Praca przejściowa inżynierska

Sterowanie dronem z wykorzystaniem metody LQR

Wydział Mechaniczny Energetyki i Lotnictwa

Politechnika Warszawska

Michał Siniarski

304481

(strona pusta)

Spis treści

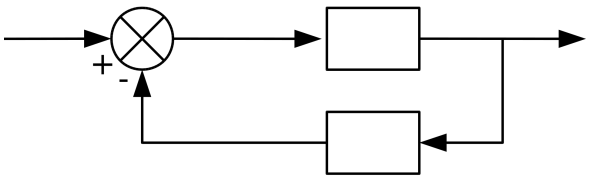
1 Wstęp

Dążenie do jak największej efektywności sterowania, przyspieszenie działania układu, a jednocześnie optymalizacja zużycia energii, prowadzi do ograniczenia udziału człowieka w sterowaniu nim. Taki zabieg ujednolica również osiągalne wyniki, czyniąc je przewidywalnymi. Jak realizuje się opisane powyżej sterowanie? Najpopularniejszym (i opisanym w dalszych podpunktach) sposobem jest zastosowanie układu sterowania.

1.1 Układ sterowania ze sprzężeniem zwrotnym

Możemy wyróżnić dwa podstawowe układy:

- Układ sterowania otwartego - jego cechą charakterystyczną jest brak informacji o wartości sygnału wyjściowego i gwarancji uzyskania odpowiedniej dokładności sterowania, czy nawet braku osiągnięcia zadanego celu. Tak mała dokładność powiązania jest z pojawianiem się zakłóceń
- Układ sterowania zamkniętego - żeby skompensować wpływ powyższych zakłóceń mierzy się wartość wyjściową i porównuje z wartością zadaną. Wynik tego porównania wykorzystywany jest do wprowadzenia ewentualnej korekty sterowania. Taka pętla nosi nazwę sprzężenia zwrotnego w układzie zamkniętym.



Rysunek 1: Schemat układu sterowania ze sprzężeniem zwrotnym ujemnym

Takie układy znajdują zastosowanie na przykład w:

- Autonomicznych pojazdach, w celu utrzymania się na jednym pasie,
- Robotach utrzymujących się na zadanej ścieżce,
- Konstrukcjach wymagających utrzymania równowagi,
- Dronach utrzymujących zadaną wysokość, bądź poruszających się po ścieżce.

1.2 Metoda LQR

W układzie zamkniętym ważny jest odpowiedni dobór biegunów. Jak można się domyślić ręczny dobór nigdy nie zapewni optymalności sterowania. W tym celu opracowano regulator LQR - regulator kwadratowo liniowy, określający rozwiązanie dla układu dynamicznego opisanego serią liniowych równań różniczkowych. Dla układu liniowego:

$$\dot{x} = Ax + Bu \tag{1}$$

Wyznacza się funkcję kosztu:

$$J = \int_0^\infty (x'Qx + u'Ru)\, dt \tag{2}$$

Sterowanie ze sprzężeniem zwrotnym odbywa się poprzez modyfikację wektora sterującego:

$$u = -Kx, \tag{3}$$

Należy jednak znaznaczyć, że tak otrzymany wektor nie prowadzi układu do stanu zadanego, a jedynie do zerowego. Dlatego też algorytmy, w których sterowanie odbywa się w sposób bardziej złożony, wektor u znajduje się następująco:

$$u = -Ke, \tag{4}$$

gdzie e jest wektorem błędu, szerzej opisanym w kolejnych podpunktach.
Macierz K wyznacza się ze wzoru:

$$K = R^{-1}B^T P \tag{5}$$

W skład powyższego równania wchodzi macierz P, którą wyznacza się z równania Riccatiego w postaci:

$$A^T P + PA - (PB + N))\, R^{-1}B^T P + Q = 0 \tag{6}$$

Do rozwiązania powyższego równania, to jest znalezienia macierzy P, wykorzystuje się funkcje *lqr*, dostępne w ogólnodostępnych bibliotekach w środowiskach *Python* lub *Matlab* Do realizacji opisywanego projektu, użyto kodu dostarczonego przez Promotora. Każdy ze wspomnianych solverów, jako dane wejściowe przyjmuje 4 macierze:

```
K, _, _ = lqr2(A, B, Q, R)
```

Macierze A i B to kolejno macierz stanu oraz macierz wejść układu. Dokładny algorytm znajdujący ich wartość, zostanie przedstawiony w dalszej części pracy.

Macierze Q i R to natomiast diagonalne macierze wag - odpowiadają za dobranie sterowania o odpowiedniej jakości, przy danym koszcie energetycznym.

Celem powyższego algorytmu jest minimalizacja kosztu J (dla zadanych Q i R), przy jednoczesnym doprowadzeniu układu do zadanego celu (do stanu najbliższego temu celowi). Owa minimalizacja odbywa się w sposób analityczny - nie jest potrzebna wartość całki J, a jedynie argumenty dla których osiąga minimum.

1.3 Cel pracy

Na podstawie powyższego przeglądu rozwiązań i opisu algorytmu można łatwo objaśnić cel poniższej pracy. Jest nim wykorzystanie algorytmu LQR do sterowania dronem na zadanej ścieżce.

2 Budowa rozwiązania

Do realizacji zadania wykorzystano język programistyczny *Python*. Taki dobór jest uzasadniony przede wszystkim dostępnością środowiska i licznymi bilbiotekami przyspieszającymi pracę w nim. Umożliwia to stosunkowo szybkie osiągnięcie zadowalających efektów.

Znaczna większość kodu wykonywana jest w pętli *while* ze skokiem *dt*:

```
1 while t < t_end + dt:
2     do_something1()
3     do_something2()
```

Umożliwia to sterowanie rozdzielczością obliczeń, a także zakresem ich wykonywania. Poniżej opisane są główne funkcje wykonujące większość operacji, wywoływane w wyżej wymienionej pętli.

Macierze A i B deklarowane są jako:

```
def Jacob_AB(RHS, y, t, u_control, n, m, az_turbulence, ax_wind, az_wind):
    A = declare_matrix(n, n)
    B = declare_matrix(n, m)
    dy = 1.0e-6
    f0 = RHS(y, t, u_control, az_turbulence, ax_wind, az_wind)
    for i in range(0, n):
        yp = array(y)
        yp[i] += dy
        f = RHS(yp, t, u_control, az_turbulence, ax_wind, az_wind)
        for j in range(0, n):
            A[j, i] = (f[j] - f0[j]) / dy

    for i in range(0, m):
        up = array(u_control)
        up[i] += dy
        f = RHS(y, t, up, az_turbulence, ax_wind, az_wind)
        for j in range(0, n):
            B[j, i] = (f[j] - f0[j]) / dy
    return A, B

    A, B = Jacob_AB(RHS, y, t, u_control, n, m, az_turbulence, ax_wind, az_wind)
```

Powyższy algorytm dla poszczególnych elementów obu macierzy oblicza wartość pochodnej:

$$A\,[i,j] = \left.\frac{\partial f_i}{\partial x_j}\right|_{x_j^*} \qquad \text{dla wymiarów (n, n),} \tag{7}$$

oraz:

$$B\,[i,j] = \left.\frac{\partial f_i}{\partial x_j}\right|_{x_j^*} \qquad \text{dla wymiarów (n, m)} \tag{8}$$

Mecierze Q i R otrzymuje się za pomocą kodu:

```
def getQR(n, m):
    Q = np.diag(np.diag(np.ones((n, n))))
    R = np.diag(np.diag(np.ones((m, m))))

    Q[0, 0] = 100.
    Q[1, 1] = 100.
    Q[2, 2] = 0.1
    Q[3, 3] = 10.
    Q[4, 4] = 100.
    Q[5, 5] = 0.001

    if n == 8:
        Q = 10000 * Q

    return Q, R

Q, R = getQR(n, m)
```

Funkcja powyżej zwraca dwie macierze diagonalne. Wartości diagonali dla Q są wartościami podanymi przez Promotora (ich wpływ na wyniki zostanie omówiony później) Wartości diagonali dla R to jedynki - nie mają one większego znaczenia i w tej pracy pozostają stałe.

Aby symulacja ruchu drona mogła się odbyć potrzebny jest jego model fizyczny. Ten został opisany za pomocą algorytmu:

```
def RHS(x, t, u_control, az_turbulence, az_wind, ax_wind):
    n = len(x)
    dx_dt = np.zeros((n, 1))

    g = 9.81
    S = 1.

    mass = 25.
    Iy = 100.

    vx = x[0] + ax_wind
    vz = x[1] + az_wind

    alpha = np.arctan2(vz, vx)
    V = np.sqrt(vz * vz + vx * vx)

    CD_0 = 0.30
    CD = CD_0

    rho_0 = 1.225
    rho = rho_0 * math.pow((1.0 - math.fabs(x[4]) / 44300.0), 4.256)

    Q_dyn = 0.5 * rho * V * V
    L = 0.
    D = Q_dyn * S * CD
    G = mass * g
    Th = 1.

    Thrust_1 = 0.5 * G + u_control[0]
    Thrust_2 = 0.5 * G + u_control[1]

    if n == 8:
        Thrust_1 = x[6]
        Thrust_2 = x[7]
    cm_q = -0.01

    Tau = 0.05

    beta = 0. * deg2rad
```

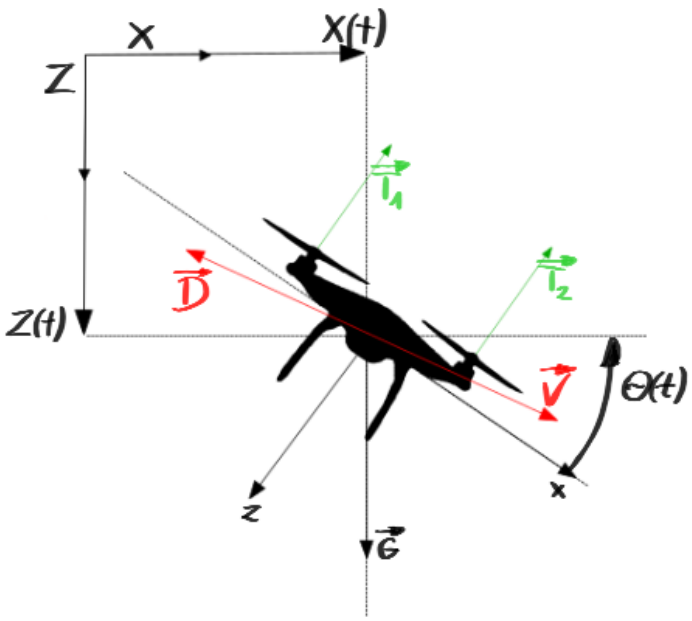
```
    cb = np.cos(beta)
    sb = np.sin(beta)
    dx_dt[0] = (-D * math.cos(alpha) + L * math.sin(alpha) - G * math.sin(
        x[5]) - Thrust_1 * sb + Thrust_2 * sb) / mass - x[
                    2] * vz
    dx_dt[1] = (-D * math.sin(alpha) - L * math.cos(alpha) + G * math.cos(
        x[5]) - Thrust_1 * cb - Thrust_2 * cb) / mass + x[
                    2] * vx + az_turbulence
    dx_dt[2] = (0.5 * (Thrust_2 * cb - Thrust_1 * cb) + cm_q * x[2]) / Iy
    dx_dt[3] = np.cos(x[5]) * vx + np.sin(x[5]) * vz
    dx_dt[4] = -np.sin(x[5]) * vx + np.cos(x[5]) * vz
    dx_dt[5] = x[2]

    if n == 8:
        dx_dt[6] = (1.0 / Tau) * (-x[6] + Th * u_control[0])
        dx_dt[7] = (1.0 / Tau) * (-x[7] + Th * u_control[1])
    return dx_dt
```

Powyższy kod zakłada:

- masę drona $m = 25kg$
- powierzchnię rzutu drona na płaszczyznę prostopadłą do wektora prędkości S=1
- moment bezwładności wzdłuż osi y $I_y = 100kg \cdot m^2$

Model fizyczny obiektu przedstawia poniższa grafika:



Rysunek 2: Model fizyczny drona

Warto zwrócić uwagę na pojawienie się dwóch układów współrzędnych - jeden z nich jest stały, drugi ruchomy i związany z dronem, natomiast oba są odwrócone. Dron jest odchylony względem poziomu o kąt θ.

Prędkości względem powietrza, definiuje się jako sumę prędkości zapisanej w wektorze stanu i prędkości wiatru w danej osi. Następnie znajduje się wartość wypadkową składowych wzdłuż dwóch obu osi.

W czasie ruchu na drona działają na niego następujące siły:

- grawitacja - $G = m \cdot g$
- opór aredodynamiczny - $D = 0.5 \cdot \rho \cdot V^2 \cdot S \cdot CD$, gdzie:
  - rho - gęstość powietrza uwzględniająca zmiany związane ze zmianami wysokości - zgodnie z atmosferą wzorcową,
  - V - prędkość wypadkowa drona,

  - CD - bezwymiarowy współczynnik oporu aerodynamicznego.

- ciąg - dron posiada dwa silniki. Każdy z nich produkuje ciąg potrzebny do wyrównania połowy ciężaru obiektu oraz dodatkową wartość potrzebną do sterowania.

Tutaj też nasuwa się wniosek co w rzeczywistości pełni funkcję sterowania dronem - są to wartości ciągu.
W tym miejscu warto również wspomnieć o wektorach, którymi u ma sterować. Wektor stanu wygląda następująco:

$$x = \begin{bmatrix} v_x & v_z & 0 & x & z & \theta & 0 & Thrust_1 & Thrust_2 \end{bmatrix} \tag{9}$$

Wektor e natomiast, nazywany również wektorem błędu jest zdefiniowany jako:

$$e = \begin{bmatrix} x_0 - (cos(\theta) \cdot v_x + sin(\theta) \cdot v_z) \\ x_1 - (sin(\theta) \cdot v_x - cos(\theta) \cdot v_z) \\ x_2 \\ 0 \\ x_4 - (-z_{ref}) \\ x_5 \end{bmatrix} \tag{10}$$

Kluczowym dla całej pracy algorytmem jest funkcja *lqr2* zmieszczona poniżej:

```
def lqr2(A,B,Q,R):

#ref Bertsekas, p.151

#first, try to solve the ricatti equation
 X = np.matrix(scipy.linalg.solve_continuous_are(A, B, Q, R))

#compute the LQR gain
 K = np.matrix(scipy.linalg.inv(R)*(B.T*X))

 eigVals, eigVecs = scipy.linalg.eig(A-B*K)

 return K, X, eigVals


 K, _, _ = lqr2(A, B, Q, R)
```

Funkcja rozwiązuje równanie Ricattiego (wyznacza wartość dla której całka J jest najmniejsza), po czym znajduje K. Program pobiera z funkcji wyłącznie wartość tego wektora.

Do obliczeń potrzebny jest również algorytm całkujący. W pracy wykorzystano algorytm rk4.

Trasa wzorcowa, do której dron powinien dążyć została opisana następująco:

```
def trajectory(X, Vel, dt):

    Z = 1.
    if X <= 0.5:
        Z = 1.
    elif 0.5 < X < 1.:
        Z = 6. * X - 2.
    elif 1. <= X <= 1.5:
        Z = 4.
    elif 1.5 <= X <= 2.:
        Z = 6. * X - 5.
    elif 2. <= X <= 2.5:
        Z = 2. * X + 3.
    elif 2.5 <= X <= 3.:
        Z = -6 * X + 23
    elif 3. <= X <= 4.:
        Z = -4. * X + 17.
    else:
        Z = 1.

    Z1 = Z

    dx = Vel * dt
    X = X + dx

    if X <= 0.5:
        Z = 1.
    elif 0.5 < X < 1.:
        Z = 6. * X - 2.
    elif 1. <= X <= 1.5:
        Z = 4.
    elif 1.5 <= X <= 2.:
        Z = 6. * X - 5.
    elif 2. <= X <= 2.5:
        Z = 2. * X + 3.
    elif 2.5 <= X <= 3.:
        Z = -6 * X + 23
    elif 3. <= X <= 4.:
        Z = -4. * X + 17.
    else:
```

```
41          Z = 1.
42
43      alpha = math.atan2(Z - Z1, dx)
44
45      return Z, alpha
```

Zgodnie z powyższym algorytmem dla współrzędnych:

- $X \leqslant 0.5$ - współrzędna Z jest równa 1.
- $0.5 \leqslant X \leqslant 1$ - Z jest opisane za pomocą krzywej $6 \cdot X - 2$
- $1 \leqslant X \leqslant 1.5$ - $Z = 4$
- $1.5 \leqslant X \leqslant 2$ - $Z(X) = 6 \cdot X - 5$
- $2 \leqslant X \leqslant 2.5$ - $Z(X) = 2 \cdot X + 3$
- $2.5 \leqslant X \leqslant 3$ - $Z(X) = -6 \cdot X + 23$
- $3 \leqslant X \leqslant 4.$ - $Z(X) = -4 \cdot X + 17$

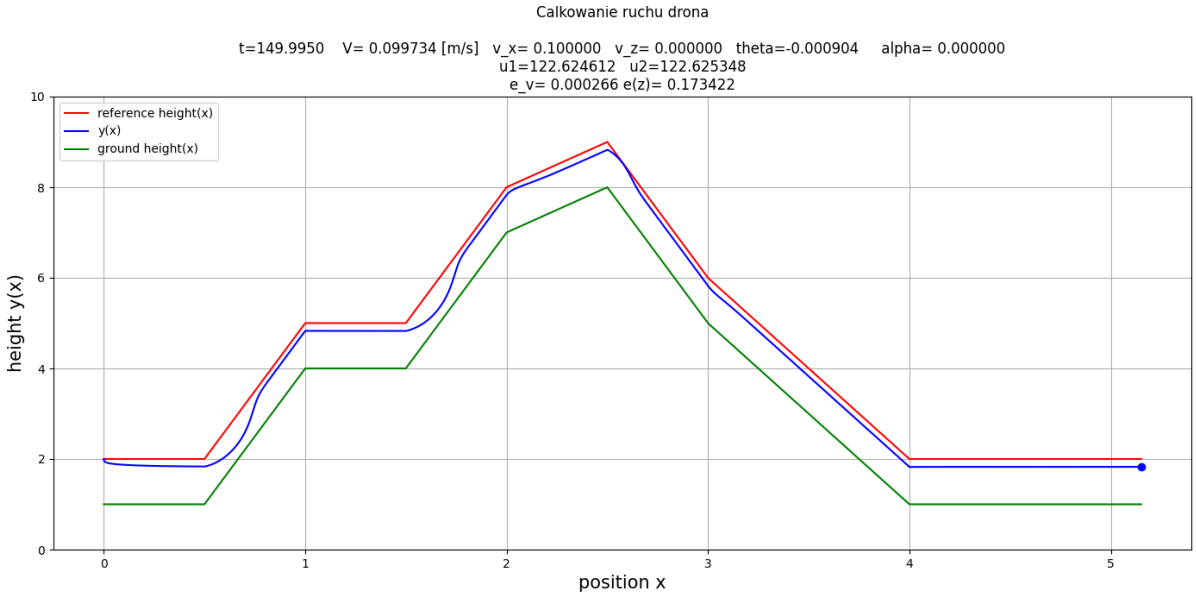
3 Założenia przyjęte do obliczeń

Obliczeń dokonano dla następnujących parametrów początkowych:

- $Vel = 0.1 \left[\frac{m}{s}\right]$ - prędkość ruchu,
- $t = 0.0[s]$ - początek ruchu,
- $t_{end} = 150[s]$ - koniec ruchu,
- $dt = 0.005[s]$ - rozdzielczość całkowania, krok czasowy

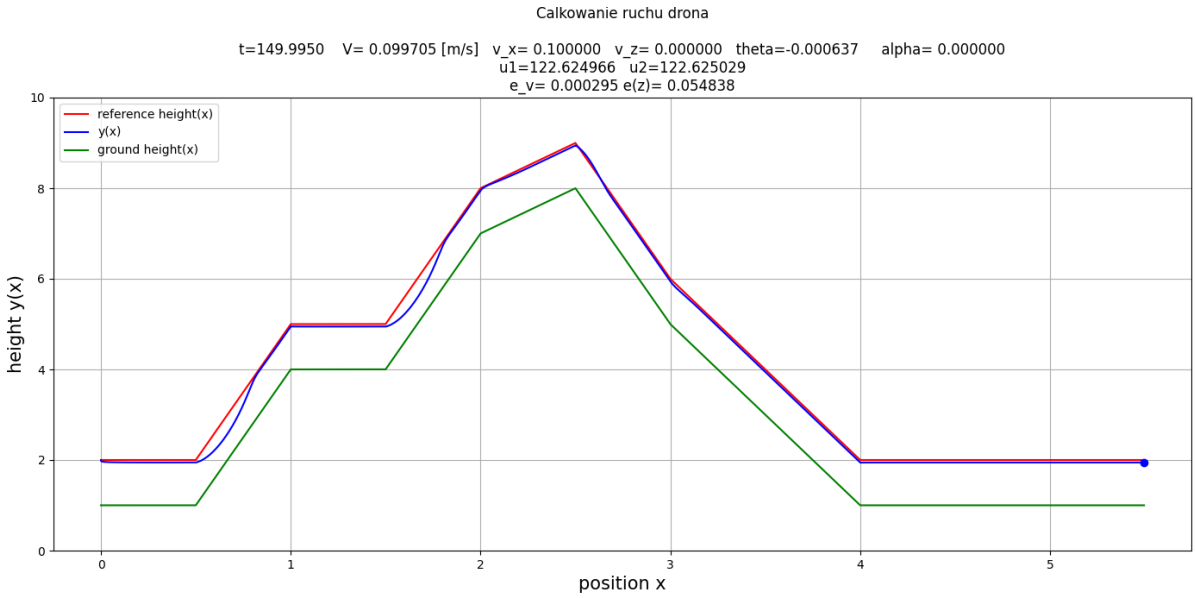
4 Przegląd wyników

Poniżej przedstawione są osiągalne wyniki i odpowiadające im macierze Q. Na podstawie takiego zestawienia można dobrać sterowanie tak, aby ruch drona jak najlepiej odzwierciadlał zadaną ścieżkę, przy jednoczesnej minimalizacj zużycia energii. Zadaniem poniższego zbioru wyników jest pokazanie sterowań o różnym charakterze - od zbyt łagodnego, przez prawidłowe, po zbyt mocne i ich powiązania z kosztem energetycznym.



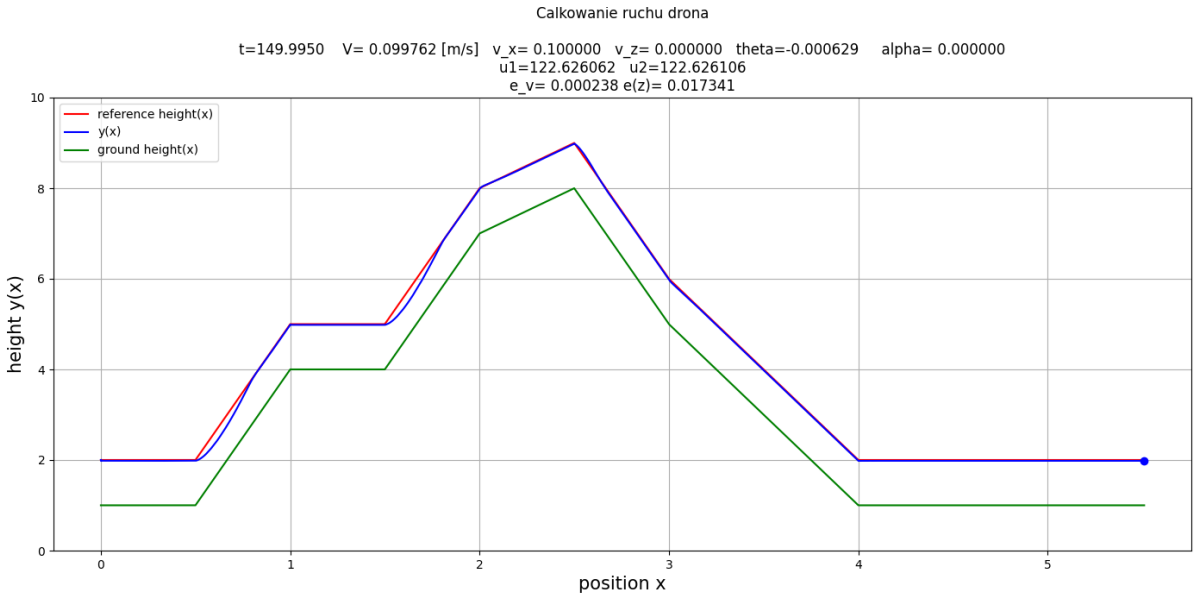
Rysunek 3: Trajektoria otrzymana dla macierzy Q_1

$$Q_1 = \begin{bmatrix} 1000000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1000000 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 100000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1000000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10000 \end{bmatrix} \tag{11}$$



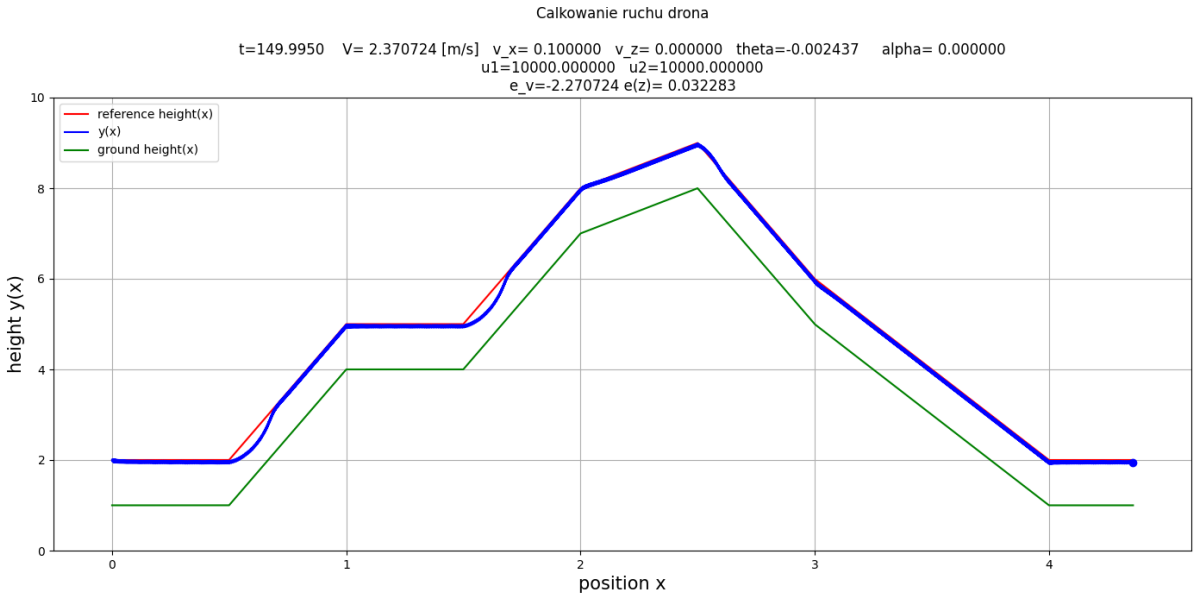
Rysunek 4: Trajektoria otrzymana dla macierzy Q_2

$$Q_2 = \begin{bmatrix} 1000000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1000000 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10000000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10000 \end{bmatrix} \tag{12}$$



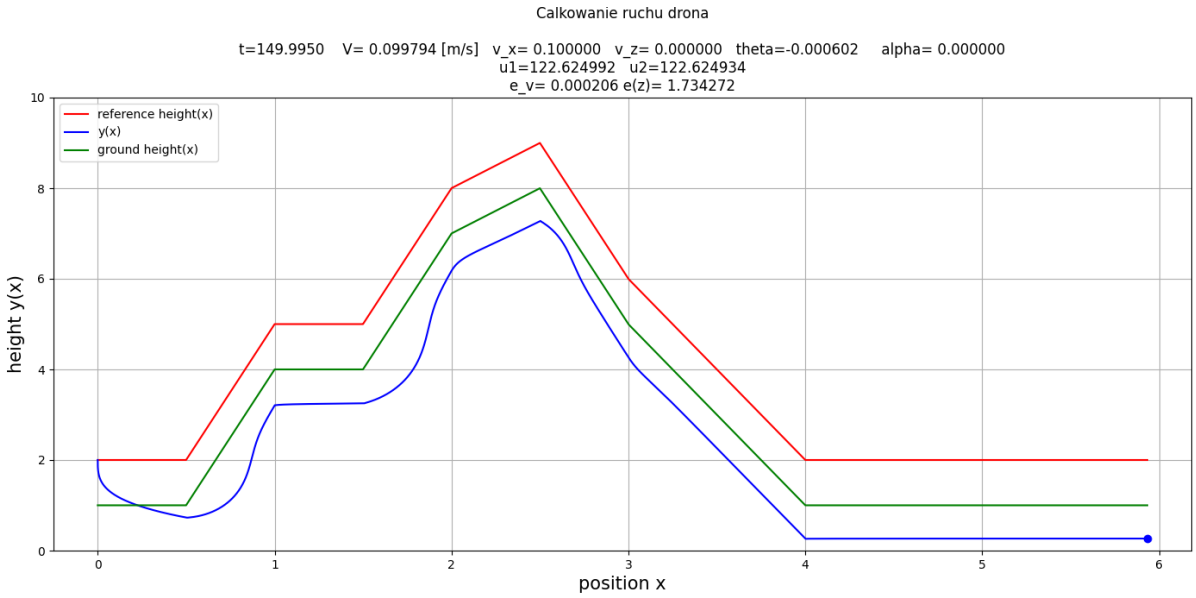
Rysunek 5: Trajektoria otrzymana dla macierzy Q_3

$$Q_3 = \begin{bmatrix} 1000000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1000000 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100000000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10000 \end{bmatrix} \tag{13}$$



Rysunek 6: Trajektoria otrzymana dla macierzy Q_4

$$Q_4 = \begin{bmatrix} 10000000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 100000000 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 100000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100000000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10000 \end{bmatrix} \tag{14}$$



Rysunek 7: Trajektoria otrzymana dla macierzy Q_5

$$Q_5 = \begin{bmatrix} 1000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1000 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10000 \end{bmatrix} \tag{15}$$

Powyższe wyniki można sklasyfikować jako:

- Q_1 (odpowiadające macierzy Q_1) - sterowanie "referencyjne", wyznaczone przez Promotora
- Q_2 - sterowanie prawidłowe,
- Q_3 - sterowanie prawidłowe, najdokładniejsze,
- Q_4 - sterowanie zbyt mocne,
- Q_5 - sterowanie zbyt słabe.

5 Wnioski

Otrzymane wyniki (zgodnie z oczekiwaniami) wskazują na powiązanie kosztów energetycznych z dokładnością sterowania. Dokładność rośnie wraz z powyższym kosztem, jednakże dzieje się tak tylko do pewnego momentu. Gdy wsrtości diagonali staną się zbyt duże, linia po której porusza się dron, prztestaje stale dążyć do referencyjnej.
Nieprawidłowe sterowanie dobrano również na wykresie 6. Koszty energetyczne są na tyle niskie, że dron nigdy nie osiąga zadanej wysokości i uderza w ziemię.

Powyższe rozważania pozwalają zrozumieć związek kosztu w macierzy Q oraz jakości działania sterowania, a także wskazują odpowiedni kierunek doboru tych kosztów. Należy również pamiętać, że kształt wyników byłby inny po zmianie parametrów początkowych. Nawet przy nieznacznie zwiększonej prędkości, wymagany byłby wzrost wartości diagonali.

6 Źródła

- https://automaticaddison.com/linear-quadratic-regulator-lqr-with-python-code-exampl
- https://pl.wikipedia.org/wiki/Regulator_liniowo-kwadratowy
- https://eia.pg.edu.pl/documents/184139/28396400/LQR.pdf
- https://yadda.icm.edu.pl/baztech/element/bwmeta1.element.baztech-09fb503c-9ef4-4cb3-c/document.pdf