



**A G H**

**AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE**

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,  
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA AUTOMATYKI I INŻYNIERII BIOMEDYCZNEJ

Praca dyplomowa magisterska

*Hybrydowa aplikacja sterowania optymalnego dla systemu zbiorników*

*Hybrid application of optimal control for a tanks system*

Autor: Łukasz Dudek  
Kierunek studiów: Automatyka i Robotyka  
Opiekun pracy: dr hab. inż. Adam Piłat

Kraków, 2017

*Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpozna bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, videogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.*

*Serdecznie dziękuję mojemu promotorowi, dr hab.  
inż. Adamowi Piłatowi za cierpliwość.*



# **Spis treści**

<b>Wstęp.....</b>	7
<b>1. Fizyczny opis zagadnienia .....</b>	9
1.1. Wprowadzenie z zakresu dynamiki płynów .....	9
1.1.1. Równanie Bernoulliego i prawo Torricellego .....	9
1.1.2. Bilans masy .....	11
1.1.3. Rodzaje przepływów .....	12
1.2. Model matematyczny zestawu zbiorników .....	13
<b>2. Regulacja optymalna .....</b>	15
2.1. Regulacja czasooptymalna .....	15
2.1.1. Ogólna definicja zagadnienia.....	15
2.1.2. Nieliniowość układu a sterowanie czasooptymalne.....	18
2.1.3. Analityczne metody wyznaczania sterowania czasooptymalnego.....	18
2.1.4. Numeryczne metody wyznaczania sterowania czasooptymalnego.....	19
2.2. Regulacja liniowo - kwadratowa .....	22
2.2.1. Ogólna definicja zagadnienia.....	22
2.2.2. Linearyzacja modelu.....	23
2.2.3. Dobór wag w zagadnieniu liniowo - kwadratowym .....	24
<b>3. Architektura zaproponowanego rozwiązania.....</b>	25
3.1. Podział zadań między elementami oprogramowania .....	25
3.2. Komunikacja między elementami oprogramowania .....	27
<b>4. Techniczny opis oprogramowania .....</b>	31
4.1. Część obliczeniowa („wyższego poziomu”) .....	31
4.1.1. Opis systemu Tango Controls .....	31
4.1.2. Architektura klasy urządzeń systemu Tango.....	35
4.1.3. Wybór pakietu do optymalizacji dynamicznej.....	40
4.1.4. Środowisko testowe części obliczeniowej .....	46
4.2. Część realizująca sterowanie bezpośrednie („niższego poziomu”). ....	47

4.2.1. Pakiet MATLAB/Simulink jako narzędzie realizujące sterowanie bezpośrednie.....	48
<b>5. Badania symulacyjne .....</b>	<b>51</b>
5.1. Optymalizacja przy użyciu pakietu JModelica.org .....	51
5.1.1. Opis algorytmu optymalizacji dynamicznej .....	51
5.1.2. Inicjalizacja optymalizacji dynamicznej .....	52
5.1.3. Uzyskana postać sterowania .....	53
5.1.4. Dokładność wyznaczania rozwiązania.....	54
5.2. Symulacja i weryfikacja .....	56
5.2.1. Weryfikacja przy użyciu pakietu JModelica.org .....	56
5.2.2. Weryfikacja przy użyciu oprogramowania MATLAB/Simulink .....	62
<b>Zakończenie .....</b>	<b>67</b>
<b>Spis rysunków .....</b>	<b>69</b>
<b>Bibliografia .....</b>	<b>70</b>

# Wstęp

Rozwój techniki komputerowej w ostatnich dziesięcioleciach spowodował szeroki dostęp do metod numerycznych obliczeń problemów analitycznie trudnych bądź niemożliwych do rozwiązywania. W tym momencie skomplikowane algorytmy potrzebujące dużych zasobów sprzętowych można uruchomić na urządzeniach o relatywnie niewielkich rozmiarach. Skorzystała na tym w oczywisty sposób również automatyka, gdyż metody optymalizacji sterowania w układach nieliniowych stały się możliwe do stosowania w czasie rzeczywistym ze względu na krótki czas obliczeń i niewielkie gabaryty urządzeń wykonujących je.

W obecnych czasach istnieje w automatyce tendencja, aby rozdzielać elementy odpowiedzialne za bezpośrednią komunikację z urządzeniami wykonawczymi od elementów uruchamiających skomplikowane matematycznie algorytmy, aby oba rodzaje można było udoskonalać w wykonywaniu tylko jednej klasy zadań. Takie podejście zostało również zaprezentowane w niniejszej pracy. *Hybrydowa* (z łac.: *hybrida*: mieszaniec, krzyżówka) struktura polega na zastosowaniu dwóch poziomów: obliczeniowego i realizującego sterowanie bezpośrednie.

Niewątpliwą zaletą takiego podejścia jest modułowość: posiadając dobrze określoną specyfikację komunikacji, można bez większego problemu wymienić jeden z elementów składowych takiej aplikacji na inny, szybko tworząc prototypy i testując różne technologie w sposób, który nie zakłóca działania całości aplikacji.

Celem niniejszej pracy jest przygotowanie oprogramowania, będącego w stanie wyliczać i aplikować sterowanie czasooptymalne dla układu trzech zbiorników z wodą. Powinno ono móc również symulować funkcjonowanie tego układu i weryfikować w ten sposób działanie optymalizacji. Dodatkowo powinno umieć podtrzymywać stan docelowy zagadnienia czasooptymalnego przy użyciu regulatora liniowo-kwadratowego.

Pierwszy rozdział niniejszej pracy zawiera opis praw i zjawisk fizycznych, które użyto do wyznaczania modelu matematycznego rozważanego układu zbiorników. Podano tam również inżynierskie uproszczenia skutkujące prostą strukturą tegoż modelu oraz sam model wraz z jego ograniczeniami.

W drugim rozdziale przytoczono matematyczne podstawy optymalizacji dynamicznej służące do wyznaczenia dwóch rodzajów sterowań optymalnych potrzebnych w niniejszej pracy: czasooptymalnego i liniowo-kwadratowego. Opisano też analityczne i numeryczne metody rozwiązywania pierwszego z tych zagadnień.

Trzeci rozdział jest poświęcony architekturze oprogramowania realizującego cele pracy. Przedstawiono tam konkretne zadania dwóch elementów aplikacji o hybrydowej strukturze oraz opisano komunikację między nimi z wyszczególnieniem parametrów wysyłanych przez obie strony.

Techniczny opis zagadnień związanych z oprogramowaniem znajduje się w rozdziale czwartym. Zawiera on opisy użytego oprogramowania, w szczególności tego wykorzystywanego do rozwiązywania zadań optymalizacji dynamicznej.

W ostatnim rozdziale zawarto podsumowanie wszystkich przeprowadzonych badań symulacyjnych. Przedstawiono użyty algorytm optymalizacyjny oraz omówiono jego wyniki. Pokazano sposób na weryfikację otrzymanego rozwiązania na obu poziomach aplikacji i zawarto wnioski z niej płynące.

# **1. Fizyczny opis zagadnienia**

Rozważany układ składa się z trzech zbiorników połączonych szeregowo, czy też kaskadowo. W związku z tym płyn (woda), którym jest napełniony pierwszy zbiornik, przepływa przez otwór o zadanym oporze wypływu do drugiego zbiornika. Stamtąd wypływa przez drugi otwór o zadanym oporze do trzeciego zbiornika, skąd przez kolejny taki otwór wypływa do zewnętrznego naczynia. Z niego woda jest pompowana z powrotem do pierwszego zbiornika. Całość układu jest przedstawiona schematycznie na rys. 1.1.

## **1.1. Wprowadzenie z zakresu dynamiki płynów**

W niniejszym podrozdziale zostały przypomniane podstawowe prawa fizyki związane z przepływem cieczy oraz jego związkiem z poziomem tej cieczy w zbiorniku.

### **1.1.1. Równanie Bernoulliego i prawo Torricellego**

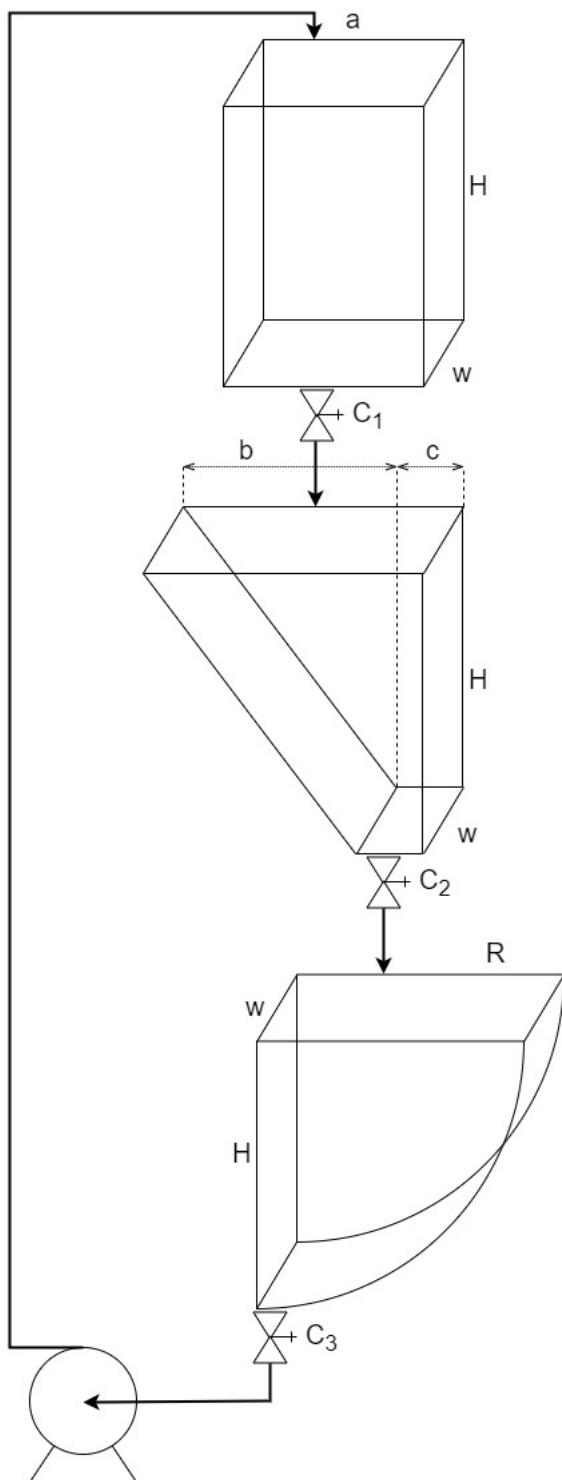
Równanie Bernoulliego jest jednym z podstawowych praw termodynamiki płynów idealnych. Mówiąc o nim, że wzrost prędkości przepływu cieczy musi wiązać się ze spadkiem ciśnienia lub energii potencjalnej. Ma kilka postaci; najpopularniejszą jest tzw. szczególna równanie Bernoulliego, które wiąże energię mechaniczną płynu z jego prędkością w danym miejscu, wysokością w układzie odniesienia służącym do wyznaczania energii potencjalnej, ciśnieniem i gęstością. W takiej formie można je stosować tylko do cieczy nieściśliwych i nielepkich, jednocześnie zakładając stacjonarność i bezwirowość przepływu.

Ta szczególna postać równania Bernoulliego jest przedstawiona jako równanie 1.1.

$$e_m = \frac{v^2}{2} + gh + \frac{p}{\rho} = const \quad (1.1)$$

Oznaczenia:

- $e_m$  - energia jednostki masy cieczy,
- $v$  - prędkość cieczy w danym miejscu,
- $g$  - przyspieszenie grawitacyjne,
- $h$  - wysokość w układzie odniesienia, w którym jest wyznaczana energia potencjalna,



Rys. 1.1. Układ zbiorników z zaznaczonymi wymiarami. Źródło: własne.

- $p$  - ciśnienie cieczy w danym miejscu,
- $\rho$  - gęstość cieczy.

Z równania Bernoulliego można wyprowadzić bezpośrednią zależność między prędkością cieczy a jej poziomem w zbiorniku. Jest ona znana pod nazwą prawa Torricellego i przedstawiona jako równanie 1.2 (przyjęto oznaczenia takie jak w przypadku równania 1.1). Można owo prawo zapisać w bardziej ogólnej formie słownej:

**Prawo Torricellego 1.1.** *Prędkość wypływu cieczy jest proporcjonalna do pierwiastka kwadratowego z poziomu cieczy w zbiorniku.*

$$v = \sqrt{2gh} \quad (1.2)$$

Takie sformułowanie tego prawa będzie istotne w dalszych krokach wyznaczania modelu matematycznego rozważanego układu.

### 1.1.2. Bilans masy

Kolejnym zjawiskiem fizycznym, którego zrozumienie jest potrzebne, aby wyznaczyć model matematyczny rozważanego w niniejszej pracy układu zbiorników, jest bilans masy, czyli bezpośrednia konsekwencja *prawa zachowania masy*.

**Prawo zachowania masy 1.1.** *Masa układu ciał (suma mas wszystkich ciał wchodzących w skład tego układu) nie zmienia się podczas przemian i oddziaływań fizycznych w nim zachodzących.*

$$m_{uk} = const \quad (1.3)$$

Rozważając układ pojedynczego zbiornika z ciecą, do którego ta ciecz jest nalewana i z którego się ona wylewa, można sformułować następstwo tego prawa dane równaniem 1.4. Mówiąc ono, że zmiana masy w rozważanym zbiorniku -  $m_{zb}$  - jest równa zmianie masy do niego wpływającej -  $m_{we}$  - i wypływającej -  $m_{wy}$ .

$$\frac{\partial m_{we}}{\partial t} - \frac{\partial m_{wy}}{\partial t} = \frac{\partial m_{zb}}{\partial t} \quad (1.4)$$

Przyjmując założenie, że ciecz w zbiorniku i poza nim ma stałą gęstość  $\rho$  oraz stosując następujące podstawienia:

- $m_{zb} = V_{zb} \cdot \rho$ , gdzie  $V_{zb}$  to objętość cieczy w zbiorniku,
- $V_{zb} = A_{zb} \cdot h_{zb}$ , gdzie  $A_{zb}$  to pole przekroju poprzecznego zbiornika, a  $h_{zb}$  to wysokość słupa cieczy w tym zbiorniku

można przedstawić powyższą zależność w postaci opisanej zależnością 1.5 (za: [21]).

$$A_{zb} \cdot \frac{\partial h_{zb}}{\partial t} = \frac{\partial V_{we}}{\partial t} - \frac{\partial V_{wy}}{\partial t} \quad (1.5)$$

Strumień (zmiana objętości cieczy w czasie) wypływający z takiego zbiornika można otrzymać na podstawie prawa Torricellego - jest on dany zależnością 1.6, gdzie  $C$  to stała proporcjonalności wypływu. W rozważanym układzie będzie on zależeć od ustawienia zaworu wyjściowego z danego zbiornika, a więc można powiedzieć, że strumień opisuje opór wypływu ze zbiornika (za: [20]).

$$\frac{\partial V_{wy}}{\partial t} = C \cdot \sqrt{h_{zb}} \quad (1.6)$$

Jeśli chodzi o strumień wpływający, to dla drugiego i trzeciego zbiornika jest on równy strumieniowi wypływającemu z poprzedniego zbiornika. Można przyjąć, że dla pierwszego zbiornika ten strumień to sterowanie pompą. Będzie ono oznaczone symbolem  $u$ .

Pola powierzchni przekrójów poprzecznych wszystkich trzech zbiorników przedstawiono jako równanie 1.7. Zastosowano oznaczenia z 1.1.

$$\begin{aligned} A_1 &= w \cdot a \\ A_2 &= c \cdot w + \frac{h_2}{h_{max}} \cdot b \cdot w \\ A_3 &= w \cdot \sqrt{R^2 - (R - h_3)^2} \end{aligned} \quad (1.7)$$

### 1.1.3. Rodzaje przepływów

Przytoczona wcześniej szczególna postać równania Bernoulliego (równanie 1.1) jest obwarowana założeniem stacjonarności przepływu. Oznacza to dwie rzeczy:

1. Wartości wektorów prędkości cieczy są stałe w czasie.
2. Poszczególne „warstwy” cieczy nie wpływają na siebie.

Drugi z tych warunków jest znany pod nazwą przepływu laminarnego, który zwykle ma miejsce przy niskich prędkościach cieczy. W takim typie przepływu nie występują żadne jego zaburzenia (ruchy wirowe, prądy przeciwe itp.), a zachowanie poszczególnych „warstw” cieczy porównać można do tasowania kart: przepływają obok siebie bez wpływania jedna na drugą. Jej cząstki będące blisko powierzchni przemieszczają się po liniach równoległych do tafli cieczy.

Niestety, w rzeczywistości ciężko jest spełnić założenie laminarności przepływu, nie mówiąc już o jego stacjonarności. W związku z tym można zastosować pewne praktyczne uogólnienie zależności 1.6 w stosunku do cieczy wypływających w sposób nielaminarny. Uogólnienie to polega na zastąpieniu pierwiastka we wspomnianym wzorze parametrem  $\alpha$ , którego wartość można dobrać na podstawie pomiarów w rzeczywistym układzie (przykład podany w [20]). Uwzględniając to, można zapisać nowe sformułowanie zależności 1.6 jako równanie 1.8.

$$\frac{\partial V_{wy}}{\partial t} = C \cdot h_{zb}^\alpha \quad (1.8)$$

## 1.2. Model matematyczny zestawu zbiorników

Na podstawie podanych wcześniej zależności można zdefiniować model matematyczny rozważanego układu zbiorników. Jest on dany równaniem 1.9 (za: [20]).

$$\begin{cases} \frac{\partial h_1}{\partial t} = \frac{u - C_1 h_1^{\alpha_1}}{aw} \\ \frac{\partial h_2}{\partial t} = \frac{C_1 h_1^{\alpha_1} - C_2 h_2^{\alpha_2}}{cw + \frac{h_2}{h_{max}} bw} \\ \frac{\partial h_3}{\partial t} = \frac{C_2 h_2^{\alpha_2} - C_3 h_3^{\alpha_3}}{w \sqrt{R^2 - (R - h_3)^2}} \end{cases} \quad (1.9)$$

Oznaczenia:

- $h(t) = [h_1(t) \ h_2(t) \ h_3(t)]^T$  - poziomy wody w zbiornikach,
- $u(t)$  - sterowanie pompą,
- $a$  - szerokość pierwszego zbiornika,
- $b$  - szerokość trójkątnej części drugiego zbiornika,
- $c$  - szerokość prostopadłościennej części drugiego zbiornika,
- $R$  - promień trzeciego zbiornika,
- $w$  - głębokość zbiorników,
- $h_{max}$  maksymalna wysokość słupa wody w zbiornikach,
- $C_i$  - opór wypływu z  $i$ -tego zbiornika,
- $\alpha_i$  - współczynnik wypływu z  $i$ -tego zbiornika.

Wszystkie wymiary w powyższym wzorze zostały przedstawione na rys. 1.1. Są na nim również oznaczone opory wypływu  $C_1$  -  $C_3$  przy odpowiednich zaworach.

Przyjmując  $\alpha_i = \frac{1}{2}, \forall i \in \{1,2,3\}$ , można uszczegółowić powyższy model, zakładając tylko przepływ laminarny. Taka właśnie jego postać będzie wykorzystywana przy analitycznym wyznaczeniu współczynników równania sprężonego (definicja znajduje się w sekcji 2.1.1.1), które jest przeprowadzone w podrozdziale 2.1.3. W rzeczywistości, jak zostało wspomniane w podrozdziale 1.1.3, wartości tych współczynników będą musiały być trochę mniejsze, aby oddać faktyczny sposób przepływu wody między zbiornikami.

W rozważanym układzie zbiorników przyjmuje się następujące ograniczenia (pomijając oczywiste  $t \geq 0$ ):

- ograniczenia równościowe:

$$\begin{aligned} h_1(0) &= h_{10} \\ h_2(0) &= h_{20} \\ h_3(0) &= h_{30} \end{aligned} \quad (1.10)$$

– ograniczenia nierównościowe:

$$\begin{aligned} \forall_{t \in [0, T]} : 0 \leq h_1(t) \leq h_{max} \\ \forall_{t \in [0, T]} : 0 \leq h_2(t) \leq h_{max} \\ \forall_{t \in [0, T]} : 0 \leq h_3(t) \leq h_{max} \\ \forall_{t \in [0, T]} : 0 \leq u(t) \leq u_{max} \end{aligned} \quad (1.11)$$

Parametry  $h_{10}$ ,  $h_{20}$  i  $h_{30}$  oraz  $h_{max}$  traktuje się jako zadane.

Punkty równowagi takiego systemu, nazywane również stanami ustalonimi, będą opisane przez zależność 1.12 i wyznaczone przez parę wektora wartości zmiennych stanu  $h_r$  oraz sterowania  $u_r$ . Brak zależności owej pary od czasu został odpowiednio odnotowany.

$$\frac{\partial h(t)}{\partial t} = 0 \Rightarrow \begin{cases} \frac{\partial h_1}{\partial t} = 0 \\ \frac{\partial h_2}{\partial t} = 0 \\ \frac{\partial h_3}{\partial t} = 0 \end{cases} \quad (1.12)$$

Dla rozważanego układu zbiorników punkty równowagi będą miały postać daną zależnością 1.13.

$$\begin{aligned} u_r &= C_1 h_1^{\alpha_1} = C_2 h_2^{\alpha_2} = C_3 h_3^{\alpha_3} \\ h_r &= \begin{bmatrix} h_{1r} \\ h_{2r} \\ h_{3r} \end{bmatrix} = \begin{bmatrix} \left(\frac{u_r}{C_1}\right)^{\frac{1}{\alpha_1}} \\ \left(\frac{u_r}{C_2}\right)^{\frac{1}{\alpha_2}} \\ \left(\frac{u_r}{C_3}\right)^{\frac{1}{\alpha_3}} \end{bmatrix} \end{aligned} \quad (1.13)$$

Wynika z tego, że układ otwarty jest asymptotycznie stabilny, gdyż posiada tylko jeden, zerowy punkt równowagi (pokazany jako wzór 1.14). Jest to również zgodne z fizyczną naturą tego systemu zbiorników: przy braku zasilania go pompą, cała woda wycieknie ze wszystkich trzech zbiorników.

$$h_r^0 = \begin{bmatrix} h_{1r}^0 \\ h_{2r}^0 \\ h_{3r}^0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad (1.14)$$

## 2. Regulacja optymalna

W niniejszym rozdziale zostały opisane matematyczne podstawy wyznaczania sterowania optymalnego według dwóch wskaźników jakości występujących w niniejszej pracy:

- czasowego (regulacja czasooptymalna),
- liniowo-kwadratowego (regulacja liniowo-kwadratowa).

### 2.1. Regulacja czasooptymalna

W niniejszym podrozdziale przedstawiona została koncepcja regulacji czasooptymalnej. Zostały podane założenia zagadnienia, twierdzenia, na których podstawie można wyliczyć rozwiązanie oraz jego proponowana forma analityczna.

Zaznacza się, że mimo iż podane niżej definicje i założenia są wzięte z ogólnych zagadnień optymalizacji dynamicznej, to w podanym brzmieniu stosują się tylko do zagadnienia wyznaczania sterowania czasooptymalnego.

#### 2.1.1. Ogólna definicja zagadnienia

##### 2.1.1.1. Założenia wstępne

Dany jest układ opisany stacjonarnym, zwyczajnym równaniem różniczkowym (pokazanym jako równanie 2.1), w którym:

$$\frac{\partial x(t)}{\partial t} = f(x(t), u(t)), \quad 0 \leq t \leq T \quad (2.1)$$

- wektor zmiennych stanu w chwili  $t$  -  $x(t)$  spełnia następujące założenia:
  - $\forall_{t \geq 0} : x(t) \in \mathbb{R}^n$  - ma  $n$  składowych, a więc rozważany system ma  $n$  równań różniczkowych zwyczajnych,
  - $x(0) = x_0 \in \mathbb{R}^n$  - spełnia warunek początkowy  $x_0$ ;
- wektor sterowań w chwili  $t$  -  $u(t)$ :

- $\forall_{t \geq 0} : u(t) \in D \subset \mathbb{R}^m$  - ma  $m$  składowych zawierającym się w zbiorze  $D$  ograniczającym wartości sterowań,
- $u(0) = u_0$  - spełnia warunek początkowy  $u_0$ ,
- funkcja  $u$  jest przedziałami ciągła na przedziale  $[0, T]$  (dokładny opis w [16]), czyli:
  - \* ma co najwyżej skończoną liczbę punktów nieciągłości,
  - \* w każdym z nich ma skończoną granicę lewostronną,
  - \* jest prawostronnie ciągła,
  - \* w lewym końcu przedziału jest lewostronnie ciągła;
- funkcja  $f : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$ :
  - $f \in C^1$  - jest ciągła i różniczkowalna ze względu na pierwszy argument,
  - $\frac{\partial f(t)}{\partial x} \in C^0$  - jej pochodna ze względu na pierwszy argument jest ciągła.

Rozwiązaniem takiego równania jest oczywiście funkcja  $x : [0, T] \mapsto \mathbb{R}^n$  nazywana *trajektorią układu*.

Trajektoria będąca rozwiązaniem zagadnienia minimalnozasowego musi spełniać następujący warunek końcowy (nazywany również stanem docelowym):

$$x(T) = x_f \in \mathbb{R}^n \quad (2.2)$$

Ów czas  $T$ , po którym przy danym sterowaniu stan systemu osiągnie warunek końcowy (nazywany również czasem optymalnym), będzie wskaźnikiem jakości:

$$Q(u(t)) = q(x_f) = T \quad (2.3)$$

Na tej podstawie można określić *sterowanie optymalne*  $\hat{u}(t)$ , które spełnia wszystkie wspomniane przy opisie równania 2.1 warunki oraz zależność 2.4. Trajektoria układu wygenerowana przez zastosowanie sterowania optymalnego nazwana jest *trajektorią optymalną* i opisana symbolem  $\hat{x}(t)$ .

$$\forall_{u(t)} : Q(u(t)) \leq Q(\hat{u}(t)) \quad (2.4)$$

W opisie zagadnienia czasooptymalnego potrzebne są jeszcze dwa pojęcia. Pierwsze to *funkcja sprzężona*  $\psi : [0, T] \mapsto \mathbb{R}^n$  będąca rozwiązaniem tzw. równania sprzężonego 2.5.

$$\frac{\partial \psi(t)}{\partial t} = -\frac{\partial f(x(t), u(t))}{\partial x} \quad (2.5)$$

Tak, jak w przypadku trajektorii optymalnej układu, trajektoria  $\psi(t)$  wyznaczona w układzie, w którym zastosowane zostało sterowanie optymalne  $\hat{u}(t)$ , nosi nazwę *trajektorii sprzężonej optymalnej* i oznaczona jest symbolem  $\hat{\psi}(t)$ .

Ostatnim pojęciem potrzebnym w niniejszym zagadnieniu jest *hamiltonian*, zwany również *funkcją Hamiltona*, czyli funkcja  $H : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^m \mapsto \mathbb{R}$  który dla trajektorii układu  $x(t)$  wygenerowanej przy

pomocy sterowania  $u(t)$  i odpowiadającej im trajektorii sprzężonej  $\psi(t)$  zdefiniowany jest zależnością 2.6.

$$H(\psi(t), x(t), u(t)) = \psi(t) \circ f(x(t), u(t)) = \psi(t)^T \cdot f(x(t), u(t)) \quad (2.6)$$

### 2.1.1.2. Zasada maksimum Pontriagina

Aby jednoznacznie opisać, a następnie wyznaczyć sterowanie czasooptymalne, potrzebne jest przytoczenie zasadniczego twierdzenia w optymalizacji dynamicznej. Jest ono znane pod nazwą *zasada maksimum Pontriagina*. Zostało opracowane w 1956 r. przez rosyjskiego matematyka Lwa Pontriagina. Twierdzenie podaje się w brzmieniu z [16].

**Zasada maksimum Pontriagina 2.1.** *Zakładając układ opisany równaniem 2.1 z warunkiem końcowym 2.2 i wskaźnikiem jakości 2.3 oraz równanie sprzężone 2.5: jeśli dla trajektorii układu  $\hat{x}(t)$  wygenerowanej przy pomocy sterowania  $\hat{u}(t)$  i odpowiadającej im trajektorii sprzężonej  $\hat{\psi}(t)$  zachodzi:*

$$\forall_{u(t) \in D} \forall_{t \in [0, T]} : H(\hat{\psi}(t), \hat{x}(t), \hat{u}(t)) \geq H(\hat{\psi}(t), \hat{x}(t), u(t)) \quad (2.7)$$

*to sterowanie  $\hat{u}(t)$  jest sterowaniem optymalnym.*

Dowód zasady maksimum nie został uwzględniony w niniejszej pracy. Można go znaleźć w [16] oraz w [19].

Powyższe twierdzenie należy obwarować dodatkowymi warunkami koniecznymi optymalności. Niech funkcja  $g : \mathbb{R}^{2n} \rightarrow \mathbb{R}^l$  opisuje zestaw ograniczeń nierównościowych, a funkcja  $h : \mathbb{R}^{2n} \rightarrow \mathbb{R}^k$  - ograniczeń równościowych. Obie dane są wzorem 2.8. Dodatkowo zakłada się, że  $g, h \in C^1$ .

$$\begin{aligned} g(x_0, x_f) &\leq 0 \\ h(x_0, x_f) &= 0 \end{aligned} \quad (2.8)$$

Ponadto, zakłada się, że istnieją  $\lambda \in \mathbb{R}$ ,  $\mu \in \mathbb{R}^l$  oraz  $\rho \in \mathbb{R}^k$ , które wraz z uprzednio zdefiniowanymi wielkościami i funkcjami spełniają następujące *warunki konieczne optymalności*:

– warunek nieujemności:

$$\lambda \geq 0 \wedge \|\mu\| \geq 0 \quad (2.9)$$

– warunek nietrywialności:

$$\lambda + \|\mu\| + \|\rho\| > 0 \quad (2.10)$$

– warunek komplementarności:

$$\mu \circ g(x_0, x_f) = 0 \quad (2.11)$$

– warunki transwersalności:

$$\begin{aligned} \hat{\psi}(0) &= \frac{\partial(\mu \circ g + \rho \circ h)}{\partial x_0} \\ \hat{\psi}(T) &= -\frac{\partial(\mu \circ g + \rho \circ h)}{\partial x_f} \end{aligned} \quad (2.12)$$

$$\forall_{t \in [0, T]} : H(\psi(t), x(t), u(t)) = \frac{\partial(\lambda T + \mu \circ g + \rho \circ h)}{\partial T} = \lambda$$

- równanie sprzężone dane wzorem 2.5,
- warunek maksimum hamiltonianu dany nierównością 2.7.

### 2.1.2. Nieliniowość układu a sterowanie czasooptymalne

Poszukiwanie sterowania czasooptymalnego w systemach nieliniowych jest skomplikowane, przede wszystkim ze względu na trudność rozwiązań analitycznego nieliniowych (lub nawet niestacjonarnych) równań różniczkowych. Układ zbiorników omawiany w niniejszej pracy jest również takimi opisany: zarówno równania dynamiki układu 1.9, jak i równania sprzężone 2.15 są nieliniowe.

Aby uprościć analizę problemu oraz fizyczne zastosowanie wyznaczonego sterowania, zakłada się, że poszukuje się go tylko w postaci „bang - bang”. To znaczy, że będą przyjmowały tylko wartości z brzegów jednowymiarowego zbioru dopuszczalnego  $D \subset \mathbb{R} \wedge D = [0, u_{max}] \Rightarrow \forall_{t \in [0, T]} : u(t) \in [0, u_{max}]$ .

Można przyjąć takie założenie, że względu na to, iż dla układów, których równania mają postać opisaną równaniem 2.13, funkcja przełączająca ma postać:  $\phi(t) = \hat{\psi}(t) \circ g(\hat{x}(t))$ . Opisano taką sytuację w [7] oraz w rozdziale 7.10 [1]. Taka sytuacja zachodzi również w omawianym układzie, gdzie  $m = 1$ .

$$\frac{\partial x}{\partial t} = f(x(t)) + \sum_{i=1}^m g_i(x(t)) \cdot u_i(t) \quad (2.13)$$

Funkcja przełączająca opisuje momenty, w których sterowanie zmienia swoją wartość z jednego krańca zbioru  $D$  na drugi. W przypadku rozważanego układu zbiorników sterowanie optymalne będzie dane wzorem 2.14.

$$\begin{aligned} \hat{u}(t) &= \frac{sgn(\phi(t))+sgn(\phi(t))^2}{2} \cdot u_{max} \wedge \phi(t) = \frac{\hat{\psi}_1(t)}{aw} \Rightarrow \\ \hat{u}(t) &= \frac{sgn(\hat{\psi}_1(t))+sgn(\hat{\psi}_1(t))^2}{2} \cdot u_{max} \end{aligned} \quad (2.14)$$

Dodatkowo, jak wspomniano w podrozdziale 1.2, układ otwarty jest stabilny, a ograniczone sterowanie nie może tego zmienić.

Podobne założenia są częstą praktyką w analizie stabilnych systemów nieliniowych ze względu na prostotę fizycznej aplikacji sterowania „bang - bang”. Przykłady znajdują się m.in. w [2], [3] oraz [17].

### 2.1.3. Analityczne metody wyznaczania sterowania czasooptymalnego

Analityczne rozwiązania poszukiwania sterowania czasooptymalnego zwykle opierają się bezpośrednio na przytoczonej powyżej zasadzie maksimum i warunkach koniecznych optymalności. W niniejszym podrozdziale zostanie pokrótko przedstawiona droga mogąca zmierzać do wyznaczenia analitycznego czasooptymalnego sterowania w rozważanym układzie zbiorników. W niniejszej pracy całe rozwiązanie analityczne nie zostało przeprowadzone ze względu na fakt, iż równania sprzężone są niestacjonarne, a więc ich rozwiązanie analityczne byłoby bardzo trudne lub wręcz niemożliwe. Poniżej przedstawiono

tylko kroki, które mogłyby prowadzić do takiego rozwiązania, gdyby dało się rozwiązać analitycznie równania sprzężone.

Pierwszym krokiem ku wyliczeniu analitycznego rozwiązania jest wyznaczenie równań sprzężonych za pomocą wzoru 2.5. Przyjmując współczynniki  $\alpha_i = \frac{1}{2} \forall i \in \{1,2,3\}$  w modelu matematycznym zestawu zbiorników danym równaniem 1.9, można wyznaczyć równania sprzężone rozwijanego układu. Są one dane wzorem 2.15. Pominięto w nim zależności wszystkich funkcji  $\psi$  oraz  $h$  od czasu, aby uprościć zapis.

$$\left\{ \begin{array}{l} \frac{\partial \psi_1}{\partial t} = \psi_1 \frac{C_1}{2aw\sqrt{h_1}} - \psi_2 \frac{C_1}{2\sqrt{h_1}(cw + \frac{h_2}{h_{max}}bw)} \\ \frac{\partial \psi_2}{\partial t} = -\psi_2 \frac{1}{cw + \frac{h_2}{h_{max}}bw} \left( \frac{b(C_1\sqrt{h_1} - C_2\sqrt{h_2})}{ch_{max} + bh_2} - \frac{C_2}{2\sqrt{h_2}} \right) - \psi_3 \frac{1}{w\sqrt{h_3}(2R - h_3)} \\ \frac{\partial \psi_3}{\partial t} = \psi_3 \frac{-C_3(3R - 2h_3)}{wh_3(2R - h_3)^{\frac{3}{2}}} \end{array} \right. \quad (2.15)$$

Następnie trzeba by przedefiniować ograniczenia równościowe (dane wzorem 1.10) i nierównościowe (1.11) tak, aby spełniały założenia funkcji  $g$  i  $h$  opisane zależnościami 2.8. Dodatkowo należy dopisać od ograniczeń równościowych to wynikające z faktu poszukiwania sterowania czasooptymalnego (wzór 2.2). W tym przypadku będzie ono miało postać opisaną przez 2.16, gdzie  $h_{1f}$ ,  $h_{2f}$  i  $h_{3f}$  są dane.

$$\begin{aligned} h_1(T) &= h_{1f} \\ h_2(T) &= h_{2f} \\ h_3(T) &= h_{3f} \end{aligned} \quad (2.16)$$

Korzystając z warunków komplementarności (2.11) oraz nieujemności (2.9), powinno się wyznaczyć składowe wektora  $\mu$  oraz założyć pewną postać wektora  $\rho$  (bazując takie założenie na 2.10).

Na podstawie tych danych należało wyznaczyć warunki początkowy i końcowy danym wzorem 2.12 dla powyższych równań sprzężonych, co pozwoliłoby wyznaczyć analityczne wzory opisujące wszystkie składowe trajektorii sprzężonych systemu.

Na koniec, korzystając z zależności 2.14, można by wyznaczyć analityczny wzór sterowania optymalnego.

#### 2.1.4. Numeryczne metody wyznaczania sterowania czasooptymalnego

W niniejszej sekcji zostały przekrojowo zaprezentowane numeryczne metody poszukiwania sterowania czasooptymalnego dla układów nieliniowych. Nie jest jej celem przedstawienie dokładnego działania wszystkich istniejących algorytmów, lecz dokonanie klasyfikacji i naświetlenie interesujących metod oraz odesłanie do odpowiedniej literatury przedmiotu.

Ogólna definicja problemu optymalizacji rozwiązywanego przez takie metody została dana w sekcji 2.1.1. Tutaj należy ją uzupełnić o ogólną postać funkcjonału  $Q$  będącego wskaźnikiem jakości w rozpatrywanym zadaniu. Jest ona dana równaniem 2.17 (za: [10]) i składa się z:

– wyrażenia podcałkowego  $L$  zależącego od:

- czasu,
- stanu  $x(t)$ ,
- sterowania  $u(t)$ ,
- parametrów statycznych układu  $p$ ,
- czasu końcowego  $T$ ;

– wyrażenia  $M$  zależącego od:

- czasu początkowego optymalizacji  $t_0$ ,
- stanu początkowego  $x(t_0)$ ,
- stanu końcowego  $x(T)$ ,
- parametrów statycznych układu  $p$ ,
- czasu końcowego  $T$ .

$$Q[x(\cdot), u(\cdot), p, t_0, T] = \int_{t_0}^T L(\tau, x(\tau), u(\tau), p, T) d\tau + M(t_0, x(t_0), x(T), p, T) \quad (2.17)$$

Gdy we wskaźniku jakości występuje tylko człon podcałkowy, to mówi się, że jest postaci Lagrange'a. Jeśli ma tylko drugi człon, nazywa się go postacią Mayera. Sytuację, w której występują oba, określa się mianem postaci Boltza (za: [10]). Oczywiście, podział między członami jest umowny i można ten sam funkcjonał zapisać zarówno w postaci Lagrange'a, jak i Mayera. Przykładem jest rozpatrywany w niniejszej pracy wskaźnik jakości dany postacią Mayera we wzorze 2.3. Równie dobrze można by go zapisać w postaci Lagrange'a:  $Q = \int_0^T 1 d\tau$ .

Podstawowy podział metod obliczeniowych to metody *pośrednie* oraz *bezpośrednie*, choć ich rozróżnienie nie jest ścisłe i istnieją metody łączące cechy obu tych klas (za: [4]).

Pierwsze z nich polegają na dyskretyzacji sterowania (czasem również stanu), aby móc przybliżyć wyjściowy problem optymalizacji dynamicznej zadaniem optymalizacji statycznej: programowania nieliniowego (ang. *NLP - Non-Linear Programming*). To przybliżenie jest zwykle aproksymacją sterowań (oraz stanów w tych metodach, które z nich korzystają) w konkretnych przedziałach czasu wielomianami. Dzięki temu zastępuje się nieskończoność wymiarowy problem optymalizacji takim o skończonym wymiarze. Metody bezpośrednie dzieli się na:

- metody *sekwencyjne*, które używają wysokiej klasy oprogramowania do całkowania sterowania, aproksymowanego wielomianami i traktują wyliczone na jego podstawie trajektorie jako dokładne,
- metody *równoczesne*, które korzystają z parametrycznej aproksymacji trajektorii układu uzyskanej w wyniku kolokacji.

Ta klasa metod jest bardziej uniwersalna i mniej wrażliwa na błędy, ale stosunkowo wolniej zbieżna i przez to mniej dokładna (za: [16]). Przykładowe metody bezpośrednie przedstawione są poniżej:

- optymalizacja czasów przełączeń (do zastosowania tylko w przypadku sterowania o postaci „bang-bang”),
- parametryzacja sterowania (np. aproksymacja funkcją schodkową),
- bezpośredni metody kolokacyjne,
- metody strzałów bezpośrednich.

Metody pośrednie wykorzystują warunki konieczne optymalności oryginalnego zadania i spowadzają je do problemu granicznego lub dwugranicznego, który jest dany układem równań bazującym na zasadzie maksimum Pontriagina. Rozwiążanie takiego układu równań skutkuje wyznaczeniem trajektorii będących potencjalnie rozwiązaniem problemu. Każda z tych trajektorii bada się, sprawdzając, jakim jest ekstremum: minimum, maksimum lub punktem siodłowym. Następnie wybiera się taką, której koszt jest najmniejszy. Metody pośrednie wymagają dobrze dobranej trajektorii początkowej układu, która zawiera informacje na temat struktury sterowania (za: [16]). Jest to wada tego rodzaju metod - wymagają większego nakładu pracy analitycznej przed rozpoczęciem obliczeń, aby wyznaczyć odpowiedni punkt startowy dla algorytmu. Zaletą za to jest duża szybsza zbieżność w otoczeniu rozwiązania (za: [16]). Przykładowe metody pośrednie to:

- pośrednia metoda strzałów,
- pośrednia metoda strzałów wielokrotnych,
- pośrednie metody kolokacyjne.

Rys. 2.1 przedstawia operacje matematyczne wykorzystywane przez oba rodzaje metod rozwiązywania problemów optymalizacji dynamicznej. Dokładniejszy opis metod wykonujących te operacje znajduje się w [10].



**Rys. 2.1.** Trzy główne komponenty wyznaczania sterowania optymalnego i klasy metod, które z nich korzystają. Źródło: [10]. Tłumaczenie: własne.

## 2.2. Regulacja liniowo - kwadratowa

W niniejszym podrozdziale zostało przedstawione zagadnienie szukania sterowania w układach liniowych z tzw. liniowo-kwadratowym wskaźnikiem jakości oraz nieskończonym horyzontem czasowym. Pokazana została również metoda zastosowania takiego typu regulacji dla układów nieliniowych przy użyciu linearyzacji w punkcie pracy. Znajduje się tu również krótkie omówienie zagadnienia doboru wag przy wyznaczaniu regulatora liniowo-kwadratowego.

### 2.2.1. Ogólna definicja zagadnienia

Dany jest układ opisany liniowym równaniem różniczkowym 2.18 z wektorem zmiennych stanu  $x(t) \in \mathbb{R}^n$  i warunkiem początkowym dla nich  $x(0) = x_0$ . Sterowaniem w tym układzie jest  $u(t) \in \mathbb{R}^m$ . Jak zostało wspomniane wyżej, horyzont czasowy jest nieskończony. Cały układ jest analogiczny do opisanego w sekcji 2.1.1.

$$\frac{\partial x(t)}{\partial t} = Ax(t) + Bu(t), \quad 0 \leq t < \infty \quad (2.18)$$

Przyjmuje się również wskaźnik jakości w tym układzie dany wzorem 2.19.

$$Q(x(t), u(t)) = \frac{1}{2} \int_0^\infty (x(t)^T W x(t) + u(t)^T R u(t)) dt \quad (2.19)$$

Dodatkowo zakłada się następujące właściwości macierzy wymienionych w powyższym wzorze:

- macierz  $W = W^T \geq 0$  - jest symetryczna i nieujemnie określona,
- macierz  $R = R^T > 0$  - jest symetryczna i dodatnio określona (jest również czasem w literaturze oznaczana symbolem  $Q$ , gdy wskaźnik jakości jest oznaczony symbolem  $J$ ).

Oczywiście, aby istniało sterowanie optymalne, musi istnieć również całka będąca wskaźnikiem jakości - jest to warunek konieczny i wystarczający. Sterowalność pary  $A$  i  $B$  lub asymptotyczna stabilność macierzy  $A$  jest warunkiem wystarczającym. Oba są podane w [1] i w [16].

Wyznaczenie sterowania optymalnego w tak zdefiniowanym problemie opiera się na rozwiązaniu algebraicznego równania Ricattiego (danego zależnością 2.20) przy założeniu, że macierz  $K$  jest symetryczna i nieujemnie określona. Jest ono opisane szerzej w [1], [16] i [5].

$$KBR^{-1}B^T K - A^T K - KA - W = 0 \quad (2.20)$$

Sterowanie optymalne  $\hat{u}(t)$  ma wówczas postać daną wzorem 2.21.

$$\hat{u}(t) = -R^{-1}B^T K x(t), \quad t \geq 0 \quad (2.21)$$

Aby taki regulator zapewnił asymptotyczną stabilność kontrolowanego układu, para  $(W, A)$  musi być wykrywalna, a para  $(A, B)$  - stabilizowalna (za: [16]).

### 2.2.2. Linearyzacja modelu

Mimo iż podstawowa definicja zagadnienia liniowo kwadratowego zakłada liniowość sterowanego układu (jak we wzorze 2.18), to można rozszerzyć jego zastosowanie również na układy nieliniowe przez użycie linearyzacji w punkcie pracy.

Niech będzie dany układ opisany przez 2.1 wraz ze wszystkimi założeniami opisanymi w sekcji 2.1.1. Można go stabilizować tylko w jednym z jego punktów równowagi  $(x_r, u_r)$  (definiowanych równaniem 2.22) ze względu na to, iż tylko do takich punktów układ może dążyć w nieskończoności, która jest horyzontem czasowym analizowanego zagadnienia.

$$\frac{\partial x(t)}{\partial t} = 0 \Rightarrow f(x_r(t), u_r(t)) = 0 \quad (2.22)$$

Na podstawie takiego punktu równowagi definiuje się odchyłki od stanu i sterowania ustalonego dane zależnością 2.23.

$$\begin{aligned} \Delta x(t) &= x(t) - x_r \\ \Delta u(t) &= u(t) - u_r \end{aligned} \quad (2.23)$$

Przy użyciu tak zdefiniowanych odchyłek wyjściowy nieliniowy układ można aproksymować układem liniowym (dany wzorem 2.24) w pewnym otoczeniu owych odchyłek.

$$\frac{\partial \Delta x(t)}{\partial t} = \bar{A} \Delta x(t) + \bar{B} \Delta u(t) \quad (2.24)$$

Macierze  $A$  i  $B$  zostały uzyskane w procesie linearyzacji funkcji  $f(x(t), u(t))$  w punkcie  $(x_r, u_r)$  - wzór 2.25 pokazuje sposób, w jaki są zdefiniowane.

$$\begin{aligned} \bar{A} &= \left( \left. \frac{\partial f(x(t), u_r)}{\partial x(t)} \right|_{x(t)=x_r} \right)^T \\ \bar{B} &= \left( \left. \frac{\partial f(x_r, u(t))}{\partial u(t)} \right|_{u(t)=u_r} \right)^T \end{aligned} \quad (2.25)$$

Wskaźnik jakości dla takiego systemu również przyjmuje postać odchyłkową zaprezentowaną jako równanie 2.26. Założenia co do macierzy  $W$  i  $R$  są dalej takie same.

$$Q(\Delta x(t), \Delta u(t)) = \frac{1}{2} \int_0^\infty (\Delta x(t)^T W \Delta x(t) + \Delta u(t)^T R \Delta u(t)) dt \quad (2.26)$$

Aby stabilizować układ w punkcie  $x_r$ , potrzebna jest definicja regulatora bazującego na odchyłkach. Jego wzór jest analogiczny do 2.21 i ma postać daną zależnością 2.27 (znaleźć ją można m.in. w [6] i w [16]). Tak definiuje się regulator optymalny dla zagadnienia liniowo-kwadratowego w układach nieliniowych.

$$\Delta \hat{u}(t) = -R^{-1} \bar{B}^T K \Delta x(t), t \geq 0 \quad (2.27)$$

Tak jak w przypadku układów liniowych, aby system z zamkniętą pętlą sprzężenia zwrotnego był asymptotycznie stabilny, para  $(W, \bar{A})$  musi być wykrywalna, a para  $(\bar{A}, \bar{B})$  - stabilizowalna (znów za: [16]).

### 2.2.3. Dobór wag w zagadnieniu liniowo - kwadratowym

Kluczowym aspektem rzeczywistego zastosowania regulatorów liniowo-kwadratowych jest dobór wag, które wpływają na jego funkcjonowanie. Wagi zawierają się we współczynnikach dwóch macierzy obecnych we wskaźniku jakości (wzór 2.19):  $W$  i  $R$ .

Ze względu na to, iż macierz odwrotna do macierzy  $R$  występuje bezpośrednio we wzorach 2.21 oraz 2.27, można założyć, że im mniejsza norma macierzy  $R$  (a więc również im mniejsze jej poszczególne współczynniki), tym regulator będzie generował mocniejszy sygnał. Wpływa to oczywiście na szybkość zmian w układzie oraz na ewentualne przeregulowania (efekt opisany w [6] i w [16]).

Jeśli chodzi o macierz  $W$ , to jej współczynniki odpowiadają za sterowanie poszczególnymi zmiennymi stanu oraz na wzajemne zależności między nimi. Najprostsza metoda doboru wag, opisana w [5], polega na założeniu, że macierz  $W$  jest diagonalna i wyznaczeniu maksymalnych dopuszczalnych błędów dla każdej zmiennej stanu. Jako wartość danej wagi należy przyjąć kwadrat odwrotności owego błędu. Przykład takiego postępowania jest podany poniżej:

- zakłada się, że istnieje zmienna stanu  $x_1(t)$ , dla której dopuszczalny jest błąd  $\delta_{x_1} = 0.01$ ,
- wartość odpowiadającego jej współczynnika macierzy  $W$  -  $w_1$  - powinna wynosić  $\delta_{x_1}^{-2} = 100^2 = 10000$ ,
- wtedy przy wyliczaniu wskaźnika jakości pod całką znajdzie się wyrażenie  $\Delta x_1^2 w_1$ , które da wartość 1, gdy odchyłka między wartością zadaną  $x_{r1}$  a aktualną wartością  $x_1(t)$  będzie wynosiła  $\delta_{x_1}$ .

Ogólna zasada może więc być podsumowana następująco: największe współczynniki w macierzy  $W$  i najmniejsze w macierzy  $R$  przyporządkowuje się tym zmiennym stanu, których minimalizacja ma priorytet (za: [6]).

### **3. Architektura zaproponowanego rozwiązania**

W niniejszej pracy została zaproponowana hybrydowa struktura aplikacji realizującej zadania wyliczania i aplikowania sterowania czasooptymalnego, jak i optymalnego w sensie liniowo-kwadratowego wskaźnika jakości. Taka jej architektura jest odbiciem faktycznej tendencji w automatyce ostatnich lat: aby skomplikowane zadania obliczeniowe zadawać nie tym elementom, które realizują bezpośrednie sterowanie, ale zlecać je innym urządzeniom o architekturze sprzętu odpowiedniejszej do ich realizacji.

W niniejszym rozdziale opisano cele każdego z dwóch poziomów aplikacji: obliczeniowego i realizującego sterowanie bezpośrednie oraz podano specyfikację komunikacji między nimi.

#### **3.1. Podział zadań między elementami oprogramowania**

W związku z zaproponowaną w niniejszej pracy architekturą oprogramowania, opierającą się na dwóch poziomach, kluczowym aspektem jest odpowiedni podział zadań między tymi poziomami. Założono również, że oba poziomy powinny mieć możliwość przeprowadzania symulacji obiektu, aby można było weryfikować poprawność funkcjonowania obu w niezależny sposób.

Pierwszy z nich to część „wyższego” poziomu - została tak określona ze względu na to, iż nie ma bezpośredniego wpływu na kontrolowany fizyczny obiekt, a zajmuje się tylko modelem matematycznym. Jej zadania są przedstawione na poniższej liście.

1. Symulacja modelu nieliniowego w pętli otwartej, aby zainicjować algorytm optymalizacji dynamicznej.
2. Wyznaczanie sterowania czasooptymalnego dla zadanych wartości początkowych i końcowych.
3. Symulacja modelu nieliniowego w pętli zamkniętej, aby zweryfikować wyliczone sterowanie czasooptymalne.
4. Linearyzacja modelu w punkcie pracy.
5. Wyznaczanie sterowania optymalnego w sensie liniowo-kwadratowego wskaźnika jakości (dla modelu linearyzowanego w punkcie pracy).

Dodatkowo przyjęto następujące założenia w związku z tymi zadaniami (podzielone ze względu na to, którego zagadnienia optymalizacji dotyczą):

1. Założenia związane ze sterowaniem czasooptymalnym:

- (a) Sterowanie czasooptymalne jest postaci „bang-bang” (wyjaśnienie w sekcji 2.1.2).
- (b) W związku z tym wystarczy wyznaczyć czasy przełączeń między konkretnymi sterowaniami maksymalnym i minimalnym oraz to, które z nich ma być aplikowane jako pierwsze.

2. Założenia związane ze sterowaniem liniowo-kwadratowym:

- (a) Punkt pracy (w którym jest dokonywana linearyzacja) jest stanem docelowym zagadnienia czasooptymalnego, jeśli ten jest punktem równowagi systemu (definicja dana w podrozdziale 1.2).
- (b) Jeśli stan docelowy zagadnienia czasooptymalnego nie jest stanem ustalonym rozważanego układu, stosuje się przybliżenie go do pewnego punktu równowagi i tam dokonuje się linearyzacji modelu matematycznego.
- (c) Sterowanie liniowo-kwadratowe jest wyliczane jako odchyłka od sterowania ustalonego dla punktu równowagi.

Na podstawie powyższych zadań oraz założeń sformułowano poniższą listę parametrów, które musi przyjmować część optymalizacyjną od użytkownika:

- parametry statyczne modelu matematycznego:
  - fizyczne rozmiary zbiorników (parametry  $a, b, c, R, w$  i  $h_{max}$ ),
  - opory wypływu ze zbiorników (parametry  $C_1, C_2$  oraz  $C_3$ ),
  - współczynniki wypływu ze zbiorników (parametry  $\alpha_1, \alpha_2$  i  $\alpha_3$ );
- wielkości związane z ograniczeniami w zagadnieniu czasooptymalnym:
  - maksymalne sterowanie ( $u_{max}$ ),
  - wartości początkowe poziomów wody w zbiornikach (parametry  $h_{10}, h_{20}$  oraz  $h_{30}$ ),
  - wartości końcowe poziomów wody w zbiornikach (parametry  $h_{1f}, h_{2f}$  i  $h_{3f}$ );
- wagi w zagadnieniu liniowo-kwadratowym:
  - wartość wagi sterowania  $R \in \mathbb{R}$ ,
  - wartości wag stanów dane jako macierz  $Q \in \mathbb{R}^3$ .

Drugi element zaproponowanego w niniejszej pracy oprogramowania to część „niższego” poziomu - jej nazwa jest związana z tym, że bezpośrednio wpływa na sterowany układ. Nie zawiera tak skomplikowanych narzędzi obliczeniowych, ale za to powinna cechować się dużą niezawodnością. Jej zadania są przedstawione na poniższej liście.

1. Aplikacja sterowania czasooptymalnego przez czas, który jest wyliczony przez część „wyższego poziomu” oraz w odpowiedniej postaci („bang-bang”).
2. Po upływie tego czasu aplikacja sterowania optymalnego w sensie liniowo-kwadratowego wskaźnika jakości aż do czasu otrzymania kolejnego sterowania czasooptymalnego.
3. Symulacja układu modelu matematycznego w celach testowych i weryfikacyjnych.

W związku z tym, że jest to element realizujący bezpośrednie sterowanie, użytkownik nie powinien mieć wpływu na jego funkcjonowanie. Cały interfejs między nim a programem sterującym powinien dotyczyć tylko i wyłącznie części „wyższego poziomu”.

## 3.2. Komunikacja między elementami oprogramowania

Hybrydowa struktura aplikacji wymusza dokładne zdefiniowanie schematów komunikacyjnych między oboma jej poziomami. Poniżej znajduje się podsumowanie wartości wysyłanych przez oba poziomy.

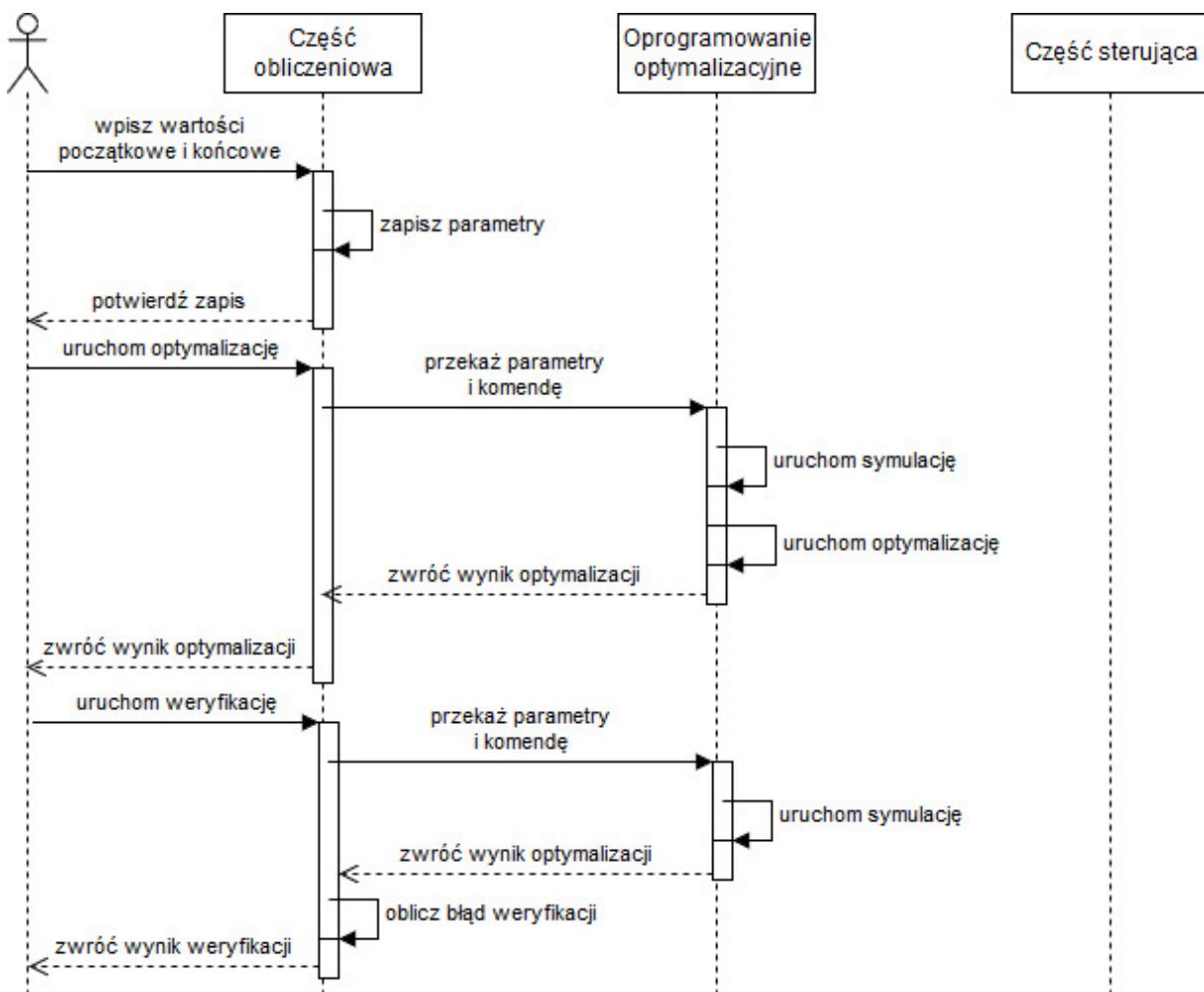
1. Część obliczeniowa wysyła:
  - (a) wartości końcowe poziomów w zbiornikach (będące stanem docelowym w zadaniu czasooptymalnym oraz punktem linearyzacji modelu służącym do wyliczenia nastaw regulatora liniowo-kwadratowego),
  - (b) dla sterowania czasooptymalnego:
    - i. czasy przełączeń,
    - ii. wartość początkową sterowania czasooptymalnego,
    - iii. wartość „drugorzędna” tego sterowania (założono, że niższy poziom aplikacji nie musi znać ograniczeń nałożonych na sterowanie),
    - iv. czas optymalny;
  - (c) dla sterowania liniowo-kwadratowego:
    - i. wektor K współczynników regulatora,
    - ii. sterowanie ustalone, od którego są liczone odchyłki.
2. Część sterowania bezpośredniego wysyła:
  - (a) aktualne poziomy wody w zbiornikach,
  - (b) aktualną wartość sterowania.

Poniżej znajduje się ich lista wyszczególnionych 3 możliwych akcji zachodzących w systemie. Zilustrowano je odpowiednimi (aczkolwiek uproszczonymi do poziomu ogólnej specyfikacji) diagramami sekwencji według konwencji UML (ang. *Unified Modelling Language*). Wyszczególniono na nich użytkownika, część obliczeniową i sterującą oraz oprogramowanie optymalizacyjne, aby zaznaczyć, że jest

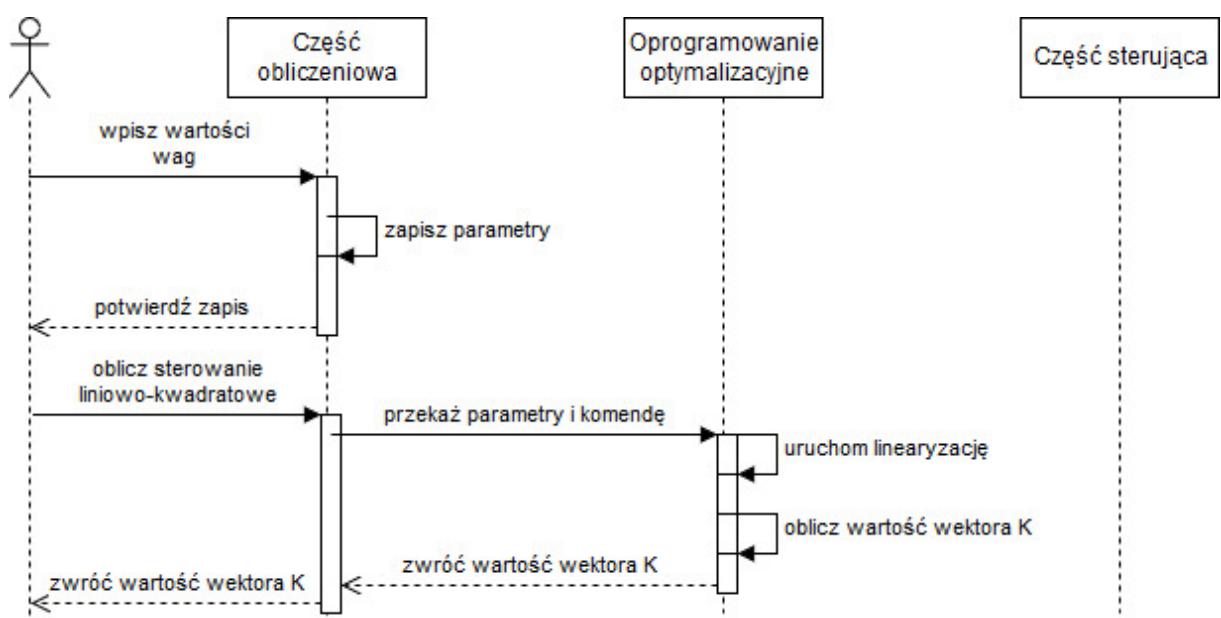
ono de facto osobnym elementem, zewnętrznym i niestanowiącym części aplikacji będącej przedmiotem niniejszej pracy.

1. Akcja obliczania sterowania czasooptymalnego (przedstawiona na rys. 3.1).
2. Akcja obliczania sterowania liniowo-kwadratowego (pokazana na rys. 3.2).
3. Akcja wysłania obliczonych sterowań między poziomami aplikacji (zaprezentowana na rys. 3.3 zawierającym pozostałe akcje w uproszczonej formie).

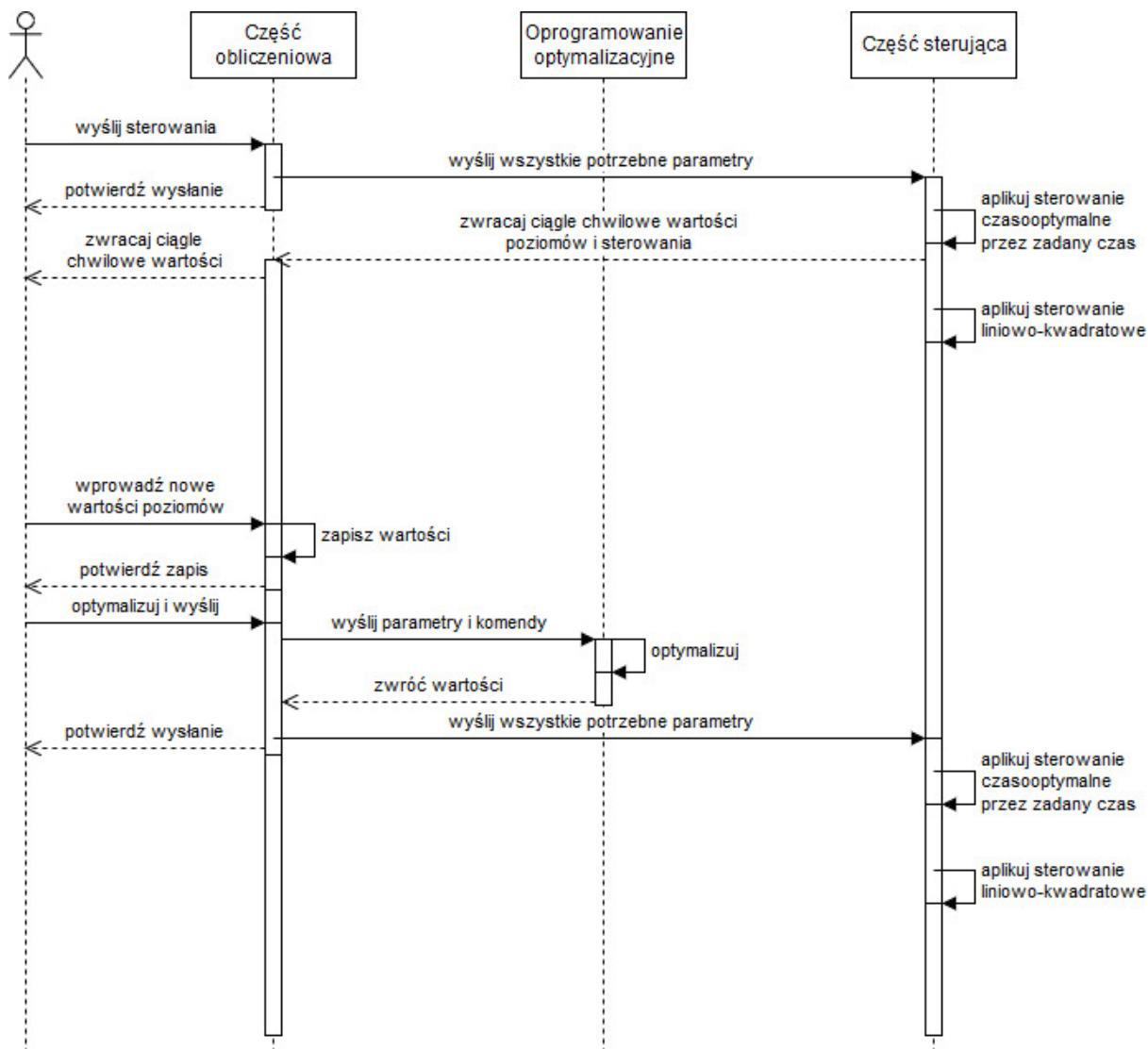
Zdecydowano, że protokołem przesyłu danych między poziomami aplikacji będzie TCP (ang. *Transmission Control Protocol*). Jest to zdecydowanie najpopularniejszy sposób przesyłu danych w sieci Internet między dwoma urządzeniami, z których jedno jest serwerem, a drugie klientem. Ze względu na to, że wyższy poziom aplikacji dostarcza większej ilości danych, zdecydowano, że to właśnie on będzie działał jako serwer TCP.



Rys. 3.1. Diagram sekwencji ilustrujący akcję obliczenia sterowania czasooptymalnego. Źródło: własne.



Rys. 3.2. Diagram sekwencji ilustrujący akcje obliczenia sterowania liniowo-kwadratowego. Źródło: własne.



Rys. 3.3. Diagram sekwencji ilustrujący akcję wysyłania sterowań. Źródło: własne.

## **4. Techniczny opis oprogramowania**

W niniejszym rozdziale znajduje się opis napisanej na potrzeby tej pracy aplikacji realizującej postawione we wstępnie zadanie. W poszczególnych podrozdziałach opisano wszystkie techniczne aspekty mające wpływ na ostateczny kształt obu poziomów przygotowanego oprogramowania.

### **4.1. Część obliczeniowa („wyższego poziomu”)**

Aby zapewnić jak najlepszą realizację zadań postawionych przed częścią obliczeniową, należało wybrać odpowiednie oprogramowanie używane przez ten poziom aplikacji. Przyjęto dwa podstawowe założenia co do niego:

- powinien być napisana w języku programowania Python,
- powinien używać biblioteki Tango Controls jako swojej podstawy.

Pierwsze założenie zostało przyjęte ze względu na niewątpliwe zalety tego języka. Jest on mocno zorientowany na programowanie obiektowe, posiada przejrzystą, nieskomplikowaną składnię oraz, jako język interpretowany, umożliwia szybkie prototypowanie i korzystanie z interaktywnego wiersza poleceń.

Dodatkowo jest to język dostępny w całości na otwartej licencji, a społeczność skupiona wokół niego dostarcza wielu bibliotek ogólnego i szczególnego zastosowania, które również często są otwartym oprogramowaniem. Podjęto w związku z tym próbę znalezienia wśród tych bibliotek takich, które byłyby pomocne w rozwiązywaniu zagadnień optymalizacji dynamicznej - więcej na ten temat w sekcji 4.1.3.

Drugie założenie zostanie wyjaśnione w kolejnej sekcji.

#### **4.1.1. Opis systemu Tango Controls**

Tango Controls to zestaw narzędzi służący do zarządzania rozproszonymi systemami sterowania umożliwiający przyłączanie dowolnych urządzeń oraz fragmentów oprogramowania do jednej magistrali programowej. Jest on również dystrybuowany na otwartej licencji (tzw. słabszej powszechniej licencji publicznej GNU - ang. *Lesser GNU Public Licence* - w wersji 3) i utrzymywany przez konsorcjum placówek naukowych z całej Europy (więcej informacji w [24]).

#### 4.1.1.1. Ogólne założenia systemu

Początki tego oprogramowania sięgają roku 1999, kiedy to w Europejskim Ośrodku Synchrotronu Atomowego w Grenoble podjęto próbę implementacji nowego systemu pozwalającego na połączenie wszystkich urządzeń jedną magistralą programową i zaczęto pracę nad jej uogólnieniem, aby mogła przyjąć dowolny sprzęt, który tylko będzie miał napisany odpowiedni do niej sterownik. Osiągnięto ten cel dzięki użyciu odpowiedniej technologii komunikacyjnej (oryginalnie implementacji architektury CORBA, od wersji 7 również biblioteki ZeroMQ) oraz zastosowaniu własnego protokołu i ogólnej przestrzeni adresowej dla wszystkich urządzeń w systemie. Zastosowano tutaj również podejście „klient - serwer”, a więc istnieje wyraźne rozgraniczenie między interfejsem dla klienta (aplikacji pobierającej dane z systemu) oraz serwera (aplikacji dostarczającej dane). Dostępne są 3 tryby komunikacji między klientami a serwerami:

- synchroniczny,
- asynchroniczny,
- zdarzeniowy.

Przyłączanie nowego sprzętu do tego systemu opiera się na zasadzie opakowywania istniejącego kodu służącego do komunikacji z danym urządzeniem kodem realizującym operacje związane z systemem Tango Controls. W związku z tym, że jednym z podstawowych założeń tego systemu jest obiektość, to opakowanie będzie miało postać *klasy urządzeń*. Dzięki temu przejmuje on wszystkie zalety programowania obiektowego: możliwość dziedziczenia między klasami, co ułatwia utrzymywanie hierarchicznej struktury pisanej oprogramowania, jak i możliwość współdzielenia bardziej ogólnych klas między różnymi systemami, które korzystają z Tango Controls. W szczególności wszystkie klasy muszą dziedziczyć po bazowej klasie - *Device*.

#### 4.1.1.2. Urządzenia i ich serwery

Podstawowym pojęciem opisywanego systemu jest *urządzenie*, które jest obiektem w sensie programistycznym (instancją klasy danego typu urządzeń). Może to być:

- logiczna abstrakcja istniejącego fizycznie sprzętu, niezależnie od tego, czy jest to jeden prosty czujnik, czy bardziej skomplikowane urządzenie z własnym HMI (ang. *Human-Machine Interface*: interfejs człowiek-maszyna),
- logiczna abstrakcja grupy urządzeń, również niezależnie od ich stopnia skomplikowania,
- fragment oprogramowania - mówi się wtedy o czysto programowym urządzeniu.

Definicja przestrzeni adresowej systemu zakłada, że każde urządzenie w systemie ma nazwę składającą się z 3 członów przedzielonych prawymi ukośnikami („/”):

- domeny,

- rodziny,
- członka.

Przykładowa nazwa wygląda następująco: sys/tg\_test/1.

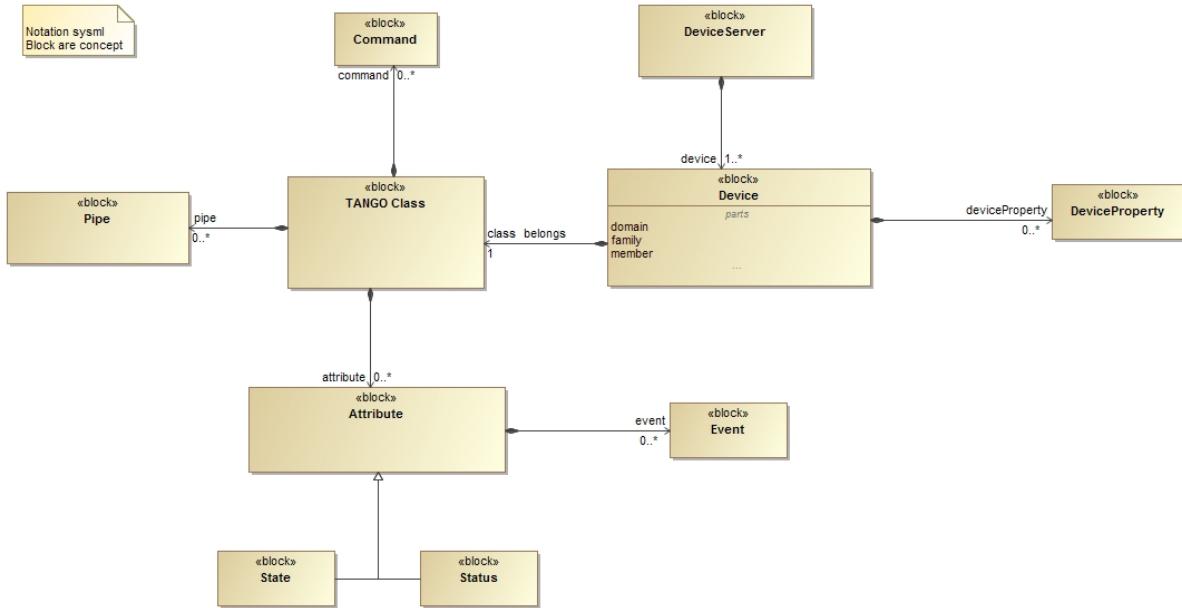
Dodatkowo podając nazwę urządzenia, można również podać adres bazowy systemu, czyli adres i port, na którym można nawiązać komunikację TCP z urządzeniem zarządzającym dostępem do warstwy persystencji systemu - bazy danych MySQL. Tam przechowywane są informacje na temat urządzeń kiedykolwiek uruchomionych w systemie. Ten adres bazowy systemu określa się zmienną środowiskową w systemie operacyjnym o nazwie TANGO\_HOST. Określenie takiej pary adresu i portu jednoznacznie definiuje, z którym systemem zarządzanym przez Tango Controls klient się chce połączyć. Uzupełniona o nią (i o nazwę protokołu) przykładowa nazwa urządzenia wygląda następująco: tango://tango-host.org:10000/sys/tg\_test/1.

Urządzenia systemu Tango są grupowane w *serwery urządzeń* - procesy w systemie operacyjnym, które przy swoim starcie pytają warstwy persystencji o to, jakie urządzenia mają uruchomić. Każdy serwer urządzeń ma określony zestaw klas urządzeń, których instancjami może zarządzać: uruchamiać je, edytować, wyłączać, kasować itp. Umożliwia on również włączenie odpytywania konkretnego elementu interfejsu urządzenia z okresem zadanym w milisekundach. Proponuje się rozróżnienie na klasę urządzeń i typ serwera urządzeń, aby uniknąć nieścisłości. Każdy serwer jest również obiektem klasy urządzeń *DServer*, a jego typ określa to, jakimi klasami urządzeń dysponuje.

Na interfejs urządzenia składają się cztery typy pól w klasie tego urządzenia:

1. Atrybuty (ang. *attributes*) - reprezentują wielkości fizyczne, zmieniające się w czasie działania urządzenia. Muszą mieć zdefiniowany typ (jeden z podstawowych typów danych numerycznych lub ciągu znaków) oraz wymiar (Tango Controls obsługuje atrybuty skalarne, wektorowe i macierzowe). Każde urządzenie musi mieć przynajmniej 2 atrybuty: stan i status (testowy opis tego, co się dzieje z urządzeniem). Przykładowy atrybut urządzenia będącego odwzorowaniem silnika krokowego w systemie Tango to pozycja (zmiennoprzecinkowa wartość skalarna) lub stan wyłączników krańcowych (dwuelementowy wektor wartości logicznych).
2. Strumienie (ang. *pipes*) - pozwalają na przesyłanie dowolnego typu surowych danych do urządzenia lub na jego zewnętrz. Dla każdego ich pakietu trzeba jednak określić ten typ przed transmisją. Mogą być jedno- lub dwukierunkowe. Jest to nowy typ pola w klasach urządzeń, obecny od wersji 9.
3. Komendy (ang. *commands*) - będące operacjami, których wykonanie urządzenie umożliwia. Mogą być wywoływanie z maksymalnie jednym argumentem wejściowym i również tylko jeden argument wyjściowy mogą zwracać. Przykładem komendy urządzenia będącego logiczną abstrakcją silnika krokowego jest operacja bazowania - znalezienie określonej pozycji sprężonego enkodera i wyzerowanie rejestru położenia.

4. Właściwości (ang. *properties*) - zawierające statyczną konfigurację urządzenia. Jako jedyne są przechowywane w bazie danych, skąd serwer urządzeń przy starcie pobiera ich wartości.



Rys. 4.1. Uproszczony model obiektów w systemie Tango Controls. Źródło: [18].

Klasa urządzeń może zawierać definicję maszyny stanów, która opisuje warunki przejść między dowolnymi z 14 dostępnych stanów. Są w tym zbiorze ogólne stany opisujące poprawne działanie urządzenia (INIT, ON, OFF, STANDBY, RUNNING, MOVING), błędy w funkcjonowaniu (ALARM, FAULT, DISABLE), takie nadające się tylko dla określonych typów sprzętu (OPEN, CLOSE, INSERT, EXTRACT) oraz stan nieustalony (UNKNOWN). Maszyna stanów może również ograniczać dostęp do innych elementów interfejsu urządzenia (atrybutów i komend) na podstawie tego, w jakim stanie ono się znajduje.

Wszystkie opisane wyżej pojęcia oraz relacje między nimi zostały przedstawione na diagramie 4.1. Znajduje się tam klasa urządzenia (ang. *Device Class*, *TANGO Class*), która definiuje elementy interfejsu urządzeń będących jej instancjami (atrybuty, komendy i strumienie). Na schemacie zostały dodatkowo oznaczone zdarzenia (ang. *Events*), których każdy atrybut może wysyłać 3 typy: okresowe, archiwizacji i zmiany. Dodatkowo zaznaczono, że urządzenie musi posiadać atrybuty stanu i statusu. Urządzenie jest obiektem takiej klasy i należy do serwera urządzeń (ang. *Device Server*). Każde dodatkowo ma konkretne wartości właściwości przechowywane w bazie danych systemu Tango.

#### 4.1.1.3. Interfejs programistyczny systemu

Specyfikacja Tango Controls definiuje API (ang. *Application Programming Interface*: interfejs programowania aplikacji), do którego jest dostęp w różnych językach programowania na wielu systemach operacyjnych (m.in.: różne dystrybucje systemu Linux, Windows, Mac OS oraz Solaris). Jest on dokładnie opisany w dokumentacji dostępnej w [18].

Tabela 4.1 podsumowuje wszystkie możliwe technologie, z których można korzystać, aby łączyć się z systemem zarządzanym przez Tango Controls.

Oprócz tego istnieje też bogaty zestaw narzędzi służących do zarządzania systemem, nadzorowania go, tworzenia interfejsów graficznych oraz realizowania innych usług wymaganych w rozproszonym systemie sterowania (np. archiwizacji, zbierania logów czy automatycznej konfiguracji).

Język	Typ implementacji	Co umożliwia
C++	pełna	klient i serwer
Java	pełna	klient i serwer
Python	opakowanie do implementacji w C++	klient i serwer
C	opakowanie do implementacji w C++	klient
LabView	implementacja pomocnicza w języku C++	klient i serwer
MATLAB/Octave	tylko warstwa komunikacyjna w języku C++	klient
IgorPro	tylko warstwa komunikacyjna w języku C++	klient
Panorama	wtyczka do aplikacji	klient
JavaScript	tylko warstwa komunikacyjna z użyciem protokołu REST	klient

**Tabela 4.1.** Podsumowanie języków programowania, w których istnieje możliwość połączenia z systemem Tango Controls. Źródło: [24].

W związku ze wszystkimi opisanymi cechami uznano Tango Controls za odpowiednią bibliotekę do oparcia na niej części obliczeniowej aplikacji stanowiącej podstawę niniejszej pracy.

#### 4.1.2. Architektura klasy urządzeń systemu Tango

Architektura systemu Tango Controls wymusiła pewne decyzje co do struktury tej części aplikacji. Zdecydowano, że będzie ona napisana jako klasa urządzeń, która określa interfejs dla wszystkich operacji wymaganych do spełnienia zadań postawionych przed tym poziomem aplikacji w podrozdziale 3.1. Nie zdecydowano się na podział zadań na mniejsze klasy urządzeń komunikujące się ze sobą. Przyjęto, że prostsza struktura z jedną klasą ułatwi użycie napisanego kodu w przyszłości, np. w formie części zajęć laboratoryjnych.

Dokumentacja kodu klasy urządzeń jest wpisana w pliki źródłowe. Można również zbudować ją osobno, na przykład w środowisku testowym dla wyższego poziomu aplikacji, którego opis znajduje się w sekcji 4.1.4.

Dodatkowym elementem wyższego poziomu aplikacji jest serwer TCP, który zarządza komunikacją z aplikacją sterowania bezpośredniego. Został on zaimplementowany jako osobny moduł uruchamiany z poziomu urządzenia systemu Tango.

#### 4.1.2.1. Interfejs dla systemu Tango

Określając interfejs dla systemu Tango, przyjęto założenie, że wszystkie parametry statyczne modelu matematycznego powinny być również statyczną konfiguracją urządzenia, a więc należy opisać je jako jego właściwości. Głównym powodem przyjęcia takiej reguły jest fakt, iż w czasie działania - realizacji sterowania optymalnego - te wielkości nie będą się zmieniać. Parametry dynamiczne, czyli takie, które zmieniają się między uruchomieniami algorytmów optymalizacyjnych, powinny zostać opisane jako atrybuty urządzenia. Wszystkie dostępne akcje na modelu matematycznym zostały zaimplementowane jako komendy. Dodano tam również wszystkie akcje opisane w podrozdziale 3.2.

Poniższa lista zawiera wszystkie istotne dla użytkownika elementy interfejsu systemu Tango urządzenia realizujące zadania wyższego poziomu aplikacji. Każdy z nich jest pokrótko opisany.

##### 1. Właściwości:

- (a) *IPOPTTolerance* - tolerancja błędów oprogramowania do rozwiązywania zagadnień programowania nieliniowego (więcej o wpływie na działanie optymalizacji w sekcji 5.1.4).
- (b) *ModelFile* - nazwa pliku z modelem dla oprogramowania do optymalizacji (więcej w sekcji 4.1.3).
- (c) *MaxControl* - górne ograniczenie sterowania.
- (d) *Tank\*Outflow* - opory wypływu ze zbiorników  $C_i$ . Dane jako 3 osobne właściwości - w miejscu \* jest liczba 1, 2 lub 3.
- (e) *Tank\*FlowCoeff* - współczynniki wypływu ze zbiorników  $\alpha_i$ . Dane j.w.
- (f) *SimulationFinalTime* - czas symulacji używanej w celu inicjalizacji algorytmu optymalizacji (więcej o tym procesie w sekcji 5.1.2).
- (g) *TCPServerEnabled* - wartość logiczna zawierająca informację o tym, czy urządzenie ma uruchomić serwer TCP w celu komunikowania się z aplikacją niższego poziomu.
- (h) *TCPServerAddress* - para adres i port (przedzielone dwukropkiem). Pod tym adresem serwer TCP będzie nasłuchiwał przychodzących połączeń.

##### 2. Atrybuty (nie umożliwiają zapisu, chyba że zaznaczono inaczej):

- (a) *H\*Current* - zawierają aktualne poziomy wody w zbiornikach odebrane od aplikacji niższego poziomu. Dane jako 3 osobne atrybuty - w miejscu \* jest liczba 1, 2 lub 3.
- (b) *H\*Initial* - zawierają początkowe wartości poziomów dla optymalizacji dynamicznej. Dane j.w., możliwy zapis.
- (c) *H\*Final* - zawierają końcowe wartości poziomów dla optymalizacji dynamicznej. Dane j.w., możliwy zapis.
- (d) *H\*Simulated* - zawierają wektory przebiegu poziomów w zbiornikach uzyskane w symulacji. Dane j.w.

- (e) *OptimalH\** - zawierają wektory przebiegu poziomów w zbiornikach uzyskane w czasie optymalizacji (wyznaczania sterowania czasooptymalnego). Dane j.w.
- (f) *OptimalTime* - liczba będąca wartością wskaźnika jakości uzyskaną w procesie wyznaczania sterowania czasooptymalnego.
- (g) *SimulationTime* - wektor wartości czasu uzyskany w czasie symulacji potrzebny w celurysowania przebiegów symulacyjnych *H\*Simulated*.
- (h) *OptimalControl* - wektor zawierający sterowanie czasooptymalne.
- (i) *ControlCurrent* - zawiera aktualną wartość sterowania odebraną od aplikacji niższego poziomu.
- (j) *SwitchTimes* - wektor zawierający czasy przełączeń po normalizacji sterowania czasooptymalnego.
- (k) *Q* - macierz 3 na 3 zawierająca wagi poziomów do zagadnienia liniowo-kwadratowego. Możliwy zapis.
  - (l) *R* - wartość wagi sterowania do zagadnienia liniowo-kwadratowego. Możliwy zapis.
- (m) *K* - wektor współczynników regulatora liniowo-kwadratowego.
- (n) *VerificationError* - wartość błędu weryfikacji (więcej o jego wyznaczaniu w podrozdziale 5.2).
- (o) *FEMElements* - liczba elementów metody elementów skończonych, używanej przez algorytm optymalizacyjny (więcej o jej wpływie na optymalizację w sekcji 5.2.1). Możliwy zapis.

### 3. Komendy:

- (a) *LoadInitialModel* - wczytuje model początkowy służący do inicjalizacji symulacji i optymalizacji.
- (b) *ResetModel* - resetuje wczytany model początkowy.
- (c) *GetEquilibriumFromControl* - przyjmuje wartość sterowania i na jej podstawie wylicza punkt równowagi, a następnie ustawia odpowiednie wartości poziomów końcowych.
- (d) *RunSimulation* - uruchamia symulację, a po jej zakończeniu ustawia wartości zmiennych symulacyjnych.
- (e) *RunVerification* - uruchamia symulację, a po jej zakończeniu ustawia wartości zmiennych symulacyjnych oraz wylicza błąd weryfikacji.
- (f) *Optimise* - uruchamia procedurę optymalizacji, a następnie ustawia wartości poziomów optymalnych, sterowania optymalnego i osiągniętego czasu. Przyjmuje jeden argument - wartość logiczną, która określa, czy jako wartości początkowych powinna użyć aktualnych poziomów wody w zbiornikach. Jeśli tak jest, to natychmiast po zakończeniu optymalizacji uruchamia komendę *SendControl*.

- (g) *NormaliseOptimalControl* - przeprowadza operację normalizacji sterowania czasooptymalnego (więcej na ten temat w sekcji 5.1.4) i ustawia wartości czasów przełączeń.
- (h) *SendControl* - sprawdza, czy są gotowe wartości do wysłania do aplikacji niższego poziomu, czyli czasy przełączeń i wartości wektora  $K$  (jeśli nie są gotowe, uruchamia komendę *GetLQR*), a następnie wysyła je do serwera TCP.
- (i) *GetDataFromDirectControl* - odbiera od serwera TCP dane od aplikacji sterowania bezpośredniego. Jest odpisywana standardowo co 50 milisekund. Ustawia aktualne poziomy i sterowanie.
- (j) *GetLQR* - uruchamia procedurę linearyzacji i wyliczania parametrów regulatora liniowo-kwadratowego. Ustawia wartości wektora  $K$ .
- (k) *GetEigenvaluesFromClosedLoopWithLQR* - zwraca w formie tekstowej wartości własne macierzy  $A - BR^{-1}B^T K$  opisującej układ zamknięty sterowany regulatorem liniowo-kwadratowym.

4. Dostępne stany:

- (a) OFF - model początkowy jest niewczytany.
- (b) STANDBY - model początkowy jest wczytany, ale nie trwa żadna dłuża operacja.
- (c) ON - trwa symulacja.
- (d) RUNNING - trwa optymalizacja.
- (e) ALARM - dłuża operacja się nie udała. W przypadku optymalizacji oznacza to, iż nie udało się znaleźć sterowania optymalnego. W przypadku weryfikacji, że zwróciła ona błąd (więcej na ten temat w sekcji 5.2.1).

#### 4.1.2.2. Problem dostępności interfejsu systemu Tango

Biblioteka Tango Controls nakłada jeszcze jedno ograniczenie na część obliczeniową aplikacji: interfejs powinien być dostępny z maksymalnie trzysekundowym opóźnieniem. To znaczy, że każda prośba klienta musi zostać obsłużona w ciągu 3 sekund. W przypadku operacji symulacji i optymalizacji to założenie nie jest spełnione, ponieważ szczególnie ta druga może trwać nawet kilka minut.

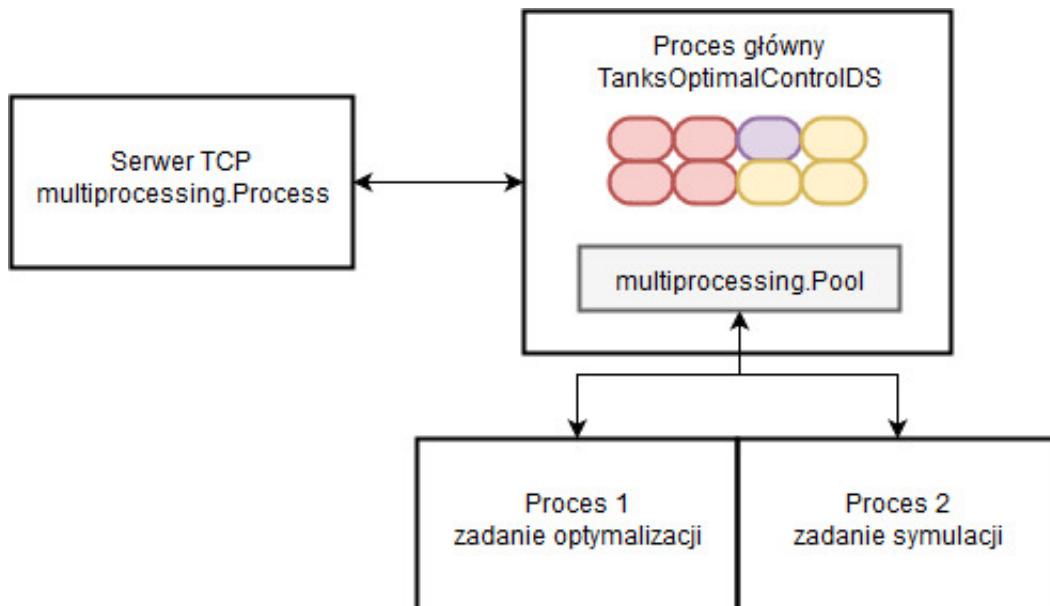
W API Tango Controls do języka Python (nazywanym się PyTango) istnieją sposoby na asynchroniczne wywoływanie operacji (tzw. *green modes*), ale mają dwie wady:

- używają tylko jednego wątku głównego aplikacji,
- procedury pakietu JModelica.org są z nimi niekompatybilne.

W związku z tym zaproponowano własne, ogólne rozwiązanie problemu potrzeby asynchronicznego wykonywania operacji wewnętrz klasy urządzeń. W tym celu użyto biblioteki `multiprocessing` wchodzącej w skład standardowego zestawu narzędzi języka Python. Za jej pomocą zaimplementowano

asynchroniczne wywoływanie długich operacji w osobnych procesach, które po swoim końcu wywołują metody odwołania (ang. *callback*) w głównym procesie aplikacji. Dzięki temu wszystkie komendy, które wywołują długie operacje (*Optimise*, *RunSimulation* oraz *RunVerification*) tylko uruchamiają osobny proces i przypisują odpowiednią metodę odwołania, nie blokując interfejsu urządzenia.

Osobnym procesem jest również serwer TCP. Posiada on własny system logowania i komunikuje się z głównym procesem aplikacji wyższego poziomu poprzez potok danych (ang. *pipe*, nie mylić z częścią interfejsu urządzenia systemu Tango). Jest on dwukierunkowy: serwer może przez niego wysyłać aktualne dane odebrane od aplikacji niższego poziomu, a główny proces wszystkie informacje o sterowaniu. Odbiór danych po stronie serwera znajduje się na początku pętli obsługi komunikacji z klientem TCP (aplikacją niższego poziomu). Odbiór danych po stronie urządzenia odbywa się w odpytywanej co 50 milisekund komendzie *GetDataFromDirectControl*.



**Rys. 4.2.** Schemat architektury procesów i wątków wyższego poziomu aplikacji. Źródło: własne.

Na rys. 4.2 przedstawiono schematycznie użycie procesów (oznaczonych prostokątami o ostrych wierzchołkach) i wątków (oznaczonych prostokątami o obłych wierzchołkach) w wyższym poziomie aplikacji. Kolorem szarym zaznaczono kontener procesów klasy `multiprocessing.Pool`. Wywołuje on odpowiednie procesy robocze dla zadań optymalizacji i/lub symulacji. Wszystkie wątki odpalane przez główny proces tego poziomu aplikacji są uruchamiane przez system Tango Controls. Kolorem czerwonym oznaczono wątki związane z komunikacją synchroniczną i asynchroniczną, kolorem żółtym - zdarzeniową, a fioletowym wątek sygnałowy (dokładniejszy opis znajduje się w [18]).

Użyto w tym celu procesów, a nie wątków ze względu na tzw. *Global Interpreter Lock*. Jest to cecha języka Python, która powoduje, że wątki nie mogą być uruchamiane równolegle (nawet w przypadku dostępnych rdzeni procesora) - zawsze są tylko współbieżne. Procesy nie są objęte tym ograniczeniem i można ich używać w prawdziwie równoległy sposób.

### 4.1.3. Wybór pakietu do optymalizacji dynamicznej

Podstawowym elementem istotnym dla powodzenia zadań postawionych przed wyższym poziomem aplikacji był wybór pakietu realizującego operacje optymalizacji dynamicznej. Istnieje na rynku kilka płatnych bibliotek bądź pakietów oprogramowania, które są w stanie rozwiązywać tego typu problemy (np. MATLAB, Mathematica itp.), ale postanowiono poszukać darmowego odpowiednika, który posiadałby interfejs w języku programowania Python (w związku z ogólnymi założeniami co do struktury tej części aplikacji).

#### 4.1.3.1. Opis pakietu ACADO

Pierwszym z pakietów optymalizacyjnych, który był rozważany, jest ACADO (ang. *Automatic Control and Dynamic Optimization*). Jest to zestaw algorytmów napisanych w języku C++ służących do różnych zastosowań związanych ze sterowaniem optymalnym, dostępny na otwartej licencji (za: [13]). W jego skład wchodzą między innymi metody:

- bezpośrednie optymalizacji dynamicznej (w tym wielokryterialnej),
- sterowania predykcyjnego (MPC, ang. *Model Predictive Control*),
- estymacji parametrów i stanu.

Dodatkowo opisywane oprogramowanie umożliwia generowanie kodu w języku C rozwiązującego konkretny problem, który może być użyty w mikrokontrolerze. Oprócz tego istnieje również możliwość użycia go w programie MATLAB, jako dodatkowego pakietu.

Twórcom tej biblioteki przyświecały cztery główne założenia:

1. Oprogramowanie musi być dostępne na otwartej licencji.
2. Składnia musi być jak najbardziej intuicyjna, aby nie odrzucać potencjalnych użytkowników (które mogą nie mieć dużego doświadczenia z językiem C++) ilością kodu szablonowego (ang. *boilerplate code*). Powinna umożliwiać prostą definicję problemów sterowania optymalnego. Udało się to osiągnąć, stosując różne bardziej zaawansowane elementy języka programowania, w szczególności przeładowywania operatorów, aby składnia opisu w ACADO była jak najbardziej zbliżona do zapisu matematycznego.
3. Kod powinien być napisany w taki sposób, aby ułatwić rozszerzanie go o własne algorytmy oraz używanie metod z tej biblioteki w innych, większych projektach. To założenie jest spełnione przez odpowiednie zaplanowanie struktury oprogramowania oraz przestrzeganie paradygmatu programowania obiektowego.
4. Brak zewnętrznych zależności. Użytkownik może zastosować ACADO, jeśli tylko posiada kompilator języka C++. Dodatkowe biblioteki nie są wymagane, ale mogą być użyte przy rysowaniu wykresów oraz specjalistycznych obliczeniach algebraicznych.

Aby zilustrować prostotę składni, poniżej załączono definicję modelu matematycznego opisywanego w niniejszej pracy zestawu zbiorników. Znajduje się tu definicja stanu, sterowania i parametrów (linie 4 - 6), samego równania różniczkowego (linie 8 - 15) oraz problemu optymalizacji wraz z ograniczeniami (linie 20 - 38).

```

1   // INTRODUCE THE VARIABLES:
2   // -----
3
4   DifferentialState h1, h2, h3;
5   Control           u      ;
6   Parameter          T      ;
7
8   DifferentialEquation f( 0.0, T );
9
10
11  // DEFINE A DIFFERENTIAL EQUATION:
12  // -----
13  f << dot(h1) == (u - C1*sqrt(h1))/a*w;
14  f << dot(h2) == (C1*sqrt(h1) - C2*sqrt(h2))/(c*w + h2*b*w/h_max);
15  f << dot(h3) == (C2*sqrt(h2) - C3*sqrt(h3))/(w*sqrt(pow(R,2) - pow((R-h3),2)));
16
17
18  // DEFINE AN OPTIMAL CONTROL PROBLEM:
19  // -----
20  float t_max = 100.0;
21  OCP ocp( 0, T, t_max );
22
23  ocp.minimizeMayerTerm( T );
24  ocp.subjectTo( f );
25
26  ocp.subjectTo( AT_START, h1 == h10 );
27  ocp.subjectTo( AT_START, h2 == h20 );
28  ocp.subjectTo( AT_START, h3 == h30 );
29
30  ocp.subjectTo( AT_END , h1 == h1_final );
31  ocp.subjectTo( AT_END , h2 == h2_final );
32  ocp.subjectTo( AT_END , h3 == h3_final );
33
34  ocp.subjectTo( 0.0 <= h1 <= h_max );
35  ocp.subjectTo( 0.0 <= h2 <= h_max );
36  ocp.subjectTo( 0.001 <= h3 <= h_max ); //to avoid hitting singularity
37  ocp.subjectTo( 0.0 <= u <= u_max );
38  ocp.subjectTo( 20.0 <= T <= t_max );

```

Mimo iż pakiet ACADO jest napisany w języku C++, a nie w języku Python, została przeprowadzona próba użycia go do rozwiązania problemu znajdowania sterowania czasooptymalnego w opisywanym w

niniejszej pracy układzie zbiorników. Wykorzystano tutaj metodę bezpośrednią strzałów wielokrotnych opartą na programowaniu nieliniowym. Niestety, nie zakończyła się ona powodzeniem: bez względu na zmiany opcji algorytmu (w tym tolerancji rozwiązania) oraz na zastosowane techniki inicjalizacji nie udało się przeprowadzić ani jednej poprawnej optymalizacji. W związku z tym zdecydowano się zrezygnować z dalszych prób użycia pakietu ACADO i poszukać oprogramowania bardziej odpowiedniego do rozpatrywanego problemu.

#### 4.1.3.2. Opis pakietu Modelica i JModelica.org

Dalsze poszukiwania oprogramowania do optymalizacji dynamicznej przyniosły odkrycie pakietu Modelica. Jest to obiektowy język opisu dynamiki układów fizycznych oparty na definicji równań i matematycznych zależności między nimi (za: [23]). Pozwala on na definicję równań różniczkowych i algebraicznych, zarówno ciągłych, jak i dyskretnych. Celem przyświecającym jego twórcom jest dostarczenie narzędzia służącego do prostego przedstawiania w formie tekstowej i graficznej komponentów układów fizycznych:

- mechanicznych (jedno-, dwu- i trójwymiarowych),
- elektrycznych,
- termodynamicznych,
- hydraulicznych,
- pneumatycznych.

Opisywany układ może zawierać dowolną kombinację komponentów każdego z tych rodzajów. Każdy z nich ma odpowiadającą sobie część biblioteki standardowej języka Modelica, która oprócz tego zawiera również ogólne komponenty (m.in. definicje jednostek układu SI, maszyn stanów, zaawansowanych funkcji matematycznych czy stałych fizycznych). Zgodnie z paradygmatem obiektowości, wszystkie komponenty można rozszerzać, aby tworzyć własne ich wariacje. Całe oprogramowanie jest dostępne na własnej, otwartej licencji, która również może być wykorzystywana do licencjonowania modeli napisanych w tym języku.

Poniżej podano definicję modelu opisywanego w niniejszej pracy układu zbiorników w języku Modelica w formie tekstowej, aby zaprezentować jego składnię. Przygotowano paczkę `TanksPkg` rozszerzającą bibliotekę standardową o następujące komponenty:

- model częściowy zawierający definicje parametrów i równań różniczkowych opisujących układ (na podstawie wzoru 1.9), ale bez wejścia sterowania  $u$  (linie 3 - 33),
- model podstawowy rozszerzający powyższy o definicję wejścia sterowania (linie 35 - 37),
- model inicjalizacyjny służący do wyznaczania punktów równowagi układu (linie 39 - 45).

Każdy z nich zawiera definicję parametrów i zmiennych określającą ich typy (i jednostki, jeśli to możliwe) oraz równania (lub równania początkowe w przypadku modelu inicjalizacyjnego). Wyjątkiem jest tutaj tylko model podstawowy, który tylko rozszerza model częściowy i nie zawiera żadnych dodatkowych definicji poza redefinicją typu parametru  $u$ .

```

1 package TanksPkg
2     partial model ThreeTanksNoInput
3         parameter Modelica.SIunits.Length a = 31.0 "1st tank's width" annotation(
4             Evaluate=true);
5         parameter Modelica.SIunits.Length b = 40.0 "2nd tank's triangular part's
6             width" annotation(Evaluate=true);
7         parameter Modelica.SIunits.Length c = 10.0 "2nd tank's rectangular part's
8             width" annotation(Evaluate=true);
9         parameter Modelica.SIunits.Length R = 40.0 "3rd tank's side's radius"
10            annotation(Evaluate=true);
11        parameter Modelica.SIunits.Length w = 5.0 "Tanks' depth" annotation(
12            Evaluate=true);
13        parameter Modelica.SIunits.Length h_max = 40.0 "Maximum height of liquid
14            in tanks" annotation(Evaluate=true);
15
16        parameter Modelica.SIunits.VolumeFlowRate C1 = 26.0 "1st tank's outflow
17            rate";
18        parameter Modelica.SIunits.VolumeFlowRate C2 = 26.0 "2nd tank's outflow
19            rate";
20        parameter Modelica.SIunits.VolumeFlowRate C3 = 28.0 "3rd tank's outflow
21            rate";
22
23        parameter Real alpha1 = 0.5 "1st tank's flow coefficient";
24        parameter Real alpha2 = 0.5 "2nd tank's flow coefficient";
25        parameter Real alpha3 = 0.5 "3rd tank's flow coefficient";
26
27        parameter Modelica.SIunits.Length h10 = 20.0 "1st tank's initial value";
28        parameter Modelica.SIunits.Length h20 = 20.0 "2nd tank's initial value";
29        parameter Modelica.SIunits.Length h30 = 20.0 "3rd tank's initial value";
30
31        parameter Real u_max = 20.0 "Maximum control value";
32
33        Real h1(start=h10, fixed=true, min=0, max=h_max);
34        Real h2(start=h20, fixed=true, min=0, max=h_max);
35        Real h3(start=h30, fixed=true, min=0, max=h_max);
36
37        replaceable Real u;
38
39        equation
40            der(h1)=(u - C1*(h1^alpha1))/(a*w);
41            der(h2)=(C1*(h1^alpha1) - C2*(h2^alpha2))/(c*w + h2*b*w/h_max);
42            der(h3)=(C2*(h2^alpha2) - C3*(h3^alpha3))/(w*sqrt(R^2 - (R - h3)^2));
43
44    end ThreeTanksNoInput;

```

```

34
35   model ThreeTanks
36     extends ThreeTanksNoInput (redeclare Modelica.Blocks.Interfaces.RealInput u
37       );
38
39   end ThreeTanks;
40
41   model ThreeTanksInit
42     extends ThreeTanks (h1(fixed=false) , h2(fixed=false) , h3(fixed=false));
43     initial equation
44       der(h1) = 0;
45       der(h2) = 0;
46       der(h3) = 0;

```

Modelica jest wykorzystywana jako podstawa opisu modeli w różnego rodzaju komercyjnym oprogramowaniu do symulacji (m.in. CATIA, Dymola czy MapleSim), ale są dostępne również jego bezpłatne odpowiedniki: OpenModelica i JModelica.org. Drugi z nich zasługuje na szczególną uwagę ze względu na dwa wzgłydy:

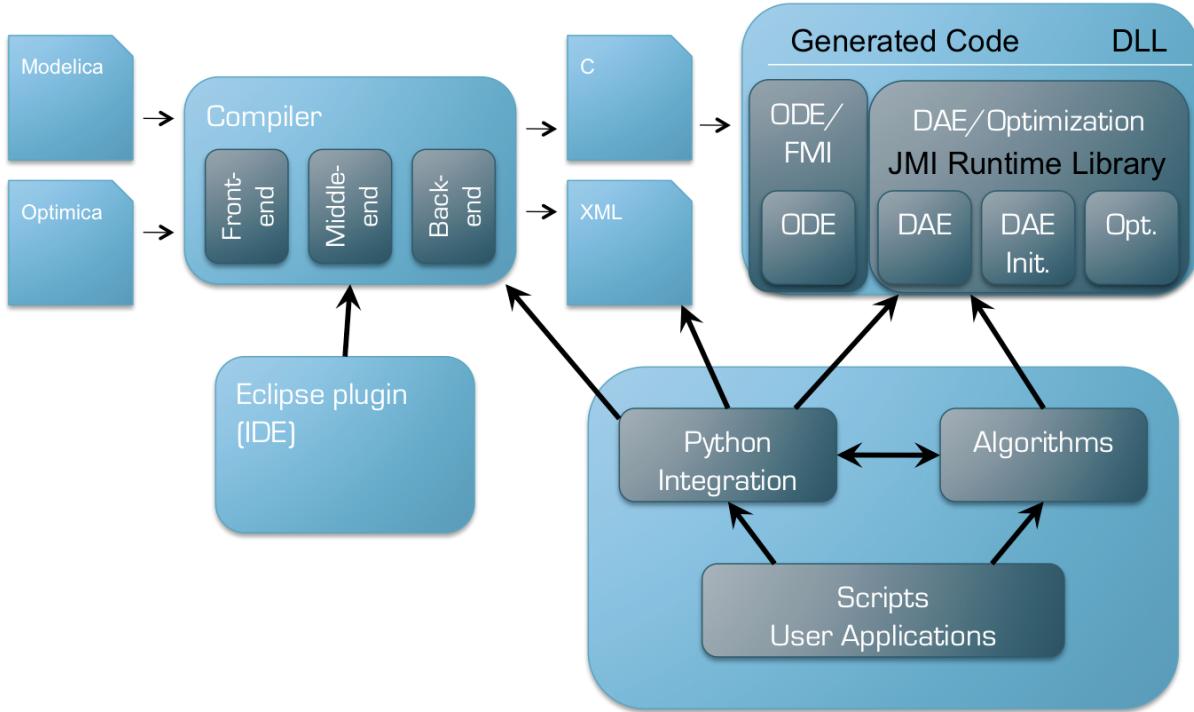
- rozszerzenie biblioteki Modelica o opis problemów optymalizacji,
- dostępność API w języku Python do modeli napisanych w języku Modelica.

Ze względu na te fakty podjęta została próba wykorzystania go w wyższym poziomie aplikacji.

JModelica.org jest platformą do symulacji, optymalizacji i analizy układów opisanych w języku Modelica. Jak wspomniano wcześniej, dostarcza również możliwości opisu problemów optymalizacji dynamicznej i metody do ich rozwiązywania zebrane we wspólnym pakiecie Optimica (dokładniejszy opis w [15]). Jest dostępna na otwartej licencji (GPL w wersji 3) i utrzymywana przez firmę Modelon AB, choć sam pomysł stworzenia takiej platformy powstał na Uniwersytecie w Lund (za: [22]).

Architektura platformy JModelica.org składa się z następujących komponentów (pokazanych schematycznie na rys. 4.3):

- kompilatora, który sprawdza poprawność modeli opisanych w językach Modelica i Optimica, a następnie generuje kod w języku C zawierający równania układów i pliki w formacie XML z metadanymi,
- biblioteki wykonywalnej JModelica.org napisanej w języku C, która pozwala skompilować model i dostarczyć go do zewnętrznego oprogramowania do wyliczania pochodnych w formie przez nie zrozumiałej,
- bibliotek w językach C++, Java i Python zawierającej implementacje metod optymalizacji dynamicznej (używana jest tu metoda kolokacji bezpośredniej opisana w [9] i w [11]),
- API do języka Python dającego użytkownikowi dostęp do wszystkich pozostałych komponentów; w jego skład wchodzą 3 pakiety:



Rys. 4.3. Schemat architektury platformy JModelica.org. Źródło: [22].

- PyModelica dostarcza interfejs do kompilatorów języków Modelica i Optimica,
- PyFMI umożliwia manipulację parametrami modeli i ich symulację,
- PyJMI służy jako interfejs do algorytmów optymalizacji;
- wtyczki do środowiska programistycznego Eclipse.

Poniżej pokazano definicję problemu optymalizacji rozważanego w niniejszej pracy napisaną w języku Optimica. Jest ona rozszerzeniem modelu podstawowego opisanego wyżej i zawiera dodatkowe definicje wskaźnika jakości, parametrów i ograniczeń zawartych w problemie. Jest ona również częścią przygotowanej paczki TanksPkg.

```

1   optimization three_tanks_time_optimal(objective=finalTime, startTime=0,
2                                         finalTime(free=true,initialGuess=50))
3   extends ThreeTanks(u(free=true, initialGuess=20.0));
4
5   parameter Real h1_final = 5.0;
6   parameter Real h2_final = 5.0;
7   parameter Real h3_final = 5.0;
8
9   constraint
10    h1(finalTime)=h1_final;
11    h2(finalTime)=h2_final;
12    h3(finalTime)=h3_final;
13    finalTime >= 0;

```

```

14      h1 <= h_max;
15      h1 >= 0;
16      h2 <= h_max;
17      h2 >= 0;
18      h3 <= h_max;
19      h3 >= 0.00001;
20      u>=0;
21      u<=u_max;
22 end three_tanks_time_optimal;

```

Ze względu na opisane cechy platformy JModelica.org oraz na fakt, iż stosowano ją do rozwiązywania podobnych problemów (np. [8]), zdecydowano się zastosować ją jako oprogramowanie rozwiązujące rozważany w niniejszej pracy problem sterowania czasooptymalnego. Pozwala ona również na linearyzację modeli, co zostało wykorzystane w celu przygotowania do obliczenia nastaw regulatora liniowo-kwadratowego.

#### 4.1.4. Środowisko testowe części obliczeniowej

Ze względu na długi i skomplikowany proces instalacji pakietu JModelica.org z jego zależnościami zdecydowano przygotować przenośne środowisko testowe części obliczeniowej opisywanego w niniejszej pracy oprogramowania. W tym celu użyto platformy Docker służącej do zarządzania kontenerami. Są one lekkimi opakowaniami na aplikacje i ich zależności symulującymi zachowanie systemu operacyjnego, które są gotowe do uruchomienia przez silnik Dockera. Do ich zapisu używa się specjalnego systemu kontroli wersji, który zachowuje konkretny stan kontenera (nazywany obrazem), aby można go było w szybki sposób odtworzyć. Oprócz tego istnieje również możliwość grupowania kontenerów w środowiska pracy, które uruchamiane są jednym poleciением.

Jest to idealne środowisko do testowania aplikacji wymagających wielu zależności, w tym również sieciowych. Na potrzeby części obliczeniowej przygotowano obraz platformy Docker zawierający:

- oprogramowanie JModelica.org wraz z zależnościami:
  - pakiet do optymalizacji nieliniowej IPOPT (ang. *Interior Point Optimizer*) oraz jego zależności:
    - \* bibliotekę operacji wektorowych i macierzowych BLAS (ang. *Basic Linear Algebra Subprograms*),
    - \* bibliotekę operacji algebraicznych LAPACK (ang. *Linear Algebra Package*),
    - \* bibliotekę opisu problemów matematycznych AMPL/MP,
    - \* oprogramowanie do rozwiązywania układów liniowych MUMPS;
  - pakiety do wyliczania wartości pochodnych funkcji CppAD i CasADI,
  - bibliotekę do rozwiązywania równań różniczkowych i algebraicznych Sundials (ang. *Suite of Nonlinear and Differential/Algebraic Equation Solvers*),

- pakiet funkcji matematycznych w języku Python SciPy;
- oprogramowanie Tango Controls w wersji klienckiej (bez serwera do komunikacji z bazą danych) wraz z zależnościami:
  - pakiet implementujący architekturę komunikacji CORBA omniORB,
  - pakiet do komunikacji zdarzeniowej ZeroMQ,
  - bibliotekę obliczeń numerycznych języka Python NumPy;
- pakiet PyTango, czyli API systemu Tango Controls w języku Python,
- pakiet *control*, który został wykorzystany do wyznaczenia regulatora liniowo-kwadratowego.

Przygotowano również środowisko pracy zawierające następujące kontenery:

- bazę danych z silnikiem MariaDB,
- serwer urządzeń DatabaseDS systemu Tango Controls służący jako host systemu,
- opisany wyżej kontener służący do uruchamiania części obliczeniowej aplikacji.

Takie podejście pozwoliło uniezależnić testowanie tego poziomu aplikacji od posiadanego sprzętu - jedynym oprogramowaniem potrzebnym do uruchomienia opisanego środowiska pracy jest sama platforma Docker.

## 4.2. Część realizująca sterowanie bezpośredni („niższego poziomu”)

Ze względu na dostępność sprzętu służącego do sterowania fizycznym układem zbiorników, wybór technologii mogącej stanowić podstawę niższego poziomu aplikacji był ograniczony do dwóch możliwości.

Stanowisko laboratoryjne jest wyposażone w komputer, który za pomocą karty przemysłowej RTD-A-C/PCI komunikuje się z układem czujników i pompą. Komputer komunikuje się z kartą za pomocą pakietu MATLAB/Simulink i to właśnie jest pierwsza opcja oprogramowania do komunikacji z urządzeniami wykonawczymi.

Alternatywnie, można skorzystać ze sterownika PLC firmy GE Fanuc z serii Versa Max. Wybrano pierwszą możliwość ze względu na fakt, iż pakiet MATLAB/Simulink umożliwia symulację modelu zbiorników, w której można przeprowadzić dodatkową weryfikację sterowania czasooptymalnego obliczonego przez aplikację wyższego poziomu.

#### 4.2.1. Pakiet MATLAB/Simulink jako narzędzie realizujące sterowanie bezpośredni

Oprogramowanie firmy Mathworks jest jedynym z najbardziej popularnych środowisk do wykonywania obliczeń i analizy ich wyników, tworzenia interfejsów użytkownika, symulacji i sterowania urządzeniami zewnętrznymi. Zawiera ono zestaw metod numerycznych realizujących szerokie spektrum zadań z różnych obszarów inżynierii. Definiuje również własny język oprogramowania zbliżony do języka Java cechujący się podejściem obiektowym i prostym zapisem operacji macierzowych.

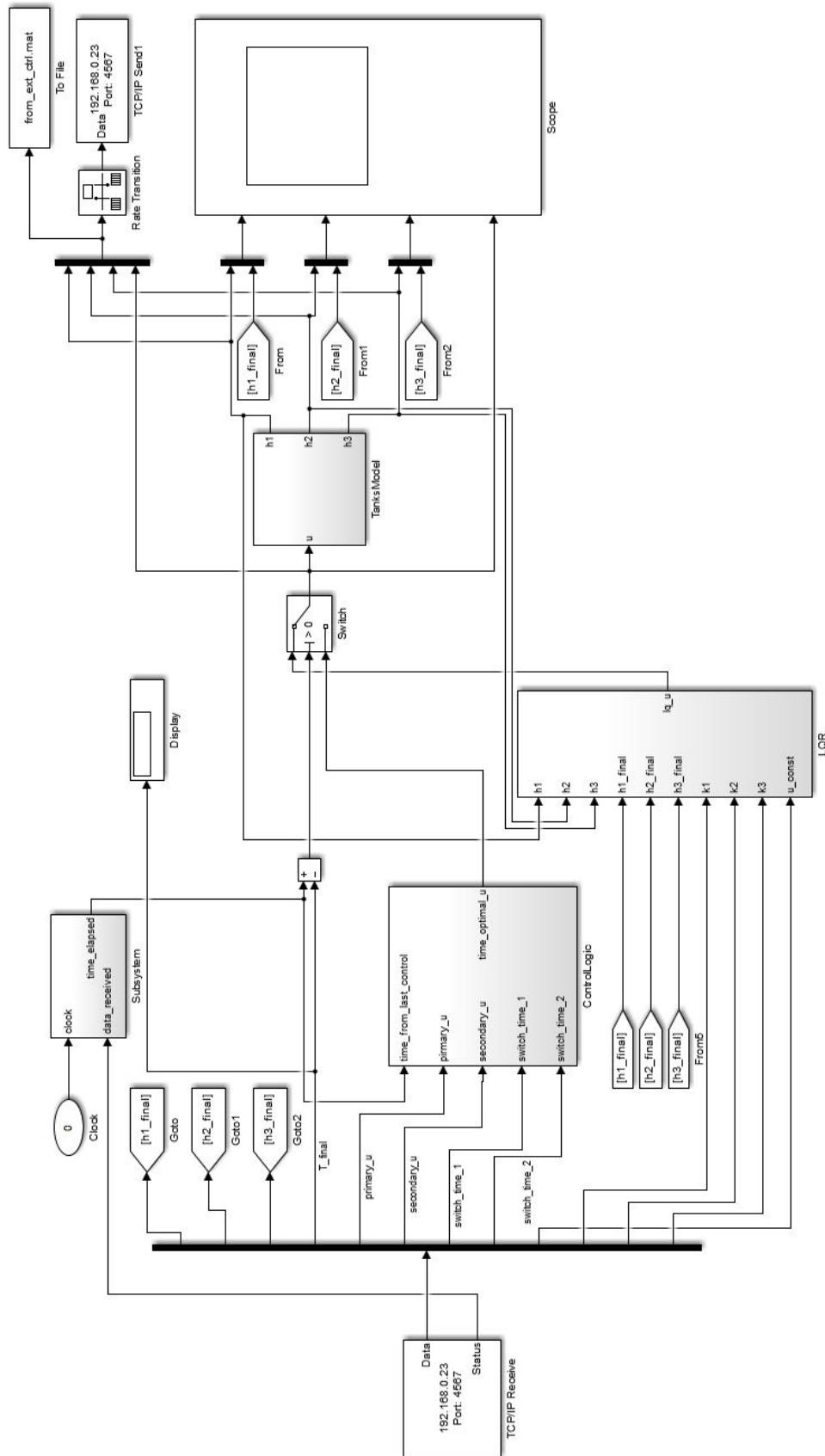
Pakiet Simulink, będący częścią oprogramowania MATLAB, umożliwia opis układów fizycznych przy pomocy języka graficznego, którego modele składają się z bloczków opisujących operacje i połączeń między nimi. Daje dostęp do całkowania takich modelów przy użyciu różnych metod, zarówno z czasem ciągłym, jak i dyskretnym. Poza tym można używać go do realizacji sterowania bezpośredniego albo przez bezpośrednie połączenie z urządzeniami wykonawczymi, albo przez generację kodu opisującego modele w języku C lub VHDL na odpowiednie platformy sprzętowe. Niestety, to oprogramowanie jest dostępne tylko na płatnych licencjach różnego typu.

Istnieje wiele przykładów zastosowań pakietu MATLAB/Simulink w celu realizacji zadań sterowania czasooptymalnego (np. [14]), ale ze względu na założoną z góry hybrydowość aplikacji opisywanej w niniejszej pracy, nie zdecydowano się na wykorzystanie go również w takim celu.

Przygotowano model w programie Simulink, pokazany na rys. 4.4, który stanowi niższy poziom aplikacji. Zawiera on następujące komponenty:

- odbiór danych z wyższego poziomu aplikacji za pomocą protokołu TCP,
- układ wyznaczający czas rozpoczęcia realizacji zadania sterowania czasooptymalnego,
- układ wyznaczający aktualną wartość sterowania czasooptymalnego na podstawie czasów przełączeń,
- układ wyznaczający sterowanie liniowo-kwadratowe,
- blok wybierający sterowanie do zaaplikowania w modelu (kryterium przełączenia jest czas będący wartością wskaźnika jakości w zadaniu czasooptymalnym),
- podsystem zawierający definicję modelu układu zbiorników,
- wysyłanie danych uzyskanych w symulacji (aktualnych poziomów wody w zbiornikach i sterowania) do wyższego poziomu aplikacji i zapisywanie ich do pliku w celu późniejszej analizy,
- blok pozwalający na rysowanie wykresów trajektorii systemu w czasie symulacji.

Przeprowadzone testy komunikacji między poziomami aplikacji wykazały, że serwer urządzeń systemu Tango Controls, będący wyższym jej poziomem, nie jest w stanie obsłużyć wysyłanych w każdym kroku symulacji danych. Zastosowano więc blok wyzwalający wysyłanie z mniejszą częstotliwością. Dobrano eksperymentalnie wartość 100 ms jako okres wysyłania, ponieważ stwierdzono, że dynamika



Rys. 4.4. Model niższego poziomu aplikacji w programie Simulink. Źródło: własne.

układu jest stosunkowo wolno zmienna, a więc taki okres nie spowoduje większych błędów w obliczaniu sterowania czasooptymalnego przez część obliczeniową aplikacji.

Na potrzeby niniejszej pracy nie prowadzono żadnych badań eksperymentalnych z rzeczywistym układem zbiorników i systemem pomiarowo-sterującym, ale niższy poziom aplikacji został przygotowany do tego, żeby takie badania przeprowadzić w przyszłości.

## **5. Badania symulacyjne**

W niniejszym rozdziale opisano przeprowadzone eksperymenty optymalizacyjne i symulacyjne (przy użyciu platform JModelica.org oraz MATLAB/Simulink), przedstawiono wpływ różnych parametrów na oba te procesy oraz zdiagnozowano problemy mające wpływ na ich funkcjonowanie na obu poziomach aplikacji.

### **5.1. Optymalizacja przy użyciu pakietu JModelica.org**

Podstawowym zadaniem wyższego poziomu aplikacji było przeprowadzenie optymalizacji w celu wyznaczenia sterowania czasooptymalnego. Jak wspomniano w sekcji 4.1.3, wybrano do tego celu platformę JModelica.org.

#### **5.1.1. Opis algorytmu optymalizacji dynamicznej**

Algorytm wykorzystywany do optymalizacji dynamicznej można ogólnie opisać jako bezpośrednią metodę kolokacji. Polega ona na aproksymacji układu równań różniczkowych w dyskretnych chwilach czasu i konstrukcji na tej podstawie zadania programowania nieliniowego, zwykle o rzadkiej strukturze. Jest napisana w języku Python i używa biblioteki CasADi do wyznaczenia wartości pochodnych funkcji oraz oprogramowania IPOPT do obliczenia wynikowego zagadnienia programowania nieliniowego. Można z niej korzystać pod warunkiem, że model matematyczny nie zawiera nieciągłości (za: [15]). W związku z tym, że w rozważanym modelu układu zbiorników występuje osobliwość dla zerowego poziomu w trzecim zbiorniku, należało unikać trajektorii zawierających ten punkt. Rzeczywiście, przy próbie uruchomienia optymalizacji dla warunku początkowego niedaleko tego punktu, algorytm zwrócił błąd niepoprawnej wartości pochodnej. Nie można więc stosować tego algorytmu do napełniania zbiorników od stanu zerowego, co zmniejsza jego praktyczne zastosowanie.

Programy rozwiązujące problemy NLP wykazują się dużo większą zbieżnością w przypadku, gdy dostarczy się im informacje o pochodnych pierwszego i drugiego rzędu (za: [11]). W związku z tym w użytym algorytmie użyto najpierw oprogramowania CasADi, aby ze skompilowanego modelu w języku Modelica wyznaczyć wartości tych pochodnych (w formie jakobianu), a następnie używa się metody elementów skończonych. Dzieli ona wejściowy czas działania na określoną liczbę przedziałów nazywanych elementami skończonymi i dostarcza aproksymacji nieznanych wartości sterowania i stanów systemu

w określonej liczbie punktów kolokacji. Wpływ liczby elementów na jakość rozwiązania jest przedstawiony w sekcji 5.2.1. W każdym punkcie kolokacji zastosowano pakiet IPOPT, który rozwiązuje zadanie w postaci danej wzorem 5.1 za pomocą metody punktu wewnętrznego. Metoda ta wykorzystuje funkcję graniczną do poruszania się tylko wewnątrz zbioru dopuszczalnego, gdzie metodą Newtona szuka lokalnego minimum.

$$\min_{x \in [x_l, x_u] \wedge g_l \leq g(x) \leq g_u} f(x) \quad (5.1)$$

Algorytm optymalizacyjny zwraca wektor sterowań oraz trajektorie układu w czasie będącym wartością wskaźnika jakości w tym zagadnieniu. Te wartości są prezentowane użytkownikowi jako odpowiednie atrybuty urządzenia systemu Tango Controls.

Brak uwzględnienia ograniczenia czasu końcowego  $T \geq 0$  w niektórych przypadkach skutkowało znalezieniem sterowania z ujemnym czasem optymalnym. Wynika z tego, że opisana metoda nie jest wrażliwa na to, że symuluje fizyczny układ, co może dziwić, zważywszy na fakt, iż została napisana dla oprogramowania służącego tylko do symulacji. To ograniczenie zostało uwzględnione w definicji optymalizacji w języku Optimica, które pokazano w sekcji 4.1.3, aby uniknąć pojawiania się ujemnego czasu optymalnego.

### 5.1.2. Inicjalizacja optymalizacji dynamicznej

Kluczowym aspektem funkcjonowania algorytmów optymalizacji dynamicznej jest odpowiednie określenie punktu początkowego, czyli początkowej struktury sterowania i wygenerowanej przez nią trajektorii układu. Jest to zagadnienie szeroko opisywane w literaturze, m.in w [4], [10], [16], [12] oraz [15]. Początkowo próbowało inicjalizować algorytm stałym sterowaniem, którego wartość jest niezależna od punktów początkowych i końcowych. Niestety, takie podejście okazało się błędne i nie udało się wyznaczyć żadnego sterowania optymalnego.

W związku z tym przyjęto następujący algorytm inicjalizacyjny:

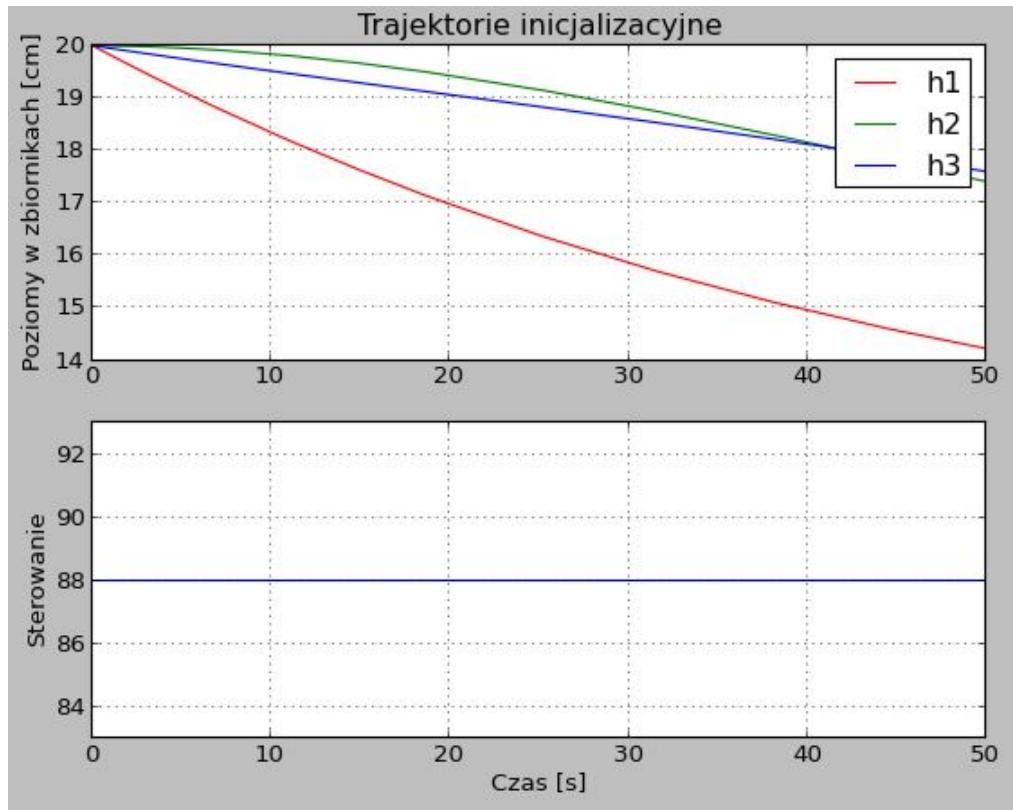
- dla każdej z trzech wartości poziomów docelowych wyznacz wartość sterowania ustalonego na podstawie odwrotności wzoru 1.13 danego wzorem 5.2,

$$u_i = C_i h_i^{\alpha_i} \quad (5.2)$$

- wyznacz wartość średnią z trzech sterowań ustalonych, wyznaczonych w poprzednim kroku według wzoru 5.3

$$u_r = \frac{1}{3} \cdot \sum_{i=1}^3 u_i \quad (5.3)$$

- przeprowadź symulację z czasem końcowym podanym jako wartość właściwości *SimulationFinal-time* (opisanej w sekcji 4.1.2),
- podaj wyniki uzyskane w symulacji jako trajektorie inicjalizacyjne do algorytmu optymalizacji.



Rys. 5.1. Przykładowa trajektoria początkowa dla algorytmu optymalizacji. Źródło: własne.

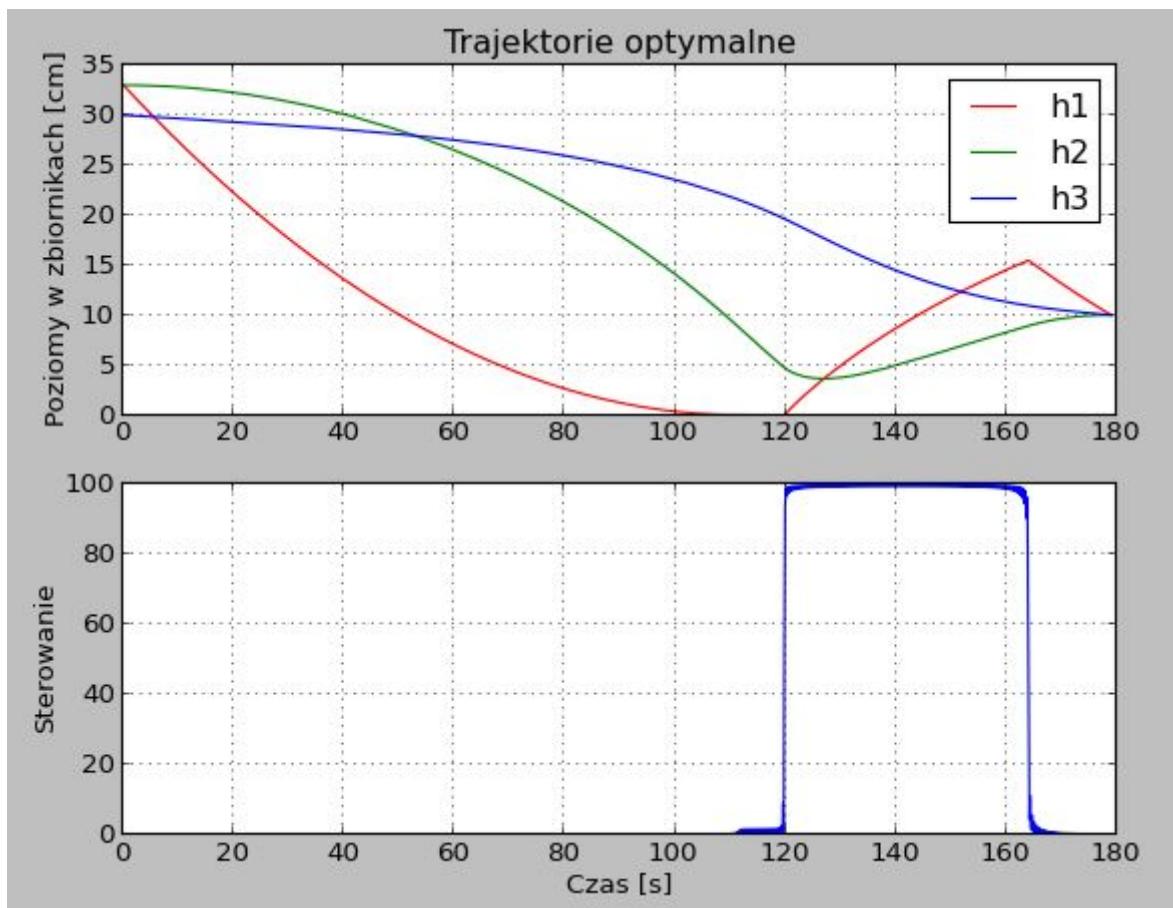
Rysunek 5.1 przedstawia przykładowe sterowanie ustalone i przebiegi zmiennych stanu uzyskane według powyższego algorytmu.

### 5.1.3. Uzyskana postać sterowania

W związku z tym, że użyte oprogramowanie jest w stanie dokonywać optymalizacji sterowania w układach nieliniowych, nie ma możliwości, aby ograniczyć postać sterowania tylko do postaci „bang-bang”. Sterowanie wyznaczane przez opisany wyżej algorytm optymalizacji jest wektorem o liczbie elementów danej wzorem 5.4, gdzie  $k$  to liczba punktów kolokacji, a  $e$  to liczba elementów skończonych. Dodatkowy element wektora sterowania bierze się od wartości początkowej.

$$l_u = k \cdot e + 1 \quad (5.4)$$

Niezależnie od tego okazało się, że sterowanie optymalne wyliczone przy pomocy pakietu JModelica.org ma postać zbliżoną do „bang-bang” ze spodziewaną liczbą dwóch przełączeń. Dowodzi to, iż założenie opisane w sekcji 2.1.2 jest prawdziwe, przynajmniej dla tych stanów początkowych i docelowych, dla których udało się wyznaczyć poprawnie sterowanie optymalne w ramach testów prowadzonych na potrzeby niniejszej pracy.



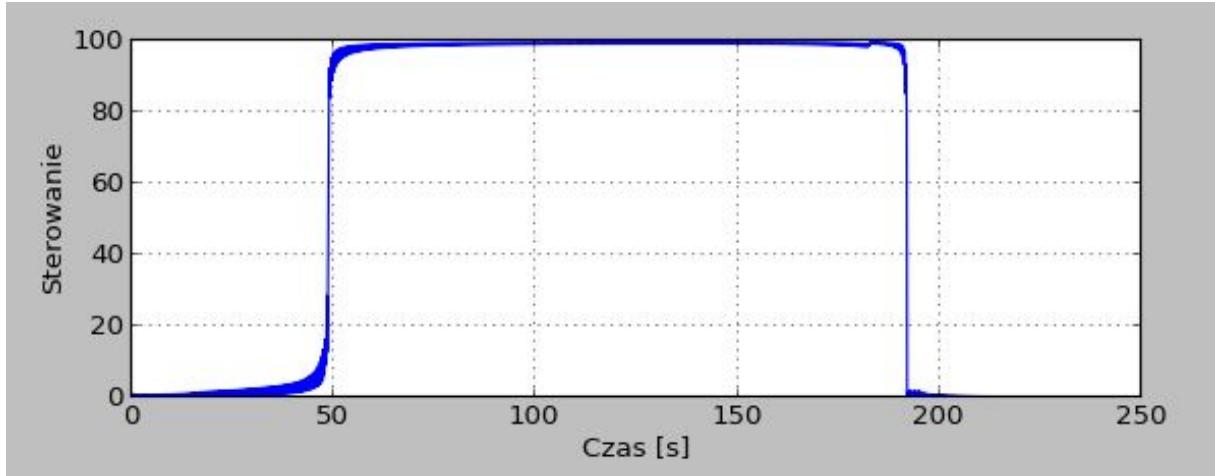
Rys. 5.2. Przykładowy wynik optymalizacji. Źródło: własne.

Rysunek 5.2 przedstawia przykładowe sterowanie optymalne oraz trajektorie układu przez nie wygenerowane. Można na nim zauważyć, że sterowanie nie jest dokładnie postaci „bang-bang”, ale jest to niego zbliżone.

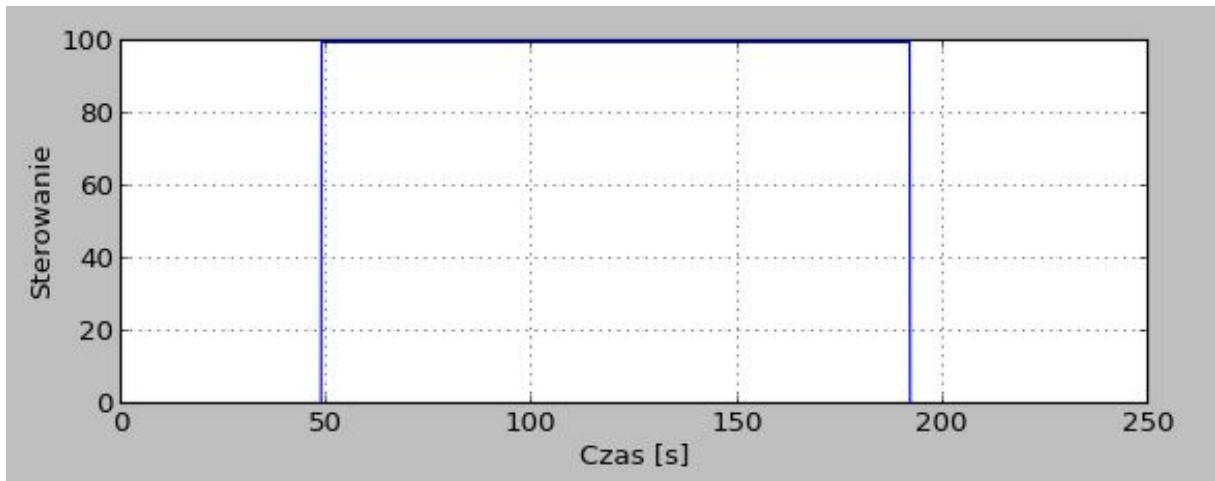
Przyjęto więc prostą metodę normalizacji sterowania do postaci „bang-bang”: każdemu punktowi powyżej połowy różnic między ograniczeniami nałożonymi na sterowanie zostaje przypisana wartość maksymalna, a każdemu poniżej - minimalna. Jeśli występuje tylko jeden czas przełączenia, to dodaje się drugi w ostatniej chwili czasu optymalizacji, a więc w momencie odpowiadającym wartości czasu optymalnego. Taka operacja pozwala wyznaczyć czasy przełączeń potrzebne niższemu poziomowi aplikacji, jeśli tylko sterowanie otrzymane z algorytmu optymalizacji ma spodziewaną strukturę. Przykład działania operacji normalizacji jest dany poniżej: przedstawiono sterowanie w postaci „surowej” obliczonej przez algorytm optymalizacyjny i po normalizacji odpowiednio na rysunkach 5.3 i 5.4.

#### 5.1.4. Dokładność wyznaczania rozwiązania

Jedynym parametrem, który wyznacza dokładność wyznaczania rozwiązania przez pakiet JModelica.org jest relatywna tolerancja zbieżności programu IPOPT, którą można ustawić we właściwości *IPOPTTolerance* (opisanej w sekcji 4.1.2). Domyślną wartością jest  $10^{-5}$ , która jednak okazała się zbyt



Rys. 5.3. Przykład sterowania „surowego” otrzymanego w wyniku optymalizacji. Źródło: własne.



Rys. 5.4. Przykład sterowania znormalizowanego. Źródło: własne.

mała dla opisywanego problemu i prowadziła do błędów zbieżności i niemożliwości wyznaczenia odpowiedniego rozwiązania. Dobrano doświadczalnie wartość  $10^{-3}$  ze względu na fakt, iż w tym przypadku niemożliwość wyliczenia sterowania optymalnego zdarzała się relatywnie rzadko.

Dodatkowo wartość wspomnianej właściwości *IPOPTTolerance* jest również wykorzystywana do sprawdzenia, czy rozwiązanie zwrócone przez algorytm optymalizacyjny jest poprawne. Sprawdza się, czy odległość między wartościami końcowymi trajektorii systemu a stanem docelowym jest mniejsza niż wartość tej właściwości.

Opisane wyżej algorytmy inicjalizacji i optymalizacji pozwoliły osiągnąć zadowalające wyniki dla stanów znajdujących się blisko punktów równowagi układu. Nie przeprowadzono jednak pełnej analizy stanów osiągalnych systemu, a więc nie stwierdzono, czy brak poprawnego wyniku dla stanów docelowych bardziej oddalonych od punktów równowagi jest związany błędny działaniem algorytmu, czy z faktem, iż te stany nie są osiągalne.

## 5.2. Symulacja i weryfikacja

Aby sprawdzić poprawność działania algorytmu optymalizacji dynamicznej, wykonano symulacje weryfikacyjne przy użyciu oprogramowania JModelica.org na wyższym poziomie aplikacji oraz środowiska MATLAB/Simulink na niższym. W obu tych przypadkach błąd weryfikacji jest liczony według wzoru 5.5, gdzie  $h_i^s(T)$  to wartość odpowiedniego poziomu po upływie czasu optymalnego uzyskana w symulacji, a  $h_i^f$  to wartość docelowa tego poziomu.

$$e_{wer} = \sum_{i=1}^3 (h_i^s(T) - h_i^f)^2 \quad (5.5)$$

Należy zwrócić uwagę na to, iż weryfikacja poprawności rozwiązania korzysta z innych danych, niż sprawdzenie dokładności rozwiązania opisane w sekcji 5.1.4. Pierwsza procedura używa danych symulacyjnych uzyskanych po optymalizacji, a druga - danych pochodzących bezpośrednio z algorytmu optymalizacyjnego.

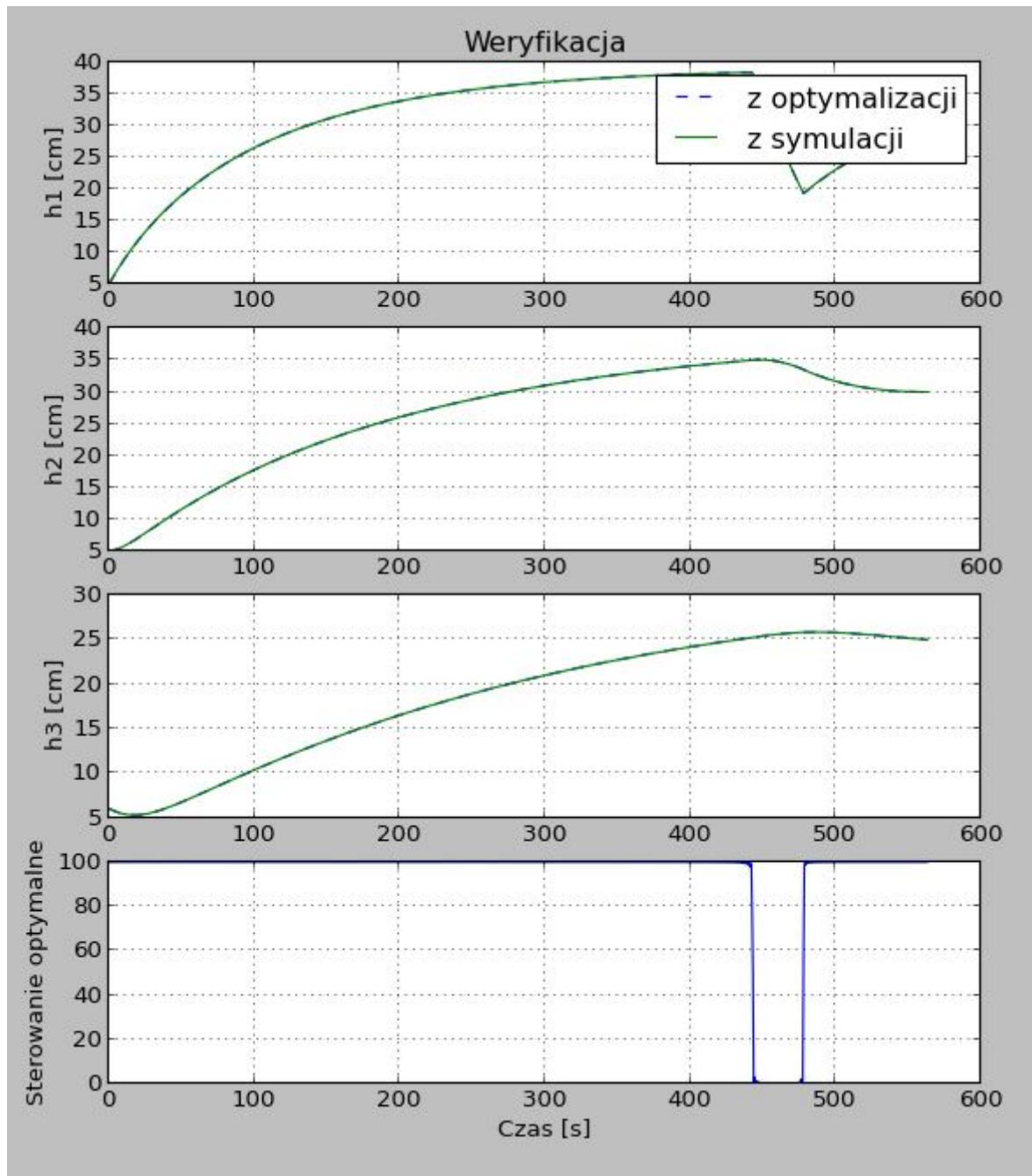
### 5.2.1. Weryfikacja przy użyciu pakietu JModelica.org

Po zakończeniu optymalizacji wyższy poziom aplikacji umożliwia przeprowadzenie symulacji weryfikacyjnej. Podaje się wtedy cały uzyskany wektor sterowania optymalnego (w postaci „surowej” lub znormalizowanej) jako trajektorię wejściową do symulacji, a na koniec wylicza się błąd weryfikacji.

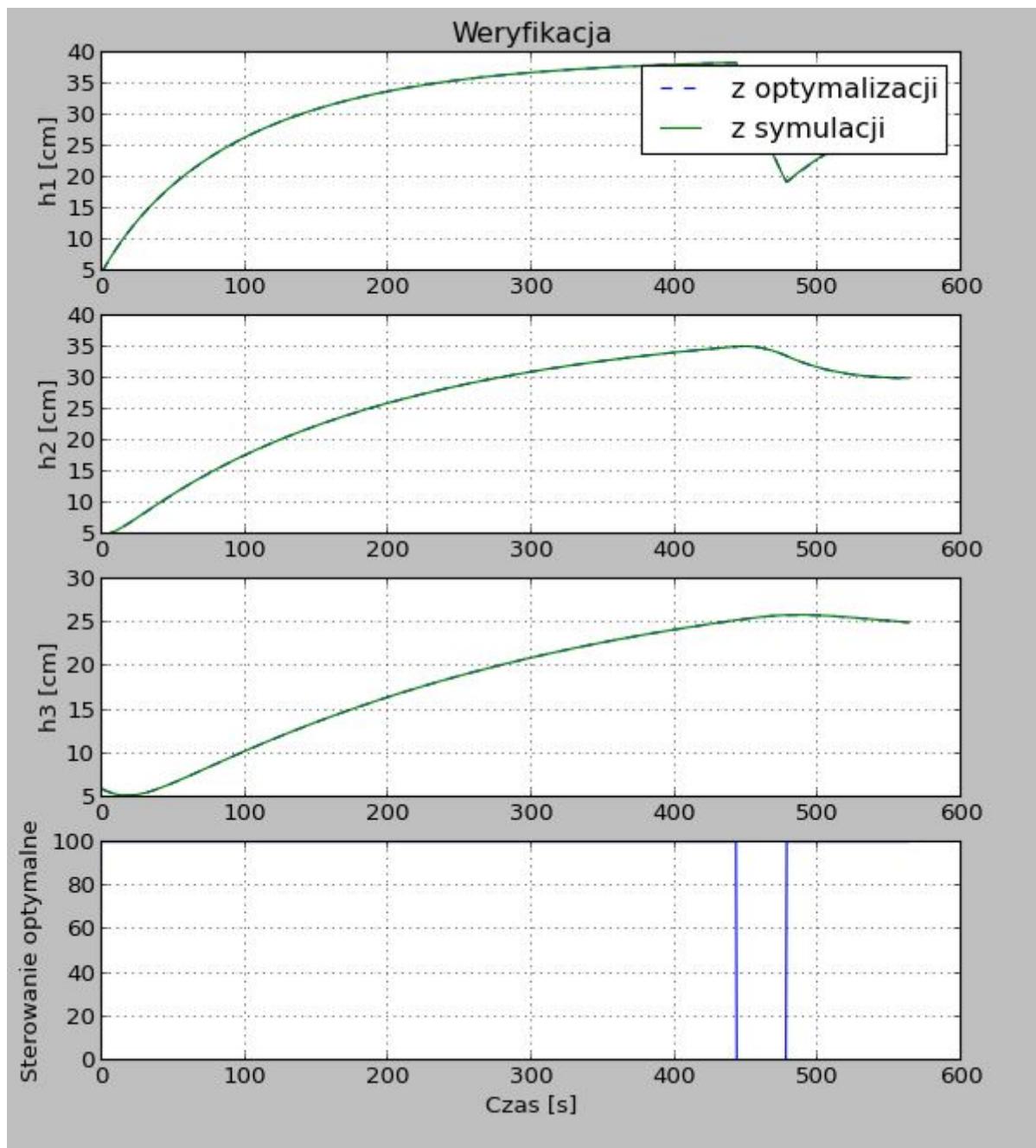
Poniżej przedstawiono 2 przykłady wykresów uzyskanych w czasie weryfikacji i porównano wartości błędów przy zastosowaniu sterowania w postaci „surowej” oraz znormalizowanej.

Pierwszy przykład to napełnianie zbiorników od poziomów  $h_1^0 = h_2^0 = 5\text{cm}$ ,  $h_3^0 = 6\text{cm}$  do poziomów  $h_1^f = h_2^f = 30\text{cm}$ ,  $h_3^f = 25\text{cm}$ . Sterowanie w postaci z algorytmu optymalizacji osiągnęło wartość błędu weryfikacji  $e_{wer}^s = 0,0002$ , a po znormalizowaniu:  $e_{wer}^n = 0,0021$ . W tym przypadku sterowanie w postaci „surowej” (przedstawione na rys. 5.5) wizualnie nie różni się bardzo od sterowania znormalizowanego (pokazanego na rys. 5.6), ale nawet taka niewielka różnica może prowadzić do zwiększenia się błędu weryfikacji o rząd wielkości. Jest to jednak również związane z tym, iż pierwszy błąd jest bardzo niewielki i w czasie wielu prób rzadko uzyskiwano tak niewielkie wartości. W tej optymalizacji użyto 350 elementów skończonych.

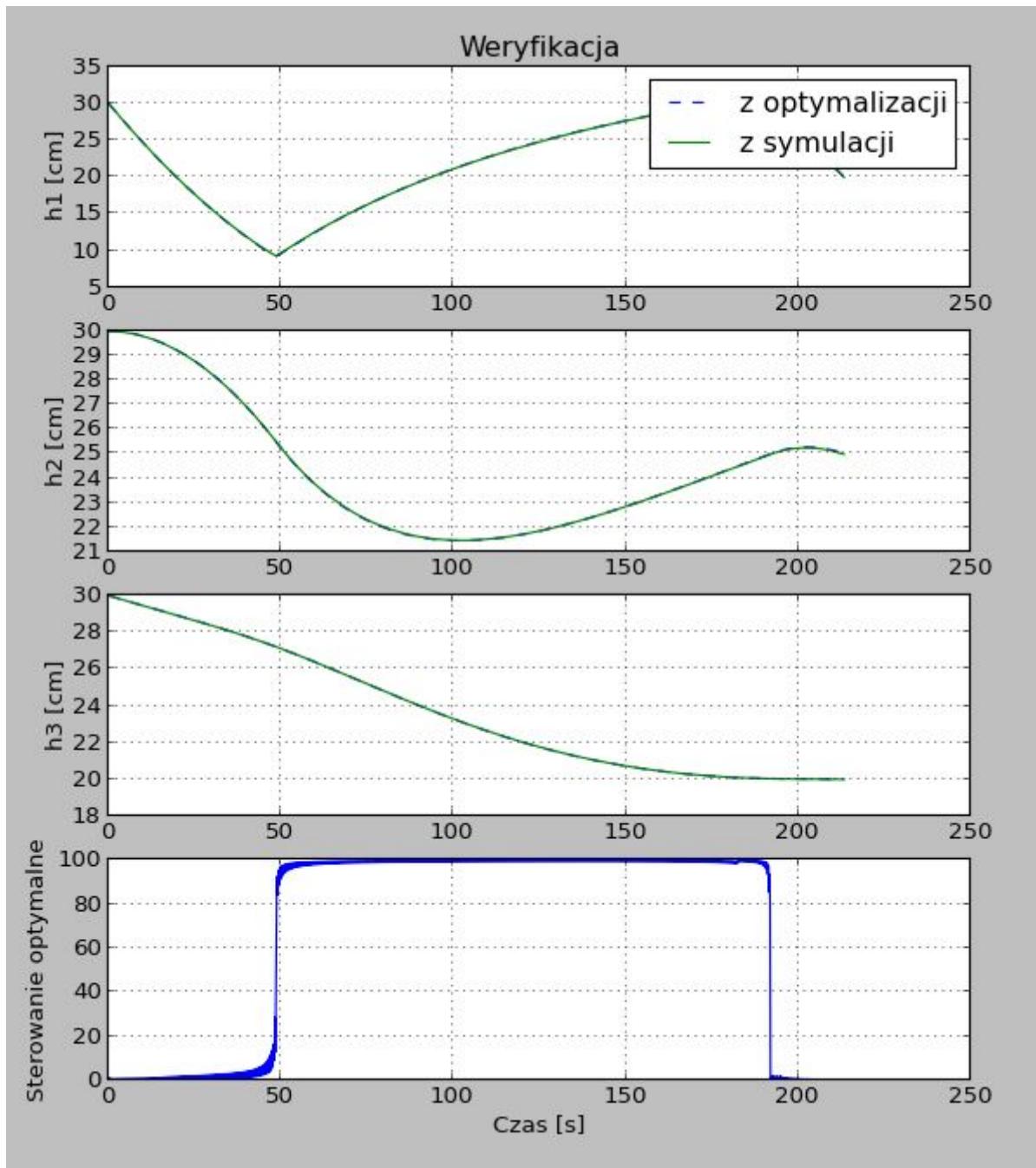
Drugi przykład to upuszczanie niewielkiej ilości wody ze zbiorników od poziomów  $h_1^0 = h_2^0 = h_3^0 = 30\text{cm}$  do  $h_1^f = 20\text{cm}$ ,  $h_2^f = 25\text{cm}$ ,  $h_3^f = 20\text{cm}$ . Sterowanie w postaci „surowej” osiągnęło wartość błędu weryfikacji  $e_{wer}^s = 0,007$ , a w postaci znormalizowanej:  $e_{wer}^n = 0,019$ . Tutaj różnica między błędami jest mniejsza niż w poprzednim przypadku, mimo bardziej skomplikowanej struktury sterowania wyznaczonego przez algorytm optymalizacyjny (pokazanego na rys. 5.7). Sterowanie znormalizowane zaprezentowano na rys. 5.8. W tej optymalizacji użyto 200 elementów skończonych, więc trzeba to wziąć pod uwagę, porównując wartości błędów między oboma opisanymi przypadkami.



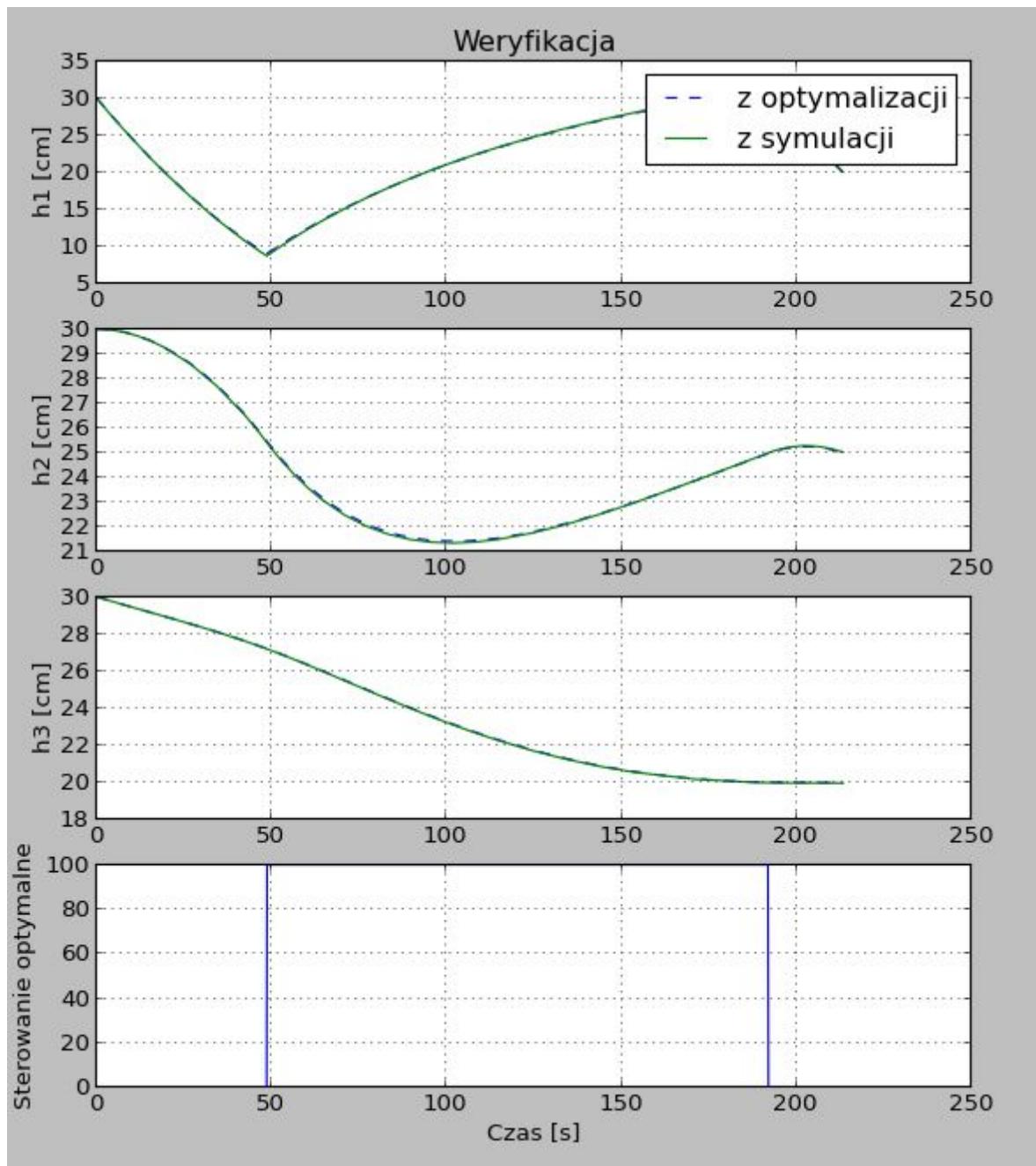
Rys. 5.5. Przykładowa weryfikacja rozwiązania przy użyciu pakietu JModelica.org.  
Źródło: własne.



Rys. 5.6. Przykładowa weryfikacja po normalizacji sterowania. Źródło: własne.



Rys. 5.7. Druga przykładowa weryfikacja rozwiązania przy użyciu pakietu JModelica.org. Źródło: własne.



Rys. 5.8. Druga przykładowa weryfikacja po normalizacji sterowania. Źródło: własne.

### 5.2.1.1. Wpływ liczby elementów metody elementów skończonych na rozwiązanie

W związku z tym, że zastosowana metoda przynosiła dobre rezultaty, lecz wartości błędu weryfikacji były dosyć duże (rzędu jednej dziesiątej), podjęto próbę zwiększenia liczby elementów skończonych i przeprowadzenia ponownej optymalizacji i weryfikacji. Odkryto, iż istotnie ma ono wpływ na wartość błędów weryfikacji i na czas obliczeń.

Przeprowadzono więc eksperiment, który miał pokazać ten wpływ liczby elementów skończonych na błędy weryfikacji dla sterowania „surowego” i znormalizowanego oraz czas obliczeń. Przyjęto zmianę tej liczby w przedziale 50 do 500 z krokiem 50 tak, aby uzyskać 10 punktów pomiarowych.

W pierwszym przypadku skorzystano z napełniania zbiorników od poziomu  $h_1^0 = h_2^0 = h_3^0 = 1\text{cm}$  do  $h_1^f = h_2^f = h_3^f = 15\text{cm}$ . Uzyskane wyniki przedstawiono na rys. 5.9. Zauważono, iż:

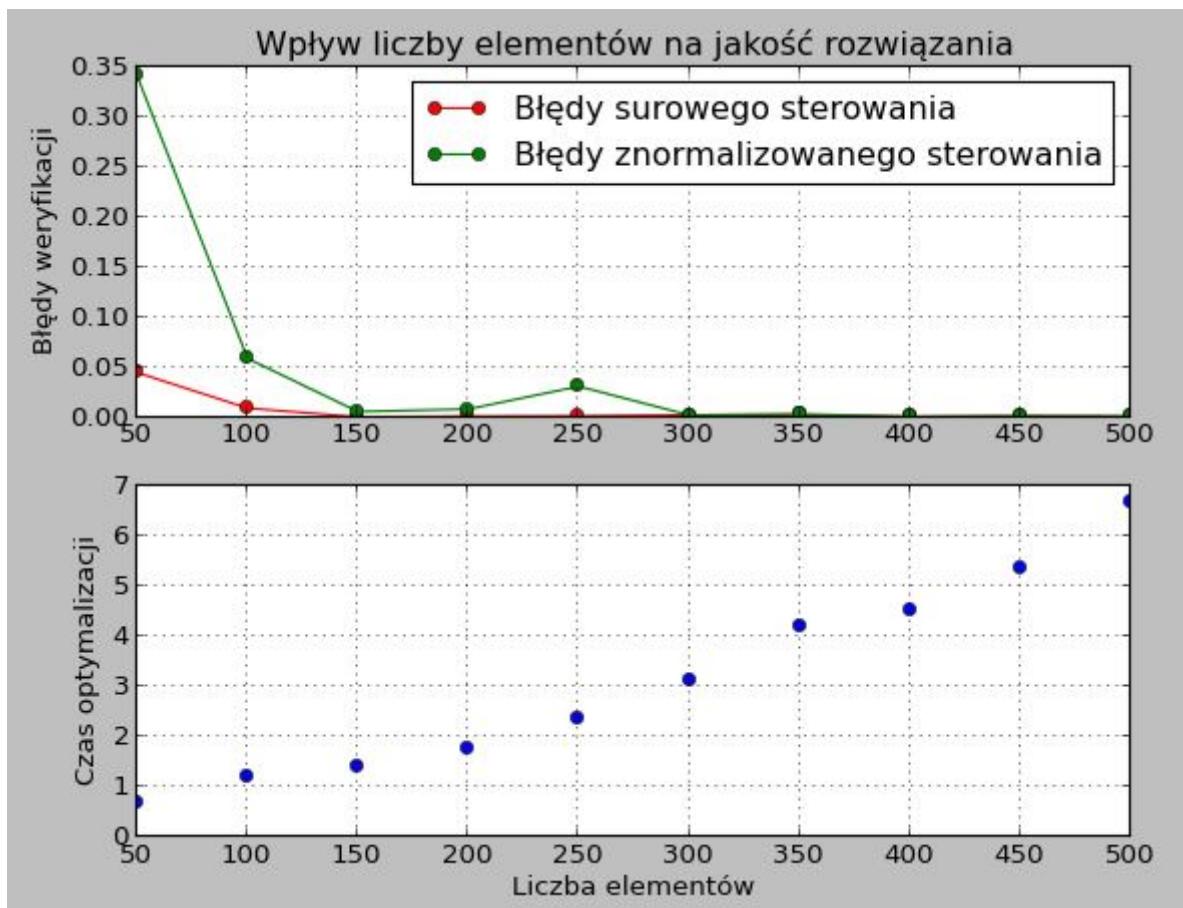
- różnica absolutna i relatywna między błędem weryfikacji dla sterowania „surowego” i znormalizowanego jest największa w przypadku najmniejszej rozważanej liczby elementów,
- wartości czasów obliczeń rosną najprawdopodobniej wielomianowo,
- charakter zmniejszania się błędów jest podobny do charakteru wzrostu czasu obliczeń,
- należy stosować wartości liczby elementów z przedziału między 200 a 350, gdyż tam występuje najlepszy stosunek czasu obliczeń do wartości błędów.

Przeprowadzono drugi eksperiment dla innych danych w celu weryfikacji powyższych wniosków. Tym razem skorzystano z napełniania zbiorników od poziomu  $h_1^0 = h_2^0 = h_3^0 = 10\text{cm}$  do  $h_1^f = h_2^f = h_3^f = 20\text{cm}$ . Wyniki pokazano na rys. 5.10. Tym razem zaobserwowano, iż:

- wartości błędów są zależne od warunków zadania - w tym przypadku były one około 2 razy większe przy wszystkich wartościach liczby elementów skończonych,
- czasy obliczeń nie zachowują się tak regularnie, jak w powyższym przypadku i osiągają wartości o rząd wielkości większe, a więc silnie zależą od warunków zadania,
- potwierdziła się hipoteza, iż należy wybierać wartości liczby elementów między 200 a 350.

### 5.2.1.2. Problemy z błędą wartością pochodną

W przypadku niektórych wartości początkowych i końcowych oraz znalezionej dla nich sterowania optymalnego napotykano problem związany z tym, iż pakiet JModelica.org nie był w stanie obliczyć pochodnej jednego ze stanów w punkcie czasu niedługo po pierwszym przełączeniu. Rozwiązano go poprzez zwiększenie liczby elementów skończonych przy rozwiązywaniu danego zadania. Nie zaobserwowano takich problemów przy liczbie elementów powyżej 50.



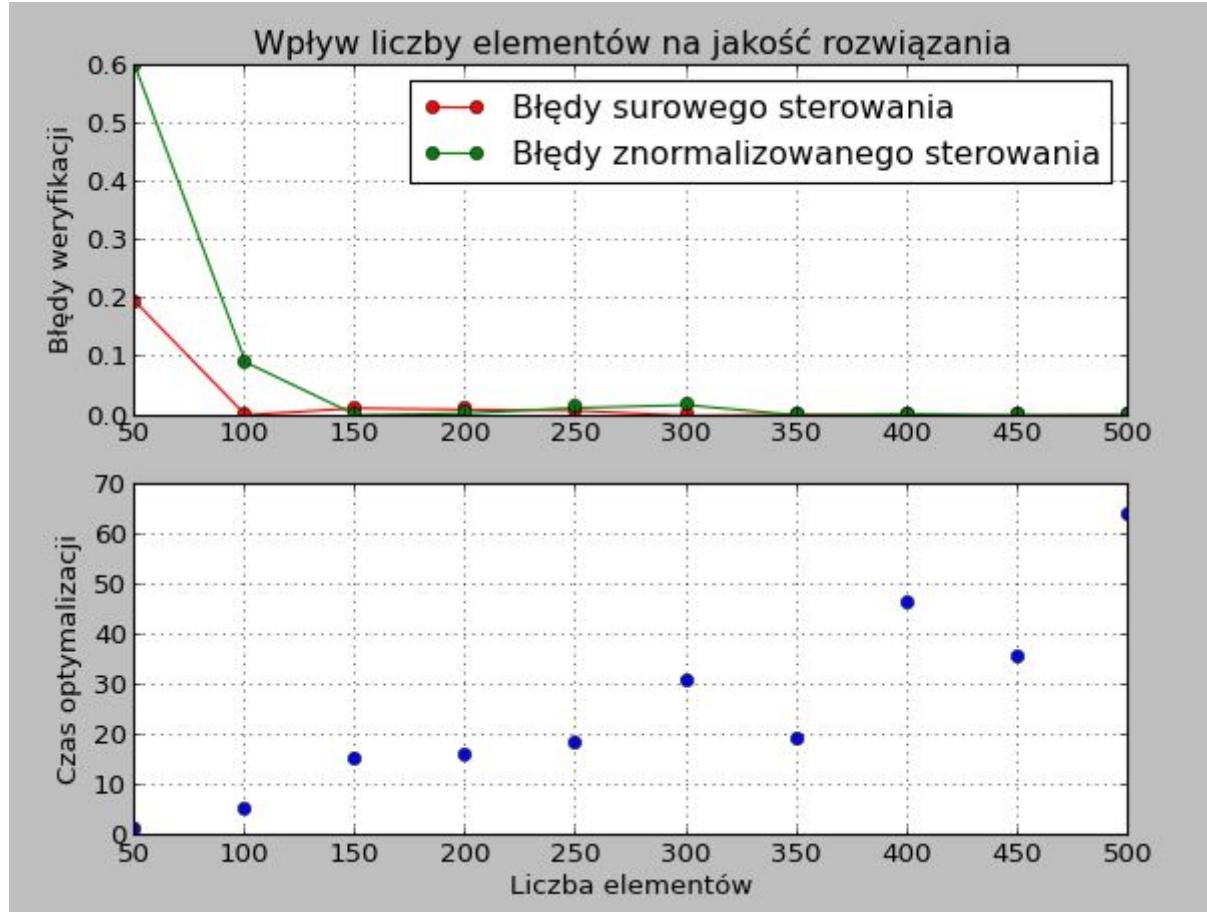
Rys. 5.9. Wpływ liczby elementów skończonych na rozwiązanie. Źródło: własne.

### 5.2.2. Weryfikacja przy użyciu oprogramowania MATLAB/Simulink

Połączenie obu poziomów aplikacji w całość umożliwiło przeprowadzenie symulacji weryfikacyjnej w programie MATLAB/Simulink. Zastosowano tu pewne uproszczenie: symulację uruchamiano dopiero po obliczeniu pierwszego sterowania optymalnego. Pokazano jednak, że w przy kolejnych optymalizacjach dokonywano ich w czasie trwania symulacji i niższy poziom aplikacji był w stanie przełączać się w czasie rzeczywistym między sterowaniem czasooptymalnym a liniowo-kwadratowym.

Poniżej pokazano dwa przykłady funkcjonowania symulacji weryfikacyjnej niższego poziomu aplikacji. W obu użyto metody Runge-Kutty 4 rzędu i stałego kroku o wartości 0.001 s. Dobrano taką metodę i wartość kroku czasowego ze względu na przewidywanie możliwości wykorzystania całej opisywanej aplikacji również z rzeczywistym układem zbiorników, gdzie komunikacja ze sprzętem wymusza stosowanie stałego kroku symulacji. Symulacje przeprowadzono z nieskończonym czasem końcowym, ale prezentowane są tylko interesujące przedziały czasu.

Pierwszy przykład jest pokazany na rys. 5.11. Widoczne są tam 3 procesy aplikacji sterowania czasooptymalnego przedzielone okresami, w których działa regulator liniowo-kwadratowy. Drugi przykład jest



Rys. 5.10. Drugi przykład wpływu liczby elementów skończonych na rozwiązanie.

Źródło: własne.

przedstawiony na rys. 5.12 i zawiera 2 procesy aplikacji sterowania czasooptymalnego. Można je zauważyć na wykresie przedstawiającym sterowanie, gdyż ma ono wtedy postać „bang-bang”. W czasie działania regulatora liniowo-kwadratowego sterowanie ma wartość zbliżoną do sterowania ustalonego, które nie leży na ograniczeniach. Nie obserwuje się zbyt dużych zmian sterowania liniowo-kwadratowego, gdyż stan układu jest już w niedalekim otoczeniu punktu równowagi.

Jak wspomiano w podrozdziale 3.1, punkt równowagi wyznacza się na podstawie stanu docelowego zagadnienia optymalizacji dynamicznej. Używa się w tym celu metody identycznej z algorytmem inicjalizacyjnym opisany w sekcji 5.1.2: wyznacza się 3 sterowania ustalone odpowiadające wartościom stanów docelowych w zbiornikach, a następnie wylicza się z nich średnią i traktuje się ją jako sterowanie ustalone dla regulatora liniowo-kwadratowego.

Wyniki weryfikacji porównano z wynikami analogicznych operacji przeprowadzonych przy użyciu wyższego poziomu aplikacji. Podsumowanie wyników zostało przedstawione w tabeli 5.1 dla pierwszego przykładu i 5.2 dla drugiego. W jej kolumnach przedstawiono:

- czas zakończenia optymalizacji liczony od początku symulacji,
- stan początkowy układu użyty do wyznaczenia sterowania czasooptymalnego,

- stan docelowy,
- błąd weryfikacji obliczony na niższym poziomie aplikacji w programie MATLAB/Simulink,
- błąd weryfikacji obliczony na wyższym poziomie aplikacji przy użyciu platformy JModelica.org; obliczono go, używając 350 elementów skończonych w algorytmie optymalizacji i normalizując sterowanie.

Czas	Stan początkowy	Stan docelowy	$e_{MATLAB}$	$e_{JModelica.org}$
81,623	$h^0 = [20 \ 20 \ 20]^T$	$h^f = [15 \ 15 \ 15]^T$	0,021	0,008
219,268	$h^0 = [15.77 \ 15.37 \ 13.02]^T$	$h^f = [10 \ 10 \ 10]^T$	0,047	0,0095
466,422	$h^0 = [10.62 \ 10.39 \ 8.37]^T$	$h^f = [15 \ 13 \ 12]^T$	0,221	0,0007

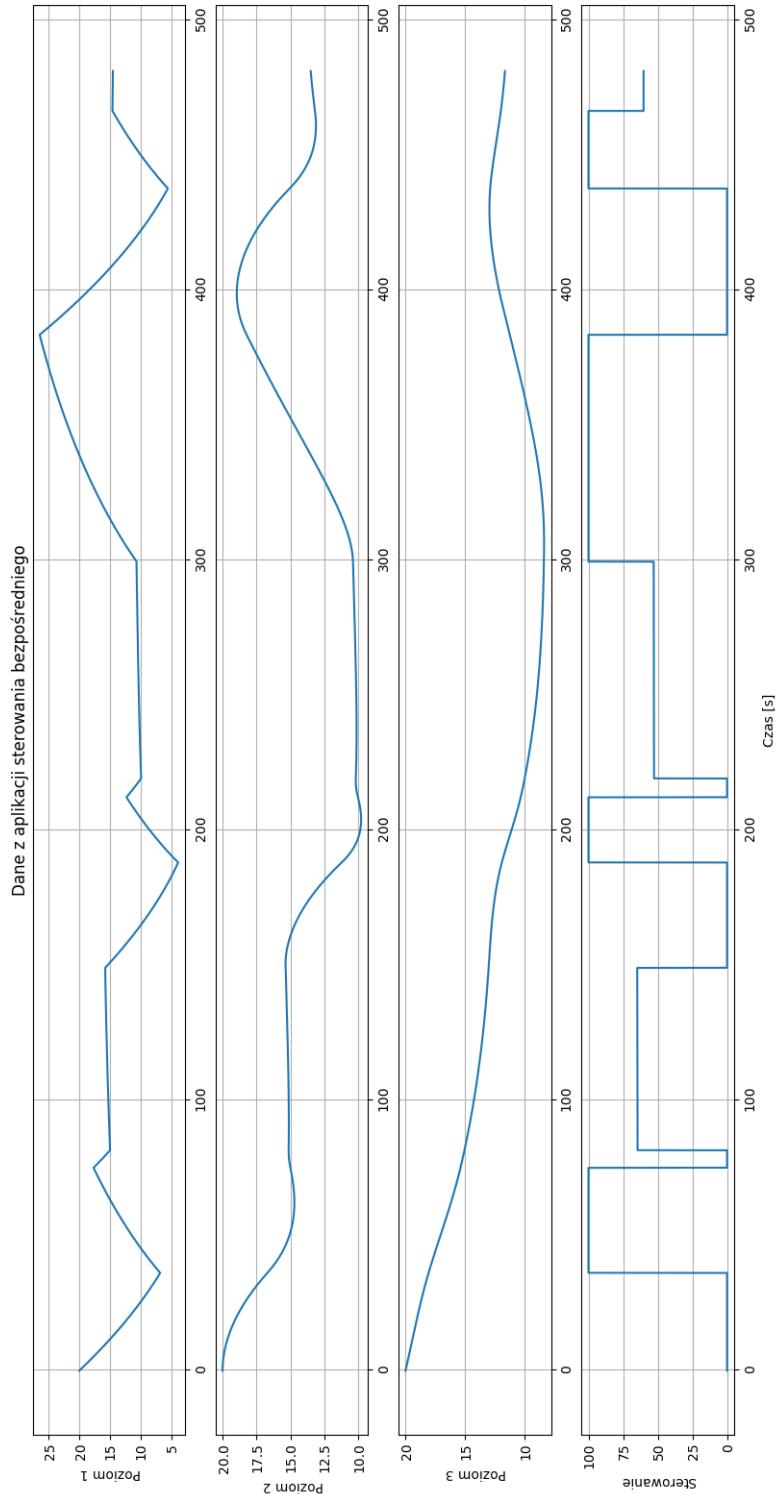
**Tabela 5.1.** Podsumowanie wyników weryfikacji wyższego i niższego poziomu aplikacji dla trzech optymalizacji przedstawionych na rys. 5.11. Źródło: własne.

Czas	Stan początkowy	Stan docelowy	$e_{MATLAB}$	$e_{JModelica.org}$
114,141	$h^0 = [20 \ 20 \ 20]$	$h^f = [10 \ 10 \ 10]^T$	0,081	0,002
168,743	$h^0 = [9.82 \ 9.99 \ 9.37]$	$h^f = [10 \ 12 \ 9]^T$	0,203	0,0005

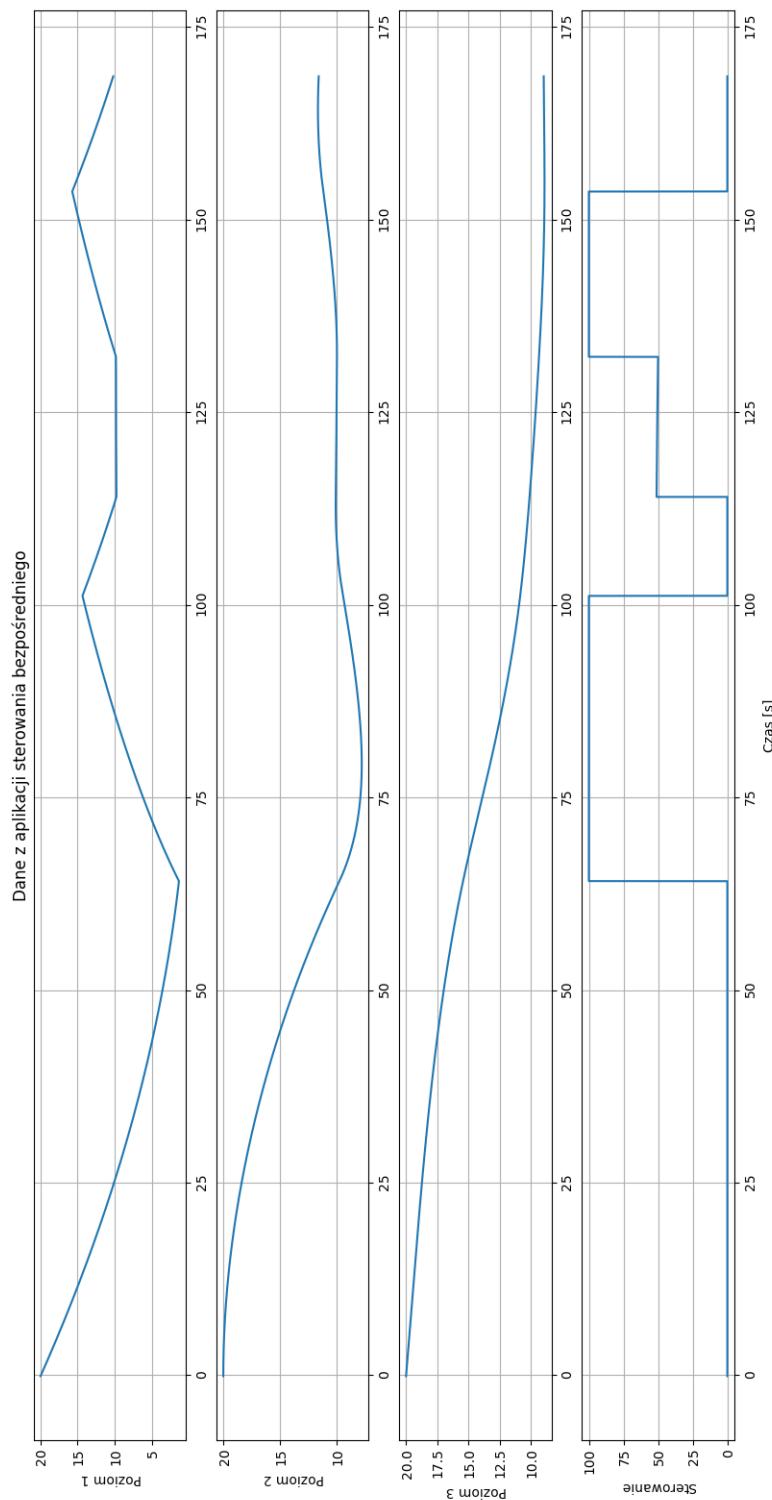
**Tabela 5.2.** Podsumowanie wyników weryfikacji wyższego i niższego poziomu aplikacji dla dwóch optymalizacji przedstawionych na rys. 5.12. Źródło: własne.

Zauważono, iż błędy weryfikacji na niższym poziomie rosną przy kolejnych optymalizacjach. Dzieje się tak ze względu na fakt, iż aplikacja wyższego poziomu przeprowadza optymalizację na podstawie danych pobranych przed uruchomieniem algorytmu, ale sterowanie jest wysyłane na niższy poziom po jego zakończeniu. Przez czas działania algorytmu poziomy zmieniają się jednak i dążą dalej do odpowiedniego punktu równowagi. Ten problem mógłby rozwiązać układ predykcyjny działający po stronie wyższego poziomu aplikacji, który wyznaczałby spodziewane wartości poziomów początkowych w momencie rozpoczęcia aplikacji sterowania czasooptymalnego. Jednakże jak pokazano w sekcji 5.2.1, czas działania algorytmu optymalizacji jest różny dla różnych zestawów stanów początkowych i końcowych, a jego oszacowanie mogłoby być trudnym zadaniem.

Błędy pierwszych optymalizacji nie powinny być obarczone tym problemem, ale i tak są o rząd wielkości większe od tych uzyskanych na wyższym poziomie aplikacji. Jest to najprawdopodobniej spowodowane niedokładnością wyznaczenia czasów przełączeń: w aplikacji wyższego poziomu do weryfikacji podaje się cały wektor sterowania, a w aplikacji niższego poziomu używa się tylko czasów przełączeń przesyłanych przez sieć, które mogą być obarczone numerycznym błędem obcięcia.



Rys. 5.11. Trzy procesy optymalizacji zweryfikowane w symulacji niższego poziomu aplikacji. Źródło: własne.



Rys. 5.12. Dwa procesy optymalizacji zweryfikowane w symulacji niższym poziomem aplikacji. Źródło: własne.

## Zakończenie

Przygotowanie aplikacji realizującej postawione we wstępie niniejszej pracy cele okazało się zadaniem tyleż trudnym, co fascynującym. Obie te cechy wynikały przede wszystkim z założeń postawionych przed wyższym poziomem aplikacji opisanych na początku podrozdziału 4.1. Znalezienie i opisanie otwartego oprogramowania, które potrafi rozwiązywać problemy optymalizacji dynamicznej układów nieliniowych i posiada przystępny interfejs użytkownika, jest niewątpliwą wartością tej pracy.

Udało się zrealizować wszystkie cele: pokazano, iż wyższy poziom aplikacji jest w stanie wyznaczać sterowanie optymalne w rozumieniu dwóch wskaźników jakości: liniowo-kwadratowego i czasowego. Umożliwia on również przeprowadzenie symulacji weryfikacyjnych wyznaczonych sterowań, które opisano w rozdziale 5. Niższy poziom aplikacji został przygotowany z myślą o komunikacji z układem pomiarowo-sterującym, ale ona nie została przetestowana. Używano tego poziomu tylko w celach symulacyjnych, co dało możliwość sprawdzenia funkcjonowania wyznaczonego sterowania optymalnego poza środowiskiem, w którym zostało obliczone.

Podstawowym kierunkiem dalszych prac byłoby połączenie napisanej aplikacji z rzeczywistym obiektem, co na pewno dostarczyłyby istotnych wniosków co do funkcjonowania całości aplikacji. Byłoby dobrze uzupełnić również opis problemu o identyfikację elementów wykonawczych oraz wyznaczenie zbioru stanów osiągalnych, aby ograniczyć użytkownikowi zbiór możliwych punktów końcowych optymalizacji. Można by również przygotować układ predykcyjny, w celu poprawy błędów sterowania obliczanego w czasie działania aplikacji. Zapewne byłoby to potrzebne przy użyciu jej z rzeczywistym układem zbiorników. Nie testowano również wszystkich możliwości dostosowania algorytmu optymalizacji do potrzeb tego zagadnienia.

Myśląc o faktycznym uruchomieniu takiej aplikacji w środowisku przemysłowym, należałoby udoskonalić inicjalizację optymalizacji, gdyż to zagadnienie nie zostało rozpoznane dogłębnie w niniejszej pracy. Jest to jednak problem na tyle złożony, że z powodzeniem mógłby stać się tematem kolejnej pracy dyplomowej.

# Spis rysunków

1.1	Układ zbiorników z zaznaczonymi wymiarami. Źródło: własne.	10
2.1	Trzy główne komponenty wyznaczania sterowania optymalnego i klasy metod, które z nich korzystają. Źródło: [10]. Tłumaczenie: własne.	21
3.1	Diagram sekwencji ilustrujący akcję obliczenia sterowania czasooptymalnego. Źródło: własne.	28
3.2	Diagram sekwencji ilustrujący akcję obliczenia sterowania liniowo-kwadratowego. Źródło: własne.	29
3.3	Diagram sekwencji ilustrujący akcję wysyłania sterowań. Źródło: własne.	30
4.1	Uproszczony model obiektów w systemie Tango Controls. Źródło: [18].	34
4.2	Schemat architektury procesów i wątków wyższego poziomu aplikacji. Źródło: własne.	39
4.3	Schemat architektury platformy JModelica.org. Źródło: [22].	45
4.4	Model niższego poziomu aplikacji w programie Simulink. Źródło: własne.	49
5.1	Przykładowa trajektoria początkowa dla algorytmu optymalizacji. Źródło: własne.	53
5.2	Przykładowy wynik optymalizacji. Źródło: własne.	54
5.3	Przykład sterowania „surowego” otrzymanego w wyniku optymalizacji. Źródło: własne.	55
5.4	Przykład sterowania znormalizowanego. Źródło: własne.	55
5.5	Przykładowa weryfikacja rozwiązania przy użyciu pakietu JModelica.org. Źródło: własne.	57
5.6	Przykładowa weryfikacja po normalizacji sterowania. Źródło: własne.	58
5.7	Druga przykładowa weryfikacja rozwiązania przy użyciu pakietu JModelica.org. Źródło: własne.	59
5.8	Druga przykładowa weryfikacja po normalizacji sterowania. Źródło: własne.	60
5.9	Wpływ liczby elementów skończonych na rozwiązanie. Źródło: własne.	62
5.10	Drugi przykład wpływu liczby elementów skończonych na rozwiązanie. Źródło: własne.	63
5.11	Trzy procesy optymalizacji zweryfikowane w symulacji niższego poziomu aplikacji. Źródło: własne.	65

5.12 Dwa procesy optymalizacji zweryfikowane w symulacji niższym poziomem aplikacji. Źródło: własne.	66
---	----



## Bibliografia

- [1] M. Athans i P. L. Falb. *Optimal Control: An Introduction to the Theory and its Applications*. Nowy Jork, Stany Zjednoczone Ameryki: McGraw-Hill Book Company, 1966.
- [2] I. Vakilzadeh i A. Keshavarz. „Bang-Bang Control of a Second-Order Non-Linear Stable Plant with Second-Order Nonlinearity”. W: *Kybernetika* 18.1 (1982). Dostęp on-line: <http://www.kybernetika.cz/content/1982/1/66/paper.pdf> Stan na: 15.08.2017 r.
- [3] K. Balachandran i D. Somasundaram. „Bang-Bang Control of a Second-Order Non-Linear Stable Plant with Fourth-Order Nonlinearity”. W: *Kybernetika* 19.2 (1983). Dostęp on-line: <http://www.kybernetika.cz/content/1983/2/158/paper.pdf> Stan na: 15.08.2017 r.
- [4] J. T. Betts. „Survey of numerical methods for trajectory optimization.” W: *Journal of Guidance, Control and Dynamics* 21.2 (1998), s. 193–207.
- [5] R. M. Murray. *Lecture notes for course CDS 110b: Introduction to Control Theory*. Dostęp on-line: <https://www.cds.caltech.edu/~murray/courses/cds110/wi06/lqr.pdf>. Stan na: 15.08.2017 r. Sty. 2006.
- [6] W. Mitkowski i in. *Teoria Sterowania. Materiały pomocnicze do ćwiczeń laboratoryjnych*. Kraków, Polska: Akademia Górnictwa-Hutnicza im. S. Staszica, 2007.
- [7] Y. Ma. *Lecture notes for course ECE553: Optimum Control Systems*. Dostęp on-line: <http://yima.csl.illinois.edu/ECE553.html> Stan na: 15.08.2017 r. Champaign, Illinois, Stany Zjednoczone Ameryki: Uniwersytet Illinois w Urbana-Champaign, 2008.
- [8] M. Hast, J. Åkesson i A. Robertsson. „Optimal Robot Control Using Modelica and Optimica”. W: *7 Międzynarodowa Konferencja Modelica*. Como, Włochy, paź. 2009.
- [9] J. Åkesson i in. „Modeling and Optimization with Optimica and JModelica.org—Languages and Tools for Solving Large-Scale Dynamic Optimization Problems”. W: *Computers and Chemical Engineering* 34.11 (list. 2010), s. 1737–1749.
- [10] A. Rao. „A Survey of Numerical Methods for Optimal Control”. W: *Advances in the Astronautical Sciences* 135 (2010).
- [11] J. Andersson i in. „Integration of CasADI and JModelica.org”. W: *8 Międzynarodowa Konferencja Modelica*. Drezno, Niemcy, mar. 2011.

- [12] F. Casella, F. Donida i J. Åkesson. „Object-Oriented Modeling and Optimal Control: A Case Study in Power Plant Start-Up”. W: *18 Światowy Kongres IFAC*. Mediolan, Włochy, sierp. 2011.
- [13] B. Houska i in. „ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization”. W: *Optimal Control Applications and Methods* 32 (2011), 289–312.
- [14] T. Trawiński. „Sterowanie czasooptymalne systemem pozycjonowania głowic dysków twardych”. W: *Modelowanie Inżynierskie* 42 (2011), s. 433–440.
- [15] *JModelica.org User Guide: Version 1.17*. Dostęp on-line: <http://www.jmodelica.org/api-docs/usersguide/JModelicaUsersGuide-1.17.0.pdf> Stan na: 15.06.2017 r. Modelon AB. Lund, Szwecja, 2015.
- [16] A. Korytowski. *Notatki z wykładów na przedmiot: Optymalizacja w Systemach Sterowania*. Kraków, Polska: Akademia Górnictwo-Hutnicza im. S. Staszica, 2015.
- [17] M. Itik. „Optimal control of nonlinear systems with input constraints using linear time varying approximations”. W: *Nonlinear Analysis: Modelling and Control* 21.3 (2016). Dostęp on-line: [https://www.mii.lt/na/issues/NA\\_2103/NA2137.pdf](https://www.mii.lt/na/issues/NA_2103/NA2137.pdf) Stan na: 15.08.2017 r.
- [18] *Dokumentacja Tango Controls*. Dostęp on-line <http://tango-controls.readthedocs.io/en/latest> Stan na: 5.09.2017 r. Grenoble, Francja.
- [19] C. Evans. *An Introduction to Mathematical Optimal Control Theory. Lecture notes version 0.2*. Dostęp on-line: <https://math.berkeley.edu/~evans/control.course.pdf> Stan na: 15.08.2017 r. Berkeley, Kalifornia, Stany Zjednoczone Ameryki: Uniwersytet Kalifornijski.
- [20] *Multitank System - User's Manual*. Inteco sp. z o.o. Kraków, Polska.
- [21] B. Postlethwaite. *Introduction to Process Control - Online Course*. Dostęp on-line: <http://www.online-courses.vissim.us/Strathclyde/> Stan na: 15.08.2017 r. Uniwersytet Strathclyde.
- [22] *Strona internetowa pakietu JModelica.org*. Dostęp on-line: <http://www.jmodelica.org> Stan na: 15.08.2017 r. Lund, Szwecja: Modelon AB.
- [23] *Strona internetowa pakietu Modelica*. Dostęp on-line: <https://www.modelica.org> Stan na: 15.08.2017 r. Stowarzyszenie Modelica.
- [24] *Strona internetowa Tango Controls*. Dostęp on-line <http://www.tango-controls.org> Stan na: 15.08.2017 r. Grenoble, Francja: Konsorcjum Tango Controls.