyznaczanie współczynników tłumienia b_i. Brak dostępu do odpowiednich mierników wymusił oszacowanie ich wartości w sposób eksperymentalny. Symulacja uruchamiana była w czasie rzeczywistym z zerowym pobudzeniem oraz określonymi warunkami początkowymi. W tych samych warunkach stawiano rzeczywisty obiekt. Kolejno dostrajano współczynniki tłumienia tak, aby działanie symulowanego wahadła odpowiadało zachowaniu rzeczywistego obiektu.

4.1.2. Wizualizacja działania symulacji



Rys. 4.1 Wizualizacja działania wahadła

W celu obserwacji wyników działania symulacji zbudowany został blok wizualizacji 3-D. Parametrami wejściowymi bloku wizualizacji są położenia kątowe obu ramion. Dobierając odpowiednią częstotliwość odświeżania obrazu otrzymujemy realistycznie poruszający się model wahadła.

4.2. Program Sterujący

Agent odpowiedzialny za naukę oraz wykonywanie akcji, zaimplementowany został przy użyciu klasy. Jest on rdzeniem całego programu, gdyż reszta komponentów jest od niego pochodna. Wszelkie hiperparametry użyte podczas wykonywania algorytmu są jej właściwościami, czyli ściśle związanymi zmiennymi.

4.2.1. Pamięć doświadczeń

Pamięć rejestrująca wszystkie doświadczenia jest właściwością agenta. Omawiana metoda klasy głównej mieści się w osobnym skrypcie. Jedną z jej funkcji jest dodawanie do struktury, zawierającej doświadczenia, kolejno zarejestrowanych rekordów. Początkowo dodaje się kolejne elementy, jednak gdy ich liczba przekroczy zdeklarowany rozmiar, dodawane doświadczenia zastępują te najstarsze. Działanie takiej struktury znane jest w środowisku programistycznym jako kolejka, a przytoczona metoda przechowywania danych kieruje się zasadą najstarszy wychodzi pierwszy (ang. *LIFO - Last In, First Out*). Zawartość pojedynczej próbki doświadczeń opisana została w podrozdziale 3.2.4.3.

4.2.2. Implementacja głębokiej sieci neuronowej

Sieć neuronowa jest jedynym elementem w napisanym kodzie, której utworzenie i na której operacje wykonywane są przy pomocy gotowych bibliotek związanych z głębokimi sieciami neuronowymi. Pochodzą one ze specjalnego narzędzia dostępnego w źródle [14]. Początkowo definiowane są rozmiary warstw sieci. Następnie warstwom ukrytym przypisywane są funkcje aktywacyjne, które w rozwiązywanym problemie określone zostały na ReLU (ang. *Rectified Linear Units*) (punkt 3.13.1). W stworzonym oprogramowaniu warstwa wejściowa zawiera liczbę neuronów równą 4, a wyjściowa 3. Jako wejścia sieci podawane są kolejno: sinus oraz cosinus położenia kątowego wahadła, jego prędkość kątowa oraz prędkość ramienia nim obracającego. Wyjściami są trzy możliwe do podjęcia akcje, interpretowane jako moment obrotowy silnika. Pierwsze dwie ustalone są jako dodatnia oraz ujemna wartość tej samej akcji. Przyjęta wartość musi być precyzyjnie dobrana. Zbyt mały moment siły sprawi, że rozbujanie wahadła będzie niemożliwe. Zbyt duży spowoduje za małą precyzje ruchów na szczycie, uniemożliwiającą balansowanie wahała w pozycji pionowej. Wnikliwie analizując efekt tego parametru oraz biorąc pod uwagę możliwości zakupionego silnika, moment siły wytwarzany przez silnik przyjął wartość 0,036 Nm. Trzecie wyjście oznacza moment siły równy zero, czyli bezczynność silnika. Wartość ta jest pomocna w przypadku, gdy wahadło zachowuje się w sposób pożądany i żadne zmiany nie są wymagane.

Sieć użyta w algorytmie DQN (punkt 3.2.4.6) sterującym wahadłem przeważnie posiada dwie warstwy ukryte a liczba neuronów w nich zamieszonych mieści się między 24 a 96. Kolejno konieczne jest przypisanie wartości do wag utworzonej sieci. Domyślną metodą stosowaną w omawianym narzędziu jest inicjalizacja wartości wag uzależniona od rozkładu gaussowskiego o wzorze $\sqrt{\frac{2}{m}}$, gdzie m jest równe rozmiarowi poprzedzającej wagę warstwy.

Wybór akcji

Taktyka stosowana w algorytmie DQN determinująca wybór akcji nazywana jest chciwym epsilonem (ang. $\boldsymbol{\varepsilon}$ *Greedy Policy*). Jej efektywność opiera się na znalezieniu

kompromisu między eksploracją a eksploatacją rozwiązań (punkt 3.2.4.2). Wraz z postępem uczenia zmniejsza się prawdopodobieństwo wybrania akcji losowej, które zaczyna się od wartości równej 100%. Zalecanym jest, aby szansa ta nigdy nie zeszła do wartości zerowej, gdyż nawet przy końcu procesu nauki aktor powinien decydować się na niewielką liczbę przypadkowych ruchów. Wartość graniczna determinuje minimalną wartość epsilona. W omawianym wypadku reguła ta została zmieniona ze względu na ewidentne problemy ze złapaniem równowagi przez wahadło, nawet przy bardzo rzadkich ruchach losowych. Z tego powodu w naszym algorytmie współczynnik akcji losowych jest zerowany po odbyciu 100 sesji uczących. Współczynnik zaniku epsilona ρ (ang. *Epsilon Decay Rate*) warunkuje tempo jego malenia. Dla prostych problemów powinien on być względnie mały, żeby niepotrzebnie nie eksplorować rozwiązań. Wraz ze wzrostem poziomu złożoności obiektu, gdzie proces nauki jest czasochłonny, współczynnik ten powinien zbliżać się do jedności. Zmiana prawdopodobieństwa z którym wybierane są akcje losowe wyznacza równanie: $\varepsilon = \varepsilon * \rho$.

4.2.3. Nagroda

Właściwe dobranie funkcji nagrody jest kluczem do poprawnego nauczenia agenta. W naszym przypadku powinna ona nagradzać za zbliżenie się kąta położenia wahadła do zadanej wartości (pozycja pionowa wahadła). Drugorzędnym priorytetem jest karanie za dużą prędkość obrotową wahadła, utrudniającą jego ustabilizowanie. Kolejno istotna jest minimalizacja zużycia energii przez silnik. Interpretowane jest to jako wykonywanie jak najmniejszej liczby niezerowych akcji. Proponowane równanie nagrody ma postać:

$$R_t = -\theta_t^2 - 0{,}004(\theta_t')^2 - 0{,}36 \cdot 10^{-3}\tau_{t-1}^2.$$

R_t przyjmuje wartości ujemne, przez co można je interpretować bardziej w kontekście kary, a im bliższe jest wartości równej zero tym lepiej. Zmienna θ jest kątowym położeniem wahadła, jej pochodna θ' prędkością kątową a τ reprezentuje zadaną na silnik wartość momentu siły. Dla poprawnej interpretacji równania kary kluczowe jest, aby kąt położenia wahadła w pozycji odwróconej (pionowej) interpretowany był jako θ równy *0*, a w pozycji dolnej przyjmował wartości *-π* bądź *π*.

4.2.4. Aktualizacja funkcji celu

W algorytmie DQN, zaprezentowanym w rozdziale 3.2.4.6, zawarte jest polecenie, aby sieć celu aktualizowana była periodycznie co określoną liczbę kroków. Nie jest to jednak jedyna stosowana w tym algorytmie metoda uaktualnianie sieci celu. Opracowany został sposób zwany „wygładzającym", dokonujący zmian w każdej sesji uczącej. Jest on preferowany w gotowych narzędziach Matlaba [15]. W tym rozwiązywaniu parametry sieci nie są przepisywane całkowicie, lecz w określonym przez architekta tempie. Szybkość zmiany określa

parametr zwany współczynnikiem wygładzania celu (ang. *Target Smooth Factor*). Równanie opisujące wygładzoną aktualizację parametrów sieci celu ma postać:

$$\varphi' = (1 - \beta)\varphi' + \beta\varphi,$$

gdzie φ' opisuje parametry sieci celu, φ jest parametrem trenowanej sieci Q a β jest współczynnikiem wygładzania celu. Należy dodać, że im współczynnik β jest większy, tym sieć celu aktualizowana jest w szybszym tempie. Dobieranie wspomnianej wartości odbywa się w sposób heurystyczny, gdyż każdy problem zupełnie inaczej reaguje na zmiany. Periodycznie wygładzająca metoda (ang. *Periodic Smoothing*) jest kompromisowym podejściem uwzględniającym wady i zalety każdego z wcześniejszych rozwiązań. W napisanym przez nas programie sieć celu aktualizowana jest właśnie w sposób periodycznie wygładzającym.

4.2.5. Aktualizacja parametrów sieci

Trening sieci odbywa się raz na sesję uczącą, zawierającą 1000 kroków (trwających 25 milisekund każdy). Używana biblioteka umożliwia wybór jednego z siedmiu algorytmów optymalizujących, a ich zadaniem jest znalezienie globalnego minimum funkcji. Do predykcji wyjść sieci używana jest funkcja ‘SimulateNN’. Poprzez jej wywołanie otrzymywane są wyjścia sieci, odpowiadające zadanym wejściom. Po deklaracji metod oraz parametrów uczących wykonywana jest funkcja ‘TrainNN’. Aktualizuje ona wagi sieci tak, aby dopasować wyjścia sieci do zadanych w argumentach wyjść pożądanych (celów), wyznaczonych za pomocą równania Bellmana (podrozdział 3.2.4). Liczba poprawy parametrów za pomocą jednej mini-partii definiowana jest epokami. W poszukiwaniu minimum funkcji straty używany jest optymalizator ADAM, którego parametry wewnętrzne wspomniane w akapicie 3.2.4.5 zainicjalizowane zostały jako domyślne, to jest: $\beta_1 = 0{,}9$; $\beta_2 = 0{,}999$; $\eta = 0{,}01$.

4.3. Symulacja obiektu

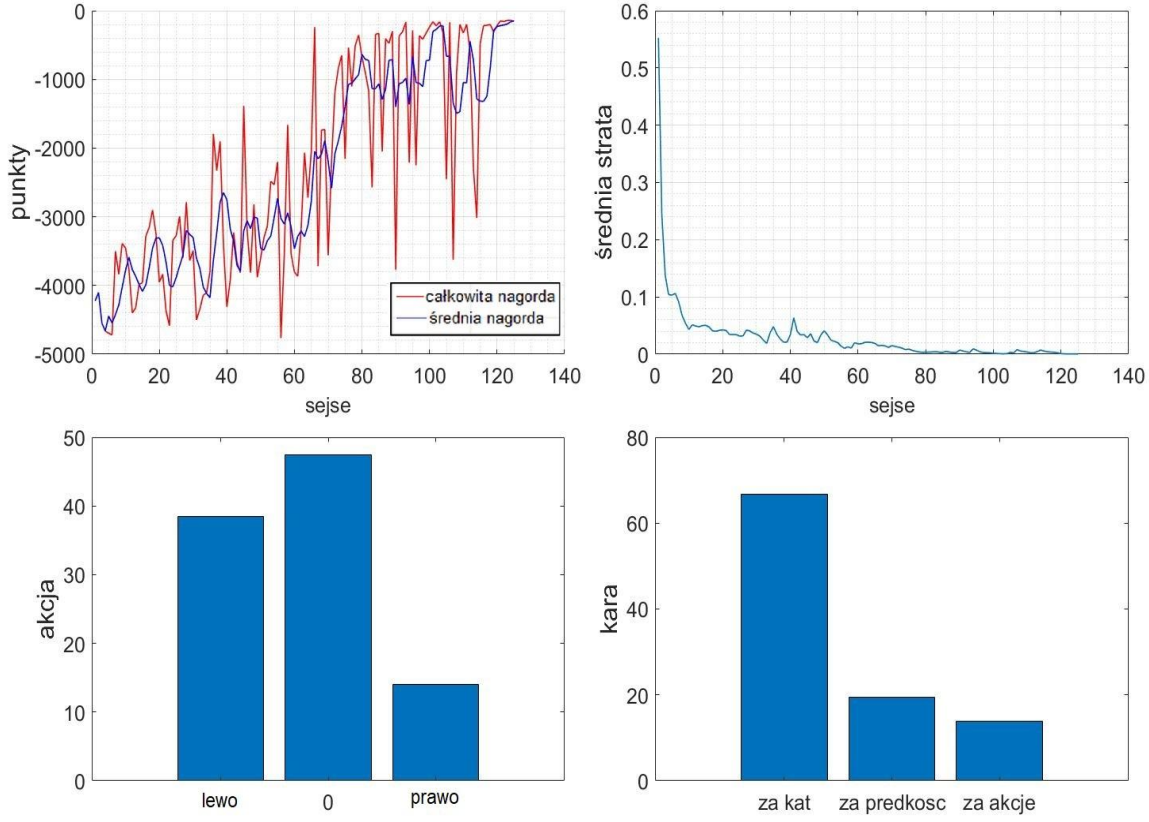
Symulacja wahadła odbyła się na modelu opisanym w podrozdziale 4.1. W pierwszej symulacji założone zostało działanie systemu w środowisku w którym pominięte zostały opóźnienia związane z transportem. Warunek zakończenia symulacji określony został jako zdobycie przez agenta średniej nagrody z pięciu ostatnich sesji, nie mniejszej niż -155 punktów. Taka liczba punktów zapewnia poprawne rozbujanie i stabilizację wahadła w pozycji odwróconej (w pionie). Proces uczenia uruchamiany jest w pliku ‘działanie.m’, gdzie również przypisywane są wartości parametrów uczących. Parametry wymienione w tabeli 1 służą jako parametry sieci bazowej.

Tabela 1 parametrów uczących

minimalny ε	**0,02**
współczynnik zaniku ε, ρ	**0,9995**
współczynnik obniżający γ	**0,9**

Tempo uczenia α	**0,001**
Liczby neuronów w warstwach ukrytych	**48 96**
Rozmiar pamięci doświadczeń D	**6000**
Rozmiar partii	**6000**
Rozmiar mini-partii	**64**
Epoki	**10**
Optymalizator	**ADAM**
Współczynnik wygładzenia celu	**0,4**

W tabeli 1 widzimy, że rozmiar pamięci ustawiony został na 6000, co oznacza przechowywanie 6 ostatnich sesji. Z taką samą wartością określony został rozmiar partii, co wskazuje, że trening odbywa się z wszystkimi próbkami pamięci. Takie ustawienie sprawdziło się najlepiej w zadaniu sterowania zamodelowanego wahadła. Każda mini-partia trenowana jest dziesięciokrotnie, o czym świadczy liczba epok. Liczba neuronów w warstwach ukrytych ustalona została eksperymentalnie (rozdział 4.4).



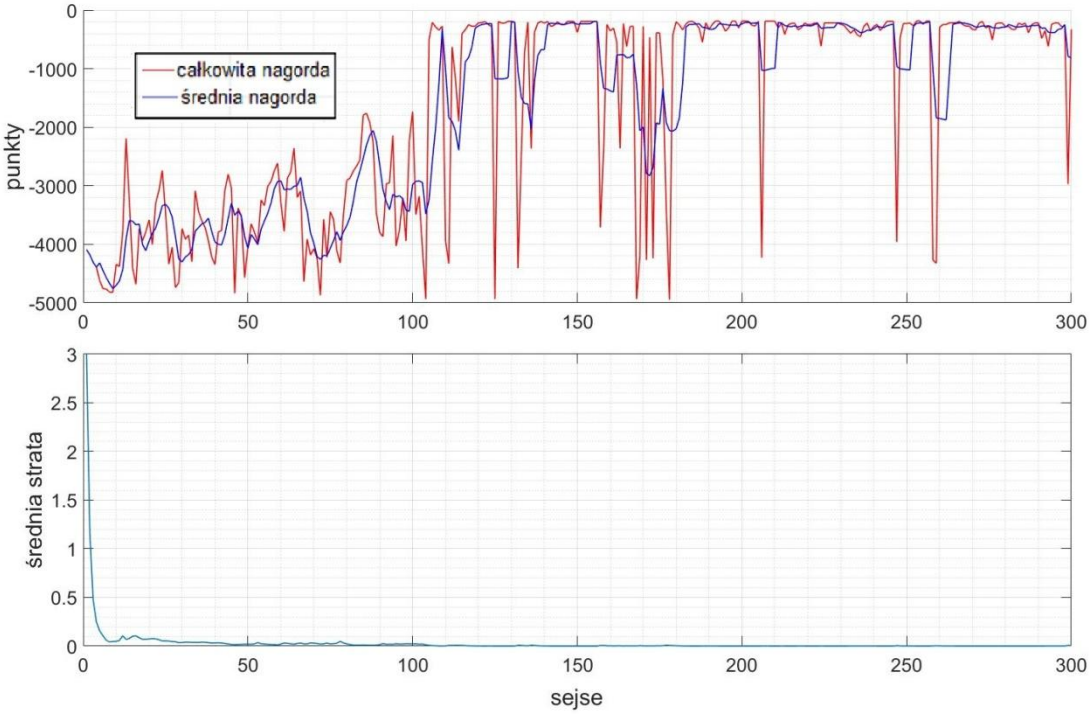
Rys. 4.2 Wykresy wskaźników uczenia

Wszystkie wskaźniki przedstawione na rysunku 4.2 zostały wprowadzone w celu usprawnienia procesu treningu. Na, lewym górnym wykresie przedstawiono wykres całkowitej uzyskanej nagrody. Całkowita nagroda sesji jest skumulowanymi nagrodami wszystkich kroków w danej sesji, zaś średnia nagroda jest średnią arytmetyczną z pięciu ostatnich całkowitych nagród. Algorytm dąży do zminimalizowania otrzymanej kary, czyli do położenia wahadła

w kącie zerowym oraz wyzerowaniu jego prędkości obrotowej. Całkowite wyeliminowanie kary nie jest możliwe ze względu na start wahadła w pozycji swobodnie zwisającej, w której kara równa jest -π^2 punktów za krok. Znaczne spadki punktów w zaawansowanym etapie działania programu spowodowane są efektem nadmiernego rozpędzenia wahadła, które osiąga znaczne prędkości obrotowe. Dzięki znacznie zmniejszonej punktacji, algorytm jednak w miarę postępu działania jest w stanie przeciwdziałać temu zjawisku. Jak widać, uczenie zakończyło się w sesji 125, w której zadane cele dotyczące średniej nagrody zostały osiągnięte. W górnym prawym wykresie przedstawiono (rys. 4.2) wykres średniej funkcji straty dla każdej sesji (punkt 3.2.4.1). Trening sieci dąży do minimalizacji tego wskaźnika, co ostatecznie osiąga. NA lewym dolnym wykresie pokazano procentowy rozkład wyboru poszczególnych akcji w ostatniej sesji. Wizualizacja okazała się pomocna w sytuacji, kiedy sieć trenowana była zbyt małą liczbą doświadczeń lub ze zbyt małym krokiem. Z początku procentowe wybory poszczególnych akcji powinny być zmieniane w bardzo dynamiczny sposób, a jeżeli nie są, to należy zwiększyć parametry trenowania sieci. Na prawym dolnym wykresie przedstawiono procentowy rozkład składowych kary dla ostatniej sesji. Dane to okazały się pomocne podczas doboru oraz skalowania funkcji nagrody. Ustawienie do którego dążymy ma uwzględniać karanie za prędkość i położenie kątowe ramienia 2 w odpowiednich proporcjach. Konieczne jest wyeliminowanie wspomnianego wcześniej zjawiska nadmiernego obracania się wahadła, lecz niezniechęcające algorytmu do jego rozbujania.

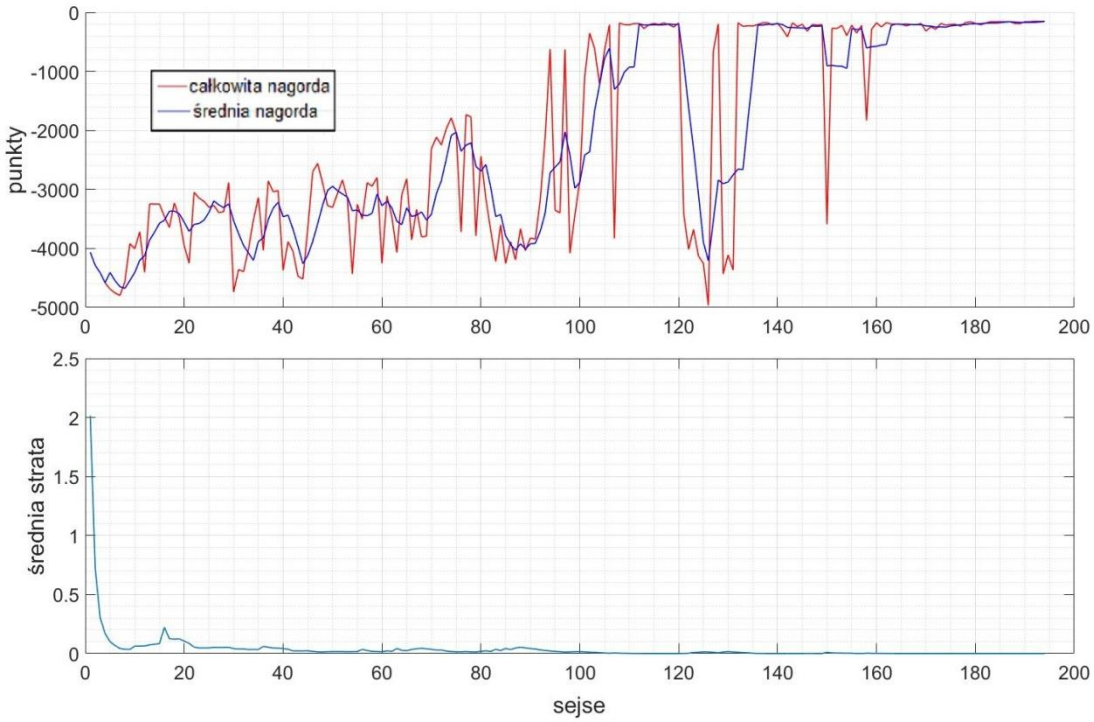
4.4. Dobranie rozmiaru sieci

W celu osiągnięcia najlepszych wyników procesu uczenia, znaczące było dobranie właściwej liczby neuronów w warstwach ukrytych sieci. Nie istnieje powszechnie stosowana zasada mówiąca jaki rozmiar sieci nadaje się do konkretnego problemu, więc należy go dobierać eksperymentalnie. Wiadomo tylko, że do sterowania odwróconym wahadłem najlepiej pasuje sieć z dwiema warstwami ukrytymi. W pracy [8] liczby neuronów warstw ukrytych ustawiane były na wartościach z przedziału (24;64). Tak więc te wartości zostały przyjęte jako punkt wyjściowy. Opracowanie wyników ma za zadanie znalezienie takich rozmiarów warstw ukrytych, dla których algorytm najszybciej jest w stanie osiągnąć zakładany cel, określony jako zdobycie średniej nagrody nie mniejszej niż -155 punktów. W celu uśrednienia wyniku badania, każdy rozmiar sieci badany jest na pięciu próbach. Jako omawianą szerzej próbę wybierano tę, której wynik był najbardziej zbliżony do średniej.



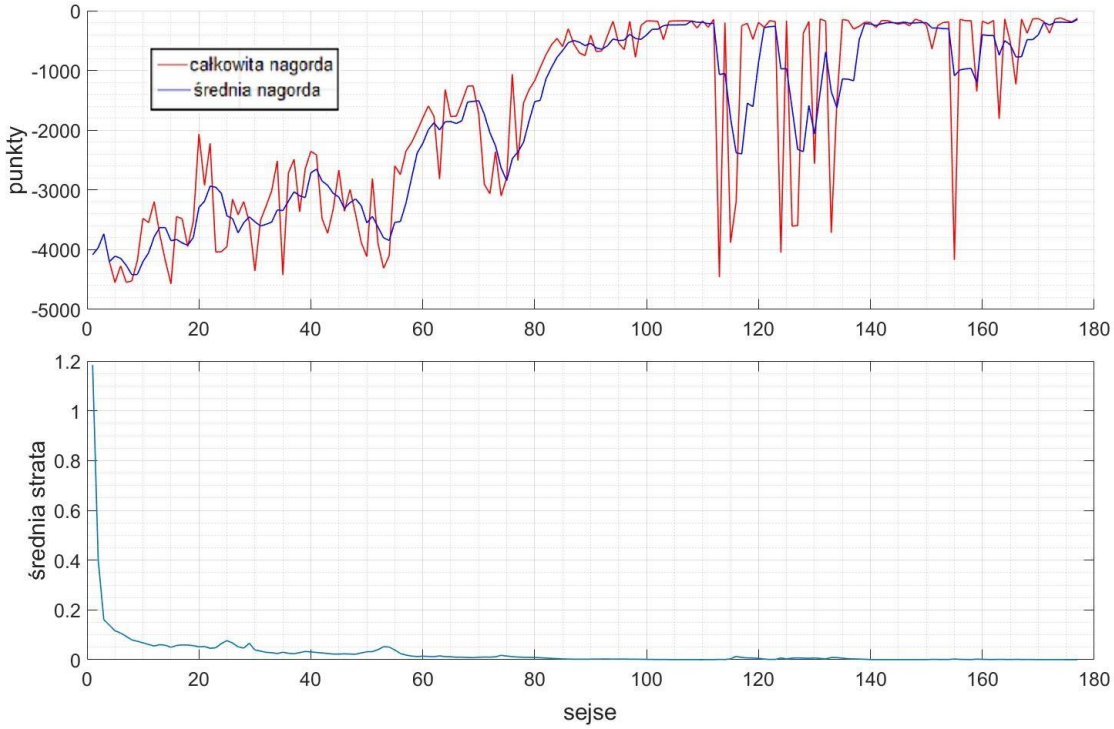
Rys. 4.3 Wykres osiągów sieci 12,24

Jako pierwsza badana była bardzo mała sieć zawierająca kolejno 12 oraz 24 neurony. Maksymalny czas w jakim sieć miała szansę na osiągnięcie celu wynosi 300 sesji. Możliwe jest, że w przypadku dłuższego treningu sieć nauczyłaby się rozbujać oraz poprawnie balansować wahadłem, jednakże taki wynik byłby niesatysfakcjonujący. W danym czasie sieć poprawnie opanowała problem rozbujania wahadła, jednak nie poradziła sobie w zadowalającym stopniu z jego utrzymywaniem w pozycji pionowej odwróconej. O bliskości osiągnięcia zakładanego celu świadczy względnie wysoka maksymalna osiągnięta liczba punktów, wynosząca -210.



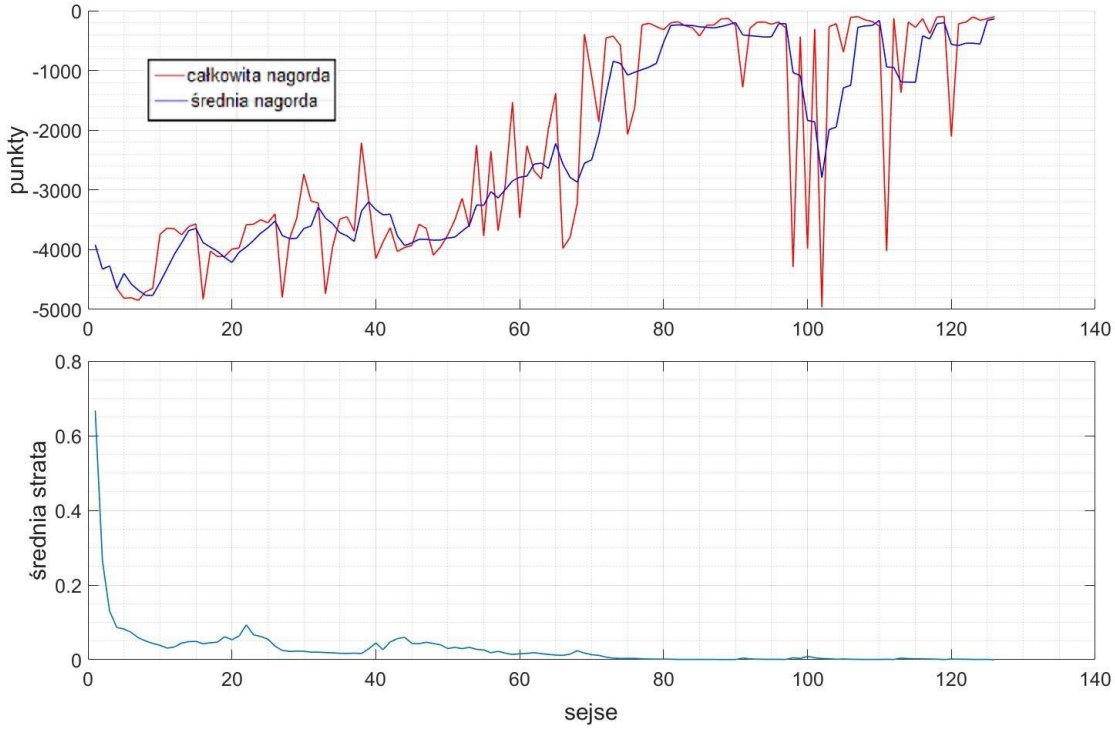
Rys. 4.4 Wykres osiągów sieci 24,48

Kolejno testowana była sieć o rozmiarze warstw wynoszącym kolejno 24 oraz 48. W przeglądanej przez nas literaturze jest to najczęściej spotykany rozmiar sieci stosowany w sterowaniu odwróconym wahadłem. W przeprowadzonym teście sieć o takim rozmiarze była w stanie osiągnąć założony cel w 80% badanych przypadków. Taki wynik nie jest zły, jednakże pożądane jest rozwiązanie gwarantujące pewność w osiągnięciu celu. Pokazany na rysunku 4.4 wykres jest średnim otrzymanym rozwiązaniem, uzyskanym używając sieci o omawianym rozmiarze. Osiągnięcie celu nastąpiło w 194 sesji.



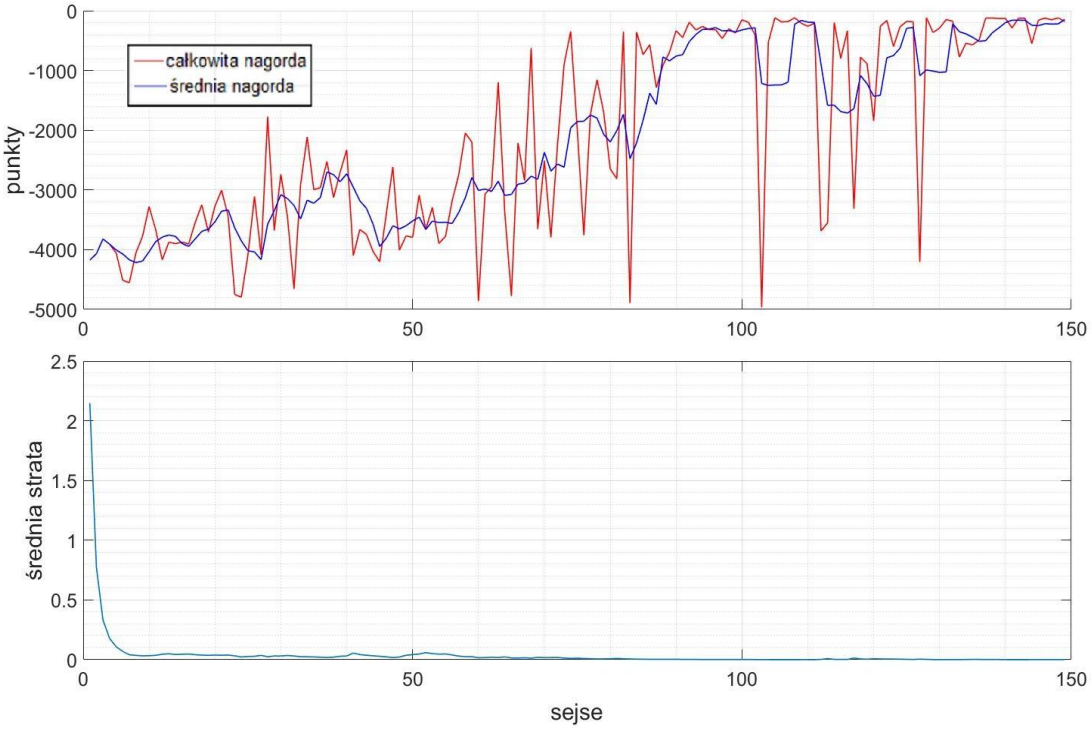
Rys. 4.5 Wykresy osiągów sieci 32,64

Tym razem wielkości warstw ukrytych ustawione zostały na 32 oraz 64 neurony. W próbie, której przebieg zobrazowano na rysunku 4.5, algorytm osiągnął zakładaną nagrodę w 177 sesjach, co jest akceptowalnym wynikiem.



Rys. 4.6 Wykresy osiągów sieci 48,96

Sieć o rozmiarze 48 oraz 96 neuronów, której sesja ucząca udokumentowana została na rysunku 4.6 poradziła sobie najlepiej z problemem sterowania odwróconego wahadła. Nauczenie w stopniu spełniającym założenie nastąpiło tutaj w 126 sesji. Z tego powodu została wybrana jako sieć bazowa, która szczegółowo opisana została w podrozdziale 4.3.



Rys. 4.7 Wykresy osiągów sieci 64,128

Rozwiązanie otrzymane przy użyciu sieci ustawionej na rozmiar 64 oraz 128 neuronów okazało się również bardzo dobre. Rezultaty były bardzo obiecujące poza jednorazowym zdarzeniem, w którym sieć nie osiągnęła założonej liczby punktów. W badaniu przeprowadzonym w tym podrozdziale najważniejszym czynnikiem jest niezawodność oraz najkrótszy średni czas w jakim algorytm zakończył proces nauki, przez co rozmiar sieci 64 oraz 128 neuronów musiał ustąpić pierwszeństwa rozmiarowi 48 oraz 96 neuronów.

Tabela 2 Zestawienie wyników badania adekwatnego rozmiaru sieci

Liczby neuronów w warstwach ukrytych	Próba 1	Próba 2	Próba 3	Próba 4	Próba 5
12 , 24	232	300+	300+	230	300+
24 , 48	177	300+	202	163	121
32 , 64	188	200	187	300+	177
48 , 96	119	154	125	114	126
64 , 128	149	300+	123	149	135

Z danych zawartych w tabeli 2 można wyciągnąć wniosek, że najbardziej adekwatnym rozmiarem sieci do sterowania modelem odwróconego obrotowego wahadła (punkt 4.1.1) jest 48 neuronów w pierwszej oraz 96 neuronów w drugiej warstwie ukrytej. Jest to jedyny rozmiar, który w każdej z badanych sesji osiągnął zakładany cel. Komórka z zawartością ‘300+’ oznacza brak osiągnięcia celu w maksymalnym założonym czasie, równym 300 sesji. Każdy eksperymentalnie sprawdzany rozmiar sieci zawiera podwojoną liczbę neuronów w drugiej

warstwie ukrytej w porównaniu z warstwą pierwszą. Gdyby założony na początku eksperymentu cel był łatwiejszy do osiągnięcia, wiele z powyżej zestawionych sieci bezproblemowo podołałoby problemowi. Celem było dobranie rozmiaru sieci, z którym algorytm konsekwentnie osiąga wymaganą liczbę punktów w optymalnym czasie.

4.5. Symulacja z uwzględnieniem opóźnień

W tym podrozdziale opisana jest symulacja, która wzbogacona została o bloki opóźniające. Sieć używana w tym badaniu była siecią bazową opisaną w podrozdziale 4.3. Początkowo nadawane opóźnienia dołączano do wyjść i wejść zamodelowanego obiektu. W takiej sytuacji proces uczenia nie osiągał zakładanego celu pomimo względnie małych opóźnień, rzędu kilku milisekund. W celu identyfikacji problemu podawano bardzo rozbieżne wartości opóźnień na wejście oraz wyjście obiektu. Zaobserwowano, że niezależnie od wielkości opóźnienia sygnałów wychodzących z wahadła (kąta oraz prędkości kątowej), wahadło nie było w stanie poprawnie balansować. Prawdopodobną przyczyną jest fakt, że jakakolwiek wartość opóźnienia na wyjściu wahadła powoduje pominięcie jednej próbki wyjściowej, a w konsekwencji nieadekwatne dopasowanie akcji do reakcji w pamięci doświadczeń. Spowodowane jest to działaniem programu symulującego, który nie przewiduje takich sytuacji. Badanie musiało odbyć się więc za pomocą ujednolicenia wszystkich czynników opóźniających i ich implementacji w postaci jednego bloku umieszczonego na sygnale wejściowym do obiektu (czyli akcji). Kolejno testowane było wiele wartości opóźniających, a wyniki tego badania zaprezentowane zostały w tabeli 3. Wartość średnia liczby sesji obliczana była na podstawie pięciu kolejnych uruchomień.

Tabela 3 Wpływ opóźnienia na czas uczenia

Opóźnienie sygnału sterującego [ms]:	Średnia liczba sesji, w których wahadło osiągnęło cel:
0	128
5	142
10	149
15	175
20	161
25	300 (Brak osiągnięcia zakładanego celu)

Pierwszym nasuwającym się wnioskiem jest fakt, że w miarę wzrostu opóźnienia transportowego, coraz bardziej wydłużany jest czas uczenia. Wyniki przedstawione w tabeli 3 nie zawsze jednak wykazują taką zależność. Pojedynczy wyjątek (20 ms) spowodowany jest dużym wpływem losowości na wynik każdego treningu. Jak widać, przełomowa była sytuacja, gdzie opóźnienie wyniosło 25 ms. Była to wartość dla której wahadło nie osiągnęło zakładanego celu. Obserwacja wspomnianej próby pozwoliła określić, że agentowi niewiele brakowało aby osiągnąć wymagane punkty. Sprawnie przebiegł proces rozbujania oraz dopracowany

on został w kierunku osiągnięcia przez wahadło niskiej prędkości obrotowej w pozycji odwróconej. Sterownik potrafił nawet przez pewien moment balansować w pozycji odwróconej pionowej. Zbyt opóźnione polecenia sterujące uniemożliwiły całkowitą stabilizację. Graniczna wartość opóźnienia, umożliwiająca skuteczne sterowanie, mieści się między 20 a 25 milisekund. Górną granicą jest również czas trwania pojedynczego kroku symulacji. Wynika z tego, że w opisywanym przypadku sterowania modelem odwróconego wahadła, opóźnienia nie mogą być większe bądź równe czasowi wykonywania pojedynczej akcji. Opóźnienie wykraczające poza opisaną granicę powoduje zbyt nieadekwatne dopasowanie akcji sterującej do obecnego stanu obiektu.

4.6. Implementacja priorytetowej pamięci doświadczeń

Opracowano kilka sposobów na usprawnienie procesu uczenia algorytmu DQN. Jednym z nich jest dodanie wag do losowanych z pamięci doświadczeń rekordów. W teorii uczenia maszynowego wybieranie próbek w taki sposób jest nazywane priorytetową pamięcią doświadczeń (ang. *Prioritized Experience Replay*). W eksperymencie używano bazowej sieci opisanej w podpunkcie 4.3. Dotychczas przeprowadzane treningi uczyły sieć wszystkimi dostępnymi próbkami z pamięci doświadczeń, zawierającej 6000 rekordów. Wprowadzając wspomniane ulepszenie, wyliczane jest prawdopodobieństwo na wybranie każdej próbki. Szansa ta jest proporcjonalna do policzonego dla przejścia błędu, będącego pierwiastkiem z wartości funkcji straty (punkt 3.2.4.1). Dzięki temu próbka zawierająca bardziej wartościowe dla sieci dane zostanie częściej wybrana. Żeby algorytm losujący miał z czego wybierać, maksymalna zawartość rekordów w pamięci doświadczeń zwiększona została dwukrotnie na 12000.

Tabela 4 Wybór próbek uczących za pomocą priorytetowej pamięci doświadczeń

Próba:	1	2	3	4	5
Sesje:	107	137	160	134	159

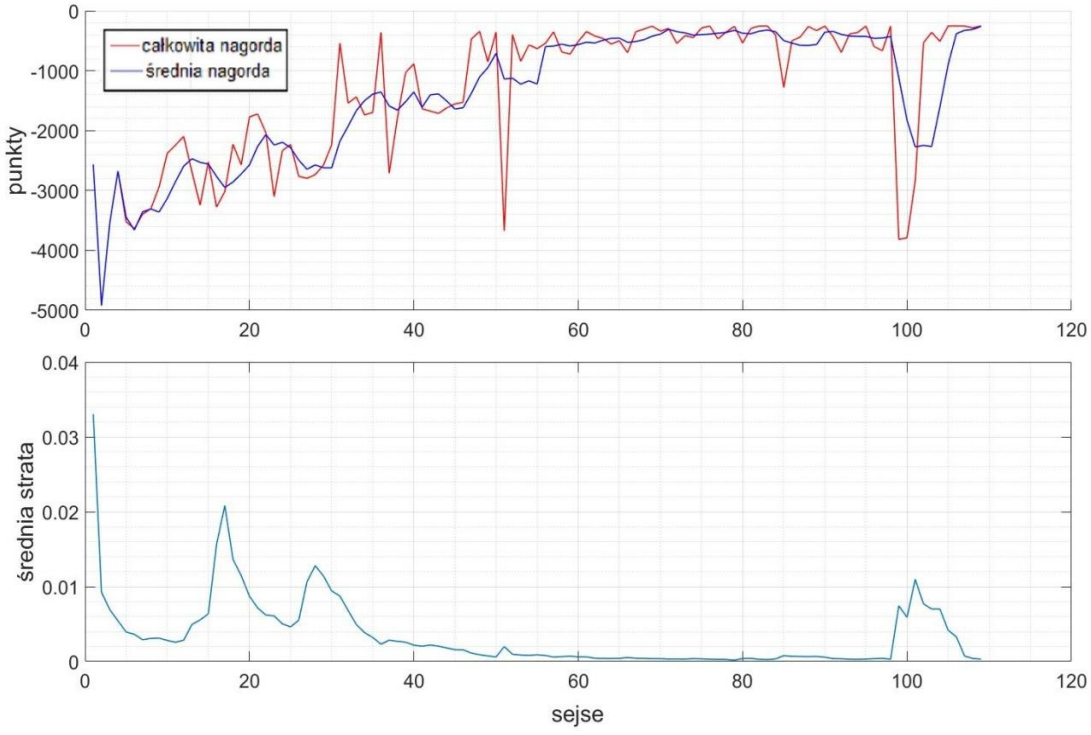
Schemat działania priorytetowej pamięci doświadczeń szerzej opisany jest w pracy [11]. Jednym z kroków algorytmu jest zmiana sposobu obliczania poprawy wag sieci. Przyjmując jednak jeden ze współczynników jako zerowy, można pominąć krok mający na celu zniwelowanie zjawiska nadmiernego dopasowania. Ulepszenie zastosowane w badaniu, którego wyniki prezentuje tabela 4, jest najprostszą wersją PER. Średnia liczba sesji uzyskana z pięciu kolejnych prób wynosi 139. Jest to wynik gorszy, lecz bardzo zbliżony do oryginalnie stosowanego rozwiązania. Istnieje wiele powszechnie stosowanych ulepszeń algorytmu DQN. Nie ma gwarancji, że po zastosowaniu każdego z nich wyniki rozwiązania problemu ulegną polepszeniu. W badaniu rozważanym w tym podrozdziale nie uzyskaliśmy lepszych rezultatów. możliwą przyczyną może być zastosowanie najprostszej formy priorytetowej pamięci

doświadczeń. Niewykluczone jest, że gdyby parametry były lepiej dostrojone a aktualizacja wag w odpowiedni sposób skalowana to uzyskane rezultaty byłyby lepsze.

4.7. Adaptacyjność, odporność sterownika i wpływ zakłóceń

Jednym z założeń pracy było stworzenie adaptacyjnego sterownika. Z początku należy rozwiać wątpliwości związane z pojęciami adaptacyjności oraz odporności kontrolera. Odporność oznacza tolerancję dla zmian parametrów obiektu. Sterowanie odporne gwarantuje, że jeśli zmiany mieszczą się w pewnych granicach, to nie trzeba zmieniać przyjętej zasady sterowania. W kontekście adaptacyjności mamy do czynienia z aktualizującym swoje parametry sterownikiem. W ten sposób dopasowuje się do zmieniającego swoje właściwości obiektu.

Adaptacyjność sterownika zaprojektowanego na potrzeby tej pracy sprawdzona została poprzez wprowadzenie zmian w parametrach fizycznych wahadła. Po ukończeniu przez sieć procesu uczącego na opracowanym modelu matematycznym (punkt 4.1.1), zwiększono dwukrotnie masę oraz długość ramienia 2 wahadła. Jest to duża zmiana parametrów obiektu wykraczająca poza zakres odporności sterownika. Kolejno wznowiono proces uczący na tak zmodyfikowanym obiekcie.



Rys. 4.8 Wyniki wznowienia procesu uczącego na zmienionym obiekcie

Jak pokazano na rysunku 4.8, algorytm zakończył swoje działanie w 109 sesji. Wyniki wskazują, że algorytm potrafił dostosować się do znaczących zmian obiektu. Ponadto opanował sterowanie nim o wiele szybciej, niż gdyby rozpoczynał proces uczący od początku.

Adaptacyjność jest własnością powszechne znaną wśród sterowników, których działanie oparte jest na sieci neuronowej. W końcu każdy z nich poprzez proces nauki przeprowadza swego rodzaju identyfikacje obiektu.

Następnie sprawdzona została odporność sterownika. Badanie przeprowadzone zostało w następujący sposób. Zmiany właściwości fizycznych wahadła rosły tak długo, aż sterownik nie będzie w stanie dłużej poprawnie stabilizować wahadła. Jako przykład zmian właściwości obiektu wybrano równoczesne, procentowe zwiększenie masy oraz długości ramienia 2. Do badania użyto dwóch bazowych sieci, opisanych w podrozdziale 4.3. Wyróżniała je jedynie długość przeprowadzonego treningu. Pierwsza sieć trenowana była do momentu spełnienia minimalnych wymagań poprawnej stabilizacji, określonej w 4.3. Uczenie to trwało 130 sesji. Druga sieć trenowana była aż przez 300 sesji (w celu maksymalizacji osiągów). Jako obiekt sterowania użyto modelu zaproponowanego w podrozdziale 4.1.1. Pierwsza sieć potrafiła poprawnie stabilizować wahadło, którego zmieniane wielkości fizyczne maksymalnie wzrosły o 40%. Sieć druga uzyskała wynik gorszy, gdyż równy zaledwie 19%.

Na końcu sprawdzony został wpływ zakłóceń na wahadło. W tym badaniu użyto również dwóch sieci, identycznych jak w poprzednim akapicie. Zakłóceniem w naszym przypadku był dodatkowy moment obrotowy nadany na ramię 2. Miało to imitować destabilizację ramienia 2 (np. pchnięciem) podczas jego pracy. Zakłócenie to trwało 2,5 milisekundy i podawane było periodycznie co 5 sekund. Eksperymentalnie sprawdzono, jak dużą amplitudę sygnału zakłócenia sterownik jest w stanie skompensować by nie ulec całkowitej destabilizacji. Dla sieci pierwszej (130 sesji), maksymalna wartość amplitudy zakłóceń wyniosła $25 \cdot 10^{-3}\, Nm$. Druga sieć (300 sesji) potrafiła utrzymać wahadło przy maksymalnej amplitudzie zakłócenia równej $65 \cdot 10^{-3}\, Nm$.

Na podstawie powyższych rozważań można wyciągnąć kilka kluczowych wniosków. Co może wydawać się nie intuicyjne, sieć druga poradziła sobie lepiej z zakłóceniami, zaś gorzej w momencie testowania odporności. Poprzez dłuższy trening sieć coraz bardziej dostraja się do kontrolowanego obiektu, polepszając swoje osiągi. Druga sieć precyzyjniej reaguje na aktualny stan wahadła, a jest to jedyna rzecz ulegająca zmianie pod wpływem zewnętrznych zakłóceń. Z tego powodu sieć druga lepiej radzi sobie z zakłóceniami. Sieć pierwsza nie jest ściśle zoptymalizowana pod kontrolowany obiekt. Jej działanie jest bardziej uniwersalne, co czyni ją bardziej odporną.

5. BUDOWA WAHADŁA (NORMAN DRYŚ)

Budując rzeczywiste wahadło kluczowym jest odpowiednie zaprojektowanie konstrukcji oraz dobór właściwych elementów wykonawczych. Powinniśmy również zminimalizować wpływ zakłóceń oraz dodatkowych nieliniowości. Obiekt sterowania powinien być również, w miarę

możliwości, systemem powtarzalnym i przewidywalnym. W tym rozdziale przedstawione zostaną wybrane, optymalne rozwiązania spełniające powyższe założenia.

Rys. 5.1 Rzeczywiste wahadło

5.1. Konstrukcja mechaniczna

Budując rzeczywiste wahadło przede wszystkim należy zapewnić stabilność konstrukcji. Niesymetrycznie rozłożona masa ramienia 1 obiektu może wprowadzać konstrukcję w drgania, a nawet rezonans skutkujący przewróceniem się urządzenia. Ważne jest również zminimalizowanie wszelkich luzów występujących w wahadle. Jest to nieprzewidziane zjawisko mające duży wpływ na zachowanie się wahadła w trakcie jego działania. Wszystkie te zjawiska wprowadzają zakłócenia i dodatkowe nieliniowości, powodując że zbudowane urządzenie przestaje być deterministyczne.

5.1.1. Stabilność konstrukcji

Pierwszym, najważniejszym aspektem jest stabilność samej konstrukcji mechanicznej. Stabilność można poprawić na kilka sposobów:

- zwiększenie masy podstawy wahadła,
- zwiększenie średnicy podstawy wahadła,
- zrównoważenie masy wirującej,
- obniżenie konstrukcji,
- wykonanie zwartej, mniej sprężystej konstrukcji.

Pierwszy i drugi podpunkt zrealizowany został dzięki zastosowaniu szerokiego, żeliwnego obciążenia o wadze dwóch kilogramów. Największym czynnikiem, niemal całkowicie niwelującym drgania konstrukcji, było zrównoważenie masy obrotowego ramienia 1. Prototyp wahadła posiadał jedno łożysko, zamontowane na ramieniu 1 przyjmującym kształt litery „L”. W takiej konstrukcji punkt środka masy ramienia 1 znajduje się względnie daleko od jego osi obrotu. W finalnej wersji zastosowane zostały dwa łożyska, zamocowane na ramieniu 1 posiadającym kształt litery „U”. Taka konstrukcja zapewniła symetryczność względem osi obrotu ramienia 1, równoważąc rozkład masy. Wysokość konstrukcji została ustalona na minimalną, jednocześnie zapewniającą swobodny ruch ramienia 2. Całość została wykonana ze stali i stopów aluminium tworząc stabilną, mało sprężystą konstrukcję. W celu zminimalizowania wpływu niezrównoważonej masy, ramię 1 w głównej mierze zostało zbudowane z aluminiowych elementów. Statyczna część konstrukcji wykonana została ze stalowych elementów. Poprawiło to stosunek masy podstawy do masy ruchomych ramion, tym samym zwiększając stabilność całego wahadła.

5.1.2. Minimalizacja zakłóceń i dodatkowych nieliniowości

Wiadomym jest, że niektórych niedoskonałości wahadła nie można zniwelować. Nie ma pewności, czy pomimo wielu utrudnień, poprawne sterowanie obiektem jest wykonalne. Z tego powodu należało zminimalizować zakłócenia oraz dodatkowe nieliniowości, powstające

na skutek budowy mechanicznej i pracy elementów wykonawczych. Następnie, dla tak zoptymalizowanego modelu rzeczywistego, sprawdzono skuteczność sterowania. Wspomniana wcześniej zmiana konstrukcji ramienia 1 na symetryczną i lekką poprawiła nie tylko stabilność wahadła. Poprawie uległa również responsywność silnika z uwagi na znacznie mniejszy moment bezwładności ramienia 1. Umożliwiło to sterowanie z większą częstotliwością. Łożyska niskooporowe umożliwiają swobodne bujanie ramienia 2. Użyty wewnątrz łożyska rzadki smar minimalizuje opory toczenia. Zastosowanie dwóch łożysk jako punktów podparcia dla osi ramienia 1 pozwoliło na ich równomierne obciążenie. Zmniejszyło to znacząco opory toczenia w porównaniu z zastosowaniem jednego, niesymetrycznie zamontowanego łożyska. Stosując dwa punkty podparcia osi ramienia 1, minimalizujemy naprężenia działające na łożyska, wpływając na ich prawidłowe działanie. Rozwiązanie to miało największy wpływ na zniwelowanie luzów występujących między ośką ramienia 1 a łożyskiem. Luzy w łożyskach były głównym czynnikiem utrudniającym sterowanie wahadłem. Trudno przewidzieć, czy bez minimalizacji omawianej histerezy, bylibyśmy w stanie prawidłowo stabilizować model. Wszelkie tarcia w modelu są niepożądane ze względu na wprowadzaną dodatkową nieliniowość. Jako ramię 2 zastosowano, stosunkowo ciężki jak na swoje wymiary, stalowy pręcik. Zmniejsza to wpływ oporu powietrza na ruch wahadła, tym samym ułatwiając jego kontrolowanie.

5.2. Zastosowana elektronika i silnik

Dobór elementów elektronicznych ma duży wpływ na jakość sterowania wahadłem. Elementy powinny zapewnić pożądaną dokładność pomiaru kąta wahadła jak i wykonywać odpowiednie obliczenia z wymaganą prędkością i precyzją. Konieczne było znalezienie kompromisu między jakością zakupionych podzespołów a ich ceną.

5.2.1. Silnik

Bardzo ważny jest dobór odpowiedniego silnika. Od szybkości reakcji silnika jak i od jego innych parametrów w głównej mierze będzie zależeć jakość sterowania. Wybrany silnik powinien mieć jak najmniejszą induktancję. Powinien również posiadać wystarczającą moc, by był w stanie obracać ramię 1 z zadanym momentem obrotowym. W projekcie zastosowano silnik typu BLDC, ponieważ umożliwia precyzyjne sterowanie. Z uwagi na zastosowanie elektronicznego komutatora są one bardziej niezawodne i trwalsze niż tradycyjne silniki szczotkowe. Mniejsza waga i wielkość w stosunku do silników szczotkowych o tej samej mocy przekłada się na bardziej kompaktową konstrukcję. Wybrany model to T-motor GB-54-1. Posiada pożądaną, niską stałą prędkościową KV o wartości 33 rpm/V. Generuje maksymalny moment obrotowy o wartości 0,27 Nm, przy zasilaniu napięciem 22 V. Jest to silnik trójfazowy. Atutem silników BLDC jest fakt, że mogą być one skonstruowane z otworem przechodzącym przez oś obrotu silnika. Był to główny powód dla którego wybrany został

właśnie ten model. Otwór pozwala na bezproblemowe przepuszczenie przez niego przewodów łączących zamontowany na ramieniu 1 enkoder z minikomputerem. Żeby zapobiec skręcaniu się przewodów przy wielokrotnych obrotach silnika, w otworze zamontowano elektryczne złącze obrotowe.

5.2.2. Mikrokontroler / mikrokomputer

Mikrokomputer jest główną jednostką zarządzającą systemem sterowania. Podstawowym jej zadaniem jest akwizycja danych z enkodera oraz generowanie odpowiednich sygnałów sterujących silnikiem. Koniecznym jest, by system musi działać w czasie rzeczywistym ze stałą częstotliwością. Pierwszym wybranym do tego celu urządzeniem było Arduino Uno. Z początku naszą ideą było wykonywanie wszystkich obliczeń algorytmu uczenia maszynowego na komputerze. Pojedyncza pętla układu sterowania wyglądała następująco:

1. Pomiar wychylenia kątowego ramienia 2 przez enkoder.
2. Arduino odbiera pomiar przy użyciu interfejsu I2C i przesyła go do komputera za pomocą USB.
3. Komputer po przetworzeniu pomiaru wybiera akcję sterującą.
4. Sygnał sterujący przesyłany jest z powrotem do Arduino.
5. Arduino, po przetworzeniu sygnału sterującego na odpowiedni sygnał napięciowy, przesyła go do sterownika silnika.
6. Sterownik silnika zamienia sygnał napięciowy na odpowiednio wzmocniony sygnał trójfazowy i bezpośrednio steruje silnikiem.
7. Powrót do punktu 1.

Cykl odbywał się z częstotliwością 20 Hz. W celu zniwelowania opóźnień transportowych postanowiono sterować obiektem bezpośrednio przez mikrosterownik. Arduino Uno nie było odpowiednim urządzeniem do wyliczania w czasie rzeczywistym przejść sieci neuronowej. Rozwiązaniem okazał się zakup minikomputera Raspberry Pi Zero W – 32 bitowego komputera jednopłytkowego oferującego bardzo dużą moc obliczeniową jak na swoje wymiary i cenę. Minikomputer Raspberry Pi bezpośrednio zbiera dane, przetwarza je i generuje odpowiedni sygnał sterujący. Wykluczenie roli komputera z pojedynczej pętli układu sterowania niweluje opóźnienia transportowe do minimum.

5.2.3. Sterownik silnika

Wybrany kontroler silnika - MIKROE-2766 jest modułem ‘All-In-one’. Jest to płytka drukowana zawierająca układ sterownika z zintegrowanymi tranzystorami mocy typu MOSFET. Układ posiada szerokie możliwości konfiguracyjne dostępne poprzez wbudowaną pamięć EEPROM. System umożliwia również sterowanie silnikiem BLDC w sposób bezczujnikowy. Warto wymienić kluczowe możliwości sterownika:

1. IPD - system wykrywający położenie początkowe wirnika względem cewek. Działa poprzez sekwencyjne generowanie pulsującego prądu i pośrednim pomiarze indukcyjności na uzwojeniach. Pozycja Silnika BLDC musi być znana do jego prawidłowego uruchomienia. System ten umożliwia pracę bez czujnika pozycji wirnika;
2. praca w pętli zamkniętej dzięki estymacji BEMF.

5.2.4. Enkoder

Najmniejsza odchyłka wychylenia wahadła od pionu skutkuje jego natychmiastową destabilizacją. Ze względu na posiadaną wysoką rozdzielczość pomiaru wybrany został enkoder magnetyczny. Wyższość nad czujnikami optycznymi zapewniana jest przez lepszą dokładność odczytów, niższą cenę oraz mniejsze rozmiary. Enkodery magnetyczne są enkoderami absolutnymi, co ułatwia interpretację danych pomiarowych. Dużą ich zaletą jest realizacja pomiaru w sposób bezinwazyjny. Brak ingerencji w oskę ramienia 1 nie zwiększa niepotrzebnie oporów obrotu oski. Zastosowany model to AS5600 firmy AMS. Enkoder dostępny był wraz z gotową płytką drukowaną i wyprowadzonymi złączami. Jest to 12-bitowy enkoder, co przekłada się na dokładność pomiaru wynoszącą około 0,1 stopnia. Układ AS5600 posiada również szerokie możliwości konfiguracyjne jak i post-processingową filtrację pomiaru.

5.2.5. Zasilanie

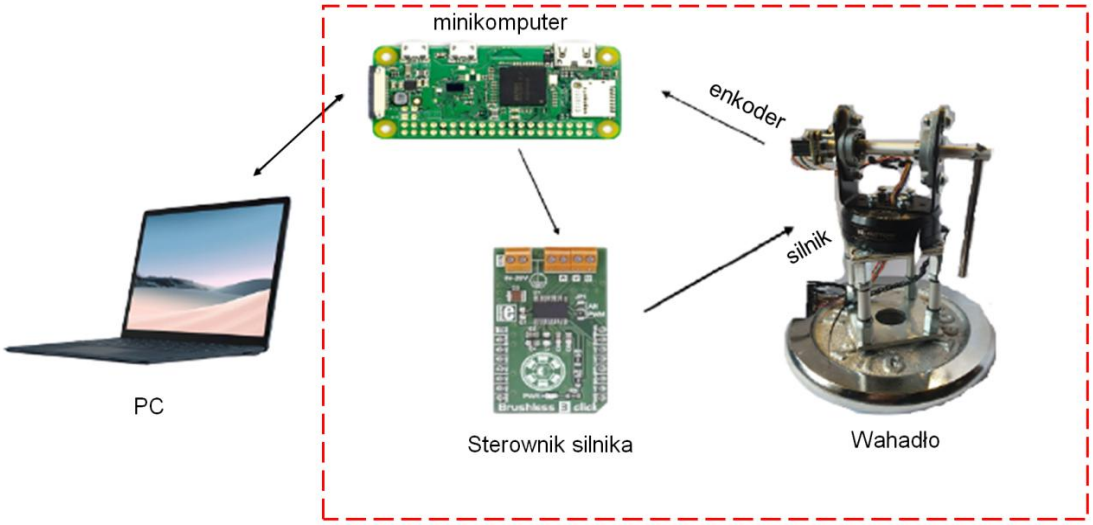
Głównym elementem wymagającym oddzielnego zasilania jest sterownik silnika. Odpowiednią moc dostarcza zasilacz laboratoryjny MPS-3005LK-3 firmy Matrix. Dzięki potencjometrom możliwe jest dokładne testowanie i dopasowanie napięcia zasilania z zakresu 0-30 V, jak i obciążenia prądowego do 3 A. Ograniczenia prądowe skutecznie zmniejszają negatywne skutki pomyłek podczas łączeń komponentów. Dzięki wbudowanej przetwornicy step-down, układ sterownika silnika nie wymaga oddzielnego zasilania. Mikrokomputer pobiera energię z zasilacza USB o mocy 15 W. Enkoder jest zasilany napięciem 3.3 V wyprowadzonym z Raspberry Pi.

6. STEROWANIE RZECZYWISTYM OBIEKTEM (NORMAN DRYŚ)

W tym rozdziale przedstawiony został proces implementacji algorytmu sterującego (opisanego w rozdziale 4) na rzeczywistym obiekcie odwróconego wahadła. Pojawiło się wiele problemów, które dokładnie opisano w dalszej części pracy. Rzeczywiste wahadło posiada dodatkowe nieliniowości, pominięte podczas procesu symulacyjnego. Zdając sobie sprawę z uproszczenia symulacji, wiadome było, że proces nauki na fizycznym obiekcie będzie bardziej złożony.

6.1. Układ sterowania

Podrozdział ten ma na celu przedstawienie układu sterowania rzeczywistego obiektu.



Rys. 6.1 Schemat układu sterowania

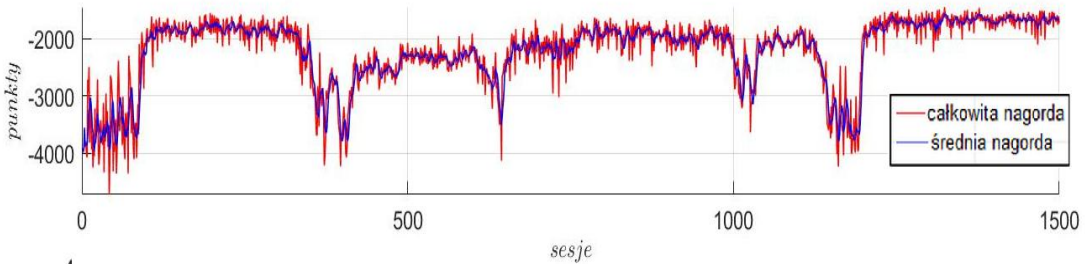
Schemat przedstawiony na rysunku 6.1 jest fizyczną implementacją omówionego wcześniej modelu MDP (punkt 3.2.1). Porównując rysunek 6.1 z rysunkiem 3.4 można zauważyć analogię schematu działania. Jedyną zmianą jest dualna forma agenta w postaci PC oraz minikomputera. Środowiskiem jest model rzeczywistego wahadła. Zgodnie z algorytmem MDP proces zaczyna się odebraniem przez agenta informacji o aktualnym stanie środowiska w chwili czasu t. Informacja o aktualnym stanie wahadła zbierana jest poprzez enkoder. Następnie mikrokomputer, przetwarzając dane stanu wahadła, generuje odpowiedni sygnał sterujący zwany również akcją. Decyzja podejmowana jest na podstawie wyliczenia pojedynczego przejścia przez naszą sieć neuronową. Operacja ta wykonywana jest bezpośrednio na mikrokomputerze. Kolejno sygnał zawierający informację o podjętej decyzji zostaje przekazany do sterownika silnika. Sterownik odpowiednio interpretuje sygnał sterujący i przetwarza go na użyteczny, trójfazowy sygnał o odpowiednim wzmocnieniu. Sygnał ten jest podawany bezpośrednio na fazy zasilające silnik. W ten sposób pętla sterowania zamyka się, a cały proces powtarza się 500 razy w ciągu jednej sesji uczącej. Omówione wyżej pojedyncze przejście pętli sterowania zakreślone zostało na rysunku 6.1 czerwoną linią. Jest to część „Online" całego algorytmu sterowania. Celem tej części jest zbieranie doświadczeń i magazynowanie ich podczas całej sesji treningowej w minikomputerze. Po zakończeniu sesji treningowej wahadło zatrzymuje się, a minikomputer przesyła wszystkie zapisane dane stanowe do PC. Jest to część „Offline" programu. Komputer przeprowadza wymagające obliczenia ucząc sieć neuronową. Tak zmodyfikowana sieć wgrywana jest z powrotem do minikomputera. Z uwagi na problemy sprzętowe (punkt 6.3) częstotliwość sterowania wynosi 20 Hz zamiast symulacyjnych 40 Hz.

6.2. Redukcja opóźnienia komunikacyjnego

Komputerowo symulowany system sterowania początkowo uczony był z pominięciem opóźnień. W celu urzeczywistnienia warunków symulacji, przetestowany został wpływ opóźnienia na proces nauki (punkt 4.5). Sprawdzono maksymalne opóźnienie przy którym układ w dalszym ciągu działał poprawnie. Opóźnienia zostały ujednolicone i umieszczone w jednym miejscu. Symulacja wykazała, że suma opóźnień działań poszczególnych elementów nie może przekroczyć czasu wykonania pojedynczego kroku sterującego.

6.2.1. Komunikacja USB

Początkowo używany mikrosterownik służył głównie jako komunikator między urządzeniami, gdyż wszelkie akcje sterowania podejmowane były bezpośrednio przez komputer. Rozkazy sterujące poprzez USB trafiały na mikrosterownik. Po wnikliwej analizie okazało się, że sterowanie obiektami działającymi w czasie rzeczywistym za pomocą łącza USB jest niewydajne. W przypadku, gdy pożądana jest szybka wymiana danych w obu kierunkach, standard komunikacyjny USB nie jest zalecanym wyborem. W pracy [9] głębiej analizującej przytoczony problem, dokonano dokładnych wyliczeń. Otóż analiza opóźnień dla danych wyjściowych wynosiła około 6 milisekund. Przeważająca liczba prób wykazała opóźnienie dla danych wejściowych na poziomie 12 milisekund. Suma wspomnianych opóźnień czasowych wynosi zatem 18 milisekund.



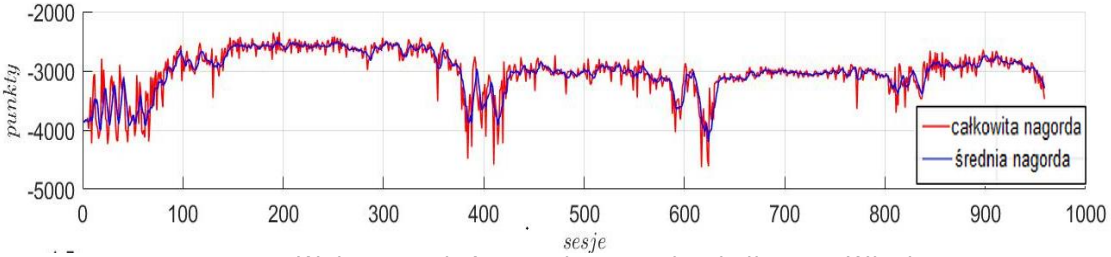
Rys. 6.2 Wykres punktów przekazywania akcji przez USB

Na rysunku 6.2 widać efekty uczenia w sytuacji, gdy łączem pomiędzy komputerem a sterownikiem było USB. Z wykresu widać, że uzyskane punkty osiągnęły swoje maksimum już w około 120 sesji. Kolejno algorytm, działający aż do 1500 sesji nie był w stanie znacząco poprawić wyników uzyskanych już na początku działania. Czynników wpływających na uzyskanie takiego wyniku było wiele, jednakże tym, który uważaliśmy za najbardziej decydujący, były opóźnienia związane z komunikacją przy użyciu standardu USB.

6.2.2. Komunikacja Wi-Fi

Kolejno przetestowano komunikację komputer-sterownik za pomocą łącza bezprzewodowego. Próba odbyła się w sposób analogiczny do tego opisanego w poprzednim

podrozdziale. Komputer wciąż podejmował wszelkie decyzje związane z wyborem akcji, a jedynym wyróżnikiem był sposób komunikacji ze sterownikiem Raspberry Zero W.



Rys. 6.3 Wykres punktów przekazywania akcji przez Wi-Fi

Również tym razem proces nauki nie osiągnął zadowalających rezultatów. Wykresu z rysunku 6.3 nie można bezpośrednio porównywać z wykresem z rysunku 6.2 ze względu na różniący się sposób wyliczania punktów. W eksperymencie przeprowadzonym w tym podrozdziale zwiększony został wpływ prędkości obrotu wahadła na sumę uzyskanych punktów. Nie przeszkadza to jednak w wyciągnięciu wniosku dotyczącego sterowania bardzo czułym obiektem, pracującym w czasie rzeczywistym przy pomocy Wi-Fi. Nie można trafnie podać wartości opóźnień zaistniałych podczas komunikacji bezprzewodowej. Jest ona zależna od zbyt wielu czynników, takich jak chwilowe obciążenie routera lub rodzaje modułów połączeniowych w sterowniku i w komputerze. Sądząc po uzyskanych rezultatach opóźnienie to jest większe niż w przypadku połączenia kablem USB. Sposób komunikacji ze sterownikiem w sposób bezprzewodowy (Wi-Fi) nie nadaje się do sterowania odwróconym wahadłem.

6.2.3. Bezpośredni wybór akcji przez sterownik

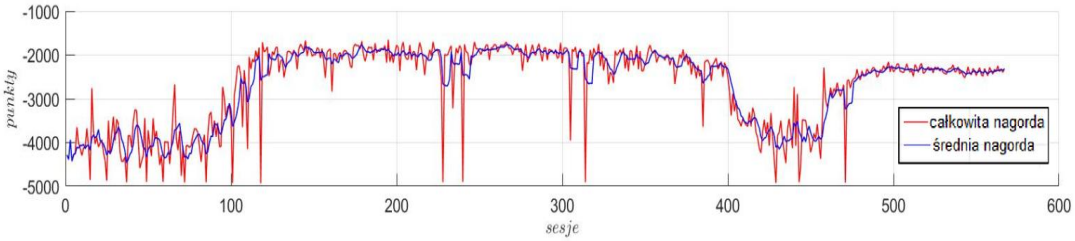
Utworzenie modelu za pomocą Simulinka umożliwia sterowanie rzeczywistym obiektem, za pośrednictwem zewnętrznego urządzenia, na dwa sposoby. Pierwszym jest działanie programu na komputerze (z zainstalowanym środowiskiem Matlab) oraz wysyłanie na minikomputer podjętych akcji. W ten sposób odbywało się sterowanie w dwóch wcześniej opisanych przykładach. Drugim sposobem jest wgranie programu na zewnętrzny minikomputer. Raspberry Zero W posiada dedykowaną nakładkę nazwaną „Simulink Support Package for Raspberry Pi Hardware”. W celu wgrania modelu na sterownik, kod działania musi zostać przetłumaczony na jeden z języków obsługiwanych przez system operacyjny zainstalowany na Raspberry.

Proces sterowania wahadłem bezpośrednio z minikomputera opisuje algorytm:

1. Założenie sieci neuronowej na komputerze (PC)
2. Wgranie modelu Simulinka na minikomputer
3. Uruchomienie programu na Raspberry
4. Przesłanie doświadczeń z sesji z minikomputera na komputer
5. Trening sieci przez komputer

6. Powrót do punktu 2, tyle razy, na ile zdeklarowana została liczba sesji.

Najbardziej czasochłonnym etapem w sterowaniu wahadłem bezpośrednio ze sterownika było tłumaczenie modelu na program w języku C, wysyłanie go oraz budowanie na zewnętrznym urządzeniu. Proces ten zajmował około 1,3 minuty, co jest wartością znacznie większą niż czas trwania pojedynczej sesji uczącej, wynoszący 25 sekund. Niestety operacja ta była konieczna ze względu na obecność sieci neuronowej wewnątrz przetłumaczonego programu.



Rys. 6.4 Wykres punktów doboru akcji przez sterownik

Wszystkie trzy metody opisane w tym rozdziale nie poradziły sobie z poprawnym wysterowaniem obiektu. Mimo, że wykres uzyskanej nagrody jest najbardziej adekwatnym narzędziem do podglądu osiągnięć agenta, to jego znaczenie staje się tym mniejsze, im program jest dalej od poprawnego działania. Wszystkie trzy sposoby, pomimo posiadania różnych opóźnień komunikacyjnych uzyskały zbliżone do siebie rezultaty, których ocena została wyciągnięta na podstawie obserwacji. Wahadło potrafiło nauczyć się rozbujać i wzbić w pozycję odwróconą. Problem następował podczas stabilizacji w obrębie kąta docelowego. Wahadło sprawiało wrażenie jakby nie do końca w sposób logiczny korygowało odchyłki kąta. Dodatkowo, wahadło nie zachowywało się przewidywalnie i powtarzalnie w okolicy górnego punktu położenia.

6.3. Potencjalne przyczyny niepowodzeń

Doszukując się przyczyn nieudanych prób sterowania obiektem, jedynym tropem były niezgodności w zachowywaniu się wahadła w pozycji odwróconej. Opóźnienia transportowe zostały zminimalizowane do wartości zaniedbywalnej, co zdyskwalifikowało to jako potencjalną przyczynę problemu.

Powodem mógł być nieodpowiedni dobór rozmiaru sieci neuronowej. Zbyt mało neuronów w warstwach ukrytych może niedostatecznie odwzorować skomplikowaną, nieliniową charakterystykę rzeczywistego wahadła. Złożoność rzeczywistego problemu z pewnością przewyższa idealizowany model symulacyjny. W przypadku niewystarczającej liczby neuronów możliwy jest przypadek, w którym sieć nie jest w stanie jednocześnie nauczyć się rozbujać wahadło oraz stabilizować je w pozycji odwróconej. Z tego powodu algorytm minimalizuje realne do osiągnięcia, globalne minimum. Jeśli sieć nie ma możliwości jednoczesnego

nauczenia się rozbujania i stabilizacji, wybiera to co przyniesie jej więcej korzyści na podstawie zdefiniowanej funkcji nagrody. W omawianym przypadku jest to szybkie i skuteczne rozbujanie ramienia 2 minimalizujące czas spędzany w jego dolnym położeniu. W przypadku zbyt małej sieci neuronowej może być to optymalna strategia sterownika. Zwiększając rozmiary sieci neuronowej nie uzyskano jednak zauważalnej poprawy w działaniu.

Ostateczną przyczyną niepowodzenia okazało się przyjęte, łatwiejsze z punktu widzenia implementacji, bezczujnikowe sterowanie silnikiem BLDC. Po dokładniejszej analizie okazało się, że systemy IPD oraz ISD nie działają dobrze z rozważanymi częstotliwościami sterowania. Przeprowadzane testy reakcji wahadła na pojedynczy impuls sygnału sterującego, okazały się mylące. Sterując wahadło z częstotliwością 20 Hz trudno jest zauważyć wpływ pojedynczych akcji na ruch ramienia 1. Mylący był również przeprowadzony test na poprawność reakcji ramienia. Badany był w nim efekt działania sygnału sterującego w postaci zmiennych ruchów lewo-prawo z częstotliwością 20 Hz. Ramię 1 wykonywało drgania o małej amplitudzie. Zwiększając częstotliwość do 40 Hz ramię 1 wciąż było w stanie wykonywać ruchy oscylacyjne o zmniejszonej amplitudzie. Drgania te wydawały się bardzo precyzyjne i w zupełności wystarczające do stabilizacji wahadła w pozycji odwróconej. Sugerując się obserwowanym działaniem ramienia 1 założono, że akcje wykonywane są poprawnie. Prawdziwą przyczyną drgań była sekwencja startowa systemu IPD. Drgania spowodowane były próbą zidentyfikowania położenia wirnika. System pracuje z maksymalną częstotliwością 95 Hz, jednak wysyła on w tym czasie 6 impulsów prądowych na odpowiednie fazy silnika. Przekłada się to na czas trwania sekwencji równy około 60 milisekund. Jest to więcej niż czas wykonania pojedynczego kroku sterowania, wynoszącego 50 milisekund. Oznacza to, że w trakcie wykonywania sekwencji startowej wahadło nie porusza się w oczekiwany przez nas sposób. Sekwencja ta powtarzana jest przy każdej zmianie kierunku ruchu. System zapewniający pracę i kontrolę silnika w pętli zamkniętej zaczyna działać dopiero po uzyskaniu odpowiedniej BEMF. Z tego powodu wybierając akcję inną niż poprzednia, system na nowo musi wykryć pozycję wirnika. Jeśli jedna z trzech wybieranych akcji jest taka sama jak poprzednia, system nie ulegnie desynchronizacji. Z uwagi na rodzaj problemu sterowania, wymagający częstej zmiany kierunku ruchu ramienia 1, system IPD działa niemal nieprzerwanie. Mając na uwadze zasadę działania algorytmu sterującego, nie jest możliwe nauczenie się stabilizacji w tak nieprzewidywalnym środowisku. Sieć neuronowa uczy się dobierania akcji, adekwatnych do aktualnego stanu obiektu. Algorytm nie zapamiętuje sekwencji, gdyż nie ma dostępu do wcześniej podjętych akcji. Obiekt musi być powtarzalny w taki sposób, by przy takim samym jego stanie, efekt podjętej akcji był zawsze taki sam. Sekwencja rozbujania oparta jest na rzadkiej zmianie kierunków ruchu. Z tego powodu, sterując rzeczywistym modelem wahadła, sieć była w stanie nauczyć się rozbujania i minimalizacji otrzymywanej kary. Inna sytuacja występuje w trakcie próby stabilizacji obiektu w pozycji odwróconej. Ze względu na dużą

częstotliwość zmian kierunku ruchu ramienia 1, wahadło nie jest w stanie wykonywać zadanych przez nas akcji. Wszystko to możliwe jest do zaobserwowania podczas sterowania rzeczywistym obiektem. Sterownik nie podejmuje żadnych logicznych działań w próbie stabilizacji wahadła w pozycji odwróconej.

Wybrany sterownik silnika bardziej nadaje się do sterowania prędkością kątową niż momentem obrotowym silnika. Sterowanie momentem siły uzyskiwane jest pośrednio poprzez tempo wzrostu prędkości oraz prądu w pętli otwartej, w której działamy. W zamyśle jest to natomiast pewien rodzaju soft-start do momentu uzyskania wystarczającej BEMF i przejścia w sterowanie w pętli zamkniętej. System bezczujnikowy jest bardziej zasadny do rozwiązań takich jak np. drony lub wentylatory. Tam takie rozwiązanie sprawdza się, gdyż śmigła poruszają się z dużą prędkością obrotową zazwyczaj w jedną stronę. System taki, pomijając chwilę startu, cały czas pracuje w pętli zamkniętej, bazując na BEMF. Małe opóźnienie rzędu setnych sekundy w chwili startu nie jest utrudnieniem dla tego typu zastosowań. Naszemu rozwiązaniu bliżej jest do precyzyjnego sterowania pozycją silnika niż jego prędkością obrotową. Konieczne jest zatem zastosowanie dodatkowego enkodera wykrywającego aktualne położenie silnika, a tym samym ramienia 1. Znając aktualne położenie silnika możemy wytworzyć zadany przez nas moment siły (niezależnie od kierunku). Przeszukując dostępne rozwiązania trudno jest znaleźć sterowniki kontrolujące moment obrotowy silników BLDC. Jest to niszowe zastosowanie. Zdecydowaną większość stanowią sterowniki kontrolujące prędkość obrotową. Sterownik kontrolujący obiekt za pomocą momentu siły, musi mieć dostępne wejście na sygnał z enkodera (mierzącego pozycję wału silnika). Jedyne znalezione rozwiązania to surowe sterowniki w postaci układów scalonych wykonanych w technologii SMT. Rozwiązanie takie wymagałoby utworzenia i wykonania własnego projektu płytki drukowanej. Dodatkowo, dostępność takich układów w Polsce jest znikoma, przez co okres oczekiwania jest znaczny (głównym dostawcą są USA). Alternatywą jest zbudowanie sterownika od podstaw, co byłoby jeszcze bardziej czasochłonne. W projekcie prawidłowo zdiagnozowano przyczynę niepowodzenia w sterowaniu rzeczywistym obiektem.

7. PODSUMOWANIE (PAWEŁ ZIELIŃSKI, NORMAN DRYŚ)

Celem projektu było stworzenie sterownika, którego działanie oparte jest na sieci neuronowej. Jako testowy obiekt wybrane zostało odwrócone, obrotowe wahadło znane jako wahadło Furuta. Dogłębna teoretyczna analiza działania algorytmu DQN umożliwiła poprawne zaimplementowanie go w postaci programu napisanego w języku Matlab.

Dostrajanie hiperparametrów treningowych, użytych w programie sterującym, odbywało się w sposób eksperymentalny. Istotnym czynnikiem, jeśli chodzi o niezawodność procesu uczącego, był rozmiar warstw ukrytych sieci neuronowej. Podczas testowania różnych

wielkości, okazało się, że z problemem sterowania wahadła Furuta najlepiej poradziła sobie sieć o warstwach ukrytych liczących kolejno 48 oraz 96 neuronów.

Kolejne badanie skupiające się przede wszystkim na sprawdzeniu dwóch sterowników pod kątem odporności oraz wpływu zakłóceń wykazało również bardzo ciekawe zależności. Dłuższy trening sieci neuronowej zmniejsza odporność sterownika, jednakże poprawia jego zdolności radzenia sobie z zakłóceniami. Odwrotne właściwości posiada sieć mniej trenowana. W zależności od wymagań sterowania należy dobrać odpowiednią długość treningu. Użycie sieci mniej trenowanej może mieć zastosowanie w systemach wbudowanych, gdzie mała moc obliczeniowa uniemożliwia uczenie sieci. Mała ilość treningu zapewnia wyższą skuteczność sterownia, co pożądane jest w systemach wbudowanych, ze względu na brak możliwości douczenia sieci. W ogólnym przypadku najlepszym rozwiązaniem jest ciągłe douczanie sterownika. Dzięki adaptacyjności osiągi są maksymalizowane w czasie rzeczywistym, przy jednoczesnym, poprawnym sterowaniu obiektem.

Następnie opisany został proces budowy rzeczywistego obiektu. Dokonano zakupu oraz montażu wszystkich elementów mechanicznych niezbędnych do przetestowania algorytmu sterującego w praktyce. Badane były rozwiązania komunikacji z minikomputerem przez USB oraz bezprzewodowo (Wi-Fi). W sytuacji gdy żadne z testowanych rozwiązań nie przyniosło zadowalających efektów, postanowiono wgrywać program sterujący na minikomputer. Proces przenoszenia programu sterującego był czasochłonny. Było to jedyne rozwiązanie niwelujące opóźnienia komunikacyjne.

Głównym problemem okazał się sterownik silnika, a dokładniej wbudowany system IPD. Korzystanie z systemu IPD uniemożliwia poprawne sterowania obiektem tak czułym, jak odwrócone wahadło Furuta. Rozwiązaniem problemu jest dobór sterownika umożliwiającego bezpośrednie sterowanie momentem obrotowym. Konieczny jest również zakup i odpowiedni montaż enkodera identyfikującego pozycję silnika.

Algorytm głębokiego uczenia sieci Q jest bardzo uniwersalnym oraz skutecznym narzędziem do sterowania obiektami silnie nieliniowymi. Takie rozwiązanie nie nadaję się w zastosowaniach, gdzie niemożliwe jest przeprowadzenie wielu cykli treningowych w czasie rzeczywistym. Kierując się zasadą brzytwy Ockhama, jeśli identyfikacja obiektu i zaimplementowanie sterownika w bardziej tradycyjny sposób (np. PID) jest prostsze i równie skuteczne od algorytmu DQN, powinniśmy wybrać to pierwsze. Obecnie stawiane są przed automatykami nowe wyzwania sterowania coraz to bardziej złożonymi obiektami. Postęp nauki, poprzez opracowanie takich algorytmów jak DQN, ułatwia sprostanie specjalistycznym zadaniom.

WYKAZ LITERATURY

1. Reinforcement learning, https://deeplizard.com/learn/playlist/PLZbbT5o_s2xoWNVdDudn51XM8lOuZ_Njv, (data dostępu: grudzień 2020)
2. Deep learning MIT, http://introtodeeplearning.com/, (data dostępu: grudzień 2020)
3. Batch, and an epoch, https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/, (data dostępu: grudzień 2020)
4. Activation functions, https://docs.paperspace.com/machine-learning/wiki/activation-function, (data dostępu: listopad 2020)
5. Aproksymacja funkcji wartości, https://www.davidsilver.uk/wp-content/uploads/2020/03/FA.pdf, (data dostępu: grudzień 2020)
6. Vincent Boucher (2019) "MONTRÉAL.AI ACADEMY: ARTIFICIAL INTELLIGENCE 101" Montreal, Quebec, Canada.
7. V. Mnih, K. Kavukcuoglu, D. Silver, A. a. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. *(2015). "Human-level control through deep reinforcement learning".* Nature Journal.
8. Recep Özalp, Nuri Köksal Varol, Burak Ta¸sci, and Ay¸segül Uçar. (2020). *"A Review of Deep Reinforcement Learning Algorithms and Comparative Results on Inverted Pendulum System"* Firat University Turkey
9. Niels Korver. (2003) *"Adequacy of the Universal Serial Bus for real-time systems"*. University of Twente.
10. Benjamin Seth Cazzolato and Zebb Prime. (2010). *"On the Dynamics of the Furuta Pendulum"* University of Adelaide
11. Tom Schaul, John Quan, Ioannis Antonoglou and David Silver. (2016). *"PRIORITIZED EXPERIENCE REPLAY"* Google DeepMind
12. Algorytm DQN wraz z rozszerzeniem Double DQN, https://ai.science/e/ddqn-deep-reinforcement-learning-with-double-q-learning--2019-02-28,(data dostępu: grudzień 2020)
13. Jannick Verlie (2016) "*Control of an inverted pendulum with deep reinforcement learning".* Ghent University
14. Biblioteki Deep Learning, https://github.com/rasmusbergpalm/DeepLearnToolbox, (data dostępu: listopad 2020)
15. Gotowe rozwiązanie Matlaba, ,https://www.mathworks.com/help/reinforcement-learning/ug/train-dqn-agent-to-swing-up-and-balance-pendulum.html, (data dostępu: listopad 2020)

WYKAZ RYSUNKÓW

Rys. 2.1 Schemat ruchu obrotowego, odwróconego wahadła [10] 9
Rys. 3.1 Warstwowa budowa sieci 13
Rys. 3.2 Wykresy funkcji aktywacji [4] 14
Rys. 3.3 Schemat blokowy działania neuronu 14
Rys. 3.4 Schemat działania MDP [1] 15
Rys. 3.5 Drzewo algorytmów RL [6] 16
Rys. 4.1 Wizualizacja działania wahadła 23
Rys. 4.2 Wykresy wskaźników uczenia 27
Rys. 4.3 Wykres osiągów sieci 12,24 29
Rys. 4.4 Wykres osiągów sieci 24,48 30
Rys. 4.5 Wykresy osiągów sieci 32,64 31
Rys. 4.6 Wykresy osiągów sieci 48,96 32
Rys. 4.7 Wykresy osiągów sieci 64,128 33
Rys. 4.8 Wyniki wznowienia procesu uczącego na zmienionym 36
Rys. 5.1 Rzeczywiste wahadło 38
Rys. 6.1 Schemat układu sterowania 43
Rys. 6.2 Wykres punktów przekazywania akcji przez USB 44
Rys. 6.3 Wykres punktów przekazywania akcji przez Wi-Fi 45
Rys. 6.4 Wykres punktów doboru akcji przez sterownik 46

WYKAZ TABEL

Tabela 1 parametrów uczących……..……..…...26
Tabela 2 Zestawienie wyników badania adekwatnego rozmiaru sieci…….....32
Tabela 3 Wpływ opóźnienia na czas uczenia……..….....33
Tabela 4 Wybór próbek uczących za pomocą priorytetowej pamięci doświadczeń…...34

ZAŁĄCZNIK NR 1: MATERIAŁY PREZENTUJĄCE DZIAŁANIE

W materiałach zamieszczono filmiki prezentujące działanie symulacji oraz rzeczywistego obiektu. Załączono również skrypt programu realizującego algorytm DQN.