Санкт–Петербургский Государственный Университет
факультет Прикладной Математики - Процессов Управления
кафедра Механики Управляемого Движения

ЛИТВИНОВ Никита Константинович

Выпускная квалификационная работа.

Управление мультикоптером в аварийной ситуации

Уровень образования: магистратура
Направление 01.04.02 «Прикладная математика и информатика»
Образовательная программа ВМ.5517.2020 «Методы прикладной математики и информатики в задачах управления»

Научный руководитель:
доцент, кафедра механики управляемого движения,
к.ф. - м.н. Шиманчук Дмитрий Викторович

Рецензент:
доцент, кафедра общенаучных дисциплин,
«Военная академия материально-технического обеспечения имени генерала армии А.В.Хрулёва»
к.т.н. Волков Юрий Александрович

Санкт-Петербург
2022 г.

Содержание

Введение.

В последнее время такое устройство как мобильный квадрокоптер небольших размеров и мощностей вышло на массовое производство. И не удивительно, так как простая конструкция, удовлетворительные показали грузоподъёмности и относительная дешевизна сделали его де-факто стандартом для съёмочного процесса в труднодоступных местах. Также в следствии конструкции, квадрокоптер имеет слабую парусность, что делает его мало зависимым от ветра, в отличие от других беспилотных летательных аппаратов.

Однако, при удешевлении и облегчении конструкции и моторов пропеллеров возникает риск отказа двигателей или механической поломки двигателей. А на борту коптер может нести дорогостоящее или хрупкое оборудование, поэтому тема сохранения общей целостности устройства в аварийной ситуации с помощью специального управления нуждается в тщательном и всестороннем исследовании.

Распределения тяги для ротации положения квадрокоптера в пространстве изображено на рис.(1). Для стационарного положения суммарная тяга винтов должна равняться силе тяжести, для поворота на месте ослабляется тяга на двух симметричных винтах и увеличивается на двух других (два верхних на рис.(1)).

Для движения в каком либо направлении ослабляется тяга на винтах, ближайших к вектору направления движения, а на противоположных увеличивается, при сохранении суммарной тяги для поддержания высоты (то есть противопоставление силе тяжести).

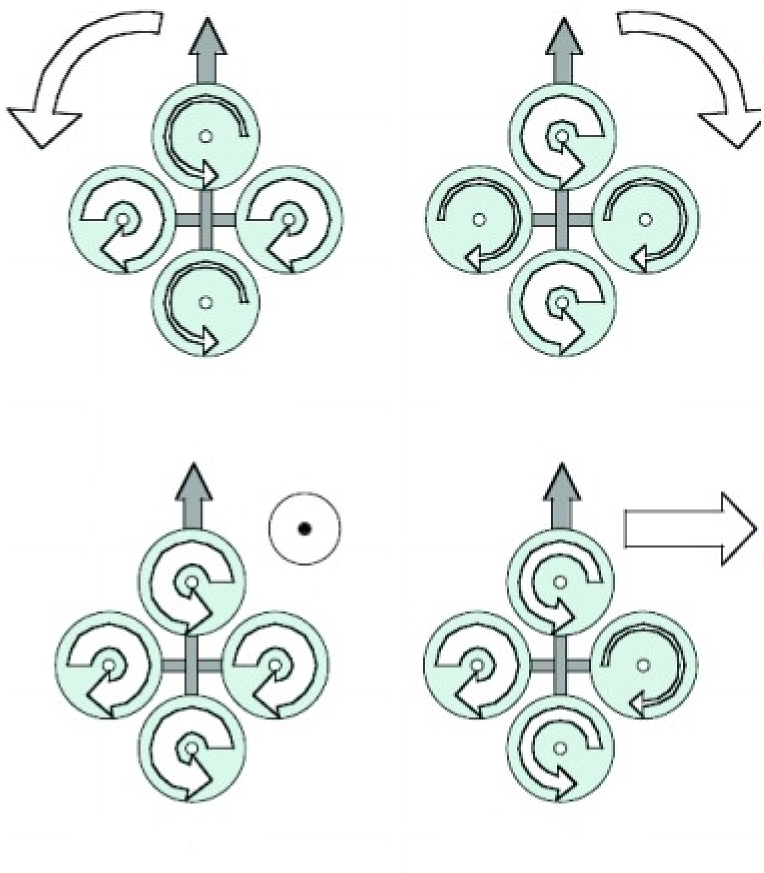

Рис. 1: Тяга двигателей для корректировки положения.

Из-за схемы перемещения, которая подразумевает наклоны корпуса коптера, придётся ввести в управление способ нейтрализации отклонений от начального положения на программной кривой.

Обзор литературы

С управлением квадрокоптера связано большое количество работ, среди них можно выделить [2], в которой проведена классификация алгоритмов управления для различных моделей квадрокоптеров с качественным анализом этих алгоритмов. . Однако, как и с любым устройством, возникают проблемы нештатных ситуаций, когда отказывают двигатели, падает тяга или временно теряется управление. На эту тему уже имеется широкий ряд исследований, например, в [3] проведён анализ таких ситуаций, а в [1] подробно остановились на динамической модели и провели численные эксперименты для различных случаев.

Классификация аварийных ситуациаций представлена в [3] и [1]. Приведём эти результаты в общем списке:

- падение тяги не ниже критической (как правило берётся 30%);
- временный отказ двигателя (время меньше чем частота управления или расчёта нового);
- падение тяги ниже критической отметки, здесь рассматривается три случая:
  - одного из двигателей;
  - двух симметричных;
  - двух смежных;
  - трёх двигателей (очевидно что при отказе всех четырёх устройство окажется неуправляемым);
- те же три случая из предыдущего пункта, но с полным отказом двигателя;

Для каждого случая существует много решений, в большинстве из перечисленных случаев это алгоритм плавной посадки в автоматическом режиме, направленный на снижение силы удара и максимальное сокращение расстояния до целевой точки на земле, иногда можно стабилизировать полёт для достижения предполагаемого оператором места призмеления.

В работе [14] подробно останавливаются на общих принципах работы линейно-квадратичных регуляторов, и регулируемых ими системах. В работах [1] - [13] рассматриваются математические модели коптеров и методы управления ими.

А источники [15] - [20] служили фундаментальной основой вывода уравнений и методов исследования.

Глава 1. Постановка задачи.

В работе ставятся следующие задачи:

- Построить программное движение для выхода из аварийной ситуации;
- Построить управление на программном движении, учитывающее отклонения от программной кривой;
- Численно промоделировать полученное решение

Цель работы: Проанализировать возможные аварийные ситуации и разработать безопасное решение.

Глава 2. Математическая модель квадрокоптера.

Схематически квадрокоптер изображён на рис.2, система координат, жёсто связанная с телом, помещена в центр масс устройства, ось Oz направлена перпендикулярна плоскости вращения винтов.

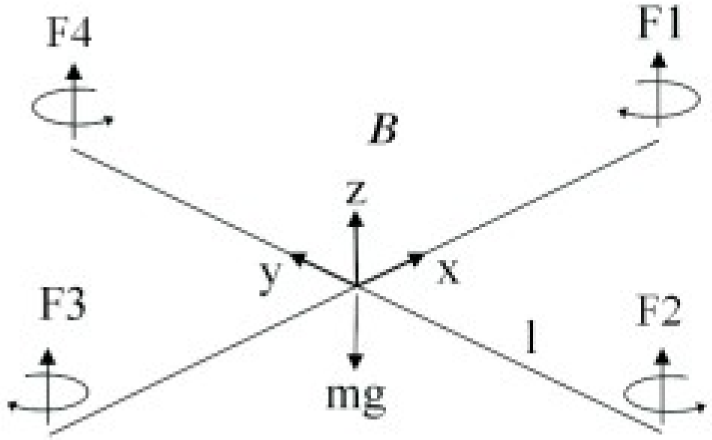

Рис. 2: Схематическое изображение квадрокоптера и действующих на него сил

Также на рис. (2) изображены основные силы, действующие на квадрокоптер (вследствии слабой парусности и относительно малой скорости полёта сопротивление воздуха учитывается только во вращении винтов), l – расстояние от винта до центра масс, B – body, тело квадрокоптера, буквой P будем обозначать всё что связано непосредственно с пропеллерами.

2.1 Кинематика.

Рассмотрим построение матрицы ориентации через углы Эйлера:

$$R(\varphi, \psi, \gamma) = R(\psi)R(\theta)R(\varphi). \tag{1}$$

Матрицы поворота относительно осей координат в (1):

$$R(y,\psi) = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{2}$$

$$R(z,\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & cos\theta \end{pmatrix}, \tag{3}$$

$$R(\varphi) = \begin{pmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{4}$$

Результирующая матрица ориентации, из уравнений (1), (2), (3) и (4):

$$R(\varphi,\psi,\gamma) = \begin{pmatrix} \cos\psi\cos\varphi - \sin\psi\sin\varphi\cos\theta & -\cos\psi\sin\varphi - \sin\psi\cos\varphi\cos\theta & \sin\psi\sin\theta \\ \sin\psi\cos\varphi + \cos\psi\sin\varphi\cos\theta & -\sin\psi\sin\varphi + \cos\psi\cos\varphi\cos\theta & -\cos\psi\sin\theta \\ \sin\varphi\sin\theta & \cos\varphi\sin\theta & \cos\theta \end{pmatrix} \tag{5}$$

Тогда радиус вектор любой точки квадрокоптера в абсолютной системе координат выражается через матрицу ориентации 5 как:

$$\overline{r_B} = R\begin{pmatrix} x \\ y \\ z \end{pmatrix} + \overline{r}. \tag{6}$$

Из 6, продифференцировав, можно получить скорость любой точки тела коптера через радиус-вектор центра.

Обозначим вектор угловой скорости тела как

$$\omega^B = \begin{pmatrix} p \\ q \\ r \end{pmatrix}, \tag{7}$$

где p, q, r – составляющие угловой скорости по осям декартовой системы координат, связанной с телом коптера.

Тогда зависимость углов ориентации от угловой скорости будет выражаться как:

$$\begin{pmatrix} \dot{\psi} \\ \dot{\theta} \\ \dot{\varphi} \end{pmatrix} = \begin{pmatrix} (p \sin\varphi + q\cos\varphi)\frac{1}{\sin\theta} \\ p\cos\varphi - q\sin\varphi \\ r - (p\sin\varphi - q\cos\varphi)\frac{1}{\tan\theta} \end{pmatrix} \tag{8}$$

Интегрируя 8, и используя 5 и 6 можно получить положение тела коптера в пространстве.

2.2 Динамика.

По 2му закону Ньютона:

$$m\ddot{\overline{r}} = R\overline{e_z}\sum_{n=1}^{4} F_n + m\overline{g}, \tag{9}$$

где R - матрица ориентации в абсолютной системе координат, $\overline{r}$ – радиус вектор центра масс квадрокоптера, $\overline{e_z}$ – вектор, задающий ось Oz абсолютной системы координат, F_n – сила тяги винта.

Для упрощения математической модели сделаем несколько предположений:

- сила тяги прямо пропорциональна квадрату его угловой скорости, то есть
$$F_n = k_f\omega_n^2, \tag{10}$$
где k_f – некоторый коэффициент, получающийся из конкретной конфигурации винта;

- реактивный момент пропеллера линейно пропорционален силе тяги с коэффициентом k_M, то есть:
$$M_i = (-1)^{n+1}k_M F_N. \tag{11}$$

- аэродинамическое сопротивление вращению винта пропорционально угловой скорости вращения тела коптера по оси O_z системы координат, жёстко связанной с телом:

$$M_a = (0, 0, -k_z r). \tag{12}$$

Рассмотрим моменты всех сил, действующих на тело. Моменты внешних сил (учитывая (11) и (12)):

$$M_{res} = \begin{pmatrix} 0 \\ F_3 l \\ 0 \end{pmatrix} - \begin{pmatrix} 0 \\ F_1 l \\ 0 \end{pmatrix} + \begin{pmatrix} F_2 l \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} F_4 l \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ k_M \sum_{n=1}^{4} F_n \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ -k_z r \end{pmatrix}, \tag{13}$$

С другой стороны, результирующий момент равен:

$$M_{res} = I^B \dot{\omega}^B + \sum_{n=1}^{4} I^P \dot{\omega}^{P_n} + \underbrace{\omega^B \times (I^B \omega^B + \sum_{n=1}^{4} I^P (\omega^B + \omega^{P_n}))}_{=M_S}, \tag{14}$$

где M_S – момент, вызванный силой Кориолиса и реактивным моментом всех пропеллеров, учитывая (7) он запишется как:

$$M_S = \begin{pmatrix} ((I_{zz}^B + 4I_{zz}^P) - (I_{yy}^B + 4I_{xx}^P))qr + I_{zz}^P q \sum_{n=1}^{4} \omega_n \\ -((I_{zz}^B + 4I_{zz}^P) - (I_{xx}^B + 4I_{xx}^P))pr - I_{zz}^P p \sum_{n=1}^{4} \omega_n \\ ((I_{yy}^B + 4I_{xx}^P) - (I_{xx}^B + 4I_{xx}^P))pq \end{pmatrix}. \tag{15}$$

Из уравнений (10), (13), (14) и (15) получаем дифференциальные уравнения для составляющих угловой скорости тела квадрокоптера:

$$\begin{cases} I_{xx}^B \dot{p} = k_f(\omega_2^2 - \omega_4^2)l - ((I_{zz}^B + 4I_{zz}^P) - (I_{yy}^B + 4I_{xx}^P))qr - I_{zz}^P q \sum_{n=1}^{4} \omega_n \\ I_{xx}^B \dot{q} = k_f(\omega_3^2 - \omega_1^2)l + ((I_{zz}^B + 4I_{zz}^P) - (I_{xx}^B + 4I_{xx}^P))pr - I_{zz}^P p \sum_{n=1}^{4} \omega_n \\ I_{zz}^B \dot{r} = -k_z r + k_M k_f(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2). \end{cases} \tag{16}$$

Третья составляющая из уравнения (15) обнулилась, так как $I_{yy}^B = I_{xx}^B$

из-за симметричности тела коптера.

Далее для любой аварийной ситуации в уравнениях (16) будет обнуляться угловая скорость вышедшого из строя двигателя либо накладываться ограничения при падении тяги в двигателе. Управлять мы можем лишь тягой двигателей, то есть угловыми скоростями вращения пропеллеров ω_n

Глава 3. Управление при отказе двигателя

Отказ двигателя означает остановку и падение тяги до нуля, не умаляя общности, возьмём 4й. То есть $\widehat{\omega_4} = 0$ и $F_4 = 0$, а это в свою очередь означает и обнуление реактивного момента этого двигателя:

$$M_4 = (-1)^5 k_M F_4 = 0. \tag{17}$$

Однако, посмотрев на рис.(2), можно заметить что симметричный отказавшему двигатель будет выводить систему из равновесия относительно горизонтальной плоскости, и коптер войдёт в непредсказуемое неуправляемое падение. Поэтому для сохранения устойчивости системы необходимо отключение ещё одного двигателя, что делает случаи отказа одного и двух симметричных двигателей абсолютно аналогичными (в некоторых работах предлагается управление квадрокоптером при трёх работающих двигателях, но в данной работе остановимся подробнее на стабилизации положения коптера на одной высоте и с неизменным положением центра масс, что без отключения симметричного двигателя невозможно), то есть:

$$\widehat{\omega_2} = 0, F_2 = 0, M_2 = 0. \tag{18}$$

Также невозможно управление квадрокоптером в случае отказа двух соседних двигателей, так как нечем будет уравновесить опрокидывающий эффект рычага относительно центра тяжести.

3.1 Программное движение

Итак, квадрокоптер превращается в биспинер, что будет вращаться относительно вертикальной оси через центр масс под действием реактивного момента двух симметричных двигателей, которые в силу конструкции вращаются в одну сторону. Управлять в такой системе возможно лишь двумя значениями силы тяги, вернее их разностью и суммарной тягой: $F_1 + F_3$ и $F_1 - F_3$, а точнее угловыми скоростями пропеллеров ω_1 и ω_3. Для поддержания или набора высоты нужно управлять суммарным значение тяги, что видно из рис. (2).

Уравнение (8) для углов ориентации через составляющие угловой скорости тела коптера не подвергнутся измениям, но выражения (16) с учётом (17) и (18) преобразуются в:

$$\begin{cases} I_{xx}^B \dot{p} = -(I_{zz}^B + 4I_{zz}^P - I_{xx}^B - 4I_{xx}^P)qr - I_{zz}^P q(\omega_1 + \omega_3), \\ I_{xx}^B \dot{q} = (I_{zz}^B + 4I_{zz}^P - I_{xx}^B - 4I_{xx}^P)pr + I_{zz}^P p(\omega_1 + \omega_3) + k_f l(\omega_1^2 - \omega_3^2), \\ I_{zz}^B \dot{r} = -k_z r + k_M k_f(\omega_1^2 + \omega_3^2). \end{cases} \tag{19}$$

А движение по траектории рассчитывается посредством второго закона Ньютона (9), перепишем его в виде:

$$\ddot{d} = \begin{pmatrix} 0 \\ 0 \\ -g \end{pmatrix} + R\widehat{e_z}\frac{k_f}{m}\begin{pmatrix} 0 \\ 0 \\ \omega_1^2 + \omega_3^2 \end{pmatrix}. \tag{20}$$

Из уравнения (20) делаем вывод, что для поддержания квадрокоптера на определённой высоте при двух работающих двигателях необходимо:

$$\omega_n = \sqrt{\frac{mg}{2k_f}}, \tag{21}$$

а при всех четырёх работающих двигателях:

$$\omega_n = \sqrt{\frac{mg}{4k_f}}. \tag{22}$$

Очевидно, что угловая скорость не может изменяться мгновенно, основываясь на работе [1] возьмём закон изменения угловой скорости в простейшем случае:

$$\dot{\omega_c} = \frac{1}{T_\omega}(\omega_d - \omega_c), \tag{23}$$

где ω_d – желаемая(desired) угловая скорость пропеллера, ω_c – текущая(current), T_ω – задержка, обычно равная примерно 0,01-0,02.

То есть для того, чтобы избежать падения квадкоптера, необходимо довести угловую скорость пропеллера из начальной (для моделирования предположим что это (22)) в конечную угловую скорость для дрейфа (21). А далее придерживаться решения системы (19) при нулевых угловых ускорениях и угловых скоростях пропеллера (21):

$$\widehat{p} = 0, \widehat{q} = 0, \widehat{r} = \frac{k_M m g}{k_z} \tag{24}$$

Однако, недостаточно просто достигнуть скорости дрейфа и придерживаться (24), из-за задержки (23) появится некоторая абсолютная скорость при первом достижении угловой скорости дрейфа. Значит нужно обеспечить режим торможения.

3.2 Стабилизация на программной кривой

Как было сказано выше, при стабилизирующем воздействии на вращение относительно осей O_x, O_y жёстко связанной с телом системы координат, суммарное значение тяги, то есть $k_f(\omega_1^2 + \omega_3^2)$ в третьем уравнении системы (19), должно оставаться постоянным для поддержания или стабильного набора высоты, а управлять мы можем лишь разностью $\omega_1^2 - \omega_3^2$, учитывая упомянутое ограничение.

Для стабилизации системы недостаточно обнулить угловые ускорения $\dot{p}$ и $\dot{q}$, необходимо также вернуться к начальной ориентации, для этого возьмём вектор нормали, который меняется по закону:

$$\dot{n} = \omega^B \times n. \tag{25}$$

Будет достаточно взять лишь две компоненты нормали n_x и n_y, так как треться выражается через них как $n_z = \sqrt{1 - n_x^2 - n_y^2}$.

Введём переменную новую переменную:

$$s = \begin{pmatrix} p \\ q \\ n_x \\ n_y \end{pmatrix}, \tag{26}$$

и тогда отклонение от программной кривой:

$$\widetilde{s} = s - \widehat{s},$$

где s – действительная ориентация в данный момент времени, а $\widehat{s}$ – ориентация на программной кривой.

Линеаризуем систему, используя уравнения для (19), (25) и (26):

$$\dot{\widetilde{s}} = A\widetilde{s} + Bu, \tag{27}$$

где

$$A = \frac{\partial \dot{s}}{\partial s}\Big|_{s=\overline{s}} = \begin{pmatrix} 0 & -f & 0 & 0 \\ f & 0 & 0 & 0 \\ 0 & -\widehat{n}_z & 0 & \widehat{r} \\ \widehat{n}_z & 0 & -\widehat{r} & 0 \end{pmatrix}, B = \begin{pmatrix} 0 \\ \frac{l}{I_{xx}^B} \\ 0 \\ 0 \end{pmatrix},$$

где

$$f = (I_{zz}^B + 4I_{zz}^P - I_{xx}^B - 4I_{xx}^P)\frac{\widehat{r}}{I_{xx}^B} + \frac{I_{zz}^P}{I_{xx}^B}(\omega_1 + \omega_3).$$

По критерию Калмана система (27) полностью управляема когда ранг матрицы управляемости не равен 4, то есть она невырождена, для проверки найдём её определитель и приравняем к нулю:

$$det(BABA^2BA^3B) = -f(\frac{l}{I_{xx}^B})^4\widehat{n}_z^2\widehat{r}(\widehat{r} - f)^2 = 0.$$

Очевидно что определитель равен нулю только в случае $\widehat{r} = f$, в том числе когда оба равны нулю.

Для стабилизации системы будем строить линейно-квадратичный регулятор – LQR, когда управление ищется в виде $u = K^{opt}\widetilde{s}$, где K^{opt} – аргумент для минимума функционала качества:

$$J(K) = \int_0^\infty (y^T R_J y + u^T Q u)\, dt, \tag{28}$$

где R_J, Q – весовые матрицы, y – наблюдение.

Есть множество программно реализованных алгоритмов LQR, поэтому ограничусь кратким описанием работы, а сам для моделирования воспользуюсь в Python 3.10 встроенной функцией control.lqr().

Глава 4. Численный эксперимент

Для численной реализации задержка T_ω для более наглядных графиков в этой главе взята равной 0.5.

Эксперимент проводился для наглядной проверки валидности предложенной математической модели, а в частности того что биспиннер стабилизируется и двигается по описанным динмическим уравнениям. Вес тела коптера был взят 0.5, расстояние от центра масс до пропеллера 0.3, ускорение свободного падения 9.81. Коэффициенты связи тяги с угловой скоростью, реактивного момента, аэродинамического сопротивления вращению винта соотвественно: $k_f = 6.41 * 10^{-6}, k_M = 1.69 * 10^{-2}, k_z = 2.75 * 10^{-3}$. Коптер стартовал зависнув на высоте 5.

Моменты инерции пропеллера и тела коптера: $I_{zz}^B = 5.5 * 10^{-3}, I_{xx}^B = 3.14 * 10^{-3}, I_{zz}^P = 1.5 * 10^{-9}, I_{xx}^P = 1.5 * 10^{-5}$.

Проиллюстрируем что происходит при управлении на достижение тяги для дрейфа без торможения:

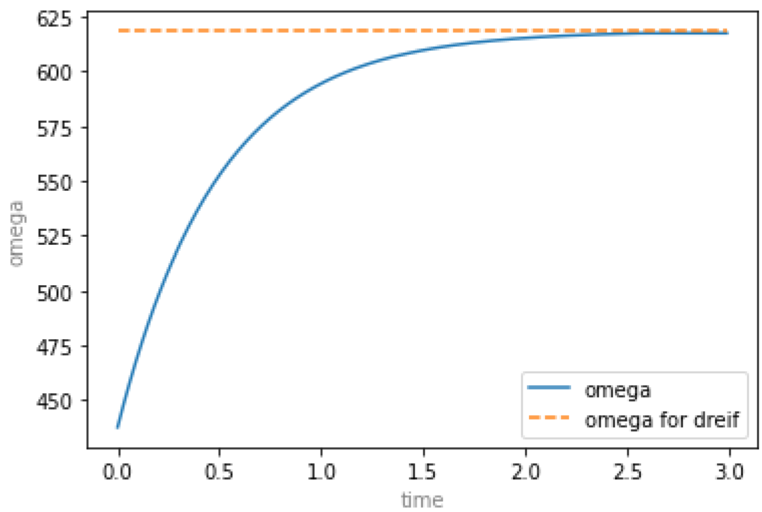

Рис. 3: Изменение угловой скорости от начального до достижения тяги дрейфа

В этом случае сила тяги пропеллеров уравновешивает силу тяжести, то есть сумма всех сил, действующих на тело равняется нулю, и по инерции не изменяя скорости коптер продолжает лететь вниз до 0, что показано на рис. (4):

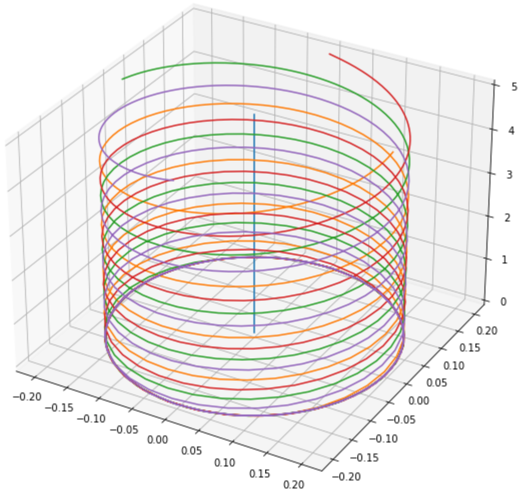

Рис. 4: Движение основных точек во время падения биспиннера.

При добавлении отклонения (0.1, 0.1, 0.1) от нулевой начальной угловой скорости вращения срабатывает алгоритм LQR, что проиллюстрировано на рис.(5)

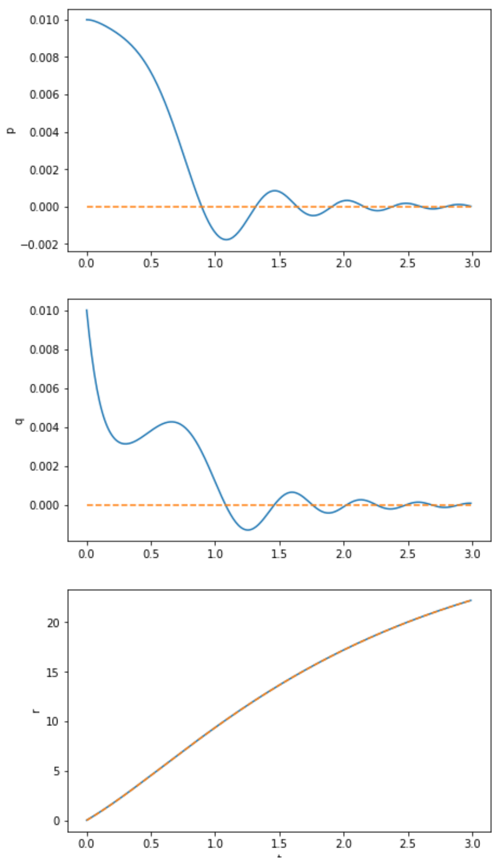

Рис. 5: Стабилизация угловой скорости на программном движении.

Смоделируем поднятие тяги выше тяги дрейфа и посмотрим на поведение угловой скорости в этом процессе. Изменение угловой скорости пропеллера на такой программном движении изображено на рис. (6)

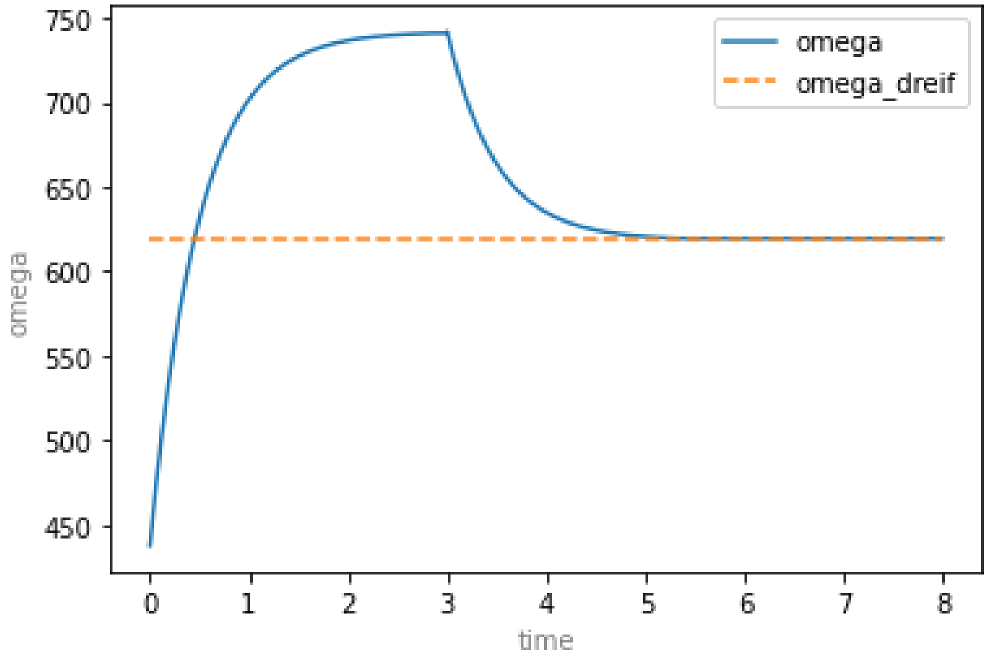

Рис. 6: Изменение угловой скорости пропеллера на программном движении с отскоком.

Резкий отскок от максимальной угловой скорости пропеллера при моделировании оправдан физически, при подаче другого напряжения на ротор разгон сначала происходит резко и замедляется при подходе скорости вращения к соответствующей ей подаваемой мощности тока. Это определяется магнитным моментом ротора, который определяется из конфигурации мотора, установленных на квадрокоптере. Магнитный момент зависит от количества и качества магнитов на роторе, сечения проводов на статоре, магнитными свойствами материалов и т.д. и т.п.

Отработка линейно-квадратичного регулятора в этом процессе показано на рис.(7)

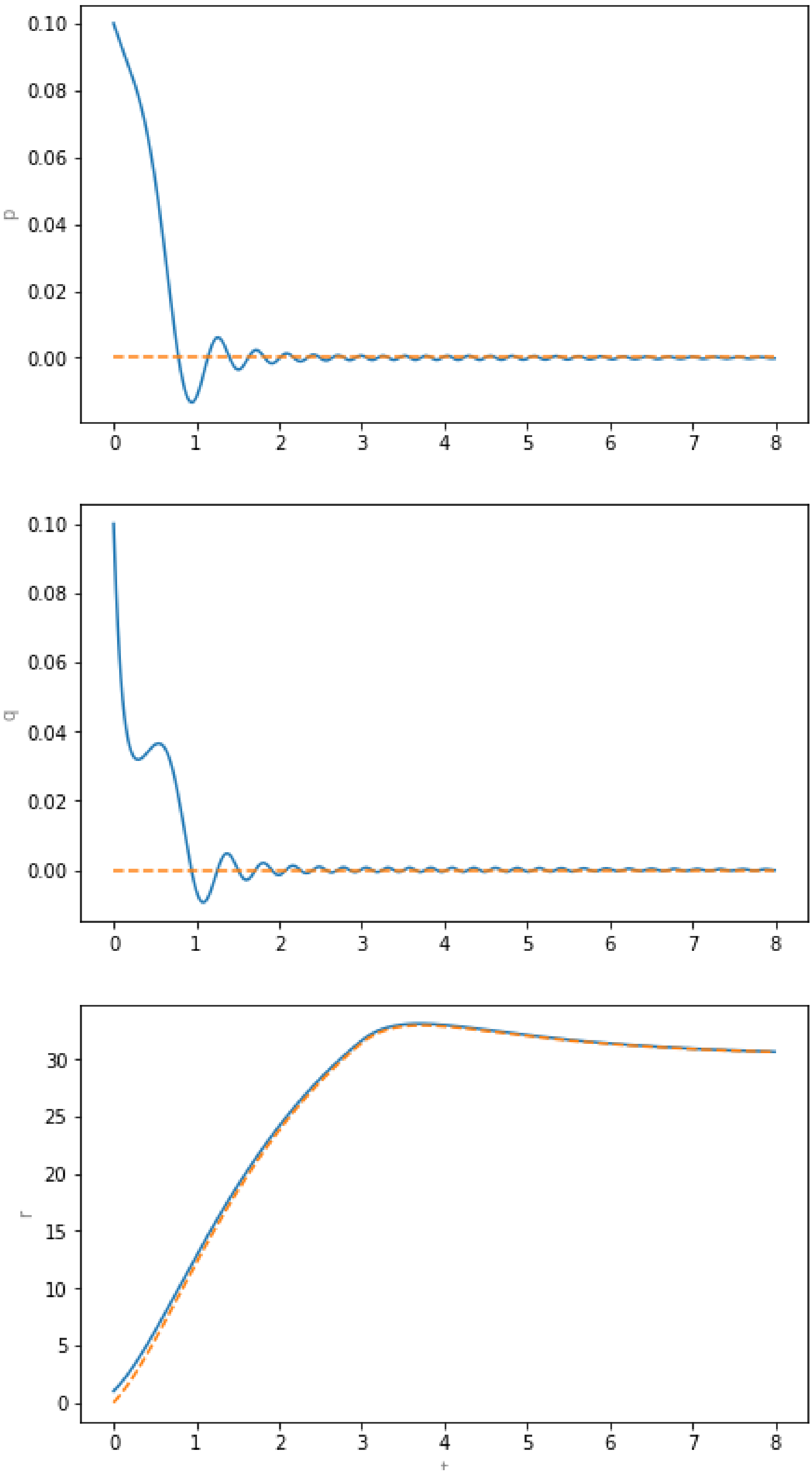

Рис. 7: Стабилизация угловой скорости на программном движении с отскоком.

Опытным путём было определено что оптимальные параметры lqr (28) равны:

$$Rlqr = 60, Qlqr = \begin{pmatrix} 0.01 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0.01 \end{pmatrix}$$

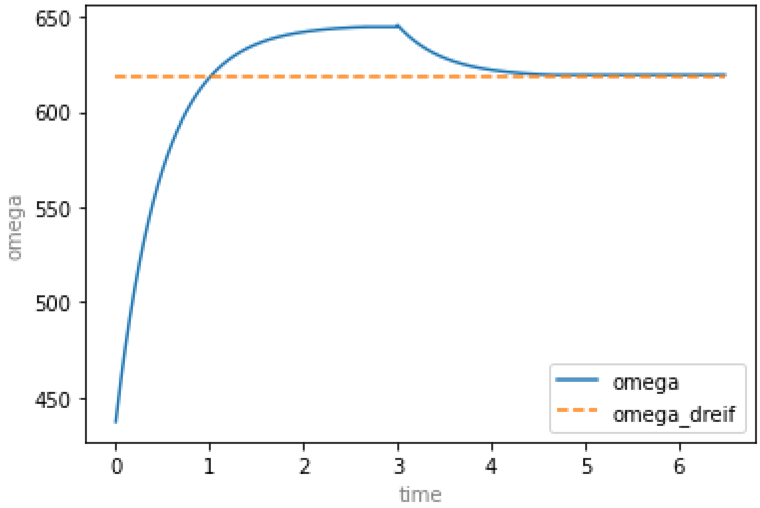

Рис. 8: Изменение угловой скорости пропеллера на программном движении с торможением.

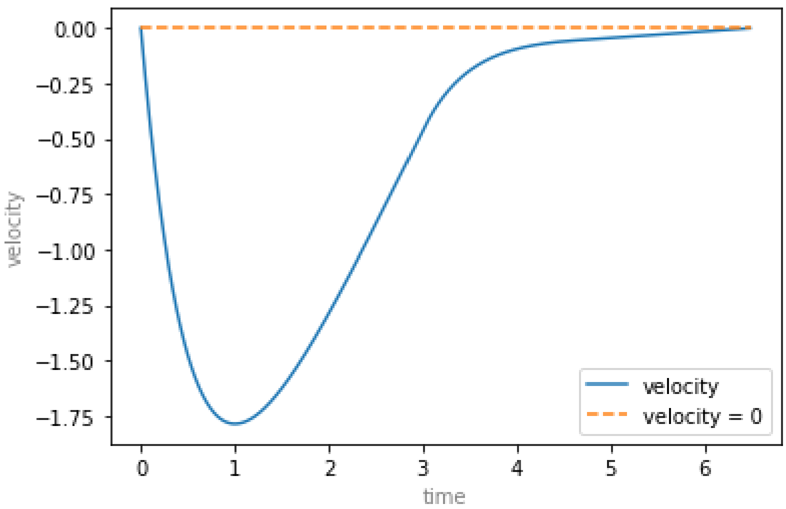

Рис. 9: Абсолютная скорость на программном движении с торможением.

Обеспечении режима торможения по угловой скорости путём изначального достижения большей тяги, подобранной эмпирически, а затем тяги дрейфа показано на рис.(8) и рис. (9).

Небольшое положительное ускорение после достижения тяги дрейфа на рис. (9) обусловлено накоплением ошибки интегрирования, так как для этого во всех случаях использовался простейший метод, Эйлера. Однако общих тенденций к торможению это не меняет, и описанные в данной работе методы можно считать подтверждёнными.

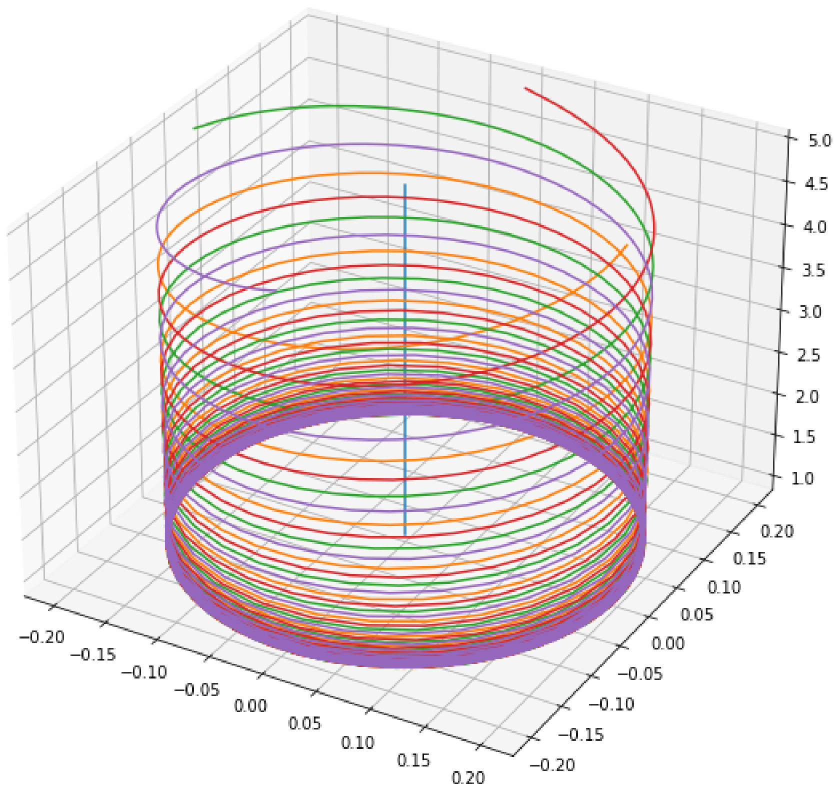

Рис. 10: Движение основных точек во время стабилизации.

Движение центра тела коптера и точек крепления пропеллера во время стабилизации изображено на рис. (10). На нём наглядно показано что происходит торможение и зависание на высоте равной примерно 1.0.

Код для моделирования был написан на Python 3.10, приложен в дополнительной главе "Приложение"

Заключение

В ходе исследования были решены следующие задачи:

- в главе 2 построена общая математическая модель движения квадрокоптера;
- в главе 3 построено программное движение для выхода из некоторых аварийных ситуаций;
- в главе 3 построено управление на программном движении, учитывающее отклонения от программной кривой;
- в главе 4 численно промоделировано полученное решение.

В работе был рассмотрен возможный выход из всех нефатальных ситуаций (когда система оставалась управляемой) путём превращения коптера в биспинер и дрейфа на ненулевой высоте с учётом задержки в управлении тягой пропеллеров.

Список литературы

[1] Ю.В. Морозов, «Экстренное управление квадрокоптером при отказе двух симметричных винтов», Автоматика и телемеханика № 3, 2018, 19 с.

[2] Zulu A., John S. «A Review of Control Algorithms for Autonomous Quadrotors»Open Journal of Applied Sciences, 2014, 4, pp. 547-556

[3] Ranjbaran M., Khorasani K. «Fault recovery of an underactuated quadrotor aerial vehicle»49th IEEE Conf. Decision Control (CDC-2010). Atlanta, 2010. P. 4385–4392.

[4] Mark W. Mueller and Raffaello D'Andrea «Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers»2014 IEEE International Conference on Robotics & Automation (ICRA), May 31–June 7, Hong Kong(China), 2014. P. 45–52.

[5] Y. Zhang, A. Chamseddine, C. Rabbath, B. Gordon, C.-Y. Su, S. Rakheja, C. Fulford, J. Apkarian, and P. Gosselin, »Development of advanced FDD and FTC, techniques with application to an unmanned quadrotor helicopter testbed», Journal of the Franklin Institute, 2013. V. 350. No. 9. P. 2396–2422.

[6] Freddi A., Lanzon A., Longhi S. «A feedback linearization approach to fault tolerance in quadrotor vehicles»IFAC World Congr. 2011. V. 44. No. 1. P. 5413–5418.

[7] Lippiello V., Ruggiero F., Serra D. «Emergency landing for a quadrotor in case ofa propeller failure: A backstepping approach»IEEE/RSJ Int. Conf. Intelligent Robots Syst. 2014. P. 4782–4788.

[8] R. Mahony, V. Kumar, and P. Corke, «Aerial vehicles: Modeling, estimation, and control of quadrotor»IEEE robotics & automation magazine, 2012, vol. 19, no. 3, pp. 20–32.

[9] Minh Duc Hua. «Contributions to the automatic control of aerial vehicles.»Automatic. Université Nice Sophia Antipolis, 2009. 196 p.

[10] S. Bouabdallah and R. Siegwart, «Full control of a quadrotor»in Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on. Ieee, 2007, pp. 153–158.

[11] Wei Dong, Guo-Ying Gu, Xiangyang Zhu, Han Ding «Modeling and Control of a Quadrotor UAV with Aerodynamic Concepts »International Journal of Mechanical, Aerospace, Industrial, Mechatronic and Manufacturing Engineering Vol:7 No: 5, 2013, 6 p.

[12] Xu, R. and Ozguner, U. «Sliding mode control of a class of underactuated systems.»Automatica, 2008, 44:233–241.

[13] Гэн К., Чулин Н.А. «Алгоритмы стабилизации для автоматического управления траекторным движением квадрокоптера»Наука и Образование МГТУ им Н.Э. Баумана. 2015. № 05 с.218-235.

[14] Kwakernaak, Huibert & Sivan, Raphael «Linear Optimal Control Systems. First Edition.»Wiley-Interscience. 1972

[15] Борисов О.И., Громов В.С., Пыркин А.А., «Методы управления робототехническими приложениями. Учебное пособие». СПб.: Уни-верситет ИТМО, 2016, 108 с.

[16] Boyd S. «EE363: Linear Dynamical Systems »Lecture. Stanford Univer., Winter Quarter 2008-09.

[17] Пименов В.Г., Ложников А.Б. «Численные методы. Часть 2.»Издательство Уральского университета, 2014

[18] Бабаджанянц Л.К., Пупышев Ю.А., Пупышева Ю.Ю. «Классическая механика.»Учебное пособие. Издание третье, исправленное. СПб, 2013. 259с.

[19] Д.В. Шиманчук«Введение в современную робототехнику»2018, 203 с.

[20] Д.В. Шиманчук, «ТЕОРЕТИЧЕСКАЯ МЕХАНИКА кинематика твёрдого тела», 2021, 90с.

Приложение

```python
from numpy import arange, sqrt, cos, sin, tan

kf = 6.41 * (10 ** -6)
km = 1.69 * (10 ** -2)
gamma = 2.75 * (10 ** -3)

# force
def f(w):
    return kf * (w ** 2)

radvec0 = [0, 0, 12]
l = 0.2
m = 0.5
g = 9.81
omega0 = [sqrt(m*g/(4*kf)), 0, sqrt(m*g/(4*kf)), 0]
omegadreif = [sqrt(m*g/(2*kf)), 0, sqrt(m*g/(2*kf)), 0]


# momentums of inertia
IQx = 3.2 * (10 ** -3)
IQz = 5.5 * (10 ** -3)
IPx = 1.5 * (10 ** -5)
IPz = 1.5 * (10 ** -9)
IBx = IQx - 4 * IPx
IBz = IQz - 4 * IPz

kr = gamma / IBz

h = 0.01                    # step
Tw = 0.5                    # delay
T = 3                       # end time
```

```
t = arange(0, T, h)

# diff of angular velocity
def domega(start, end):
    if abs(end - start) <= 1:
        return 0
    else:
        return (end - start) / Tw

# array of angular velocites
def omega_gen(omega0, omegaend, t):
    omega = [omega0[0]]
    for i in range(len(t) - 1):
        omega.append(omega[-1] + h * domega(omega[-1],
                                 omegaend[0]))
    return omega

omega = list(omega_gen(omega0, omegadreif, t))

print(omega[0], omega[-1])

#%matplotlib inline
import matplotlib.pyplot as plt

plt.plot(t, omega)

plt.plot(t, [omegadreif[0] for _t in t], "--")
plt.legend(['omega', 'omega for dreif'])
plt.xlabel('time', color='gray')
plt.ylabel('omega',color='gray')
plt.show()

from numpy import matmul
```

```python
from numpy import array as arr

from numpy import cross

def dpqr(w, p, q, r, u=list([0, 0])):
    return [(- (IQz - IQx) * q * r -
                      IPz * q * (w[0] + w[1])) / IBx,
                  ((f(w[0]) - f(w[1])) * l +
                      (IQz - IQx) * p * r +
                      IPz * p * (w[0] + w[1]) + u[1]) / IBx,
                  (IPz * w[0] + IPz * w[1] - gamma * r +
                      km * (f(w[0]) + f(w[1]))) / IBz]

# angular velocity on axes
def pqr_gen(omega, pqr0, t):
    pqr = [pqr0]
    for i in range(len(t) - 1):
        pqr.append([pqr[-1][0] + h * dpqr([omega[i+1], omega[i+1]], pqr[-1][0], pqr[-1][1], pqr[-1][2])[0],
                    pqr[-1][1] + h * dpqr([omega[i+1], omega[i+1]], pqr[-1][0], pqr[-1][1], pqr[-1][2])[1],
                    pqr[-1][2] + h * dpqr([omega[i+1], omega[i+1]], pqr[-1][0], pqr[-1][1], pqr[-1][2])[2]])
    return pqr

pqr = pqr_gen(omega, [0, 0, 0], t)

p = [item[0] for item in pqr]
q = [item[1] for item in pqr]
r = [item[2] for item in pqr]

# orientation matrix
def R(psi, tetta, phi):
```

```
    Rpsi = arr([[cos(psi), -sin(psi), 0],
                [sin(psi),  cos(psi), 0],
                [       0,         0, 1]])
    Rtetta = arr([[1,             0,             0],
                  [0, cos(tetta), -sin(tetta)],
                  [0, sin(tetta),  cos(tetta)]])
    Rphi = arr([[cos(phi), -sin(phi), 0],
                [sin(phi),  cos(phi), 0],
                [       0,         0, 1]])
    return matmul(matmul(Rpsi, Rtetta), Rphi)

# diff angles of Euler
def dangles(psi, tetta, phi, p, q, r):
    dpsi   = (p*sin(phi) + q*cos(phi)) / sin(tetta)
             if tetta != 0 else 0
    dtetta = p*cos(phi) - q*sin(phi)
    dphi   = r - (p*sin(phi) - q*cos(phi)) / tan(tetta)
             if tetta != 0 else r
    return dpsi, dtetta, dphi

angles = [arr([0, 0, 0])]

R0 = arr([[1, 0, 0],
          [0, 1, 0],
          [0, 0, 1],])

def R_gen(angles, p, q, r):
    Rar = [R0]
    for i in range(len(t) - 1):
        angles.append(angles[-1] + h *
            arr(dangles(angles[-1][0], angles[-1][1],
                        angles[-1][2],
                        p[i+1], q[i+1], r[i+1])))
```

```
            Rar.append(R(angles[-1][0], angles[-1][1],
                         angles[-1][2]))
    return Rar

Rar = R_gen(angles, p, q, r)

# 2nd Newton law
def d2d(R, w1, w3):
    result = [R[i][2] * (f(w1) + f(w3)) / m
                 for i in range(3)]
    result[2] -= g
    return result

def d_v(omega):

    vd = [arr([0, 0, 0])]
    for i in range(len(t) - 1):
        vd.append(vd[-1] + arr([h, h, h]) *
                  arr(d2d(Rar[i+1], omega[i+1],
                              omega[i+1])))


    d = [[0, 0, 5]]
    for i in range(len(t) - 1):
        d.append(arr(d[-1]) + arr([h, h, h]) * vd[i+1])
        if d[-1][2] < 0:
            d[-1][2] = 0

    return d, vd

d_current, v = d_v(omega)
plt.plot(t, [item[2] for item in v])
plt.show()
```

```
from control import lqr

def A(r, w):
    a = ((IQx - IQz) * r - IPz * (w[0] + w[1])) / IBx
    return [[ 0, a, 0, 0],
            [-a, 0, 0, 0],
            [ 0,-1, 0, r],
            [ 1, 0,-r, 0]]

B = [[0], [1 / IBx], [0], [0]]

Rlqr = 100
Qlqr = [[0.01, 0, 0, 0],
        [0, 0.01, 0, 0],
        [0, 0, 0.01, 0],
        [0, 0, 0, 0.01]]

def gen_lqr(omega, pqr0, t):
        _pqr = [pqr0]
        for i in range(len(t) - 1):
                K, S, E = lqr(A(pqr[i][2],
                                        [omega[i], omega[i]]),
                                      B, Qlqr, Rlqr)
                u = [K[0][j] * (pqr[i][j] - _pqr[-1][j])
                        for j in range(2)]
                _pqr.append([_pqr[-1][0] + h *
                                    dpqr([omega[i+1],
                                          omega[i+1]],
                                          _pqr[-1][0],
                                          _pqr[-1][1],
                                          _pqr[-1][2], u)[0],
                                    _pqr[-1][1] + h *
```

```
                                        dpqr([omega[i+1],
                                              omega[i+1]],
                                              _pqr[-1][0],
                                              _pqr[-1][1],
                                              _pqr[-1][2], u)[1],
                                       _pqr[-1][2] + h *
                                        dpqr([omega[i+1],
                                              omega[i+1]],
                                              _pqr[-1][0],
                                              _pqr[-1][1],
                                              _pqr[-1][2], u)[2]])
        return _pqr

_pqr = gen_lqr(omega, [0.01, 0.01, 0.01], t)

print(_pqr[-1], pqr[-10])

fig, (ax1, ax2, ax3) = plt.subplots(3, 1, sharey=False,
    figsize=(7,14))
p = [item[0] for item in pqr]
_p = [item[0] for item in _pqr]
ax1.plot(t, _p, '-', t, p, '--')
ax1.set_ylabel('p')
#
q = [item[1] for item in pqr]
_q = [item[1] for item in _pqr]
ax2.plot(t, _q, '-', t, q, '--')
ax2.set_ylabel('q')
#ax2.plot(t, q)

r = [item[2] for item in pqr]
_r = [item[2] for item in _pqr]
ax3.plot(t, _r, '-', t, r, '--')
```

```
ax3.set_ylabel('r')
ax3.set_xlabel('t')
#ax3.plot(t, r)
plt.show()

plt.plot(t, [(Rar[i][2] * 2 * f(omega[i])/m -
              [0, 0, g])[2]
              for i in range(len(t))])
plt.show()
d_current, v = d_v(omega)

plt.plot(t, [item[2] for item in v])
plt.show()

fig, (ax1, ax2, ax3) = plt.subplots(3, 1, sharey=False,
    figsize=(5,5))
ax1.plot(t, [d_current[i][0]
              for i in range((len(d_current)))])
ax1.set_xlabel('t')
ax1.set_ylabel('x_i')
ax2.plot(t, [d_current[i][1]
              for i in range((len(d_current)))])
ax2.set_xlabel('t')
ax2.set_ylabel('y_i')
ax3.plot(t, [d_current[i][2]
              for i in range((len(d_current)))])
plt.show()

from mpl_toolkits.mplot3d import Axes3D

x = [d_current[i][0] for i in range(len(t))]
y = [d_current[i][1] for i in range(len(t))]
z = [d_current[i][2] for i in range(len(t))]
```

```
xi = [arr(matmul(arr([1, 0, 0]), Rar[i]))[0] + arr(x[i])
        for i in range(len(x))]
yi = [arr(matmul(arr([1, 0, 0]), Rar[i]))[1] + arr(y[i])
        for i in range(len(x))]

fig, (ax1, ax2) = plt.subplots(2, 1, sharey=False,
    figsize=(5,5))
ax1.plot(t, xi)
#ax1.set_ylim(0, w0[0] * 2)
ax1.set_xlabel('t')
ax1.set_ylabel('x_i')
ax2.plot(t, yi)
ax2.set_xlabel('t')
ax2.set_ylabel('y_i')

x1 = [arr(matmul(arr([1, 0, 0]), Rar[i]))[0] + arr(x[i])
        for i in range(len(t))]
y1 = [arr(matmul(arr([1, 0, 0]), Rar[i]))[1] + arr(y[i])
        for i in range(len(t))]
z1 = [arr(matmul(arr([1, 0, 0]), Rar[i]))[2] + arr(z[i])
        for i in range(len(t))]
x2 = [arr(matmul(arr([-1, 0, 0]), Rar[i]))[0] + arr(x[i])
        for i in range(len(t))]
y2 = [arr(matmul(arr([-1, 0, 0]), Rar[i]))[1] + arr(y[i])
        for i in range(len(t))]
z2 = [arr(matmul(arr([-1, 0, 0]), Rar[i]))[2] + arr(z[i])
        for i in range(len(t))]
x3 = [arr(matmul(arr([0, 1, 0]), Rar[i]))[0] + arr(x[i])
        for i in range(len(t))]
y3 = [arr(matmul(arr([0, 1, 0]), Rar[i]))[1] + arr(y[i])
        for i in range(len(t))]
z3 = [arr(matmul(arr([0, 1, 0]), Rar[i]))[2] + arr(z[i])
```

```
        for i in range(len(t))]
x4 = [arr(matmul(arr([0, -1, 0]), Rar[i]))[0] + arr(x[i])
        for i in range(len(t))]
y4 = [arr(matmul(arr([0, -1, 0]), Rar[i]))[1] + arr(y[i])
        for i in range(len(t))]
z4 = [arr(matmul(arr([0, -1, 0]), Rar[i]))[2] + arr(z[i])
        for i in range(len(t))]

fig = plt.figure(figsize=(30,10))
ax = fig.add_subplot(111, projection='3d')
ax.plot(x, y, z, label='parametric curve')
ax.plot(x1, y1, z1, label='parametric curve')
ax.plot(x2, y2, z2, label='parametric curve')
ax.plot(x3, y3, z3, label='parametric curve')
ax.plot(x4, y4, z4, label='parametric curve')
plt.show()

t_temp = arange(T, T + 3.5, h)

v_0 = v[-1][2]
print(v_0)

om1 = omega_gen(omega0, 1.0439 * arr(omegadreif), t)
om1 = list(om1) + list(omega_gen(1.0439 * arr(omegadreif)
    , 0.9999*arr(omegadreif), t_temp))
plt.plot(list(t)+ list(t_temp), om1)
plt.plot(list(t)+ list(t_temp), [omegadreif[0] for _t in
    (list(t)+ list(t_temp))], '--')
plt.legend(['omega', 'omega_dreif'])
plt.xlabel('time', color = 'gray')
plt.ylabel('omega', color = 'gray')
plt.show()
```

```
t = list(t)+ list(t_temp)

Rlqr = 60
Qlqr = [[0.01, 0, 0, 0],
        [0, 0.01, 0, 0],
        [0, 0, 0.01, 0],
        [0, 0, 0, 0.01]]

pqr = pqr_gen(om1, [0, 0, 0], t)
_pqr = gen_lqr(om1, [0.1, 0.1, 1], t)

fig, (ax1, ax2, ax3) = plt.subplots(3, 1, sharey=False, figsize=(7,14))
p = [item[0] for item in pqr]
_p = [item[0] for item in _pqr]
ax1.plot(t, _p, '-', t, p, '--')
ax1.set_ylabel('p', color = 'grey')
#
q = [item[1] for item in pqr]
_q = [item[1] for item in _pqr]
ax2.plot(t, _q, '-', t, q, '--')
ax2.set_ylabel('q', color = 'grey')
#ax2.plot(t, q)

r = [item[2] for item in pqr]
_r = [item[2] for item in _pqr]
ax3.plot(t, _r, '-', t, r, '--')
ax3.set_ylabel('r', color = 'grey')
ax3.set_xlabel('t', color = 'grey')
#ax3.plot(t, r)
plt.show()

angles = [arr([0, 0, 0])]
```

```python
Rar = R_gen(angles, p, q, r)

d_current, v = d_v(om1)

print(v[-1])

plt.plot(t, [item[2] for item in v])
plt.plot(t, [0 for _t in t], '--')
plt.legend(['velocity', 'velocity = 0'])
plt.xlabel('time', color = 'gray')
plt.ylabel('velocity', color = 'gray')
plt.show()

x = [d_current[i][0] for i in range(len(t))]
y = [d_current[i][1] for i in range(len(t))]
z = [d_current[i][2] for i in range(len(t))]

x1 = [arr(matmul(arr([1, 0, 0]), Rar[i]))[0] + arr(x[i])
        for i in range(len(t))]
y1 = [arr(matmul(arr([1, 0, 0]), Rar[i]))[1] + arr(y[i])
        for i in range(len(t))]
z1 = [arr(matmul(arr([1, 0, 0]), Rar[i]))[2] + arr(z[i])
        for i in range(len(t))]
x2 = [arr(matmul(arr([-1, 0, 0]), Rar[i]))[0] + arr(x[i])
        for i in range(len(t))]
y2 = [arr(matmul(arr([-1, 0, 0]), Rar[i]))[1] + arr(y[i])
        for i in range(len(t))]
z2 = [arr(matmul(arr([-1, 0, 0]), Rar[i]))[2] + arr(z[i])
        for i in range(len(t))]
x3 = [arr(matmul(arr([0, 1, 0]), Rar[i]))[0] + arr(x[i])
        for i in range(len(t))]
y3 = [arr(matmul(arr([0, 1, 0]), Rar[i]))[1] + arr(y[i])
        for i in range(len(t))]
```

```
z3 = [arr(matmul(arr([0, 1, 0]), Rar[i]))[2] + arr(z[i])
        for i in range(len(t))]
x4 = [arr(matmul(arr([0, -1, 0]), Rar[i]))[0] + arr(x[i])
        for i in range(len(t))]
y4 = [arr(matmul(arr([0, -1, 0]), Rar[i]))[1] + arr(y[i])
        for i in range(len(t))]
z4 = [arr(matmul(arr([0, -1, 0]), Rar[i]))[2] + arr(z[i])
        for i in range(len(t))]

fig = plt.figure(figsize=(30,10))
ax = fig.add_subplot(111, projection='3d')
ax.plot(x, y, z, label='parametric curve')
ax.plot(x1, y1, z1, label='parametric curve')
ax.plot(x2, y2, z2, label='parametric curve')
ax.plot(x3, y3, z3, label='parametric curve')
ax.plot(x4, y4, z4, label='parametric curve')
plt.show()
```