ALGORYTMY STEROWANIA QUADROTORA

Politechnika Wrocławska 2013

Autorzy

Dorota Moskal – rozdziały 4.2, 6.4, 7.2
Michał Kret – rozdział 4.2
Mateusz Sawicki, Michał Dziergwa – rozdziały 6.6, 6.7, 7.5
Mariusz Orda, Krzysztof Szydłowski – rozdziały 1, 2, 4.1, 5, 6.3, 7.1.
Natalia Czop, Maciej Gawron – rozdziały 3.1, 4.3, 6.5, 7.3
Michał Adamczyk, Paweł Kułakowski – rozdziały 3.2, 6.1, 6.2, 7.4

Skład raportu wykonano w systemie LaTeX

Praca udostępniana na licencji Creative Commons: *Uznanie autorstwa-Użycie niekomercyjne-Na tych samych warunkach 3.0*, Wrocław 2012. Pewne prawa zastrzeżone na rzecz Autorów. Zezwala się na niekomercyjne wykorzystanie treści pod warunkiem wskazania Autorów jako właścicieli praw do tekstu oraz zachowania niniejszej informacji licencyjnej tak długo, jak tylko na utwory zależne będzie udzielana taka sama licencja. Tekst licencji dostępny na stronie: http://creativecommons.org/licenses/by-nc-sa/3.0/pl/

Spis treści

II. Implementacja

Spis oznaczeń

Oznaczenia zostały uszeregowane alfabetycznie. W pierwszej kolejności przedstawiono symbole rozpoczynające się literami alfabetu łacińskiego. Następnie wprowadzono oznaczenia wykorzystujące alfabet grecki.

— A_X, A_Y, A_Z – przekrój robota odpowiednio w osi OX, OY, OZ
— b – współczynnik ciągu
— d – współczynnik oporu środowiska powietrza
— $F(t)$ – siła elektrodynamiczna
— F_X, F_Y, F_Z – siła oporu powietrza działająca odpowiednio w osi OX, OY, OZ
— f – współczynnik tarcia lepkiego
— G – obiekt sterowania
— g – przyspieszenie ziemskie
— I_X, I_Y i I_Z – momenty główne (bezwładności) ramy quadrotora
— $i(t)$ – natężenie prądu płynącego przez silnik
— J_r – moment bezwładności pojedynczego śmigła
— K – stała konstrukcyjna silnika
— K_1 – sterownik H_∞ do śledzenia sygnału W_i (też Ki)
— K_2 – sterownik H_∞ do pomiaru sygnału zwrotnego (też $K2$)
— L – indukcyjność uzwojenia twornika silnika
— l – odległość od środka ciężkości quadrotora do każdego z rotorów
— $M(t)$ – moment elektryczny
— M_f – moment tarcia suchego
— $M_r(t)$ – moment obciążenia
— m – masa całkowita quadrotora
— $Q(s)$ – funkcja przenoszenia
— q – współrzędne uogólnione
— R – rezystancja uzwojenia twornika silnika
— R_r – promień śmigieł (też r)
— T – energia kinetyczna
— T – ciąg z uwzględnionym efektem ziemi (podłoża)
— T_∞ – ciąg uzyskiwany przez silniki na nieskończenie dużej wysokości
— U – napięcie na zaciskach silnika
— U_1, U_2, U_3, U_4 – sterowania modelu quadrotora
— u – wektor wejścia sterowania
— v – wartość zmierzona na wyjściu układu sterowania
— V – energia potencjalna
— W_i – model dokładny obiektu wyznaczony przy stanie stacjonarnym (też Wi)
— w – sygnały egzogenne (zakłócenia zewnętrzne)
— $\dot{x}$ – prędkość w płaszczyźnie dla której liczymy opory powietrza
— x, y, z – współrzędne układu związanego z ziemią.
— X, Y, Z – współrzędne układu związanego z quadrotorem.
— z – sygnał błędu
— z_h – pionowa odległość quadrotora od powierzchni płaskiej (podłoża/sufitu)
— ρ_{air} – gęstość powietrza
— ϕ, θ, ψ – kąty KKM (kiwanie, kołysanie, myszkowanie), inaczej kąty RPY (roll, pitch, yaw) wyrażone w radianach
— Ω – suma prędkości obrotowych śmigieł
— Ω_1, Ω_2, Ω_3, Ω_4 – wartości prędkości obrotowej śmigieł

— $\omega(t)$ – prędkość obrotowa silnika
— $\omega_X, \omega_Y, \omega_Z$ – prędkości obrotowe quadrotora względem osi X, Y, Z.

1. Wstęp

1.1. Badany obiekt

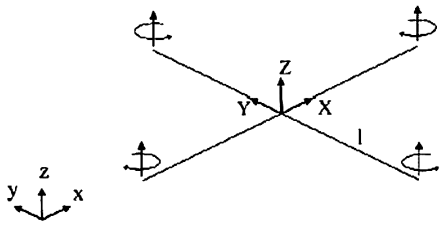
W pracy rozpatrzono zagadnienia dotyczące latającego robota typu quadrotor (quadrocopter). Napędzany jest on za pomocą czterech śmigieł umieszczonych w jednej płaszczyźnie, których osie obrotu są do niej prostopadłe. Taka konstrukcja umożliwia kontrolowany ruch w trójwymiarowej przestrzeni. Wygląd przykładowej realizacji fizycznej obiektu przedstawia rysunek 1.1.

Rysunek 1.1. Przykład robota typu quadrotor [5]

1.1.1. Własności obiektu

Obiekt badany w pracy posiada następujące własności:
— Ciało robota jest bryłą sztywną.
— Robot posiada cztery ramiona o równej długości, ułożone pod kątem 90 stopni względem siebie. Na ich końcach zamontowane są śmigła.
— Środek ciężkości robota znajduje się w jego środku geometrycznym.
— Śmigła są bryłami sztywnymi.
— Ciąg śmigieł jest proporcjonalny do kwadratu ich prędkości obrotowych.
— Opór aerodynamiczny jest proporcjonalny do kwadratu prędkości ciała.
— Ponieważ praca śmigieł dostarcza dodatkowej energii do układu quadrotora, układ ten jest niezachowawczy.

Schemat poglądowy badanego obiektu został przedstawiony na rysunku 1.2.

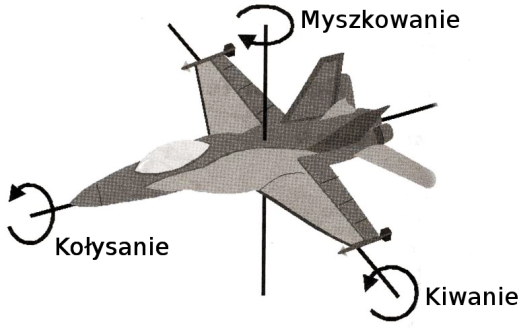


Rysunek 1.2. Schemat poglądowy quadrotora.

1.2. Układ współrzędnych

Środowisko działania modelu zostało opisane za pomocą dwóch układów kartezjańskich, związanych odpowiednio z otoczeniem oraz z ciałem robota. Początе Do opisu współrzędnych pierwszego układu wykorzystano symbole *x,y,z*, natomiast dla drugiego użyto *X,Y,Z*. Początek układu współrzędnych związanego z ciałem robota umieszczono w środku geometrycznym quadrotora, zaś osie X, Y w płaszczyźnie śmigieł. W celu określenia orientacji robota w przestrzeni wykorzystano kąty obrotu poprzecznego – *kołysania* (ang. *roll*), wzdłużnego – *kiwania* (*pitch*) oraz pionowego – *myszkowania* (*yaw*), w skrócie *KKM*. Reprezentację tą przedstawia rysunek 1.3. Jako oznaczenie wprowadzonych kątów przyjęto odpowiednio symbole ϕ, θ, ψ. W dalszej części dla przejrzystości zapisu wykorzystano skróconą notację zapisu funkcji trygonometrycznych: $\cos\phi \rightarrow c_\phi$, $\sin\theta \rightarrow s_\theta$ itp.

Przyjęty wektor zmiennych stanu to $q = (x, y, z, \phi, \theta, \psi)$.



Rysunek 1.3. Ilustracją kątów *KKM*

1.3. Cel pracy

Głównym celem pracy jest opracowanie środowiska umożliwiającego porównywanie wybranych algorytmów sterowania dla quadrotorów i realizacji zadania śledzenia trajektorii, uwzględniając wybrane zjawiska fizyczne działające na robota.

W rozdziale 2 przedstawiono sposób uzyskania przykładowego modelu dynamiki quadrotora przy użyciu formalizmu Eulera-Lagrange'a. Następnie zaprezentowano model

z nowym sterowaniem, wykorzystany w dalszej części pracy, oraz zwrócono uwagę na różnice w porównaniu do innych publikacji naukowych.

Kolejny rozdział omawia kwestie związane z modelowaniem matematycznym ciągu (sekcja 3.1) oraz numerycznym rozwiązywaniem równań różniczkowych 3.2. Modelowanie matematyczne ciągu dotyczy modelowania silników prądu stałego używanych w rzeczywistym modelu obiektu oraz modelowania (lub ich uzasadnionego pomijania) zaburzeń oddziałujących na quadrotor.

Rozdział 4 wprowadzono podstawy teoretyczne trzech algorytmów wykorzystanych do późniejszego sterowania modelu quadrotora. W sekcji 4.1 opisano regulator PID (proporcjonalno–całkująco–różniczkujący), który wykorzystywany jest do realizacji w środowisku symulacyjnym określonych zadań, w szczególności śledzenia trajektorii robota.

Kolejno, w 4.2, omówiony jest sterownik wykorzystujący algorytm całkowania wstecznego w zastosowaniu dla quadrotora.

Sekcja 4.3 wprowadza teorię sterowania algorytmem H_∞ podstawowego układu sterowania podając przekształcenia i warunki niezbędne do jego poprawnego zaimplementowania.

Część drugą rozpoczyna rozdział 5, w którym opisano zadania śledzenia trajektorii oraz sterowania do punktu.

W rozdziale 6.1 przedstawiono opis interfejsów danych używanych w aplikacji w celu wymiany informacji pomiędzy modelem, sterownikami oraz wizualizacją. W kolejnym rozdziale 6.2 przedstawiono szczegółowy opis implementacji modelu quadrotora z wykorzystaniem pakietu Simulink, opis wejść i wyjść, instrukcję inicjalizacji oraz przykład wdrożenia. Sekcje PID, 6.4 oraz 6.5 przybliżają czytelnikowi sposób zamodelowania w pakiecie Matlab/Simulink rozpatrywanych algorytmów sterowania quadrotorem, ze szczególnym zwróceniem uwagi na pojawiające się problemy implementacji i sposób ich rozwiązania.

Rozdziały 7.1 - 7.3 prezentują badania sterowania quadrotorem przy pomocy trzech algorytmów sterowania opisanych w niniejszej publikacji. Rozdział 7.4 przedstawia testy numerycznych metod rozwiązywania układów równań różniczkowych, a w szczególności zaimplementowanego modelu quadrotora.

Część I

Podstawy teoretyczne

2. Model dynamiki quadrotora

2.1. Wprowadzenie

Komputerowa symulacja robotów w kontekście sterowania wymaga implementacji modelu fizycznego obiektu o wymaganej dokładności. W celu jego uzyskania stosuje się formalizm Newtona–Eulera bądź Eulera–Lagrange'a. Pierwszy z nich opiera się o zasadę zachowania sił oraz zasadę zachowania momentów. Jego zaletą jest łatwość uzyskania dynamika odwrotnej, dzięki czemu jest wykorzystywany w regulatorach opartych na modelu. Formalizm Eulera–Lagrange'a jest podejściem energetycznym. Może być stosowany na dowolnych współrzędnych opisujących układ, a uzyskane przez niego równania dynamiki są w postaci jawnej. Będzie on wykorzystany w niniejszym rozdziale do uzyskania modelu dynamiki quadrotora. Wyprowadzenie zostanie przeprowadzone w oparciu o pracę [5].

2.2. Równania Eulera-Lagrange'a

W ujęciu mechaniki klasycznej lagranżjan L jest różnicą energii kinetycznej T i potencjalnej V obiektu

$$L = T - V. \tag{2.1}$$

Według *Zasady Najmniejszego Działania Hamiltona* układ zachowawczy przyjmuje trajektorie dla których funkcjonał

$$I = \int_{t_0}^{t_1} L(q(t), \dot{q}(t))\mathrm{d}t \tag{2.2}$$

osiąga wartość najmniejszą z możliwych. Praca [15] przedstawia rozwiązanie problemu minimalizacji przy pomocy rachunku wariacyjnego. W rezultacie uzyskuje się równania Eulera–Lagrange'a

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\delta L}{\delta \dot{q}_i}\right) - \frac{\delta L}{\delta q_i} = 0. \tag{2.3}$$

Zasadniczą zaletą formalizmu Eulera–Lagrange'a jest niezmienniczość formy równań, bez względu na wybór współrzędnych uogólnionych q_i. Badanie układów niezachowawczych, takich jak quadrotor, wymaga skorzystania z uogólnionych równań Eulera-Lagrange'a. Po uwzględnieniu działania sił niezachowaczych F_i względem współrzędnych q_i równanie (2.3) przyjmuje postać:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\delta L}{\delta \dot{q}_i}\right) - \frac{\delta L}{\delta q_i} = F_i \tag{2.4}$$

Siły uogólniony nie zawsze są tożsame z siłami fizycznymi. Ich postać różni się ze względu na dobór współrzędnych uogólnionych. W przypadku współrzędnych kątowych mają

wymiar momentów. Nie dla każdego doboru współrzędnych uogólnionych istnieje interpretacja fizyczna sił uogólnionych. Powyższe równania można zapisać w typowej formie równań dynamiki

$$M(q_i)\ddot{q}_i + C(q_i, \dot{q}_i)\dot{q}_i + D(q_i) = F_i, \tag{2.5}$$

gdzie M,C i D są odpowiednio macierzami bezwładności, sił odśrodkowych i Coriolisa oraz sił grawitacji obiektu.

2.3. Momenty niezachowawcze

Układ quadrorotora jest niezachowawczy ze względu na pracę wykonywaną przez silniki wirników. W wyprowadzanym modelu uwzględniono następujące siły i momenty niezachowawcze:

— Suma ciągów silników:

$$F_c = b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2), \tag{2.6}$$

gdzie Ω_i jest prędkością obrotową śmigła i, a b współczynnikiem ciągu.

— Różnica pomiędzy ciągiem generowanym przez silniki 2 i 4 (rys. 2.1):

$$F_\phi^{(1)} = bl(-\Omega_2^2 + \Omega_4^2), \tag{2.7}$$

gdzie l jest długością ramienia.

— Różnica pomiędzy ciągiem generowanym przez silniki 1 i 3 (rys. 2.2):

$$F_\theta^{(1)} = bl(-\Omega_1^2 + \Omega_3^2). \tag{2.8}$$

— Różnica pomiędzy ciągiem generowanym przez pary silników 1 i 3 oraz 2 i 4 (rys. 2.3):

$$F_\psi = d(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2), \tag{2.9}$$

gdzie d jest współczynnikiem oporu powietrza.

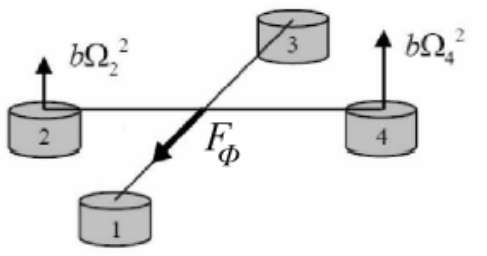
— Efekt żyroskopowy względem osi X(kąt ϕ) (rys. 2.4):

$$F_\phi^{(2)} = J_r\omega_Y(-\Omega_1 + \Omega_2 - \Omega_3 + \Omega_4), \tag{2.10}$$

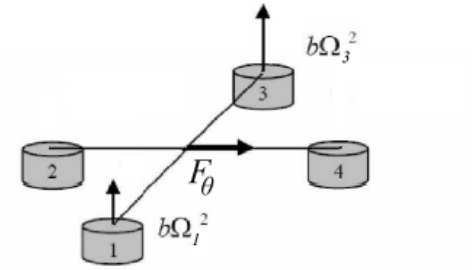
gdzie J_r jest macierzą bezwładności śmigiła, ω_Y prędkością obrotową ciała względem osi Y.

— Efekt żyroskopowy względem osi Y(kąt θ) (rys. 2.5):

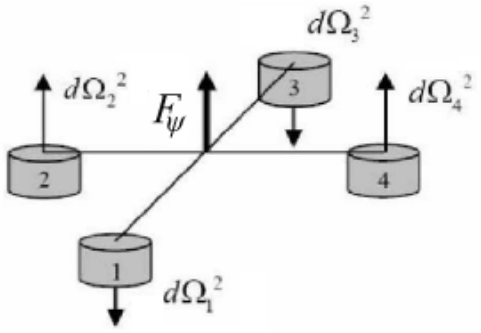
$$F_\theta^{(2)} = J_r\omega_X(\Omega_1 - \Omega_2 + \Omega_3 - \Omega_4). \tag{2.11}$$



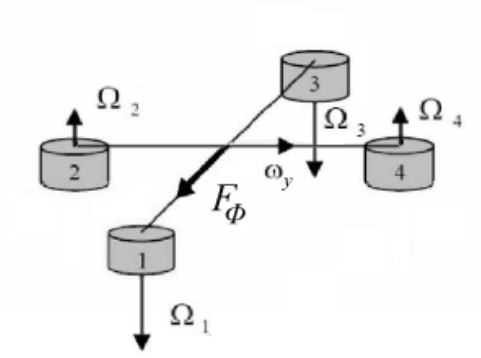
Rysunek 2.1. Moment spowodowany różnicą ciągów generowanych przez silniki 2 i 4.



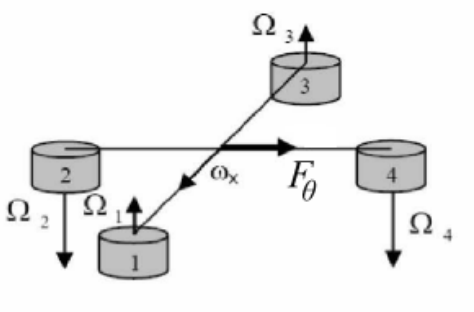
Rysunek 2.2. Moment spowodowany różnicą ciągów generowanych przez silniki 1 i 3.



Rysunek 2.3. Moment spowodowany różnicą ciągów generowanych przez pary silników 1 i 3 oraz 2 i 4.



Rysunek 2.4. Efekt żyroskopowy względem osi X(kąt ϕ)



Rysunek 2.5. Efekt żyroskopowy względem osi Y(kąt θ)

2.4. Współrzędne położenia

Do wyznaczenia równań dynamiki dla współrzędnych położenia wystarczy skorzystać z drugiej zasady dynamiki Newtona

$$a = \frac{1}{m}F, \tag{2.12}$$

gdzie m jest masą obiektu, F siłą wypadkową działającą na niego, zaś a przyśpieszeniem ciała.
Wypadkowa siła działającą na quadrotora składa się z siły grawitacji

$$F_g = mg \tag{2.13}$$

oraz sumy ciągów silników

$$F_c = b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2), \tag{2.14}$$

gdzie Ω_i jest prędkością obrotową śmigła i. Załóżmy, że siła grawitacji oddziaływuje wzdłuż globalnej osi z, i zauważmy, że kierunek siły ciągu silników jest zgodny z normalną quadrotora Z. W celu wyrażenia wektora opisanego we współrzędnych quadrotora w układzie współrzędnych związanym z ziemią, posłużono się wzorem

$$\vec{r}_{x,y,z}(X,Y,Z) = R(\phi,\theta,\psi)\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}, \tag{2.15}$$

gdzie

$$R(\phi,\theta,\psi) = \begin{bmatrix} c_\psi c_\theta & c_\psi s_\theta s_\phi - s_\psi c_\phi & c_\psi s_\theta c_\phi + s_\psi s_\phi \\ s_\psi c_\theta & s_\psi s_\theta s_\phi + c_\psi c_\phi & s_\psi s_\theta c_\phi - c_\psi s_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix}. \tag{2.16}$$

Podstawiając siłę wypadkową do równania (2.12) uzyskujemy

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} = -\begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} + \frac{1}{m}R(\phi,\theta,\psi)\begin{pmatrix} 0 \\ 0 \\ b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \end{pmatrix}. \tag{2.17}$$

Po uproszczeniu równanie (2.17) otrzymuje następującą postać

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} = -\begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} + \frac{b}{m}(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2)\begin{pmatrix} c_\psi s_\theta c_\phi + s_\psi s_\phi \\ s_\psi s_\theta c_\phi - c_\psi s_\phi \\ c_\theta c_\phi \end{pmatrix}. \tag{2.18}$$

2.5. Współrzędne kątowe

Do wyprowadzenia równań dynamiki dla współrzędnych kątowych zostaną użyte uogólnione równania Eulera–Lagrange'a (2.4). Jako współrzędne uogólnione posłużą kąty *KKM* – (ϕ, θ, ψ).

2.5.1. Energia kinetyczna

Pierwszą składową lagranżjanu jest energia kinetyczna. Ponieważ współrzędnymi uogólnionymi są kąty *KKM*, należy uzyskać wzór energii w odpowiadającym im prędkościach obrotowych. W tym celu różniczkowane jest wyrażenie (2.15). Dla przejrzystości zapisu rozdzielono zapis macierzowy do układu równań.

$$\begin{cases} v_x(X,Y,Z) & = (-s_\theta c_\psi \dot{\theta} - c_\theta s_\psi \dot{\psi})X \\ & +(-c_\psi c_\phi \dot{\psi} + s_\psi s_\phi \dot{\phi} - s_\psi s_\phi s_\theta \dot{\psi} + c_\psi c_\phi s_\theta \dot{\phi} + c_\psi s_\phi c_\theta \dot{\theta})Y \\ & +(c_\psi s_\phi \dot{\psi} + s_\psi c_\phi \dot{\phi} - s_\psi c_\phi s_\theta \dot{\psi} - c_\psi s_\phi s_\theta \dot{\phi} + c_\psi c_\phi c_\theta \dot{\theta})Z \\ v_y(X,Y,Z) & = (-s_\theta s_\psi \dot{\theta} - c_\theta c_\psi \dot{\psi})X \\ & +(-s_\psi c_\phi \dot{\psi} - c_\psi s_\phi \dot{\phi} + c_\psi s_\phi s_\theta \dot{\psi} + s_\psi c_\phi s_\theta \dot{\phi} + s_\psi s_\phi c_\theta \dot{\theta})Y \\ & +(s_\psi s_\phi \dot{\psi} - c_\psi c_\phi \dot{\phi} + c_\psi c_\phi s_\theta \dot{\psi} - s_\psi s_\phi s_\theta \dot{\phi} + s_\psi c_\phi c_\theta \dot{\theta})Z \\ v_z(X,Y,Z) & = (-c_\theta \dot{\theta})X \\ & +(c_\phi c_\theta \dot{\phi} - s_\phi s_\theta \dot{\theta})Y \\ & +(-s_\phi c_\theta \dot{\phi} - c_\phi s_\theta \dot{\theta})Z \end{cases} \tag{2.19}$$

Kwadrat prędkości dowolnego punktu układu quadrotora w układzie ziemi przyjmuje postać:

$$v^2(X,Y,Z) = v_x^2(X,Y,Z) + v_y^2(X,Y,Z) + v_z^2(X,Y,Z) \tag{2.20}$$

Po wykonaniu przekształceń uzyskano następujące wyrażenia:

$$\begin{aligned} v^2(X,Y,Z) = & (Y^2+Z^2)(\dot{\phi} - \dot{\psi}s_\theta)^2 \\ & + (Z^2+X^2)(\dot{\theta}c_\phi + \dot{\psi}s_\phi c_\theta)^2 \\ & + (X^2+Y^2)(\dot{\theta}s_\phi + \dot{\psi}c_\phi c_\theta)^2 \\ & + 2XY(\dot{\psi}^2 s_\phi s_\theta c_\theta + \dot{\psi}(c_\phi s_\theta \dot{\theta} - s_\phi c_\theta \dot{\phi}) - c_\phi \dot{\phi}\dot{\theta}) \\ & + 2ZX(\dot{\psi}^2 c_\phi s_\theta c_\theta + \dot{\psi}(-s_\phi s_\theta \dot{\theta} - c_\phi c_\theta \dot{\phi}) + s_\phi \dot{\phi}\dot{\theta}) \\ & + 2YZ(-\dot{\psi}^2 s_\phi c_\theta c_\phi + \dot{\psi}(s^2\phi c_\theta \dot{\theta} - c^2\phi c_\theta \dot{\theta}) + s_\phi c_\phi \dot{\theta}^2) \end{aligned} \tag{2.21}$$

Energia kinetycznego dla bryły sztywnej jest wyrażona wzorem [15]

$$T = \frac{1}{2}\int_{m_q} v^2 \mathrm{d}m \tag{2.22}$$

Całkując wyrażenie (2.21) po masie na obszarze quadrotora m_q uzyskano wzór na energie kinetyczną ruchu obrotowego robota (2.22).

$$\begin{aligned}
T = &\frac{1}{2}\int_{m_q}(Y^2+Z^2)\mathrm{d}m\cdot(\dot{\phi}-\dot{\psi}s_\theta)^2 \\
&+\frac{1}{2}\int_{m_q}(Z^2+X^2)\mathrm{d}m\cdot(\dot{\theta}c_\phi+\dot{\psi}s_\phi c_\theta)^2 \\
&+\frac{1}{2}\int_{m_q}(X^2+Y^2)\mathrm{d}m\cdot(\dot{\theta}s_\phi+\dot{\psi}c_\phi c_\theta)^2 \\
&+\int_{m_q}XY\mathrm{d}m\cdot(\dot{\psi}^2 s_\phi s_\theta c_\theta+\dot{\psi}(c_\phi s_\theta\dot{\theta}-s_\phi c_\theta\dot{\phi})-c_\phi\dot{\phi}\dot{\theta}) \\
&+\int_{m_q}ZX\mathrm{d}m\cdot(\dot{\psi}^2 c_\phi s_\theta c_\theta+\dot{\psi}(-s_\phi s_\theta\dot{\theta}-c_\phi c_\theta\dot{\phi})+s_\phi\dot{\phi}\dot{\theta}) \\
&+\int_{m_q}YZ\mathrm{d}m\cdot(-\dot{\psi}^2 s_\phi c_\theta c_\phi+\dot{\psi}(s^2\phi c_\theta\dot{\theta}-c^2\phi c_\theta\dot{\theta})+s_\phi c_\phi\dot{\theta}^2).
\end{aligned} \tag{2.23}$$

Ze względu na założoną symetryczną budowę quadrotora, wyrażenia związane z momentami dewiacyjnymi można pominąć. Momenty główne zostały oznaczone I_X, I_Y, I_Z. Ostatecznie energia kinetyczna:

$$\begin{aligned}
T = &\frac{1}{2}I_X(\dot{\phi}-\dot{\psi}s_\theta)^2 \\
&+\frac{1}{2}I_Y(\dot{\theta}c_\phi+\dot{\psi}s_\phi c_\theta)^2 \\
&+\frac{1}{2}I_Z(\dot{\theta}s_\phi+\dot{\psi}c_\phi c_\theta)^2.
\end{aligned} \tag{2.24}$$

2.5.2. Energia potencjalna

Drugą składową lagranżjanu jest energia potencjalna układu. Wyraża się następującym wzorem:

$$\begin{aligned}
V &= -g\int_{m_q}z\mathrm{d}m \\
&= -g\int_{m_q}X\mathrm{d}m\cdot(s_\theta) \\
&- g\int_{m_q}Y\mathrm{d}m\cdot(s_\phi c_\theta) \\
&- g\int_{m_q}Z\mathrm{d}m\cdot(c_\phi c_\theta).
\end{aligned} \tag{2.25}$$

2.5.3. Równanie Eulera-Lagrange'a

Posiadając analityczną postać energii kinetycznej (2.24) i potencjalnej (2.25) oraz uogólnionych sił niezachowawczych (2.3) możliwe jest zastosowanie uogólnionych równań Eulera-Lagrange'a (2.4).

$$\begin{aligned}
F_\phi^{(1)}+F_\phi^{(2)} &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\delta L}{\delta\dot{\phi}}\right)-\frac{\delta L}{\delta\phi}, \\
F_\psi &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\delta L}{\delta\dot{\psi}}\right)-\frac{\delta L}{\delta\psi}, \\
F_\theta^{(1)}+F_\theta^{(2)} &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\delta L}{\delta\dot{\theta}}\right)-\frac{\delta L}{\delta\theta}.
\end{aligned} \tag{2.26}$$

W fizycznym quadrotorze niemożliwe jest bezpośrednie mierzenie kątów *KKM*. Powszechnie wykorzystywane jest urządzenie *IMU* (z ang. *inertial measurement unit*), będące kombinacją akcelerometrów oraz żyroskopów. Za jego pomocą odczytywane są prędkości i przyśpieszenia kątowe związane z układem ciała robota.Z tego względu dokonujemy w modelu (2.26) zamiany prędkości $\dot{\phi}, \dot{theta}, \dot{psi}$ ma przez *IMU* i oznaczone przez $\omega_X, \omega_Y, \omega_Z$. Transformacja wykonywana jest następującym wzorem:

$$\begin{bmatrix}\omega_X\\ \omega_Y\\ \omega_Z\end{bmatrix} = \begin{bmatrix}1 & 0 & -s_\theta\\ 0 & c_\phi & s_\phi c_\theta\\ 0 & -s_\phi & c_\phi c_\theta\end{bmatrix}\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} \tag{2.27}$$

W ten sposób otrzymujemy

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\delta L}{\delta\dot{\phi}}\right) - \frac{\delta L}{\delta\phi} &= I_X\dot{\omega}_X - (I_Y - I_Z)\omega_Y\omega_Z\\
&\quad + \int_{m_q} Y\mathrm{d}m\cdot(-gc_\phi c_\theta)\\
&\quad + \int_{m_q} Z\mathrm{d}m\cdot(gs_\phi c_\theta)\\
\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\delta L}{\delta\dot{\theta}}\right) - \frac{\delta L}{\delta\theta} &= -s_\phi(I_Z\dot{\omega}_Z - (I_X - I_Y)\omega_X\omega_Y)\\
&\quad + c_\phi(I_Y\dot{\omega}_Y - (I_Z - I_X)\omega_X\omega_Z)\\
&\quad + \int_{m_q} X\mathrm{d}m\cdot(-gc_\theta)\\
&\quad + \int_{m_q} Y\mathrm{d}m\cdot(-gs_\phi s_\theta)\\
&\quad + \int_{m_q} Z\mathrm{d}m\cdot(-gc_\phi s_\theta)\\
\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\delta L}{\delta\dot{\phi}}\right) - \frac{\delta L}{\delta\phi} &= -s_\theta(I_X\dot{\omega}_X - (I_Y - I_Z)\omega_Y\omega_Z)\\
&\quad + s_\phi c_\theta(I_Y\dot{\omega}_Y - (I_Z - I_X)\omega_X\omega_Z)\\
&\quad + c_\phi c_\theta(I_Z\dot{\omega}_Z - (I_X - I_Y)\omega_X\omega_Y)
\end{aligned} \tag{2.28}$$

Po zastosowaniu w wyrażeniach (2.28) przybliżenia liniowego dla małych wartości kątów kiwania i kołysania ($\alpha \approx 0 \rightarrow \sin\alpha \approx 0, \cos\alpha \approx 1$) uzyskano równania dynamiki układu dla kątów KKM.

$$\begin{cases}
\ddot{\phi} &= \frac{bl(\Omega_2^2-\Omega_4^2)+J_r\dot{\theta}(-\Omega_1^2+\Omega_2^2-\Omega_1^3+\Omega_4^2)+(I_Y-I_Z)\dot{\psi}\dot{\theta}}{I_X}\\
\ddot{\theta} &= \frac{bl(\Omega_3^2-\Omega_1^2)+J_r\dot{\phi}(\Omega_1^2-\Omega_2^2+\Omega_1^3-\Omega_4^2)+(I_Z-I_X)\dot{\psi}\dot{\phi}}{I_Y}\\
\ddot{\psi} &= \frac{d(\Omega_1^2-\Omega_2^2+\Omega_3^2-\Omega_4^2)+(I_X-I_Y)\dot{\theta}\dot{\phi}}{I_Z}
\end{cases} \tag{2.29}$$

2.6. Przyjęty model

W celu nadania wyjściom sterującym sensu sił (momentów oddziałujących na quadrotor) opisanych w podrozdziale 2.3 wprowadzimy nowy wektor sterowań $U = \begin{pmatrix} U_1 & U_2 & U_3 & U_4 \end{pmatrix}^T$ oraz zmienną pomocniczą Ω w następujący sposób [5]

$$\begin{cases} U_1 = b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ U_2 = b(-\Omega_2^2 + \Omega_4^2) \\ U_3 = b(-\Omega_1^2 + \Omega_3^2) \\ U_4 = d(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2) \\ \Omega = \Omega_2 + \Omega_4 - \Omega_1 - \Omega_3. \end{cases} \tag{2.30}$$

Łatwo można wyznaczyć wartości prędkości obrotowych śmigieł Ω_i w zależności od wektora U, a mianowicie

$$\begin{cases} \Omega_1^2 = \frac{U_1}{4b} - \frac{U_3}{2bl} - \frac{U_4}{4d} \\ \Omega_2^2 = \frac{U_1}{4b} - \frac{U_2}{2bl} + \frac{U_4}{4d} \\ \Omega_3^2 = \frac{U_1}{4b} + \frac{U_3}{2bl} - \frac{U_4}{4d} \\ \Omega_4^2 = \frac{U_1}{4b} + \frac{U_2}{2bl} + \frac{U_4}{4d}. \end{cases} \tag{2.31}$$

Zestawiając ze sobą równania (2.18) oraz (2.29) z uwzględnieniem zależności (2.30) otrzymujemy kompletny model dynamiki quadrotora w postaci

$$\begin{cases} \ddot{x} = (c_\psi s_\theta c_\phi + s_\psi s_\phi)\frac{1}{m}U_1 \\ \ddot{y} = (s_\psi s_\theta c_\phi - c_\psi s_\phi)\frac{1}{m}U_1 \\ \ddot{z} = -g + (c_\theta c_\phi)\frac{1}{m}U_1 \\ \ddot{\phi} = \dot{\theta}\dot{\psi}(\frac{I_Y - I_Z}{I_X}) - \frac{J_r}{I_X}\dot{\theta}\Omega + \frac{l}{I_X}U_2 \\ \ddot{\theta} = \dot{\phi}\dot{\psi}(\frac{I_Z - I_X}{I_Y}) + \frac{J_r}{I_Y}\dot{\phi}\Omega + \frac{l}{I_Y}U_3 \\ \ddot{\psi} = \dot{\phi}\dot{\theta}(\frac{I_X - I_Y}{I_Z}) + \frac{1}{I_Z}U_4. \end{cases} \tag{2.32}$$

W literaturze pojawia się kilka wersji modelu quadrotora. Ważne, aby w przypadku wszystkich działań porównawczych zawsze opierać się na tej samej podstawie teoretycznej. Tutaj zdecydowano się zaimplementować powyższy model z kilku powodów. Po pierwsze, jest to jeden z pierwszych przedstawionych modeli, wszystkie kolejne odwołują się do niego i na nim bazują. Model ten jest bardzo dobrze opisany w literaturze, zdecydowana większość publikacji znanych autorom przyjmuje ten opis. Należy jednakże przy próbie powtórzenia symulacji zwrócić uwagę na różnice występujące w używanych modelach. Przykładowo praca [8] wprowadza przeskalowanie sterowań U_2 oraz U_3 przez odległość śmigła od środka modelu l, dzięki czemu prędkość zmian kątów kiwania i myszkowania zostaje powiązana z modelem fizycznym. Z kolei w [11] oprócz tego przyjęto odwrotny kierunek osi z, czego skutkiem jest zmiana znaku przed sterowaniem U_3.

3. Modelowanie matematyczne

3.1. Modelowanie matematyczne ciągu

Siła nośna quadrotora uzyskiwana jest z 4 śmigieł kręcących się parami (ustawionymi przeciwlegle) w przeciwnych kierunkach. Śmigła napędzane są silnikami elektrycznymi, które są sterowane przez sterownik. Każde z nich generuje własny ciąg zależny od prędkości obrotowej.

Poniżej przedstawiono modelowanie ciągu dla badanego obiektu jakim jest quadrotor. Następnie przybliżono czytelnikowi problem modelowania silników prądu stałego (gdyż taki typ silników stosowany jest w badanym obiekcie). Ostatnia zaś sekcja niniejszego działu omawia zaburzenia jakie mogą oddziaływać na quadrotor.

3.1.1. Modelowanie ciągu

W przypadku prac rozważających jedynie algorytmy sterowania i samolokalizacji robota często ten aspekt jest pomijany [22]. W przypadku, gdy autorzy chcą jedynie pokazać, że takie zjawisko istnieje, stosowana jest kwadratowa zależność pomiędzy prędkością obrotową silników, a ciągiem śmigieł – taki model silników jest najczęściej spotykany w literaturze [4]. Jest to uproszczony model wyprowadzony z teorii momentu [7]. W przypadku, gdy prace omawiają wpływ ciągu na działanie algorytmów stosowany jest pełny model wyprowadzony z teorii momentu [12].

Istnieją także prace poświęcone jedynie modelowaniu ciągu quadrotorów [13]. W ich przypadku model wyprowadzony z teorii momentu uzupełniany jest o nieliniowe współczynniki wynikające z licznych zjawisk aerodynamicznych występujących na łopatach śmigieł obracających się z dużą prędkością.

W modelu quadrotora stworzonym na potrzeby tego projektu została zamodelowana zależność kwadratowa pomiędzy wartością prędkości obrotowej łopat śmigieł (Ω_1, Ω_2, Ω_3, Ω_4) a ciągiem przez nie generowanym (U_1, U_2, U_3, U_4). Jest ona opisana następującymi równaniami:

$$\begin{cases} U_1 = b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ U_2 = b(-\Omega_2^2 + \Omega_4^2) \\ U_3 = b(-\Omega_1^2 + \Omega_3^2) \\ U_4 = d(-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2) \end{cases} . \tag{3.1}$$

Przyjęty model pozwala na wyliczenie prędkości obrotowych śmigieł dla narzuconego sterowania. Transformacja taka jest opisana równaniami:

$$\begin{cases} \Omega_1^2 = \frac{1}{4b}U_1 - \frac{1}{2b}U_3 - \frac{1}{4d}U_4 \\ \Omega_2^2 = \frac{1}{4b}U_1 - \frac{1}{2b}U_2 + \frac{1}{4d}U_4 \\ \Omega_3^2 = \frac{1}{4b}U_1 + \frac{1}{2b}U_3 - \frac{1}{4d}U_4 \\ \Omega_4^2 = \frac{1}{4b}U_1 + \frac{1}{2b}U_2 + \frac{1}{4d}U_4 \end{cases} . \tag{3.2}$$

Ponieważ wszystkie prędkości obrotowe są dodatnie, transformacja (3.2) daje jednoznaczne rozwiązanie. Dzięki temu możliwe jest symulowanie efektów oddziaływających na quadrotor wynikających z prędkości obrotowych śmigieł w przypadku algorytmów, które sterują jedynie ciągiem silników.

3.1.2. Modelowanie silników prądu stałego

Poza modelowaniem zależności pomiędzy prędkością obrotową a ciągiem śmigła, model quadrotora można uzupełnić o model matematyczny napędów – zazwyczaj silników prądu stałego.

Podobnie jak w przypadku modelowania ciągu i tutaj zależność pomiędzy napięciem zadawanym na silniki, a prędkością obrotową wału może zostać pominięta [22].

Model matematyczny silników prądu stałego wynikający z praw Ohma, Kirchoffa i Newtona przedstawiony w [20] można opisać równaniami:

$$\begin{cases} F(t)r = Ki(t) = M(t) \\ U = Ri(t) + L\frac{di(t)}{dt} + K\omega(t) \\ M(t) = J\frac{d\omega}{dt}(t) + f\omega(t) + M_f(\omega(t)) + M_r(t) \end{cases} \tag{3.3}$$

Z równań (3.3) można wyprowadzić następującą zależność pomiędzy wyjściem ω a wejściami U i M_r:

$$JL\frac{d^2\omega}{dt^2}(t) + (JR + fL)\frac{d\omega}{dt}(t) + (Rf + K^2)\omega(t) = -L\frac{dM_r}{dt}(t) - RM_r(t) + KU(t). \tag{3.4}$$

Często, zważywszy na niewielką wartość indukcyjności silników stosowanych w konstrukcjach tego typu, równanie (3.4) jest upraszczane do

$$JR\frac{d\omega}{dt}(t) + (Rf + K^2)\omega(t) = -RM_r(t) + KU(t). \tag{3.5}$$

Podobne, uproszczone modele są stosowane w [12] i [4]. Ponadto w [4] została pokazana linearyzacja modelu silnika w punkcie pracy.

Model quadrotora przyjęty na potrzeby tego projektu nie uwzględnia matematycznego modelowania napędów.

3.1.3. Zaburzenia oddziałujące na quadrotor

Na unoszący się quadrotor oddziałuje duża liczba sił i momentów, które mogą zostać uwzględnione w modelu. Poniżej przedstawiono te, które są najczęściej opisywane w literaturze, np. w [13].

1. Zaburzenia ciągu.
   a) Zaburzenia ciągu wynikające z bliskości ziemi/sufitu – quadrotor na niewielkich wysokościach (rzędu kilku cm) doświadcza zwiększonej siły nośnej wynikającej z odbitego ciągu generowanego przez łopaty śmigieł od ziemi. Zjawisko to można opisać równaniem:

   $$\frac{T}{T_\infty} = \frac{1}{1 - (\frac{R_r}{4z_h})^2}. \tag{3.6}$$

   Podobne zjawisko obserwowane jest w przypadku bliskości sufitu. W tym przypadku nad śmigłami tworzy się podciśnienie zwiększając siłę nośną quadrotora. Efekt sufitu jest często modelowany tym samym wzorem co efekt bliskości ziemi.
   b) Zaburzenia ciągu wynikające ze zmiany kąta natarcia łopat – siła nośna zależy od kąta natarcia śmigła. Wraz ze zmianą wiatru lub też ruchem quadrotora kąt natarcia może ulec zmianie, co wpływa na siłę ciągu generowaną przez śmigła.

c) Zaburzenia ciągu wynikające z drgania łopat (blade flapping) – zmiana geometrii śmigieł wynikająca z drgania łopat podczas lotu poziomego. Efekt powoduje także zmianę kąta natarcia śmigła.
d) Zaburzenia ciągu wynikające z oddziaływania sił horyzontalnych na elementy śmigła (hub force) – pomijalna wartość przy występowaniu wcześniej opisanych oddziaływań.
e) Zaburzenia ciągu wynikające z innego udźwigu łopaty atakującej i łopaty cofającej podczas lotu do przodu (rolling moment) – pomijalna wartość przy występowaniu innych efektów.
f) Zaburzenia ciągu wynikające z zależności gęstości powietrza od wysokości również są pomijalne, gdyż quadrotory poruszają się na stosunkowo niewielkich wysokościach i podczas pojedynczego lotu nie zmieniają znacznie swojej wysokości. Większe zmiany wysokości są niemożliwe ze względu na niewielką pojemność akumulatorów stosowanych w popularnych konstrukcjach.
g) Zaburzenia ciągu wynikające z zawirowań powietrza generowanych przez inne maszyny latające – zauważalne w przypadku latania w zwartej formacji.

2. Efekt żyroskopowy ramy quadrotora – każde ciało obracające się ma zdolność do utrzymania swojej orientacji względem płaszczyzny prostopadłej do osi obrotu. Wynika to z zasady zachowania pędu. Podobnie jest z obracającym się quadrocopterem. Wpływ efektu żyroskopowego na kąty KKM jest zazwyczaj zawarty bezpośrednio w równaniach opisujących modele matematyczne quadrotorów

$$\begin{cases} \ddot{\phi} = \dot{\theta}\dot{\psi}\frac{I_y - I_z}{I_x} \\ \ddot{\theta} = \dot{\phi}\dot{\psi}\frac{I_z - I_x}{I_y} \\ \ddot{\psi} = \dot{\theta}\dot{\psi}\frac{I_x - I_y}{I_z} \end{cases} . \tag{3.7}$$

3. Efekt żyroskopowy śmigieł – obok efektu opisanego wcześniej jest to najczęściej modelowane zjawisko zaburzające quadrotor. Bezwładność obracających się z dużą prędkością śmigieł przeciwdziała kiwaniu i kołysaniu się quadrokoptera. Wpływ tego efektu na kąty KKM zazwyczaj modelowany jest następująco:

$$\begin{cases} \ddot{\phi} = -\dot{\theta}\frac{J_r}{I_x}\Omega \\ \ddot{\theta} = -\dot{\phi}\frac{J_r}{I_y}\Omega \\ \ddot{\psi} = 0 \end{cases} . \tag{3.8}$$

4. Opory powietrza – zazwyczaj modelowane są jako:

$$F_X = \frac{1}{2} d \rho_{air} \dot{x}^2 A_X, \tag{3.9}$$

gdzie oznaczenia zgodnie ze spisem oznaczeń umieszczonym na początku pracy.

5. Wiatr – często modelowany jako dodatkowa prędkość dodawana do oporów powietrza opisanych równaniem (3.9).

Zakłóceniami zawartymi w analizowanym modelu są efekty żyroskopowe ramy quadrotora i łopat jego śmigieł opisane równaniami (3.7) - (3.8).

3.2. Numeryczne metody rozwiązywania równań różniczkowych

3.2.1. Wstęp do całkowania numerycznego

Równania różniczkowe są częstym sposobem opisu rozmaitych procesów i zjawisk takich jak np. mechanika płynów, rozchodzenie się fal dźwiękowych. W niniejszym dokumencie podstawą naszych zainteresowań jest rozwiązywanie układów dynamiki które również zapisane są za pomocą równań różniczkowych.

Gdy analizujemy układy równań teoretycznie najlepszym sposobem ich rozwiązywania (pod względem jakościowym) jest rozwiązanie ich w sposób analityczny – dzięki temu otrzymujemy wzór który może być dalej badany. Niestety często wyprowadzenie rozwiązania jest skomplikowane, czasochłonne lub też niemożliwe. W takich należy zastosować metody numeryczne otrzymując numeryczne rozwiązanie problemu.

3.2.2. Klasy równań różniczkowych

Równania różniczkowe ze względu na metody rozwiązywania równań numerycznych dzielą się na dwie klasy:

— Równania różniczkowe zwyczajne – *ODE - ordinary differential equations* posiadają funkcje jednej zmiennej (np. zmiennej czasu t)

— Równania różniczkowe cząstkowe – *PDE - partial differential equations* posiadają funkcje wielu zmiennych

Równania ODE mają ogólną formę przedstawioną w równaniu 3.10

$$F(t, q, \frac{dq}{dt}, \frac{d^2q}{dt^2}, \ldots, \frac{d^kq}{dt^k}) = 0 \tag{3.10}$$

gdzie t – zmienna, q – $q \in C^k((a, b), \mathbb{R}^n)$ jest funkcją zmiennej t. Przykładem takiego równania może być równanie 3.11.

$$\frac{d^2q}{dt^2} + a\frac{dq}{dt} + bq = 0 \tag{3.11}$$

Równania PDE różnią się od ODE tym że zmiennych po których następuje całkowanie jest więcej niż w przypadku ODE przykładem takiego równania może być 3.12

$$\frac{d^2q}{dx} + \frac{d^2q}{dy} = 0 \tag{3.12}$$

gdzie q to funkcja zmiennych x, y.

W tej pracy będziemy stosowane będą równania typu ODE, gdzie zmienną po której różniczkujemy będzie czas, w związku z tym omówione zostaną metody numeryczne dla tej klasy równań.

3.2.3. Szacowanie błędu

Aby oszacować błąd poszczególnych metod rozwiązywania równań różniczkowych stosuje się notację $O(h)$ jej stosowanie jest bardzo intuicyjne: Jeśli po zmniejszeniu kroku o połowę błąd całkowania maleje o połowę to będzie to klasa $O(h)$ jeżeli zaś maleje czterokrotnie to jest to klasa $O(h^2)$ i tak dalej. Klasę błędu szacujemy z następującego wzoru:

$$e_h \leqslant Ch^{p+1} \tag{3.13}$$

Gdzie e_h to błąd lokalny ("*local truncation error*"), C jest stałą dodatnią, a p jest rzędem danej metody.

3.2.4. Metoda Eulera

Chcąc wyznaczyć rozwiązanie równania różniczkowego w postaci 3.14 wyznaczamy jego przybliżenie dla przyrostu $h > 0$:

$$y'(t) \approx f(t, y(t)) = \frac{y(t+h) - y(t)}{h} \tag{3.14}$$

Przekształcając równanie 3.14 do postaci

$$y(t+h) = y(t) + h * f(t, y(t)) \tag{3.15}$$

Otrzymujemy podstawową metodę Eulera. Metoda ta jest najprostszą i najbardziej intuicyjną metodą całkowania, jej błąd silnie zależy od przyjętego kroku całkowania a także od sztywności równania. Metoda Eulera jest metodą pierwszego rzędu $O(h^2)$

3.2.5. Metoda Rungego-Kutty

Metoda Rungego-Kutty (w skrócie RK) drugiego rzędu bazuje na wyznaczeniu wyniku na podstawie średniego przyrostu (jak w m. Eulera) oraz punktu pośredniego [23].

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1) \\ y_{n+1} &= y_n + k_2 \end{aligned} \tag{3.16}$$

Na założeniu przyjęcia jednego punktu pośredniego bazuje cała rodzina metod Runge-Kutty drugiego rzędu (np. w metodzie Heuna wyliczamy wartość kroku z wzoru $y_{n+1} = y_n + \frac{1}{2}(k_1 + k_2)$). Błąd polepsza się do rzędu drugiego $O(h^3)$.

Metoda Rungego-Kutty rzędu czwartego bazuje natomiast na czterech punktach pośrednich a błąd polepsza się do klasy czwartej $O(h^5)$.

Istnieje wiele odmian metod Rungego-Kutty różnią się one współczynnikami dla długości kroku (przed h), sposobie uwzględniania poprzednich kroków w krokach kolejnych i sposobie wyliczania wyniku z kroków, jednak ogólna idea pozostaje ta sama.

3.2.6. Równania sztywne

Największym zagrożeniem dla metod całkowania numerycznego są równania sztywne (ang. *stiff equations*). Niestety nie ma jednolitej definicji dla równań sztywnych. Możemy jednak zdefiniować to intuicyjnie: są to takie równania które przy całkowaniu niektórymi metodami numerycznymi okazują się numerycznie niestabilne. Aby zapewnić numeryczną stabilność należy albo znacząco zmniejszyć krok całkowania lub zmienić metodę.

3.2.7. Podwajanie kroku

Aby zwiększyć dokładność metod całkowania można zastosować metodę podwajania kroku. Metoda ta jest prosta w zastosowaniu:

Wykonujemy krok całkowania h, a potem powtarzamy obliczenia wykonując podwójne całkowanie z krokiem $\frac{1}{2}h$, różnica będzie estymatorem błędu całkowania który pomoże nam poprawić wynik:

$$\begin{aligned} \Delta &= y_2 - y_1 \\ y(x+h) &= y_2 + \frac{\Delta}{15} \end{aligned}$$

Gdzie y_1 to całkowanie pojedyncze z krokiem h a y_2 to dwa całkowania z krokiem $\frac{h}{2}$.

3.2.8. Zmienny krok całkowania

Na schemacie podwójnego obliczania kroku bazują metody rozszerzone.

$$y_{n+1} = y_n + c_1 k_1 + c_2 k_2 + ... + c_m k_m$$
$$y^*_{n+1} = y^*_n + c^*_1 k_1 + c^*_2 k_2 + ... + c^*_m k_m$$

W metodach podstawowych RK bazujemy jedynie na y_{n+1} natomiast w rozszerzonych (*embedded RK methods*) korzystamy dodatkowo z y^*_{n+1} (czyli całkujemy z krokiem o połowę mniejszym) co pozwala na estymację błędu za pomocą błędu metody (*truncation error*):

$$\Delta = y_{n+1} - y^*_{n+1} \tag{3.17}$$

dzięki temu mamy szansę na poprawę kroku algorytmu.

Określamy dokładność za pomocą Δ_0, następnie obliczamy bieżącą wartość błędu Δ_1, jeśli jest ona znacząco mniejsza od zadanej dokładności to podwajamy krok. Jeśli niewiele mniejsza to zostawiamy bieżący krok. Jeśli natomiast przekracza błąd zmniejszamy krok całkowania o połowę.

Innym sposobem doboru kroku jest wyznaczanie jego wartości na podstawie wzoru:

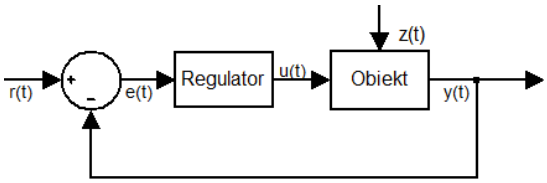
$$h_0 = h_1 \left| \frac{\Delta_0}{\Delta_1} \right|^{0.2} \tag{3.18}$$

W przypadku gdy $\frac{\Delta_0}{\Delta_1} > 1$ powtarzamy całkowanie z nowym krokiem h_0, w przeciwnym wypadku wykonujemy krok naprzód o wartości h_0. Ten sposób postępowania nazywany jest adaptacyjnym krokiem całkowania.

4. Algorytmy sterowania

4.1. Regulator PID

Ogólnie, w układzie regulacji (rysunek 4.1) uchybem regulacji $e(t)$ nazywamy różnicę pomiędzy wartością zadaną $r(t)$ a wielkością wyjściową obiektu $y(t)$. Uchyb (błąd sterowania) jest



Rysunek 4.1. Układ regulacji

sygnałem, na podstawie którego regulator generuje sygnał sterujący $u(t)$ dla obiektu. Na obiekt dodatkowo działają jeszcze zakłócenia $z(t)$ [1]. Regulator na podstawie wartości $e(t)$ steruje obiektem w pożądany przez nas sposób (na przykład minimalizując błąd śledzenia trajektorii).

Jednym z najbardziej znanych przykładów sterowników, obecnie najczęściej wykorzystywanym w przemyśle, jest regulator PID – rozpatrywany pod względem zastosowania w robotach typu quadrotor m. in. w [3, 8, 10]. Składa się on z trzech członów kompensujących uchyb – proporcjonalnego, całkującego oraz różniczkującego, pracujących w pętli sprzężenia zwrotnego. Zachowanie takiego sterownika opisuje równanie [1]

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}, \tag{4.1}$$

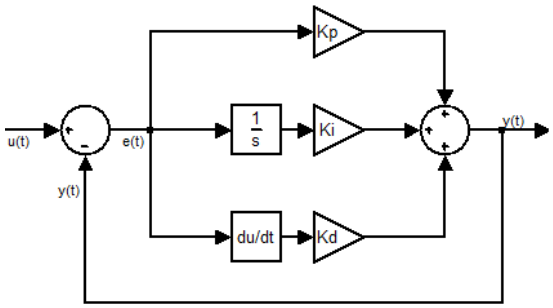
gdzie jako K_p określamy wzmocnienie części proporcjonalnej, jako K_i wzmocnienie części całkującej a jako K_d wzmocnienie części różniczkującej. W dziedzinie liczb zespolonych po zastosowaniu transformacji Laplace'a (co jest potrzebne przy implementacji algorytmu w środowisku MATLAB Simulink), można zapiać równanie (4.1) jako

$$u(s) = (K_p + \frac{K_i}{s} + sK_d)e(s). \tag{4.2}$$

Regulator PID jest szeroko wykorzystywany w konstrukcjach robotów typu quadrocopter, ponieważ:
— posiada bardzo prostą strukturę,
— jest łatwo implementowany sprzętowo,
— jest wydajnym obliczeniowo regulatorem nawet dla wielu procesów,
— posiada szereg algorytmów opartych na metodach Zieglera-Nicholsa pozwalających dostroić go nawet bez znajomości modelu.

Modyfikacje regulatora PID

Równanie (4.2) opisuje regulator PID w tzw. postaci równoległej, w której wszystkie składowe są od siebie niezależne. Z tego powodu postać ta jest zwana algorytmem PID–IND (od *INDependent algorithm*) i używana najczęściej w pracach naukowych (chociażby w [8,11]). Przedstawia ją rysunek (4.2).



Rysunek 4.2. Schemat ideowy regulatora PID–IND

Drugą postać regulatora PID, zwaną algorytmem PID–ISA (od *Ideal Standard Algorithm*), określa równanie

$$u(t) = k_r \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de(t)}{dt} \right), \tag{4.3}$$

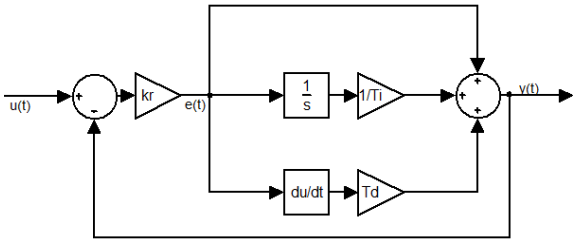
gdzie T_i jest czasem zdwojenia (całkowania, po którym przy działaniu członu proporcjonalnego i całkującego od podania sygnału skokowego wyjście ulega zdwojeniu). T_d jest czasem wyprzedzenia (różniczkowania, po którym przy działaniu członu proporcjonalnego i różniczkującego w czasie liniowo narastającego sygnału wyjściowego ulega on zdwojeniu). k_r jest współczynnikiem skalującym wzmocnienie regulatora. W dziedzinie liczb zespolonych po zastosowaniu transformacji Laplace'a równanie (4.3) przyjmuje postać

$$u(s) = k_r(1 + \frac{1}{T_i} + sT_d)e(s). \tag{4.4}$$

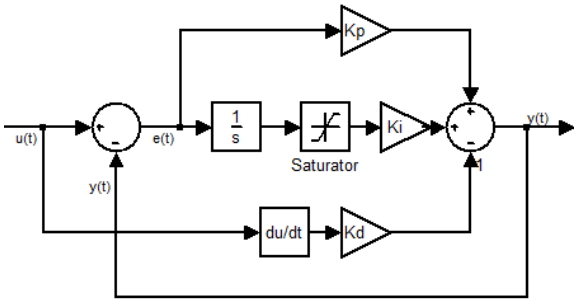
Jak łatwo zauważyć w celu przeliczania nastaw do postaci (4.2) należy wykorzystać wzory

$$\begin{aligned} K_p &= k_r \\ K_i &= \frac{k_r}{T_i} \\ K_d &= k_r T_d. \end{aligned} \tag{4.5}$$

Postać PID–ISA, często stosowana w przemyśle [2], przedstawiono na rysunku 4.3.



Rysunek 4.3. Schemat ideowy regulatora PID–ISA



Rysunek 4.4. Regulator PID z saturatorem

Pomimo szerokiej popularności i wielu metod doboru nastaw regulator PID nie zawsze działa zadowalająco. Opracowano szereg usprawnienień przeciwdziałającym najczęściej występującym niepożądanym zjawiskom. Pierwszym takim występującym zjawiskiem jest *integral windup* [2] (tłumaczone jako nawijanie całkowania). Wszystkie układy rzeczywiste charakteryzują się pewnymi osiągalnymi wartościami granicznymi. Zdarza się, iż kiedy zmienna sterująca osiągnie wartość maksymalną, błąd dalej jest całkowany. Modyfikację temu zapobiegającą implementuje się dodając do członu całkującego blok saturatora. Wyrażenie całkujące osiąga wówczas bardzo dużą wartość, czemu przeciwdziała blok saturatora wprowadzający nasycenie.

Kolejna modyfikacja wynika z faktu, że w czasie działania algorytmu regulacji zwykłego PID mogą wystąpić gwałtowne zmiany zmiennych (np. kiedy zmienne zadane są w sposób skokowy). By zminimalizować to zjawisko częstokroć w miejsce pochodnej błędu do regulatora podawana jest pochodna wyjścia obiektu. Konstrukcję regulatora PID, uwzględniającą oba usprawnienia, przedstawiono na rysunku 4.4.

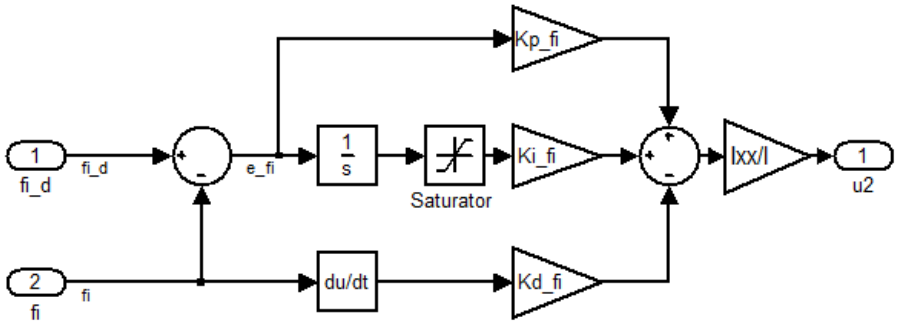
Zdarza się, iż w praktyce stosowane są regulatory PID nie posiadające wszystkich występujących w nich członów. I tak regulator *PI* jest regulatorem, w którym całkowicie pominięto blok różniczkujący, dzięki czemu system jest mniej wrażliwy na szybkie, rzeczywiste zmienne stanu (nie będące szumami). Jest bardziej stabilny w przypadku zaszumianych danych. Dzieje się tak, gdyż pochodna jest wrażliwa na wysokie częstotliwości. Jest to szczególnie istotne przy implementacji na rzeczywistym robocie typu quadrotor, który nie jest wyposażony w wiele czujników lub nie stosuje ich fuzji. W przypadku, kiedy dynamika obiektu jest całkowicie lub częściowo nieznana (lub występują duże różnice pomiędzy modelem matematycznym a rzeczywistym obiektem) stosuje się regulator *PD*. Obecność członu całkującego nie daje w takim przypadku wymiernej poprawy sterowania. Wadą takiego rozwiązania jest fakt, że regulator PD daje niezerowy uchyb ustalony.

4.1.1. Implementacja regulatora PID dla quadrotora

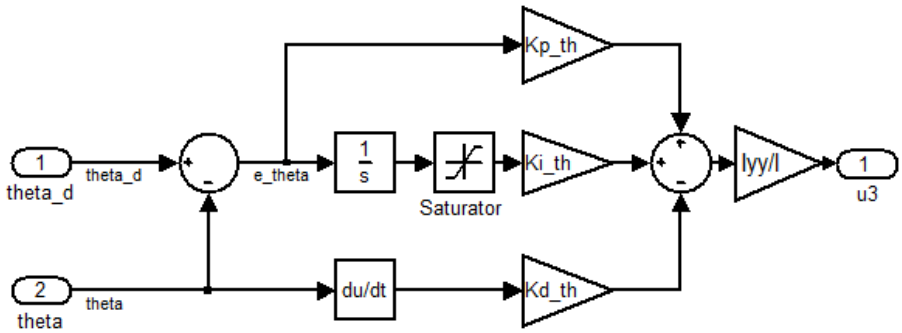
Dysponując w quadrotorze 4 wejściami sterującymi wybieramy 4 składowe wektora stanu, których przebieg będziemy kontrolować za pomocą regulatora PID. Będą to kąty ϕ, θ i ψ oraz położenie z [5,8,10,11]. W celu doboru struktury i nastaw tych regulatorów zauważmy, że równanie dynamiki (2.32) może zostać uproszczone po założeniu małych wartości kątów myszkowania ψ i kołysania θ do postaci

$$\begin{cases} \ddot{x} = 0 \\ \ddot{y} = 0 \\ \ddot{z} = -g + (c_\phi c_\theta)\frac{U_1}{m} \\ \ddot{\phi} = \frac{lU_2}{I_X} \\ \ddot{\theta} = \frac{lU_3}{I_Y} \\ \ddot{\phi} = \frac{U_4}{I_Z}. \end{cases} \tag{4.6}$$

Jak widać dla współrzędnych kątowych uzyskano w ten sposób liniowy model dynamiki. Przy założeniu małych wartości kątów ϕ i θ podobnie możemy spojrzeć na równanie dla składowej z (ze stałym pochodzącym od grawitacji wymuszeniem g, które w procesie sterowania należy kompensować). Poniżej zostanie omówiony sposób implementacji regulatora dla poszczególnych współrzędnych. Strukturę sterowników dla kątów kiwania i kołysania pokazano na rysunkach 4.5 i 4.6. Są to typowe regulatory PID z nasyceniem w części całkującej, z dodatkowym czło-

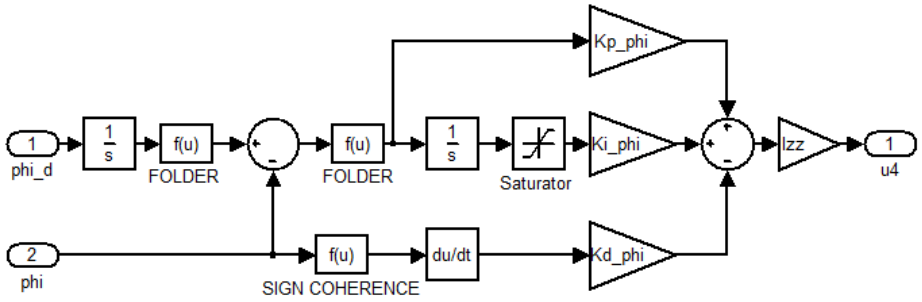


Rysunek 4.5. Sterownik kąta ϕ



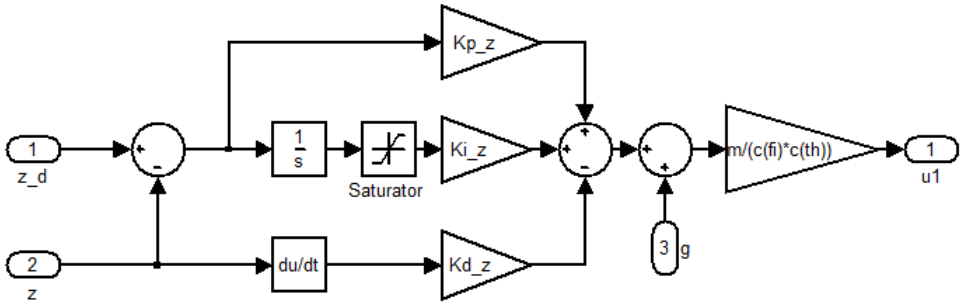
Rysunek 4.6. Sterownik kąta θ

nem wzmacniającym o współczynniku wzmocnienia wyznaczonym na podstawie modelu (4.6). $K_{P\phi}[s^{-2}]$, $K_{I\phi}[s^{-3}]$, i $K_{D\phi}[s^{-1}]$ są parametrami sterownika kąta ϕ, zaś $K_{P\theta}[s^{-2}]$, $K_{I\theta}[s^{-3}]$, i $K_{D\theta}[s^{-1}]$ są parametrami sterownika kąta θ. Dla kąta myszkowania struktura sterownika jest analogiczna z tą różnicą, że tutaj ze względu na nieciągłość reprezentacji kąta myszkowania (przy $\pm\pi$), pojawiła się potrzeba zastosowania dodatkowych bloków FOLDER i SIGN COHERENCE. Pozwalają one na zachowanie ciągłości kąta ψ. (zobacz rysunek 4.7).
Sterownik współrzędnej z posiada na wyjściu wynikający z modelu (4.6) współczynnik skalujący



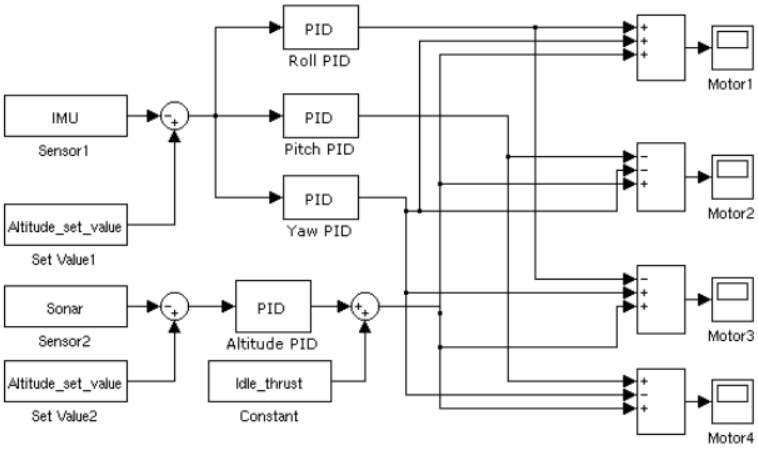
Rysunek 4.7. Sterownik kąta myszkowania

oraz człon służący do kompensacji grawitacji. Został przedstawiony na rysunku 4.8.



Rysunek 4.8. Sterownik współrzędnej z

Zestawiając w jedną całość opisane sterowniki otrzymujemy kompletny układ sterowania quadrotorem o strukturze pokazanej na rysunku 4.9.



Rysunek 4.9. Przykład implementacji algorytmu (za: [10])

Algorytm sterowania otrzymuje jako wejścia dane o wartościach aktualnych zmiennych stanu, które pochodzą z sensorów - inercyjnej jednostki pomiarowej *IMU*, mierzącej aktualne wartości kątów ϕ, θ i ψ oraz danych z sonaru mierzących wartość współrzędnej z. Jako *Altitude_set_value* przyjmujemy wartości zadane dla tych zmiennych. Rysunek przedstawia sterowanie za pomocą prędkościami kątowymi Ω, które pochodzi bezpośrednio ze wzoru (2.31). W implementacji na fizycznym obiekcie można uprościć dynamikę quadrotora do postaci z równania (4.6) - praca [8] potwierdza pożądane działanie regulatora PID w zadaniu śledzenia trajektorii przez modelowany w ten sposób quadrotor.

4.2. Algorytm całkowania wstecznego

W niniejszym podrozdziale omówiony zostanie algorytm całkowania wstecznego[1] potocznie nazywany algorytmem *backstepingu* (od pierwotnej nazwy angielskiej *integral backstepping*) w zastosowaniu do sterowania quadrotorem. W kolejnych podrozdziałach czytelnik znajdzie:
— omówienie metody całkowania wstecznego,
— przegląd istniejących prac na temat sterowania quadrotorem przy użyciu algorytmu backsteppingu;
— opis sterownika wykorzystującego ideę całkowania wstecznego.

4.2.1. Przegląd literatury

Zagadnienie sterowania quadrotorem za pomocą algorytmu wstecznego całkowania było wielokrotnie podejmowane w literaturze fachowej. Jako jedni z pierwszych tematykę tę podjęli S. Bouabdallah oraz R. Siegwart. W pracy [6] wyprowadzają oni model dynamiki quadrotora i podają jego równania (por. rozdział 6.2). Następnie przedstawiają dynamikę w przestrzeni stanu i, w jasny dla czytelnika sposób, przekształcają ją do postaci łańcuchowej[2] (por. rozdział 4.2.3). Zdefiniowanie funkcji błędów dla kolejnych podsystemów sterownika pozwala na czytelną konstrukcję kolejnych funkcji Lapunowa, a w efekcie prowadzi do otrzymania dwustopniowego algorytmu sterowania quadrotorem. Praca ta stała się swoistą inspiracją dla kolejnych naukowców – jest cytowana w niemalże każdym kolejnym artykule dotyczącym omawianego zagadnienia.

Rok po opublikowaniu pracy Bouabdallaha i Siegwarta pojawiły się prace naukowców z Francji – Madiniego oraz Benallequeá [18, 19]. Przedstawione w nich zostało inne podejście do zagadnienia dynamiki, m.in. położono większy nacisk na opisanie sił składających się na dynamikę quadrotora, takich jak opory powietrza czy efekt żyroskopowy. Zaproponowany algorytm znacznie odbiegał od typowego schematu całkowania wstecznego, warto natomiast zwrócić uwagę na fakt, iż dokonano jego implementacji na laboratoryjnym modelu quadrotora.

Prace [14, 24] również traktują o zagadnieniu sterowania obiektem typu quadrotor, niemniej algorytm całkowania wstecznego jest omówiony pobieżnie i przyjmuje postać podobną do tej, jaką zaproponowali Bouabdallah i Siegwart. Autorzy podanych prac skupiają się w nich na dwóch innych algorytmach, odpowiednio na sterowaniu ślizgowym oraz algorytmie H_∞.

Spośród omówionych prac za najlepszą podstawę do opracowania i zamodelowania sterownika uznano tą Bouabdallaha i Siegwarta [6]. Zdecydowano się na takie podejście ponieważ:
— przedstawiona konstrukcja algorytmu całkowania wstecznego jest *najbliższa* duchowi prac Kokotovića,
— jest to podstawowa praca, do której odwołują się kolejni badacze,
— algorytm opisany jest czytelnie i nie ma w nim żadnych niejasności.

4.2.2. Schemat algorytmu

Algorytm całkowania wstecznego zaproponowany został w 1990 przez Petara V. Kokotovića [16, 17]. Pozwala on na stabilizację układów o postaci kaskadowej i znalazł zastosowanie przy realizacji zadania sterowania różnorodnych obiektów.

Rozważamy następujący układ kaskadowy [21]

$$\begin{cases} \dot{x}_1 = f_1(x_1) + g_1(x_1)x_2, \\ \dot{x}_2 = f_2(x_1, x_2) + g_2(x_1, x_2)x_3, \\ \vdots \\ \dot{x}_n = f_n(x_1, \ldots, x_n) + g_n(x_1, \ldots, x_n)u. \end{cases} \tag{4.7}$$

[1] Polska nazwa *całkowanie wsteczne* zaproponowana została przez prof. K. Tchonia w [26].
[2] zwanej również postacią kaskadową; nazwy te będą używane zamiennie

W takim układzie funkcje f_i i g_i w i-tym podsystemie zależą tylko od tych zmiennych x_j, dla których $j \leqslant i$, zaś zmienna x_{i+1} pełni rolę sterowania. W ostatnim podsystemie, zamiast kolejnej zmiennej występuje rzeczywiste sterowanie systemu.

Projektowanie sterownika odbywa się w sposób rekurencyjny. Zaczynamy od stabilizacji podsystemu

$$\dot{x}_1 = f_1(x_1) + g_1(x_1)x_2. \tag{4.8}$$

Definiujemy dla niego odpowiednią funkcję Lapunowa zależną tylko od zmiennej x_1. Zwykle przyjmuje ona formę kwadratową

$$V_1(x_1) = \frac{1}{2} z_1^T z_1, \tag{4.9}$$

gdzie $z_1 = x_1$ lub $z_1 = x_1 - x_{1d}$ (x_{1d} – trajektoria zadana). Funkcja ta musi spełniać dwa warunki:

$$\begin{array}{rcl} V_1(x_1) = 0 & \Leftrightarrow & x_1 = 0, \\ V_1(x_1) > 0 & \Leftrightarrow & x_1 \neq 0. \end{array} \tag{4.10}$$

Kolejno wyliczamy $\dot{V}_1(x_1)$ wzdłuż trajektorii układu. Na mocy zasady niezmienniczości LaSalle'a, układ jest stabilny, jeśli spełniona jest nierówność

$$\dot{V}_1(x_1) \leqslant 0. \tag{4.11}$$

Z nierówności (4.11) otrzymujemy wyrażenie opisujące x_2, dane z dokładnością do pewnego parametru, zwykle oznaczanego jako α_1.

W kolejnym kroku algorytmu, podsystem składa się z dwóch równań

$$\begin{cases} \dot{x}_1 = f_1(x_1) + g_1(x_1)x_2, \\ \dot{x}_2 = f_2(x_1, x_2) + g_2(x_1, x_2)x_3. \end{cases} \tag{4.12}$$

Ponownie definiujemy funkcję Lapunowa, pamiętając, że należy zachować wyniki poprzedniego kroku. I tak V_2 przyjmuje postać

$$V_2(x_1, x_2) = V_1(x_1) + \frac{1}{2} z_2^T z_2, \tag{4.13}$$

gdzie $z_2 = x_2 - \alpha_1 x_1$ lub $z_2 = x_2 - x_{2d} - \alpha_1(x_1 - x_{1d})$. Wykorzystując nierówność

$$\dot{V}_2(x_1, x_2) \leqslant 0. \tag{4.14}$$

uzyskujemy postać x_3, ponownie daną z dokładnością do pewnego parametru, oznaczanego zwykle jako α_2.

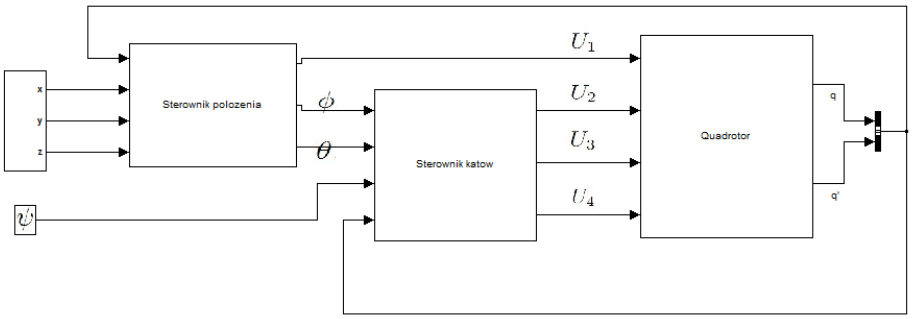
Postępując analagicznie dla układów obejmujących coraz więcej podsystemów dochodzimy do momentu, w którym uzyskujemy postać sterowania u.

4.2.3. Sterownik

Jak przyjęto w rozdziale 1.2 stan quadrototra określa sześciowymiarowy wektor $q = (x, y, z, \phi, \theta, \psi)$.

Dostępne są jednak tylko cztery sterowania, obiekt jest więc niedosterowany. Nie jest więc możliwe śledzenie trajektorii dla więcej niż czterech składowych wektora q. Logicznym rozwiązaniem jest zadawanie położenia w przestrzeni (x, y, z) oraz orientacji w płaszczyźnie XY – ψ. Kąty kołysania i kiwania przyjmują wartości zależne od pozostałych współrzędnych.

Schemat sterownika implementującego algorytm całkowania wstecznego pokazuje rysunek 4.10. Sterownik podzielony jest na dwa moduły – pierwszy wylicza kąty θ i ϕ konieczne do uzyskania



Rysunek 4.10. Schemat sterownika.

zadanej pozycji (x, y) oraz łączną siłę ciągu silników potrzebną do uzyskania zadanej wysokości z. Drugi wylicza pozostałe sterowania pozwalające na uzyskanie wymaganych wartości kątów.

Do wyznaczenia sterowań oba moduły sterownika wykorzystują algorytm całkowania wstecznego. W celu sprowadzenia układu quadrotora danego równaniem (2.32) do przestrzeni stanu stosuje się podstawienie wektora $x = (x_1, x_2, ..., x_{12})^T$, gdzie:

$$\begin{cases} x_1 = \phi, \\ x_2 = \dot{x}_1 = \dot{\phi}, \\ x_3 = \theta, \\ x_4 = \dot{x}_3 = \dot{\theta}, \\ x_5 = \psi, , \\ x_6 = \dot{x}_5 = \dot{\psi}, \\ x_7 = x, \\ x_8 = \dot{x}_7 = \dot{x}, \\ x_9 = y, \\ x_{10} = \dot{x}_9 = \dot{y}, \\ x_{11} = z, \\ x_{12} = \dot{x}_{11} = \dot{z}. \end{cases} \tag{4.15}$$

Z (4.15) i (2.32) otrzymujemy:

$$\begin{cases} \dot{x}_1 = x_2, \\ \dot{x}_2 = x_4 x_6 a_1 + x_4 a_2 \Omega + b_1 U_2, \\ \dot{x}_3 = x_4, \\ \dot{x}_4 = x_2 x_6 a_3 + x_2 a_4 \Omega + b_2 U_3, \\ \dot{x}_5 = x_6, \\ \dot{x}_6 = x_4 x_2 a_5 + b_3 U_4, \\ \dot{x}_7 = x_8, \\ \dot{x}_8 = -g + (\cos x_1 \cos x_3) \frac{1}{m} U_1, \\ \dot{x}_9 = u_x \frac{1}{m} U_1, \\ \dot{x}_{10} = u_x \frac{1}{m} U_1, \\ \dot{x}_{11} = x_{12}, \\ \dot{x}_{12} = u_y \frac{1}{m} U_1, \end{cases} \tag{4.16}$$

gdzie: $a_1 = \dfrac{I_y - I_z}{I_x}$, $a_2 = \dfrac{J_R}{I_x}$, $a_3 = \dfrac{I_z - I_x}{I_y}$, $a_4 = \dfrac{J_R}{I_y}$, $a_5 = \dfrac{I_x - I_y}{I_z}$, $b_1 = \dfrac{l}{I_x}$, $b_2 = \dfrac{l}{I_y}$, $b_3 = \dfrac{l}{I_z}$,

oraz

$$u_x = \cos x_1 \sin x_3 \cos x_5 + \sin x_1 \sin x_5, \tag{4.17}$$

$$u_x = \cos x_1 \sin x_3 \sin x_5 - \sin x_1 \cos x_5. \tag{4.18}$$

Otrzymany układ ma postać łańcuchową – a ściślej można go potraktować jako sześć łańcuchowych podukładów o zmiennych x_i i x_{i+1}.

Sterowanie kątami natarcia

Sterowanie U_2 można wyliczyć wykorzystując pierwszy ze wspomnianych podukładów opisanych równaniami

$$\dot{x}_1 = x_2, \tag{4.19}$$
$$\dot{x}_2 = x_4 x_6 a_1 + x_4 a_2 \Omega + b_1 U_2. \tag{4.20}$$

Dla zadanej trajektorii x_{d1}, błąd śledzenia e_{b1} zdefiniowany jest jako

$$e_{b1} = x_{d1} - x_1. \tag{4.21}$$

W celu ustabilizowania (4.19) zdefiniujmy następującą funkcję Lapunowa (czytelnik powiniem zwrócić uwagę, że tutaj wszystkie współrzędne są skalarne)

$$V_1(x_1) = \frac{1}{2} e_{b1}^T e_{b1} = \frac{1}{2} e_{b1}^2. \tag{4.22}$$

Funkcja ta spełnia warunek nieujemności (4.10). Jej pochodna wyliczona wzdłuż trajektorii układu wynosi

$$\dot{V}_1(e_{b1}) = e_{b1}\dot{e}_{b1} = e_{b1}(\dot{x}_{d1} - \dot{x}_1) = e_{b1}(\dot{x}_{d1} - x_2). \tag{4.23}$$

Spełnienie warunku na niedodatnią pochodną (4.11) zapewnia

$$x_2 = \dot{x}_{d1} + \alpha_1 e_{b1}, \tag{4.24}$$

przy parametrze $\alpha_1 > 0$.

Dla drugiego stopnia kaskady (4.20) błąd ma postać

$$e_{b2} = x_2 - \dot{x}_{d1} - \alpha_2 e_{b1}. \tag{4.25}$$

Zdefiniujmy funkcję Lapunowa spełniającą (4.10)

$$V_2(x_1, x_2) = V_1(x_1) + \frac{1}{2} e_{b2}^2 = \frac{1}{2}(e_{b1}^2 + e_{b2}^2), \tag{4.26}$$

i wyliczmy jej pochodną wzdłuż trajektorii układu

$$\begin{aligned} &\dot{V}_2(e_{b1}, e_{b2}) = \\ &e_{b2}(x_4 x_6 a_1 + x_4 a_2 \Omega + b_1 U_2) - e_{b2}(\ddot{x}_{1d} - \alpha_1(e_{b2} + \alpha_1 e_{b1})) - e_{b1} e_{b2} - \alpha_1 e_{b1}^2 = \\ &\qquad e_{b2}(b_1 U_2 + x_4 x_6 a_1 + x_4 a_2 \Omega - \ddot{x}_{1d} + \alpha_1(e_{b2} + \alpha_1 e_{b1}) - e_{b1}) - \alpha_1 e_{b1}^2. \end{aligned} \tag{4.27}$$

Spełnienie warunku (4.14) w punkcie równowagi ($\ddot{x}_1 = 0$) zapewnia sterowanie

$$U_2 = \frac{1}{b_1}(e_{b1} - x_4 x_6 a_1 - x_4 a_2 \Omega - \alpha_1(e_{b2} + \alpha_1 e_{b1}) - \alpha_2 e_{b2}), \tag{4.28}$$

gdzie $\alpha_2 > 0$, zaś człon $\alpha_2 e_{b2}$, przeciwdziała zmianom e_{b1}.

W analogiczny sposób można wyznaczyć sterowania U_3 i U_4 uzyskując

$$\begin{aligned} U_3 &= \tfrac{1}{b_2}(e_{b3} - x_2 x_6 a_3 - x_2 a_4 \Omega - \alpha_3(e_{b3} + \alpha_3 e_{b3}) - \alpha_4 e_{b4})), \\ U_4 &= \tfrac{1}{b_3}(e_{b5} - x_2 x_4 a_5 - \alpha_5(e_{b6} + \alpha_5 e_{b5}) - \alpha_6 e_{b6}), \end{aligned} \tag{4.29}$$

gdzie

$$\begin{aligned} e_{b3} &= x_{d3} - x_3, \\ e_{b4} &= x_4 - \dot{x}_{d3} - \alpha x_3 e_{b3}, \\ e_{b5} &= x_{d5} - x_5, \\ e_{b6} &= x_6 - \dot{x}_{d5} - \alpha x_5 e_{b5}. \end{aligned} \tag{4.30}$$

Sterowanie położeniem

Wyznaczenie U_1 odbywa się identycznie jak dla pozostałych sterowań. Przy błędach

$$\begin{aligned} e_{b7} &= x_{d7} - x_7, \\ e_{b8} &= x_8 - \dot{x}_{d7} - \alpha_7 e_{b7}, \end{aligned} \tag{4.31}$$

przyjmuje ono postać

$$U_1 = \frac{m}{\cos x_1 \cos x_3}(e_{b7} + g - \alpha_7(e_{b8} + \alpha_7 e_{b7}) - \alpha_8 e_{b8}). \tag{4.32}$$

Przemieszczenie quadrotora w płaszczyźnie XY związane jest bezpośrednio z kątami kiwania i kołysania. Znając sterowanie U_1, dwa ostatnie podukłady stabilizujemy podobnie jak poprzednie, traktując jako sterowanie odpowiednio u_x i u_y. Otrzymujemy

$$\begin{aligned} u_x &= \tfrac{m}{U_1}(e_{b9} - \alpha_9(e_{b10} + \alpha_9 e_{b9}) - \alpha_{10} e_{b10}), \\ u_y &= \tfrac{m}{U_1}(e_{b11} - \alpha_{11}(e_{b12} + \alpha_{11} e_{b11}) - \alpha_{12} e_{b12}), \end{aligned} \tag{4.33}$$

przy

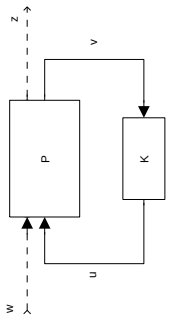
$$\begin{aligned} e_{b9} &= x_{d9} - x_9, \\ e_{b10} &= x_{10} - \dot{x}_{d9} - \alpha_9 e_{b9}, \\ e_{b11} &= x_{d11} - x_{11}, \\ e_{b12} &= x_{12} - \dot{x}_{d11} - \alpha_1 1 e_{b11}. \end{aligned} \tag{4.34}$$

Kąty θ i ϕ wyznaczyć można przyrównując uzyskane wyniki do zależności (4.17) i (4.18).

Ostatecznie uzyskany został układ sterowania o dwunastu regulowanych parametrach α_1 – α_{12}, pozwalający na śledzenie trajektorii w przestrzeni zadaniowej określonej położeniem i kątem myszkowania quadrotora.

4.3. Algorytm H_∞

Rysunek 4.11 przedstawia podstawowy układ sterowania [25] zbudowany z regulatora P oraz obiektu K.



Rysunek 4.11. Podstawowy układ sterowania

Równania opisujące układ można zapisać następująco:

$$\begin{bmatrix} z \\ v \end{bmatrix} = P(s) \begin{bmatrix} w \\ u \end{bmatrix} = \begin{bmatrix} P_{11}(s) & P_{12}(s) \\ P_{21}(s) & P_{22}(s) \end{bmatrix} \begin{bmatrix} w \\ u \end{bmatrix} \tag{4.35}$$
$$u = K(s)v.$$

Równanie zamkniętej pętli sprzężenia zwrotnego dla powyższego układu sterowania z w do z opisuje transformacja:

$$z = F_t(P, K)w \tag{4.36}$$

gdzie

$$F_t(P, K) = P_{11} + P_{12}K(I - P_{22}K)^{-1}P_{21} \tag{4.37}$$

Algorytm H_∞ ma za zadanie minimalizować wartość normy $F_t(P, K)$.

Algorytm sterowania H_∞ bazuje na rozwiązywaniu równań stanu przedstawionych w 1988 roku przez Glovera i Doyle'a. Zakładając reprezentację P równaniami stanu postaci

$$P = \begin{bmatrix} A & B_1 & B_2 \\ C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{bmatrix} \tag{4.38}$$

można określić najczęściej występujące założenia odnośnie stosowania algorytmu H_∞:

(I) A, B_2, C_2 są stabilne i wykrywalne,
(II) D_{12} i D_{21} są pełnego rzędu,
(III) $\begin{bmatrix} A - j\omega I & B_2 \\ C_1 & D_{12} \end{bmatrix}$ jest pełnego rzędu dla wszystkich ω,
(IV) $\begin{bmatrix} A - j\omega I & B_1 \\ C_2 & D_{21} \end{bmatrix}$ jest pełnego rzędu dla wszystkich ω,
(V) $D_{11} = 0$ i $D_{22} = 0$.

Założenie (I) wymaga istnienia sterownika K stabilizującego układ. Drugie założenie wystarcza, aby zapewnić istnienie takiego sterownika. Założenia (III) i (IV) wystarczają aby zapewnić, by układ nie został wytrącony z równowagi i jego pierwiastki nie wpadały w obszar niestabilny. Założenie (V) nie jest wymagane dla algorytmu H_∞, stosuje się go raczej w algorytmie H_2, pozwala jednak na znaczne uproszczenie formuł algorytmu. Dodatkowo dla uproszczeń często zakłada się (VI), iż $D_{12} = \begin{bmatrix} 0 \\ I \end{bmatrix}$ oraz $D_{21} = \begin{bmatrix} 0 & I \end{bmatrix}$.

4.3.1. Sterowanie układu algorytmem H_∞

Standardowy problem sterowania algorytmem H_∞ dla układu przedstawionego na rysunku 4.11 sprowadza się do znalezienia wszystkich sterowników K, które minimalizują

$$||F_t(P,K)||_\infty = \max_\omega \bar{\delta}(F_t(P,K)(j\omega)) \tag{4.39}$$

W praktyce zazwyczaj nie jest wymagane uzyskanie optymalnego sterownika dla problemu H_∞. Za wystarczająco dobry sterownik uznaje się taki, który spełnia nierówność:

$$||F_t(P,K)||_\infty < \gamma \tag{4.40}$$

gdzie γ_{min} jest minimalną wartością $||F_t(P,K)||_\infty$ spośród wszystkich sterowników K oraz $\gamma > \gamma_{min}$. Tak postawiony problem sterowania można rozwiązać stosując algorytm przedstawiony przez Doyle'a w 1989 roku.

Dla układu 4.11 przy założeniach (I)-(VI) sterownik spełniający nierówność (4.40) istnieje wtedy i tylko wtedy gdy:

(i) $X_\infty \geqslant 0$ jest rozwiązaniem algebraicznego równania Riccatiego
$$A^T X_\infty + X_\infty A + C_1^T C_1 + X_\infty(\gamma^{-2} B_1 B_1^T - B_2 B_2^T) X_\infty = 0 \tag{4.41}$$
takim, że $Re\lambda_i[A + (\gamma^{-2} B_1 B_1^T - B_2 B_2^T) X_\infty] < 0,\ \forall i$

(ii) $Y_\infty \geqslant 0$ jest rozwiązaniem algebraicznego równania Riccatiego
$$AY_\infty + X_\infty A^T + B_1 B_1^T + Y_\infty(\gamma^{-2} C_1^T C_1 - C_2^T C_2) Y_\infty = 0 \tag{4.42}$$
takim, że $Re\lambda_i[A + Y_\infty(\gamma^{-2} C_1^T C_1 - C_2^T C_2)] < 0,\ \forall i$

(iii) $\rho(X_\infty Y_\infty) < \gamma^2$

Wszystkie sterowniki spełniające powyższe warunki dane są przez $K = F_t(K_c, Q)$, gdzie:

$$\begin{aligned} K_c(s) &= \begin{bmatrix} A_\infty & -Z_\infty L_\infty & Z_\infty B_2 \\ F_\infty & 0 & I \\ -C_2 & I & 0 \end{bmatrix} \\ F_\infty &= -B_2^T X_\infty, \qquad L_\infty = -Y_\infty C_2^T, Z_\infty = (I - \gamma^{-2} Y_\infty X_\infty)^{-1} \\ A_\infty &= A + \gamma^{-2} B_1 B_1^T X_\infty + B_2 F_\infty + Z_\infty L_\infty C_2 \end{aligned} \tag{4.43}$$

oraz $Q(s)$ jest dowolną stabilną funkcją przenoszenia spełniającą nierówność $||Q||_\infty < \gamma$. Dla $Q(s) = 0$ mamy:

$$K(s) = K_{c_{11}}(s) = -Z_\infty L_\infty (sI - A_\infty)^{-1} F_\infty \tag{4.44}$$

$K(s)$ nazywane jest 'głównym' sterownikiem i ma taki sam wymiar co $P(s)$. Sterownik ten może zostać podzielony na część odpowiedzialną za estymację stanu:

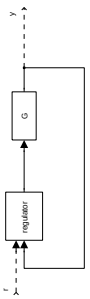
$$\dot{\hat{x}} = A\hat{x} + B_1 \gamma^{-2} B_1^T X_\infty \hat{x} + B_2 u + Z_\infty L_\infty (C_2 \hat{x} - y) \tag{4.45}$$

oraz sprzężenie zwrotne:

$$u = F_\infty \hat{x} \tag{4.46}$$

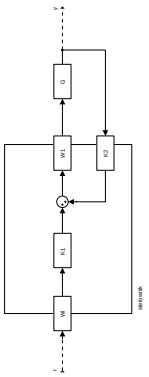
Wiele układów wymagających sterowania określa się jako układy 2DOF czyli układy o dwóch stopniach swobody, gdzie pierwszy dotyczy pomiaru sygnału uzyskanego ze sprzężenia zwrotnego, a drugi podążania za wartością referencyjną.

Podstawowy układ tego typu został przedstawiony na rysunku 4.12 (r – sygnał referencyjny, G – obiekt, y – wyjście).



Rysunek 4.12. Podstawowy układ 2DOF

W przypadku algorytmu kształtowania pętli H_∞ układ z rysunku 4.12 można przekształcić do postaci z rysunku 4.13.



Rysunek 4.13. Sterowanie kształtowaniem pętli H_∞ układu 2DOF

Kształtowanie pętli H_∞ łączy klasyczne metody teorii sterowania ze sterowaniem optymalnym zapewniając odporność stabilności systemu. Niniejsze rozwiązanie zostało zaprezentowane w 1990 roku przez McFarlane'a i Glovera.

Kształtowanie pętli H_∞ można zasadniczo podzielić na dwa etapy. Najpierw w otwartej pętli sygnały z obiektu sterowania zwiększone są poprzez pre- i post-kompensatory, aby uzyskać pożądany kształt odpowiedzi częstotliwościowej otwartej pętli z dokładnością do pojedynczych wartości. Następnie w drugim etapie algorytmu uzyskany 'kształt' jest poddany stabilizacji odpornościowej przy użyciu optymalizacji H_∞.

Kolejne kroki algorytmu aby uzyskać układ z rysunku 4.13 można zapisać jako:

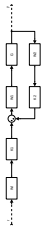
1. Zaprojektuj sterownik kształtowania pętli H_∞ dla układu 1DOF bez pre-kompensatora W_i.
2. Wybierz funkcję przenoszenia T_{ref} między W_1/W_i a kontrolowanymi wyjściami.
3. Ustaw parametr skalowania ρ na małą wartość większą niż 1 (najlepsza byłaby wartość z przedziału [1,3]).

4. Dla $G_s = GW_1$, T_{ref} oraz ρ rozwiąż standardowy problem optymalizacji H_∞. Wyznacz $K = \begin{bmatrix} K_1 & K_2 \end{bmatrix}$.
5. Zastąp prefiltr K_1 przez $K_1 W_i$ aby uzyskać dokładny model w stanie ustalonym.
6. Przeanalizuj układ i jeśli jest to wymagane popraw ρ, W_1 oraz T_{ref}.

4.3.2. Sterowanie układu quadrotora algorytmem H_∞

W przypadku quadrotorów stosuje się zazwyczaj algorytm kształtowania pętli H_∞.

Struktura podstawowego układu sterowania kształtowaniem pętli H_∞ przedstawiona jest na rysunku 4.14.



Rysunek 4.14. Podstawowy układ sterowania kształtowaniem pętli H_∞

Niech dany będzie nieliniowy model quadrotora UAV.

Za sterowania układu przyjęto równania (2.30), gdzie F_i zgodnie z rysunkiem 1.2. W związku z tym równania dynamiki quadrotora opisane są równaniami (2.32).

Aby móc zastosować sterowanie quadrotora kształtowaniem pętli H_∞ należy obiekt sterowania poddać linearyzacji, ukształtować jego odpowiedź częstotliwościową w otwartej pętli sprzężenia zwrotnego i następnie nałożyć na ukształtowany system pętlę sterowania wysyntezowaną za pomocą optymalizacji H_∞. Zgodnie z [9] zastosowano prekompensator W_1 i postkompensator W_2, które opisują system z otwartą pętlą jako $G_s = W_2 G W_1$. Zastosowanie kompensatorów pozwala dowolnie kształtować odpowiedź częstotliwościową systemu w otwartej pętli sprzężenia zwrotnego. Diagonalny kompensator W_1 zawiera filtry całkujące i proporcjonalne. Filtry proporcjonalne odpowiadają za zredukowanie opóźnienia fazowego systemu w okolicach częstotliwości krzyżowania. Część całkująca W_1 pomaga w filtrowaniu zakłóceń. Postkompensator W_2 składa się z filtru dolnoprzepustowego pierwszego stopnia odrzucającego szumy polepszając odporność układu.

Algorytm postępowania oraz implementację sterownika przy użyciu oprogramowania Matlab/Simulink przedstawiono w sekcji 6.5

Część II

Implementacja

5. Zadania realizowane przez quadrotor

Do zadań sterowania realizowanych przez robot mobilny jakim jest quadrotor należą:
— śledzenie trajektorii,
— śledzenie ścieżki,
— sterowanie do punktu.
W celu analizy algorytmów sterowania, przeprowadzono badania nad pierwszym i ostatnim zaganieniem. Poniżej zostaną omówione te zadania i zaprezentowane krzywe użyte przy śledzeniu trajektorii.

5.1. Śledzenie trajektorii

Zadanie śledzenia trajektorii polega na wykonaniu przez robot mobilny ruchu wzdłuż krzywej z określoną prędkością. Do określenia zadawanego położenia wykorzystuje się krzywe parametryzowane czasem. Zagadnienie zrealizowano przy wykorzystaniu rodzin krzywych zapewniających wysoką kontrole nad generowanym kształtem. Krzywe Lissajous umożliwiają tworzenie torów sinusoidalnych w przestrzeni kartezjańskiej. W szczególności mogą posłużyć do generacji trajektorii kołowej bądź eliptycznej. Krzywa Béziera zapewnia możliwość tworzenia gładkich torów o dowolnym kształcie. Jest funkcją wielomianową i może posłużyć do zbadania ruchu po prostej, czy paraboli. Jako uzupełnienie została zaprezentowana krzywa łańcuchowa, posiadająca kształt optymalny energetycznie.

5.1.1. Krzywe Lissajous

Krzywe Lissajosu są rodziną krzywych realizowanych w przestrzeni kartezjańskiej. Dla współrzędnych x i y są dane równaniem parametrycznym

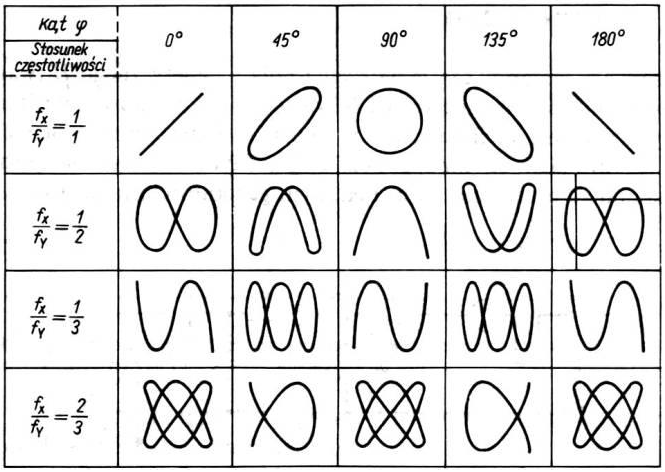
$$\begin{cases} x(t) = A\sin\left(\frac{2\pi f_x t}{T} + \varphi\right) + \Delta_x \\ y(t) = B\sin\frac{2\pi f_y t}{T} + \Delta_y \end{cases}, \tag{5.1}$$

gdzie A i B są amplitudami krzywej,T jej okresem,Δ_x i Δ_y służą wypozycjonowaniu w przestrzeni, natomiast f_x i f_y są częstotliwościami składowych i razem z opóźnieniem fazowym φ określają charakter kształtu. Dla stosunku $\frac{f_x}{f_y} = 1$ otrzymywane są dwa szczególne przypadki, dla $\varphi = \frac{\pi}{2}$ jest to elipsa, zaś przy $\varphi = 0$ odcinek. Gamy kształtów w zależności od tych parametrów przedstawiono na rysunku 5.1. Implementacja trajektorii sprowadza się do przypisania równań (5.1) dwóm współrzędnym przestrzeni kartezjańskiej.

5.1.2. Krzywe Béziera

Krzywe Béziera są wielomianowymi krzywymi parametrycznymi. Dzięki prostej konstrukcji, możliwości odwzorowania dowolnego kształtu oraz wysokiej kontroli nad krzywizną są powszechnie używane m.in. w grafice komputerowej, środowiskach *CAD* czy wzornictwie przemysłowym. Niech $P0$ i $P1$ będą dowolnymi punktami w przestrzeni. Wtedy krzywa Béziera pierwszego stopnia z parametryzacją $t \in [0, 1]$ jest wyrażona wzorem

$$B_{P_0}^{P_1}(t) = (1-t)P_0 + tP_1. \tag{5.2}$$



Rysunek 5.1. Krzywe Lissajous

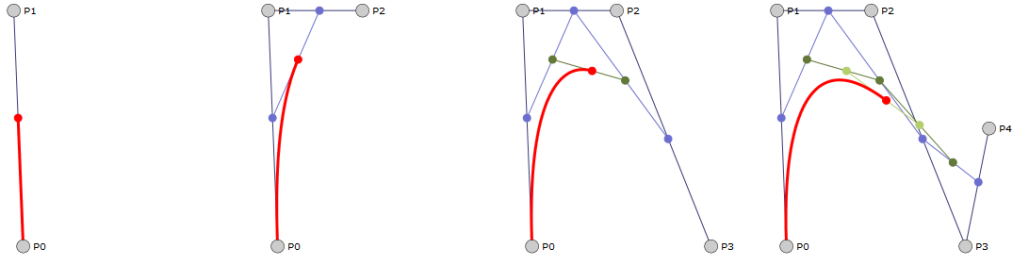
Jak łatwo zauważyć jest to odcinek łączący oba punkty. Dołączmy do naszego zbioru punkt P_2. Krzywa Béziera stopnia drugiego jest analogiczną konstrukcją przy wykorzystaniu krzywych $B_{P_0}^{P_1}(t)$ i $B_{P_1}^{P_2}(t)$.

$$B_{P_0}^{P_2}(t) = (1-t)B_{P_0}^{P_1}(t) + tB_{P_1}^{P_2}(t) \tag{5.3}$$

Uzyskana w powyższy sposób krzywa łączy punkty P_0 i P_2 oraz jest styczna na końcach do odcinków P_0P_1 i P_1P_2. Ostatecznie krzywą stopnia n jest rekurencyjnie określona równaniami

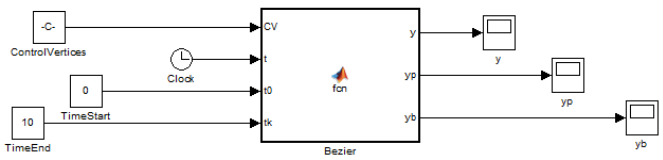
$$\begin{cases} B_{P_i}^{P_{i+1}}(t) = (1-t)P_i + tP_{i+1} \\ B_{P_0}^{P_n}(t) = (1-t)B_{P_0}^{P_{n-1}}(t) + tB_{P_1}^{P_n}(t) \end{cases} . \tag{5.4}$$

Stopień krzywej jest liczbą punktów pomniejszoną o jeden. Proces powstawania krzywej został zaprezentowany na rysunku 5.2 W celu zapewnienia możliwości zadawania trajektorii o swobod-



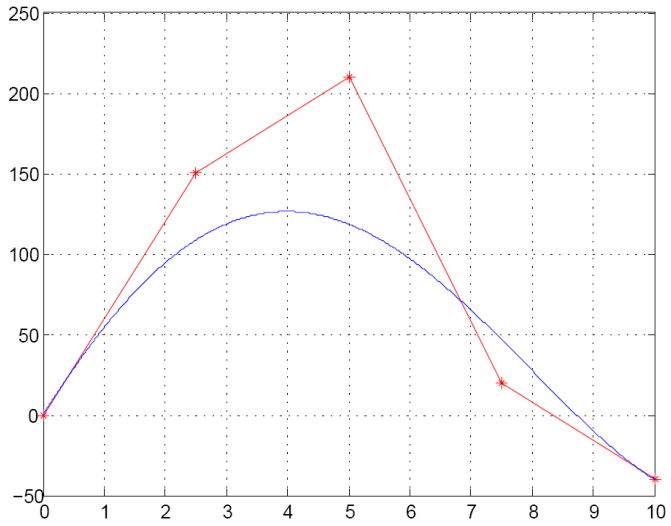
Rysunek 5.2. Ilustracja rekurencji krzywej Béziera

nym kształcie zaimplementowano krzywe Béziera w środowisku *Simulink*. Układ reprezentuje rysunek 5.3. Wejścia i wyjścia układu:



Rysunek 5.3. Implementacja krzywych Béziera w środowisku *Simulink*

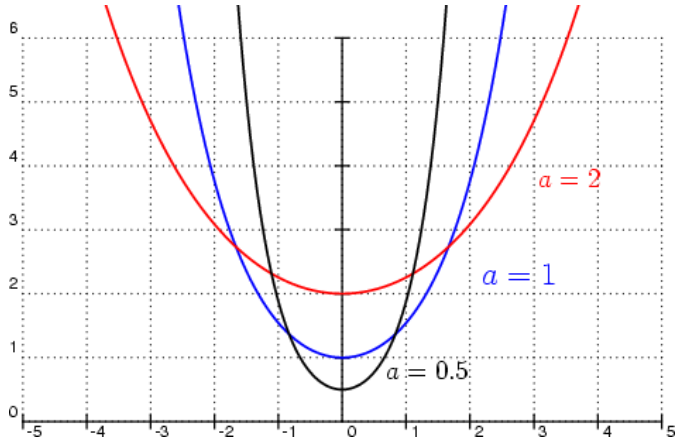
— CV – wektor wierzchołków (punktów) kontrolnych,
— t – zmienną czasu,
— $t0$ – czas startu,
— tk – czas końca,
— y – krzywa Béziera,
— yp – pierwsza pochodna krzywej Béziera,
— yb – druga pochodna krzywej Béziera.
Stopień krzywej jest określany na podstawie długości wektora CV. Przykładową realizacje przedstawia rysunek 5.4.



Rysunek 5.4. Krzywa Béziera zrealizowana w środowisku *Simulink*

5.1.3. Krzywa łańcuchowa

Krzywa łańcuchowa odzwierciedla krzywiznę nierozciągliwej, idealnie wiotkiej liny o niezerowej masie, zawieszonej pomiędzy dwiema podporami. Jej kształt minimalizuje energię potencjalną takiego układu, i zatem może zostać uzyskany z równania Eulera-Lagrange'a (2.3). Została przedstawiona na rysunku 5.5.



Rysunek 5.5. Wykres krzywych łańcuchowych w zależności od parametru a

Krzywa łańcuchowa jest dana wzorem

$$y(x) = a\cosh\left(\frac{x-\Delta_x}{a}\right) + \Delta_y, \tag{5.5}$$

gdzie y oraz x są współrzędnymi kartezjańskimi, Δ_x i Δ_y odpowiadają za umiejscowienie w przestrzeni, zaś a determinuje zarówno wierchołek krzywej, jak i tempo jej otwarcia. Wzór (5.5) można przekształcić do postaci parametrycznej

$$\begin{cases} x(t) = t - t_0 + \Delta_x \\ y(t) = a\cosh\left(\frac{t-t_0}{a}\right) + \Delta_y \end{cases}, \tag{5.6}$$

gdzie t_0 jest przesunięciem parametryzacji t.

5.2. Sterowanie do punktu

Sterowanie do punktu ma na celu osiągniecie i utrzymanie przez robota mobilnego zadanego punktu przestrzeni zadaniowej w badanych współrzędnych. Regulacja może dotyczyć zarówno współrzędnych położenia jak i kątowych (utrzymanie orientacji). Implementacja zadania sprowadza się do zadania trajektorii stałej w czasie. Nie jest wymagane, aby sterowanie do punktu dotyczyło wszystkich sterowalnych współrzędnych. Powszechnie łączy się je z zadaniem śledzenia trajektorii, zadając trajektorie, w której ruch jest dozwolono jedynie dla części składowych. Przykładową stałą trajektorie dla regulatora PID zaprezenowano równaniu (5.7)

$$\begin{cases} z = 2 \\ \phi = -\frac{pi}{6} \\ \theta = \frac{pi}{10} \\ \psi = \frac{pi}{2} \end{cases}. \tag{5.7}$$

Warto zauważyć, iż pomimo w przykładzie zadawane są wartości stałe w czasie, quadrotor będzie się poruszał we współrzędnych x oraz y.

6. Realizacja w środowisku MATLAB

6.1. Interfejs danych

6.1.1. Wymiana danych w modelu

Zadanie symulacji modelu wraz z jego sterowaniem wymaga stworzenia wielu różnych struktur, takich jak model obiektu, sterowniki, wyniki symulacji, interfejs użytkownika. Każda z tych struktur zawiera w sobie parametry, pobiera oraz zwraca dane. Aby uniknąć chaosu związanego z występowaniem tylu różnych parametrów, (konfliktom nazw, dublowaniu parametrów, braku zrozumienia przeznaczenia parametrów), należy przyjąć jednolity sposób agregacji danych.

Wymiana danych następuje w trzech punktach:

1. Model-Algorytm
2. Wizualizacja-Model
3. Wizualizacja-Algorytmy

W przypadku 1, czyli wymiany danych pomiędzy modelem i algorytmami interfejsem danych są połączenia pomiędzy bloczkami modelu i sterownika, dane tam przekazywane zostały oparte o model ustalony pomiędzy grupami projektowymi. Natomiast w dwóch pozostałych przypadkach (2, 3) zastosowano interfejs dobrany i omówiony w kolejnej części raportu.

6.1.2. Dobór interfejsu danych

Agregacja danych w Matlabie może odbywać się na różne sposoby, każdy z nich posiada swoje wady i zalety. W tej części zostały omówione podstawowe metody agregacji danych w Matlabie i został uzasadniony wybór metody zastosowanej w projekcie.

Zmienne w przestrzeni roboczej Matlaba

Podstawowa, najczęściej stosowana metoda. Zmienne zapisujemy bezpośrednio w przestrzeni roboczej, starając się przy tym nadawać im czytelne nazwy. Ten sposób jest dobry przy niewielkich rozmiarach modelu, przy rozbudowanym modelu prowadzi do bałaganu w przestrzeni roboczej – dotarcie do potrzebnej zmiennej nie jest proste a jeszcze trudniejsze jest znalezienie zbioru parametrów określonej struktury ponadto powoduje konflikty nazw.

Zastosowanie struktur

Zastosowanie struktur pozwala na zagregowanie danych w większe bloki co porządkuje przestrzeń roboczą. Trzymanie się szablonu przy implementacji pozwala na łatwy dostęp do przechowywanych tam wartości, i ich modyfikację np. przez GUI. Do wad należy zaliczyć brak kontroli nad zmianami co może prowadzić do błędów pracy oprogramowania[1] oraz brak możliwości zagregowania w ten sposób funkcji i metod.

Programowanie obiektowe z użyciem klas

Przy porządkowaniu interfejsu możliwe jest wykorzystanie klas co umożliwia zastosowanie metod stosowanych przy programowaniu obiektowym: kontrolę dostępu atrybutów (public, pri-

[1] Tę wadę można zmniejszyć przez zastosowanie odpowiednich funkcji i kontrolę wartości - zostało to opisane w dalszej części dokumentu

Tabela 6.1. Struktura parametrów modelu i sterowania

Nazwa	Opis
`val`	Wartość bieżąca
`min_val`	minimalna wartość
`max_val`	maksymalna wartość
`unit_sym`	symbol jednostki SI
`desc`	opis parametru

vate), stosowanie metod. Niestety prócz zalet te rozwiązanie posiada znaczące wady. Implementacja zmienia się wraz z kolejnymi wersjami Matlaba. Tworzenie klas jest również skomplikowane i powoduje problemy przy wykorzystaniu przez inne osoby.

6.1.3. Opis struktur zawartych w projekcie

W projekcie zastosowano struktury, jest to kompromis pomiędzy zaletami stosowania klas a prostotą zapisywania zmiennych bezpośrednio w przestrzeni roboczej. Jak okazało się później dokonany wybór był właściwy - pozwalał na szybkie zapoznanie się z nim pozostałych grup projektowych.

Dane zawarte w modelu zostały zagregowane w strukturach przedstawionych na rys. 6.1. Krótkie streszczenie znaczenia poszczególnych struktur:

— `Simoptions` – *"Simulation Options"* opcje bieżącej symulacji, są one częścią modelu implementowanego w Simulinku [2] składają się na nie m.in. metoda całkowania, krok całkowania, maksymalny błąd.
— `QP` – *"Quadrotor Parameters"* parametry modelu, w tej części znajdują się parametry związane z modelem quadrokoptera np. warunki początkowe, parametry fizyczne, opcje sterowania.
— `CP` – *"Controler Parameters"* parametry kontrolera, obsługiwane i dostarczane przez grupy algorytmiczne.
— `QL` – *"Quality"* wskaźniki jakości służące ocenie algorytmów, nie implementowane na tym etapie prac.
— `traj` – *"trajectory"* zapis trajektorii, służy do zapisywania zadanej/otrzymanej trajektorii
— `simOut` –*"simulation output"* agregacja wszystkich wymienionych powyżej struktur stosowanych/otrzymanych przy pojedyńczej symulacji, oraz zapis przebiegu sterowania

Parametry wchodzące w skład struktur `QP` i `CP` posiadają ustrukturyzowaną formę (tabela 6.1) pozwalającą na ich łatwą obsługę zarówno w module GUI jak i z poziomu skryptów.

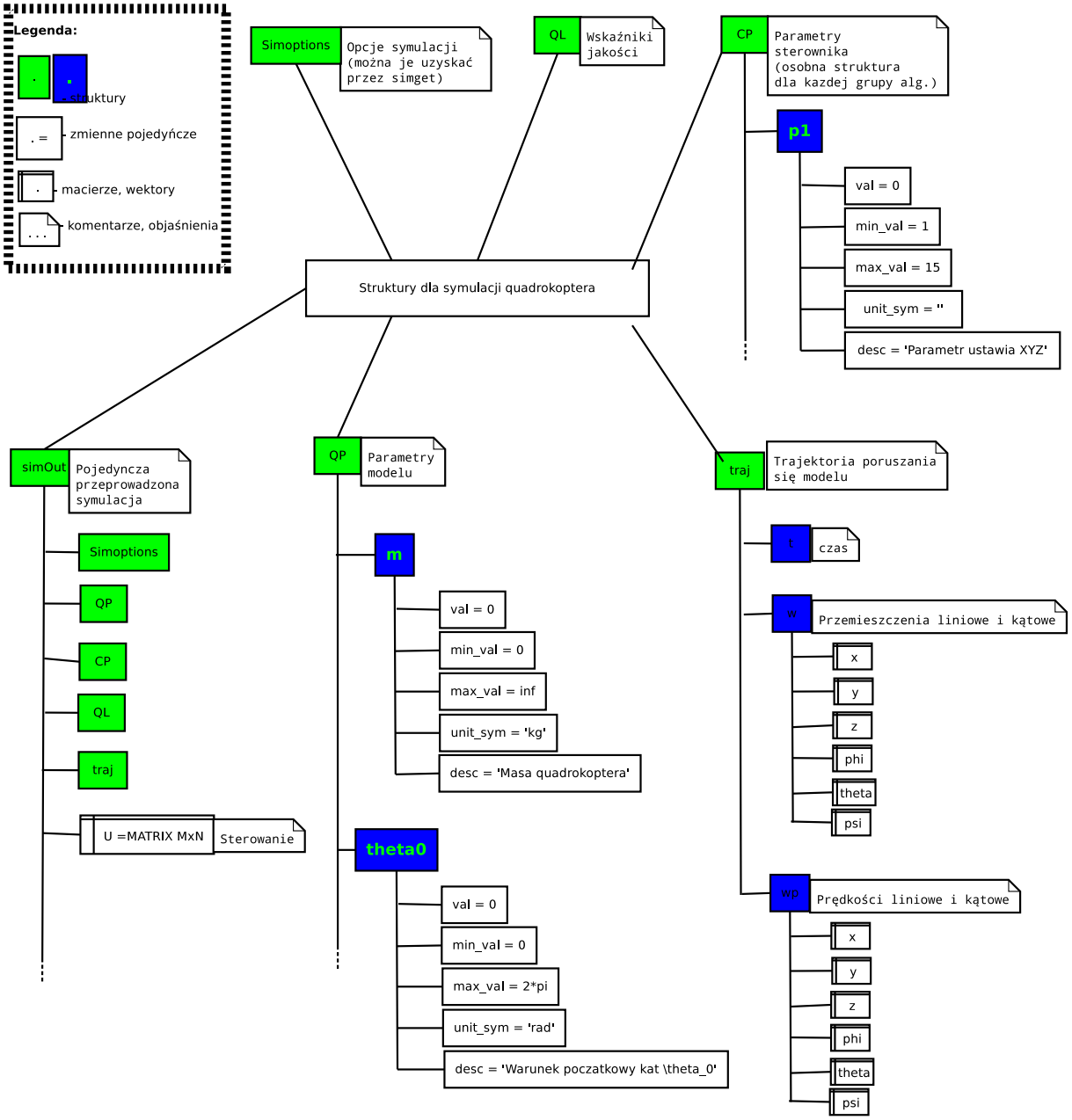
6.1.4. Funkcja pomocnicza generująca struktury

W celu unifikacji i uniknięcia błędów przy implementacji struktur należy korzystać z funkcji pomagającej tworzyć struktur (listing 6.1). Celem tej funkcji jest wstępna walidacja danych. Dane sprawdzane są pod względem poprawności typu (np. tekst, liczba) następnie sprawdzany jest zakres (czy bieżąca wartość znajduje się w zadeklarowanym zakresie). Na końcu zadane wartości przypisywane są do poszczególnych parametrów struktury danych. Funkcja pozwala więc na proste utworzenie parametru – jest to znacząco krótsze niż wypisywanie poszczególnych pól struktury. Jej konstrukcja wyłapuje błędy co pomaga w poprawnym utworzeniu parametru.

Wydruk 6.1. Funkcja wspomagająca tworzenie parametrów dla struktur

```
1 function N = nparam(val, min_val, max_val, unit_sym, desc)

3 VarC = class(val);
```

[2] Te parametry można uzyskać za pomocą komendy `simget('nazwa_pliku_modelu')` i ustawiać za pomocą `simset`



Rysunek 6.1. Struktury

```
if ( ~(strcmp(VarC, 'tf') ...
    || strcmp(VarC, 'ss') ...
    || strcmp(VarC,'logical')) )
    if (min_val > max_val)
        error('New param. error! min_val > max_val');
    end

    if ((val < min_val) || (val > max_val))
      error('New param. error! Current value out of range');
    end
end

if (~ischar(desc))
  error('nparam. error! Description (desc) is not char!');
end

if (~ischar(unit_sym))
  error('nparam. error! Unit symbol (unit_sym) is not char!');
end

N = struct('val',        val,      ... % current vaule
           'min_val',    min_val,  ... % value >= min_val
           'max_val',    max_val,  ... % value <= max_val
           'unit_sym',   unit_sym, ... % SI symbol (m, rad)
           'desc',       desc);        % parameter description

end
```

Podanie parametrów przebiega w następujący sposób (tutaj na skróconym przykładzie 6.2).

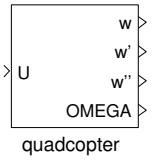
Wydruk 6.2. Przykład zadania parametrów

```
%%% Quadrotor parameters
% Source: "Design and Control of an Indoor Micro Quadrotor"

% nparam(val, min_val, max_val, unit_sym, desc)
QP.m= nparam(5, 0, 1000, 'kg', 'Masa quadrokoptera');
QP.l= nparam(0.25, 0, 1, 'm', ...
             'Dlugosc ramienia quadrokoptera');
QP.g= nparam(9.81, 9.78, 9.85, 'm/s^2', ...
            'Przyspieszenie ziemskie');
% (...)
```

6.2. Model

Model zaimplementowano z wykorzystaniem środowiska **Simulink** będącego częścią pakietu numerycznego **Matlab**. Środowisko to umożliwia szybką i efektywną realizację zadań z zakresu Teorii sterowania. Implementacja modeli i algorytmów sterowania odbywa się z wykorzystaniem intuicyjnych narzędzi graficznych. Efektem prac, oprócz poprawnie działających modeli i sterowników umożliwiających przeprowadzanie badań, jest przejrzysta struktura całego układu sterowania przedstawiająca różne bloki funkcjonalne oraz przepływ informacji.

Efektem realizacji zadania jest blok `quadcopter` przedstawiony na rysunku 6.2. Reprezentuje



Rysunek 6.2. Blok programu **Simulink** z zaimplementowanym modelem

obiekt sterowania z wektorem sterowania na wejściu i zmiennymi stanu oraz jedną ze zmiennych wejściowych modelu wyliczaną na podstawie wektora sterowania `U` (zmienna wymagana przez niektóre algorytmy sterowania) na wyjściu. Opis wejść i wyjść przedstawiono w podrozdziale 6.2.1. Wykorzystanie modelu wymaga wcześniejszej inicjalizacji kilku zmiennych, których opis przedstawiono w podrozdziale 6.2.2. Umożliwiają one ustawienie odpowiednich parametrów fizycznych modelu i wartości początkowych symulacji oraz wybór typu sterowania (sterowanie „U" i „omega", podrozdział 6.2.1).

Implementację modelu oparto na przedstawionym w pierwszej części publikacji równaniu dynamiki quadrotora (2.32) opisującego w wymaganym stopniu zachowanie rzeczywistego obiektu. Model zawiera opis efektów przedstawionych w rozdziale 3.1 i opisanych równaniami (3.7) i (3.8). Opisują one efekty żyroskopowe ramy quadrotora oraz łopat śmigieł.

Parametry fizyczne modelu wczytywane są poprzez strukturę `QP_param`, która musi zostać zainicjowana odpowiednimi wartościami przed uruchomieniem symulacji. Tabela 6.2 zawiera wartości oraz opis parametrów fizycznych zastosowanych domyślnie podczas symulacji w niniejszej publikacji. W podrozdziale 6.2.3 przedstawiono przykład uruchomienia symulacji z wykorzystaniem bloku `quadcopter` wraz z przykładowym zestawem parametrów wczytanych przy pomocy skryptu 6.4. W podrozdziale 6.2.4 przedstawiono szczegóły implementacji.

Tabela 6.2. Przykładowe parametry fizyczne modelu

Nazwa	Wartość	Opis
`m`	0.686	masa $[kg]$
`g`	9.81	g $[\frac{m}{S^2}]$
`Ix`	0.0075	body inertia $[\frac{kg}{m^2}]$
`Iy`	0.0075	body inertia $[\frac{kg}{m^2}]$
`Iz`	0.014	body inertia $[\frac{kg}{m^2}]$
`Jr`	0.001	body inertia $[\frac{kg}{m^2}]$
`l`	0.315	lever $[m]$
`b`	2.07	thrust factor
`d`	0.035	drag factor
`isUcontrol`	true	wybór sterowania (0 - „omega", 1 - „U")

6.2.1. Opis wejść/wyjść

Wejściem modelu (`U`) jest czteroelementowy wektor sterowań. W zależności od ustawienia parametru `QP.isUcontrol` (6.2.2) model sterowany jest wektorem prędkości obrotowych silników $[\Omega_1, \Omega_2, \Omega_3, \Omega_4]$ (sterowanie „omega") lub wektorem sterowań $[U_1, U_2, U_3, U_4]$ (sterowanie „U"). W zależności od wyboru sterowania wektory wejściowe są zadawane wprost („U") lub w przypadku sterowania „omega" wartości są przeliczane na wektor $[U_1, U_2, U_3, U_4]$ przy pomocy równania (2.30). Wartość Ω w zależności od sterowania jest wyliczana jako suma prędkości obrotowych silników z odpowiednimi znakami (2.30) lub wyliczana na podstawie wektora sterowań `U` przy pomocy równania (2.31).

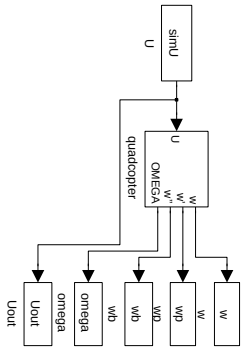
Wyjściem modelu są sześcioelementowe wektory `w`, `w'` oraz `w''` reprezentujące zmienne stanu $(x, y, z, \phi, \theta, \psi)$ oraz ich prędkości i przyspieszenia. Dodatkowo wyprowadzono zmienną `OMEGA` reprezentującą zmienną Ω z modelu dynamiki (2.32).

6.2.2. Inicjalizacja

Uruchomienie modelu wymaga wcześniejszej inicjalizacji parametrów początkowych pobieranych z przestrzeni roboczej **Matlaba**. W tym celu należy zainicjować następujące zmienne:
— `W0` - wektor [1x6] - położenia początkowe zmiennych stanu,
— `Wp0` - wektor [1x6] - prędkości początkowe zmiennych stanu,
— `paramStruct` - struktura parametrów (Tab. 6.2).

6.2.3. Przykład uruchomienia

Wydruk 6.3 przedstawia przykładowy skrypt realizujący przygotowanie i uruchomienie symulacji z modelem. Obiekt umieszczono w pliku `obiektUni.mdl` przedstawionym na rysunku 6.3.



Rysunek 6.3. Model `obiektUni.mdl`

Inicjalizację struktury `QP_param` przedstawiono na wydruku 6.4.

Wydruk 6.3. Przykład inicjalizacji i uruchomienia modelu

```
QP_param;
PP_param;
%===init====================
W0=PP.W0;
Wp0=PP.Wp0;
%===paramStruct==============
paramStruct.m = QP.m.val;
paramStruct.g = QP.g.val;
paramStruct.Ix = QP.Ix.val;
paramStruct.Iy = QP.Iy.val;
paramStruct.Iz = QP.Iz.val;
paramStruct.Jr = QP.J.val;
paramStruct.l = QP.l.val;
paramStruct.b = QP.b.val;
paramStruct.d = QP.d.val;
```

```
paramStruct.isUcontrol = QP.isUcontrol.val;
%===paramBus=================
clear slBus*;
busInfo = Simulink.Bus.createObject(paramStruct);
%===sim======================
t=sim('obiektUni', 10);
```

Wydruk 6.4. QP_param

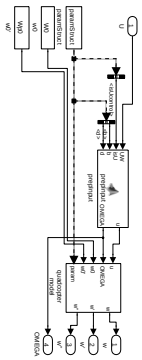
```
%%% Quadrotor parameters
%
% Source: "Design and Control of an Indoor Micro Quadrotor" (325.pdf)

% nparam(val, min_val, max_val, unit_sym, desc)
QP.isUcontrol = nparam(true, false, true, '', ...
   'Przelacznik miedzy sposobem sterowania wirnikami (omega) a (U)');

QP.m = nparam(0.686, 0, 1000, 'kg', 'Masa quadrokoptera');
QP.l = nparam(0.315, 0, 1, 'm', 'Dlugosc ramienia quadrokoptera');
QP.b = nparam(2.07, 0, 1000, '', ...
   'Wspolczynnik obciazenia (thrust factor)');
QP.d = nparam(0.035, 0, 1000, '', ...
   'Wspolczynnik oporu areodynamicznego (drag factor)');
QP.g = nparam(9.81, 9.78, 9.85, 'm/s^2', 'Przyspieszenie ziemskie');

QP.Ix = nparam(0.0075, 0, 1000, 'kg*m^2', ...
   'Moment bezwladnosci wzdloz osi x quadrokoptera');
QP.Iy = nparam(0.0075, 0, 1000, 'kg*m^2', ...
   'Moment bezwladnosci wzdloz osi y quadrokoptera');
QP.Iz = nparam(0.014, 0, 1000, 'kg*m^2', ...
   'Moment bezwladnosci wzdloz osi z quadrokoptera');
QP.J  = nparam(0.001, 0, 1000, 'kg*m^2', ...
   'Moment bezwladnosci wirnika quadrokoptera');

QP.x0 = nparam(0, -100, 100, 'm', 'Poczatkowe polozenie x0');
QP.y0 = nparam(0, -100, 100, 'm', 'Poczatkowe polozenie y0');
QP.z0 = nparam(0, -100, 100, 'm', 'Poczatkowe polozenie z0');

QP.xprim0 = nparam(0, -100, 100, 'm/s', ...
   'Poczatkowa predkosc liniowa x0');
QP.yprim0 = nparam(0, -100, 100, 'm/s', ...
   'Poczatkowa predkosc liniowa y0');
QP.zprim0 = nparam(0, -100, 100, 'm/s', ...
   'Poczatkowa predkosc liniowa z0');

QP.phi0     = nparam(0, 0, 2*pi, 'rad', ...
   'Poczatkowy kat kolysania (roll) phi0');
QP.theta0   = nparam(0, 0, 2*pi, 'rad', ...
   'Poczatkowy kat kiwania (pitch) theta0');
QP.psi0     = nparam(0, 0, 2*pi, 'rad', ...
   'Poczatkowy kat myszkowania (yaw) psi0');

QP.phiprim0     = nparam(0, 0, 2*pi, 'rad/s', ...
   'Poczatkowa predkosc kolysania (roll) phiprim0');
QP.thetaprim0   = nparam(0, 0, 2*pi, 'rad/s', ...
   'Poczatkowa predkosc kiwania (pitch) thetaprim0');
QP.psiprim0     = nparam(0, 0, 2*pi, 'rad/s', ...
   'Poczatkowa predkosc myszkowania (yaw) psiprim0');

assignin('base','QP',QP);
```

6.2.4. Implementacja

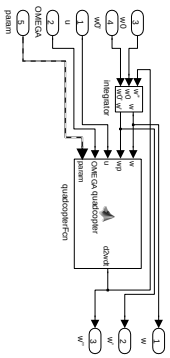
Wydruk 6.5. prepInput

```
function [u, OMEGA] = prepInput(UW, isU, b, d)
% prepInput
%
% param:
% - UW = [U1, U2, U3, U4] lub [omega1, omega2, omega3, omega4]
```

```
% - isU - true/false
% - b - scalar param
% - d - scalar param

u = zeros(4,1);
OMEGA = 0;

A = [ 1/(4*b),         0, -1/(2*b), -1/(4*d);
      1/(4*b), -1/(2*b),         0,  1/(4*d);
      1/(4*b),         0,  1/(2*b), -1/(4*d);
      1/(4*b),  1/(2*b),         0,  1/(4*d)];

if isU == 0
    u(1) = b * (UW(1)^2 + UW(2)^2 + UW(3)^2 + UW(4)^2);
    u(2) = b * (UW(4)^2 - UW(2)^2);
    u(3) = b * (UW(3)^2 - UW(1)^2);
    u(4) = d * (UW(2)^2 + UW(4)^2 - UW(1)^2 - UW(3)^2);
    OMEGA = UW(2) + UW(4) - UW(1) - UW(3);
else
    u = UW;
    om = A*UW;
    om = sqrt(om);
    OMEGA = om(2) + om(4) - om(1) - om(3);
end
```



Rysunek 6.4. Wnętrze bloku `quadcopter`

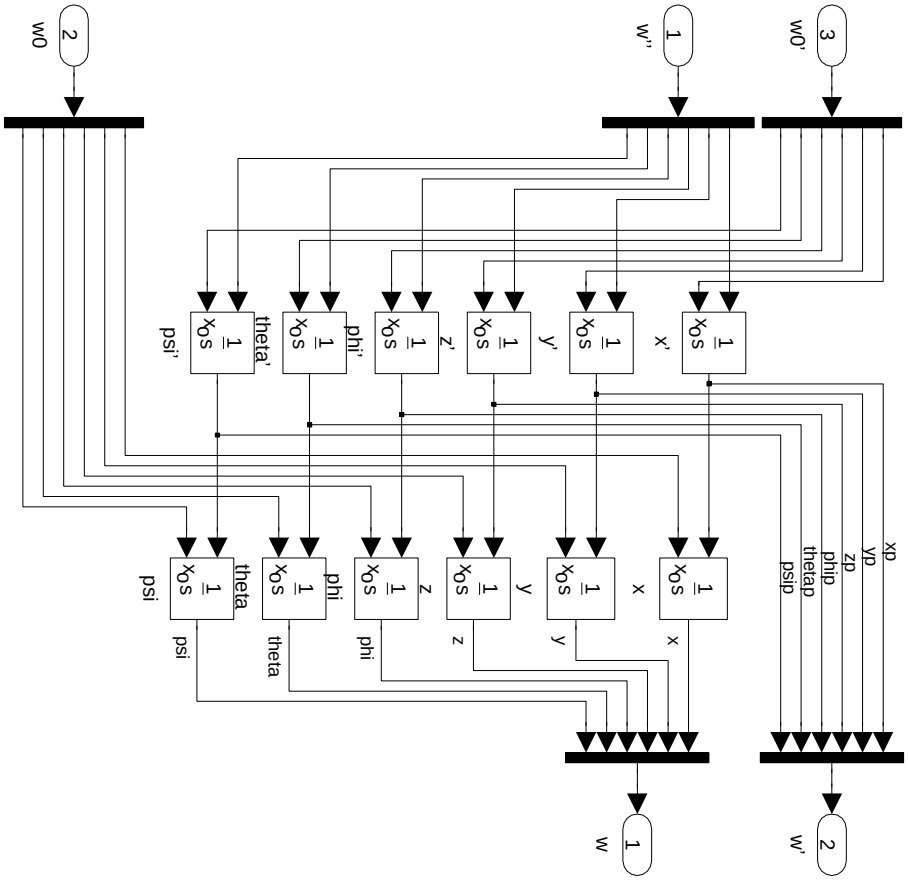


Rysunek 6.5. Wnętrze bloku `quadcopter model`

Wydruk 6.6. quadcopterFcn

```
function d2wdt  = quadcopter(w, wp , u, OMEGA, param)
%quadcopter
%
% w = [x, y, z, phi, theta, psi]
% wp = [xprim, yprim, zprim, phiprim, thetaprim, psiprim]
% u = [U1, U2, U3, U4]
```

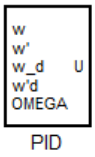
```
% OMEGA = scalar param
% param = paramStruct
%

x = w(1); xprim = wp(1);
y = w(2); yprim = wp(2);
z = w(3); zprim = wp(3);
phi = w(4); phiprim = wp(4);
theta = w(5); thetaprim = wp(5);
psi = w(6); psiprim = wp(6);

U1 = u(1); U2 = u(2); U3 = u(3); U4 = u(4);
omega = OMEGA;

m = param.m;
g = param.g;
Ix = param.Ix;
Iy = param.Iy;
Iz = param.Iz;
Jr = param.Jr;
l = param.l;

dwdt = zeros(6,1);
d2wdt = zeros(6,1);

dwdt(1) = xprim;

d2wdt(1) = (cos(phi) * sin(theta) * cos(psi) + ...
            sin(phi) * sin(psi)) * (1/m) * U1;

dwdt(2) = yprim;

d2wdt(2) = (cos(phi) * sin(theta) * sin(psi) - ...
            sin(phi) * cos(psi)) * (1/m) * U1;

dwdt(3) = zprim;

d2wdt(3) = -g + (cos(phi) * cos(theta)) * (1/m) * U1;

dwdt(4) = phiprim;

d2wdt(4) = thetaprim * psiprim * ((Iy-Iz)/Ix) - ...
            (Jr/Ix) * thetaprim * omega + ...
            (l/Ix) * U2;

dwdt(5) = thetaprim;

d2wdt(5) = phiprim * psiprim * ((Iz-Ix)/Iy) + ...
            (Jr/Iy) * phiprim * omega + ...
            (l/Iy) * U3;

dwdt(6) = psiprim;

d2wdt(6) = phiprim * thetaprim * ((Ix-Iy)/Iz) + ...
            (1/Iz) * U4;
```



Rysunek 6.6. Wnętrze bloku `integrator`

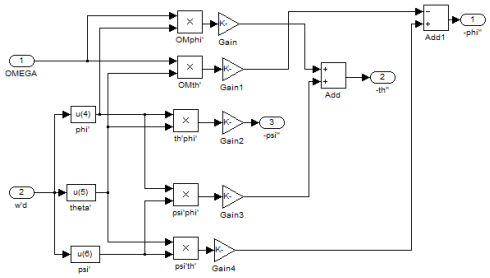
6.3. Implementacja sterownika PID

Zaimplementowany sterownik reguluje cztery zmienne - wysokość z oraz kąty ϕ, θ i ψ. Jesgo struktura zakłada pobieranie przez niego aktualnych wartości położenia i prędkości wszystkich sześciu współrzędnych z modułu modelu quadrotora. Są one zapisywane w wektorze ω i $\omega\prime$. Analogicznie, z modułu generującego trajektorię regulator poprzez wektory ω_d $\omega_d\prime$ pobiera zadawane przez użytkownika położenia i prędkości. Dodatkowo, regulator odczytuje poprzez wektor *OMEGA* prędkości obrotowe śmigieł Ω_i. Efektem działania bloku jest wygenerowanie sterowanie dla quadrotora w wektorze U. Wygląd stworzonego bloku w środowisku Matlab Simulink pokazuje rysunek 6.7.



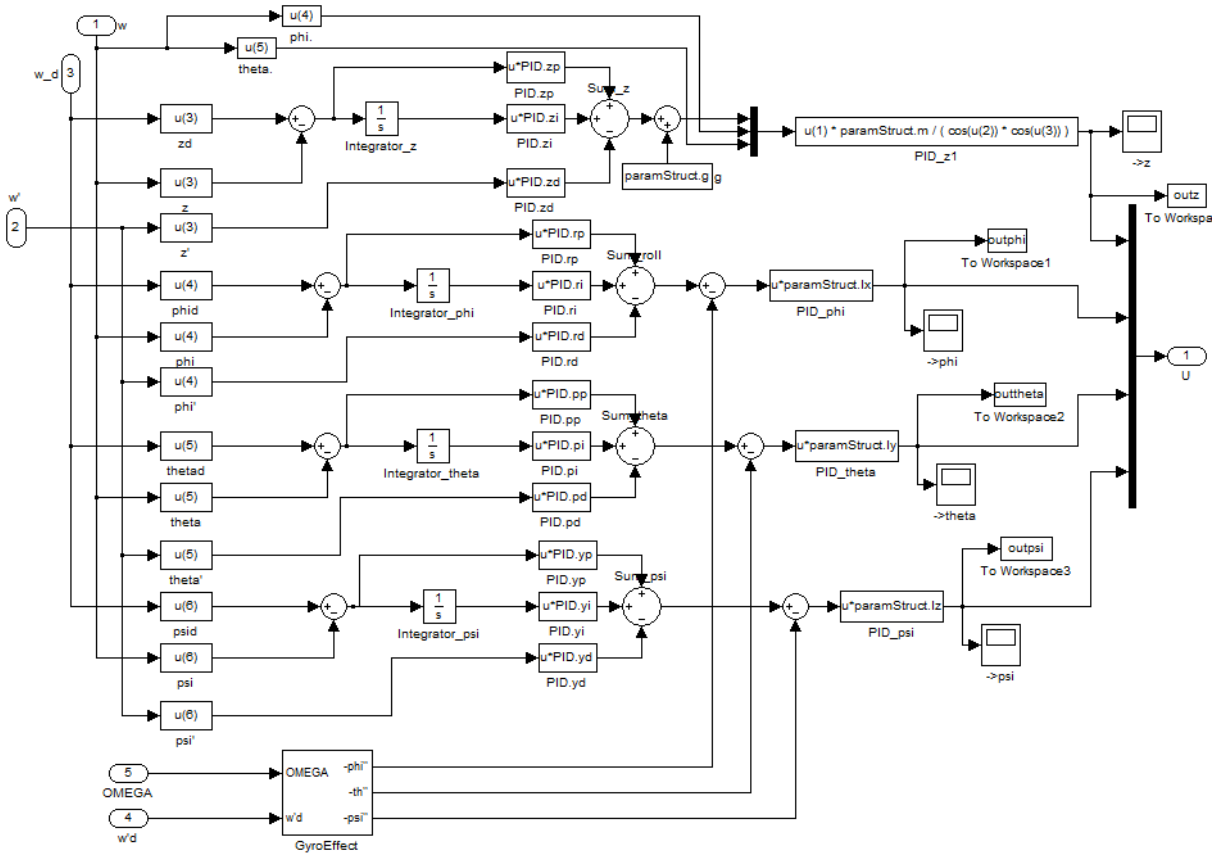
Rysunek 6.7. Postać bloku stworzonego regulatora PID

Wektor *OMEGA* i $\omega_d\prime$ służą do kompensacji efektu żyroskopowego. Zaimplementowany człon spowalnia działanie symulacji, ale umożliwia dokładniejsze odwzorowanie rzeczywistego obiektu. Blok ten przedstawiono na rysunku 6.8.



Rysunek 6.8. Człon kompensujący efekt żyroskopowy

Różnicą wartości wektorów ω_d i ω jest uchyb. Trafia on do bloków parametrów poszczególnych części regulatora PID. Sterownik wykorzystuje ich 12. Odpowiadają one za wartości wzmocnień części proporcjonalnych K_p, całkujących K_i i różniczkujących K_d dla każdej współrzędnej. Na wejście członów różniczkujących podawana jest pochodna zmiennej w celu uniknięcia gwałtownych zmian zmiennych. Dla zapewnienia większej dokładności wszystkie współrzędne kompensowane są przez odpowiednie stałe fizyczne (momenty bezwładności I_{xx}, I_{yy}, I_{zz} i odległość od środka ciężkości ll) wynikające ze wzorów. Regulacja wysokości quadrotora dodatkowo uwzględnia wpływ grawitacji oraz zmian kątów θ i ϕ. Model sterownika w środowisku Matlab Simulink przedstawia rysunek 6.9.

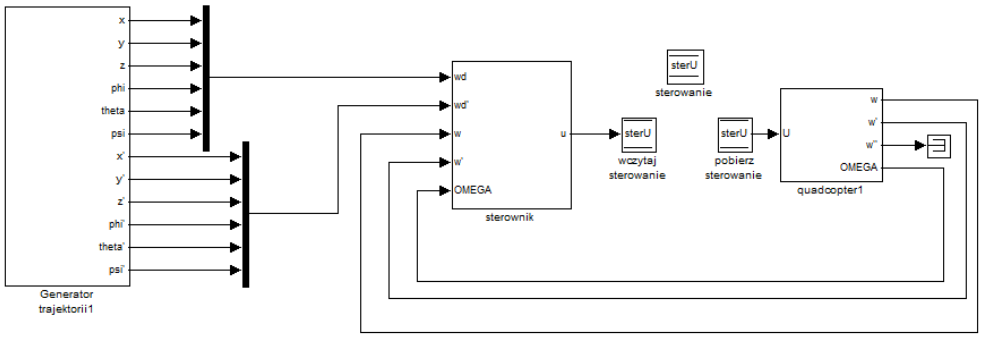


Rysunek 6.9. Zrealizowany regulator PID

6.4. Algorytm całkowania wstecznego

W tej części przedstawiona zostanie implementacja sterownika wykorzystującego całkowanie wsteczne (opisanego w 4.2.3) w środowisku MATLAB/Simulink. Podczas realizacji sterowania pojawiły się pewne problemy, o których nie wspominano w literaturze, a które są istotne z punktu widzenia implementacji. Zostaną one uwzględnione w kolejnych paragrafach.

Poszczególne elementy sterownika zamodelowane zostały w postaci zagnieżdżonych podsystemów. Rysunek 6.10 pokazuje blok sterownika podłączony do generatora trajektorii zadanej oraz modelu.



Rysunek 6.10. Podłączenie sterownika do układu

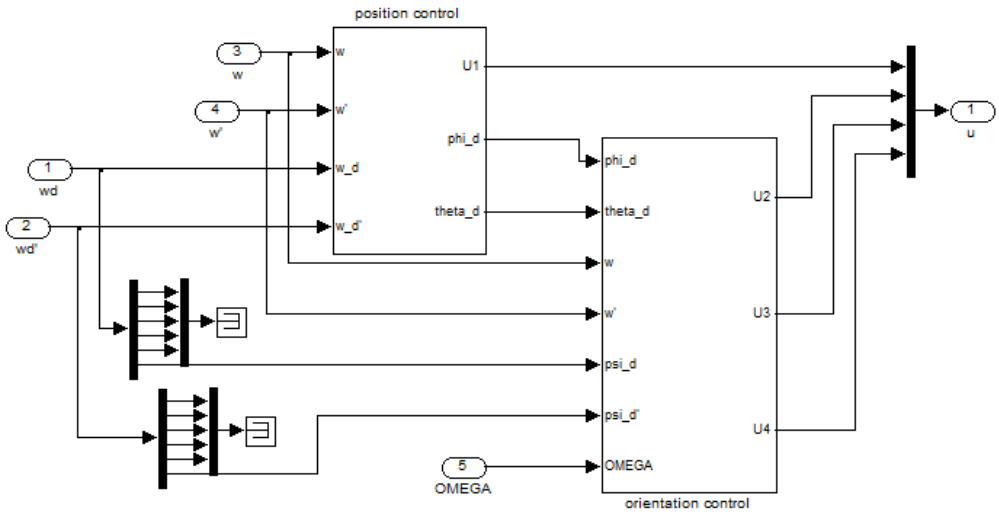
Obliczanie wektora stanu quadrotora q, jego pochodnych $\dot{q}$ i $\ddot{q}$ oraz sterowania zamknięte jest w pętli algebraicznej[3]. Z tego też powodu konieczne stało się dodanie do systemu opóźnienia. Aby model działał dla różnych metod całkowania (zarówno stało- jak zmiennokrokowych), zdecydowano się na wykorzystanie bloków pamięci (odczyt, zapis i rezerwacja przestrzeni na dane), w których można ustawić zmienny czas próbkowania (w przeciwieństwie do popularniejszego bloku opóźnienia).

Rysunek 6.11 pokazuje wnętrze bloku sterownika. Widoczna jest na nim modułowa budowa pokazana wcześniej na rysunku 4.10.

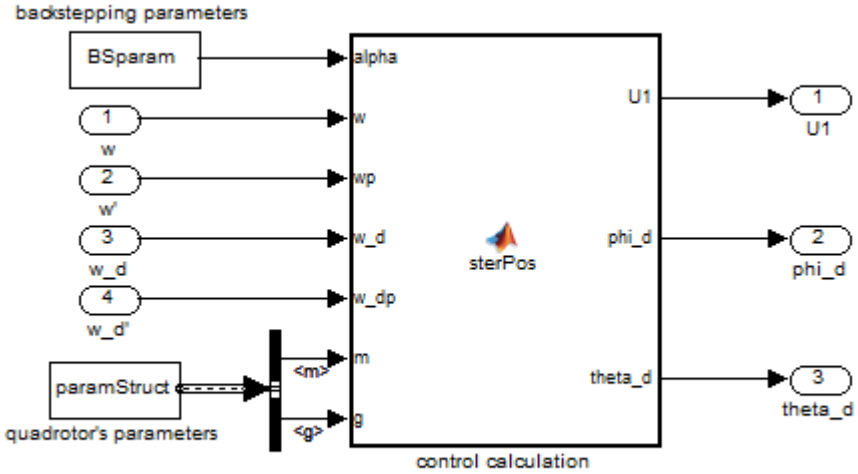
Bloki reprezentujące sterownik położenia i kątów – *position control* i *orientation control* – zbierają sygnały im sygnnały, by przekazać je do funkcji wyliczających wartości zadawane U zaimplementowanych w *MATLAB Function Blok*.

Sterownik położenia pokazany jest na rysunku 6.12, zaś funkcja wyliczająca sterowanie U_1 i kąty ϕ_d oraz θ_d – na wydruku 6.7. W trakcie testowania systemu okazało się, że konieczne jest nałożenie na ten sterownik dodatkowych ograniczeń, tak aby żaden z zadawanych kątów ϕ i θ nie przekroczył wartości bezwzględnej $\pi/4$. Ograniczenia te są zgodne z intuicją – przechylenie quadrotora o większy kąt zapoczątkowałoby pełny obrót wokół osi X lub Y. Takie zachowanie nie jest uwzględnione ani w modelu sterownika, ani samego quadrotora, szybko prowadzi więc do destabilizacji systemu oraz dziwnych zachowań systemu (np. sterownik zadaje nierealizowalne sterowania, definiujące zespolone prędkości obrotowe śmigieł). Ze względu na możliwy błąd śledzenia trajektorii, a więc możliwość uzyskania większych kątów niż zadawane, wprowadzony został margines bezpieczeństwa i wartość bezwzględna kątów została ograniczona do $\pi/5$.

[3] ang. algebraic loop



Rysunek 6.11. Ogólny model sterownika



Rysunek 6.12. Sterownik położenia

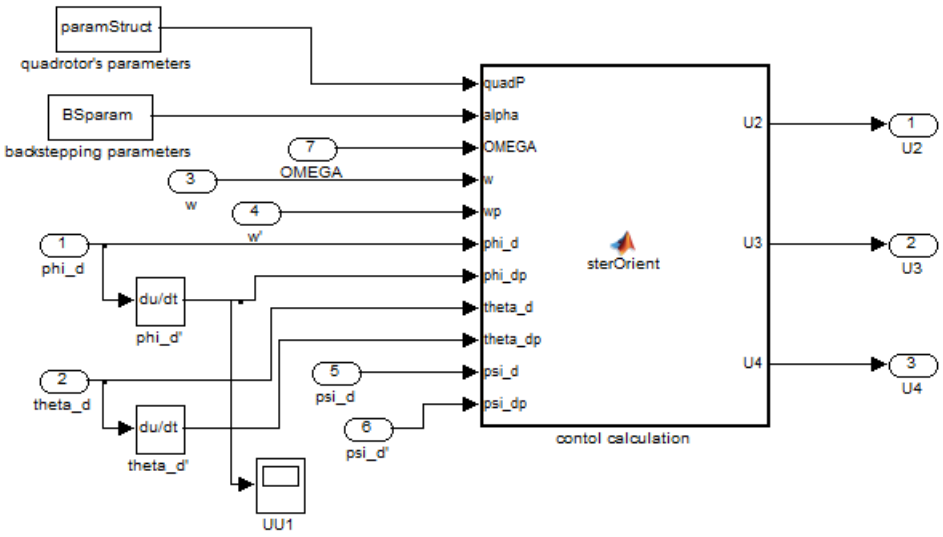
Wydruk 6.7. Funkcja sterPos

```
function [U1, phi_d, theta_d] = sterPos(alpha, w, wp, w_d, w_dp, m, g)

% sterPos
%
% param:
% w = [x, y, z, phi, theta, psi]
% wp = [xprim, yprim, zprim, phiprim, thetaprim, psiprim]
% - m - quadrocopter mas (scalar)
% - BSP - backstepping algorithm parameters
% - w, wp - trajectory and its derivative
% - w_d, w_dp - given trajectory and its derivative

x = w(1); xprim = wp(1);
y = w(2); yprim = wp(2);
z = w(3); zprim = wp(3);
phi = w(4);
theta = w(5);
psi = w(6);
x_d = w_d(1); x_dprim = w_dp(1);
y_d = w_d(2); y_dprim = w_dp(2);
z_d = w_d(3); z_dprim = w_dp(3);

e_bs7 = z_d - z;
e_bs8 = zprim - z_dprim - alpha(7) * e_bs7;
e_bs9 = x_d - x;
e_bs10 = xprim - x_dprim - alpha(9) * e_bs9;
e_bs11 = y_d - y;
e_bs12 = yprim - y_dprim - alpha(11) * e_bs11;

U1 = (e_bs7+g-alpha(7)*(e_bs8 + alpha(7)*e_bs7)-alpha(8)*e_bs8)* m / (cos(phi)*cos(theta));

ux = (e_bs9-alpha(9)*(e_bs10+alpha(9)*e_bs9)-alpha(10)*e_bs10)* m / U1;

uy = (e_bs11-alpha(11)*(e_bs12+alpha(11)*e_bs11)-alpha(12)*e_bs12)* m / U1;

s1 = ux*sin(psi)+uy*cos(psi);

%angle bounds
if s1 > sin(pi/5)
    phi_d = -pi/5;
elseif s1 < sin(-pi/5)
    phi_d = pi/5;
else
    phi_d = -asin(s1);
end

s3 = (ux*cos(psi)+uy*sin(psi)) / cos(phi_d);

%angle bounds
if s3 > sin(pi/5)
    theta_d = -pi/5;
elseif s3 < sin(-pi/5)
    theta_d = pi/5;
else
    theta_d = asin(s3);
end
```

Sterownik orientacji pokazany jest na rysunku 6.13, funkcja wyliczająca sterowania U_2, U_3 i U_4 – na wydruku 6.8. W przypadku sterowania orientacją wstawione zostały dodatkowe bloki na wejściach sterownika. Wyliczenie błędów trajektorii zadanych równaniami (4.25) i (4.30) wymaga znajomości pochodnych kątów ϕ i θ, a sterownik położenia ich nie dostarcza. Konieczne okazało się wykorzystanie bloków pochodnej. Zazwyczaj unika się różniczkowania numerycznego, ze względu na duże błędy w obliczeniach, jednak w przypadku quadrotora wspomniane kąty są wolnozmienne i ich różniczkowanie nie stanowi problemu.

Wydruk 6.8. Funkcja sterOrient

```
function [U2, U3, U4] = sterOrient(quadP, alpha, OMEGA, w, wp, phi_d, phi_dp, theta_d,theta_dp, psi_d, psi
% sterOrient
%
```



Rysunek 6.13. Sterownik orientacji

```
% param:
% - quadP - quadrocopter parameters structure
% - BSP - backstepping algorithm parameters
% - OMEGA - scalar
% - w, wp - trajectory and its derivative
% - psi_d, ..., phi_dp - given trajectory and its derivative

phi = w(4); phip = wp(4);
theta = w(5); thetap = wp(5);
psi = w(6); psip = wp(6);

a1 = (quadP.Iy - quadP.Iz) / quadP.Ix;
a2 = - quadP.Jr / quadP.Ix;
a3 = (quadP.Iz - quadP.Ix) / quadP.Iy;
a4 = quadP.Jr / quadP.Iy;
a5 = (quadP.Ix - quadP.Iy) / quadP.Iz;

b1 = quadP.l / quadP.Ix;
b2 = quadP.l / quadP.Iy;
b3 = quadP.l / quadP.Iz;


e_bs1 = phi_d - phi;
e_bs2 = phip - phi_dp - alpha(1) * e_bs1;
e_bs3 = theta_d - theta;
e_bs4 = thetap - theta_dp - alpha(3) * e_bs3;
e_bs5 = psi_d - psi;
e_bs6 = psip - psi_dp - alpha(5) * e_bs5;

U2 = (e_bs1-a1*thetap*psip-a2*thetap*OMEGA-...
    alpha(1)*(e_bs2 + alpha(1)*e_bs1)-alpha(2)*e_bs2) / b1;

U3 = (e_bs3-a3*phip*psip-a4*phip*OMEGA-...
    alpha(3)*(e_bs4 + alpha(3)*e_bs3)-alpha(4)*e_bs4) / b2;

U4 = (e_bs5-a5*thetap*phip-alpha(5)*(e_bs6 + alpha(5)*e_bs5)-alpha(6)*e_bs6) / b3;
```

6.4.1. Inicjalizacja

W celu uruchomienia sterownika (poza dostarczonym GUI) należy:

— zainicjalizować model quadrotora,

— utworzyć wektor parametrów sterownika `BSparam` z wybranymi nastawami sterownika,
— ustawić wybraną trajektorię w generatorze.

Utworzenie parametrów może się odbyć za pomocą przygotowanych skryptów: *BackSteppingParam.m* tworzącego strukturę `BSP` z opisem parametrów sterownika oraz *BackStep.m* inicjalizującego wektor `BSparam` wartościami zadanymi w `BSP`, co pokazano na wydrukach 6.9 i 6.10.

Wydruk 6.9. Utworzenie struktury BSP

```
%%% Backstepping algorithm parameters
%
% Source: S. Bouabdallah, R. Siegwart \
% "Backstepping and Sliding-mode Techniques Applied
%                                   to an Indoor Micro Quadrotor"

% nparam(val, min_val, max_val, unit_sym, desc)
BSP.alpha01 = nparam(1, 0, Inf, '',...
                        '1. parametr stabilizujacy kat fi');
BSP.alpha02 = nparam(1, 0, Inf, '',...
                        '2. parametr stabilizujacy kat fi');
BSP.alpha03 = nparam(1, 0, Inf, '',...
                        '1. parametr stabilizujacy kat teta');
BSP.alpha04 = nparam(1, 0, Inf, '',...
                        '2. parametr stabilizujacy kat teta');
BSP.alpha05 = nparam(1, 0, Inf, '',...
                        '1. parametr stabilizujacy kat psi');
BSP.alpha06 = nparam(1, 0, Inf, '',...
                        '2. parametr stabilizujacy kat psi');
BSP.alpha07 = nparam(3.1, 0, Inf, '',...
                        '1. parametr stabilizujacy wsp. z');
BSP.alpha08 = nparam(3.4, 0, Inf, '',...
                        '2. parametr stabilizujacy wsp. z');
BSP.alpha09 = nparam(1, 0, Inf, '',...
                        '1. parametr stabilizujacy wsp. x');
BSP.alpha10 = nparam(1, 0, Inf, '',...
                        '2. parametr stabilizujacy wsp. x');
BSP.alpha11 = nparam(2, 0, Inf, '',...
                        '1. parametr stabilizujacy wsp. y');
BSP.alpha12 = nparam(1, 0, Inf, '',...
                        '2. parametr stabilizujacy wsp. y');
```

Wydruk 6.10. Inicjalizacja wektora parametrów

```
BSparam = zeros (1,12);

BSparam(1) = BSP.alpha01.val;
BSparam(2) = BSP.alpha02.val;
BSparam(3) = BSP.alpha03.val;
BSparam(4) = BSP.alpha04.val;
BSparam(5) = BSP.alpha05.val;
BSparam(6) = BSP.alpha06.val;
BSparam(7) = BSP.alpha07.val;
BSparam(8) = BSP.alpha08.val;
BSparam(9) = BSP.alpha09.val;
BSparam(10) = BSP.alpha10.val;
BSparam(11) = BSP.alpha11.val;
BSparam(12) = BSP.alpha12.val;

paramStruct.isUcontrol = true;
```

6.5. Implementacja algorytmu H_∞ dla quadrotora

Niech dany będzie model dynamiki quadrotora zgodny z (2.32), na podstawie którego dokonano implementacji badanego obiektu w pakiecie Matlab/Simulink.

Jak zostało wspomniane wcześniej aby umożliwić implementację algorytmu H_∞ model quadrotora musi zostać przedstawiony w postaci układu zapisanego przy użyciu równań stanu, w których macierze współczynników są funkcjami zewnętrznych zmiennych szedulujących α, $\dot{\alpha}$. Zgodnie z [9] reprezentacja układu macierzowego takiego układu wygląda następująco:

$$\begin{bmatrix}\dot{x}\\ z\\ \dot{y}\end{bmatrix} = \begin{bmatrix} A(\alpha,\dot{\alpha}) & \begin{bmatrix}B_1(\alpha,\dot{\alpha}) & B_2(\alpha,\dot{\alpha})\end{bmatrix} \\ \begin{bmatrix}C_1(\alpha,\dot{\alpha})\\ C_2(\alpha,\dot{\alpha})\end{bmatrix} & \begin{bmatrix}D_{11}(\alpha,\dot{\alpha}) & D_{12}(\alpha,\dot{\alpha})\\ D_{21}(\alpha,\dot{\alpha}) & D_{22}(\alpha,\dot{\alpha})\end{bmatrix}\end{bmatrix}\begin{bmatrix}x\\ v\\ u\end{bmatrix}, \tag{6.1}$$

gdzie x jest wektorem stanu, u wejścia, a z sygnału błędu, y oznacza mierzalne wyjściem, v zakłócenia.

UWAGA: notacja zastosowana w niniejszym rozdziale zgodna jest z [9], aby ułatwić czytelnikowi porównanie zaprezentowanego rozwiązania z literaturą i może momentami powielać wcześniej wprowadzone oznaczenia – należy je zatem od siebie odróżniać.

Jeśli dla modelu (6.1) dobrze dobierze się zmienne α i $\dot{\alpha}$ system będzie charakteryzował się odpornością stabilności w zamkniętej pętli sprzężenia zwrotnego. Jednakże tak przedstawiony model nie jest akceptowalny przez problem sterowania quadratorem. Należy przekształcić go do quasi-modelu podającego formę, która jest od razu gotowa do linearyzacji.

$$\begin{bmatrix}\dot{\alpha}\\ q-q_{eq}(\alpha)\\ \delta-\delta_{eq}(\alpha)\end{bmatrix} = \begin{bmatrix}0 & A_{12}(\alpha) & B_{11}(\alpha)\\ 0 & A_{22}-\frac{d}{d\alpha}[q_{eq}(\alpha)]A_{12}(\alpha) & B_{21}-\frac{d}{d\alpha}[q_{eq}(\alpha)]B_{11}(\alpha)\\ 0 & -\frac{d}{d\alpha}[q_{eq}(\alpha)]A_{12}(\alpha) & -\frac{d}{d\alpha}[q_{eq}(\alpha)]B_{11}(\alpha)\end{bmatrix}\begin{bmatrix}\alpha\\ q-q_{eq}(\alpha)\\ \delta-\delta_{eq}(\alpha)\end{bmatrix} \tag{6.2}$$

6.5.1. Model dynamiki quadrotora opisany równaniami stanu

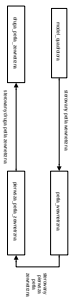
Model (2.32) po przekształceniu do quasi-modelu (6.2) przyjmuje postać:

$$\frac{d}{dt}\begin{bmatrix}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\\ \dot{\theta}\\ \dot{\phi}\\ \dot{\psi}\\ q\end{bmatrix}\\ \begin{bmatrix}\theta\\ \phi\\ \psi\end{bmatrix}\end{bmatrix} = \begin{bmatrix}0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&-1&0&0&0\\ 0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0&0\\ 0&0&0&0&0&1&0&0&0&0\end{bmatrix}\begin{bmatrix}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\\ \dot{\theta}\\ \dot{\phi}\\ \dot{\psi}\\ q\end{bmatrix}\\ \begin{bmatrix}\theta\\ \phi\\ \psi\end{bmatrix}\end{bmatrix} + \begin{bmatrix}\frac{\cos\psi\sin\theta\cos\phi+\sin\psi\sin\phi}{m} & 0&0&0\\ \frac{\cos\phi\sin\theta\sin\psi-\sin\phi\cos\psi}{m} & 0&0&0\\ \frac{\cos\phi\cos\theta}{m} & 0&0&0\\ 0 & \frac{l}{I_y}&0&0\\ 0 & 0&\frac{l}{I_y}&0\\ 0 & 0&0&I_z\\ 0 & 0&0&0\\ 0 & 0&0&0\\ 0 & 0&0&0\\ 0 & 0&0&0\end{bmatrix}\begin{bmatrix}U_1\\ U_2\\ U_3\\ U_4\end{bmatrix} \tag{6.3}$$

gdzie kąty KKM $\begin{bmatrix}\theta & \phi & \psi\end{bmatrix}^T$ pełnią rolę zmiennych szedulujących.

6.5.2. Architektura sterownika H_∞

Zaimplementowaną architekturę sterownika H_∞ dla quadrotora ilustruje rysunek 6.14.



Rysunek 6.14. Architektura sterownika H_∞

Linearyzacja modelu

Wewnętrzna pętla sterownika ma za zadanie zapewnić odporność stabilności układu quadrotora przy wznoszeniu się i podczas lotu zakładając występowanie zewnętrznych zakłóceń w powietrzu (np. wiatr). Jej drugim zadaniem jest odsprzęgnięcie systemu, który jest systemem MIMO. Wektor sterowań przyjmuje postać:

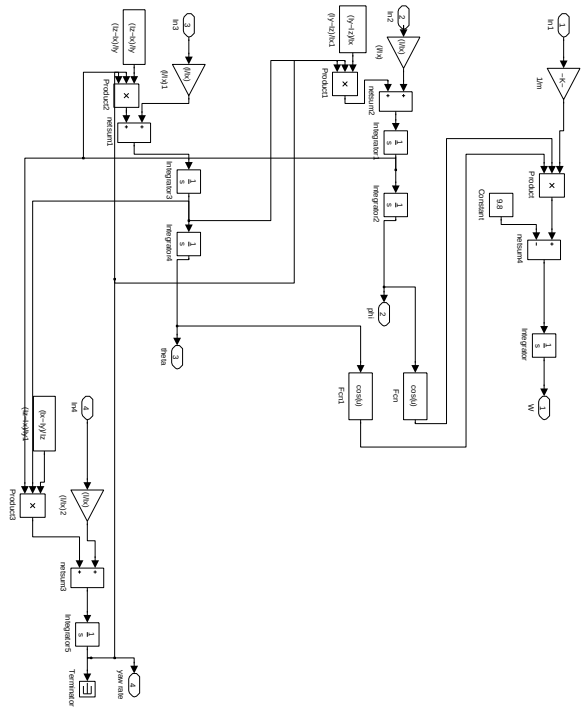
$$u = \begin{bmatrix} z & y & z & \psi \end{bmatrix}^T. \tag{6.4}$$

Aby wewnętrzna pętla sterownika zadziałała poprawnie model quadrotora musiał zostać zlinearyzowany.

Plik `quad00nh.mdl` (rysunek 6.15) zawiera model samego quadrotora (bez pętli kształtowania). Nie jest to ten sam model co wyprowadzony w (2.32), a którego implementacja została przedstawiona w rozdziale 6.2. Zastosowany w nim blok `fcn` do obliczania modelu, powodował wykrywanie powiązania między całkowaniem położeń wejściowych x, y, a wyjściami (położenia, kąty KKM) co do linearyzacji modelu nie było potrzebne, a powodowało problemy podczas rozwiązywania podwójnego algebraicznego równania Riccatiego (wydruk 6.11). Usunięcie nieużywanych na tym etapie bloków całkujących oraz wyzerowanie wartości x i y wraz z ich pochodnymi dalej generowało przy linearyzacji nieprawidłowy model – nawet podczas wybrania w opcjach linearyzacji, aby pomijał niewykorzystane bloki. Dopiero stworzenie nowego modelu `quad00nh.mdl` rozwiązało ten problem i pozwoliło swobodnie generować sterowniki H_∞ dla quadrotora. Przy budowie modelu H_∞ pominięto efekt żyroskopowy śmigieł.

Wydruk 6.11. Wyliczenie równań Riccatiego

```
function [K gam]=hinf2dof(Gs, Tref)
%HINF2DOF  synthesizes the H_inf 2-DOF controller in (9.80)
% Uses MATLAB RCAST toolbox
% Usage: K=hinf2dof(Gs, Tref);
% INPUTS:  Shaped plant Gs and reference model Tref
% OUTPUT: Two degrees-of-freedom controller K
%
% Coprime factorization of Gs
[As,Bs,Cs,Ds] = unpck(Gs);
[Ar,Br,Cr,Dr] = unpck(Tref);
[nr,nr] = size(Ar); [lr,mr] = size(Dr);
[ns,ns] = size(As); [ls,ms] = size(Ds);
Rs = eye(ls)+Ds*Ds.'; Ss = eye(ms)+Ds'*Ds;
A1 = (As - Bs*inv(Ss)*Ds'*Cs);
R1 = Cs'*inv(Rs)*Cs; Q1 = Bs*inv(Ss)*Bs';

% [Z1, Z2, fail, reig_min] = ric_schr([A1' -R1; -Q1 -A1]);
% fail
% [Z1,Z2,eig,zerr,wellposed,Zs] = aresolv(A1',Q1,R1);
% Zs = Z2/Z1;
% Alt. Robust Control toolbox:
[Z1,Z2,eig,zerr,wellposed,Zs] = aresolv(A1',Q1,R1);
%
% Choose rho=1 (Designer's choice) and
```



Rysunek 6.15. Linearyzowany model quadrotora (`quad00nh.mdl`)

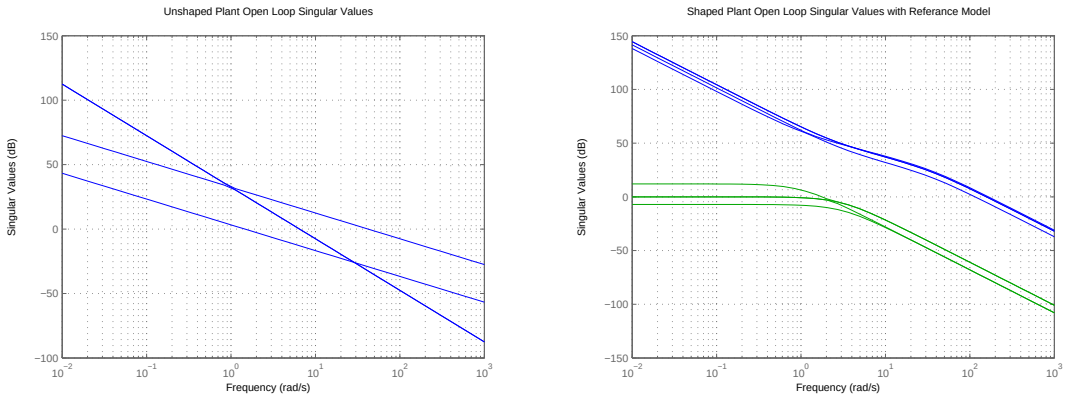
```
% build the generalized plant P in (9.87)
%

rho=1;
A = blkdiag(As,Ar);
B1 = [zeros(ns,mr) ((Bs*Ds')+(Zs*Cs'))*inv(sqrtm(Rs));
     Br zeros(nr,ls)];
B2 = [Bs;zeros(nr,ms)];
C1 = [zeros(ms,ns+nr);Cs zeros(ls,nr);rho*Cs -rho*rho*Cr];
C2 = [zeros(mr,ns+nr);Cs zeros(ls,nr)];
D11 = [zeros(ms,mr+ls);zeros(ls,mr) sqrtm(Rs);
-rho*rho*Dr rho*sqrtm(Rs)];
D12 = [eye(ms);Ds;rho*Ds];
D21 = [rho*eye(mr) zeros(mr,ls);zeros(ls,mr) sqrtm(Rs)];
D22 = [zeros(mr,ms);Ds];
B = [B1 B2]; C = [C1;C2]; D = [D11 D12;D21 D22];
P = ss(A,B,C,D);
% Alternative: Use sysic to generate P from Figure 9.21
% but may get extra states, since states from Gs may enter twice.
%
% Gamma iterations to obtain H-infinity controller
%
[l1,m2] = size(D12); [l2,m1] = size(D21);
nmeas = l2; ncon = m2; gmin = 1; gmax = 15; gtol = 0.01;

% [K, Gnclp, gam] = hinfsyn(P, nmeas, ncon, gmin, gmax, gtol);
% Alt. Robust toolbox, use command: hinfopt
[K, Gnclp, gam] = shinfopt(A,B,C,D, nmeas, ncon, gmax, gmin, gtol);
```

Wydruk 6.12. Linearyzacja modelu i analiza wartości dla modelu bez kształtowania pętli (appendix1ca.m)

```
% Quad Rotor UAV 2-DOF H - I n f i n i t y Analysis and Synthesis
% Designed by Ming Chen
% Date: A p r i l 2003

% updated implementation and adaptation to the project
% by Natalia Czop & Maciej Gawron
```

```
% Date: January 2013

% Section One: Calcuate the H-inf controller for the inner loop

% Step 1: Linearize the plant and analysis
% the singular values of the unshaped model
[x,u] = trim('quad00nh');
[A,B,C,D] = linmod('quad00nh',x,u);
sys = ss(A,B,C,D);
sys_tf = tf( sys );
G = pck(A,B,C,D);
figure(1);
clf;
hold on;
sigma(A,B,C,D,{0.01,1000})
title('Unshaped Plant Open Loop Singular Values'); grid pause;
```

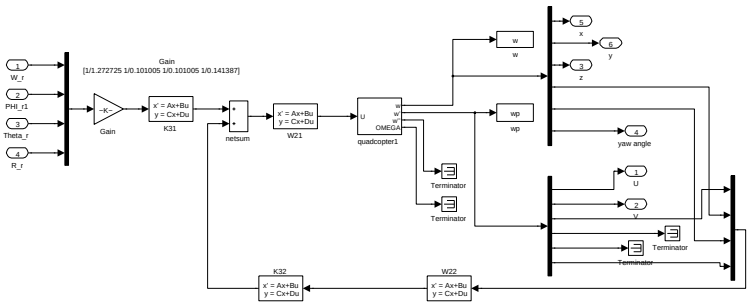
Wykresy na rysunku 6.16 przedstawiają porównanie odpowiedzi częstotliwościowej modelu quadrotora przed i po filtracji dokonanej za pomocą W_1 i W_2 zgodnie ze wskazówkami zawartymi w sekcji 4.3.2 oraz [25] i [9]. Wartości dobranych parametrów umieszczone są w wydruku 6.12. Kompensatory pozwoliły skupić częstotliwości krzyżowania sterowanych wartości w okolicy 150 rad. Niestety w przypadku niższych częstotliwości krzyżowania układ zaczynał być niestabilny. W prekompensatorze W_2 konieczne okazało się dodanie zer umieszczonych w lewej półpłaszczyźnie płaszczyzny liczb zespolonych w częściach odpowiedzialnych za kształtowanie kątów kiwania i kołysania. Pozwoliły one zagwarantować nachylenie zbocza wzmocnienia tych kątów w okolicach częstotliwości krzyżowania na poziomie -1.



Rysunek 6.16. Analiza wartości dla otwartej pętli przed i po kształtowaniu

Wydruk 6.13. Wybranie funkcji przenoszenia (appendix1ca.m)

```
% Step 2: Choose weights and Tref and analysis the singular
% values of the shaped plant system matrix Gs
WA = [ 0.2 0 ];
w11 = nd2sys([70 70], WA);
w13 = nd2sys(8, 1);
w14 = nd2sys([1.2, 3.6] , WA ) ;
W1 = daug(w11,w13,w13,w14);

WB = [ 1 35];
w21 = nd2sys([55], WB);
w23 = nd2sys(75*[1 2.31],WB);
W2 = daug(w21,w23,w23,w21);

[Aw1,Bw1,Cw1,Dw1] =unpck(W1);
[Aw2,Bw2,Cw2,Dw2] = unpck(W2);
[W21A,W21B,W21C,W21D] = unpck(W1);
[W22A,W22B,W22C,W22D] = unpck(W2);
```



Rysunek 6.17. Wewnętrzna pętla sterowania dla sterownika H_∞ (`quad11.mdl`)

```
Tref_model = nd2sys(2^2, [1 2*0.95*1 1^2]);
Tref_model1 = nd2sys(3^2,[1 2*0.95*3 3^2]);
Tref_model2 = nd2sys(2^2,[1 2*0.95*3 3^2]);
Tref = daug(Tref_model,Tref_model1,Tref_model1,Tref_model2);

Gs = mmult(W2,G,W1);
[As,Bs,Cs,Ds] = unpck(Gs);
figure(2);
clf;
hold on;
sigma (As, Bs, Cs, Ds, {0.01,1000})
sigma (Tref, {0.01,1000})
title('Shaped Plant Open Loop Singular Values with
                    Referance Model')
grid
```

Kolejnym etapem postępowania jest synteza sterownika oraz wyznaczenie $K = \begin{bmatrix} K_1 & K_2 \end{bmatrix}$.

Wydruk 6.14. Synteza sterownika i wyznaczenie K (appendix1ca.m)

```
% Step 3: Synthesize 2-DOF controller K and get gamma= 2.8444
K = hinf2dof(Gs, Tref ) ;

% Step 4: Seperate K to K l and K2
minfo(K) % system: 20 states 4 outputs 8 inputs
K1 = sel ( K , [ 1 2 3 4] , [1 2 3 4]) ;
K2 = sel(K, [1 2 3 4], [5 6 7 8]);
[Ak1,Bk1,Ck1,Dk1] = unpck(K1);
[Ak2,Bk2,Ck2,Dk2] = unpck(K2);
[K31A,K31B,K31C,K31D] = unpck(K1);
[K32A,K32B,K32C,K32D] = unpck(K2);
```

Zlinearyzowanie modelu pozwala na podłączenie do obiektu sterowania pętli wewnętrznej.

Implementacja wewnętrznej pętli w pakiecie Matlab/Simulink przedstawiona jest na rysunku 6.17.

Musi ona spełniać następujące wymagania:

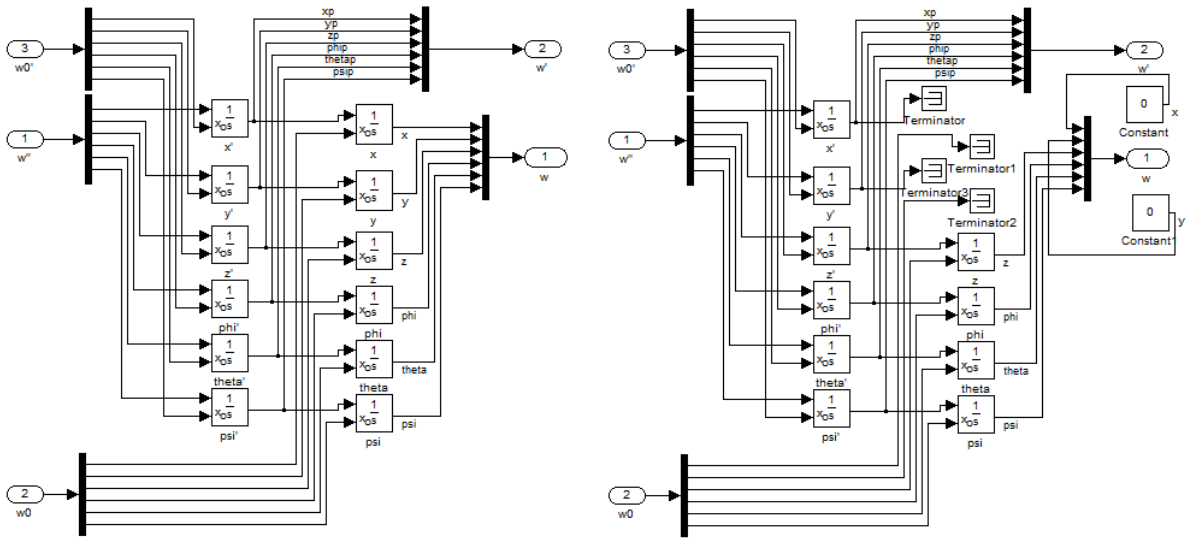
— wszystkie bieguny zamkniętej pętli muszą leżeć w lewej półpłaszczyźnie płaszczyzny zespolonej, aby zapewnić stabilność układu;

— układ musi być w stanie odrzucić szybkie (krótko trwające) zakłócenia impulsowe i krokowe.

Układ z rysunku 6.17 również należy zlinearyzować, następnie przeprowadzić syntezę sterownika i wyznaczyć $K = \begin{bmatrix} K_1 & K_2 \end{bmatrix}$. Ponownie konieczne było wyeliminowanie błędów modelu związanych z całkowaniem niewykorzystanych na tym etapie projektowania sterownika współrzędnych. Schemat całkowania – oryginalny i po wprowadzeniu zmian przedstawiono na rysunku 6.18.

Wydruk 6.15. Obliczenie parametrów sterownika H_∞ dla pierwszej pętli zewnętrznej – linearyzacja pętli wewnętrznej i wyznaczenie współczynników (appendix2ca.m)

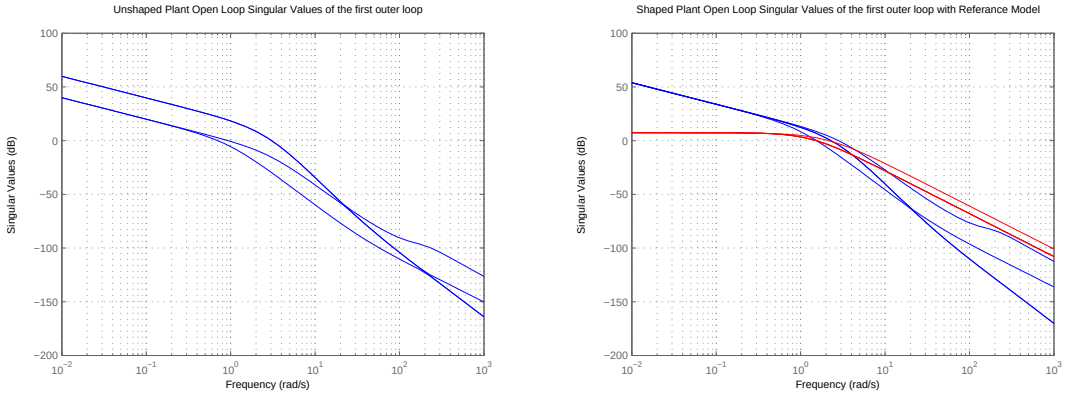
```
% Section Two: Calcuate the H_inf controller the first outer loop
```



Rysunek 6.18. Schemat różniczkowania dla modelu oryginalnego i H_∞

```
% Step 1: Linearize the plant and analysis
% the singular values of the unshaped model

[x,u] = trim('quad11_work');
[Aoi,Boi,Coi,Doi] = linmod('quad11_work',x,u, 1);
syso = ss(Aoi,Boi,Coi,Doi);
sys_tfo = tf ( syso ) ;

Go = pck(Aoi,Boi,Coi,Doi);

figure(1)
clf
sigma(Go, {0.01,1000});
title('Unshaped Plant Open Loop Singular Values of
        the first outer loop')
grid

% Step 2: Choose weight Wl(s) and Tref and analysis
% the singular values of the shaped plant
WA1 = 5;
WB1 = 0.5;

wo11 = nd2sys(WA1,1);
wo12 = nd2sys(WB1,1);
wo13 = wo12;
wo14 = wo11;

Wo1 = daug(wo11,wo12,wo13,wo14);
[W11A,W11B,W11C,W11D] = unpck(Wo1);

Tref_modelo1 = nd2sys(2^2, [1 2*1*1.3 1.3^2]);
Tref_modelo2 = nd2sys(3^2,[1 2*1*2.1 2^2]);
Trefo = daug(Tref_modelo1,Tref_modelo1,Tref_modelo1,Tref_modelo2);

Gso = mmult(Go,Wo1);

figure(2)
clf
hold on;
sigma(Gso, {0.01,1000});
sigma(Trefo, {0.01,1000}, 'r');
title('Shaped Plant Open Loop Singular Values of the
 first outer loop with Referance Model')
grid
```

```
% Step 3: Synthesize 2-DOF controller K and get gamma= 3.6056
Ko = hinf2dof(Gso, Trefo) ;


% Step 4: Reduce Ko and seperate to Kol and Ko2
minfo(Ko) % system: 64 states     4 outputs     8 inputs

% Reduce Ko to 20 states, 4 outputs and 8 inputs
[Ko_cf,sigKo] = sncfbal(Ko);
Ko_20 = cf2sys(hankmr(Ko_cf,sigKo,20,'d'));
gapKo_20= nugap(Ko,Ko_20) % gap is closed to zero.

Ko1 = sel(Ko_20, [1 2 3 4], [1 2 3 4]);
Ko2 = sel(Ko_20, [1 2 3 4], [5 6 7 8]);

[K21A,K21B,K21C,K21D] = unpck(Ko1);
[K22A,K22B,K22C,K22D] = unpck(Ko2);
```



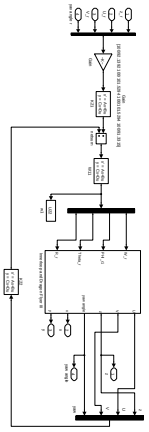
Rysunek 6.19. Analiza wartości dla pętli wewnętrznej przed i po kształtowaniu

Pętle zewnętrzne

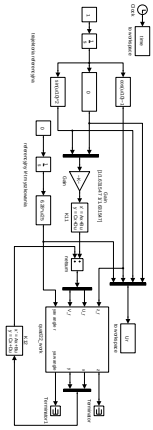
Kolejne dwie pętle zgodne z architekturą sterownika z rysunku 6.14 przedstawią rysunki 6.20 i 6.21. Postępowanie z nimi jest analogiczne do postępowania z pętlą wewnętrzną – najpierw należy dokonać linearyzacji, a następnie kształtowania pętli. Jednak zważywszy, że sam model quadrotora został ukształtowany w pętli wewnętrznej stosowany jest jedynie proporcjonalny prekompensator W_1, którego zadaniem jest ukształtowanie częstotliwości krzyżowania w okolicach 3 rad. Wykresy odpowiedzi częstotliwościowych systemu z zamkniętą wewnętrzną pętlą sprzężenia zwrotnego przed i po wstępnym kształtowaniu kompensatorem W_1 uzyskane po wykonaniu kodu z wydruku 6.15 widoczne są na rysunku 6.19.

Wydruk 6.16. Obliczenie parametrów sterownika H_∞ dla drugiej pętli zewnętrznej – linearyzacja pierwszej pętli zewnętrznej i wyznaczenie współczynników (appendix3ca.m)

```
% Step 1: Linearize the plant and analysis
% the singular values of the unshaped model
[x,u] = trim('quad22_work');
[A_4,B_4,C_4,D_4] = linmod('quad22_work',x,u);
G_4 = pck(A_4,B_4,C_4,D_4);
Gxy = sel(G_4,[2 3], [2 3]);
[Axy,Bxy,Cxy,Dxy] = unpck(Gxy);

% Step 2: Choose Tref
Tref_model = nd2sys(2^2,[1^2 2*1*1 1^2]);

Tref = daug(Tref_model,Tref_model);

figure(1);
```



Rysunek 6.20. Pierwsza zewnętrzna pętla sterowania dla sterownika H_∞ (`quad22.mdl`)

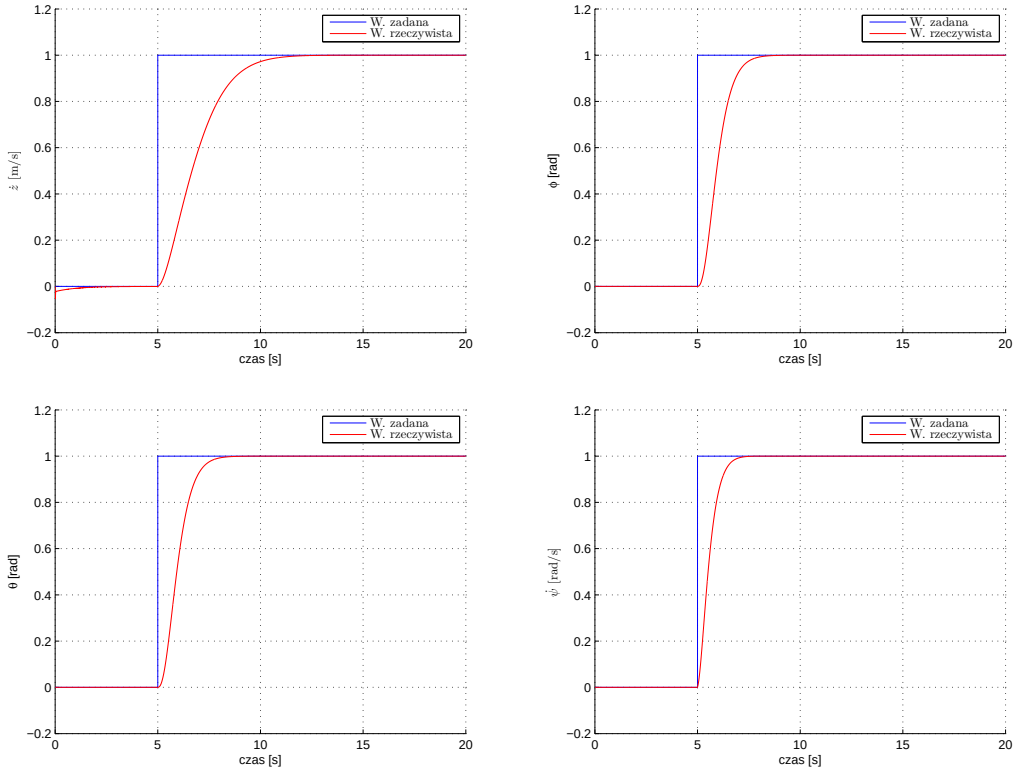


Rysunek 6.21. Druga zewnętrzna pętla sterowania dla sterownika H_∞ (`quad33.mdl`)

```
clf;
hold on;
sigma(Axy,Bxy,Cxy,Dxy,{0.01,1000})
sigma(Tref_model,{0.01,1000},'r')
title('Unshaped Plant Open Loop Singular Values with Referance Model')
grid


% Step 3: Synthesize 2-DOF controller Kxy and get gamma=3.0659
Kxy = hinf2dof(Gxy, Tref ) ;

% Step 4: Reduce the Kxy and seperate t o Kxyl and Kxy2
minfo(Kxy) % system: 102 states 2 outputs 4 input

% Reduct Kxy model to 20-states, 2 outputs and 4 inputs
[Kxy_cf,sigKxy] = sncfbal(Kxy);
Kxy_20 = cf2sys(hankmr(Kxy_cf,sigKxy,20,'d'));
gapKxy_20 = nugap(Kxy,Kxy_20) % gap is closed to zero

Kxy1 = sel(Kxy_20, [1 2], [1 2]);
Kxy2 = sel(Kxy_20, [1 2 ] , [3 4 ] ) ;
[K11A,K11B,K11C,K11D] = unpck(Kxy1);
[K12A,K12B,K12C,K12D] = unpck(Kxy2);
```

Analiza sterowania

Zarówno pętla wewnętrzna jak i pierwsza pętla zewnętrzna związane są bezpośrednio z algorytmem sterowania kształtowaniem pętli H_∞.



Rysunek 6.22. Odpowiedzi skokowe dla `quad11.mdl`

Pętla wewnętrzna odpowiada za sterowanie kątów kiwania i kołysania oraz stabilizację odchylenia i prędkości wznoszenia.

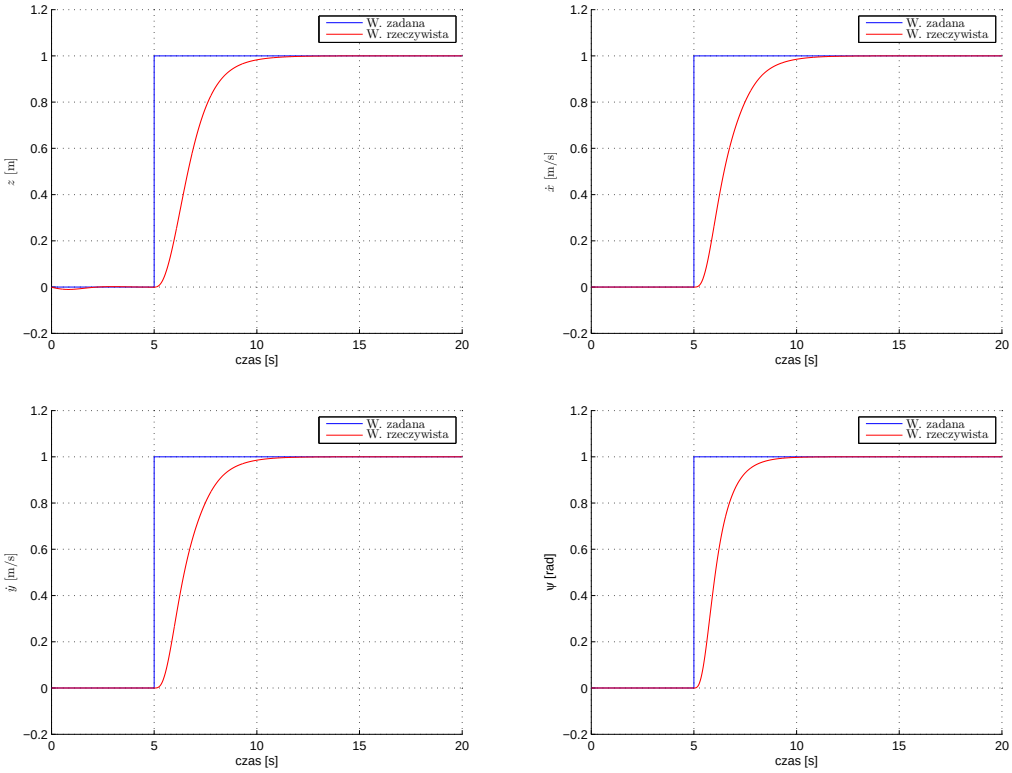
Pierwsza pętla zewnętrzna zapewnia kontrolę pozostałych prędkości (wzdłużnych i poprzecznych) oraz kontrolę kąta myszkowania.

Druga pętla zewnętrzna zapewnia kontrolę położenia. W jej miejscu można dołączyć przykładowo sterownik predykcyjny czy sterownik PID badając jakość regulacji we współpracy ze sterownikiem H_∞.

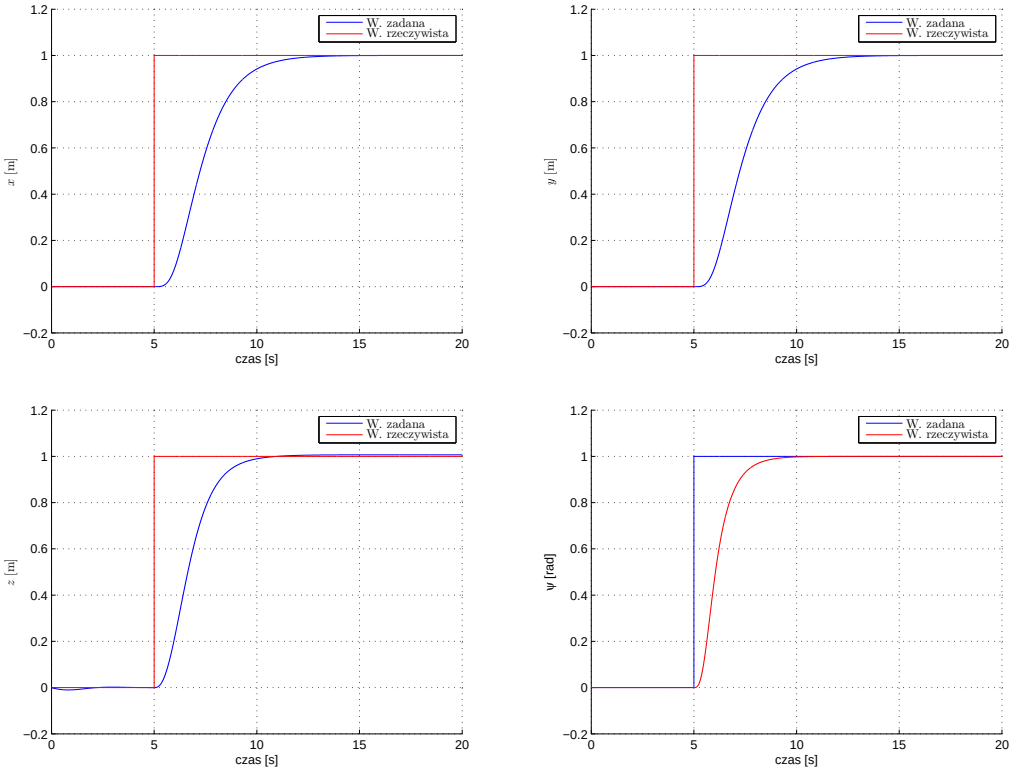
Odpowiedzi skokowe dla każdej pętli (`quad11.mdl`, `quad22.mdl`, `quad33.mdl`) przedstawiono na rysunkach 6.22-6.24.

Czas symulacji wynosił 20 sekund, skok o wartości 1 był zadawany w 5 sekundzie symulacji na wszystkie wejścia danej pętli. W przypadku pętli `quad22.mdl` i `quad33.mdl` konieczne okazało się zadanie skoku na kąt myszkowania osobno w stosunku do innych skoków. Podczas symulacji, w których wszystkie skoki były zadawane jednocześnie, układ destabilizował się – położenie x i y zaczynało oscylować w okolicach zadanej wartości. Jednak testy z mniejszymi skokami lub lekko opóźnionym skokiem dla kąta myszkowania nie wykazywały podobnych zjawisk.

Z uzyskanych odpowiedzi wynika, że quadrotor stabilizuje się maksymalnie w ciągu 7 sekund od zadanego wymuszenia na pożądanej wartości. W przypadku wcześniejszych pętli quadrotor stabilizuje się nawet dwukrotnie szybciej – po około 3 sekundach. Jednocześnie odpowiedzi skokowe nie przekraczają zadanej wartości – odpowiedź łagodnie dochodzi do wartości wymuszenia.



Rysunek 6.23. Odpowiedzi skokowe dla `quad22.mdl`



Rysunek 6.24. Odpowiedzi skokowe dla `quad33.mdl`

6.6. Interfejs użytkownika

6.6.1. Wprowadzenie

Nowoczesny użytkownik wymaga od współpracy z dowolnym środowiskiem prostoty i intuicyjności w jego obsłudze, tak aby w najszybszy możliwy sposób mógł uzyskać wymagane przez siebie wyniki. Sterowanie procesami zachodzącymi w środowisku MATLAB poprzez linię poleceń ma wiele zalet: jest to elastyczny sposób interakcji z programem, dający użytkownikowi największe możliwości. Wymaga jednak nauczenia się syntaktyki i semantyki języka skryptowego, przy czym ich znajomość nie gwarantuje uniknięcia błędów wynikających chociażby z niepoprawnego wprowadzenia danych czy też nazwy polecenia. Naturalnie rzutuje to na powtarzalność otrzymywanych wyników. W celu wyeliminowania podobnych problemów zdecydowano się stworzyć graficzny interfejs użytkownika (Graphical User Inteface - GUI) pozwalający na obsługę opisanych w niniejszej pracy funkcjonalności w sposób całkowicie powtarzalny i niezawodny.

6.6.2. Struktura GUI

Przy zaproponowanym podejściu całe GUI mieści się w jednym m-pliku. Można go podzielić na trzy wyraźne części:
1. Kod wykonywany podczas uruchomienia GUI
2. Deklaracje wszystkich obiektów widniejących w GUI
3. Wywołania zwrotne

Obecnie funkcjonalności GUI pozwalają na przeglądanie i modyfikację parametrów modelu oraz sterowań. Możliwe jest też wyświetlenie trajektorii quadrotora.

Kod wykonywany podczas uruchomienia GUI

Głównym celem tej części kodu jest wstępne przygotowanie danych w taki sposób, by mogły zostać wyświetlone. W tym celu niezbędne jest przejście ze struktur używanych przez grupy projektowe do tablic komórek. Proces ten, dla struktury QP zawierającej parametry modelu, realizuje poniższy fragment kodu:

Wydruk 6.17. Konwersja struktury QP

```
cQP=struct2cell(QP);
x=size(cQP);
N=x(1);

for i=1:N
fQP{i}=struct2cell(cQP{i});
val(i)=fQP{i}(1);
min_val(i)=fQP{i}(2);
max_val(i)=fQP{i}(3);
unit_sym(i)=fQP{i}(4);
desc(i)=fQP{i}(5);
end
```

W celu konwersji struktur do tablic komórek użyto polecenia *struct2cell*. Proces realizowany jest w pętli zależnej od ilości elementów struktury. Dzięki temu dodanie nowego parametru w którejkolwiek ze struktur nie pociąga za sobą potrzeby zmian w GUI. Poszczególne elementy struktury przypisywane są do tablic, które staną się kolumnami tablicy. Po konwersji, kolumny są łączone w jedną całość za pomocą polecenia *cat*. Następną czynnością jest ustalenie nazw kolumn, które wyświetlane będą w GUI, ustawienie formatu danych, które znajdować się będą w poszczególnych kolumnach oraz zaznaczenie które z nich będą mogły być edytowalne. Wszystkie te funkcje realizuje się przez stworzenie odpowiednich wektorów, które zostaną następnie przekazane do odpowiedniego obiektu GUI. W przypadku struktury QP wektory te przedstawiają się następująco:

Wydruk 6.18. Formatowanie tablic

```
columnname =   {'Val', 'Min_Val', 'Max_Val', 'Unit_sym','Opis'};
columnformat = {'numeric', 'numeric', 'numeric', 'char', 'char'};
columneditable =  [true false false false false];
```

Proces ten powtarzany jest dla wszystkich struktur, które mają zostać wyświetlone w tabelach.

Deklaracje obiektów

Deklaracje obiektów rozpoczynamy od stworzenia tła dla pozostałych obiektów. Jest nim zwykły obiekt *figure*. Przy tworzeniu możemy ustalić wymiary oraz własności tego obiektu, w szczególności możemy wyłączyć możliwość zmiany jego rozmiaru przez użytkownika. Wszystkie obiekty GUI tworzone są jako elementy jednej struktury, która jest potem wykorzystywana w wywołaniach zwrotnych. Deklaracja tła odbywa się w następujący sposób:

Wydruk 6.19. Deklaracja tła

```
S.fh =  figure('position',[0 0 1000 700],...
'menubar','none',...
'name','GUI',...
'numbertitle','off',...
'resize','off');
```

Na tym etapie ustalamy wszystkie parametry tworzonego obiektu, takie jak jego pozycja, nazwa, itp. Na stworzonej w ten sposób palecie umieszczane są następnie wszystkie pozostałe obiekty. W tym przypadku są to *uitable*, tworzący tablicę oraz *uicontrol* odpowiadający za wszelkiego rodzaju przyciski oraz pola tekstowe. W tym miejscu obiektom *uitable* przypisywane są też opisane w poprzedniej sekcji struktury danych w sposób przedstawiony poniżej.

Wydruk 6.20. Deklaracja tablicy

```
S.t = uitable('Units','normalized','Position',...
[0.33 0.56 0.67 0.3], 'Data', dat,...
'ColumnName', columnname,...
'ColumnFormat', columnformat,...
'ColumnEditable', columneditable,...
'ColumnWidth', {'auto' 'auto' 'auto' 'auto' 350},...
'RowName',[]);
```

Jedną z najważniejszych czynności dokonywanych w tej części kodu (z punktu widzenia funkcjonalności aplikacji) jest przypisanie poszczególnym przyciskom funkcji, które zostaną wykonane po ich przyciśnięciu. Poniżej znajduje się przykład kodu realizującego to zadanie dla przycisku 'Symuluj'.

Wydruk 6.21. Deklaracja przycisku

```
S.pb6 = uicontrol('style','push',...
'units','pix',...
'position',[85 330 180 40],...
'fontsize',14,...
'string','Symuluj',...
'callback',{@symuluj,S});
```

W parametrach obiektu ustawiamy własność 'push' (stworzony obiekt będzie typowym przyciskiem), nazwę przycisku i na końcu w sekcji 'callback' podajemy nazwę funkcji, która ma zostać wywołana po jego naciśnięciu wraz z parametrami, które mają zostać do niej przekazane (w tym przypadku jedynym parametrem jest struktura S w której zapisane są wszystkie obiekty graficzne przynależące do GUI, które zostały zadeklarowane do tego momentu).

Wywołania zwrotne

Aby zapewnić poprawne działanie interfejsu należy wszystkim elementom GUI, które mają spełniać konkretne zadania (przyciski, checkbox'y, itp.) przypisać odpowiednie funkcje. Taki mechanizm nazywany jest wywołaniem zwrotnym (callback). Poniżej przedstawiono przykładową funkcję dla przycisku 'Symuluj'.

Wydruk 6.22. Deklaracja funkcji

```
function [] = symuluj(varargin)
simuUni;
```

Funkcja ta uruchamia m-plik odpowiedzialny za poprawne przeprowadzenie symulacji w Simulinku. Należy zauważyć, że jedynym argumentem przyjmowanym przez funkcję jest *varargin*. Jest to struktura zawierająca wszystkie argumenty przekazane do funkcji podczas etapu tworzenia obiektu, któremu jest przypisana. W szczególności struktura zawiera też uchwyt do tego obiektu.

Ważnym problemem podczas tworzenia wywołań zwrotnych jest przekazywanie struktur danych. Niestety, jeśli podamy dane jako argument wywołania to zostaną one przekazane przez wartość i zmodyfikowane jedynie wewnątrz funkcji - oryginalne dane pozostaną bez zmian. Problem ten został rozwiązany przez użycie funkcji *setappdata*, która pozwala na wysłanie danych do przestrzeni powiązanej z konkretnym obiektem. Mogą one zostać następnie pobrane w dowolnym momencie za pomocą funkcji *getappdata*. Rozwiązanie to pozwala na przekazywanie danych pomiędzy poszczególnymi elementami GUI, jednak nie jest możliwe ich podglądanie co znacząco utrudnia proces tworzenia GUI. Planowane jest znalezienie rozwiązania tego problemu. Jednym z możliwych rozwiązań jest zastosowanie funkji *assignin* oraz *evalin*, które pozwalają na przesyłanie danych pomiędzy obszarami roboczymi.

6.6.3. Uruchamianie GUI

Uruchomienie GUI sprowadza się do wpisania w linii komend Matlaba polecenia *GUI* a co za tym idzie do do uruchomienia m-pliku *GUI.m*, zawierającego wstępną konfigurację potrzebną do poprawnego funkcjonowania aplikacji. Po kolei wykonywane są następujące czynności:

1. Oczyszczenie obszaru roboczego oraz zamknięcie niepotrzebnych okien
2. Uruchomienie m-plików odpowiedzialnych za wczytanie parametrów modelu
3. Uruchomienie m-plików odpowiedzialnych za wczytanie parametrów każdego z zaimplementowanych algorytmów
4. Uruchomienie symulacji tak, aby użytkownik mógł bez wprowadzenia żadnych zmian obserwować zachowanie układu dla domyślnych ustawień
5. Uruchomienie m-pliku GUI_1.m zawierającego właściwą aplikację

6.6.4. Struktura GUI z punktu widzenia użytkownika

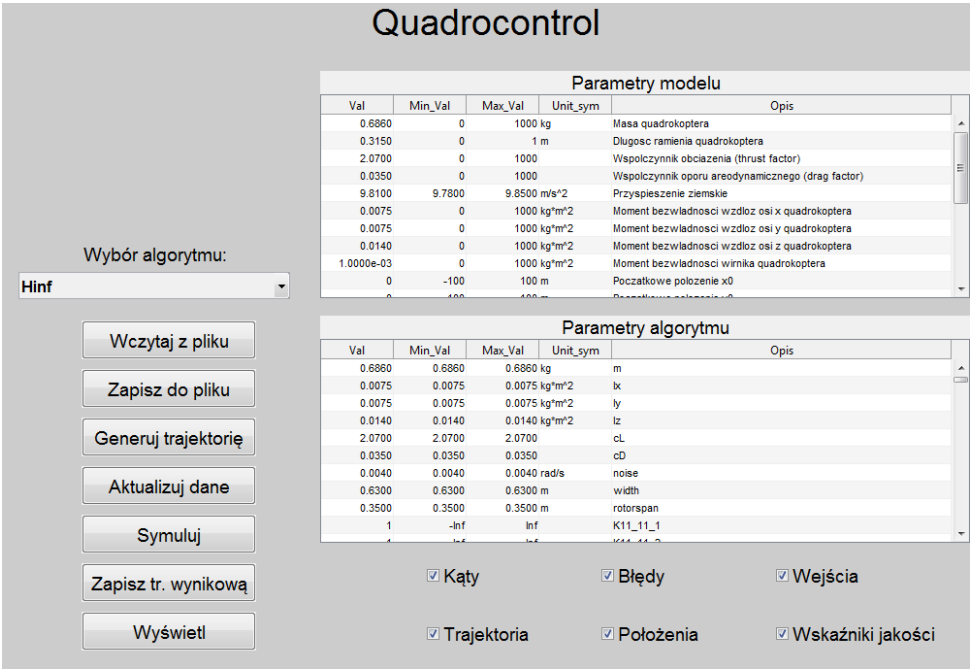
Po uruchomieniu oczom użytkownika okazuje się okno przedstawione na rys. 6.25.

Zgodnie z założeniami, najważniejsze parametry oraz funkcjonalności zgromadzone są w jednym oknie, co zwiększa przejrzystość oraz łatwość korzystania z interfejsu. Można wyróżnić kilka charakterystycznych obszarów okna. Zgromadzone w tych obszarach widżety spełniają podobne funkcje.

Parametry modelu oraz algorytmu

Największą część głównego okna zajmują dwie tabele zawierające parametry modelu (górna) oraz obecnie wybranego algorytmu (dolna). Każda z tabel zawiera następujące informacje o danym parametrze:

— Val - obecna wartość parametru



Rysunek 6.25. Ostateczny wygląd GUI

— Min_Val, Max_Val - wartość minimalna oraz maksymalna parametru
— Unit_sym - jednostka w jakiej wyrażony jest parametr
— Opis - krótki opis danego parametru

Jedyną modyfikowalną częścią tabeli jest kolumna obecnych wartości parametrów. Zmiana parametru w tabeli nie pociąga za sobą bezpośrednio jego zmiany w przestrzeni roboczej Matlaba - po dokonaniu zmian należy je zaakceptować za pomocą przycisku *Aktualizuj dane*. Wygląd tej części interfejsu dla algorytmu całkowania wstecznego przedstawia rys. 6.26

Pasek wyboru algorytmu

Do wyboru algorytmu służy rozwijalna lista przedstawiona na rys. 6.27.

Do wyboru mamy każdy z zaimplementowanych algorytmów (PID, backstepping, H_{inf}) oraz dodatkowo *brak algorytmu*. Zależnie od wybranego ustawienia zmieniać się będzie tabela z parametrami algorytmu. W szczególności tabela ta zniknie całkowicie w przypadku wyboru ostatniej pozycji. Ustawienie to pozwala na testowanie samego modelu poprzez zadawanie stałych wejść.

Zadajnik stałych wejść

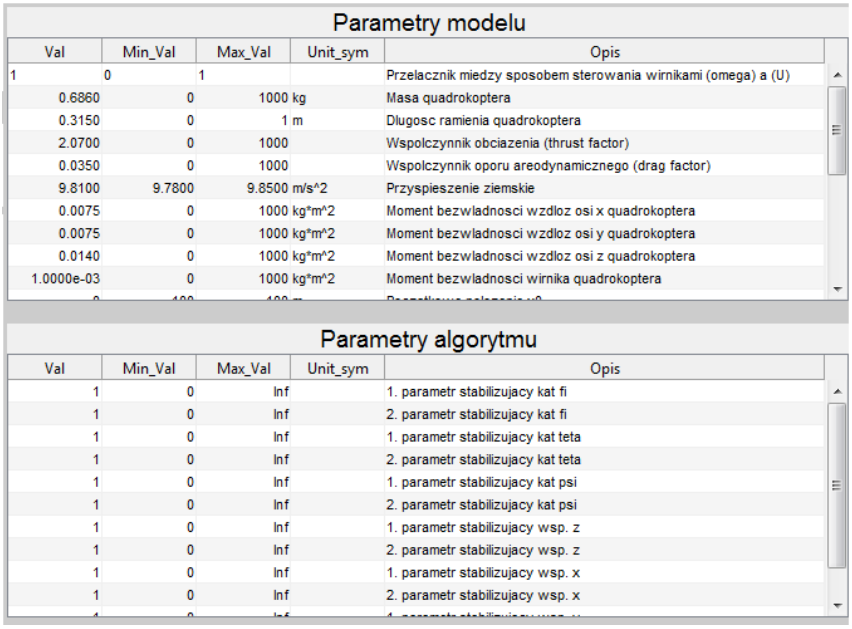
W lewym górnym rogu głównego okna GUI znajdują się dwie tabele, przedstawione na rys. 6.28.

W tych tablicach można wprowadzać stałe sterowania w przypadku, gdy przeprowadzamy testy na samym modelu, bez wykorzystania algorytmu. W innym przypadku nie są one wykorzystywane. W lewej tablicy znajdują się sterowania zdefiniowane za pomocą równania 2.30, natomiast w prawej tablicy sterowania zdefiniowane za pomocą równania 2.31.

Przyciski wyboru wizualizowanych elementów

Przyciski wyboru (checkbox'y) znajdujące się w dolnej części okna służą do wyboru wizualizowanych danych. Przedstawia je rys. 6.29.

Po zaznaczeniu konkretnego przycisku należy nacisnąć przycisk *Wyświetl* w celu uzyskania zamierzonego efektu. Pięć przycisków powoduje wyświetlenie wykresów (dwu- lub trójwymiaro-



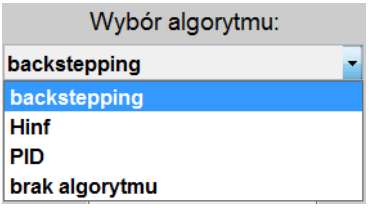
Parametry modelu

Val	Min_Val	Max_Val	Unit_sym	Opis
1	0	1		Przelacznik miedzy sposobem sterowania wirnikami (omega) a (U)
0.6860	0	1000	kg	Masa quadrokoptera
0.3150	0	1	m	Dlugosc ramienia quadrokoptera
2.0700	0	1000		Wspolczynnik obciazenia (thrust factor)
0.0350	0	1000		Wspolczynnik oporu areodynamicznego (drag factor)
9.8100	9.7800	9.8500	m/s^2	Przyspieszenie ziemskie
0.0075	0	1000	kg*m^2	Moment bezwladnosci wzdloz osi x quadrokoptera
0.0075	0	1000	kg*m^2	Moment bezwladnosci wzdloz osi y quadrokoptera
0.0140	0	1000	kg*m^2	Moment bezwladnosci wzdloz osi z quadrokoptera
1.0000e-03	0	1000	kg*m^2	Moment bezwladnosci wirnika quadrokoptera

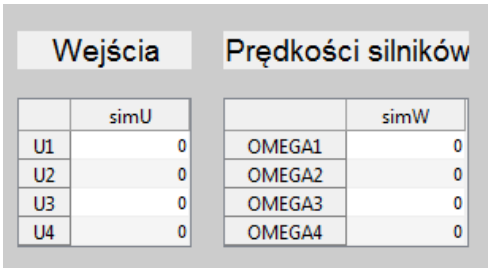
Parametry algorytmu

Val	Min_Val	Max_Val	Unit_sym	Opis
1	0	Inf		1. parametr stabilizujacy kat fi
1	0	Inf		2. parametr stabilizujacy kat fi
1	0	Inf		1. parametr stabilizujacy kat teta
1	0	Inf		2. parametr stabilizujacy kat teta
1	0	Inf		1. parametr stabilizujacy kat psi
1	0	Inf		2. parametr stabilizujacy kat psi
1	0	Inf		1. parametr stabilizujacy wsp. z
1	0	Inf		2. parametr stabilizujacy wsp. z
1	0	Inf		1. parametr stabilizujacy wsp. x
1	0	Inf		2. parametr stabilizujacy wsp. x

Rysunek 6.26. Parametry modelu i algorytmu



Rysunek 6.27. Lista wyboru algorytmów



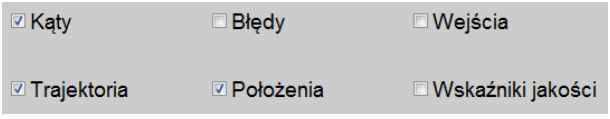
Wejścia

	simU
U1	0
U2	0
U3	0
U4	0

Prędkości silników

	simW
OMEGA1	0
OMEGA2	0
OMEGA3	0
OMEGA4	0

Rysunek 6.28. Zadajnik wejść modelu

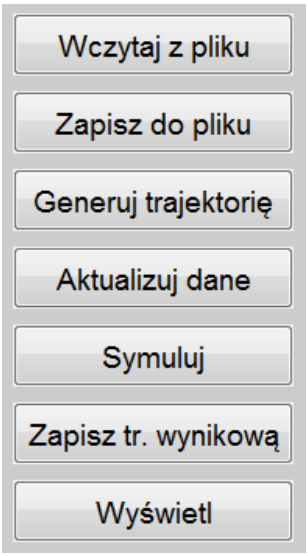


Rysunek 6.29. Przyciski wyboru

wych) zawierających kluczowe dla badanego informacje o przebiegu symulacji. Ostatni przycisk - *Wskaźniki jakości* - powoduje wyświetlenie osobnego okna.

Przyciski funkcyjne

Najważniejszym elementem GUI są przyciski funkcyjne przedstawione na rys. 6.30.



Rysunek 6.30. Przyciski funkcyjne

Przyciski te odpowiadają za poprawne wykonanie najważniejszych funkcji GUI. Są to kolejno:

— Wczytaj z pliku - powoduje zapisanie struktury GUIOut zawierającej najważniejsze informacje o symulacji do wybranego pliku
— Zapisz do pliku - powoduje wczytanie struktury GUIOut z wybranego pliku
— Generuj trajektorię - odpowiada za otworzenie osobnego okna z generatorem trajektorii
— Aktualizuj dane - powoduje aktualizacje danych w obszarze roboczym Matlaba
— Symuluj - uruchamia symulację na podstawie danych zapisanych w obszarze roboczym Matlaba
— Zapisz tr. wynikową - zapisuje trajektorię wyjściową symulacji jako trajektorię wejściową do następnych symulacji
— Wyświetl - powoduje wyświetlenie zaznaczonych za pomocą przycisków wyboru danych

Wskaźniki jakości

Rozwiązaniem proponowanym w celu zbadania jakości regulacji jest wystartowanie symulowanego quadrotora z tego samego punktu i śledzenie tej samej trajektorii przy użyciu poszczególnych algorytmów. Następnie możemy obserwować błędy śledzenia trajektorii ($e_x(t) = q_x(t) - q_x d(t)$, $e_y(t) = q_y(t) - q_y d(t)$, itd.) i na ich podstawie wnioskować o jakości regulacji. W przypadku zadania śledzenia trajektorii nie jest możliwe zastosowanie wszystkich wskaźników używanych powszechnie w automatyce, takich jak np. czas narastania czy czas regulacji. Można jednak wykorzystać takie wskaźniki jak:
— Błąd minimalny (najmniejszy z błędów śledzenia trajektorii na przestrzeni całej symulacji)
— Błąd maksymalny (największy z błędów śledzenia trajektorii na przestrzeni całej symulacji)
— Błąd średni wartości absolutnej błędu (średni błąd z całego przebiegu symulacji)
Do uzyskania całościowej informacji o procesie regulacji zostaną wykorzystane również poniższe wskaźniki całkowe:
— Całka z kwadratu uchybu (ISE - Integral Square Error)

$$\int_0^\infty e(t)^2 dt$$

Większą wagę mają duże błędy. Minimalizowanie ISE prowadzi do szybkiego eliminowania błędów ale tolerowane będą małe, długo trwające błędy. Może to powodować powstawanie znaczących oscylacji o niskich częstotliwościach.

— Całka z modułu uchybu (IAE - Integral Absolute Error)

$$\int_0^\infty |e(t)| dt$$

IAE nie wnosi kar zależnych od rozmiaru błędu. Minimalizowanie IAE prowadzi do wolniejszych odpowiedzi od ISE ale zwykle z mniejszymi oscylacjami.

— Całka z modułu pomnożonego przez czas (ITAE - Integral Time Absolute Error)

$$\int_0^\infty |e(t)| t dt$$

ITAE wprowadza kary za błędy pojawiające się później w czasie. Minimalizowanie ITAE prowadzi do szybkiego ustalania odpowiedzi za cenę powolnej reakcji w początkowych etapach.

Wskaźniki wyliczone dla przykładowej symulacji z wykorzystaniem algorytmu całkowania wstecznego przedstawia rys. 6.31

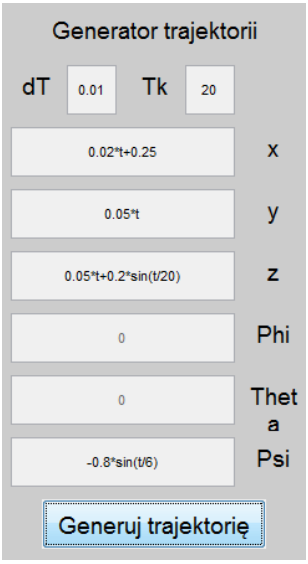
	MIN	MAX	MEAN	ISE	IAE	ITAE
x	-0.0296554	0.257944	0.0481364	12.6135	96.321	499.988
y	-0.060357	0.129145	0.0444164	5.89561	88.8772	845.572
z	-0.000936417	0.0264462	0.00227771	0.0842724	4.5577	6.79135
Phi	-0.99412	1.00878	0.628802	979.286	1258.23	12464.5
Theta	-0.999586	1.00585	0.629487	981.322	1259.6	12484.5
Psi	-0.0166243	0.00358671	0.00308311	0.0339398	6.16931	49.5759

Rysunek 6.31. Wskaźniki jakości

6.6.5. Generator trajektorii

Zadaniem postawionym przed generatorem trajektorii było poprawne generowanie dyskretnych trajektorii wprowadzanych przez użytkownika w sposób naturalny z zachowaniem składni zrozumiałej dla środowiska Matlab. W tym celu zaprojektowano okno przedstawione na rys. 6.32.

Użytkownik może wprowadzić dowolną akceptowaną przez algorytm funkcję zmiennej t w odpowiednim polu. Zmienne dT oraz Tk odpowiadają za utworzenie wektora czasu. Ma on postać $t = 0 : dT : Tk$. W celu odpowiedniego przetworzenia podanej przez użytkownika funkcji w środowisku Matlab jest ona zapisywana w zmiennej typu *string*. Poprawne zinterpretowanie



Rysunek 6.32. Generator trajektorii

tak zapisanej funkcji umożliwia polecenie *eval*. Pierwsze oraz drugie pochodne otrzymywane są z użyciem polecenia *diff*, które realizuje różniczkowanie symboliczne. Po wygenerowaniu odpowiednich wektorów są one zapisywane w przestrzeni roboczej Matlaba za pomocą polecenia *assignin*.

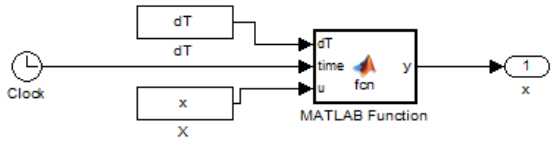
Wydruk 6.23. Generowanie trajektorii

```
dT=0.01;
Tk=20;

gx='0.02*t+0.25';
gy='0.05*t';
gz='0.05*t+0.2*sin(t/20)';
gxp=diff(sym(gx));
gyp=diff(sym(gy));
gzp=diff(sym(gz));

t=0:dT:Tk;

x=eval(gx);
y=eval(gy);
z=eval(gz);
xp=eval(gxp);
yp=eval(gyp);
zp=eval(gzp);

assignin('base','x',x);
assignin('base','y',y);
assignin('base','z',z);
assignin('base','xp',xp);
assignin('base','yp',yp);
assignin('base','zp',zp);
```

Największym problemem okazało się odpowiednie przetworzenie wygenerowanych trajektorii w Simulinku. Ostatecznie zastosowano następujące rozwiązanie:

— Wektor zawierający trajektorię, obecny czas oraz krok czasu (dT) podawane są w Simulinku na wejście bloczka *MATLAB Function*
— Na podstawie obecnego czasu i kroku czasu wyliczany indeks elementu, który należy pobrać z wektora
— Pobrana z wektora wartość podawana jest na wyjście

Schemat blokowy realizujący w Simulinku ten proces przedstawia rys. 6.33.



Rysunek 6.33. Generator - schemat w Simulinku dla trajektorii x

Kod zawarty w *MATLAB Function* odpowiadający za generowanie odpowiedniego indeksu przedstawia poniższy wydruk.

Wydruk 6.24. Obliczanie indeksu dla trajektorii x

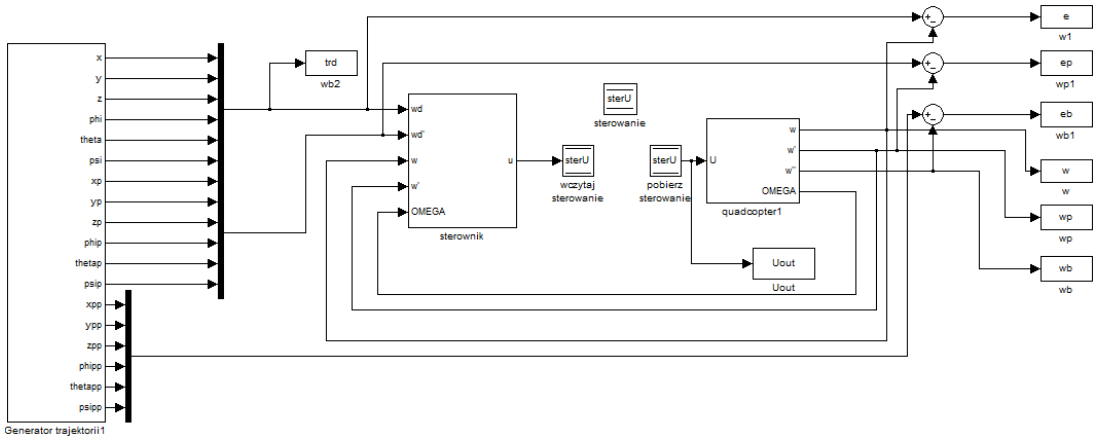
```
function y = fcn(dT,time,u)

n=size(u);
if n(2)>1
    index=round(time/dT)+1;
    y = u(index);
else
    y=u;
end
```

6.6.6. Połączenie algorytmu z GUI

W celu połączenia konkretnego algorytmu z GUI spełnione muszą być dwa warunki:

1. Algorytm musi współpracować z opisanym w poprzednim rozdziale generatorem (lub z innym przyjmującym wygenerowane wektory)
2. Odpowiednie sygnały muszą zostać wyprowadzone do obszaru roboczego Matlaba. Są to:
   — wyjścia obiektu
   — sygnały błędu
   — sygnały sterujące

   Przykład modelu podłączonego do GUI przedstawia rys. 6.34



Rysunek 6.34. Podłączenie do GUI modelu algorytmu całkowania wstecznego.

Wszystkie dane uzyskane w ten sposób są po zakończeniu symulacji zapisywane w jednej strukturze o nazwie *GUIOut*, która następnie zapisywana jest w obszarze roboczym matlaba za pomocą polecenia *assignin* . Proces ten obrazuje poniższy wydruk.

Wydruk 6.25. Tworzenie struktury GUIOut

```
GUIOut.x=w(:,1); GUIOut.y=w(:,2); GUIOut.z=w(:,3);
GUIOut.phi=w(:,4); GUIOut.theta=w(:,5); GUIOut.psi=w(:,6);
GUIOut.xp=wp(:,1); GUIOut.yp=wp(:,2); GUIOut.zp=wp(:,3);
GUIOut.phip=wp(:,4); GUIOut.thetap=wp(:,5); GUIOut.psip=wp(:,6);
GUIOut.xb=wb(:,1); GUIOut.yb=wb(:,2); GUIOut.zb=wb(:,3);
GUIOut.phib=wb(:,4); GUIOut.thetab=wb(:,5); GUIOut.psib=wb(:,6);

GUIOut.ex=e(:,1); GUIOut.ey=e(:,2); GUIOut.ez=e(:,3);
GUIOut.ephi=e(:,4); GUIOut.etheta=e(:,5); GUIOut.epsi=e(:,6);
GUIOut.exp=ep(:,1); GUIOut.eyp=ep(:,2); GUIOut.ezp=ep(:,3);
GUIOut.ephip=ep(:,4); GUIOut.ethetap=ep(:,5); GUIOut.epsip=ep(:,6);
GUIOut.exb=eb(:,1); GUIOut.eyb=eb(:,2); GUIOut.ezb=eb(:,3);
GUIOut.ephib=eb(:,4); GUIOut.ethetab=eb(:,5); GUIOut.epsib=eb(:,6);
GUIOut.t=tt;

GUIOut.xd=trd(:,1); GUIOut.yd=trd(:,2); GUIOut.zd=trd(:,3);
GUIOut.phid=trd(:,4); GUIOut.thetad=trd(:,5); GUIOut.psid=trd(:,6);

GUIOut.u1=Uout(:,1); GUIOut.u2=Uout(:,2); GUIOut.u3=Uout(:,3);
 GUIOut.u4=Uout(:,4);

assignin('base','GUIOut',GUIOut);
```

Elementy tak przygotowanej struktury są następnie wyświetlane za pomocą poleceń *plot* oraz *plot3* w wywołaniu zwrotnym zdefiniowanym dla przycisku *Wyświetl*.

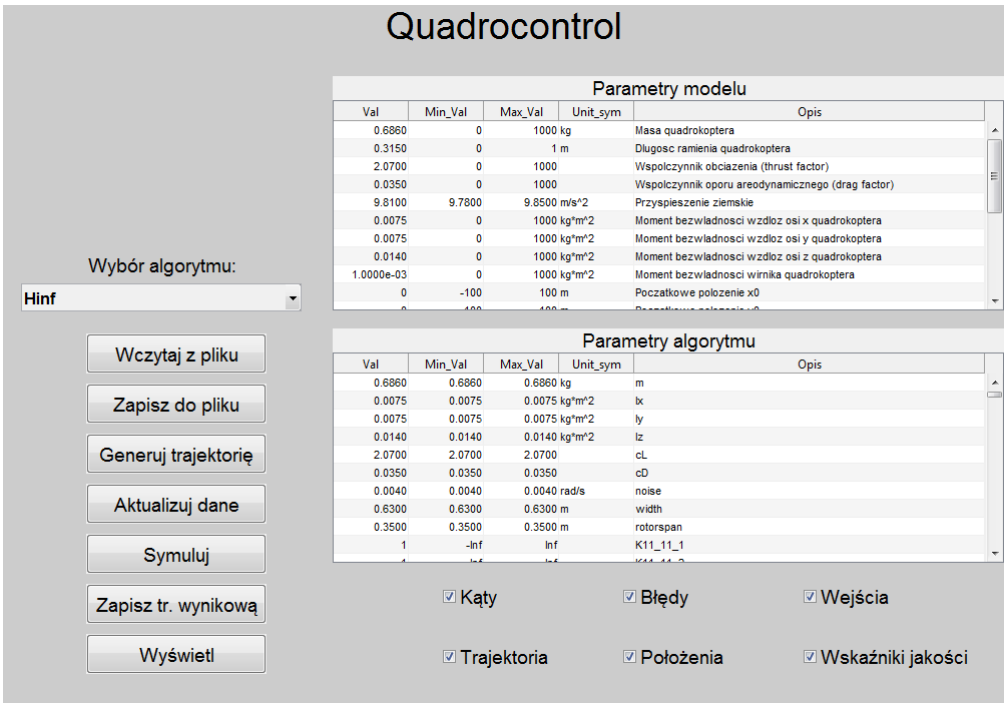
6.7. Obsługa interfejsu użytkownika

6.7.1. Uruchamianie GUI

Aby rozpocząć pracę z graficznym interfejsem użytkownika należy w wierszu poleceń programu MATLAB wprowadzić polecenie:

```
>> GUI
```

W wyniku tej operacji powiiniśmy otrzymać okno robocze zgodne z tym przedstawionym na rysunku 6.35.



Rysunek 6.35. Graficzny interfejs użytkownika

6.7.2. Ustawienia parametrów modelu quadrotora

Aby dostosowac model do potrzeb symulacji należy w tabeli *Parametry modelu*, przedstawionej na rysunku 6.36, dobrać odpowiednio wartości parametrów. Poszczególne wartości wprowadza się w kolumnie *Val* oznaczonej na rysunku czerwoną elpisą. Z kolejnych kolumn można odczytać przedział wartości w którym powinien znajdować się dany parametr:

— *Min_Val* oznacza wartość minimalną,
— *Max_Val* oznacza wartość maksymalną.

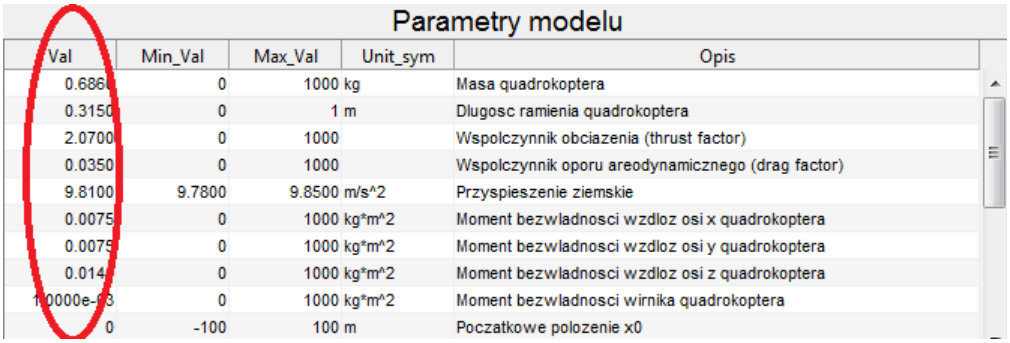
Następna kolumna zawiera jednostki w jakich wyrażają się parametry, a ostatnia kolumna dostarcza słownego opisu danego parametru.

Należy pamiętać aby po wprowadzeniu wszystkich wymaganych zmian, nacisnąć przycisk *Aktualizuj dane* (rysunek 6.37), aby zmiany stały się widoczne dla planowanej symulacji.

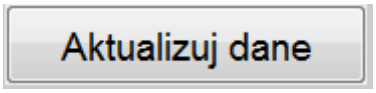
6.7.3. Wybór algorytmu sterowania

Aby wybrać odpowiedni algorytm sterowania, należy w rozwijanym menu (rysunki 6.38 i 6.39) wybrać odpowiednią pozycję:

— *backstepping*, dla algorytmu całkowania wstecznego,

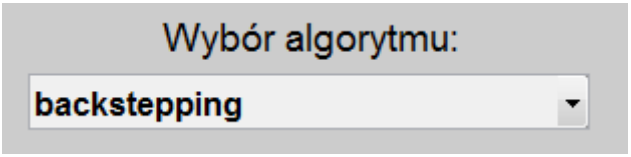


Rysunek 6.36. Parametry modelu — przedstawienie danych



Rysunek 6.37. Przycisk aktualizujący ustawienia modelu i algorytmów sterowania

— *Hinf*, dla algorytmu H_{inf},
— *PID*, dla algorytmu PID,
— *brak algorytmu*, aby umożliwić symulację na modelu quadrotora pozbawionym algorytmu sterowania



Rysunek 6.38. Rozwijane menu wyboru algorytmu sterowania — zwinięte

6.7.4. Ustawienia parametrów algorytmu sterowania

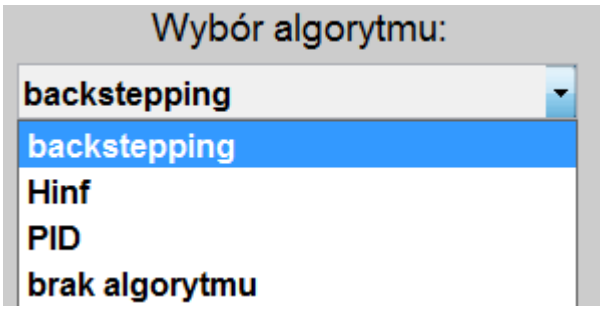
Aby dostosowac algorytm sterowania do potrzeb symulacji należy w tabeli *Parametry algorytmu*, przedstawionej na rysunku 6.40, dobrać odpowiednio wartości parametrów. Poszczególne wartości wprowadza się w kolumnie *Val* oznaczonej na rysunku czerwoną elipsą. Z kolejnych kolumn można odczytać przedział wartości w którym powinien znajdować się dany parametr:
— *Min_Val* oznacza wartość minimalną,
— *Max_Val* oznacza wartość maksymalną.
Następna kolumna zawiera jednostki w jakich wyrażają się parametry, a ostatnia kolumna dostarcza słownego opisu danego parametru.

W przypadku wyboru opcji *brak algorytmu* w menu wyboru algorytmu, tabela *Parametry algorytmu* zostaje ukryta, a zamiast niej wyświetlone zostają tabele *Wejścia* i *Prędkości silników* (rysunek 6.41), pozwalające na zadanie stałych sterowanań na modelu.

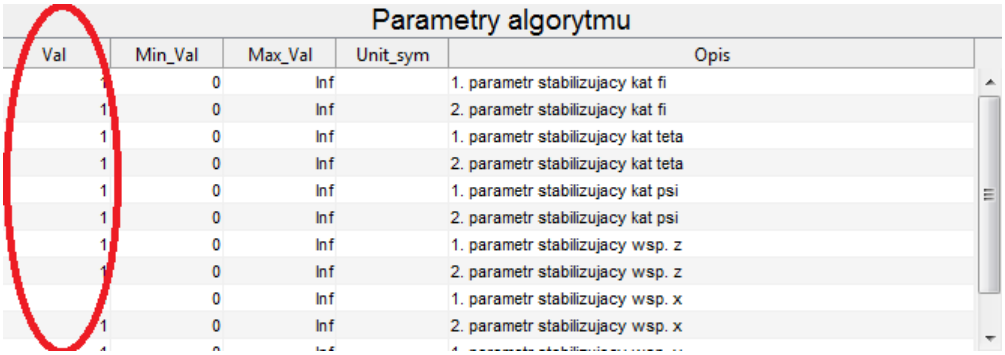
Należy pamiętać aby po wprowadzeniu wszystkich wymaganych zmian, nacisnąć przycisk *Aktualizuj dane* (rysunek 6.37), aby zmiany stały się widoczne dla planowanej symulacji.

6.7.5. Generowanie trajektorii zadanej

Jeżeli istnieje potrzeba uruchomienia symulacji dla trajektorii domyślnej, można pominąć ten krok. Aby wygenerować trajektorię zadaną dla modelu quadrotora należy wcisnąć przycisk *Generuj trajektorię* (rysunek 6.42). Spowoduje te otworzenie nowego okna (rysunek 6.43) oraz



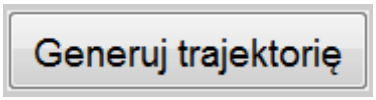
Rysunek 6.39. Rozwijane menu wyboru algorytmu sterowania — rozwinięte



Rysunek 6.40. Parametry algorytmu — przedstawienie danych

Rysunek 6.41. Ustawienia wejść modelu, w przypadku wyboru braku algorytmu sterowania

zablokowanie paska wyboru algorytmu i przycisku *Generuj trajektorię*. Odblokowane one zostaną dopiero po zamknięciu okna generatora. Okno generatora umożliwia wprowadzenie w polu *dT*

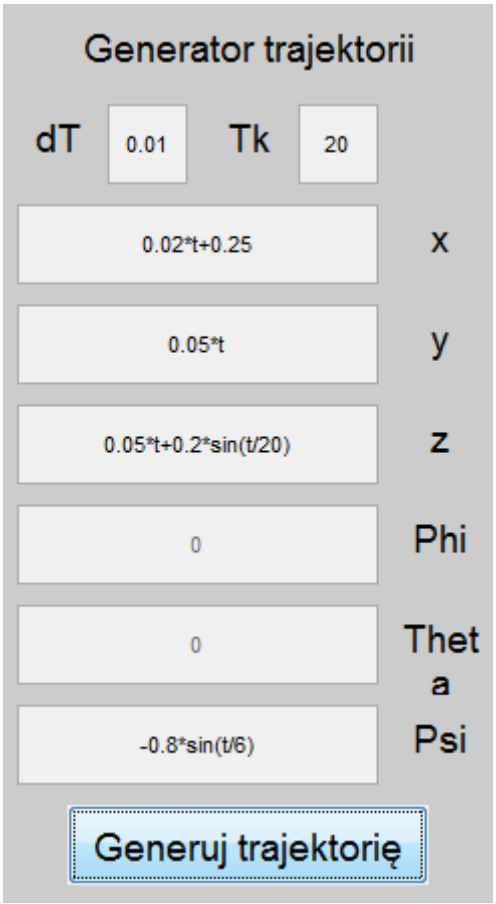


Rysunek 6.42. Przycisk otwierający okno generatora trajektorii

kroku symulacji, w polu *Tk* czasu symulacji, a w pozostałych polach funkcji czasu opisujących trajektorię zadaną dla poszczególnych zmiennych stanu (*x y z Phi Theta Psi*). W oknie aktywne poza *Tk* i *dT* pozostają jedynie te pola, które odpowiadają współrzędnym stanu sterowanym bezpośrednio przez algorytm sterowania.

Funkcje opisujące generowaną trajektorię należy wprowadzac w notacji programu MATLAB, w zależności od zmiennej *t* opisującej czas. Przykład również na rysunku 6.43.

Dokonane zmiany należy potwierdzić przyciskiem *Generuj trajektorię*, aby generator wytworzył wszystkie wymagane przez symulację wektory.

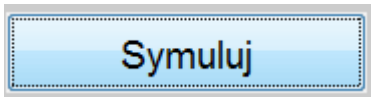


Rysunek 6.43. okno generatora trajektorii

6.7.6. Uruchomienie symulacji

Aby uruchomić symulację dla zadanych parametrów i trajektorii, należy wcisnąć przycisk *Symuluj* (rysunek 6.44). Na czas trwania symulacji zostanie on zablokowany, aby uchronić użytkownika, przed uruchomieniem kilku symulacji jednocześnie oraz zmieni swój wygląd (rysunek

6.45). Przycisk powróci do swojego normalnego wygladu po zakończeniu symulacji. Po wykonaniu symulacji można zapisać trajektorię wynikową quadrotora w celu wykorzystania jej do dalszych symulacji za pomocą przycisku *Zapisz tr. wynikową*. Należy uważać aby nie uruchomić po tej czynności generatora trajektorii, gdyż wtedy zapisana trajektoria wynikowa zostanie nadpisana.



Rysunek 6.44. Przycisk uruchamiający symulację

Rysunek 6.45. Przycisk uruchamiający symulację w trakcie pracy

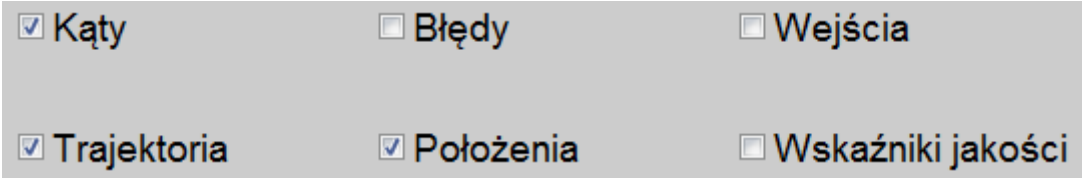
6.7.7. Prezentacja wyników

Wyniki symulacji można obejrzeć poprzez naciśnięcie przycisku *Wyświetl* (rysunek 6.46), co spowoduje prezentację wybranych za pomocą przycisków wyboru (rysunek 6.47) wykresów (rysunek 6.48), bądź okna prezentującego wskaźniki jakości.

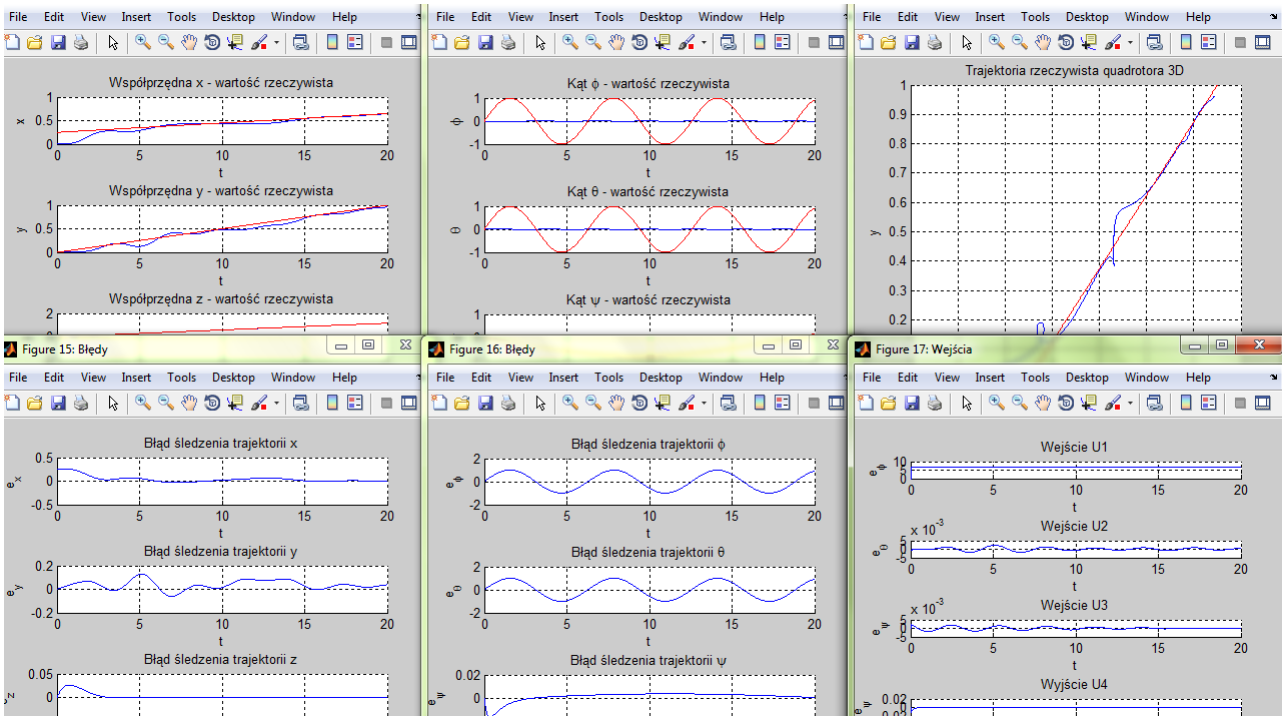
Rysunek 6.46. Przycisk uruchamiający wizualizację wyników

6.7.8. Zapisywanie i odczytywanie wyników

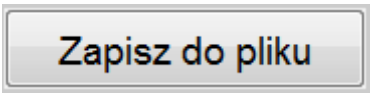
Aby móc zachować otrzymane w trakcie symulacji wyniki do dalszej pracy należy użyć przycisku *Zapisz do pliku* (rysunek 6.49). Spowoduje to otworzenie okna menadżera (rysunek 6.51), który pozwoli zapisać plik w dowolnym miejscu. Aby otworzyć plik z danymi i wykorzystać je w pracy z programem należy użyć przycisku *Wczytaj z pliku* (rysunek 6.50). Spowoduje to otworzenie okna menadżera (rysunek 6.52), który pozwoli wczytać zapisany plik do pamięci programu.



Rysunek 6.47. Przyciski wyboru pozwalające na wybór wizualizowanych danych

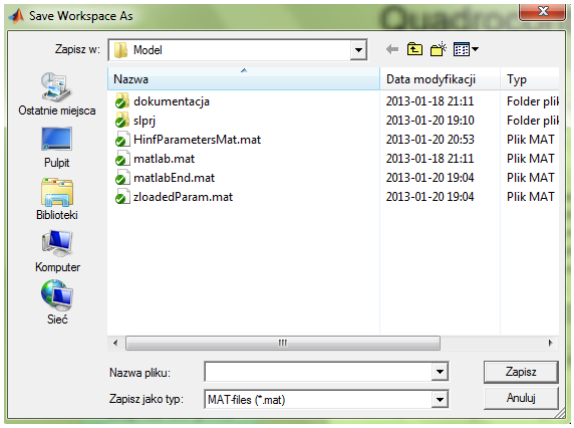


Rysunek 6.48. Zbiorcze przedstawienie generowanych wykresów

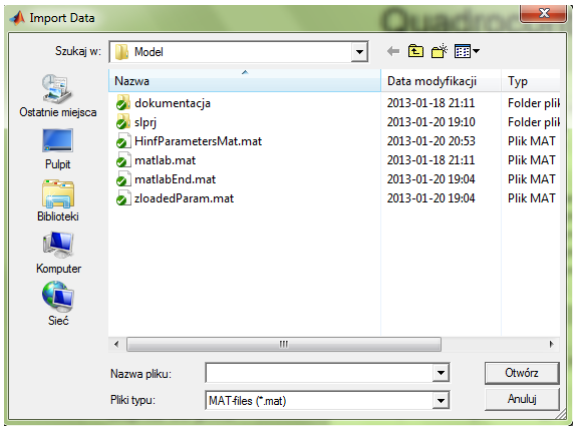


Rysunek 6.49. Przycisk pozwalający na zapis uzyskanych danych

Rysunek 6.50. Przycisk pozwalający na wczytanie zapisanych danych



Rysunek 6.51. Menadżer pozwalający zapisać plik wynikowy symulacji



Rysunek 6.52. Menadżer pozwalający wczytać zapisany plik wynikowy symulacji

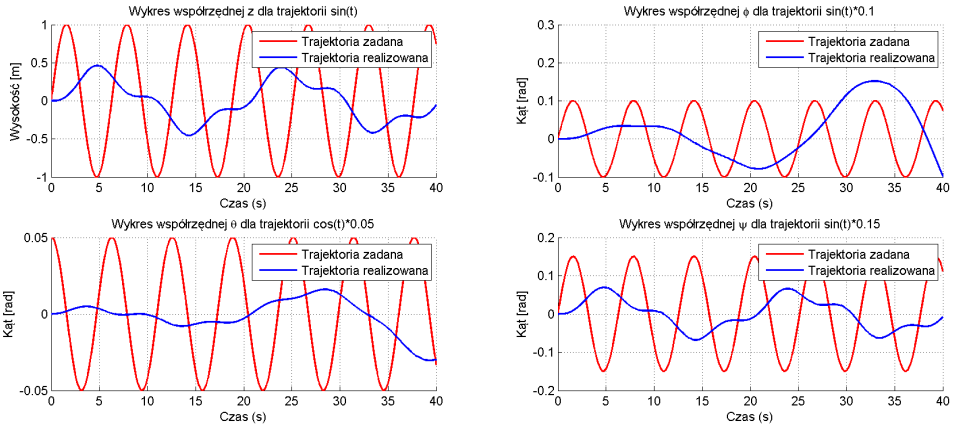
7. Badania

7.1. Regulator PID

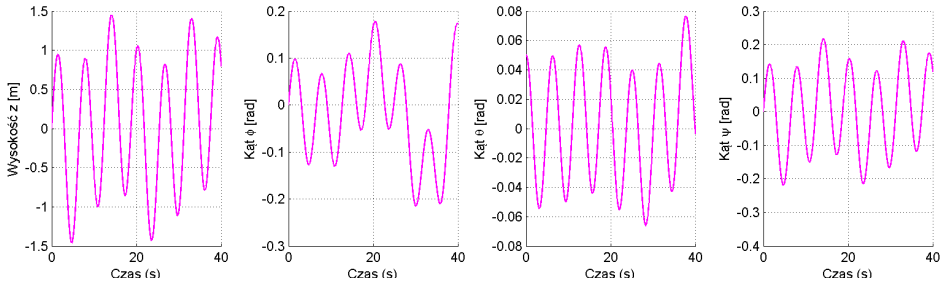
Pierwszy etap przeprowadzonych badań dotyczył ustalenia wpływu poszczególnych parametrów na zachowanie się sterownika w zadaniu śledzenia trajektorii sinusoidalnych. Poniżej autorzy starali się przedstawić pewne zauważone zależności. Potwierdzają się one w odniesieniu do badanego obiektu robota typu quadrotor nie tylko na zaprezentowanych trajektoriach, ale w szerszym kontekście opisują jego zachowanie w realizacji zadań śledzenia trajektorii i stabilizacji do punktu. W poniższych badaniach poszczególnych członów przyjęto takie same wartości parametrów wzmocnienia proporconalnego, całkującego i różniczkującego we wszystkich czterech regulatorach. Dokonano tego, aby pokazać trend identyczny dla wszystkich współrzędnych, który został zauważony w czasie wstępnych badań. W celu bardzo dokładnej realizacji zadania śledzenia trajektorii zalecą się odpowiednią modyfikację poszczególnych nastawy w zależności od otrzymywanych wyników.

7.1.1. Człon proporcjonalny P

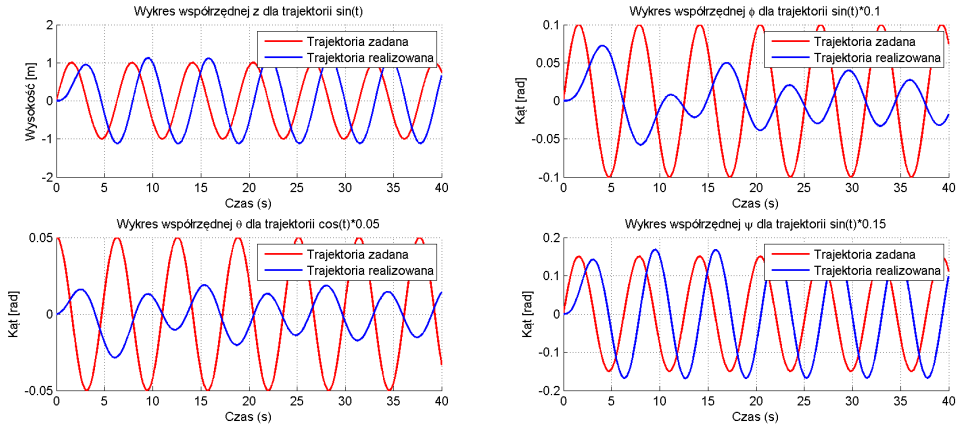
Badano zachowanie modelu quadrotora w zależności od zmiany wszystkich 4 parametrów K_p dla współrzędnych z, ϕ, θ i ψ przy wartościach parametrów K_i=0.1 i K_d=1 (w legendzie rysunków parametry są oznaczone kolejno jako P, I oraz D). Nie obserwowano wartości zadawanych sterowań – dopiero po dobraniu optymalnych nastaw stwierdzono, iż są one fizycznie realizowalne. Przyjmowano wartości różniące się rzędem wielkości; jako podstawę ustalono współczynnik wzmocnienia równy 10 (w oparciu o wstępne wyniki doświadczalne). Parametry zmieniano od wartości 0.1 do 1000. Nastawy porównywano dla tych samych trajektorii zadanych w celu zauważenia pewnego trendu. Rysunki przedstawiające wyniki doświadczeń dla parametrów K_p = 0.1 (7.1 i 7.2) oraz $k_p = 1$ (7.3 i 7.4) pozwalają stwierdzić, iż przy wzmocnieniu (K_p=1) lub jego obniżeniu obiekt nie jest w stanie śledzić zadanej trajektorii.



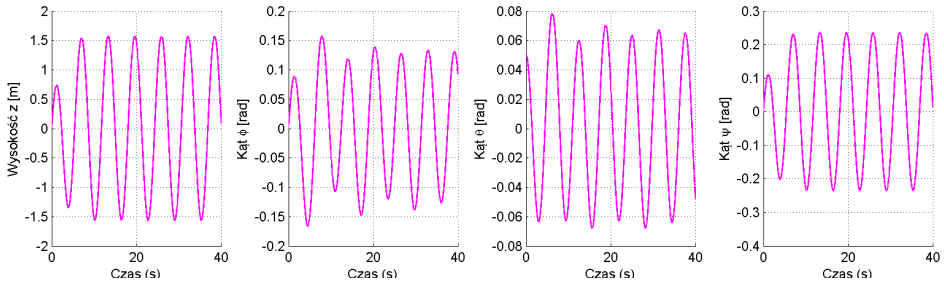
Rysunek 7.1. Śledzenie trajektorii dla parametrów K_p=0.1



Rysunek 7.2. Uchyb śledzenia trajektorii dla parametrów K_p=0.1

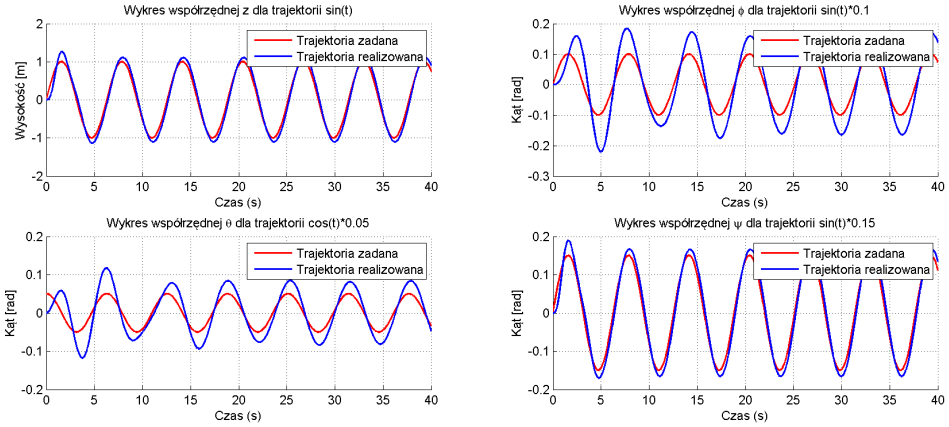


Rysunek 7.3. Śledzenie trajektorii dla parametrów K_p=1

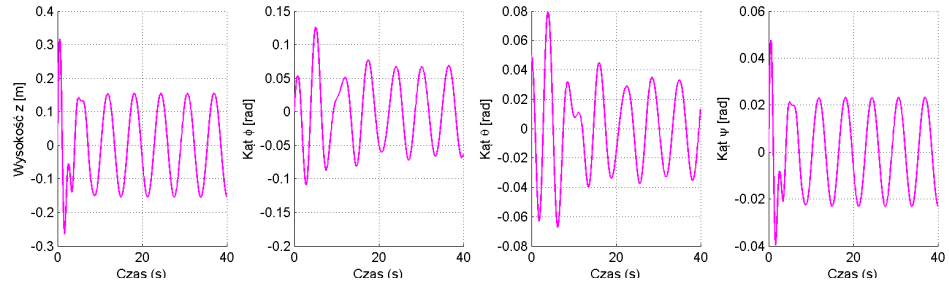


Rysunek 7.4. Uchyb śledzenia trajektorii dla parametrów K_p=1

Każdorazowe zwiększanie o rząd wielkości wzmocnienia członu proporcjonalnego powoduje dużo lepsze śledzenie trajektorii (szybsza zbieżność uchybu do 0). Prezentują to rysunki 7.5 i 7.6, przedstawiające śledzenie trajektorii dla parametru $K_p = 10$.

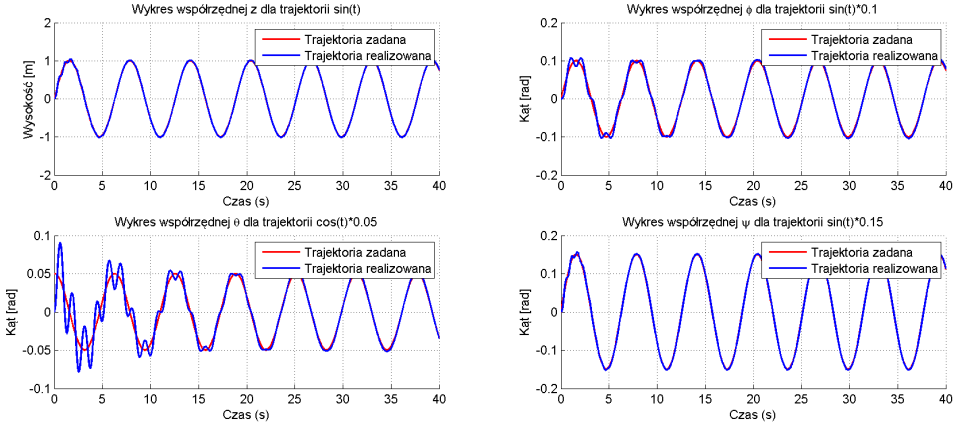


Rysunek 7.5. Śledzenie trajektorii dla parametrów K_p=10

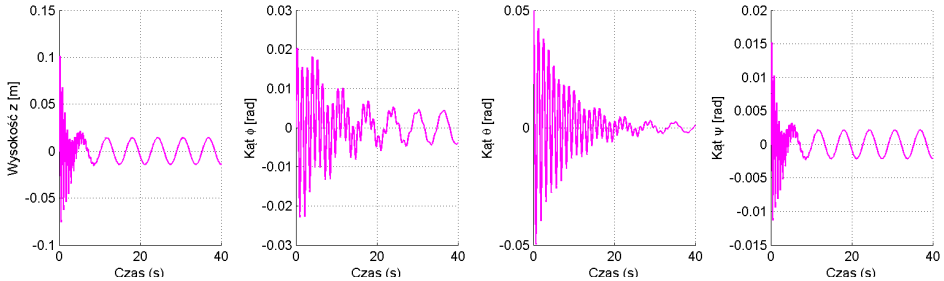


Rysunek 7.6. Uchyb śledzenia trajektorii dla parametrów K_p=10

Niestety, każdorazowe zwiększenie tegoż współczynnika znacząco wydłuża czas symulacji z powodu coraz większych wartości liczbowych, które są przemnażane przez siebie. Ponadto, duże wartości wzmocnień mogą nie być realizowane na fizycznym obiekcie. Należy zwrócić szczególną uwagę na pojawiające się oscylacje, które zmniejszają stabilność systemu. Efektem zbyt dużych nastaw są widoczne w pierwszych 10 sekundach przy nastawach K_p=100 przeregulowania, pokazane na rysunkach 7.7 i 7.8.

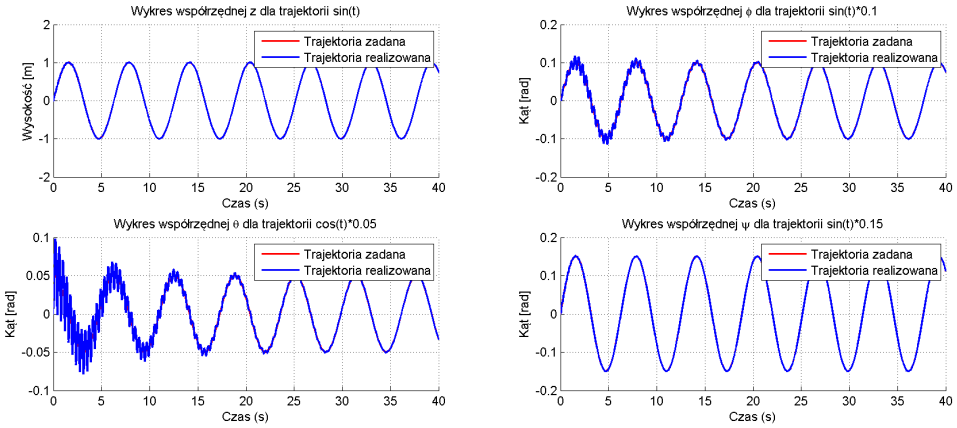


Rysunek 7.7. Śledzenie trajektorii dla parametrów K_p=100

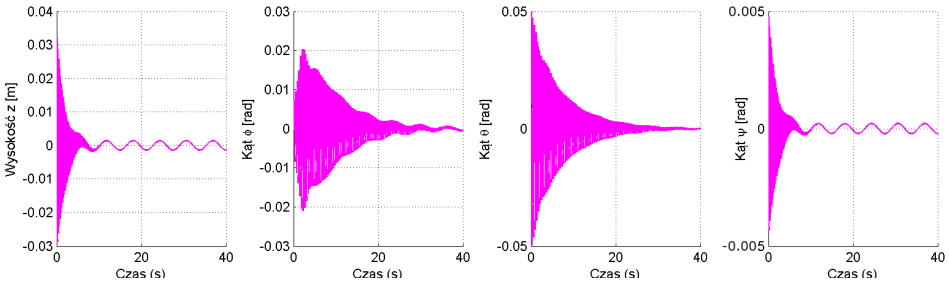


Rysunek 7.8. Uchyb śledzenia trajektorii dla parametrów K_p=100

W przypadku pracy z rzeczywistym układem takie zjawiska mogą powodować jego nieprawidłowe działanie, o czym koniecznie należy pamiętać. Dalsze zwiększanie parametrów do K_p = 1000 powoduje większe oscylacje, co zostało przedstawione na rysunkach 7.9 i 7.10.



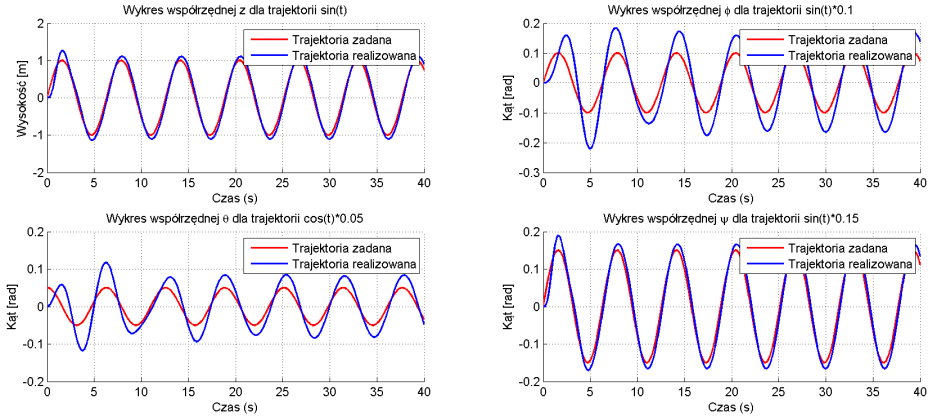
Rysunek 7.9. Śledzenie trajektorii dla parametrów K_p=1000



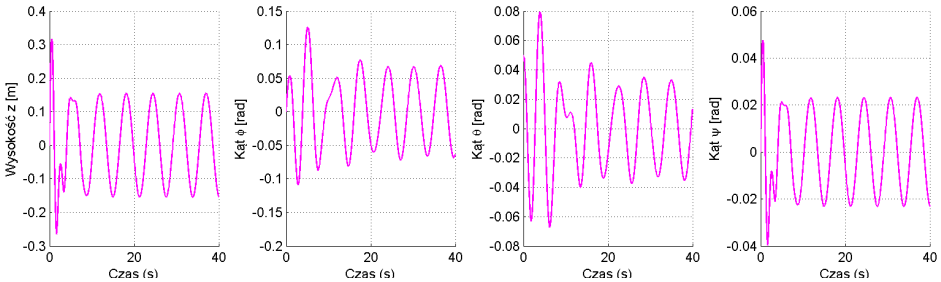
Rysunek 7.10. Uchyb śledzenia trajektorii dla parametrów K_p=1000

7.1.2. Człon całkujący I

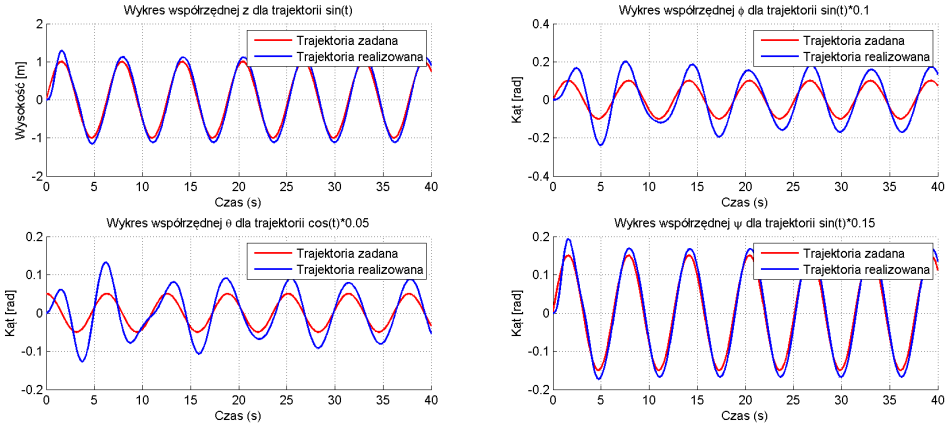
Wpływ zmian parametrów wzmocnień członów całkujących badano w analogiczny sposób. Zakres zmian parametru K_i ustalono na wartość od 0.01 do 100 na podstawie wstępnych badań obiektu, przy parametrach K_p=10 i K_d=1 (bezpiecznymi nastawami również znalezionymi we wstępnych badaniach). Wartości K_i¡1 nie różnią się znacząco otrzymanymi wynikami, dlatego też pominięto ukazanie wykresów dla wartości K_i=0.01. Wyniki doświadczeń pokazują, iż dla tych wartości współrzędna z i kąt ψ są bardzo dobrze śledzone, zaś pewne problemy pojawiają się dla kątów ϕ i θ. Pokazują to rysunki 7.11 i 7.12 dla K_i = 0.1 oraz rysunki 7.13 i 7.14 dla K_i = 1.



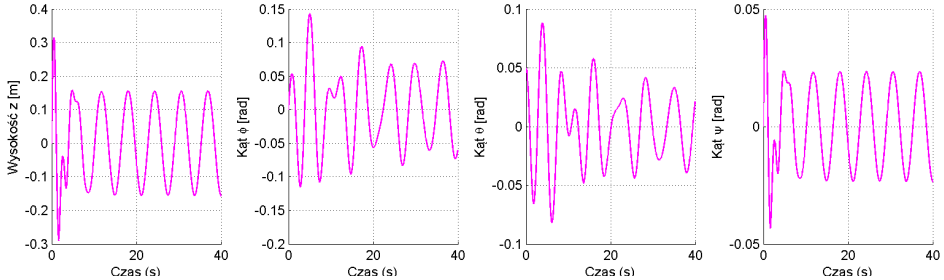
Rysunek 7.11. Śledzenie trajektorii dla parametrów K_i=0.1



Rysunek 7.12. Uchyb śledzenia trajektorii dla parametrów K_i=0.1

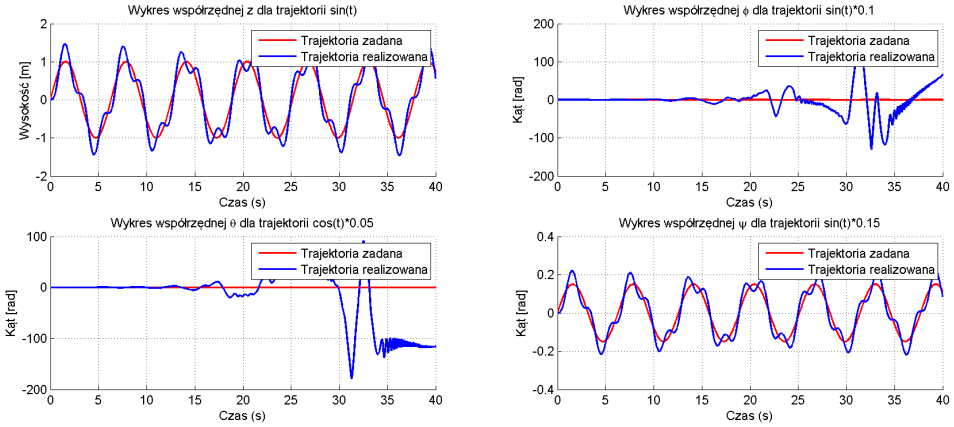


Rysunek 7.13. Śledzenie trajektorii dla parametrów K_i=1

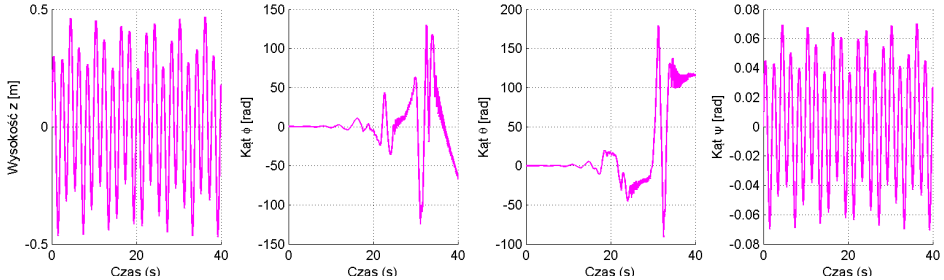


Rysunek 7.14. Uchyb śledzenia trajektorii dla parametrów K_i=1

Wartość $K_i = 10$ destabilizuje układ, powodując ruch wirowy wokół osi ϕ oraz θ. Podobnie jak w przypadku parametru K_p, duża wartość powoduje wzbudzenie się układu – ukazują to rysunki 7.15 i 7.16. Z tego powodu większe wartości K_i nie zostały sprawdzone.



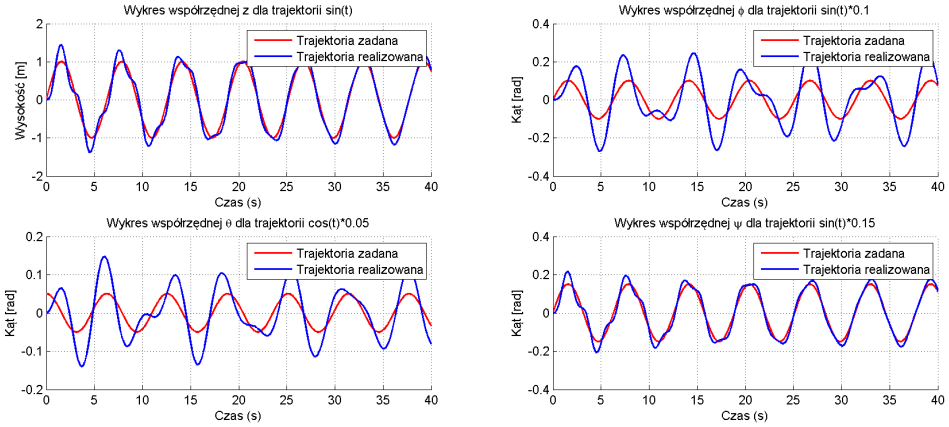
Rysunek 7.15. Śledzenie trajektorii dla parametrów K_i=10



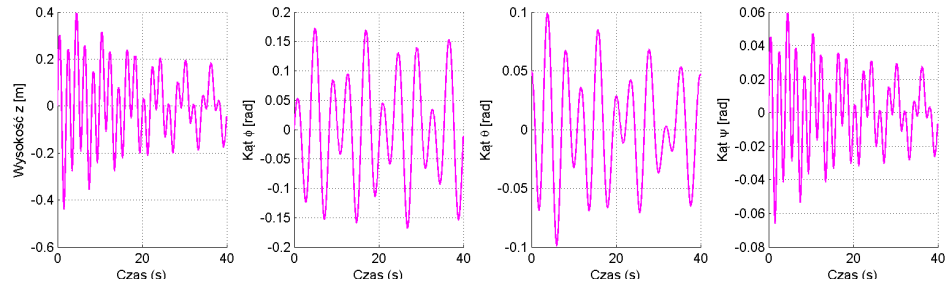
Rysunek 7.16. Uchyb śledzenia trajektorii dla parametrów K_i=10

7.1.3. Człon różniczkujący D

Parametry wzmocnień członów różniczkujących K_d rozpatrywano w zakresie od 0.1 do 1000 dla K_p=10 i K_i=0.1. Przy wartości K_d=0.1 układ nieprawidłowo śledził trajektorię dla kątów θ oraz ϕ, co przedstawiają rysunki 7.17 i 7.18.

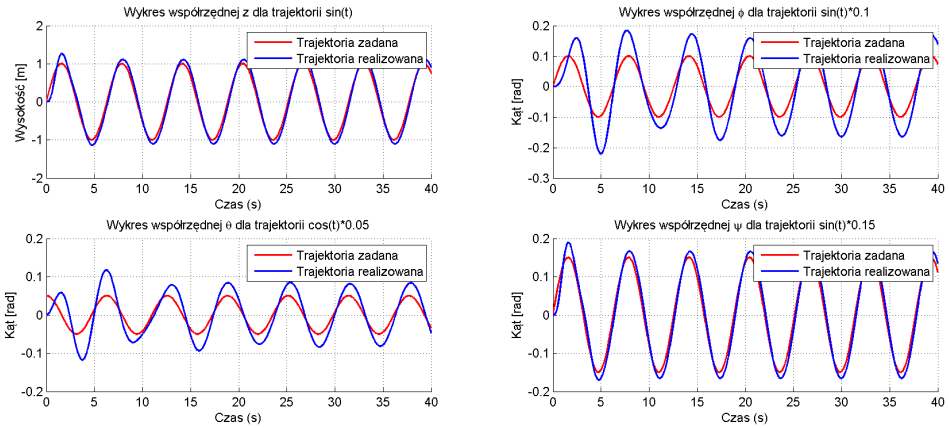


Rysunek 7.17. Śledzenie trajektorii dla parametrów K_d=0.1

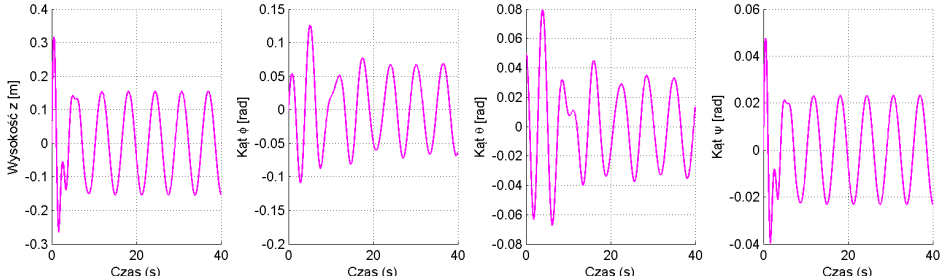


Rysunek 7.18. Uchyb śledzenia trajektorii dla parametrów K_d=0.1

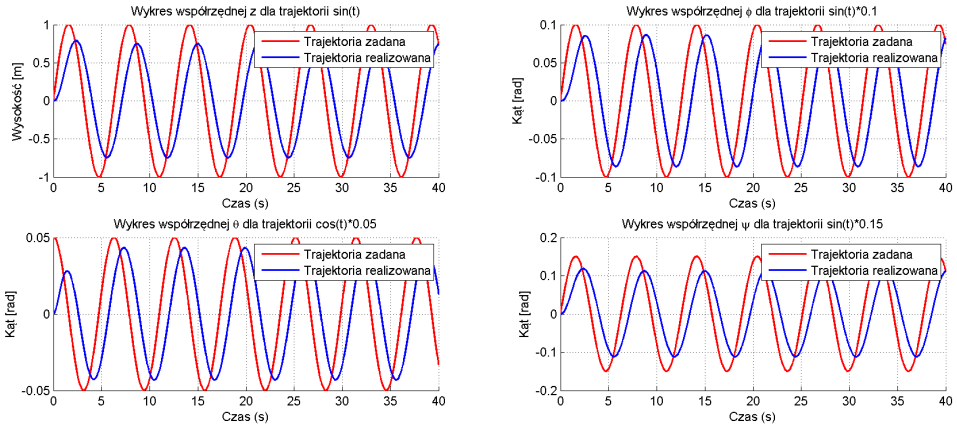
Zwiększenie parametru K_d o rząd wielkości umożliwiło polepszenie sterowania i śledzenia kątów. Niestety, dalsze zwiększanie parametru do wartości K_d=10 skutkuje zbyt dużą wartością odpowiedzi i w efekcie przeregulowaniem układu. Pokazują to rysunki 7.19 i 7.20 (dla K_d=1) i 7.21 i 7.22 (dla K_d=10).



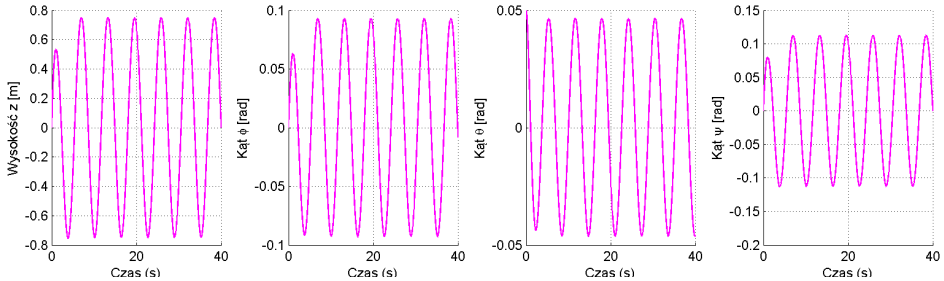
Rysunek 7.19. Śledzenie trajektorii dla parametrów K_d=1



Rysunek 7.20. Uchyb śledzenia trajektorii dla parametrów K_d=1

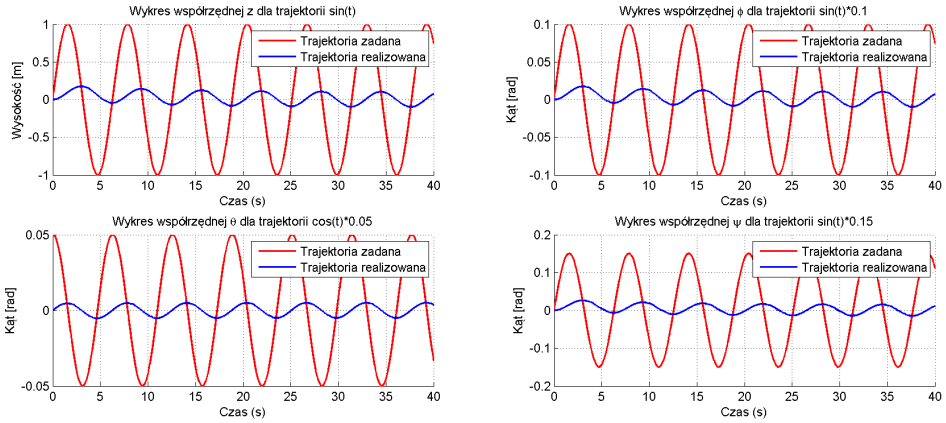


Rysunek 7.21. Śledzenie trajektorii dla parametrów K_d=10

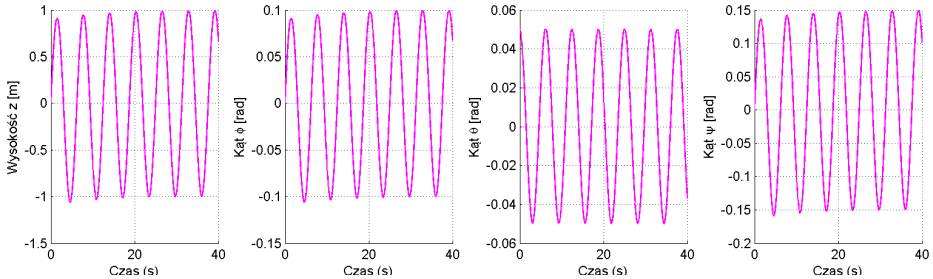


Rysunek 7.22. Uchyb śledzenia trajektorii dla parametrów K_d=10

Dalsza inkrementacja K_d jest bezcelowa, gdyż sterownik przestaje wówczas regulować układ co ukazują rysunki 7.23 i 7.24.



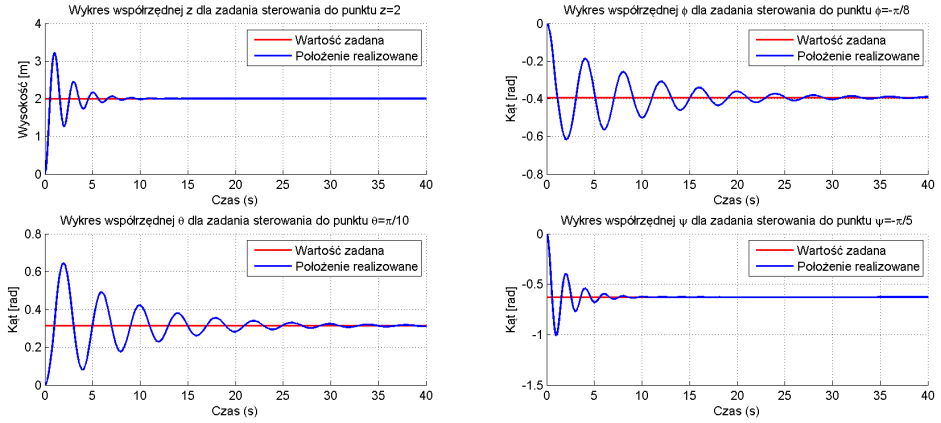
Rysunek 7.23. Śledzenie trajektorii dla parametrów K_d=100



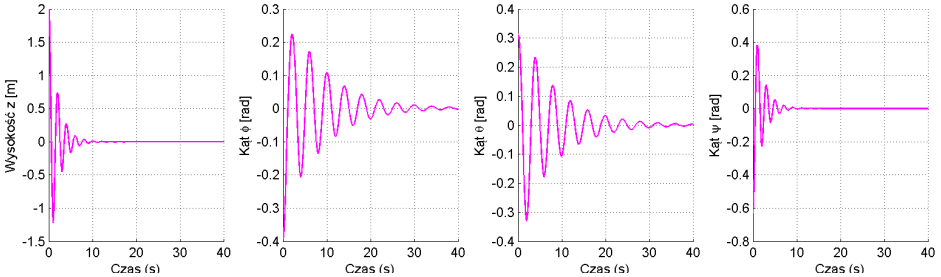
Rysunek 7.24. Uchyb śledzenia trajektorii dla parametrów K_d=100

7.1.4. Badania – sterowanie do punktu i śledzenie wybranych trajektorii

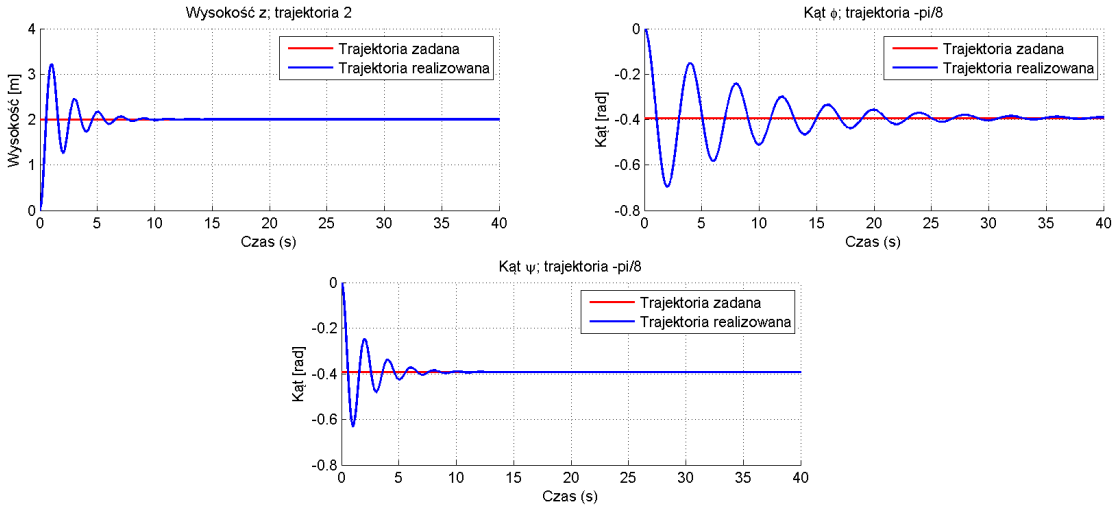
Zbadano sterowanie PID dla zadań sterowania do punktu oraz śledzenia trajektorii opartych o krzywe Béziera i łańcuchowe. Symulacje wykonano dla nastaw $K_p = 10$, $K_i = 0.1$, $K_d = 1$. Dla każdego z badanych przykładów uzyskano zbliżoną tendencję. Wyszczególniono dwie pary błędów o podobnej jakości regulacji. Pierwsza, składająca się z wysokości z i kąta myszkowania ψ, osiąga zbieżność znacznie szybciej, w przeciągu 20 sekund. Druga, złożona z kąta kiwania ϕ i kołysania θ, do osiągnięcia zbieżności potrzebuje czasu 60 sekund, co zostało przedstawione na rysunku 7.30. Ponadto zmienna z jako jedyna wykazuje zmiany w procesie regulacji w zależności od badanej trajektorii. Dla krzywych Béziera i łańcuchowych po osiągnięciu zbieżności uchyb wzrasta do poziomu 0.02. Pozostałe zmienne, oraz z przy sterowaniu do punktu uzyskują rząd wielkości błędów na poziomie 10^{-4}–10^{-5}. Zbadano, czy gorsza jakość regulacji kątów ϕ i θ może wynikać z wzajemnego wzbudzania, wpływu oscylacji jednego z kątów na drugi. W tym celu odłączono regulator kąta θ i zadano sterowanie do punktu. Rezultat umieszczono na rysunkach 7.27 i 7.28. Nie zauważono istotnych różnic w jakości sterowania kątem ϕ. Wykresy trajektorii jak i uchybu nie odbiegają od analogicznych uzyskancych przy sterowaniu czterema współrzędnymi. Przyczyną gorszej regulacji kątów kiwania i kołysania nie jest ich wzajemny wpływ.



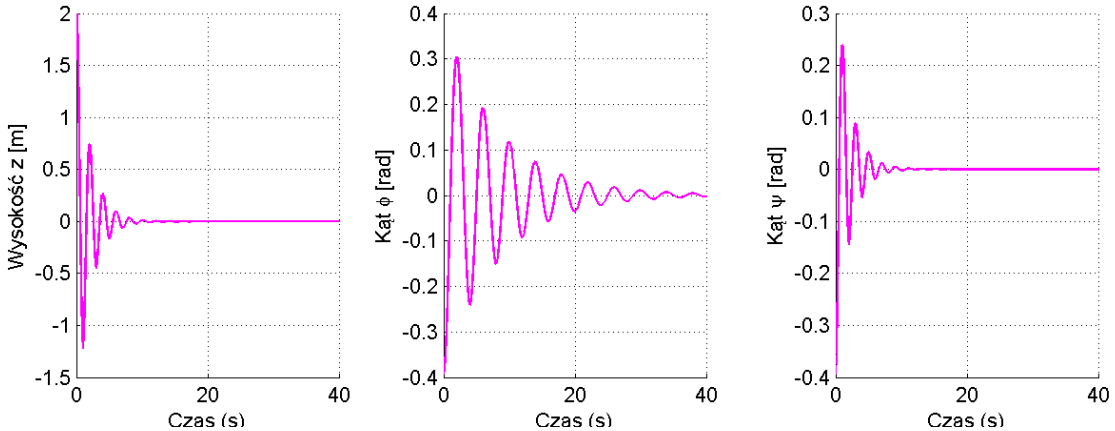
Rysunek 7.25. Sterowanie do punktu



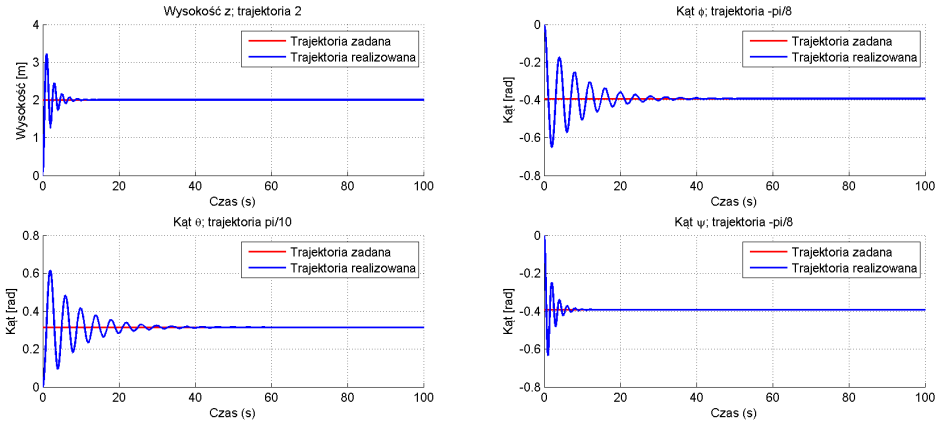
Rysunek 7.26. Uchyb przy sterowaniu do punktu



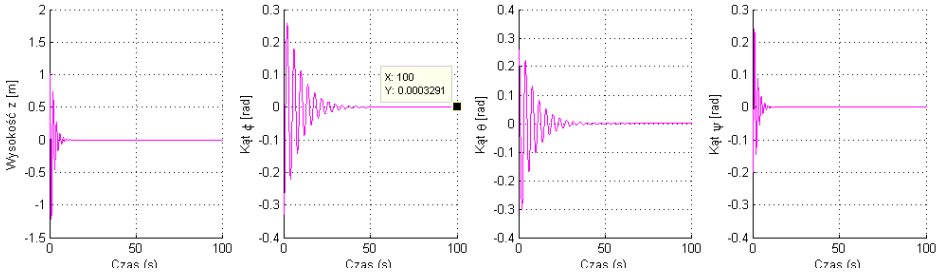
Rysunek 7.27. Sterowanie do punktu, przy odłączonej współrzędnej θ



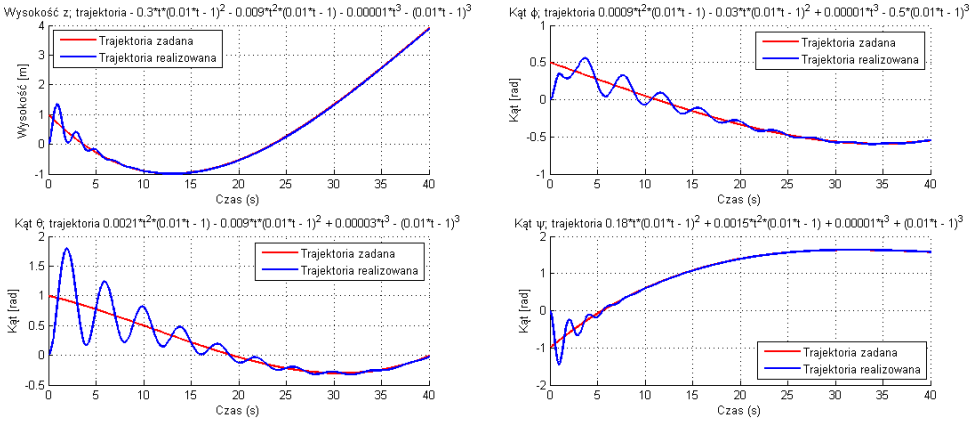
Rysunek 7.28. Uchyb przy sterowaniu do punktu, przy odłączonej współrzędnej θ



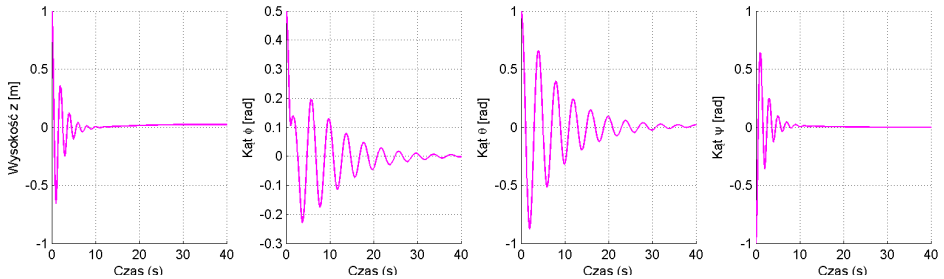
Rysunek 7.29. Sterowanie do punktu, dla czasu symulacji $t = 100$



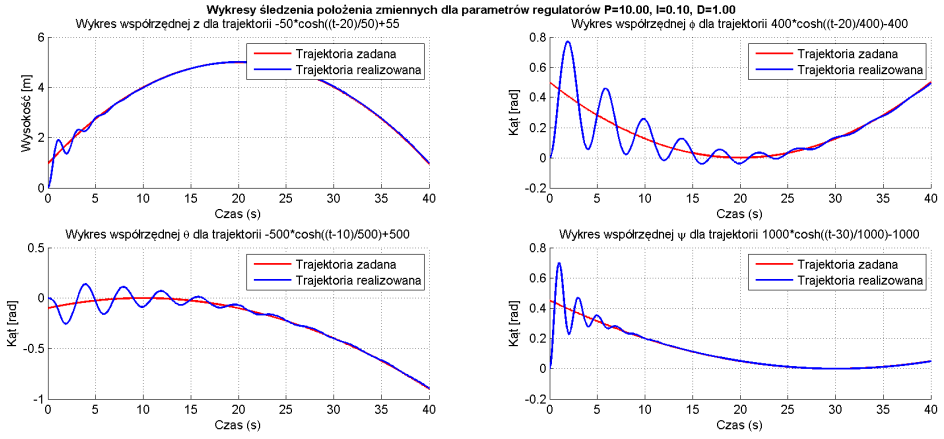
Rysunek 7.30. Uchyb przy sterowaniu do punktu, dla czasu symulacji $t = 100$



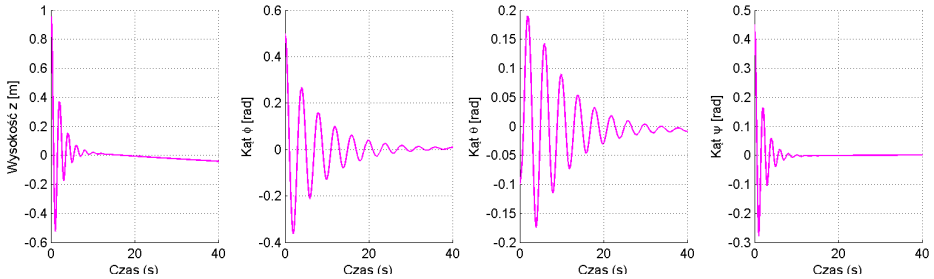
Rysunek 7.31. Śledzenie trajektorii dla zadanych krzywych Béziera



Rysunek 7.32. Uchyb śledzenia trajektorii dla zadanych krzywych Béziera



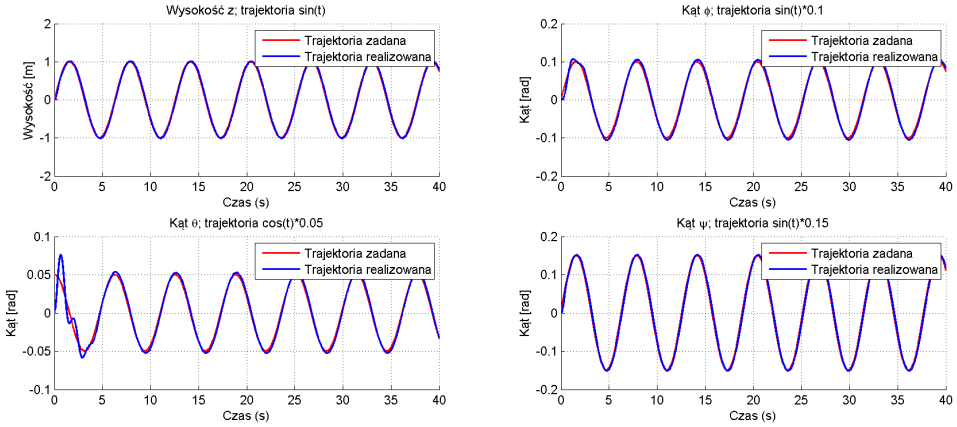
Rysunek 7.33. Śledzenie trajektorii dla krzywych łańcuchowych



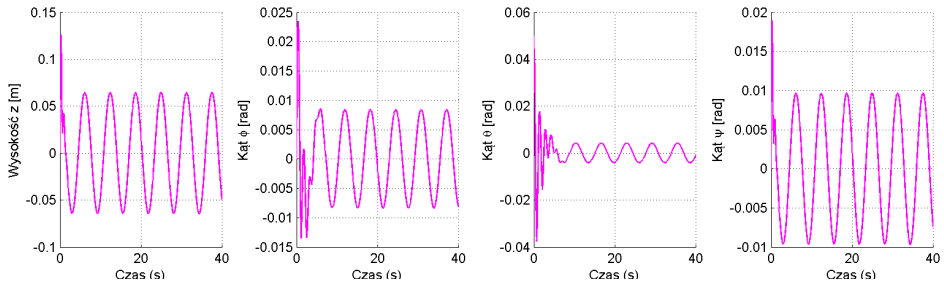
Rysunek 7.34. Uchyb śledzenia trajektorii dla krzywych łańcuchowych

7.1.5. Podsumowanie

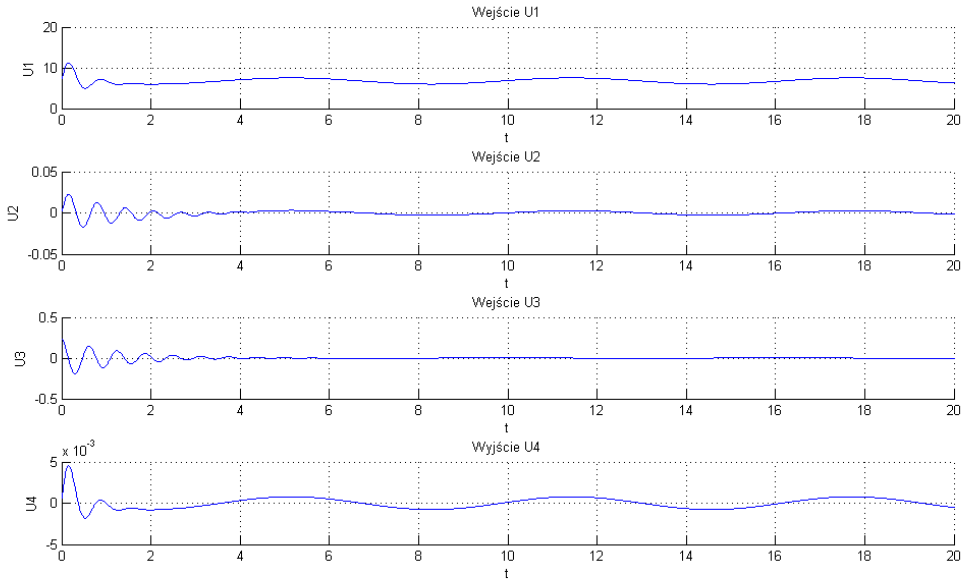
Po serii dodatkowych badań dla opisywanego obiektu, przy zadanych wartościach stałych fizycznych stwierdzono, że najlepiej radzi sobie regulator o nastawach K_p=80, K_i=0.1, K_d=5 – pokazują to rysunki 7.35 i 7.36 oraz wykres sterowań 7.37.



Rysunek 7.35. Śledzenie trajektorii dla parametrów K_p=80, K_i=0.1, K_d=5

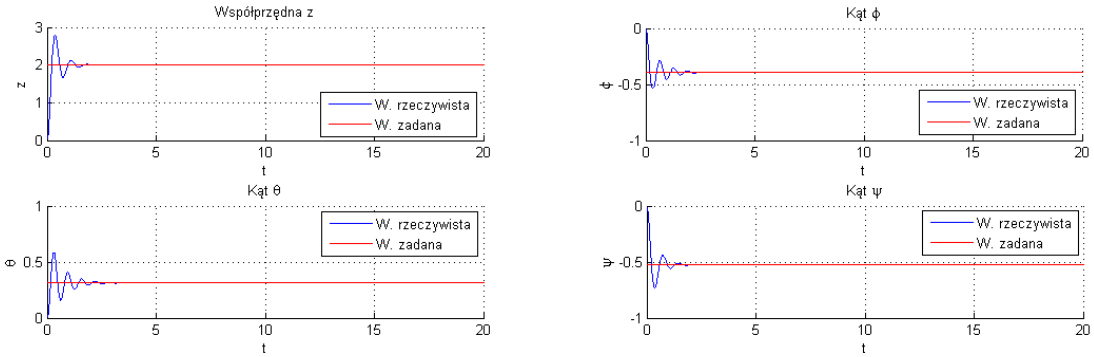


Rysunek 7.36. Uchyb śledzenia trajektorii dla parametrów K_p=80, K_i=0.1, K_d=5

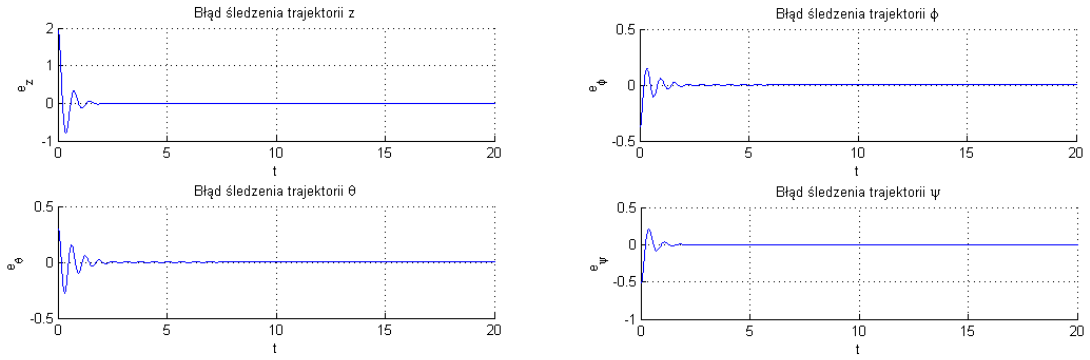


Rysunek 7.37. Wykresy zadawanego sterowania przy śledzeniu trajektorii dla parametrów K_p=80, K_i=0.1, K_d=5

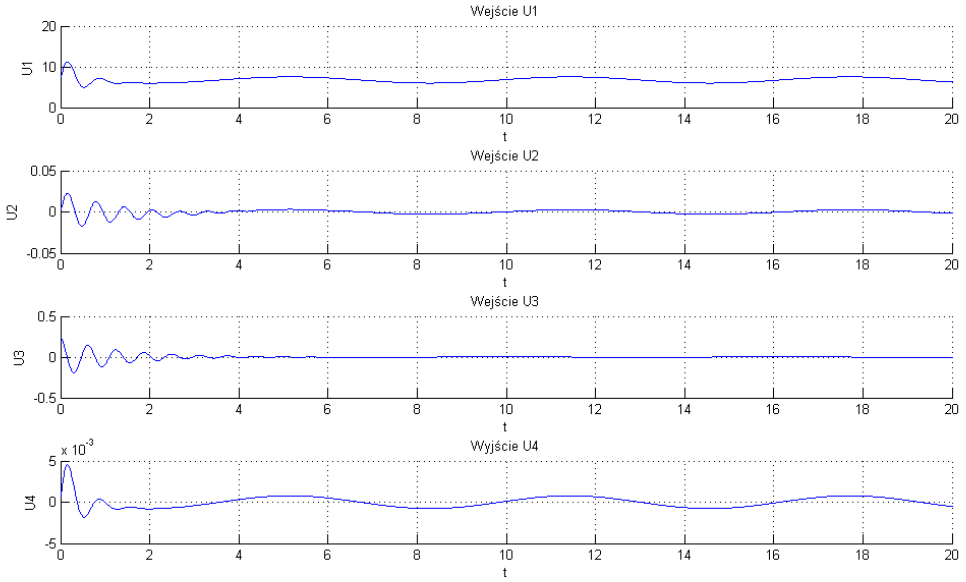
Błędy śledzenia trajektorii, które się pojawiają przy sterowaniu quadrotorem dla ustalonych parametrów w oparciu o trajektorię $\sin x$ (o amplitudzie 1 m), wynoszą 0.05 m; dla kątów skala błędów jest analogicznia. Z uwagi na gorszą jakość regulacjii kątów ϕ i θ, zaleca się zastosować dla nich nastaw o wyższych warościach. Sprawdzono, iż zbliżone przebiegi sterowania dla wszystkich zmiennych przy około czterokrotnym zwiększeniem K_p dla kątów kiwania i kołysania. Rezultat przedstawiono na rysunkach 7.38 i 7.39, wraz z wykresem sterowań 7.40 .



Rysunek 7.38. Sterowanie do punktu dla parametrów K_p=80, K_i=0.1, K_d=5 oraz podwyższonym K_p=320 dla kątów ϕ i θ



Rysunek 7.39. Uchyb sterowania do punktu i dla parametrów K_p=80, K_i=0.1, K_d=5 oraz podwyższonym K_p=320 dla kątów ϕ i θ



Rysunek 7.40. Wykresy zadawanego sterowania dla sterowania do punktu i dla parametrów K_p=80, K_i=0.1, K_d=5 oraz podwyższonym K_p=320 dla kątów ϕ i θ

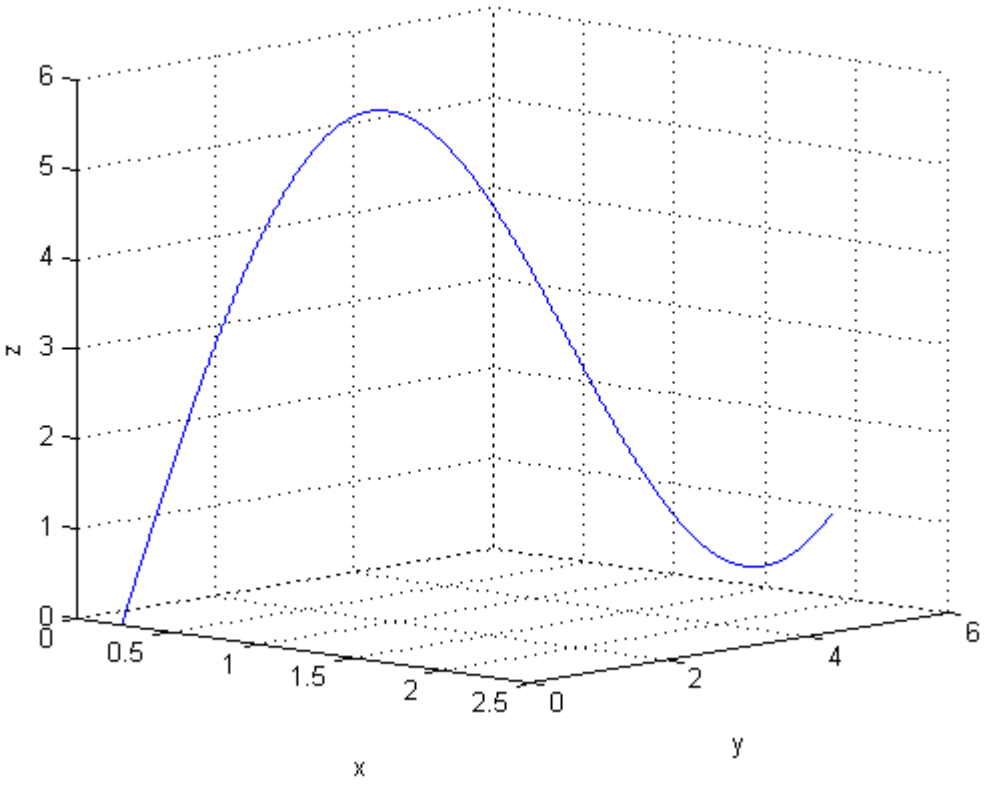
7.2. Algorytm całkowania wstecznego

7.2.1. Trajektorie

W przypadku śledzenia trajektorii układu łańcuchowego konieczne jest jej odpowiednie dobranie. Zadawane równania powinny być możliwe do wygenerowania przez dynamikę układu. W przypadku quadrotora opisanego równaniami (2.32) dopuszczalne są więc kombinacje funkcji stałych, liniowych oraz sinusa i kosinusa. Na nich też skupiły się badania algorytmu. Symulacje pokazały jednak, że niektóre spośród trajektorii niedopuszczalnych z powodzeniem można realizować za pomocą zaimplementowanego sterownika, co zostało pokazane w części 7.2.4.

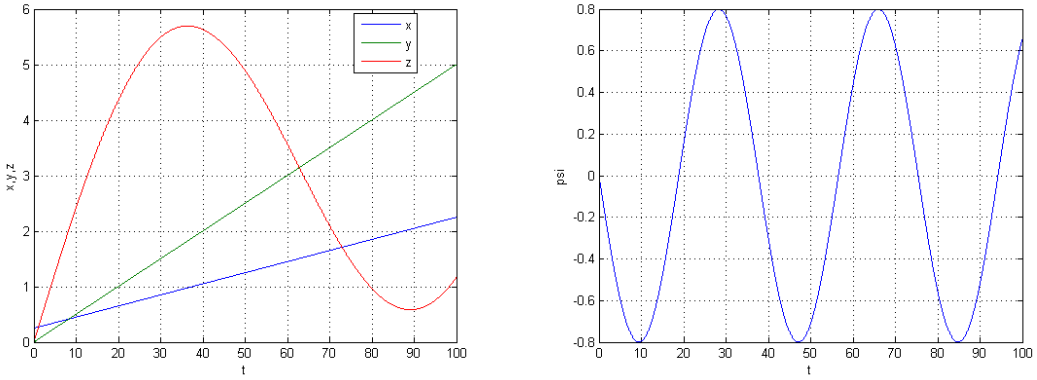
Zachowanie algorytmu pokazane zostanie na przykładzie wybranej trajektorii zadanej równaniem (7.1) pokazanej na rysunkach 7.41 i 7.42.

$$\begin{cases} x_d &= 0,02t + 0,25 \\ y_d &= 0,05t \\ z_d &= 0,05t + 4\sin(\frac{t}{20}) \\ \psi_d &= -0,8\sin(\frac{t}{6}) \end{cases} \tag{7.1}$$



Rysunek 7.41. Trajektoria zadana w przestrzeni trójwymiarowej

Wszystkie symulacje przeprowadzane były dla stałego kroku całkowania wynoszącego $dT = 0.01s$, metody całkowania ode3 (Bogacki-Shampine) oraz parametrów modelu pokazanych na wydruku 6.4, chyba że zostało wyraźnie zaznaczone, że jest inaczej. Czas symulacji wynosił 100s.



Rysunek 7.42. Trajektoria zadana równaniem (7.1)

7.2.2. Dobór parametrów sterownika

Nastawy algorytmu dobierane były na podstawie kryterium minimalizującego całkę K_b kwadratów błędów trajektorii

$$K_b = \int_0^{T_k} \Big((x(t) - x_d(t))^2 + (y(t) - y_d(t))^2 + (z(t) - z_d(t))^2 + (\psi(t) - \psi_d(t))^2 \Big)\, dt. \tag{7.2}$$

Sterowanie charakteryzuje się prostą zależnością – im większa wartość parametrów α, tym mniejsza wartość kryterium K_b, a błędy szybciej zbiegają do zera. Oczywiście spełnione jest to w pewnym zakresie i pod pewnymi warunkami.

Pierwszy problem, jaki pojawił się przy wyborze nastaw to obliczenia. Przy niektórych zestawach parametrów całkowanie numeryczne zaczynało generować bardzo duże błędy. System ulegał wzbudzeniu i współrzędna x zaczynała dążyć do nieskończoności. Zwiększenie zadawanej dokładności (przy całkowaniu o zmiennym kroku) niewiele dawało. MATLAB informował o problemach z doborem długości kroku wynikających z tzw. wysokiej sztywności systemu albo przeprowadzał bardzo dokładne i jednocześnie wolne symulacje, po których nie był w stanie wyeksportować danych z Simulinka do przestrzeni roboczej ze względu na ich rozmiar. Wykorzystanie metod całkowania przeznaczonych specjalnie dla systemów sztywnych nie poprawiało sytuacji, zwiększając jedynie czas obliczeń.

W celu uniknięcia problemu należało skorygować parametry α_9 i α_{10}, które okazały się kluczowym czynnikiem powodującym błędy. Unikać należy pewnych konfiguracji tych dwóch nastaw i dziesięciu pozostałych – zwykle wystarczające jest zadbanie, by co najmniej jeden z α_9 i α_{10} nie przekraczał 2.

W przypadku pozostałych nastaw, zwiększanie ich wartości poprawiało wskaźnik jakości, choć nie zawsze były to znaczące różnice. Większe wartości, pomimo spełnienia ograniczeń na α_9 i α_{10} ponownie prowadziły do niepoprawnego całkowania numerycznego. Przebiegi dla wybranych nastaw pokazane zostały w rozdziale 7.2.3.

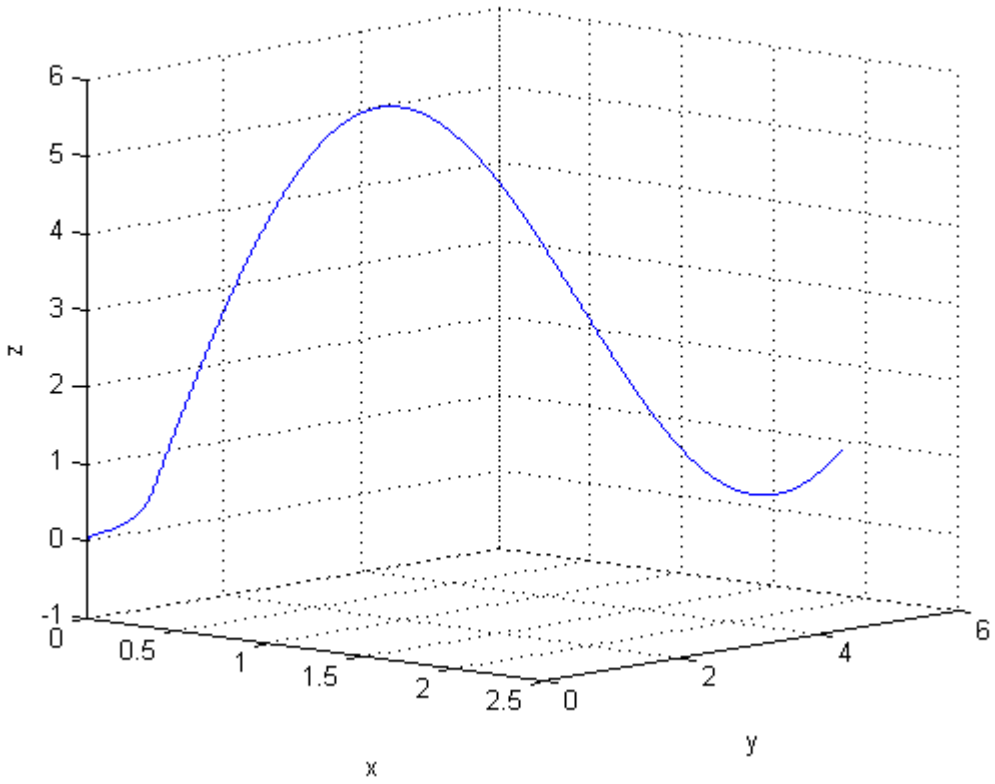
Przy wyborze sterowania wzięto pod uwagę możliwość implementacji sterownika na rzeczywistym układzie. W takim wypadku czas obliczeń pełni ważną rolę, a moc obliczeniowa mikroprocesorów jest znacznie mniejsza od tej, którą dysponuje standardowy procesor. Zwykle też implementuje się obliczenia o stałym kroku. Ostatecznie wybrane nastawy to

```
BSparam = [5 5 5 5 5 5 5 5 2 5 5 5].
```

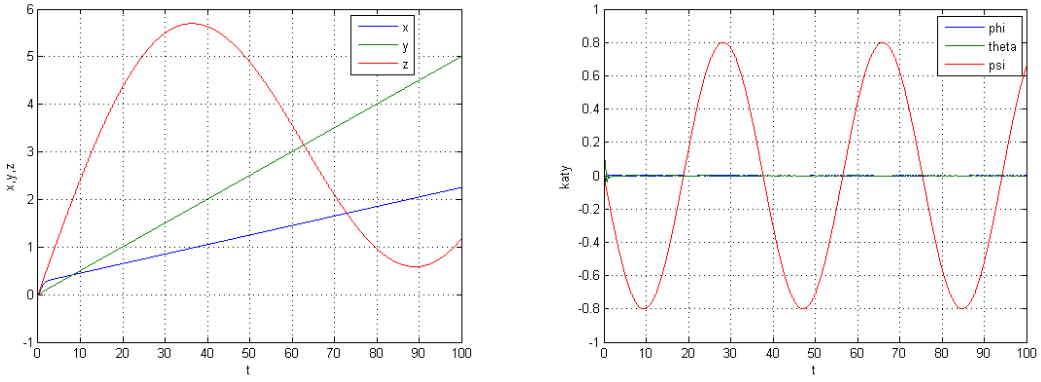
Wyniki śledzenia trajektorii (7.1) pokazują rysunki 7.43 do 7.46. Wartość całki wyniosła $K_b = 0.0429$.

Rysunek (7.47) pokazuje wyniki symulacji po zmianie parametrów modelu quadrotora na

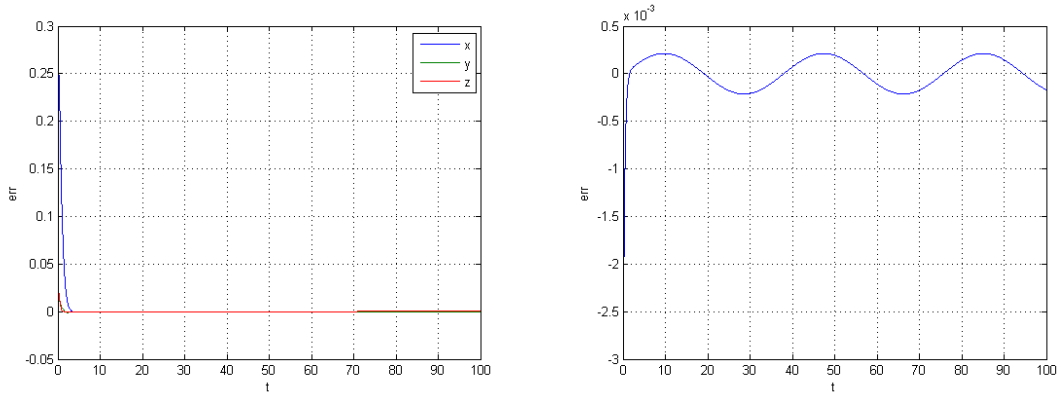
$$QP.m = 5, QP.l = 0.25, QP.b = 1, QP.d = 1, QP.Ix = 1, QP.Iy = 1, QP.Iz = 1, QP.J = 1.$$



Rysunek 7.43. Realizacja trajektorii (7.1) w przestrzeni trójwymiarowej

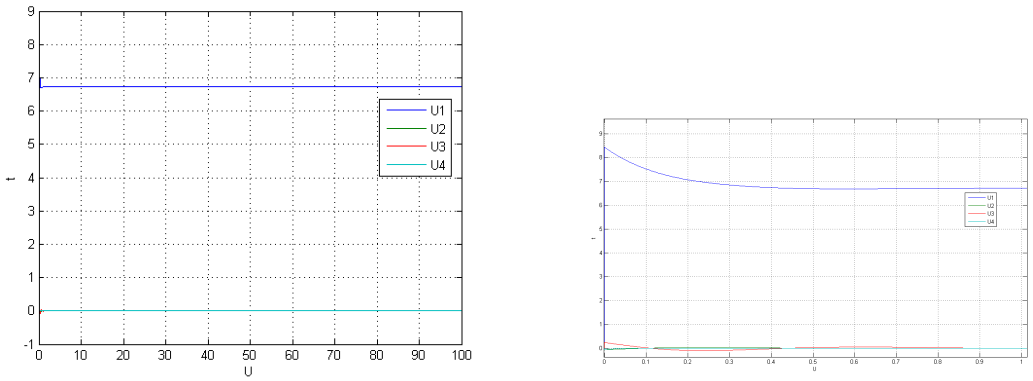


Rysunek 7.44. Realizacja trajektorii (7.1).



(a) położenie (b) ψ

Rysunek 7.45. Błędy śledzenia trajektorii (7.1)

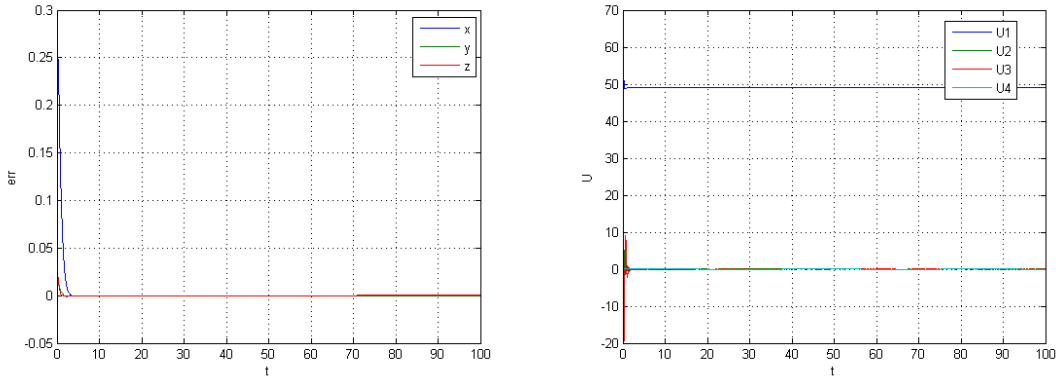


(a) pełny przebieg (b) przebieg w pierwszej sekundzie

Rysunek 7.46. Zadawane sterowanie podczas realizacji (7.1)

(7.3)

Trajektoria nadal była wiernie śledzona, a wartość kryterium nie uległa zmianie. Ze względu na zmianę masy przeskalowane zostało sterowanie.



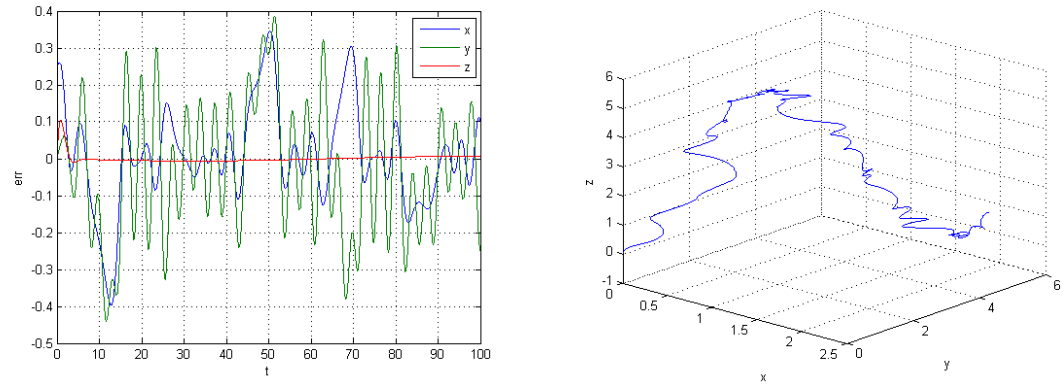
(a) położenie (b) ψ

Rysunek 7.47. Błędy śledzenia współrzędnych x,y i z oraz sterowanie po zmianie parametrów modelu quadrotora

7.2.3. Przegląd wyników symulacji dla różnych nastaw sterownika

Poniżej zamieszczony został przegląd wyników symulacji dla wybranych nastaw sterownika. Przedstawione w nim wykresy błędów dla współrzędnyche x, y, z, wykres błędu dla kąta ψ nie został pokazany ze względu na niewielkie wartości oraz nieznaczne różnice dla różnych nastaw.

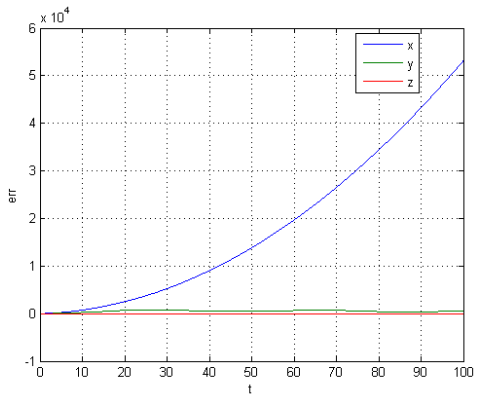
BSparam = [0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8]
— Wykres błędu i trajektorię pokazano na rysunku 7.48,
— $K_b = 5.6506$,
— zdecydowanie zbyt małe wartości nastaw powodują występowanie znacznych oscylacji w systemie.



(a) błędy (b) trajektoria w przestrzeni

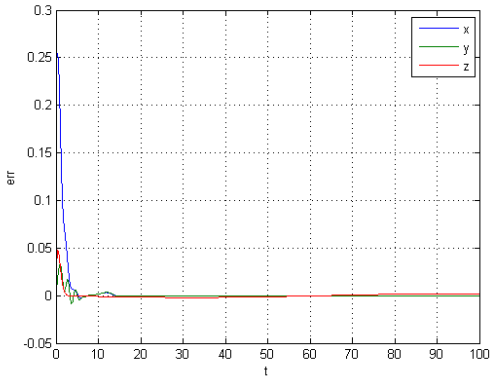
Rysunek 7.48. Wyniki symulacji dla BSparam = [0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8]

BSparam = [5 5 5 5 5 5 5 5 5 5 5 5]
— Wykres błędu pokazano na rysunku 7.49,
— $K_b = 5.6875 * 10^{10}$,
— przykład nastaw powodujących błędy w całkowaniu numerycznym.



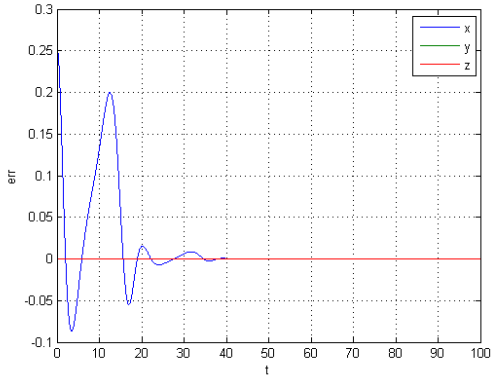
Rysunek 7.49. Wyniki symulacji dla BSparam = [5 5 5 5 5 5 5 5 5 5 5 5].

BSparam =[2 2 2 2 2 2 2 2 2 2 2 2]
— Wykres błędu pokazano na rysunku 7.50,
— $K_b = 0.0758$,
— nastawy pozwalają na dobrą realizację zadania śledzenia trajektorii, jednak dające gorszą wartość całki niż uzyskana w części 7.2.2.

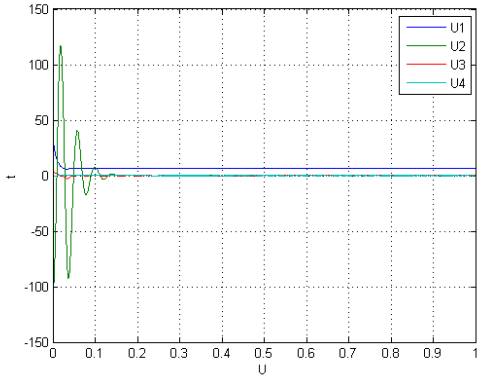


Rysunek 7.50. Wyniki symulacji BSparam =[2 2 2 2 2 2 2 2 2 2 2 2]

BSparam = [80 80 80 80 80 80 80 80 0.4 0.4 80 80]
— Wykres błędu pokazano na rysunku 7.51,
— $K_b = 0.0374$,
— parametry dają mniejszą wartość kryterium, jednak generują sterowania powyżej 100,
— ze względu na błędy numeryczne, badanie musiało zostać przeprowadzone przy zmiennym kroku całkowania (przyjęto metodę ode45 i dokładność 10^{-10}).



(a) błędy



(b) sterowanie w pierwszej sekundzie

Rysunek 7.51. Wyniki symulacji dla BSparam = [80 80 80 80 80 80 80 80 0.4 0.4 80 80]

7.2.4. Przegląd wyników symulacji dla różnych trajektorii

Poniżej zamieszczony został przegląd wyników dla różnych trajektorii. Wszystkie symulacje przeprowadzone zostały dla wartości sterownika dobranych w części 7.2.2.

Przemieszczanie się w poziomie Trajektoria zadana równaniem (7.4) wyznaczona jest jako swoisty patrol – quadrotor ma za zadanie przemieszczać się na stałej wysokości, zmieniając orientację i wykonując "ósemki". Zadanie to okazało się trudniejsze niż postawione w 7.2.1, pojawiają się większe błędy. Całka wynosi $K_b = 0.7789$

Trajektorię w przestrzeni trójwymiarowej i błędy śledzenia współrzędnych x, y, z pokazuje rysunek 7.52.

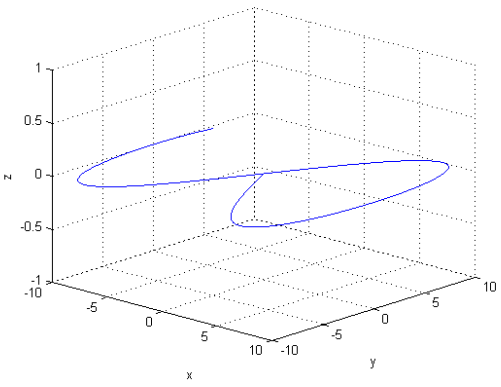
$$\begin{cases} x_d &= 10\sin(\frac{t}{20}) \\ y_d &= -10\sin(\frac{t}{10}) \\ z_d &= 0 \\ \psi_d &= -0,8\sin(\frac{t}{6}) \end{cases} \tag{7.4}$$

Funkcja kwadratowa Trajektoria zadana równaniem (7.5) różni się od trajektorii (7.1) współrzędnymi x_z i y_z . Teoretycznie należy ona do grupy trajektorii niedopuszczalnych ze względu na występowanie funkcji kwadratowej. Przeprowadzone symulacje pokazały jednak, że układ sterowania działa poprawnie, choć śledzenie obarczone jest większym błędem. Pojawiają się 'piki' wartości błędu. Częstość ich występowania określona jest przez pół okresu sinusa zadawanego dla kąta ψ, zaś wartość błędu współrzędnej x – z początkowym błędem trajektorii. Kryterium całkowej wyniosło $K_d = 0.9749$.

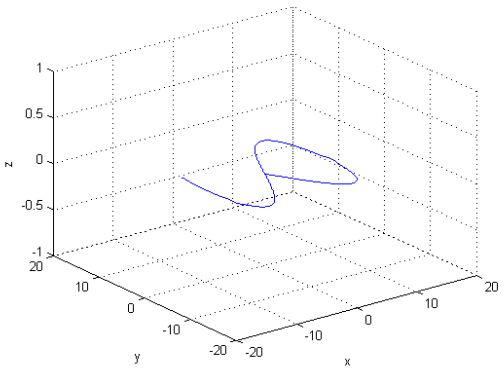
Realizację trajektorii w przestrzeni trójwymiarowej i błędy śledzenia współrzędnych x, y, z pokazuje rysunek 7.53.

$$\begin{cases} x_d &= 0,02t^2 + 0,25 \\ y_d &= 0,03t^2 \\ z_d &= 0,05t + 0,2\sin(\frac{t}{20}) \\ \psi_d &= -0,8\sin(\frac{t}{6}) \end{cases} \tag{7.5}$$

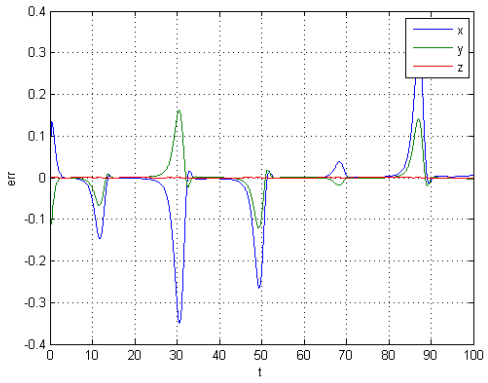
Zawis Trajektoria zadana równaniem (7.6) to po prostu zadanie quadrotorowi pozostania w zawisie i "rozglądania" się po okolicy. Również w tym wypadku sterownik doskonale sobie radzi. Całka wyniosła $K_b = 4.2111 * 10^{-6}$.



(a) trajektoria zadana

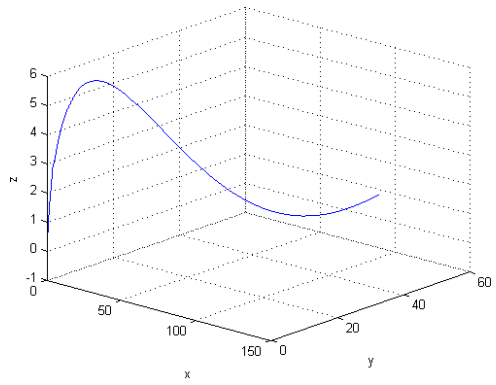


(b) trajektoria zrealizowana

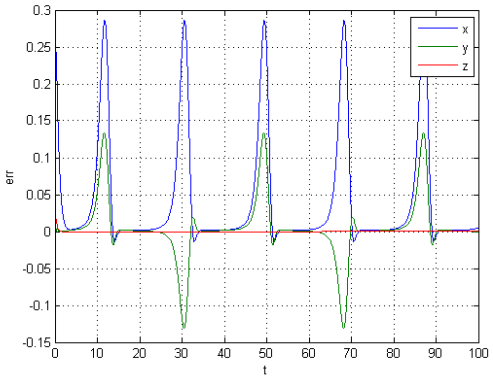


(c) błędy

Rysunek 7.52. Wyniki symulacji dla trajektorii zadanej równaniem (7.4)



(a) trajektoria zrealizowana

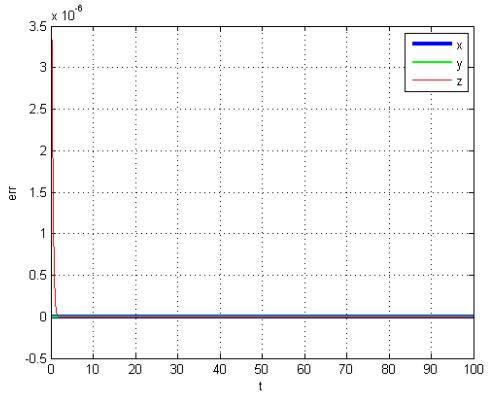


(b) błędy

Rysunek 7.53. Wyniki symulacji dla trajektorii zadanej równaniem 7.5.

Błędy śledzenia współrzędnych x, y, z pokazuje rysunek 7.54.

$$\begin{cases} x_d & = 0 \\ y_d & = 0 \\ z_d & = 0 \\ \psi_d & = -0,8\sin(\frac{t}{6}) \end{cases} \tag{7.6}$$



Rysunek 7.54. Wyniki symulacji dla trajektorii zadanej równaniem (7.6)

7.2.5. Podsumowanie

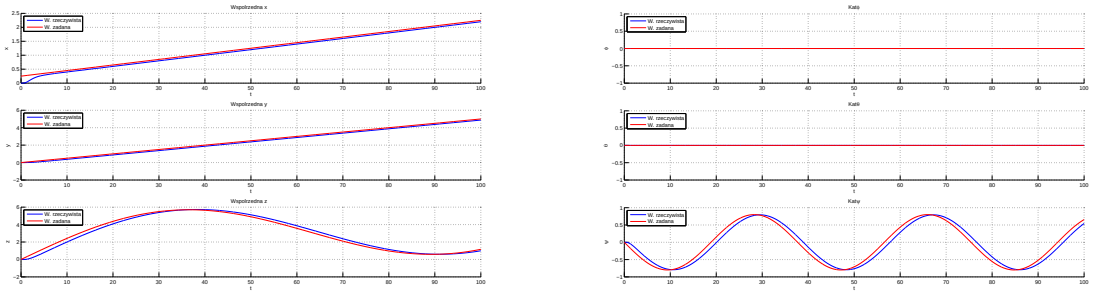
Przeprowadzone symulacje pokazują, że sterownik wykorzystujący algorytm całkowania wstecznego bardzo dobrze wykonuje zadanie śledzenia trajektorii. Podstawowym problemem przy realizacji zadania są obliczenia numeryczne – mogą powodować tak duże błędy, że system staje się niestabilny. Dlatego też ważne jest odpowiednie dobranie parametrów. Same parametry można zmieniać w szerokiej skali wartości (zależnej m.in. od parametrów modelu). Możliwe jest także zadawanie trajektorii teoretycznie niedopuszczalnej – choć w takim przypadku znacznie częściej pojawiają się problemy z obliczeniami. Algorytm sprawdza się także w zadaniu stabilizacji w punkcie.

7.3. Badania jakości sterowania H_∞ dla quadrotora

Podobnie jak w przypadku wcześniej analizowanych algorytmów sterowania po zaimplementowaniu sterownika H_∞ przeprowadzono badania jego jakości. W literaturze nie znaleziono żadnych warunków nakładających restrykcje na postać trajektorii wynikające z właściwości samego sterownika.

Na początku wybrano trajektorię opisaną równaniami (7.1), tak aby móc porównać jakość sterownika H_∞ ze sterownikiem opartym o algorytm całkowania wstecznego, gdyż w obu tych algorytmach wektor sterowań jest taki sam.

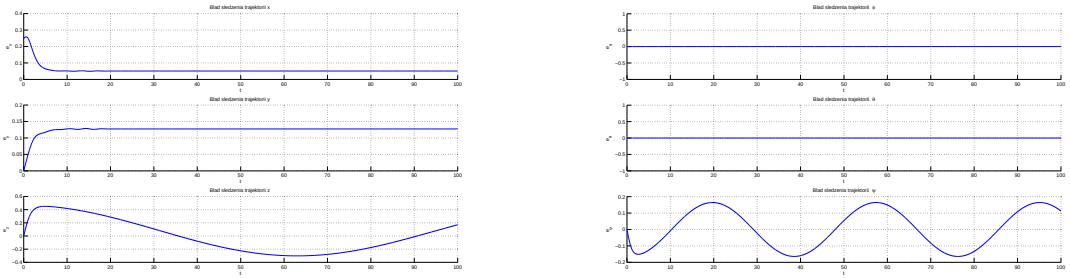
Na wykresach z rysunku 7.55 przedstawiono uzyskane położenia rzeczywiste obok położeń zadanych oraz zadany i rzeczywisty kąt ψ (kąt myszkowania). Kąty kiwania i kołysania nie wchodzą w skład wektora sterowań, jednakże udostępnione użytkownikowi GUI (patrz sekcja 6.6, 6.7) rysuje ich przebiegi automatycznie stąd ich obecność na wykresach



Rysunek 7.55. Położenia i kąty zadane i rzeczywiste (uzyskane symulacyjne) dla H_∞ podczas realizacji (7.1)

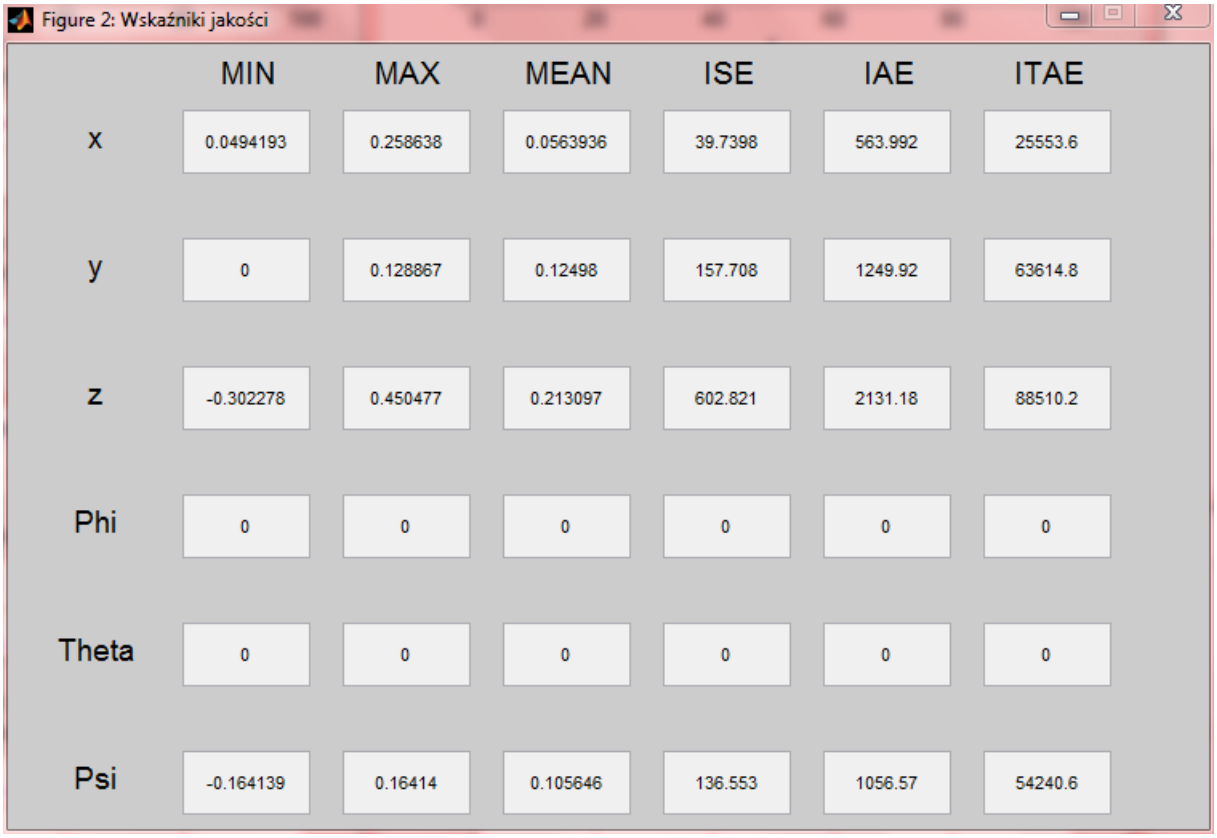
Niezależnie od analizowanej współrzędnej (x, y, z) można zaobserwować przesunięcie trajektorii rzeczywistej w stosunku do trajektorii zadanej. Podobnie dzieje się w przypadku kąta myszkowania. Gdyby w procesie sterowania przesunięcie można było wyeliminować działanie sterownika H_∞ można by określić jako idealne. Niestety algorytm sterowania będzie podążał za przebiegiem zadanej trajektorii z pewnym opóźnieniem w celu wyeliminowania szumów. Filtry występujące w pierwszej pętli mają właściwości całkujące wprowadzające owe opóźnienie. Jest to jednak własność systemu sterowania wynikająca ze sposobu jego zaprojektowania i tego typu opóźnień należy się spodziewać.

Występujące przesunięcie wpływa zatem na postać i wymiar uzyskanych błędów. Wykresy z rysunku 7.56 pokazują błąd trajektorii i kątów.



Rysunek 7.56. Błędy śledzenia trajektorii i kątów dla H_∞ podczas realizacji (7.1)

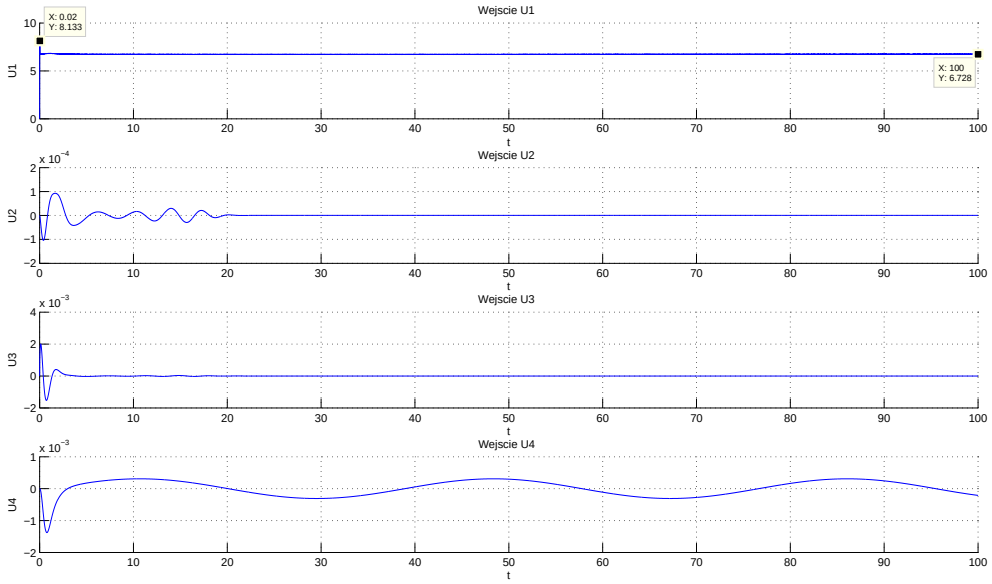
Biorąc pod uwagę występujące opóźnienie kryterium oceny przyjęte do analizy jakości sterowania algorytmem całkowania wstecznego nie jest miarodajnym kryterium dla algorytmu H_∞. Pozostałe przyjęte przy wizualizacji kryteria błędu również nie są miarodajne.



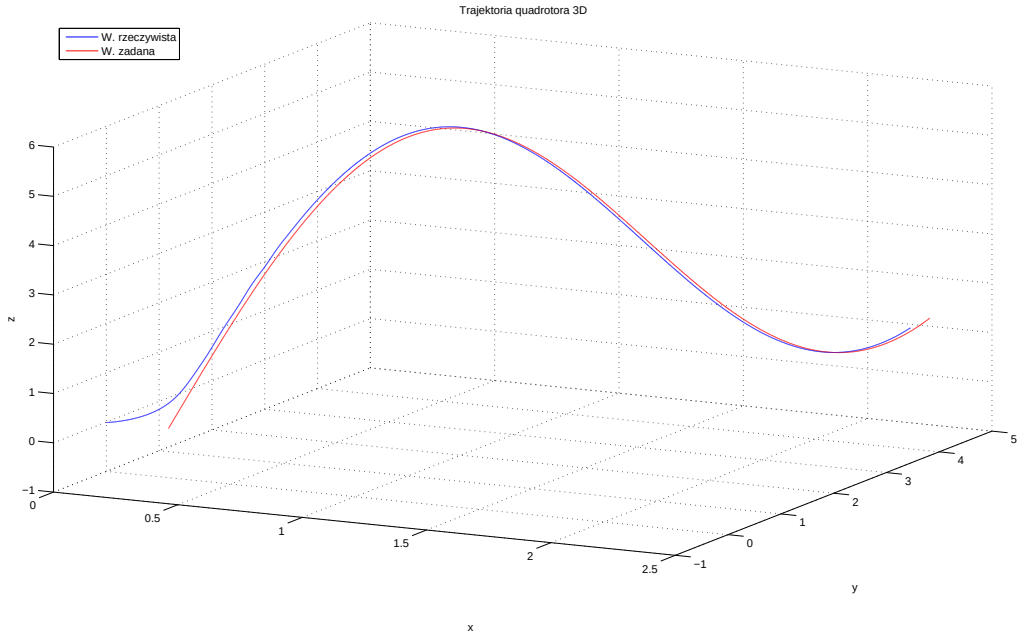
Rysunek 7.57. Wskaźniki jakości sterowania dla H_∞ podczas realizacji (7.1)

Warto zwrócić uwagę na wartości sterowań z rysunku 7.58, które w przypadku trajektorii (7.1) mają podobny przebieg do sterowań uzyskanych dla algorytmu całkowania wstecznego. Jednakże w innych przypadkach sterowania to może być znacznie łagodniejsze (mniejsze piki) niż w przypadku dwóch pozostałych algorytmów – patrz sekcja 7.5. Trajektorię zadaną i uzyskaną przedstawiono na rysunku 7.59.

W związku z charakterem swojej implementacji sterownik oparty o algorytm H_∞ nie oferuje użytkownikowi żadnych nastaw regulacji do samodzielnej manipulacji. Parametry lineryzacji wyznaczane są dla konkretnego modelu i jedynie zmiana jego parametrów wpływa na ich wartości.



Rysunek 7.58. Zadawane sterowanie dla H_∞ podczas realizacji (7.1)



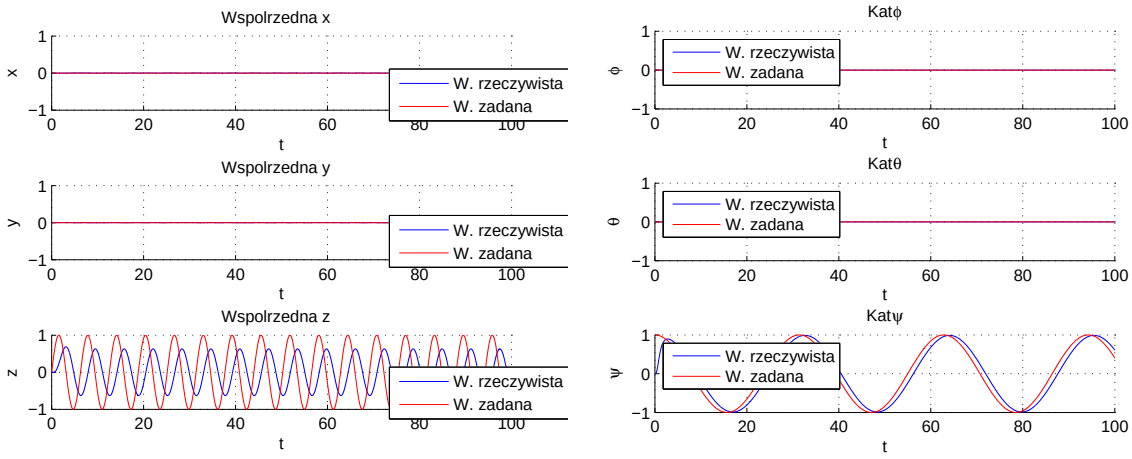
Rysunek 7.59. Trajektoria zadana i rzeczywista dla H_∞ podczas realizacji (7.1)

7.3.1. Wpływ zmian trajektorii współrzędnej z i kąta myszkowania na jakość sterowania

Jeśli za trajektorię przyjmiemy

$$\begin{cases} x_d &= 0 \\ y_d &= 0 \\ z_d &= \sin t \\ \psi_d &= \cos \frac{t}{5} \end{cases} \tag{7.7}$$

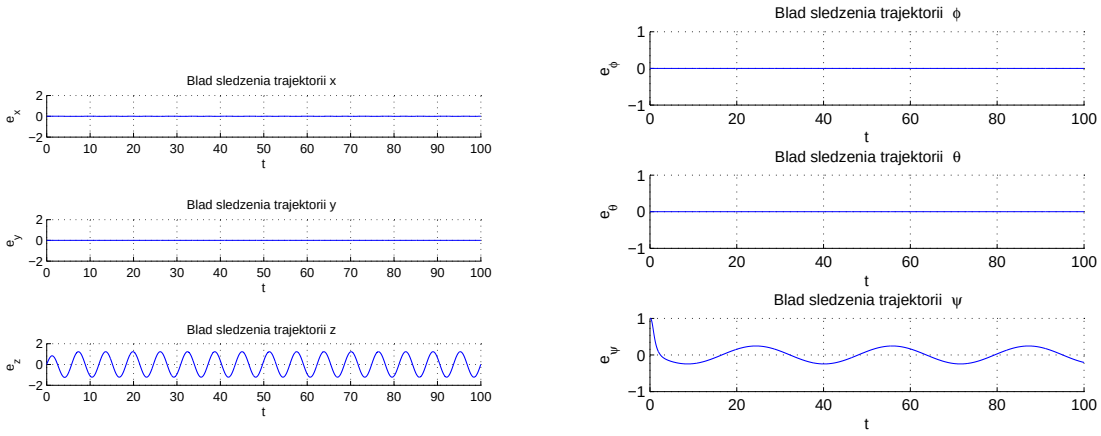
to na sterowanie układem będzie miała wpływ głównie pętla wewnętrzna zaimplementowanego sterownika.



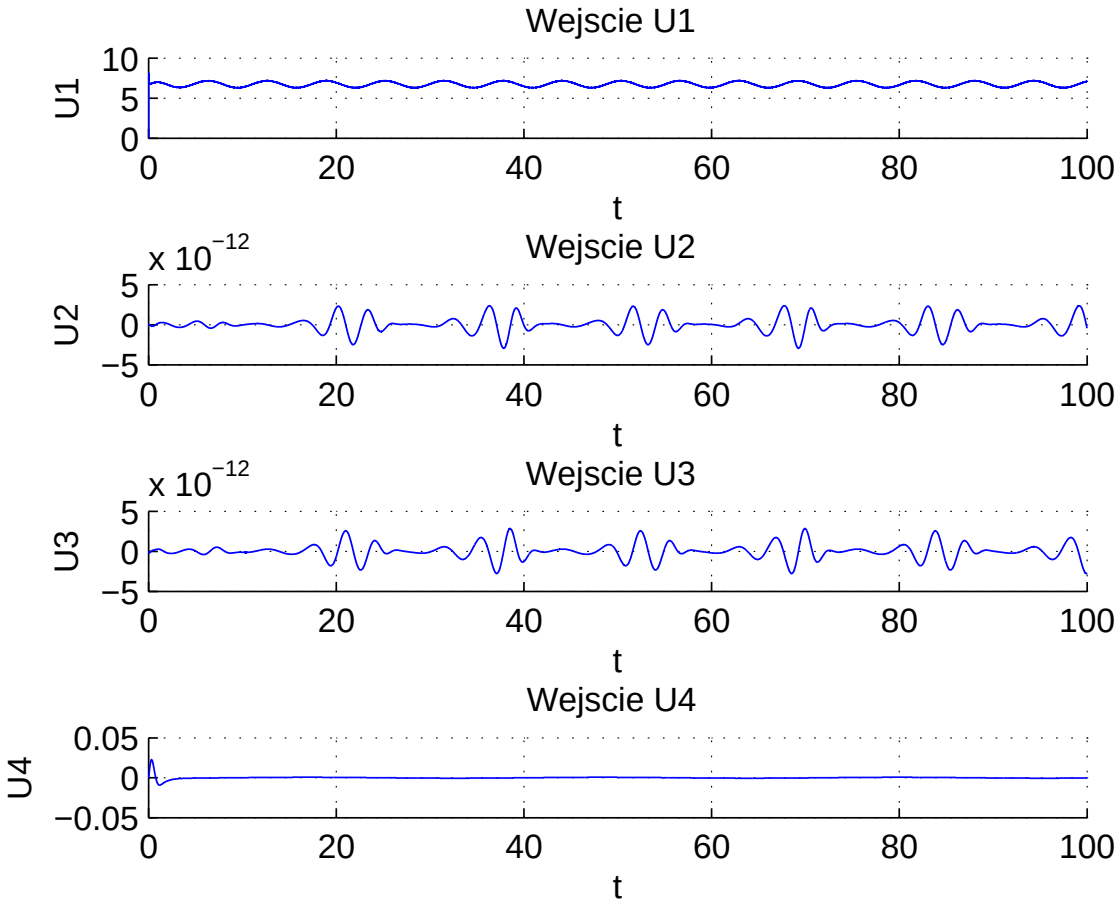
Rysunek 7.60. Położenia i kąty zadane i rzeczywiste (uzyskane symulacyjne) dla H_∞ podczas realizacji (7.7)

Błędy śledzenia trajektorii i kąta myszkowania zaprezentowano na wykresach z rysunku 7.61. Błędy śledzenia dla trajektorii współrzędnych x oraz y są na poziomie błędów numerycznych Matlaba, nie da się jednak stwierdzić czy faktycznie są to błędy numeryczne czy może wynikają ze złej jakości sterowania. Błąd trajektorii współrzędnej z wynika z opóźnienia śledzenia trajektorii zadanej. Układ próbkując sygnał, za jego maksymalną amplitudę uznaje wartość przesuniętą w stosunku do wartości prawidłowej – zamiast 1 metra wynosi ona około 0.6 metra – i właśnie tę wartość śledzi, zamiast wartości zadanej. Również oscylacja błędu śledzenia zachowania kąta myszkowania utrzymuje się na stałym poziomie i wynosi około 0.2 radiana.

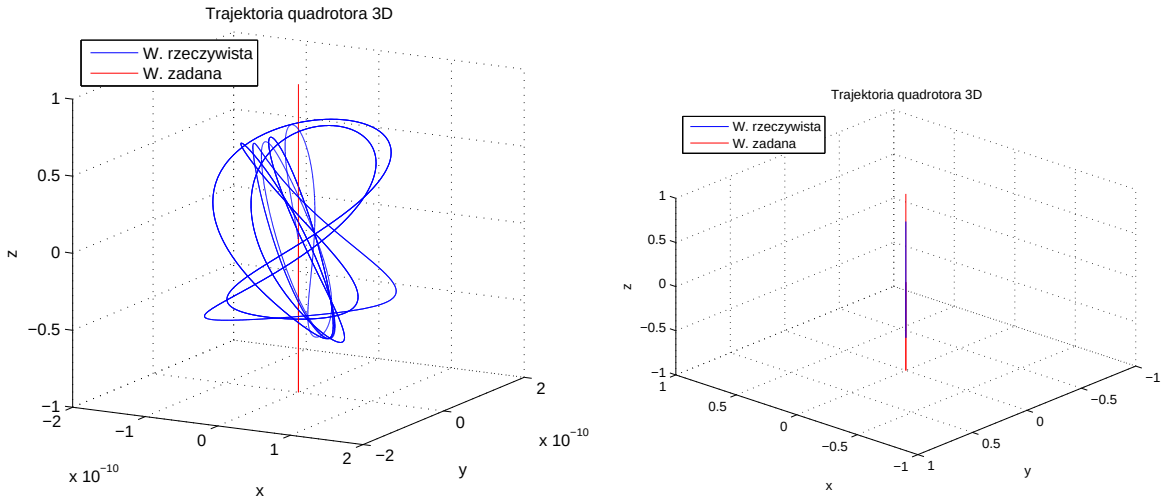
Występujące błędy dla trajektorii współrzędnej x oraz y powodują, iż uzyskana trajektoria rzeczywista podana w skali błędów $(1e-10)$ wygląda niestabilnie i w przypadku nie podania skali błędów może sugerować, iż quadrotor nie porusza się prawidłowo.



Rysunek 7.61. Błędy śledzenia trajektorii i kątów dla H_∞ podczas realizacji (7.7)



Rysunek 7.62. Zadawane sterowanie dla H_∞ podczas realizacji (7.7)



Rysunek 7.63. Trajektoria zadana i rzeczywista dla H_∞ podczas realizacji (7.7) – osie w skali uzyskanych błędów, osie wyskalowane ręcznie

7.3.2. Wpływ zmian trajektorii współrzędnych x i y na jakość sterowania

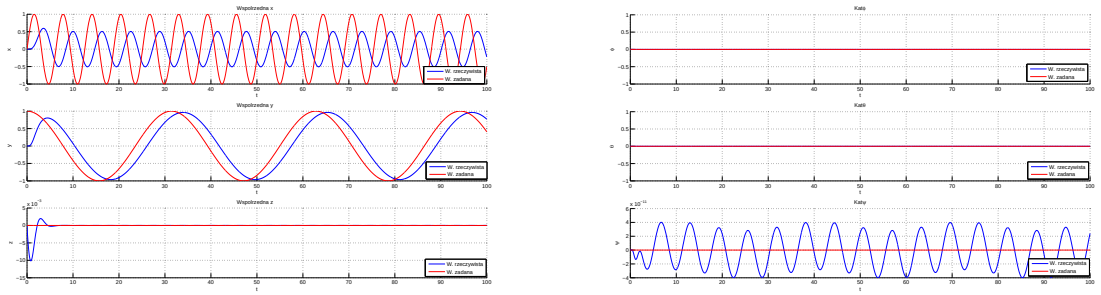
Jeśli za trajektorię przyjmiemy

$$\begin{cases} x_d &= \sin t \\ y_d &= \cos \frac{t}{5} \\ z_d &= 0 \\ \psi_d &= 0 \end{cases} \tag{7.8}$$

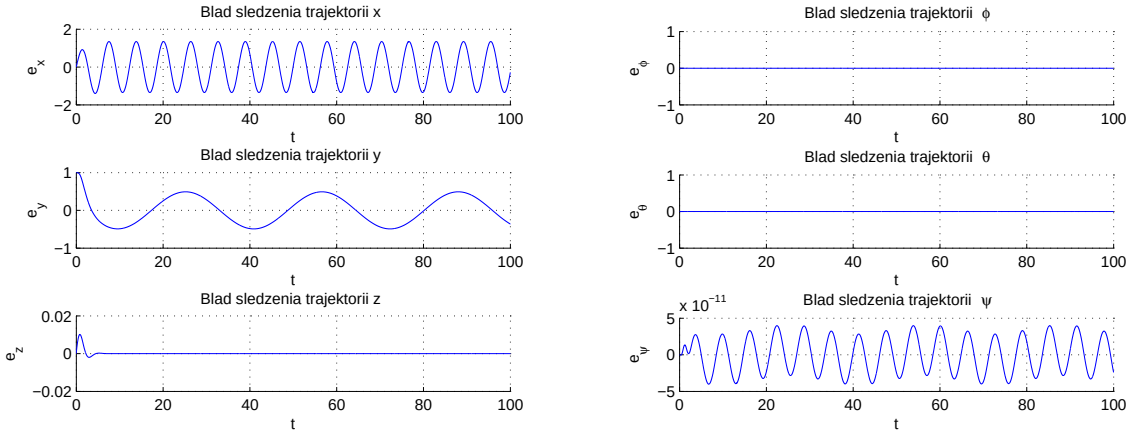
to na sterowanie układem będzie miała wpływ głównie pierwsza pętla zewnętrzna zaimplementowanego sterownika. Wykresy przedstawiające wyniki symulacji podążania za tą trajektorią widoczne są na rysunku 7.64. Należy zwrócić uwagę, że w przypadku współrzędnej położenia x wartość amplitudy sinusoidy została znacznie ograniczona. W przypadku dalszego zwiększania częstotliwości sinusoidy zostałaby ona uśredniona (potraktowana jako szumy) i quadrotor poruszałby się na wprost. W przypadku współrzędnej y widoczne jest szybkie dotarcie do wartości zadanej w pierwszych sekundach ruchu (błąd początkowy wynosił 1).

Błędy śledzenia trajektorii i kąta myszkowania zaprezentowano na wykresach z rysunku 7.65. Błędy śledzenia dla trajektorii po współrzędnej z oraz kąta myszkowania są na poziomie błędów numerycznych Matlaba, co potwierdza niezależność tych sterowań od sterowań współrzędnymi x i y.

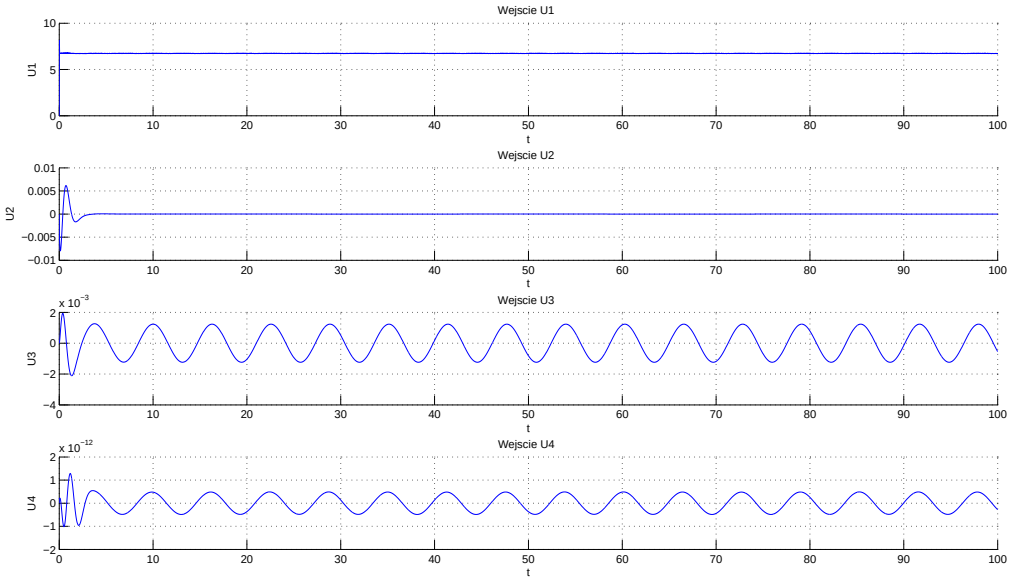
Występujące błędy dla trajektorii po współrzędnej z oraz kąta myszkowania powodują, iż uzyskana trajektoria rzeczywista wygląda niestabilnie i w przypadku nie podania skali błędów może sugerować, iż quadrotor nie porusza się prawidłowo.



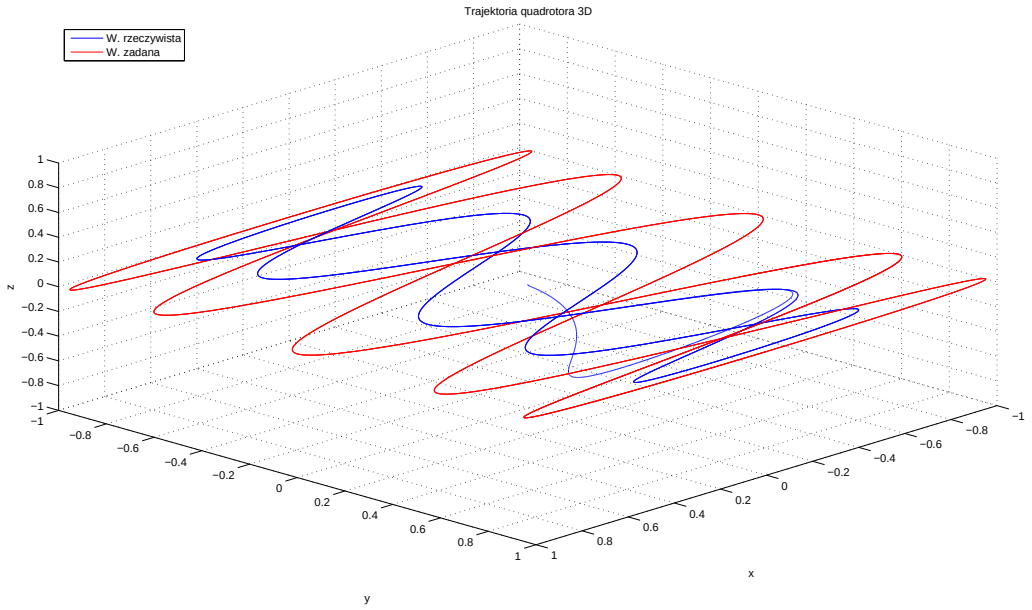
Rysunek 7.64. Położenia i kąty zadane i rzeczywiste (uzyskane symulacyjne) dla H_∞ podczas realizacji (7.8)



Rysunek 7.65. Błędy śledzenia trajektorii i kątów dla H_∞ podczas realizacji (7.8)



Rysunek 7.66. Zadawane sterowanie dla H_∞ podczas realizacji (7.8)



Rysunek 7.67. Trajektoria zadana i rzeczywista dla H_∞ podczas realizacji (7.8)

7.3.3. Wpływ zmian kąta myszkowania na jakość sterowania

Dla trajektorii danej układem:

$$\begin{cases} x_d &= \frac{t}{2} \\ y_d &= \frac{t}{2} \\ z_d &= 0 \\ \psi_d &= 10\sin(2\pi\frac{t}{40}) \end{cases} \tag{7.9}$$

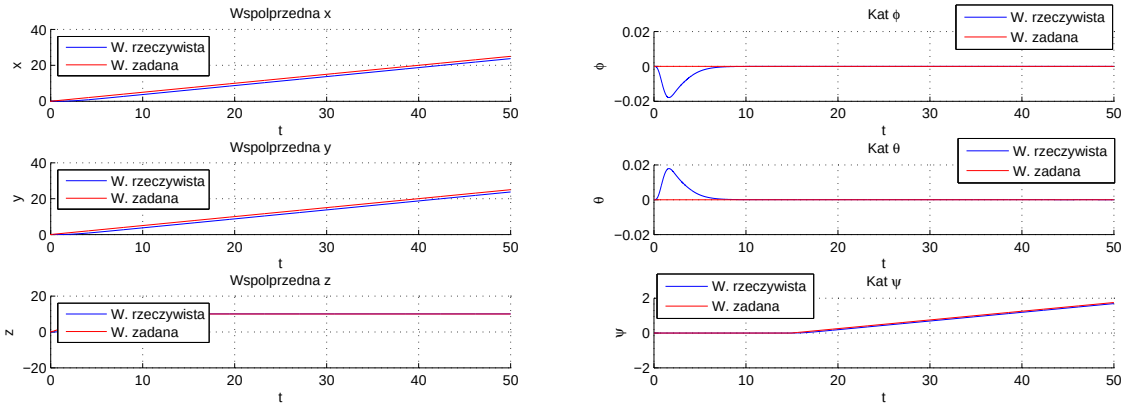
dla czasu t symulacji od 0 do 10s,

$$\begin{cases} x_d &= \frac{t}{2} \\ y_d &= \frac{t}{2} \\ z_d &= 10 \\ \psi_d &= 0 \end{cases} \tag{7.10}$$

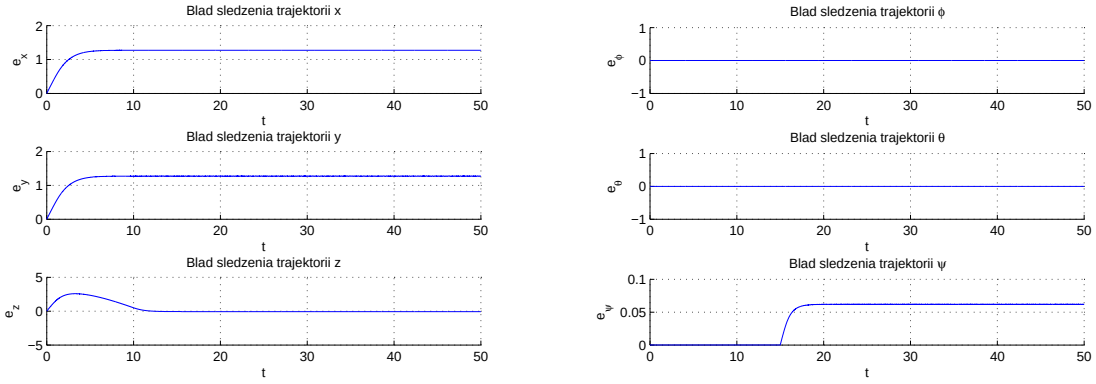
dla czasu t od 10 do 15 sekundy symulacji oraz

$$\begin{cases} x_d &= \frac{t}{2} \\ y_d &= \frac{t}{2} \\ z_d &= 10 \\ \psi_d &= \frac{t-15}{20} \end{cases} \tag{7.11}$$

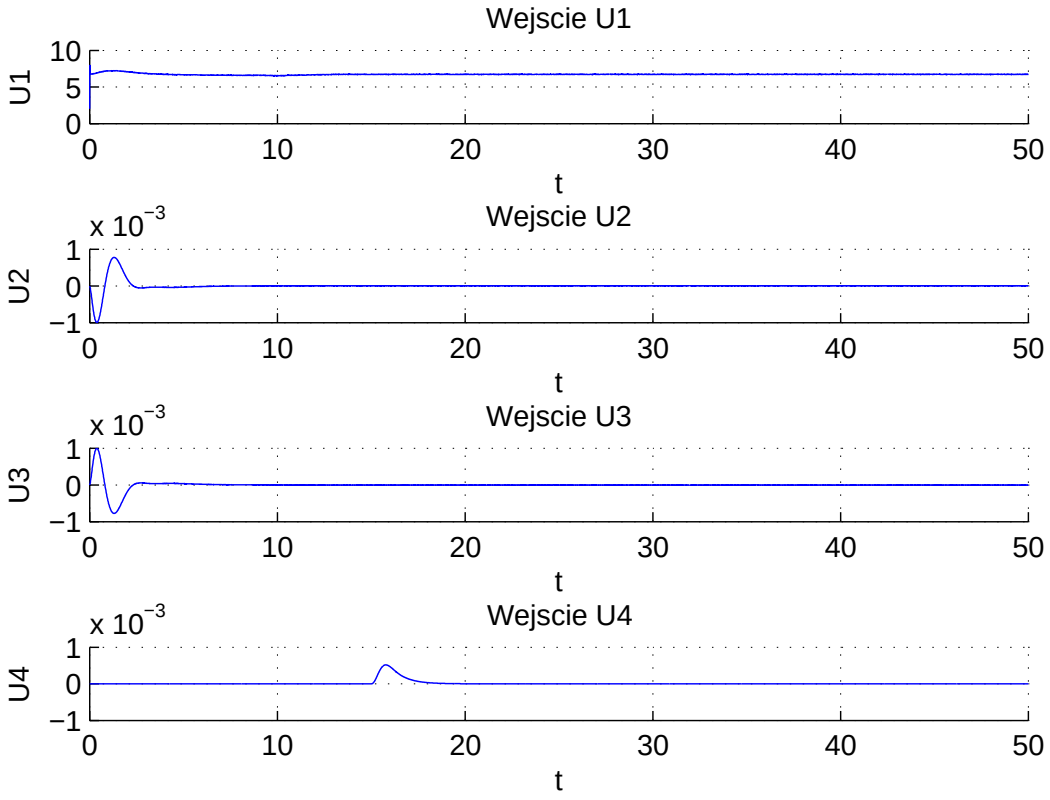
dla czasu t symulacji od 15 do 50 sekundy symulacji, można zaobserwować obracanie się quadratora wokół własnej osi (zgodnie z trajektorią zadaną na kąt myszkowania). Przeprowadzone symulacje dla tej trajektorii widoczne są na rysunku 7.69. Błąd śledzonej trajektorii zrzutowany na płaszczyznę xy widoczny jest na rysunku 7.71.



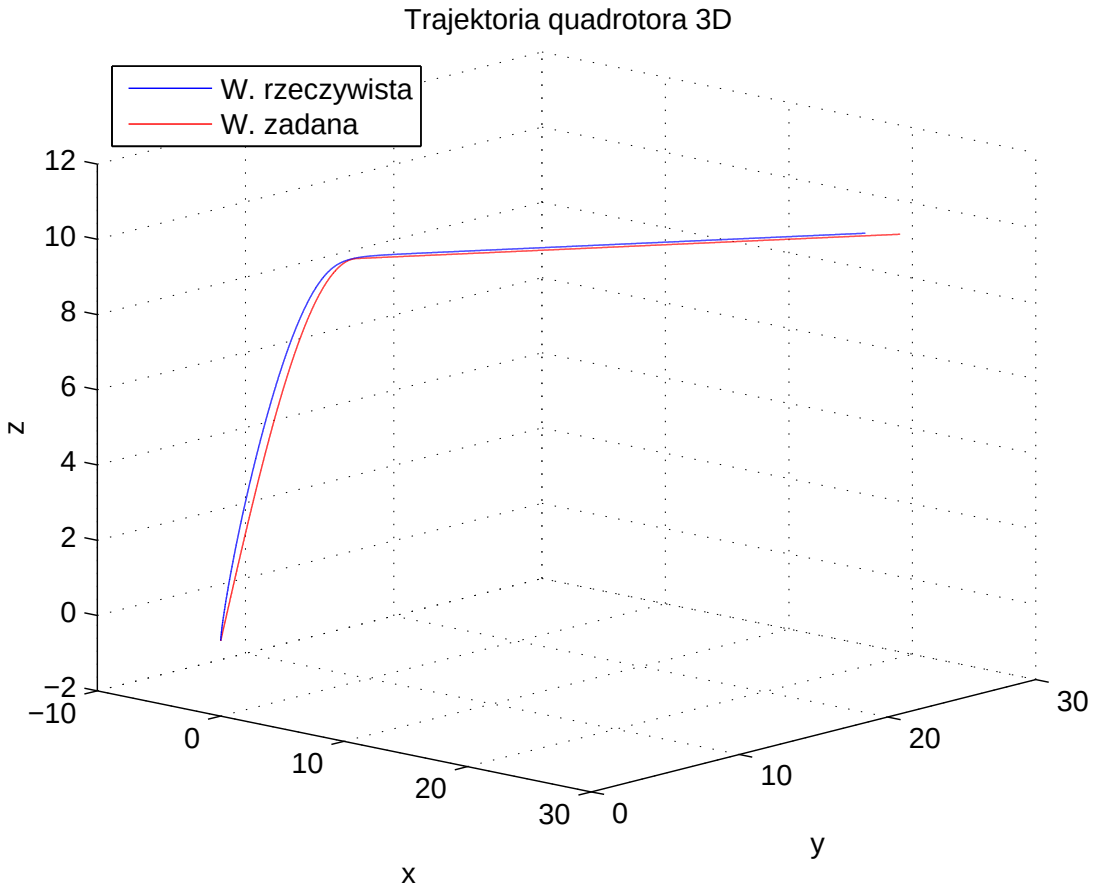
Rysunek 7.68. Położenia i kąty zadane i rzeczywiste (uzyskane symulacyjne) dla H_∞ podczas realizacji (7.9)-(7.11)



Rysunek 7.69. Błędy śledzenia trajektorii i kątów dla H_∞ podczas realizacji (7.9)-(7.11)



Rysunek 7.70. Zadawane sterowanie dla H_∞ podczas realizacji (7.9)-(7.11)



Rysunek 7.71. Trajektoria zadana i rzeczywista dla H_∞ podczas realizacji (7.9)-(7.11)

7.3.4. Przypadek na granicy stabilności układu

Dla trajektorii danej układem:

$$\begin{cases} x_d &= 0 \\ y_d &= 0 \\ z_d &= 2t \\ \psi_d &= 0 \end{cases} \tag{7.12}$$

dla czasu t symulacji od 0 do 15s,

$$\begin{cases} x_d &= 2t - 30 \\ y_d &= -4\sin\frac{t}{8} + 4\sin\frac{15}{8} \\ z_d &= 30 \\ \psi_d &= 0 \end{cases} \tag{7.13}$$

dla czasu t od 15 do 40 sekundy symulacji oraz

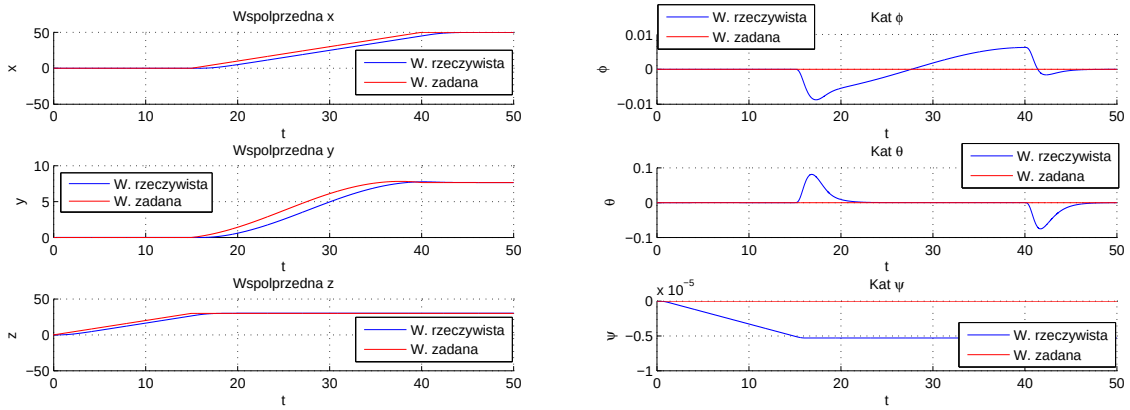
$$\begin{cases} x_d &= x_d(40s) \\ y_d &= y_d(40s) \\ z_d &= z_d(40s) \\ \psi_d &= 0 \end{cases} \tag{7.14}$$

dla czasu t symulacji od 40 do 50 sekundy symulacji, otrzymane przebiegi przedstawiono na wykresach z rysunku 7.72 - 7.75.

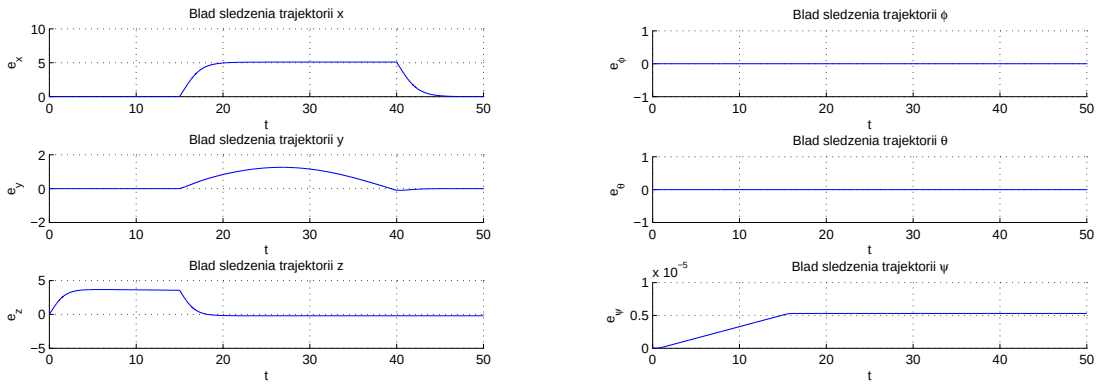
Gdyby jednak w zadanej trajektorii wprowadzić manipulację kątem myszkowania

$$\psi_d = \sin\frac{t}{4} \tag{7.15}$$

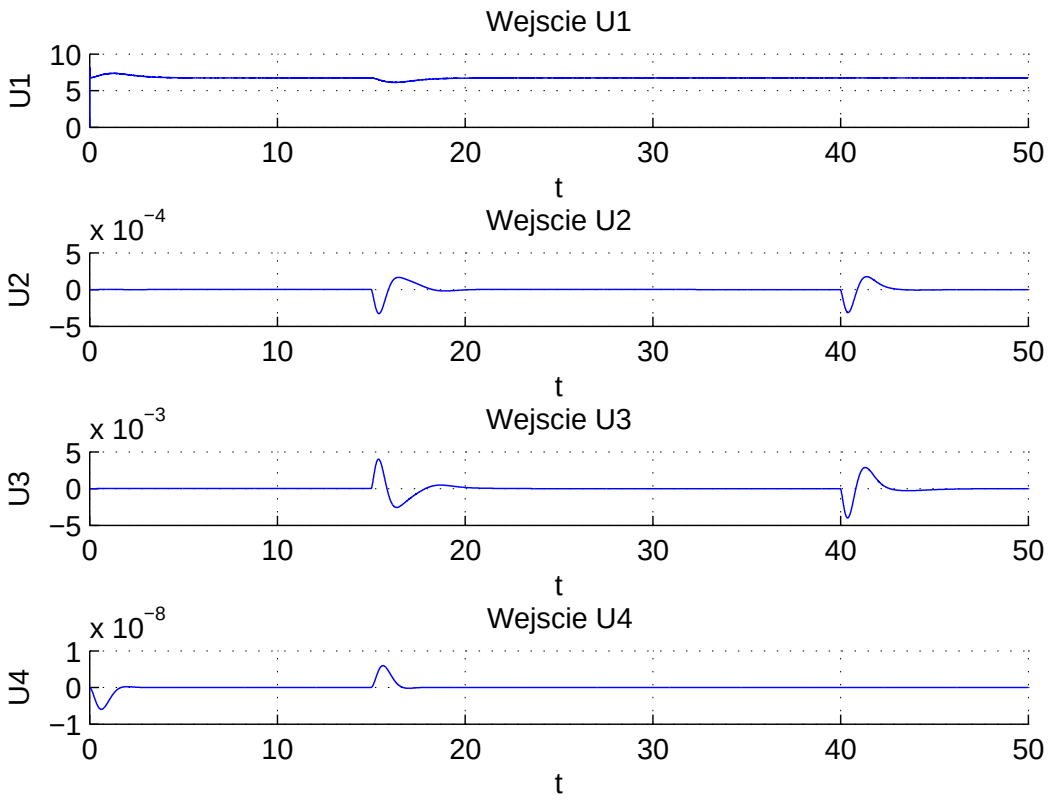
wtedy układ destabilizuje się i zaczyna oscylować w okolicach wartości zadanej (jest to najlepiej widoczne na wykresie przedstawiającym zależność y od czasu – 7.76), czego przyczyną jest prawdopodobnie nieprawidłowe dobranie nastaw (pre- i post-kompensatorów) w pierwszej pętli zewnętrznej sterownika. Niestety próby wyeliminowania tego zjawiska nie powiodły się. W przypadku dalszego zwiększania prędkości zmiany kąta myszkowania system staje się niestabilny i wartości współrzędnych położenia quadrotora na płaszczyźnie xy dążą szybko do (minus) nieskończoności.



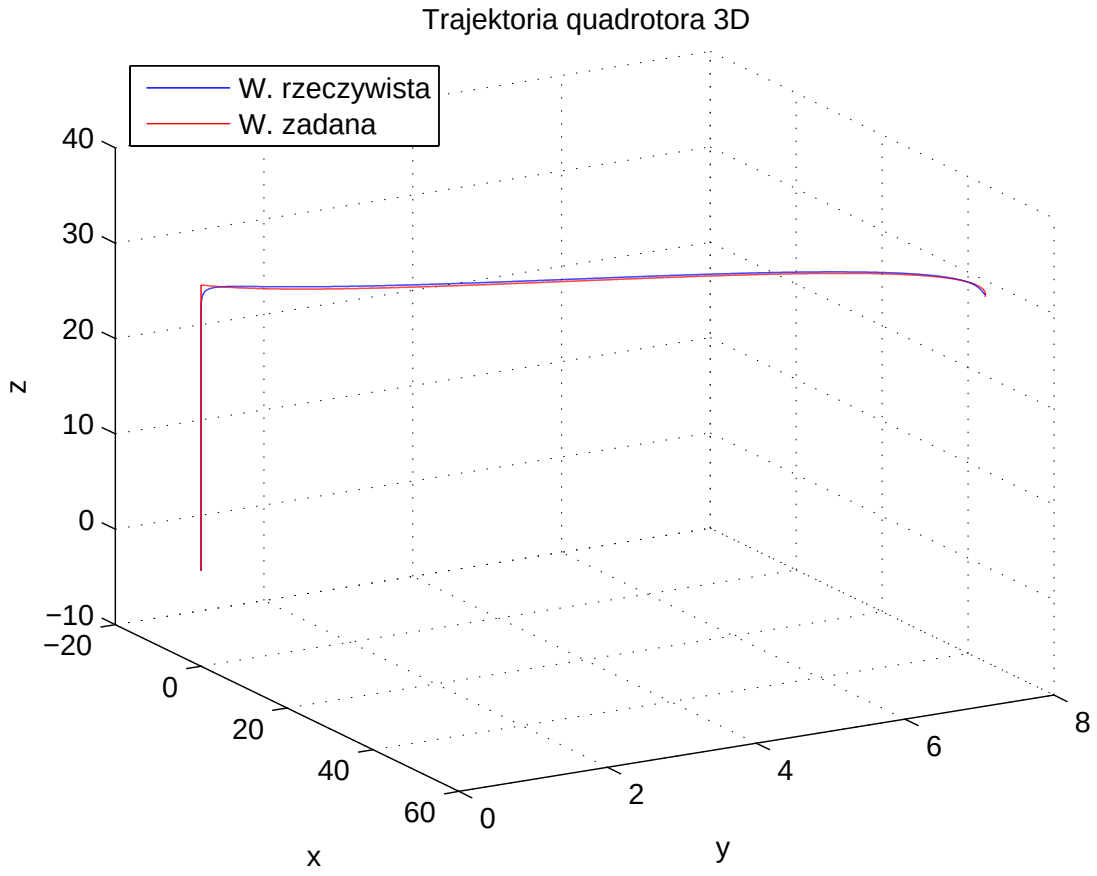
Rysunek 7.72. Położenia i kąty zadane i rzeczywiste (uzyskane symulacyjne) dla H_∞ podczas realizacji (7.12)-(7.14) – układ stabilny



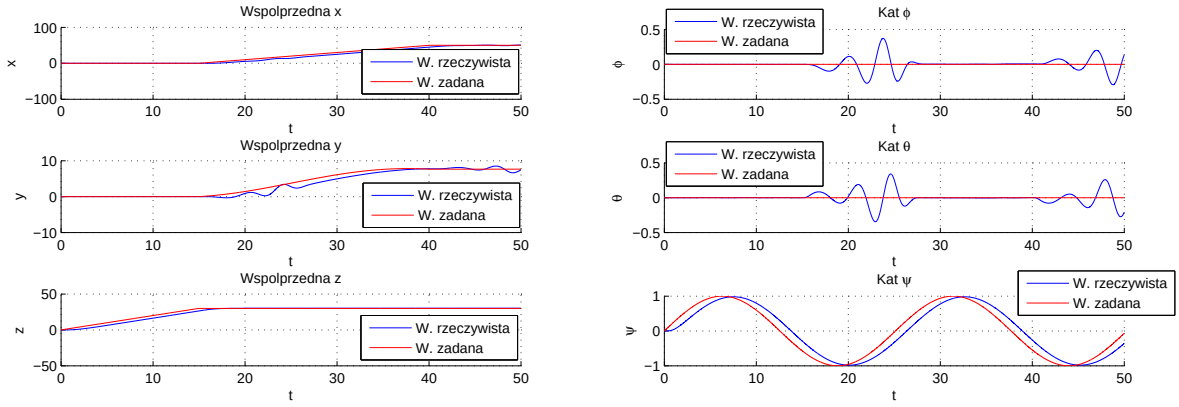
Rysunek 7.73. Błędy śledzenia trajektorii i kątów dla H_∞ podczas realizacji (7.12)-(7.14) – układ stabilny



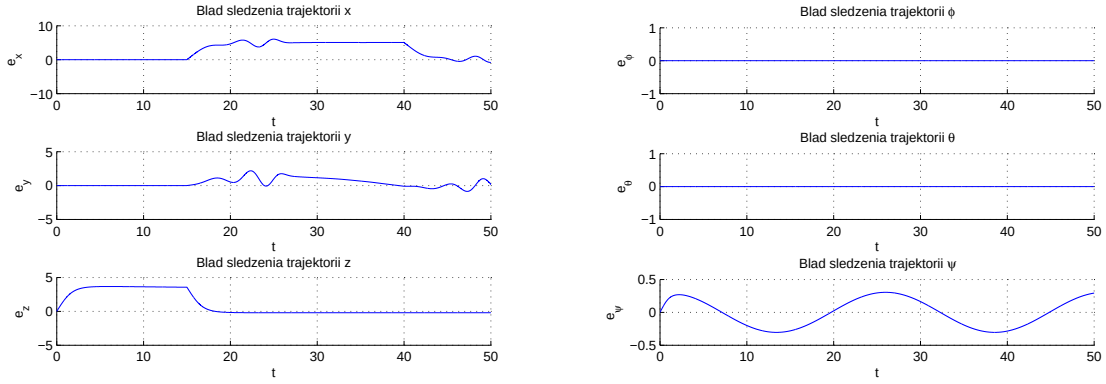
Rysunek 7.74. Zadawane sterowanie dla H_∞ podczas realizacji (7.12)-(7.14) – układ stabilny



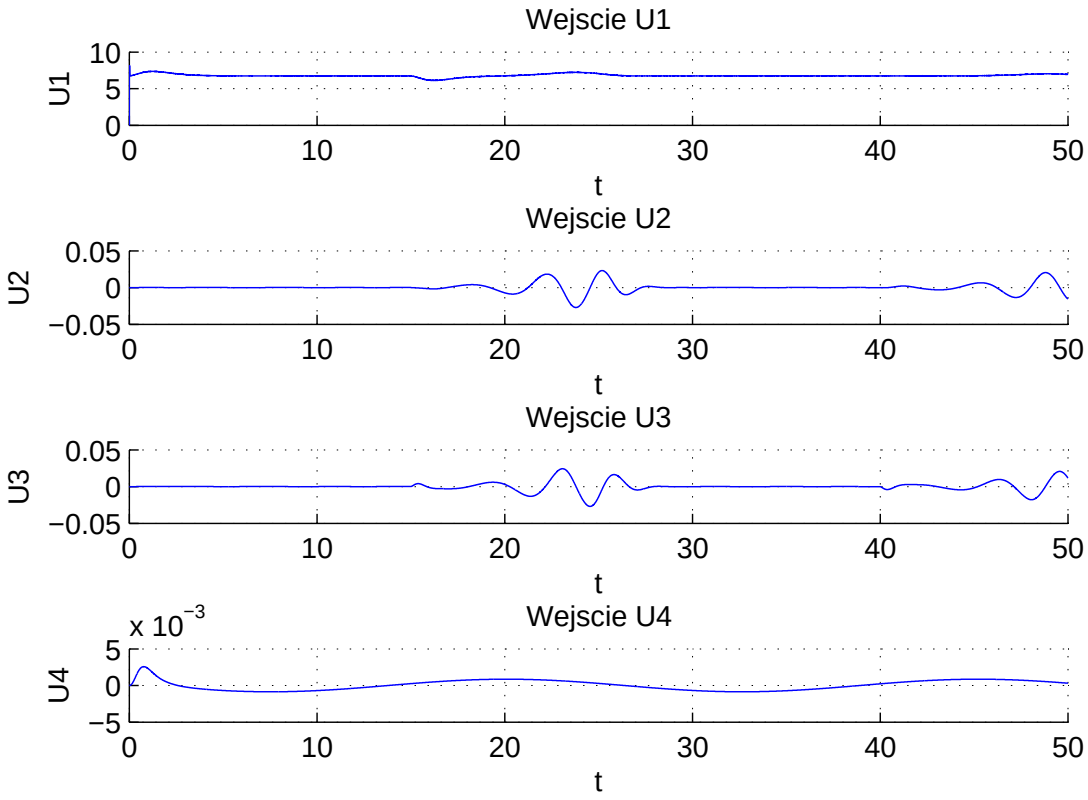
Rysunek 7.75. Trajektoria zadana i rzeczywista dla H_∞ podczas realizacji (7.12)-(7.14) – układ stabilny



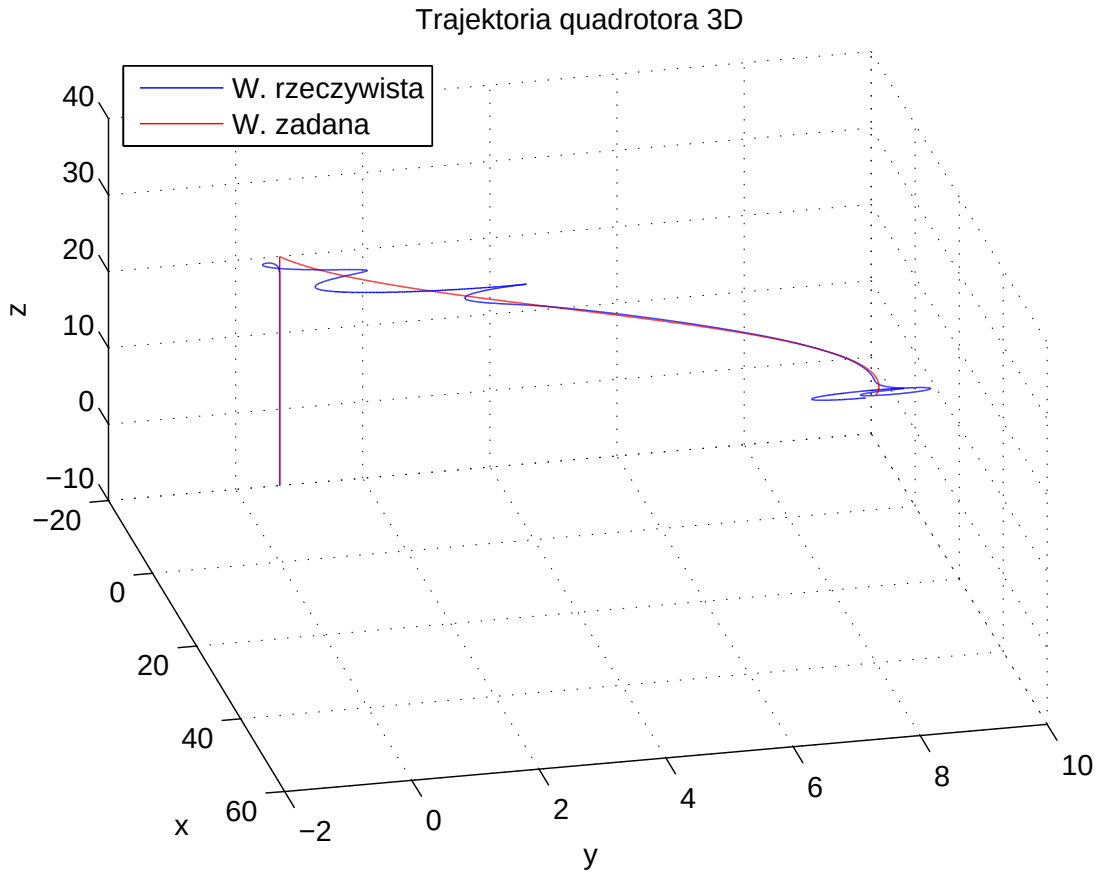
Rysunek 7.76. Położenia i kąty zadane i rzeczywiste (uzyskane symulacyjne) dla H_∞ podczas realizacji (7.12)-(7.14) z manipulacją kąta myszkowania – układ na granicy stabilności



Rysunek 7.77. Błędy śledzenia trajektorii i kątów dla H_∞ podczas realizacji (7.12)-(7.14) z manipulacją kąta myszkowania – układ na granicy stabilności



Rysunek 7.78. Zadawane sterowanie dla H_∞ podczas realizacji (7.12)-(7.14) z manipulacją kąta myszkowania – układ na granicy stabilności



Rysunek 7.79. Trajektoria zadana i rzeczywista dla H_∞ podczas realizacji (7.12)-(7.14) z manipulacją kąta myszkowania – układ na granicy stabilności

7.3.5. Podsumowanie

Przeprowadzone symulacje pokazują, że utworzony sterownik nie jest w stanie kontrolować wszystkich 4 stopni swobody qudrotora. W przypadku sterowania jedynie jego położeniem (pomijając kąt myszkowania) sterownik jest stabilny i testy nie wykazały żadnych zachowań niestabilnych. Kąt myszkowania podczas tych symulacji pozostawał równy 0 (w granicach błędów numerycznych Simulinka).

W przypadku, gdy sterowano jednocześnie położeniem i kątem myszkowania w większości przeprowadzonych symulacji quadrotor zachowywał się niestabilnie. Objawiało się to nagłym wzrostem wartości bezwzględnej (do nieskończoności) przynajmniej jednej ze współrzędnych położenia quadrotora: x, y i z. Jednocześnie wartość kąta myszkowania jest zachowana i dalej podąża za zadaną trajektorią. Próby wyeliminowania tego zjawiska nie powiodły się.

7.4. Badania metod numerycznych

Przeprowadzono badania różnych metod numerycznych rozwiązywania układu równań różniczkowych dostępnych w programie `Matlab` (Tab. 7.1). Obiektem badania był zaimplementowany wcześniej model quadrotora 6.2.

Tabela 7.1. Badane metody całkowania

Nazwa	Metoda
`ode1`	Euler
`ode4`	Runge-Kutta
`ode8`	Dormand-Prince
`ode45`	Dormand-Prince
`ode23`	Bogacki-Shampine
`ode23s`	mod. dla równań sztywnych

7.4.1. Stałe sterowania

Pierwszym podejściem do badań było zastosowanie stałych sterowań oraz różnych warunków początkowych. Przykładowo sterowanie (7.16) dla zerowych prędkości początkowych powoduje zawis quadrotora w punkcie.

$$\begin{cases} U_1 = & g * m \\ U_2 = & 0 \\ U_3 = & 0 \\ U_4 = & 0 \end{cases} \tag{7.16}$$

W większości przypadków (różne położenia początkowe oraz różne metody) układ pozostawał w pozycji początkowej. Jedynym ciekawym przypadkiem była sytuacja kiedy wszystkie położenia początkowe były równe zero. Układ (niezależnie od metody całkowania) wzbudzał się w osi OZ. Mogło to być spowodowane niedokładnością przybliżenia funkcji cos w punkcie 0.W pozostałych równaniach występują zera bez przybliżeń, co powoduje brak problemów z całkowaniem.

W następnym podejściu spróbowano zadać stałe sterowanie powodujące utrzymanie stałej wysokości, ale przy różnych od zera wartościach początkowych dla orientacji (układ powinien "uciekać" w płaszczyźnie prostopadłej do grawitacji). Równanie (7.17) reprezentuje zastosowane sterowanie.

$$\begin{cases} U_1 = & \frac{g*m}{\cos\phi\cos\theta} \\ U_2 = & 0 \\ U_3 = & 0 \\ U_4 = & 0 \end{cases} \tag{7.17}$$

W tym przypadku możliwe było analityczne wyliczenie wartości przesunięcia na płaszczyźnie jakie powinno zostać osiągnięte przez układ po czasie równym T. Otrzymaną wartość porównano z wartościami otrzymanymi podczas symulacji. W przypadku każdej z metod oraz różnych wartości początkowych układ dążył do spodziewanych wartości.

Następnie postanowiono przetestować sekwencję stałych sterowań realizujących zadanie:

1. zawis,
2. przechył,
3. lot,
4. przechył,
5. zawis.

Porównano drogę przebytą przez układ wyliczoną przy pomocy różnych metod całkowania. We wszystkich przypadkach otrzymano ten sam wynik z tą samą dokładnością.

Obserwacje zachowania układu w przypadku stałych sterowań nie przyniosła żadnych ciekawych spostrzeżeń. Odpowiedzią jest równanie dynamiki, z którego wynika, że w przypadku stałych sterowań większość zmiennych zeruje się, a układ pozostaje w spoczynku. Nie zaobserwowano również samoczynnego wzbudzenia się stosowanych metod numerycznych.

Zaobserwowano również, iż metody w przypadku braku ustalonego kroku całkowania (metody stałokrokowe) lub maksymalnego kroku całkowania (metody zmiennokrokowe) optymalizowały ilość kroków do ilości zmian sterowania. Czyli, np. w przypadku sekwencji stałych sterowań metody stałokrokowe wyliczały wartości tylko w punktach zmiany sterowania, a cała symulacja odbywała się w kilku krokach.

Następnym krokiem badań było zastosowanie sterowań zmiennych.

7.4.2. Sterowanie zmienne

Aby móc dobrze porównywać metody całkowania numerycznego należało zastosować zmienne sterowanie – musiało być ono w dodatku odpowiednio szybko zmienne, dzięki temu można obserwować błędy wynikające z zastosowania różnych metod.

Zdecydowano że na model zostanie podane sterowanie postaci:

$$U_1 = 5 * \sin(\frac{t * 10}{\pi}) + m * g \tag{7.18}$$

Dzięki temu sterowaniu quadrotor zawieszony poziomo powinien oscylować wokół położenia równowagi lekko spadając. Jednakże aby uzyskać odpowiednio zmienną trajektorię do badań zdecydowano się zastosować warunki początkowe:

$$\begin{cases} \phi_0 = & \frac{\pi}{4} \\ \theta_0 = & \frac{\pi}{4} \\ \psi_0 = & 0 \end{cases} \tag{7.19}$$

Dzięki temu uzyskano odpowiednio szybką zmianę trajektorii możliwą do zastosowania w analizowaniu algorytmów.

7.4.3. Wyniki badań

Badania zostały przeprowadzone zarówno dla metod całkowania stałokrokowych (takich jak metody Matlaba (`ode1, ode4, ode8`), jak też zmiennokrokowych (`ode23, ode23s, ode45`).

Badanymi czynnikami dzięki którym porównano algorytmy były:

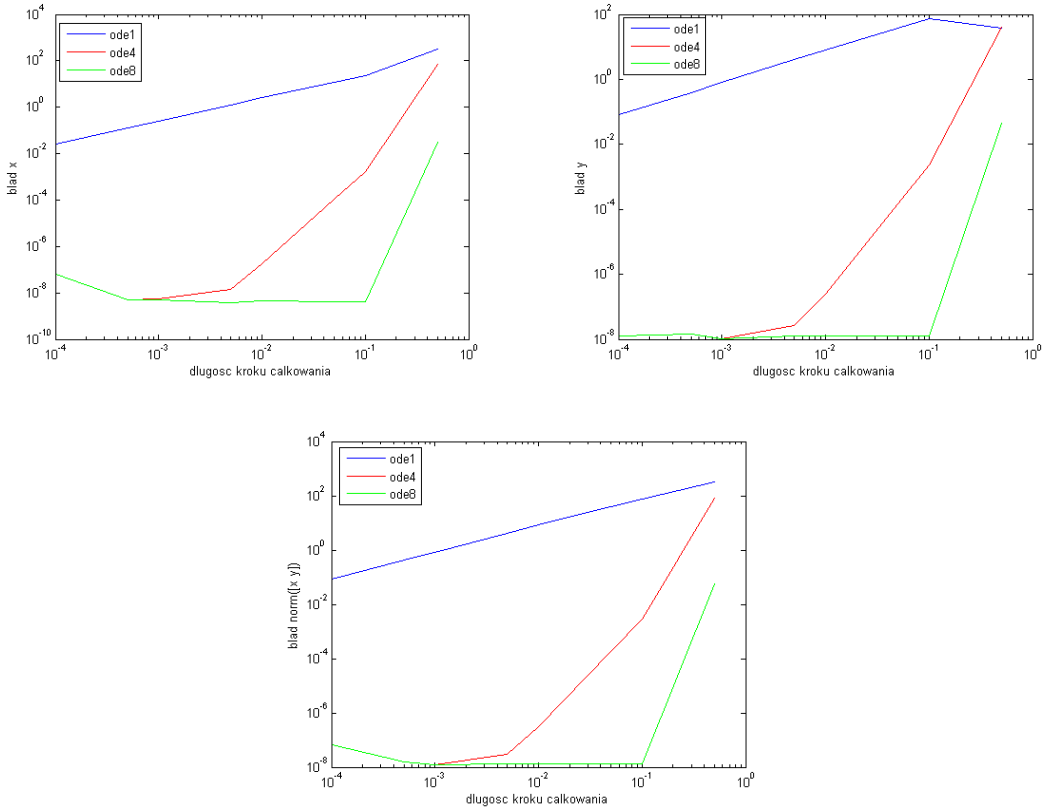
— czas symulacji – im bardziej skomplikowany model tym więcej czasu na wykonanie się potrzebuje w związku z tym przebadano czasy wykonywania, uzyskane wyniki zostały uśrednione w celu zwiększenia miarodajności.
— błąd położenia XY – porównywano końcowe stany osiągane przez model a następnie wyznaczano ich normę, co pozwoliło na oszacowanie dokładności zadanego algorytmu całkowania.
— błąd trajektorii – aproksymowany za pomocą całki z przebiegu.

Aby móc liczyć błędy, odwoływano się do metody która w testach pierwotnych okazała się najbardziej wiarygodna. Do porównywania algorytmów wykorzystano więc metodę `ode23` z następującymi parametrami: tolerancja błędu = 10^{-12}, maksymalny krok całkowania = 10^{-3}. Z takimi parametrami zasymulowano układ i wyznaczono przebiegi wzorcowe.

Stałokrokowe metody całkowania

Głównym i wspólnym parametrem metod stałokrokowych jest stały krok całkowania określany jako `fixed-step`. Porównano trzy metody całkowania (`ode1`, `ode4`, `ode8`) określając długość kroku.

Na wykresach 7.4.3 można zaobserwować spadek błędu wraz ze zmniejszającym się krokiem, jednak widać też zwiększenie się błędu dla najmniejszych wartości kroku - jest to spowodowane niestabilnością rozwiązywanego problemu, widać to szczególnie dzięki drugiemu użytemu wskaźnikowi błędu (rys. 7.4.3) załamanie występuje tam dla innej długości kroku. Metoda `ode1` wykazuje większe błędy w porównaniu do dwóch pozostałych metod całkowania. Różnicę widać również w przypadku metod `ode4` i `ode8`, jednak można zaobserwować że dla odpowiednio dużego kroku te błędy są tego samego rzędu, co pozwala wybrać metodę prostszą metodę `ode4`.



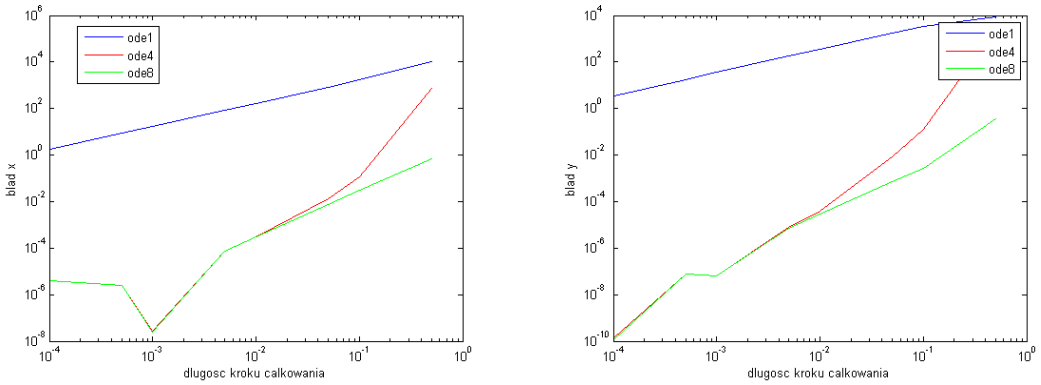
Rysunek 7.80. Porównanie błędów dla stałokrokowych metod całkowania (błąd celu trajektorii)

Podczas przeprowadzania symulacji dla złożonych modeli ważna jest nie tylko wiedza jak zwiększenie kroku wpływa na zmniejszenie błędu, ale też zależność pomiędzy błędami a czasami przeprowadzania symulacji – celem jest uzyskanie wiarygodnego wyniku, z jak najmniejszym błędem przy zachowaniu rozsądnego czasu całkowania. Te zależności widać na rys. 7.4.3. Dla metody `ode1` błąd przy narastającym czasie obliczeń zmniejsza się bardzo wolno, natomiast dla dwóch pozostałych metod znacznie szybciej.

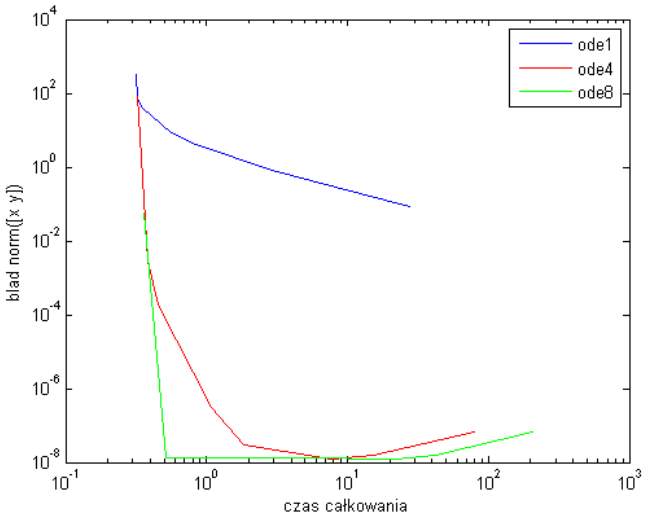
Na wykresach łatwo jest zauważyć ogólną tendencję, ale w danych zgromadzonych w tabelach 7.4.3 można zauważyć ciekawe zależności. Czas obliczeń dla tych samych długości kroku spełnia zależność $T_{ode1} < T_{ode4} < T_{ode8}$. Można zauważyć że przy krokach 0.01 i 0.001 dla metod `ode4` i `ode8` wartości obu błędów są porównywalne, a dodatkowo błąd położenia jest identyczny, co w tym przypadku oznacza że można zastosować metodę `ode4` przy kroku 0.01 uzyskując optymalny czas całkowania w stosunku do błędu.

Metody zmiennokrokowe

W tej części były porównywane metody zmiennokrokowe, w tym przypadku możemy algorytmy regulować za pomocą maksymalnego rozmiaru kroku (w Matlabie to parametr `MaxStepSize`) oraz tolerancji błędu (odpowiednio parametr `RelTol`).



Rysunek 7.81. Porównanie błędów dla stałokrokowych metod całkowania (błąd realizacji trajektorii)



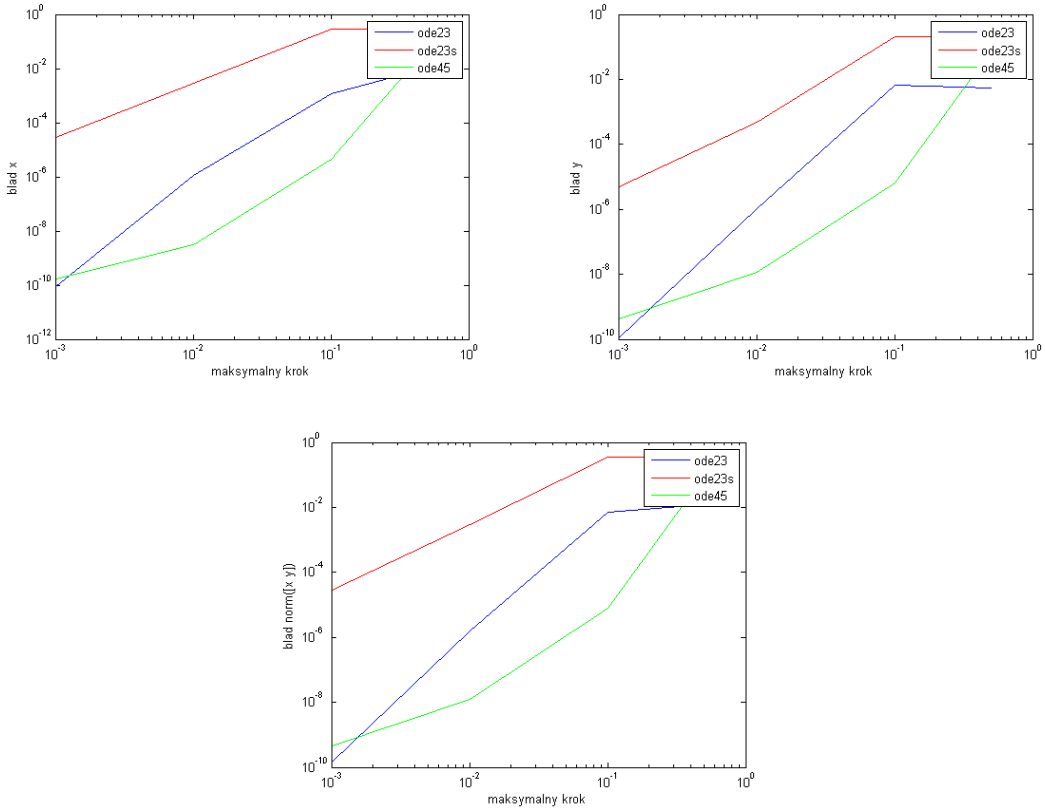
Rysunek 7.82. Wykres zależności czasu całkowania od błędu dla metod stałokrokowych

Metoda	Krok	Czas	errXY	errTraj
ode1	0.1	0.326	48.667	3514.4
ode1	0.01	0.555	5.21	358.523
ode1	0.001	2.915	0.525	35.919
ode4	0.1	0.377	0.0021	0.129
ode4	0.01	1.258	1.612e-05	7.161e-05
ode4	0.001	7.884	1.615e-05	1.118e-04
ode8	0.1	0.534	1.615e-05	0.0027
ode8	0.01	2.487	1.615e-05	8.387e-05
ode8	0.001	22.047	1.615e-05	1.118e-04

Tabela 7.2. Porównanie metod stałokrokowych

Dla metod zmiennokrokowych dokładność rośnie wraz ze zmniejszaniem się maksymalnej długości kroku. Na wykresach błędu punktu końcowego trajektorii (rys. 7.4.3) widać przewagę dla metody `ode45` nad `ode23`, jest jednak ona nieznaczna i przy najmniejszych ustawieniach

zmienia się na korzyść `ode23`. Natomiast metoda dla rozwiązywania równań sztywnych `ode23s` wykazuje wyraźnie większe błędy niż pozostałe metody.



Rysunek 7.83. Porównanie błędów dla zmiennokrokowych metod całkowania (błąd celu trajektorii)

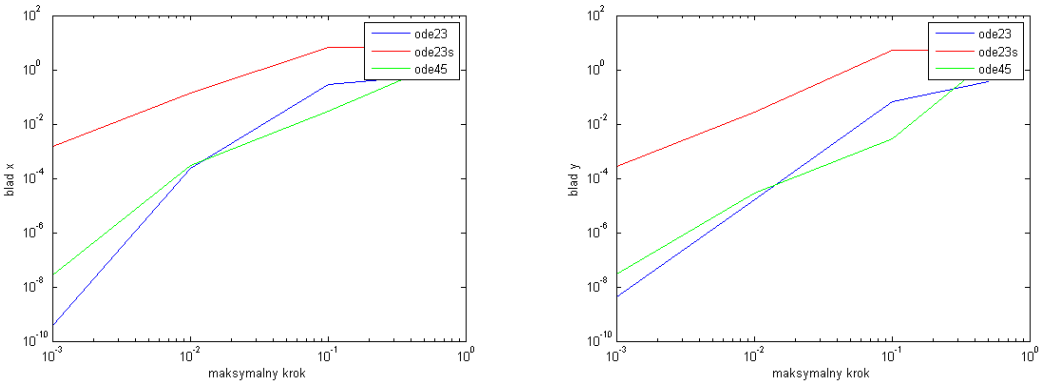
Zależność błędu od czasu całkowania (rys. 7.4.3) wskazuje że przy rosnącym czasie całkowania błąd najszybciej maleje w metodzie `ode45`, metoda `ode23` ma podobny przebieg, jednak gorsze dokładności (prócz bardzo małych kroków - wtedy tendencja się zmienia). Metoda `ode45` wykazuje znaczne błędy pomimo długich czasów obliczeń.

W tabelach wyników (7.4.3, 7.4.3) można zaobserwować że dla ustalonych parametrów (maksymalnego kroku i tolerancji) czasy całkowania spełniają zależność: $T_{ode23} < T_{ode45} < T_{ode23s}$. Natomiast błędy są największe dla `ode23s` najmniejsze zaś dla `ode45`. Błędy stają się porównywalne przy ustaleniu maksymalnego kroku rzędu 0.001 dla obu dokładności tolerancji – wpływ na dokładność ma maksymalny zadany krok.

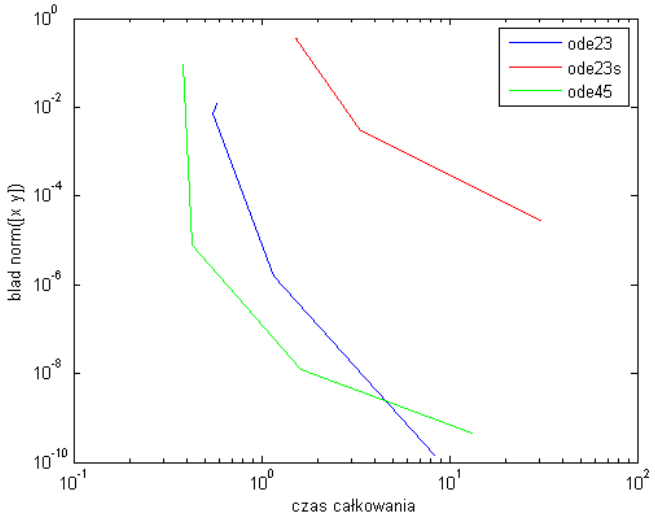
7.4.4. Wnioski

Spośród metod stałokrokowych najwydajniejszą okazała się być metoda `ode4`, zaś spośród metod zmiennokrokowych `ode23`. W przypadku metod stałokrokowych małe błędy uzyskiwano już dla długości kroku $0.01s$. Dalsze zmniejszanie długości kroku nie powodowało zmniejszenia poziomu błędu, tylko wydłużało czas obliczeń. W przypadku metod zmiennokrokowych osiągnięcie niskich błędów wymaga ustalenia maksymalnej długości kroku na poziomie zbliżonym do kroku metod stałokrokowych, czyli $0.01s$. Co ciekawe, szczególnie niezadowalające wyniki otrzymano dla opcji „auto”, która całkowicie automatycznie dobiera długości kroków.

Niezadowalające wyniki uzyskano również dla metody `ode23s`, czyli zmodyfikowanej metody `ode23` przeznaczonej do rozwiązywania układów sztywnych jakim jest nasz model quadrotora.



Rysunek 7.84. Porównanie błędów dla zmiennokrokowych metod całkowania (błąd realizacji trajektorii)



Rysunek 7.85. Wykres zależności czasu całkowania od błędu dla metod zmiennokrokowych

Metoda	Czas	errXY	errTraj
ode45	0.555	1.586e-05	0.0027
ode23	0.521	0.044	0.225
ode23s	0.811	1.789	32.102

Tabela 7.3. Maksymalny Krok: *auto*

Metoda	Czas	errXY	errTraj
ode45	0.55	1.586e-05	0.0027
ode23	0.505	0.044	0.225
ode23s	0.79	1.789	32.102

Tabela 7.4. Maksymalny Krok: 0.1

Metoda	Czas	errXY	errTraj
ode45	1.886	1.615e-05	8.379e-05
ode23	1.39	1.611e-05	1.288e-04
ode23s	3.75	0.002	0.027

Tabela 7.5. Maksymalny Krok: 0.01

Metoda	Czas	errXY	errTraj
ode45	14.944	1.616e-05	1.119e-04
ode23	9.616	1.616e-05	1.119e-04
ode23s	33.599	2.93e-05	1.591e-04

Tabela 7.6. Maksymalny Krok: 0.001

Tabela 7.7. Wyniki dla tolerancji $1.0 * 10^{-3}$

Metoda	Czas	errXY	errTraj
ode45	0.54	1.817e-05	0.003
ode23	0.719	0.004	0.067
ode23s	1.827	0.25	5.424

Tabela 7.8. Maksymalny Krok: *auto*

Metoda	Czas	errXY	errTraj
ode45	0.54	1.817e-05	0.003
ode23	0.723	0.004	0.067
ode23s	1.76	0.25	5.424

Tabela 7.9. Maksymalny Krok: 0.1

Metoda	Czas	errXY	errTraj
ode45	1.863	1.615e-05	8.393e-05
ode23	1.392	1.611	1.276
ode23s	3.723	0.002	0.027

Tabela 7.10. Maksymalny Krok: 0.01

Metoda	Czas	errXY	errTraj
ode45	14.996	1.616e-05	1.119e-04
ode23	9.957	1.616e-05	1.118e-04
ode23s	34.305	2.931e-05	1.591e-04

Tabela 7.11. Maksymalny Krok: 0.001

Tabela 7.12. Wyniki dla tolerancji $1.0 * 10^{-5}$

Z badań wynika, że dobre wyniki można uzyskać przy pomocy zarówno metod stałokrokowych jak i zmiennokrokowych. Oba typy metod posiadają swoje wady i zalety, które wynikają ze zmiennej długości kroku. Metody polecane do wykonywania badań to: `ode23` z metod zmiennokrokowych i `ode4` z metod stałokrokowych. Dla tych metod należy eksperymentalnie dobrać krok, który może się różnić w zależności od zastosowanego sterownika i zadanej trajektorii. W przypadku braku ograniczeń dotyczących stałości długości kroku całkowania zaleca się wybór metody `ode23` z ustawionym maksymalnym krokiem całkowania.

7.5. Badania porównanwcze algorytmów sterowania

7.5.1. Wstęp

Aby móc porównać jakość sterowania za pomocą poszczególnych algorytmów, przeprowadzono szereg symulacji dla różnych trajektorii (każdy z algorytmów miał sterować obiektem na identycznym zestawie trajektorii zadanych). Po każdej symulacji zebrano zestaw wskaźników jakości oraz maksymalne absolutne wartości sterowań obiektu, które posłużyły do późniejszej oceny jakości sterowania. Trajektorie zadane były uzyskiwane na dwa sposoby:
— za pomocą generatora trajektorii,
— jako trajektorę zadaną przypisywano trajektorię uzyskaną z dynamiki obiektu.
Dla wszystkich symulacji przyjęto nastepujące warunki początkowe:

$$\begin{cases} x = 0 \\ y = 0 \\ z = 0 \\ \phi = 0 \\ \theta = 0 \\ \psi = 0 \end{cases} \tag{7.20}$$

7.5.2. Porównanie algorytmów sterowania

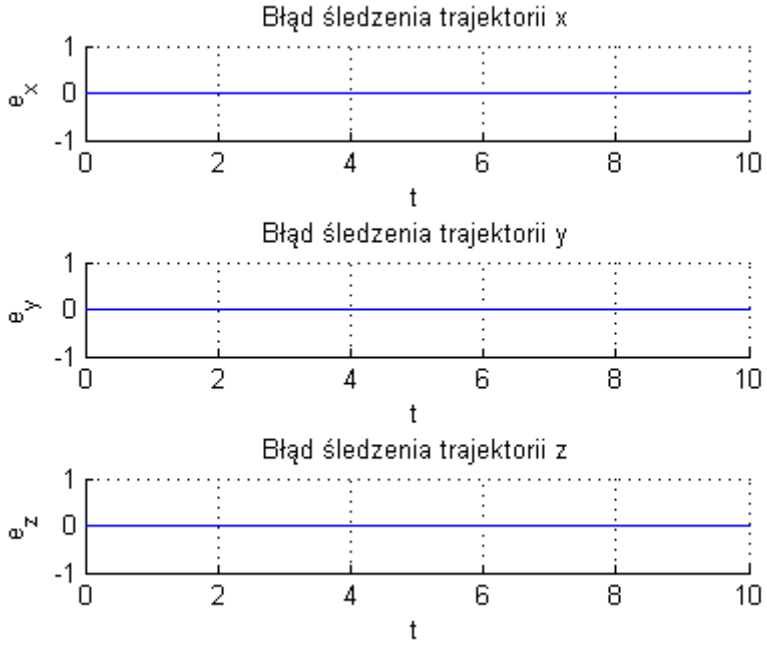
Sterowanie w zawisie

Trajektoria zadana została opisana następującym zestawem równań:

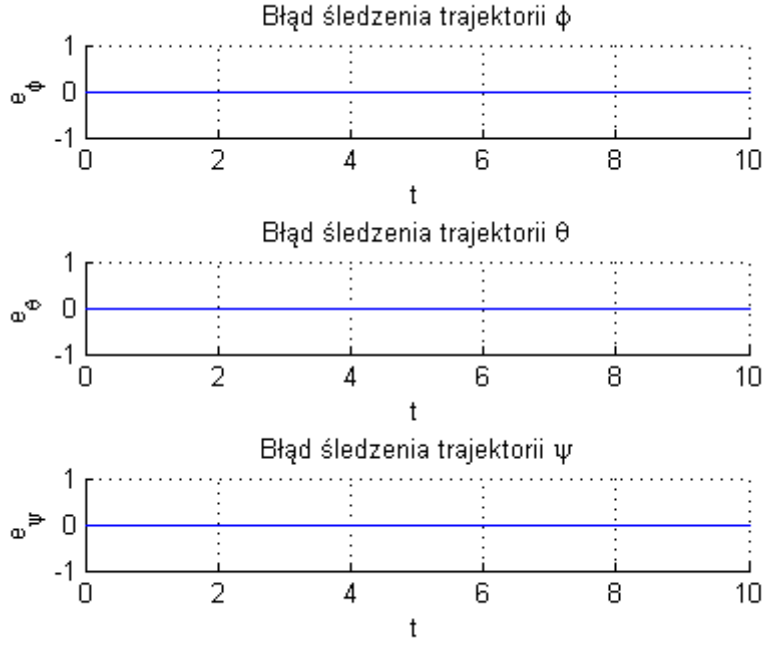
$$\begin{cases} x_d = 0 \\ y_d = 0 \\ z_d = 0 \\ \phi_d = 0 \\ \theta_d = 0 \\ \psi_d = 0 \end{cases} \tag{7.21}$$

Czas sterowania ustalono na 10s, a krok czasowy na 0,01s. Poszczególne wskaźniki jakości dla algorytmu sterowania PID zebrano w tabeli 7.13, dla całkowania wstecznego w 7.14, a dla H_∞ w 7.15. Błędy poszczególnych zmiennych wewnętrznych podczas sterowania dla algorytmu PID przedstawione są na wykresach 7.86 i 7.87, dla algorytmu całkowania wstecznego na 7.88 i 7.89, a dla H_∞ na 7.90 i 7.91.

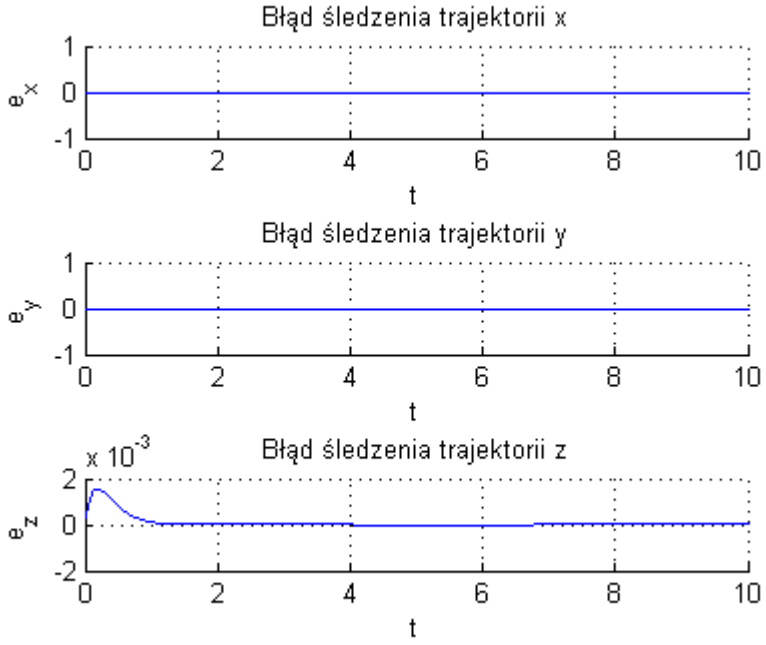
Maksymalne absolutne wartości wejść dla algorytmu PID:
— $max(U_1) = 6,73$
— $max(U_2) = 0$
— $max(U_3) = 0$
— $max(U_4) = 0$
Maksymalne absolutne wartości wejść dla algorytmu całkowania wstecznego:
— $max(U_1) = 6,859$
— $max(U_2) = 0$
— $max(U_3) = 0$
— $max(U_4) = 0$
Maksymalne absolutne wartości wejść dla algorytmu H_∞:
— $max(U_1) = 8,133$
— $max(U_2) = 4,403e-015$
— $max(U_3) = 1.816e-014$
— $max(U_4) = 3.101e-014$



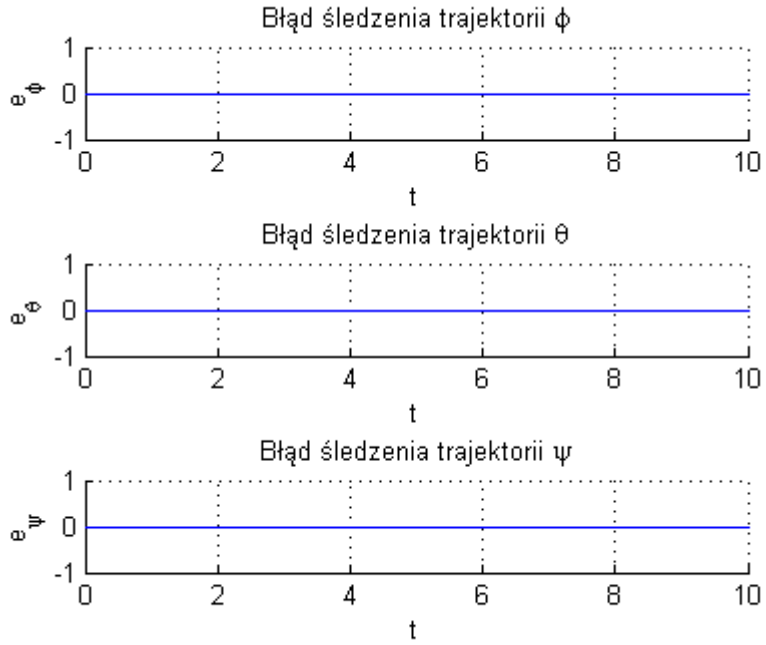
Rysunek 7.86. Błędy współrzędnych położenia - algorytm PID, sterowanie w zawisie



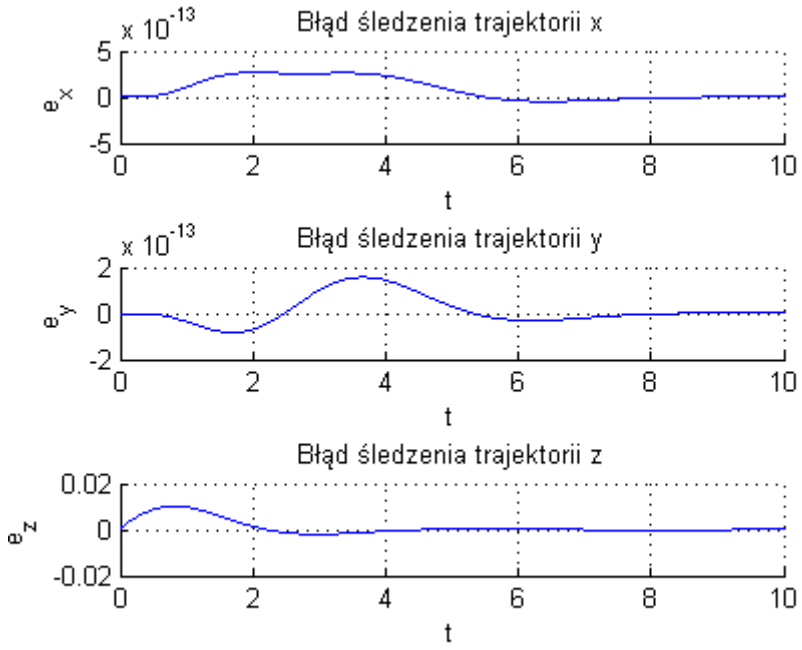
Rysunek 7.87. Błędy współrzędnych orientacji - algorytm PID, sterowanie w zawisie



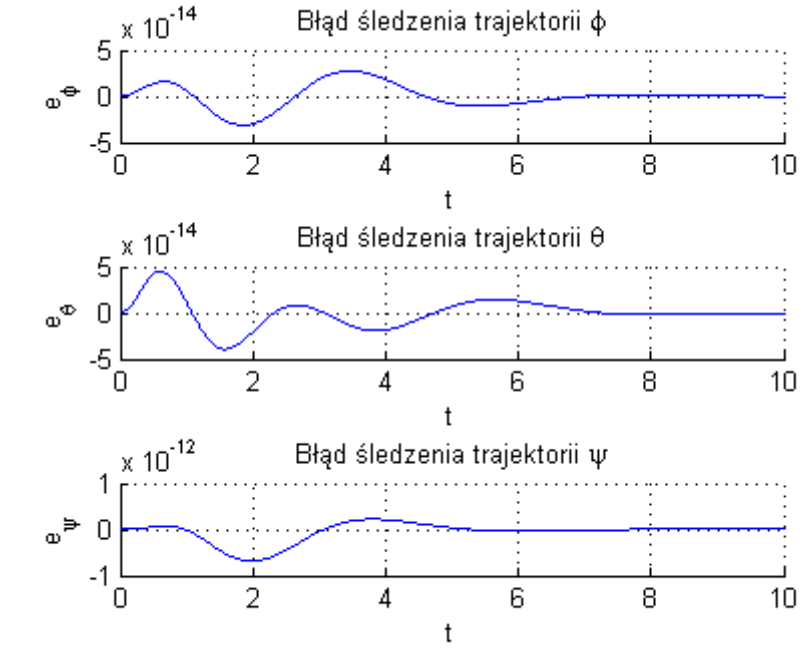
Rysunek 7.88. Błędy współrzędnych położenia - algorytm całkowania wstecznego, sterowanie w zawisie



Rysunek 7.89. Błędy współrzędnych orientacji - algorytm całkowania wstecznego, sterowanie w zawisie



Rysunek 7.90. Błędy współrzędnych położenia - algorytm H_∞, sterowanie w zawisie



Rysunek 7.91. Błędy współrzędnych orientacji - algorytm H_∞, sterowanie w zawisie

Tabela 7.13. Poszczególne wskaźniki jakości dla algorytmu PID - sterowanie w zawisie

	e_{MIN}	e_{MAX}	e_{MEAN}	ISE	IAE	ITAE
x	0	0	0	0	0	0
y	0	0	0	0	0	0
z	0	0	0	0	0	0
ϕ	0	0	0	0	0	0
θ	0	0	0	0	0	0
ψ	0	0	0	0	0	0

Tabela 7.14. Poszczególne wskaźniki jakości dla algorytmu całkowania wstecznego - sterowanie w zawisie

	e_{MIN}	e_{MAX}	e_{MEAN}	ISE	IAE	ITAE
x	0	0	0	0	0	0
y	0	0	0	0	0	0
z	-1.46276e-012	0.00155312	8.07265e-005	8.56317e-005	0.0808072	0.0309876
ϕ	0	0	0	0	0	0
θ	0	0	0	0	0	0
ψ	0	0	0	0	0	0

Tabela 7.15. Poszczególne wskaźniki jakości dla algorytmu H_∞ - sterowanie w zawisie

	e_{MIN}	e_{MAX}	e_{MEAN}	ISE	IAE	ITAE
x	-4.68583e-014	2.67827e-013	1.01513e-013	2.14456e-023	1.01615e-010	3.24995e-010
y	-8.32453e-014	1.57104e-013	4.12455e-014	3.90828e-024	4.12867e-011	1.49725e-010
z	-0.00196796	0.0101557	0.00164948	0.0109402	1.65113	2.34826
ϕ	-3.12133e-014	2.73727e-014	8.99668e-015	1.67662e-025	9.00568e-012	2.82039e-011
θ	-3.92615e-014	4.51438e-014	1.05455e-014	2.41099e-025	1.05561e-011	3.05078e-011
ψ	-6.85414e-013	2.21231e-013	1.19878e-013	4.86245e-023	1.19998e-010	3.13954e-010

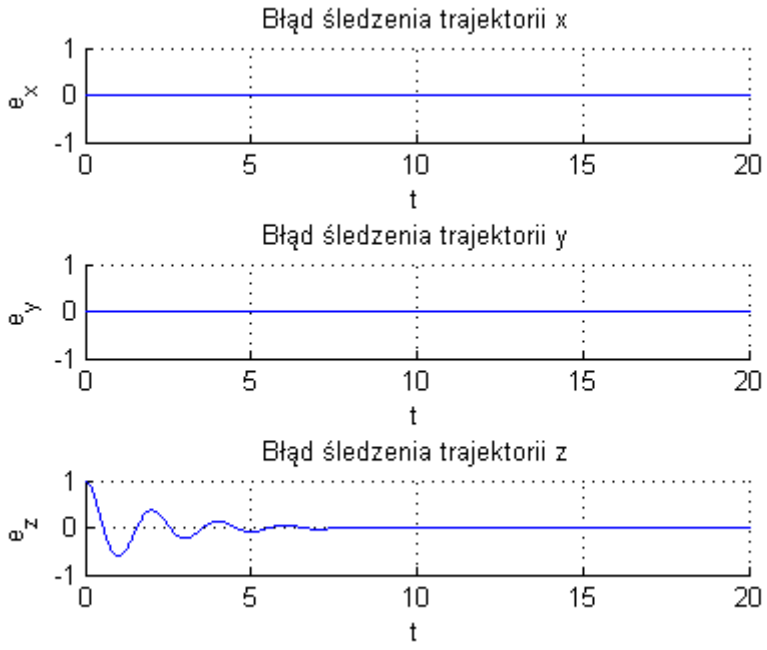
Sterowanie do punktu

Trajektoria zadana została opisana następującym zestawem równań:

$$\begin{cases} x_d = 0 \\ y_d = 0 \\ z_d = 1 \\ \phi_d = 0 \\ \theta_d = 0 \\ \psi_d = 0 \end{cases} \tag{7.22}$$

Czas sterowania ustalono na 20s, a krok czasowy na 0,01s. Poszczególne wskaźniki jakości dla algorytmu sterowania PID zebrano w tabeli 7.16, dla całkowania wstecznego w 7.17, a dla H_∞ w 7.18. Błędy poszczególnych zmiennych wewnętrznych podczas sterowania dla algorytmu PID przedstawione są na wykresach 7.92 i 7.93, dla algorytmu całkowania wstecznego na 7.94 i 7.95, a dla H_∞ na 7.96 i 7.97.

Maksymalne absolutne wartości wejść dla algorytmu PID:

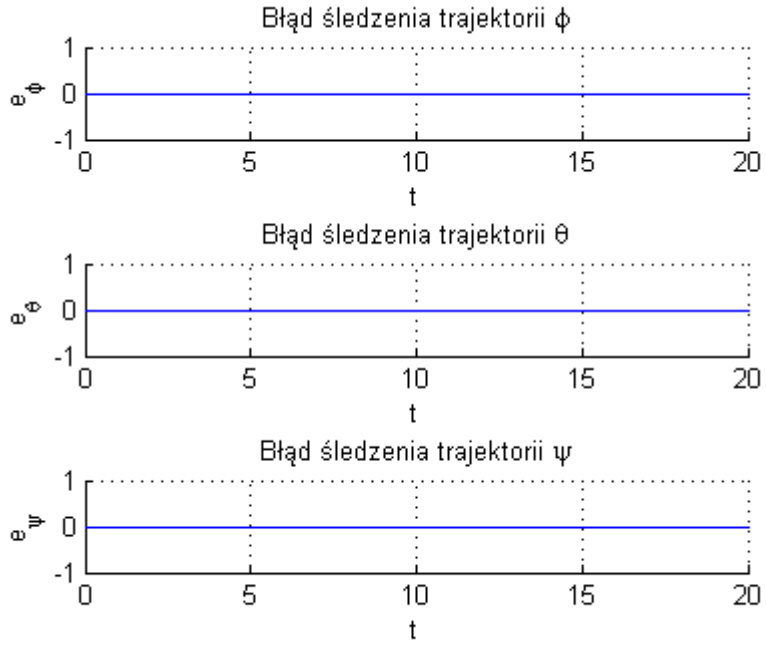


Rysunek 7.92. Błędy współrzędnych położenia - algorytm PID, sterowanie do punktu

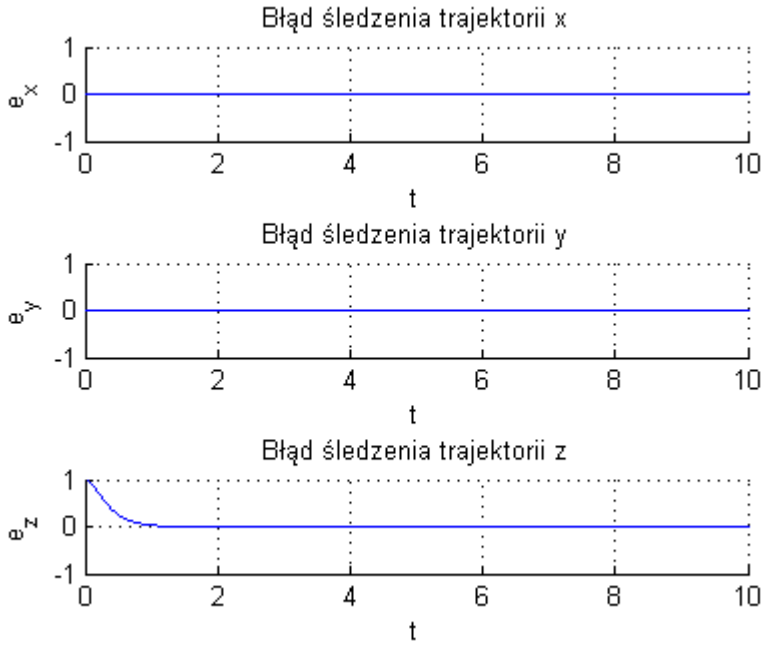
— $max(U_1) = 13,59$
— $max(U_2) = 0$
— $max(U_3) = 0$
— $max(U_4) = 0$

Maksymalne absolutne wartości wejść dla algorytmu całkowania wstecznego:
— $max(U_1) = 23,23$
— $max(U_2) = 0$
— $max(U_3) = 0$
— $max(U_4) = 0$

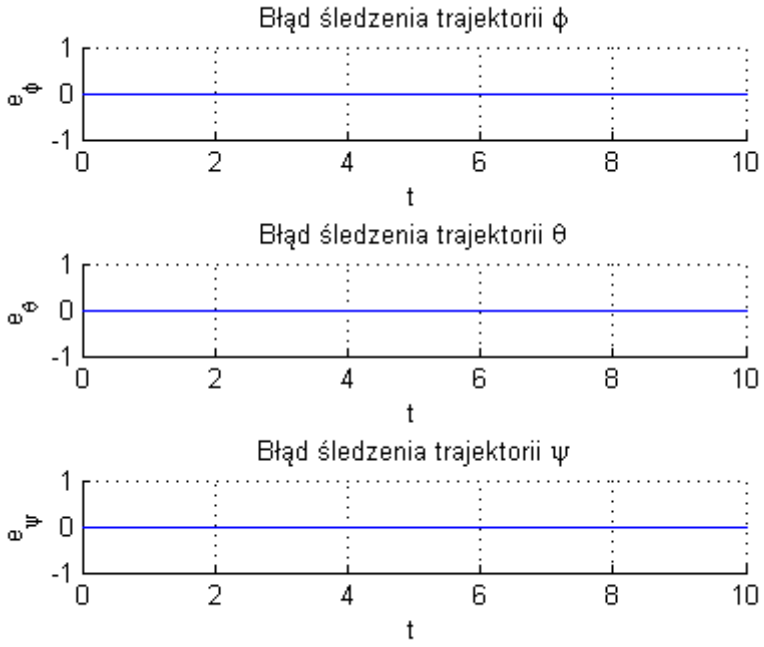
Maksymalne absolutne wartości wejść dla algorytmu H_∞:
— $max(U_1) = 8,238$
— $max(U_2) = 2,192e-013$



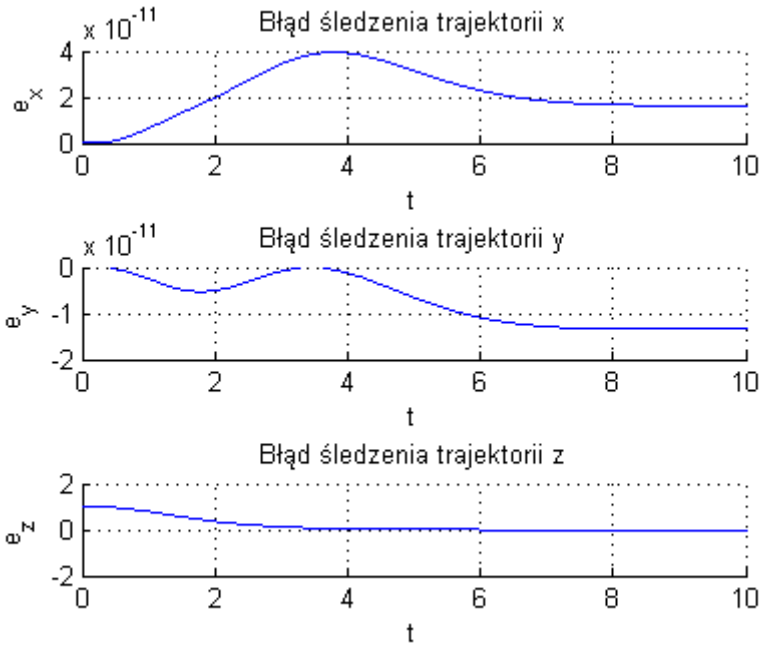
Rysunek 7.93. Błędy współrzędnych orientacji - algorytm PID, sterowanie do punktu



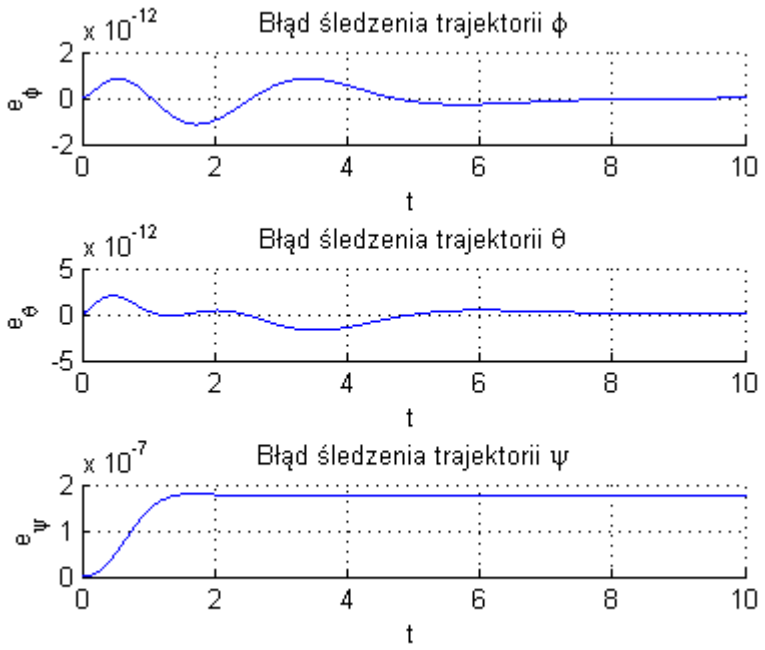
Rysunek 7.94. Błędy współrzędnych położenia - algorytm całkowania wstecznego, sterowanie do punktu



Rysunek 7.95. Błędy współrzędnych orientacji - algorytm całkowania wstecznego, sterowanie do punktu



Rysunek 7.96. Błędy współrzędnych położenia - algorytm H_∞, sterowanie do punktu



Rysunek 7.97. Błędy współrzędnych orientacji - algorytm H_∞, sterowanie do punktu

— $max(U_3) = 6,263e-013$
— $max(U_4) = 7,482e-009$

Tabela 7.16. Poszczególne wskaźniki jakości dla algorytmu PID - sterowanie w zawisie

	e_{MIN}	e_{MAX}	e_{MEAN}	ISE	IAE	ITAE
x	0	0	0	0	0	0
y	0	0	0	0	0	0
z	-0.609199	1	0.0667595	56.0519	133.586	269.493
ϕ	0	0	0	0	0	0
θ	0	0	0	0	0	0
ψ	0	0	0	0	0	0

Śledzenie trajektorii zmiennej w czasie

Trajektoria zadana została opisana następującym zestawem równań:

$$\begin{cases} x_d = 0 \\ y_d = 0 \\ z_d = 1 - e^{-t} \\ \phi_d = 0 \\ \theta_d = 0 \\ \psi_d = -0,4\sin(\frac{t}{6}) \end{cases} \tag{7.23}$$

Czas sterowania ustalono na 20s, a krok czasowy na 0,01s. Poszczególne wskaźniki jakości dla algorytmu sterowania PID zebrano w tabeli 7.19, dla całkowania wstecznego w 7.20, a dla H_∞ w 7.21. Błędy poszczególnych zmiennych wewnętrznych podczas sterowania dla algorytmu PID przedstawione są na wykresach 7.98 i 7.99, dla algorytmu całkowania wstecznego na 7.100 i 7.101, a dla H_∞ na 7.102 i 7.103.

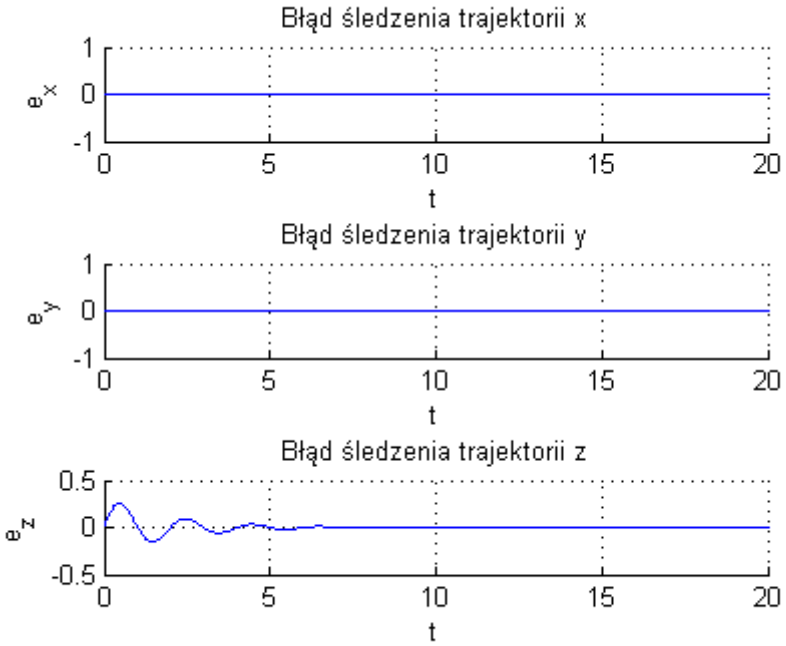
Maksymalne absolutne wartości wejść dla algorytmu PID:

Tabela 7.17. Poszczególne wskaźniki jakości dla algorytmu całkowania wstecznego - sterowanie w zawisie

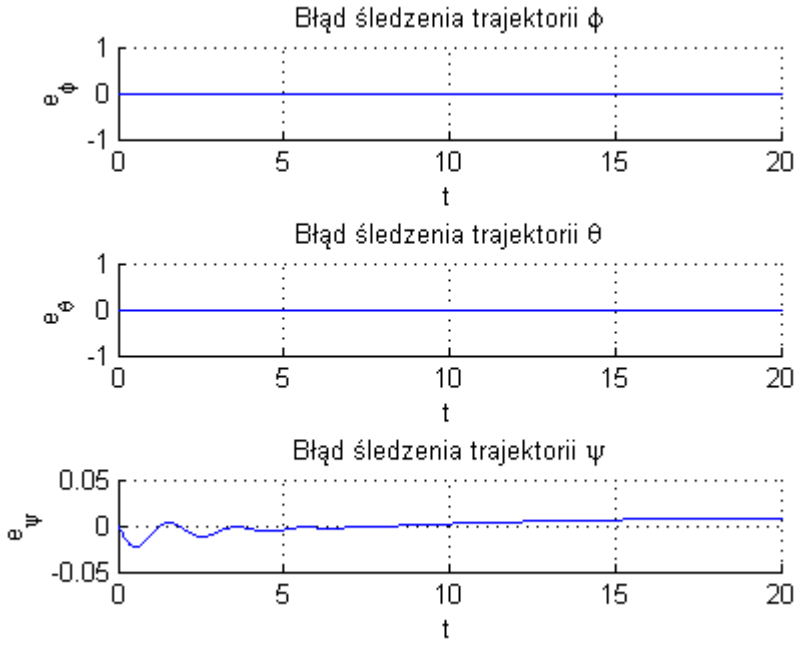
	e_{MIN}	e_{MAX}	e_{MEAN}	ISE	IAE	ITAE
x	0	0	0	0	0	0
y	0	0	0	0	0	0
z	-9.41319e-010	1	0.0388836	24.7334	38.9225	10.9318
ϕ	0	0	0	0	0	0
θ	0	0	0	0	0	0
ψ	0	0	0	0	0	0

Tabela 7.18. Poszczególne wskaźniki jakości dla algorytmu H_∞ - sterowanie w zawisie

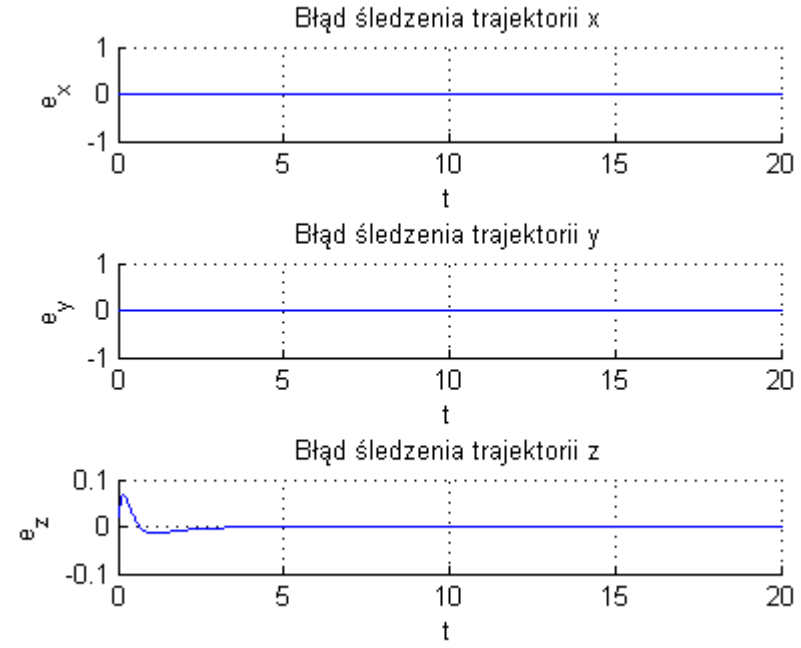
	e_{MIN}	e_{MAX}	e_{MEAN}	ISE	IAE	ITAE
x	0	3.94359e-011	2.13363e-011	5.68146e-019	2.13576e-008	1.09187e-007
y	-1.3438e-011	0	7.32458e-012	8.09516e-020	7.33191e-009	5.01987e-008
z	-0.00669964	1.00174	0.186333	129.511	186.519	237.449
ϕ	-1.13227e-012	8.48226e-013	3.3002e-013	2.13886e-022	3.3035e-010	9.8176e-010
θ	-1.72434e-012	2.03702e-012	5.17873e-013	5.82894e-022	5.18391e-010	1.73118e-009
ψ	0	1.79929e-007	1.64214e-007	2.85177e-011	0.000164378	0.000879168



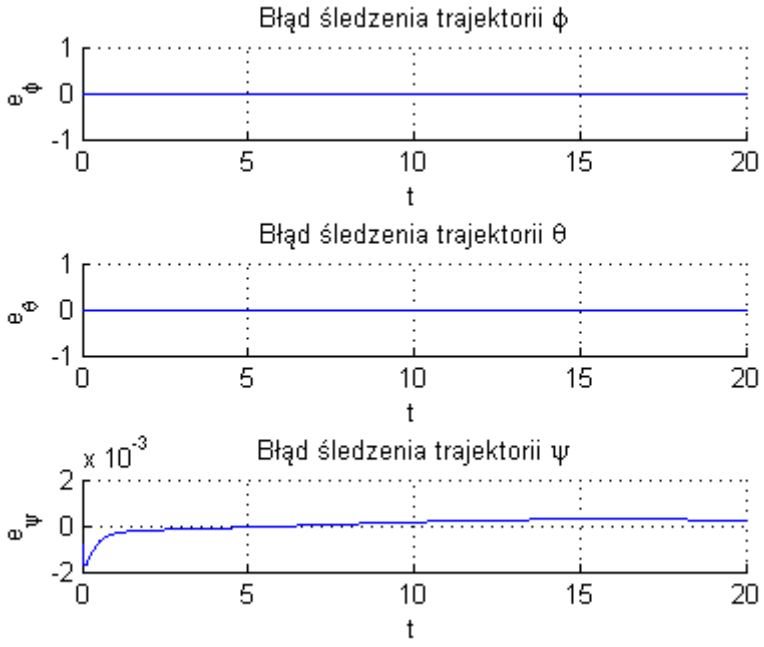
Rysunek 7.98. Błędy współrzędnych położenia - algorytm PID, śledzenie trajektorii zmiennej w czasie



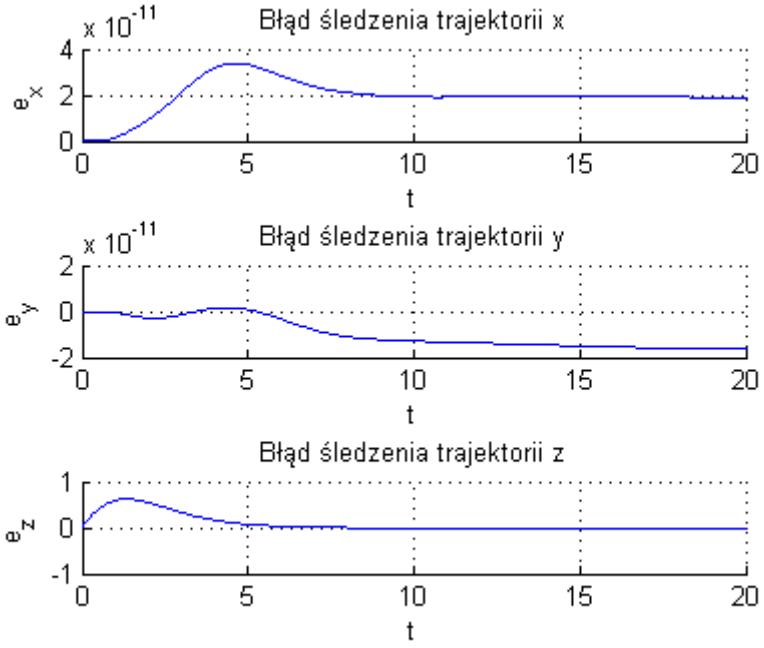
Rysunek 7.99. Błędy współrzędnych orientacji - algorytm PID, śledzenie trajektorii zmiennej w czasie



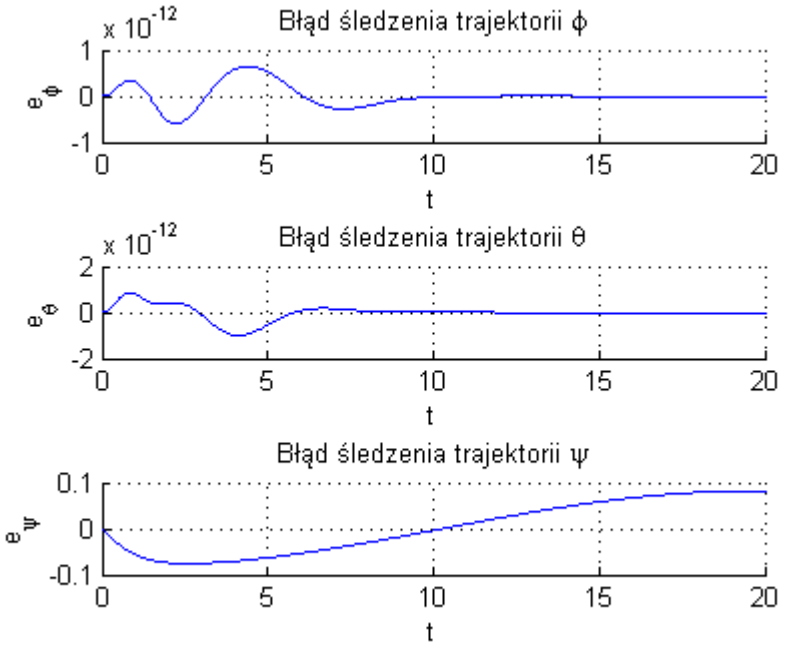
Rysunek 7.100. Błędy współrzędnych położenia - algorytm całkowania wstecznego, śledzenie trajektorii zmiennej w czasie



Rysunek 7.101. Błędy współrzędnych orientacji - algorytm całkowania wstecznego, śledzenie trajektorii zmiennej w czasie



Rysunek 7.102. Błędy współrzędnych położenia - algorytm H_∞, śledzenie trajektorii zmiennej w czasie



Rysunek 7.103. Błędy współrzędnych orientacji - algorytm H_∞, śledzenie trajektorii zmiennej w czasie

— $max(U_1) = 8,097$
— $max(U_2) = 0$
— $max(U_3) = 0$
— $max(U_4) = 2,403e-003$

Maksymalne absolutne wartości wejść dla algorytmu całkowania wstecznego:
— $max(U_1) = 13,08$
— $max(U_2) = 0$
— $max(U_3) = 0$
— $max(U_4) = 2,257e-002$

Maksymalne absolutne wartości wejść dla algorytmu H_∞:
— $max(U_1) = 8,134$
— $max(U_2) = 6,863e-014$
— $max(U_3) = 1,421e-013$
— $max(U_4) = 6,863e-004$

Tabela 7.19. Poszczególne wskaźniki jakości dla algorytmu PID - śledzenie trajektorii zmiennej w czasie

	e_{MIN}	e_{MAX}	e_{MEAN}	ISE	IAE	ITAE
x	0	0	0	0	0	0
y	0	0	0	0	0	0
z	-0.153622	0.255272	0.0209307	5.1313	41.8822	93.9549
ϕ	0	0	0	0	0	0
θ	0	0	0	0	0	0
ψ	-0.0229159	0.00708101	0.00505768	0.0817068	10.1204	101.254

Tabela 7.20. Poszczególne wskaźniki jakości dla algorytmu całkowania wstecznego - śledzenie trajektorii zmiennej w czasie

	e_{MIN}	e_{MAX}	e_{MEAN}	ISE	IAE	ITAE
x	0	0	0	0	0	0
y	0	0	0	0	0	0
z	-0.0138403	0.0664601	0.00229127	0.139386	4.58483	4.72204
ϕ	0	0	0	0	0	0
θ	0	0	0	0	0	0
ψ	-0.00177206	0.000282596	0.000216753	0.000173952	0.433723	4.25809

Tabela 7.21. Poszczególne wskaźniki jakości dla algorytmu H_∞ - śledzenie trajektorii zmiennej w czasie

	e_{MIN}	e_{MAX}	e_{MEAN}	ISE	IAE	ITAE
x	0	3.37942e-011	1.96327e-011	8.77709e-019	3.9285e-008	4.05165e-007
y	-1.62981e-011	1.62633e-012	9.81294e-012	2.66645e-019	1.96357e-008	2.61874e-007
z	-0.00681674	0.622129	0.0955024	80.1482	191.1	505.696
ϕ	-5.9741e-013	6.49564e-013	1.38976e-013	1.10332e-022	2.7809e-010	1.27251e-009
θ	-1.00278e-012	8.37092e-013	1.70342e-013	2.03446e-022	3.40855e-010	1.29399e-009
ψ	-0.0756237	0.0820699	0.050421	6.28684	100.892	1058.47

Śledzenie trajektorii zmiennej w czasie

Trajektoria zadana została uzyskana z dynamiki modelu, po zastosowaniu algorytmu całkowania wstecznego, dla trajektorii danej układem równań:

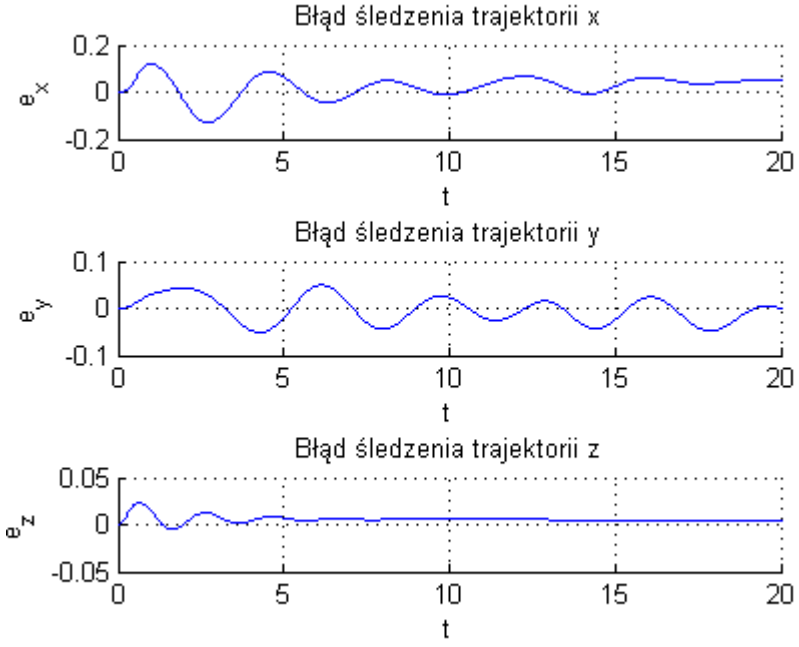
$$\begin{cases} x_d = 0.02t + 0.25 \\ y_d = 0.05t \\ z_d = 0.05t + 0.2\sin(\frac{t}{20}) \\ \phi_d = 0 \\ \theta_d = 0 \\ \psi_d = -0.8\sin(\frac{t}{6}) \end{cases} \tag{7.24}$$

Czas sterowania ustalono na 20s, a krok czasowy na 0,01s. Poszczególne wskaźniki jakości dla algorytmu sterowania PID zebrano w tabeli 7.22, dla całkowania wstecznego w 7.23, a dla H_∞ w 7.24. Błędy poszczególnych zmiennych wewnętrznych podczas sterowania dla algorytmu PID przedstawione są na wykresach 7.104 i 7.105, dla algorytmu całkowania wstecznego na 7.106 i 7.107, a dla H_∞ na 7.108 i 7.109.

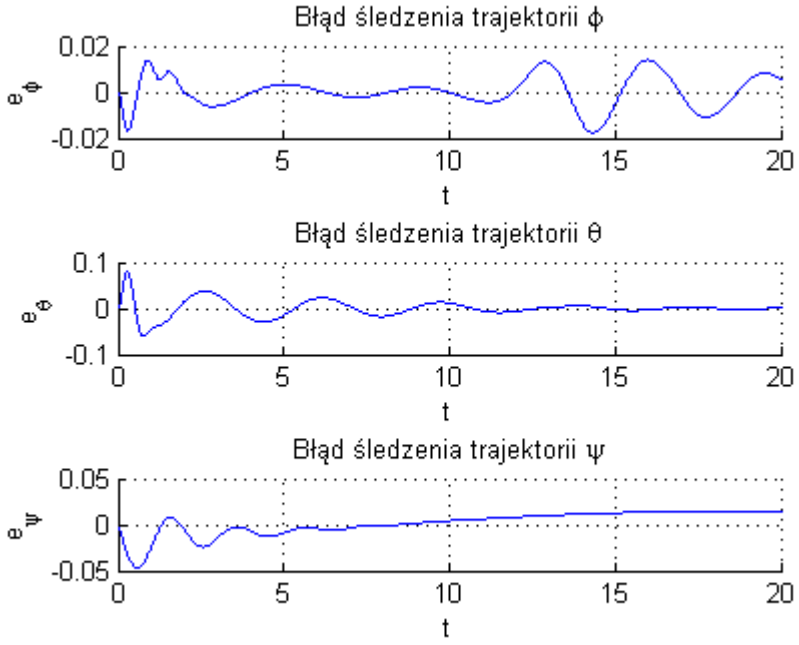
Maksymalne absolutne wartości wejść dla algorytmu PID:
- $max(U_1) = 6,853$
- $max(U_2) = 1,302e-003$
- $max(U_3) = 5,828e-003$
- $max(U_4) = 4,962e-003$

Maksymalne absolutne wartości wejść dla algorytmu całkowania wstecznego:
- $max(U_1) = 6,985$
- $max(U_2) = 2,718e-002$
- $max(U_3) = 8,994e-002$
- $max(U_4) = 2,183e-002$

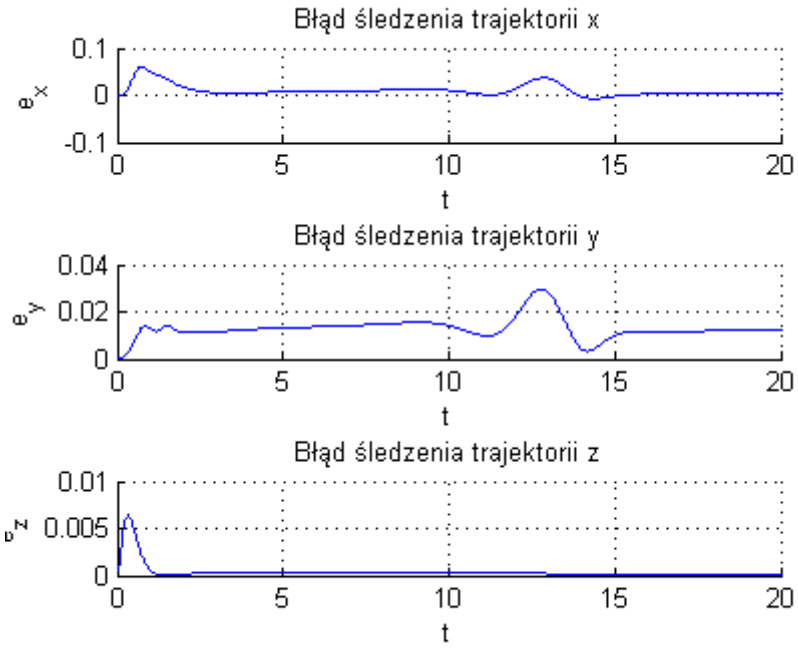
Maksymalne absolutne wartości wejść dla algorytmu H_∞:
- $max(U_1) = 8,133$
- $max(U_2) = 2,272e-004$
- $max(U_3) = 4,852e-004$
- $max(U_4) = 1,406e-003$



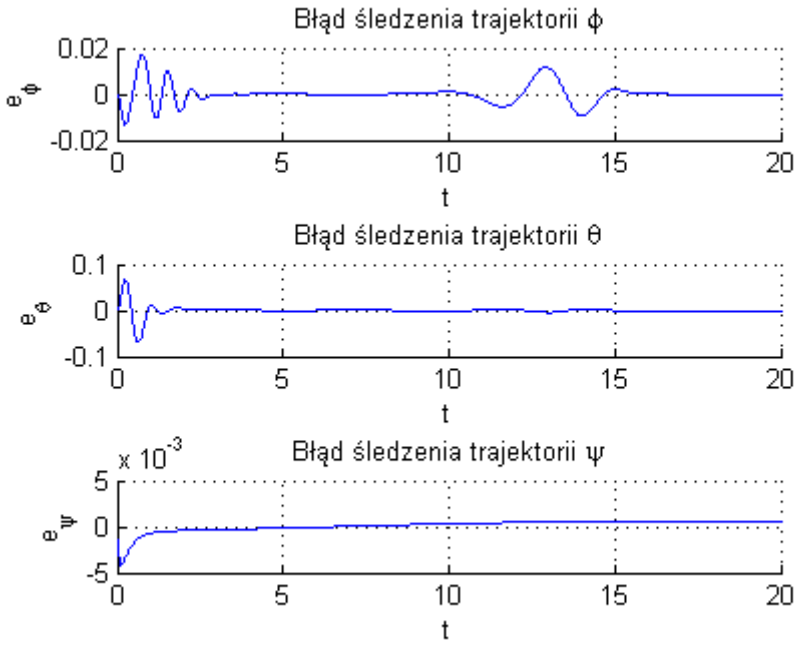
Rysunek 7.104. Błędy współrzędnych położenia - algorytm PID, śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu całkowania wstecznego



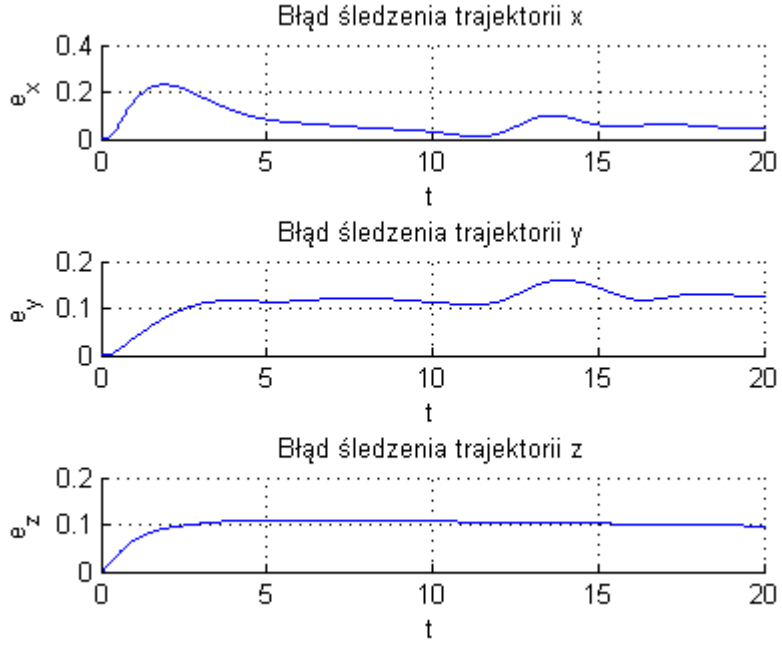
Rysunek 7.105. Błędy współrzędnych orientacji - algorytm PID, śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu całkowania wstecznego



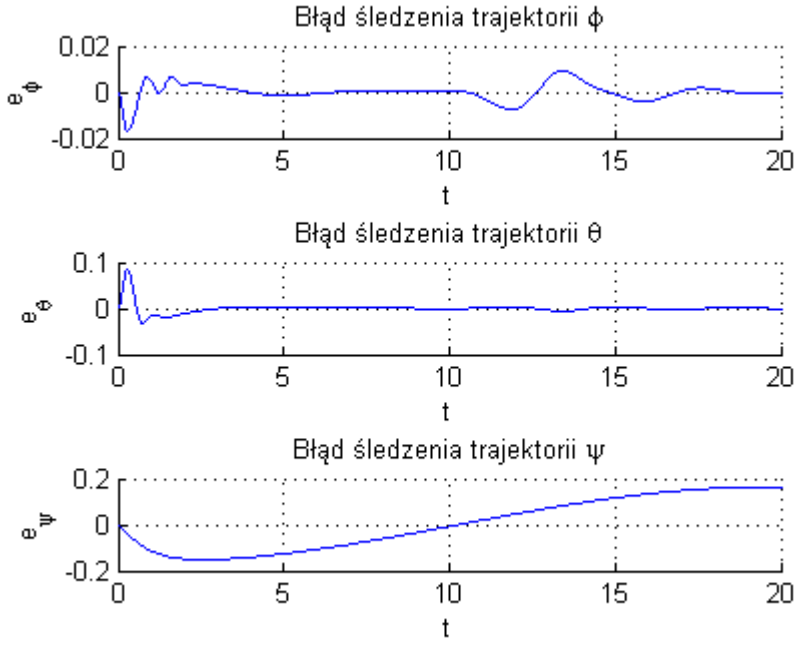
Rysunek 7.106. Błędy współrzędnych położenia - algorytm całkowania wstecznego, śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu całkowania wstecznego



Rysunek 7.107. Błędy współrzędnych orientacji - algorytm całkowania wstecznego, śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu całkowania wstecznego



Rysunek 7.108. Błędy współrzędnych położenia - algorytm H_∞, śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu całkowania wstecznego



Rysunek 7.109. Błędy współrzędnych orientacji - algorytm H_∞, śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu całkowania wstecznego

Tabela 7.22. Poszczególne wskaźniki jakości dla algorytmu PID - śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu całkowania wstecznego

	e_{MIN}	e_{MAX}	e_{MEAN}	ISE	IAE	ITAE
x	-0.129212	0.119084	0.0436722	5.62549	87.388	780.835
y	-0.0511982	0.0496459	0.02367	1.56247	47.3636	435.311
z	-0.00502402	0.0235786	0.00599127	0.0929181	11.9885	105.956
ϕ	-0.0175286	0.0139254	0.00525285	0.0942715	10.5109	120.187
θ	-0.0608534	0.0809655	0.0116959	0.654065	23.4036	117.25
ψ	-0.0471706	0.0141671	0.0101647	0.33449	20.3396	202.71

Tabela 7.23. Poszczególne wskaźniki jakości dla algorytmu całkowania wstecznego - śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu całkowania wstecznego

	e_{MIN}	e_{MAX}	e_{MEAN}	ISE	IAE	ITAE
x	-0.00730702	0.0601638	0.0108093	0.517482	21.6294	158.256
y	0	0.029701	0.0129831	0.377739	25.9792	263.071
z	0	0.0063876	0.000371538	0.00166183	0.743448	4.35201
ϕ	-0.013204	0.0172239	0.00206239	0.0324445	4.12684	32.8548
θ	-0.0693628	0.0683056	0.0027265	0.203335	5.45572	15.7508
ψ	-0.00417421	0.000583438	0.000458379	0.000862456	0.917215	8.78462

Tabela 7.24. Poszczególne wskaźniki jakości dla algorytmu H_∞ - śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu całkowania wstecznego

	e_{MIN}	e_{MAX}	e_{MEAN}	ISE	IAE	ITAE
x	0	0.231285	0.0778177	18.1775	155.713	1221.43
y	-3.80275e-023	0.159844	0.114268	28.0032	228.649	2520.75
z	0	0.109803	0.0997281	20.4922	199.556	2060.85
ϕ	-0.0172602	0.0091239	0.00238171	0.0274846	4.76579	42.6091
θ	-0.0319772	0.0838135	0.00418146	0.231293	8.36711	35.848
ψ	-0.151506	0.164195	0.100845	25.1762	201.791	2117.35

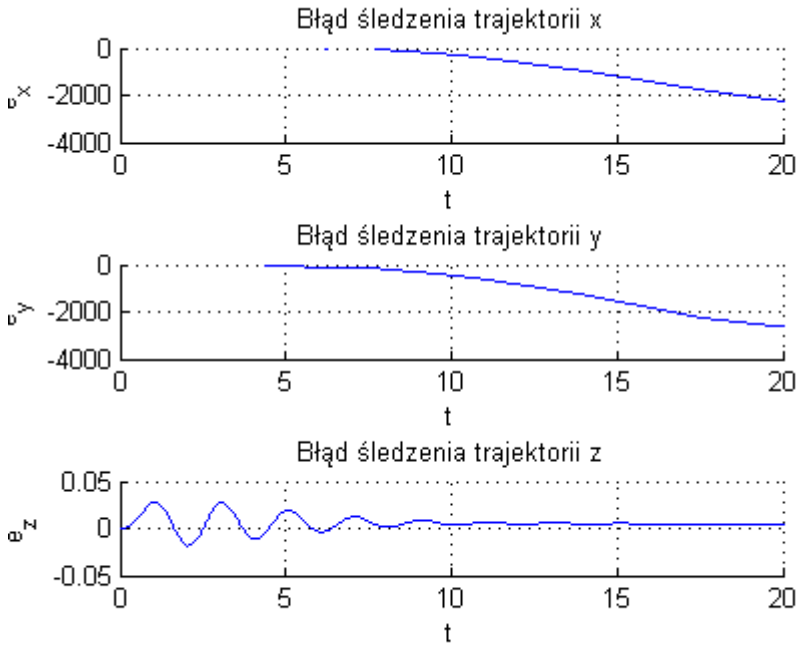
Śledzenie trajektorii zmiennej w czasie

Trajektoria zadana została uzyskana z dynamiki modelu, po zastosowaniu algorytmu PID , dla trajektorii danej układem równań:

$$\begin{cases} x_d = 0 \\ y_d = 0 \\ z_d = 0.05t + 0.2\sin(\frac{t}{20}) \\ \phi_d = 1 - e^{-t} \\ \theta_d = -0.4\sin(\frac{t}{6}) \\ \psi_d = -0.8\sin(\frac{t}{6}) \end{cases} \tag{7.25}$$

Czas sterowania ustalono na 20s, a krok czasowy na 0,01s. Poszczególne wskaźniki jakości dla algorytmu sterowania PID zebrano w tabeli 7.25, dla całkowania wstecznego w 7.26, a dla H_∞ w 7.27. Błędy poszczególnych zmiennych wewnętrznych podczas sterowania dla algorytmu PID przedstawione są na wykresach 7.110 i 7.111, dla algorytmu całkowania wstecznego na 7.112 i 7.113, a dla H_∞ na 7.114 i 7.115.

Maksymalne absolutne wartości wejść dla algorytmu PID:

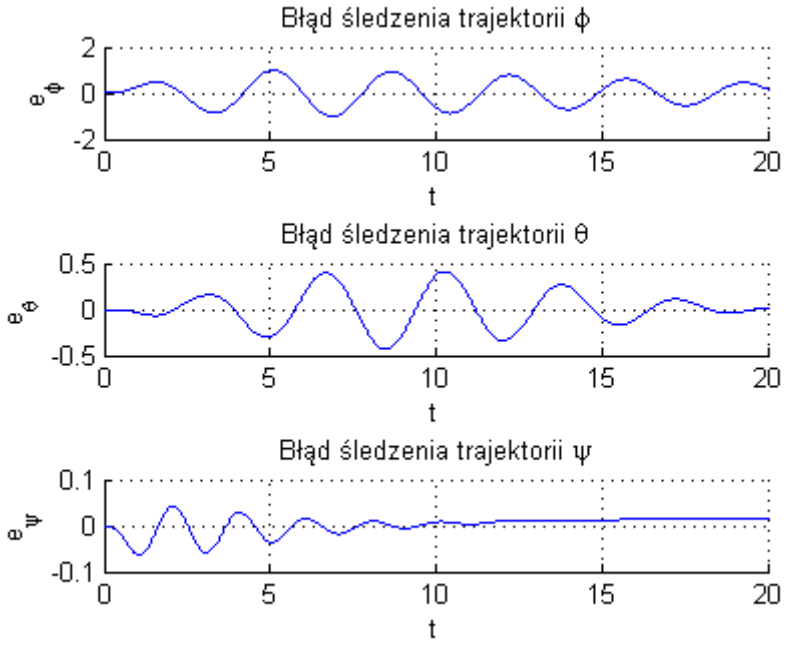


Rysunek 7.110. Błędy współrzędnych położenia - algorytm PID, śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu PID

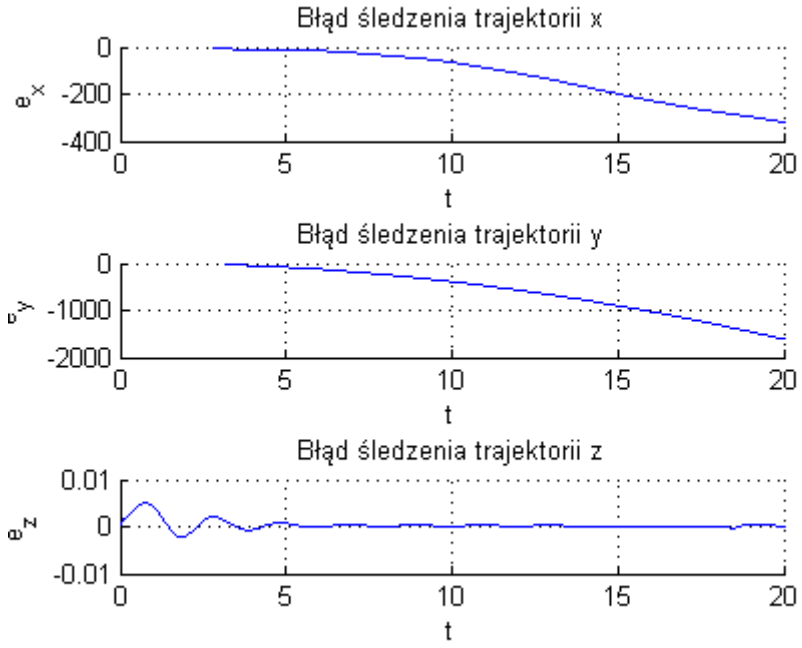
— $max(U_1) = 1,072e + 004$
— $max(U_2) = 7,225e - 002$
— $max(U_3) = 2,948e - 002$
— $max(U_4) = 7,263e - 003$

Maksymalne absolutne wartości wejść dla algorytmu całkowania wstecznego:
— $max(U_1) = 10,28$
— $max(U_2) = 4,653e - 001$
— $max(U_3) = 42,33$
— $max(U_4) = 3,669e - 003$

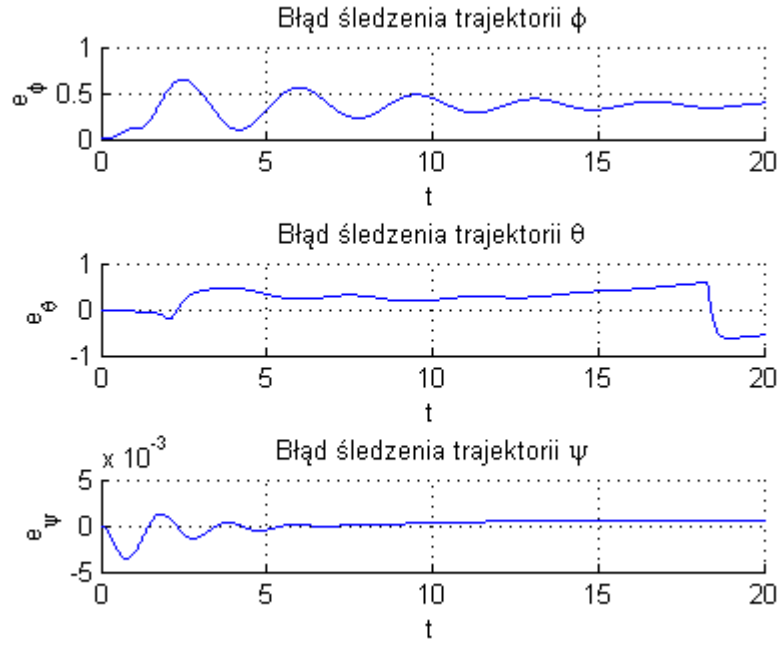
Maksymalne absolutne wartości wejść dla algorytmu H_∞:



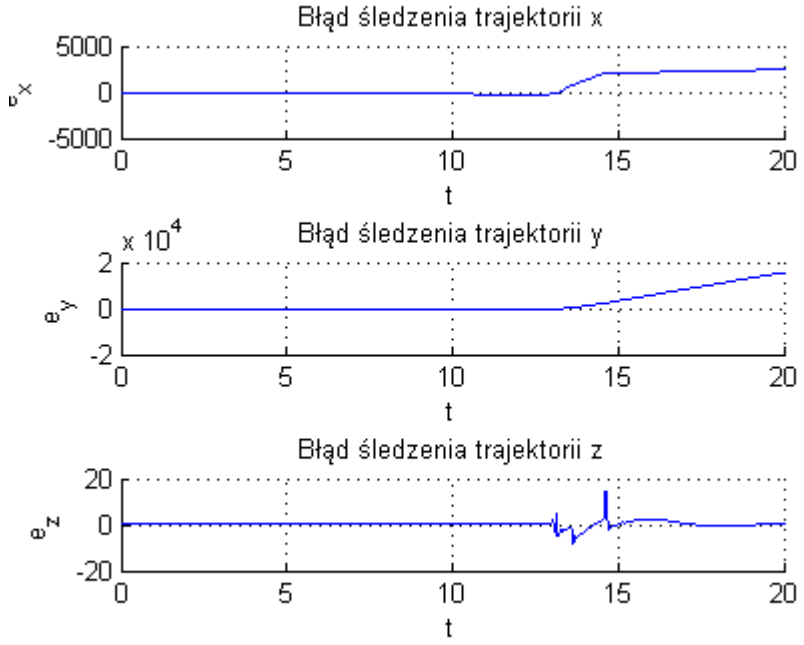
Rysunek 7.111. Błędy współrzędnych orientacji - algorytm PID, śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu PID



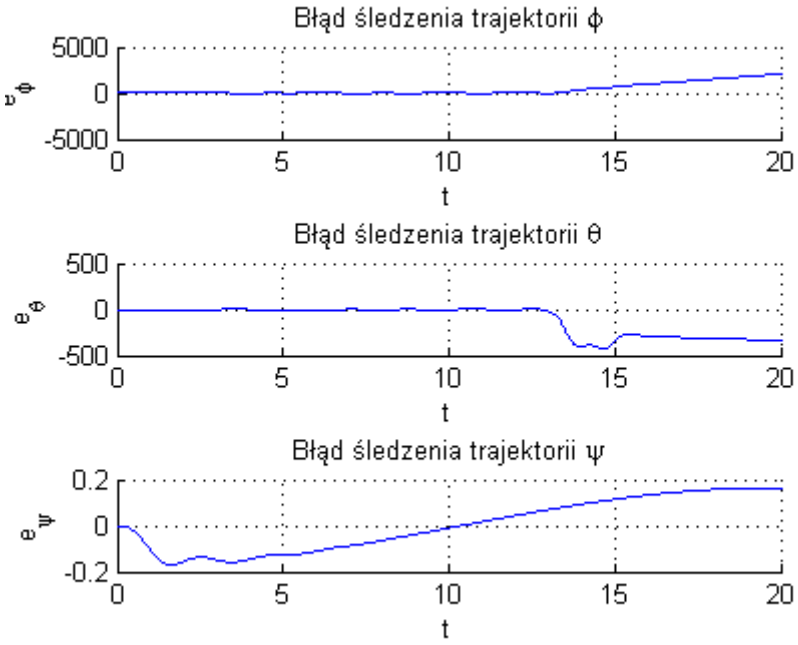
Rysunek 7.112. Błędy współrzędnych położenia - algorytm całkowania wstecznego, śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu PID



Rysunek 7.113. Błędy współrzędnych orientacji - algorytm całkowania wstecznego, śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu PID



Rysunek 7.114. Błędy współrzędnych położenia - algorytm H_∞, śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu PID



Rysunek 7.115. Błędy współrzędnych orientacji - algorytm H_∞, śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu PID

— $max(U_1) = 9,204e + 004$
— $max(U_2) = 67,51$
— $max(U_3) = 61,3$
— $max(U_4) = 1,961e - 003$

Tabela 7.25. Poszczególne wskaźniki jakości dla algorytmu PID - śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu PID

	e_{MIN}	e_{MAX}	e_{MEAN}	ISE	IAE	ITAE
x	-2249.27	0	628.829	1.83428e+009	1.25829e+006	2.03061e+007
y	-2635.57	0	828.443	2.91717e+009	1.65772e+006	2.61424e+007
z	-0.0182355	0.028147	0.00746839	0.179557	14.9443	116.017
ϕ	-1.00495	0.998117	0.469193	591.248	938.855	9075.56
θ	-0.429503	0.412009	0.152132	76.864	304.417	2897.48
ψ	-0.0623421	0.041949	0.0152647	0.79888	30.5446	245.564

Śledzenie trajektorii zmiennej w czasie

Trajektoria zadana została uzyskana z dynamiki modelu, po zastosowaniu algorytmu H_∞, dla trajektorii danej układem równań:

$$\begin{cases} x_d = 0 \\ y_d = 0 \\ z_d = 0.05t + 0.2\sin(\frac{t}{20}) \\ \phi_d = 1 - e^{-t} \\ \theta_d = -0.4\sin(\frac{t}{6}) \\ \psi_d = -0.8\sin(\frac{t}{6}) \end{cases} \tag{7.26}$$

Tabela 7.26. Poszczególne wskaźniki jakości dla algorytmu całkowania wstecznego - śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu PID

	e_{MIN}	e_{MAX}	e_{MEAN}	ISE	IAE	ITAE
x	-319.834	0	106.39	4.43687e+007	212886	3.27978e+006
y	-1606.99	0	524.729	1.02059e+009	1.04998e+006	1.59117e+007
z	-0.00212821	0.00506974	0.000566675	0.00231327	1.13392	5.47013
ϕ	0	0.647219	0.352004	281.034	704.36	7384
θ	-0.620674	0.588624	0.315842	246.062	632	7546.09
ψ	-0.00360862	0.00126932	0.000544208	0.00126295	1.08896	9.12534

Tabela 7.27. Poszczególne wskaźniki jakości dla algorytmu H_∞ - śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu PID

	e_{MIN}	e_{MAX}	e_{MEAN}	ISE	IAE	ITAE
x	-291.842	2459.08	750.384	2.98276e+009	1.50152e+006	2.46151e+007
y	-337.198	15579.2	2578.6	5.15822e+010	5.15978e+006	9.06141e+007
z	-8.36431	14.3202	0.533241	3205.32	1067.02	14892.3
ϕ	-4.83177	2077.03	381.591	1.08015e+009	763564	1.34386e+007
θ	-429.053	1.32029	108.514	7.08424e+007	217136	3.60431e+006
ψ	-0.168711	0.164735	0.10036	25.4259	200.82	2113.14

Czas sterowania ustalono na 20s, a krok czasowy na 0,01s. Poszczególne wskaźniki jakości dla algorytmu sterowania PID zebrano w tabeli 7.25, dla całkowania wstecznego w 7.26, a dla H_∞ w 7.27. Błędy poszczególnych zmiennych wewnętrznych podczas sterowania dla algorytmu PID przedstawione są na wykresach 7.110 i 7.111, dla algorytmu całkowania wstecznego na 7.112 i 7.113, a dla H_∞ na 7.114 i 7.115.

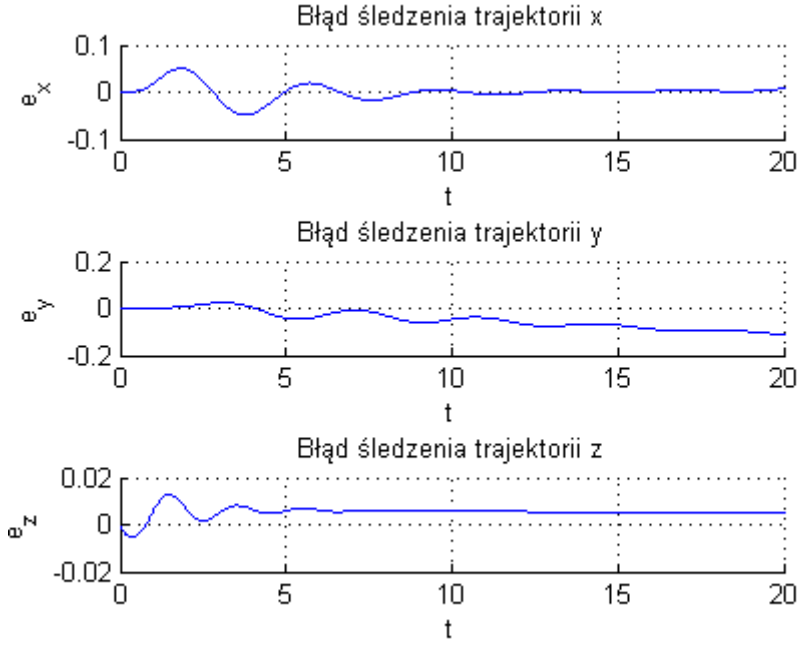
Maksymalne absolutne wartości wejść dla algorytmu PID:
- $max(U_1) = 6,799$
- $max(U_2) = 2,638e-004$
- $max(U_3) = 9,607e-004$
- $max(U_4) = 2,019e-003$

Maksymalne absolutne wartości wejść dla algorytmu całkowania wstecznego:
- $max(U_1) = 6,763$
- $max(U_2) = 1,167e-002$
- $max(U_3) = 1,387e-002$
- $max(U_4) = 7,019e-004$

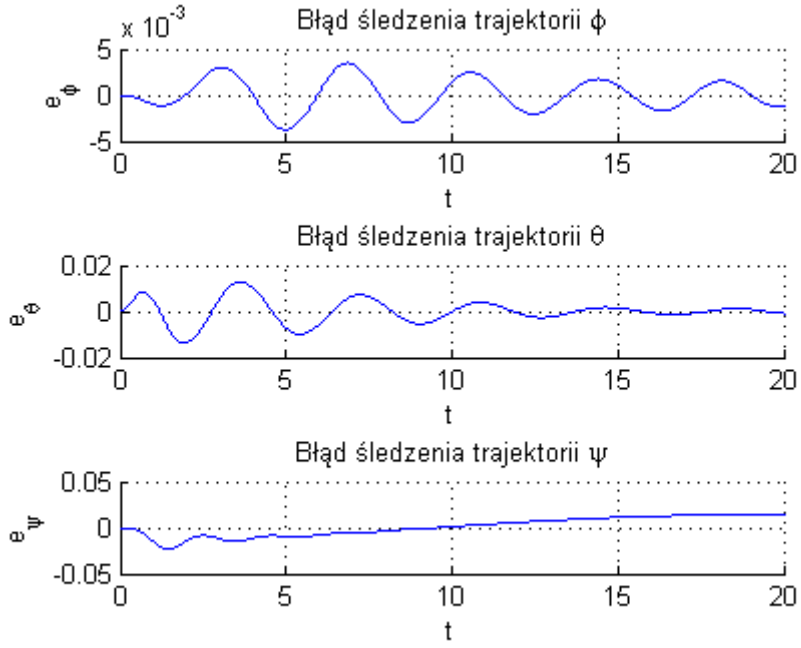
Maksymalne absolutne wartości wejść dla algorytmu H_∞:
- $max(U_1) = 8,133$
- $max(U_2) = 1,256e-004$
- $max(U_3) = 1,242e-004$
- $max(U_4) = 8,178e-004$

7.5.3. Wnioski

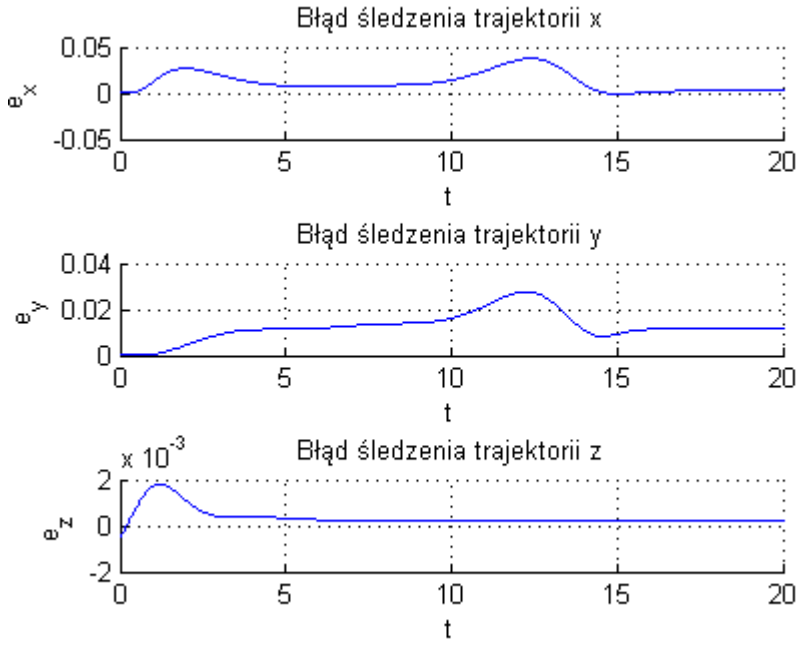
W trakcie badań stwierdzono, że najbardziej czasochłonnym obliczeniowo algorytmem jest H_∞, co może stanowić problem w jego implementacji na rzeczywistym obiekcie. Algorytm ten, jeżeli byłby zastosowany na mikrokontrolerze posiadającym niską moc obliczeniową, mógłby doprowadzić do uszkodzenia, albo wręcz zniszczenia obiektu sterowania, właśnie ze względu na zbyt długi czas obliczeń. Zanim sterownik wyznaczyłby odpowiednie sterowanie, które pozwoliłoby quadrotorowi utrzymać się na trajektorii zadanej, obiekt sterowania oddaliłby się od niej i mógłby zderzSyć sie z przeszkodą terenową, albo wręcz spaść na podłoże. Dużo szybsze są



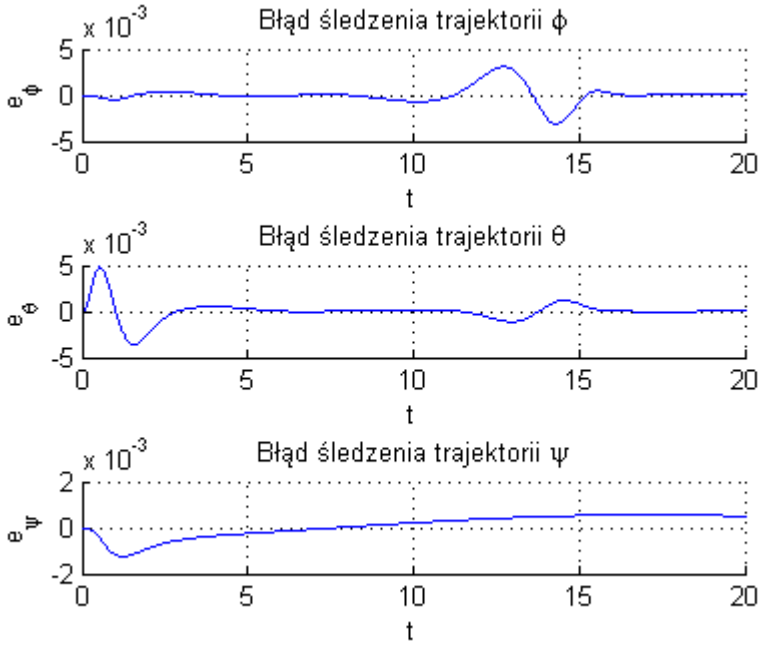
Rysunek 7.116. Błędy współrzędnych położenia - algorytm PID, śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu H_∞



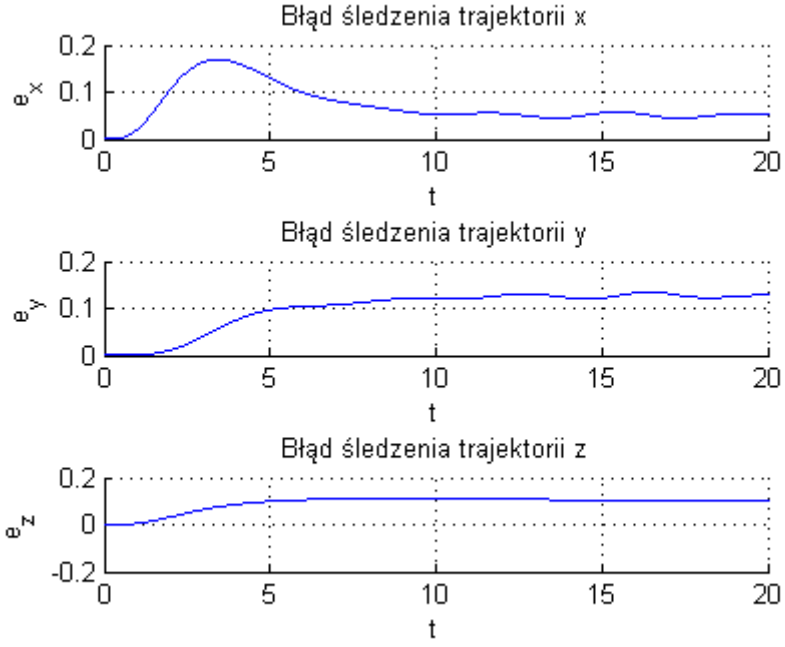
Rysunek 7.117. Błędy współrzędnych orientacji - algorytm PID, śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu H_∞



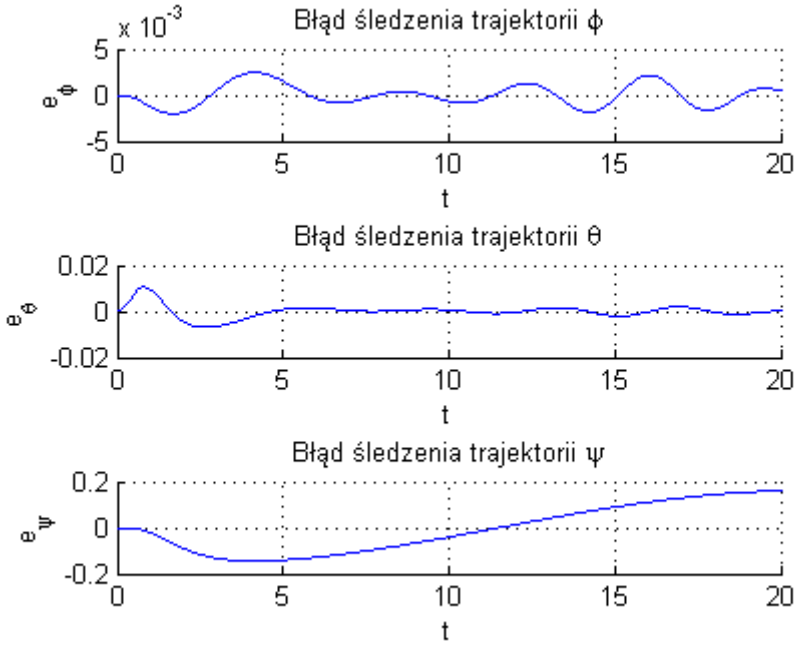
Rysunek 7.118. Błędy współrzędnych położenia - algorytm całkowania wstecznego, śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu H_∞



Rysunek 7.119. Błędy współrzędnych orientacji - algorytm całkowania wstecznego, śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu H_∞



Rysunek 7.120. Błędy współrzędnych położenia - algorytm H_∞, śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu H_∞



Rysunek 7.121. Błędy współrzędnych orientacji - algorytm H_∞, śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu H_∞

Tabela 7.28. Poszczególne wskaźniki jakości dla algorytmu PID - śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu H_∞

	e_{MIN}	e_{MAX}	e_{MEAN}	ISE	IAE	ITAE
x	-0.0483569	0.0501813	0.0103244	0.57266	20.6591	111.094
y	-0.109878	0.0241722	0.0499055	7.09447	99.8608	1354.97
z	-0.00538918	0.0128887	0.00564659	0.068769	11.2988	108.854
ϕ	-0.00372742	0.00348691	0.00147058	0.00614098	2.94263	27.0012
θ	-0.0136925	0.0128163	0.00364985	0.0518873	7.30335	44.4799
ψ	-0.0226752	0.0140377	0.00897326	0.211829	17.9555	189.915

Tabela 7.29. Poszczególne wskaźniki jakości dla algorytmu całkowania wstecznego - śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu H_∞

	e_{MIN}	e_{MAX}	e_{MEAN}	ISE	IAE	ITAE
x	-0.000735006	0.0375408	0.0120373	0.503912	24.0866	209.165
y	-8.90074e-020	0.0276108	0.0125277	0.389962	25.068	280.059
z	-0.000498348	0.0018116	0.000356309	0.000508359	0.712974	4.75856
ϕ	-0.00309947	0.00309342	0.000515312	0.00183285	1.03114	12.0828
θ	-0.00361146	0.00486167	0.000552843	0.00226003	1.10624	6.26427
ψ	-0.00124913	0.000560469	0.000413692	0.000481955	0.827797	8.34486

Tabela 7.30. Poszczególne wskaźniki jakości dla algorytmu H_∞ - śledzenie trajektorii uzyskanej z dynamiki modelu, po zastosowaniu algorytmu H_∞

	e_{MIN}	e_{MAX}	e_{MEAN}	ISE	IAE	ITAE
x	0	0.169102	0.071646	13.4641	143.364	1225.92
y	-8.90454e-020	0.134682	0.0995699	23.3189	199.239	2385.39
z	-4.86753e-016	0.109519	0.0911565	18.5408	182.404	2055.35
ϕ	-0.00201076	0.0024575	0.000945595	0.00268068	1.89214	18.3345
θ	-0.00695748	0.0105605	0.00177064	0.0161512	3.54304	21.4398
ψ	-0.144797	0.160738	0.0899669	20.9825	180.024	1927.82

algorytmy sterowania PID i całkowania wstecznego (rzeczywisty czas trwania symulacji, a więc także wyliczania sterowań był - w przeciwieństwie do H_∞ - znacznie krótszy od czasu lotu quadrotora). Analizując maksymalne wartości zadawanego sterowania tylko algorytm całkowania wstecznego nie zadawał wartości przekraczających 100 na obiekt (H_∞ i PID zadawały co jakiś czas szpilki dochodzące nawet do 10e+006), przy czym odwzorowanie zadanej trajektorii przy zastosowaniu tego algorytmu dawało najelpsze efekty w większości sytuacji. Kryteria wskazują, że najlepszym algorytmem sterowania jest algorytm całkowania wstecznego, co może się zmienić jeżeli udałoby się do pozostałych algorytmów dobrać lepsze nastawy.

Bibliografia

[1] Aström, Hägglund. PID controllers: theory, design and tunings, 2nd edition. 1995.

[2] K. J. Aström, R. M. Murray. Feedback systems: An introduction for scientists and engineers. Raport instytutowy, Department of Automatic Control Lund Institute of Technology, Control and Dynamical Systems California Institute of Technology, 2007.

[3] C. Balas. Autonomous quadrotor robot controller. Master's thesis, Cranfield University, 2007.

[4] S. Bouabdallah. Design and control of an indoor micro quadrotor. *In Proc. of Int. Conf. on Robotics and Automation*, 2004.

[5] S. Bouabdallah. *Design and control of quadrotors with application to autonomous flying.* Praca doktorska, STI, Lausanne, 2007.

[6] S. Bouabdallah, R. Siegwart. Backstepping and sliding-mode techniques apllied to an indoor micro quadrotor. *IEEE International Conference on Robotics and Automation*, Barcelona, Hiszpania, 2005.

[7] A. R. S. Bramwell, G. Done, D. Balmford. Bramwell's helicopter dynamics. *In Proc. of Int. Conf. on Robotics and Automation*, 2001.

[8] T. Bresciani. Modeling, identification and control of a quadrotor helicopter. Master's Thesis ISRN LUTFD2/TFRT--5823--SE, Department of Automatic Control, Lund University, Wrocław University of Technology, Sweden, 2008.

[9] M. Cheng. Formation and flight control of affordable quad-rotor unmanned air vehicles. *The University of Brithish Columbia*, 2003.

[10] M. Dymczyk. Autonomous quadrotor robot controller, 10th Students' Science Conference. 2012.

[11] A. Y. Elruby, M. M. El-khatib, N. H. El-Amary, A. I. Hashad. Dynamic modeling and control of quadrotor vehicle. *Fifteenth International Conference on Applied Mechanics and Mechanical Engineering, AMME-15*, 2012.

[12] R. Goela, S. M. Shahb, N. K. Guptac, N. Ananthkrishnanc. Modeling, simulation and flight testing of an autonomous quadrotor. *ICEAE - International Conference on Environmental and Agriculture Engineering*, 2009.

[13] G. M. Hoffmann, H. Huang, S. L. Wasl, E. C. J. Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. *In Proc. of the AIAA Guidance, Navigation, and Control Conference*, 2007.

[14] A. Kacimi, A. Mokhtari, B. Kouadri. Sliding mode control based on adaptive backstepping approach for a quadrotor unmanned aerial vehicle. *Przegląd elektrotechniczny*, 2012.

[15] T. W. B. Kibble, F. H. Berkshire. *Classical Mechanics; 5th ed.* Imperial Coll., London, 2004.

[16] P. V. Kokotovic. The joy of feedback: nonlinear and adaptive. *Control System Magazine*, 12:7–17, 1992.

[17] M. Krsti, I. Kanellakopoulos, P. V. Kokotovic. Nonlinear and adaptive control design. 1995.

[18] T. Madini, A. Benallegue. Backstepping control for a quadrotor helicopter. *IEEE International Conference on Intelligente Robots and Systems*, Pekin, Chiny, 2006.

[19] T. Madini, A. Benallegue. Control of a quadrotor mini-helicopter via full state backstepping technique. *45th IEEE Conference on Decision & Control*, San Diego, USA, 2006.

[20] J. Malewicz. Środowisko do badania prostych podzespołów robotycznych. *Praca magisterska, PWr*, 2008.

[21] A. Mazur. *Sterowanie oparte na modelu*. Politechnika Wrocławska, Wrocław, Polska, 2009.

[22] I. Palunko, R. Fierro, P. Cruz. Trajectory generation for swing-free maneuvers of a quadrotor with suspended payload: A dynamic programming approach. *ICRA*, strony 2691–2697, 2012.

[23] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, Cambridge, 1992.

[24] G. V. Raffo, M. G. Orteg, F. R. Rubio. Backstepping/h_∞ control for path tracking of a quadrotor unmanned aerial vehicle. *American Control Conference*, Washington, USA, 2008.

[25] S. Skogestad, I. Postlethwaite. Multivariable feedback control. analysis and design. *John Wiley and Sons*, 2001.

[26] K. Tchoń, A. Mazur, I. Dulęba, R. Hossa, R. Muszyński. *Manipulatory i roboty mobilne: Modelowanie, planowanie ruchu, sterowanie*. Akademicka Oficyna Wydawnicza PLJ, Warszawa, 2000.