```python
# Симуляция квадрокоптера
import numpy as np
import math

def c(x):
    return np.cos(x)
def s(x):
    return np.sin(x)
def t(x):
    return np.tan(x)


class Quadcopter():

    def __init__(self, pos, vel, angle, ang_vel, pos_ref, dt):

        mass = 0.468  # масса квадрокоптера, кг
        gravity = 9.81  # ускорение свободного падения, м/с^2
        num_motors = 4  # количество моторов
        kt = 1e-7  # коэффициент для перевода скорости вращения мотора в тягу, Н/(об/мин)^2

        # Начальные параметры симуляции
        self.pos = pos  # положение в инерциальной системе координат [x, y, z], м
        self.vel = vel  # скорость в инерциальной системе координат [ẋ, ẏ, ż], м/с
        self.angle = angle  # ориентация (углы Эйлера) в радианах [крен (phi), тангаж (theta), рысканье (psi)]
        self.ang_vel = ang_vel  # угловая скорость [phi, thetȧ, psi], рад/с
        self.lin_acc = np.array([0., 0., 0.])  # линейное ускорение, м/с^2
        self.ang_acc = np.array([0., 0., 0.])  # угловое ускорение, рад/с^2

        # Желаемые состояния
        self.pos_ref = pos_ref  # целевая позиция [x, y, z], м
        self.vel_ref = [0., 0., 0.]  # целевая скорость, м/с
        self.lin_acc_ref = [0., 0., 0.]  # целевое ускорение, м/с^2
        self.angle_ref = [0., 0., 0.]  # целевые углы, рад
        self.ang_vel_ref = [0., 0., 0.]  # целевая угловая скорость, рад/с
        self.ang_acc_ref = [0., 0., 0.]  # целевое угловое ускорение, рад/с^2

        # Параметры времени
        self.time = 0
        self.dt = dt

        # Параметры окружающей среды
        self.gravity = gravity  # ускорение свободного падения, м/с^2
        self.density = 1.225  # плотность воздуха, кг/м^3

        # Границы среды
        self.bound_x = 10.
        self.bound_y = 10.
        self.bound_z = 10.

        # Константы квадрокоптера
        self.num_motors = num_motors  # количество моторов
        self.mass = mass  # масса квадрокоптера
        self.Ixx = 4.856e-5  # момент инерции по оси X, кг·м²
        self.Iyy = 4.856e-5  # момент инерции по оси Y, кг·м²
        self.Izz = 8.8e-5  # момент инерции по оси Z, кг·м²
        self.A_ref = 0.02  # эффективная площадь для расчета сопротивления, м²
        self.L = 0.2  # расстояние от центра до мотора, м
        self.kt = kt  # коэффициент тяги
        self.b_prop = 1.14e-7  # коэффициент момента, Н·м/(об/мин)^2
        self.Cd = 1  # коэффициент сопротивления
        self.thrust = mass * gravity  # начальная тяга
        self.speeds = np.ones(num_motors) * ((mass * gravity) / (kt * num_motors))  # начальные скорости моторов
        self.tau = np.zeros(3)  # начальный момент

        self.maxT = 2.5   # максимальная тяга на мотор, Н
        self.minT = 0.2  # минимальная тяга на мотор, Н
        self.max_angle = math.pi / 12  # максимальный угол наклона, рад

        self.I = np.array([[self.Ixx, 0, 0], [0, self.Iyy, 0], [0, 0, self.Izz]])  # тензор инерции
        self.g = np.array([0, 0, -gravity])  # вектор силы тяжести

        self.done = False

    def calc_pos_error(self, pos):
```

```
        ''' Возвращает ошибку по положению '''
        return self.pos_ref - pos

    def calc_vel_error(self, vel):
        ''' Возвращает ошибку по скорости '''
        return self.vel_ref - vel

    def calc_ang_error(self, angle):
        ''' Возвращает ошибку по углам ориентации '''
        return self.angle_ref - angle

    def calc_ang_vel_error(self, ang_vel):
        ''' Возвращает ошибку по угловой скорости '''
        return self.ang_vel_ref - ang_vel

    def body2inertial_rotation(self):
        '''
        Преобразование из системы координат тела в инерциальную (углы Эйлера)
        angle 0 = крен (ось X, phi)
        angle 1 = тангаж (ось Y, theta)
        angle 2 = рысканье (ось Z, psi)
        '''

        c1 = c(self.angle[0])
        s1 = s(self.angle[0])
        c2 = c(self.angle[1])
        s2 = s(self.angle[1])
        c3 = c(self.angle[2])
        s3 = s(self.angle[2])

        R = np.array([[c2 * c3, c3 * s1 * s2 - c1 * s3, s1 * s3 + c1 * s2 * c3],
                      [c2 * s3, c1 * c3 + s1 * s2 * s3, c1 * s3 * s2 - c3 * s1],
                      [-s2, c2 * s1, c1 * c2]])

        return R

    def inertial2body_rotation(self):
        '''
        Преобразование из инерциальной системы в систему тела
        (транспонированная матрица из body2inertial_rotation)
        '''
        return np.transpose(self.body2inertial_rotation())

    def thetadot2omega(self):
        ''' Перевод углов Эйлера (Euler_dot) в угловую скорость (omega) '''
        R = np.array([[1, 0, -s(self.angle[1])],
                      [0, c(self.angle[0]), c(self.angle[1]) * s(self.angle[0])],
                      [0, -s(self.angle[0]), c(self.angle[1]) * c(self.angle[0])]])
        return np.matmul(R, self.ang_vel)

    def omegadot2Edot(self, omega_dot):
        ''' Перевод угловой скорости (omega) в производные углов Эйлера (Euler_dot) '''
        R = np.array([[1, s(self.angle[0]) * t(self.angle[1]), c(self.angle[0]) * t(self.angle[1])],
                      [0, c(self.angle[0]), -s(self.angle[0])],
                      [0, s(self.angle[0]) / c(self.angle[1]), c(self.angle[0]) / c(self.angle[1])]])
        self.ang_acc = np.matmul(R, omega_dot)

    def find_omegadot(self, omega):
        ''' Вычисляет угловое ускорение в инерциальной системе, рад/с² '''
        omega = self.thetadot2omega()
        return np.linalg.inv(self.I).dot(self.tau - np.cross(omega, np.matmul(self.I, omega)))

    def find_lin_acc(self):
        ''' Вычисляет линейное ускорение, м/с² '''
        R_B2I = self.body2inertial_rotation()
        R_I2B = self.inertial2body_rotation()

        # Силы в системе тела
        Thrust_body = np.array([0, 0, self.thrust])
        Thrust_inertial = np.matmul(R_B2I, Thrust_body)

        vel_bodyframe = np.matmul(R_I2B, self.vel)
        drag_body = -self.Cd * 0.5 * self.density * self.A_ref * (vel_bodyframe) ** 2
        drag_inertial = np.matmul(R_B2I, drag_body)
        weight = self.mass * self.g
```

```
        acc_inertial = (Thrust_inertial + drag_inertial + weight) / self.mass
        self.lin_acc = acc_inertial

    def des2speeds(self, thrust_des, tau_des):
        ''' Вычисляет скорости моторов для заданной тяги и моментов '''
        e1 = tau_des[0] * self.Ixx
        e2 = tau_des[1] * self.Iyy
        e3 = tau_des[2] * self.Izz

        n = self.num_motors
        weight_speed = thrust_des / (n * self.kt)

        motor_speeds = []
        motor_speeds.append(weight_speed - (e2 / ((n / 2) * self.kt * self.L)) - (e3 / (n * self.b_prop)))
        motor_speeds.append(weight_speed - (e1 / ((n / 2) * self.kt * self.L)) + (e3 / (n * self.b_prop)))
        motor_speeds.append(weight_speed + (e2 / ((n / 2) * self.kt * self.L)) - (e3 / (n * self.b_prop)))
        motor_speeds.append(weight_speed + (e1 / ((n / 2) * self.kt * self.L)) + (e3 / (n * self.b_prop)))

        # Проверка пределов тяги
        thrust_all = np.array(motor_speeds) * self.kt
        over_max = np.argwhere(thrust_all > self.maxT)
        under_min = np.argwhere(thrust_all < self.minT)

        if over_max.size != 0:
            for i in range(over_max.size):
                motor_speeds[over_max[i][0]] = self.maxT / self.kt
        if under_min.size != 0:
            for i in range(under_min.size):
                motor_speeds[under_min[i][0]] = self.minT / self.kt

        self.speeds = motor_speeds

    def find_body_torque(self):
        ''' Вычисляет момент на теле от разности тяги моторов '''
        tau = np.array([
            (self.L * self.kt * (self.speeds[3] - self.speeds[1])),
            (self.L * self.kt * (self.speeds[2] - self.speeds[0])),
            (self.b_prop * (-self.speeds[0] + self.speeds[1] - self.speeds[2] + self.speeds[3]))
        ])
        self.tau = tau

    def step(self):
        # Суммарная тяга от моторов
        self.thrust = self.kt * np.sum(self.speeds)

        # Линейное и угловое ускорения
        self.find_lin_acc()
        self.find_body_torque()

        # Угловое ускорение в инерциальной системе
        omega = self.thetadot2omega()
        omega_dot = self.find_omegadot(omega)

        # Угловое ускорение в теле
        self.omegadot2Edot(omega_dot)

        # Обновление состояний по шагу времени
        self.ang_vel += self.dt * self.ang_acc
        self.angle += self.dt * self.ang_vel
        self.vel += self.dt * self.lin_acc
        self.pos += self.dt * self.vel
        self.time += self.dt
```