МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний технічний університет

«Харківський політехнічний інститут»

Кафедра «Комп'ютерне моделювання процесів та систем»

з дисципліни «Програмування компонентів БПЛА»

**Лабораторна робота 1**

«Моделювання руху БПЛА»

Виконав студент групи ІКМ-220Б

Пономаренко Олексій Валерійович

Перевірив викладач

Успенський Валерій Борисович

Харків 2023

**Мета:** Cкласти програму для моделювання руху БПЛА за формулами зі статті. Дослідити, як впливає швидкість ротору ДВЗ на динаміку БПЛА.

Я написав программу для моделювання руху БПЛА, яка у кожний момент дискретного часу інтегрування обраховує значення координат, кутів орієнтації, проекцій кутової швидкості, кутової швидкості роторів двигунів системи управління та проекції кутової швидкості за наданими формулами.

*import numpy as np*

*import quaternion # noqa*

*from scipy.integrate import quad*

*import sympy*

*class BPLA:*

*e1 = np.array([1, 0, 0])*

*e2 = np.array([0, 1, 0])*

*e3 = np.array([0, 0, 1])*

*def \_\_init\_\_(*

*self, time,*

*I, m, dt, J, J0, Kf, Kf0, # noqa*

*Km, Km0, p, G, Ks, H\_, end\_pos,*

*Vx\_, Vymax, Vz\_, c,*

*state*

*):*

*self.time = time*

*self.t = 0.0*

*self.I = I # noqa*

*self.m = m*

*self.dt = dt*

*self.J = J*

*self.J0 = J0*

*self.Kf = Kf*

*self.Kf0 = Kf0*

*self.Km = Km*

*self.Km0 = Km0*

*self.p = p*

*self.G = G*

*self.Ks = Ks*

*self.H\_ = H\_*

*self.end\_pos = end\_pos*

*self.Vx\_ = Vx\_*

*self.Vymax = Vymax*

*self.Vz\_ = Vz\_*

*self.c = c*

*[*

*self.pos,*

*self.v,*

*self.psi,*

*self.teta,*

*self.gamma,*

*self.Om,*

*self.omega,*

*self.eps*

*] = state*

*self.current\_state = []*

*self.F = np.array([*

*[0.0, 0.0, 0.0],*

*[0.0, 0.0, 0.0],*

*[0.0, 0.0, 0.0],*

*[0.0, 0.0, 0.0],*

*[0.0, 0.0, 0.0],*

*[0.0, 0.0, 0.0]*

*])*

*self.Lam = np.quaternion(0.0, 0.0, 0.0, 0.0)*

*self.v\_zsk = np.array([0.0, 0.0, 0.0])*

*self.Fs\_zsk = np.array([0.0, 0.0, 0.0])*

*# Вектор локальної похідної кінетичного моменту двигунів*

*self.dH = np.array([0.0, 0.0, 0.0])*

*# Рівнодіючs силb тяги двигунів у проекціях на осі ІСК*

*self.Fe = np.array([0.0, 0.0, 0.0])*

*# Проекції сили опору повітря*

*self.Fs = np.array([0.0, 0.0, 0.0])*

*# Вектор гіроскопічного моменту в проекціях на осі ЗСК*

*self.Mg\_zsk = np.array([0.0, 0.0, 0.0])*

*# Cумарний момент сил тяги двигунів*

*self.MFe = np.array([0.0, 0.0, 0.0])*

*# Cумарний аеродинамічний момент від шести двигунів*

*self.MAe = np.array([0.0, 0.0, 0.0])*

*# повертає поточний стан*

*def getCurrentState(self):*

*return [*

*self.pos[0],*

*self.pos[1],*

*self.pos[2],*

*self.v[0],*

*self.v[1],*

*self.v[2],*

*self.psi,*

*self.teta,*

*self.gamma,*

*self.Om[0],*

*self.Om[1],*

*self.Om[2],*

*self.omega[0],*

*self.omega[1],*

*self.omega[2],*

*self.omega[3],*

*self.omega[4],*

*self.omega[5],*

*self.eps[0],*

*self.eps[1],*

*self.eps[2],*

*self.eps[3],*

*self.eps[4],*

*self.eps[5],*

*self.F[0][0],*

*self.F[0][1],*

*self.F[0][2],*

*self.F[1][0],*

*self.F[1][1],*

*self.F[1][2],*

*self.F[2][0],*

*self.F[2][1],*

*self.F[2][2],*

*self.F[3][0],*

*self.F[3][1],*

*self.F[3][2],*

*self.F[4][0],*

*self.F[4][1],*

*self.F[4][2],*

*self.F[5][0],*

*self.F[5][1],*

*self.F[5][2],*

*self.Lam.w,*

*self.Lam.x,*

*self.Lam.y,*

*self.Lam.z,*

*self.Fe[0],*

*self.Fe[1],*

*self.Fe[2],*

*self.v\_zsk[0],*

*self.v\_zsk[1],*

*self.v\_zsk[2],*

*self.Fs\_zsk[0],*

*self.Fs\_zsk[1],*

*self.Fs\_zsk[2],*

*self.Fs[0],*

*self.Fs[1],*

*self.Fs[2],*

*self.Mg\_zsk[0],*

*self.Mg\_zsk[1],*

*self.Mg\_zsk[2],*

*self.dH[0],*

*self.dH[1],*

*self.dH[2],*

*self.MFe[0],*

*self.MFe[1],*

*self.MFe[2],*

*self.MAe[0],*

*self.MAe[1],*

*self.MAe[2]*

*]*

*def launch(self):*

*np.seterr(all='raise')*

*states = []*

*ts = self.time*

*num\_steps = int(ts / self.dt)*

*for i in range(num\_steps):*

*state = self.getCurrentState()*

*if (i % (num\_steps / ts) == 0):*

*self.t = i / (num\_steps / ts)*

*print(f'{self.t} {self.pos} [{self.psi}, {self.teta}, {self.gamma}]') # noqa*

*self.calc\_forces()*

*# self.main\_controller()*

*self.derivative()*

*states.append(state)*

*self.states = states*

*def derivative(self):*

*pos = self.v*

*v = self.Fe / self.m + self.G + self.Fs / self.m*

*psi = (*

*self.Om[2] \* np.sin(self.gamma) - self.Om[1] \* np.cos(self.gamma)*

*) / np.cos(self.teta)*

*teta = self.Om[1] \* np.sin(*

*self.gamma*

*) + self.Om[2] \* np.cos(self.gamma)*

*gamma = self.Om[0] + np.tan(self.teta) \* (*

*self.Om[2] \* np.sin(self.gamma) - self.Om[1] \* np.cos(self.gamma)*

*)*

*Om = (-self.Mg\_zsk - self.dH + self.MFe + self.MAe) / self.I*

*omega = self.eps*

*self.pos = self.pos + pos \* self.dt*

*self.v = self.v + v \* self.dt*

*self.psi = self.psi + psi \* self.dt*

*self.teta = self.teta + teta \* self.dt*

*self.gamma = self.gamma + gamma \* self.dt*

*self.Om = self.Om + Om \* self.dt*

*self.omega = self.omega + omega \* self.dt*

*def main\_controller(self):*

*self.vertical\_controller()*

*self.values\_formation()*

*self.eps[0] = self.ay \* self.m / (2 \* self.c[0] \* self.omega[0])*

*def vertical\_controller(self):*

*vy\_dot = self.Fe[1] / self.m + self.G[1] + self.Fs[1] / self.m*

*if abs(self.H\_ - self.pos[1]) > 20:*

*b = 1*

*k = np.array([*

*-b \*\* 2,*

*-2 \* b*

*])*

*ay\_ = k[1] \* self.v[1] + k[0] \* (*

*vy\_dot - self.Vymax \* np.sign(self.H\_ - self.pos[1])*

*)*

*else:*

*b = 0.5*

*k = np.array([*

*-b \*\* 3,*

*-3 \* b \*\* 2,*

*-3 \* b*

*])*

*ay\_ = k[2] \* vy\_dot + k[1] \* self.v[1] + k[0] \* (*

*self.pos[1] - self.H\_*

*)*

*self.ay = ay\_*

*def angle\_controller(self):*

*b = 5*

*k = np.array([*

*-b \*\* 3,*

*-3 \* b \*\* 2,*

*-3 \* b*

*])*

*f = (*

*self.Om3 \* np.sin(self.gamma) - self.Om2 \* np.cos(self.gamma)*

*) / np.cos(self.teta)*

*df = sympy.diff(f, x)*

*def values\_formation(self):*

*dx, \_, dz = self.end\_pos - self.pos*

*dx\_plus\_dz = (dx \*\* 2 + dz \*\* 2) \*\* 0.5*

*if dx\_plus\_dz < 10:*

*psi\_ = self.psi*

*else:*

*psi\_ = np.arccos(dx / dx\_plus\_dz) \* np.sign(dz)*

*b = 0.5*

*k = np.array([*

*-b \*\* 2,*

*-2 \* b*

*])*

*f = lambda t: self.v\_zsk - self.Vz\_ # noqa*

*int\_a = self.t # ???*

*int\_b = self.t + self.dt # ???*

*self.gamma\_ = 1 / (self.G[1] \* self.m) \* (*

*self.m \* (*

*k[1] \* (self.v\_zsk - self.Vz\_) + k[0] \* np.array([*

*quad(lambda t: f(t)[i], int\_a, int\_b)[0] for i in range(3)*

*])*

*) - self.Fs\_zsk[2] + abs(self.c[5]) \* self.omega[5] \*\* 2*

*)*

*k1 = -0.1*

*self.teta\_ = -(*

*self.m \* k1 \* (self.v[1] - self.Vx\_) - self.Fs[0]*

*) / (self.G[1] \* self.m)*

*b = 0.1*

*k = np.array([*

*-b \*\* 2,*

*-2 \* b*

*])*

*x\_, z\_ = self.v[0], self.v[2]*

*d\_R = np.quaternion(0, self.pos[0] - x\_, 0, self.pos[2] - z\_) # ???*

*d\_R\_zsk = (self.Lam.inverse() \* d\_R \* self.Lam).vec*

*d\_vx, \_, d\_vz = d\_R\_zsk*

*self.gamma\_ = (*

*self.m \* (*

*k[1] \* self.v[2] + k[0] \* d\_vz*

*) - self.Fs[2] + abs(self.c[5]) \* self.omega[5] \*\* 2*

*) / (self.G[1] \* self.m)*

*self.teta\_ = -(*

*self.m \* (*

*k[1] \* self.v[0] + k[0] \* d\_vx*

*) - self.Fs[0]*

*) / (self.G[1] \* self.m)*

*def calc\_forces(self):*

*F = np.array([*

*self.Kf0 \* self.omega[0] \*\* 2 \* self.e2,*

*self.Kf \* self.omega[1] \*\* 2 \* self.e2,*

*self.Kf \* self.omega[2] \*\* 2 \* self.e2,*

*self.Kf \* self.omega[3] \*\* 2 \* self.e2,*

*self.Kf \* self.omega[4] \*\* 2 \* self.e2,*

*self.Kf \* self.omega[5] \*\* 2 \* self.e3*

*])*

*Fe\_zsk = np.sum(F, axis=0)*

*Lam\_psi = np.quaternion(*

*np.cos(self.psi/2),*

*0,*

*-np.sin(self.psi/2),*

*0*

*)*

*Lam\_teta = np.quaternion(*

*np.cos(self.teta/2),*

*0,*

*0,*

*np.sin(self.teta/2)*

*)*

*Lam\_gamma = np.quaternion(*

*np.cos(self.gamma/2),*

*np.sin(self.gamma/2),*

*0,*

*0*

*)*

*Lam = Lam\_psi \* Lam\_teta \* Lam\_gamma*

*Phiez = np.quaternion(*

*0,*

*Fe\_zsk[0],*

*Fe\_zsk[1],*

*Fe\_zsk[2]*

*)*

*Fe = (Lam \* Phiez \* Lam.inverse()).vec*

*v = np.quaternion(0, self.v[0], self.v[1], self.v[2])*

*v\_zsk = (Lam.inverse() \* v \* Lam).vec*

*Fs\_zsk = np.array([*

*-self.Ks[0] \* v\_zsk[0] \*\* 2 \* np.sign(v\_zsk[0]),*

*-self.Ks[1] \* v\_zsk[1] \*\* 2 \* np.sign(v\_zsk[1]),*

*-self.Ks[2] \* v\_zsk[2] \*\* 2 \* np.sign(v\_zsk[2])*

*])*

*Phis\_zsk = np.quaternion(*

*0,*

*Fs\_zsk[0],*

*Fs\_zsk[1],*

*Fs\_zsk[2]*

*)*

*Fs = (Lam \* Phis\_zsk \* Lam.inverse()).vec*

*H = np.array([*

*self.I[0] \* self.Om[0],*

*self.I[1] \* self.Om[1] + self.J \* (*

*self.omega[1] - self.omega[2] + self.omega[3] - self.omega[4]*

*) + self.J0 \* self.omega[0],*

*self.I[2] \* self.Om[2] + self.J \* self.omega[5]*

*])*

*Mg\_zsk = np.cross(self.Om, H) # векторний доб*

*dH = np.array([*

*0,*

*self.J \* (*

*self.eps[1] - self.eps[2] + self.eps[3] - self.eps[4]*

*) + self.J0 \* self.eps[0],*

*self.J \* self.eps[5]*

*])*

*MFe = np.sum(np.cross(self.p, F), axis=0)*

*Ma = np.array([*

*self.Km0 \* self.omega[0] \*\* 2 \* self.e2,*

*self.Km \* self.omega[1] \*\* 2 \* self.e2,*

*-self.Km \* self.omega[2] \*\* 2 \* self.e2,*

*self.Km \* self.omega[3] \*\* 2 \* self.e2,*

*-self.Km \* self.omega[4] \*\* 2 \* self.e2,*

*self.Km \* self.omega[5] \*\* 2 \* self.e3*

*])*

*MAe = np.sum(Ma, axis=0)*

*self.F = F*

*self.Lam = Lam*

*self.Fe = Fe*

*self.v\_zsk = v\_zsk*

*self.Fs\_zsk = Fs\_zsk*

*self.Fs = Fs*

*self.Mg\_zsk = Mg\_zsk*

*self.dH = dH*

*self.MFe = MFe*

*self.MAe = MAe*

Далі я передав початкові параметри в цю модель і отримав такі результати (рис. 1-4).



Рисунок 1 - поведінка координат відносно ІСК під час польоту

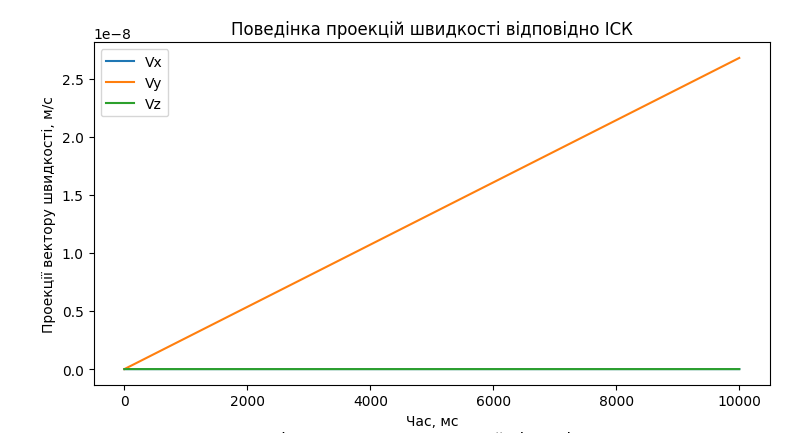


Рисунок 2 - поведінка проекцій швидкості відносно ІСК

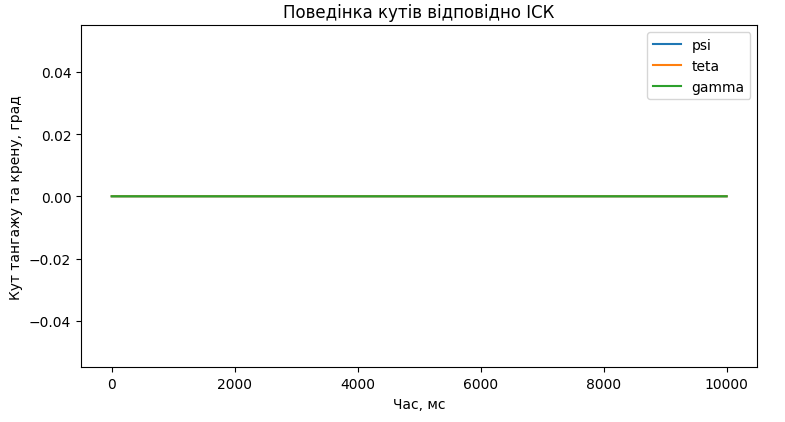


Рисунок 3 - поведінка кутів відносно ІСК

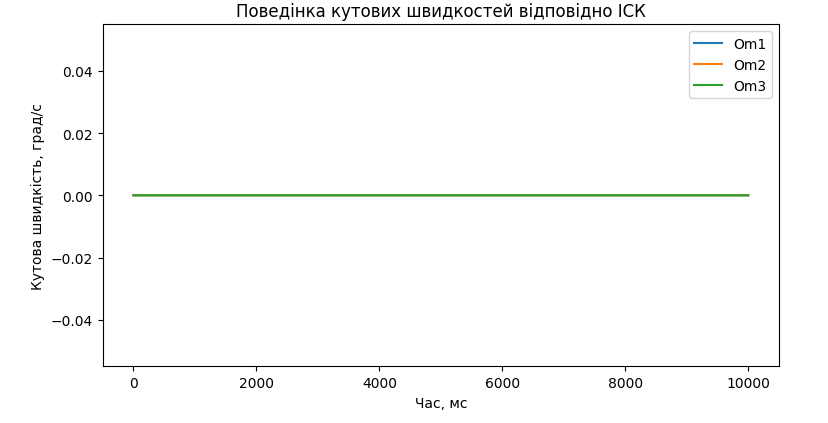


Рисунок 4 - поведінка кутів відносно ІСК

Також до цього файлу додається таблиця даних отриманих у кожний момент дискретного часу.

**Висновки:** Отже я спробував створити модель беспілотного літального апарату та меня вдалося втримати його в повітрі. Хоча все ж таки вийшов дуже спрощений варіант, адже я не використовував 2 двигуни, які відповідають за керування цим літальним апаратом. Мені сподобалось виконувати цю лабораторну роботу, було доволі таки цікаво. Стало трошки зрозуміліше як працюють беспілотні літальні апарати.