POLITECHNIKA ŚLĄSKA

WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI

Modele i Systemy Sterowania w Robotyce

Raport końcowy

AiR S2-I/Rob

Sekcja nr 1:

Piotr Gruchalski

Paweł Kaźmieruk

Tomasz Kuś

Maksymilian Skibiński

Paweł Szczech

Mateusz Szczepanik

29 sierpnia 2021 r.

Spis treści

Wstęp 3

1 Zadanie proste i odwrotne kinematyki (27.05.2021) 4

2 Planowanie trajektorii (31.05.2021) 9

3 Jakobian (07.06.2021) 12

4 Jakobian – Simulink (10.06.2021) 20

5 Dynamika (14.06.2021, 17.06.2021) 23

6 Roboty mobilne (21.06.2021, 24.06.2021) 32

Wstęp

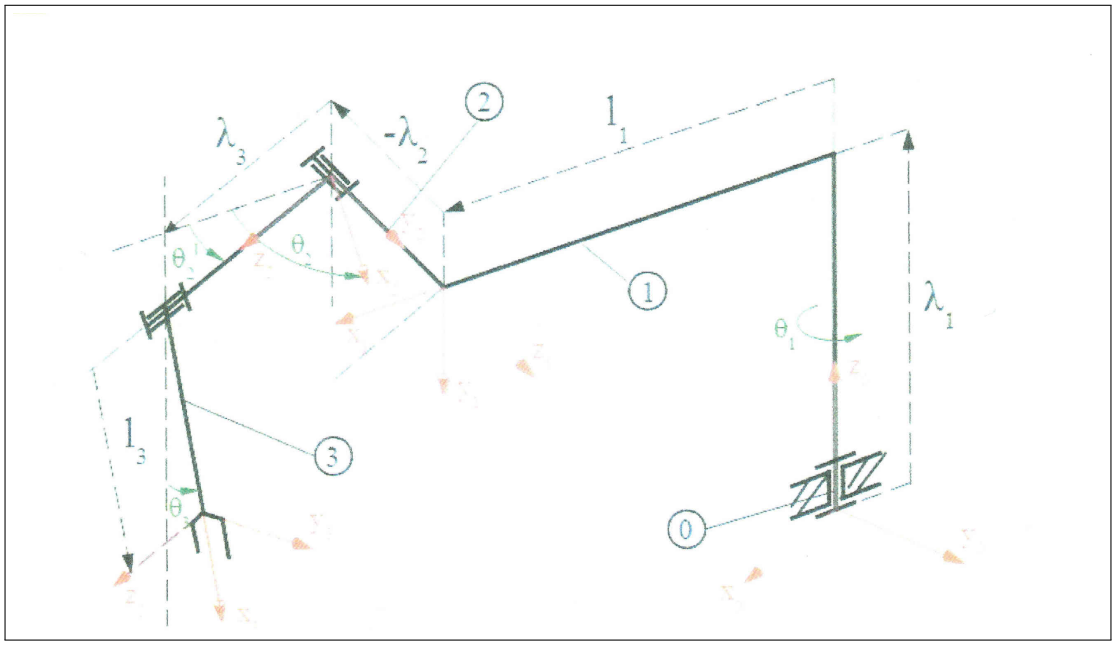
W ramach laboratorium przerobiono materiał zawarty w książce Petera Corke'a „*Robotics, Vision and Control*". Potrzebne informacje zawarte były w czterech rozdziałach:

- rozdział 4: Roboty mobilne
- rozdział 7: Kinematyka manipulatora
- rozdział 8: Prędkość manipulatora
- rozdział 9: Dynamika i kontrola

Środowiskiem pracy w trakcie wirtualnych zajęć był MATLAB, a głównym narzędziem wykorzystywanym do wykonywania ćwiczeń był Robotics Toolbox, również autorstwa Corke'a.

1 Zadanie proste i odwrotne kinematyki (27.05.2021)

W ramach tych ćwiczeń laboratoryjnych należało zapoznać się z treścią rozdziałów 7.1 i 7.2 książki i wykorzystać omawiane skrypty dla przydzielonych sekcjom manipulatorów. Schemat kinematyczny manipulatora przydzielony sekcji nr 1 jest przedstawiony na rysunku 1.



Rysunek 1: Schemat kinematyczny manipulatora

Struktura przedstawionego manipulatora to OOO[1] – składa się z 3 par kinematycznych obrotowych. Korzystając z zapisu Denavita-Hartenberga można zadeklarować manipulator w MATLABie.

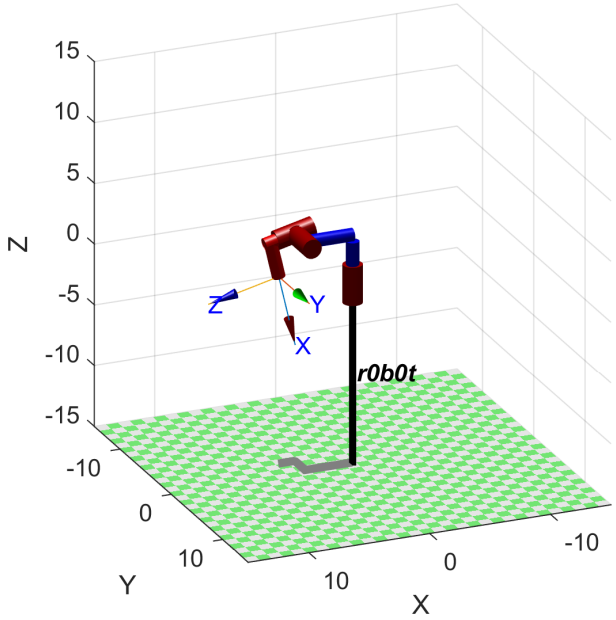
```
% Stałe parametry (przykładowe wartości).
lambda_1 = 4;
lambda_2 = 2;
lambda_3 = 2;

l_1 = 4;
l_3 = 3;

% Deklaracja członów.
L1 = Revolute('d', lambda_1, 'a', l_1, 'alpha', -pi/2);
L2 = Revolute('d', -lambda_2, 'a', 0, 'alpha', pi/2);
L3 = Revolute('d', lambda_3, 'a', l_3, 'alpha', 0);

% Deklaracja manipulatora.
rob = SerialLink([L1 L2 L3], 'name', 'r0b0t');
```

[1] W książce, i innych materiałach w języku angielskim, taki manipulator jest oznaczony jako RRR, gdzie R to skrót od *revolute*

Za pomocą metody `plot`[2] można przedstawić manipulator dla wybranych współrzędnych naturalnych członów (umieszczonych w wektorze $\mathbf{q}$).

```
% Wektor współrzędnych naturalnych.
q = [0, 1.8, 0];

% Manipulator dla tych współrzędnych.
rob.plot(q);
view(157, 22);
```

Ten kod tworzy rysunek 2.



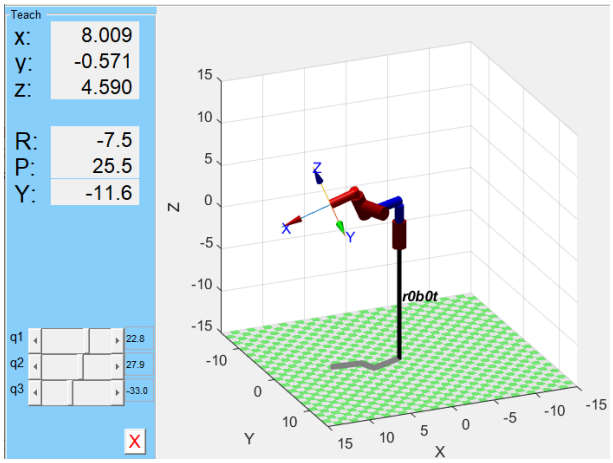
Rysunek 2: Użycie funkcji `plot`

Dla użytego wektora $\mathbf{q}$ oraz funkcji `view` (która zmienia położenie obserwatora względem robota) obraz manipulatora jest zbliżony do obrazu, jaki widać na schemacie kinematycznym (rys. 1).

Bardziej zaawansowanym sposobem na obserwowanie, jak zmiana współrzędnych naturalnych manipulatora wpłynie na położenie i orientację elementu wykonawczego, jest używanie metody `teach` (rys. 3). Tym razem nie podaje się konkretnego wektora $\mathbf{q}$, tylko korzysta z suwaków do określenia dokładnych wartości składowych tego wektora. Ponadto, w górnej części menu jest zapisane położenie i orientacja Roll-Pitch-Yaw układu skojarzonego z elementem wykonawczym.

Zadanie proste kinematyki jest problemem geometrycznym polegającym na obliczeniu położenia i orientacji manipulatora na podstawie współrzędnych naturalnych członów. Opis manipulatora przechodzi zatem z przestrzeni wewnętrznej (wektor $\mathbf{q}$) do przestrzeni zewnętrznej. Dotychczas wspomniane metody `plot` oraz `teach` rozwiązywały omawiany problem, ale

[2]Mowa o metodzie `plot` obiektu klasy `SerialLink`, a nie o funkcji `plot` występującej w podstawowym MATLABie



Rysunek 3: Użycie metody `teach`

zadanie może być rozwiązane również z użyciem metody `fkine` – tym razem na podstawie wektora **q** stworzona zostanie macierz **T** opisującą układ współrzędnych związany z elementem wykonawczym. Przykładowo:

```
>> rob.fkine([0, 0, 0])


ans =
         1         0         0         7
         0         1         0        -2
         0         0         1         6
         0         0         0         1
>> rob.fkine([0, pi/2, 0])


ans =
         0         0         1         6
         0         1         0        -2
        -1         0         0         1
         0         0         0         1
```

Co więcej, metoda `fkine` może zwrócić nie tylko macierz **T**, ale także macierze pośrednie **A**, które opisują relację pomiędzy poszczególnymi układami współrzędnych związanymi z członami manipulatora.

```
1 [T, ALL] = rob.fkine([pi/2, pi/3, 0])
```

W omawianym przypadku, wyjściowa macierz `ALL` zawiera 3 macierze pośrednie **A**.

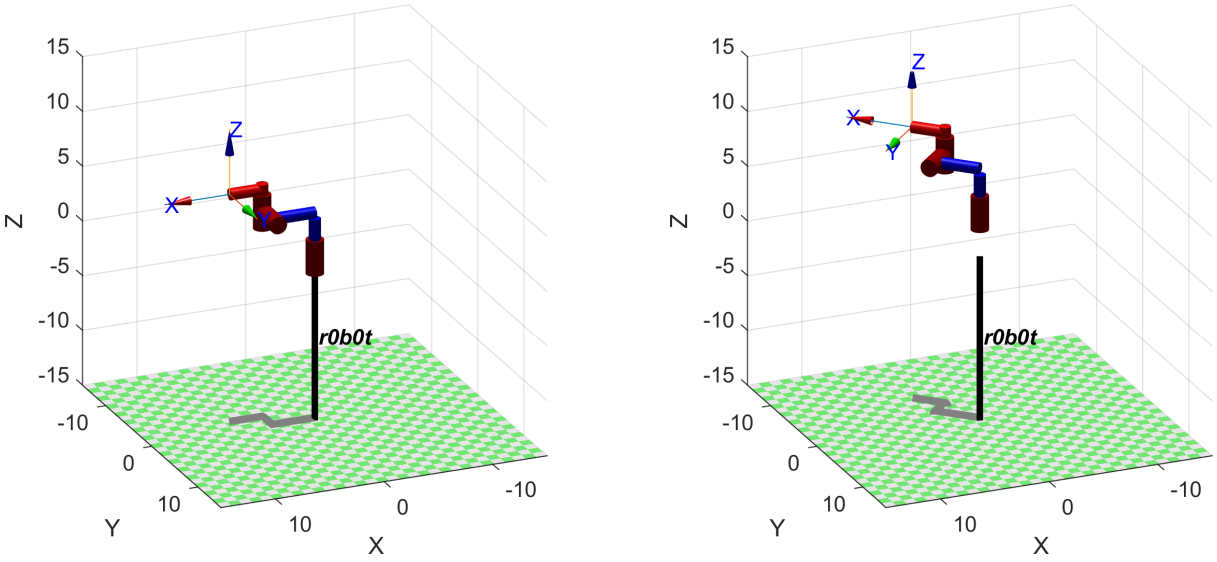
Korzystając z własności `base` oraz `tool`, można zdefiniować macierze przekształceń wpływające na położenie i orientację układów:

– związanego z podstawą robota,

– związanego z elementem wykonawczym.

W kodzie poniżej zadeklarowano przesunięcie o 4 wzdłuż osi z oraz obrót o $\frac{\pi}{4}$ wokół osi z. Rezultat widoczny jest na rysunku 4.

```
1 rob.base = SE3(0, 0, 4) * SE3.Rz(-pi/4);
```



(a) bez zmiany

(b) użycie wspomnianego kodu

Rysunek 4: Wpływ zmiany własności `base`

Podobny efekt można uzyskać zmieniając parametry Denavita-Hartenberga.

Zadanie odwrotne kinematyki

W zadaniu odwrotnym kinematyki *wejściem* jest pożądane położenie i orientacje elementu wykonawczego, a *wyjściem* współrzędne naturalne członów manipulatora. Ten problem jest znacznie trudniejszy: może istnieć więcej niż jedno rozwiązanie wartości zadanych, ale może także nie być żadnego rozwiązania – o tym decyduje przestrzeń robocza manipulatora.

Rozwiązań tego problemu można szukać na dwa sposoby:

- analitycznie – korzystając ze zmiennych symbolicznych wyprowadza się równania kinematyki prostej, a następnie przekształca je tak, by otrzymać wzory jawne wyrażające zadaną składową położenia/orientacji w funkcji współrzędnych naturalnych,
- numerycznie – bazując na iteracyjnym rozwiązywaniu zadania prostego kinematyki, aż błąd pomiędzy zadanym położeniem a otrzymanym będzie jak najmniejszy – jest to pewnego rodzaju problem optymalizacji.

Metody analityczne są szybsze, ale wraz ze wzrostem liczby par kinematycznych (i tym samym zmiennych w równaniach) równania znacznie się komplikują i otrzymanie rozwiązania w postaci jawnej może być niemożliwe. Używając funkcji `ikine6s` można otrzymać rozwiązania analityczne dla robotów o 6 stopniach swobody z kiścią sferyczną.

Metody numeryczne działają wolniej, ale są znacznie bardziej uniwersalne. Można z nich korzystać poprzez funkcję `ikine`.

Badany manipulator ma 3 stopnie swobody, przez co jego przestrzeń robocza jest ograniczona – nie będzie można osiągnąć każdej pozycji efektora. Zatem przy używaniu funkcji `ikine` należy zdefiniować *maskę*, poprzez którą deklaruje się, które składowe położenia/orientacji mają zostać zignorowane, a które nie – w omawianym przypadku co najmniej 3 składowe muszą zostać zignorowane. Przykładowe użycie:

```
% Pożadane położenie i orientacja el. wykonawczego.
T = SE3(7, -2, 6.2) * SE3.rpy(pi/4, 0, pi/6);

% Rozwiązanie zadania odwrotnego kinematyki, dla użytej maski.
q_ikine = rob.ikine(T, 'mask', [1, 1, 1, 0, 0, 0]);

% Rozwiązanie zadania prostego dla otrzymanych współrzędnych naturalnych.
T_ikine = rob.fkine(q_ikine);
T_ikine.print('xyz')
```

W efekcie otrzymano:

```
t = (7, -2, 6.2), RPY/xyz = (-7.03, -3.99, 0.374) deg
```

Powyższy kod zaczyna się od deklaracji macierzy $\mathbf{T}$ określającej pożądane położenie i orientację efektora. Następnie rozwiązywane jest zad. odwr. kin. dla tej macierzy, przy czym maska wskazuje, że orientacja ma zostać zignorowana. By sprawdzić otrzymane rozwiązanie program oblicza macierz $\mathbf{T}$ dla uzyskanych współrzędnych naturalnych. Jak widać, położenie pokrywa się z zadanym, a orientacje się różnią, na co należało przystać.

Tego kodu można także używać dla innych macierzy $\mathbf{T}$ oraz innych masek, należy jednak pamiętać że:

- nie znajdzie on rozwiązania dla każdej macierzy $\mathbf{T}$, gdyż manipulator jest ograniczony poprzez swoją strukturę,
- można zmieniać wektor wartości binarnych określający maskę, ale mogą tam się znaleźć co najwyżej 3 jedynki. Przy mniejszej liczbie jedynek niż 3 rozwiązanie może bardziej odbiegać od wartości zadanych.

2 Planowanie trajektorii (31.05.2021)

Jednym z najczęstszych zadań stawianych manipulatorom jest płynny ruch pomiędzy dwoma zadanymi punktami. To zadanie można rozwiązać na dwa sposoby:

– ruch prostoliniowy w przestrzeni konfiguracyjnej,

– ruch prostoliniowy w przestrzeni kartezjańskiej.

Pierwszy sposób polega na użyciu funkcji `jtraj` – funkcja ta dla dwóch zadanych wektorów współrzędnych naturalnych i wektora czasu zwróci wektory współrzędnych naturalnych tworzących trajektorię pomiędzy dwoma zadanymi wartościami granicznymi. Ponadto można także otrzymać odpowiadające danym chwilom czasu prędkości i przyspieszenia członów. Przykładowo, dla pewnych dwóch macierzy $\mathbf{T}_1$, $\mathbf{T}_2$ określających zadane pozycje efektora:

```
% Rozwiązanie zadania odwrotnego kinematyki dla
% pozycji początkowej i końcowej.
q1 = rob.ikine(T1, 'mask', [1, 1, 1, 0, 0, 0]);
q2 = rob.ikine(T2, 'mask', [1, 1, 1, 0, 0, 0]);

% Przykładowy wektor czasu.
t = [0 : 0.05 : 2]';

% Generowanie trajektorii w przestrzeni wewnętrznej.
[q, qd, qdd] = jtraj(q1, q2, t);
```

Teraz, dla otrzymanej *macierzy* `q`, która zawiera współrzędne naturalne dla zadanych chwil czasu, można skorzystać z metody `plot`, by uruchomić animację, która przedstawi ruch manipulatora. Można także rozwiązać zadanie proste kinematyki dla tych współrzędnych, by otrzymać macierze $\mathbf{T}$ określające położenie el. wykonawczego na każdym etapie animacji.

Do wykonywania ruchu prostoliniowego efektora w przestrzeni kartezjańskiej służy z kolei funkcja `ctraj`.

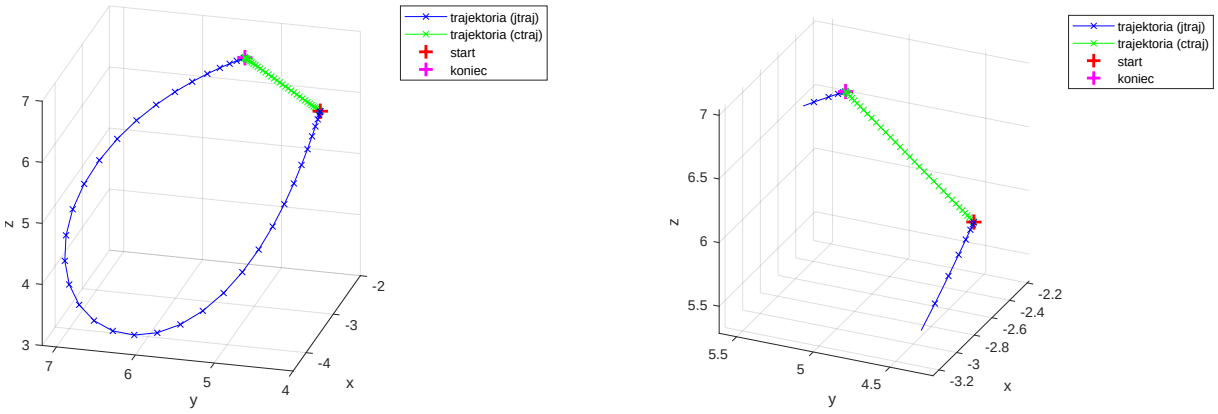
```
Ts = ctraj(T1, T2, length(t));
```

Porównanie otrzymanych trajektorii znajduje się na rysunku 5.

Trajektorie różnią się znacząco, chociaż oczywiście punkty początkowe i końcowe są takie same. Dla generacji trajektorii prostoliniowej w przestrzeni kartezjańskiej, punkty pośrednie tworzą linie prostą, a dla generacji traj. w przestrzeni konfiguracyjnej nie. Porównanie wyznaczonych współrzędnych naturalnych znajduje się na rysunku 6.

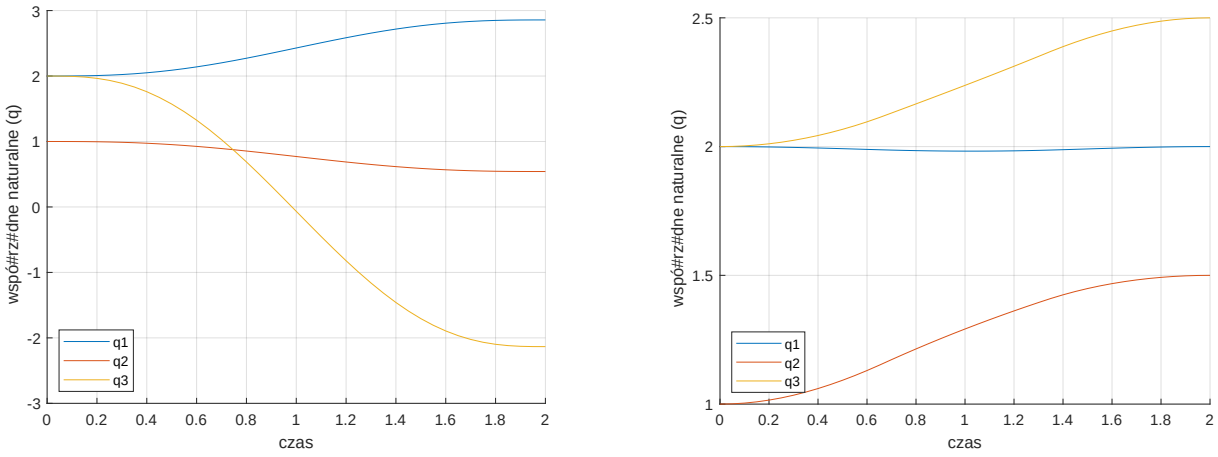
Wykresy pokazują, że nie tylko zmieniły się wyznaczone wartości *pośrednie* $\mathbf{q}$, ale także wartości *końcowe*. Wartości końcowe $\mathbf{q}$ dla obu wariantów skutkują jednak w tym samym zadanym położeniu punktu końcowego, choć w innej orientacji, na co należało przystać.

Można zadać pytanie: co to znaczy, że trajektoria jest liniowa w przestrzeni konfiguracyjnej? Należy sporządzić wykresy analogiczne jak z rysunków 5, ale inaczej opisać osie, tak by przedstawiały wartości współrzędnych naturalnych. Porównanie dla obu trajektorii widać na rysunku 7.



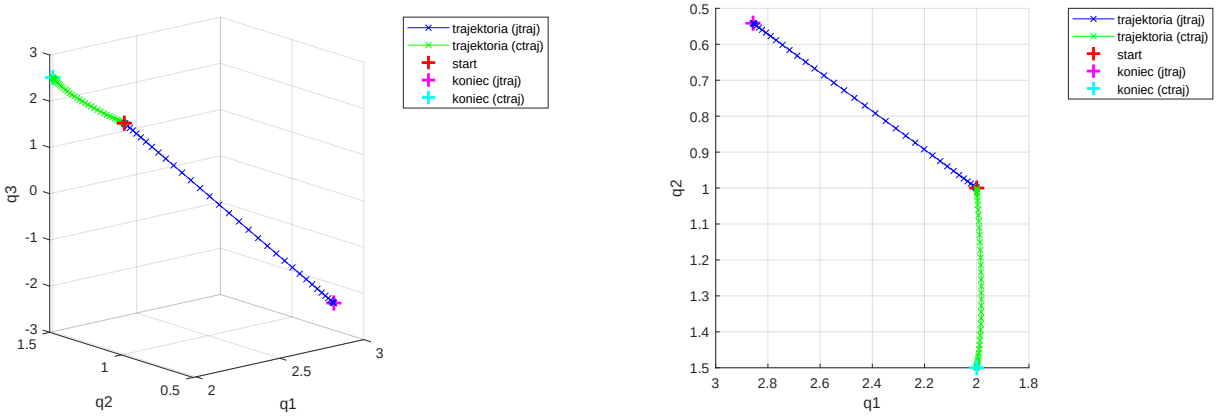
(a) (b) zbliżenie na `ctraj`

Rysunek 5: Porównanie funkcji `jtraj` i `ctraj` – trajektorie



(a) (b) zbliżenie na `ctraj`

Rysunek 6: Porównanie funkcji `jtraj` i `ctraj` – współrzędne naturalne



(a) (b) zbliżenie na `ctraj`

Rysunek 7: Porównanie funkcji `jtraj` i `ctraj` – przestrzeń konfiguracyjna

Tym razem to trajektoria wyznaczona poprzez `jtraj` jest liniowa, a wyznaczona poprzez `ctraj` już nie (choć odbiega od liniowej tylko nieznacznie). Na rysunkach widać także zauważony na wcześniejszych wykresach fakt: wartości końcowe współrzędnych naturalnych różnią się.

Przy wykorzystywaniu omówionych sposobów do generacji trajektorii należy pamiętać, że ruch prostoliniowy nie zawsze będzie możliwy. Nawet jeżeli punkty początkowe i końcowe należą do przestrzeni roboczej manipulatora, punkty pośrednie trajektorii niekoniecznie muszą do niej należeć, szczególnie w omawianym przypadku, gdy do dyspozycji mamy manipulator o jedynie 3 stopniach swobody. Wszystko zależy od kształtu przestrzeni roboczej robota. Ponadto mogą wystąpić sytuacje gdy ruch prostoliniowy jest możliwy w jednej przestrzeni, a w drugiej nie.

3 Jakobian (07.06.2021)

Jakobian jest macierzą, która określa zależność pomiędzy tempem zmian współrzędnych naturalnych manipulatora $\mathbf{q}$ a położeniem i orientacją członu wykonawczego w przestrzeni kartezjańskiej i vice versa.

$$\boldsymbol{\nu} = \mathbf{J}\dot{\mathbf{q}}$$
$$\dot{\mathbf{q}} = \mathbf{J}^{-1}\boldsymbol{\nu}$$

gdzie:

– $\boldsymbol{\nu}$ – oznacza prędkości w przestrzeni kartezjańskiej:

$$\boldsymbol{\nu} = [\nu_x\ \nu_y\ \nu_z\ \omega_x\ \omega_y\ \omega_z]$$

– $\dot{\mathbf{q}}$ – oznacza prędkości w przestrzeni konfiguracyjnej:

$$\dot{\mathbf{q}} = [\dot{q}_1\ \dot{q}_2\ \dots\ \dot{q}_N] \qquad N \text{ – liczba stopni swobody}$$

– $\mathbf{J}$ – jakobian, czyli macierz, która zawiera wszystkie kombinacje pochodnych cząstkowych składowych $\boldsymbol{\nu}$ i $\dot{\mathbf{q}}$.

$$\mathbf{J} = \frac{\partial \boldsymbol{\nu}}{\partial \dot{\mathbf{q}}} = \begin{bmatrix} \frac{\partial \nu_1}{\partial \dot{q}_1} & \dots & \frac{\partial \nu_1}{\partial \dot{q}_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial \nu_6}{\partial \dot{q}_1} & \dots & \frac{\partial \nu_6}{\partial \dot{q}_N} \end{bmatrix}$$

Jest ona macierzą o wymiarach 6 x N. Istotne jest, że wartości jej elementów zależą od współrzędnych naturalnych $\mathbf{q}$, czyli tak naprawdę: $\mathbf{J}(\mathbf{q})$.

Do wyznaczania jakobianów wykorzystuje się funkcje:

– `jacob0` – jakobian wyznaczony względem układu bazowego: ${}^0\mathbf{J}(\mathbf{q})$,

– `jacobe` – jakobian wyznaczony względem układu członu wykonawczego: ${}^E\mathbf{J}(\mathbf{q})$.

Przykładowo:

```
% Przykładowe współrzędne naturalne.
q = [0, 0, 0];

% Wyznaczenie jakobianu.
J = rob.jacob0(q);
```

da w rezultacie:

```
J =

    2.0000    2.0000         0
```

```
    7.0000    0.0000    3.0000
         0   -3.0000         0
         0         0         0
         0    1.0000         0
    1.0000    0.0000    1.0000
```

O czym mówią takie wartości?

Zaczynając od wymiarów – macierz jest prostokątna o wymiarach 6 x 3, co wynika z faktu, że pozycja obiektu w przestrzeni kartezjańskiej ma 6 składowych (3 dla położenia i 3 dla orientacji), a omawiany manipulator ma 3 stopnie swobody. Przy analizie jakobianu, warto zaznaczyć, że rzędom odpowiada przestrzeń kartezjańska, a kolumnom przestrzeń konfiguracyjna.

W pierwszej kolumnie widzimy zera i wartości niezerowe. Zero oznacza, że zmiany wartości *tej konkretnej* współrzędnej naturalnej nie wpływają na zmiany tej konkretnej współrzędnej kartezjańskiej. W przykładowym jakobianie zmiany współrzędnej naturalnej q_1 w ogóle nie wpływają na współrzędną z położenia i na obroty wokół osi x i y elementu wykonawczego.

Niezerowe liczby w komórkach macierzy świadczą o tym że zmiana ν_i wpłynie na tę konkretną współrzędną q_j. Im liczba większa tym silniejszy efekt wywrze zmiana danej współrzędnej.

Jak już zostało wspomniane, ta konkretna macierz $\mathbf{J}$ została wyznaczona dla konkretnych współrzędnych naturalnych $\mathbf{q}$ i ulegnie ona zmianie gdy zmianie ulegną wsp. naturalne. Dla przykładu wprowadzono drobną zmianę: niech q_1 będzie teraz równe $5°$.

```
% Przykładowe współrzędne naturalne -- poprzednie.
q = [0, 0, 0];

% Przykładowe współrzędne naturalne -- zmiana.
q = [5 * pi/180, 0, 0];
```

W rezultacie otrzymano:

```
J =

    1.3823    1.9924   -0.2615
    7.1477    0.1743    2.9886
         0   -3.0000         0
         0   -0.0872         0
         0    0.9962         0
    1.0000    0.0000    1.0000
```

Niewielki ruch spowodował, że pewne elementy macierzy $\mathbf{J}$ się zmieniły, a inne nie. Przydatne w takiej analizie jest posługiwanie się metodą `teach`.

Jakobian można także wyznaczyć względem układu związanego z elementem wykonawczym przy pomocy metody `jacobe`. Macierze wyznaczone przez `jacob0` i `jacobe` mogą się pokrywać dla pewnych wektorów $\mathbf{q}$. Ponadto zmiany konkretnej składowej wektora $\mathbf{q}$ mogą mieć wpływ na zmiany jednej z macierzy, a na drugą nie.

Jakobiany wyznaczane przez wymienione funkcje domyślnie wyrażają wpływ na orientację efektora poprzez prędkości kątowe: $\omega_x, \omega_y, \omega_z$. Korzystając z dodatkowych opcjonalnych argumentów funkcji, można sprecyzować, by stosowany był zapis względem kątów Roll-Pitch-Yaw lub kątów Eulera.

Zgodnie z równaniem przedstawionym na początku rozdziału, jakobiany można wykorzystać do wyznaczenia odpowiednich prędkości $\dot{\mathbf{q}}$ lub $\boldsymbol{\nu}$. Należy jednak zdawać sobie sprawę, że omawiany manipulator ma tylko 3 stopnie swobody.

Przykład:

```
% Współrzędne naturalne w danej pozie.
qn = [1.5, 1, 0.5];

% Jakobian dla współrzędnych qn.
J = rob.jacob0(qn);

% Wektor pożądanych prędkości.
vel = [0, 0.25, 0, 0, 0, 0];

% Wyznaczenie odpowiednich prędkości q.
qd = pinv(J) * vel';

% Rezultat.
xd = J * qd; xd = xd';

vel
xd
```

Manipulator znajduje się w pewnej pozie opisanej poprzez wektor współrzędnych naturalnych $\mathbf{q}_n$ – wyznaczono jakobian dla tej pozy. Następnie podano pewien wektor pożądanych prędkości w przestrzeni zewnętrznej. Biorąc pod uwagę równanie przedstawione na początku rozdziału, można skorzystać ze wzoru, by wyznaczyć odpowiednie prędkości w przestrzeni konfiguracyjnej. Problem jednak polega na tym, że omawiany manipulator ma tylko 3 stopnie swobody – nie można zatem po prostu odwrócić macierzy $\mathbf{J}$. Pewnym rozwiązaniem jest skorzystanie z *pseudoodwracania*[3] poprzez funkcję `pinv`. Takie podejście spowoduje, że zminimalizowana zostanie norma: $||\mathbf{J}\dot{\mathbf{q}} - \boldsymbol{\nu}|$, czyli różnica jest pomiędzy prędkościami zadanymi a faktycznie otrzymanymi.

Porównanie prędkości zadanych i otrzymanych:

```
vel =

         0    0.2500         0         0         0         0


xd =
```

[3] Prawdopodobnie najlepsze tłumaczenie terminu *pseudo-inverse*

```
   -0.0134    0.1765    0.0111    0.0426   -0.1029   -0.0165
```

Różnice są znaczne – zadano jedynie prędkość wzdłuż osi x przy pozostałych zerowych, a otrzymane rozwiązanie posiada wszystkie prędkości niezerowe. Funkcja `pinv` starała się zminimalizować błędy, ale te i tak pojawiły się w rozwiązaniu.

Inny sposób: omawiany manipulator ma 3 stopnie swobody, zatem wykorzystano je w inny sposób zadając tylko 3 prędkości.

```
% ,,Kwadratyzacja'' jakobianu, poprzez usunięcie 3 ostatnich wierszy.
J_xyz = J(1:3, :);

% Wyznaczenie odpowiednich prędkości q,
% ale tylko dla prędkości względem osi x, y, z.
qd = inv(J_xyz) * vel(1:3)';

% Rezultat.
xd = J * qd; xd = xd'
```

Usunięto 3 ostatnie wiersze jakobianu, tak by stał się kwadratowy. W ten sposób zignorowane zostaną wpływy zmian współrzędnych naturalnych na orientację. Wektor pożądanych prędkości także ograniczono do pierwszych 3 składowych – w ten sposób jakobian i zadane prędkości są zgodne ze sobą. Teraz można po prostu odwrócić uproszczony jakobian.

Rezultat:

```
xd =

         0    0.2500         0    0.0572   -0.1506   -0.0270
```

Tym razem udało się osiągnąć zadane prędkości: ν_x, ν_y, ν_z. Wciąż prędkości obrotowe są niezerowe, ale jawnie zezwolono na to, upraszczając jakobian i wektor prędkości zadanych. Pewnych ograniczeń manipulatora o niewystarczającej liczbie stopni swobody nie sposób obejść.

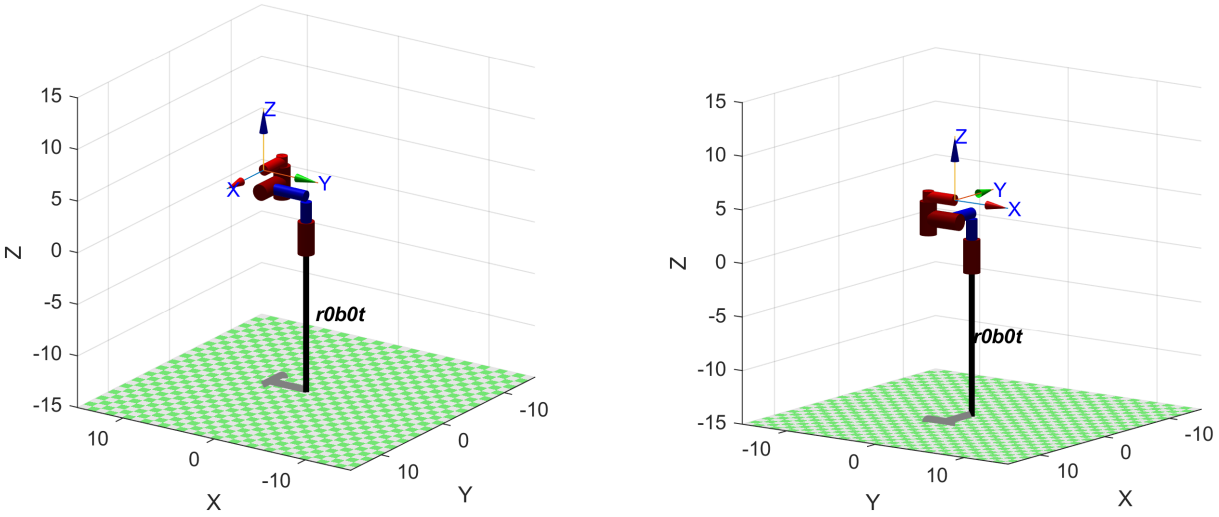
Kolejną ważną kwestią istotną z punktu widzenia jakobianów są osobliwości. Osobliwości kinematyczne to takie konfiguracje manipulatora dla których $\det(\mathbf{J}) = 0$. Możemy je podzielić na dwie grupy:

– osobliwości graniczne przestrzeni roboczej – pojawiają się przy w pełni złożonych/rozłożonych członach,

– osobliwości wnętrza przestrzeni roboczej – pojawiają się gdy w jednej linii ustawione są co najmniej dwie osie par obrotowych lub przesuwnych.

Manipulator, który znajduje się w konfiguracji osobliwej utracił jeden lub więcej stopni swobody, co znacznie utrudnia ruch elementu wykonawczego.

Przykładowo, dla badanego manipulatora OOO konfiguracja osobliwa wystąpi gdy współrzędne naturalne to: $\mathbf{q} = \begin{bmatrix} 0 & 0 & \frac{\pi}{2} \end{bmatrix}$. Manipulator w tej konfiguracji widoczny jest na rysunku 8 poniżej.



Rysunek 8: Manipulator w konifguracji osobliwej

Kod:

```
% Współrzędne naturalne.
q = [0, 0, pi/2];

% Jakobian.
J = rob.jacob0(q);

% Część jakobianu odpowiadająca prędkościom liniowym
J_xyz = J(1:3, :);

% Czy wystąpiła osobliwość?
jsingu(J_xyz)
det(J_xyz)
rank(J_xyz)
```

Górna „kwadratowa” część jakobianu (`J_xyz`) to:

```
J_xyz =

   -1.0000    2.0000   -3.0000
    4.0000         0    0.0000
         0   -0.0000         0
```

Należy zwrócić uwagę na dwie rzeczy:

– ostatni rząd macierzy, odpowiadający prędkości wzdłuż osi z, stanowią zera,

– druga i trzecia kolumna macierzy, odpowiadające współrzędnym naturalnym q_2 i q_3, są do siebie proporcjonalne.

Wyznacznik tej macierzy jest zerowy[4], a jej rząd zmniejszył się o jeden do 2.

```
ans =

   2.2044e-15


ans =

     2
```

Użycie funkcji `jsingu` zwraca informację, że:

```
1 linearly dependent joints:
  q3 depends on: q2
```

Co zaobserwowano już wcześniej, analizując jakobian (fragment wyżej).

Dokonano drobnej zmiany wektora $\mathbf{q}$, wychodząc z konfiguracji osobliwej

```
% Współrzędne naturalne -- poprzednie.
q = [0, 0, pi/2];

% Współrzędne naturalne -- aktualne.
q = [0, 0, pi/2 + 1 * pi/180];
```

Zmianie uległa ostatnia współrzędna – dodany został kąt $1°$. Teraz jakobian nie jest macierzą osobliwą

```
ans =

   -0.6227
```

choć wyznacznik jest wciąż bardzo małą liczbą. Zbadano wpływ tej konfiguracji na wyznaczanie prędkości.

```
% Pożądane prędkości liniowe.
vel = [0.0, 0.0, 0.1];

% Odpowiadające im prędkości q.
qd = inv(J_xyz) * vel'
```

Zadany został wektor prędkości liniowych `vel` i korzystając z jakobianu obliczono odpowiednie prędkości w przestrzeni wewnętrznej $\dot{\mathbf{q}}$. W rezultacie otrzymano:

```
qd =

    0.0168
    1.9100
    1.2679
```

[4]Precyzyjniej: jest pomijalnie mały, a jego niezerowość wynika z niedokładności obliczeń.

Dla porównania dokonano tych samych obliczeń dla innej, znacznie dalszej od punktu osobliwego konfiguracji manipulatora. Wtedy:

```
qd =

    0.0217
    0.0975
    0.0628
```

Gdy zadano prędkość liniową wzdłuż osi z, dla konfiguracji bliskiej osobliwej, prędkości wewnętrzne q_2, q_3 rosną znacząco – im bliżej osobliwości tym bliższe są one nieskończoności.

Osobliwość nie oznacza jednak, że manipulator *zupełnie* nie potrafi wykonywać zadanych ruchów. Dla przykładu zadano inny wektor prędkości: $\boldsymbol{\nu} = [0 \quad 0.1 \quad 0]$, czyli zamiast osi z wybrano oś y, i w rezultacie otrzymano:

```
qd =

    0.0252
   -0.0000
   -0.0084
```

Tym razem otrzymane prędkości są znacznie mniejsze, co dowodzi, że analizowana konfiguracja nie jest problematyczna dla tego ruchu.

Przy analizie osobliwości manipulatora przydatne mogą być elipsoidy prędkości. Dla kodu:

```
1 rob.teach('callback', @(r, q) r.vellipse(q));
```

pojawi się manipulator, którym można sterować, a przy członie wykonawczym będzie odpowiednia elipsoida. Wygodniej jednak, dla celów raportu, będzie wykorzystać metodę `vellipse`, która pokaże samą elipsoidę dla zadanej konfiguracji.

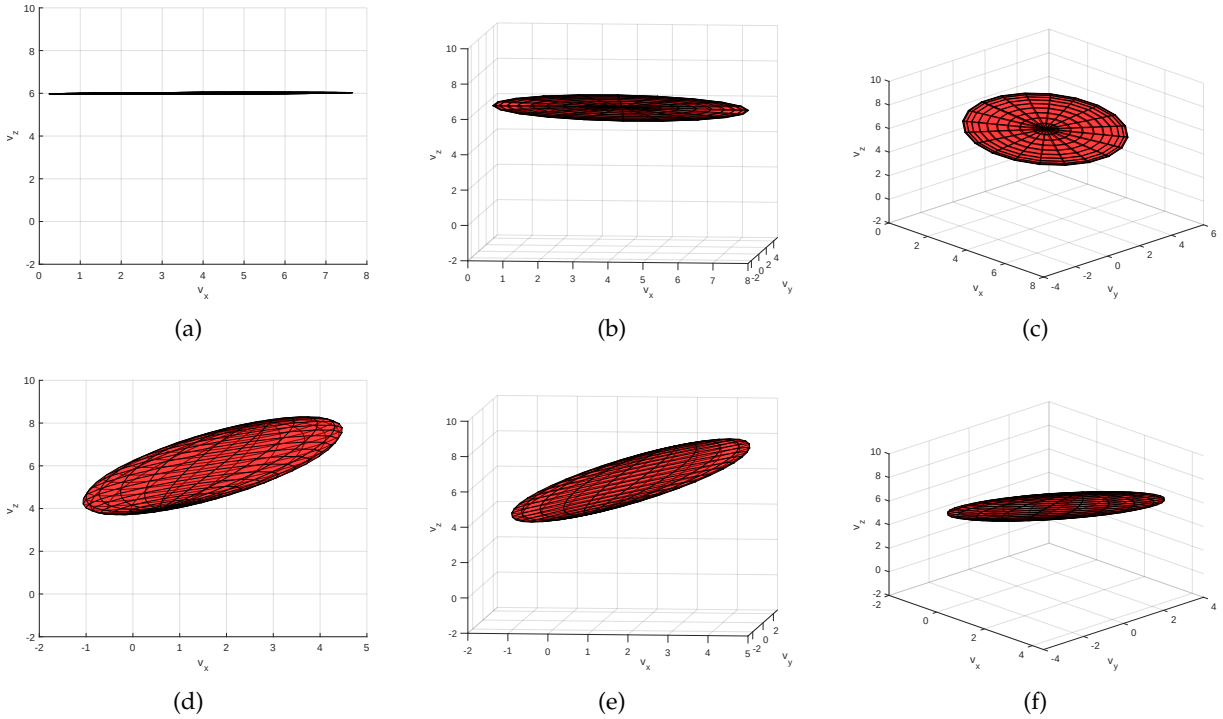
Przykładowy kod:

```
% Konfiguracja.
qns = [0, 0, pi/2 + 1 * pi/180];

% Elipsoida prędkości -- dla ruchu liniowego.
rob.vellipse(qns, 'trans');

% Współczynnik manipulacyjności.
m = rob.maniplty(qns, 'trans')
```

Porównano dwie konfiguracje: jedną bliską osobliwej i drugą dalszą od niej. Elipsoidy dla nich można zobaczyć na rysunku 9.

Dla konfiguracji bliskiej osobliwej (rysunki a, b, c), elipsoida jest bardzo płaska – blisko jej do elipsy z pominięciem wymiaru z. W przypadku drugiej konfiguracji, elipsoidę cechuje już znacznie większa objętość i bliżej jej do kuli w porównaniu do konf. prawie osobliwej.

Do podobnych wniosków można dojść badając odpowiedni współczynnik, który jest propocjonalny do objętości elipsoidy.



(a) (b) (c)

(d) (e) (f)

Rysunek 9: Elipsoidy prędkości
a, b, c – blisko osobliwości d, e, f – znacznie dalej

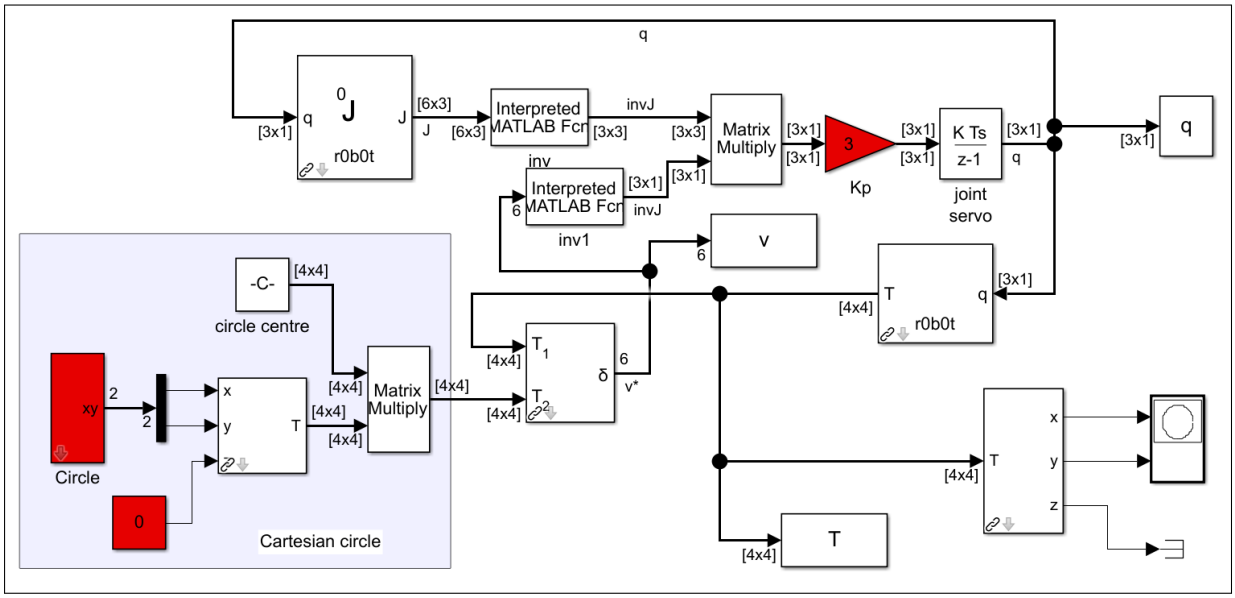
```
m =

    0.6227

m =

    7.1637
```

Pierwsza wartość dotyczy konfiguracji bliskiej osobliwej. Większa wartość oznacza lepszą konfigurację. Taki współczynnik nie daje natomiast informacji o tym, wzdłuż której osi ruch jest trudniejszy.

Podobną analizę można prowadzić dla ruchu obrotowego.

4 Jakobian – Simulink (10.06.2021)

W ramach tych ćwiczeń należało przygotować jeden z przykładowych programów w Simulinku, który domyślnie jest napisany dla robota Puma 560, dla badanego manipulatora. Schemat po odpowiednich zmianach widoczny jest na rysunku 10.



Rysunek 10: Schemat w Simulinku

Dokonane zmiany to:

- blok wyznaczający jakobian ${}^0\mathbf{J}$ na podstawie aktualnej konfiguracji $\mathbf{q}$ jest teraz wyznaczany dla omawainego robota (o nazwie „r0b0t"),
- podobnie z blokiem rozwiązującym zadanie proste kinematyki,
- a także z blokiem *circle centre*, który wyznacza środek okręgu względem wybranej początkowej konfiguracji manipulatora,
- blok *Interpreted MATLAB Fcn* następujący po bloku ${}^0\mathbf{J}$, który domyślnie odwraca jakobian, dodatkowo usuwa z niego 3 ostatnie rzędy dotyczące prędkości obrotowych, by był macierzą kwadratową,
- dodany został kolejny blok *Interpreted MATLAB Fcn*, który z wektora $\boldsymbol{\nu}^*$ usuwa trzy ostatnie elementy dotyczące prędkości obrotowych.

Poza tym, w ramach testów, pewnym zmianom ulegały: parametry okręgu (blok *Circle*) oraz wzmocnienie *Kp*.

Zadanie, które stoi przed manipulatorem to ruch (elementem wykonawczym) wzdłuż trajektorii w kształcie okręgu. Przedstawiony schemat stanowi zamknięty układ regulacji z regulatorem typu P. By wykonać takie zadanie, system sterowania manipulatorem:

- wyznacza jakobian dla aktualnej konfiguracji,

– wyznacza pożądane prędkości w przestrzeni kartezjańskiej potrzebne do realizacji celu,

– na podstawie dwóch powyższych oblicza prędkości w przestrzeni wewnętrznej ($\dot{\mathbf{q}}$).

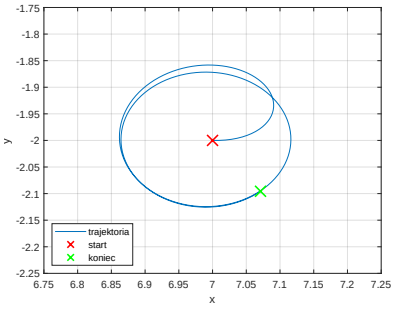
Zamknięcie układu regulacji w pętli sprzężenia zwrotnego pozwala na likwidację problemów związanych z kumulacją błędów. Regulując wzmocnienie części P, można wpływać na intensywność, z jaką sygnały sterujące będą starały się likwidować wspomniane błędy.

Porównanie otrzymanych trajektorii dla kilku ze sprawdzonych wzmocnień widać na rysunku 11 poniżej.

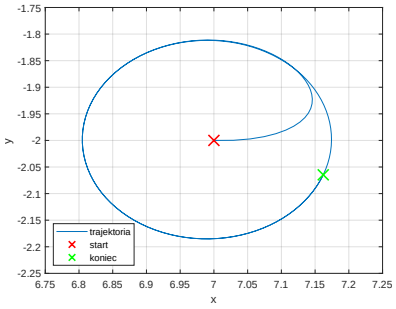
Dla małych wzmocnień ($K_p = 1$, $K_p = 3$) narysowany przez efektor okrąg nie jest zgodny z zadanymi parametrami (zbyt mały promień). Poczynając od $K_p = 5$ okrąg ma już właściwy promień. Dla wzmocnienia znacznie większego, czyli $K_p = 20$, efektor pod sam koniec zaczyna schodzić z poprawnej trasy – drobne błędy zbyt mocno wpływają na sygnały sterujące.

Schemat pozwala w łatwy sposób odczytać wyznaczone prędkości $\dot{\mathbf{q}}$ i $\boldsymbol{\nu}$. Przykładowo, dla wzmocnienia $K_p = 5$ zostały umieszczone na rysunku 12.

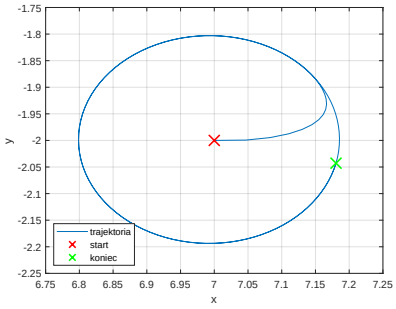
Zgodnie z oczekiwaniami, zadanie ruchu po obwodzie okręgu na płaszczyźnie XY, nie wygenerowało prędkości wzdłuż osi z. Co do członów manipulatora, jak widać do ruchu nie została wykorzystana druga para kinematyczna.



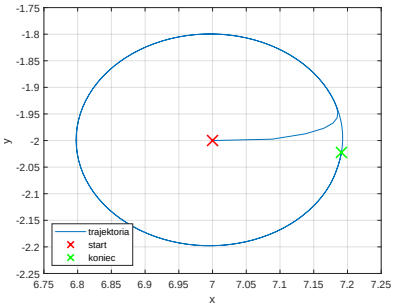
(a) $K_p = 1$



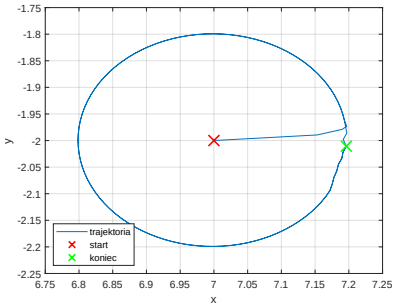
(b) $K_p = 3$



(c) $K_p = 5$

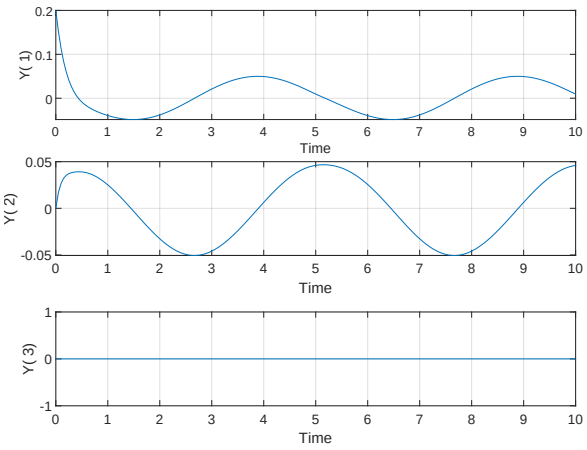


(d) $K_p = 10$

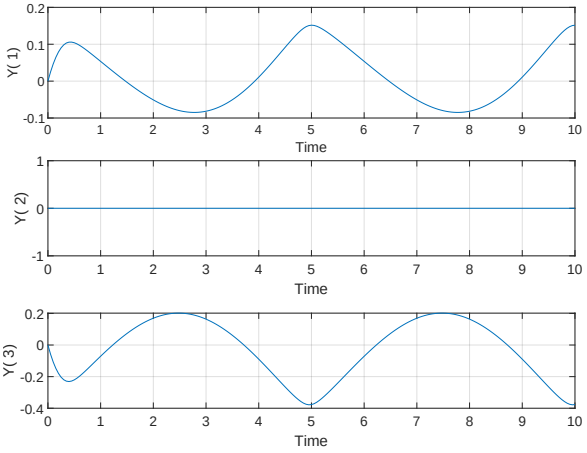


(e) $K_p = 20$

Rysunek 11: Porównanie otrzymanych trajektorii dla zmian wzmocnienia K_p



(a) prędkości $\boldsymbol{\nu}$ – dla ruchu liniowego



(b) prędkości $\dot{\mathbf{q}}$

Rysunek 12: Wykresy prędkości dla $K_p = 5$

5 Dynamika (14.06.2021, 17.06.2021)

Dynamika robota jest opisywana poprzez następujące równanie macierzowe:

$$\mathbf{Q} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{F}(\dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q})$$

gdzie:

- $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$ – wektory współrzędnych, prędkości i przyspieszeń uogólnionych,
- $\mathbf{M}$ – macierz inercji/bezwładności, która jest symetryczna,
- $\mathbf{C}$ – macierz sił Coriolisa i odśrodkowych (którą można rozdzielić na dwie osobne macierze dla wspomnianych sił),
- $\mathbf{F}$ – wektor sił tarcia,
- $\mathbf{G}$ – wektor sił grawitacji,
- $\mathbf{Q}$ – wektor sił i momentów działających na manipulator.

Proste zadania dynamiki to takie, w których należy znaleźć wektor $\mathbf{Q}$ dla podanej konfiguracji manipulatora, czyli wektorów $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$. Odwrotne zadanie zajmuje się tym samym, ale w odwrotnej kolejności.

Do wyznaczania wektora $\mathbf{Q}$ można wykorzystać metodę `rne`, np. w następujący sposób:

```
% Konfiguracja
q = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1];
qd = [0.05, 0, 0, 0.05, 0, 0];
qdd = [0, 0, 0, 0, 0, 0];

% Wektor Q
Q = p560.rne(q, qd, qdd)
```

Ta metoda wyznacza odpowiednie siły i momenty napędowe wykorzystując iteracyjny algorytm Newtona-Eulera, który składa się z dwóch faz:

faza I: od podstawy do el. wykonawczego wyznaczane są prędkości oraz przyspieszenia każdego członu,

faza II: od el. wykonawczego do podstawy wyznaczane są siły i momenty.

Do wyznaczenia elementów macierzy występujących w równaniu dynamiki można skorzystać z kilku poleceń:

- `gravload` – wektor sił grawitacji,
- `inertia` – macierz bezwładności,
- `coriolis` – macierz sił Coriolisa,
- `friction` – wektor sił tarcia.

Tym poleceniom podawane są na wejście wektory odpowiednich współrzędnych, zgodnie z równaniem dynamiki.

Macierze te zależą nie tylko od konfiguracji robota, ale także innych parametrów, których wartość można sprawdzić korzystając z pola `dyn` odpowiednich członów, na przykład:

```
>> p560.links(3).dyn
Revolute(std): theta=q, d=0.15005, a=0.0203, alpha=-1.5708, offset=0
  m    = 4.8
  r    = -0.0203     -0.0141     0.07
  I    = | 0.066       0           0           |
         | 0           0.086       0           |
         | 0           0           0.0125      |
  Jm   = 0.0002
  Bm   = 0.00138
  Tc   = 0.132      (+) -0.105     (-)
  G    = -53.71
  qlim = -3.926991 to 0.785398
```

gdzie:

– najpierw wypisane są parametry kinematyczne,

– `m` oznacza masę członu,

– `r` – pozycję środka masy (COM),

– `I` – bezwładność członu,

– `Jm` – bezwładność silnika,

– `Bm` – tarcie silnika,

– `Tc` – tarcie Coulomba (dla obu kierunków ruchu),

– `G` – przełożenie przekładni,

– `qlim` – dostępne zakresy współrzędnych uogólnionych.

Niestety, wprowadzenie tych parametrów dla badanego manipulatora okazało się przerastać możliwości członków sekcji, dlatego zadania z tej sekcji wykonano dla robota Puma.

zad. 1

a)

Dalsze zwiększanie wzmocnienia P regulatora powoduje zwiększanie przeregulowania oraz pojawienie się oscylacji, nie powoduje natomiast zmniejszenia błędu w stanie ustalonym. Z kolei zmniejszanie tego wzmocnienia tylko pogarsza jakość regulacji – odpowiedź układu jest znacznie wolniejsza.

Dodanie części I znacznie poprawia pracę układu. Łącząc części P i I, uzyskano poprawnie działający układ regulacji, który odpowiada na zmiany wejścia z satysfakcjonującą prędkością, a uchyb dąży do zera.

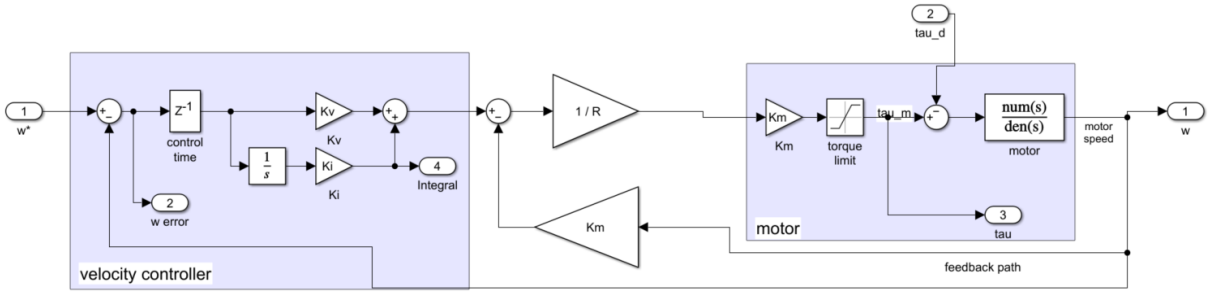
b)

By wykonać zadanie, zgodnie ze schematem blokowym zapisano zmienne odpowiadające za: regulator P/PI, transmitancję silnika, wzmocnienie Km. Korzystając z polecenia `feedback` zamknięto układ w pętli sprzężenia zwrotnego, a korzystając z `rlocus` zbadano linie pierwiastkowe.

Zmiany parametrów regulatora wykazały, że nastawy regulatora nie mają wpływu na stabilność układu, który zawsze jest stabilny (więc maksymalne dozwolone wzmocnienie jest nieskończone), a jedynie na jakość regulacji.

c)

Zaproponowano sposób modyfikacji `vloop`, który widać na rysunku 13.



Rysunek 13: Sterowanie napięciem oraz efekt EMF

Dodane zostały dwa bloki w środku pomiędzy *velocity controller*, a *motor*. Dzielenie przez rezystancję związane jest z konwersją napięcia na wyjściu regulatora na prąd, którym sterowany jest motor, a ujemne napięcie EMF zależy od prędkości silnika i jest proporcjonalne do stałej K_m.

d)

Zjawisko windupu całkowania występuje, gdy wypracowywany przez regulator sygnał jest wyższy niż wartości graniczna urządzenia wykonawczego. Uchyb regulacji jest cały czas całkowany. Powoduje to zauważalne opóźnienia sygnału wyjściowego, np. w sytuacji, gdy nagle skokowo zmienimy wartość zadaną.

Najłatwiejszym rozwiązaniem tego problemu jest zatrzymanie pracy części całkującej, gdy sygnał sterujący osiąga zbyt duże (lub zbyt małe) wartości.

zad. 2

Należy wykazać, że:

$$\mathrm{N\,m\,A^{-1}} = \mathrm{V\,s\,rad^{-1}}$$

Wiedząc, że:

$$\mathrm{N} = \mathrm{kg\,m\,s^{-2}} \qquad \mathrm{V} = \mathrm{kg\,m^2\,A^{-1}\,s^{-3}}$$

Biorąc pod uwagę wspomniane zależności, po przekształceniach otrzymano:

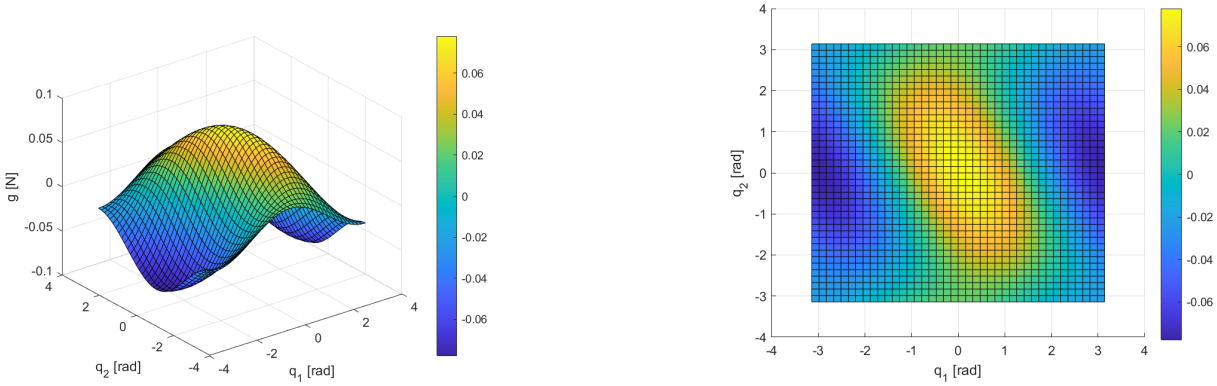
$$\mathrm{kg\ m^2\ s^{-2}\ A^{-1}} = \mathrm{kg\ m^2\ s^{-2}\ A^{-1}\ rad^{-1}}$$

Wiedząc, że radian jest jednostką bezwymiarową, można usunąć ją z wyrażenia, dzięki czemu obie strony równania będą sobie równe.

zad. 3

a)

Zakłócenie pochodzące ze strony siły grawitacji w funkcji współrzędnych uogólnionych widać na rysunkach 14.



(a) (b)

Rysunek 14: Zadanie 3a

Siła ta jest największa, gdy człony są wyprostowane.

b)

Bezwładność członu 1 w funkcji położenia członu 2 widać na rysunku 15.

zad. 4

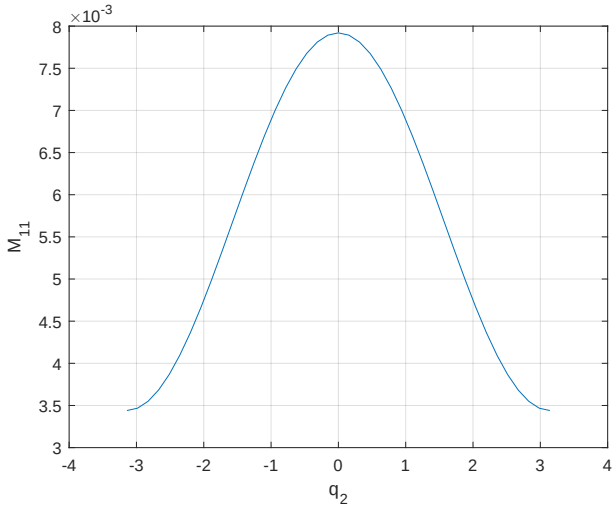
Na wykresach 16 zostały przedstawione siły grawitacji działające na człony w konfiguracji współrzędnych q_2 i q_3. Na rysunkach dolnych widać zmiany wprowadzone poprzez dodanie obciążenia do efektora manipulatora.

Dodanie obciążenia (2.5 kg) spowodowało wzrost sił grawitacji.

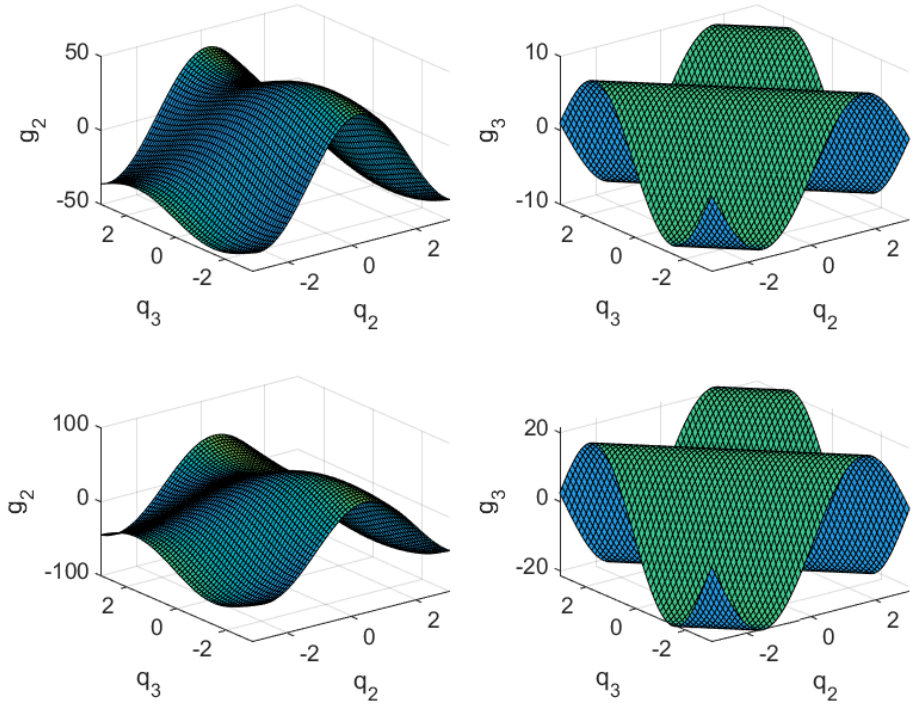
zad. 5

Podobnie jak w zadaniu poprzednim, na wykresach 17 widać wpływ konfiguracji manipulatora na bezwładność członów. Na dolnych wykresach widać wpływ obciążenia.

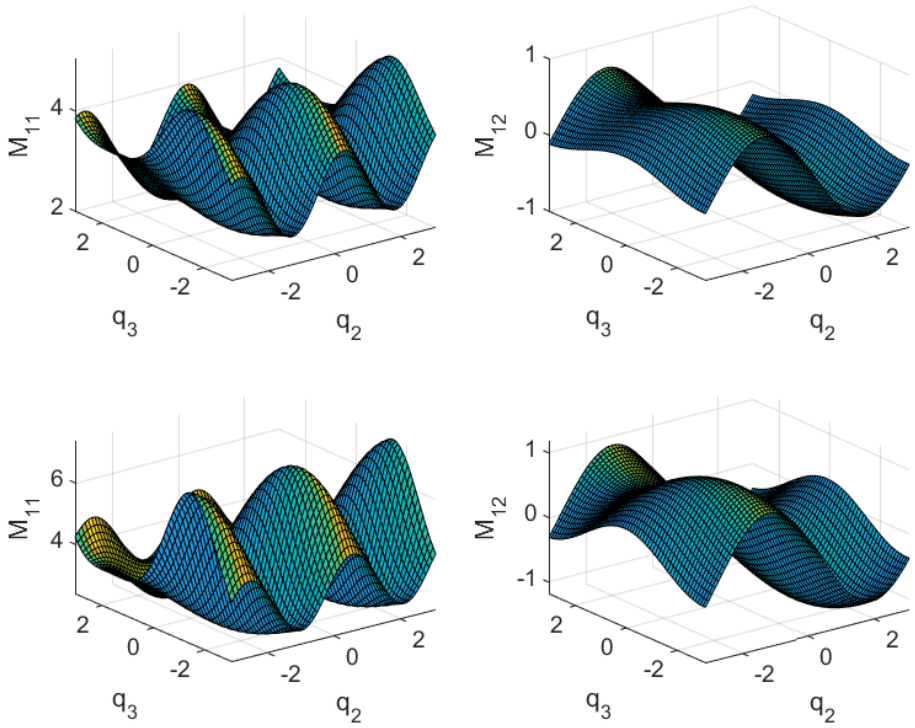
Zgodnie z oczekiwaniami, dodanie obciążenia spowodowało odpowiedni wzrost sił bezwładności.



Rysunek 15: Bezwładność członu 1 w funkcji położenia członu 2



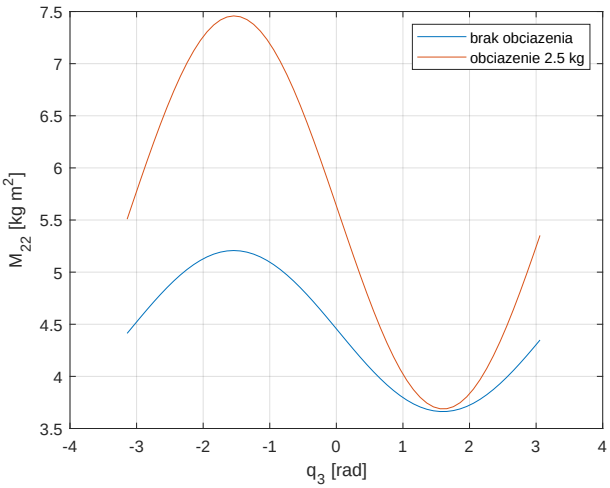
Rysunek 16: Siły grawitacji w funkcji konfiguracji, oraz wpływ obciążenia



Rysunek 17: Siły bezwładności w funkcji konfiguracji, oraz wpływ obciążenia

zad. 6

Wpływ obciążenia na siłę bezwładności działającą na człon 2 w funkcji położenia członu 3 widać na rysunku 18.



Rysunek 18: Wpływ obciążenia na siłę bezwładności działającą na człon 2 w funkcji położenia członu 3

Tak jak w poprzednim zadaniu widać, że dodanie obciążenia prowadzi do zwiększenia tej siły.

zad. 7

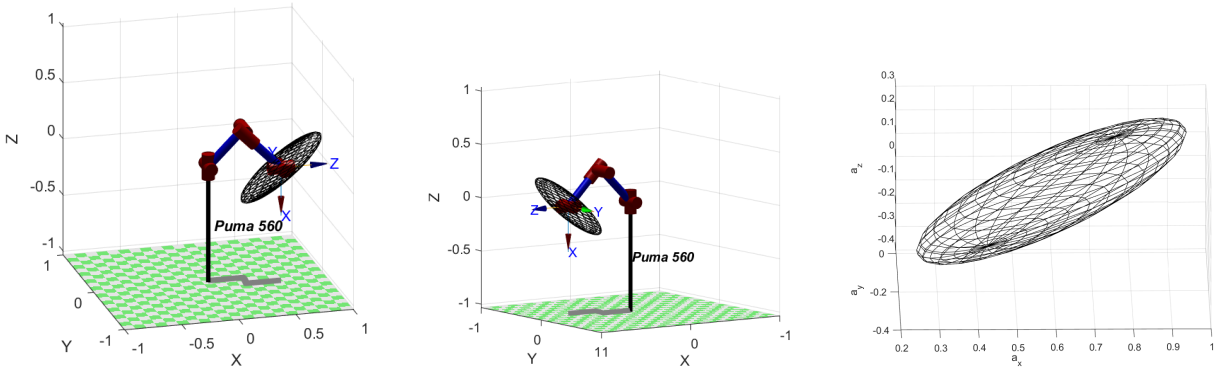
Dodanie obciążenia powoduje wzrost wartości średniej bezwładności w całym dostępnym zakresie współrzędnej uogólnionej, ale także znacznie zwiększa zakres odczuwanej bezwładności.

zad. 8

Można to powiązać z trzecią zasadą dynamiki Newtona. Na głównej diagonali znajdują się siły bezwładności odczuwane przez konkretny j-ty człon, a poza nią znajdują się siły, które wynikają z wpływu bezwładności jednego członu na inny człon.

zad. 10

Na rysunkach 19 pokazana została elipsoida przyspieszenia przestrzennego dla danej konfiguracji manipulatora.



(a) (b) (c)

Rysunek 19: Elipsoida przyspieszenia przestrzennego

Główna oś elipsoidy określa kierunek w którym efektor ma największe przyspieszenia dla danej konfiguracji. Im elipsoida bliższa jest kuli, tym łatwiej jest manipulatorowi przyspieszać w dowolnym kierunku. Miarę tej wartości można wyznaczyć dzieląc najmniejszy promień przez największy.

zad. 11

a)

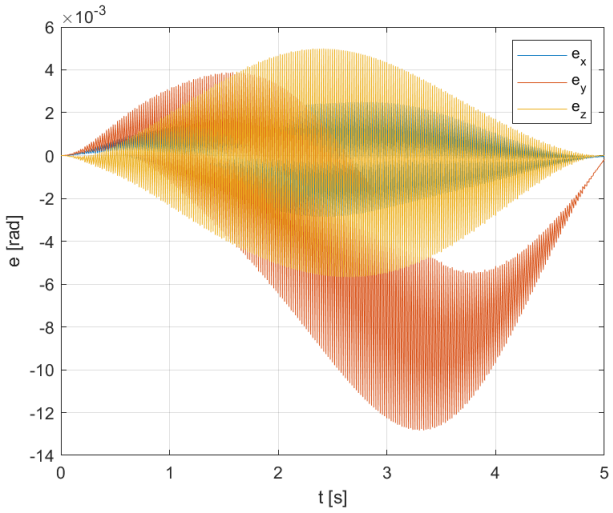
Przykładowe błędy dla ustawień domyślnych widać na rysunku 20.

Przy czym warunki początkowe, końcowe i czas zadania to:

$$\text{początek:} \quad q_0 = [0\ 0\ 0\ 0\ 0\ 0]$$

$$\text{koniec:} \quad q_g = \left[\frac{\pi}{4}\ \frac{\pi}{2}\ -\frac{\pi}{2}\ 0\ 0\ 0\right]$$

$$\text{czas:} \quad 5\ \text{s}$$



Rysunek 20: Błędy

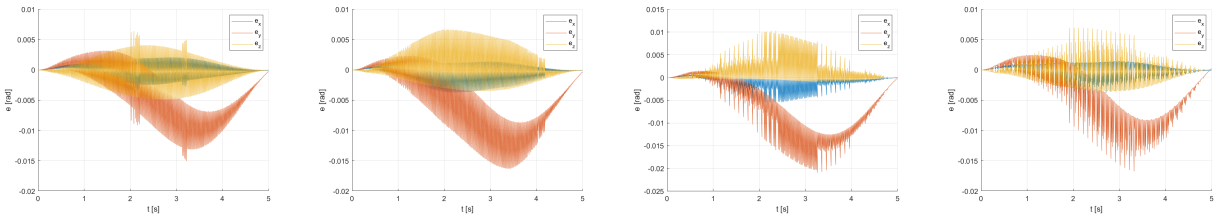
Błędy w trakcie realizacji zadania są stosunkowo niewielkie, rzędu 10^{-3}.
Przeprowadzono kilka eksperymentów związanych ze zmianami parametrów:

- Ustawienie zbyt małego oczekiwanego czasu realizacji trajektorii powodowało zwiększenie błędów.
- Zwiększanie wzm. części P pozwoliło zmniejszyć błędy regulacji, ale zwiększało oscylacje. Natomiast zmniejszanie P, powodowało znaczne błędy w fazie końcowej symulacji.
- Zwiększanie oraz zmniejszanie wzm. D nie przyniosło większych zmian. Przy całkowicie wyłączonym D, drobnemu pogorszeniu uległ błąd w stanie końcowym.

Badania w pewnym stopniu komplikował fakt, że symulacja jest bardzo zasobożerna i trwa stosunkowo długo.

b)

Na wykresach21 widać wpływ zwiększania czas obliczeń T_{fb} na podążanie manipulatora za trajektorią. Czas domyślny to: $T_{fb} = 0.002$ s (rysunek20).



(a) $T_{fb} = 0.005$ s (b) $T_{fb} = 0.01$ s (c) $T_{fb} = 0.02$ s (d) $T_{fb} = 0.05$ s

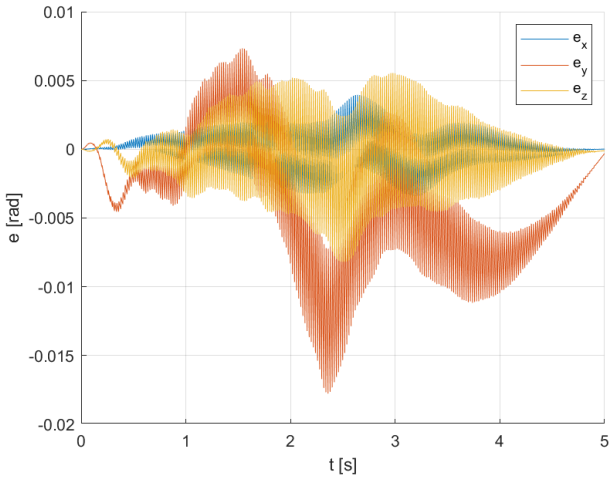
Rysunek 21: Zmniejszanie tempa obliczania momentu

Wraz ze zwiększaniem tego czasu, pogarsza się praca manipulatora. System sterowania musi generować większe siły, przez to że wyznacza odpowiednie momenty i siły rzadziej. Dla

jeszcze wyższych czasów T_{fb}, system sterowania zaczyna tracić stabilność.

c)

W tym podpunkcie użyta została metoda `perturb` do symulacji nie w pełni znanego modelu manipulatora. Zmian można dokonać w następujący sposób: *RNE* → *MATLAB Fcn* → jako pierwszy argument funkcji `rne` podaje się nie sam manipulator (`robot`), ale jego zakłócony model (`robot.perturb(0.1)`). Wpływ takich zmian widać na rysunku 22.



Rysunek 22: Wpływ zakłóconego modelu na błędy

Porównując ten wykres z 20 można zauważyć, że błędy nieco urosły (maksymalny błąd wzrósł o kilkadziesiąt procent). Znacząco zmienił się natomiast kształt wykresu błędów - po perturbacjach wykres jest bardziej chaotyczny, zawiera element losowości, co jest zgodne z oczekiwaniami.

6 Roboty mobilne (21.06.2021, 24.06.2021)

zad. 1

a)

Na początku wyprowadzono odpowiednie wzory:

$$v = 20\ \tfrac{\text{km}}{\text{h}} \approx 5.56\ \tfrac{\text{m}}{\text{s}} \qquad \dot{\theta} = 10\ \tfrac{^\circ}{\text{s}} \approx 0.175\ \tfrac{\text{rad}}{\text{s}}$$

$$\gamma = \tan^{-1}\left(\frac{\dot{\theta}L}{v}\right)$$

Zatem, zakładając model roweru dla pojazdu, pożądany kąt skrętu γ to 0.0629 rad. Stosując sterowanie Ackermanna:

$$\gamma_{\text{L}} = \tan^{-1}\left(\frac{L}{R_{\text{B}} - 0.5 \cdot w}\right) \qquad \gamma_{\text{R}} = \tan^{-1}\left(\frac{L}{R_{\text{B}} + 0.5 \cdot w}\right)$$

Wtedy kąty skrętu dla kół przednich to: γ_{L} = 0.0644 rad, γ_{R} = 0.0614 rad.

b)

Tym razem zmieniono promień skrętu dla modelu roweru R_{B}. Wzory na kąty skrętów są identyczne jak powyżej. Odpowiedź:

R_{B} [m]	10	50	100
γ_{L} [rad]	0.2129	0.0406	0.0201
γ [rad]	0.1974	0.0400	0.0200
γ_{R} [rad]	0.1839	0.0394	0.0198

c)

Obie strony pojazdu muszą poruszać się z tą samą prędkością kątową względu okręgu (ze środkiem w punkcie ICR), zatem:

$$\dot{\theta} = \frac{v}{R_{\text{B}}} = \frac{v_L}{R_{\text{L}}} = \frac{v_R}{R_{\text{R}}}$$

gdzie v oznaczają odpowiednie prędkości liniowe, a R promienie okręgu zataczanych przez odpowiednie części pojazdu. Dalej wyprowadzono wzory na prędkości obrotowe:

$$\omega_{\text{L}} = \frac{\dot{\theta}R_{\text{L}}}{r} \qquad \omega_{\text{R}} = \frac{\dot{\theta}R_{\text{R}}}{r}$$

Przez r oznaczono promień opony kół. Założono, że są takie same. Wtedy:

R_B [m]	10	50	100
ω_L $[\frac{\mathrm{rad}}{\mathrm{s}}]$	82.2222	87.5556	88.2222
ω_R $[\frac{\mathrm{rad}}{\mathrm{s}}]$	95.5556	90.2222	89.5556

zad. 2

W poprzednim zadaniu wspomniano, że:

$$\dot{\theta} = \frac{v}{R_\mathrm{B}} = \frac{v_L}{R_\mathrm{L}} = \frac{v_R}{R_\mathrm{R}}$$

Dodatkowo wiadomo, że prędkość obrotowa koła zależy od prędkości liniowej w następujący sposób:

$$\omega = \frac{v}{r}$$

gdzie r oznacza promień koła. Łącząc powyższe:

$$\dot{\theta}_L = \frac{\omega_L \, r_L}{R_L} \qquad \dot{\theta}_R = \frac{\omega_R \, r_R}{R_R}$$

Jeśli promienie kół się nieznacznie różnią, do wyznaczenia prędkości $\dot{\theta}$ pojazdu można wykorzystać średnią z $\dot{\theta}_L$ i $\dot{\theta}_R$

zad. 3

Znów można zastosować podobne wzory. Przez γ_S oznaczono promień skrętu dla samochodu (L = 4 m), a przez γ_A – promień skrętu dla autobusu (L = 12 m).

R [m]	10	20	50
γ_S [rad]	0.3805	0.1974	0.0798
γ_A [rad]	0.8761	0.5404	0.2355

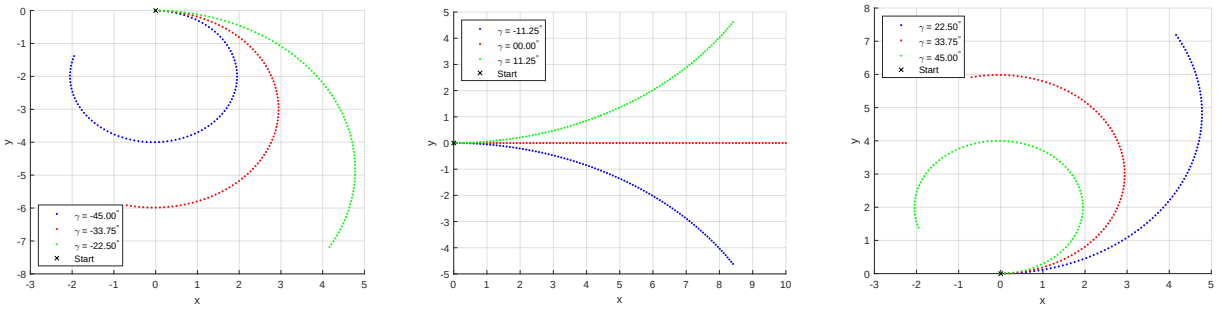
zad. 4

Trajektorie dla badanych stałych kątów skrętu γ umieszczone są na rysunku 23.

zad. 5

Przykładowa implementacja operatora „minus w kole”, inaczej funkcji `angdiff` w MATLABie.

```
function ret = minusOperator(alpha, beta)
    % Różnica pomiędzy kątami.
    diff = alpha - beta;

    % Różnicę należy sprowadzić do przedziału [-pi, pi). W zależności
    % od tego ile aktualnie wynosi, rozważono 3 przypadki.
    if diff >= pi
```



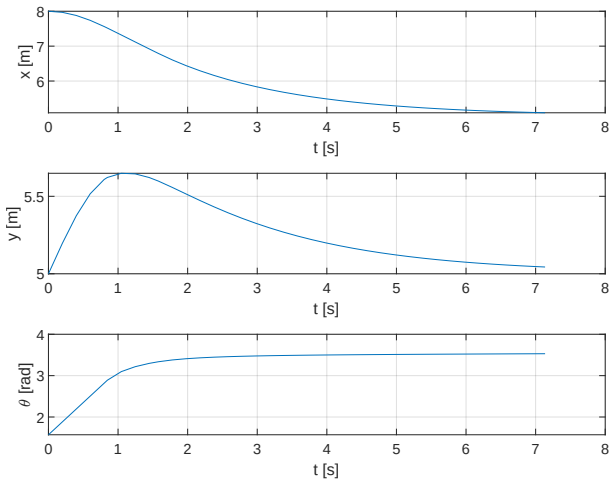
(a) (b) (c)

Rysunek 23: Trajektorie

```
8           tmp = ceil((diff - pi) / (2*pi));
9           ret = diff - tmp * 2*pi;
10      elseif diff < -pi
11          tmp = floor((diff + pi) / (2*pi));
12          ret = diff - tmp * 2*pi;
13      else
14          ret = diff;
15      end
16  end
```

zad. 6

Przebiegi czasowe widać na rysunku 24.



Rysunek 24: Przebiegi x, y i θ względem czasu t

zad. 7

a)

- Zmiany odległości *look-ahead* powodowały zauważalne zmiany w trajektorii pojazdu – im dystans był mniejszy, tym ślad trasy pozostawianej przez pojazd dokładniej pokrywał się z trajektorią ruchomego celu.
- Zmiany wzmocnienia (*heading gain*) regulatora kąta skrętu działały odwrotnie – im wzmocnienie było większe, tym obliczany kąt skrętu był gwałtowniejszy, przez co trajektorie (pojazdu i celu) lepiej ze sobą się pokrywały. Gdy wzmocnienie zostało gwałtownie zmniejszone, wyznaczane kąty skrętu były na tyle małe, że pojazd nie zawsze potrafił nadążyć za zmianą trasy ruchomego celu i oscylował względem niej.
- Zmiany wzmocnienia części P regulatora prędkości nie miały dużego wpływu na samą trajektorię pojazdu. Wpływ widoczny jest przy analizie z perspektywy czasu – gdy cel zmienia swoją trasę, pojazd proporcjonalnie do wzmocnienia P zwiększa swoją prędkość.

b)

Gdy ustawiono wzmocnienie części I na zero, nie uzyskano zadanego dystansu od celu w stanie ustalonym (ale pojazd jest w stanie podążać za celem w linii prostej) – występuje stały błąd o pewnej wartości. Jest to naturalna konsekwencja wyłączenia całkowania w regulatorze.

c)

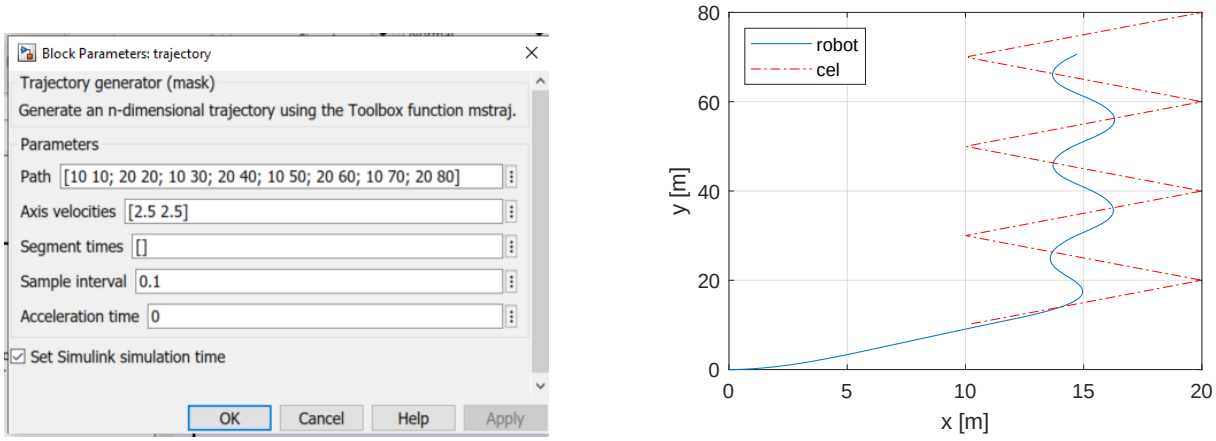
W tej sytuacji, stały błąd w stanie ustalonym zależy od wzmocnienia części P – przy wzmocnieniu dążącym do nieskończoności, błąd ten dąży do zera. Wraz ze wzrostem K_P zaczynają jednak pojawiać się inne niepożądane zjawiska (np. przeregulowania), więc można je zwiększyć tylko do pewnego stopnia.

e)

By robot poruszał się slalomem, określono ruchy celu tak, by jego trajektoria przypominała sygnał trójkątny (rysunek 25).

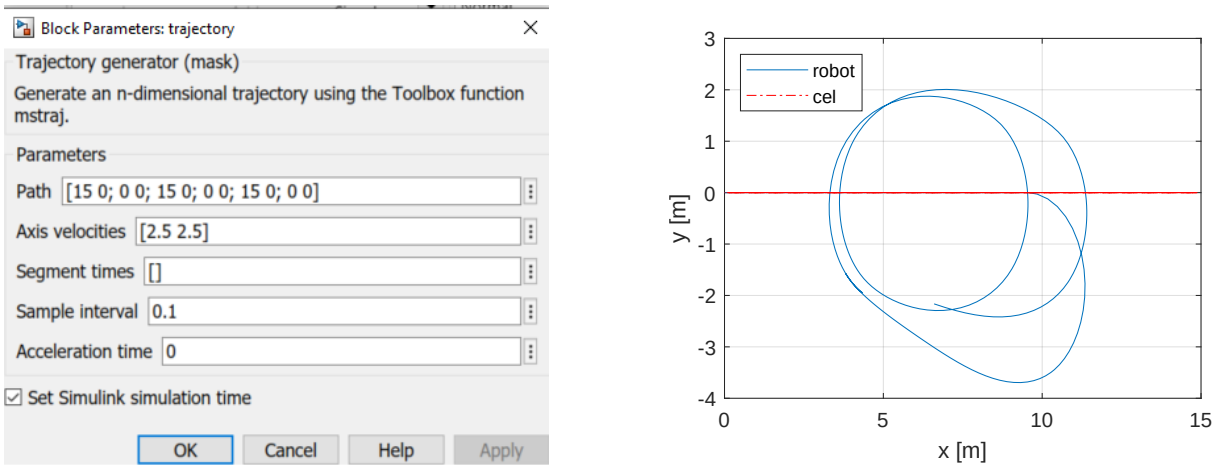
f)

Ponownie należy odpowiednio przystosować punkty trajektorii ruchomego celu do tego zadania. Tym razem podawano na przemian dwa punkty (rysunek 26).



(a) przykładowe ustawienia (b) trajektorie

Rysunek 25: Slalom



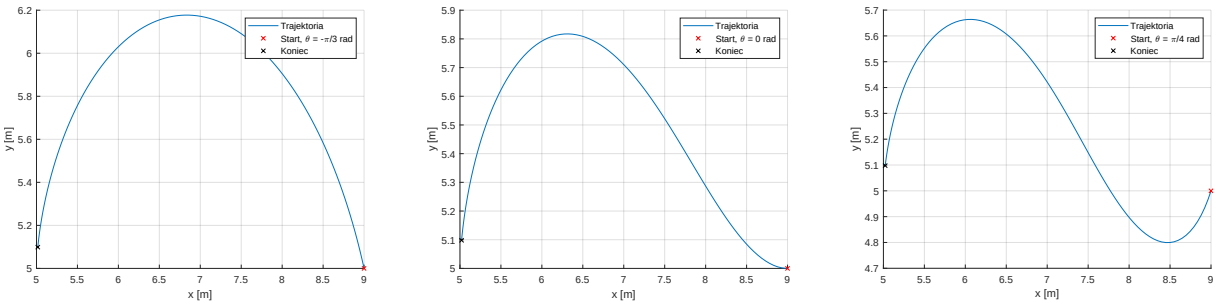
(a) przykładowe ustawienia (b) trajektorie

Rysunek 26: Ruch

zad. 8

a)

Zgodnie z prośbą autora książki, wykonano przykład ponownie dla innych orientacji początkowych. Kilka testów widać na rysunku 27.



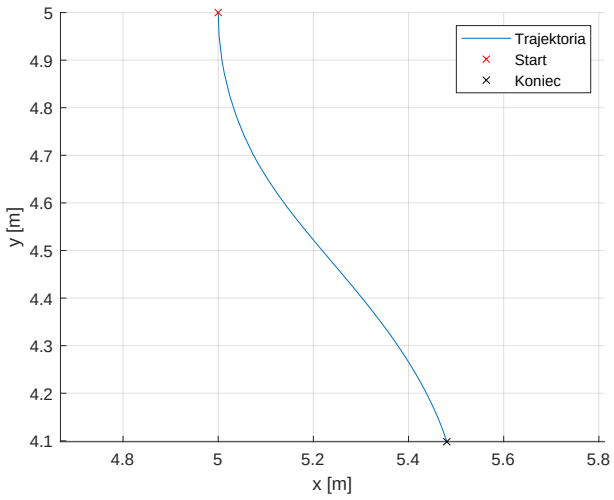
(a) (b) (c)

Rysunek 27: Wpływ orientacji początkowych θ na trajektorię

Dla takich warunków początkowych i takiego położenia/orientacji celowej, zmiana orientacji początkowej nie wpływa na kierunek ruchu pojazdu, tzn. pojazd zawsze porusza się do tyłu[5] .

b)

Manewr parkowania równoległego odtworzono poprzez odpowiedni dobór pozycji początkowych i końcowych.



Rysunek 28: Parkowanie równoległe

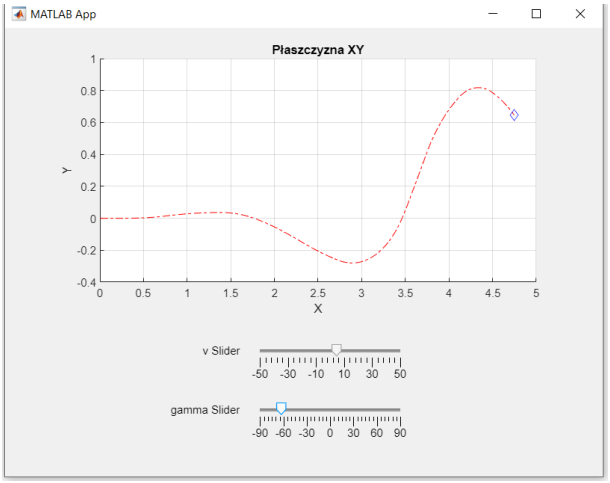
[5]Dobrze to widać na wykresach, które automatycznie uruchamiają się wraz z symulacją w Simulinku, ale niestety nie da się ich osobno zapisać.

c)

- Zmiany wzmocnienia k_ρ wpływały na prędkość pojazdu – im większe wzmocnienie tym szybciej się on poruszał, oczywiście uwzględniając także odległość od samego celu, gdyż prędkość jest do niej proporcjonalna. W przypadku bardzo niskich wzmocnień należało zwiększyć horyzont symulacji, gdyż domyślny nie wystarczał. Dla dużych wzmocnień pojazd, będąc już blisko celu, wciąż poruszał się na tyle szybko, że pozostałe składowe algorytmu sterowania nie były w stanie zapobiec nadmiernym skrętom i oscylacjom.
- Zmiana wzmocnienia k_α, podobnie jak k_ρ wpływała na sposób dotarcia do celu, choć w sposób łatwiej zauważalny. Zwiększanie tego wzmocnienia powoduje, że trajektoria jest bardziej płaska, bardziej przypomina linię prostą – robot wtedy szybciej osiąga odpowiednią orientację, która pozwoli mu dotrzeć do celu.
- Wzmocnienie k_β, tak jak i k_α ingeruje w prędkość kątową pojazdu, ale to wzmocnienie tak oblicza sygnał, by orientacja robota w punkcie celu pokrywała się z zadaną. Odpowiednio dobierając wzmocnienia k_α i k_β można osiągnąć podobne rezultaty.

zad. 9

Na rysunku poniżej pokazane zostało okno prostej aplikacji do sterowania pojazdem, który jest modelowany poprzez rower.



Rysunek 29: GUI do sterowania pojazdem

Pojazdem sterowano przesuwając suwakami: pierwszy z nich dotyczy prędkości liniowej pojazdu (*v Slider*), drugi dotyczący kątu skrętu (*gamma Slider*). Ponad suwakami znajduje się mapa, która aktualizuje się co stały krok czasu, a na nią nanoszone są kolejne punkty, określające aktualne położenie pojazdu.

zad. 10

By przystosować regulatory omówione w podanym rozdziale do pracy z pojazdem wyposażonym w napęd różnicowy, należy zaobserwować kluczową różnicę pomiędzy nim a pojazdem 4-kołowym: sterowanie pojazdem 4-kołowym odbywa się poprzez regulację prędkości liniowej v oraz kąta skrętu γ, a pojazdem z napędem różnicowym – poprzed regulację prędkości liniowych obu kół v_L, v_R. By zaobserwować jak te zmiany wpłyną na dobierane regulatory, rozpatrzono dla przykładu punkt poświęcony docieraniu pojazdu do celu.

Prędkości liniowej pojazdu jest wyznaczana w ten sam sposób, co w rozdziale:

$$v^* = K_v\sqrt{(x^* - x)^2 + (y^* - y)^2}$$

gdzie: (x^*, y^*) – położenie celu, (x, y) – położenie robota, K_v – wzmocnienie. Tym razem jednak tę prędkość należy podać dla obu kół pojazdu, zatem na tym etapie:

$$v_L = v_R = v^*$$

Teraz trzeba uwzględnić orientację robota:

$$\theta^* = \tan^{-1}\left(\frac{y^* - y}{x^* - x}\right)$$

Poprzednio korzystając z tej pożądanej orientacji wyznaczany był kąt skrętu γ[6]:

$$\gamma = K_h(\theta^* \ominus \theta)$$

Tym razem użyto tego samego wyrażenia, ale by wyznaczyć odpowiednią różnicę prędkości kół pojazdu. Zatem, ostatecznie:

$$v_L = \begin{cases} v^* & \gamma \leqslant 0 \\ v^* - \gamma & \gamma > 0 \end{cases} \qquad v_R = \begin{cases} v^* & \gamma \geqslant 0 \\ v^* + \gamma & \gamma < 0 \end{cases}$$

Chodzi zatem o to, że v^* wyznacza pewien poziom prędkości bazowej, wyznaczanej względem celu, a γ zmniejsza *tylko jedną z prędkości*, by uzyskać odpowiednią orientację.

zad. 11

Wiadomo, że:

$$\dot{\theta} = \frac{v_L}{R_L} = \frac{v_R}{R_R}$$

Wartości R_L i R_R są ze sobą powiązane poprzez odległość W, która oznacza dystans dzielący oba koła:

$$R_R = R_L + W \qquad R_L = R_R - W$$

[6]Przez symbol $\ominus$ oznaczono użyty w instrukcji symbol „minusa w kole", który oznacza użycie funkcji `angdiff` w MATLABie

Stosując takie podstawienie do wzorów powyżej otrzymano:

$$\frac{v_L}{R_L} = \frac{v_R}{R_L + W}$$

$$v_L(R_L + W) = v_R R_L$$

$$R_L(v_L - v_R) + wV_L = 0$$

$$\frac{v_L - v_R}{W} = -\frac{v_L}{R_L}$$

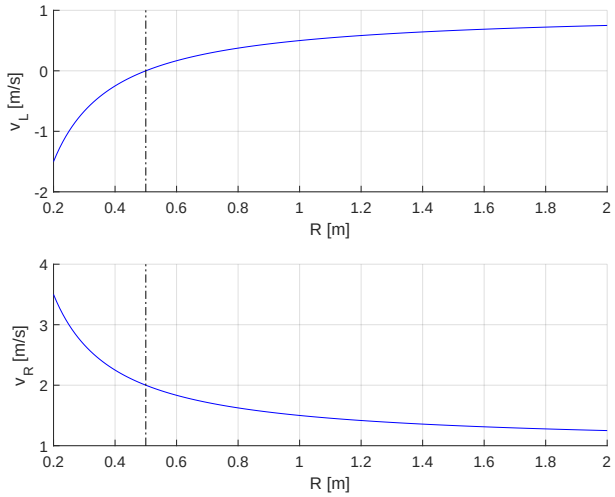
$$\frac{v_R - v_L}{W} = \frac{v_L}{R_L}$$

Iloraz $\frac{v_L}{R_L}$ z definicji oznacza $\dot{\theta}$, zatem:

$$\frac{v_R - v_L}{W} = \dot{\theta}$$

zad. 12

Zakładając pewną odległość pomiędzy kołami W = 1m, wykreślono wykresy.



Rysunek 30: Prędkości v_L, v_R w funkcji promienia okręgu ICR

Czarną linią oznaczony został kluczowy punkt – środek okręgu, czyli punkt ICR, leży dokładnie *pod* jednym z kół, w tym przypadku pod kołem lewym. W takiej sytuacji ta prędkość jest zerowa ($v_L = 0$), a robot obraca się korzystając tylko z drugiego koła. Natomiast gdy promień koła jest jeszcze mniejszy, czyli środek okręgu leży gdzieś pod robotem, prędkość v_L zmienia znak.

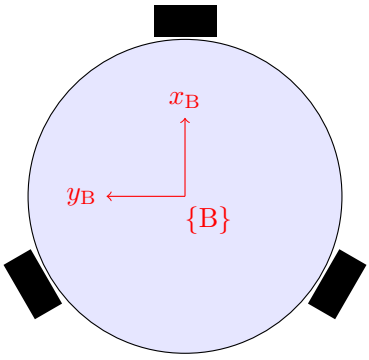
W przypadku drugiego koła, prędkość cały czas odpowiednio się zmienia, ale ciągle ma znak dodatni. Analogiczne rozważania można przeprowadzić dla skrętu w drugą stronę.

zad. 13

Jeśli robotowi wyposażonemu w napęd różnicowy zada się, by poruszał się ze stałą prędkością wzdłuż jego osi y, to robot zacznie się obracać w miejscu. Oś y przebiega wzdłuż osi kół, i zmienia ona swoje położenie wraz ze zmianą położenia robota.

zad. 14

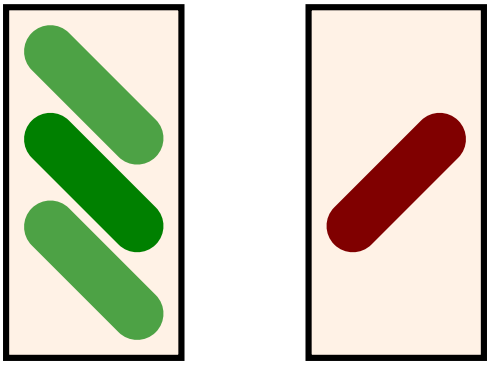
Struktura pojazdu widoczna jest na rysunku 31.



Rysunek 31: Szkic pojazdu

Platforma pojazdu to koło o pewnym promieniu. 3 koła są rozmieszczone tak, że ich osie przecinają się w jednym punkcie.

Robot porusza się korzystając ze szwedzkich kół, których szkice pokazane są na rysunku 32.



(a) widok z góry (b) widok z dołu

Rysunek 32: Koła robota

Na rysunku b, przedstawiona jest jedna rolka, która ma kontakt z podłożem w danej chwili. Umieszczona ona jest pod kątem $\alpha = 45°$ względem osi koła.

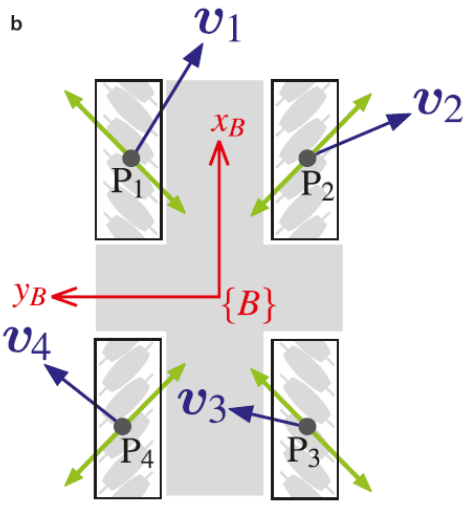
Kod, który przekształca zadane prędkości liniowe i obrotowe pojazdu na prędkości obrotowe kół platformy pokazany jest poniżej.

```
clear; close; clc;

% Parametry pojazdu.
alpha = 45 * pi/180;
r = 0.4;
R = 0.1;

% Zadane prędkości.
```

```
v_star = [1, 0, 0]';
omega_star = 0.5;

% Wektory p, określające położenie kół względem układ współrzędnych B.
p_1 = [1, 0, 0]';
p_2 = [-sin(30 * pi/180), -cos(30 * pi/180), 0]';
p_3 = [-sin(30 * pi/180), cos(30 * pi/180), 0]';

% Wyznaczanie pożądanych prędkości liniowych dla każdego koła.
v_1 = v_star + cross(omega_star * [0, 0, 1]', p_1);
v_2 = v_star + cross(omega_star * [0, 0, 1]', p_2);
v_3 = v_star + cross(omega_star * [0, 0, 1]', p_3);

% Wyznaczanie prędkości obrotowych dla każdego koła.
v_1_w = (v_1(1) - v_1(2) * cot(alpha)) / R
v_2_w = (v_2(1) - v_2(2) * cot(alpha)) / R
v_3_w = (v_3(1) - v_3(2) * cot(alpha)) / R
```

zad. 15

Schemat pojazdu z zadania widoczny jest na rysunku 33.



Rysunek 33: Schemat pojazdu
Źródło: książka, strona nr 113, rysunek 4.18.b

Układ {B} związany z pojazdem umieszczony jest w jego środku geometrycznym i porusza/obraca się wraz z ruchem robota. By zrealizować ruch pojazdu po okręgu, należy:

– ustalić pewną stałą wartość prędkości liniowej układu {B} względem jego osi y,

– ustalić pewną stałą wartość prędkości obrotowej robota.

W ten sposób pojazd będzie poruszał się ze stałą prędkością w kierunku jego osi y, oś ta będzie obracała się wraz z prędkością obrotową. W ten sposób oś x układu będzie cały czas wskazywała środek okręgu.

Teraz trzeba odpowiednio dobrać prędkości ${}^{B}v_y$ i ${}^{B}\omega$, by zadany ruch był realizowany po okręgu o promieniu 0.5m.

$$\text{promień:} \qquad r = 0.5 \text{ m}$$

$$\text{obwód:} \qquad L = 2\pi r = \pi \text{ m}$$

Założono pewną stałą wartość prędkości liniowej:

$${}^{B}v_y = \frac{\pi}{4} \ \frac{\text{m}}{\text{s}}$$

Jeśli w ciągu sekundy robot pokona taki dystans, to oznacza to, że pokona $1/4$ obwodu koła – zatem jego orientacja musi się w ciągu sekundy zmienić o $90°$, czyli:

$${}^{B}\omega = \frac{\pi}{2} \frac{\text{rad}}{\text{s}}$$

Korzystając z kodu przedstawionego dla poprzedniego zadania, można obliczyć prędkości kątowe dla wszystkich 4 kół robota, które pozwolą mu poruszać się w taki sposób.

```
clear; close; clc;

% Parametry pojazdu: kąty rolek, promień koła.
alpha_1 = 45 * pi/180;
alpha_2 = -alpha_1;
R = 0.1;

% Zadane prędkości.
v_star = [0, 1, 0]';
omega_star = 0.5;

% Wektory p, określające położenie kół względem układu współrzędnych B.
p_1 = [1, 0.5, 0];
p_2 = [1, -0.5, 0];
p_3 = [-1, -0.5, 0];
p_4 = [-1, 0.5, 0];

% Wyznaczacznie pożądanych prędkości liniowych dla każdego koła.
v_1 = v_star + cross(omega_star * [0, 0, 1]', p_1);
v_2 = v_star + cross(omega_star * [0, 0, 1]', p_2);
v_3 = v_star + cross(omega_star * [0, 0, 1]', p_3);
v_4 = v_star + cross(omega_star * [0, 0, 1]', p_4);

% Wyznaczenia prędkości obrotowych dla każdego koła.
v_1_w = (v_1(1) - v_1(2) * cot(alpha_2)) / R
v_2_w = (v_2(1) - v_2(2) * cot(alpha_1)) / R
v_3_w = (v_3(1) - v_3(2) * cot(alpha_2)) / R
v_4_w = (v_4(1) - v_4(2) * cot(alpha_1)) / R
```

Założono, że platforma pojazdu to prostokąt o wymiarach: 1m x 0.5m, a korzystając z tego założenia wyznaczono wektory $\mathbf{p}$ określające położenie kół. Założono też pewien promień kół

pojazdu: 0.1m. Ostateczna odpowiedź to:

$$\omega_1 = 7.5\,\tfrac{\text{rad}}{\text{s}}, \quad \omega_2 = -10\,\tfrac{\text{rad}}{\text{s}}, \quad \omega_3 = 12.5\,\tfrac{\text{rad}}{\text{s}}, \quad \omega_4 = -10\,\tfrac{\text{rad}}{\text{s}},$$

zad. 16

a)

By czterowirnikowiec znajdował się w punkcie równowagi pionowej, jego całkowita siła ciągu musi być równa sile ciążenia, zatem musi zachodzić:

$$T = -mg$$

gdzie: T – całkowita siła ciągu, m – masa quadrotora, g – przyspieszenie ziemskie.
Równanie macierzowe pozwalające wyznaczyć prędkości śmigieł to:

$$\begin{bmatrix} \varpi_1^2 \\ \varpi_2^2 \\ \varpi_3^2 \\ \varpi_4^2 \end{bmatrix} = \mathbf{A}^{-1} \begin{bmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix}$$

W omawianym przypadku, momenty τ_x, τ_y, τ_z są równe zero. Macierz $\mathbf{A}$ to:

$$\mathbf{A} = \begin{bmatrix} -b & -b & -b & -b \\ 0 & -db & 0 & db \\ db & 0 & -db & 0 \\ k & -k & k & -k \end{bmatrix}$$

Ostatecznie rozwiązaniem są prędkości:

$$\varpi_i = \sqrt{\frac{mg}{4b}} \qquad i \in \{1, 2, 3, 4\}$$

Współczynnik b to wartość stała, określająca zależność siły ciągu od prędkości obrotowej śmigła.

b)

W systemie sterowania znajdują się w sumie 4 regulatory PD:

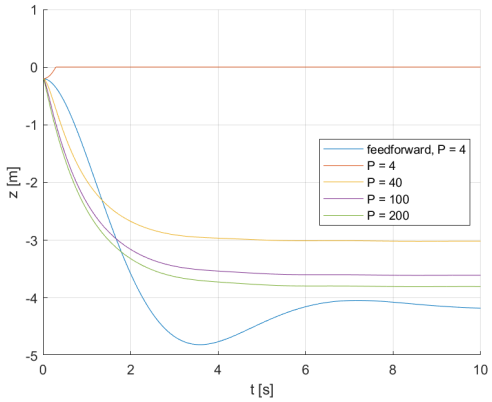
- wyznaczanie całkowitej siły ciągu dla zadanej wysokości (*Height control*),
- wyznaczanie momentu względem osi z dla zadanego kąta yaw (*Yaw control*),
- wyznaczanie kątów pitch i roll dla pożądanego wektora prędkości (*Velocity control*),
- wyznaczanie momentów względem osi x i y dla zadanych kątów pitch i roll (*Attitude control*).

Jeśli są 4 regulatory PD, to znaczy że w sumie potrzeba dobrać 8 nastaw. Wektor stanu quadrotora zawiera 12 składowych. Trudno jest przeprowadzić kompletną analizę, która wskaże, jaka poszczególne nastawy wpłyną jak na wszystkie składowe.. Zaobserwowano, że:

- dla regulatora *Height control*: Zmniejszanie wzm. P powodowało większe oscylacje wysokości pojazdu i dłuższe ustalanie się wartości zadanej, natomiast zwiększenie jej poprawiło jakość regulacji – przeregulowanie znacznie się zmniejszyło. Zwiększanie wzm. D także poprawiło jakość regulacji poprawiając dynamikę sygnału – krócej trwała faza przejściowa.
- dla regulatora *Yaw control*: Zmiany wzm. P prawie w ogóle nie wpłyneły podążanie za zadanym kątem yaw, który narastał liniowo. Zmiany D miały dużo większe znaczenie – im większe wzmocnienie D tym kąt yaw zmieniał się z mniejszą prędkością, czyli powodowało to gorsze nadążanie za sygnałem zadanym. Wzmocnienie to nie moze być jednak zerowe, bo wtedy kąt yaw obiektu silnie oscyluje względem sygnału zadanego.
- dla regulatora *Velocity control*: Zwiększanie wzm. P powodowało, że kąty pitch i roll obiektu zmieniały się ze znacznie większą częstotliwością. Zmiany D miały jednak większy wpływ na zmiany tych kątów – zmniejszenie tego wzm. powodowało, że wirnikowiec zaczynał wykonywać trajektorię koła o coraz większym promieniu, zaczynał uciekać z zadanej trajektorii.
- dla regulatora *Attitude control*: Zmiany wzm. P nie powodowały zmian żadnej ze składowych wektora stanu (być może wynikało to z zadanych parametrów). Zmiany D wpływały natomiast na wiele składowych stanu drona, co – upraszczając – powodowało, że nie zawsze poprawnie śledził zadaną trajektorię, szczególnie gdy badano większe (względem domyślnego) wzmocnienia.

c)

Wpływ zmian wzmocnienia P, przy usunięciu składnika sprzężenia w przód, na wysokość quadrotora widać na rysunku 34.



Rysunek 34: Wpływ P na osiąganą wysokość

Gdy usunięto składnik $T_0 \approx mg$, który zapewnia pionową równowagę, w stanie ustalonym wysokość nie osiągnęła wartości zadanej. Zwiększanie wzmocnienia zmniejsza ten błąd, ale i tak cały czas występuje.

d)

Całkowita siła ciągu wyznaczana przez regulator wysokości to: $\mathbf{T} = [0\,0\,T]^T$. Regulator wyznacza siłę ciągu na podstawie kąta yaw oraz jego pochodnej, natomiast nie bierze pod uwagę kątów pitch i roll. Kąty te wpływają na orientację wirnikowca, przez co wektor $\mathbf{T}$ nie jest skierowany prostopadle do podłoża, zgodnie z równaniem macierzowym:

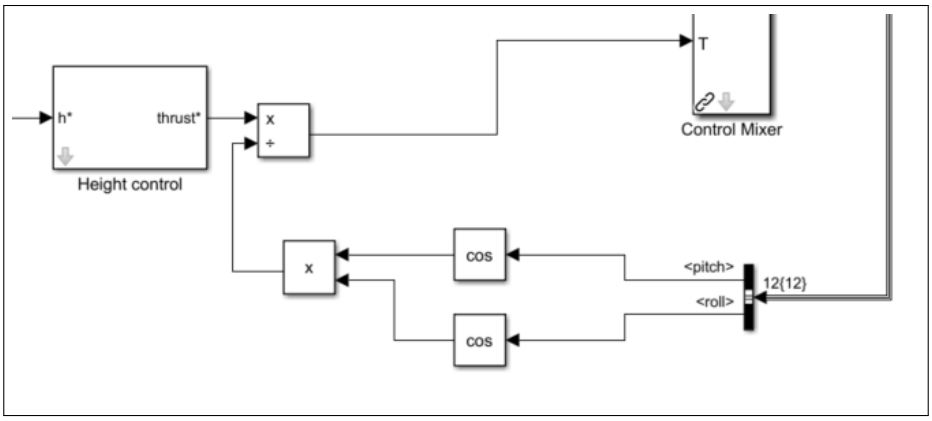
$$\mathbf{R}_{\mathrm{x,y,z}}(\phi, \theta, \psi) \cdot \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}$$

gdzie: ϕ – pitch, θ – roll, ψ – yaw.

Jeśli kąty pitch lub roll są niezerowe, wtedy $T \neq T_z$. By jednak wymusić tę równość we wspomnianych przypadkach, można odwrócić macierz rotacji $\mathbf{R}$, a następnie znaleźć rząd odpowiedzialny za T_z. Ostatecznie, by osiągnąć zamierzony cel, wyznaczoną przez regulator siłę ciągu T przepuszczono przez wyrażenie:

$$T^* = T \cdot (\cos(\phi) \cdot \cos(\theta))^{-1}$$

Omówioną modyfikację w Simulinku widać na schemacie poniżej.



Rysunek 35: Modyfikacja schematu

e)

Z symulacją lotu wirnikowca do góry nogami wystąpiły duże problemy. Wypróbowano wiele pomysłów:

– wymuszano zadany kąt 180 stopni na jednym z kątów orientacji pojazdu,

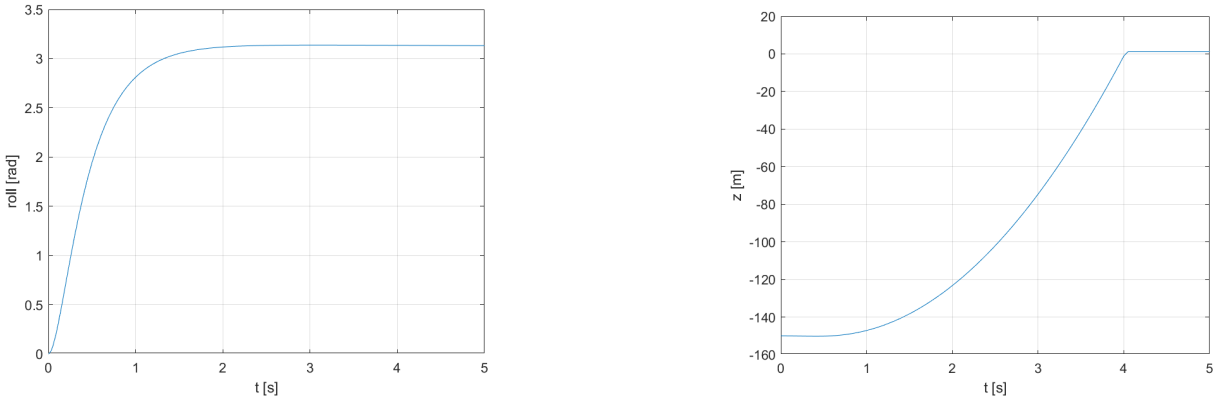
– starano się sterować pojazdem poprzez zadawanie odpowiednich momentów τ_x, τ_y, τ_z,

– wyłączano regulator wysokości i zadawano stałą siłę ciągu,

– umożliwiono śmigłom obrót w drugą stronę,

– zmieniano wzmocnienia w regulatorach (tych, których nie wyłączono).

Niestety, nie udało się uzyskać zamierzonych celów w zadowalającym stopniu. Być może pomysły były niewłaściwe, a być może symulator nie był przygotowany do tego zadania. Najlepszy rezultat widoczny jest na rysunku 36.



(a) (b)

Rysunek 36: Próby lotów do góry nogami

Jak widać obrót na drugą stronę udało się uzyskać, ale później pomimo wielu prób nie udało się opanować spadającego pojazdu.

f)

Ruch balistyczny zaprogramowano w następujący sposób:

– rozwarto sprzężenie zwrotne – sterowanie w układzie otwartym,

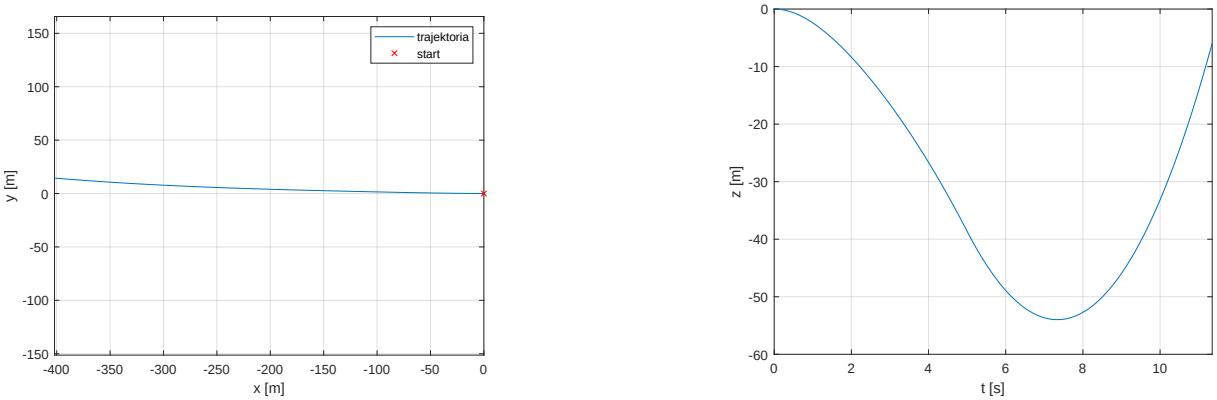
– zadane kąty orientacji pojazdu ustawiono jako wartości stałe:

$$\text{pitch} = \text{yaw} = 0^\circ, \qquad \text{roll} = 45^\circ,$$

– całkowitą siłę ciągu na początku zadano jako pewną dużą wartość: po pierwsze, musi być ona większa od siły ciążenia, a po drugie musi być wystarczająco duża, by podczas zmiany nachylenia pojazdu względem ziemi rzut wektora siły ciągu na płaszczyznę xy układu związanego z powierzchnią wciąż był większy od siły ciążenia. Następnie, w pewnej chwili – wybrano 5 sekund – wyłączono napędy.

Rezultat widać na rysunku 37.

Co ciekawe, wirnikowiec nie porusza się jedynie względem osi x – w bardzo niewielkim stopniu zmienia się także składowa y jego położenia. Wysokość zmienia się zgodnie z oczekiwaniami – w okolicach 5 sekundy wyłączony zostaje napęd, prędkość pojazdu względem ziemi zaczyna maleć, aż w końcu staje się ujemna i wirnikowiec spada.



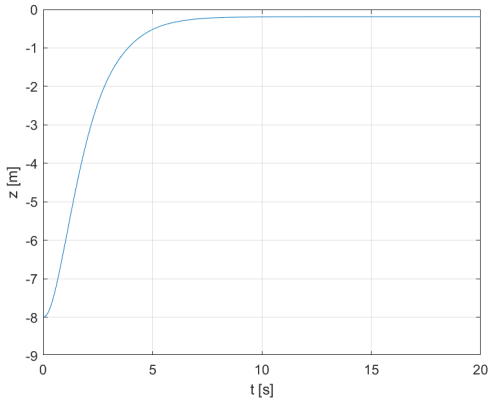
(a) płaszczyzna XY

(b) wysokość

Rysunek 37: Ruch balistyczny

g)

By zrealizować gładkie lądowanie, należy poprawnie dobrać nastawy regulatora PD wyznaczającego siłę ciągu dla osiągnięcia zadanej wysokości – nie może być przeregulowań.



Rysunek 38: Gładkie lądowanie

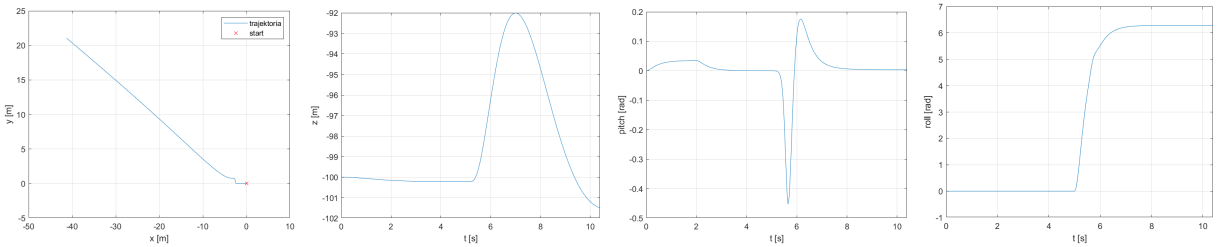
W stanie ustalonym jest minimalny uchyb – jest to konsekwencja zastosowania regulatora PD bez członu I. By go zmniejszyć można dodać wspomniany człon, lub zmienić wartość składnika sprzężenia w przód, które ustala stały poziom siły ciągu równoważącej siłę ciążenia (ta domyślna jest wpisana nieco niedokładnie).

h)

Manewr nazywany „beczką” czy „barrel roll” wykonano, zgodnie z instrukcją, w następujący sposób:

- pojazd, który znajduje się w stanie równowagi pionowej, lekko przechylono (kąt pitch), a następnie ustawiono ten kąt z powrotem na 0. W ten sposób pojazd zyskuje pewną wolno gasnącą prędkość w kierunku osi x,
- następnie, po kilku sekundach, gwałtownie zmieniono kąt roll z 0 na $360°$.

Efekty tych eksperymentów widać na rysunkach poniżej.



(a) trajektoria (b) wysokość (c) pitch (d) roll

Rysunek 39: Barrel roll

Wirnikowiec po lekkim przechyleniu (kąt pitch) uzyskał pewną prędkość v_x. Później dokonano gwałtownej zmiany kątu roll. Zmiana ta wpływa znacząco na wysokość pojazdu, który spada, ale później odzyskuje wysokość. Spowodowało to także zmiany trajektorii lotu. Być może eksperymenty były prowadzone przy zbyt niskich prędkościach, przez co niesymetryczne ułożenie pojazdu w trakcie „beczki" spowodowało rozłożenie prędkości liniowej na dwie składowe. Próbowano zmieniać prędkości, a także parametry regulatorów, ale nie do końca udało się całkowicie zapobiec temu zjawisku.

i)

To zadanie jest analogiczne co do poprzedniego, z jedną drobną różnicą:

- pojazd, który znajduje się w stanie równowagi pionowej, lekko przechylono (kąt pitch), a następnie ustawiono ten kąt z powrotem na 0. W ten sposób pojazd zyskuje pewną wolno gasnącą prędkość w kierunku osi x,
- następnie, po kilku sekundach, gwałtownie zmieniono kąt *pitch* z 0 na $360°$.

Niestety, symulator nie jest przygotowany do tych eksperymentów – kąt pitch może zmieniać się tylko od $-180°$ do $180°$, przez co wprowadzona zmiana powoduje niekończące się obracanie.

j)

Do zmiany liczy wirników trzeba wykorzystać inny blok opisujący dynamikę pojazdu oraz inny blok wyznaczający prędkości wirnków dla pożądanych momentów. W MATLABie użyto polecenia `roblocks`, co otworzyło w Simulinku panel z różnymi blokami związanymi z toolboxem. Kliknięto w zakładkę *Vehciles*, a następnie wybrano: *N-Rotor* oraz *Control Mixer N*. Te bloki umieszczono dokładnie w tych samych miejscach, gdzie były podobne bloki, które dotyczyły quadrotora.

Wspomniane bloki trzeba zparametryzować poprzez wprowadzenie liczby wirników (w naszym przypadku 8). Także model wirnikowca, wprowadzony do MATLABa poprzez komendę `mdl_quadrotor`, należy zmodyfikować zmieniając wartość pola `nrotors` na 8.

Teoretycznie te zmiany powinny wystarczyć do przeprowadzenia pierwszych testów, ale jak się okazało należy także wprowadzić inne zmiany:

– w bloku *N-Rotor* wybrano *Dynamics* i zmieniono nazwę parametru `ground` na `groundflag`,

– należy też zmodyfikować kod bloku symulującego dynamikę, co uzyskano klikając w *Edit* – w linijce 131 dodano zmienną globalną `globalFlag`. Linijka po zmianach powinna wyglądać tak:

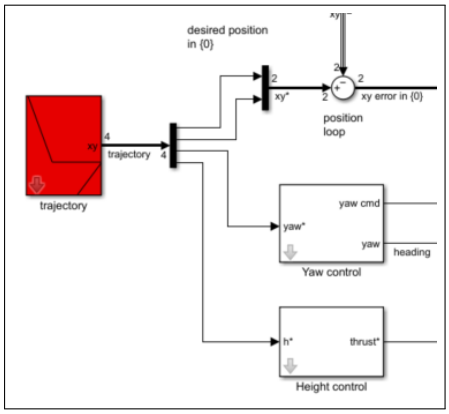
```
131     global a1s b1s groundFlag
```

Teraz program powinien działać poprawnie, choć by wirnikowiec się odpowiednio poruszał, należy zmienić wzmocnienia regulatorów.

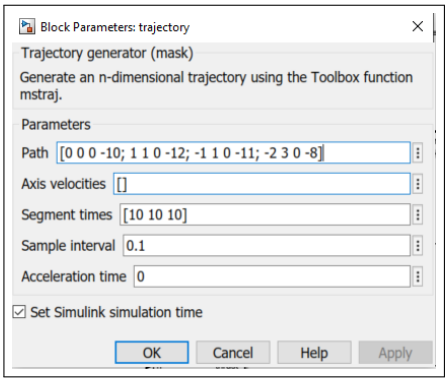
k)

By zrealizować zadanie podążania wzdłuż trajektorii można posłużyć się blokiem *trajectory*, który wykorzystywany był w innej symulacji Simulinka (`sl_pursuit`).

Na rysunkach poniżej przedstawiony został sposób dodania bloku do schematu oraz parametry bloku.



(a) schemat



(b) parametry

Rysunek 40: Blok *trajectory*

W parametryzacji istotne są dwa pola:

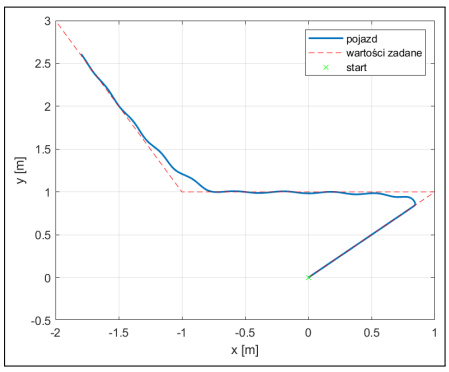
– *Path* – macierz o 4 kolumnach (x, y, z, θ) określających punkty pośrednie trajektorii,

– *Segment Times* – czas dla każdej części trajektorii.

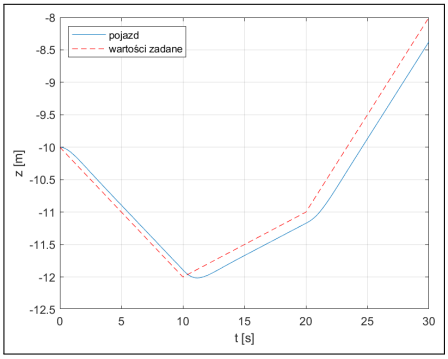
Punkty łączone są ze sobą liniami prostymi.

Następnie należy zmienić wzmocnienia regulatorów w systemie sterowania, tak by lepiej radziły sobie z nadążaniem. Domyślne wzmocnienia co prawda zapewniają płynne przejście z jednego miejsca na drugie, ale przy nadążaniu można zezwolić na większe przeregulowania, co umożliwi uzyskanie lepszych wskaźników czasowych.

Na rysunkach 41 widać efekty pracy.



(a) płaszczyzna XY



(b) wysokość

Rysunek 41: Podążanie za trajektorią