Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki
Katedra Automatyki

Rozprawa doktorska

Problemy sterowania robotami kroczącymi — generatory chodu hexapoda

mgr inż. Marcin Piątek

Promotor:
dr hab. inż. Andrzej Turnau, prof. AGH

Kraków 2012

Serdecznie dziękuję za wszelką pomoc
i ogromną wiarę w moje możliwości
***Panu Profesorowi Andrzejowi Turnau**.*
Swoją wdzięczność wyrażam także wszystkim
tym, którzy udzielili mi wsparcia podczas
tworzenia tej pracy.

Spis treści

Rozdział 1

Wstęp

Roboty kroczące wśród mobilnych stanowią szczególną grupę. Naśladują sposób poruszania się człowieka i zwierząt [56], [61] i [62]. Większa ilość nóg u zwierzęcia czy robota to lepsza równowaga czyli stabilność statyczna i dynamiczna. Mniejsza ilość nóg to prostsze nimi poruszanie, czyli łatwiejsze sterowanie chodem.

Nieprzypadkowo w pracy wybór pada na sześcionoga. Taki robot pokonuje przeszkody, kroczy stabilnie po schodach, nie przewraca się po zatrzymaniu, korzysta z kilku rodzajów chodu. Wymaga niestety rozbudowanego układu sterowania Głównego Generatora Wzorców (ang. Central Pattern Generator) [13], [59] i [29]. Każdą nogą robota sterują trzy serwomechanizmy. Trzeba zatem sterować w czasie rzeczywistym osiemnastoma serwomechanizmami.

Celem głównym pracy jest zbudowanie i przebadanie generatora chodu hexapoda. Cel jest pojęty w miarę szeroko. Po pierwsze chodzi o wszelkie możliwe praktycznie realizowalne rodzaje chodu, zarówno te, które obserwujemy w przyrodzie wśród sześcionogów jak i takie, dla których nie ma wzorców biologicznych. Po drugie przedmiotem zainteresowania jest nie tylko pojedynczy rodzaj chodu, ale przede wszystkim płynna zmiana poruszania się przy przejściu z jednego rodzaju chodu w drugi. Tym zagadnieniom jest poświęcony największy wysiłek w pracy. Poczynając od oscylatorów liniowych przez nieliniowe skupiono się ostatecznie na oscylatorach Toda-Rayleigha.

Celem pobocznym pracy jest konstrukcja hexapoda i jego otoczenia w postaci stanowiska uruchomieniowego programowo-sprzętowego. Zadanie wykonania robota kroczącego i puszczenie go w ruch wymaga wiele pracy, ale w obecnej dobie po wejściu do internetu można zauważyć wiele rozwiązań poruszających się na ekranie. Twórcy tych rozwiązań czerpią niekłamaną przyjemność z uruchomienia własnego kroczącego robota. W przedstawionej pracy, jak można zauważyć w kolejnych jej rozdziałach, chodzi o pewne systematyczne podejście do problemów kroczenia. Autor, podobnie jak wspomniani internauci, czerpie satysfakcję z pokazania własnych filmów w internecie, na których można śledzić nie tylko, że robot porusza się, ale jak zmienia rodzaje chodu i jak płynnie przechodzi z jednego rodzaju chodu w drugi.

W pracy postawiono następujące tezy:

- Obserwacja działania wzorców biologicznych jest najbardziej naturalnym sposobem budowy nieznanych algorytmów ruchu gwarantującym zadowalający efekt kroczenia, a nawet w wielu przypadkach najlepszym z możliwych.
- Zastosowanie zbiorów oscylatorów nieliniowych w modelu CPG – Central Pattern Generator pozwala na osiągnięcie płynnych przejść pomiędzy trybami chodu.
- Zastosowanie oscylatorów nieliniowych umożliwia efektywną realizację CPG w układzie mikroprocesorowym.

W rozdziale drugim opisano cztery rodzaje chodów sześcionogów ze szczególnym podkreśleniem zapasów stabilności statycznej. Rozdział trzeci poświęcono konstrukcji stanowiska badawczego algorytmów chodu. Przedstawiono układ mechaniczny robota. Opisano szczegółowo odwrotne zadanie kinematyki. Pokazano warstwę sprzętową i programową. W rozdziale czwartym prowadzono identyfikację serwomechanizmów stosowanych w robocie. Przedstawiono modele symulacyjne prostego zadania kinematyki położenia i prędkości nogi robota.

Zasadnicze treści pracy są zawarte w rozdziałach piątym, szóstym i siódmym. Dotyczą one symulacji i eksperymentów rzeczywistych oscylatorów – generujących chód robota. Oscylatory liniowe pokazano tylko symulacyjnie dla pełności obrazu. Oscylatory nieliniowe Van der Pola, Rayleigha i kraty Toda-Rayleigha przedstawiono w dwóch postaciach: symulacyjnej i w formie eksperymentu rzeczywistego.

Pracę kończy podsumowanie, dodatek A – modele, dodatek B – kody źródłowe algorytmów.

Filmy przedstawiające robota podczas eksperymentów można znaleźć na stronie www pod adresem:
http://www.youtube.com/channel/UCeJVPT-RBaS3TphwwmFMsZg/videos

Rozdział 2

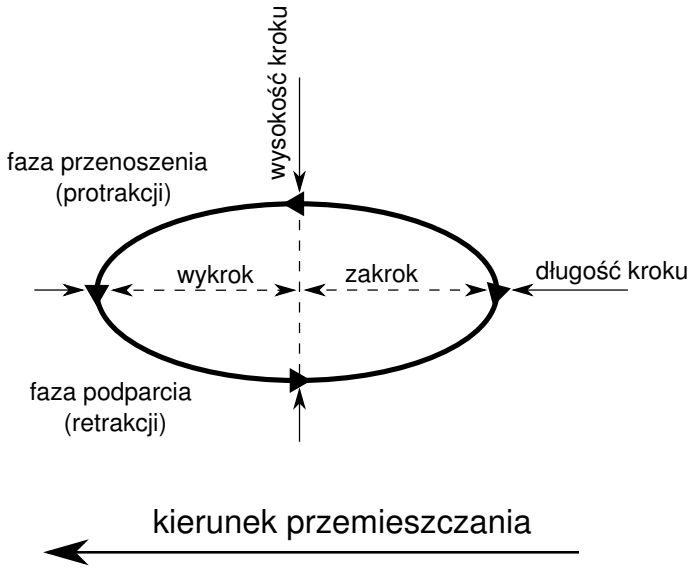
Algorytmy chodu

Chodem nazywamy formę lokomocji człowieka lub zwierząt polegającą na synchronicznym (często cyklicznym) przestawianiu odnóży (nóg) i powodującą przemieszczanie się. Na ogół w procesie chodu biorą udział także elementy tułowia. W trakcie chodu człowiek lub zwierzę porusza się w określonym kierunku (do przodu, do tyłu czy na boki) przez co rozumiemy zmianę położenia środka ciężkości w płaszczyźnie równoległej do podłoża. Efektem ubocznym może być unoszenie i opadanie środka ciężkości lub jego niewielkie wahania na boki [60].

Bieg jest to charakterystyczny rodzaj chodu, podczas którego występują takie odcinki czasu, gdy żadne z odnóży nie ma kontaktu z podłożem.

2.1 Pojęcia podstawowe

Zarówno podczas chodu jak i w trakcie biegania każda z nóg jest kolejno podnoszona do góry, przesuwana w kierunku ruchu, stawiana na podłożu, a następnie przesuwana w kierunku przeciwnym do kierunku przemieszczania. Na rysunku 2.1 przedstawiono trajektorię zakreśloną przez końcówkę nogi w trakcie przestawiania.



Rysunek 2.1: Proces przestawiania nogi z jego charakterystycznymi parametrami

Krzywa po jakiej porusza się końcówka nogi (stopa) ma kształt przypominający elipsę wyrysowaną na płaszczyźnie prostopadłej do podłoża i równoległej do kierunku przemieszczania się. Fragment ruchu nogi, gdy znajduje się ona nad powierzchnią podłoża,

nazywamy fazą podnoszenia lub fazą protrakcji. Fragment ruchu gdy noga jest oparta o podłoże nazywamy fazą podparcia lub fazą retrakcji. Faza retrakcji skutkuje ruchem postępowym środka ciężkości poruszającego się obiektu. Faza protrakcji w uproszczeniu jest przygotowaniem odpowiedniej pozycji początkowej nogi przed właściwym ruchem czyli retrakcją. Dodatkowo wyróżniamy dwie wielkości związane z procesem przestawiania nogi. Są to:

- wykrok – długość odcinka na jaką noga wysuwa się do przodu,
- zakrok – długość odcinka na jaką noga wysuwa się do tyłu.

Razem te dwa odcinki składają się na pełną długość kroku. Możemy zdefiniować także wysokość kroku czyli różnicę wysokości pomiędzy najwyższym, a najniższym położeniem nogi. Linią pogrubioną oznaczono zarys trajektorii jaką pokonuje końcówka nogi.

Analiza chodu jest przydatna do jego odtworzenia. Podstawowym wyróżnikiem chodu jest ilość odnóży wykorzystywana do przemieszczania [22], [18] i [55]. Zwierzęta posiadają parzyste ilości odnóży, możliwe więc są następujące konfiguracje:

- dwunożne (ptaki, człowiek),
- czteronożne (większość ssaków, płazów, gadów np. psy, konie),
- sześcionożne (część owadów np. mucha, karaluch),
- ośmionożne (pajęczaki),
- dziesięcionożne i więcej (stonoga, wij, prosinek).

Budowa maszyny kroczącej o nieparzystej ilości nóg np. trzech jest możliwa, ale takie konstrukcje nie występują w naturze.

Kolejnym ważnym wyróżnikiem rodzaju chodu jest kolejność przestawiania nóg. W zależności od ilości nóg można wyznaczyć różne sekwencje ich przestawiania, a zatem różne rodzaje chodów. Pewne rodzaje chodów są charakterystyczne tylko dla określonych układów nóg (np. chód trójpodporowy dla zwierząt sześcionożnych). Istnieją też chody niezależne od ilości nóg (np. chód metachroniczny).

Z punktu widzenia kolejności przestawiania nóg możemy wyróżnić kategorię chodów periodycznych (cyklicznych, okresowych) i chodów swobodnych. Chody periodyczne charakteryzują się tym, że kolejność przestawiania nóg jest okresowo powtarzana. Chód okresowy jest wybierany przez zwierzęta w sytuacji gdy poruszają się one po mało zróżnicowanym terenie jak płaskie lub prawie płaskie powierzchnie i przy braku przeszkód terenowych. Trajektoria po jakiej przemieszcza się środek ciężkości zwierzęcia w chodzie cyklicznym jest linią prostą lub ma kształt lekkiego łuku.

Gdy chód staje się aperiodyczny to zmienia się w chód swobodny[1]. W tym chodzie decyzja o tym, która noga będzie przestawiana zapada na bieżąco. Znajduje on zastosowanie w sytuacjach nietypowych, na trudnym terenie. Całość ruchu jest planowana na podstawie szybko zmieniających się parametrów zewnętrznych, odczytywanych przez zmysły.

Sposoby poruszania się konia: stęp, kłus, galop, cwał, inochód (występuje tylko u niektórych gatunków) są dobrze znanymi przykładami chodów periodycznych.

Jednym z najważniejszych parametrów chodu, mającym bezpośredni związek z ilością nóg, jest jego stabilność. Wyróżniamy:

[1]Chód swobodny można traktować jako jeden rodzaj chodu, bądź jako kategorię w ramach której wydzielimy szczególne przypadki jak np. chód za przewodnikiem czy reakcje odruchowe w sytuacjach niespodziewanej utraty stabilności lub napotkania przeszkody.

- chód stabilny statycznie, dla którego obiekt pozostaje stabilny (tzn. nie przewraca się), nawet gdyby doszło do pełnego zatrzymania wszystkich stawów na dowolnym etapie chodu,
- chód stabilny quasi-statycznie, dla którego w trakcie przemieszczania możemy znaleźć takie odcinki czasu, że obiekt jest stabilny statycznie jak i takie odcinki czasu w trakcie, których obiekt przestaje być stabilny,
- chód stabilny dynamicznie, dla którego obiekt traci stabilność w przypadku zatrzymania stawów w dowolnym momencie ruchu.

Człowiek lub jednonożne roboty skaczące poruszają się chodem dynamicznie stabilnym [63]. Większość zwierząt czteronożnych porusza się chodem quasi-statycznie stabilnym. Owady sześcionożne stosują chód stabilny statycznie.

W trakcie przestawiania nóg w chodach periodycznych często określa się tak zwany współczynnik obciążenia, czyli stosunek czasu styku nogi z podłożem do okresu chodu czyli czasu realizacji pełnej sekwencji przestawienia wszystkich nóg. Fazą względną nazywamy odcinek czasu od początku okresu chodu do postawienia danej nogi w stosunku do długości okresu chodu. Chody symetryczne posiadają identyczne współczynniki obciążenia wszystkich nóg, a fazy względne sąsiadujących nóg różnią się o połowę (prawa i lewa lub przednia i tylna).

Z ilości nóg, rodzaju chodu i wymienionych wyżej parametrów wynikają kolejne wielkości np.

- prędkość w fazie protrakcji – prędkość nogi podczas przestawiania jej do przodu,
- prędkość w fazie retrakcji – prędkość nogi podczas jej cofania czyli w istocie prędkość chodu.

2.2 Chody sześcionogów

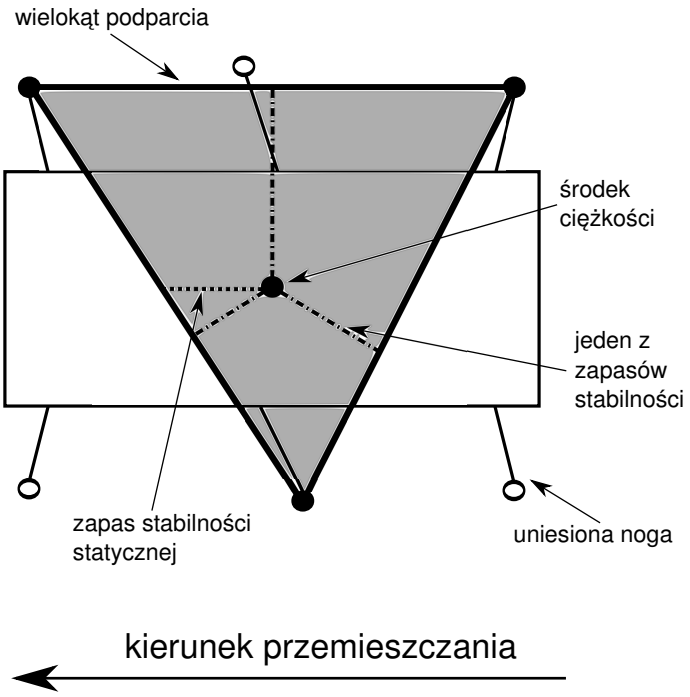
Roboty sześcionożne celowo są budowane z zachowaniem pewnego stopnia podobieństwa do owadów występujących w przyrodzie. Algorytmy chodu są wzorowane na sposobie poruszania się owadów [46], [3], [19] i [42]. Chód robota, a także owada sześcionożnego w przeciwieństwie do istot o mniejszej ilości nóg, charakteryzuje się tzw. stabilnością statyczną.

Przyjmuje się, że robot (owad) jest stabilny, gdy rzut środka ciężkości jego konstrukcji na podłoże w kierunku zgodnym z kierunkiem siły grawitacji mieści się w granicach ustanowionych przez figurę geometryczną wyznaczoną przez aktualne położenie nóg podpierających konstrukcję. Figurę tą nazywamy wielokątem podparcia. Definicja ta została zilustrowana na rysunku 2.2. Dodatkowo zaznaczono tzw. zapasy stabilności i zapas stabilności statycznej.

Zapasem stabilności nazywamy najmniejszą odległość rzutu środka ciężkości od krawędzi wielokąta podparcia. Zapasów stabilności możemy wyznaczyć tyle ile krawędzi posiada wielokąt podparcia. Wartość zapasów stabilności pozwala na określenie czy podczas ruchu konstrukcja nie przewróci się. Jest to jedno z podstawowych kryteriów badanych podczas analizy chodu.

Zapasem stabilności statycznej nazywamy odległość rzutu środka ciężkości od krawędzi wielokąta podparcia mierzoną wzdłuż prostej o kierunku zgodnym z kierunkiem ruchu. Wartość tego kryterium w trakcie chodu pozwala odróżnić chody stabilne statycznie od chodów quasi-statycznie stabilnych i stabilnych dynamicznie.

Algorytm przemieszczania się musi zapewniać pewne minimalne wartości zapasów stabilności, aby konstrukcja nie przewróciła się i zachowała stabilność statyczną. Wartości



Rysunek 2.2: Zarys sześcionożnego owada lub robota kroczącego wraz z wielokątem podparcia i zapasami stabilności

te są najczęściej dobierane eksperymentalnie i uwzględniają pewną tolerancję, która sprawia, że algorytm przemieszczania jest bardziej odporny na zakłócania niż w przypadku nie zachowania tolerancji.

Podczas analizy ruchu maszyn kroczących lub owadów definiuje się także wiele innych kryteriów stabilności [60], ale w niniejszej pracy będziemy korzystać jedynie z dwóch powyższych.

Łatwo można wywnioskować, że aby konstrukcja sześcionoga pozostała stabilna podczas chodu, powinien on w każdej chwili podpierać się na co najmniej trzech nogach. Możemy zatem zdefiniować trzy podstawowe rodzaje chodu:

a) jednocześnie przestawiana jest jedna noga w trakcie kroczenia, a pozostałe pięć podpiera konstrukcję,

b) jednocześnie przestawiane są dwie nogi w trakcie kroczenia, a pozostałe cztery podpierają konstrukcję,

c) jednocześnie przestawiane są trzy nogi w trakcie kroczenia, a pozostałe trzy podpierają konstrukcję,

d) kombinacja kilku powyższych np. w części faz przestawiana jest jedna noga, a w pozostałych dwie.

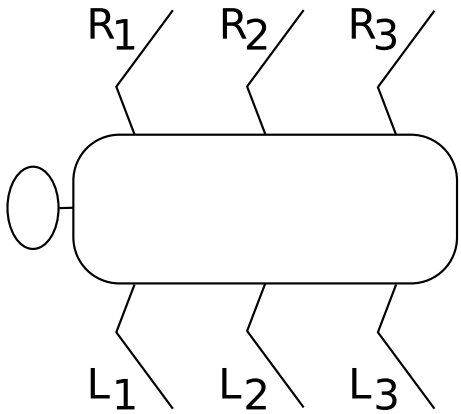
Pierwszy rodzaj chodu jest najbardziej powolnym, a trzeci najszybszym sposobem poruszania się sześcionożnych owadów i robotów kroczących. Periodyczne wersje tych czterech rodzajów chodu poprzez analogię można odnieść do szeroko opisywanych w literaturze sposobów poruszania się zwierząt czworonożnych [46], [2], [38] i [47].

Obserwacja chodu owadów prowadzona przez entomologów pozwoliła na zdefiniowanie sześciu zwięzłych prawidłowości dotyczących kolejności przestawiania nóg [7] i [58][2]:

[2]polskie tłumaczenie za [60]

- fale przemieszczeń nóg przenoszą się od tyłu do przodu ciała i żadna następna noga nie zostanie podniesiona, zanim noga znajdująca się za nią nie zostanie postawiona,
- naprzeciwległe nogi znajdujące się w tym samym segmencie ciała są przesunięte w fazie ruchu (nie są przemieszczane równocześnie),
- czas przenoszenia jest stałą dla wszystkich nóg, niezależnie od prędkości ruchu,
- czas podparcia (styku z podłożem) maleje w miarę wzrostu prędkości ruchu, rośnie więc częstość kroczenia (częstość przestawień nóg),
- odstęp czasu między podniesieniem nogi tylnej, a środkowej oraz między podniesieniem nogi środkowej i przedniej są porównywalne, podczas gdy taka różnica czasu pomiędzy nogą tylną, a przednią zmienia się proporcjonalnie do częstości kroczenia,
- noga przednia (lub środkowa) jest podnoszona tylko wtedy, gdy noga poprzedzająca ją w sekwencji zostanie postawiona na podłoże.

Wychodząc z zaobserwowanych własności możemy zdefiniować kolejność przestawiania nóg dla kilku podstawowych rodzajów chodu sześcionoga. Rysunek 2.3 przedstawia schematyczny kształt owada lub robota sześcionożnego, na którym oznaczono poszczególne nogi.



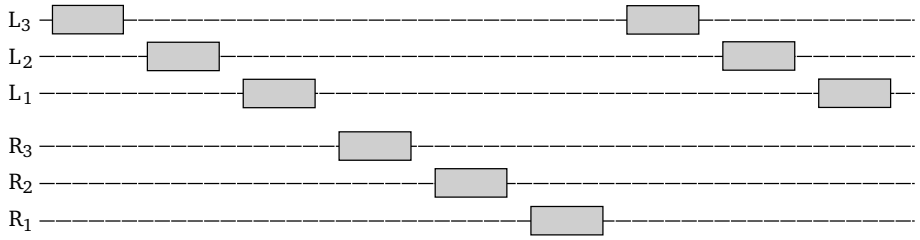
Rysunek 2.3: Schematyczna postać owada lub robota sześcionożnego wraz z oznaczeniem nóg

2.2.1 Chód metachroniczny

Najbardziej powolnym chodem sześcionoga jest ten, w którym nogi przestawiane są jedna po drugiej. Chód taki nazywamy pełzającym lub metachronicznym. Rysunek 2.4 przedstawia kolejność przestawiania nóg w tym chodzie. Współczynnik obciążenia dla każdej z nóg wynosi $\frac{1}{6}$.

Na wykresie 2.5 możemy zaobserwować wszystkie sześć cykli chodu metachronicznego wraz z zaznaczonym wielokątem stabilności. Zapasy stabilności są stosunkowo duże co oznacza, że nawet w przypadku napotkania nieoczekiwanej przeszkody obiekt z dużym prawdopodobieństwem zachowa stabilność. Chód możemy więc określić jako bezpieczny lub zachowawczy.

Podczas chodu metachronicznego w trakcie przestawiania jednej z nóg pozostałe pięć podtrzymuje konstrukcję. Oznacza to, że prędkość w fazie retrakcji jest pięć razy mniejsza niż w fazie przenoszenia. Rysunek 2.6 pokazuje trajektorie wszystkich sześciu nóg podczas



Rysunek 2.4: Wykres kolejności przestawiania nóg w chodzie metachronicznym

przemieszczania się. Ich charakterystyczny kształt wynika z dysproporcji prędkości w fazie protrakcji w stosunku do fazy retrakcji.

2.2.2 Chód czterotakowy

Drugim pod względem prędkości poruszania się jest chód składający się z czterech faz. Współczynnik obciążenia każdej z nóg wynosi $\frac{1}{4}$.

Charakterystyczną cechą tego chodu jest fakt, iż w dwóch fazach przestawiane są równocześnie dwie nogi, a w dwóch pozostałych fazach tylko jedna. Można to zaobserwować na wykresie 2.7.

Na rysunku 2.8 możemy prześledzić zmianę wielokąta stabilności dla wszystkich czterech faz chodu. Chód pomimo większej prędkości w dalszym ciągu pozostaje stabilny i posiada stosunkowo duże zapasy stabilności. Dla tego rodzaju chodu nie zostały wyznaczone trajektorie ruchu ponieważ autor nie będzie go analizował w dalszej części pracy.

2.2.3 Chód gąsienicowy

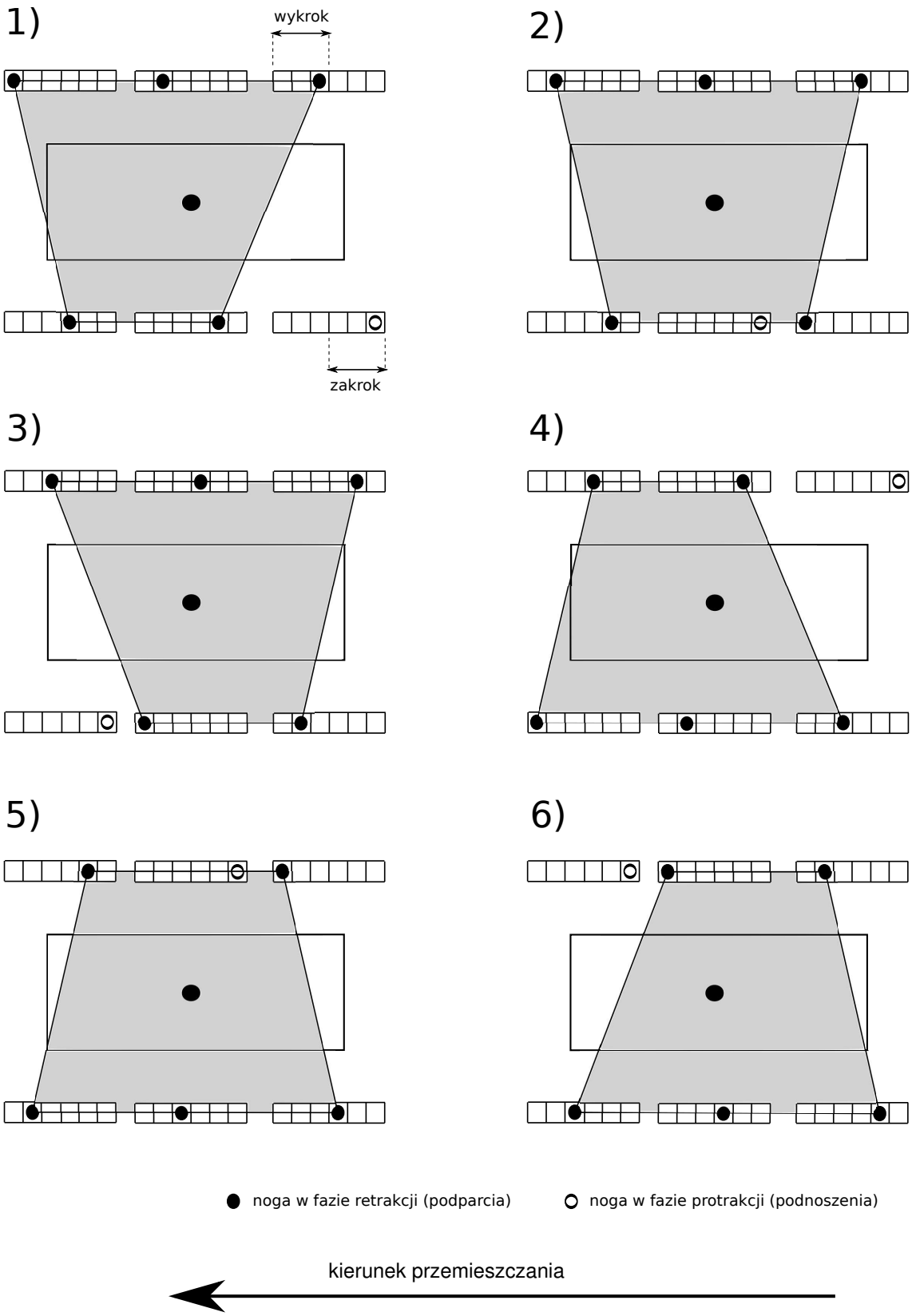
Chód gąsienicowy (nazwany w ten sposób ze względu na podobieństwo do charakterystycznego sposobu poruszania się gąsienic) składa się z trzech faz w trakcie, których kolejno przestawiane są nogi tylne, środkowe i przednie. Przedstawia to diagram na rysunku 2.9. Współczynnik obciążenia poszczególnych nóg wynosi $\frac{1}{3}$.

Na rysunku 2.10 możemy zaobserwować zmiany wielokąta stabilności podczas poszczególnych faz tego chodu. Ze względu na niewielki zapas stabilności wielkości wykroku i zakroku powinny zostać precyzyjnie dobrane. Przy odpowiednio dużych zakłóceniach na przykład w postaci przeszkód terenowych czy nierówności podłoża istnieje niebezpieczeństwo utraty stabilności. W założeniach ten rodzaj chodu wydaje się przystosowany do pokonywania przeszkód o regularnym kształcie na przykład schodów lub krawężnika. Nie spełnia drugiej z pięciu prawidłowości ruchu sześcionogów zaobserwowanych przez entomologów u owadów.

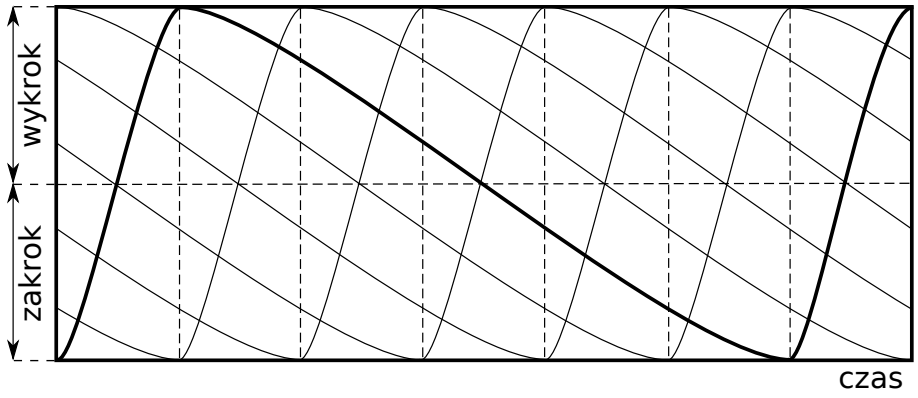
Wykres na rysunku 2.11 przedstawia trajektorie ruchu nóg w osi kierunku przemieszczania. Nogi są przestawiane parami więc widoczne są tylko trzy wykresy, pozostałe nałożyły się na siebie. Prędkość ruchu nogi w fazie retrakcji jest dwa razy mniejsza niż w fazie protrakcji.

2.2.4 Chód trójpodporowy

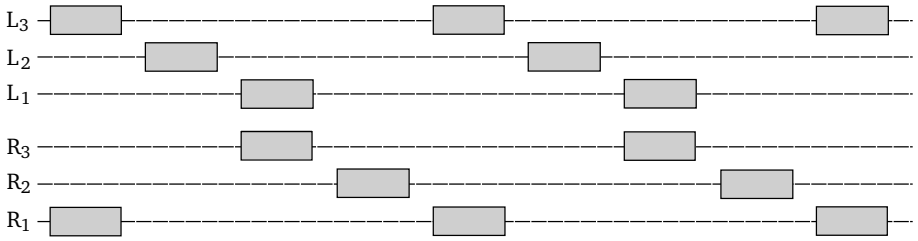
Chód trójpodporowy jest najszybszym chodem stabilnym jaki jest możliwy do realizacji w przypadku sześcionogów. Rysunek 2.12 przedstawia kolejność przestawiania nóg. Pełen cykl chodu składa się z dwóch faz, a współczynnik obciążenia wynosi $\frac{1}{2}$.



Rysunek 2.5: Sześć faz chodu metachronicznego z oznaczonym wielokątem stabilności



Rysunek 2.6: Trajektorie ruchu nóg sześcionoga w osi kierunku przemieszczania dla chodu metachronicznego (trajektoria pierwszej nogi została pogrubiona)

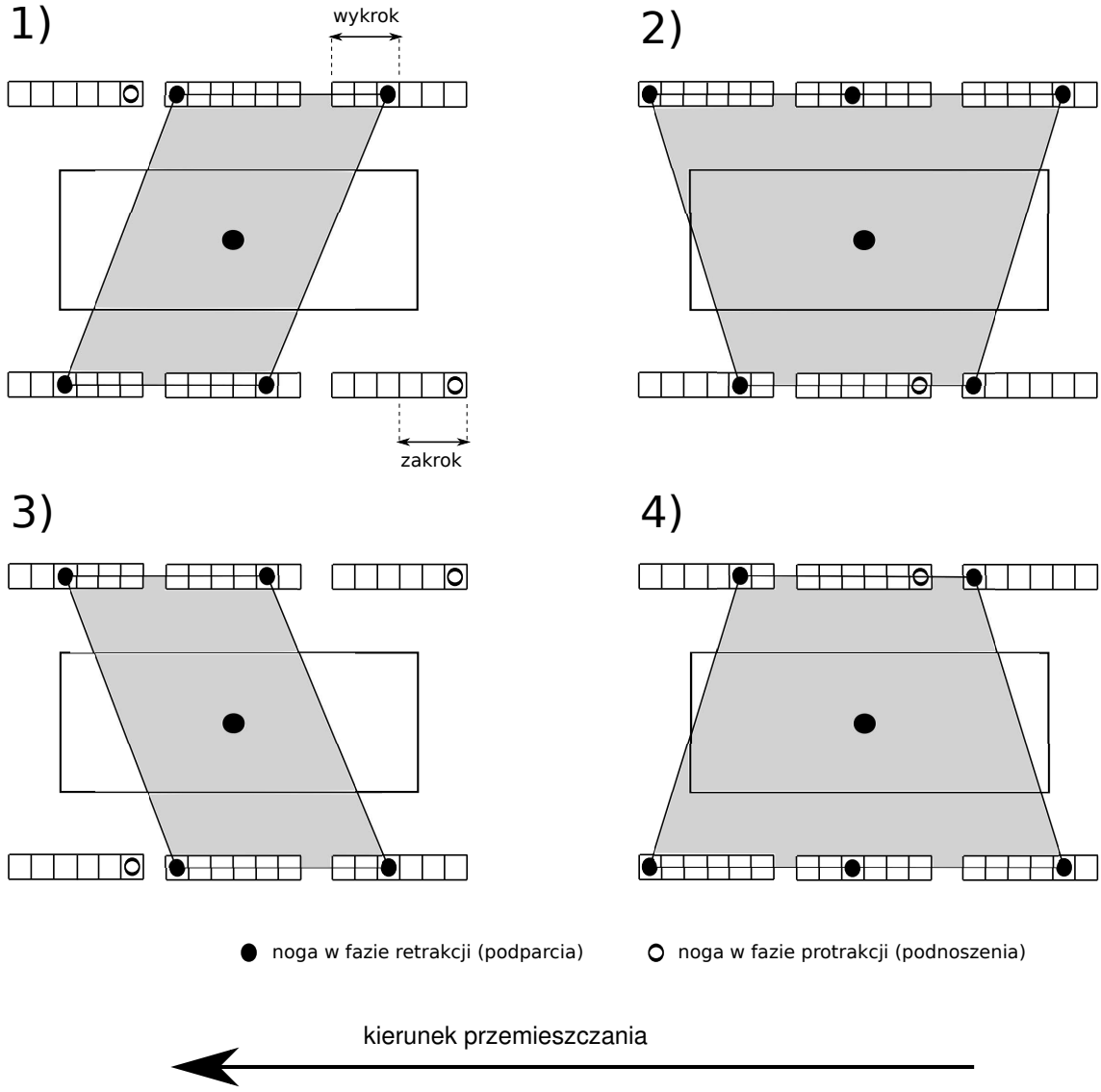


Rysunek 2.7: Wykres kolejności przestawiania nóg w chodzie czterotaktowym

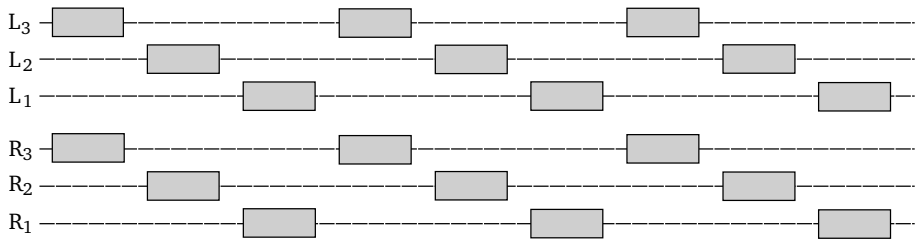
Rysunek 2.13 pokazuje obydwie fazy ruchu z zaznaczonymi wielokątami stabilności. Nogi podpierające korpus sześcionoga układają się w kształt trójkąta.

Rysunek 2.14 przedstawia trajektorie ruchu wszystkich sześciu nóg w osi kierunku przemieszczania. Nogi przestawiane są trójkami więc widoczne są tylko dwie trajektorie, a pozostałe nałożyły się na siebie. Prędkości nóg w fazie retrakcji i protrakcji są takie same.

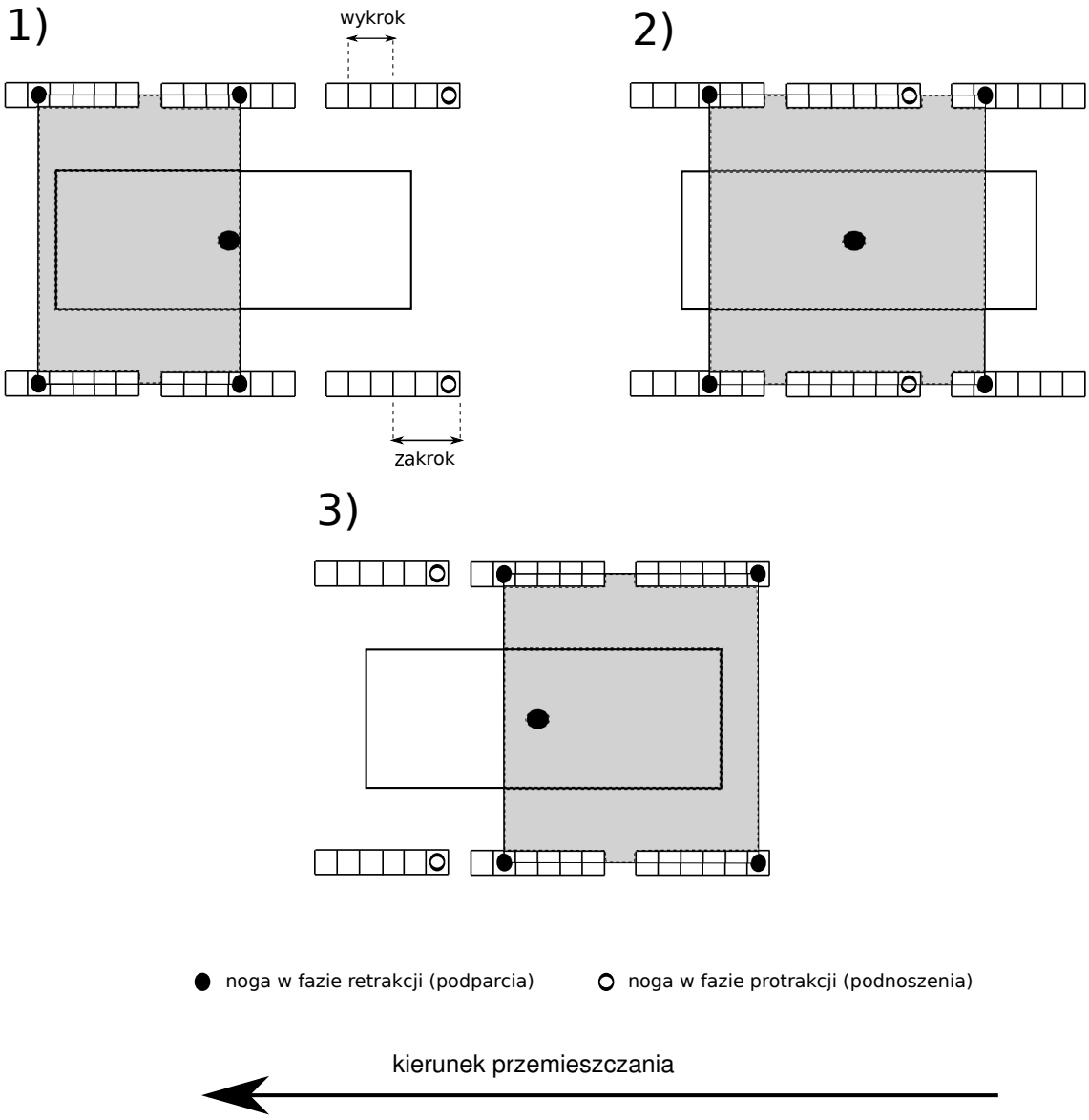
Chód trójpodporowy jest najczęściej stosowany przez owady sześcionożne. Charakterystyczny rozstaw nóg owadów (nogi środkowe rozstawione szerzej, przednie i tylne bliżej [19]) pozwala na zwiększenie zapasu stabilności podczas poruszania się. Dobrze służy podczas pokonywania długich, płaskich odcinków.



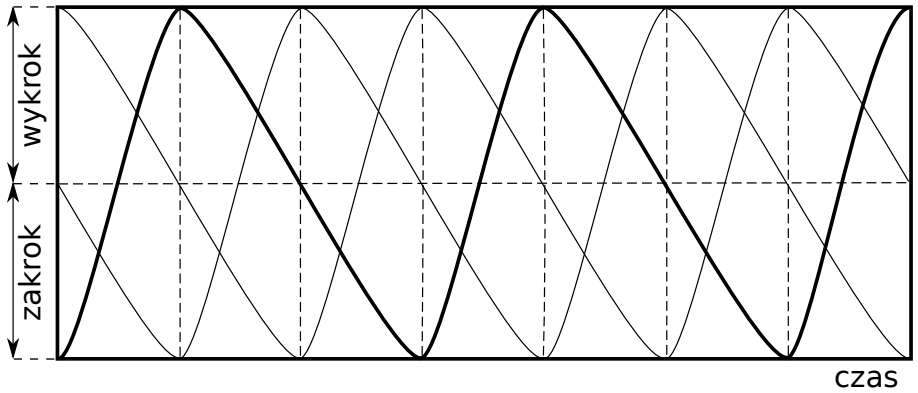
Rysunek 2.8: Cztery fazy chodu czterotaktowego z oznaczonym wielokątem stabilności



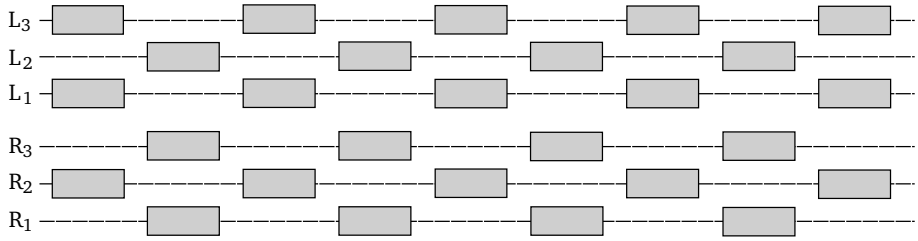
Rysunek 2.9: Wykres kolejności przestawiania nóg w chodzie gąsienicowym



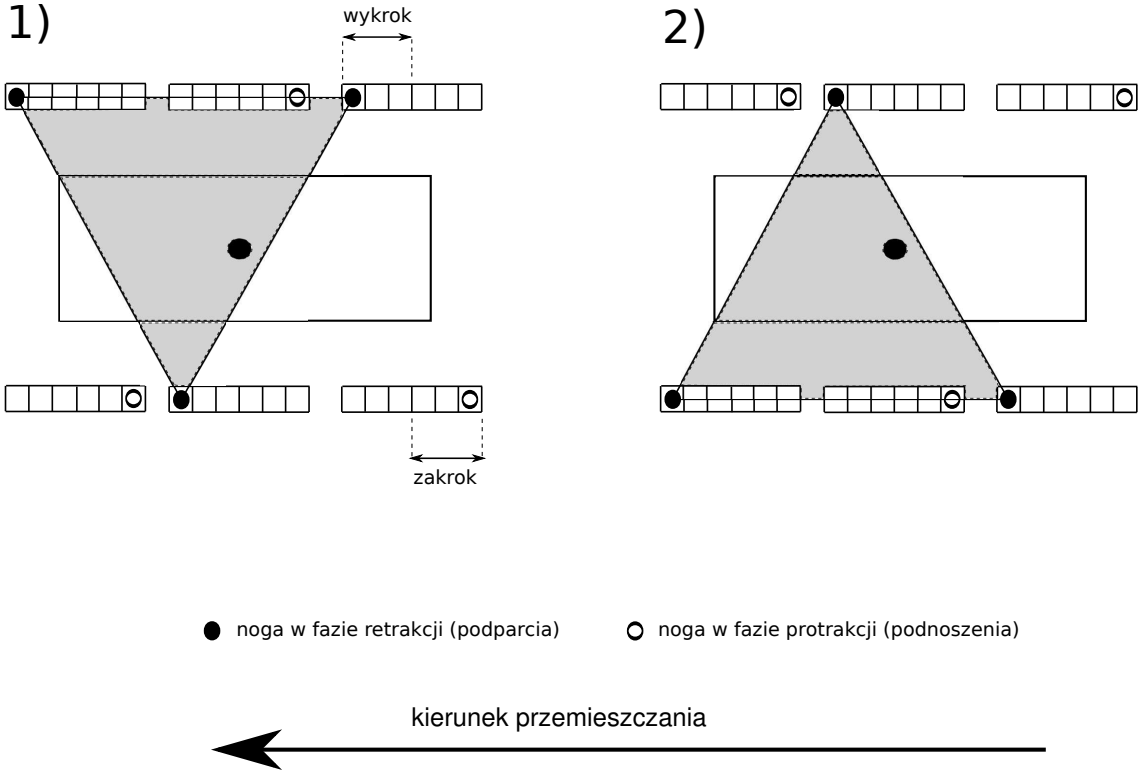
Rysunek 2.10: Trzy fazy chodu gąsienicowego z oznaczonym wielokątem stabilności



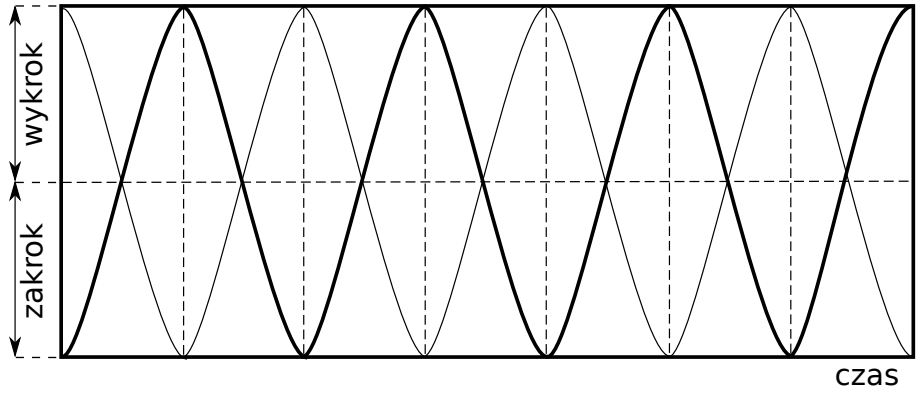
Rysunek 2.11: Trajektorie ruchu nóg sześcionoga w osi kierunku przemieszczania dla chodu gąsienicowego (trajektoria pierwszej nogi została pogrubiona)



Rysunek 2.12: Wykres kolejności przestawiania nóg w chodzie trójpodporowym



Rysunek 2.13: Sześć faz chodu trójpodporowego z oznaczonym wielokątem stabilności



Rysunek 2.14: Trajektorie ruchu nóg sześcionoga w osi kierunku przemieszczania dla chodu trójpodporowego (trajektoria pierwszej nogi została pogrubiona)

Rozdział 3

Konstrukcja stanowiska badawczego algorytmów chodu

W celu prowadzenia badań nad algorytmami ruchu zbudowano sześcionożnego robota kroczącego wraz z całym otoczeniem programowo-sprzętowym. Ten zbudowany warsztat – robot i otoczenie – nazwano stanowiskiem. Do elementów stanowiska należą:

- układ mechaniczny robota kroczącego,
  - łączniki aluminiowe stanowiące korpus konstrukcji robota,
  - elementy wykonawcze w postaci silników prądy stałego,
  - czujniki pomiarowe w postaci układów potencjometrycznych,
- warstwa sprzętowa układu sterowania,
  - układ FPGA,
  - przetworniki analogowo cyfrowe,
  - komputer klasy PC / układ mikroprocesorowy,
- oprogramowanie sterujące,
  - system operacyjny czasu rzeczywistego,
  - dedykowane oprogramowanie (biblioteki),
- oprogramowanie do wspomagania projektowania algorytmów i analizy wyników,
  - pakiet MATLAB & Simulink z odpowiednimi rozszerzeniami.

3.1 Układ mechaniczny robota kroczącego

Przygotowana konstrukcja sześcionożnego robota kroczącego składa się z prostokątnej podstawy o wymiarach 110 milimetrów na 370 milimetrów. Do podstawy przymocowano sześć identycznych nóg. Rozmieszczone je symetrycznie: trzy z prawej strony i trzy z lewej strony. Konstrukcja jest przedstawiona na rysunku 3.1.

Każda z nóg ma trzy stopnie swobody i składa się z aluminiowych elementów oraz trzech serwomechanizmów firmy Hitec model HS-475HB. Zastosowany serwomechanizm posiada element wykonawczy (silnik prądu stałego) i czujnik pomiarowy (czujnik rezystancyjny położenia kątowego). Układ zawiera także prosty regulator w pętli sprzężenia zwrotnego. Rysunek 3.2 przedstawia nogę robota w powiększeniu.

Rysunek 3.1: Robot kroczący

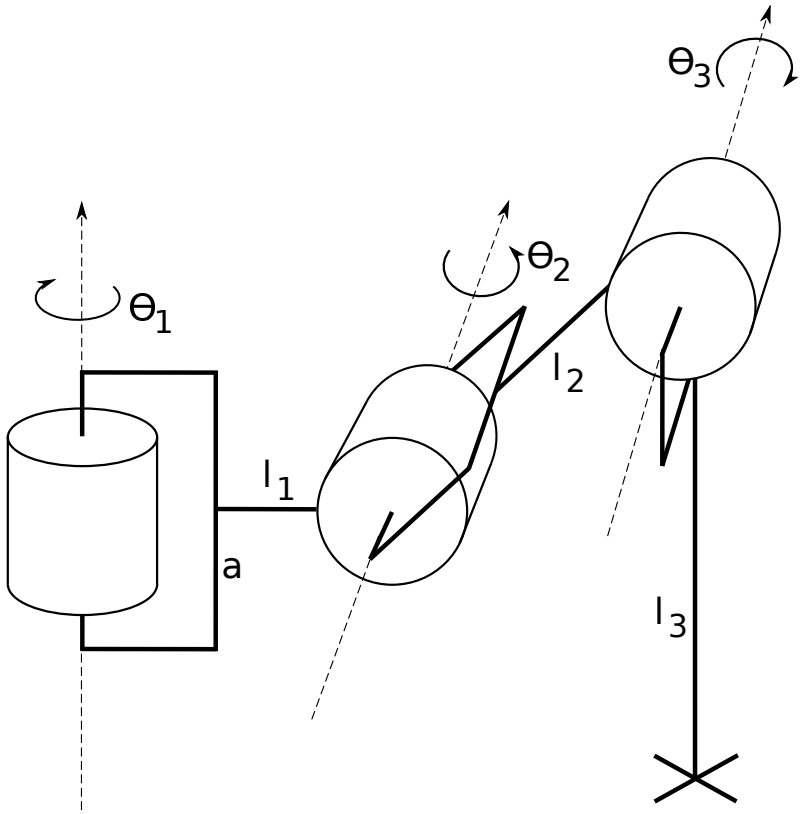
Rysunek 3.2: Jedna z nóg robota

Kształt i konstrukcja nogi robota są wzorowane na kształcie i konstrukcji nogi owadów sześcionożnych. Występują trzy przeguby obrotowe czyli jest to tak zwana geometria stawowa (RRR). Długości poszczególnych członów nogi robota zostały oznaczone l_1, l_2, l_3 i wynoszą:

$$\begin{aligned} l_1 &= 37\,\text{mm}, \\ l_2 &= 57\,\text{mm}, \\ l_3 &= 160\,\text{mm}\,. \end{aligned} \tag{3.1}$$

Kąty zakreślane przez kolejne człony robota oznaczono symbolami θ_1, θ_2 i θ_3 (patrz rysunek 3.3). Dodatkowo pomiędzy mocowaniami członów l_1 i l_2 występuje przesunięcie pionowe a o wartości:

$$a = 10\,\text{mm} \tag{3.2}$$

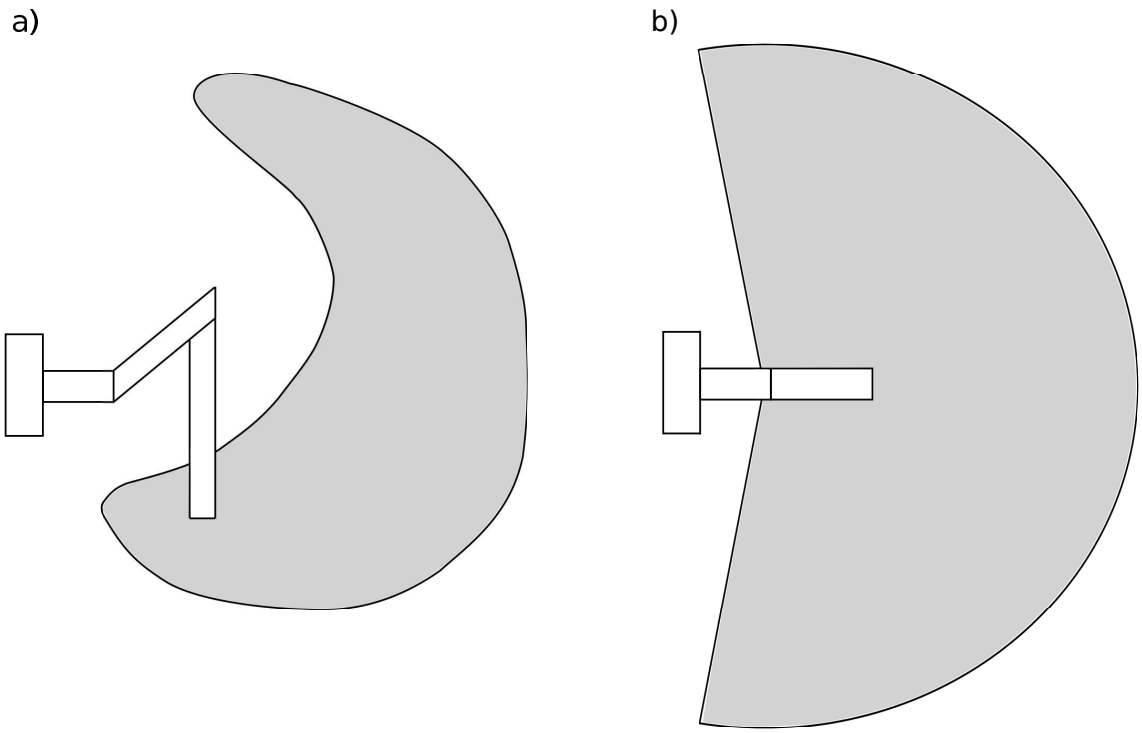


Rysunek 3.3: Struktura nogi robota

Konstrukcja nogi jest bardzo podobna do manipulatora z łokciem [50]. Różnica polega na tym, że oś pierwszego przegubu jest ustawiona poziomo do podstawy, a nie pionowo. Na ostatnim z członów zainstalowano gumową nakładkę redukującą ewentualny poślizg podczas kroczenia. Konfiguracja kinematyki nogi robota jest pokazana na rysunku 3.3. Zarys przestrzeni roboczej konstrukcji nogi przedstawiono na rysunku 3.4.

Na wszystkich przedstawionych wykresach noga przyjmuje tak zwaną pozycję bazową. Jest to pozycja wyjściowa, gdy robot stoi w miejscu. Od tej pozycji rozpoczyna się chód i w tej pozycji powinien się zakończyć. Wartości położenia kątowego przyjmowane przez poszczególne przeguby w pozycji bazowej nogi wynoszą odpowiednio:

$$\begin{aligned} \theta_1 &= 0^\circ \\ \theta_2 &= 30^\circ \\ \theta_3 &= 120^\circ \end{aligned} \tag{3.3}$$



Rysunek 3.4: Przestrzeń robocza nogi robota a) widok z boku, b) widok z góry

W algorytmach chodu robota generowane są pozycje zadane końcówek dla sześciu nóg. Sterowanie tego typu wymaga rozwiązania zadania odwrotnej kinematyki nogi robota.

3.2 Odwrotne zadanie kinematyki

Odwrotne zadanie kinematyki jest to funkcja lub algorytm pozwalające na obliczenie wartości współrzędnych naturalnych łańcucha kinematyki przy zadanej pozycji końcówki we współrzędnych kartezjańskich. Z punktu widzenia algorytmu sterującego ruchem łańcucha kinematyki istotne jest korzystanie ze współrzędnych kartezjańskich. Pozwala na naturalne dla człowieka postrzeganie ruchu łańcucha kinematyki. Dodatkowo korzystanie z układu kartezjańskiego jest bardziej uniwersalne niż stosowanie współrzędnych naturalnych, gdyż wartości współrzędnych kartezjańskich nie zależą od budowy samego łańcucha, czyli ilości i rozmiarów poszczególnych członów i rodzaju przegubów. Te zalety skłaniają do stosowania układu kartezjańskiego podczas projektowania algorytmów sterujących chodem robota kroczącego [20].

Niestety korzystanie z układów współrzędnych kartezjańskich podczas prac z łańcuchem kinematyki, jakim jest noga robota, posiada także kilka wad. Są nimi:

- stosunkowo duża złożoność obliczeniowa odwrotnego zadania kinematyki,
- utrudnienia w wyznaczaniu obszaru roboczego łańcucha kinematyki ze względu na jego nieregularny kształt w układzie kartezjańskim,
- występowanie osobliwości w odwrotnym zadaniu kinematyki.

Projektując i budując nogi robota kroczącego w sposób naturalny wykorzystujemy przeguby obrotowe ponieważ symulują one ruchy stawów organizmów biologicznych. Przemawiają za tym także względy praktyczne: koszty i prostsza budowa mechaniczna. Od-

wrotne zadanie kinematyki, gdy zastosowano przeguby obrotowe, wymaga wyliczania funkcji trygonometrycznych. Zwłaszcza w przypadku mniejszych mikrokontrolerów może to być poważny problem ze względu na narzut obliczeniowy[1] [28]. W przypadku robotów kroczących, które mają stanowić systemy autonomiczne, z jednej strony należy dążyć do zastosowania jak najprostszych, najmniejszych i możliwie energooszczędnych mikrokontrolerów, a z drugiej strony należy pamiętać o zapewnieniu odpowiednio dużej mocy obliczeniowej dla projektowanego algorytmu.

Podczas obliczania współrzędnych naturalnych nogi robota na podstawie położenia końcówki można natrafić na rozwiązania niejednoznaczne [6] i [50]. Na przykład ten sam punkt w przestrzeni kartezjańskiej można osiągnąć podsuwając końcówkę skierowaną w dół, do góry, w prawo lub lewo. Dla ruchu nogi robota w większości przypadków należy spodziewać się, że noga będzie skierowana w dół. W tej pracy wielu niejednoznaczności kinematyki nogi udało się uniknąć ze względu na fizyczne ograniczenia ruchu poszczególnych przegubów. Niestety nie wyeliminowano w ten sposób wszystkich punktów niejednoznacznych.

Projektowane algorytmy sterujące chodem robota sześcionożnego odnoszą się z osobna do ruchu końcówki każdej nogi robota sześcionożnego. Założono, że każda noga pracuje w swoim własnym układzie współrzędnych. To znaczy, że początek układu współrzędnych $(0, 0, 0)$ jest umieszczony w punkcie połączenia nogi z podstawą. Upraszcza to konstrukcję pewnych elementów algorytmu i pozwala na wielokrotne wykorzystanie fragmentów kodu źródłowego algorytmu sterującego. Dodatkowo założono, że kierunki osi układu kartezjańskiego dla każdej z nóg przebiegają w następujący sposób:

- oś X – przebiega w kierunku przód-tył całego robota z dodatnią półosią zwróconą do przodu,
- oś Y – przebiega w kierunku od prawa do lewa całego robota z dodatnią półosią zwróconą do zewnętrznej strony robota to oznacza, dla nóg po prawej stronie w prawo, dla nóg po lewej stronie w lewo,
- oś Z – przebiega w kierunku góra-dół robota z dodatnią półosią zwróconą w górę.

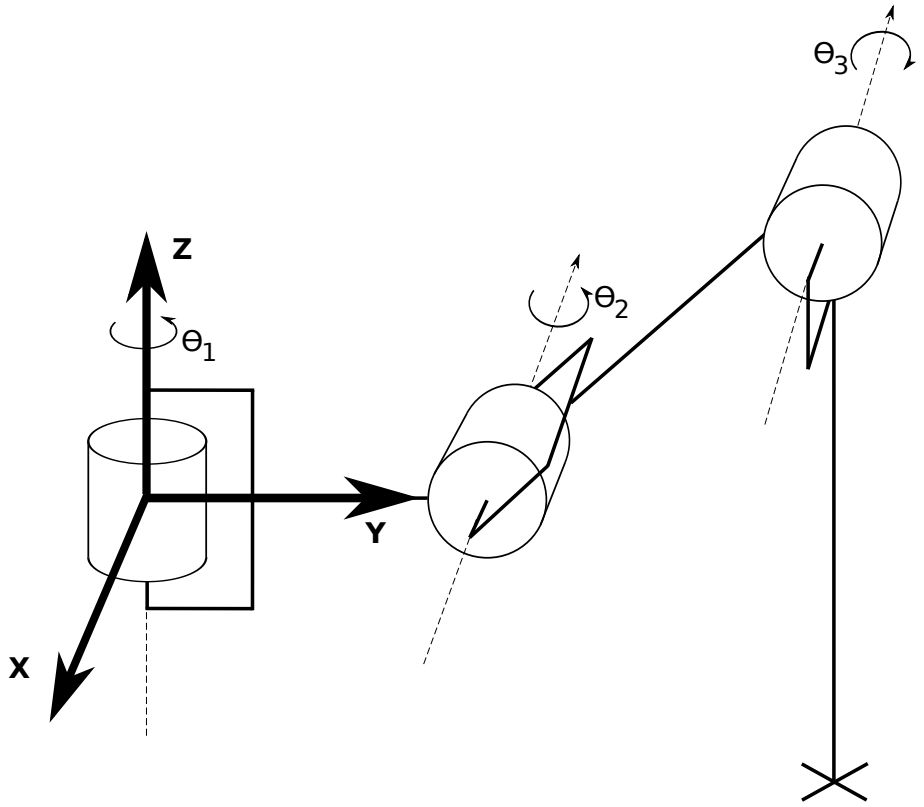
Rysunek 3.5 przedstawia prawą nogę robota z zaznaczonym układem współrzędnych. Noga lewa jest lustrzanym odbiciem nogi prawej.

Rozwiązując odwrotne zadanie kinematyki na ogół wystarczy wykorzystać prostą trygonometrię. Tak zostało to rozwiązane w tym przypadku. Podczas analizy kinematyki nogi robota należy wyprowadzić wzory na trzy kąty oznaczone odpowiednio θ_1, θ_2 i θ_3 będące położeniami kątowymi trzech serwomachanizmów. Argumentami szukanych wzorów będą wartości p_x, p_y i p_z czyli współrzędne końcówki nogi robota w układzie kartezjańskim. Na rysunku 3.6 przedstawiono rzut płaszczyzny XY (czyli widok z góry na nogę robota). Noga robota ułożona jest dowolnie. Zaznaczono poszukiwany kąt θ_1. Długości rzutów nogi p_x i p_y są zarazem współrzędnymi końcówki nogi. Łatwo można zapisać zależność:

$$\theta_1 = \operatorname{arctg}\left(\frac{p_y}{p_x}\right) \tag{3.4}$$

Przed wyprowadzeniem zależności na pozostałe dwa kąty należy zdefiniować płaszczyznę oznaczoną literą R. Jest to płaszczyzna prostopadła do poziomej płaszczyzny XY, przecinająca układ współrzędnych w punkcie $(0, 0, 0)$ i przecinająca płaszczyznę YZ pod

[1] W praktyce funkcje trygonometryczne w układach mikroprocesorowych są wyliczane albo za pomocą rozwinięcia szeregu Taylora, albo za pomocą tablic LUT (look-up table). Pierwsza metoda jest czasochłonna, druga wymaga wykorzystania odpowiednio dużej tablicy pamięci.



Rysunek 3.5: Schemat nogi prawej robota z zaznaczonym kartezjańskim układem współrzędnych

kątem θ_1. Płaszczyzna ta jest wyznaczona przez nogę robota w taki sposób, że wszystkie człony łańcucha kinematyki nogi leżą zawsze na niej. Rysunek 3.7 przedstawia przykładowy rzut nogi robota na płaszczyznę R. Zaznaczono na nim długości poszczególnych członów l_1, l_2 i l_3, kierunek i wartość przesunięcia pomiędzy mocowaniem członu l_2 do l_1 oznaczone jako a i wartości przykładowych kątów θ_2 i θ_3. Zaznaczono także wartości położenia w osi Z oznaczone jako p_z i odległość końcówki nogi robota od osi Z na płaszczyźnie R oznaczoną p_r. Z rysunku 3.6 wynika, że wartość p_r wynosi:

$$p_r = \sqrt{p_x^2 + p_y^2} \tag{3.5}$$

Z kolei na rysunku 3.8 oznaczono odcinek c stanowiący przeciwprostokątną trójkąta prostokątnego. Pozostałe boki tego trójkąta mają długości: $p_r - l_1$ oraz $p_z - a$ (p_z w tym przykładzie ma wartość ujemną). Długość odcinka c zapisujemy wzorem:

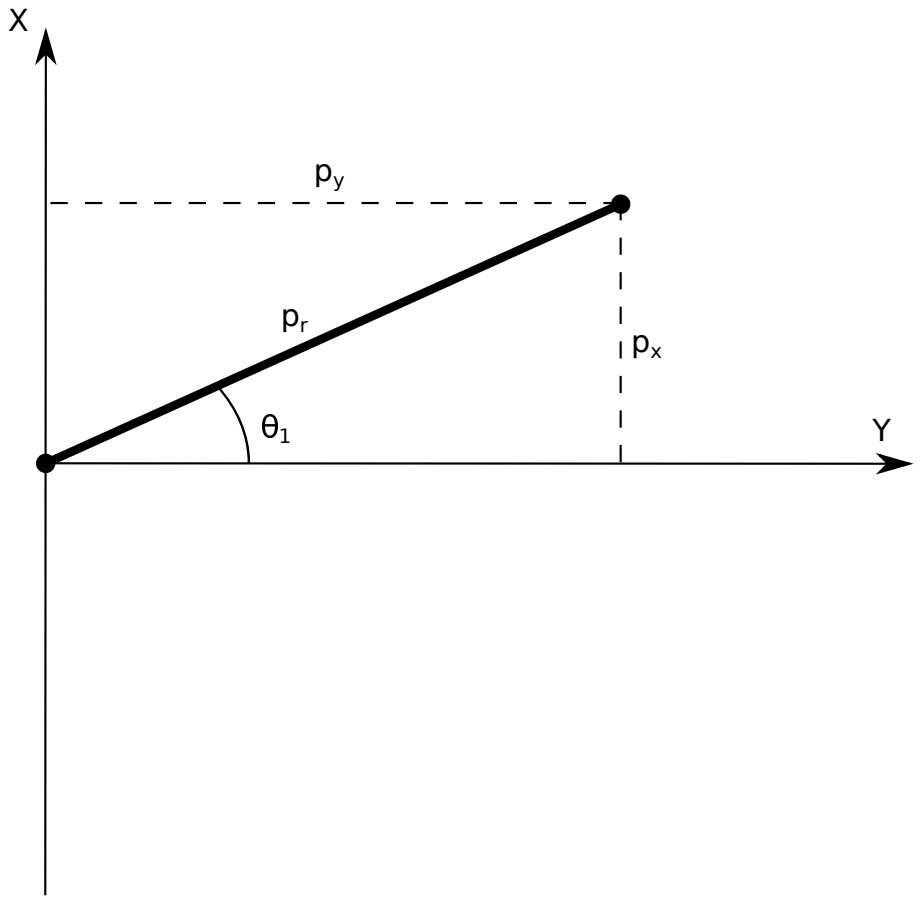
$$c = \sqrt{(p_z - a)^2 + (p_r - l_1)^2} \tag{3.6}$$

Z twierdzenia Carnota możemy wyliczyć:

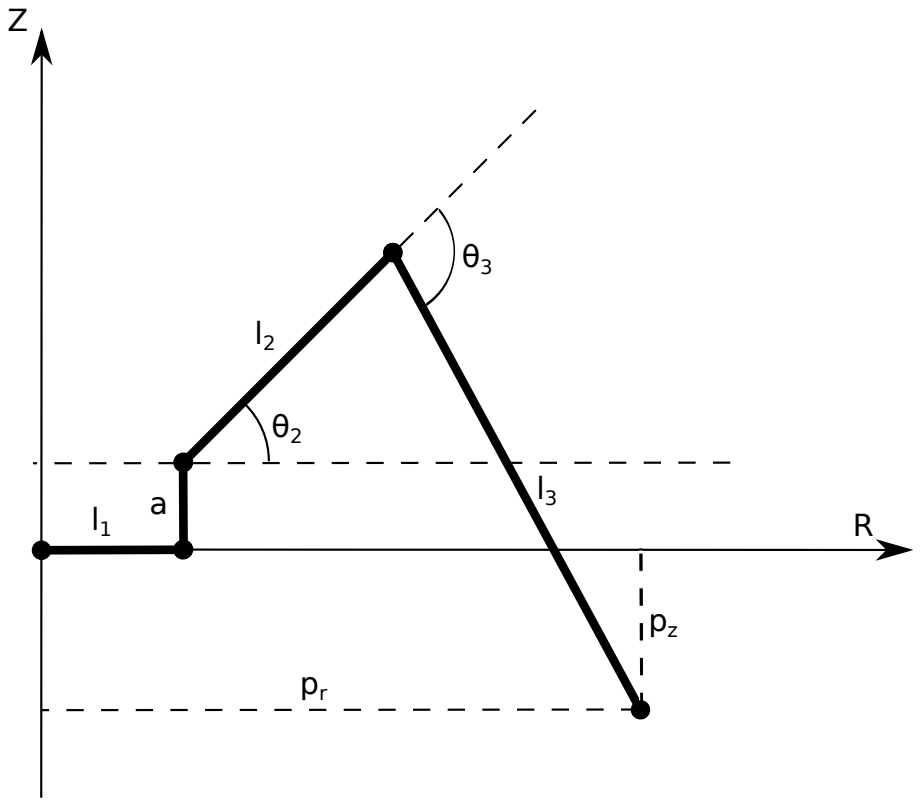
$$\cos(\pi - \theta_3) = \frac{c^2 - l_2^2 - l_3^2}{-2l_2l_3} \tag{3.7}$$

i uprościć do postaci:

$$\cos(\theta_3) = \frac{c^2 - l_2^2 - l_3^2}{2l_2l_3} \tag{3.8}$$



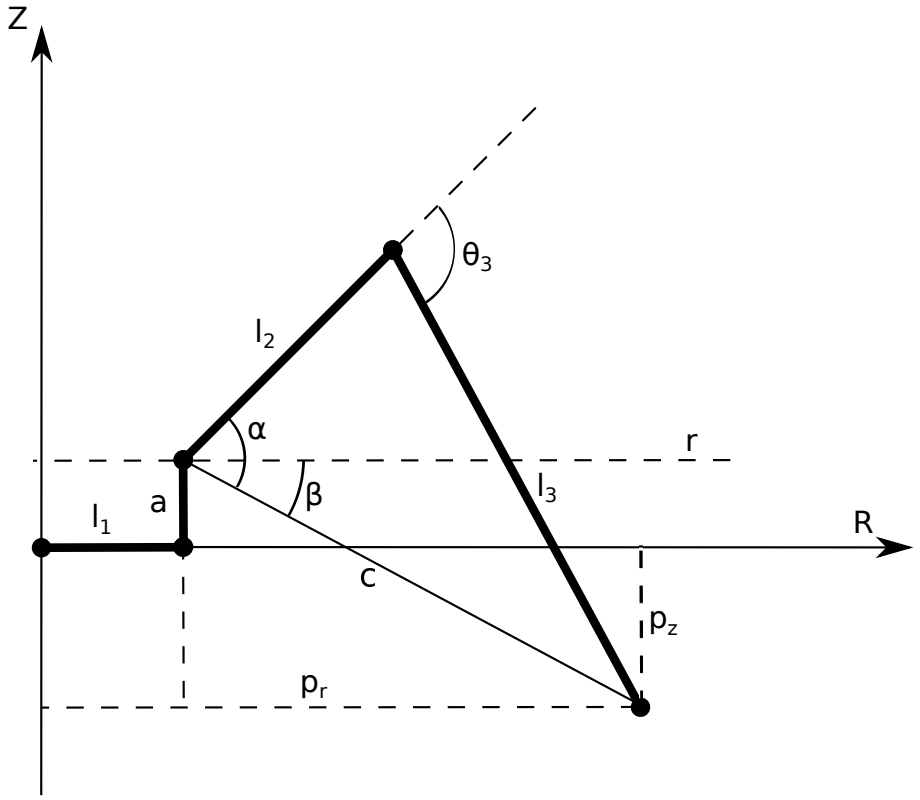
Rysunek 3.6: Rzut nogi robota w płaszczyźnie XY z zaznaczonym kątem θ_1



Rysunek 3.7: Rzut nogi robota w płaszczyźnie R z zaznaczonym kątem θ_2 i θ_3

Ostatecznie otrzymuje się wzór na kąt wychylenia trzeciego przegubu nogi robota:

$$\theta_3 = \arccos\left(\frac{c^2 - l_2^2 - l_3^2}{2l_2l_3}\right) \tag{3.9}$$



Rysunek 3.8: Rzut nogi robota w płaszczyźnie R z zaznaczonym kątem α i β

Najbardziej złożone jest wyprowadzenie zależności dla kąta θ_2. Należy zdefiniować jedną prostą oznaczoną jako r i dwa kąty oznaczone jako α i β.

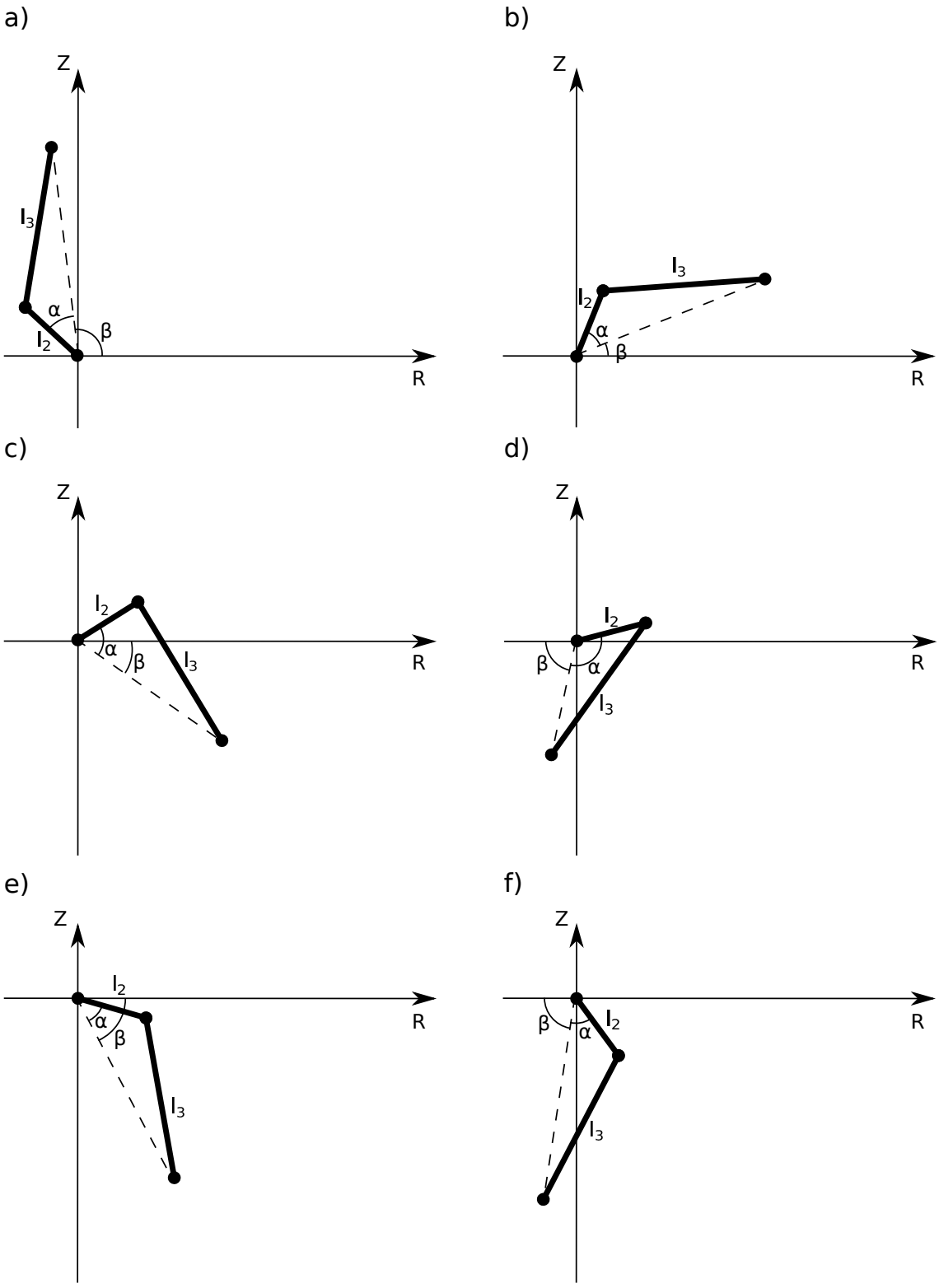
Prosta r jest prostą prostopadłą do osi Z, leży na płaszczyźnie R i przechodzi przez punkt mocowania członu l_2. Kąt pomiędzy członem l_2, a prostą r oznaczono α. Kąt pomiędzy wcześniej zdefiniowanym odcinkiem c i prostą r oznaczono β. Prostą r i kąty α oraz β pokazano na rysunku 3.8. Kąty α i β można łatwo wyliczyć. Dla α należy skorzystać z twierdzenia Carnota:

$$\alpha = \arccos\left(\frac{l_3^2 - l_2^2 - c^2}{-2l_2c}\right) \tag{3.10}$$

Dla kąta β można zauważyć, że stanowi on jeden z kątów trójkąta prostokątnego tego samego, dla którego wyprowadzono wzór na długość odcinka c. Dzięki temu można zapisać, że:

$$\beta = \operatorname{arctg}\left(\frac{p_z - a}{r - l_1}\right) \tag{3.11}$$

Kąt θ_2 można uzależnić od kątów α i β. Zależność ta ma inną postać dla różnych ułożeń poszczególnych członów nogi robota. Gdy końcówka nogi robota jak i przegub na łączeniu członów l_2 i l_3 przyjmują inne położenie w różnych ćwiartkach płaszczyzny R, kąt β także zmienia znak i położenie. Kąt α jest zawsze dodatni ponieważ został zdefiniowany jako



Rysunek 3.9: Rzut nogi robota w płaszczyźnie R z zaznaczonym kątem α i β oraz z przesuniętym środkiem układu współrzędnych (sześć różnych przypadków ułożenia przegubów l_2 i l_3)

kąt w trójkącie stanowiącym człony l_2 i l_3 nogi robota, a fizyczne ograniczenia ruchu tych członów uniemożliwiają rozwarcie tak by kąt θ_3 stał się ujemny.

Uwzględniając fizyczne ograniczenia ruchu nogi robota i poszczególnych przegubów, wyznaczono sześć różnych przypadków położenia przegubów l_2 i l_3. Zostały one zilustrowane na rysunku 3.9. Dla ułatwienia interpretacji na rysunku przesunięto środek układu współrzędnych do punktu mocowania członu l_2. Zależności pomiędzy kątem θ_2, a kątami α i β dla wszystkich analizowanych przypadków zostały przedstawione w tablicy 3.1. Z przedstawionych danych wynika, że wzór opisujący zależność między kątami przyjmuje tylko dwie różne postacie, a różnica pojawia się gdy odcinek będący odległością końcówki nogi robota, a punktem mocowania członu l_2 wzdłuż prostej r zmienia znak. Długość tego odcinka wynosi $p_r - l_1$ (patrz rysunek 3.8). W związku z tym możemy zapisać, że:

przypadek	znak β	znak θ_2	wartość θ_2
a)	dodatni	dodatni	$\theta_2 = \alpha + \beta$
b)	dodatni	dodatni	$\theta_2 = \alpha + \beta$
c)	ujemny	dodatni	$\theta_2 = \alpha + \beta$
d)	dodatni	dodatni	$\theta_2 = \alpha - (\pi - \beta)$
e)	ujemny	ujemny	$\theta_2 = \alpha + \beta$
f)	dodatni	ujemny	$\theta_2 = -(\pi - \alpha - \beta) = \alpha - (\pi - \beta)$

Tablica 3.1: Przypadki zależności kąta θ_2 od kątów α i β

$$\theta_2 = \begin{cases} \alpha + \beta & , p_r - l_1 \geq 0 \\ \alpha - (\pi - \beta) & , p_r - l_1 < 0 \end{cases} \tag{3.12}$$

Ostatecznie pełny algorytm pozwalający na obliczenie całego zadania kinematyki odwrotnej dla nogi robota ma następującą postać:

$$\begin{aligned} p_r &= \sqrt{p_x^2 + p_y^2} \\ c &= \sqrt{(p_z - a)^2 + (p_r - l_1)^2} \\ \alpha &= \arccos\left(\frac{l_3^2 - l_2^2 - c^2}{-2l_2 c}\right) \\ \beta &= \operatorname{arctg}\left(\frac{p_z - a}{r - l_1}\right) \\ \\ \theta_1 &= \operatorname{arctg}\left(\frac{p_y}{p_x}\right) \\ \theta_2 &= \begin{cases} \alpha + \beta & , p_r - l_1 \geq 0 \\ \alpha - (\pi - \beta) & , p_r - l_1 < 0 \end{cases} \\ \theta_3 &= \arccos\left(\frac{c^2 - l_2^2 - l_3^2}{2l_2 l_3}\right) \end{aligned} \tag{3.13}$$

Przedstawiony algorytm jest prosty, ale jego skuteczna implementacja powinna uwzględniać wszelkie niejednoznaczności [6]. Po eliminacji punktów i orientacji nogi robota nieosiągalnych ze względu na ograniczenia fizyczne wynikające z mechanicznej budowy układu pozostaje do analizy tylko obsługa przypadków, w których końcówka nogi robota znajduje się w osi Z to znaczy współrzędne p_x i p_y przyjmują wartość 0. Przypadki takie można łatwo wykryć podczas wykonywania obliczeń w kodzie źródłowym implementacji

ponieważ wartość obliczanej zmiennej p_r wtedy i tylko wtedy także wynosi 0. W tej sytuacji we wzorze na θ_1 pojawia się symbol nieoznaczony $\frac{0}{0}$ i niestety nie ma możliwości ustalenia wartości θ_1. W sytuacji gdy proponowany algorytm został wykorzystany do planowania całej trajektorii nogi robota można posiłkować się takimi rozwiązaniami jak przyjęcie wartości średniej kąta z poprzedniej $n-1$ i następnej $n+1$ chwili czasu:

$$\theta_1^n = \frac{\theta_1^{n-1} + \theta_1^{n+1}}{2} \tag{3.14}$$

gdzie n to bieżąca chwila czasu. Jeśli jest to niemożliwe ze względu na brak wyniku obliczeń na krok do przodu (przy planowaniu trajektorii na bieżąco) można podstawić poprzednią $n-1$ wartość położenia kąta θ_1 za wartość bieżącą.

$$\theta_1^n = \theta_1^{n-1} \tag{3.15}$$

W przypadku analizowanego robota kroczącego ta druga metoda daje bardzo dobre rezultaty. Wynika to z ograniczonych wymagań co do dokładności. Poza tym ta opisana sytuacja w praktyce zachodzi bardzo rzadko, bo nogi robota na ogół nie osiągają te pozycji podczas chodzenia.

3.3 Warstwa sprzętowa układu sterowania

System sterowania robotem kroczącym charakteryzuje się mnogością elementów wykonawczych i czujników pomiarowych. W przypadku projektowania sprzętowej części stanowiska robota utrudnieniem jest równoczesne sterowanie osiemnastoma układami serwomechanizmów. Podobna złożoność w aplikacjach robotyki czy mechatroniki występuje rzadko. Typowe ramię robota montażowego to "tylko" sześć serwomechanizmów. Otwartość projektu stanowiska laboratoryjnego determinuje przygotowanie konstrukcji łatwej dla przyszłych modyfikacji i rozbudowy.

Podczas projektowania stanowiska laboratoryjnego sześcionożnego robota kroczącego wzięto pod uwagę dwie architektury sprzętowej warstwy układu sterownia:

- rozproszonego systemu sterowania składającego się z wielu małych komunikujących się ze sobą układów i jednego układu nadrzędnego,
- systemu składającego się z jednego mikroprocesorowego układu sterującego i układu FPGA w warstwie pośredniej pomiędzy układem sterującym, a elementami wykonawczymi i pomiarowymi.

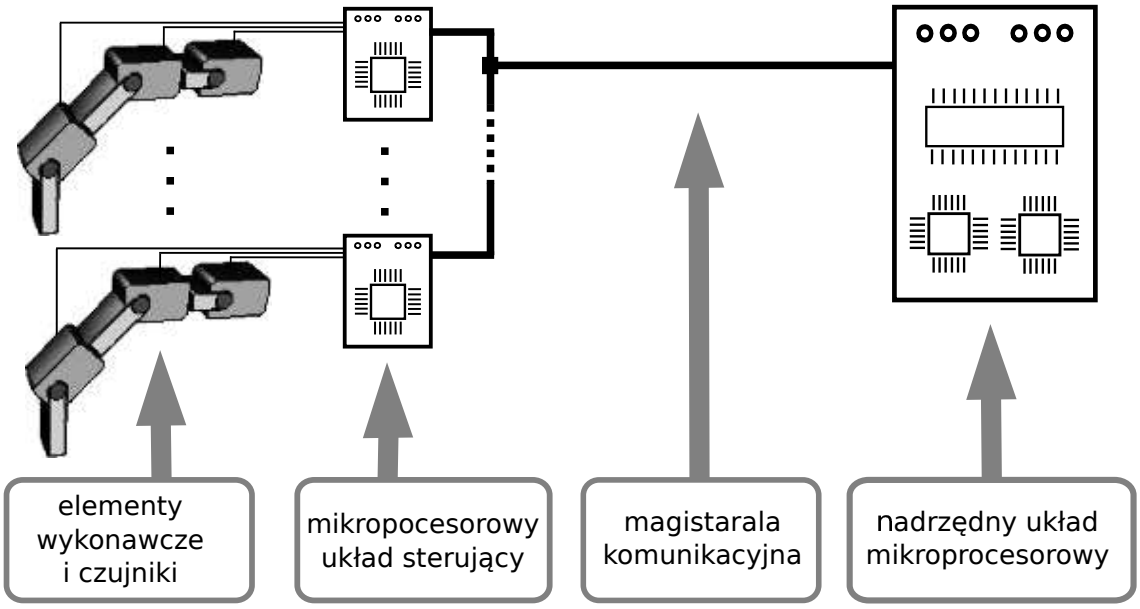
Pierwsze rozwiązanie polega na budowie sieci małych układów mikroprocesorowych sterujących poszczególnymi komponentami systemu i układu nadrzędnego zarządzającego nimi z wykorzystaniem magistrali komunikacyjnej [5]. Dosyć naturalną wydaje się koncepcja, w której mały sterownik mikroprocesorowy nadzoruje pracę tylko jednej nogi. Byłby on odpowiedzialny za:

- obliczenie odwrotnego zadania kinematyki nogi robota,
- generację sygnałów sterujących dla serwomechanizmów,
- odczyt wartości położenia kątowego przegubów,
- obliczenie prostego zadania kinematyki nogi robota,
- komunikację z układem nadrzędnym.

Każda noga robota sześcionożnego zostałaby wyposażona w jeden taki sterownik. Całość byłaby zarządzana z poziomu nadrzędnego układu sterowania, który odpowiadałby za:

- generację trajektorii końcówek nóg robota,
- nadzór nad pracą sterowników bezpośrednich,
- interfejs z użytkownikiem,
- logowanie wyników pracy.

W systemach o strukturze rozproszonej istotnym elementem jest medium i protokół komunikacji. W przypadku robotów kroczących o małych rozmiarach istnieje szeroka gama dostępnych rozwiązań, które można by zastosować [30]. Dobrym wyborem wydaje się być użycie magistrali RS-232, przemysłowej magistrali szeregowej RS-485 lub standardu stosowanego w przemyśle motoryzacyjnym czyli protokołu CAN w wersji CANOpen lub Profibus. Rysunek 3.10 przedstawia schemat ideowy systemu o architekturze rozproszonej zastosowanej w aplikacji robota kroczącego. Największą zaletą tego typu architektury jest



Rysunek 3.10: Schemat budowy rozproszonego systemu sterowania robotem kroczącym

jej skalowalność. Bardzo łatwo można uzupełniać ją o dodatkowe komponenty, przy czym projektant ma pełną swobodę podczas doboru elementów składowych, w tym architektury mikroprocesora zastosowanego w mikroprocesorowym układzie sterującym. Jest to o tyle ważne, że niektóre komponenty układu robota mogą wymagać specjalnych rozwiązań. Na przykład układ sterujący nogą robota może zostać zrealizowany za pomocą 16-bitowego mikrokontrolera ogólnego zastosowania, natomiast komponent obsługujący system kamery wizyjnej będzie wymagał zastosowania dedykowanych architektur DSP. Jedynym ograniczeniem podczas projektowania komponentów układu sterującego jest przepustowość i możliwość obsługi zastosowanej magistrali komunikacyjnej.

Rozbicie funkcjonalnych możliwości systemu sterującego pozwala na opracowanie i implementację zaawansowanych algorytmów pozwalających na pracę robota bez zatrzymań podczas awarii jednego lub wielu komponentów. Awaria jednego z komponentów pomiarowych lub awaria jednej nogi nie wyklucza całkowicie możliwości dalszej pracy robota, a jedynie ogranicza jego funkcjonalność. W pewnym sensie przypomina to sposób funkcjonowania organizmów biologicznych. W świecie istot żywych dysfunkcja pewnych

elementów organizmu niekoniecznie oznacza całkowite wykluczenie i śmierć. Oznacza jednak konieczność adaptacji do nowych warunków i ograniczonych możliwości. Temat ten z całą pewnością jest ciekawym polem do dalszych prac i analiz.

Największą wadą architektury rozproszonej jest jej względna złożoność, która zazwyczaj prowadzi do zwiększonej awaryjności i generuje problemy na etapie projektowania i implementacji.

Drugim rozważanym rozwiązaniem budowy sprzętowej warstwy systemu sterującego jest zastosowanie systemu mikroprocesorowego z układem FPGA w warstwie pośredniej [21]. Układy FPGA ze względu na swoje niekwestionowane zalety są coraz częściej stosowane w systemach sterujących [44]. Ich dwie cechy spowodowały, że z powodzeniem można je wykorzystać do budowy systemu sterującego sześcionożnym robotem kroczącym [41]. Pierwsza z nich do stosunkowo duża ilość uniwersalnych wejść i wyjść. Mogą one zostać wykorzystane zarówno do sterowania dużą ilością układów serwomechanizmów jak i innych elementów w przyszłości po rozbudowie robota. Drugą cechą układów FPGA, którą wykorzystuje się podczas realizacji omawianego projektu, jest implementacja równoległych struktur. Właściwość ta pozwala na implementację na przykład osiemnastu niezależnych sterowników układu serwomechanizmu, czy kilku niezależnych komponentów obsługujących czujniki analogowo-cyfrowe.

Ze względu na złożoność obliczeń w układzie FPGA raczej nie powinno się implementować algorytmów typu proste czy odwrotne zadanie kinematyki[2]. Układ FPGA w tej strukturze będzie głównie odpowiedzialny za:

- generację sygnału sterującego dla serwomechanizmu,
- odczyt wartości położenia kątowego przegubów,
- komunikację z układem mikroprocesorowym.

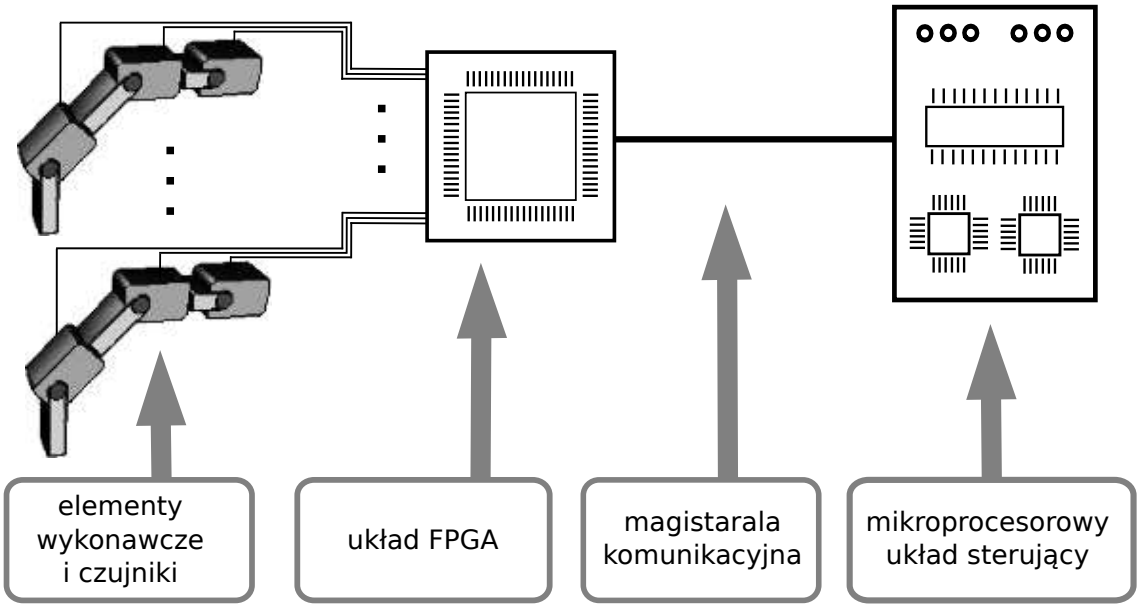
Układ mikroprocesorowy będzie wykonywał większość prac obliczeniowych i związanych z analizą danych w tym:

- obliczenie odwrotnego zadania kinematyki nogi robota,
- obliczenie prostego zadania kinematyki nogi robota,
- generację trajektorii końcówek nóg robota,
- komunikację z układem FPGA,
- interfejs z użytkownikiem,
- logowanie wyników pracy.

Schemat układu o takiej strukturze został przedstawiony na rysunku 3.11. Największe zalety tego typu architektury systemu sterującego to stosunkowo prosta budowa i sam fakt zastosowania układu FPGA. Jego użycie otwiera ogromne możliwości dla konstruktora systemu, które byłyby trudne do osiągnięcia z wykorzystaniem samych mikroprocesorów (zwłaszcza w dziedzinie przetwarzania sygnałów [57]). Jego wady to:

- mniejsza skalowalność (w porównaniu do architektury rozproszonej),
- brak naturalnej dekompozycji funkcjonalnej co utrudnia projektowanie, testowanie i przyszłą rozbudowę.

[2]Autor pracy nie twierdzi, że jest to niemożliwe [57]. Implementacja tego typu algorytmu w układzie FPGA najprawdopodobniej jest możliwa, ale ze względu na nakład pracy, rozmiar koniecznego do zastosowania układu scalonego (w przypadki sześciu nóg) i związanych z tym kosztów, prawdopodobnie będzie nieopłacalna.



Rysunek 3.11: Schemat budowy systemu sterowania robotem kroczącym z układem FPGA w warstwie pośredniej

W układzie zbudowanym w niniejszej pracy wykorzystano architekturę z układem FPGA. Zastosowano układ firmy Xilinx z rodziny Spartan 3. Podczas większości eksperymentów rolę układu mikroprocesorowego pełni komputer klasy PC stosowany wymiennie z układami wbudowanymi z procesorami o architekturze PowerPC i ARM9.

3.4 Oprogramowanie sterujące stanowiska laboratoryjnego

W warstwie sprzętowej systemu sterowania zdefiniowano składnik pełniący funkcje:

- nadrzędnego układu mikroprocesorowego – dla architektury rozproszonej,
- mikroprocesorowego układu sterującego – dla architektury z układem FPGA.

W obydwu zdefiniowanych architekturach składnik ten pełni funkcję nadrzędną i jest odpowiedzialny za realizację najbardziej złożonych, a zarazem najbardziej różnorodnych zadań. W tego typu układach mikroprocesorowych celowe wydaje się zastosowanie systemu operacyjnego. Przemawia za tym kilka zalet.

W układach mikroprocesorowych systemy operacyjne stosuje się ze względu na [48]:

Obsługę wielozadaniowości czyli obsługę wielu zadań działających na procesorze równolegle lub pseudo-równolegle. System operacyjny udostępnia odpowiednie mechanizmy, które pozwalają na implementację procesów i wątków oraz na prawidłową obsługę komunikacji międzyprocesowej (ang. IPC - Inter-Process Communication). System zajmuje się także szeregowaniem uruchomionych zadań na dostępnych procesorach.

Wirtualizację pamięci czyli kontrolę nad dostępem zadań obliczeniowych do pamięci operacyjnej. Dzięki wprowadzeniu pamięci wirtualnej system operacyjny może wychwycić błędne odwołania zadania do adresów pamięci spoza przydzielonego mu zakresu. Pozwala to głównie na zabezpieczenie pozostałych zadań obliczeniowych przed uszkodzeniem lub nieuprawnioną modyfikacją należących do nich danych co mogłoby doprowadzić do ich nieprawidłowego działania, a w efekcie zawieszenia pracy całego

układu. W przypadku wystąpienia tego typu błędu nie będzie działać tylko jedno, niepoprawnie napisane zadanie, a nie wszystkie zadania.

Wykorzystanie sterowników sprzętu czyli standardowych bibliotek pozwalających na obsługę typowego sprzętu i protokołów komunikacji, dostarczanych wraz z systemem operacyjnym. Dzięki sterownikom udaje się uniknąć nakładu pracy na pisanie oprogramowania obsługującego typowe urządzenia.

Możliwość użycia gotowego oprogramowania dostępnego od innych producentów lub społeczności. Pozwala to na znaczne przyspieszenie pracy poprzez częściowe zastosowanie gotowych rozwiązań komercyjnych lub rozwiązań typu OpenSource.

Niestety większość systemów operacyjnych dostępnych na rynku oprócz opisanych powyżej zalet wprowadza do układu brak determinizmu czasowego. Podczas implementacji systemu sterującego obiektem mechatronicznym stosujemy dyskretne algorytmy sterowania [16]. Jednym z parametrów tego typu algorytmów jest stały (niezmienny) i precyzyjnie określony krok dyskretyzacji T_s pozwalający na zdefiniowanie upływu czasu w postaci wzoru:

$$t = T_s k \tag{3.16}$$

gdzie k jest liczbą całkowitą. W praktyce oznacza to, że kolejne kroki (iteracje) dyskretnego algorytmu sterowania powinny zostać uruchomione w stałych odstępach czasu o wartości T_s, a każde zaburzenie tego parametru jest równoważne z wprowadzaniem zakłóceń do pracy układu i może doprowadzić do niestabilności algorytmu i całego obiektu. Klasyczne systemy operacyjne (Microsoft Windows, GNU/Linux, MacOS X) zostały dostosowane głównie do pracy biurowej, gdzie opisane wcześniej wymaganie nie musi być przestrzegane. Do celów sterowania wykorzystuje się systemy czasu rzeczywistego wyposażone w dedykowane systemy operacyjne czasu rzeczywistego, które pozwalają na implementację specjalnych zadań czasu rzeczywistego [11]. Konstrukcja systemu operacyjnego związana z określonym celem sprawia, że zadania czasu rzeczywistego najlepiej jak to tylko możliwe przestrzegają narzucone im rygory czasowe [39].

Definiuje się dwa podstawowe rodzaje systemów czasu rzeczywistego [52] i [51]:

- **Systemy o twardych wymaganiach czasowych (ang. hard real-time systems)** są to systemy spełniające wymagania czasowe w sposób rygorystyczny to znaczy spełniony jest warunek według, którego odpowiedź powinna być opracowana przed upływem pewnego zadeklarowanego czasu.
- **Systemy o miękkich wymaganiach czasowych (ang. soft real-time systems)** są to systemy, dla których spełniony jest warunek według, którego odpowiedź powinna być opracowana z pewnym średnim opóźnieniem czasowym, przy czym nie musi być spełniony wymóg określenia maksymalnego czasu trwania opóźnienia.

Często wyróżnia się także rodzaj pośredni, czyli system o solidnych wymaganiach czasowych (ang. firm real-time systems), dla którego definiuje się obydwa parametry to znaczy zarówno średni czas uzyskania odpowiedzi jak i maksymalny czas uzyskania odpowiedzi.

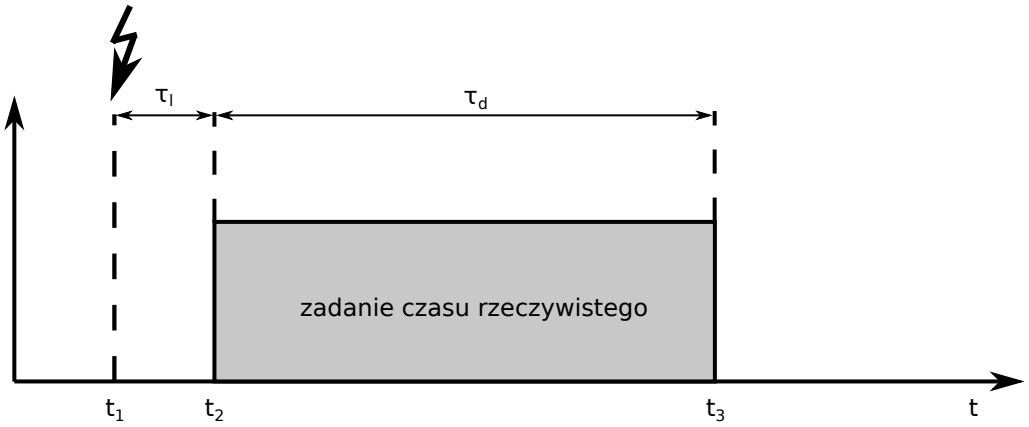
W praktyce podczas pracy z systemem czasu rzeczywistego często rozważa się wartości dwóch parametrów czasowych [4]. Są to:

- opóźnienie (ang. latency),
- rozrzut (ang. jitter).

Ilustracje tych dwóch pojęć w omawianym kontekście znajdują się na rysunkach 3.12 i 3.13.

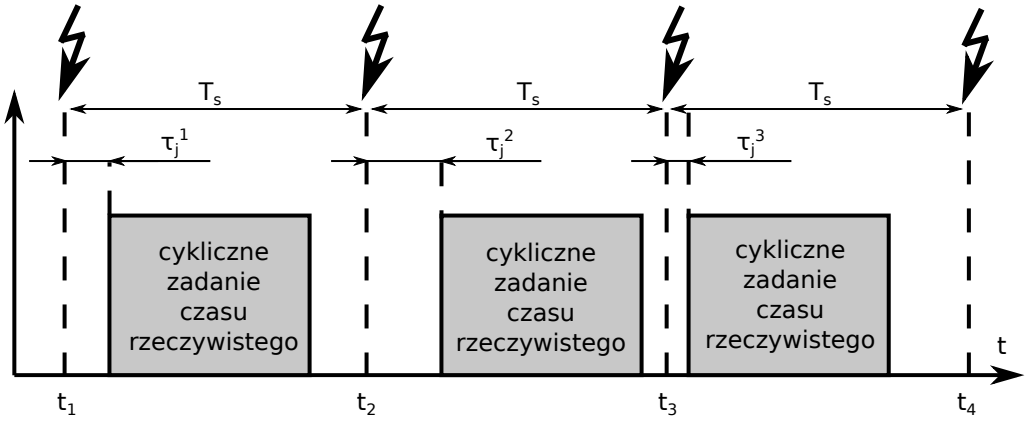
Pojęcie opóźnienia wiąże się z czasem jaki jest wymagany przez system czasu rzeczywistego by zareagować, przy czym poprzez reakcję rozumiemy uruchomienie odpowiedniej

procedury. W chwili czasu t_1 doszło do żądania uruchomienia zadania czasu rzeczywistego (patrz rysunek 3.12). Właściwe wykonywanie zadania rozpoczęto w chwili czasu t_2. Długość odcinka czasu τ_l oznacza opóźnienie, a długość odcinka czasu τ_d oznacza czas wykonywania zadania czasu rzeczywistego. W systemach mikroprocesorowych najmniejsze opóźnienia uzyskuje się stosując przerwania sprzętowe. W układach sterowania ten parametr będzie miał szczególne znaczenie przy implementacji elementów algorytmu sterującego, będących reakcją na zdarzenia (ang. event-based). Wielokrotnie realizuje się w ten sposób różne formy zabezpieczeń na przykład w układach mechatroniki obsługę sygnału pochodzącego od tak zwanych wyłączników krańcowych czyli układów zabezpieczających układ przed opuszczeniem przestrzeni roboczej. Przekroczenie zdefiniowanego obszaru pracy może zakończyć się poważną awarią, uszkodzeniem układu lub obsługujących go ludzi. Stąd odpowiednio szybka reakcja na zmianę sygnału jest niezwykle istotna.



Rysunek 3.12: Ilustracja opóźnienia τ_l (ang. latency)

Rozrzut także jest formą opóźnienia czasowego ale pojawiającą się tylko w przypadku zadań cyklicznych. Na rysunku 3.13 w chwilach czasu t_1, t_2, t_3 i t_4 doszło do żądania wywołania cyklicznego zadania czasu rzeczywistego (np. dyskretnego algorytmu sterującego). Odstęp pomiędzy chwilami czasu $t_i - t_{i-1}$ powinien wynosić T_s i być równy czasowi dyskretyzacji algorytmu sterującego. W praktyce algorytm jest uruchamiany z pewnym zmiennym opóźnieniem τ_j, którego wartość nazywamy rozrzutem. To zmieniające się opóźnienie sprawia, że odstęp czasowy pomiędzy kolejnymi wywołaniami zadania cyklicznego za każdym razem jest inny.



Rysunek 3.13: Ilustracja rozrzutu (ang. jitter)

Opóźnienie i rozrzut są ze sobą powiązane i ich wartości w systemie czasu rzeczywistego powinny być zbliżone. Niestety ze względu na szczegóły technicznej implementacji zadań wywoływanych zdarzeniem i zadań cyklicznych w różnych systemach czasu rzeczywistego wartości czasów mogą od siebie znacznie odbiegać. Należy także pamiętać, że te dwie wartości są parametrami całego systemu czasu rzeczywistego i mogą się różnić w zależności od:

- zastosowanej architektury i częstotliwości pracy procesora,
- rodzaju zastosowanej pamięci operacyjnej,
- rodzaju i dokładności zegara czasu rzeczywistego w systemie,
- zastosowanej magistrali dostępu do urządzeń peryferyjnych,
- wykorzystanych urządzeń peryferyjnych,
- zastosowanego systemu operacyjnego czasu rzeczywistego,
- i wielu innych czynników.

W przypadku użycia komputera klasy PC udaje się osiągać wartości rozrzutu w granicach $10\mu s$.

Podczas realizacji stanowiska dla robota kroczącego stosowano system operacyjny QNX Neutrino w wersji 6.5. O tym wyborze zdecydowano ze względu na dobre wsparcie techniczne ze strony producenta zarówno w postaci dokumentacji, podręczników jak i dostarczanych sterowników do typowych urządzeń i protokołów. System operacyjny QNX Neutrino stosuje architekturę mikrojądra to znaczy jądro systemu zostało zredukowane do niezbędnego minimum [23]. W przypadku systemów o jądrze monolitycznym elementy takie jak:

- sterowniki urządzeń,
- implementacja protokołów,
- system obsługi plików,

znalazłyby się w samym jądrze. W systemie QNX Neutrino zaimplementowano je tak, jak zwykłe procesy. Podejście to ma dwie istotne zalety. Po pierwsze znacznie poprawia bezpieczeństwo samego systemu operacyjnego. Jedną z zalet stosowania systemów operacyjnych jest wirtualizacja pamięci operacyjnej procesów, co znacząco ogranicza zwykłym procesom możliwość uszkodzenie pozostałych elementów systemu w przypadku awarii. Samo jądro systemu operacyjnego jako element zarządzający wirtualizacją jest poza jego kontrolą. Każdy dodatkowy element jądra systemu operacyjnego to dodatkowe potencjalne źródła błędu, które mogą uniemożliwić poprawną pracę całego systemu. Elementy wydzielone z jądra i działające w tak zwanej przestrzeni użytkownika (jako procesy) w przypadku awarii mogą być po prostu wywłaszczone i nie przerwą dalszej pracy systemu.

Drugą zaletą architektury mikrojądra jest jego skalowalność. W przypadku mniejszych układów mikroprocesorowych o małej ilości dostępnej pamięci możemy łatwo ograniczyć rozmiar systemu operacyjnego usuwając niepotrzebne elementy. Pozwala to na stworzenie systemu operacyjnego skrojonego na miarę konkretnego układu mikroprocesorowego z jego konkretnymi urządzeniami peryferyjnymi.

System operacyjny QNX Neutrino jest dostępny dla wielu architektur procesora, a dodatkowo producent udostępnia bibliotekę tak zwanych BSP (ang. Board Support Package) czyli gotowych rozwiązań dla zestawów mikroprocesorowych dostępnych na rynku. Aplikacja (algorytm sterujący) przygotowana dla jednego układu mikroprocesorowego

z systemem QNX Neutrino może być łatwo wdrożona dla innego układu. Tę właściwość wykorzystano podczas projektowania systemów sterowania sześcionożnym robotem kroczącym. Algorytm łatwiej prototypuje się w dużym systemie takim, jak komputer PC. W komputerze jest dostępne wiele urządzeń peryferyjnych: interfejs w postaci monitora i klawiatury, dysk twardy do logowania danych. Po opracowaniu, weryfikacji i przetestowaniu algorytmu stosunkowo małym nakładem pracy można go przenieść do systemu wbudowanego.

3.5 Oprogramowanie wspomagające projektowanie

Na etapie projektowania systemu sterowania chodem sześcionożnego robota kroczącego do wielu prac wykorzystywany jest pakiet MATLAB & Simulink wraz z wieloma rozszerzeniami (ang. toolbox and blockset). Pakiet został stworzony tak by wspierać i promować metodykę **projektowania z użyciem modelu (ang. Model-Based Design)**. Metodyka ta z grubsza polega na wykorzystaniu modeli matematycznych jako głównego nośnika informacji o projekcie. Model przewija się na każdym etapie rozwoju projektu przy czym każda kolejna faza sprawia, że jest on coraz dokładniejszy i coraz lepiej opisuje realizowany projekt [34] i [31].

Zgodnie z metodyką projektowania z użyciem modelu czas realizacji projektu dzielimy na kilka głównych etapów:

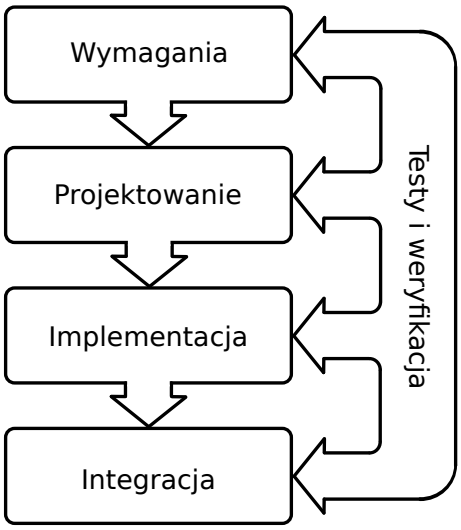
- wymagania (ang. requirements),
- projektowanie (ang. design),
- implementacja (ang. implementation),
- integracja (ang. integration),
- testy i weryfikacja (ang. tests and verification).

Na zakończenie każdego etapu możemy zdefiniować kamienie milowe w postaci:

- specyfikacji,
- projektu algorytmu,
- kodu źródłowego,
- prototypu,
- raportu z testów i weryfikacji.

Poniżej zamieszczono wyjaśnienie dotyczące etapów rozwoju projektu i na czym polegają osiągane w nich rezultaty. Rysunek 3.14 przedstawia powiązania pomiędzy poszczególnymi etapami rozwoju projektu realizowanego zgodnie z metodyką projektowania z użyciem modelu.

Wymagania. Na etapie tworzenia wymagań powstaje kilka dokumentów opisujących jakie rezultaty chce się osiągnąć i co jest celem projektu. Pierwszym z tworzonych dokumentów jest lista formalnych wymagań. Poprzez wymaganie jest rozumiany opis pożądanej cechy lub właściwości końcowego projektu. Zbiór takich wymagań stanowi cel, do którego się dąży. Podczas tworzenia wymagań ważne jest by miały one odpowiednią formę i spełniały kilka warunków. Za najistotniejsze uważa się warunki:



Rysunek 3.14: Powiązania pomiędzy poszczególnymi etapami rozwoju projektu realizowanego zgodnie z metodyką projektowania z użyciem modelu

- spójności, który mówi o tym, że jedno wymaganie odnosi się tylko jednej sprawy (cechy, właściwości),
- jednoznaczności, który mówi o tym, że wymaganie musi być zdefiniowane w sposób precyzyjny, na przykład niedopuszczalne są wymagania mówiące, że system ma być szybki, bez określenia konkretnych wartości,
- weryfikowalności, który mówi o tym, że dla każdego wymagania możliwe jest wykonanie testu, który jednoznacznie potwierdzi jego prawidłową realizację.

Dokument zawierający listę wymagań stanowi podstawę do tworzenia dalszych części specyfikacji. Dalszymi elementami specyfikacji będzie opis ogólnej struktury i zasady działania projektu, dekompozycja na mniejsze elementy i opis interfejsów. Najtrudniejszym krokiem będzie prawidłowa dekompozycja systemu. Podział na mniejsze elementy powinien zakładać, że każdy z nich może być realizowany niezależnie i testowany niezależnie. Pozwoli to na przykład na podział prac w większym zespole badawczym, ułatwi proces weryfikacji i znacznie uprości proces wyszukiwania i naprawy błędów (ang. debugging). Posiadając taki zestaw dokumentów możemy przystąpić do podziału prac i przejść do kolejnego etapu. Metodyka projektowania z użyciem modelu zakłada, że już na tak wczesnym etapie możemy stosować modele. Pozwolą one na przedstawienie dekompozycji i zdefiniowanie interfejsów. W modelu na tym etapie brakuje jeszcze właściwych algorytmów, ale zostanie już odpowiednio wyrażona hierarchia i zostaną narzucone ilości i rodzaje wejść/wyjść do każdego składnika.

Projektowanie. Posiadając specyfikację (częściowo wyrażoną w postaci modelu) można przystąpić do prac projektowych. Modelowanie pozwala na weryfikowanie nowych pomysłów i rozwiązań bez potrzeby budowania prototypu urządzenia czy żmudnej implementacji. W sytuacji, gdy przygotowujemy projekt algorytmu sterującego, konieczne jest także zbudowanie modelu obiektu lub przynajmniej jego substytutu. Dzięki temu można symulować pracę obiektu w różnych warunkach, w tym jego reakcję na projektowany algorytm sterujący.

Implementacja. Poprzez proces implementacji rozumiana jest faza przenoszenia wcześniej zrealizowanego modelu na układ fizyczny. Oznacza to budowę fizycznego prototypu. W szczególności oznacza przepisanie fragmentów algorytmu na kod źródłowy.

Możemy do tego wykorzystać oprogramowanie do automatycznej generacji kodu bezpośrednio z modeli. Stosowanie odpowiednich technik podczas projektowania algorytmu pozwala osiągnąć kod źródłowy o jakości porównywalnej do kodu napisanego przez doświadczonego programistę. Dodatkowo unika się błędów związanych z czynnikiem ludzkim podczas procesu kodowania. Wynikiem tego etapu są zrealizowane rzeczywiste składniki zdefiniowane na etapie tworzenia specyfikacji.

Integracja. Proces integracji sprowadza się do połączenia wcześniej zrealizowanych komponentów, dostrojenia i kalibracji całego systemu.

Testy i weryfikacja. Podczas realizacji projektu testy i weryfikacja wykonywane są po każdym etapie. Pozwala to na możliwie wczesne wykrywanie błędów i w efekcie przyspieszenie prac nad projektem. W sytuacji gdy został wykryty błąd należy wrócić do etapu, w którym został on popełniony.

W trakcie realizacji projektu sześcionożnego robota kroczącego starano się przestrzegać wytycznych metodyki projektowania z użyciem modelu. Oznacza to, że stworzono:

- wstępne założenia projektu (forma specyfikacji),
- zrealizowano symulacyjny model matematyczny sześcionożnego robota kroczącego, który pozwalał na weryfikację algorytmów sterowania,
- wykonano projekt algorytmu (algorytmów) w postaci modelu symulacyjnego,
- zweryfikowano działanie algorytmu korzystając z symulacyjnego modelu robota,
- zrealizowano przykładowy system sterowania robotem kroczącym i wykorzystano narzędzie do automatycznej generacji kodu, aby przenieść model algorytmu do kodu źródłowego,
- wykonano integrację systemu i zweryfikowano działanie algorytmu na rzeczywistym obiekcie.

Z perspektywy czasu należy szczególnie podkreślić, że metodyka pozwoliła przede wszystkim na szybką weryfikację wielu pomysłów algorytmów sterowania co autor pracy uważa za największą korzyść czerpaną z jej użycia. Drugą ważną zaletą metodyki projektowania z użyciem modelu jest przyspieszenie prac nad kodowaniem dzięki zastosowaniu narzędzi do automatycznej generacji kodu.

Rozdział 4

Model symulacyjny układu

Do budowy modelu symulacyjnego użyto pakietu narzędzi MATLAB&Simulink.

Zastosowane podejście wymaga odwzorowania w modelu symulacyjnym sterowanego obiektu, tak aby podczas symulacji sterownika badać jego odpowiedzi na sygnały sterujące. Zadanie to sprowadza się do przeprowadzenia procedury identyfikacji obiektu tak by odwzorować jego zachowanie za pomocą modelu matematycznego. W przypadku robota sześcionożnego dosyć łatwo można wprowadzić podział na składniki, z których każdy może być identyfikowany niezależnie. Dzięki temu zadanie to upraszcza się znacznie. Do realizacji modelu matematycznego pozwalającego ocenić zachowanie rzeczywistej konstrukcji mechatronicznej wytypowano następujące modele:

- serwomechanizm,
- obserwator stanu serwomechanizmu,
- zadanie proste kinematyki położenia,
- zadanie proste kinematyki prędkości.

Model serwomechanizmu pozwala na ocenę dynamicznego zachowania poszczególnych nóg robota. Przyjęto założenie, że konstrukcja nogi robota jest konstrukcją doskonale sztywną. W związku z tym, wszelkie efekty dynamiczne podczas ruchu nogi robota pochodzą od zjawisk wynikających z dynamiki serwomechanizmu.

Obserwator stanu serwomechanizmu, jest stosowany podczas analizy sygnału pomiarowego położenia kątowego pochodzącego z rzeczywistych eksperymentów z serwomechanizmem. Spełnia w układzie dwie funkcje:

- filtracji sygnału położenia kątowego serwomechanizmu,
- estymacji sygnału prędkości kątowej serwomechanizmu.

Zadanie proste kinematyki położenia pozwala na obliczenie położenia końcówki nogi robota w przestrzeni współrzędnych kartezjańskich. W tym przypadku, jako początek układu współrzędnych przyjęto punkt mocowania nogi robota do konstrukcji tułowia. Ten model pozwala na ocenę czy trajektoria ruchu końcówki nogi robota zaplanowana w układzie sterującym jest prawidłowo odwzorowana. Umożliwia także ocenę wpływu dynamiki serwomechanizmu na odwzorowywanie trajektorii.

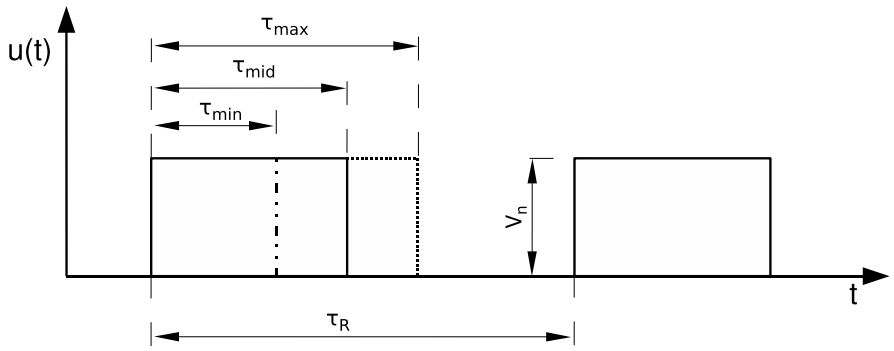
Zadanie proste kinematyki prędkości umożliwia obliczenie prędkości końcówki nogi robota w układzie współrzędnych kartezjańskich. Model ten służy do oceny prawidłowego odwzorowania planowanej trajektorii ze szczególnym uwzględnieniem granicznych prędkości ruchu końcówki nogi robota.

4.1 Identyfikacja serwomechanizmu

Podczas budowy konstrukcji mechanicznej robota sześcionożnego zastosowano 18 serwomechanizmów firmy Hitec model HS-475HB. Każdy z serwomechanizmów wyposażony jest w:

- silnik prądu stałego,
- mechaniczną przekładnię,
- potencjometr służący do pomiaru wartości położenia kątowego za przekładnią,
- układ elektroniczny sterujący silnikiem.

Sterowanie położeniem kątowym serwomechanizmu odbywa się przez podanie odpowiedniej wartości wypełnienia fazy sygnału wartości zadanej. Szczegółowe parametry sygnału wartości zadanej zostały przedstawione na rysunku 4.1. Położenie kątowe serwomechanizmu może przyjmować wartości od -60 do $+60$ stopni od położenia uznanego za położenie bazowe.



Rysunek 4.1: Sygnał wartości zadanej dla układu serwomechanizmu: $V_n = 3-4V$, $\tau_{max} = 2.1ms$ dla prawej skrajnej pozycji serwomechanizmu, $\tau_{mid} = 1.5ms$ dla bazowej pozycji serwomechanizmu, $\tau_{min} = 0.9ms$ dla lewej skrajnej pozycji serwomechanizmu, $\tau_R = 20ms$ częstotliwość sygnału sterującego

Do pomiaru rzeczywistego położenia kątowego serwomechanizmu służy potencjometr. Układ regulacji wbudowany w mechanizm korzysta z pomiaru spadku napięcia na potencjometrze by określić aktualną wartość położenia kątowego, a następnie wylicza wartość sterowania na podstawie wartości zadanej i aktualnego położenia korzystając z odpowiedniego algorytmu. Niestety algorytm ten nie jest znany to znaczy nie został opublikowany przez producenta. Korzystając jednak z odpowiednich metod identyfikacji można spróbować odtworzyć przybliżony model matematyczny tego algorytmu i zastosować go podczas symulacji [10] i [40].

Do przeprowadzenia procedury identyfikacji omawianego serwomechanizmu zbudowano odpowiednie stanowisko kontrolno-pomiarowe. W skład stanowiska wchodzi:

- komputer klasy PC z systemem operacyjnym czasu rzeczywistego QNX Neutrino RTOS,
- karta pomiarowa RT-DAC4 z możliwością:
  - generacji sygnału typu PWM,
  - obsługą enkoderów inkrementalnych,

  - odczytu poziomu napięcia analogowego poprzez przetworniki analogowo-cyfrowe,

- enkoder inkrementalny z podstawką umożliwiającą sztywne mocowanie do układu serwomechanizmu,

Korzystając z możliwości tego stanowiska przeprowadzono serię eksperymentów pomiarowych opisanych poniżej. W praktyce najistotniejsze okazuje się zastosowanie systemu pozwalającego na implementację zadań uruchamianych z rygorem twardego czasu rzeczywistego oraz poprawna implementacja zadania sterującego i zadania zapisującego wyniki pomiarów.

4.1.1 Analiza statyczna

Do prawidłowej interpretacji wyników identyfikacji konieczna jest analiza statycznego zachowania serwomechanizmu. W tym celu przeprowadzono eksperyment zadawania kolejnych wartości sygnału wartości zadanej położenia kątowego (sygnał PWM), a następnie przeprowadzano pomiar:

- położenia kątowego serwomechanizmu przy pomocy enkodera inkrementalnego,
- spadku napięcia na potencjometrze serwomechanizmu przy pomocy woltomierza,
- spadku napięcia na potencjometrze serwomechanizmu przy pomocy kanału analogowego karty pomiarowej.

Pomiary wykonywano po ustabilizowaniu się położenia kątowego serwomechanizmu, tak by nie uwzględniały one dynamiki obiektu, dla całego zakresu pracy układu. Cały eksperyment został przeprowadzony dla czterech losowo wybranych serwomechanizmów co pozwoliło zbadać różnice występujące pomiędzy seryjnymi elementami układu.

Uzyskane w ten sposób punkty pomiarowe wykorzystano do określenia dwóch relacji. Są to:

- relacja wejściowa $\tau = f(\alpha)$, gdzie τ oznacza czas trwania sygnału wysokiego w sygnale wartości zadanej serwomechnizmu, a α oznacza odpowiadającą mu wartość położenia kątowego,
- relacja wyjściowa $\alpha = f(u)$, gdzie α to także wartość aktualnego położenia kątowego, a u oznacza odpowiadający temu położeniu poziom spadku napięcia na potencjometrze zmierzony przez kartę pomiarową i przetwornik analogowo-cyfrowy.

Informacja o charakterze i współczynnikach funkcji wejściowej pozwala zadawać konkretną wartość położenia kątowego z poziomu programu komputerowego. Równocześnie informacja o charakterze i współczynnikach funkcji wyjściowej pozwala na określenie wartości położenia kątowego na podstawie wartości zwróconej przez kartę pomiarową. W obu przypadkach, na etapie wyznaczania tych funkcji, wartość położenia kątowego była odczytywana przy pomocy enkodera jako czujnika o liniowej charakterystyce przejściowej i dającego znacznie dokładniejszy pomiar niż zastosowany w układzie potencjometr.

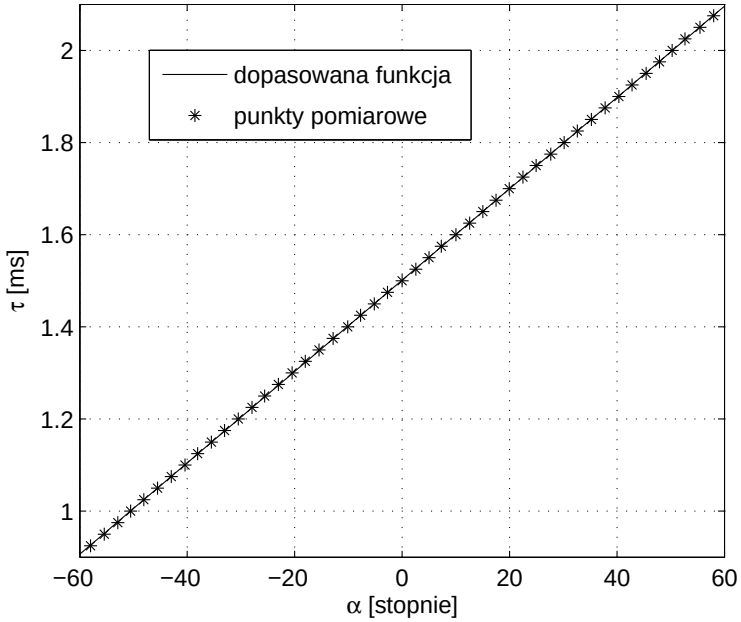
Do punktów pomiarowych, uzyskanych dla wszystkich czterech serwomechanizmów, dopasowano funkcje liniowe metodą regresji liniowej. Wykres przykładowej funkcji wejściowej i wyjściowej po linearyzacji wraz z punktami pomiarowymi przedstawiono odpowiednio na rysunku 4.2 i na rysunku 4.3. Maksymalny błąd dokładności działania zarówno sterowania jak i odczytu, a wynikający z linearyzacji wynosił $1,02\%$. Dla potrzeb dalszych prac z serwomechanizmami została obliczona jedna uśredniona funkcja wejściowa i jedna uśredniona funkcja wyjściowa. Uśredniona funkcja wejściowa stosowana dla dowolnego z osiemnastu wykorzystywanych serwomechanizmów ma postać:

$$\tau = 0,0098 * \alpha + 1,5006 \tag{4.1}$$

Natomiast uśredniona funkcja wyjściowa stosowana w dalszych pracach ma postać:

$$\alpha = -137,0609 * u + 120,8288 \tag{4.2}$$

Największy zaobserwowany podczas eksperymentów błąd wynikający z uśrednienia funkcji



Rysunek 4.2: Punkty pomiarowe wykorzystane do identyfikacji zależności $\tau = f(\alpha)$ przykładowego serwomechanizmu (gdzie τ oznacza czas trwania sygnału wysokiego w sygnale wartości zadanej serwomechnizmu, a α oznacza odpowiadającą mu wartość położenia kątowego) oraz wykres funkcji dopasowanej do punktów pomiarowych metodą regresji liniowej

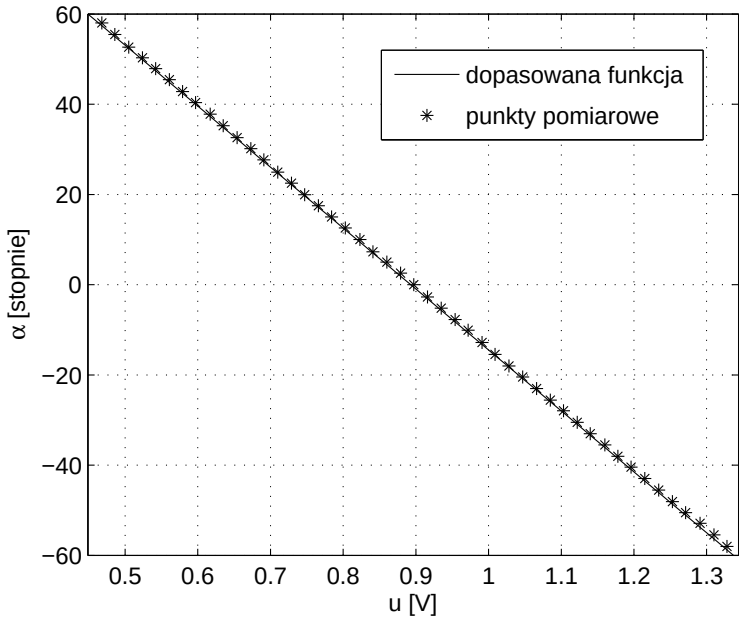
wynosił $3,4\%$ co stanowi równowartość $4,0814°$.

4.1.2 Analiza dynamiki serwomechanizmu

Dla zbadania dynamiki układu serwomechanizmu przeprowadzono eksperymenty polegające na wysyłaniu nowej wartości zadanej do układu co $1ms$ i odczycie wartości położenia kątowego (pośrednio poprzez odczyt spadku napięcia na potencjometrze przy pomocy kanału analogowego karty pomiarowej) przez pewien okres czasu. Ciągi wartości zadanej zostały zdefiniowane na podstawie wytycznych typowych dla procesu identyfikacji [24]. Wytyczne te można podsumować następująco:

- ciągi danych do identyfikacji powinny obejmować pełny zakres pasma przenoszenia identyfikowanego systemu,
- ciągi danych powinny reprezentować wszystkie typowe zachowania systemu jakie będą później wykorzystywane podczas pracy z systemem,
- do prawidłowej identyfikacji należy wykorzystać co najmniej dwa ciągi danych pochodzących z rzeczywistych eksperymentów:
  - ciąg danych służący do procedury identyfikacji,
  - ciąg danych służący tylko do weryfikacji.

Po przeprowadzeniu kilku eksperymentów z układem serwomechanizmu do właściwej identyfikacji wytypowano trzy ciągi sterowania, każdy trwający 30 sekund. Zostały one przedstawione na rysunkach 4.4, 4.5 i 4.6. Charakteryzują się one kolejno:



Rysunek 4.3: Punkty pomiarowe wykorzystane do identyfikacji zależności $\alpha = f(u)$ przykładowego serwomechanizmu (gdzie α to wartość aktualnej położenia kątowego, a u oznacza odpowiadający temu położeniu poziom spadku napięcia na potencjometrze) oraz wykres funkcji dopasowanej do punktów pomiarowych metodą regresji liniowej

- wolnymi zmianami wartości zadanej położenia kątowego, brakiem gwałtownych ruchów i skoków,
- szybkimi zmianami wartości zadanej położenia kątowego, obecnością gwałtownych ruchów i skoków,
- obecnością zarówno wolnych jak i szybkich zmian wartości zadanej położenia kątowego.

Do weryfikacji wytypowano jeden ciąg trwający 10 sekund, charakteryzujący się obecnością zarówno wolnych jak i gwałtownych zmian wartości zadanej.

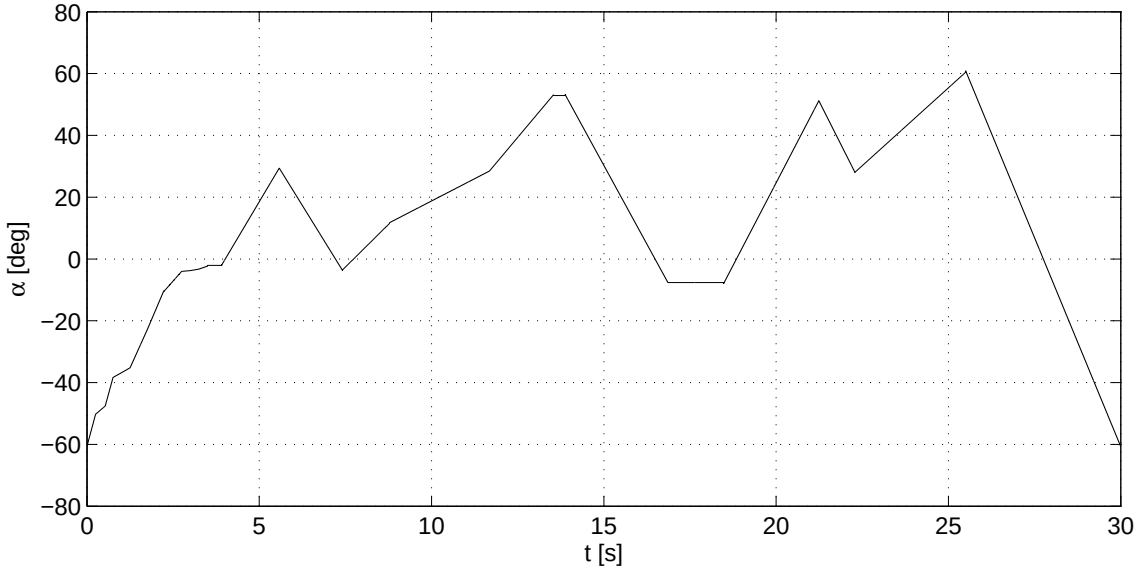
Wykresy zawierające odpowiedzi na wszystkie cztery, wytypowane ciągi sterowania dla serwomechanizmu zostały przedstawione na rysunkach 4.8, 4.9, 4.10 i 4.11.

W następnej kolejności wykonano prace związane z identyfikacją modelu serwomechanizmu. Do oceny jakości dobranego modelu zdefiniowano wskaźnik jakości w postaci:

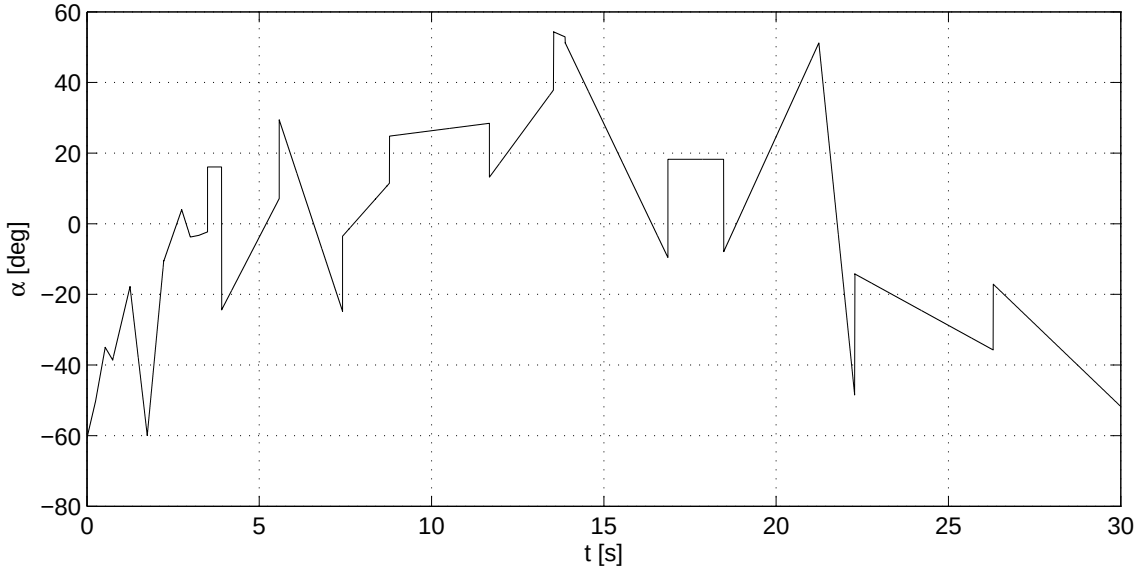
$$J = \sum_{k=0}^{N} (y(k) - y_m(k))^2 \tag{4.3}$$

gdzie wartość N to ilość próbek w ciągu weryfikującym, $y(k)$ to sygnał dyskretny odpowiedzi rzeczywistego układu serwomechanizmu, a $y_m(k)$ to sygnał dyskretny odpowiedzi identyfikowanego modelu serwomechanizmu.

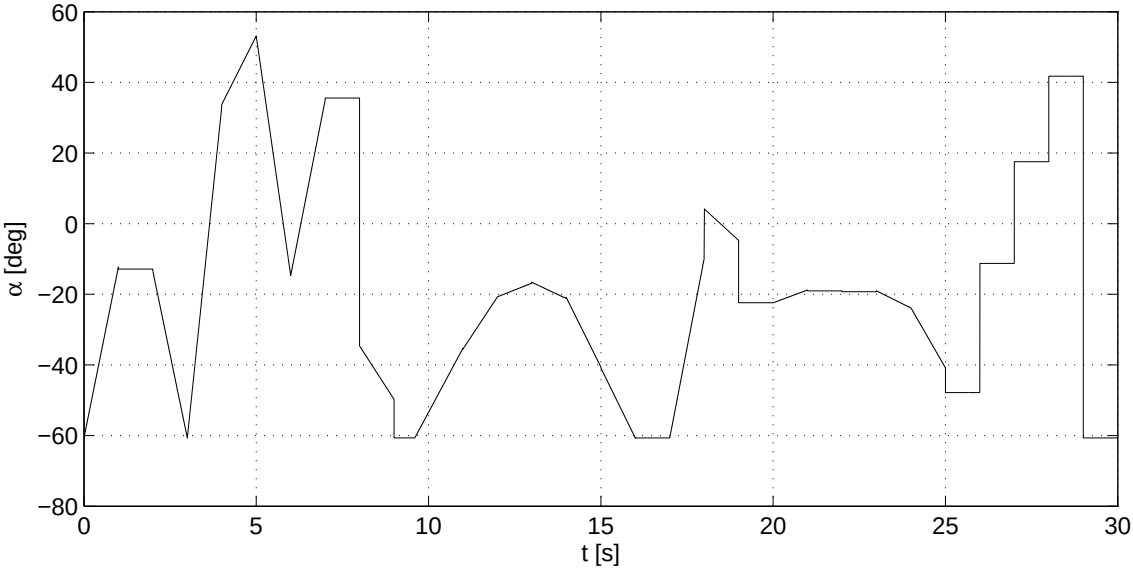
Identyfikację rozpoczęto od próby dopasowania modeli liniowych w postaci dyskretnych równań stanu do odpowiedzi rzeczywistego układu. Niestety nie przyniosło to oczekiwanych rezultatów. Natomiast szczegółowa analiza odpowiedzi układu na skoki o różnej wartości (patrz rysunek 4.12) podsunęła pomysł na strukturę modelu nieliniowego [54]. W odpowiedzi układu na skok można dostrzec dwie charakterystyczne fazy, tak jak zaznaczono na rysunku 4.13.



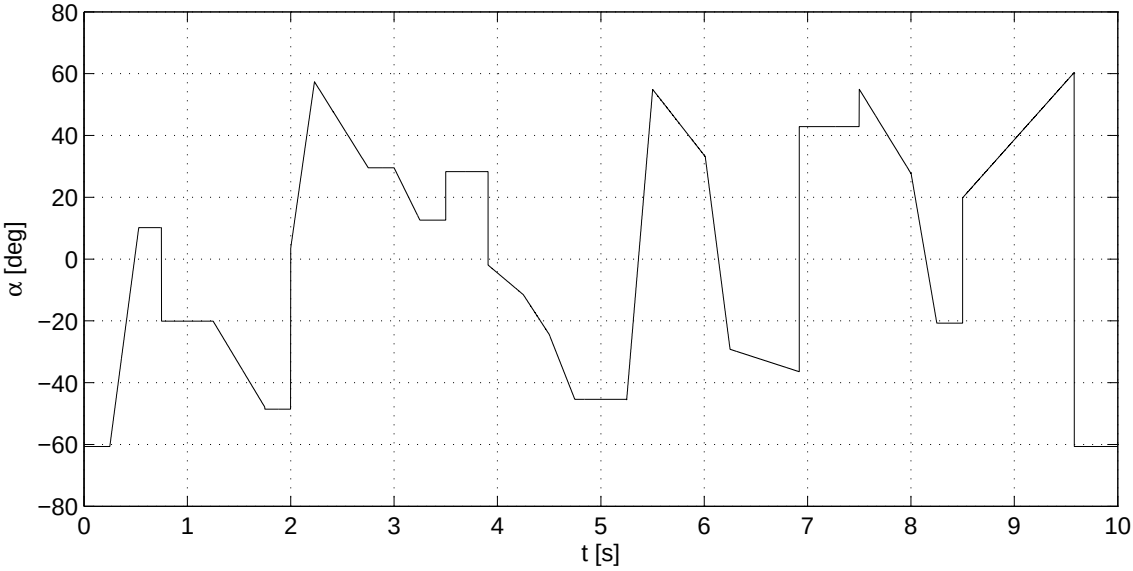
Rysunek 4.4: Sygnał wartości zadanej serwomechanizmu, przypadek wolno-zmienny (wartość sterowania została przeskalowana zgodnie z funkcją wejściową, patrz równanie (4.1))



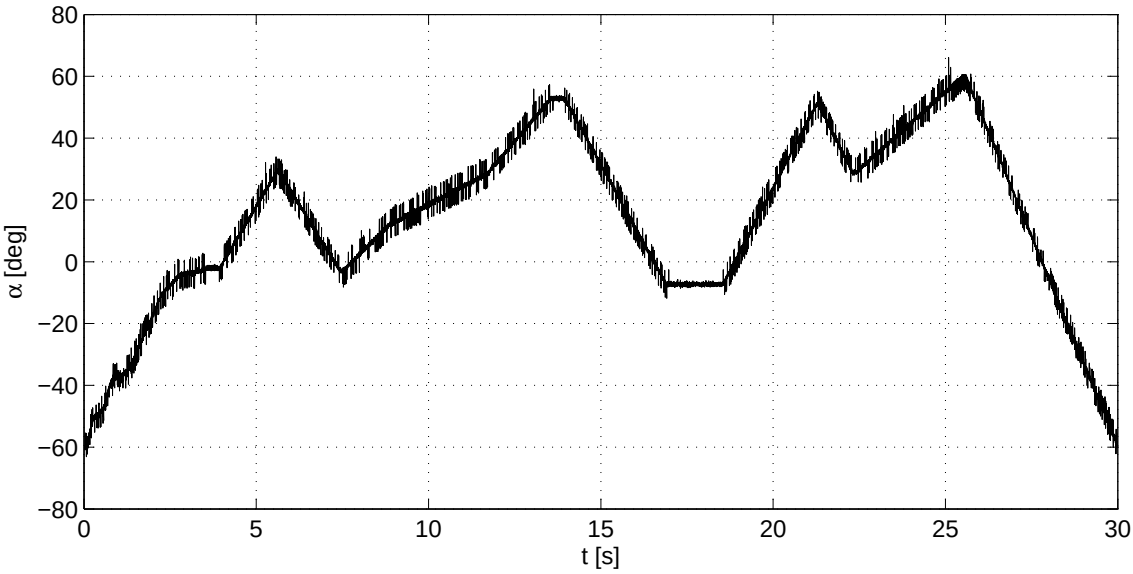
Rysunek 4.5: Sygnał wartości zadanej serwomechanizmu, przypadek szybko-zmienny (wartość sterowania została przeskalowana zgodnie z funkcją wejściową, patrz równanie (4.1))



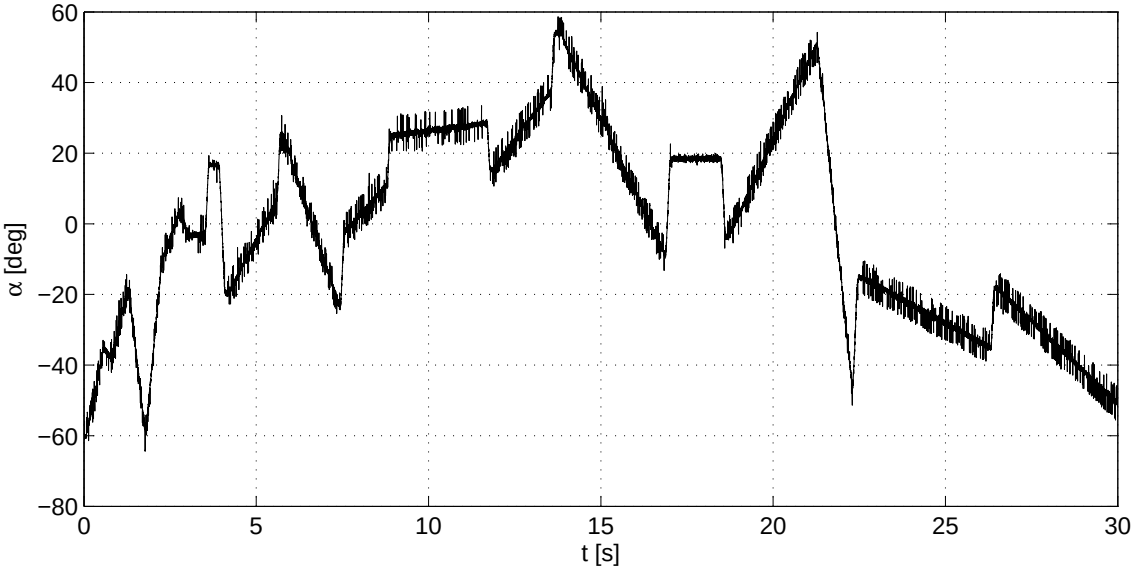
Rysunek 4.6: Sygnał wartości zadanej serwomechanizmu, przypadek ogólny (wartość sterowania została przeskalowana zgodnie z funkcją wejściową, patrz równanie (4.1))



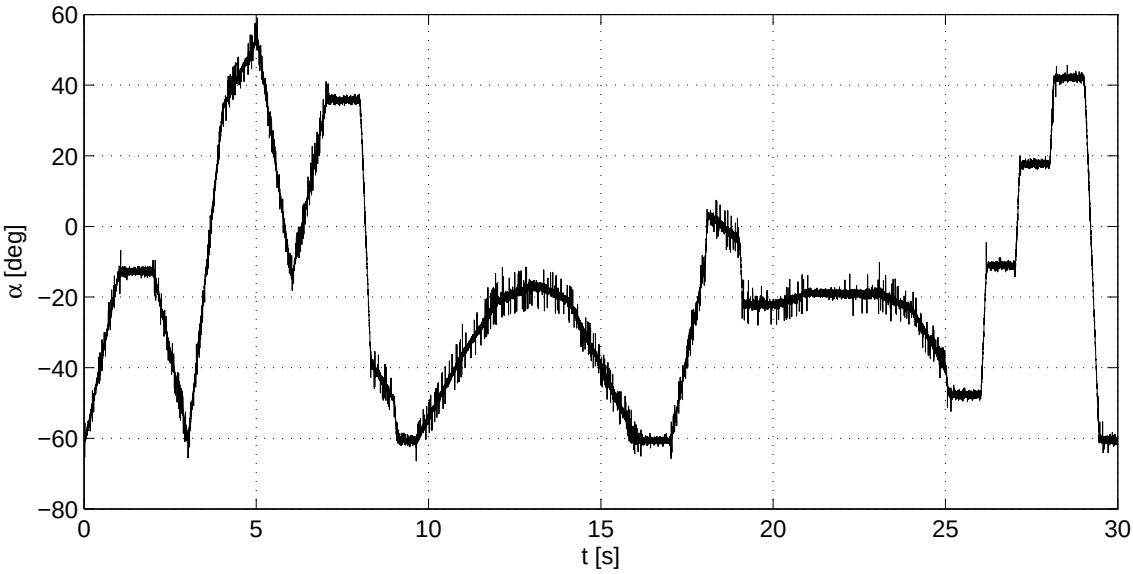
Rysunek 4.7: Sygnał wartości zadanej serwomechanizmu, przypadek weryfikacyjny (wartość sterowania została przeskalowana zgodnie z funkcją wejściową, patrz równanie (4.1))



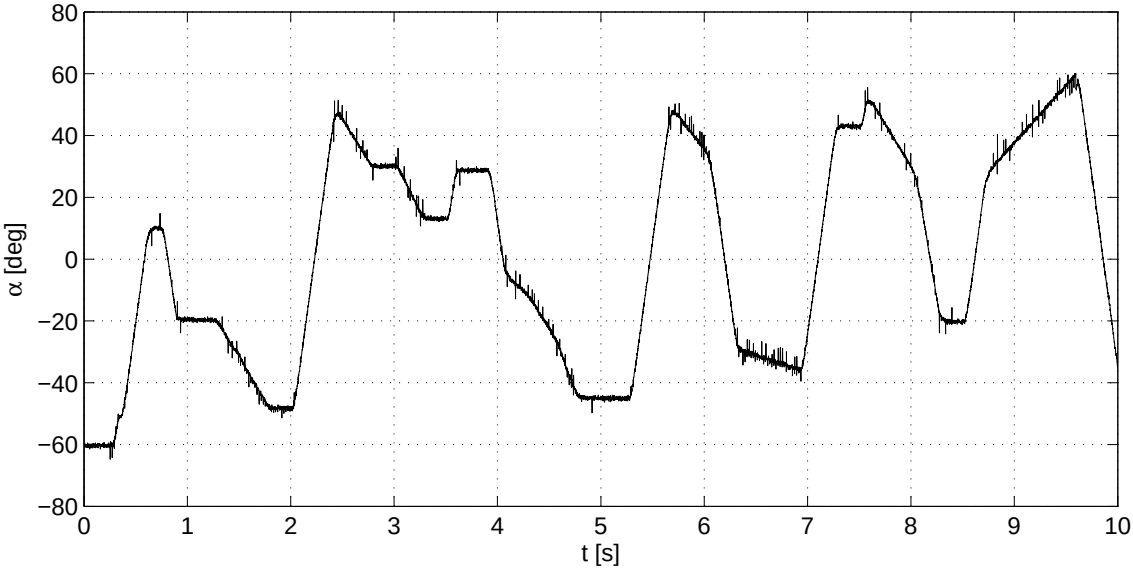
Rysunek 4.8: Odpowiedź serwomechanizmu na sygnał wartości zadanej, przypadek wolnozmienny (wartość spadku napięcia na potencjometrze serwomechanizmu została przeskalowana zgodnie z funkcją wyjściową, patrz równanie (4.2))



Rysunek 4.9: Odpowiedź serwomechanizmu na sygnał wartości zadanej, przypadek szybkozmienny (wartość spadku napięcia na potencjometrze serwomechanizmu została przeskalowana zgodnie z funkcją wyjściową, patrz równanie (4.2))



Rysunek 4.10: Odpowiedź serwomechanizmu na sygnał wartości zadanej, przypadek ogólny (wartość spadku napięcia na potencjometrze serwomechanizmu została przeskalowana zgodnie z funkcją wyjściową, patrz równanie (4.2))



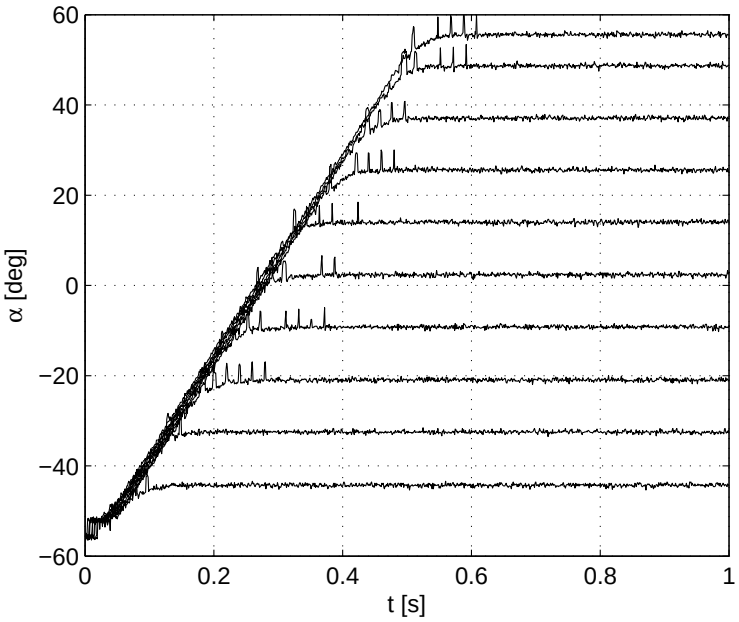
Rysunek 4.11: Odpowiedź serwomechanizmu na sygnał wartości zadanej, przypadek weryfikacyjny (wartość spadku napięcia na potencjometrze serwomechanizmu została przeskalowana zgodnie z funkcją wyjściową, patrz równanie (4.2))

Faza pierwsza Polega na ciągłej zmianie wartości położenia kątowego ze stałą prędkością kątową. Ruch taki odpowiada sytuacji gdy silnik prądu stałego zasilany jest stałym napięciem, a co za tym idzie sterowanie silnika nie zmienia się w trakcie tej fazy.

Faza druga W odpowiedzi serwomechanizmu można dostrzec podobieństwo do klasycznej odpowiedzi układu inercyjnego pierwszego rzędu na wymuszenie w postaci skoku jednostkowego. W układzie z regulatorem w zamkniętej pętli sprzężenia zwrotnego taka odpowiedź pojawi się w sytuacji, gdy układ regulatora realizuje algorytm regulatora proporcjonalnego.

Występowanie obu faz w takiej konfiguracji będzie typowe dla układu inercyjnego sterowanego regulatorem proporcjonalnym z ograniczeniem na wartość sterowania. Struktura takiego modelu została przedstawiona na rysunku 4.14. W przedstawionym układzie występują dwie charakterystyczne sytuacje odpowiadające poszczególnym fazom odpowiedzi z rysunku 4.13.

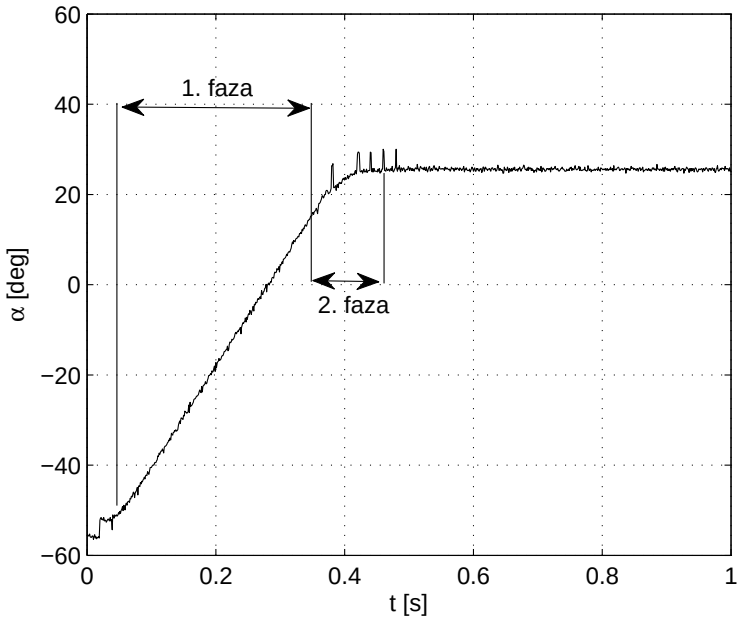
- Sytuacja gdy różnica pomiędzy wartością zadaną, a położeniem kątowym silnika jest stosunkowo duża odpowiada **fazie pierwszej** odpowiedzi na skok. Wartość sterowania przekracza dopuszczalną wartość i na układ działa jego maksymalna lub minimalna wartość.
- Fazie drugiej odpowiada sytuacja gdy wartość sterowania spada poniżej ograniczenia i układ zachowuje się jak klasyczny obiekt inercyjny.



Rysunek 4.12: Zestawienie odpowiedzi układu serwomechanizmu na sterowanie w postaci skoku o różnych wartościach

W układzie z rysunku 4.14 zostały zdefiniowane następujące parametry:

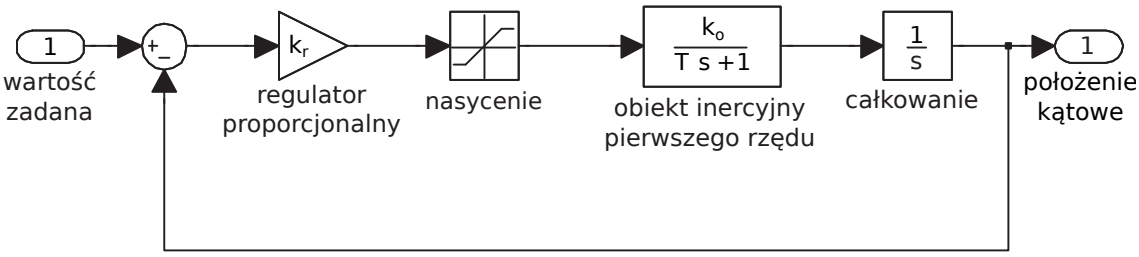
- k_r – współczynnik proporcjonalności regulatora,
- k_o – współczynnik wzmocnienia obiektu inercyjnego reprezentującego silnik prądu stałego,
- T – współczynnik inercji obiektu inercyjnego reprezentujący silnik prądu stałego,



Rysunek 4.13: Odpowiedź układu serwomechanizmu na wymuszenie w postaci skoku wartości zadanej z zaznaczeniem dwóch charakterystycznych faz odpowiedzi

- S_{min} – dolne ograniczenie na wartość sygnału sterującego,
- S_{max} – górne ograniczenie na wartość sygnału sterującego.

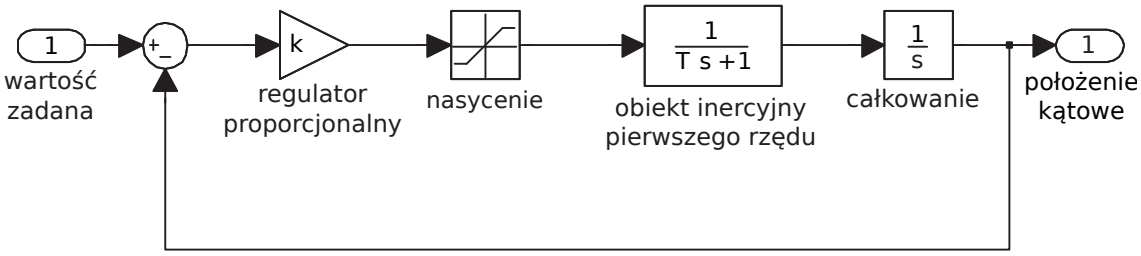
Proces identyfikacji polegał na znalezieniu wartości tych parametrów. Wykorzystano do tego narzędzia z pakietu MATLAB & Simulink, które pozwoliły na zastosowanie metod optymalizacji numerycznej. Procedura identyfikacji parametrów polegała na zdefiniowaniu zadania optymalizacji z ograniczeniami, w którym szukane jest minimum funkcji wskaźnika jakości (4.3).



Rysunek 4.14: Struktura modelu nieliniowego reprezentująca układ badanego serwomechanizmu

Podczas prac z modelem 4.14 okazało się, że dwa współczynniki: k_o i k_r, w praktyce mają taki sam wpływ na zachowanie układu, natomiast ich obecność wprowadza tożsamościową równość do algorytmów optymalizacji. W związku z tym, model został dodatkowo zmodyfikowany, tak by wyrugować jeden z dwóch współczynników. Końcową postać modelu przedstawiono na rysunku 4.15. Zależności pomiędzy współczynnikami w wersji poprawionej w stosunku do wersji pierwotnej wyglądają następująco:

$$\begin{aligned} k &= k_r k_o \\ S^*_{min} &= S_{min} k_o \\ S^*_{max} &= S_{max} k_o \end{aligned} \tag{4.4}$$



Rysunek 4.15: Uproszczona struktura modelu nieliniowego reprezentująca układ badanego serwomechanizmu

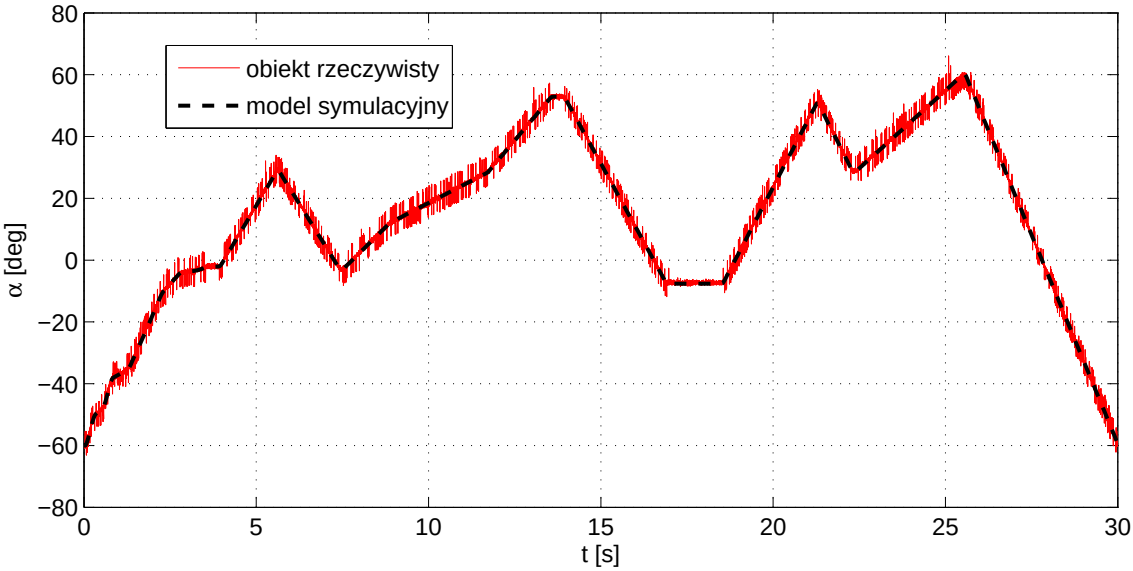
Przyjęto także następujące ograniczenia na wartości parametrów:

$$\begin{aligned} k &> 0 \\ T &> 0 \\ S_{min} &< 0 \\ S_{max} &> 0 \end{aligned} \tag{4.5}$$

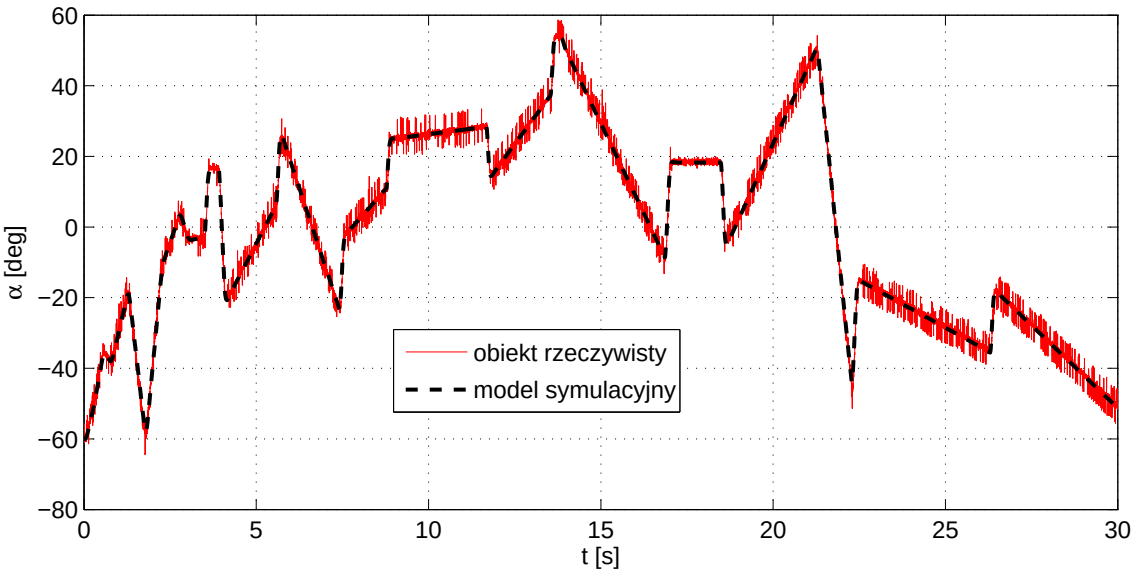
Po wielokrotnym przeprowadzeniu procedury optymalizacji z wykorzystaniem metody najmniejszych kwadratów (wstępna optymalizacja, odnalezienie sąsiedztwa minimum) i metody gradientowej (znalezienie ostatecznego rozwiązania) oraz dostępnych danych pomiarowych uzyskano satysfakcjonujące wyniki w postaci wartości parametrów:

$$\begin{aligned} k &= 0,0181 \\ T &= 20,8279 \\ S_{min} &= -220,854 \\ S_{max} &= 220,9851 \end{aligned} \tag{4.6}$$

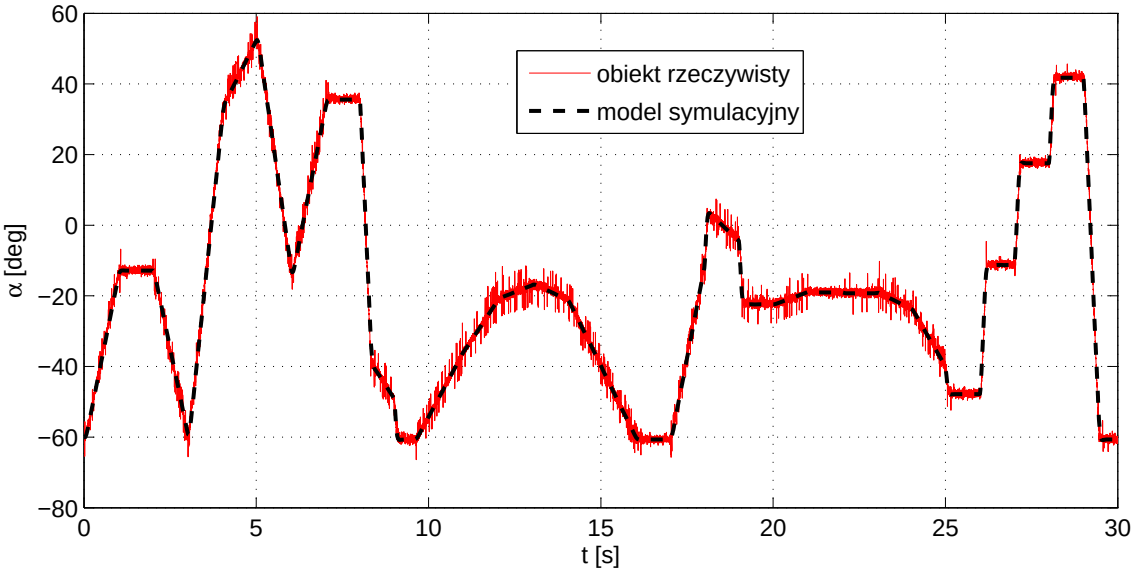
Na wykresach 4.16, 4.17, 4.18 i 4.19 można zobaczyć porównanie pomiędzy wynikami pomiarowymi pochodzącymi z eksperymentu z rzeczywistym obiektem, a wynikami symulacji opracowanego modelu ze znalezionymi wartościami parametrów.



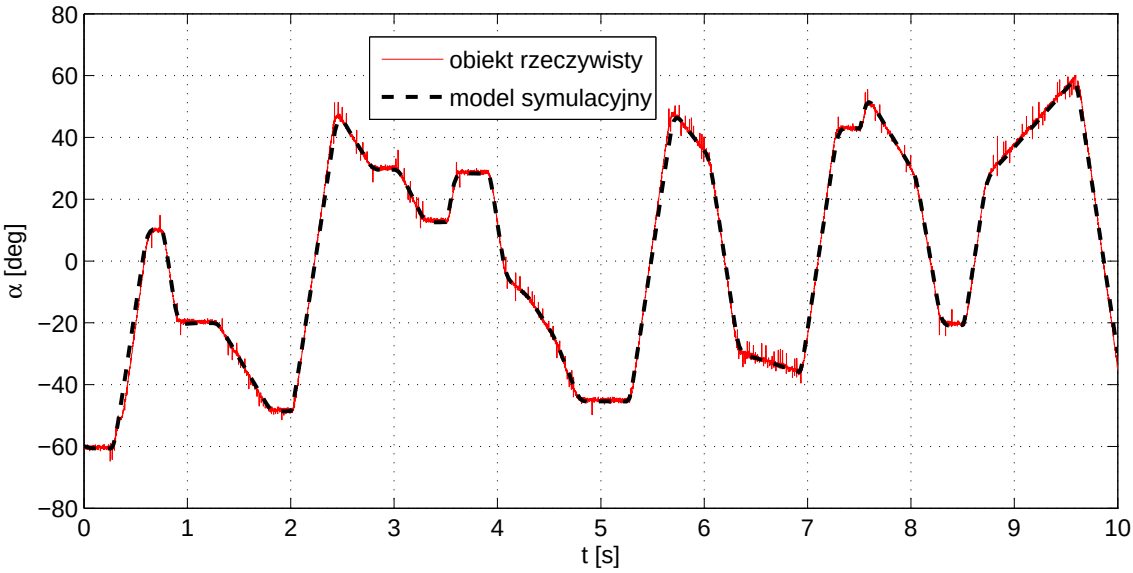
Rysunek 4.16: Porównanie wyników eksperymentu pomiarowego z wynikami symulacji zidentyfikowanego modelu, przypadek wolno-zmienny



Rysunek 4.17: Porównanie wyników eksperymentu pomiarowego z wynikami symulacji zidentyfikowanego modelu, przypadek szybko-zmienny



Rysunek 4.18: Porównanie wyników eksperymentu pomiarowego z wynikami symulacji zidentyfikowanego modelu, przypadek ogólny (przypadek był wykorzystany do właściwej identyfikacji)



Rysunek 4.19: Porównanie wyników eksperymentu pomiarowego z wynikami symulacji zidentyfikowanego modelu, przypadek ogólny do weryfikacji

4.2 Obserwator stanu serwomechanizmu

Dla analizy wyników pomiarowych pochodzących z badania zachowania robota zbudowano identycznościowy obserwator stanu układu serwomechanizmu. Algorytm obserwatora opiera się na teorii rozszerzonego filtru Kalmana natomiast nie jest jego pełną implementacją [43]. Podczas analizy danych obserwator pełni dwie funkcje:

- estymacji niemierzonego sygnału prędkości kątowej serwomechanizmu,
- filtracji rzeczywistego, zmierzonego sygnału położenia kątowego.

Dynamikę serwomechanizmu opisują równania:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\frac{1}{T}x_2 + \frac{1}{T}\mathrm{sat}\left(k(u - x_1), S^*_{min}, S^*_{max}\right) \\ y &= x_1 \end{aligned} \tag{4.7}$$

gdzie funkcję nasycenia oznaczoną symbolem sat opisuje równanie:

$$\mathrm{sat}(z, a, b) = \begin{cases} a, & z < a \\ z, & a \leq z \leq b \\ b, & z > b \end{cases} \tag{4.8}$$

W równaniach (4.7) symbolem x_1 oznaczono zmienną stanu reprezentującą położenie kątowe, symbolem x_2 zmienną stanu reprezentującą prędkość kątową układu serwomechanizmu, a u to sygnał sterowania układu czyli w naszym przypadku wartość zadana położenia kątowego. Równanie na wartość wyjściową y oznacza, że jedyną wartością mierzoną w układzie jest położenie kątowe. Pozostałe parametry występujące w równaniach (4.7) to znaczy: T, k, S^*_{min} i S^*_{max} mają interpretację fizyczną identyczną z tą jaką opisano w rozdziale 4.1.2.

Równania dynamiki serwomechanizmu można zapisać w formie:

$$\begin{aligned} \dot{x} &= f(x, u) \\ \dot{y} &= h(x) = Cx \end{aligned} \tag{4.9}$$

gdzie macierz wyjścia C przyjmuje postać:

$$C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

Przyjęto, że równanie nieliniowego, asymptotycznego obserwatora identycznościowego jest w postaci:

$$\dot{\widehat{x}} = f\left(\widehat{x}, u\right) + L\left(y - C\widehat{x}\right) \tag{4.10}$$

W praktyce zapis ten oznacza, że stan układu jest wyliczany z wykorzystaniem wcześniej zidentyfikowanego modelu nieliniowego układu (tak zwana faza predykcji algorytmu filtru Kalmana). Stan następnie jest korygowany o różnicę prognozy i rzeczywistej wartości zmierzonej części stanu (tak zwana faza korekty algorytmu filtru Kalmana) [49]. Wartość korekty jest przemnożona przez współczynnik L, którego wartość powinna być tak dobrana, by zapewnić asymptotyczną stabilność obserwatora.

Jak zapisano w równaniu (4.10) do estymacji stanu wykorzystywana jest między innymi funkcja $f(x, u)$ będąca częścią zidentyfikowanego modelu (patrz równania (4.7) i (4.9)). Oznacza to, że jakość działania obserwatora będzie zależała od dwóch elementów:

- prawidłowej identyfikacji modelu,
- prawidłowego doboru wartości współczynnika L.

Budowa tego typu obserwatora jest także testem poprawności wcześniej wykonanych prac.

Dobór wartości współczynnika L wymaga zbadania stabilności równania obserwatora. Występująca w nim funkcja $f(x, u)$ została przepisana do postaci:

$$f(x, u) = Ax + Bu + g(x, u) \tag{4.11}$$

czyli z równania stanu została wyodrębniona część nieliniowa w postaci funkcji $g(x, u)$. W tej wersji równania serwomechanizmu macierze A i B przyjmują postać:

$$A = \begin{bmatrix} 0 & 1 \\ -\dfrac{k}{T} & -\dfrac{1}{T} \end{bmatrix}, B = \begin{bmatrix} 0 \\ \dfrac{k}{T} \end{bmatrix} \tag{4.12}$$

Samo równanie obserwatora w wersji zlinearyzowanej wygląda następująco:

$$\dot{\widehat{x}} = A\widehat{x} + Bu + L\left(y - C\widehat{x}\right) \tag{4.13}$$

a po przekształceniu do postaci równania stanu:

$$\dot{\widehat{x}} = \left(A - LC\right)\widehat{x} + Bu + Ly \tag{4.14}$$

Równanie (4.14) będzie asymptotycznie stabilne gdy będzie spełniony warunek:

$$\operatorname{Re}\lambda\left(A - LC\right) < \sigma \tag{4.15}$$

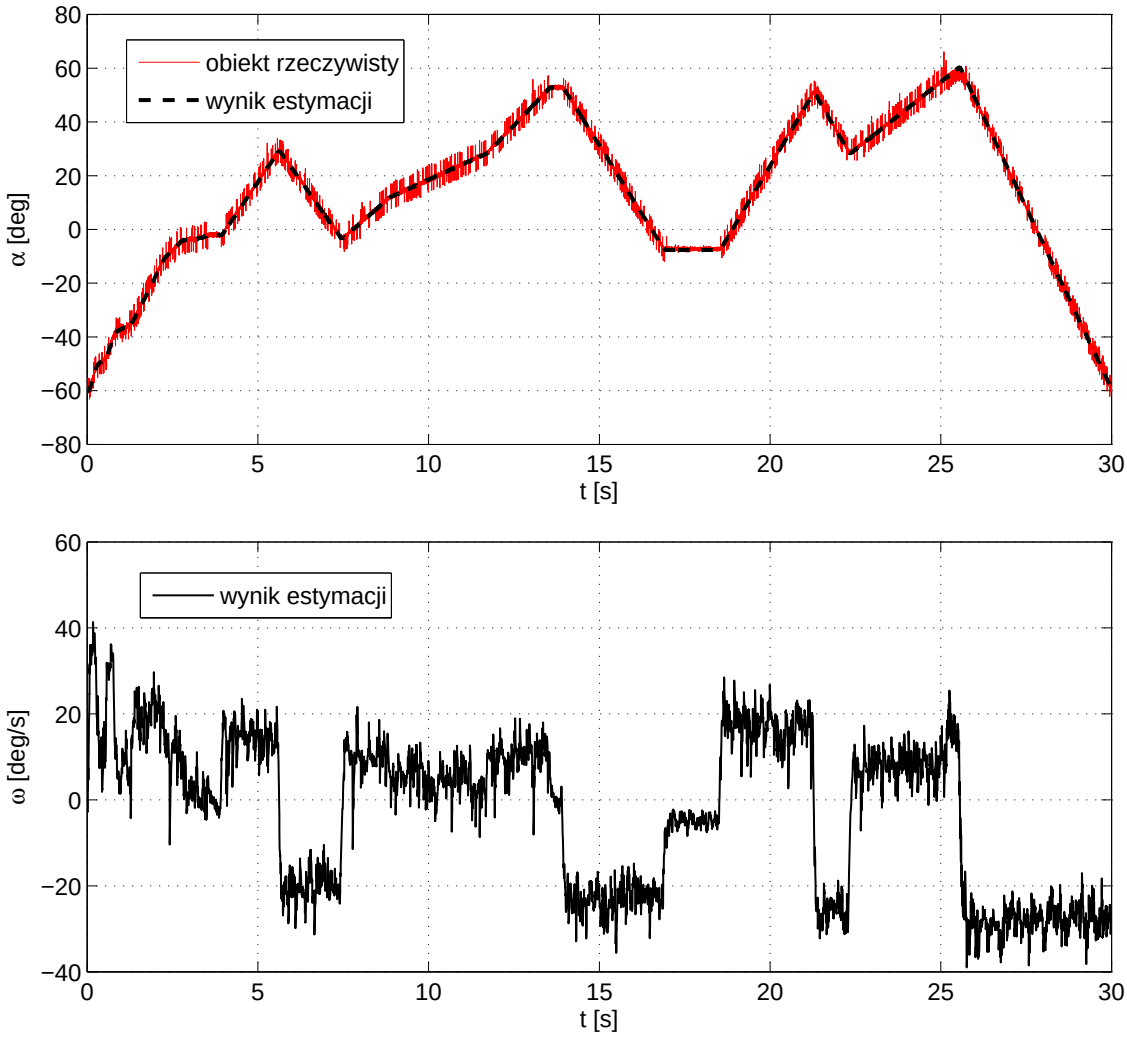
gdzie σ to dostatecznie mała liczba ujemna zapewniająca zanikanie błędu estymacji do zera. Liczba ta musi być na tyle mała by zniwelować wpływ nieliniowej części $g(x, u)$ pominiętej podczas linearyzacji układu.

Dla modelu przyjęto wartość współczynnika L jako:

$$L = \begin{bmatrix} 14,6291 \\ -763,2848 \end{bmatrix} \tag{4.16}$$

Rezultaty działania opracowanego regulatora ilustrują wykresy 4.20, 4.21, 4.22 i 4.23. Jakość działania obserwatora można zweryfikować porównując wartości zmierzonego położenia kątowego z wartościami za obserwatorem. Niestety w przypadku prędkości kątowej

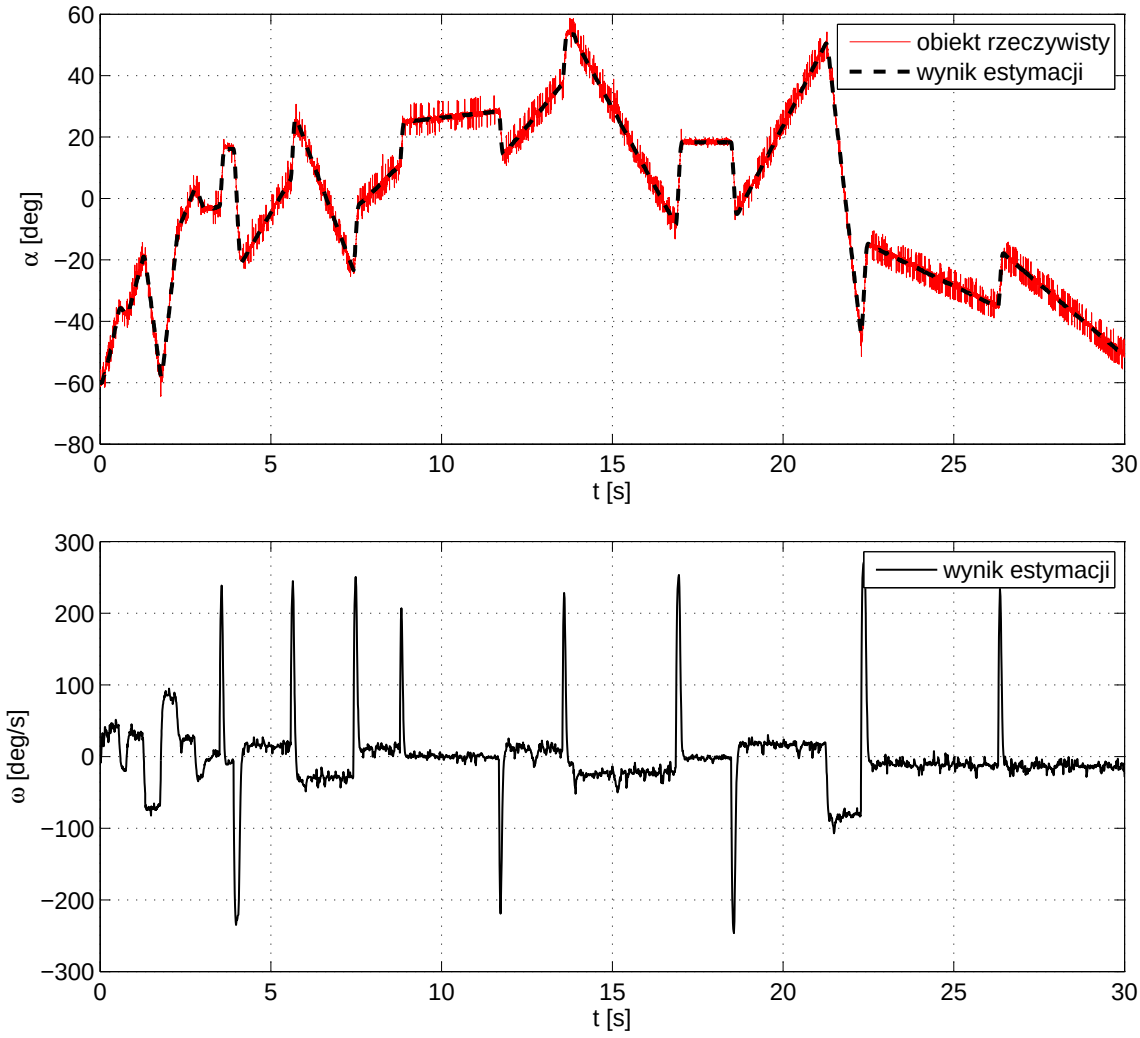
nie jesteśmy w stanie dokonać takiego porównania. W budowanym układzie robota sześcionożnego nie ma możliwości technicznych na dokonanie takiego pomiaru. Jedyne potwierdzenie jakości działania tej części obserwatora to weryfikacja czy wartość prędkości kątowej ω na wyjściu obserwatora nigdy nie przekracza maksymalnej prędkości kątowej dla serwomechanizmu podawanej przez producenta układu. Wartość ta według dokumentacji nie powinna przekraczać 300 deg/s co odpowiada maksymalnym wartościom pojawiającym się na przedstawionych wykresach (w odpowiedzi na skok).



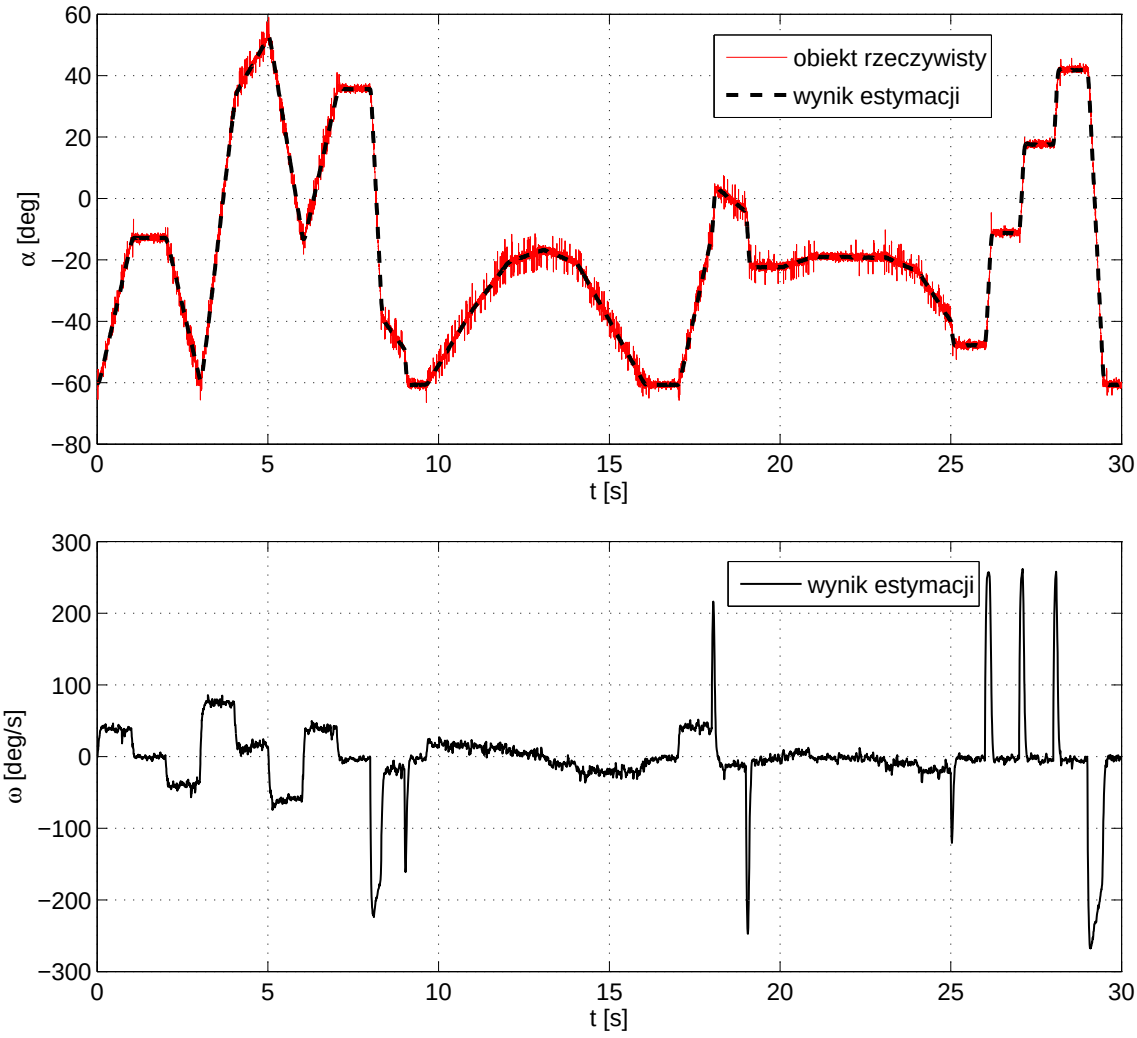
Rysunek 4.20: Wynik działania projektowanego obserwatora – odpowiedź algorytmu na zmierzony sygnał położenia kątowego, przypadek wolno-zmienny, α oznacza położenie kątowe, ω oznacza prędkość kątową

4.3 Proste zadanie kinematyki

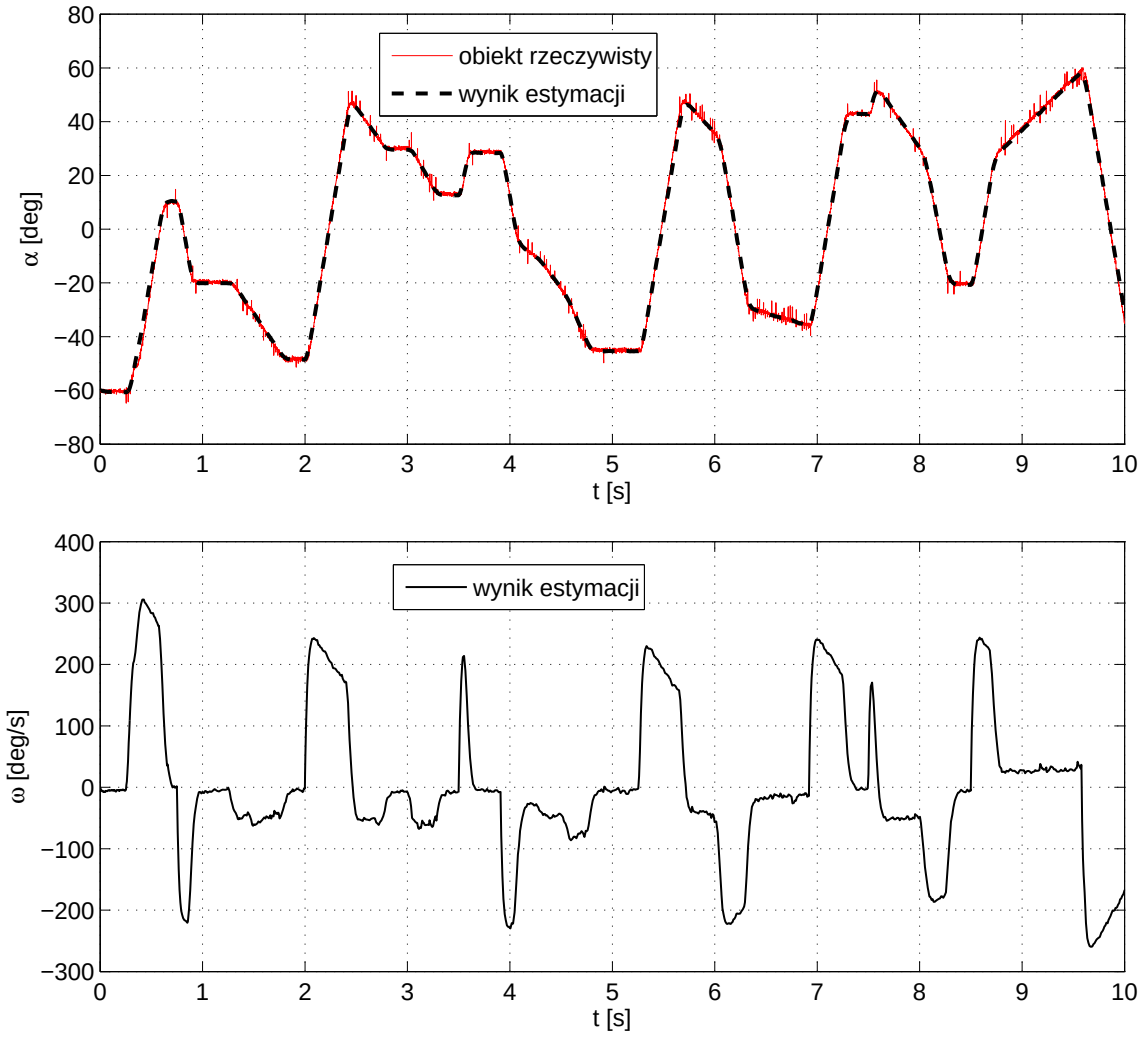
Podczas symulacji opracowanego modelu robota uwzględniana jest dynamika układu serwomechanizmu co pozwala na obserwację zmiany położenia kątowego i prędkości kątowej poszczególnych przegubów nóg robota. Wynik taki jest trudny do interpretacji, ponieważ sterując ruchem nóg robota najczęściej zwracamy uwagę tylko na położenie końcówki nogi w naturalnym dla człowieka układzie współrzędnych jakim jest układ kartezjański. W celu uproszczenia analizy wyników pracy układu zdecydowano o implementacji zadania



Rysunek 4.21: Wynik działania projektowanego obserwatora – odpowiedź algorytmu na zmierzony sygnał położenia kątowego, przypadek szybko-zmienny, α oznacza położenie kątowe, ω oznacza prędkość kątową



Rysunek 4.22: Wynik działania projektowanego obserwatora – odpowiedź algorytmu na zmierzony sygnał położenia kątowego, przypadek ogólny, α oznacza położenie kątowe, ω oznacza prędkość kątową



Rysunek 4.23: Wynik działania projektowanego obserwatora – odpowiedź algorytmu na zmierzony sygnał położenia kątowego, przypadek weryfikacyjny, α oznacza położenie kątowe, ω oznacza prędkość kątową

kinematyki prostej czyli opracowania funkcji matematycznej pozwalającej na przeliczenie współrzędnych naturalnych nogi robota, jakimi są położenia kątowe poszczególnych serwomechanizmów na współrzędne kartezjańskie końcówki nogi.

Pełne zadanie proste kinematyki polega na wyprowadzeniu tak zwanej macierzy przekształcenia [50]. Macierz ta przedstawia zależność pomiędzy dwoma układami współrzędnych: układem bazowym i układem docelowym, a w ogólnej formie przyjmuje postać:

$$H = \begin{bmatrix} R_{3\times3} & d_{3\times1} \\ f_{1\times3} & s_{1\times1} \end{bmatrix} \tag{4.17}$$

gdzie poszczególne elementy oznaczają:

- $R_{3\times3}$ – macierz rotacji mówiąca o orientacji układu docelowego względem układu bazowego,
- $d_{3\times1}$ – wektor przesunięcia środka układu docelowego względem środka układu bazowego,
- $f_{1\times3}$ – wektor perspektywy, który w przypadku analizy łańcucha kinematyki jakim jest noga robota przyjmuje wartość $[0 \quad 0 \quad 0]$,
- $s_{1\times1}$ – współczynnik skali, który w przypadku analizy łańcucha kinematyki przyjmuje wartość 1.

Dla dwóch układów współrzędnych oznaczonych jako O_0 i O_1 macierz przekształcenia z układu współrzędnych O_0 do układu O_1 oznaczamy jako H_0^1, natomiast operację odwrotną jako H_1^0. Dodatkowo dla macierzy przekształcenia zachodzi właściwość, że dla układów współrzędnych O_0, O_1, O_2 i macierzy przekształcenia H_0^1, H_1^2:

$$H_0^2 = H_0^1 H_1^2 \tag{4.18}$$

Z tej właściwości korzystamy wyprowadzając proste zadanie kinematyki dla łańcucha kinematyki definiując jeden układ współrzędnych powiązanych z każdym elementem łańcucha, a następnie wyprowadzając jedynie macierze przekształcenia dla sąsiednich układów współrzędnych H_i^{i+1}. Macierz przekształcenia pomiędzy końcowym elementem łańcucha, a bazowym układem współrzędnych O_0 wyliczamy ze wzoru:

$$H_0^N = \prod_{i=0}^{N-1} H_i^{i+1} \tag{4.19}$$

Wyprowadzenie macierzy przekształcenia dla dowolnych układów współrzędnych jest procedurą bardzo pracochłonną. Dla układów łańcucha kinematyki można zastosować tak zwaną notację Denavita-Hartenberga. Jest to stosunkowo prosty algorytm pozwalający na szybkie wyprowadzenie macierzy przekształcenia. Algorytm ten można zastosować tylko w sytuacji gdy mamy pewną dowolność w oznaczaniu układów współrzędnych powiązanych z kolejnymi przegubami łańcucha.

Notacja ta wymaga by układy współrzędnych spełniły dwa warunki:

- oś x_{i+1} jest prostopadła do osi z_i,
- oś x_{i+1} przecina oś z_i,

Jeśli te dwa warunki będą spełnione to macierz przekształcenia pomiędzy dwoma sąsiednimi układami przyjmie ogólną postać:

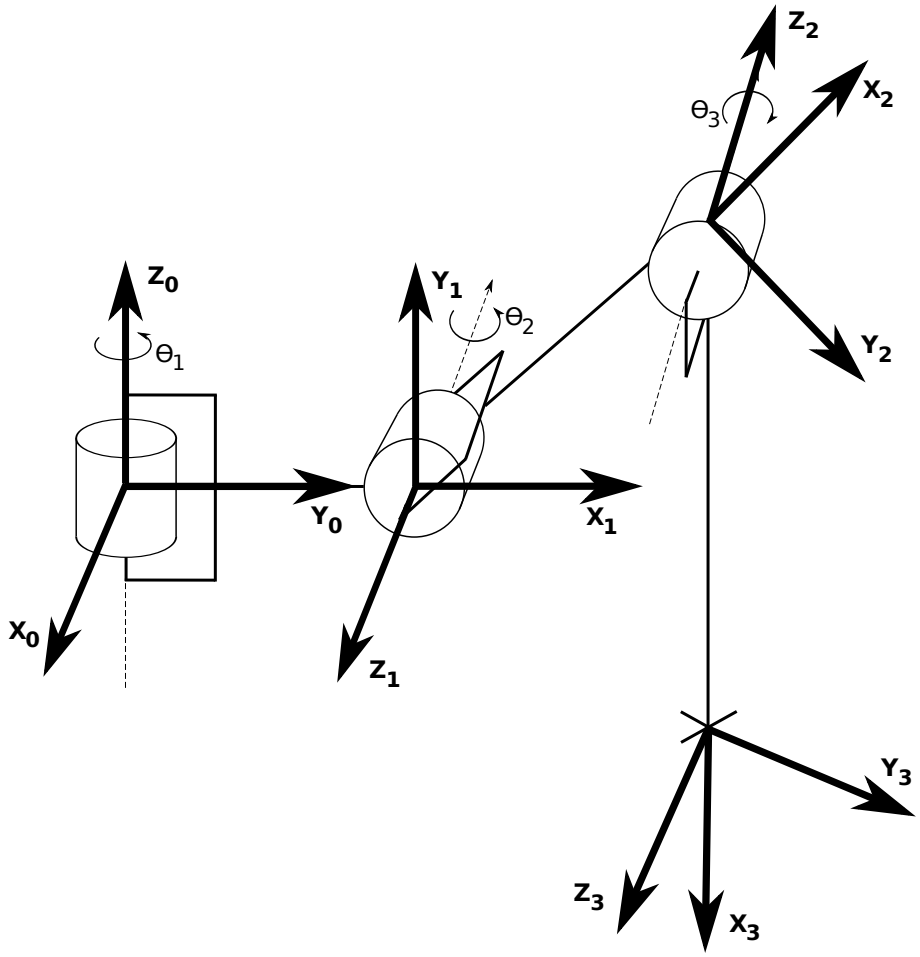
$$H_{i-1}^i = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.20}$$

gdzie poszczególne parametry oznaczają:

- a_i – długość członu,
- α_i – skręcenie członu,
- d_i – odsunięcie przegubu,
- θ_i – kąt przegubu.

Ponieważ macierz ta oznacza przekształcenie tylko dla jednego przegubu, który najczęściej jest albo obrotowy, albo pryzmatyczny to trzy parametry przyjmują wartość stałą, a jeden (α_i dla przegubu obrotowego, θ_i dla przegubu pryzmatycznego) jest zmienną. Zmienna ta zarazem jest jedną ze współrzędnych naturalnych układu.

Rysunek 4.24 przedstawia schemat nogi robota z zaznaczonymi układami współrzędnych dla każdego przegubu zgodnie z notacją Denavita-Hartenberga. Układ $O(X_0, Y_0, Z_0)$ jest układem bazowym w stosunku, do którego chcemy odnieść pozycję końcówki nogi robota. Początek układu bazowego został umieszczony w punkcie, w którym noga robota jest mocowana do podstawy robota.



Rysunek 4.24: Schemat budowy nogi robota sześcionożnego z zaznaczeniem osi współrzędny dla każdego przegubu zgodnie z notacją Denavita-Hartenberga

Dla zdefiniowanych układów współrzędnych możemy wyznaczyć listę parametrów, które następnie będzie można podstawić do wzoru (4.20). Lista parametrów dla kolejnych członów łańcucha kinematyki nogi robota jest podana w tablicy 4.1. W dalszej części

przegub	a_i	d_i	α_i	θ_i
1	37 mm	10 mm	$+90^\circ$	θ_1
2	57 mm	0	-180°	θ_2
3	160 mm	0	$+180^\circ$	θ_3

Tablica 4.1: Lista wartości parametrów przegubów dla nogi robota sześcionożnego, zgodna z notacją Denavita-Hartenberga

pracy będziemy korzystać z uproszczonego zapisu funkcji trygonometrycznych:

$$\begin{aligned} s_1 &= \sin(\theta_1) \\ s_2 &= \sin(\theta_2) \\ s_3 &= \sin(\theta_3) \\ \\ c_1 &= \cos(\theta_1) \\ c_2 &= \cos(\theta_2) \\ c_3 &= \cos(\theta_3) \end{aligned} \tag{4.21}$$

Po podstawieniu otrzymuje się trzy macierze przekształcenia dla sąsiadujących ze sobą układów współrzędnych:

$$H_0^1 = \begin{bmatrix} c_1 & 0 & s_1 & 37 \cdot c_1 \\ s_1 & 0 & -c_1 & 37 \cdot s_1 \\ 0 & 1 & 0 & 10 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H_1^2 = \begin{bmatrix} c_2 & s_2 & 0 & 57 \cdot c_2 \\ s_2 & -c_2 & 0 & 57 \cdot s_2 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.22}$$

$$H_2^3 = \begin{bmatrix} c_3 & s_3 & 0 & 160 \cdot c_3 \\ s_3 & -c_3 & 0 & 160 \cdot s_3 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Po przemnożeniu macierzy (4.22) otrzymuje rozwiązanie prostego zadania kinematyki dla nogi robota sześcionożnego w postaci:

$$\begin{aligned} H_0^3 &= H_0^1 H_1^2 H_2^3 = \\ &= \begin{bmatrix} c_1c_2c_3 + c_1s_2s_3 & c_1c_2s_3 - c_1s_2c_3 & s_1 & 37c_1 + 57c_1c_2 + 160c_1c_2c_3 + 160c_1s_2s_3 \\ s_1c_2c_3 + s_1s_2s_3 & s_1c_2s_3 - s_1s_2c_3 & -c_1 & 37s_1 + 57s_1c_2 + 160s_1c_2c_3 + 160s_1s_2s_3 \\ s_2c_3 - c_2s_3 & c_2c_3 + s_2s_3 & 0 & 57s_2 - 160c_2s_3 + 160s_2c_3 + 10 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{4.23}$$

W dalszych pracach na układem sterowania chodem robota sześcionożnego wykorzystywany będzie tylko wektor przesunięcia z całej macierzy przekształcenia. Wynika to z kształtu obszaru roboczego nogi i faktu, że końcówka nogi robota zawsze powinna być skierowana w dół podczas kroczenia, do czego będą stosować się wszystkie algorytmy przestawiania nóg. Nie wyklucza się jednak wykorzystania pozostałej części wyniku w dalszych pracach nad robotem.

Z macierzy (4.23) wynika, że wzory na współrzędne kartezjańskie końcówki nogi robota mają postać:

$$\begin{aligned} x &= 37c_1 + 57c_1c_2 + 160c_1c_2c_3 + 160c_1s_2s_3 \\ y &= 37s_1 + 57s_1c_2 + 160s_1c_2c_3 + 160s_1s_2s_3 \\ z &= 57s_2 - 160c_2s_3 + 160s_2c_3 + 10 \end{aligned} \tag{4.24}$$

4.4 Proste zadanie kinematyki prędkości

Kolejnym zaimplementowanym modelem robota sześcionożnego jest zadanie proste kinematyki prędkości. Model ten, tak samo jak i zadanie proste kinematyki położenia ułatwia interpretacje wyników działania algorytmów sterujących. Dzięki niemu na etapie analizy wyników można określić jaka była prędkość liniowa końcówki nogi robota we wszystkich trzech wymiarach przestrzeni kartezjańskiej. Ponadto model dostarcza informacji czy ruch nogi jest płynny to znaczy, czy nie dochodzi do gwałtownej zmiany kierunku ruchu. Ruch nogi pozbawiony płynności może co prawda prowadzić do skutecznego przemieszczania się robota sześcionożnego, ale wygląda nienaturalnie, a dodatkowo powoduje zwiększone zużycie energii.

Do obliczania prędkości liniowych końcówki nogi robota należy wprowadzić tak zwany jakobian manipulatora czyli macierz J_0^N o wymiarach $6 \times N$, gdzie N jest liczbą członów łańcucha kinematyki [50]. Gotowy jakobian można podstawić do wzoru:

$$\begin{bmatrix} v_0^N \\ \omega_0^N \end{bmatrix} = J_0^N \dot{q} \tag{4.25}$$

gdzie v_0^N oznacza wektor prędkości liniowych końcówki łańcucha kinematyki w bazowym układzie współrzędnych, ω_0^N oznacza wektor prędkości kątowych końcówki łańcucha kinematyki w bazowym układzie współrzędnych, a $\dot{q}$ to wektor prędkości poszczególnych przegubów w układzie zmiennych naturalnych. Ostatni element składa się z prędkości kątowych ω_i w przypadku przegubów obrotowych i z prędkości liniowych v_i w przypadku przegubów pryzmatycznych.

Ze wzoru (4.25) wynika, że część jakobianu zawiera elementy związane z obliczaniem prędkości liniowych końcówki łańcucha kinematyki, a część zawiera elementy związane z obliczaniem prędkości kątowych. W związku z tym wprowadzono podział na część J_v i część J_ω, każda o wymiarach $3 \times N$ przy czym:

$$J_0^N = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix} \tag{4.26}$$

Do wyrażenia definicji tych dwóch części jakobianu należy zdefiniować jeszcze dwa pojęcia:

- z_i jest to wektor jednostkowy w osi Z dla układu współrzędnych powiązanego z i-tym członem łańcucha, podany względem układu bazowego,
- o_i jest to wektor z początku układu bazowego O_0 do początku układu O_i.

Dokładniejsza analiza procesu obliczania macierzy przekształcenia z rozdziału 4.3 pozawala wydedukować, że wektor z_i to pierwsze trzy elementy trzeciej kolumny macierzy H_0^i, natomiast wektor o_i to pierwsze trzy elementy czwartej kolumny macierzy H_0^i.

Macierz J_ω definiujemy jako:

$$J_\omega = \begin{bmatrix} J_\omega^1 & J_\omega^2 & \dots & J_\omega^N \end{bmatrix} \tag{4.27}$$

Jeśli przegub i-ty jest przegubem obrotowym to:

$$J_\omega^i = z_{i-1} \tag{4.28}$$

a jeśli przegub i-ty jest przegubem pryzmatycznym to:

$$J_\omega^i = 0 \tag{4.29}$$

Natomiast macierz J_v definiujemy jako:

$$J_\omega = \begin{bmatrix} J_v^1 & J_v^2 & \dots & J_v^N \end{bmatrix} \tag{4.30}$$

Jeśli przegub i-ty jest przegubem obrotowym to:

$$J_v^i = z_{i-1} \times (o_N - o_{i-1}) \tag{4.31}$$

a jeśli przegub i-ty jest przegubem pryzmatycznym to:

$$J_v^i = z_{i-1} \tag{4.32}$$

Wyprowadzenie powyższych wzorów można znaleźć w pracy [50].

Z powyższych wzorów i definicji wynika, że dla nogi robota kroczącego jakobian przyjmie postać:

$$J = \begin{bmatrix} z_0 \times (o_3 - o_0) & z_1 \times (o_3 - o_1) & z_2 \times (o_3 - o_2) \\ z_0 & z_1 & z_2 \end{bmatrix} \tag{4.33}$$

Z macierzy przekształcenia H_0^1, H_0^2 i H_0^3 możemy odczytać wszystkie wektory niezbędne do wyprowadzenia jakobianu z wyjątkiem wektorów z_0 i o_0. Z definicji wektorów zamieszczonej powyżej wynika jednak, że przyjmą one postać:

$$z_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, o_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.34}$$

Pozostałe brakujące elementy to:

$$\begin{aligned} z_1 &= \begin{bmatrix} s_1 \\ -c_1 \\ 0 \end{bmatrix}, z_2 = \begin{bmatrix} -s_1 \\ c_1 \\ 0 \end{bmatrix}, \\ o_1 &= \begin{bmatrix} 37c_1 \\ 37s_1 \\ 10 \end{bmatrix}, o_2 = \begin{bmatrix} 37c_1 + 57c_1c_2 \\ 37s_1 + 57s_1c_2 \\ 57s_2 + 10 \end{bmatrix}, \\ o_3 &= \begin{bmatrix} 37c_1 + 57c_1c_2 + 160c_1c_2c_3 + 160c_1s_2s_3 \\ 37s_1 + 57s_1c_2 + 160s_1c_2c_3 + 160s_1s_2s_3 \\ 57s_2 - 160c_2s_3 + 160s_2c_3 + 10 \end{bmatrix} \end{aligned} \tag{4.35}$$

Po niezbędnych obliczeniach jakobian przyjmuje postać:

$$J = \begin{bmatrix} -37s_1 - 57s_1c_2 - 160s_1c_2c_3 - 160s_1s_2s_3 \\ 37c_1 + 57c_1c_2 + 160c_1c_2c_3 + 160c_1s_2s_3 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix},$$

$$\begin{bmatrix} -c_1(57s_2 - 160c_2s_3 + 160s_2c_3) \\ -s_1(57s_2 - 160c_2s_3 + 160s_2c_3) \\ c_1(57c_1c_2 + 160c_1c_2c_3 + 160c_1s_2s_3) + s_1(57s_1c_2 + 160s_1c_2c_3 + 160s_1s_2s_3) \\ s_1 \\ -c_1 \\ 0 \end{bmatrix},$$

$$\begin{bmatrix} -c_1(160c_2s_3 - 160s_2c_3) \\ -s_1(160c_2s_3 - 160s_2c_3) \\ -c_1(160c_1c_2c_3 + 160c_1s_2s_3) - s_1(160s_1c_2c_3 + 160s_1s_2s_3) \\ -s_1 \\ c_1 \\ 0 \end{bmatrix} \tag{4.36}$$

Korzystając z zależności (4.25), w której wektor $\dot{q}$ przyjmuje postać:

$$\dot{q} = \begin{bmatrix} \dot{\theta_1} \\ \dot{\theta_2} \\ \dot{\theta_3} \end{bmatrix} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \tag{4.37}$$

otrzymujemy zależności na prędkości liniowe końcówki nogi robota w układzie bazowym:

$$\begin{aligned} v_x = &-(37s_1 + 57c_2s_1 + 160c_2c_3s_1 + 160s_1s_2s_3) \cdot \dot{\theta_1} \\ &-c_1(160c_2s_3 - 160c_3s_2) \cdot \dot{\theta_3} \\ &-c_1(57s_2 - 160c_2s_3 + 160c_3s_2) \cdot \dot{\theta_2} \end{aligned}$$

$$\begin{aligned} v_y = &(37c_1 + 57c_1c_2 + 160c_1c_2c_3 + 160c_1s_2s_3) \cdot \dot{\theta_1} \\ &-s_1(160c_2s_3 - 160c_3s_2) \cdot \dot{\theta_3} \\ &-s_1(57s_2 - 160c_2s_3 + 160c_3s_2) \cdot \dot{\theta_2} \end{aligned}$$

$$\begin{aligned} v_z = &(c_1(57c_1c_2 + 160c_1c_2c_3 + 160c_1s_2s_3) + s_1(57c_2s_1 + 160c_2c_3s_1 + 160s_1s_2s_3)) \cdot \dot{\theta_2} \\ &-(c_1(160c_1c_2c_3 + 160c_1s_2s_3) + s_1(160c_2c_3s_1 + 160s_1s_2s_3)) \cdot \dot{\theta_3} \end{aligned} \tag{4.38}$$

Rozdział 5

Wykorzystanie oscylatorów liniowych

Oscylatorem nazywamy układ, który produkuje pewien powtarzalny, okresowy sygnał. Oscylatorem może być układ mechaniczny, hydrauliczny czy elektryczny opisany równaniem różniczkowym, którego rozwiązaniem jest funkcja okresowa [1]. W rozdziale omówiono dynamikę najprostszych oscylatorów liniowych oraz ich zastosowanie do budowy układu sterującego chodem robota sześcionożnego.

5.1 Oscylator harmoniczny

Jednym z najprostszych układów oscylatora jest oscylator harmoniczny będący układem liniowym opisanym równaniem:

$$\ddot{y} + \omega_0^2 y = 0 \tag{5.1}$$

Stosując podstawienie

$$\begin{aligned} x_1 &= y \\ x_2 &= \dot{y} \end{aligned} \tag{5.2}$$

możemy uzyskać równanie stanu, które w zapisie macierzowym przyjmuje postać:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\omega_0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{5.3}$$

Rozwiązaniem równania 5.1 jest funkcja:

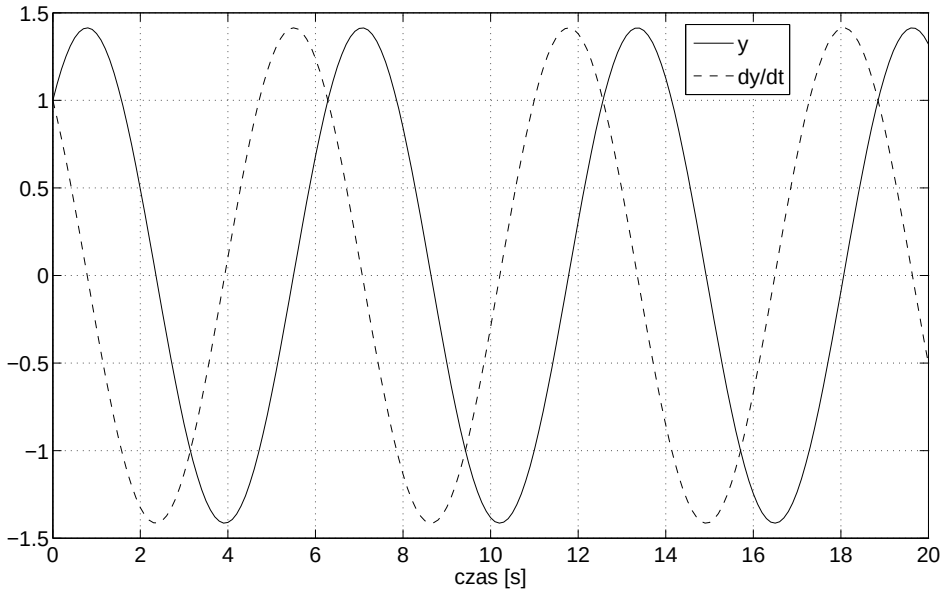
$$y(t) = A\sin(\omega_0 t) + B\cos(\omega_0 t) \tag{5.4}$$

gdzie współczynnik ω_0 oznacza pulsację drgań oscylatora podaną w $\frac{rad}{s}$, a wartości współczynników A i B są zależne od warunków początkowych y_0 i $\dot{y}_0$. Przykładowe rozwiązanie zostało przedstawione na rysunku 5.1.

Stabilność układu możemy łatwo zweryfikować obliczając wartości własne macierzy stanu ze wzoru (5.3). Przy założeniu, że współczynnik ω_0 przyjmuje wartość dodatnią, wynoszą one odpowiednio:

$$\begin{aligned} \lambda_1 &= j\sqrt{\omega_0} \\ \lambda_2 &= -j\sqrt{\omega_0} \end{aligned} \tag{5.5}$$

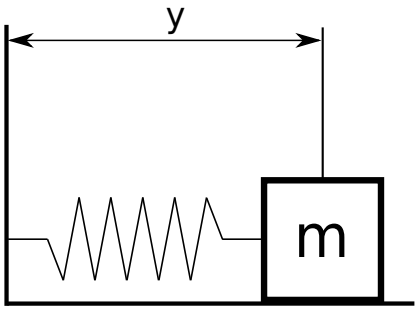
co oznacza, że układ jest stabilny (ale nie asymptotycznie stabilny) [32] i [36]. Obecność niezerowej części urojonej w wartościach własnych świadczy o występowaniu oscylacji.



Rysunek 5.1: Rozwiązanie układu oscylatora harmonicznego dla wartości początkowych $y = 1$ i $\dot{y} = 1$

Działanie układu można podsumować prostym stwierdzeniem, że rozwiązaniem układu w przestrzeni fazowej są trajektorie o kształtach okręgu wychodzące z punktów wartości początkowych położenia i prędkości. Rysunek 5.3 przedstawia przykładowe rozwiązania oscylatora harmonicznego opisanego wzorem (5.1).

Równanie to opisuje na przykład dynamikę prostego układu mechanicznego składającego się z bloczka o zadanej masie m przymocowanego na sprężynie o współczynniku sprężystości k przy założeniu, że pominięto siłę tarcia. Układ ten został przedstawiony na rysunku 5.2.

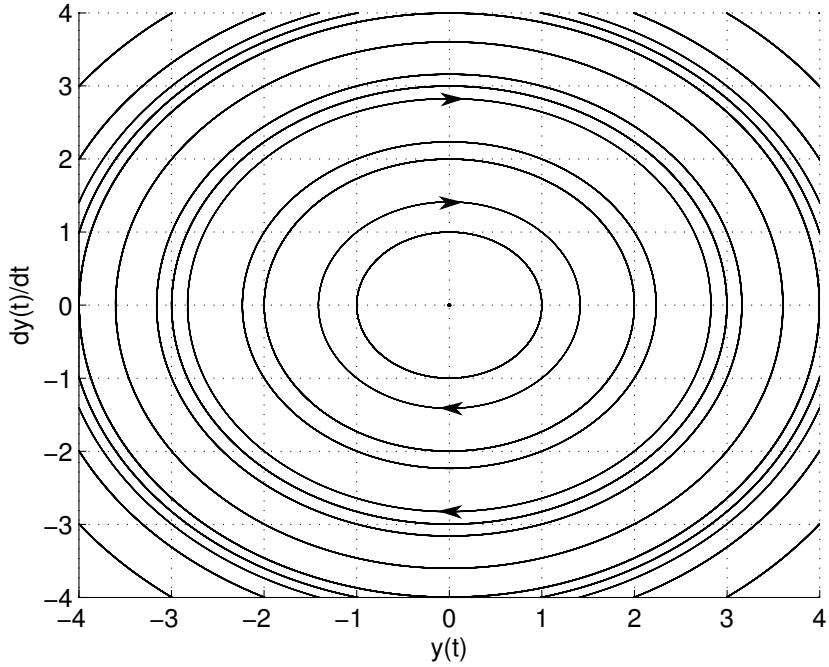


Rysunek 5.2: Interpretacja mechaniczna układu oscylatora harmonicznego

5.2 Oscylator harmoniczny tłumiony

Gdy w układzie mechanicznym przedstawionym na rysunku 5.2 uwzględnimy siłę tarcia, otrzymamy równanie:

$$\ddot{y} + 2\xi\omega_0\dot{y} + \omega_0^2 y = 0 \tag{5.6}$$



Rysunek 5.3: Portret fazowy oscylatora harmonicznego

lub stosując ponownie podstawienie (5.2) – równanie stanu w postaci macierzowej

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\omega_0 & -2\xi\omega_0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{5.7}$$

gdzie wartość ξ jest współczynnikiem tłumienia. Rozwiązaniem tego równania są oscylacje tłumione (gasnące) przy czym szybkość tłumienia oscylacji zależy od współczynnika tłumienia. Wykresy ilustrujące przykładowe rozwiązania dla różnych wartości współczynnika ξ przedstawiono na rysunku 5.4.

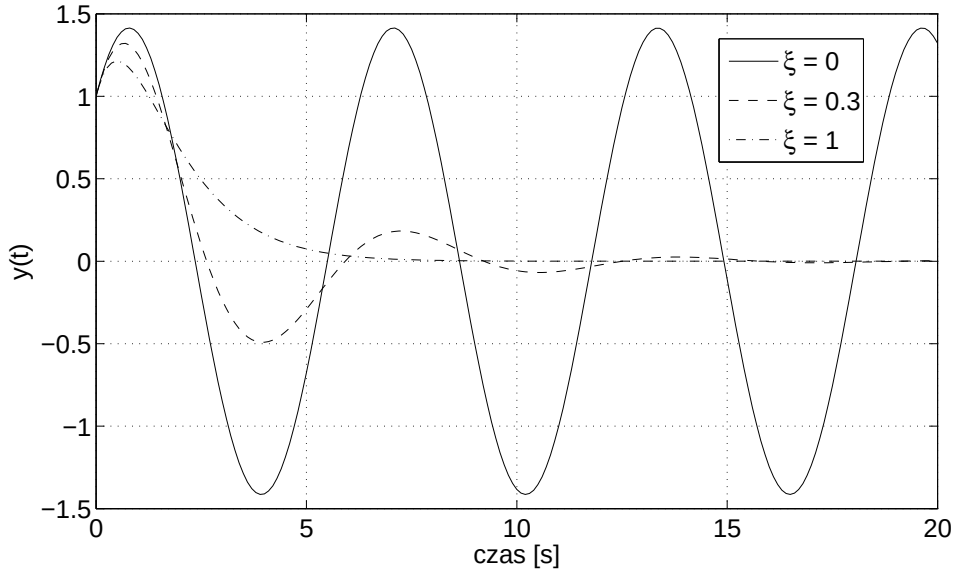
Analizując zachowanie rozwiązania równania (5.6) należy wyróżnić trzy podstawowe przypadki:

- $\xi < 1$ - gdy rozwiązanie oscyluje z częstotliwością zbliżoną do przypadku nietłumionego, ale z amplitudą stopniowo zmniejszającą się do zera, układ wraca do punktu równowagi,
- $\xi = 1$ - gdy rozwiązanie wraca do punktu równowagi najszybciej jak to tylko możliwe bez oscylacji, jest to tak zwane rozwiązanie aperiodyczne krytyczne,
- $\xi > 1$ - gdy rozwiązanie wraca do punktu równowagi bez oscylacji, ale wolniej niż w przypadku rozwiązania aperiodycznego krytycznego, czym większe wartości współczynnika tłumienia tym punkt równowagi jest osiągany wolniej.

Do analizy stabilności układu oscylatora harmonicznego z tłumieniem należy wyliczyć wartości własne macierzy stanu z równania (5.7), które można wyrazić wzorami:

$$\begin{aligned} \lambda_1 &= -2\xi\omega_0 - j2\sqrt{\xi^2\omega_0^2 - \omega_0} \\ \lambda_2 &= -2\xi\omega_0 + j2\sqrt{\xi^2\omega_0^2 - \omega_0} \end{aligned} \tag{5.8}$$

Przy założeniu, że wartości współczynników ω_0 i ξ są dodatnie otrzymujemy układ:



Rysunek 5.4: Rozwiązanie układu oscylatora harmonicznego z tłumieniem dla wartości początkowych $y = 1$, $\dot{y} = 1$ i dla różnych wartości współczynnika ξ

- asymptotycznie stabilny – ze względu na ujemne części rzeczywiste wartości własnych,
- oscylujący – ze względu na niezerowe części urojone wartości własnych.

Dla tego typu wartości własnych otrzymujemy portret fazowy typu ognisko. Dla przykładowych wartości współczynników $\omega_0 = 1$ oraz $\xi = 0.5$ został on przedstawiony na rysunku 5.5.

5.3 Oscylator harmoniczny z wymuszeniem

Oscylatorem harmonicznym z wymuszeniem nazywamy układ opisany równaniem różniczkowym

$$\ddot{y} + 2\xi\omega_0\dot{y} + \omega_0^2 y = F(t) \tag{5.9}$$

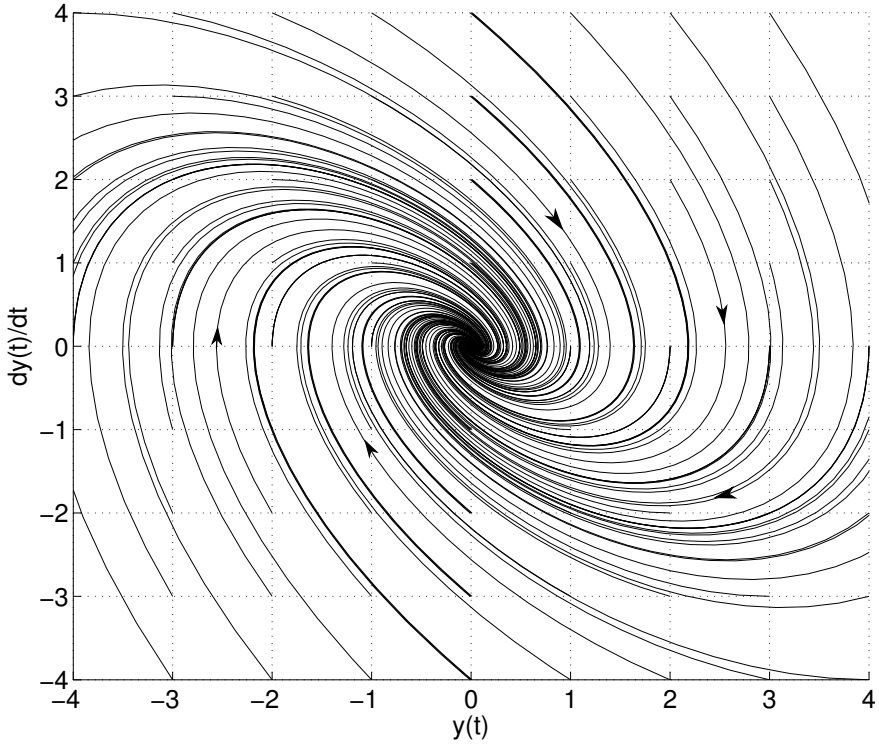
gdzie $F(t)$ jest funkcją wymuszającą zależną od czasu. Stosując podstawienie (5.2) otrzymamy postać macierzową:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\omega_0 & -2\xi\omega_0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ F(t) \end{bmatrix} \tag{5.10}$$

Wprowadzenie dodatkowej funkcji wymuszającej nie ma wpływu na wartości własne więc nie zmienia kwestii stabilności bądź asymptotycznej stabilności układu [33]. Podstawiając za $\dot{x}_1 = 0$ i $\dot{x}_2 = 0$ do równania (5.10) otrzymamy nowy punkt równowagi układu

$$\begin{aligned} x_1 &= \frac{F(t)}{\omega_0} \\ x_2 &= 0 \end{aligned} \tag{5.11}$$

Stosując różne klasy funkcji $F(t)$ możemy uzyskać efekty ułatwiające zastosowanie równań oscylatora harmonicznego podczas projektowania układów sterowania chodem robota kroczącego.



Rysunek 5.5: Portret fazowy oscylatora harmonicznego tłumionego

Najprostszą klasą funkcji wymuszającej jest

$$F(t) \equiv C \tag{5.12}$$

gdzie C oznacza wartość stałą (patrz rysunek 5.6a). Funkcja wymuszająca o takim charakterze pozwala na przesunięcie punktu równowagi na płaszczyźnie stanu. Oznacza to, że odpowiednio ustalając stałą C przesuniemy punkt, do którego dążą wszystkie rozwiązania równania (5.6). W szczególnym przypadku gdy współczynnik tłumienia $\xi = 0$ równanie (5.9) upraszcza się do postaci:

$$\ddot{y}(t) + \omega_0 y = F(t) \tag{5.13}$$

co oznacza, że wartość C przesunie punkt wokół, którego oscylują wszystkie rozwiązania (patrz rysunek 5.6b).

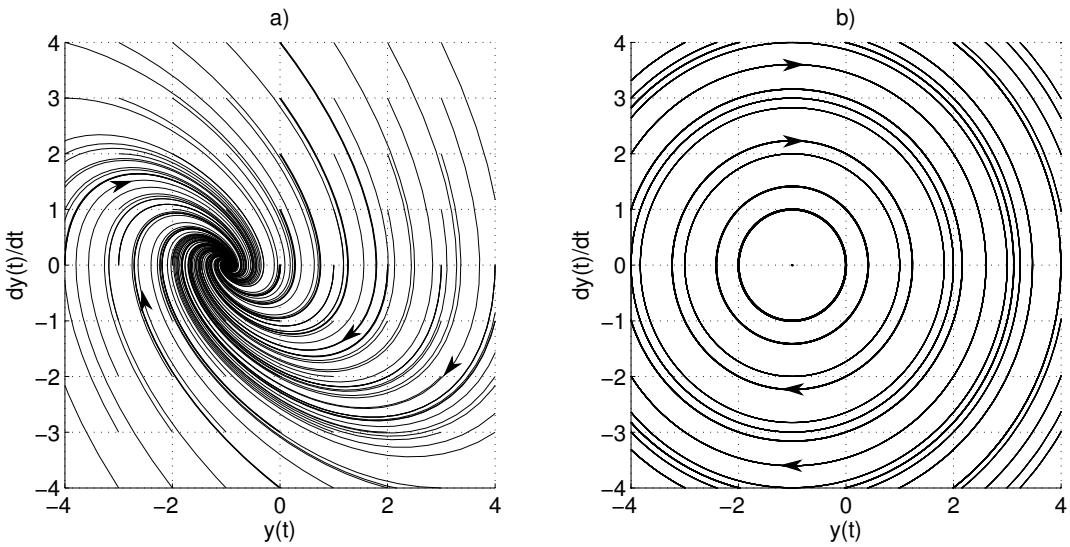
Innym typem funkcji wymuszającej może być funkcja

$$F(t) = F_0 \sin(\omega t) \tag{5.14}$$

Podstawiając funkcję wymuszenia do równania (5.11) otrzymamy równanie na punkt równowagi takiego układu w postaci:

$$\begin{aligned} x_1 &= \frac{F_0 \sin(\omega t)}{\omega_0} \\ x_2 &= 0 \end{aligned} \tag{5.15}$$

Jego położenie będzie zależne od czasu i będzie przemieszczać się po płaszczyźnie fazowej zgodnie z funkcją $F_0 \sin(\omega t)$. Z punktu widzenia praktycznego wykorzystania tego układu



Rysunek 5.6: Portret fazowy oscylatora harmonicznego a) z tłumieniem i z wymuszeniem stałym b) bez tłumienia i z wymuszeniem stałym

bardziej interesująca wydaje się jego niestacjonarna wersja, w której wymuszenie sinusoidalne zarówno jak i tłumienie jest włączane warunkowo na żądanie użytkownika. Układ taki możemy opisać równaniem:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \begin{cases} -\omega_0 x_1 & , t < t_1 \\ -\omega_0 x_1 - 2\xi\omega_0 x_2 + F_0 \sin(\omega t) & , t \geq t_1 \end{cases} \end{aligned} \tag{5.16}$$

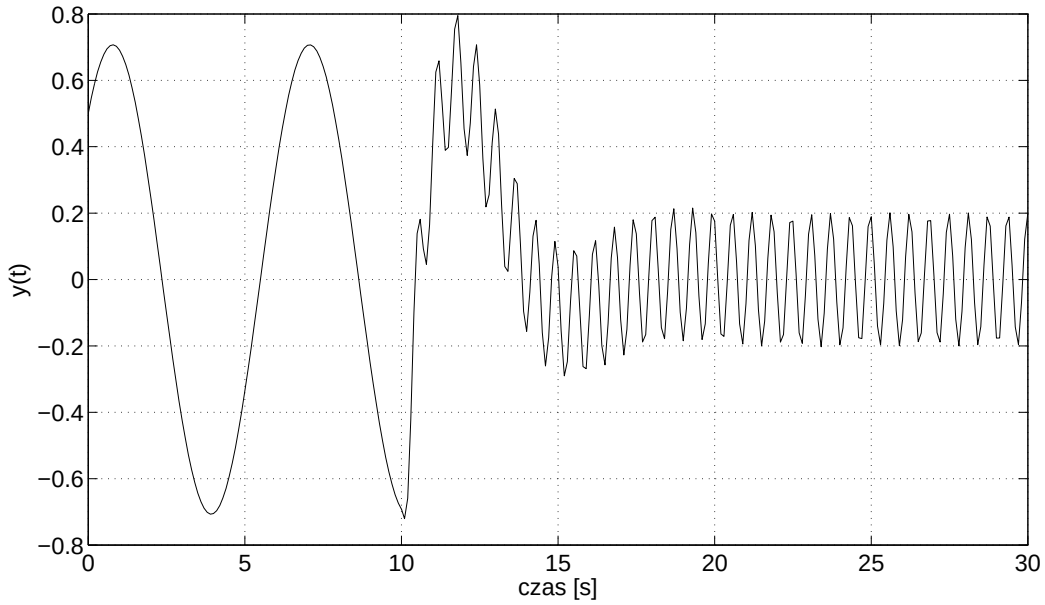
Takie wykorzystanie oscylatora harmonicznego z tłumieniem i wymuszeniem pozwala na uzyskanie trajektorii przejścia ze stanu o oscylacjach nietłumionych o zadanej amplitudzie i częstotliwości do stanu, w którym częstotliwość jest zupełnie inna.
Przykład takiej zmiany przedstawiono na rysunku 5.7. W przedstawionym przykładzie parametry równania wynoszą: $\omega_0 = 1$, $\omega = 10$, $\xi = 0,5$, $F_0 = 20$, warunek początkowy równania oscylatora harmonicznego to $[x_1, x_2]^T = [0, 0]^T$, a przełączenie układu nastąpiło w chwili czasu $t_1 = 10$ s.

5.4 Wykorzystanie oscylatorów liniowych do budowy systemu sterowania

Oscylatory liniowe można z powodzeniem zastosować do budowy układu sterującego ruchem nóg robota sześcionożnego. Ze względu na charakter ruchu poszczególnych nóg, przebiegi czasowe pozwalające na wprawienie robota w ruch postępowy oraz dynamikę układu oscylatora liniowego (harmonicznego) można jedynie uzyskać chód trójpodporowy. Wynika to z faktu, że tylko w tym rodzaju chodu przebiegi sterujące (patrz rysunek 2.14) przyjmują kształt sinusoidy.

Chód trójpodporowy został szczegółowo opisany w rozdziale 2.2.4. Jego charakterystyczną cechą jest naprzemienne przestawianie grup trzech nóg tak by robot (owad) zachował stabilność w każdej chwili chodu. Naprzemiennie poruszają się grupy nóg:

- grupa A — lewa przednia, prawa środkowa, lewa tylna
- grupa B — prawa przednia, lewa środkowa, prawa tylna



Rysunek 5.7: Odpowiedź układu oscylatora z tłumienie i wymuszeniem sinusoidalnym włączanymi warunkowo w chwili czasu $t_1 = 10$ s

Oznacza to, że układ CPG możemy ograniczyć do dwóch jednostek bazujących na oscylatorach, z których każda odpowiada za generację przebiegów dla jednej grupy.

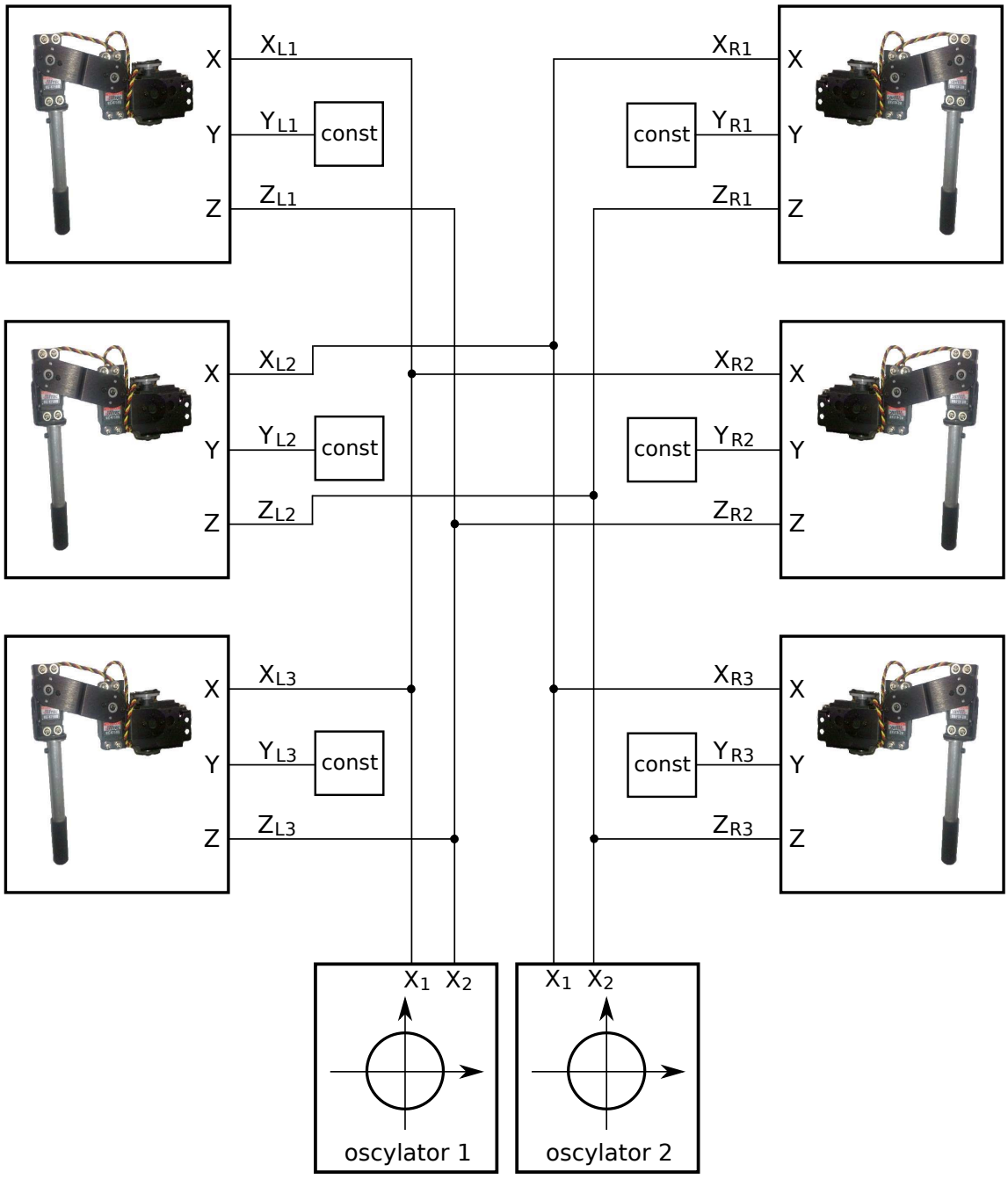
Budowę systemu sterującego rozpoczyna spostrzeżenie, że trajektoria ruchu końcówki nogi robota (patrz rysunek 2.1) przypomina kształtem trajektorię z portretu fazowego oscylatora harmonicznego. Wykorzystanie równania tego oscylatora polega na odpowiednim podłączeniu sygnałów sterujących, a mianowicie:

- y z równania (5.1) (sygnał x_1 z wersji (5.3)) jako sygnał sterujący ruchem końcówki nogi wzdłuż kierunku przemieszczania się robota (oś X),
- $\dot{y}$ z równania (5.1) (sygnał x_2 z wersji (5.3)) jako sygnał sterujący ruchem końcówki nogi wzdłuż kierunku prostopadłego do podłoża (oś Z).

Schemat prostego systemu sterującego chodem robota kroczącego wykorzystujący oscylatory harmoniczne jest przedstawiony na rysunku 5.8. Na schemacie zaznaczono współrzędną Y każdej z nóg robota utrzymywaną na stałej wartości zadanej. Należy przypomnieć, że wszystkie wartości odnoszą się do poszczególnych układów współrzędnych każdej z nóg, a początek układu współrzędnych przestrzeni fazowej oscylatora jest tożsamy z pozycją bazową nogi.

Chód robota z punktu widzenia takiego układu sterowania składa się z trzech podstawowych faz:

1. **faza rozruchu** – start robota, przejście końcówek nóg z pozycji bazowej do punktu na okręgu w płaszczyźnie (X,Z),
2. **chód właściwy** – faza płynnego przemieszczania, ruch końcówek nóg po okręgu w płaszczyźnie (X,Z),
3. **faza zatrzymania** – zatrzymanie robota, przejście końcówek nóg z punktu na okręgu w płaszczyźnie (X,Z) do pozycji bazowej.



Rysunek 5.8: Schemat ideowy systemu sterującego chodem robota sześcionożnego poruszającego się chodem trójpodporowym z wykorzystaniem oscylatorów harmonicznych

Przy pomocy przedstawionego wyżej układu można zrealizować wszystkie trzy fazy.

Faza rozruchu może zostać zrealizowana przez zadanie dla końcówek nóg odpowiednich wartości początkowych. Oscylatory harmoniczne wprowadzamy w oscylacje o zadanej amplitudzie za pomocą warunku początkowego. Dobrym rozwiązaniem wydaje się uniesienie nóg z grupy A do góry, do maksymalnego położenia w osi Z i opuszczenie nóg z grupy B w dół, do minimalnego położenia. Analizując wykres odpowiedzi równania oscylatora harmonicznego na rysunku 5.1 możemy wywnioskować, że warunek początkowy dla znormalizowanych wartości wysokości i długości kroku będzie odpowiednio wynosił:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

dla grupy A oraz

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

dla grupy B. Można także zamienić wartościami grupę A z grupą B.

Chód właściwy może zostać zrealizowany przy pomocy zwykłych oscylatorów harmonicznych, które przejmują kontrolę nad współrzędnymi końcówki nóg robota zgodnie ze schematem 5.8. Jako warunki początkowe przyjmuje się położenie końcówek nóg robota odpowiednio dla grupy A i grupy B bezpośrednio po zakończeniu fazy rozruchu.

Faza zatrzymania polega na powrocie do położenia bazowego końcówek nóg. W tej fazie można wykorzystać właściwości oscylatora harmonicznego z tłumieniem. Odpowiednio dobrany współczynnik tłumienia pozwala na realizację takiej trajektorii. Część równania oscylatora odpowiedzialna za tłumienie powinna być włączana warunkowo bezpośrednio po przejściu robota do fazy zatrzymania. Kluczowe w tym przypadku wydaje się dobranie odpowiedniej wartości współczynnika tłumienia ξ. Jego wartość powinna wynosić około 1 co pozwala na powrót do pozycji bazowej bez dodatkowych oscylacji i w krótkim czasie.

Pomimo, że układ sterowania sześcionożnym robotem kroczącym oparty na oscylatorach harmonicznych wydaje się stosunkowo prosty, to jego fizyczna realizacja już taka nie jest. Podczas realizacji fazy chodu właściwego, układ sterowania oblicza trajektorię chodu korzystając z równania oscylatora harmonicznego, będącego układem stabilnym, ale nie asymptotycznie stabilnym. Realizacja układu sterowania w układzie mikroprocesorowym (takim jak opisano w rozdziale 3.3 i 3.4) wymaga dyskretyzacji równań stanu. Niestety dyskretna wersja równania jest niestabilna [35]. Istnieje możliwość rozbudowy układu o elementy umożliwiające implementację, ale układ traci przez to swoją największą zaletę czyli niewielką złożoność obliczeniową. Z tego właśnie powodu zrezygnowano z implementacji systemu sterowania sześcionożnym robotem kroczącym opartego na oscylatorach harmonicznych. Wykonano jednak symulacje by przynajmniej częściowo zweryfikować założenia odnośnie działania tego układu.

5.5 Wyniki symulacji układu sterowania opartego na oscylatorach liniowych

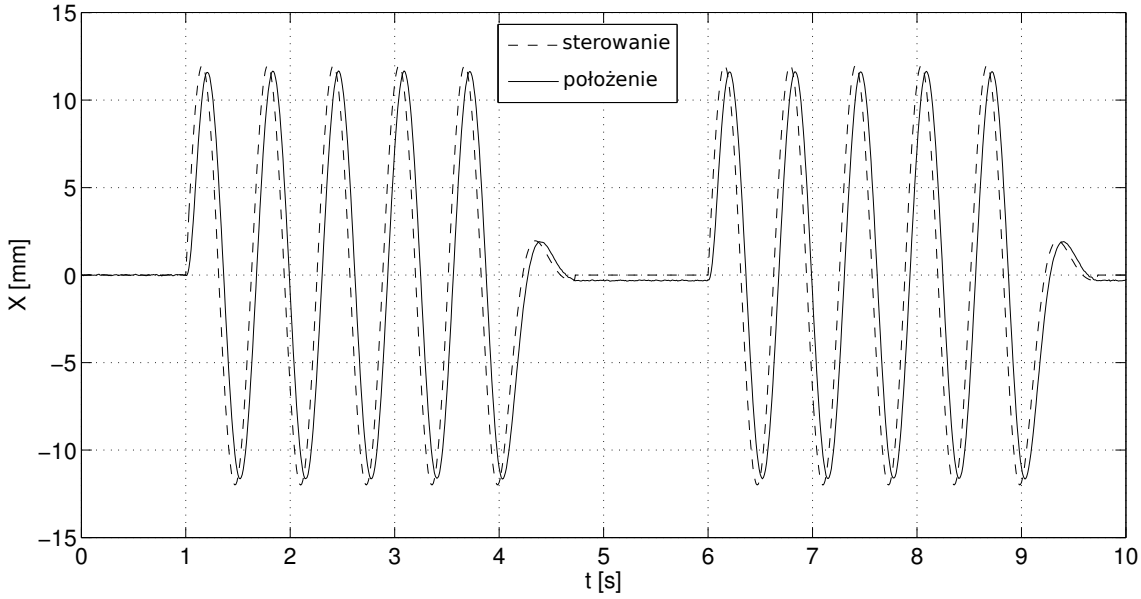
Wykonano symulacje oscylatorów liniowych sterujących modelem robota. Wyniki tych symulacji pozwalają na przynajmniej częściową ocenę algorytmu sterującego oraz pozwalają ocenić sam model robota.

Przyprowadzono wiele eksperymentów symulacyjnych. Na rysunkach 5.9, 5.10, 5.11 i 5.12 przedstawiono wyniki najbardziej interesującego z nich. Eksperyment ten trwał 10 s i miał następujący przebieg:

- 0 s – robot znajduje się w pozycji bazowej, oczekuje na sterowanie,

- 1 s – sterowanie przechodzi w fazę rozruchu,
- 4 s – sterowanie przechodzi w fazę zatrzymania,
- 6 s – sterowanie przechodzi w fazę rozruchu,
- 9 s – sterowanie przechodzi w fazę zatrzymania.

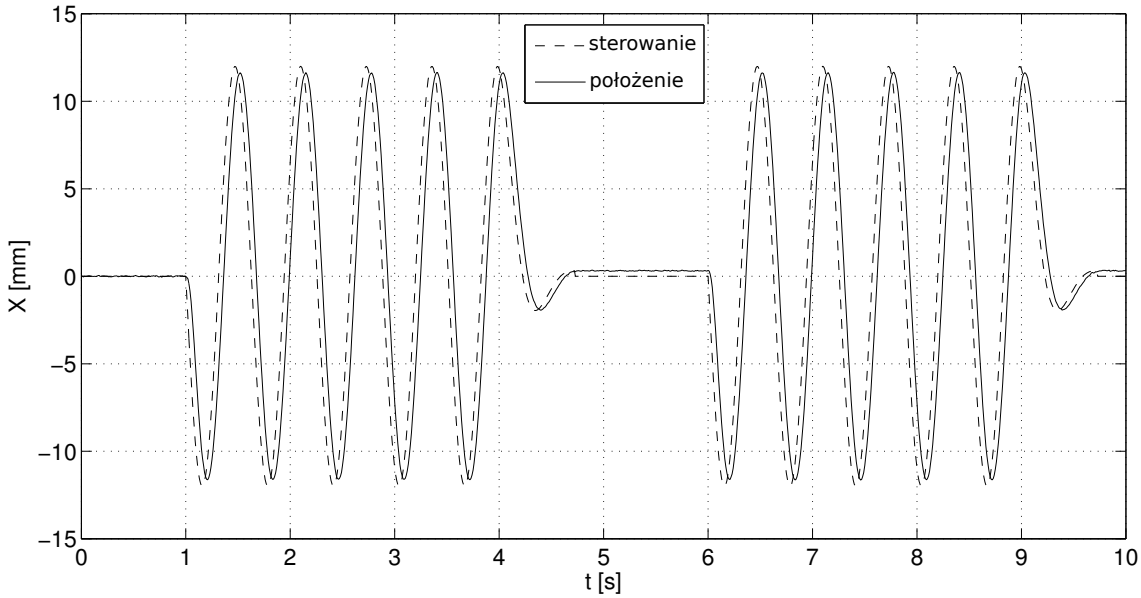
Rysunki 5.9 i 5.11 przedstawiają przebieg sterowania wysłanego do modelu i przebieg położeń jednej z końcówek nóg z grupy A w osi X i w osi Z w funkcji czasu. Składają się one na pełny ruch końcówki nogi w płaszczyźnie XZ podczas opisanego eksperymentu. Rysunki 5.10 i 5.12 przedstawiają sterowania wysłane do modelu robota i położenia jednej z końcówek nóg z grupy B w osi X i w osi Z w funkcji czasu.



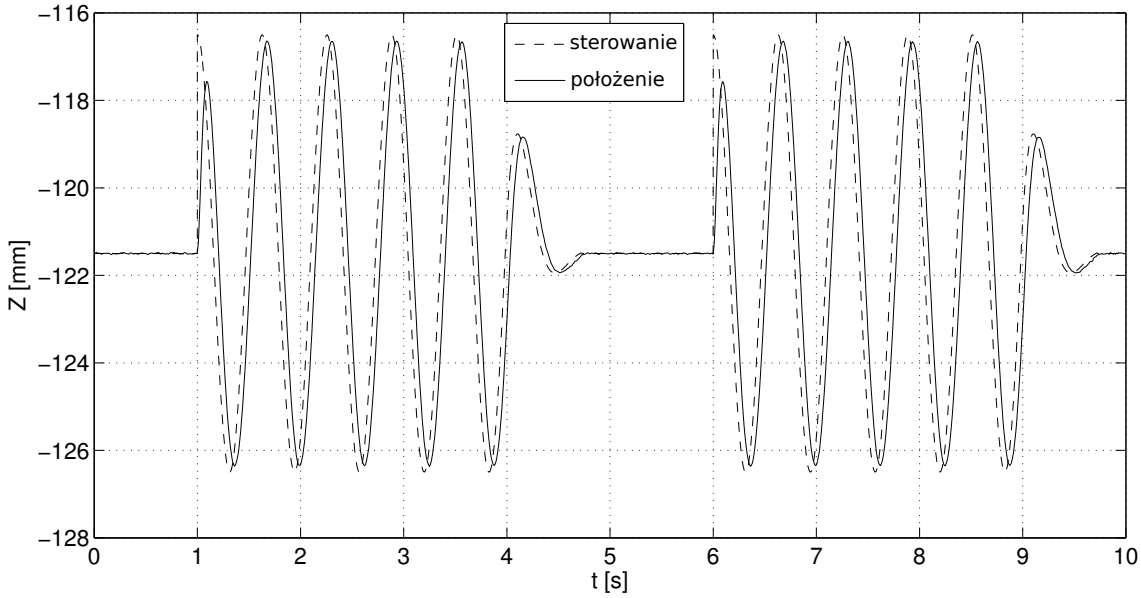
Rysunek 5.9: Sterowanie oraz położenie końcówki jednej z nóg robota z grupy A w osi X w funkcji czasu – wyniki symulacji modelu robota sterowanego algorytmem opartym na oscylatorach harmonicznych

Analiza wyników symulacji wykazała, że sam algorytm sterowania spełnia swoją rolę. Ruch końcówek nóg robota jest zgodny z oczekiwaniem. Zrealizowano wszystkie trzy fazy ruchu. Sterowanie nogami z grupy A jest w przeciwfazie w stosunku do sterowania nogami z grupy B, generuje więc ruch postępowy robota. Symulacja modelu robota sześcionożnego pokazuje w jaki sposób sterowanie końcówką nogi przekłada się na jej ruch. Wykazano między innymi opóźnienie pomiędzy sygnałem sterowaniem, a sygnałem położenia o wielkości $70 - 80$ ms. Powoduje ono między innymi wygładzenie jedynego niegładkiego punktu na charakterystykach sterujących jakim jest zadanie warunku początkowego położenia w fazie rozruchu (patrz 1 s symulacji na rysunkach 5.11 i 5.12).

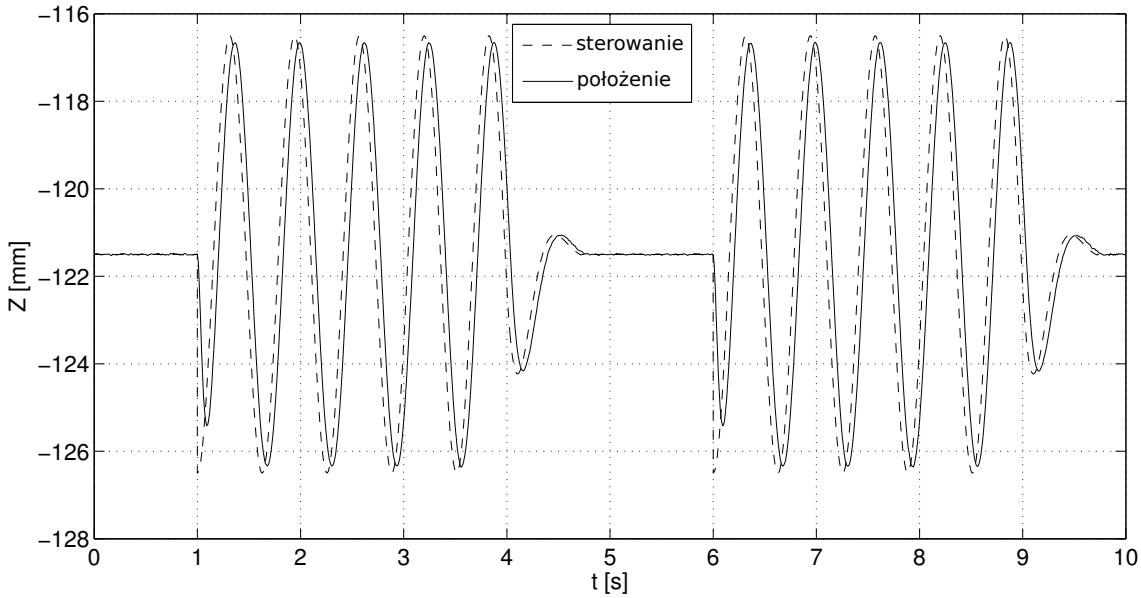
Jako interesujący przykład zamieszczono wykres przedstawiający sterowanie i ruch końcówki nogi w osi Y dla jednej z nóg robota. Zgodnie z wytycznymi algorytmu z rozdziału 5.4 położenie końcówki nogi w osi Y powinno być stałe (w przypadku tej symulacji przyjęto $86,5$ mm). Oczywiście ruch w pozostałych osiach powoduje zakłócenia tego położenia. Wartość tych zakłóceń wynika między innymi z dynamiki całej nogi. Wahania nie przekraczają $0,1$ mm.



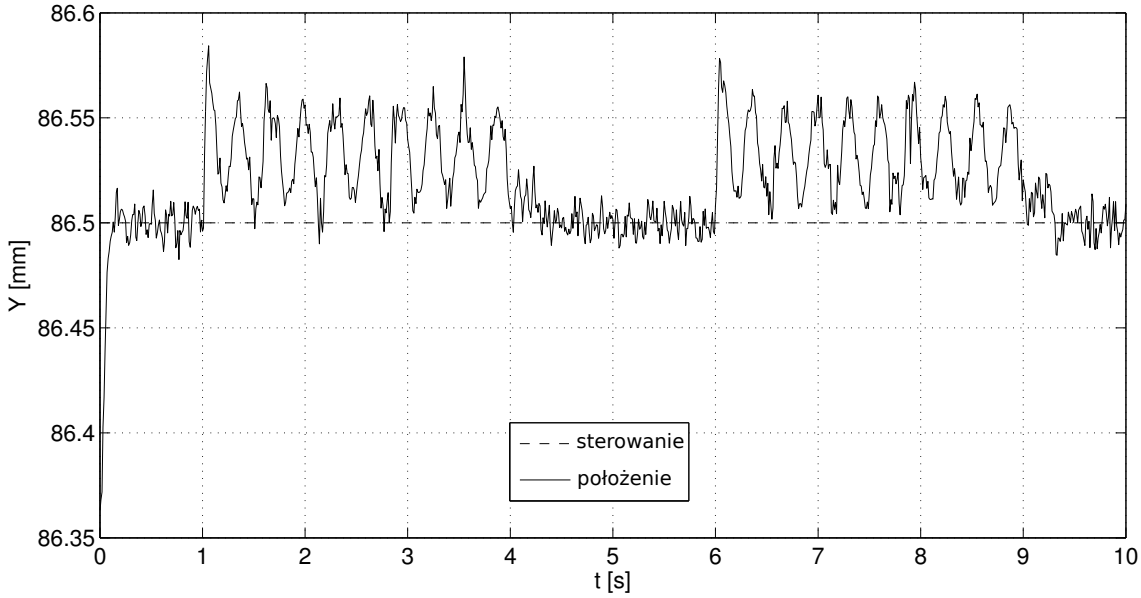
Rysunek 5.10: Sterowanie oraz położenie końcówki jednej z nóg robota z grupy B w osi X w funkcji czasu – wyniki symulacji modelu robota sterowanego algorytmem opartym na oscylatorach harmonicznych



Rysunek 5.11: Sterowanie oraz położenie końcówki jednej z nóg robota z grupy A w osi Z w funkcji czasu – wyniki symulacji modelu robota sterowanego algorytmem opartym na oscylatorach harmonicznych



Rysunek 5.12: Sterowanie oraz położenie końcówki jednej z nóg robota z grupy B w osi Z w funkcji czasu – wyniki symulacji modelu robota sterowanego algorytmem opartym na oscylatorach harmonicznych



Rysunek 5.13: Sterowanie oraz położenie końcówki jednej z nóg robota osi Y w funkcji czasu – wyniki symulacji modelu robota sterowanego algorytmem opartym na oscylatorach harmonicznych

Rozdział 6

Wykorzystanie oscylatorów nieliniowych

Odrębną kategorię oscylatorów stanowią układy oscylatorów nieliniowych z cyklem granicznym w przestrzeni fazowej. Najciekawszą kategorią są tak zwane oscylatory relaksacyjne. Oscylatorem relaksacyjnym nazywamy układ dynamiczny, który okresowo przełącza się pomiędzy dwoma fazami:

- faza dysypacji – rozpraszania energii, w którym energia z systemu jest stopniowo przekazywana do magazynu energii,
- faza relaksacji – wyładowania energii z magazynu.

Do przełączania pomiędzy tymi dwoma fazami dochodzi na skutek osiągnięcia ustalonego progu energii. Warto zauważyć, że element opisujący równanie takiego oscylatora, który będzie odpowiadał za progowe przełączanie może mieć charakter nieliniowy. Magazynem energii może być na przykład masa lub kondensator.

Rozwiązaniem układu generatora relaksacyjnego mogą być funkcje o bardzo różnym charakterze w tym także funkcje nieliniowe (przebieg prostokątny, trójkątny, piłokształtny czy będący złożeniem kilku funkcji sinusoidalnych o różnych parametrach). Jest to cecha pożądana ponieważ charakter funkcji sterujących chodem robota jest także w wielu przypadkach nieliniowy.

Cyklem granicznym nazywamy rozwiązanie okresowe w przestrzeni fazowej, w którego otoczeniu nie znajdują się inne rozwiązania okresowe. Oznacza to, że wszystkie rozwiązania układu w danym sąsiedztwie zmierzają do tego jednego rozwiązania okresowego. Gdy rozwiązania zmierzają do cyklu przy $t \to \infty$ mówimy o cyklu ω-granicznym. Gdy rozwiązania zmierzają do cyklu przy $t \to -\infty$ mówimy o cyklu α-granicznym. Cykl graniczny podobnie jak punkty równowagi, atraktory czy orbity cykliczne, jest specyficznym rodzajem zbioru granicznego na płaszczyźnie fazowej.

Jednym z podstawowych twierdzeń stosowanych podczas badania cyklu jest twierdzenie Poincare-Bendixson'a definiujące kiedy rozwiązaniem danego układu dynamicznego jest cykl (rozwiązanie periodyczne) dla układów dwuwymiarowych.

Twierdzenie Poincare-Bendixson'a. *Dany jest system dynamiczny opisany równaniem:*

$$\begin{array}{rcl} \dot{x}(t) & = & f\left[x(t)\right] \\ x(0) & = & x_0 \end{array} \tag{6.1}$$

gdzie $f : \mathbf{R}^2 \to \mathbf{R}^2$, $f \in C^1$, $x(t) \in \mathbf{R}^2$, *dla ustalonego* $t \geq 0$. *Jeśli ograniczona trajektoria* $\Lambda(x_0)$ *nie zawiera punktów równowagi to* $\Lambda(x_0)$ *jest trajektorią rozwiązania periodycznego.*

Niestety twierdzenie to w zależności od publikacji i autora przyjmuje bardzo różne postacie. Wersja podana powyżej pochodzi z pracy [15]. Dowód twierdzenia można znaleźć w [17].

6.1 Oscylator Van der Pola

Jednym z najbardziej charakterystycznych oscylatorów relaksacyjnych jest oscylator Van der Pola definiowany jako układ dynamiczny wyrażony równaniem:

$$\ddot{y} - \mu\left(1 - y^2\right)\dot{y} + \omega^2 y = 0 \tag{6.2}$$

Równanie to zostało podane przez holenderskiego fizyka i inżyniera Baltazara Van der Pola. Równanie to znalazło wiele zastosowań praktycznych między innymi w biologii i sejsmologii.

Równanie dynamiki oscylatora Van der Pola jest jednym z wielu równań spełniających postać układu Lienarda [14] i [15]. Układem Lienarda nazywamy wszystkie systemy dynamiczne, które możemy zapisać w postaci:

$$\ddot{y} + f(y)\dot{y} + g(y) = 0 \tag{6.3}$$

lub równoważnej postaci:

$$\begin{aligned} \dot{x}_1 &= x_2 - F(x_1) \\ \dot{x}_2 &= -g(x_1) \end{aligned} \tag{6.4}$$

przy czym

$$F(x) = \int_0^x f(s)ds \tag{6.5}$$

Równanie opisujące oscylator Van der Pola można zapisać także w tej drugiej formie i wtedy przyjmuje ono postać:

$$\begin{aligned} \dot{x}_1 &= x_2 + \mu\left(1 - \frac{1}{3}x_1^2\right)x_1 \\ \dot{x}_2 &= -\omega^2 x_1 \end{aligned} \tag{6.6}$$

Twierdzenie Lienarda mówi, iż każdy układ dynamiczny wyrażony równaniem w postaci (6.3) i spełniający następujące warunki:

- $F, g \in C^1(R)$
- F, g są funkcjami nieparzystymi,
- $xg(x) > 0$ dla $x \neq 0$,
- $F(0) = 0, f(0) < 0$
- $F(x)$ ma jedno zero dodatnie w a i rośnie monotonicznie do nieskończoności dla $x \geq a$ gdy $x \to \infty$

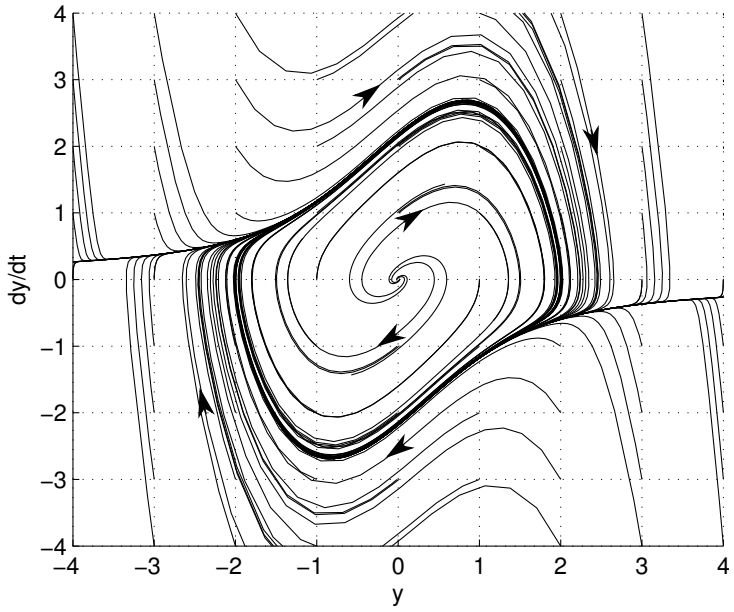
ma dokładnie jeden cykl graniczny w przestrzeni stanu. Dowód twierdzenia Lienarda jest podany w pracy [37].

Równanie oscylatora Van der Pola z funkcjami

$$\begin{aligned} F(x) &= \mu x\left(1 - \frac{1}{3}x^2\right) \\ g(x) &= \omega^2 x \end{aligned}$$

spełnia wszystkie warunki z twierdzenia Lienarda więc możemy uznać, że rozwiązanie tego równania będzie dążyło w przestrzeni stanu do cyklu granicznego.

Przykładowe rozwiązania tego równania zostały przedstawione na rysunku 6.1. Wyraźnie widać na nim, że niezależnie od wyboru punktu początkowego wszystkie trajektorie zbiegają do cyklu granicznego o dosyć charakterystycznym kształcie.



Rysunek 6.1: Portret fazowy oscylatora Van der Pola

Wartość współczynnika μ ma wpływ na kształt cyklu granicznego w przestrzeni stanu. Na rysunku 6.2 można zaobserwować w jaki sposób zmienia się kształt cyklu granicznego przy wzrastającej wartości współczynnika μ. Współczynnik ten może przyjmować tylko wartości dodatnie.

Wartość współczynnika ω oznacza częstość oscylacji pojawiających się w układzie.

6.2 Oscylator Rayleigha

Oscylator Rayleigha jest zmodyfikowaną wersją oscylatora relaksacyjnego Van der Pola. Równanie to zostało po raz pierwszy podane przez lorda Rayleigha w pracy [45]. Równanie oscylatora przyjmuje postać

$$\ddot{y} - \mu\left(1 - \dot{y}^2\right)\dot{y} + \omega^2 y = 0 \tag{6.7}$$

Na rysunku 6.3 przedstawiono portret fazowy oscylatora Rayleigha.

Stosując podstawienie:

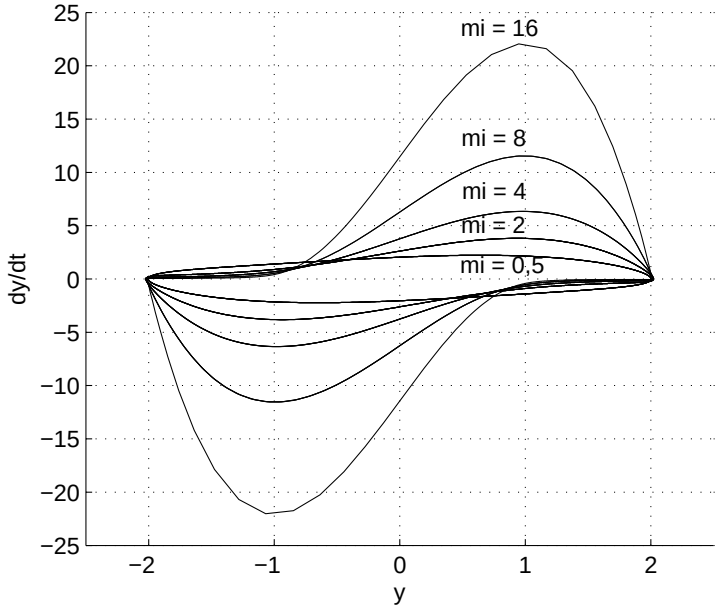
$$\begin{aligned} x_1 &= \dot{y} \\ x_2 &= -\omega^2 y \end{aligned} \tag{6.8}$$

możemy przekształcić równanie (6.7) do postaci:

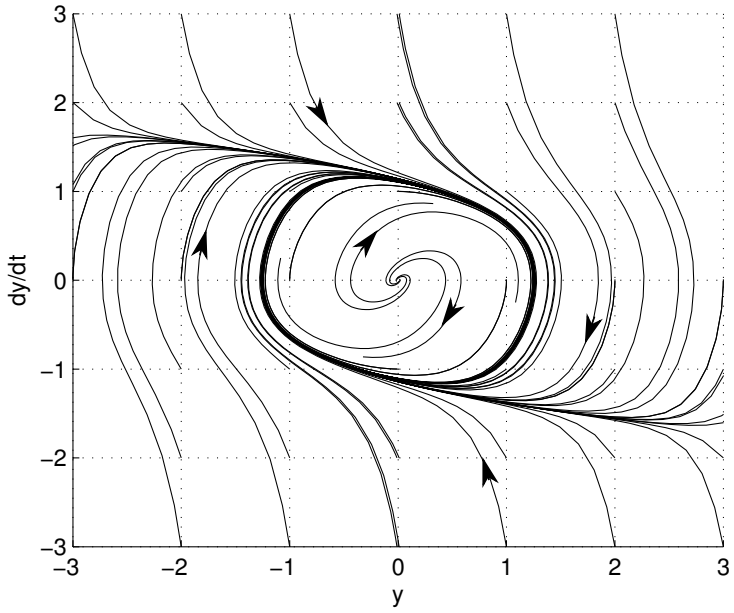
$$\begin{aligned} \dot{x}_1 &= x_2 + \mu\left(1 - x_1^2\right)x_1 \\ \dot{x}_2 &= -\omega^2 x_1 \end{aligned} \tag{6.9}$$

Jest to jedna z postaci układu Lienarda podana we wzorze (6.4). Funkcje

$$\begin{aligned} F(x) &= \mu x\left(1 - x^2\right) \\ g(x) &= \omega^2 x \end{aligned}$$



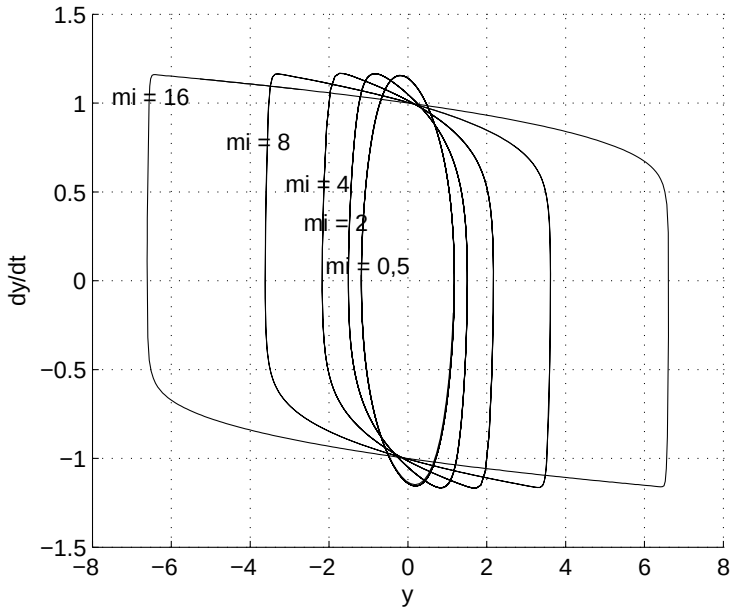
Rysunek 6.2: Cykl graniczny oscylatora Van der Pola dla różnych wartości współczynnika μ



Rysunek 6.3: Portret fazowy oscylatora Rayleigha

w przypadku równania oscylatora Rayleigha także spełniają warunki podane dla twierdzenia Lienarda co implikuje fakt, że rozwiązaniem równania będzie cykl graniczny w przestrzeni stanu [15].

W równaniu oscylatora Rayleigha (wzór (6.7)) występują dwa parametry. Współczynnik ω (ze znakiem +) reguluje dopływ energii do systemu. Współczynnik μ (ze znakiem −) reguluje sposób w jaki energia przepływa przez system. Parametr ω jest częstością oscylacji w układzie, natomiast wpływ parametru μ na kształt cyklu granicznego przedstawiono na rysunku 6.4.



Rysunek 6.4: Cykl graniczny oscylatora Rayleigha dla różnych wartości współczynnika μ

6.3 Wykorzystanie oscylatorów nieliniowych do budowy systemu sterowania

Zarówno omawiany powyżej oscylator Van der Pola jak i oscylator Rayleigha mogą zostać wykorzystane do realizacji układu sterującego chodem robota sześcionożnego w wersji stosującej chód trójpodporowy. W trakcie chodu końcówki nóg robota poruszają się w płaszczyźnie XZ zgodnie z przebiegami zdefiniowanymi na płaszczyznach fazowych oscylatorów. Wykorzystując zarówno oscylator Van der Pola jak i oscylator Rayleigha jesteśmy w stanie zrealizować wszystkie trzy fazy chodu robota, czyli:

- fazę rozruchu,
- chód właściwy,
- fazę zatrzymania.

Dla realizacji wszystkich trzech faz chodu algorytmu kroczenia należy zdefiniować system dynamiczny składający się z dwóch równań oscylatorów nieliniowych tego samego typu.

Sposób połączenia sygnałów sterujących, struktura systemu sterującego i zdefiniowane fazy ruchu pozostają analogiczne jak w omawianym wcześniej przypadku zastosowania oscylatorów liniowych (patrz rozdział 5.4). Oznacza to, że także w tym przypadku należy wprowadzić podział na grupy nóg A i B poruszające się w przeciwfazie.

Obydwa omawiane typy oscylatorów nieliniowych mogą być stosowane wymiennie, w zależności od zapotrzebowania, na przykład teren, po którym porusza się robot może być czynnikiem determinującym wybór oscylatora.

Realizacja **fazy rozruchu** i **fazy chodu właściwego** są ze sobą powiązane. Podczas chodu właściwego istotne jest, by oscylator sterujący nogami z grupy A pozostawał zawsze w przeciwfazie w stosunku do oscylatora sterującego nogami z grupy B. Efekt ten można uzyskać dzięki odpowiedniemu wymuszeniu wartości początkowej w fazie rozruchu. Wartości początkowe powinny wynosić:

$$\begin{bmatrix} x \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0 \\ C \end{bmatrix},$$

dla oscylatora sterującego grupą A oraz

$$\begin{bmatrix} x \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0 \\ -C \end{bmatrix},$$

dla oscylatora sterującego grupą B. Wartość współczynnika C jest odpowiednio małą liczbą dodatnią. Tak dobrany punkt początkowy sprawia, że oscylatory zostaną "wytrącone" z punktu równowagi $[0 \quad 0]^T$, a ich odwrotne znaki powodują, że oscylatory pozostaną w przeciwfazie. W fazie rozruchu korzystamy z największej zalety oscylatorów, jaką jest asymptotyczna stabilność cyklu granicznego w przestrzeni stanu oscylatora.

Należy zauważyć, że podane wartości początkowe odpowiadają sytuacji, w której nogi z grupy A zostały lekko uniesione do góry, a nogi z grupy B lekko opuszczone na dół. W przeciwieństwie do sytuacji, w której zastosowano oscylatory liniowe zmiana pozycji nóg w stosunku do pozycji bazowej może być niewielka (na granicy dokładności numerycznej procesora realizującego algorytm chodu) i wręcz niemierzalna z punktu widzenia układu mechanicznego robota. Rozwiązanie równania oscylatora Van der Pola lub Rayleigha będzie dążyć do cyklu granicznego gwarantując właściwe sterowanie chodem w fazie rozruchu (etap dochodzenia z punktu początkowego do cyklu granicznego) i w fazie chodu właściwego (ruch po cyklu granicznym).

Podczas chodu właściwego końcówki nóg robota będą poruszały się po trajektorii identycznej z kształtem cyklu granicznego odpowiedniego oscylatora. W przypadku zastosowania równania Van der Pola noga robota w fazie protrakcji jest unoszona wysoko i ustawiana na podłożu niemal pionowo. W fazie retrakcji noga w położeniu niemal pionowym jest unoszona z podłoża. Z tego właśnie powodu wydaje się, że zastosowanie równań Van der Pola do sterowania chodem robota sprawdzi się w sytuacji gdy robot będzie poruszał się po gruncie grząskim (miękkim) lub pokrytym drobnymi przedmiotami gdzie nie powinno dopuszczać się tak zwanego "szurania" nogami. Przykładem takiego terenu będzie trawnik, łąka, powierzchnia błotnista lub piaszczysta. Należy zauważyć, że wysokie unoszenie nóg spowolni ruch całego robota i zwiększy zapotrzebowanie na energię ze względu na wydłużenie trajektorii, po której porusza się końcówka nogi.

Dwa występujące w równaniach oscylatora parametry należy interpretować jako:

- ω – częstość pracy oscylatora równoważna z częstością przestawiania nóg (ilość cykli na sekundę),
- μ – współczynnik odpowiadający za kształt cyklu granicznego w efekcie ma wpływ na wysokość na jaką zostanie uniesiona noga robota.

O ile dobór częstości pracy wydaje się oczywisty to ustalenie wartości współczynnika μ jest problematyczne. Z jednej strony jego zmiana powoduje wyższe lub niższe unoszenie nogi robota, natomiast powoduje także zmianę czasu w jakim oscylator i noga przechodzą

z pozycji bazowej do cyklu granicznego w fazie rozruchu. Dlatego zmiana wartości współczynnika μ podczas pracy układu nie wydaje się celowa i ewentualną zmianę wysokości kroku lepiej wprowadzać poprzez dodatkowe skalowanie wartości sterującej. Współczynnik μ należy dobrać eksperymentalnie tak, by dla danej konstrukcji mechanicznej uzyskać efekt łagodnego rozpędzania w fazie rozruchu, a zarazem efekt wysokiego unoszenia nóg w fazie chodu właściwego.

W przeciwieństwie do oscylatora Van der Pola zastosowanie oscylatorów Rayleigha powoduje łagodne i szybkie przesuwanie poszczególnych grup nóg po powierzchni, bez efektu wysokiego unoszenia nóg. Dzięki sterowaniu z wykorzystaniem oscylatorów Rayleigha robot porusza się płynnie, zużywając mniej energii. Algorytm ten będzie dobrze sprawdzał się przy powierzchniach płaskich i niegrząskich. Dobór parametrów jest prostszy niż poprzednio. Wartość μ nie powoduje dużych zmian w kształcie cyklu granicznego i należy ją dobrać kierując się głównie dynamiką fazy rozruchu.

Pewne trudności powoduje próba uzyskania **fazy zatrzymania** zrealizowanej za pomocą przedstawionych wcześniej równań. Zarówno w przypadku zastosowania oscylatorów Rayleigha jak i oscylatorów Van der Pola nie jesteśmy w stanie uzyskać płynnego zatrzymania i powrotu do pozycji bazowej bez wprowadzenia pewnych modyfikacji do równań oscylatorów. Można jednak wykorzystać efekt pojawiający się w oscylatorze liniowym z tłumieniem i na okres fazy zatrzymania zamienić dynamikę oscylatorów zgodnie z równaniem (6.10) (w przypadku oscylatorów Van der Pola):

$$\ddot{y} = \begin{cases} \mu[1 - y^2]\dot{y} - \omega^2 y & , t < t_1 \\ -\mu\dot{y} - \omega^2 y & , t >= t_1 \end{cases} \tag{6.10}$$

lub równaniem (6.11) (w przypadku oscylatorów Rayleigha):

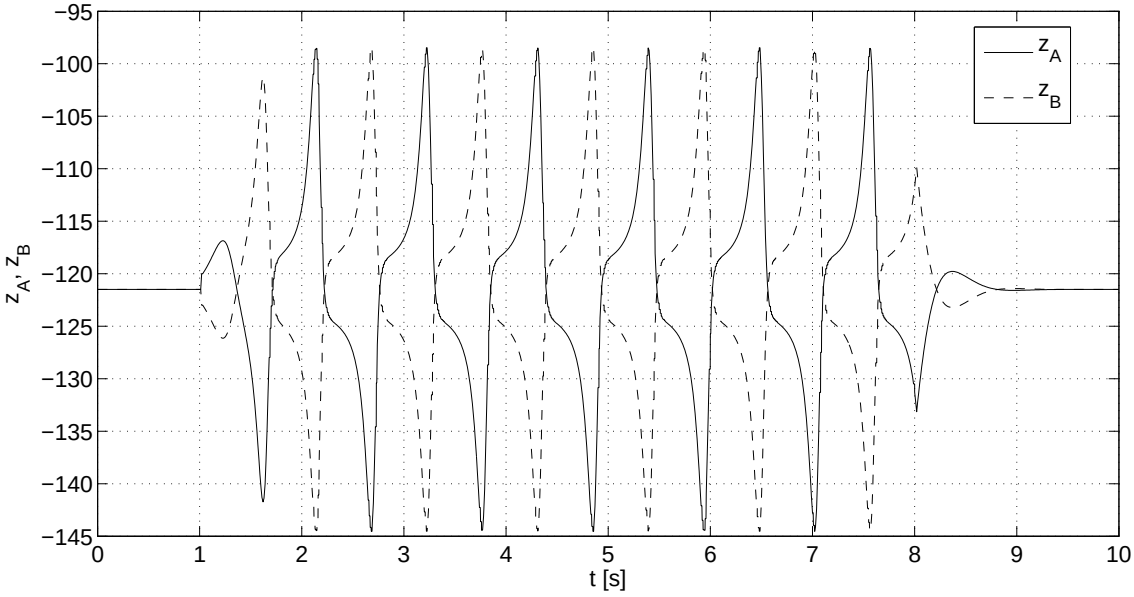
$$\ddot{y} = \begin{cases} \mu[1 - \dot{y}^2]\dot{y} - \omega^2 y & , t < t_1 \\ -\mu\dot{y} - \omega^2 y & , t >= t_1 \end{cases} \tag{6.11}$$

W obu przypadkach t_1 to chwila czasu, w której chcemy rozpocząć proces zatrzymywania się robota. Pozostawienie współczynnika μ w drugiej części równania ma na celu upodobnienie dynamiki procesu zatrzymania do procesu rozruchu.

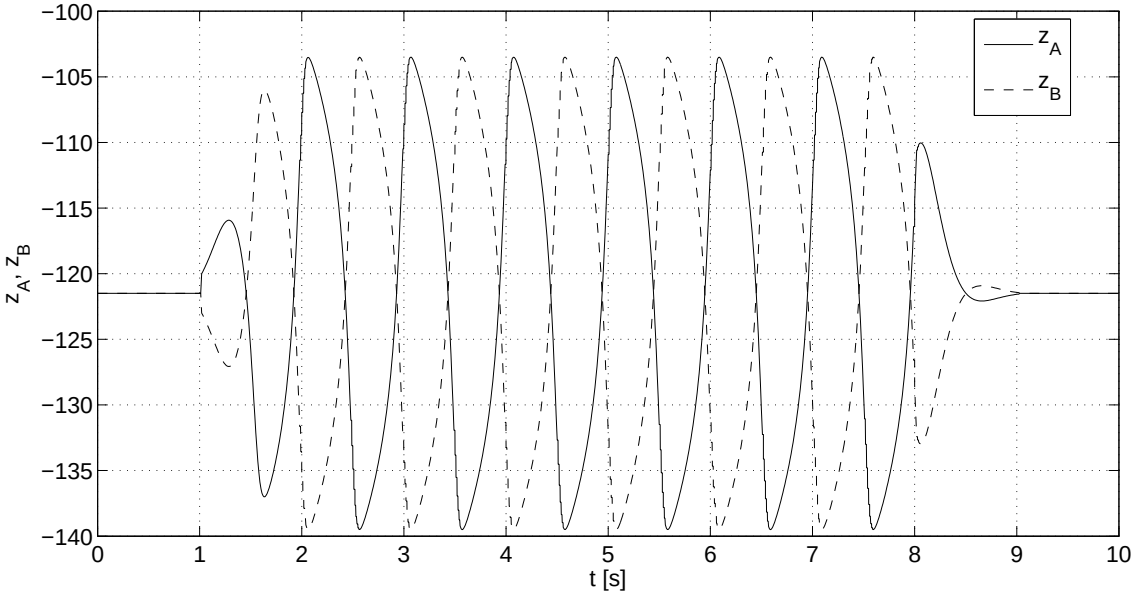
Na rysunku 6.5 przedstawiono przykładowe sterowanie ruchem nóg robota z wykorzystaniem równań oscylatorów Van der Pola we wszystkich trzech fazach ruchu. Na rysunku 6.6 przedstawiono przykładowe sterowanie z wykorzystaniem oscylatorów Rayleigha.

Oscylatory Van der Pola i oscylatory Rayleigha mogą być stosowane wymiennie w zależności od informacji z czujników pomiarowych na temat podłoża. Istnieje możliwość płynnego przełączania pomiędzy jednym układem, a drugim, należy jednak pamiętać, że wartości zmiennych stanu nie mogą być bezpośrednio przepisywane z oscylatorów Van der Pola do oscylatorów Rayleigha bez odpowiedniego skalowania. Jest to spowodowane różnicami w wartościach uzyskiwanych przez te układu przy takich samych parametrach. Rysunek 6.7 przedstawia schemat połączeń układu sterującego opartego na oscylatorach Van der Pola i Rayleigha.

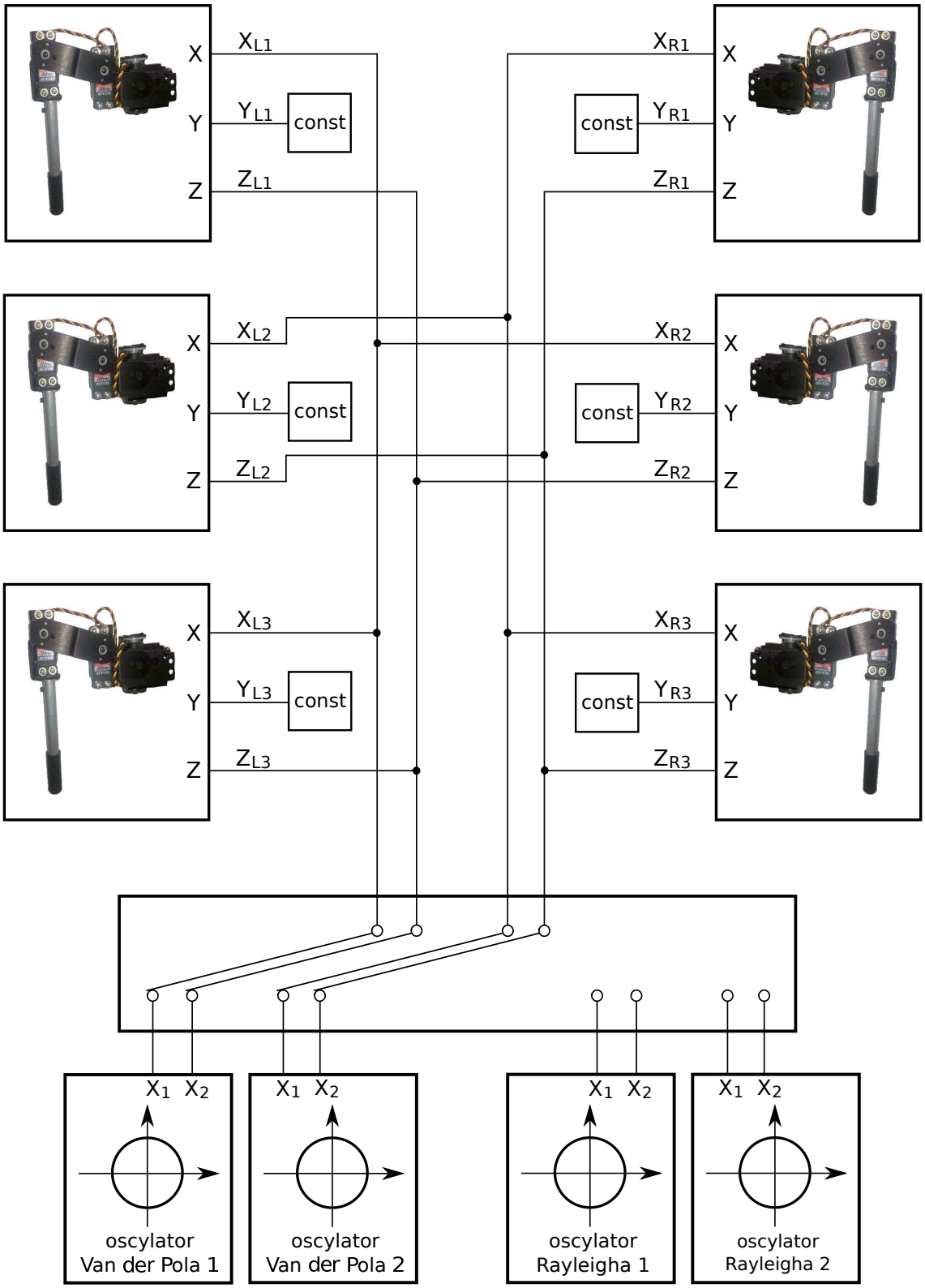
Działanie algorytmu realizującego chód trójpodporowy sześcionożnego robota kroczącego zarówno z wykorzystaniem oscylatorów Van der Pola jak i oscylatorów Rayleigha zostało zilustrowane na rysunku 6.8. Przedstawiony przypadek przedstawia sygnały sterujące ruchem końcówek nogi robota zgodnie z zasadą stosowaną dla poprzednich wykresów. Do przełączeń pomiędzy dwoma typami oscylatorów doszło odpowiednio w 3 i 7 sekundzie. Dzięki dynamice oscylatorów samo przełączenie jest procesem stosunkowo płynnym. Można to także zaobserwować na rysunku 6.9. Rysunek ten przedstawia płaszczyznę fazową jednego z oscylatorów czyli ruch końcówki jednej z grup nóg w płaszczyźnie XZ.



Rysunek 6.5: Sygnały sterujące chodem robota w układzie stosującym dwa oscylatory Van der Pola: z_A – sygnał sterujący nogami z grupy A, z_B – sygnał sterujący nogami z grupy B

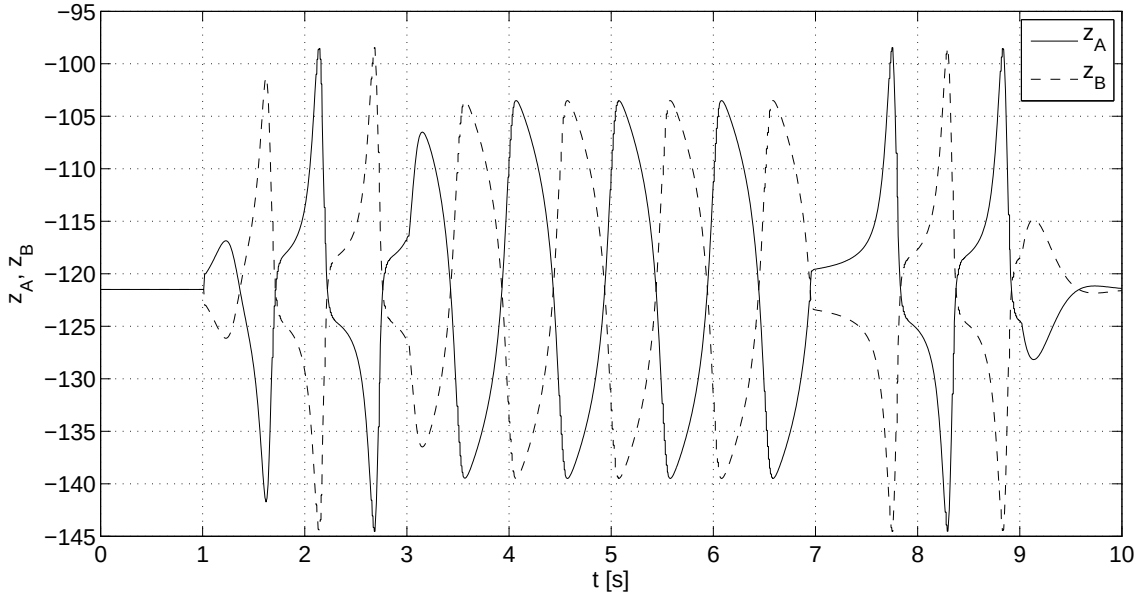


Rysunek 6.6: Sygnały sterujące chodem robota w układzie stosującym dwa oscylatory Rayleigha: z_A – sygnał sterujący nogami z grupy A, z_B – sygnał sterujący nogami z grupy B



Rysunek 6.7: Schemat ideowy systemu sterującego chodem robota sześcionożnego poruszającego się chodem trójpodporowym z wykorzystaniem oscylatorów Van der Pola i oscylatorów Rayleigha

Widoczne jest na nim nim ślad planowanej trajektorii końcówki nogi robota dla sześciu niezależnych przypadków.



Rysunek 6.8: Sygnały sterujące chodem robota w układzie stosującym oscylatory Van der Pola (od 0 do 3 sekundy oraz od 7 do 11 sekundy) i oscylatory Rayleigha (od 3 do 7 sekundy): z_A – sygnał sterujący nogami z grupy A, z_B – sygnał sterujący nogami z grupy B

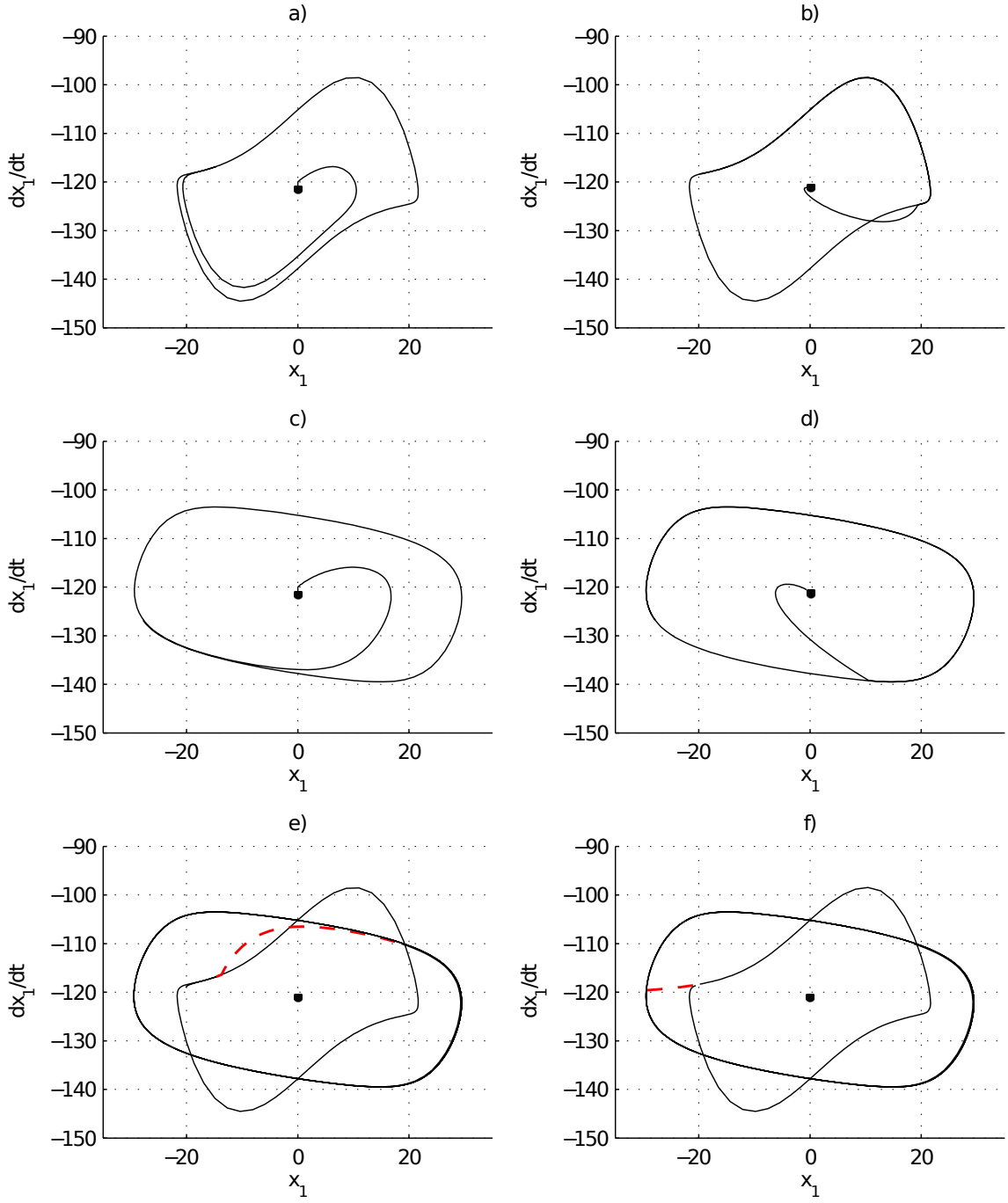
Synteza opisywanego algorytmu powinna się rozpocząć od doboru żądanych kształtów cykli granicznych dla obydwu typów oscylatora, a następnie ich normalizacji tak by osiągnąć podobne długości kroków dla obu typów oscylatora. Następnie implementuje się oba algorytmy, a do ich przełączania powinno dochodzić według następującego schematu:

- zatrzymanie algorytmu sterowania oscylatorami typu 1,
- zerowanie zmiennych stanu algorytmu sterowania oscylatorami typu 1,
- odczyt aktualnej informacji o sterowaniu,
- przeliczenie zmiennych sterujących zgodnie ze wzorem: $x_0 = x/k$, gdzie x to zmienne przechowujące wartość sterowania, a k to współczynnik normalizacji dla oscylatorów typu 2,
- podstawienie wyliczonych w poprzednim kroku wartości x_0 za zmienne stanu dla algorytmu z oscylatorami typu 2,
- start algorytmu z oscylatorami typu 2

6.4 Stabilność algorytmu opartego na oscylatorach nieliniowych

W rozdziale 6.3 przedstawiono algorytm generujący trajektorie ruchu dla końcówek nóg sześcionożnego robota kroczącego. W algorytmie wykorzystano trzy niezależne równania dynamiki:

- nieliniowe równanie oscylatora Van der Pola,



Rysunek 6.9: Sygnał sterujące końcówką nóg robota z grupy A w układzie stosującym naprzemiennie oscylatory Van der Pola i oscylatory Rayleigha: a) faza rozruchu w trybie oscylatora Van der Pola, b) faza zatrzymania w trybie oscylatora Van der Pola, c) faza rozruchu w trybie oscylatora Rayleigha, d) faza zatrzymania w trybie oscylatora Rayleigha, e) przejście z trybu oscylatora Van der Pola w tryb oscylatora Rayleigha (trajektorie przejścia oznaczono linią przerywaną), f) przejście z trybu oscylatora Van der Pola w tryb oscylatora Rayleigha

- nieliniowe równanie oscylatora Rayleigha,
- liniowe równanie oscylatora harmonicznego tłumionego.

Sam algorytm polega na regułowym przełączaniu pomiędzy tymi trzema równaniami z przepisaniem wartości zmiennych stanu z równania deaktywowanego do równania aktywowanego w momencie przełączenia. Każde z wymienionych równań, rozważane samodzielnie, posiada trajektorie asymptotyczynie stabilne. Niestety ta właściwość nie implikuje asymptotycznej stabilności trajektorii generowanych przez cały algorytm. Poprawne wyniki symulacji i eksperymentów na układzie rzeczywistym pozwalają przypuszczać, że taka właściwość zachodzi. Pozwala to na sformułowanie następującej hipotezy:

Hipoteza o ograniczoności trajektorii generowanej przez algorytm oparty na oscylatorach nieliniowych. *System dynamiczny z regułowym przełączaniem (w sposób opisany w rozdziale 6.3 niniejszej pracy) pomiędzy trzema równaniami dynamiki w postaci:*

$$\ddot{y} = \mu[1 - y^2]\dot{y} - \omega^2 y$$

$$\ddot{y} = \mu[1 - \dot{y}^2]\dot{y} - \omega^2 y$$

i

$$\ddot{y} = -\mu\dot{y} - \omega^2 y$$

generuje trajektorie ograniczone.

Dowód tej hipotezy jest trudny do przeprowadzania i pozostaje sprawą otwartą.

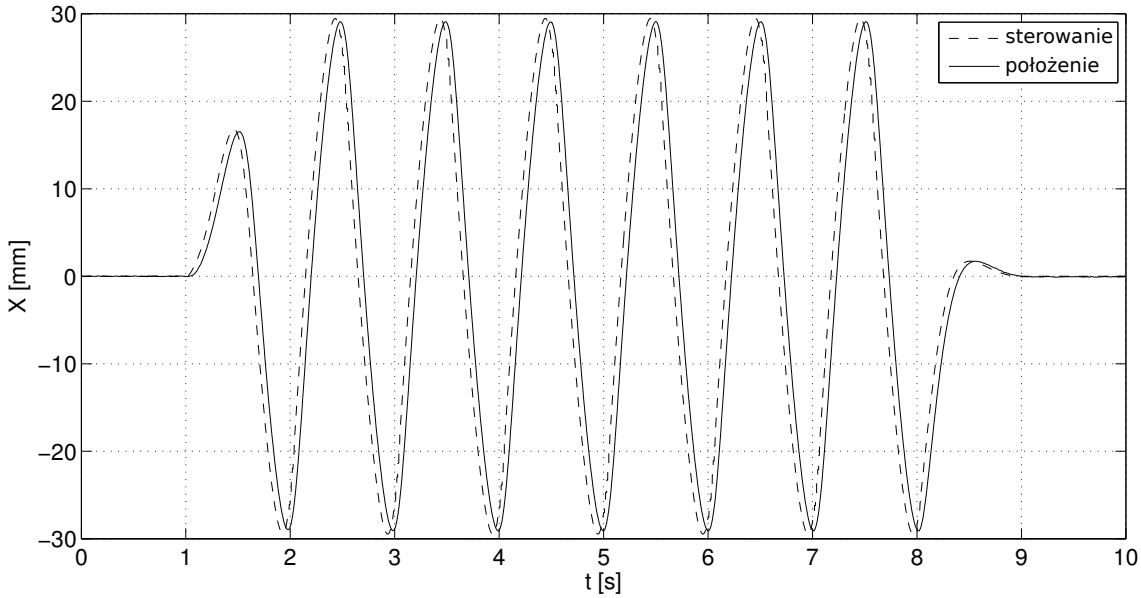
6.5 Wyniki symulacji układu sterowania opartego na oscylatorach nieliniowych

Na rysunkach 6.10 i 6.11 przedstawiono wyniki symulacji działania algorytmu sterującego chodem robota sześcionożnego wykorzystującego oscylatory Rayleigha. Przedstawiono położenie końcówki nogi w osi X i w osi Z. Najbardziej interesujące fragmenty wykresów to płynne fazy rozruchu (w 1 sekundzie symulacji) i płynne fazy zatrzymania (początek w 8 sekundzie symulacji). W porównaniu do oscylatorów liniowych znacznie poprawiły się przebiegi w fazie rozruchu. Obserwuje się opóźnienie między sygnałami sterowania i położenia końcówki nogi znane z algorytmu sterowania opartego na oscylatorach liniowych. Przyczyną opóźnienia jest dynamika modelu.

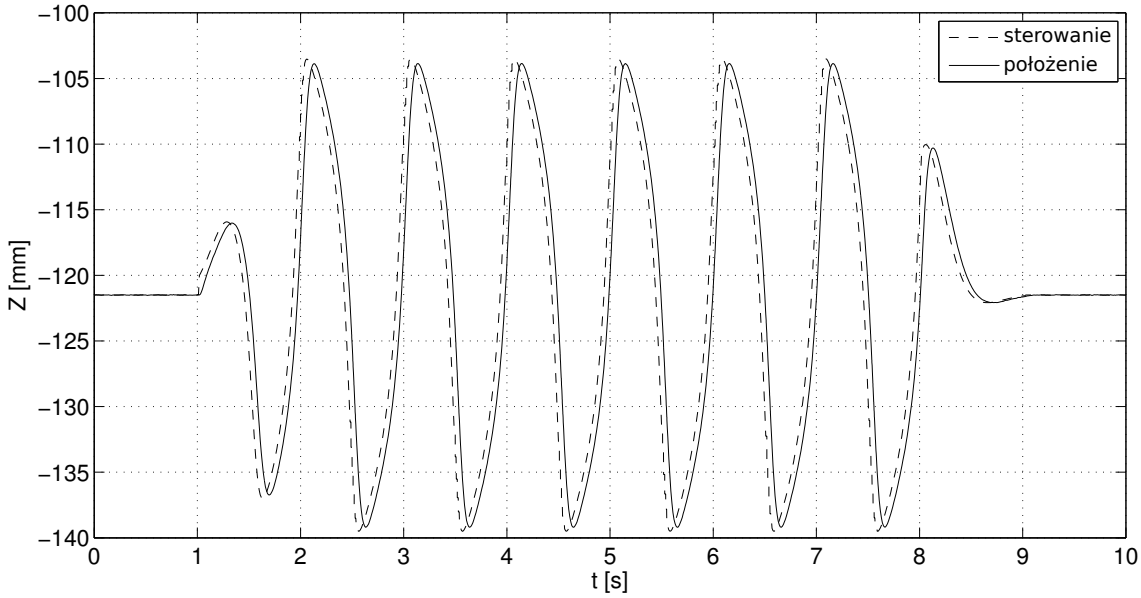
Na wykresach 6.12 i 6.13 zamieszczono wyniki symulacji sterowania chodem robota sześcionożnego. Działający algorytm wykorzystuje oscylatory Van der Pola. Obserwuje się charakterystyczne podnoszenie i opuszczenie nogi dla ułatwienie poruszania się po miękkim podłożu. Także w tym przypadku udało się uzyskać efekt płynnej realizacji fazy rozruchu (1 sekunda) i fazy zatrzymania (8 sekunda).

Na wykresach 6.14, 6.15, 6.16 i 6.17 zamieszczono wyniki symulacji algorytmu sterującego chodem robota sześcionożnego opartego na układzie oscylatorów Van der Pola i Rayleigha z możliwością przełączania pomiędzy tym dwoma rodzajami oscylatorów. Przełączenie zachodzi w 5 sekundzie trwania eksperymentów. Zmiana oscylatorów odbywa się płynnie. Jedyny niepokojący efekt pojawia się na wykresie 6.15. Efekt ten dostrzeżono, w niektórych przełączeniach (w zależności od chwili przełączenia) z trybu wykorzystującego oscylatory Rayleigha na tryb wykorzystujący oscylatory Van der Pola w osi Z (tylko i wyłącznie). Efekt ten nie powinien mieć jednak wpływu na przemieszczanie się robota.

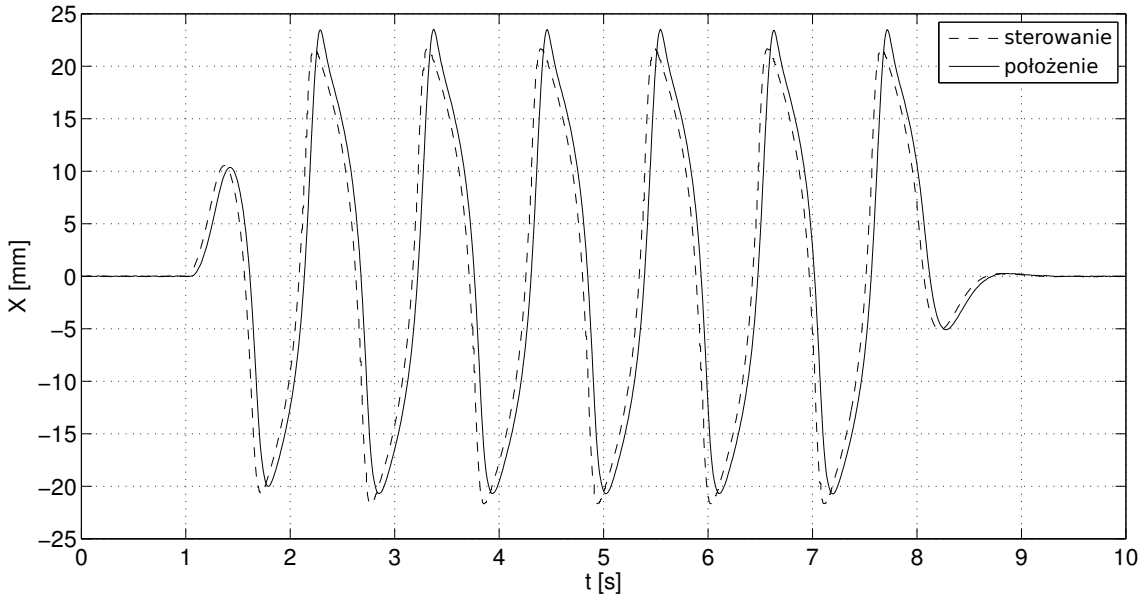
Na rysunku 6.18 przedstawiono wyniki symulacji w postaci wykresu śladu końcówki nogi robota z grupy A na płaszczyźnie XZ. Wykres ten przedstawia sześć najbardziej interesujących aspektów działania algorytmu to znaczy fazę rozruchu, zatrzymania i przejścia



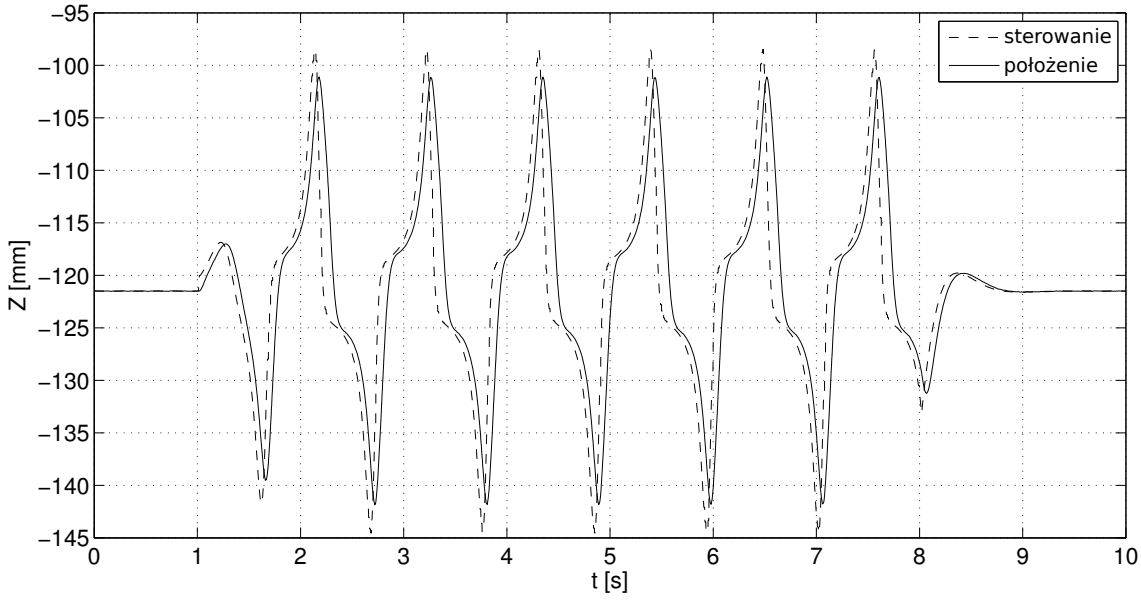
Rysunek 6.10: Sterowanie oraz położenie końcówki jednej z nóg robota w osi X w funkcji czasu – wyniki symulacji modelu robota sterowanego algorytmem opartym na oscylatorach nieliniowych z aktywnymi oscylatorami Rayleigha



Rysunek 6.11: Sterowanie oraz położenie końcówki jednej z nóg robota w osi Z w funkcji czasu – wyniki symulacji modelu robota sterowanego algorytmem opartym na oscylatorach nieliniowych z aktywnymi oscylatorami Rayleigha

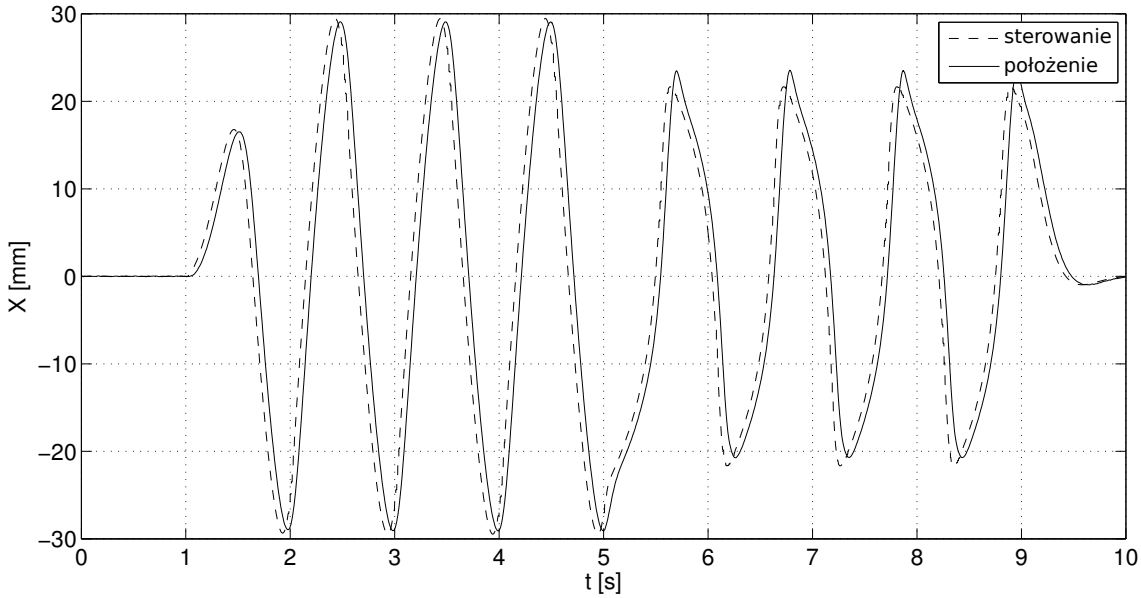


Rysunek 6.12: Sterowanie oraz położenie końcówki jednej z nóg robota w osi X w funkcji czasu – wyniki symulacji modelu robota sterowanego algorytmem opartym na oscylatorach nieliniowych z aktywnymi oscylatorami Van der Pola



Rysunek 6.13: Sterowanie oraz położenie końcówki jednej z nóg robota w osi Z w funkcji czasu – wyniki symulacji modelu robota sterowanego algorytmem opartym na oscylatorach nieliniowych z aktywnymi oscylatorami Van der Pola

w tryb oscylatora Van der Pola i w tryb oscylatora Rayleigha. Na wykresie widać zniekształcenia w stosunku do trajektorii planowanej (patrz rysunek 6.9) wynikające z dynamiki układu nogi robota.



Rysunek 6.14: Sterowanie oraz położenie końcówki jednej z nóg robota w osi X w funkcji czasu – wyniki symulacji modelu robota sterowanego algorytmem opartym na oscylatorach nieliniowych z aktywnymi oscylatorami Rayleigha przełączonymi w 5 sekundzie na oscylatory Van der Pola

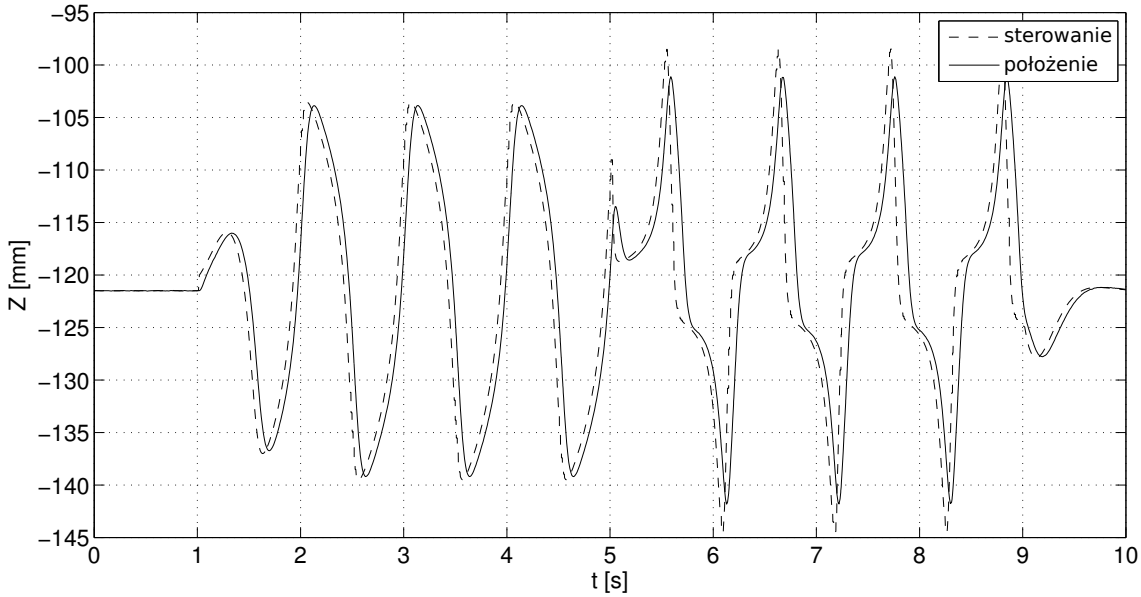
6.6 Wyniki eksperymentów rzeczywistych chodu robota

Algorytm oparty na oscylatorach nieliniowych został zaimplementowany na stanowisku rzeczywistego robota sześcionożnego. Z przygotowanego modelu symulacyjnego wygenerowano kod źródłowy w języku C w postaci dwóch funkcji:

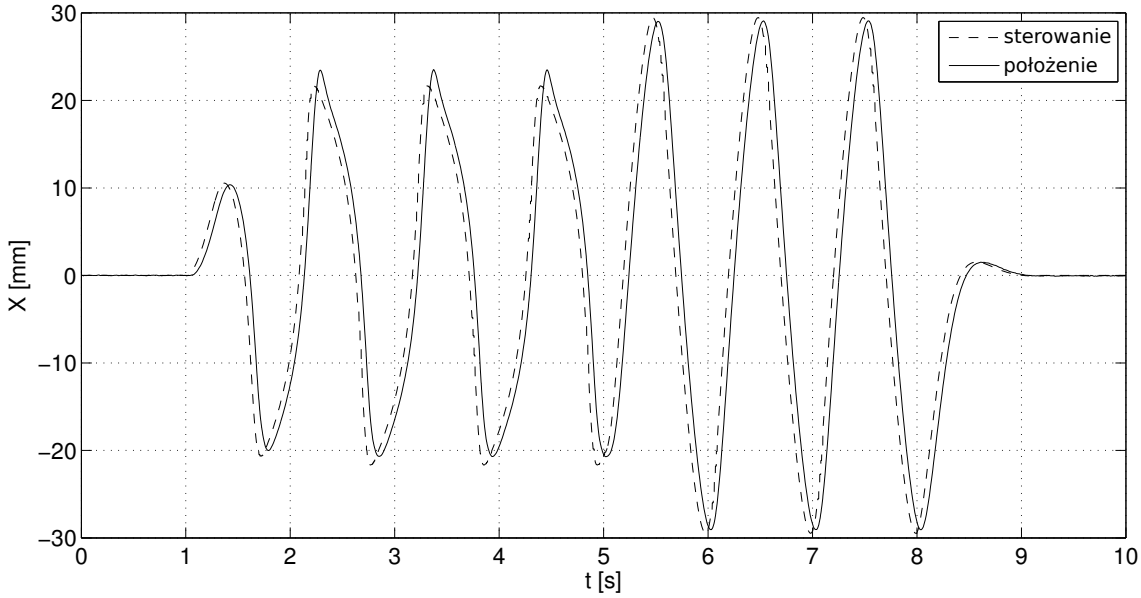
- funkcji inicjującej *CPG_nonlinear_init*,
- funkcji wyliczającej jeden krok właściwego algorytmu *CPG_nonlinear_step*.

Algorytm został zintegrowany ze stanowiskiem robota i uruchomiony. Przetestowano eksperymentalnie wszystkie fazy chodu trybu opartego na oscylatorze Van der Pola i trybu opartego na oscylatorze Rayleigha. Wyniki potwierdziły poprawność wcześniejszych eksperymentów symulacyjnych. Na rysunkach 6.19 i 6.20 przedstawiono klatki filmu z kolejnych faz ruchu jednej z nóg robota. Klatki pochodzą z eksperymentu podczas, którego robot był podwieszony na specjalnie przygotowanym statywie, co pozwoliło na nałożenie na nie oczekiwanej trajektorii końcówki nogi, pochodzącej z eksperymentów symulacyjnych. Niestety dokładność tej metody pozostawia wiele do życzenia. Pozwala jednak na weryfikację poprawności założeń algorytmu.

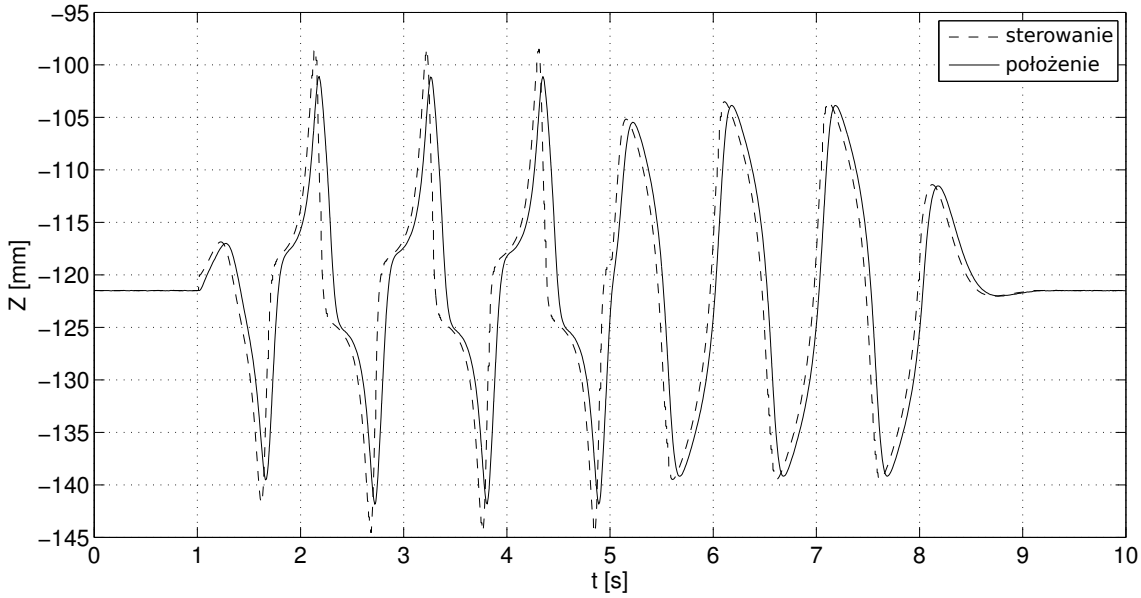
Podczas eksperymentów z robotem przemieszczającym się po podłożu zgodnie z przyjętymi założeniami udało się uzyskać ruch postępowy robota. Potwierdzono także możliwość zastosowania trybu opartego na oscylatorze Van der Pola do przemieszczania się po miękkim (grząskim) podłożu.



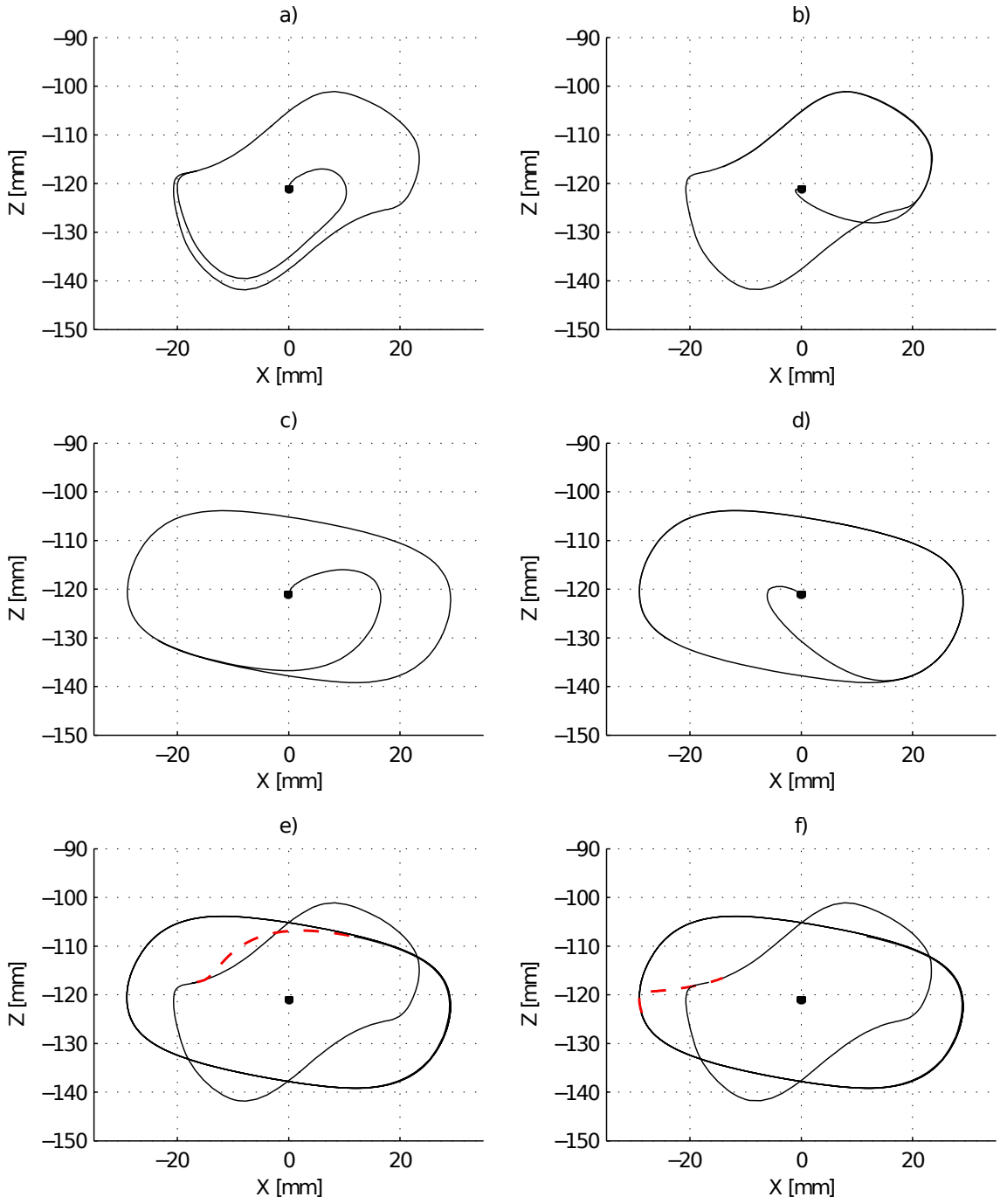
Rysunek 6.15: Sterowanie oraz położenie końcówki jednej z nóg robota w osi Z w funkcji czasu – wyniki symulacji modelu robota sterowanego algorytmem opartym na oscylatorach nieliniowych z aktywnymi oscylatorami Rayleigha przełączonymi w 5 sekundzie na oscylatory Van der Pola



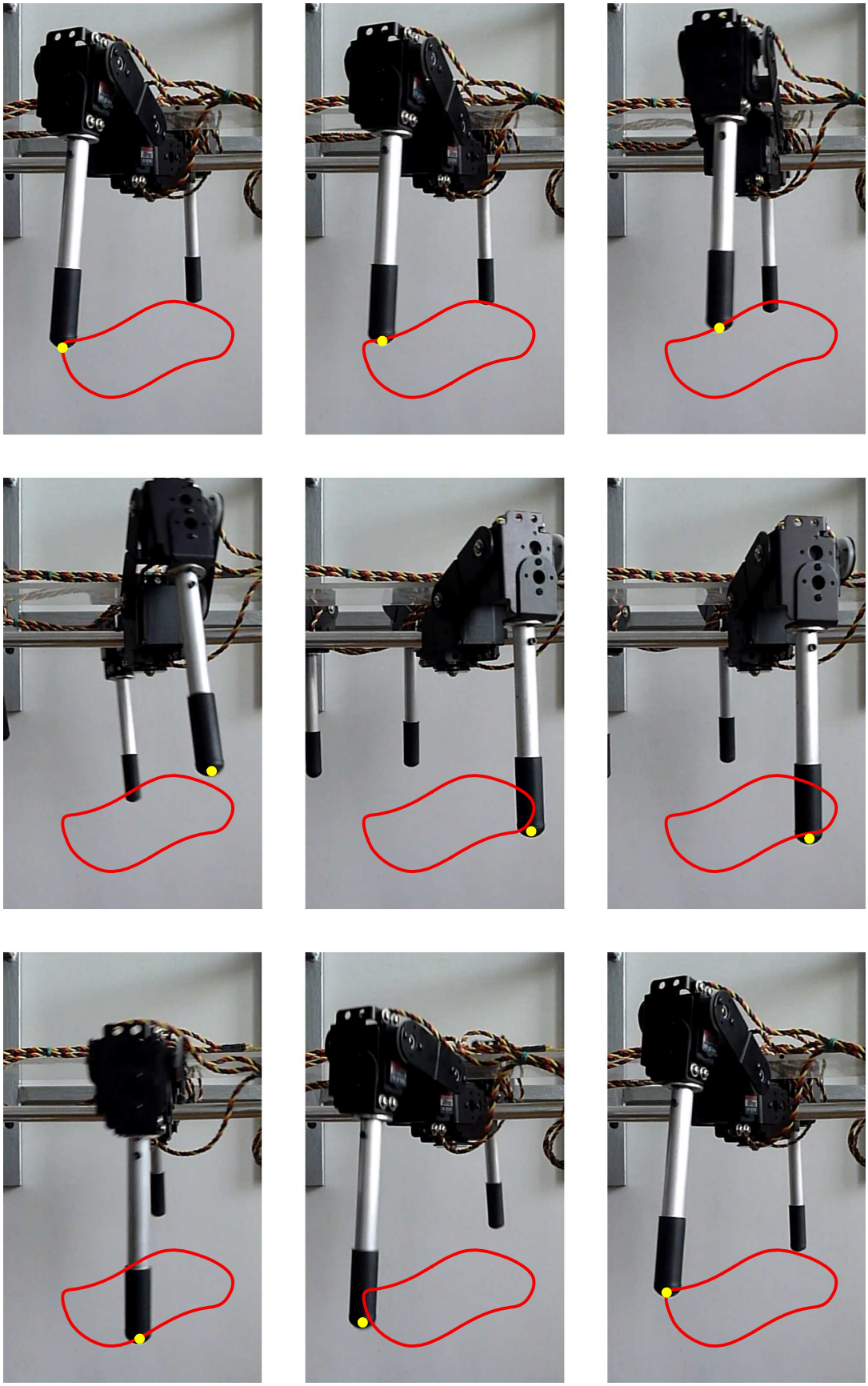
Rysunek 6.16: Sterowanie oraz położenie końcówki jednej z nóg robota w osi X w funkcji czasu – wyniki symulacji modelu robota sterowanego algorytmem opartym na oscylatorach nieliniowych z aktywnymi oscylatorami Van der Pola przełączonymi w 5 sekundzie na oscylatory Rayleigha



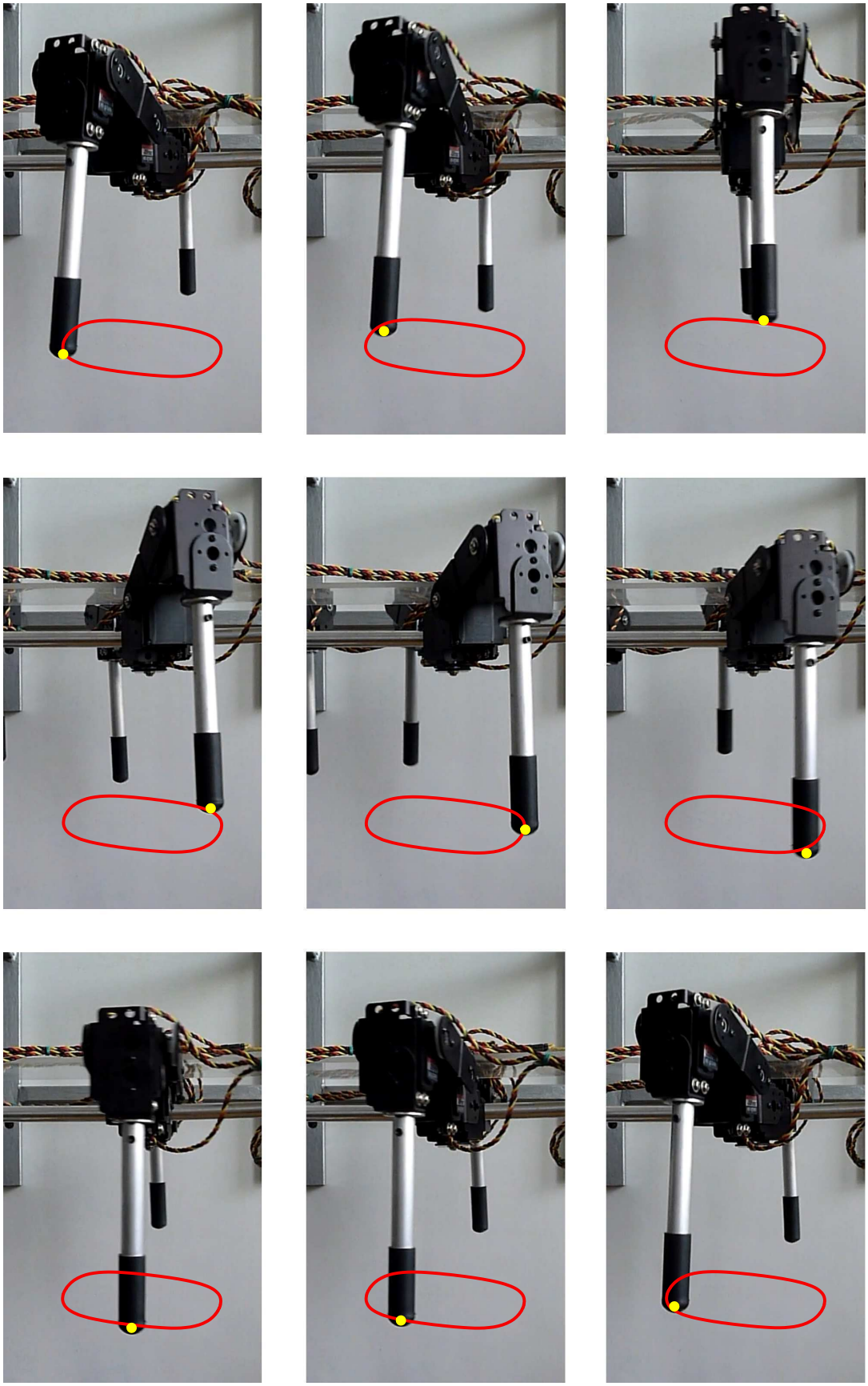
Rysunek 6.17: Sterowanie oraz położenie końcówki jednej z nóg robota w osi Z w funkcji czasu – wyniki symulacji modelu robota sterowanego algorytmem opartym na oscylatorach nieliniowych z aktywnymi oscylatorami Van der Pola przełączonymi w 5 sekundzie na oscylatory Rayleigha



Rysunek 6.18: Ślad położenia końcówki nogi robota z grupy A w układzie stosującym naprzemiennie oscylatory Van der Pola i oscylatory Rayleigha: a) faza rozruchu w trybie oscylatora Van der Pola, b) faza zatrzymania w trybie oscylatora Van der Pola, c) faza rozruchu w trybie oscylatora Rayleigha, d) faza zatrzymania w trybie oscylatora Rayleigha, e) przejście z trybu oscylatora Van der Pola w tryb oscylatora Rayleigha (trajektorie przejścia oznaczono linią przerywaną), f) przejście z trybu oscylatora Van der Pola w tryb oscylatora Rayleigha



Rysunek 6.19: Klatki filmowe ze sterowania robotem z wykorzystaniem algorytmu opartego na oscylatorze Van der Pola. Nałożono symulacyjne trajektorie ruchu końcówki nogi. Jasny punkt na trajektorii ilustruje bieżące położenie końcówki nogi.



Rysunek 6.20: Klatki filmowe ze sterowania robotem z wykorzystaniem algorytmu opartego na oscylatorze Rayleigha. Nałożono symulacyjne trajektorie ruchu końcówki nogi. Jasny punkt na trajektorii ilustruje bieżące położenie końcówki nogi.

Rozdział 7

Wykorzystanie kraty Toda-Rayleigha

Kratą Toda-Rayleigha nazywamy sieć Tody (patrz rozdział 7.1), w której każdy element jest oscylatorem Rayleigha (patrz rozdział 6.2). Konstrukcja ta została zbadana i opisana w serii artykułów niemiecko-hiszpańskiego zespołu badawczego [27], [26], [9], [25] i [8]. Już od samego początku zwrócono w nich uwagę na możliwość zastosowania kraty Toda-Rayleigha w układach sterowania chodem robotów kroczących.

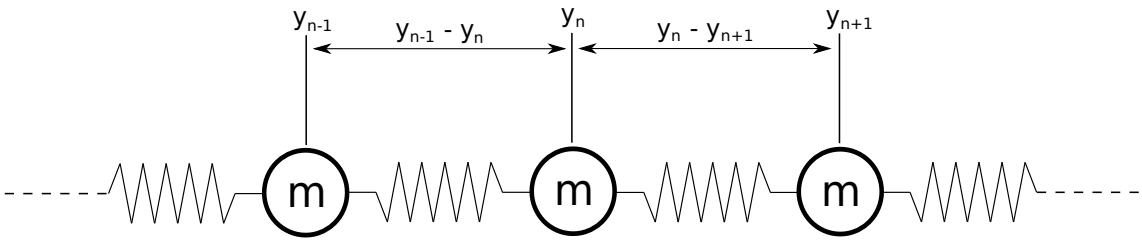
7.1 Krata Tody

Krata Tody (ang. Toda lattice) jest to prosty model jednowymiarowego kryształu stosowany w fizyce ciała stałego. Opisuje on łańcuch sąsiadujących ze sobą elementów pomiędzy, którymi dochodzi do interakcji opisanych równaniem:

$$\begin{aligned} \dot{p}_n &= e^{y_{n-1}-y_n} - e^{y_n-y_{n+1}} \\ \dot{y}_n &= p_n \end{aligned} \tag{7.1}$$

W równaniu (7.1) y_n oznacza przesunięcie n-tego elementu, a p_n oznacza pęd n-tego elementu przy założeniu, że masa każdego z elementów kraty wynosi 1. Równanie to zostało przedstawione w pracy [53] i jest jednym z klasycznych przykładów równania dynamiki generującego solitony [12].

Kratę Toda możemy przez analogię przedstawić jako układ mas połączonych ze sobą przy pomocy sprężyn o nieliniowej (ekspotencjalnej) charakterystyce sprężystości (patrz rysunek 7.1). Gdy każda z mas posiada taką samą wartość energii układ pozostaje stabilny. Gdy w jednym z elementów układu pojawi się dodatkowo porcja energii (na przykład pochodząca z zewnętrznego wymuszenia) jest ona przenoszona do sąsiednich elementów układu dzięki sprężystym połączeniom. W efekcie uzyskuje się niegasnącą falę propagowaną przez poszczególne masy, ponieważ w układzie nie występuje tarcie czy inna forma dyssypacji.

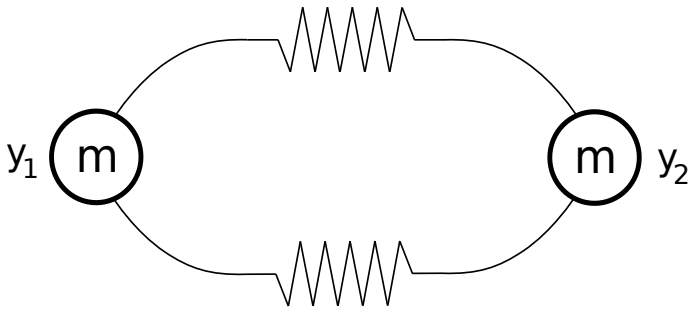


Rysunek 7.1: Układ mechaniczny opisany równaniem kraty Toda

Szczególnym przypadkiem kraty Toda jest układ, w którym istnieje skończona liczba elementów N oraz jest spełniony warunek:

$$y_{n+N} = y_n \tag{7.2}$$

Warunek ten oznacza, że elementy kraty zostały połączone w zamknięty pierścień. W tym przypadku dodatkowa energia pojawiająca się w układzie dzięki sprężystym połączeniom będzie krążyć w układzie powodując drgania mas. Rysunek 7.2 przedstawia przykładowy pierścień, dla którego $N = 2$. Na rysunku 7.3 możemy zaobserwować wyniki symulacji numerycznej takiego układu dla różnych warunków początkowych. Niezerowe warunki początkowe równania są równoważne z wystąpieniem zewnętrznego wymuszenia. Na wykresach widać wygenerowany soliton krążący w układzie w postaci funkcji sinusoidalnej.



Rysunek 7.2: Układ mechaniczny pierścienia Toda składającego się z dwóch elementów

W przypadku gdy ilość elementów układu będzie większa niż dwa np. dla wartości $N = 6$ wygenerowany soliton będzie miał znacznie bardziej złożoną postać ze względu na efekt nakładania się fal pochodzących z przeciwnych kierunków. W dalszym ciągu jest to jednak funkcja okresowa, którą można przedstawić w postaci:

$$\xi = A_0 \sin(\omega_0 t + \phi_0) + A_1 \sin(\omega_1 t + \phi_1) + ... + A_n \sin(\omega_n t + \phi_n) \tag{7.3}$$

Rysunek 7.4 pokazuje pierścień Toda składający się z sześciu elementów, a rysunek 7.5 przedstawia wyniki symulacji numerycznej takiego układu przy prostym wymuszeniu (nie zerowym warunku początkowym) dla jednego elementu.

7.2 Krata Toda-Rayleigha

W pracy [27] po raz pierwszy została opisana dynamika układu nazwanego kratą Toda-Rayleigha (ang. Toda-Rayleigh lattice). Układ ten opisuje równanie (7.4).

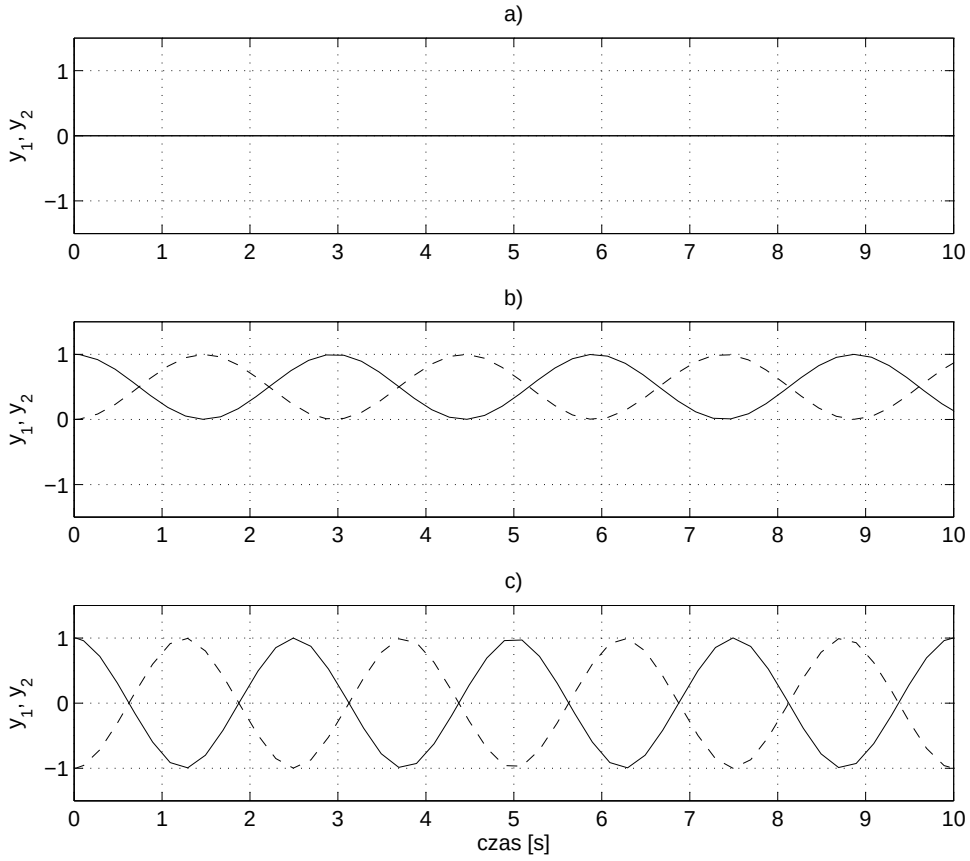
$$\begin{aligned} \dot{p}_n &= \omega_0^2 \left(e^{y_{n-1}-y_n} - e^{y_n - y_{n+1}}\right) + \left(\mu - p_n^2\right) p_n \\ \dot{y}_n &= p_n \end{aligned} \tag{7.4}$$

Jest to połączenie kraty Toda z oscylatorem Rayleigha. Identycznie jak w równaniu Toda mamy część równania opisującą wymianę energii pomiędzy poszczególnymi elementami pierścienia, ale dodatkowo pojawia się element odpowiadający za wytracanie i pompowanie energii pochodzący z równania oscylatora Rayleigha.

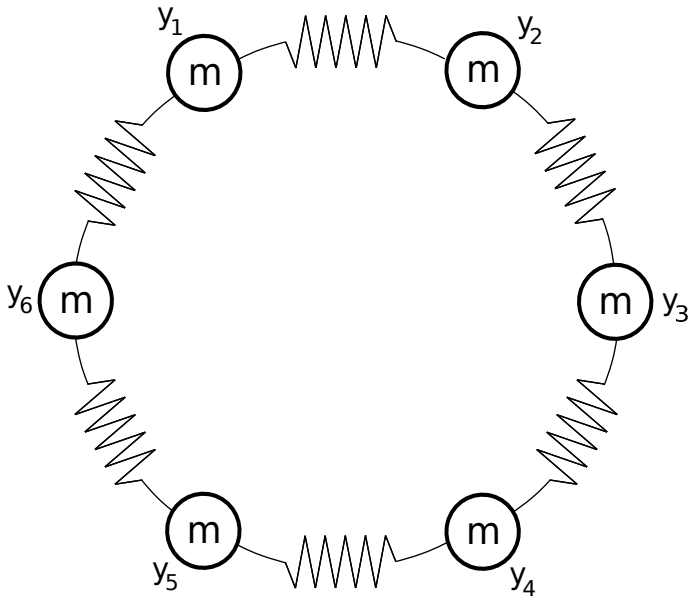
Do dalszych rozważań będziemy zakładać, że opisany równaniem (7.4) układ składa się z N elementów, przy czym zachodzi zależność:

$$y_{n+N} = y_n \tag{7.5}$$

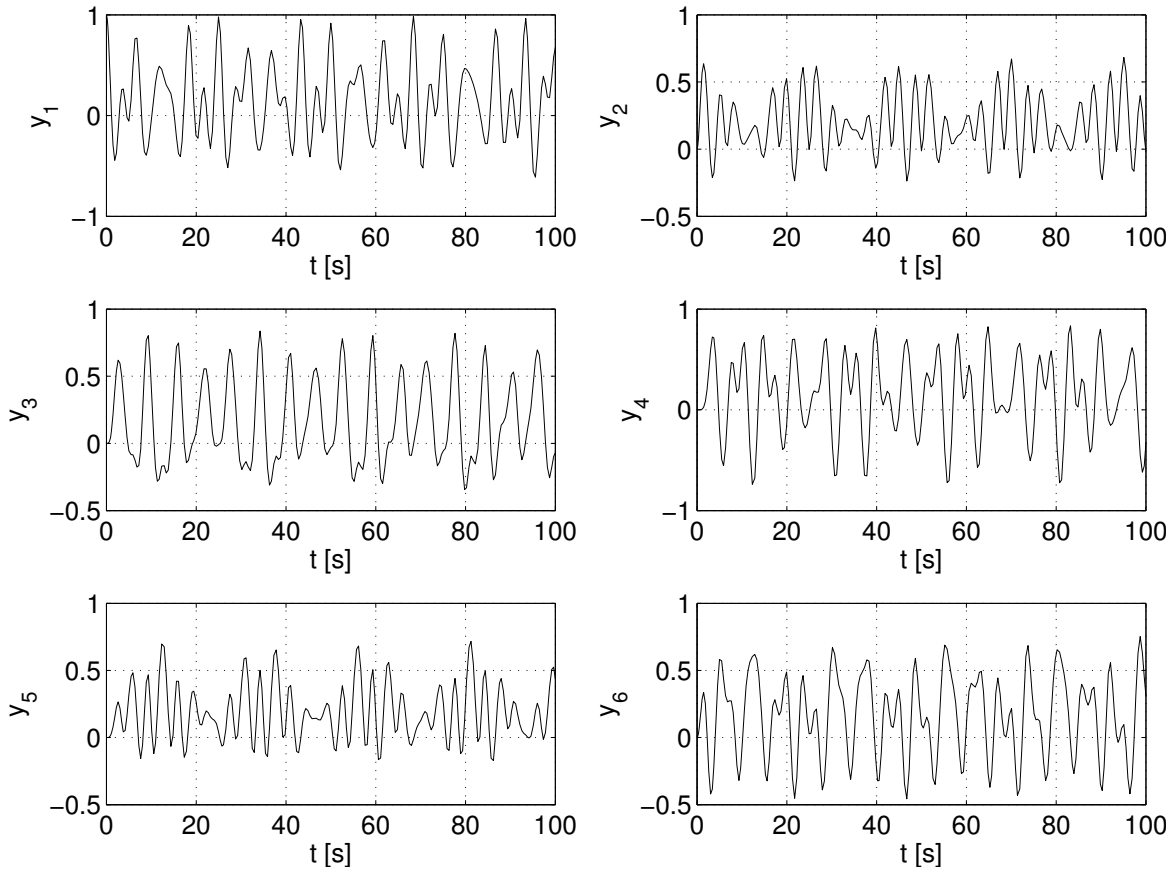
Oznacza to, że elementy kraty Toda-Rayleigha spięte są ze sobą na kształt pierścienia.



Rysunek 7.3: Wyniki symulacji pierścienia Toda składającej się z dwóch elementów gdzie warunki początkowe wynoszą a) $y_1 = 0, y_2 = 0$, b) $y_1 = 1, y_2 = 0$, c) $y_1 = 1, y_2 = -1$



Rysunek 7.4: Układ mechaniczny pierścienia Toda składającego się z sześciu elementów



Rysunek 7.5: Wyniki symulacji pierścienia Toda składającej się z sześciu elementów

Analizując równanie (7.4) można wywnioskować, że układ posiada przynajmniej trzy rozwiązania, dla których nie pojawiają się oscylacje. W takiej sytuacji wartość odchyleń pomiędzy sąsiednimi elementami kraty oraz ich przyspieszenie, a zarazem pochodna pędu przyjmują wartość zero czyli

$$\begin{aligned} \dot{p}_n &= 0 \\ y_{n-1} - y_n &= 0 \\ y_n - y_{n+1} &= 0 \end{aligned} \tag{7.6}$$

Po podstawieniu tych warunków do (7.4) otrzymujemy trzy przypadki, dla których pęd przyjmuje wartości:

$$\begin{aligned} p_n &= 0 \\ p_n &= \sqrt{\mu} \\ p_n &= -\sqrt{\mu} \end{aligned} \tag{7.7}$$

Pierwszy przypadek odpowiada sytuacji gdy cały układ stoi w miejscu i jest to niestabilny punkt równowagi. Dwa pozostałe odpowiadają sytuacji, gdy cały pierścień obraca się ze stałą prędkością w prawą lub lewą stronę (odległości pomiędzy elementami nie zmieniają się) i są to rozwiązania stabilne.

Poza trzema wyróżnionymi wyżej punktami równowagi zachowanie układu będzie się różniło w zależności od liczby N czyli od liczby elementów kraty. Najprostszy przypadek to sytuacja, w której $N = 1$. Spełniony jest warunek:

$$y_{n+1} = y_{n-1} = 0 \tag{7.8}$$

Po podstawieniu go do równania (7.4) otrzymuje się:

$$\begin{aligned} \dot{p} &= \omega_0^2 \left(e^{-y} - e^{y}\right) + \left(\mu - p^2\right) p \\ \dot{y} &= p \end{aligned} \tag{7.9}$$

Równanie to można zapisać także w postaci:

$$\ddot{y} + 2\omega_0^2 \sinh(y) = \left(\mu - \dot{y}^2\right) \dot{y} \tag{7.10}$$

Zakładając, że dla małych wartości $\sinh(y) = y$ równanie (7.10) przechodzi w postać:

$$\ddot{y} + 2\omega_0^2 y = \left(\mu - \dot{y}^2\right) \dot{y} \tag{7.11}$$

Równanie (7.11) jest oscylatorem Rayleigha (opisanym w podrozdziale 6.2).

W pracach [27] i [26] wykazano, że oprócz dwóch stanów stabilnych charakteryzujących się obrotem pierścienia ze stałą prędkością równanie (7.4) posiada dokładnie $N - 1$ stanów stabilnych, dla których rozwiązaniem jest cykl graniczny. Rozwiązania te charakteryzują się tym, że energia wpompowana do układu przepływa w postaci solitonu przez kolejne elementy pierścienia w prawą lub w lewą stronę.

Najprostszy przypadek, w którym dla równania kraty Toda-Rayleigha pojawiają się rozwiązania cykliczne to układ składający się z dwóch elementów czyli równanie w postaci:

$$\begin{aligned} \ddot{y}_1 &= \omega_0^2 \left(e^{y_2 - y_1} - e^{y_1 - y_2}\right) + \mu \left(1 - \dot{y}_1^2\right) \dot{y}_1 \\ \ddot{y}_2 &= \omega_0^2 \left(e^{y_1 - y_2} - e^{y_2 - y_1}\right) + \mu \left(1 - \dot{y}_2^2\right) \dot{y}_2 \end{aligned} \tag{7.12}$$

Dla równania (7.12) można zaobserwować trzy rozwiązania stabilne: dwa w postaci pierścienia obracającego się ze stałą prędkością w prawo lub w lewo i jedno rozwiązanie cykliczne, dla którego wartości $\dot{y}_1$ i $\dot{y}_2$ oscylują w przeciwfazie.

Dla układu składającego się z sześciu elementów istnieje siedem rozwiązań stabilnych: dwa w postaci pierścienia obracającego się ze stałą prędkością w prawo lub w lewo i pięć rozwiązań cyklicznych. Rozwiązania cykliczne zostały oznaczone numerami

$m = -2, -1, 0, 1, 2$ przy czym poszczególne rozwiązania dodatnie i ujemne na przykład $m = -2$ i $m = 2$ odpowiadają sobie w ten sposób, że kształt cyklu granicznego dla tych rozwiązań jest taki sam natomiast energia przepływa poprzez pierścień w przeciwnym kierunku.

Dla trybu oznaczonego $m = 2$ i $m = -2$ soliton rozchodzący się przez pierścień ma wartość maksymalną w jednym elemencie układu równocześnie. Można to sobie łatwo wyobrazić jako falę przepływającą przez pierścień w prawą lub lewą stronę. Dla trybu oznaczonego $m = 1$ i $m = -1$ istnieją dwie fale rozchodzące się w tym samym kierunku, a ich maksima znajdują się po dokładnie przeciwległych stronach pierścienia. Dla trybu oznaczonego jako $m = 0$ istnieją trzy fale, których maksima są równomiernie rozłożone na obwodzie pierścienia. W trybie $m = 0$ zachodzi ciekawa właściwość, która powoduje, że nie jesteśmy w stanie określić, w którym kierunku rozchodzi się fala w pierścieniu. Każda fala przyjmuje takie same wartości zarówno w swoim maksimum jak i minimum. Wartość y_n w kolejnych elementach pierścienia będzie naprzemiennie rosła i malała co powoduje, że nie można odróżnić początku jednej fali od końca poprzedniej. Tryb ten jest nazywany trybem "optycznym" (ang. optical mode).

Solitony rozchodzą się w sposób analogiczny do opisanego powyżej przypadku w kracie Toda-Rayleigha o innej liczbie elementów. W przypadku gdy układ składa się z nieparzystej liczby elementów tryb optyczny nie istnieje. W przypadku gdy układ składa się z dwóch elementów jest on jedynym rozwiązaniem z cyklem granicznym.

W przeciwieństwie do algorytmu opartego na oscylatorach nieliniowych parametry μ i ω_0 nie mają prostego przełożenia na kształt cyklu i częstość drgań pojawiających się w układzie. Szacowania rozmiaru cyklu granicznego w poszczególnych tryba chodu wyprowadzono w pracy [26]. Poszczególne parametry cyklu opisano wzorami:

$$\begin{aligned} A_y^m &= \frac{\sqrt{\mu}}{\sqrt{3}\sin\dfrac{2\pi m + N}{2N}} \\ A_{\dot{y}}^m &= \frac{2\sqrt{\mu}}{\sqrt{3}} \\ \omega_{st}^m &= 2\omega_0 \sin\frac{2\pi m + N}{2N}\left(1 + \frac{\mu}{12\omega_0^2}\right) \end{aligned} \tag{7.13}$$

gdzie A_y^m oznacza wartość amplitudy cyklu w osi y, $A_{\dot{y}}^m$ to wartość amplitudy cyklu w osi $\dot{y}$, ω_{st}^m oznacza częstość drgań odpowiedzi układu (czas obiegu cyklu granicznego), N to ilość elementów kraty Toda-Rayleigha, a m to numer cyklu.

W pracy [26] wprowadzono pewną modyfikację do równia kraty w postaci:

$$\begin{aligned} &\ddot{y}_n = \omega_0^2\left(e^{y_{n-1}-y_n} - e^{y_n - y_{n+1}}\right) + f\left(\dot{y}_n - p_{ext}\right) - \omega_R^2 y_n \\ &f(x) = (\mu - x^2)x \end{aligned} \tag{7.14}$$

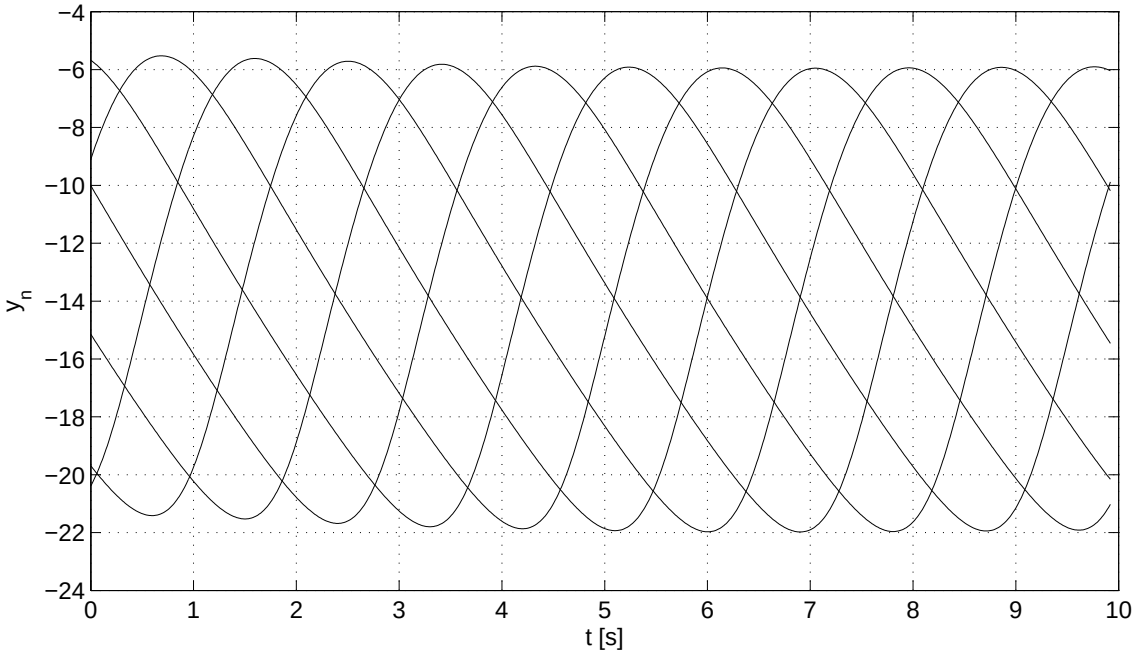
Modyfikacja ma podwójną rolę:

- człon $\omega_R^2 y_n$ pełni rolą stabilizującego, ujemnego sprzężenia zwrotnego zapobiegającego dryftowi rozwiązań cyklicznych, które może pojawić się w trybach $m \neq 0$,
- przesunięcie rozwiązań w przestrzeni stanu o wartość p_{ext} pełni rolę sygnału sterującego, które przesuwając punkt wokół, którego tworzy się cykl graniczny pozwala na wymuszenie przejścia pomiędzy sąsiadującymi rozwiązaniami cyklicznymi.

Na rysunkach 7.6, 7.7 i 7.8 przedstawiono przebiegi zmiennej y_n dla trzech rozwiązań cyklicznych, odpowiednio $m = 2$, $m = 1$ i $m = 0$. Jak łatwo dostrzec przebiegi te pokrywają się z sygnałami sterującymi ruchem nogi przedstawionymi na rysunkach 2.6, 2.11

i 2.14. Kolejne cykle graniczne będące wynikiem przepływu energii w elementach kraty Toda-Rayleigha zapewniają prawidłową synchronizację pracy nóg robota sześcionożnego odpowiednio:

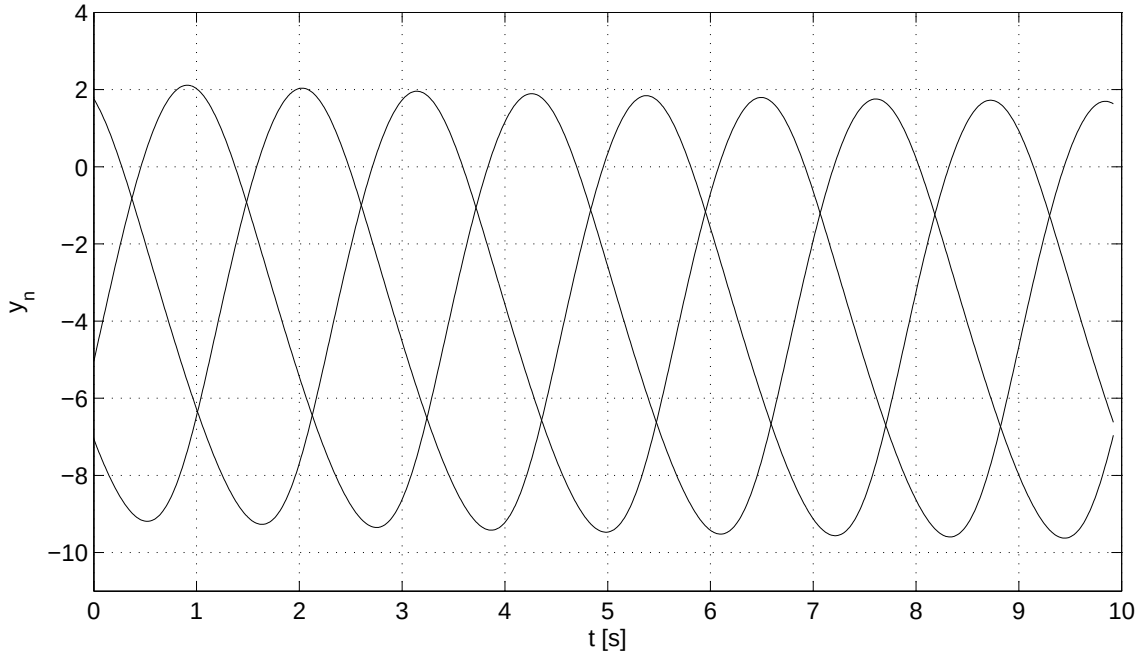
- cykl graniczny $m = 2$ dla chodu metachronicznego,
- cykl graniczny $m = 1$ dla chodu gąsienicowego,
- cykl graniczny $m = 0$ dla chodu trójpodporowego.



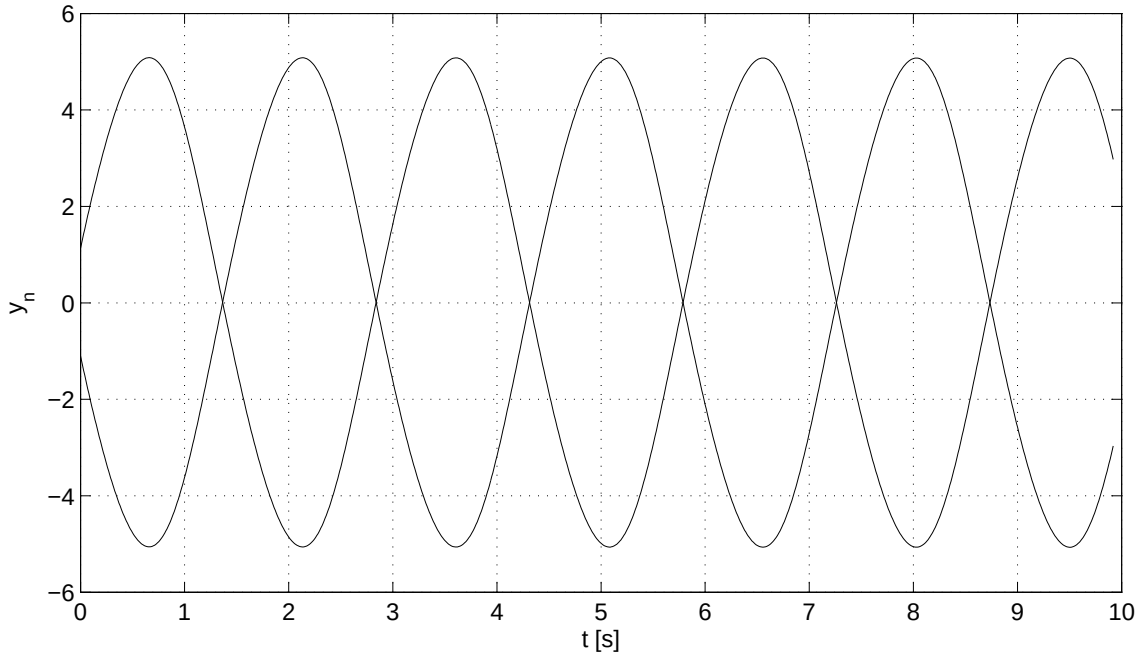
Rysunek 7.6: Wartości sygnałów y_n dla $n = 1..6$ na wyjściu kraty Toda-Rayleigha po osiągnięciu cyklu granicznego oznaczonego jako $m = 2$

Przy zastosowaniu kraty Toda-Raleigha do sterowania chodem sześcionożnego robota kroczącego ruch końcówek nóg odbywa się (analogiczne jak w poprzednich algorytmach) po trajektorii cyklu granicznego w dwuwymiarowej przestrzeni stanu $\dot{y}_n = f(y_n)$ gdzie $n = 1..6$. Kształty cykli granicznych są przedstawione na rysunkach 7.9, 7.10 i 7.11. Należy zauważyć, że pomiędzy środkiem cykli granicznych w poszczególnych trybach chodu występuje przesunięcie (wpływ parametru p_{ext}). Nie uniemożliwia ono zastosowania równania kraty Toda-Rayleigha do sterowania chodem robota, ale zdecydowanie utrudnia dobór parametrów. Wpływ tego przesunięcia można łatwo zredukować odpowiednim doborem pozycji bazowej nóg robota, tak by wszystkie punkty, wszystkich cykli granicznych znajdowały się w przestrzeni roboczej końcówki nóg robota.

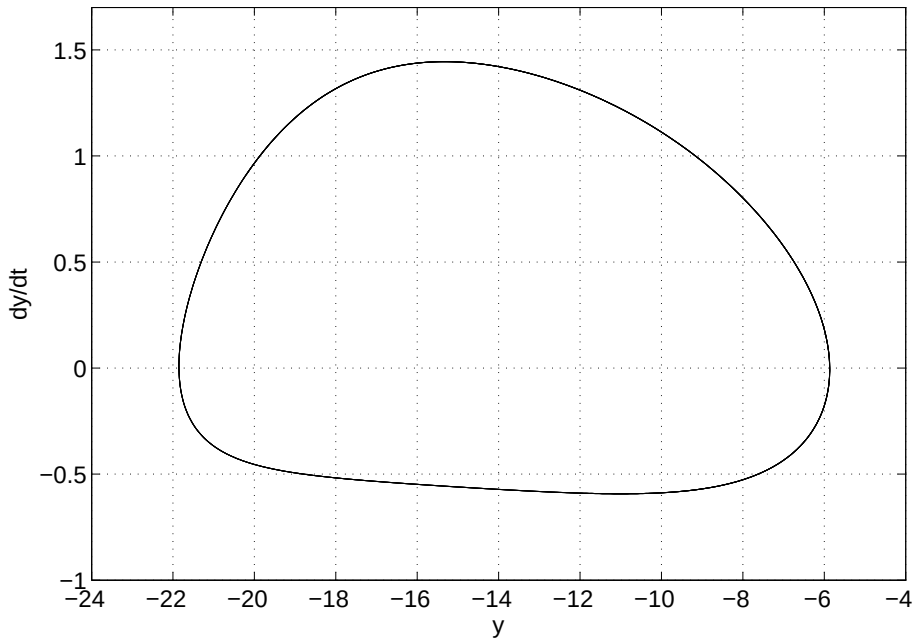
Krata umożliwia także płynne przejścia między cyklami granicznymi czyli rodzajami chodu. Niestety doświadczenia numeryczne wykazały, że dynamika przejść dla prawidłowo dobranych wartości parametrów μ i ω_0 nie nadaje się do sterowania ruchem końcówek nóg robota (patrz rysunek 7.12). Przy małych wartościach parametru μ dynamika przebiegów jest odpowiednia do sterowania chodem w fazie chodu właściwego, ale wydłuża się faza przejścia pomiędzy trybami (cyklami granicznymi). Przy dużych wartościach parametru μ poprawia się czas przejścia pomiędzy trybami, ale kształt cyklu granicznego staje się "ostry" i całe równanie nabiera charakteru układu sztywnego co uniemożliwia zastosowanie go do sterowania w czasie rzeczywistym.



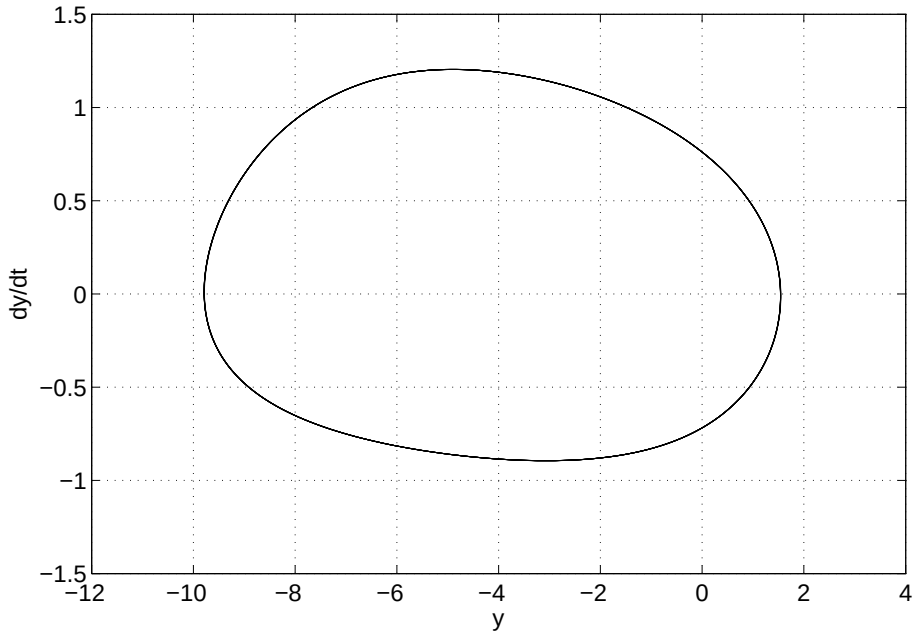
Rysunek 7.7: Wartości sygnałów y_n dla $n = 1..6$ na wyjściu kraty Toda-Rayleigha po osiągnięciu cyklu granicznego oznaczonego jako $m = 1$. Wykresy odpowiadające sygnałom y_1 i y_4, y_2 i y_5 oraz y_3 i y_6 nałożyły się na siebie, ponieważ oscylacje odpowiednich elementów przebiegają tak samo.



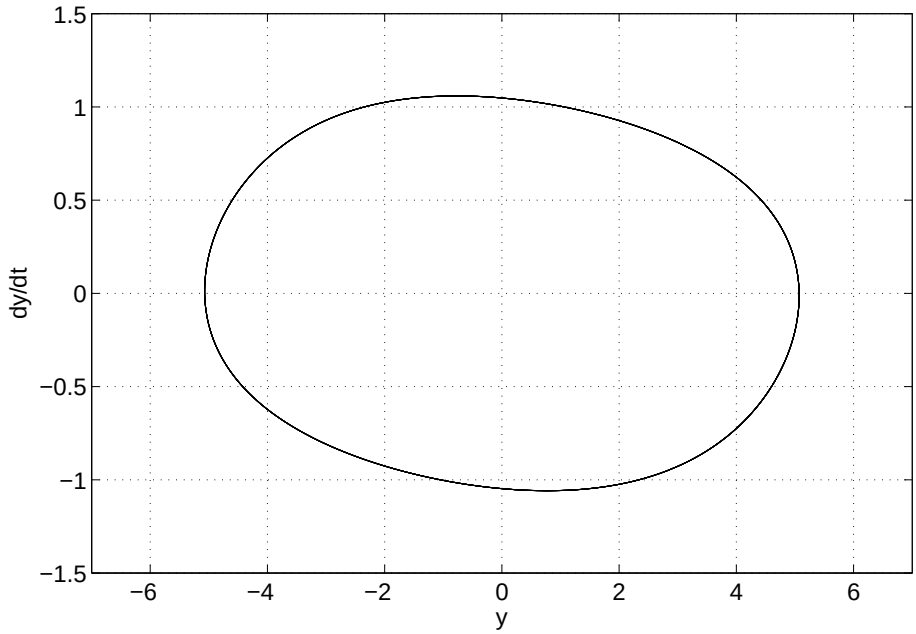
Rysunek 7.8: Wartości sygnałów y_n dla $n = 1..6$ na wyjściu kraty Toda-Rayleigha po osiągnięciu cyklu granicznego oznaczonego jako $m = 0$. Wykresy odpowiadające sygnałom y_1, y_3 i y_5 oraz y_2, y_4 i y_6 nałożyły się na siebie, ponieważ oscylacje odpowiednich elementów przebiegają tak samo.



Rysunek 7.9: Cykl graniczny w przestrzeni stanu jednego z wyjść kraty Toda-Rayleigha w trybie oznaczonym $m = 2$



Rysunek 7.10: Cykl graniczny w przestrzeni stanu jednego z wyjść kraty Toda-Rayleigha w trybie oznaczonym $m = 1$



Rysunek 7.11: Cykl graniczny w przestrzeni stanu jednego z wyjść kraty Toda-Rayleigha w trybie oznaczonym $m = 0$

7.3 Zastosowanie kraty Toda-Rayleigha do budowy generatora chodu

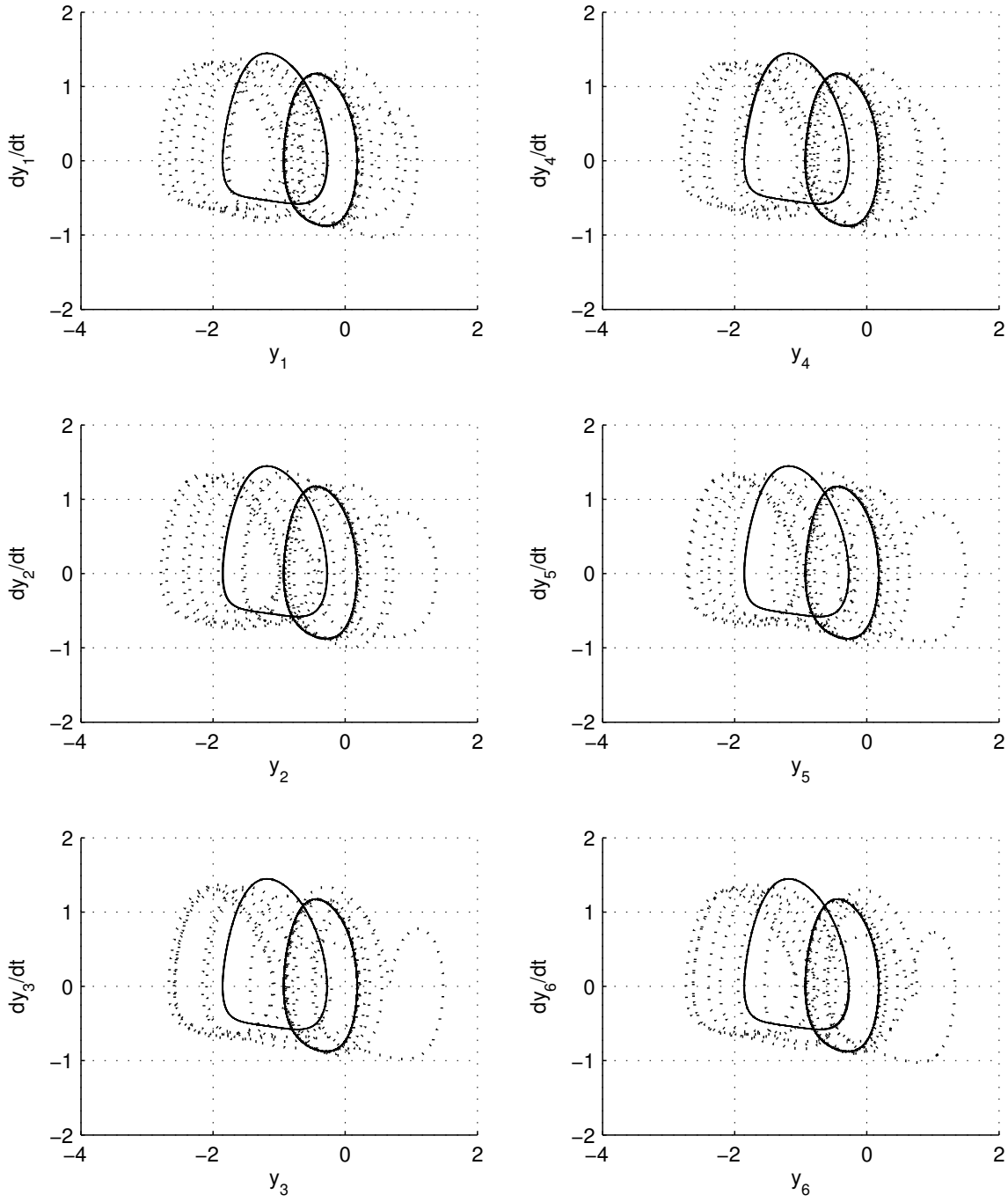
Krata Toda-Rayleigha może zostać wykorzystana do budowy algorytmu generującego trajektorie chodu sześcionożnego robota kroczącego. W przeciwieństwie do oscylatorów omawianych w poprzednich rozdziałach, algorytm oparty na kracie Toda-Rayleigha może posłużyć do generacji chodu:

- metachronicznego,
- gąsienicowego,
- trójpodporowego.

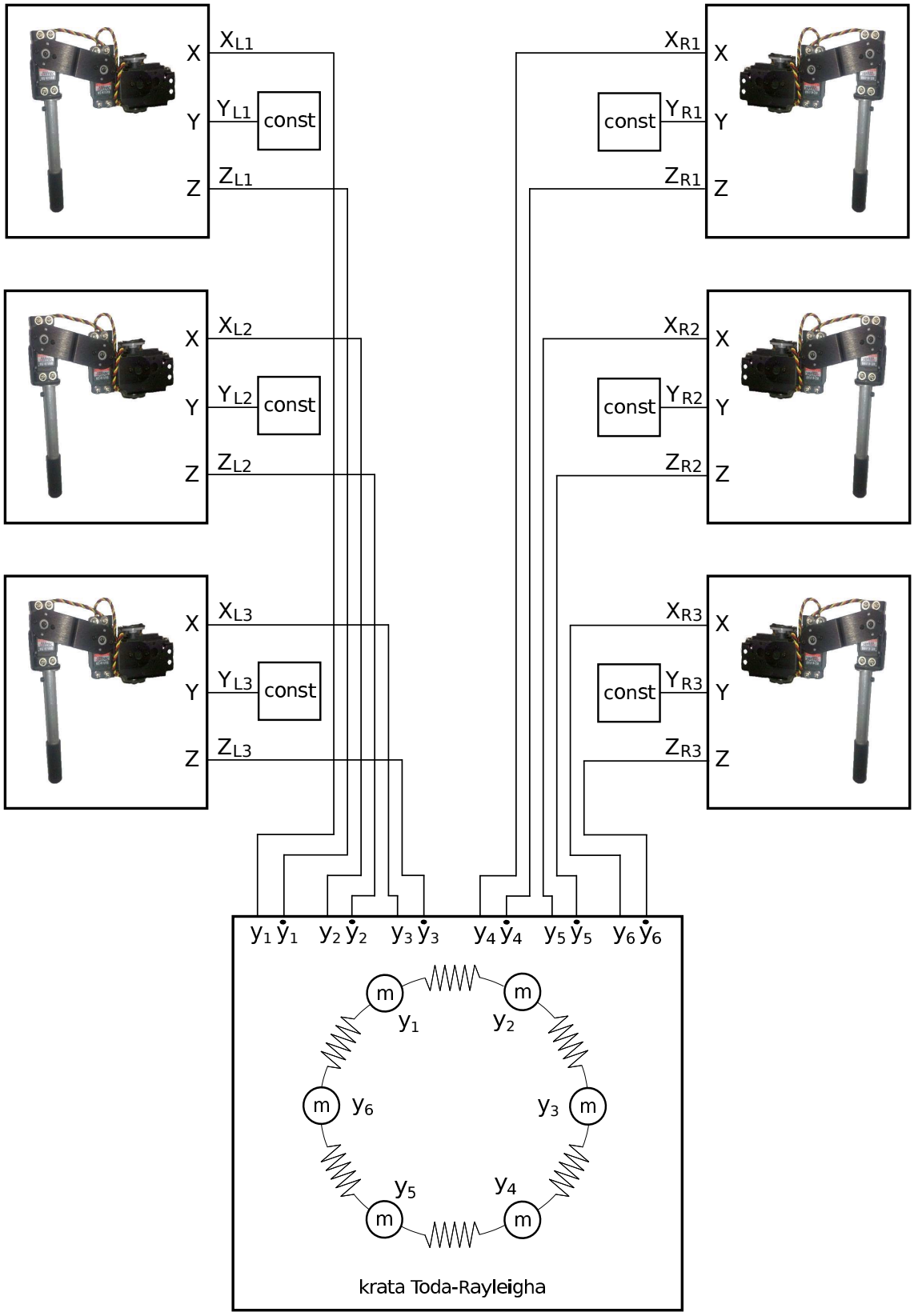
W każdym z wymienionych rodzajów chodów można zrealizować fazy rozruchu, chodu właściwego oraz fazę zatrzymania. Dodatkowo istnieje możliwość realizacji **fazy przejścia** będącej etapem pośrednim pomiędzy dwoma różnymi rodzajami chodu na przykład metachronicznym i trójpodporowym.

Do budowy algorytmu sterowania sześcionożnym robotem kroczącym należy zastosować kratę składającą się z sześciu węzłów połączonych w cykl. Każdy węzeł kraty będzie odpowiedzialny za generację trajektorii ruchu dla jednej nogi robota. Tak jak w przypadku poprzednich algorytmów opartych na oscylatorach liniowych i nieliniowych, sygnał zmiennej stanu y jest odpowiedzialny za generację trajektorii końcówki nogi wzdłuż osi X. Sygnał zmiennej stanu $\dot{y}$ jest odpowiedzialny za generację trajektorii końcówki nogi wzdłuż osi Z. Ruch wzdłuż osi Y (jak w poprzednich przypadkach) nie odbywa się to znaczy końcówka nogi robota utrzymuje stałą pozycję wzdłuż tej osi. Schemat połączeń został przedstawiony na rysunku 7.13.

W **fazie chodu właściwego** za generację każdego z chodów (metachronicznego, gąsienicowego i trójpodporowego) odpowiada inny cykl graniczny pojawiający się w przestrzeni stanu równania kraty. Wprowadzenie układu w odpowiedni cykl graniczny jest realizowane poprzez odpowiednio dobrane warunki początkowe układu. W ten sposób jest



Rysunek 7.12: Przykładowe przejście układu kraty Toda-Rayleigha z cyklu granicznego $m = 2$ do $m = 1$. Początkowy i docelowy cykl graniczny oznaczono linią ciągła, trajektorię przejścia oznaczono linią przerywaną.



Rysunek 7.13: Schemat ideowy systemu sterującego chodem robota sześcionożnego z wykorzystaniem kraty Toda-Rayleigha

realizowana **faza rozruchu** dla poszczególnych rodzajów chodu. Należy zapewnić, by skokowe wartości wymuszenia pozycji końcówki nogi robota z pozycji bazowej przyjmowały wartości niezerowe tylko dla osi Z. Oznacza to, że przyjmując pozycję zadaną przez warunki początkowe końcówka nogi robota przesuwa się jedynie w kierunku góra-dół. Zadanie takich wartości wzdłuż osi X (przód-tył) powodowałby przesuwanie końcówek nóg wzdłuż podłoża przeciwdziałając sile tarcia końcówki o podłoże. Byłoby to niekorzystne z dwóch powodów: powodowałby niepotrzebną stratę energii i generowałoby nienaturalne zachowanie nogi. Biorąc pod uwagę powyższy wniosek i schemat połączeń kraty Toda-Rayleigha do poszczególnych sygnałów sterujących nogi można zapisać, że:

$$\begin{aligned} y_n^0 &= 0 \\ \dot{y}_n^0 &= D_n \end{aligned} \tag{7.15}$$

dla $n = 1..6$.

Stałe D_n należy dobrać eksperymentalnie, a następnie zweryfikować za pomocą symulacji. Wszystkie wartości znajdujące się dostatecznie blisko cyklu granicznego, który chce się osiągnąć mogą pełnić rolę D_n. Kluczowe wydaję się więc wybór takich wartości, które pozwolą na osiągnięcie cyklu granicznego możliwie szybko i zarazem łagodnie.

Faza zatrzymania została zrealizowana tak samo jak w przypadku algorytmu sterowania z wykorzystaniem oscylatorów liniowych i oscylatorów nieliniowych. Dynamika oscylatorów na czas fazy zatrzymania jest zamieniana na równania oscylatorów harmonicznych tłumionych co powoduje wytłumienie oscylacji do zera, czyli płynne przesunięcie końcówek nóg robota sześcionożnego do pozycji bazowej. Oznacza to, że równanie dynamiki zastosowane w algorytmie przyjmuje następującą postać:

$$\ddot{y}_n = \begin{cases} \omega_0^2 \left(e^{y_{n-1}-y_n} - e^{y_n-y_{n+1}}\right) + f\left(\dot{y}_n - p_{ext}\right) - \omega_R^2 y_n & , t < t_1 \\ -\mu \dot{y}_n - \omega_0^2 y_n & , t >= t_1 \end{cases} \tag{7.16}$$

gdzie

$$f(x) = (\mu - x^2)x, \tag{7.17}$$

a t_1 jest chwilą czasu, w której rozpoczyna się faza zatrzymania. Największą zaletą tej metody jest możliwość rozpoczęcia fazy zatrzymania w każdej chwili czasu t_1 bez względu na to jaka jest aktualnie wartość zmiennych stanu kraty Toda-Rayleigha.

Realizacja **fazy przejścia** pomiędzy dowolnymi dwoma rodzajami chodu z trzech stanowi kolejny element algorytmu sterującego. Istnieje możliwość takiego doboru parametrów równania kraty Toda-Rayleigha, aby układ samoczynnie przechodził z jednego cyklu granicznego w inny po zmianie parametru p_{ext}. Jednakże dynamika takiego przejścia pozostawia wiele do życzenia. Do usprawnienia procedury można wykorzystać mechanizm zastosowany do realizacji fazy zatrzymania to znaczy czasowe przełączenie dynamiki kraty Toda-Rayleigha na równania oscylatorów harmonicznych tłumionych.

Do implementacji fazy przejścia wykorzystano możliwość przesunięcia punktu równowagi na płaszczyźnie stanu jak, dla układu oscylatora harmonicznego z wymuszeniem (patrz rozdział 5.3). Punkt równowagi można przesunąć wzdłuż osi zmiennej stanu y_n odpowiadającej za ruch końcówek nóg robota wzdłuż osi X. Proces przejścia z dowolnego trybu chodu oznaczonego jako A na inny dowolny tryb chodu oznaczony jako B składa się z następujących faz:

- aktywna jest dynamika kraty Toda-Rayleigha, rozwiązanie układu znajduje się w cyklu granicznym odpowiadającym za tryb chodu A,
- zapada decyzja o zmianie trybu chodu z chodu A na chód B,

- wybrany jest punkt równowagi na płaszczyźnie stanu części równania realizującej dynamikę oscylatora harmonicznego tłumionego poprzez wprowadzenie stałego wymuszenia o wartościach C_n, gdzie $n = 1..6$,
- układ jest przełączony na dynamikę oscylatora harmonicznego z wymuszeniem,
- jak tylko wartości zmiennych stanu y_n osiągną punkty docelowe czyli C_n dynamika układu z powrotem jest przełączna na kratę Toda-Rayleigha,
- ze względu na odpowiednio dobrane wartości stałych C_n rozpoczyna się faza rozruchu dla trybu chodu B,
- po zakończeniu fazy rozruchu chodu B układ samoczynnie przechodzi do fazy chodu właściwego.

Podczas realizacji fazy rozruchu dobór stałych D_n ma istotne znaczenie. Do prawidłowej realizacji fazy przejścia istotny jest odpowiedni dobór stałych C_n. Należy podkreślić, że o ile podczas fazy rozruchu dochodzi do "startu" kraty Toda-Rayleigha z punktu początkowego, w którym niezerowe są wartości zmiennych $\dot{y}_n^0$ (wzdłuż osi Z) to w przypadku zmiany trybu chodu dochodzi do startu kraty Toda-Rayleigha z punktu początkowego, w którym niezerowe są wartości zmiennych y_n^0 (wzdłuż osi X). Podczas doboru wartości stałych C_n (jeden zestaw dla każdego trybu chodu) należy kierować się tymi samymi wytycznymi co w przypadku doboru stałych D_n. Cały algorytm daje efekt płynnego przejścia z jednego trybu chodu w inny. Po uwzględnieniu wymuszenia dla oscylatora harmonicznego tłumionego ostateczne równanie dynamiki algorytmu sterującego przyjmuje postać:

$$\ddot{y}_n = \begin{cases} \omega_0^2 \left(e^{y_{n-1}-y_n} - e^{y_n-y_{n+1}}\right) + f\left(\dot{y}_n - p_{ext}\right) - \omega_R^2 y_n & ,t < t_1 \\ -\mu \dot{y}_n - \omega_0^2 y_n - C_n & ,t >= t_1 \end{cases} \tag{7.18}$$

gdzie

$$f(x) = (\mu - x^2)x, \tag{7.19}$$

a t_1 jest chwilą czasu, w której rozpoczyna się faza zatrzymania lub zmiany trybu chodu. Do uzyskania fazy zatrzymania wystarczy ustawić stałe C_n na 0.

Implementacja całości algorytmu wymaga nie tylko wdrożenia dyskretnej wersji równania (7.18), ale także dosyć złożonej maszyny skończenie stanowej pozwalającej na przełączanie pomiędzy fazami i trybami chodu w taki sposób by sygnały wejściowe tego algorytmu sprowadzić do dwóch niosących proste informacje:

- start/stop – czy robot ma się przemieszczać,
- tryb chodu – który z trybów chodu powinien być aktywny.

Przykładowa implementacja takiej maszyny skończenie stanowej została zrealizowana przy pomocy programów Simulink i Stateflow. Można ją znaleźć w dodatku A.

7.4 Stabilność algorytmu opartego na kracie Toda-Rayleigha

W rozdziale 7.3 przedstawiono algorytm generujący trajektorie ruchu dla końcówek nóg sześcionożnego robota kroczącego oparty na zmodyfikowanym równaniu kraty Toda-Rayleigha (równanie (7.14)). W algorytmie wykorzystano także równanie dynamiki oscylatora harmonicznego tłumionego ze stałym wymuszeniem (równanie (5.9)).

Algorytm sterowania chodem robota polega na regułowym przełączaniu pomiędzy tym dwoma równaniami (równanie (7.18)) i umiejętnym manipulowaniu wartościami początkowymi równania kraty Toda-Rayleigha (stałe C_n i D_n) i stałym wymuszeniem równania oscylatora harmonicznego tłumionego z wymuszeniem (stałe C_n).

Równanie oscylatora harmonicznego tłumionego ze stałym wymuszeniem jest równaniem globalnie asymptotycznie stabilnym. Zmodyfikowane równanie kraty Toda-Rayleigha niestety nie posiada takiej właściwości. Eksperymenty symulacyjne wykazały obecność cykli granicznych w przestrzeni stanu równania kraty. Ważne jest, by punkty początkowe C_n i D_n były tak dobrane, aby znajdowały się w obszarze przyciągania poszczególnych cykli granicznych. Powoduje to konieczność wykazania istnienia takiego obszaru.

Hipoteza o istnieniu obszaru przyciągania cyklu granicznego w przestrzeni fazowej zmodyfikowanego równania kraty Toda-Rayleigha. *Dla każdego z $N-1$ cykli granicznych N-elementowej kraty Toda-Rayleigha opisanej równaniem:*

$$\ddot{y}_n = \omega_0^2 \left(e^{y_{n-1}-y_n} - e^{y_n-y_{n+1}}\right) + f\left(\dot{y}_n - p_{ext}\right) - \omega_R^2 y_n$$
$$f(x) = (\mu - x^2)x$$

istnieje ograniczony zbiór punktów Γ_m z niepustym wnętrzem taki, że $\forall g_m \in \Gamma_m$ trajektoria wychodząca z punktu g_m dąży do m-tego cyklu granicznego równania.

Istnienie takiego obszaru przyciągania pozwoli na zdefiniowanie następującego twierdzenia:

Hipoteza o ograniczoności trajektorii generowanej przez algorytm oparty na kracie Toda-Rayleigha. *System dynamiczny z regułowym przełączaniem (w sposób opisany w rozdziale 7.3 niniejszej pracy) pomiędzy dwoma równaniami:*

$$\ddot{y}_n = \omega_0^2 \left(e^{y_{n-1}-y_n} - e^{y_n-y_{n+1}}\right) + f\left(\dot{y}_n - p_{ext}\right) - \omega_R^2 y_n$$
$$f(x) = (\mu - x^2)x$$

a równaniem

$$\ddot{y}_n = -\mu\dot{y}_n - \omega^2 y_n - C_n$$

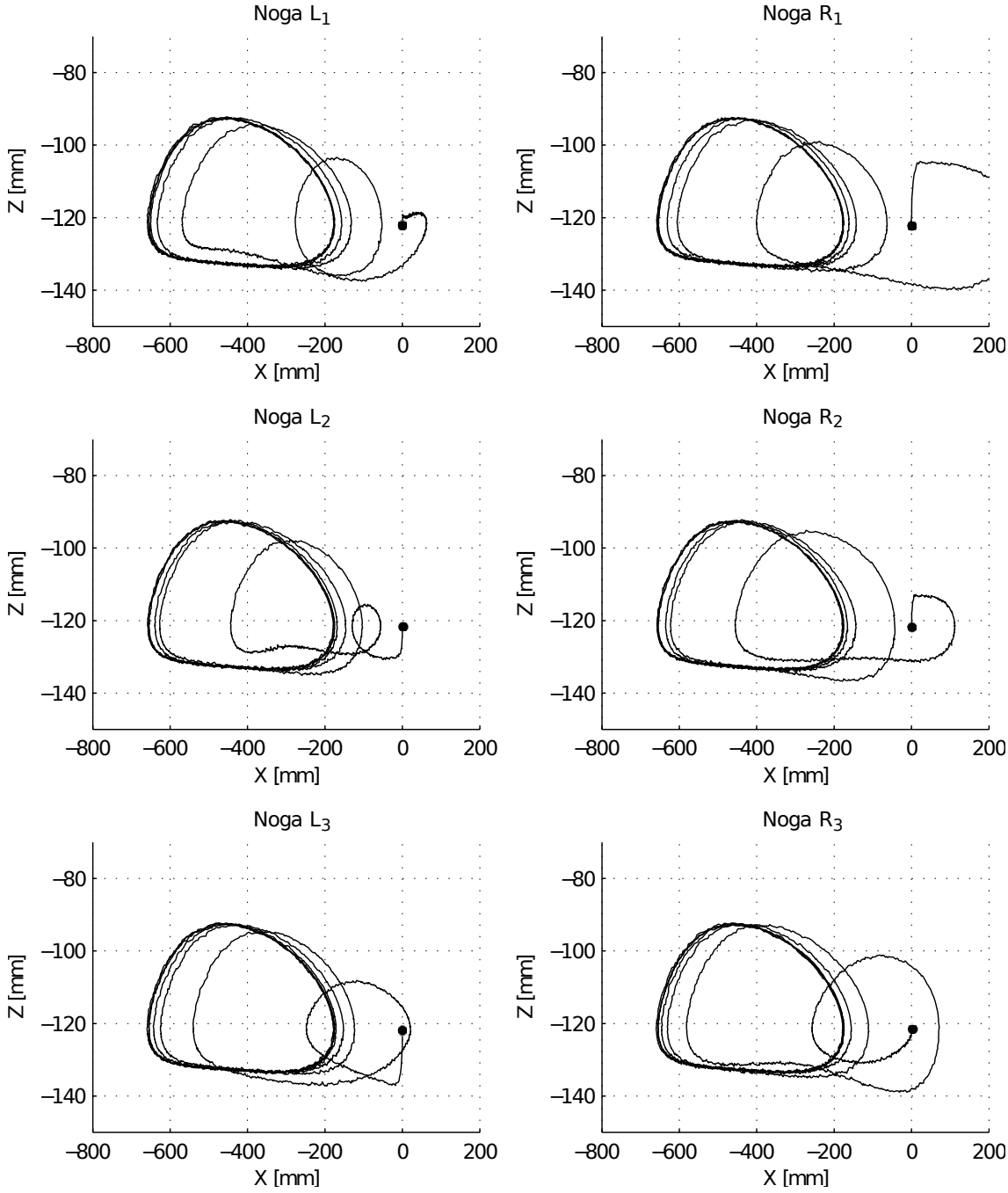
gdzie $n = 1..N$, przy założeniu, że punkty początkowe C_n i D_n, dla których aktywowane jest równanie kraty Toda-Rayleigha należą do zbioru Γ_m będącego obszarem przyciągania m-tego cyklu granicznego równania kraty, generuje trajektorie ograniczoną.

Dowód obydwu hipotez pozostaje sprawą otwartą.

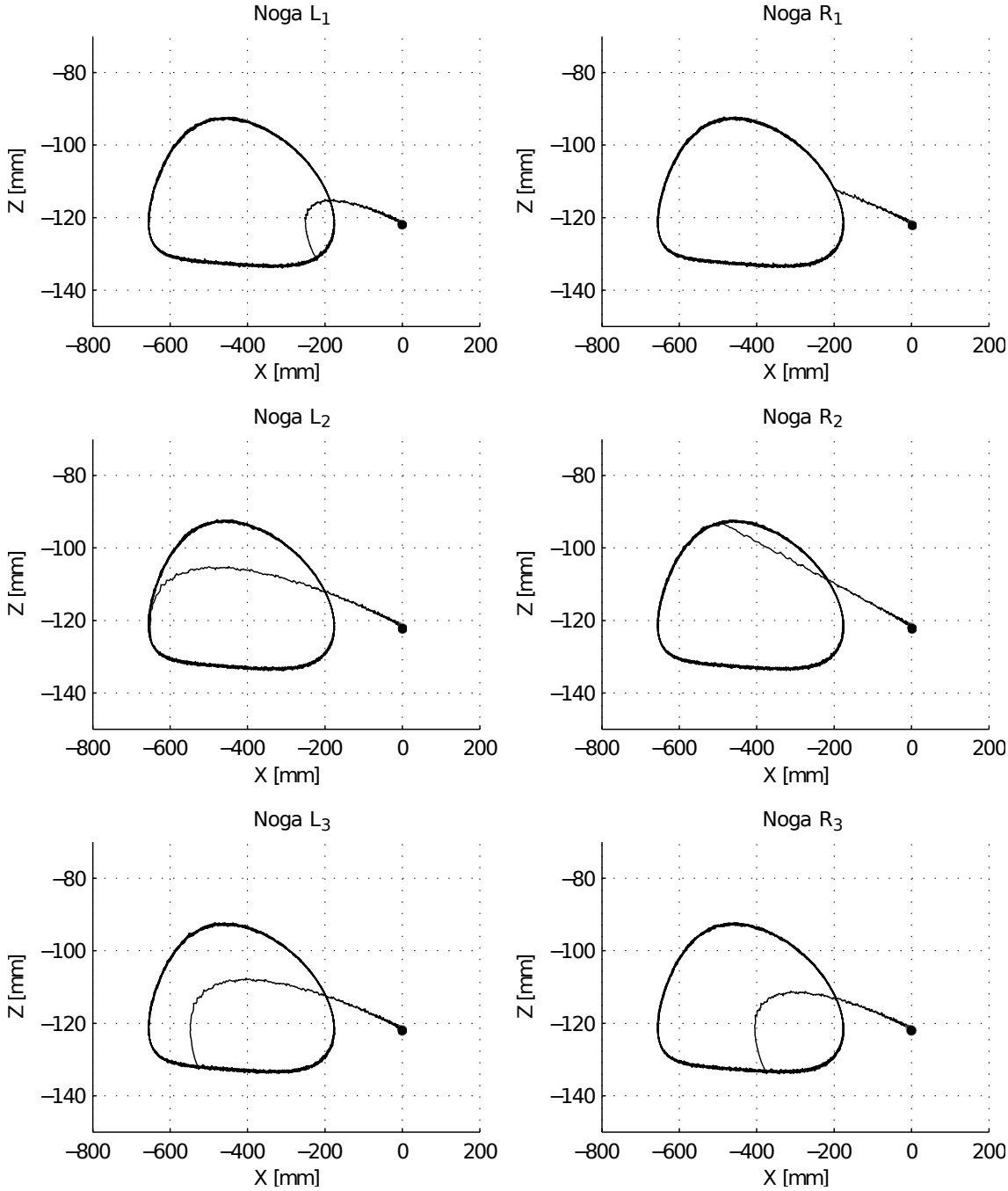
7.5 Wyniki symulacji układu sterowania opartego na kracie Toda-Rayleigha

Analiza wyników symulacji modelu robota kroczącego sterowanego algorytmem opartym na kracie Toda-Rayleigha jest trudniejsza niż w przypadku poprzednich algorytmów, ponieważ każda noga jest sterowana innym sygnałem sterującym. Z tego właśnie powodu zdecydowano o przedstawieniu wyników symulacji w postaci wykresów śladu pozostawianego przez końcówki wszystkich sześciu nóg modelu robota podczas chodu.

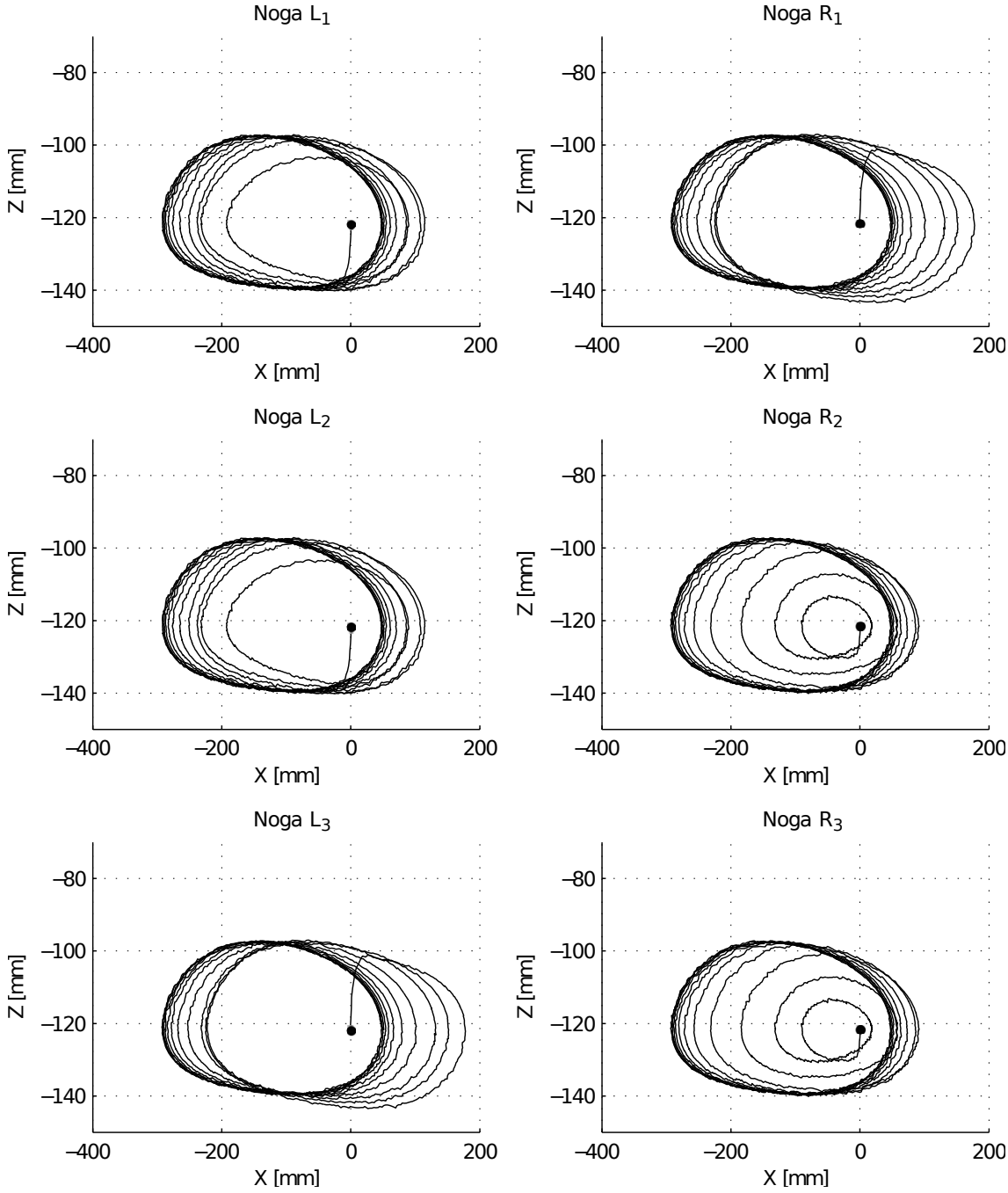
Na rysunkach 7.14 i 7.15 przedstawiono proces rozpoczęcia i zakończenia chodu w trybie $m = 2$. Zarówno rozruch jak i powrót do pozycji bazowej jest płynny i spełnia oczekiwania stawiane algorytmowi. Na rysunkach 7.16, 7.17, 7.18 i 7.19 przedstawiono proces rozruchu i zatrzymania w trybach $m = 1$ i $m = 0$. Symulacja wykazała, że tryb $m = 1$ charakteryzuje się stosunkowo wolnym zbieganiem trajektorii do cyklu granicznego co jest wyraźnie widoczne podczas rozruchu z pozycji bazowej (patrz 7.16). Nie zakłóca to jednak procesu chodzenia ponieważ odpowiedni kształt i synchronizacja trajektorii końcówek nóg jest zachowana. W trybie $m = 3$ końcówki nóg zachowują się prawidłowo, zgodnie z oczekiwaniami. Zarówno proces rozruchu i zatrzymania jest płynny i szybki.



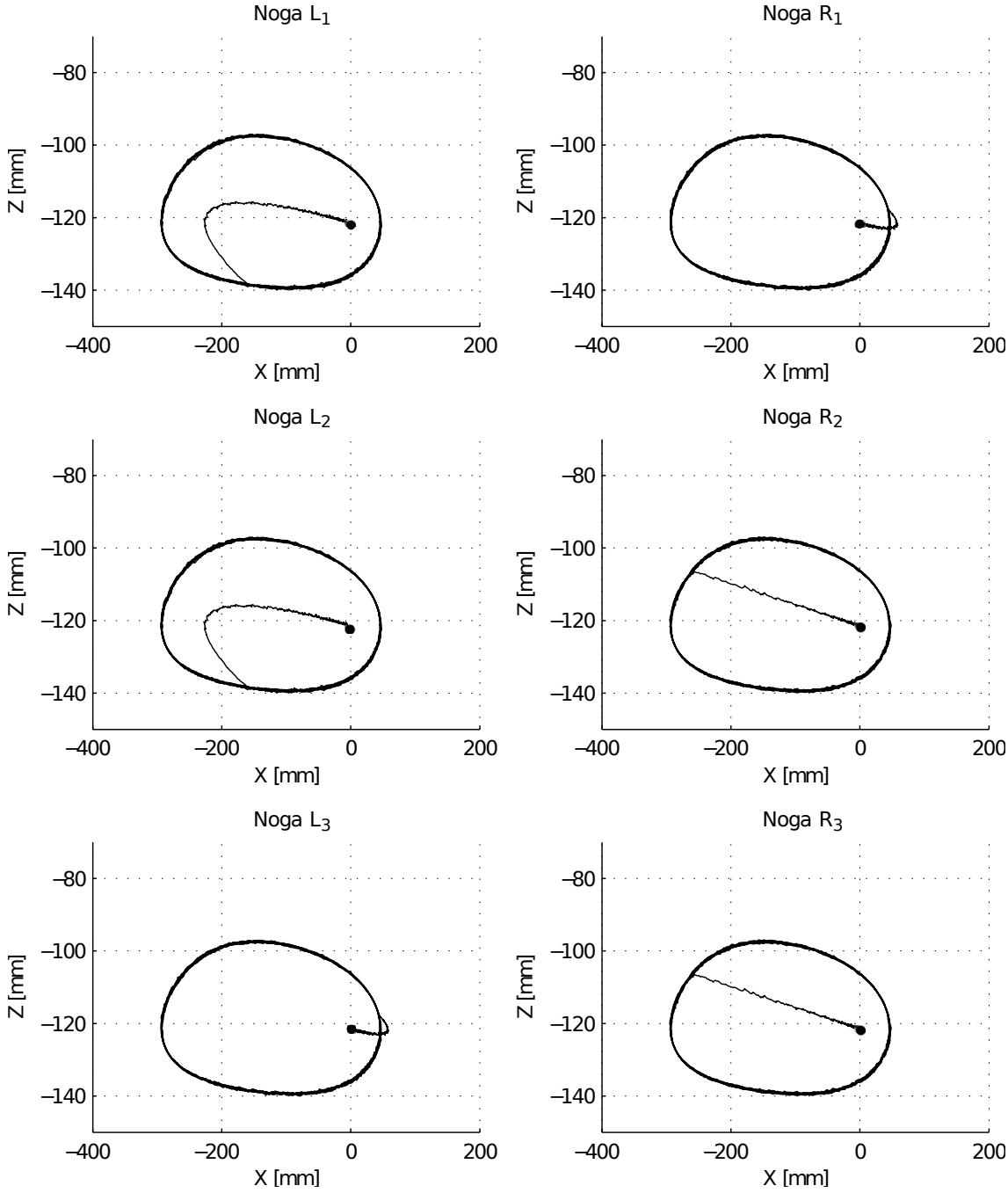
Rysunek 7.14: Ślad położenia końcówek sześciu nóg robota w płaszczyźnie XZ od rozpoczęcia chodu do osiągnięcia cyklu granicznego – wyniki symulacji modelu robota sterowanego algorytmem opartym na kracie Toda-Rayleigha w trybie chodu $m = 2$ (pozycję bazową oznaczono kropką)



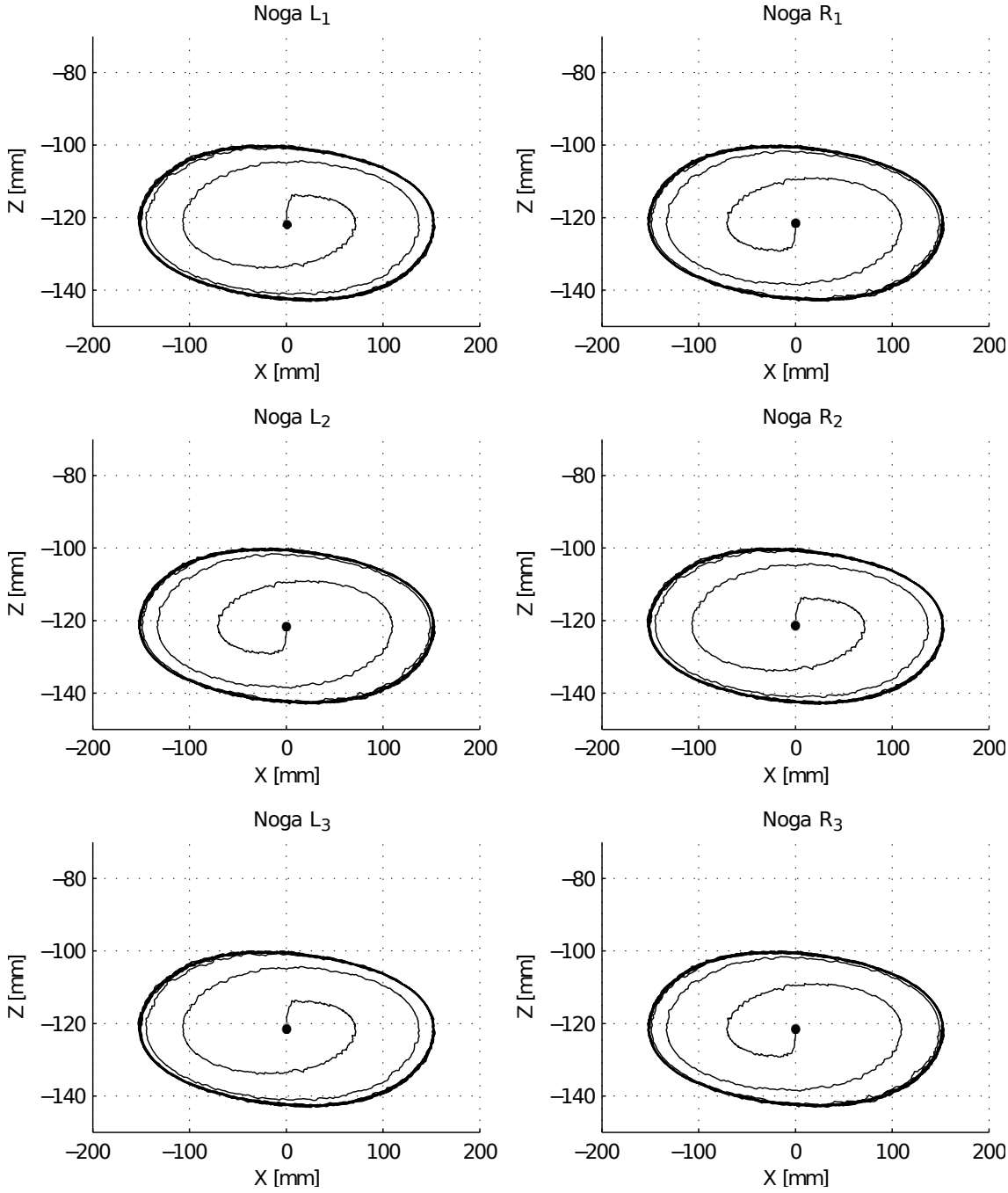
Rysunek 7.15: Ślad położenia końcówek sześciu nóg robota w płaszczyźnie XZ od cyklu granicznego do zatrzymania robota – wyniki symulacji modelu robota sterowanego algorytmem opartym na kracie Toda-Rayleigha w trybie chodu $m = 2$ (pozycję bazową oznaczono kropką)



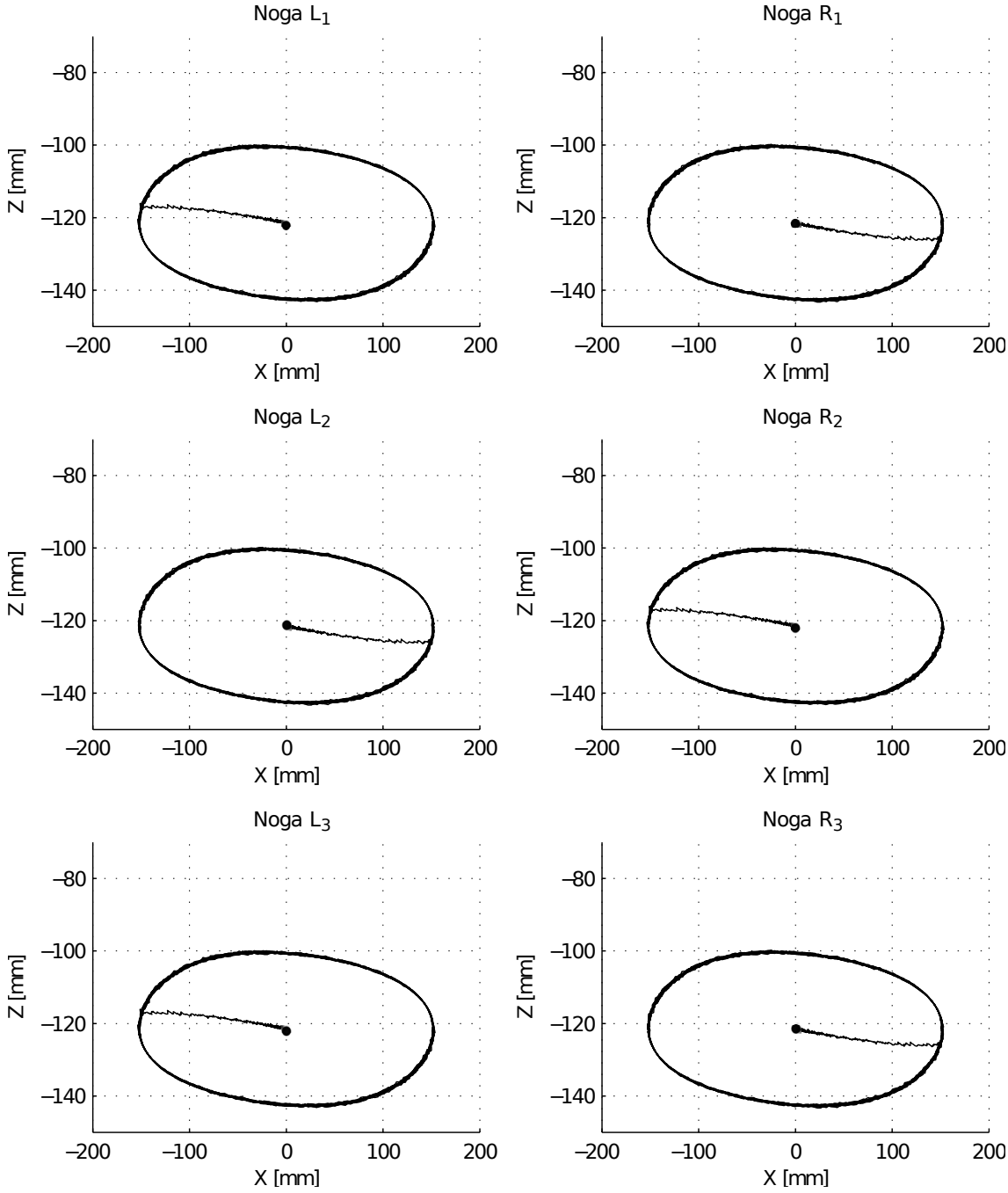
Rysunek 7.16: Ślad położenia końcówek sześciu nóg robota w płaszczyźnie XZ od rozpoczęcia chodu do osiągnięcia cyklu granicznego – wyniki symulacji modelu robota sterowanego algorytmem opartym na kracie Toda-Rayleigha w trybie chodu $m = 1$ (pozycję bazową oznaczono kropką)



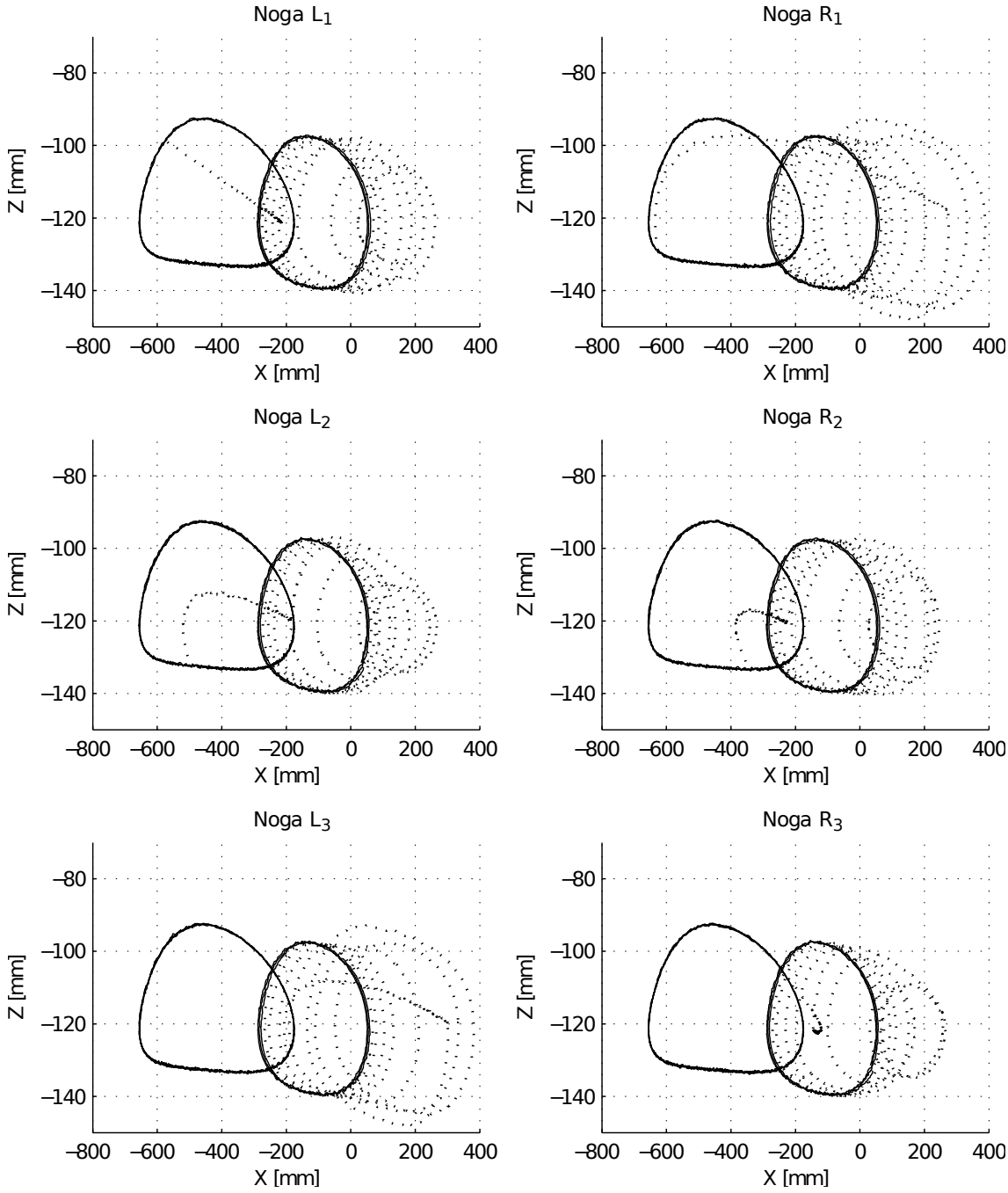
Rysunek 7.17: Ślad położenia końcówek sześciu nóg robota w płaszczyźnie XZ od cyklu granicznego do zatrzymania robota – wyniki symulacji modelu robota sterowanego algorytmem opartym na kracie Toda-Rayleigha w trybie chodu $m = 1$ (pozycję bazową oznaczono kropką)



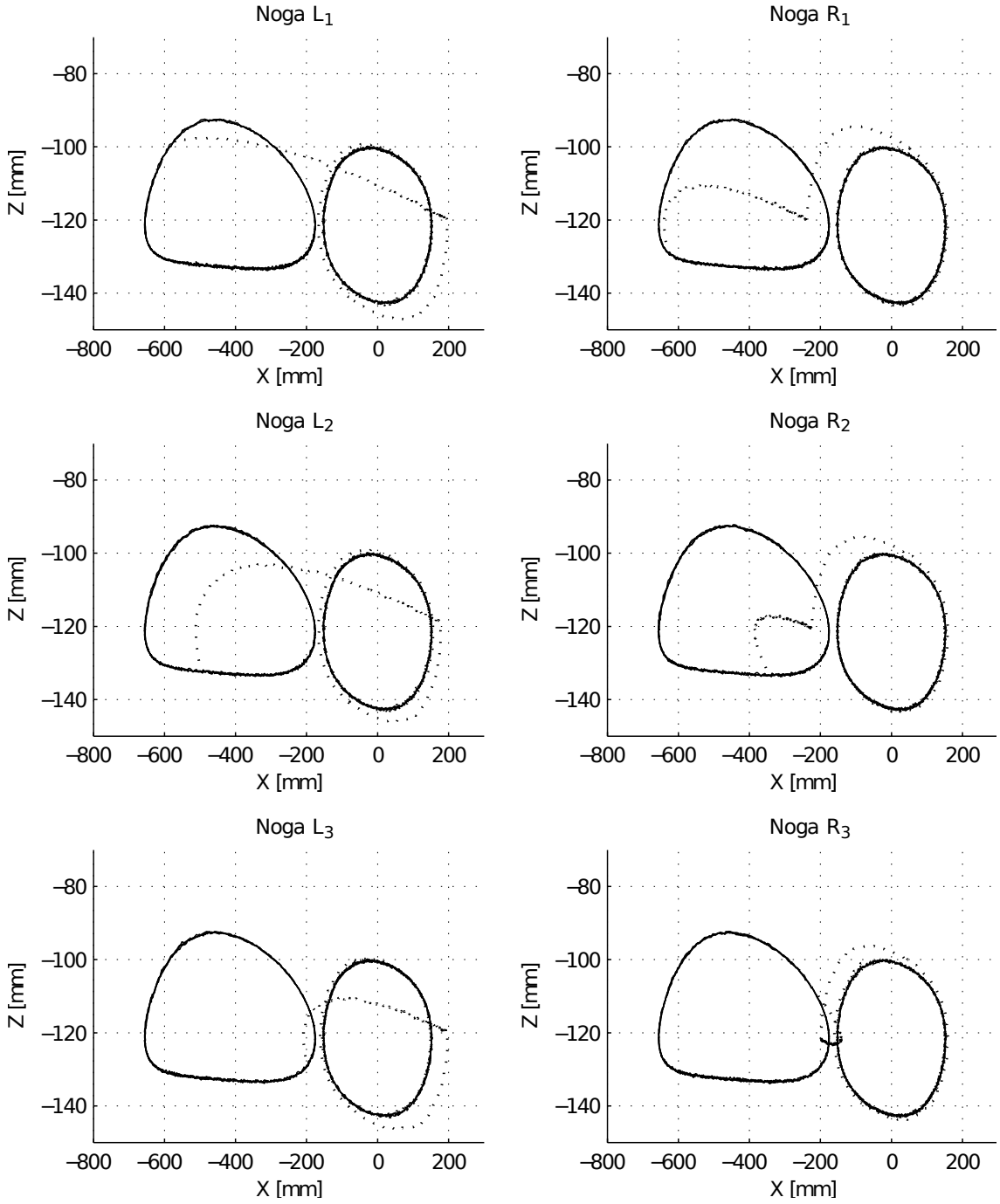
Rysunek 7.18: Ślad położenia końcówek sześciu nóg robota w płaszczyźnie XZ od rozpoczęcia chodu do osiągnięcia cyklu granicznego – wyniki symulacji modelu robota sterowanego algorytmem opartym na kracie Toda-Rayleigha w trybie chodu $m = 0$ (pozycję bazową oznaczono kropką)



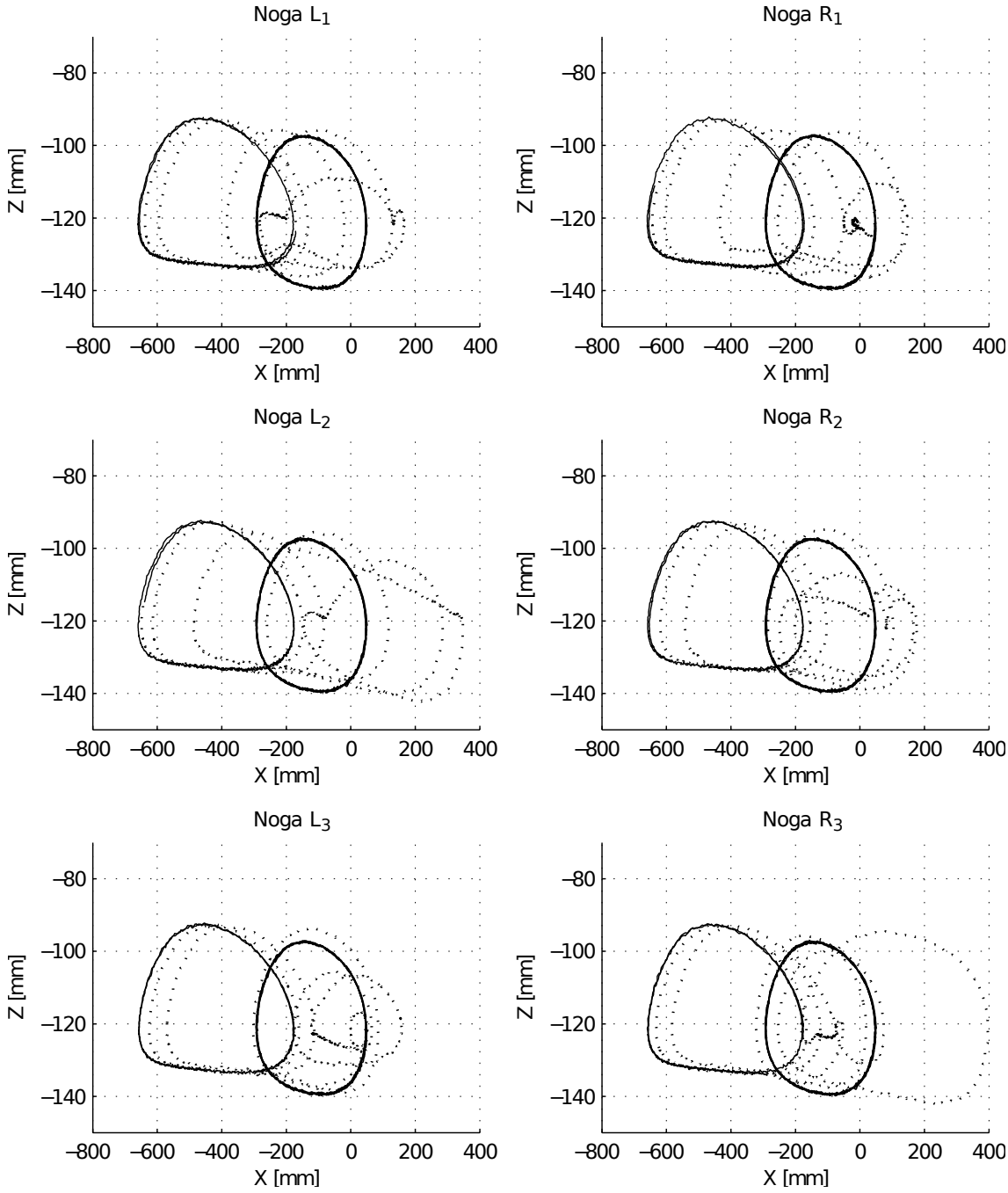
Rysunek 7.19: Ślad położenia końcówek sześciu nóg robota w płaszczyźnie XZ od cyklu granicznego do zatrzymania robota – wyniki symulacji modelu robota sterowanego algorytmem opartym na kracie Toda-Rayleigha w trybie chodu $m = 0$ (pozycję bazową oznaczono kropką)



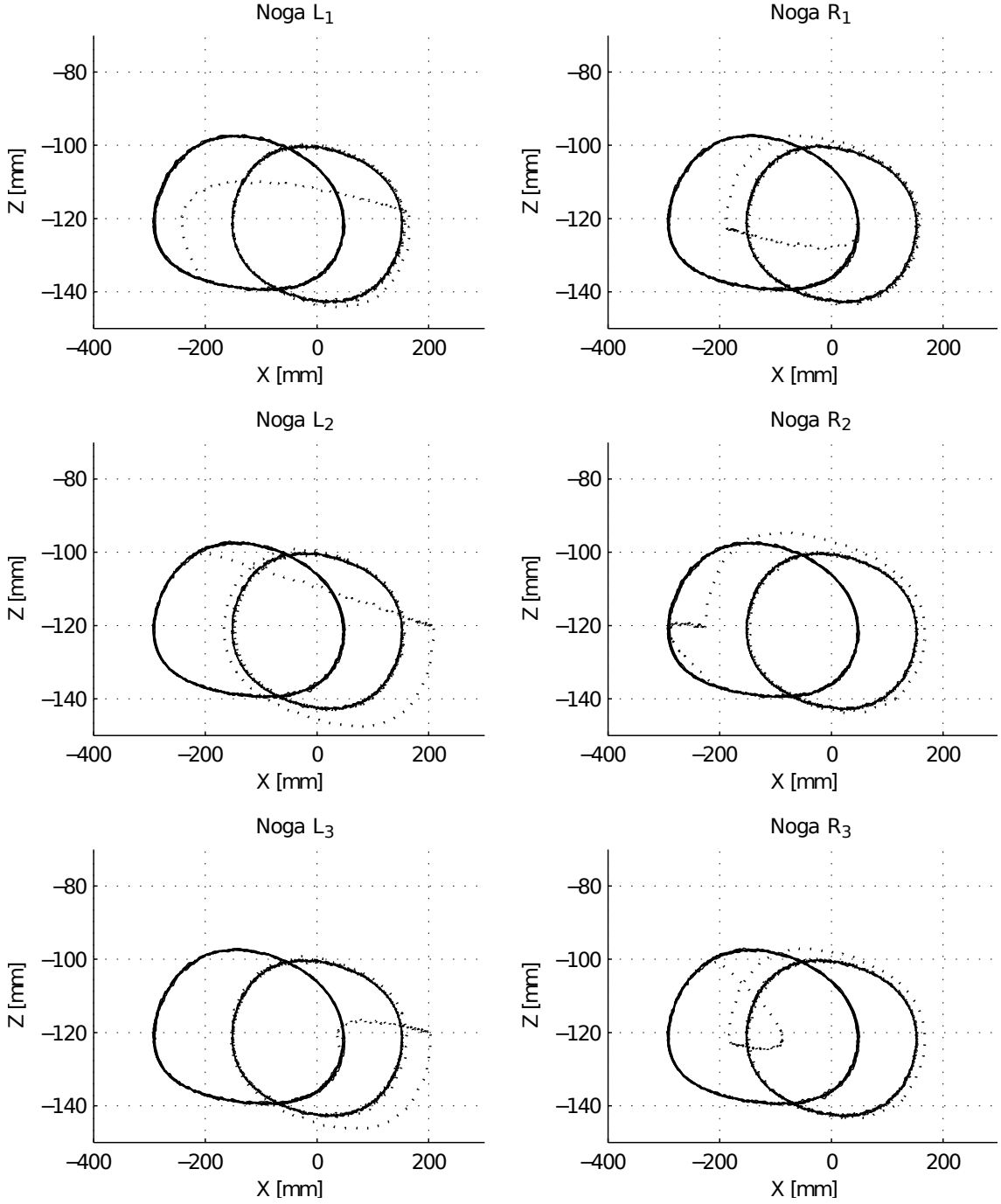
Rysunek 7.20: Ślad położenia w płaszczyźnie XZ końcówek sześciu nóg robota sterowanego algorytmem opartym na kracie Toda-Rayleigha podczas zmiany trybu chodu z trybu $m = 2$ do trybu $m = 1$ – wyniki symulacji modelu robota (cykle graniczne oznaczono linią ciągła, trajektorię przejścia oznaczono linią kropkowaną)



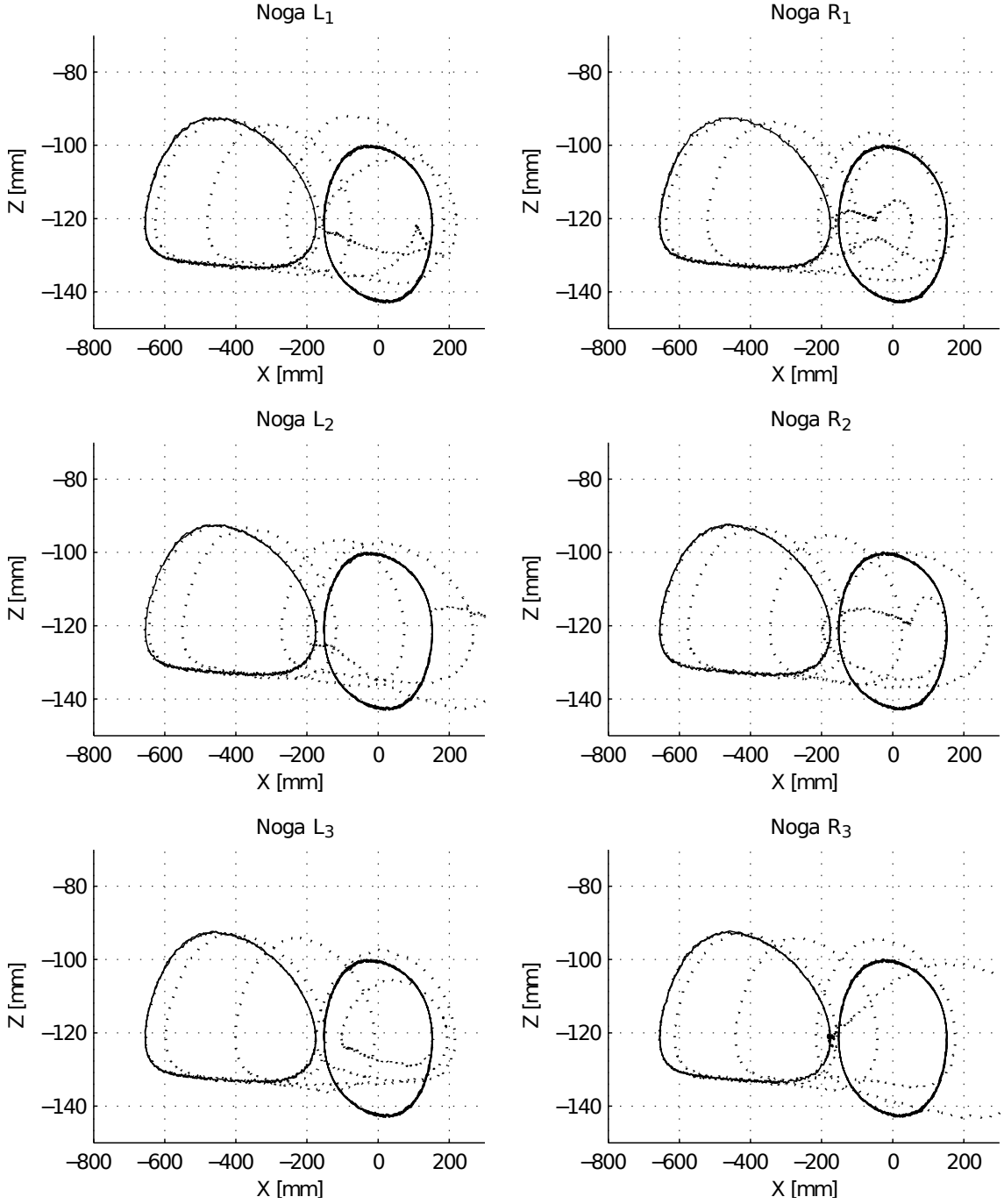
Rysunek 7.21: Ślad położenia w płaszczyźnie XZ końcówek sześciu nóg robota sterowanego algorytmem opartym na kracie Toda-Rayleigha podczas zmiany trybu chodu z trybu $m = 2$ do trybu $m = 0$ – wyniki symulacji modelu robota (cykle graniczne oznaczono linią ciągła, trajektorię przejścia oznaczono linią kropkowaną)



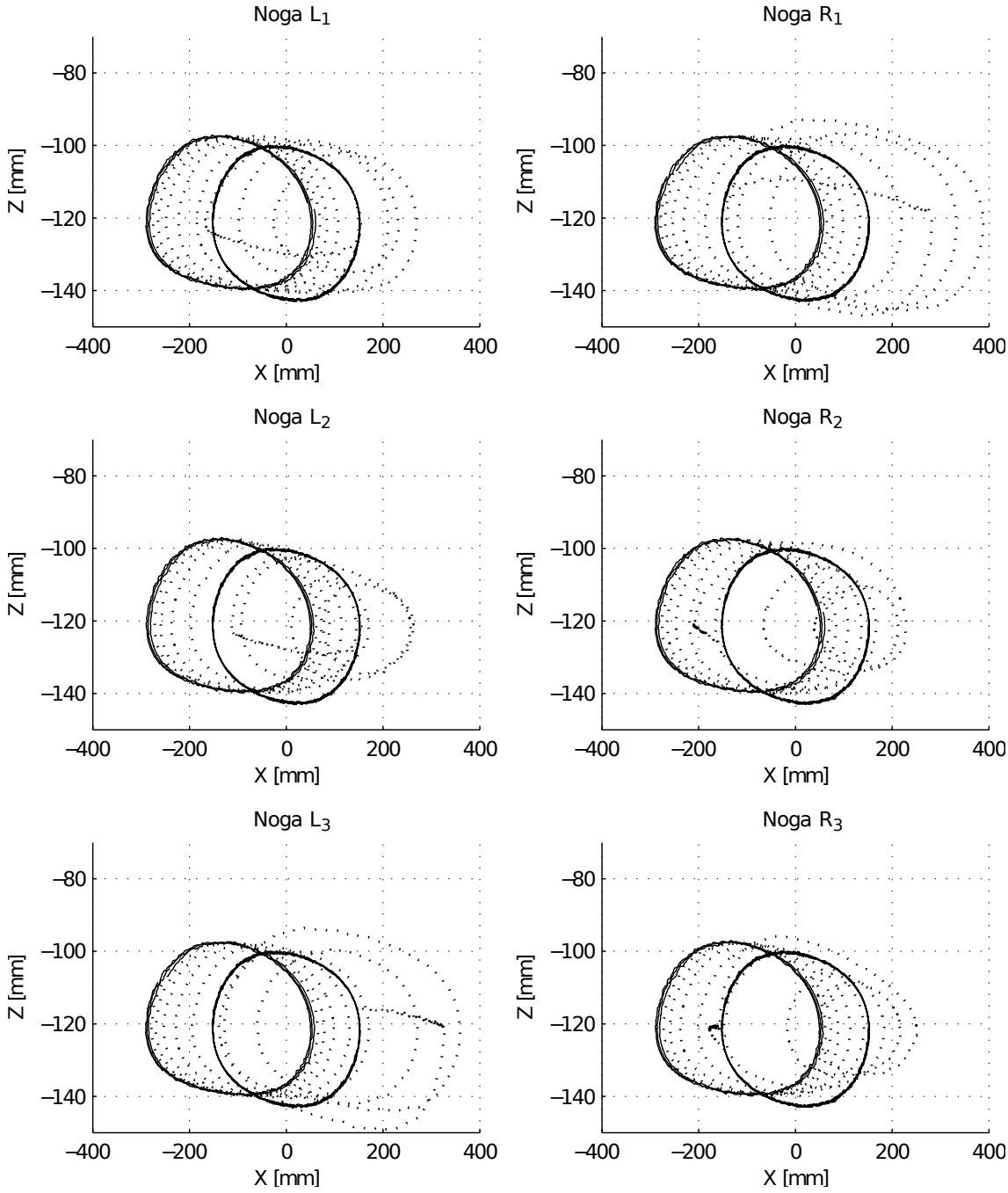
Rysunek 7.22: Ślad położenia w płaszczyźnie XZ końcówek sześciu nóg robota sterowanego algorytmem opartym na kracie Toda-Rayleigha podczas zmiany trybu chodu z trybu $m = 1$ do trybu $m = 2$ – wyniki symulacji modelu robota (cykle graniczne oznaczono linią ciągła, trajektorię przejścia oznaczono linią kropkowaną)



Rysunek 7.23: Ślad położenia w płaszczyźnie XZ końcówek sześciu nóg robota sterowanego algorytmem opartym na kracie Toda-Rayleigha podczas zmiany trybu chodu z trybu $m = 1$ do trybu $m = 0$ – wyniki symulacji modelu robota (cykle graniczne oznaczono linią ciągła, trajektorię przejścia oznaczono linią kropkowaną)



Rysunek 7.24: Ślad położenia w płaszczyźnie XZ końcówek sześciu nóg robota sterowanego algorytmem opartym na kracie Toda-Rayleigha podczas zmiany trybu chodu z trybu $m = 0$ do trybu $m = 2$ – wyniki symulacji modelu robota (cykle graniczne oznaczono linią ciągła, trajektorię przejścia oznaczono linią kropkowaną)



Rysunek 7.25: Ślad położenia w płaszczyźnie XZ końcówek sześciu nóg robota sterowanego algorytmem opartym na kracie Toda-Rayleigha podczas zmiany trybu chodu z trybu $m = 0$ do trybu $m = 1$ – wyniki symulacji modelu robota (cykle graniczne oznaczono linią ciągła, trajektorię przejścia oznaczono linią kropkowaną)

Na wykresach 7.20, 7.21, 7.22, 7.23, 7.24 i 7.25 przedstawiono wyniki symulacji procesu zmiany trybu chodu (wszystkie kombinacje). Zmiany trybów są płynne i stosunkowo szybkie. Podczas przejścia na tryb $m = 1$ można dostrzec efekt wolnego zbiegania trajektorii do cyklu granicznego zaobserwowany przy rozruchu.

Uzyskane wyniki symulacyjne wskazują na poprawność założeń przyjętych na etapie budowania algorytmu. Ostatecznym potwierdzeniem są wyniki eksperymentu z rzeczywistym robotem.

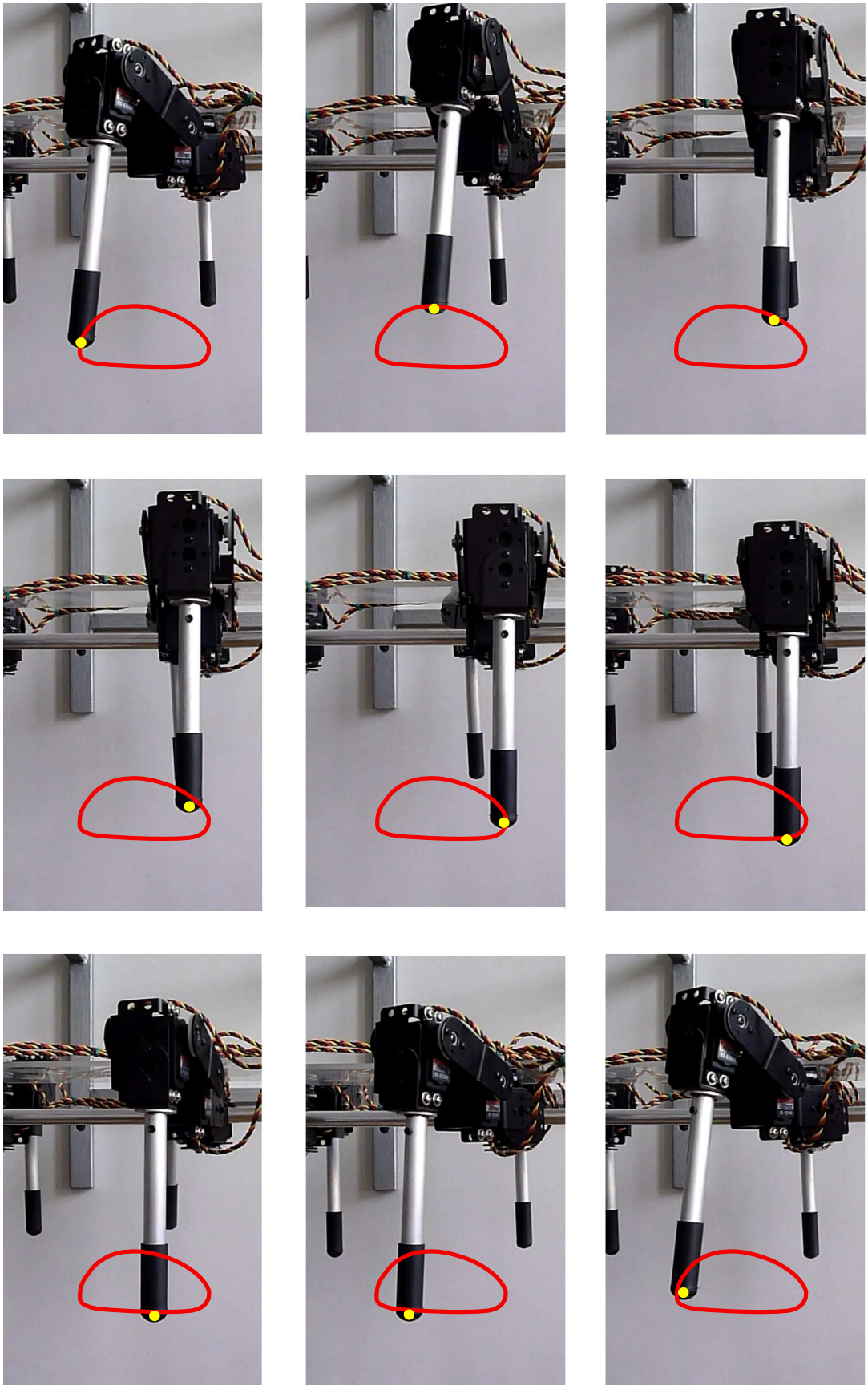
7.6 Wyniki eksperymentów rzeczywistych chodu robota

Algorytm oparty na kracie Toda-Rayleigha został zaimplementowany na stanowisku rzeczywistego robota sześcionożnego. Wykorzystano do tego wcześniejszy model symulacyjny algorytmu. Narzędzia do automatycznej generacji kodu pozwoliły na uzyskanie kodu źródłowego algorytmu w postaci dwóch głównych funkcji:

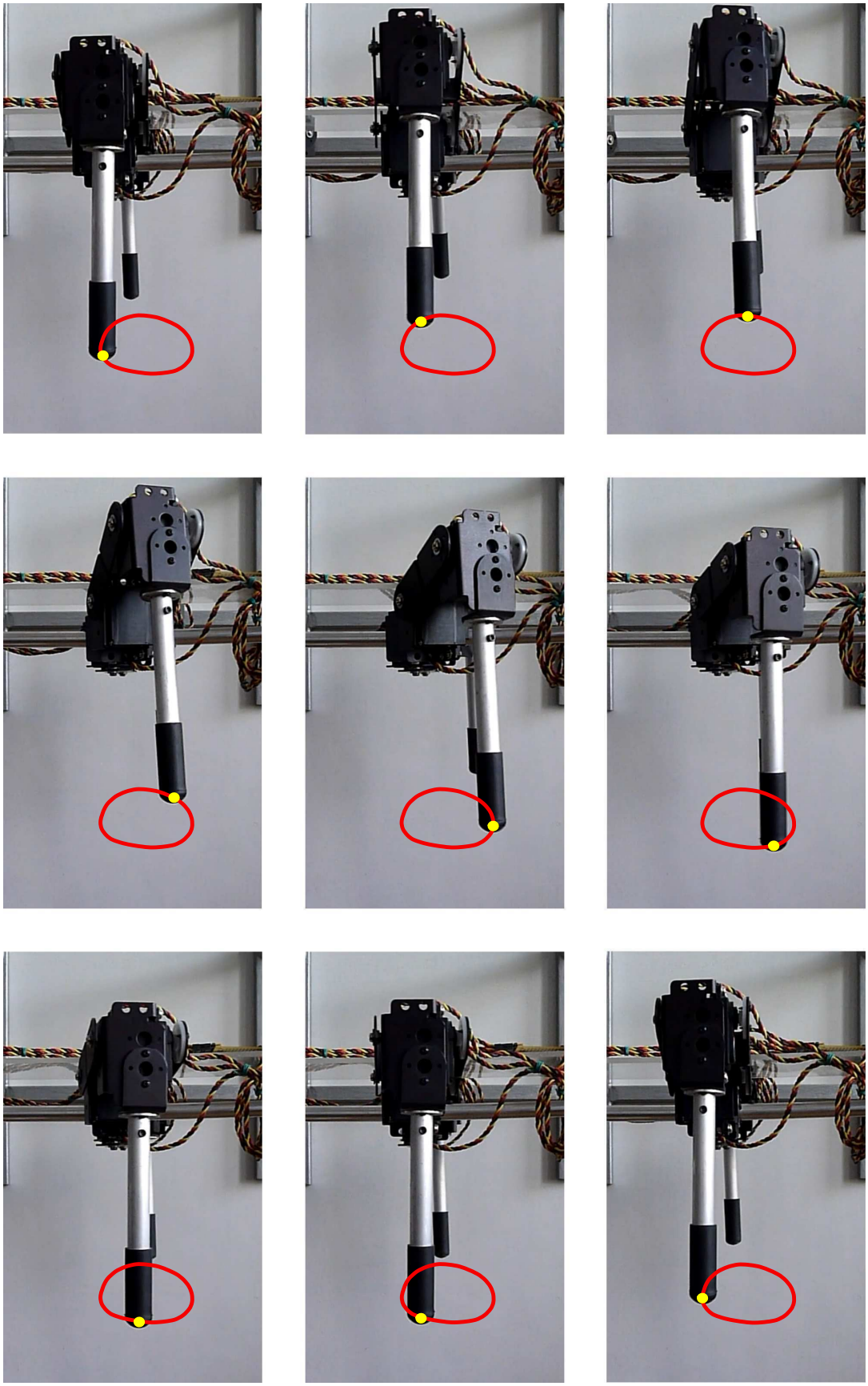
- funkcji inicjującej *CPG_TR_init*,
- funkcji wyliczającej jeden krok właściwego algorytmu *CPG_TR_step*.

Algorytm został zintegrowany ze stanowiskiem robota i uruchomiony. Przetestowano eksperymentalnie wszystkie fazy wszystkich trzech trybów chodu. Wyniki potwierdziły poprawność wcześniejszych eksperymentów symulacyjnych. Na wykresach 7.26, 7.27 i 7.28 przedstawiono klatki filmu kolejnych faz ruchu jednej z nóg robota dla wszystkich trzech trybów chodu. Klatki pochodzą z eksperymentu podczas, którego robot był podwieszony na specjalnie przygotowanym statywie. W tak umiejscowionym robocie nogi wykonują swobodne ruchy w przestrzeni nie stykając się z podłożem. Niezakłócony przez podłoże ruch nogi jest powtarzalny w przestrzeni kartezjańskiej, a statyw determinuje właściwie nałożenie trajektorii symulacyjnej końcówki nogi na kolejne klatki filmu. Klatki to właśnie chwilowe położenia rzeczywistej trajektorii końcówki nogi.

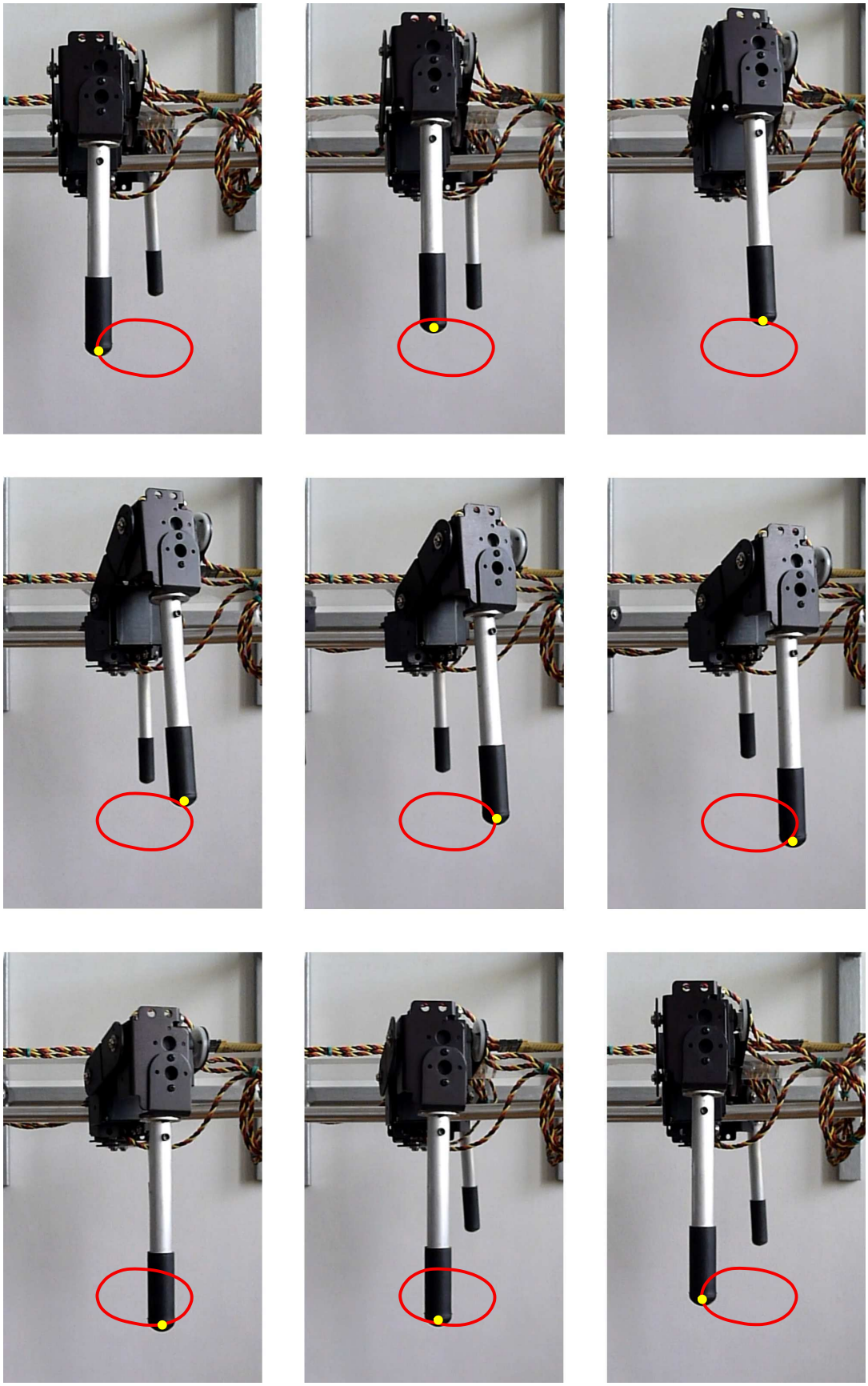
Podczas eksperymentów z robotem kroczącym na podłożu udało się uzyskać ruch postępowy robota dla wszystkich trybów chodu. Sprawdzono także wszystkie fazy przejścia. Ruch robota we wszystkich fazach chodu był płynny, fazy przejścia krótkie. Nie zamyka to drogi do poprawy jakości algorytmu w przyszłości. Wyniki są obiecujące i zachęcają do prac nad algorytmem.



Rysunek 7.26: Klatki filmowe ze sterowania robotem z wykorzystaniem algorytmu opartego na kracie Toda-Rayleigha — tryb chodu $m = 2$. Nałożono symulacyjne trajektorie ruchu końcówki nogi. Jasny punkt na trajektorii ilustruje bieżące położenie końcówki nogi.



Rysunek 7.27: Klatki filmowe ze sterowania robotem z wykorzystaniem algorytmu opartego na kracie Toda-Rayleigha — tryb chodu $m = 1$. Nałożono symulacyjne trajektorie ruchu końcówki nogi. Jasny punkt na trajektorii ilustruje bieżące położenie końcówki nogi.



Rysunek 7.28: Klatki filmowe ze sterowania robotem z wykorzystaniem algorytmu opartego na kracie Toda-Rayleigha — tryb chodu $m = 0$. Nałożono symulacyjne trajektorie ruchu końcówki nogi. Jasny punkt na trajektorii ilustruje bieżące położenie końcówki nogi.

Rozdział 8

Podsumowanie

Główny cel pracy osiągnięto. Zbudowano generatory chodu hexapoda w postaci oscylatorów liniowych i nieliniowych. Szczególną użyteczność ma krata Toda-Rayleigha z uwagi na płynną zmianę przy przełączaniu między różnymi rodzajami chodów. Tym samym zrealizowano tezę, że zastosowanie zbiorów oscylatorów nieliniowych w modelu CPG – Central Pattern Generator pozwala na osiągnięcie płynnych przejść pomiędzy trybami chodu.

Jak już wspomniano we Wstępie cel główny jest rozumiany szeroko. Obejmuje nie tylko symulacje generatorów chodu, ale przede wszystkim ich realizacje. Robot porusza się wykorzystując wszystkie założone tryby chodu. Ruch ilustrują klatki filmu przedstawione w pracy. Generatorem chodu jest program nazwany CPG (ang. Central Pattern Generator) zapisany w układzie mikroprocesorowym. W ten sposób wykazana została teza, że zastosowanie oscylatorów nieliniowych umożliwia efektywną realizację CPG w układzie mikroprocesorowym.

Kolejna teza pracy brzmi: obserwacja działania wzorców biologicznych jest najbardziej naturalnym sposobem budowy nieznanych algorytmów ruchu gwarantującym zadowalający efekt kroczenia, a nawet w wielu przypadkach najlepszym z możliwych. Zrealizowane algorytmy były wzorowane na chodach stosowanych w przyrodzie przez owady sześcionożne. Autor nie znalazł lepszych wzorców chodu niż biologiczne i będąc przekonany o słuszności teorii ewolucji uważa, że natura jest źródłem najlepszych wzorców. Na ogół nie spotyka się konstrukcji robotów kroczących o nieparzystej ilości nóg, z wyjątkiem jednonożnego robota. W tym przypadku jednak chodzi raczej o demonstrację kroczenia (skakania) na jednej nodze.

W pracy zrealizowano kilka celów pobocznych. Należy podkreślić, że zbudowano kompletne środowisko programowo-sprzętowe służące do badań i eksperymentów symulacyjnych i rzeczywistych ruchu hexpoda. Przede wszystkim uporano się ze złożonością zadania sterowania, które polega na równoczesnym zbieraniu chwilowych położeń osiemnastu członów robota i wysyłaniu sterowań do osiemnastu serwomechanizmów. Powodzeniem zakończyła się nie tylko generacja stosownego chodu i wymuszenie właściwego kroczenia robota, ale powiązanie rzeczywistego ruchu z modelem symulacyjnym dynamiki robota i identyfikacją tego modelu.

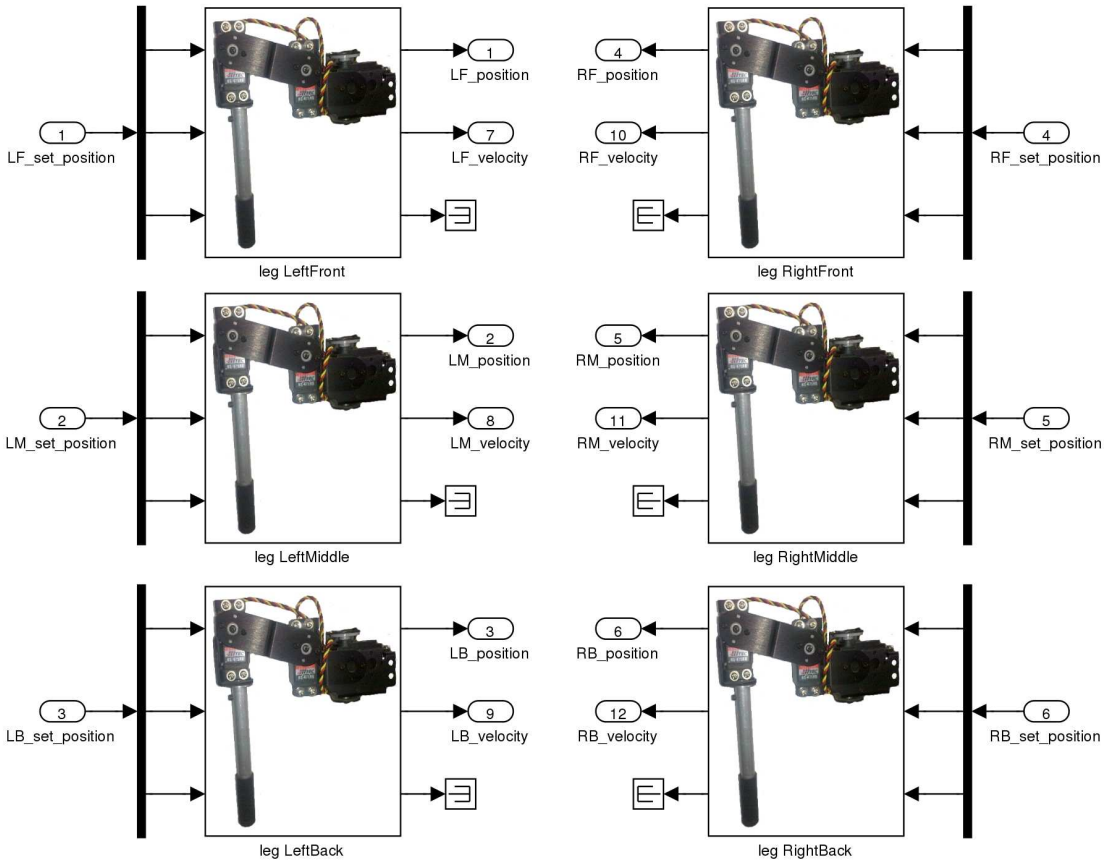
Autor planuje dalsze badania nad konstrukcją w pełni autonomicznego robota sześcionożnego, a przede wszystkim intryguje go niezwykle ciekawe zagadnienie pokonywania schodów i nim zajmie się w pierwszej kolejności. Niespełnionym dotychczas zamierzeniem jest zamiana podanych hipotez o stabilności, dokładniej ograniczoności trajektorii przy przechodzeniu z jednego cyklu granicznego w drugi cykl, w twierdzenia. Jest to zadanie istotne, otwarte dla przyszłych badań.

Dodatek A

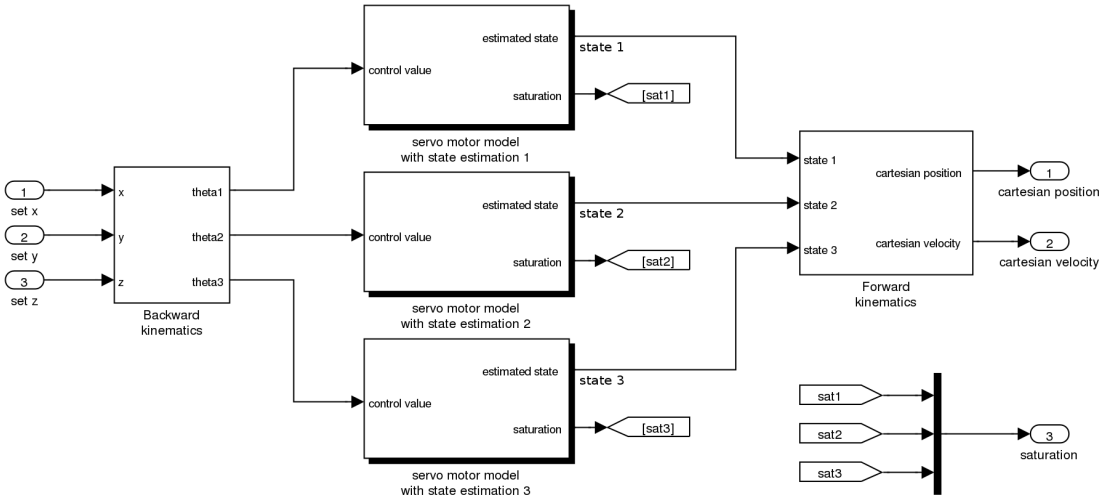
Modele

W dodatku umieszczono zrzuty ekranu najważniejszych fragmentów modeli wykorzystywanych podczas eksperymentów symulacyjnych.

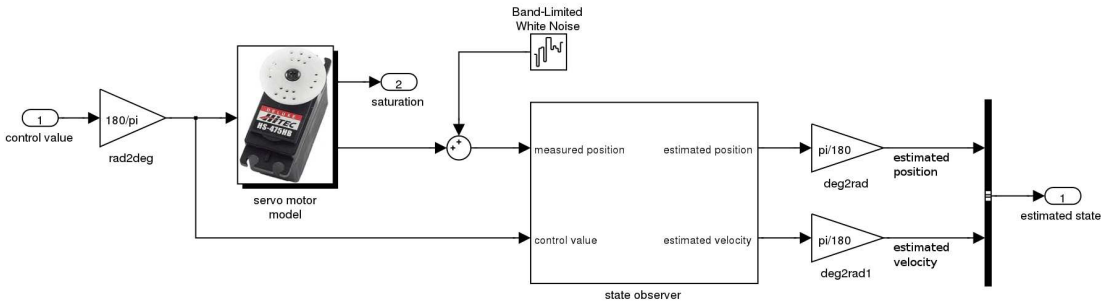
A.1 Model robota sześcionożnego



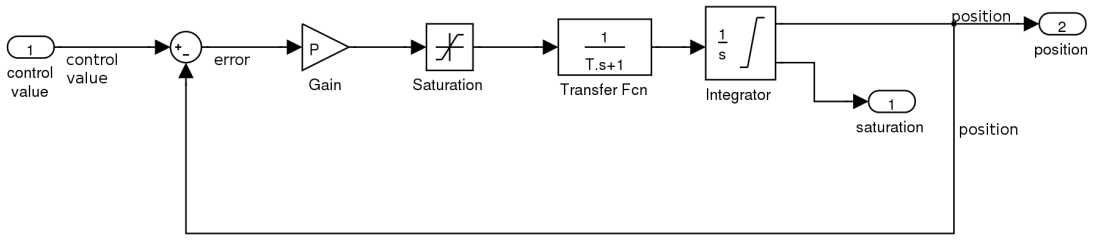
Rysunek A.1: Główny widok modelu robota



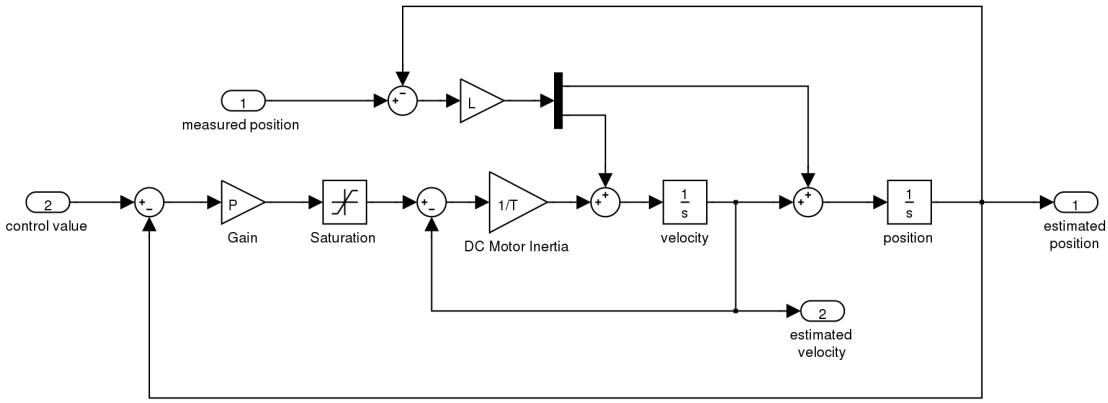
Rysunek A.2: Widok modelu nogi robota



Rysunek A.3: Widok modelu serwomechanizmu z obserwatorem stanu

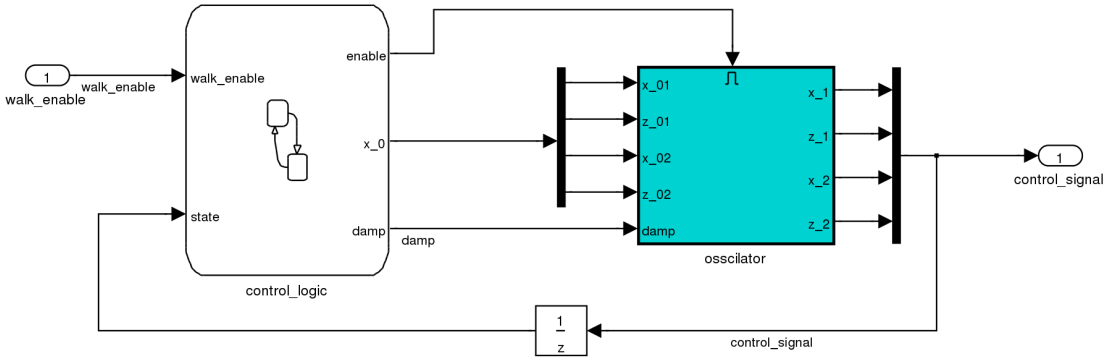


Rysunek A.4: Widok modelu serwomechanizmu

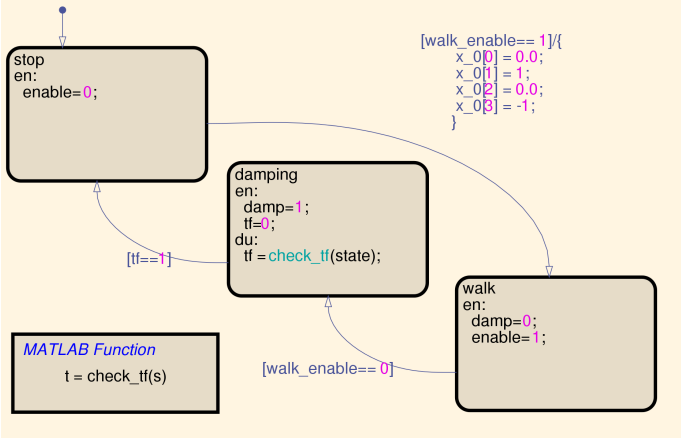


Rysunek A.5: Widok modelu obserwatora stanu

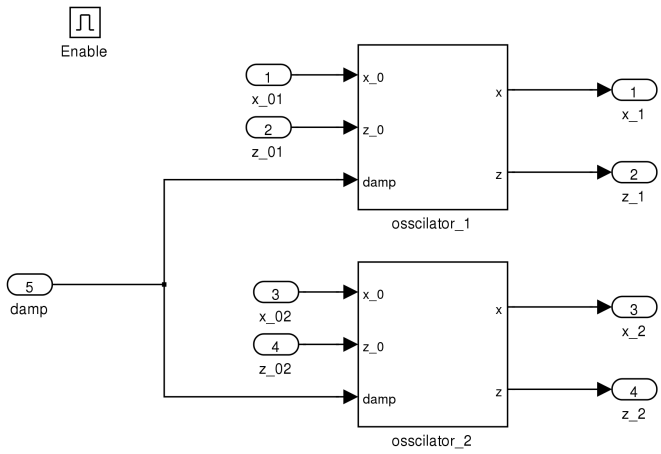
A.2 Model algorytmu opartego na oscylatorach liniowych



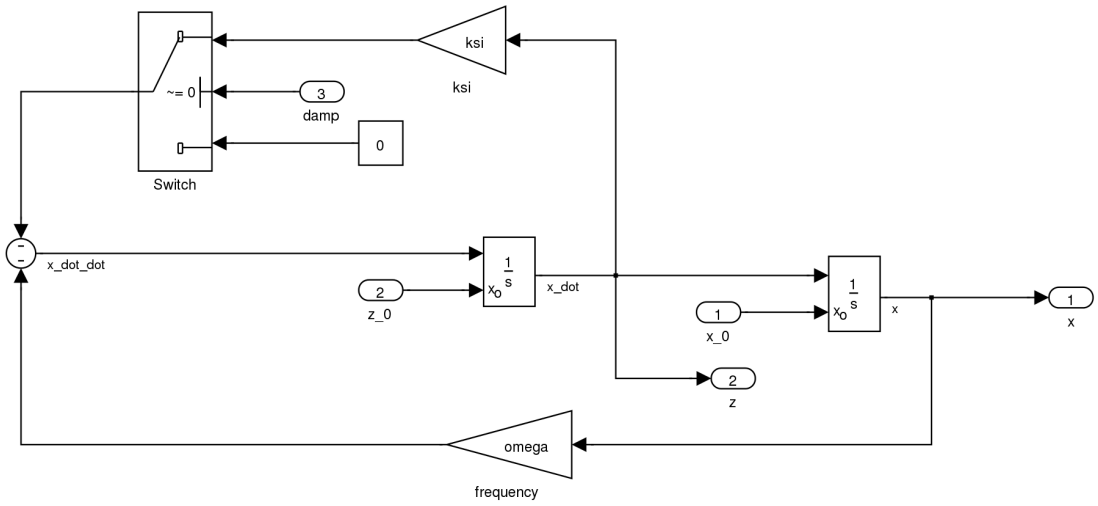
Rysunek A.6: Widok głównego modelu algorytmu sterowania opartego na oscylatorach liniowych



Rysunek A.7: Widok maszyny stanu modelu algorytmu sterowania opartego na oscylatorach liniowych

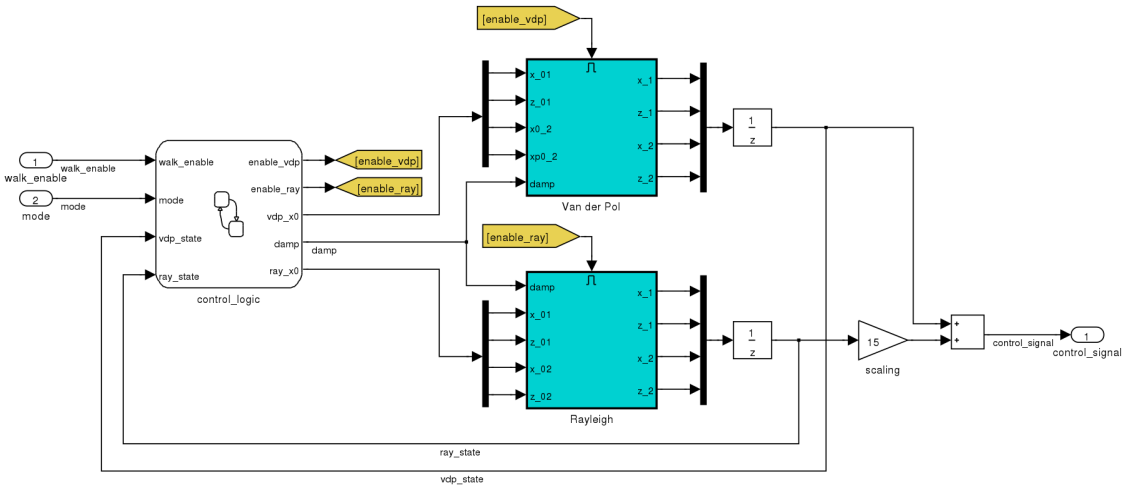


Rysunek A.8: Widok struktury oscylatorów algorytmu sterowania opartego na oscylatorach liniowych

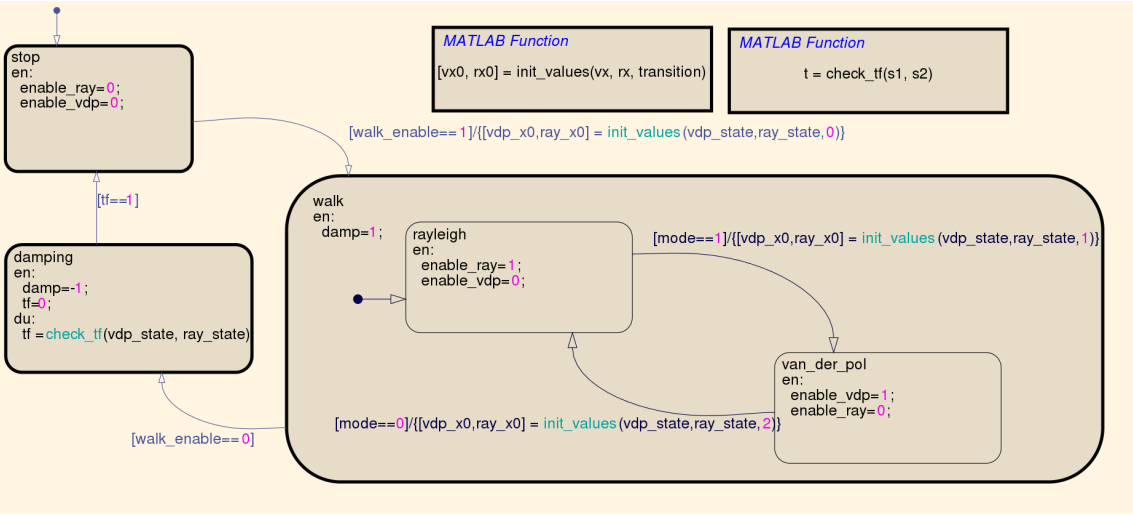


Rysunek A.9: Widok jednego z oscylatorów algorytmu sterowania opartego na oscylatorach liniowych

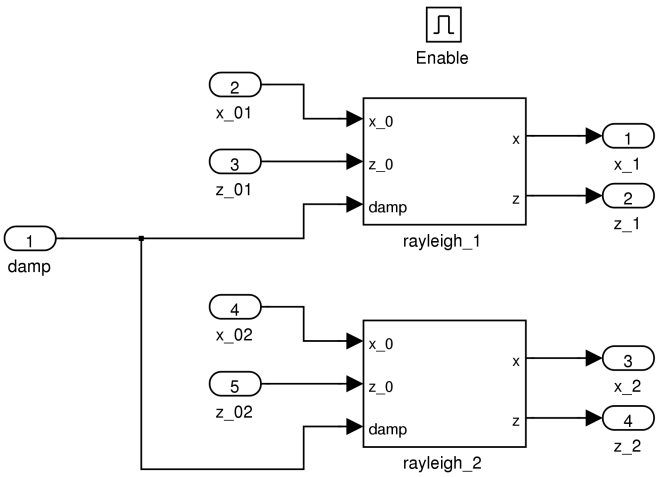
A.3 Model algorytmu opartego na oscylatorach nieliniowych



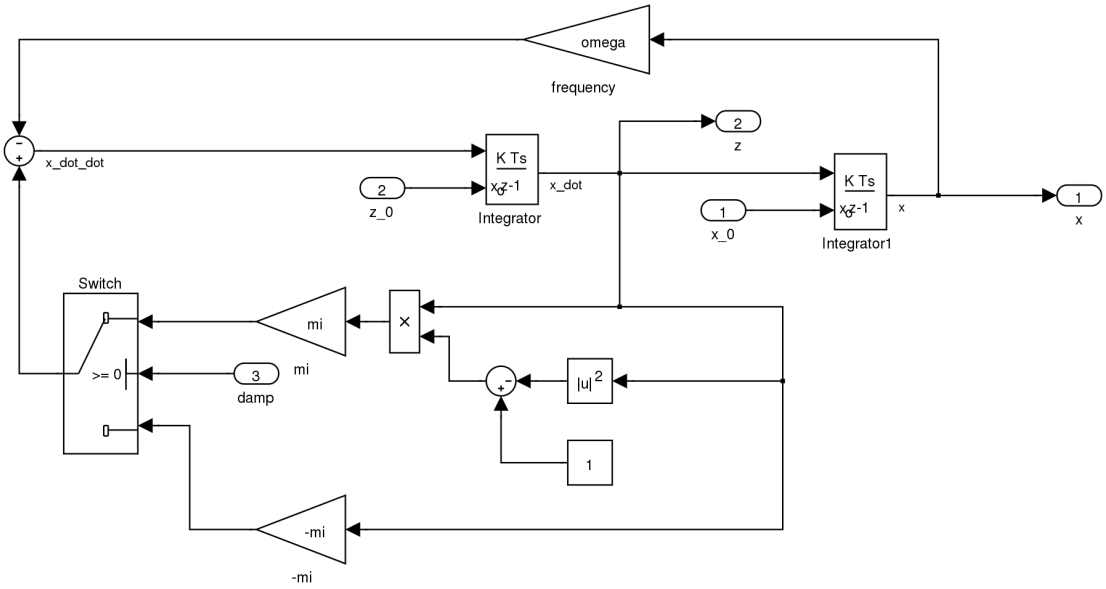
Rysunek A.10: Widok głównego modelu algorytmu sterowania opartego na oscylatorach nieliniowych



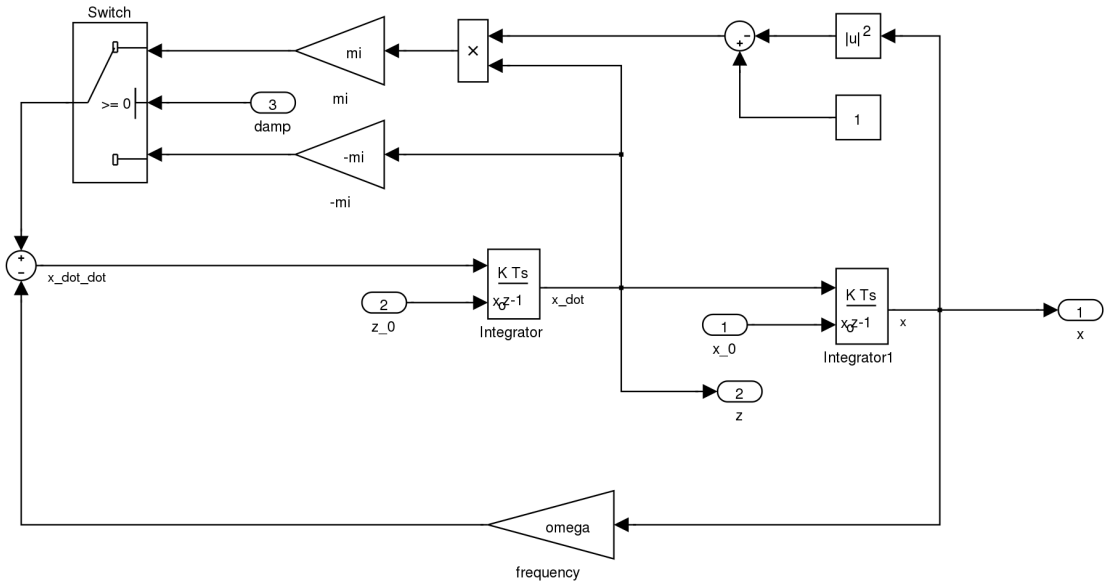
Rysunek A.11: Widok maszyny stanu modelu algorytmu sterowania opartego na oscylatorach nieliniowych



Rysunek A.12: Widok struktury oscylatorów modelu algorytmu sterowania opartego na oscylatorach nieliniowych

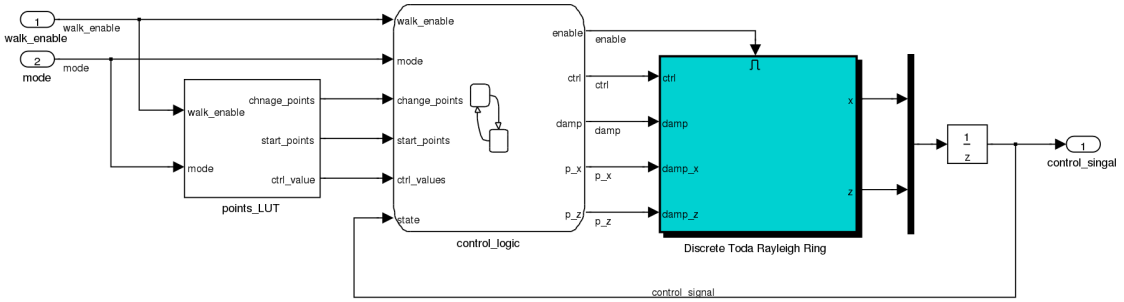


Rysunek A.13: Widok dynamiki oscylatora Rayleigha modelu algorytmu sterowania opartego na oscylatorach nieliniowych

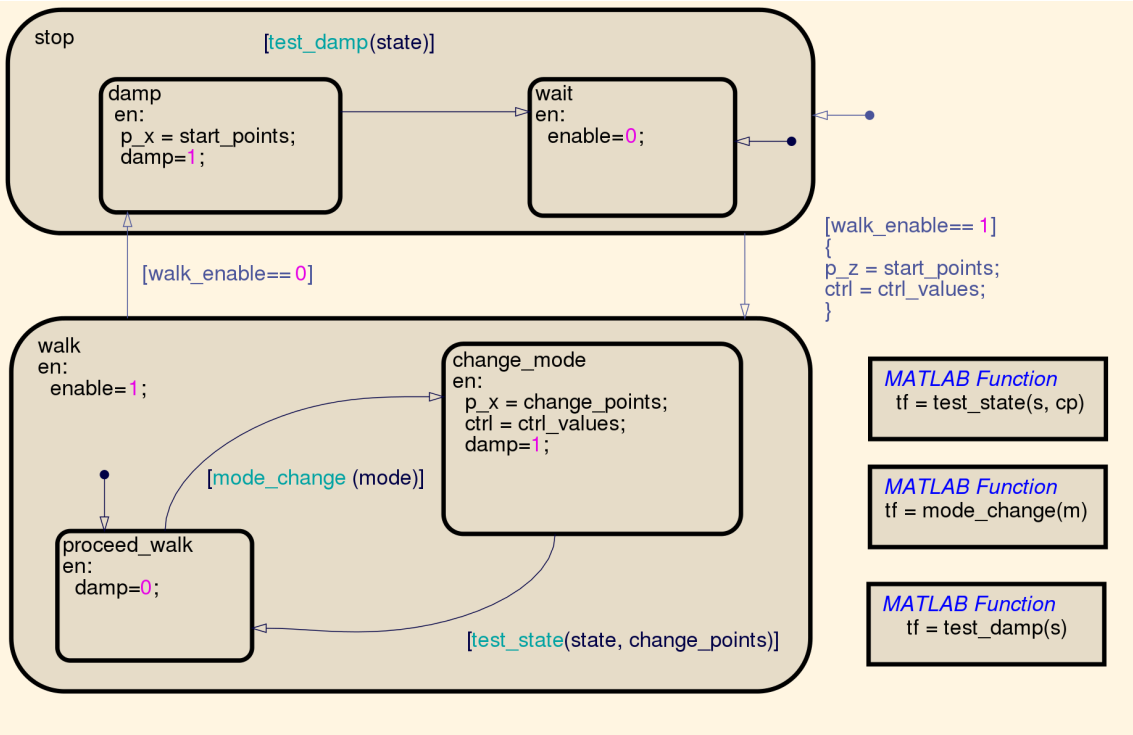


Rysunek A.14: Widok dynamiki oscylatora Van der Pola modelu algorytmu sterowania opartego na oscylatorach nieliniowych

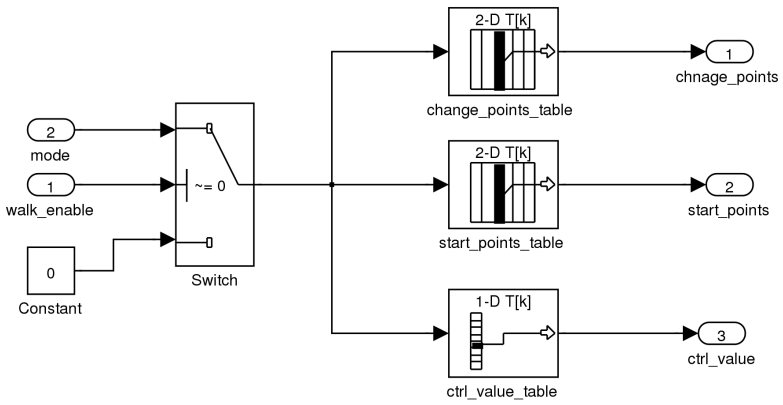
A.4 Model algorytmu opartego na kracie Toda-Rayleigha



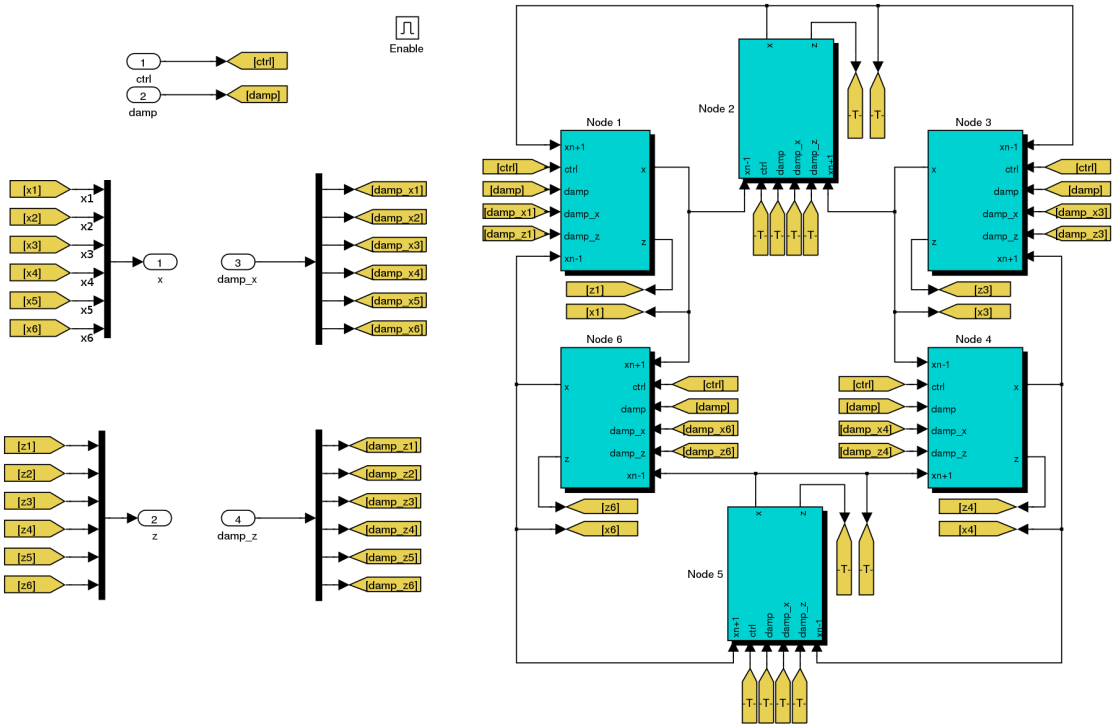
Rysunek A.15: Widok głównego modelu algorytmu sterowania opartego na kracie Toda-Rayleigha



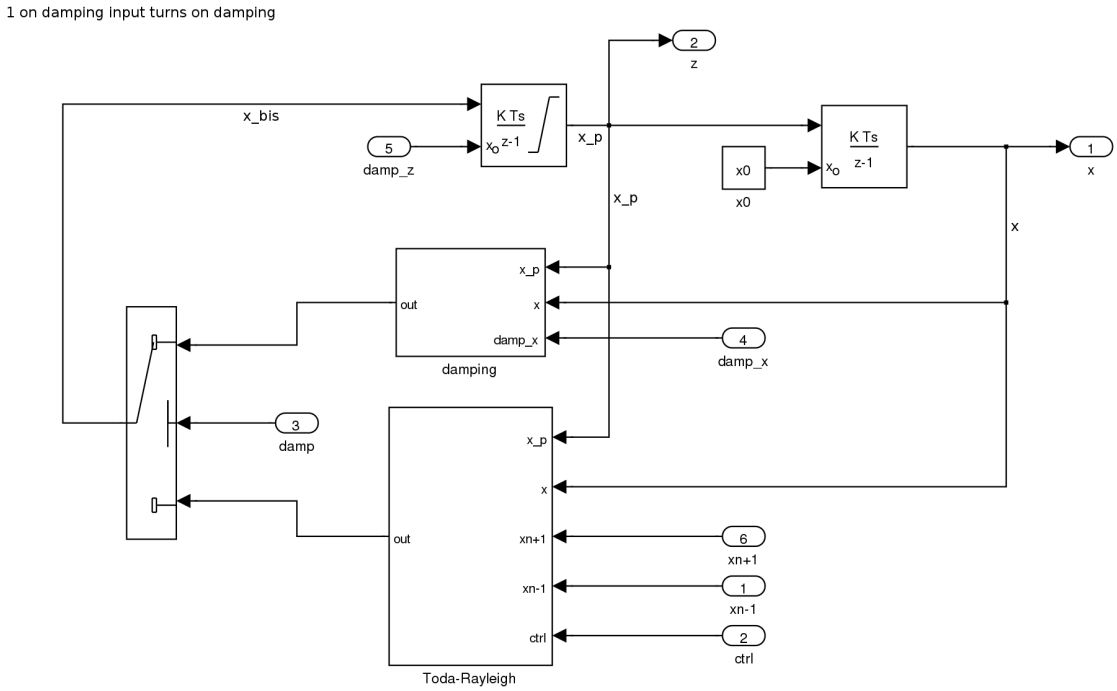
Rysunek A.16: Widok maszyny stanu modelu algorytmu sterowania opartego na kracie Toda-Rayleigha



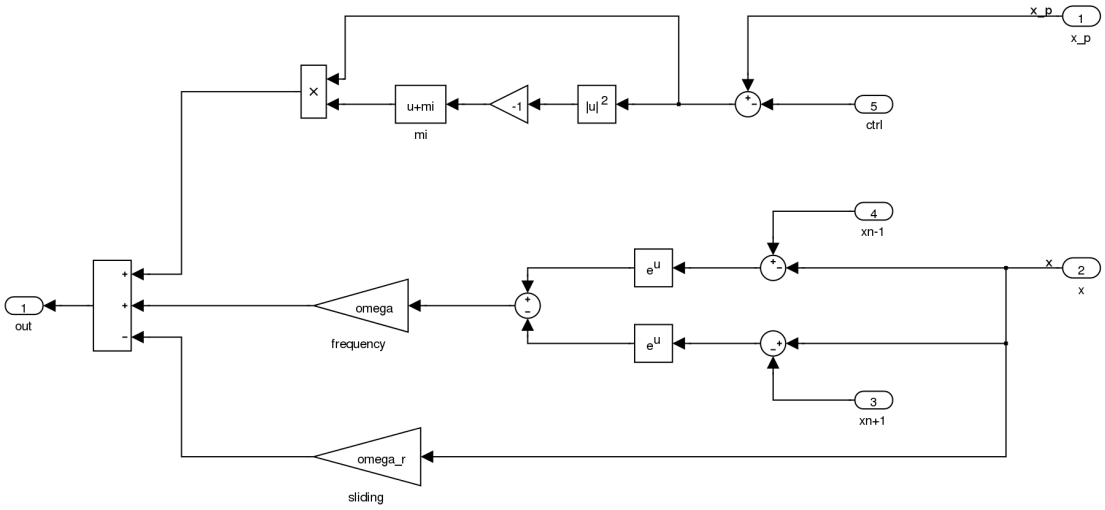
Rysunek A.17: Widok generatora punktów C_n i D_n modelu algorytmu sterowania opartego na kracie Toda-Rayleigha



Rysunek A.18: Widok struktury kraty modelu algorytmu sterowania opartego na kracie Toda-Rayleigha



Rysunek A.19: Widok jednego z elementów kraty modelu algorytmu sterowania opartego na kracie Toda-Rayleigha



Rysunek A.20: Widok równania elementu kraty modelu algorytmu sterowania opartego na kracie Toda-Rayleigha

Dodatek B

Kod źródłowy

Kod źródłowy algorytmu wykorzystującego oscylatory Rayleigha i Van der Pola:

```
/* Named constants for Chart: '<Root>/control_logic' */
#define CPG_nonlinea_IN_NO_ACTIVE_CHILD ((uint8_T)0U)
#define CPG_nonlinear_di_IN_van_der_pol ((uint8_T)2U)
#define CPG_nonlinear_discr_IN_rayleigh ((uint8_T)1U)
#define CPG_nonlinear_discre_IN_dumping ((uint8_T)1U)
#define CPG_nonlinear_discrete_IN_stop ((uint8_T)2U)
#define CPG_nonlinear_discrete_IN_walk ((uint8_T)3U)

/* Block signals (auto storage) */
BlockIO_CPG_nonlinear_discrete CPG_nonlinear_discrete_B;

/* Block states (auto storage) */
D_Work_CPG_nonlinear_discrete CPG_nonlinear_discrete_DWork;

/* Real-time model */
RT_MODEL_CPG_nonlinear_discrete CPG_nonlinear_discrete_M_;
RT_MODEL_CPG_nonlinear_discrete *const CPG_nonlinear_discrete_M =
  &CPG_nonlinear_discrete_M_;

/* Forward declaration for local functions */
static void CPG_nonlinear_discr_init_values(const real_T vx[4], const real_T rx
  [4], int32_T transition, real_T vx0[4], real_T rx0[4]);
static int32_T CPG_nonlinear_discrete_check_tf(const real_T s1[4], const real_T
  s2[4]);

/* Function for Chart: '<Root>/control_logic' */
static void CPG_nonlinear_discr_init_values(const real_T vx[4], const real_T rx
  [4], int32_T transition, real_T vx0[4], real_T rx0[4])
{
  /* MATLAB Function 'init_values': '<S3>:140' */
  /*  transition set to 0 - means initialization */
  /*  transition set to 1 - means change to Van der Pol */
  /*  transition set to 2 - means change to Rayleigh */
  if (transition == 1) {
    /* '<S3>:140:7' */
    /* '<S3>:140:8' */
    vx0[0] = 15.0 * rx[0];
    vx0[1] = 15.0 * rx[1];
    vx0[2] = 15.0 * rx[2];
    vx0[3] = 15.0 * rx[3];

    /* '<S3>:140:9' */
    rx0[0] = 0.0;
    rx0[1] = 0.0;
    rx0[2] = 0.0;
    rx0[3] = 0.0;
  } else if (transition == 2) {
    /* '<S3>:140:10' */
    /* '<S3>:140:11' */
    rx0[0] = vx[0] / 15.0;
```

```
    rx0[1] = vx[1] / 15.0;
    rx0[2] = vx[2] / 15.0;
    rx0[3] = vx[3] / 15.0;

    /* '<S3>:140:12' */
    vx0[0] = 0.0;
    vx0[1] = 0.0;
    vx0[2] = 0.0;
    vx0[3] = 0.0;
  } else {
    /* '<S3>:140:14' */
    vx0[0] = 0.0;
    vx0[1] = 0.1;
    vx0[2] = 0.0;
    vx0[3] = -0.1;

    /* '<S3>:140:15' */
    rx0[0] = 0.0;
    rx0[1] = 0.1;
    rx0[2] = 0.0;
    rx0[3] = -0.1;
  }
}

/* Function for Chart: '<Root>/control_logic' */
static int32_T CPG_nonlinear_discrete_check_tf(const real_T s1[4], const real_T
  s2[4])
{
  int32_T t;

  /* MATLAB Function 'check_tf': '<S3>:153' */
  /* '<S3>:153:3' */
  /* '<S3>:153:4' */
  /* '<S3>:153:6' */
  /* '<S3>:153:7' */
  if ((((fabs(s1[0]) + fabs(s1[1])) + fabs(s1[2])) + fabs(s1[3])) + (((fabs(s2[0])
       + fabs(s2[1])) + fabs(s2[2])) + fabs(s2[3])) <= 0.01) {
    /* '<S3>:153:9' */
    /* '<S3>:153:10' */
    t = 1;
  } else {
    /* '<S3>:153:12' */
    t = 0;
  }

  return t;
}

/* Model step function */
void CPG_nonlinear_step(int32_T walk_enable, int32_T mode, real_T
  control_signal[4])
{
  real_T rtb_Switch_j;
  real_T rtb_x_dot_dot_d;

  /* Outport: '<Root>/control_signal' incorporates:
   *  Gain: '<Root>/scaling'
   *  Sum: '<Root>/Add'
   *  UnitDelay: '<Root>/Unit Delay'
   *  UnitDelay: '<Root>/Unit Delay1'
   */
  control_signal[0] = 15.0 * CPG_nonlinear_discrete_DWork.UnitDelay1_DSTATE[0] +
    CPG_nonlinear_discrete_DWork.UnitDelay_DSTATE[0];
  control_signal[1] = 15.0 * CPG_nonlinear_discrete_DWork.UnitDelay1_DSTATE[1] +
    CPG_nonlinear_discrete_DWork.UnitDelay_DSTATE[1];
  control_signal[2] = 15.0 * CPG_nonlinear_discrete_DWork.UnitDelay1_DSTATE[2] +
    CPG_nonlinear_discrete_DWork.UnitDelay_DSTATE[2];
  control_signal[3] = 15.0 * CPG_nonlinear_discrete_DWork.UnitDelay1_DSTATE[3] +
    CPG_nonlinear_discrete_DWork.UnitDelay_DSTATE[3];

  /* Chart: '<Root>/control_logic' incorporates:
```

```
 *  Inport: '<Root>/mode'
 *  Inport: '<Root>/walk_enable'
 *  UnitDelay: '<Root>/Unit Delay'
 *  UnitDelay: '<Root>/Unit Delay1'
 */
/* Gateway: control_logic */
/* During: control_logic */
if (CPG_nonlinear_discrete_DWork.is_active_c1_CPG_nonlinear_disc == 0) {
  /* Entry: control_logic */
  CPG_nonlinear_discrete_DWork.is_active_c1_CPG_nonlinear_disc = 1U;

  /* Entry Internal: control_logic */
  /* Transition: '<S3>:148' */
  CPG_nonlinear_discrete_DWork.is_c1_CPG_nonlinear_discrete =
    CPG_nonlinear_discrete_IN_stop;

  /* Entry 'stop': '<S3>:95' */
  CPG_nonlinear_discrete_B.enable_ray = 0;
  CPG_nonlinear_discrete_B.enable_vdp = 0;
} else {
  switch (CPG_nonlinear_discrete_DWork.is_c1_CPG_nonlinear_discrete) {
   case CPG_nonlinear_discre_IN_dumping:
    /* During 'dumping': '<S3>:149' */
    if (CPG_nonlinear_discrete_DWork.tf == 1) {
      /* Transition: '<S3>:150' */
      CPG_nonlinear_discrete_DWork.is_c1_CPG_nonlinear_discrete =
        CPG_nonlinear_discrete_IN_stop;

      /* Entry 'stop': '<S3>:95' */
      CPG_nonlinear_discrete_B.enable_ray = 0;
      CPG_nonlinear_discrete_B.enable_vdp = 0;
    } else {
      CPG_nonlinear_discrete_DWork.tf = CPG_nonlinear_discrete_check_tf
        (CPG_nonlinear_discrete_DWork.UnitDelay_DSTATE,
         CPG_nonlinear_discrete_DWork.UnitDelay1_DSTATE);
    }
    break;

   case CPG_nonlinear_discrete_IN_stop:
    /* During 'stop': '<S3>:95' */
    if (walk_enable == 1) {
      /* Transition: '<S3>:146' */
      CPG_nonlinear_discrete_DWork.is_c1_CPG_nonlinear_discrete =
        CPG_nonlinea_IN_NO_ACTIVE_CHILD;
      CPG_nonlinear_discr_init_values
        (CPG_nonlinear_discrete_DWork.UnitDelay_DSTATE,
         CPG_nonlinear_discrete_DWork.UnitDelay1_DSTATE, 0,
         CPG_nonlinear_discrete_B.vdp_x0, CPG_nonlinear_discrete_B.ray_x0);
      CPG_nonlinear_discrete_DWork.is_c1_CPG_nonlinear_discrete =
        CPG_nonlinear_discrete_IN_walk;

      /* Entry 'walk': '<S3>:94' */
      CPG_nonlinear_discrete_B.dump = 1;

      /* Entry Internal 'walk': '<S3>:94' */
      /* Transition: '<S3>:105' */
      CPG_nonlinear_discrete_DWork.is_walk = CPG_nonlinear_discr_IN_rayleigh;

      /* Entry 'rayleigh': '<S3>:103' */
      CPG_nonlinear_discrete_B.enable_ray = 1;
      CPG_nonlinear_discrete_B.enable_vdp = 0;
    }
    break;

   default:
    /* During 'walk': '<S3>:94' */
    if (walk_enable == 0) {
      /* Transition: '<S3>:147' */
      /* Exit Internal 'walk': '<S3>:94' */
      CPG_nonlinear_discrete_DWork.is_walk = CPG_nonlinea_IN_NO_ACTIVE_CHILD;
      CPG_nonlinear_discrete_DWork.is_c1_CPG_nonlinear_discrete =
```

```
        CPG_nonlinear_discre_IN_dumping;

      /* Entry 'dumping': '<S3>:149' */
      CPG_nonlinear_discrete_B.dump = -1;
      CPG_nonlinear_discrete_DWork.tf = 0;
    } else if (CPG_nonlinear_discrete_DWork.is_walk ==
               CPG_nonlinear_discr_IN_rayleigh) {
      /* During 'rayleigh': '<S3>:103' */
      if (mode == 1) {
        /* Transition: '<S3>:106' */
        CPG_nonlinear_discrete_DWork.is_walk = CPG_nonlinea_IN_NO_ACTIVE_CHILD;
        CPG_nonlinear_discr_init_values
          (CPG_nonlinear_discrete_DWork.UnitDelay_DSTATE,
           CPG_nonlinear_discrete_DWork.UnitDelay1_DSTATE, 1,
           CPG_nonlinear_discrete_B.vdp_x0, CPG_nonlinear_discrete_B.ray_x0);
        CPG_nonlinear_discrete_DWork.is_walk = CPG_nonlinear_di_IN_van_der_pol;

        /* Entry 'van_der_pol': '<S3>:104' */
        CPG_nonlinear_discrete_B.enable_vdp = 1;
        CPG_nonlinear_discrete_B.enable_ray = 0;
      }
    } else {
      /* During 'van_der_pol': '<S3>:104' */
      if (mode == 0) {
        /* Transition: '<S3>:107' */
        CPG_nonlinear_discrete_DWork.is_walk = CPG_nonlinea_IN_NO_ACTIVE_CHILD;
        CPG_nonlinear_discr_init_values
          (CPG_nonlinear_discrete_DWork.UnitDelay_DSTATE,
           CPG_nonlinear_discrete_DWork.UnitDelay1_DSTATE, 2,
           CPG_nonlinear_discrete_B.vdp_x0, CPG_nonlinear_discrete_B.ray_x0);
        CPG_nonlinear_discrete_DWork.is_walk = CPG_nonlinear_discr_IN_rayleigh;

        /* Entry 'rayleigh': '<S3>:103' */
        CPG_nonlinear_discrete_B.enable_ray = 1;
        CPG_nonlinear_discrete_B.enable_vdp = 0;
      }
    }
    break;
  }
}

/* End of Chart: '<Root>/control_logic' */

/* Outputs for Enabled SubSystem: '<Root>/Rayleigh' incorporates:
 *  EnablePort: '<S1>/Enable'
 */
if (CPG_nonlinear_discrete_B.enable_ray > 0) {
  if (!CPG_nonlinear_discrete_DWork.Rayleigh_MODE) {
    /* InitializeConditions for DiscreteIntegrator: '<S4>/Integrator' */
    CPG_nonlinear_discrete_DWork.Integrator_IC_LOADING_i = 1U;

    /* InitializeConditions for DiscreteIntegrator: '<S4>/Integrator1' */
    CPG_nonlinear_discrete_DWork.Integrator1_IC_LOADING_e = 1U;

    /* InitializeConditions for DiscreteIntegrator: '<S5>/Integrator' */
    CPG_nonlinear_discrete_DWork.Integrator_IC_LOADING_h = 1U;

    /* InitializeConditions for DiscreteIntegrator: '<S5>/Integrator1' */
    CPG_nonlinear_discrete_DWork.Integrator1_IC_LOADING_c = 1U;
    CPG_nonlinear_discrete_DWork.Rayleigh_MODE = TRUE;
  }

  /* DiscreteIntegrator: '<S4>/Integrator' */
  if (CPG_nonlinear_discrete_DWork.Integrator_IC_LOADING_i != 0) {
    CPG_nonlinear_discrete_DWork.Integrator_DSTATE_c =
      CPG_nonlinear_discrete_B.ray_x0[1];
  }

  CPG_nonlinear_discrete_B.x_dot_d =
    CPG_nonlinear_discrete_DWork.Integrator_DSTATE_c;
```

```
/* End of DiscreteIntegrator: '<S4>/Integrator' */

/* DiscreteIntegrator: '<S4>/Integrator1' */
if (CPG_nonlinear_discrete_DWork.Integrator1_IC_LOADING_e != 0) {
  CPG_nonlinear_discrete_DWork.Integrator1_DSTATE_g =
    CPG_nonlinear_discrete_B.ray_x0[0];
}

CPG_nonlinear_discrete_B.x_f =
  CPG_nonlinear_discrete_DWork.Integrator1_DSTATE_g;

/* End of DiscreteIntegrator: '<S4>/Integrator1' */

/* Switch: '<S4>/Switch' incorporates:
 *  Constant: '<S4>/x1'
 *  Gain: '<S4>/-mi'
 *  Gain: '<S4>/mi'
 *  Math: '<S4>/Math Function'
 *  Product: '<S4>/Product'
 *  Sum: '<S4>/Sum'
 *
 * About '<S4>/Math Function':
 *  Operator: magnitude^2
 */
if (CPG_nonlinear_discrete_B.dump >= 0) {
  rtb_Switch_j = (1.0 - CPG_nonlinear_discrete_B.x_dot_d *
                  CPG_nonlinear_discrete_B.x_dot_d) *
    CPG_nonlinear_discrete_B.x_dot_d * 10.0;
} else {
  rtb_Switch_j = (-10.0) * CPG_nonlinear_discrete_B.x_dot_d;
}

/* End of Switch: '<S4>/Switch' */

/* Sum: '<S4>/Sum1' incorporates:
 *  Gain: '<S4>/frequency'
 */
rtb_x_dot_dot_d = rtb_Switch_j - 50.0 * CPG_nonlinear_discrete_B.x_f;

/* DiscreteIntegrator: '<S5>/Integrator' */
if (CPG_nonlinear_discrete_DWork.Integrator_IC_LOADING_h != 0) {
  CPG_nonlinear_discrete_DWork.Integrator_DSTATE_oj =
    CPG_nonlinear_discrete_B.ray_x0[3];
}

CPG_nonlinear_discrete_B.x_dot_m =
  CPG_nonlinear_discrete_DWork.Integrator_DSTATE_oj;

/* End of DiscreteIntegrator: '<S5>/Integrator' */

/* DiscreteIntegrator: '<S5>/Integrator1' */
if (CPG_nonlinear_discrete_DWork.Integrator1_IC_LOADING_c != 0) {
  CPG_nonlinear_discrete_DWork.Integrator1_DSTATE_a =
    CPG_nonlinear_discrete_B.ray_x0[2];
}

CPG_nonlinear_discrete_B.x_k =
  CPG_nonlinear_discrete_DWork.Integrator1_DSTATE_a;

/* End of DiscreteIntegrator: '<S5>/Integrator1' */

/* Switch: '<S5>/Switch' incorporates:
 *  Constant: '<S5>/x1'
 *  Gain: '<S5>/-mi'
 *  Gain: '<S5>/mi'
 *  Math: '<S5>/Math Function'
 *  Product: '<S5>/Product'
 *  Sum: '<S5>/Sum'
 *
 * About '<S5>/Math Function':
 *  Operator: magnitude^2
```

```
     */
    if (CPG_nonlinear_discrete_B.dump >= 0) {
      rtb_Switch_j = (1.0 - CPG_nonlinear_discrete_B.x_dot_m *
                      CPG_nonlinear_discrete_B.x_dot_m) *
        CPG_nonlinear_discrete_B.x_dot_m * 10.0;
    } else {
      rtb_Switch_j = (-10.0) * CPG_nonlinear_discrete_B.x_dot_m;
    }

    /* End of Switch: '<S5>/Switch' */

    /* Update for DiscreteIntegrator: '<S4>/Integrator' */
    CPG_nonlinear_discrete_DWork.Integrator_IC_LOADING_i = 0U;
    CPG_nonlinear_discrete_DWork.Integrator_DSTATE_c += 0.01 * rtb_x_dot_dot_d;

    /* Update for DiscreteIntegrator: '<S4>/Integrator1' */
    CPG_nonlinear_discrete_DWork.Integrator1_IC_LOADING_e = 0U;
    CPG_nonlinear_discrete_DWork.Integrator1_DSTATE_g += 0.01 *
      CPG_nonlinear_discrete_B.x_dot_d;

    /* Update for DiscreteIntegrator: '<S5>/Integrator' incorporates:
     *  Gain: '<S5>/frequency'
     *  Sum: '<S5>/Sum1'
     */
    CPG_nonlinear_discrete_DWork.Integrator_IC_LOADING_h = 0U;
    CPG_nonlinear_discrete_DWork.Integrator_DSTATE_oj += (rtb_Switch_j - 50.0 *
      CPG_nonlinear_discrete_B.x_k) * 0.01;

    /* Update for DiscreteIntegrator: '<S5>/Integrator1' */
    CPG_nonlinear_discrete_DWork.Integrator1_IC_LOADING_c = 0U;
    CPG_nonlinear_discrete_DWork.Integrator1_DSTATE_a += 0.01 *
      CPG_nonlinear_discrete_B.x_dot_m;
  } else {
    if (CPG_nonlinear_discrete_DWork.Rayleigh_MODE) {
      /* Disable for Outport: '<S1>/x_1' */
      CPG_nonlinear_discrete_B.x_f = 0.0;

      /* Disable for Outport: '<S1>/z_1' */
      CPG_nonlinear_discrete_B.x_dot_d = 0.0;

      /* Disable for Outport: '<S1>/x_2' */
      CPG_nonlinear_discrete_B.x_k = 0.0;

      /* Disable for Outport: '<S1>/z_2' */
      CPG_nonlinear_discrete_B.x_dot_m = 0.0;
      CPG_nonlinear_discrete_DWork.Rayleigh_MODE = FALSE;
    }
  }

  /* End of Outputs for SubSystem: '<Root>/Rayleigh' */

  /* Outputs for Enabled SubSystem: '<Root>/Van der Pol' incorporates:
   *  EnablePort: '<S2>/Enable'
   */
  if (CPG_nonlinear_discrete_B.enable_vdp > 0) {
    if (!CPG_nonlinear_discrete_DWork.VanderPol_MODE) {
      /* InitializeConditions for DiscreteIntegrator: '<S6>/Integrator' */
      CPG_nonlinear_discrete_DWork.Integrator_IC_LOADING = 1U;

      /* InitializeConditions for DiscreteIntegrator: '<S6>/Integrator1' */
      CPG_nonlinear_discrete_DWork.Integrator1_IC_LOADING = 1U;

      /* InitializeConditions for DiscreteIntegrator: '<S7>/Integrator' */
      CPG_nonlinear_discrete_DWork.Integrator_IC_LOADING_l = 1U;

      /* InitializeConditions for DiscreteIntegrator: '<S7>/Integrator1' */
      CPG_nonlinear_discrete_DWork.Integrator1_IC_LOADING_l = 1U;
      CPG_nonlinear_discrete_DWork.VanderPol_MODE = TRUE;
    }

    /* DiscreteIntegrator: '<S6>/Integrator' */
```

```
if (CPG_nonlinear_discrete_DWork.Integrator_IC_LOADING != 0) {
  CPG_nonlinear_discrete_DWork.Integrator_DSTATE =
    CPG_nonlinear_discrete_B.vdp_x0[1];
}

CPG_nonlinear_discrete_B.x_dot =
  CPG_nonlinear_discrete_DWork.Integrator_DSTATE;

/* End of DiscreteIntegrator: '<S6>/Integrator' */

/* DiscreteIntegrator: '<S6>/Integrator1' */
if (CPG_nonlinear_discrete_DWork.Integrator1_IC_LOADING != 0) {
  CPG_nonlinear_discrete_DWork.Integrator1_DSTATE =
    CPG_nonlinear_discrete_B.vdp_x0[0];
}

CPG_nonlinear_discrete_B.x = CPG_nonlinear_discrete_DWork.Integrator1_DSTATE;

/* End of DiscreteIntegrator: '<S6>/Integrator1' */

/* Switch: '<S6>/Switch' incorporates:
 *  Constant: '<S6>/x1'
 *  Gain: '<S6>/-mi'
 *  Gain: '<S6>/mi'
 *  Math: '<S6>/Math Function'
 *  Product: '<S6>/Product'
 *  Sum: '<S6>/Sum'
 *
 * About '<S6>/Math Function':
 *  Operator: magnitude^2
 */
if (CPG_nonlinear_discrete_B.dump >= 0) {
  rtb_Switch_j = (1.0 - CPG_nonlinear_discrete_B.x *
                  CPG_nonlinear_discrete_B.x) *
    CPG_nonlinear_discrete_B.x_dot * 10.0;
} else {
  rtb_Switch_j = (-10.0) * CPG_nonlinear_discrete_B.x_dot;
}

/* End of Switch: '<S6>/Switch' */

/* Sum: '<S6>/Sum1' incorporates:
 *  Gain: '<S6>/frequency'
 */
rtb_x_dot_dot_d = rtb_Switch_j - 50.0 * CPG_nonlinear_discrete_B.x;

/* DiscreteIntegrator: '<S7>/Integrator' */
if (CPG_nonlinear_discrete_DWork.Integrator_IC_LOADING_l != 0) {
  CPG_nonlinear_discrete_DWork.Integrator_DSTATE_o =
    CPG_nonlinear_discrete_B.vdp_x0[3];
}

CPG_nonlinear_discrete_B.x_dot_g =
  CPG_nonlinear_discrete_DWork.Integrator_DSTATE_o;

/* End of DiscreteIntegrator: '<S7>/Integrator' */

/* DiscreteIntegrator: '<S7>/Integrator1' */
if (CPG_nonlinear_discrete_DWork.Integrator1_IC_LOADING_l != 0) {
  CPG_nonlinear_discrete_DWork.Integrator1_DSTATE_k =
    CPG_nonlinear_discrete_B.vdp_x0[2];
}

CPG_nonlinear_discrete_B.x_p =
  CPG_nonlinear_discrete_DWork.Integrator1_DSTATE_k;

/* End of DiscreteIntegrator: '<S7>/Integrator1' */

/* Switch: '<S7>/Switch' incorporates:
 *  Constant: '<S7>/x1'
 *  Gain: '<S7>/-mi'
```

```
     *  Gain: '<S7>/mi'
     *  Math: '<S7>/Math Function'
     *  Product: '<S7>/Product'
     *  Sum: '<S7>/Sum'
     *
     * About '<S7>/Math Function':
     *  Operator: magnitude^2
     */
    if (CPG_nonlinear_discrete_B.dump >= 0) {
      rtb_Switch_j = (1.0 - CPG_nonlinear_discrete_B.x_p *
                      CPG_nonlinear_discrete_B.x_p) *
        CPG_nonlinear_discrete_B.x_dot_g * 10.0;
    } else {
      rtb_Switch_j = (-10.0) * CPG_nonlinear_discrete_B.x_dot_g;
    }

    /* End of Switch: '<S7>/Switch' */

    /* Update for DiscreteIntegrator: '<S6>/Integrator' */
    CPG_nonlinear_discrete_DWork.Integrator_IC_LOADING = 0U;
    CPG_nonlinear_discrete_DWork.Integrator_DSTATE += 0.01 * rtb_x_dot_dot_d;

    /* Update for DiscreteIntegrator: '<S6>/Integrator1' */
    CPG_nonlinear_discrete_DWork.Integrator1_IC_LOADING = 0U;
    CPG_nonlinear_discrete_DWork.Integrator1_DSTATE += 0.01 *
      CPG_nonlinear_discrete_B.x_dot;

    /* Update for DiscreteIntegrator: '<S7>/Integrator' incorporates:
     *  Gain: '<S7>/frequency'
     *  Sum: '<S7>/Sum1'
     */
    CPG_nonlinear_discrete_DWork.Integrator_IC_LOADING_l = 0U;
    CPG_nonlinear_discrete_DWork.Integrator_DSTATE_o += (rtb_Switch_j - 50.0 *
      CPG_nonlinear_discrete_B.x_p) * 0.01;

    /* Update for DiscreteIntegrator: '<S7>/Integrator1' */
    CPG_nonlinear_discrete_DWork.Integrator1_IC_LOADING_l = 0U;
    CPG_nonlinear_discrete_DWork.Integrator1_DSTATE_k += 0.01 *
      CPG_nonlinear_discrete_B.x_dot_g;
  } else {
    if (CPG_nonlinear_discrete_DWork.VanderPol_MODE) {
      /* Disable for Outport: '<S2>/x_1' */
      CPG_nonlinear_discrete_B.x = 0.0;

      /* Disable for Outport: '<S2>/z_1' */
      CPG_nonlinear_discrete_B.x_dot = 0.0;

      /* Disable for Outport: '<S2>/x_2' */
      CPG_nonlinear_discrete_B.x_p = 0.0;

      /* Disable for Outport: '<S2>/z_2' */
      CPG_nonlinear_discrete_B.x_dot_g = 0.0;
      CPG_nonlinear_discrete_DWork.VanderPol_MODE = FALSE;
    }
  }

  /* End of Outputs for SubSystem: '<Root>/Van der Pol' */

  /* Update for UnitDelay: '<Root>/Unit Delay' */
  CPG_nonlinear_discrete_DWork.UnitDelay_DSTATE[0] = CPG_nonlinear_discrete_B.x;
  CPG_nonlinear_discrete_DWork.UnitDelay_DSTATE[1] =
    CPG_nonlinear_discrete_B.x_dot;
  CPG_nonlinear_discrete_DWork.UnitDelay_DSTATE[2] =
    CPG_nonlinear_discrete_B.x_p;
  CPG_nonlinear_discrete_DWork.UnitDelay_DSTATE[3] =
    CPG_nonlinear_discrete_B.x_dot_g;

  /* Update for UnitDelay: '<Root>/Unit Delay1' */
  CPG_nonlinear_discrete_DWork.UnitDelay1_DSTATE[0] =
    CPG_nonlinear_discrete_B.x_f;
  CPG_nonlinear_discrete_DWork.UnitDelay1_DSTATE[1] =
```

```
    CPG_nonlinear_discrete_B.x_dot_d;
  CPG_nonlinear_discrete_DWork.UnitDelay1_DSTATE[2] =
    CPG_nonlinear_discrete_B.x_k;
  CPG_nonlinear_discrete_DWork.UnitDelay1_DSTATE[3] =
    CPG_nonlinear_discrete_B.x_dot_m;
}

/* Model initialize function */
void CPG_nonlinear_init(void)
{
  /* Registration code */

  /* initialize error status */
  rtmSetErrorStatus(CPG_nonlinear_discrete_M, (NULL));

  /* block I/O */
  (void) memset(((void *) &CPG_nonlinear_discrete_B), 0,
                sizeof(BlockIO_CPG_nonlinear_discrete));

  /* states (dwork) */
  (void) memset((void *)&CPG_nonlinear_discrete_DWork, 0,
                sizeof(D_Work_CPG_nonlinear_discrete));

  /* Start for Enabled SubSystem: '<Root>/Rayleigh' */
  /* InitializeConditions for DiscreteIntegrator: '<S4>/Integrator' */
  CPG_nonlinear_discrete_DWork.Integrator_IC_LOADING_i = 1U;

  /* InitializeConditions for DiscreteIntegrator: '<S4>/Integrator1' */
  CPG_nonlinear_discrete_DWork.Integrator1_IC_LOADING_e = 1U;

  /* InitializeConditions for DiscreteIntegrator: '<S5>/Integrator' */
  CPG_nonlinear_discrete_DWork.Integrator_IC_LOADING_h = 1U;

  /* InitializeConditions for DiscreteIntegrator: '<S5>/Integrator1' */
  CPG_nonlinear_discrete_DWork.Integrator1_IC_LOADING_c = 1U;

  /* End of Start for SubSystem: '<Root>/Rayleigh' */

  /* Start for Enabled SubSystem: '<Root>/Van der Pol' */
  /* InitializeConditions for DiscreteIntegrator: '<S6>/Integrator' */
  CPG_nonlinear_discrete_DWork.Integrator_IC_LOADING = 1U;

  /* InitializeConditions for DiscreteIntegrator: '<S6>/Integrator1' */
  CPG_nonlinear_discrete_DWork.Integrator1_IC_LOADING = 1U;

  /* InitializeConditions for DiscreteIntegrator: '<S7>/Integrator' */
  CPG_nonlinear_discrete_DWork.Integrator_IC_LOADING_l = 1U;

  /* InitializeConditions for DiscreteIntegrator: '<S7>/Integrator1' */
  CPG_nonlinear_discrete_DWork.Integrator1_IC_LOADING_l = 1U;

  /* End of Start for SubSystem: '<Root>/Van der Pol' */

  /* InitializeConditions for Chart: '<Root>/control_logic' */
  CPG_nonlinear_discrete_DWork.is_walk = 0U;
  CPG_nonlinear_discrete_DWork.is_active_c1_CPG_nonlinear_disc = 0U;
  CPG_nonlinear_discrete_DWork.is_c1_CPG_nonlinear_discrete = 0U;
  CPG_nonlinear_discrete_DWork.tf = 0;
  CPG_nonlinear_discrete_B.enable_vdp = 0;
  CPG_nonlinear_discrete_B.enable_ray = 0;
  CPG_nonlinear_discrete_B.vdp_x0[0] = 0.0;
  CPG_nonlinear_discrete_B.vdp_x0[1] = 0.1;
  CPG_nonlinear_discrete_B.vdp_x0[2] = 0.0;
  CPG_nonlinear_discrete_B.vdp_x0[3] = -0.1;
  CPG_nonlinear_discrete_B.dump = 0;
  CPG_nonlinear_discrete_B.ray_x0[0] = 0.0;
  CPG_nonlinear_discrete_B.ray_x0[1] = 0.1;
  CPG_nonlinear_discrete_B.ray_x0[2] = 0.0;
  CPG_nonlinear_discrete_B.ray_x0[3] = -0.1;
}
```

```
/* Model terminate function */
void CPG_nonlinear_discrete_terminate(void)
{
  /* (no terminate code required) */
}
```

Kod źródłowy algorytmu wykorzystującego kratę Toda-Rayleigha:

```
/* Named constants for Chart: '<Root>/control_logic' */
#define CPG_TR_discr_IN_NO_ACTIVE_CHILD ((uint8_T)0U)
#define CPG_TR_discrete_IN_change_mode ((uint8_T)1U)
#define CPG_TR_discrete_IN_dump        ((uint8_T)1U)
#define CPG_TR_discrete_IN_proceed_walk ((uint8_T)2U)
#define CPG_TR_discrete_IN_stop        ((uint8_T)1U)
#define CPG_TR_discrete_IN_wait        ((uint8_T)2U)
#define CPG_TR_discrete_IN_walk        ((uint8_T)2U)

/* Block signals (auto storage) */
BlockIO_CPG_TR_discrete CPG_TR_discrete_B;

/* Block states (auto storage) */
D_Work_CPG_TR_discrete CPG_TR_discrete_DWork;

/* External inputs (root inport signals with auto storage) */
ExternalInputs_CPG_TR_discrete CPG_TR_discrete_U;

/* Real-time model */
RT_MODEL_CPG_TR_discrete CPG_TR_discrete_M_;
RT_MODEL_CPG_TR_discrete *const CPG_TR_discrete_M = &CPG_TR_discrete_M_;

/* Forward declaration for local functions */
static int32_T CPG_TR_discrete_mode_change(int32_T m);
static int32_T CPG_TR_discrete_test_state(const real_T s[12], const real_T cp[6]);
static int32_T CPG_TR_discrete_test_dump(const real_T s[12]);

/* Function for Chart: '<Root>/control_logic' */
static int32_T CPG_TR_discrete_mode_change(int32_T m)
{
  int32_T tf;

  /* MATLAB Function 'mode_change': '<S2>:258' */
  /*  m - walk mode */
  if (!CPG_TR_discrete_DWork.old_mode_not_empty) {
    /* '<S2>:258:6' */
    /* '<S2>:258:7' */
    CPG_TR_discrete_DWork.old_mode = CPG_TR_discrete_U.mode_p;
    CPG_TR_discrete_DWork.old_mode_not_empty = TRUE;
  }

  if (m != CPG_TR_discrete_DWork.old_mode) {
    /* '<S2>:258:10' */
    /* '<S2>:258:11' */
    tf = 1;

    /* '<S2>:258:12' */
    CPG_TR_discrete_DWork.old_mode = m;
  } else {
    /* '<S2>:258:14' */
    tf = 0;
  }

  return tf;
}

/* Function for Chart: '<Root>/control_logic' */
static int32_T CPG_TR_discrete_test_state(const real_T s[12], const real_T cp[6])
```

```
{
  int32_T tf;
  real_T diff;
  real_T y[12];
  real_T x[12];
  int32_T i;

  /* MATLAB Function 'test_state': '<S2>:211' */
  /*  s - current legs position */
  /*  cp - current change points */
  /* '<S2>:211:6' */
  /* '<S2>:211:7' */
  /* '<S2>:211:9' */
  for (i = 0; i < 6; i++) {
    x[i] = s[i] - cp[i];
  }

  x[6] = s[6];
  x[7] = s[7];
  x[8] = s[8];
  x[9] = s[9];
  x[10] = s[10];
  x[11] = s[11];
  for (i = 0; i < 12; i++) {
    y[i] = fabs(x[i]);
  }

  diff = y[0];
  for (i = 0; i < 11; i++) {
    diff += y[i + 1];
  }

  if (diff > 1.0) {
    /* '<S2>:211:11' */
    /* '<S2>:211:12' */
    tf = 0;
  } else {
    /* '<S2>:211:14' */
    tf = 1;
  }

  return tf;
}

/* Function for Chart: '<Root>/control_logic' */
static int32_T CPG_TR_discrete_test_dump(const real_T s[12])
{
  int32_T tf;
  real_T diff;
  real_T y[12];
  int32_T i;

  /* MATLAB Function 'test_dump': '<S2>:262' */
  /*  s - current legs position */
  /* '<S2>:262:5' */
  /* '<S2>:262:6' */
  /* '<S2>:262:8' */
  for (i = 0; i < 12; i++) {
    y[i] = fabs(s[i]);
  }

  diff = y[0];
  for (i = 0; i < 11; i++) {
    diff += y[i + 1];
  }

  if (diff > 0.1) {
    /* '<S2>:262:10' */
    /* '<S2>:262:11' */
    tf = 0;
  } else {
```

```
    /* '<S2>:262:13' */
    tf = 1;
  }

  return tf;
}

/* Model step function */
void CPG_TR_step(int32_T walk_enable, int32_T mode, real_T control_singal[12])
{
  int32_T rtb_Switch;
  real_T rtb_Sum5;
  real_T rtb_x_bis;
  real_T rtb_x_bis_a;
  real_T rtb_x_bis_h;
  real_T rtb_x_bis_l;
  real_T rtb_x_bis_n;
  int32_T i;
  int32_T i_0;

  /*Copy value for root inport '/walk_enable' since it is accessed globally */
  CPG_TR_discrete_U.walk_enable_h = walk_enable;

  /* Copy value for root inport '<Root>/mode' since it is accessed globally */
  CPG_TR_discrete_U.mode_p = mode;

  /* Outport: '<Root>/control_singal' incorporates:
   *  UnitDelay: '<Root>/Unit Delay2'
   */
  memcpy(&control_singal[0], &CPG_TR_discrete_DWork.UnitDelay2_DSTATE[0], 12U *
         sizeof(real_T));

  /* Switch: '<S3>/Switch' incorporates:
   *  Constant: '<S3>/Constant'
   *  Inport: '<Root>/mode'
   *  Inport: '<Root>/walk_enable'
   */
  if (CPG_TR_discrete_U.walk_enable_h != 0) {
    rtb_Switch = CPG_TR_discrete_U.mode_p;
  } else {
    rtb_Switch = 0;
  }

  /* End of Switch: '<S3>/Switch' */

  /* LookupNDDirect: '<S3>/change_points_table'
   *
   * About '<S3>/change_points_table':
   *  2-dimensional Direct Look-Up returning a Column
   */
  if (rtb_Switch >= 3) {
    i = 3;
  } else if (rtb_Switch <= 0) {
    i = 0;
  } else {
    i = rtb_Switch;
  }

  i = (int32_T)((uint32_T)i * 6U);
  for (i_0 = 0; i_0 < 6; i_0++) {
    CPG_TR_discrete_B.change_points_table[i_0] =
      CPG_TR_discrete_ConstP.change_points_table_[(uint32_T)i + (uint32_T)i_0];
  }

  /* End of LookupNDDirect: '<S3>/change_points_table' */

  /* LookupNDDirect: '<S3>/start_points_table'
   *
   * About '<S3>/start_points_table':
   *  2-dimensional Direct Look-Up returning a Column
   */
```

```
if (rtb_Switch >= 3) {
  i = 3;
} else if (rtb_Switch <= 0) {
  i = 0;
} else {
  i = rtb_Switch;
}

i = (int32_T)((uint32_T)i * 6U);
for (i_0 = 0; i_0 < 6; i_0++) {
  CPG_TR_discrete_B.start_points_table[i_0] =
    CPG_TR_discrete_ConstP.start_points_table_t[(uint32_T)i + (uint32_T)i_0];
}

/* End of LookupNDDirect: '<S3>/start_points_table' */

/* LookupNDDirect: '<S3>/ctrl_value_table'
 *
 * About '<S3>/ctrl_value_table':
 *  1-dimensional Direct Look-Up returning a Scalar
 */
if (rtb_Switch >= 3) {
  rtb_Switch = 3;
} else {
  if (rtb_Switch <= 0) {
    rtb_Switch = 0;
  }
}

CPG_TR_discrete_B.ctrl_value_table =
  CPG_TR_discrete_ConstP.ctrl_value_table_tabl[(uint32_T)rtb_Switch];

/* End of LookupNDDirect: '<S3>/ctrl_value_table' */

/* Chart: '<Root>/control_logic' incorporates:
 *  Inport: '<Root>/mode'
 *  Inport: '<Root>/walk_enable'
 */
/* Gateway: control_logic */
/* During: control_logic */
if (CPG_TR_discrete_DWork.is_active_c1_CPG_TR_discrete == 0) {
  /* Entry: control_logic */
  CPG_TR_discrete_DWork.is_active_c1_CPG_TR_discrete = 1U;

  /* Entry Internal: control_logic */
  /* Transition: '<S2>:250' */
  CPG_TR_discrete_DWork.is_c1_CPG_TR_discrete = CPG_TR_discrete_IN_stop;

  /* Entry Internal 'stop': '<S2>:212' */
  /* Transition: '<S2>:251' */
  CPG_TR_discrete_DWork.is_stop = CPG_TR_discrete_IN_wait;

  /* Entry 'wait': '<S2>:248' */
  CPG_TR_discrete_B.enable = 0;
} else if (CPG_TR_discrete_DWork.is_c1_CPG_TR_discrete ==
           CPG_TR_discrete_IN_stop) {
  /* During 'stop': '<S2>:212' */
  if (CPG_TR_discrete_U.walk_enable_h == 1) {
    /* Transition: '<S2>:220' */
    for (i = 0; i < 6; i++) {
      CPG_TR_discrete_B.p_z[i] = CPG_TR_discrete_B.start_points_table[i];
    }

    CPG_TR_discrete_B.ctrl = CPG_TR_discrete_B.ctrl_value_table;

    /* Exit Internal 'stop': '<S2>:212' */
    CPG_TR_discrete_DWork.is_stop = CPG_TR_discr_IN_NO_ACTIVE_CHILD;
    CPG_TR_discrete_DWork.is_c1_CPG_TR_discrete = CPG_TR_discrete_IN_walk;

    /* Entry 'walk': '<S2>:213' */
    CPG_TR_discrete_B.enable = 1;
```

```
    /* Entry Internal 'walk': '<S2>:213' */
    /* Transition: '<S2>:241' */
    CPG_TR_discrete_DWork.is_walk = CPG_TR_discrete_IN_proceed_walk;

    /* Entry 'proceed_walk': '<S2>:222' */
    CPG_TR_discrete_B.dump = 0;
  } else {
    if ((CPG_TR_discrete_DWork.is_stop == CPG_TR_discrete_IN_dump) &&
        (CPG_TR_discrete_test_dump(control_singal) != 0)) {
      /* During 'dump': '<S2>:247' */
      /* Transition: '<S2>:252' */
      CPG_TR_discrete_DWork.is_stop = CPG_TR_discrete_IN_wait;

      /* Entry 'wait': '<S2>:248' */
      CPG_TR_discrete_B.enable = 0;
    }
  }
} else {
  /* During 'walk': '<S2>:213' */
  if (CPG_TR_discrete_U.walk_enable_h == 0) {
    /* Transition: '<S2>:221' */
    /* Exit Internal 'walk': '<S2>:213' */
    CPG_TR_discrete_DWork.is_walk = CPG_TR_discr_IN_NO_ACTIVE_CHILD;
    CPG_TR_discrete_DWork.is_c1_CPG_TR_discrete = CPG_TR_discrete_IN_stop;
    CPG_TR_discrete_DWork.is_stop = CPG_TR_discrete_IN_dump;

    /* Entry 'dump': '<S2>:247' */
    for (i = 0; i < 6; i++) {
      CPG_TR_discrete_B.p_x[i] = CPG_TR_discrete_B.start_points_table[i];
    }

    CPG_TR_discrete_B.dump = 1;
  } else if (CPG_TR_discrete_DWork.is_walk == CPG_TR_discrete_IN_change_mode)
  {
    /* During 'change_mode': '<S2>:223' */
    if (CPG_TR_discrete_test_state(control_singal,
         CPG_TR_discrete_B.change_points_table) != 0) {
      /* Transition: '<S2>:245' */
      CPG_TR_discrete_DWork.is_walk = CPG_TR_discrete_IN_proceed_walk;

      /* Entry 'proceed_walk': '<S2>:222' */
      CPG_TR_discrete_B.dump = 0;
    }
  } else {
    /* During 'proceed_walk': '<S2>:222' */
    if (CPG_TR_discrete_mode_change(CPG_TR_discrete_U.mode_p) != 0) {
      /* Transition: '<S2>:242' */
      CPG_TR_discrete_DWork.is_walk = CPG_TR_discrete_IN_change_mode;

      /* Entry 'change_mode': '<S2>:223' */
      for (i = 0; i < 6; i++) {
        CPG_TR_discrete_B.p_x[i] = CPG_TR_discrete_B.change_points_table[i];
      }

      CPG_TR_discrete_B.ctrl = CPG_TR_discrete_B.ctrl_value_table;
      CPG_TR_discrete_B.dump = 1;
    }
  }
}

/* End of Chart: '<Root>/control_logic' */

/* Outputs for Enabled SubSystem: 'Discrete Toda Rayleigh Ring' incorporates:
 *  EnablePort: '<S1>/Enable'
 */
if (CPG_TR_discrete_B.enable > 0) {
  if (!CPG_TR_discrete_DWork.DiscreteTodaRayleighRing_MODE) {
    /*InitializeConditions for DiscreteIntegrator: '<S4>/Discrete Integrator'*/
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LOADI = 1U;
```

```
    /*InitializeConditions for DiscreteIntegrator: '<S4>/Discrete Integrator'*/
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTATE =
      CPG_TR_discrete_ConstB.x0;

    /*InitializeConditions for DiscreteIntegrator: '<S9>/Discrete Integrator'*/
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTAT_p =
      CPG_TR_discrete_ConstB.x0_d;

    /*InitializeConditions for DiscreteIntegrator: '<S5>/Discrete Integrator'*/
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTAT_b =
      CPG_TR_discrete_ConstB.x0_e;

    /*InitializeConditions for DiscreteIntegrator: '<S5>/Discrete Integrator'*/
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LOA_g = 1U;

    /*InitializeConditions for DiscreteIntegrator: '<S6>/Discrete Integrator'*/
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTAT_c =
      CPG_TR_discrete_ConstB.x0_m;

    /*InitializeConditions for DiscreteIntegrator: '<S6>/Discrete Integrator'*/
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LOA_d = 1U;

    /*InitializeConditions for DiscreteIntegrator: '<S7>/Discrete Integrator'*/
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTAT_l =
      CPG_TR_discrete_ConstB.x0_n;

    /*InitializeConditions for DiscreteIntegrator: '<S7>/Discrete Integrator'*/
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LOA_e = 1U;

    /*InitializeConditions for DiscreteIntegrator: '<S8>/Discrete Integrator'*/
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTA_bn =
      CPG_TR_discrete_ConstB.x0_h;

    /*InitializeConditions for DiscreteIntegrator: '<S8>/Discrete Integrator'*/
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LOA_p = 1U;

    /*InitializeConditions for DiscreteIntegrator: '<S9>/Discrete Integrator'*/
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LO_dh = 1U;
    CPG_TR_discrete_DWork.DiscreteTodaRayleighRing_MODE = TRUE;
  }

  /* DiscreteIntegrator: '<S4>/Discrete-Time Integrator' */
  if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LOADI != 0) {
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE =
      CPG_TR_discrete_B.p_z[0];
    if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE >= 5.0) {
      CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE = 5.0;
    } else {
      if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE <= (-5.0)) {
        CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE = (-5.0);
      }
    }
  }

  if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE >= 5.0) {
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE = 5.0;
  } else {
    if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE <= (-5.0)) {
      CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE = (-5.0);
    }
  }

  CPG_TR_discrete_B.x_p = CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE;

  /* End of DiscreteIntegrator: '<S4>/Discrete-Time Integrator' */

  /* DiscreteIntegrator: '<S4>/Discrete-Time Integrator1' */
  CPG_TR_discrete_B.x = CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTATE;

  /* Sum: '<S10>/Sum5' */
  rtb_Sum5 = CPG_TR_discrete_B.x_p - CPG_TR_discrete_B.ctrl;
```

```
/* DiscreteIntegrator: '<S9>/Discrete-Time Integrator1' */
CPG_TR_discrete_B.x_l =
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTAT_p;

/* DiscreteIntegrator: '<S5>/Discrete-Time Integrator1' */
CPG_TR_discrete_B.x_o =
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTAT_b;

/* Switch: '<S4>/Switch' incorporates:
 *  Bias: '<S10>/mi'
 *  Gain: '<S10>/frequency'
 *  Gain: '<S10>/omega^1'
 *  Gain: '<S10>/sliding'
 *  Gain: '<S11>/dumping'
 *  Gain: '<S11>/frequency'
 *  Math: '<S10>/Math Function'
 *  Math: '<S10>/Math Function2'
 *  Math: '<S10>/Math Function3'
 *  Product: '<S10>/Product'
 *  Sum: '<S10>/Sum1'
 *  Sum: '<S10>/Sum2'
 *  Sum: '<S10>/Sum3'
 *  Sum: '<S10>/Sum4'
 *  Sum: '<S11>/Sum6'
 *
 * About '<S10>/Math Function':
 *  Operator: magnitude^2
 *
 * About '<S10>/Math Function2':
 *  Operator: exp
 *
 * About '<S10>/Math Function3':
 *  Operator: exp
 */
if (CPG_TR_discrete_B.dump != 0) {
  rtb_x_bis = (CPG_TR_discrete_B.p_x[0] - 2.0 * CPG_TR_discrete_B.x_p) - 1.0
    * CPG_TR_discrete_B.x;
} else {
  rtb_x_bis = ((rtb_Sum5 * rtb_Sum5 * (-1.0) + 0.8) * rtb_Sum5 + (exp
    (CPG_TR_discrete_B.x_l - CPG_TR_discrete_B.x) - exp(CPG_TR_discrete_B.x
    - CPG_TR_discrete_B.x_o)) * 1.0) - 0.1 * CPG_TR_discrete_B.x;
}

/* End of Switch: '<S4>/Switch' */

/* DiscreteIntegrator: '<S5>/Discrete-Time Integrator' */
if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LOA_g != 0) {
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_k =
    CPG_TR_discrete_B.p_z[1];
  if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_k >= 5.0) {
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_k = 5.0;
  } else {
    if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_k <= (-5.0)) {
      CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_k = (-5.0);
    }
  }
}

if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_k >= 5.0) {
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_k = 5.0;
} else {
  if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_k <= (-5.0)) {
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_k = (-5.0);
  }
}

CPG_TR_discrete_B.x_p_p =
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_k;

/* End of DiscreteIntegrator: '<S5>/Discrete-Time Integrator' */
```

```
/* Sum: '<S12>/Sum5' */
rtb_Sum5 = CPG_TR_discrete_B.x_p_p - CPG_TR_discrete_B.ctrl;

/* DiscreteIntegrator: '<S6>/Discrete-Time Integrator1' */
CPG_TR_discrete_B.x_d =
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTAT_c;

/* Switch: '<S5>/Switch' incorporates:
 *  Bias: '<S12>/mi'
 *  Gain: '<S12>/frequency'
 *  Gain: '<S12>/omega^1'
 *  Gain: '<S12>/sliding'
 *  Gain: '<S13>/dumping'
 *  Gain: '<S13>/frequency'
 *  Math: '<S12>/Math Function'
 *  Math: '<S12>/Math Function2'
 *  Math: '<S12>/Math Function3'
 *  Product: '<S12>/Product'
 *  Sum: '<S12>/Sum1'
 *  Sum: '<S12>/Sum2'
 *  Sum: '<S12>/Sum3'
 *  Sum: '<S12>/Sum4'
 *  Sum: '<S13>/Sum6'
 *
 * About '<S12>/Math Function':
 *  Operator: magnitude^2
 *
 * About '<S12>/Math Function2':
 *  Operator: exp
 *
 * About '<S12>/Math Function3':
 *  Operator: exp
 */
if (CPG_TR_discrete_B.dump != 0) {
  rtb_x_bis_a = (CPG_TR_discrete_B.p_x[1] - 2.0 * CPG_TR_discrete_B.x_p_p) -
    1.0 * CPG_TR_discrete_B.x_o;
} else {
  rtb_x_bis_a = ((rtb_Sum5 * rtb_Sum5 * (-1.0) + 0.8) * rtb_Sum5 + (exp
    (CPG_TR_discrete_B.x - CPG_TR_discrete_B.x_o) - exp
    (CPG_TR_discrete_B.x_o - CPG_TR_discrete_B.x_d)) * 1.0) - 0.1 *
    CPG_TR_discrete_B.x_o;
}

/* End of Switch: '<S5>/Switch' */

/* DiscreteIntegrator: '<S6>/Discrete-Time Integrator' */
if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LOA_d != 0) {
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_d =
    CPG_TR_discrete_B.p_z[2];
  if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_d >= 5.0) {
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_d = 5.0;
  } else {
    if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_d <= (-5.0)) {
      CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_d = (-5.0);
    }
  }
}

if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_d >= 5.0) {
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_d = 5.0;
} else {
  if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_d <= (-5.0)) {
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_d = (-5.0);
  }
}

CPG_TR_discrete_B.x_p_i =
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_d;

/* End of DiscreteIntegrator: '<S6>/Discrete-Time Integrator' */
```

```
/* Sum: '<S14>/Sum5' */
rtb_Sum5 = CPG_TR_discrete_B.x_p_i - CPG_TR_discrete_B.ctrl;

/* DiscreteIntegrator: '<S7>/Discrete-Time Integrator1' */
CPG_TR_discrete_B.x_b =
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTAT_l;

/* Switch: '<S6>/Switch' incorporates:
 *  Bias: '<S14>/mi'
 *  Gain: '<S14>/frequency'
 *  Gain: '<S14>/omega^1'
 *  Gain: '<S14>/sliding'
 *  Gain: '<S15>/dumping'
 *  Gain: '<S15>/frequency'
 *  Math: '<S14>/Math Function'
 *  Math: '<S14>/Math Function2'
 *  Math: '<S14>/Math Function3'
 *  Product: '<S14>/Product'
 *  Sum: '<S14>/Sum1'
 *  Sum: '<S14>/Sum2'
 *  Sum: '<S14>/Sum3'
 *  Sum: '<S14>/Sum4'
 *  Sum: '<S15>/Sum6'
 *
 * About '<S14>/Math Function':
 *  Operator: magnitude^2
 *
 * About '<S14>/Math Function2':
 *  Operator: exp
 *
 * About '<S14>/Math Function3':
 *  Operator: exp
 */
if (CPG_TR_discrete_B.dump != 0) {
  rtb_x_bis_h = (CPG_TR_discrete_B.p_x[2] - 2.0 * CPG_TR_discrete_B.x_p_i) -
    1.0 * CPG_TR_discrete_B.x_d;
} else {
  rtb_x_bis_h = ((rtb_Sum5 * rtb_Sum5 * (-1.0) + 0.8) * rtb_Sum5 + (exp
    (CPG_TR_discrete_B.x_o - CPG_TR_discrete_B.x_d) - exp
    (CPG_TR_discrete_B.x_d - CPG_TR_discrete_B.x_b)) * 1.0) - 0.1 *
    CPG_TR_discrete_B.x_d;
}

/* End of Switch: '<S6>/Switch' */

/* DiscreteIntegrator: '<S7>/Discrete-Time Integrator' */
if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LOA_e != 0) {
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_b =
    CPG_TR_discrete_B.p_z[3];
  if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_b >= 5.0) {
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_b = 5.0;
  } else {
    if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_b <= (-5.0)) {
      CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_b = (-5.0);
    }
  }
}

if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_b >= 5.0) {
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_b = 5.0;
} else {
  if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_b <= (-5.0)) {
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_b = (-5.0);
  }
}

CPG_TR_discrete_B.x_p_l =
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_b;

/* End of DiscreteIntegrator: '<S7>/Discrete-Time Integrator' */
```

```
/* Sum: '<S16>/Sum5' */
rtb_Sum5 = CPG_TR_discrete_B.x_p_l - CPG_TR_discrete_B.ctrl;

/* DiscreteIntegrator: '<S8>/Discrete-Time Integrator1' */
CPG_TR_discrete_B.x_ln =
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTA_bn;

/* Switch: '<S7>/Switch' incorporates:
 *  Bias: '<S16>/mi'
 *  Gain: '<S16>/frequency'
 *  Gain: '<S16>/omega^1'
 *  Gain: '<S16>/sliding'
 *  Gain: '<S17>/dumping'
 *  Gain: '<S17>/frequency'
 *  Math: '<S16>/Math Function'
 *  Math: '<S16>/Math Function2'
 *  Math: '<S16>/Math Function3'
 *  Product: '<S16>/Product'
 *  Sum: '<S16>/Sum1'
 *  Sum: '<S16>/Sum2'
 *  Sum: '<S16>/Sum3'
 *  Sum: '<S16>/Sum4'
 *  Sum: '<S17>/Sum6'
 *
 * About '<S16>/Math Function':
 *  Operator: magnitude^2
 *
 * About '<S16>/Math Function2':
 *  Operator: exp
 *
 * About '<S16>/Math Function3':
 *  Operator: exp
 */
if (CPG_TR_discrete_B.dump != 0) {
  rtb_x_bis_l = (CPG_TR_discrete_B.p_x[3] - 2.0 * CPG_TR_discrete_B.x_p_l) -
    1.0 * CPG_TR_discrete_B.x_b;
} else {
  rtb_x_bis_l = ((rtb_Sum5 * rtb_Sum5 * (-1.0) + 0.8) * rtb_Sum5 + (exp
    (CPG_TR_discrete_B.x_d - CPG_TR_discrete_B.x_b) - exp
    (CPG_TR_discrete_B.x_b - CPG_TR_discrete_B.x_ln)) * 1.0) - 0.1 *
    CPG_TR_discrete_B.x_b;
}

/* End of Switch: '<S7>/Switch' */

/* DiscreteIntegrator: '<S8>/Discrete-Time Integrator' */
if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LOA_p != 0) {
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_g =
    CPG_TR_discrete_B.p_z[4];
  if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_g >= 5.0) {
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_g = 5.0;
  } else {
    if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_g <= (-5.0)) {
      CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_g = (-5.0);
    }
  }
}

if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_g >= 5.0) {
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_g = 5.0;
} else {
  if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_g <= (-5.0)) {
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_g = (-5.0);
  }
}

CPG_TR_discrete_B.x_p_h =
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_g;

/* End of DiscreteIntegrator: '<S8>/Discrete-Time Integrator' */
```

```
/* Sum: '<S18>/Sum5' */
rtb_Sum5 = CPG_TR_discrete_B.x_p_h - CPG_TR_discrete_B.ctrl;

/* Switch: '<S8>/Switch' incorporates:
 *  Bias: '<S18>/mi'
 *  Gain: '<S18>/frequency'
 *  Gain: '<S18>/omega^1'
 *  Gain: '<S18>/sliding'
 *  Gain: '<S19>/dumping'
 *  Gain: '<S19>/frequency'
 *  Math: '<S18>/Math Function'
 *  Math: '<S18>/Math Function2'
 *  Math: '<S18>/Math Function3'
 *  Product: '<S18>/Product'
 *  Sum: '<S18>/Sum1'
 *  Sum: '<S18>/Sum2'
 *  Sum: '<S18>/Sum3'
 *  Sum: '<S18>/Sum4'
 *  Sum: '<S19>/Sum6'
 *
 * About '<S18>/Math Function':
 *  Operator: magnitude^2
 *
 * About '<S18>/Math Function2':
 *  Operator: exp
 *
 * About '<S18>/Math Function3':
 *  Operator: exp
 */
if (CPG_TR_discrete_B.dump != 0) {
  rtb_x_bis_n = (CPG_TR_discrete_B.p_x[4] - 2.0 * CPG_TR_discrete_B.x_p_h) -
    1.0 * CPG_TR_discrete_B.x_ln;
} else {
  rtb_x_bis_n = ((rtb_Sum5 * rtb_Sum5 * (-1.0) + 0.8) * rtb_Sum5 + (exp
    (CPG_TR_discrete_B.x_b - CPG_TR_discrete_B.x_ln) - exp
    (CPG_TR_discrete_B.x_ln - CPG_TR_discrete_B.x_l)) * 1.0) - 0.1 *
    CPG_TR_discrete_B.x_ln;
}

/* End of Switch: '<S8>/Switch' */

/* DiscreteIntegrator: '<S9>/Discrete-Time Integrator' */
if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LO_dh != 0) {
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTAT_g1 =
    CPG_TR_discrete_B.p_z[5];
  if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTAT_g1 >= 5.0) {
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTAT_g1 = 5.0;
  } else {
    if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTAT_g1 <= (-5.0)) {
      CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTAT_g1 = (-5.0);
    }
  }
}

if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTAT_g1 >= 5.0) {
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTAT_g1 = 5.0;
} else {
  if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTAT_g1 <= (-5.0)) {
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTAT_g1 = (-5.0);
  }
}

CPG_TR_discrete_B.x_p_a =
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTAT_g1;

/* End of DiscreteIntegrator: '<S9>/Discrete-Time Integrator' */

/* Sum: '<S20>/Sum5' */
rtb_Sum5 = CPG_TR_discrete_B.x_p_a - CPG_TR_discrete_B.ctrl;
```

```
/* Switch: '<S9>/Switch' incorporates:
 *  Bias: '<S20>/mi'
 *  Gain: '<S20>/frequency'
 *  Gain: '<S20>/omega^1'
 *  Gain: '<S20>/sliding'
 *  Gain: '<S21>/dumping'
 *  Gain: '<S21>/frequency'
 *  Math: '<S20>/Math Function'
 *  Math: '<S20>/Math Function2'
 *  Math: '<S20>/Math Function3'
 *  Product: '<S20>/Product'
 *  Sum: '<S20>/Sum1'
 *  Sum: '<S20>/Sum2'
 *  Sum: '<S20>/Sum3'
 *  Sum: '<S20>/Sum4'
 *  Sum: '<S21>/Sum6'
 *
 * About '<S20>/Math Function':
 *  Operator: magnitude^2
 *
 * About '<S20>/Math Function2':
 *  Operator: exp
 *
 * About '<S20>/Math Function3':
 *  Operator: exp
 */
if (CPG_TR_discrete_B.dump != 0) {
  rtb_Sum5 = (CPG_TR_discrete_B.p_x[5] - 2.0 * CPG_TR_discrete_B.x_p_a) -
    1.0 * CPG_TR_discrete_B.x_l;
} else {
  rtb_Sum5 = ((rtb_Sum5 * rtb_Sum5 * (-1.0) + 0.8) * rtb_Sum5 + (exp
    (CPG_TR_discrete_B.x_ln - CPG_TR_discrete_B.x_l) - exp
    (CPG_TR_discrete_B.x_l - CPG_TR_discrete_B.x)) * 1.0) - 0.1 *
    CPG_TR_discrete_B.x_l;
}

/* End of Switch: '<S9>/Switch' */

/* Update for DiscreteIntegrator: '<S4>/Discrete-Time Integrator' */
CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LOADI = 0U;
CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE += 0.01 * rtb_x_bis;
if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE >= 5.0) {
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE = 5.0;
} else {
  if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE <= (-5.0)) {
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE = (-5.0);
  }
}

/* End of Update for DiscreteIntegrator: '<S4>/Discrete-Time Integrator' */

/* Update for DiscreteIntegrator: '<S4>/Discrete-Time Integrator1' */
CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTATE += 0.01 *
  CPG_TR_discrete_B.x_p;

/* Update for DiscreteIntegrator: '<S9>/Discrete-Time Integrator1' */
CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTAT_p += 0.01 *
  CPG_TR_discrete_B.x_p_a;

/* Update for DiscreteIntegrator: '<S5>/Discrete-Time Integrator1' */
CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTAT_b += 0.01 *
  CPG_TR_discrete_B.x_p_p;

/* Update for DiscreteIntegrator: '<S5>/Discrete-Time Integrator' */
CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LOA_g = 0U;
CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_k += 0.01 * rtb_x_bis_a;
if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_k >= 5.0) {
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_k = 5.0;
} else {
  if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_k <= (-5.0)) {
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_k = (-5.0);
```

```
      }
    }

    /* End of Update for DiscreteIntegrator: '<S5>/Discrete-Time Integrator' */

    /* Update for DiscreteIntegrator: '<S6>/Discrete-Time Integrator1' */
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTAT_c += 0.01 *
      CPG_TR_discrete_B.x_p_i;

    /* Update for DiscreteIntegrator: '<S6>/Discrete-Time Integrator' */
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LOA_d = 0U;
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_d += 0.01 * rtb_x_bis_h;
    if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_d >= 5.0) {
      CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_d = 5.0;
    } else {
      if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_d <= (-5.0)) {
        CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_d = (-5.0);
      }
    }

    /* End of Update for DiscreteIntegrator: '<S6>/Discrete-Time Integrator' */

    /* Update for DiscreteIntegrator: '<S7>/Discrete-Time Integrator1' */
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTAT_l += 0.01 *
      CPG_TR_discrete_B.x_p_l;

    /* Update for DiscreteIntegrator: '<S7>/Discrete-Time Integrator' */
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LOA_e = 0U;
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_b += 0.01 * rtb_x_bis_l;
    if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_b >= 5.0) {
      CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_b = 5.0;
    } else {
      if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_b <= (-5.0)) {
        CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_b = (-5.0);
      }
    }

    /* End of Update for DiscreteIntegrator: '<S7>/Discrete-Time Integrator' */

    /* Update for DiscreteIntegrator: '<S8>/Discrete-Time Integrator1' */
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTA_bn += 0.01 *
      CPG_TR_discrete_B.x_p_h;

    /* Update for DiscreteIntegrator: '<S8>/Discrete-Time Integrator' */
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LOA_p = 0U;
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_g += 0.01 * rtb_x_bis_n;
    if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_g >= 5.0) {
      CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_g = 5.0;
    } else {
      if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_g <= (-5.0)) {
        CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTATE_g = (-5.0);
      }
    }

    /* End of Update for DiscreteIntegrator: '<S8>/Discrete-Time Integrator' */

    /* Update for DiscreteIntegrator: '<S9>/Discrete-Time Integrator' */
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LO_dh = 0U;
    CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTAT_g1 += 0.01 * rtb_Sum5;
    if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTAT_g1 >= 5.0) {
      CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTAT_g1 = 5.0;
    } else {
      if (CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTAT_g1 <= (-5.0)) {
        CPG_TR_discrete_DWork.DiscreteTimeIntegrator_DSTAT_g1 = (-5.0);
      }
    }

    /* End of Update for DiscreteIntegrator: '<S9>/Discrete-Time Integrator' */
  } else {
    if (CPG_TR_discrete_DWork.DiscreteTodaRayleighRing_MODE) {
      CPG_TR_discrete_DWork.DiscreteTodaRayleighRing_MODE = FALSE;
```

```
    }
  }

  /* End of Outputs for SubSystem: '<Root>/Discrete Toda Rayleigh Ring' */

  /* Update for UnitDelay: '<Root>/Unit Delay2' */
  CPG_TR_discrete_DWork.UnitDelay2_DSTATE[0] = CPG_TR_discrete_B.x;
  CPG_TR_discrete_DWork.UnitDelay2_DSTATE[1] = CPG_TR_discrete_B.x_o;
  CPG_TR_discrete_DWork.UnitDelay2_DSTATE[2] = CPG_TR_discrete_B.x_d;
  CPG_TR_discrete_DWork.UnitDelay2_DSTATE[3] = CPG_TR_discrete_B.x_b;
  CPG_TR_discrete_DWork.UnitDelay2_DSTATE[4] = CPG_TR_discrete_B.x_ln;
  CPG_TR_discrete_DWork.UnitDelay2_DSTATE[5] = CPG_TR_discrete_B.x_l;
  CPG_TR_discrete_DWork.UnitDelay2_DSTATE[6] = CPG_TR_discrete_B.x_p;
  CPG_TR_discrete_DWork.UnitDelay2_DSTATE[7] = CPG_TR_discrete_B.x_p_p;
  CPG_TR_discrete_DWork.UnitDelay2_DSTATE[8] = CPG_TR_discrete_B.x_p_i;
  CPG_TR_discrete_DWork.UnitDelay2_DSTATE[9] = CPG_TR_discrete_B.x_p_l;
  CPG_TR_discrete_DWork.UnitDelay2_DSTATE[10] = CPG_TR_discrete_B.x_p_h;
  CPG_TR_discrete_DWork.UnitDelay2_DSTATE[11] = CPG_TR_discrete_B.x_p_a;
}

/* Model initialize function */
void CPG_TR_init(void)
{
  /* Registration code */

  /* initialize error status */
  rtmSetErrorStatus(CPG_TR_discrete_M, (NULL));

  /* block I/O */
  (void) memset(((void *) &CPG_TR_discrete_B), 0,
                sizeof(BlockIO_CPG_TR_discrete));

  /* states (dwork) */
  (void) memset((void *)&CPG_TR_discrete_DWork, 0,
                sizeof(D_Work_CPG_TR_discrete));

  /* external inputs */
  (void) memset((void *)&CPG_TR_discrete_U, 0,
                sizeof(ExternalInputs_CPG_TR_discrete));

  /* Start for Enabled SubSystem: '<Root>/Discrete Toda Rayleigh Ring' */
  /* InitializeConditions for DiscreteIntegrator: '<S4>/Discrete Integrator' */
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LOADI = 1U;

  /* InitializeConditions for DiscreteIntegrator: '<S4>/Discrete Integrator1' */
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTATE =
    CPG_TR_discrete_ConstB.x0;

  /* InitializeConditions for DiscreteIntegrator: '<S9>/Discrete Integrator1' */
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTAT_p =
    CPG_TR_discrete_ConstB.x0_d;

  /* InitializeConditions for DiscreteIntegrator: '<S5>/Discrete Integrator1' */
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTAT_b =
    CPG_TR_discrete_ConstB.x0_e;

  /* InitializeConditions for DiscreteIntegrator: '<S5>/Discrete Integrator' */
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LOA_g = 1U;

  /* InitializeConditions for DiscreteIntegrator: '<S6>/Discrete Integrator1' */
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTAT_c =
    CPG_TR_discrete_ConstB.x0_m;

  /* InitializeConditions for DiscreteIntegrator: '<S6>/Discrete Integrator' */
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LOA_d = 1U;

  /* InitializeConditions for DiscreteIntegrator: '<S7>/Discrete Integrator1' */
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTAT_l =
    CPG_TR_discrete_ConstB.x0_n;

  /* InitializeConditions for DiscreteIntegrator: '<S7>/Discrete Integrator' */
```

```
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LOA_e = 1U;

  /* InitializeConditions for DiscreteIntegrator: '<S8>/Discrete Integrator1' */
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator1_DSTA_bn =
    CPG_TR_discrete_ConstB.x0_h;

  /* InitializeConditions for DiscreteIntegrator: '<S8>/Discrete Integrator' */
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LOA_p = 1U;

  /* InitializeConditions for DiscreteIntegrator: '<S9>/Discrete Integrator' */
  CPG_TR_discrete_DWork.DiscreteTimeIntegrator_IC_LO_dh = 1U;

  /* End of Start for SubSystem: '<Root>/Discrete Toda Rayleigh Ring' */

  /* InitializeConditions for Chart: '<Root>/control_logic' */
  CPG_TR_discrete_DWork.old_mode_not_empty = FALSE;
  CPG_TR_discrete_DWork.is_stop = 0U;
  CPG_TR_discrete_DWork.is_walk = 0U;
  CPG_TR_discrete_DWork.is_active_c1_CPG_TR_discrete = 0U;
  CPG_TR_discrete_DWork.is_c1_CPG_TR_discrete = 0U;
  CPG_TR_discrete_B.enable = 0;
  CPG_TR_discrete_B.ctrl = 0.0;
  CPG_TR_discrete_B.dump = 0;
  CPG_TR_discrete_B.p_x[0] = 0.0;
  CPG_TR_discrete_B.p_x[1] = 0.0;
  CPG_TR_discrete_B.p_x[2] = 0.0;
  CPG_TR_discrete_B.p_x[3] = 0.0;
  CPG_TR_discrete_B.p_x[4] = 0.0;
  CPG_TR_discrete_B.p_x[5] = 0.0;
  CPG_TR_discrete_B.p_z[0] = 0.0;
  CPG_TR_discrete_B.p_z[1] = 0.0;
  CPG_TR_discrete_B.p_z[2] = 0.0;
  CPG_TR_discrete_B.p_z[3] = 0.0;
  CPG_TR_discrete_B.p_z[4] = 0.0;
  CPG_TR_discrete_B.p_z[5] = 0.0;
}
```

Bibliografia

[1] S. Banerjee. *Dynamics for Engineers*. John Wiley & Sons, 2005.

[2] D. Belter and P. Skrzypczyński. Efficient gait learning in simulation: crossing the reality gap by evolutionary model identification. In *Mobile Robotics. Solutions and Challenges. Proceedings of the Twelfth International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, Istanbul, Turkey, 2009.

[3] Q. Bombled and O. Verlinden. Dynamic model, gait generation and control of the amru5 hexapod robot. In *Mobile Robotics. Solutions and Challenges. Proceedings of the Twelfth International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, Istanbul, Turkey, 2009.

[4] S. Chakraborty and J. Eberspäche, editors. *Advances in Real-Time Systems*. Springer, 2012.

[5] M. Colnaric and D. Verber. *Distributed Embedded Control Systems: Improving Dependability with Coherent Design*. Springer, 2007.

[6] J. J. Craig. *Introduction to Robotics: Mechanics and Control*. Pearson, 3 edition, 2003.

[7] F. Dalcomyn. *Insect Locomotion on Land. Locomotion and Excercise in Antropods*. Plenum, New York, 1981.

[8] E. del Rio, V. A. Makarov, M. G. Velarde, M. G. Bedia, and W. Ebeling. Electromechanical control of hexapod robot locomotion. *Paper Instituto Plurdisciplinar UCM Madrid*, 2008.

[9] E. del Rio, V. A. Makarov, M. G. Velarde, and W. Ebeling. Mode transistion and wave propagation in a driven-dissipative toda-rayleigh ring. *Physical Review E*, 67(056208), 2003.

[10] M. Długosz. *Problemy optymalizacji układów napędowych w automatyce i robotyce*. PhD thesis, Akademia Górniczo-Hutnicza, 2009.

[11] B. P. Douglass. *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*. Addison Wesley, 2002.

[12] P. Drazin and R. Johnson. *Solitons: an introduction*. Cambridge University Press, 1989.

[13] K. Feng, C.-M. Chew, G.-S. Hong, and T. Zielińska. Bipedal locomotion control using a four-compartmental central pattern generator. In *Mechatronics and Automation, 2005 IEEE International Conference*, Niagara Falls, Ontario, Canada, 2005.

[14] P. Grabowski. *Stabilność układów Lurie*. AGH Uczelnianie Wydawnictwa Naukowo-Techniczne, 1999.

[15] P. Grabowski. *Nonlinear control systems*. publikacja na stronie internetowej, 2012. `http://www.ia.agh.edu.pl/~pgrab/grabowski_files/nonlinear/nonlinear.xml`.

[16] W. Grega. *Sterowanie cyfrowe w czasie rzeczywistym*. Wydawnictwo Wydziału Elektrotechniki, Automatyki, Informatyki i Elektroniki AGH, 1999.

[17] M. W. Hirsh, S. Smale, and R. L. Devaney. *Differential equations, dynamical systems, and introduction to chaos*. Academic Press, 2004.

[18] L. R. P. III, E. D. Diller, and R. D. Quinn. Design aspects of a climbing hexapod. In *Mobile Robotics. Solutions and Challenges. Proceedings of the Twelfth International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, Istanbul, Turkey, 2009.

[19] K. Inagaki. Study of leg arrangement for hexapod walking robot with gait seek algorithm. In *Mobile Robotics. Solutions and Challenges. Proceedings of the Twelfth International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, Istanbul, Turkey, 2009.

[20] R. N. Jazar. *Theory of Applied Robotics: Kinematics, Dynamics, and Control*. Springer, 2007.

[21] J. Kalisz et al. *Język VHDL w praktyce*. Wydawnictwa Łączności i Komunikacji, 2002.

[22] C. Kara, T. Brandt, and D. Schramm. A comparision of static gait patterns for quadrupeds. In *Mobile Robotics. Solutions and Challenges. Proceedings of the Twelfth International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, Istanbul, Turkey, 2009.

[23] B. Krten. *QNX Neutrino RTOS*. QNX Software Systems GmbH & Co., 2008.

[24] L. Ljung. *System Identification: Theory for the User*. Prentice-Hall, 2 edition, 1999.

[25] V. A. Makarov, E. del Rio, M. G. Bedia, M. G. Velarde, and W. Ebeling. Central pattern generator incorporating the actuator dynamics for a hexapod robot. *International Journal of Mathematical and Computer Sciences*, 3(2):97–102, 2007.

[26] V. A. Makarov, E. del Rio, W. Ebeling, and M. G. Velarde. Dissipative toda-rayleigh lattice and its oscillatory modes. *Physical Review E*, 64(036601), 2001.

[27] V. A. Makarov, W. Ebeling, and M. G. Velarde. Soliton-like waves on dissipative toda lattice. *International Journal of Bifurcation and Chaos*, 10(5):1075–1089, 2000.

[28] H. Malepati. *Digital Media Processing,: DSP Algorithms Using C*. Newnes, 2010.

[29] P. Manoonpong. *Neural Preprocessing and Control of Reactive Walking Machines*. Springer-Verlag, 2007.

[30] D. Marchewka and M. Piątek. Stosowanie systemów wbudowanych do sterowania robotami mobilnymi. *Półrocznik AGH Automatyka*, 12(2):257–266, 2008.

[31] The Mathworks Inc. *Simulink, Simulation and Model-Based Design*, 2011.

[32] W. Mitkowski. *Stabilizacja systemów dynamicznych.* Wydawnictwa Naukowo-Techniczne, 1991.

[33] W. Mitkowski. *Równania macierzowe i ich zastosowania.* AGH Uczelnianie Wydawnictwa Naukowo-Techniczne, 2007.

[34] B. Mrozek and Z. Mrozek. *MATLAB i Simulink. Poradnik użytkownika.* Helion, 2 edition, 2004.

[35] K. Ogata. *Discrete-Time Control Systems.* Prentice-Hall, 1995.

[36] K. Ogata. *Modern Control Engineering.* Prentice-Hall, 5 edition, 2010.

[37] L. Perko. *Differential Equtions and Dynamical Systems.* Text in Applied Mathematics. Springer-Verlag, New York, 1991.

[38] P. Petrov and L. Dimitrov. Leg trajectory planning for hexapod robot. In *Mobile Robotics. Solutions and Challenges. Proceedings of the Twelfth International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, Istanbul, Turkey, 2009.

[39] M. Piątek. Real-Time Application Interface and Xenomai modified GNU/Linux real-time operating systems dedicated to control. In *Computer Methods and Systems 2007*, pages 179–184. Oprogramowanie Naukowo-Techniczne, 2007.

[40] M. Piątek. Identification of the servo motor used in the walking robot. *Półrocznik AGH Automatyka*, 14(1), 2010.

[41] M. Piątek and P. Piątek. Development environment for the walking robot application. In *Computer Methods and Systems 2009*, pages 295–299. Oprogramowanie Naukowo-Techniczne, Nov. 2009.

[42] M. Piątek and A. Turnau. Hexapod-six-legged walking robot controlled with toda-rayleigh lattice. *Bio-Algorithms and Med-Systems. Jagelonian University. Medical College*, 8(1):111–121, 2012.

[43] M. Piątek and A. Tutaj. Angular velocity estimation in the six-legged walking robot application. In *PPEEm 2009: Fundamental problems of power electronics electromechanics and mechatronics.* PPEM, Dec. 2009.

[44] P. Piątek. *Wykorzystanie specjalizowanych architektur sprzętowych do realizacji krytycznych czasowo zadań sterowania.* PhD thesis, Akademia Górniczo-Hutnicza, 2007.

[45] L. Rayleigh. *Theory of sound*, volume 1. Macmillan and co, 1894.

[46] A. Roennau, T. Kerscher, M. Ziegenmeyer, J. Zoellner, and R. Dillmann. Six-legged walking in rough terrain based on foot point planning. In *Mobile Robotics. Solutions and Challenges. Proceedings of the Twelfth International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, Istanbul, Turkey, 2009.

[47] A. Schneider, M. Schilling, H. Cruse, and J. Schmitz. Positive velocity feedback on a six-legged walking robot. In *Mobile Robotics. Solutions and Challenges. Proceedings of the Twelfth International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, Istanbul, Turkey, 2009.

[48] A. Silberschatz, P. Galvin, and G. Gagne. *Podstawy systemów operacyjnych.* Wydawnictwa Naukowo-Techniczne, 2005.

[49] D. Simon. *Optimal state estimation*. John Wiley & Sons, 2006.

[50] M. W. Spong and M. Vidyasagar. *Robot Dynamics and Control*. Wydawnictwa Naukowo-Techniczne, 1997.

[51] T. Szmuc. *Zaawansowane metody tworzenia oprogramowania systemów czasu rzeczywistego*. CCATIE - Krakowskie Centrum Informatyki Stosowanej. Katedra Automatyki AGH, 1998.

[52] T. Szmuc and G. Motet. *Specyfikacja i projektowania oprogramowania czasu rzeczywistego*. CCATIE - Krakowskie Centrum Informatyki Stosowanej. Katedra Automatyki AGH, 1998.

[53] M. Toda. *Theory of Nonlinear Lattices*. Springer, Berlin, 2 edition, 1989.

[54] A. Turnau. Model identification dedicated to the time-optimal control. In *17th IFAC World Congress*, pages 2655–2660, July 2008.

[55] A. Veinguertener, T. Hoinville, O. Bruneau, and J.-G. Fontain. Morphological design of the bio-inspired reconfigurable hexaquabip robot. In *Mobile Robotics. Solutions and Challenges. Proceedings of the Twelfth International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, Istanbul, Turkey, 2009.

[56] R. Vepa. *Biomimetic Robotics. Mechanisms and Control*. Cambridge University Press, 2009.

[57] K. Wiatr. *Akceleracja obliczeń w systemach wizyjnych*. WNT, 2003.

[58] M. Wilson. Insect walking. *Annual Review Of Entymology*, (11):103–122, 1966.

[59] T. Zielińska. Coupled oscillators utilised as gait rhythm generators of a two-legged walking machine. *Biological Cybernetics*, 74(3):263–273, 1996.

[60] T. Zielińska. *Maszyny kroczące*. Wydawnictwa Komunikacji i Łączności, Warszawa, 2005.

[61] T. Zielińska. Biological inspiration used for robots motion synthesis. *Journal of Phisiology - Paris*, (103):133–140, 2009.

[62] T. Zielińska, C.-M. Chew, P. Kryczka, and T. Jargilo. Robot gait synthesis using the scheme of human motion skills development. *Mechanism and Machine Theory*, 44(3):541–558, 2009.

[63] T. Zielińska and A. Chmielniak. Biologically inspired motion synthesis method of two-legged robot. In *Mobile Robotics. Solutions and Challenges. Proceedings of the Twelfth International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, Istanbul, Turkey, 2009.