Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет Систем Управления и Робототехники

Направление подготовки:

15.04.06 Мехатроника и робототехника

КУРСОВАЯ РАБОТА

по дисциплине: Моделирование и управление движением роботов

по теме: *Планирование траектории квадрокоптера методом случайного дерева и управление с помощью контроллера, использующего метод бэкстеппинга*

Выполнили студенты Веснин М.А.
Миргазов Э.Р.
Топольницкий А.А.

Преподаватель Колюбин С.А.

	Подпись преподавателя	Дата
Защита		

Санкт-Петербург

2023 г.

Оглавление

Введение

В современном мире квадрокоптеры находят всё бóльшее применение, поскольку могут выполнять целый ряд задач: профилактика и ликвидация ЧС, обеспечение обороны и национальной безопасности объектов промышленности, сельского хозяйства и продовольствия, природных ресурсов, а также служб мониторинга, длительного авиационного патрулирования земной и водной поверхностей. Помимо этого, данный тип роботов можно применять для задач поиска пропавших людей или в задачах противодействия преступникам [1].

Для успешного выполнения описанных выше задач необходимо три элемента: сам квадрокоптер, траектория, по которой должен двигаться робот, и система управления, обеспечивающая движение по спланированной траектории с высокой точностью. Существует много различных систем управления для квадрокоптеров – как линейные системы, так и нелинейные. Классическим примером является ПД-регулятор, однако это не единственный подход, как можно реализовать управление.

Целью данной работы является планирование траектории квадрокоптера с помощью метода случайного дерева и синтез системы управления для поддержания движения по заданной траектории. Для выполнения поставленной цели необходимо решить несколько задач:

1. Выполнить кинематический анализ квадрокоптера;
2. Построить динамическую модель данного робота;
3. Реализовать планирование траектории движения;
4. Синтезировать регулятор для управления движением робота;
5. Провести численное моделирование системы;
6. Сравнить результаты применяемой системы управления с классическим ПД-регулятором.

Глава 1. Кинематический анализ и динамическая модель квадрокоптера

Глава 1.1. Кинематический анализ квадрокоптера

Поведение квадрокоптера можно описать тремя углами:

1. φ – угол крена – угол вокруг оси Ox в инерциальной СК;
2. θ – угол тангажа – угол вокруг оси Oy в инерциальной СК;
3. ψ – угол рыскания – угол вокруг оси Oz в инерциальной СК;

В работе планируется использовать квадрокоптер из библиотеки для MATLAB под названием Robotics Toolbox за авторством Питера Корка, поэтому возьмём конфигурационную схему возьмём из книги автора [2]. Она представлена ниже. Можно заметить, что ось Oz имеет направление внизу, пропеллеры 1 и 3 вращаются против часовой стрелки, 2 и 4 по часовой.

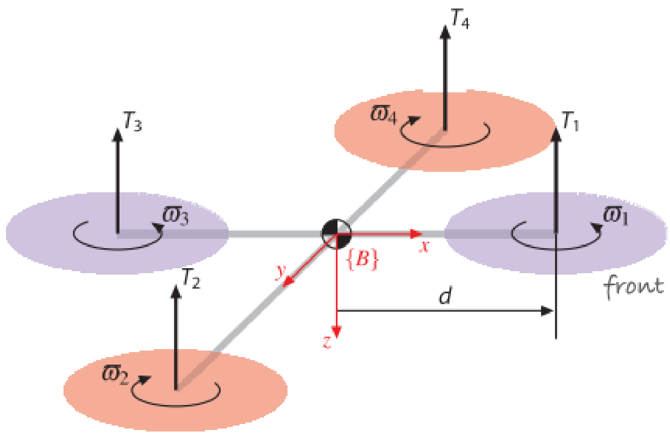

Рисунок 1. Конфигурация рассматриваемого квадрокоптера

Существует несколько способов получения матрицы поворота, необходимой для связи инерциальной системы координат и системы координат квадрокоптера. Чтобы получить в нашем случае подобную матрицу, необходимо сначала ввести три матрицы вращения вокруг осей:

$$R_\psi = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad R_\theta = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \qquad (1)$$

$$R_{\varphi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi) & -\sin(\varphi) \\ 0 & \sin(\varphi) & \cos(\varphi) \end{bmatrix} \tag{2}$$

Таким образом, результирующая матрица вращения представляет собой преобразование с использованием углов Эйлера инерциальной системы координат в систему координат тела:

$$\begin{bmatrix} x_{\text{тела}} \\ y_{\text{тела}} \\ z_{\text{тела}} \end{bmatrix} = R_{\psi} R_{\theta} R_{\varphi} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \tag{3}$$

Если перемножить матрицы R_i, то получим переход в систему координат тела. Если применить операцию транспонирования к матрице, то будет получена итоговая матрица:

$$R_{\text{тело}}^{0} = \begin{bmatrix} \mathrm{c}(\psi)\,\mathrm{c}(\theta) & \mathrm{s}(\varphi)\,\mathrm{s}(\theta)\,\mathrm{c}(\psi) - \mathrm{c}(\varphi)\,\mathrm{s}(\psi) & \mathrm{c}(\varphi)\,\mathrm{s}(\theta)\,\mathrm{c}(\psi) + s(\varphi)\,\mathrm{s}(\psi) \\ \mathrm{c}(\theta)\,\mathrm{s}(\psi) & \mathrm{s}(\varphi)\,\mathrm{s}(\theta)\,\mathrm{s}(\psi) + \mathrm{c}(\varphi)\,\mathrm{c}(\psi) & \mathrm{c}(\varphi)\,\mathrm{s}(\theta)\,\mathrm{s}(\psi) - s(\varphi)\,\mathrm{c}(\psi) \\ -s(\theta) & \mathrm{c}(\theta)\,\mathrm{s}(\varphi) & \mathrm{c}(\theta)\,\mathrm{c}(\varphi) \end{bmatrix} \tag{4}$$

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} = R_{\text{тело}}^{0} \begin{bmatrix} x_{\text{тела}} \\ y_{\text{тела}} \\ z_{\text{тела}} \end{bmatrix} \tag{5}$$

Угловые скорости от углов Эйлера связаны с системой координат тела следующим выражением [3]:

$$\Omega_0^{\text{тело}} = \dot{\varphi} + R_{\varphi}\dot{\theta} + R_{\varphi}R_{\theta}\dot{\psi} \tag{6}$$

Глава 1.2. Динамическая модель квадрокоптера

Для описания динамики квадрокоптера приняты следующие допущения [4],[5]:

1. Квадрокоптер представляет собой твёрдое тело с симметричной структурой;
2. Пропеллеры являются твёрдыми телами;
3. Центр масс совпадает с центом давления;
4. Сопротивлением воздуха пренебрегается.

С помощью уравнений Ньютона – Эйлера можно записать динамическую модель квадрокоптера в общем виде:

$$m\dot{V} = \sum F \tag{7}$$

$$J\dot{\omega} = -\omega \times J\omega + \tau, \tag{8}$$

где τ – суммарный момент, описанный ниже; J – матрица инерции 3 х 3, ω – вектор угловой скорости.

Рассмотрим моменты. Каждым пропеллером создаётся подъёмная сила, выражаемая формулой:

$$T_i = b\varpi_i^2, i = 1,2,3,4, \tag{9}$$

где $b > 0$ – константа, зависящая от плотности воздуха, куба радиуса лезвия пропеллера, количества пропеллеров и длины хорды лезвия; ϖ_i – скорость вращения пропеллера. T – суммарная тяга.

Далее, для момента по каждой оси можно получить следующие формулы. Знаки в формулах объясняются направлениями вращения пропеллеров, схема приведена была выше:

$$\begin{cases} \tau_x = db(\varpi_4^2 - \varpi_2^2) \\ \tau_y = db(\varpi_1^2 - \varpi_3^2) \\ \tau_z = k(\varpi_1^2 + \varpi_3^2 - \varpi_4^2 - \varpi_2^2) \end{cases}, \tag{10}$$

где d – расстояние от центра масс до оси вращения пропеллера, k – коэффициент, зависящий от тех же факторов, что и b. Можно заметить, что управление по рысканию можно осуществлять, управляя скоростями вращения пропеллеров.

Теперь вернёмся к уравнениям (7) и (8) и распишем их чуточку подробнее.

$$m\dot{V} = mg * \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} - R_{\text{тело}}^0 \begin{bmatrix} 0 \\ 0 \\ k(\varpi_1^2 + \varpi_3^2 + \varpi_4^2 + \varpi_2^2) \end{bmatrix} \tag{11}$$

$$\dot{V} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} -\frac{T}{m}(\cos(\varphi)\sin(\theta)\cos(\psi) + \sin(\varphi)\sin(\psi)) \\ -\frac{T}{m}(\cos(\varphi)\sin(\theta)\sin(\psi) - \sin(\varphi)\cos(\psi)) \\ -\frac{T}{m}\left(\cos(\varphi)\cos(\theta)\right) + g \end{bmatrix} \tag{12}$$

$$J\dot{\omega} = -\omega \times J\omega + \tau \ = \ M_{propellers} - \ M_{Gyro}, \tag{13}$$

где $M_{propellers}$ – моменты, создаваемые пропеллерами по осям, по сути, это сумма $\tau_x + \tau_y + \tau_z$. Для учёта результирующего гироскопического эффекта используется формула:

$$M_{Gyro} = \begin{pmatrix} \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} \wedge J_r \begin{pmatrix} 0 \\ 0 \\ \sum_{i=1}^{4} (-1)^{i+1} \varpi_i^2 \end{pmatrix} = J_r \Omega \begin{pmatrix} \dot{\theta} \\ -\dot{\varphi} \\ 0 \end{pmatrix}, \tag{14}$$

где $\Omega = \ \omega_1 - \omega_2 + \omega_3 - \omega_4, J_r$ – момент инерции ротора двигателя.

Для расчёта центростремительной силы применяется формула:

$$\mathrm{w} \wedge \mathrm{I}\ \mathrm{w} = \begin{pmatrix} \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} \wedge \begin{pmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{pmatrix} \begin{pmatrix} \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} \dot{\psi}\dot{\theta}\left(I_z - I_y\right) \\ \dot{\psi}\dot{\varphi}(I_x - I_z) \\ \dot{\varphi}\dot{\theta}\left(I_y - I_x\right) \end{pmatrix} \tag{15}$$

Если в уравнение (8) переписать с учётом (14) и (15), то вращательная составляющая динамики будет выглядеть следующим образом:

$$\begin{pmatrix} \ddot{\varphi} \\ \ddot{\theta} \\ \ddot{\psi} \end{pmatrix} = \begin{pmatrix} \dot{\psi}\dot{\theta}\left(\frac{I_y - I_z}{I_x}\right) - \frac{J_r}{I_x}\dot{\theta}\Omega + \frac{\tau_x}{I_x} \\ \dot{\psi}\dot{\varphi}\left(\frac{I_z - I_x}{I_y}\right) + \frac{J_r}{I_y}\dot{\varphi}\Omega + \frac{\tau_y}{I_y} \\ \dot{\varphi}\dot{\theta}\left(\frac{I_x - I_y}{I_z}\right) + \frac{\tau_z}{I_z} \end{pmatrix} \tag{16}$$

Наконец, объединяя линейные и вращательные составляющие в одну систему, получим (20):

$$\begin{pmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\\ \ddot{\varphi}\\ \ddot{\theta}\\ \ddot{\psi}\end{pmatrix} = \begin{pmatrix} -\frac{T}{m}(\cos(\varphi)\sin(\theta)\cos(\psi)+\sin(\varphi)\sin(\psi)) \\ -\frac{T}{m}(\cos(\varphi)\sin(\theta)\sin(\psi)-\sin(\varphi)\cos(\psi)) \\ -\frac{T}{m}\big(\cos(\varphi)\cos(\theta)\big)+g \\ \dot{\psi}\dot{\theta}\left(\frac{I_y-I_z}{I_x}\right)-\frac{J_r}{I_x}\dot{\theta}\Omega+\frac{\tau_x}{I_x} \\ \dot{\psi}\dot{\varphi}\left(\frac{I_z-I_x}{I_y}\right)+\frac{J_r}{I_y}\dot{\varphi}\Omega+\frac{\tau_y}{I_y} \\ \dot{\varphi}\dot{\theta}\left(\frac{I_x-I_y}{I_z}\right)+\frac{\tau_z}{I_z} \end{pmatrix} \tag{17}$$

Глава 2. Система управление квадрокоптером

Для начала введём обозначения, которыми удобно в дальнейшем пользоваться:

$$X^T = \left[\varphi, \dot{\varphi}, \theta, \dot{\theta}, \psi, \dot{\psi}\right]^T = [x_1, x_2, x_3, x_4, x_5, x_6]^T \tag{18}$$

Тогда система (16) будет иметь вид:

$$\dot{X} = \begin{pmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\\ \dot{x}_5\\ \dot{x}_6\end{pmatrix} = \begin{bmatrix} x_2 \\ x_4x_6a_1 - c_1\Omega x_4 + b_1\tau_x \\ x_4 \\ x_2x_6a_2 + c_2\Omega x_2 + b_2\tau_y \\ x_6 \\ x_4x_2a_3 + b_3\tau_z \end{bmatrix}, \tag{19}$$

где

$$a_1 = \frac{I_y - I_z}{I_x} \quad b_1 = \frac{1}{I_x} \quad c_1 = \frac{J_r}{I_x}$$
$$a_2 = \frac{I_z - I_x}{I_y} \quad b_2 = \frac{1}{I_y} \quad c_2 = \frac{J_r}{I_y}$$
$$a_3 = \frac{I_x - I_y}{I_z} \quad b_3 = \frac{1}{I_z} \quad c_3 = \frac{J_r}{I_z}$$

Глава 2.1. Метод бэкстеппинга для управления углами

На примере уравнений для крена рассмотрим порядок применения ***Бэкстеппинга.*** Необходимы два уравнения:

$$\begin{gathered}\dot{x}_1 = x_2 \\ \dot{x}_2 = x_4 x_6 a_1 - c_1 \Omega x_4 + b_1 \tau_x\end{gathered} \tag{20}$$

Данный метод можно удобно разложить на два последующих шага. **Шаг первый** – определяем ошибку, относящуюся x_1 к и его производной:

$$\begin{cases} e_1 = x_{1d} - x_1 \\ \dot{e}_1 = \dot{x}_{1d} - x_2 \end{cases} \tag{21}$$

Возьмём функцию Ляпунова и её производную следующего вида:

$$\begin{cases} V_1(e_1) = \dfrac{1}{2} e_1^2 \\ \dot{V}_1(e_1) = e_1 \dot{e}_1 = e_1(\dot{x}_{1d} - x_2) \end{cases} \tag{22}$$

Чтобы удовлетворять условию устойчивости функции Ляпунова ($\dot{V}_1(e_1) < 0$) вводится новое виртуальное управление x_2 в соответствии с формулой ниже:

$$\begin{gathered} x_2 = \dot{x}_{1d} + k_1 e_1 \\ e_2 = x_2 - x_{2d} = x_2 - \dot{x}_{1d} - k_1 e_1, \end{gathered} \tag{23}$$

где k_1 – положительная константа (коэффициент). Тогда производная функции Ляпунова имеет следующий вид:

$$\dot{V}_1(e_1) = -k_1 e_1^2 \tag{24}$$

Во **втором шаге** вводим новую функцию Ляпунова следующего вида:

$$V_2(e_1, e_2) = \frac{1}{2} e_1^2 + \frac{1}{2} e_2^2 \tag{25}$$

Ниже приведён расчёт производной для этой функции Ляпунова:

$$\dot{V}_2(e_1, e_2) = e_1 \dot{e}_1 + e_2 \dot{e}_2 \tag{26}$$

$$\begin{gathered}\dot{V}_2(e_1, e_2) = e_1\,(-e_2 - k_1 e_1) + e_2(\dot{x}_2 - \ddot{x}_{1d} - k_1 \dot{e}_1) = -e_1 e_2 - k_1 e_1^2 + \\ + e_2 \dot{x}_2 - e_2(\ddot{x}_{1d} - k_1(e_2 + k_1 e_1)\end{gathered} \tag{27}$$

Устойчивость всей системы, системы крена, обеспечивается с помощью второй положительной константы k_2:

$$\tau_x = \frac{1}{b_1}[e_1 - x_4 x_6 a_1 - c_1 \Omega x_4 + \ddot{x}_{1d} - k_1(e_2 + k_1 e_1) - k_2 e_1] \tag{28}$$

И тогда производная функции Ляпунова имеет вид:

$$\dot{V}_2(e_1, e_2) = -k_1 e_1^2 - k_2 e_2^2, \tag{29}$$

что говорит о том, что система асимптотически устойчива.

Используя тот же подход для управления по тангажу и рысканию, можно получить итоговую систему уравнений, обеспечивающую управление по углам Эйлера:

$$\begin{gathered}\tau_x = \frac{1}{b_1}[e_1 - x_4 x_6 a_1 - c_1 \Omega x_4 + \ddot{x}_{1d} - k_1(e_2 + k_1 e_1) - k_2 e_1] \\ \tau_y = \frac{1}{b_2}[e_3 - x_2 x_6 a_2 - c_2 \Omega x_2 + \ddot{x}_{3d} - k_3(e_4 + k_3 e_3) - k_4 e_4] \\ \tau_z = \frac{1}{b_3}[e_5 - x_2 x_4 a_3 + \ddot{x}_{5d} - k_5(e_6 + k_5 e_5) - k_6 e_6]\end{gathered} \qquad (30)$$

При этом:

$$\begin{gathered}e_i = x_{id} - x_i \\ e_{i+1} = x_{i+1} - x_{(i+1)d} = x_{(i+1)} - \dot{x}_{id} - k_i e_i\end{gathered} \qquad (31)$$

где $e_i, i = 1,2,3,4,5,6$; $k_i, i = 1,2,3,4,5,6$ соответственно ошибка и положительные константы функций Ляпунова. Коэффициенты k_i необходимо настраивать.

Глава 2.2. Способ получения значений углов

Поскольку в Robotics Toolbox существует модель с квадрокоптером, можем некоторые моменты взять оттуда. Например, управление тягой осуществляется с помощью ПД регулятора по формуле:

$$T = K_p(z_{des} - z_{act}) + K_d(\dot{z}_{des} - \dot{z}_{act}) + mg \qquad (32)$$

Контроллер, отвечающий за поведение робота, связывает ошибку в координатах x и y в системе координат тела с желаемыми углами Эйлера в инерциальной системе координат.

$$xy^{error\_body} = R_B(\theta_y) \cdot xy^{error\_0}, \qquad (33)$$

$$euler\_angles^0_{des} = K_p(p^b_{des} - p^b_{act}) + K_d(\dot{p}^b_{des} - \dot{p}^b_{act}), \qquad (34)$$

где p — это координаты x, y

Глава 3. Планирование траектории

Как уже упоминалось, планирование является важной задачей при работе с робототехническими системами. Существует много разных способ, рассмотрим некоторые из них [6].

Метод декомпозиции на ячейки. Бывает точным или приблизительным, рассмотрим на примере точной декомпозиции:

1. Всё свободное пространство разделяется на треугольные или трапецеидальные ячейки;
2. Составляются графы связности с вершинами в их центрах и ребрами, соответствующими общим сторонам смежных ячеек;
3. В итоге получаем два типа точек:
    a. тип 1, белые, соответствуют свободному пространству;
    b. тип 2, чёрные, соответствуют запрещённой зоне пространства;
4. Далее определяются вершины графа, которым соответствуют начальное и конечное положения, затем начинается поиск последовательности переходов по белым ячейкам.

Метод вероятностной дорожной карты используется для быстрой генерации пути и основан использовании случайных выборок в пространстве, состоит из четырёх шагов:

1. Осуществляется выбор нескольких узлов случайным образом;
2. Соседние узлы соединяются между собой отрезками, не пересекающими запрещённую зону, при заданной норме в этом пространстве;
3. Первые два шага повторяются, чтобы покрыть достаточно большую область, охватывающую начальное и конечное положение;
4. Выбирается последовательность отрезков, позволяющая выполнить траекторию.

Ещё одним алгоритмом планирования пути, как раз используемым в нашей работе, является алгоритм *двунаправленного быстроисследующего случайного дерева* [7]. Алгоритм работает следующим образом:

1. Задаются начальное и конечное положения робота, в нашем случае квадрокоптера. Начальное положение является первой вершиной дерева;
2. Затем в пространство добавляется случайная точка, на расстоянии, равном заданному шагу алгоритма;
3. Далее происходит проверка – не попала ли точка на наше препятствие. Если проверка пройдена, то вершина и линия между ней и предыдущей точкой добавляются к общему дереву;
4. Если новая случайная и конечная точки совпали, то построении траектории выполнено. В противном случае действия повторяются.

Для того, чтобы траектория, полученная методом случайного дерева, была более гладкой, с помощью функции *optimizePath* была проведена оптимизация пути. Данная функция работает на основе алгоритма Левенберга-Марквардта, о котором можно прочитать здесь [8], и эта функция позволяет задать желаемое расстояние, на котором объект должен держаться от разного рода препятствий.

Глава 4. Результаты численного моделирования

На рисунке ниже приведена построенная карта с препятствиями и маршрутом. Чёрным помечены преграды, красным – спланированный маршрут, полученный с помощью метода быстроисследующего случайного дерева. Синим – ветви дерева. Фиолетовым – оптимизированный вариант траектории. Начальное положение (40,2), конечное – (90,90). Как видно по рисунку, неоптимизированный маршрут хоть и создаёт траекторию от начала до конца, но в некоторых точках проходит вплотную к преградам, что в реальности недопустимо. Оптимизированный маршрут же старается поддерживать расстояние до препятствий не меньше 1 метра.

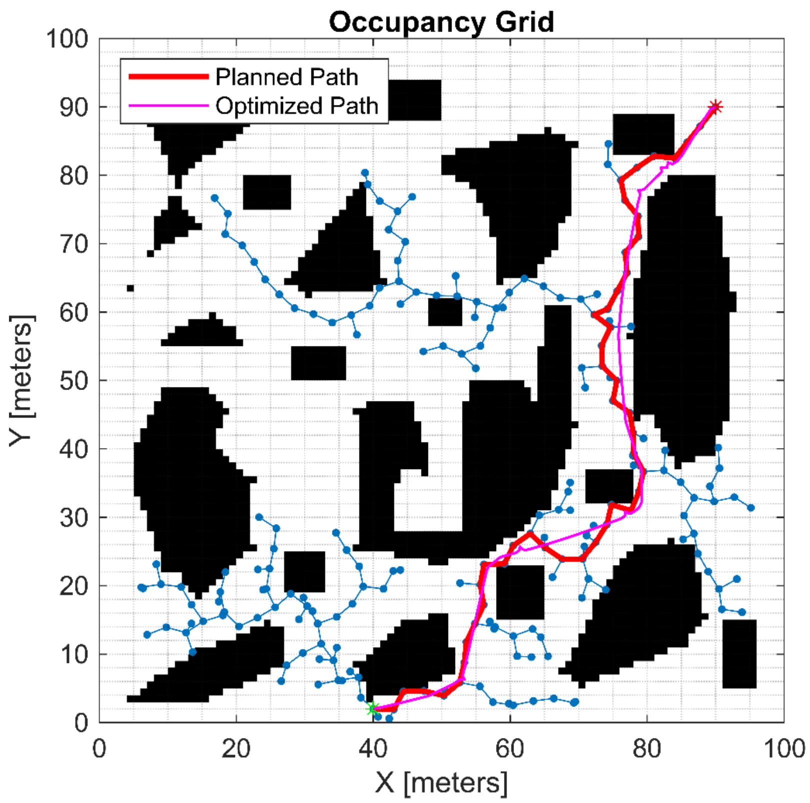

Рисунок 2. Спланированная траектория методом быстроисследуемого случайного дерева

Далее, на основе планирования алгоритма траектории были получены параметры этой траектории – координаты x и y. В дальнейшей работе предполагались следующие параметры:

Таблица 1. Параметры моделирования

Высота полёта	5 метров
Угол рыскания	Поддерживается 0 радиан
Масса квадрокоптера	4 кг
Момент инерции ротора двигателя	6,49e-5 кг * м^2

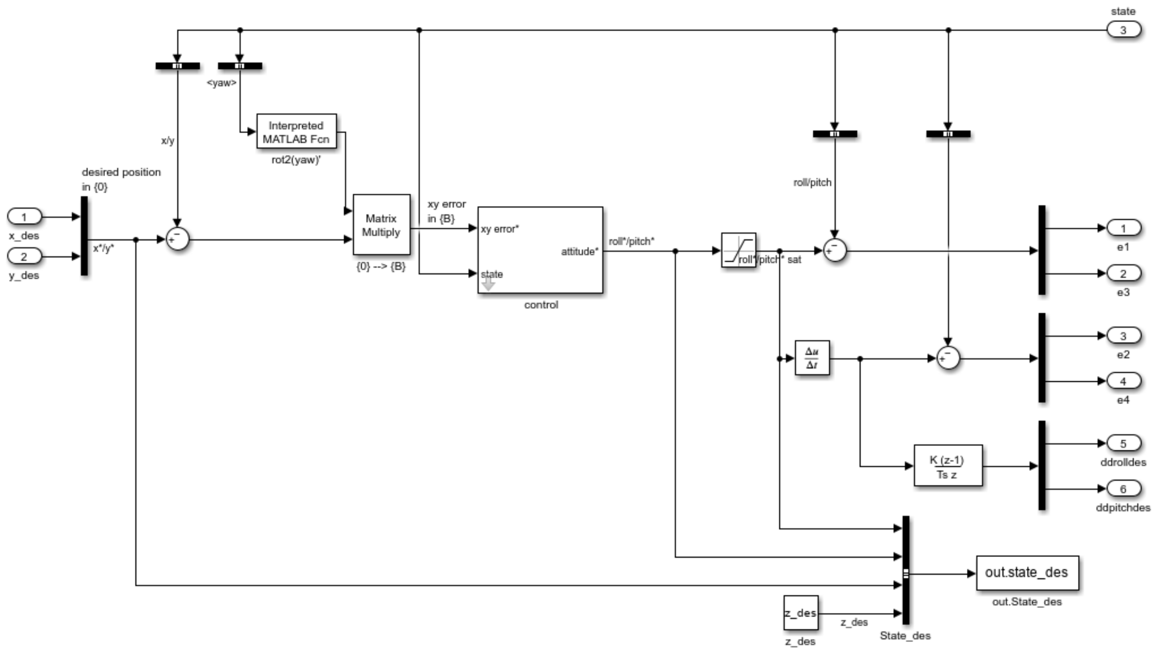

Рисунок 3. Часть внутренней системы модели в Simulink

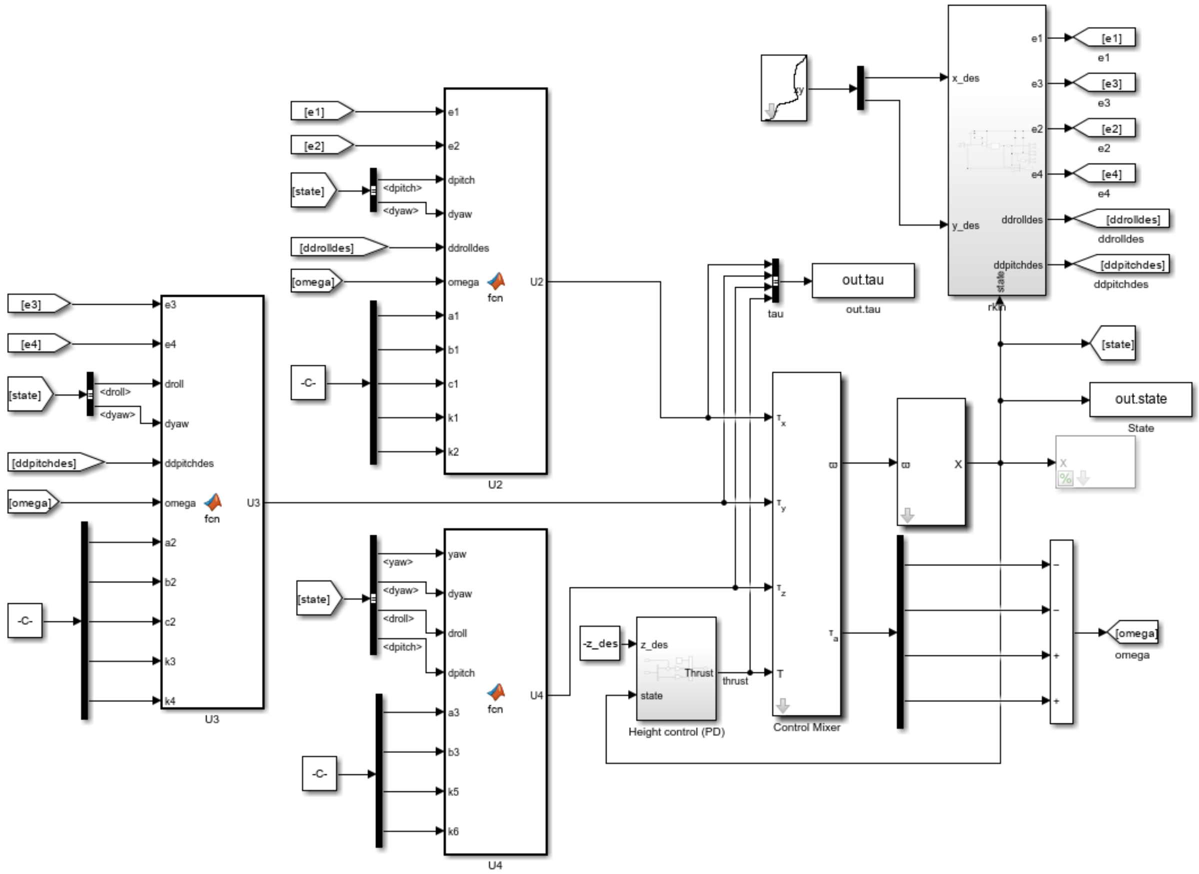

Рисунок 4. Внешняя часть системы управления в Simulink

Для того, чтобы оценить предлагаемый алгоритм, ниже приведены графики, отражающие реальное поведение квадрокоптера в сравнении с желаемым поведением.

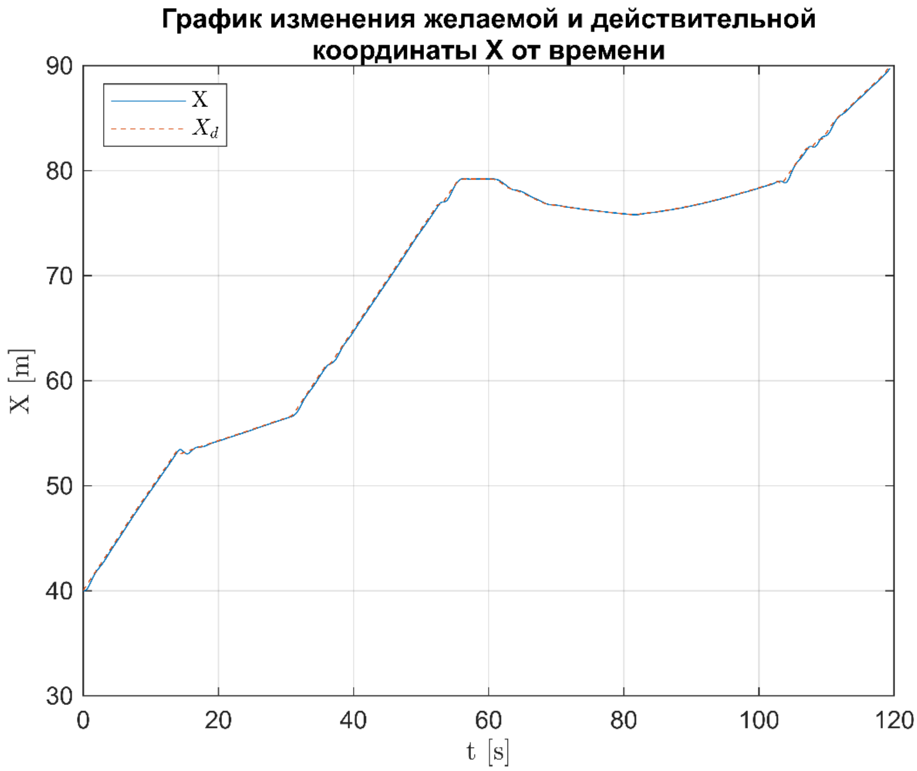

Рисунок 5. График изменения координаты X

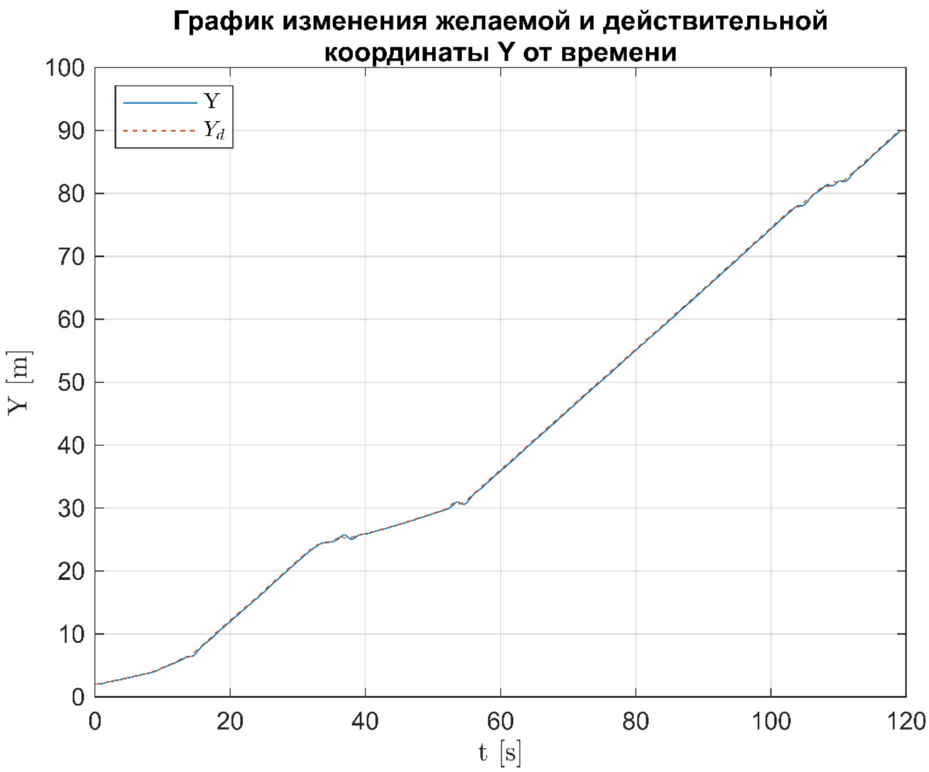

Рисунок 6. График изменения координаты Y

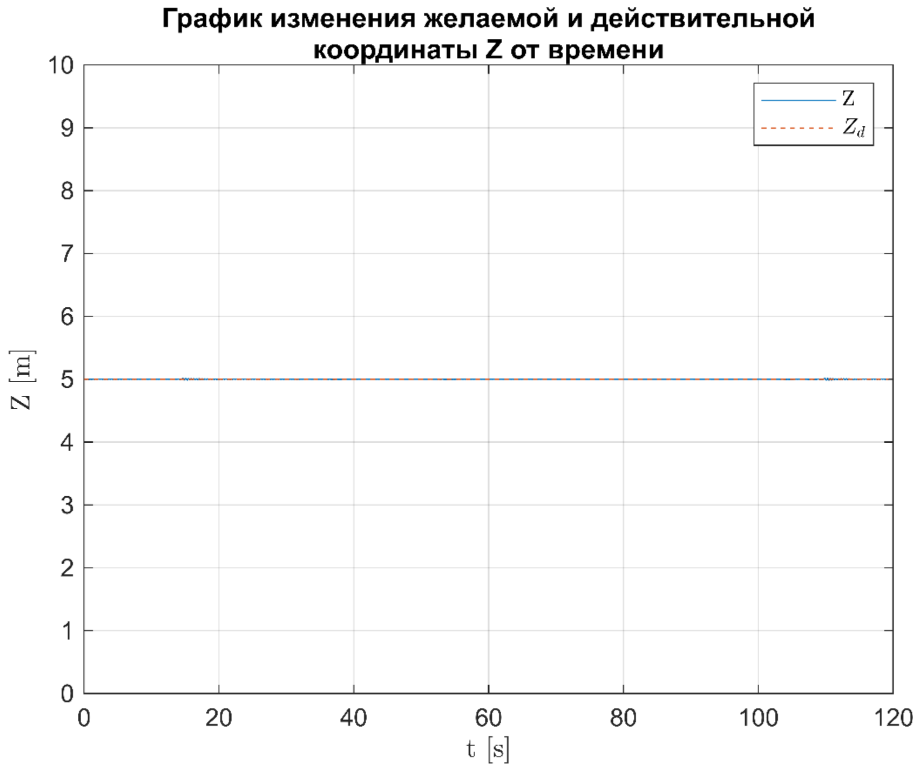

Рисунок 7. График изменения координаты Z

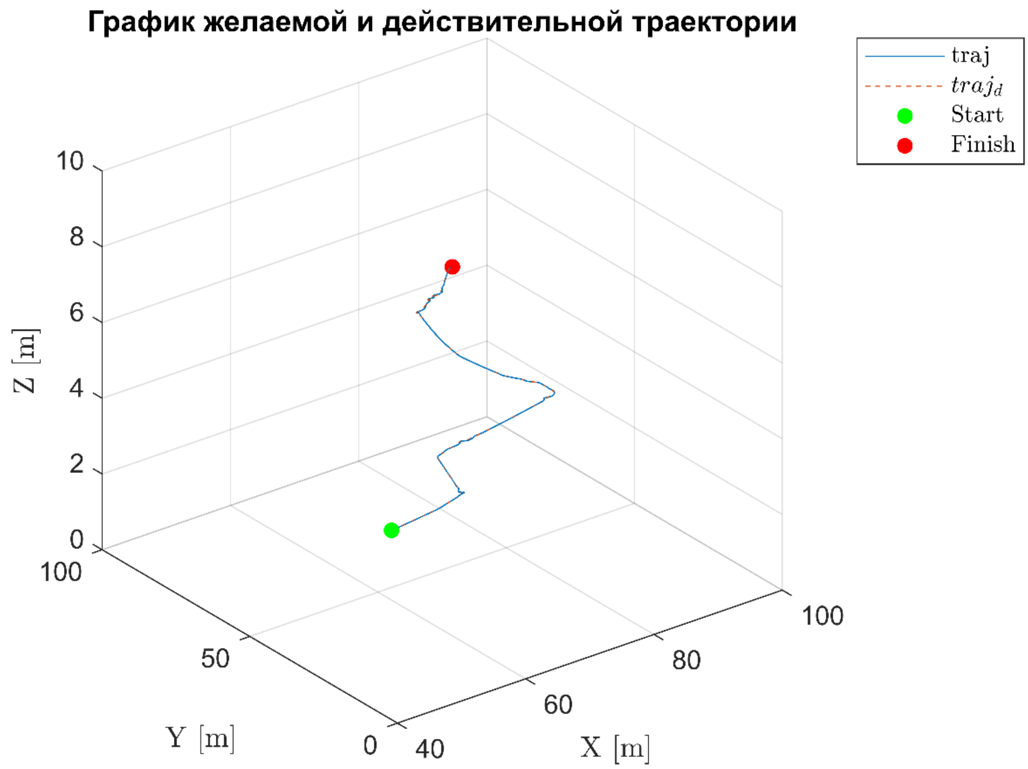

Рисунок 8. Трёхмерный вид траектории

На всех графиках видно, что дрон следует по заданной траектории с крайне небольшими отклонениями, что говорит о качественной настройке алгоритма бэкстеппинга.

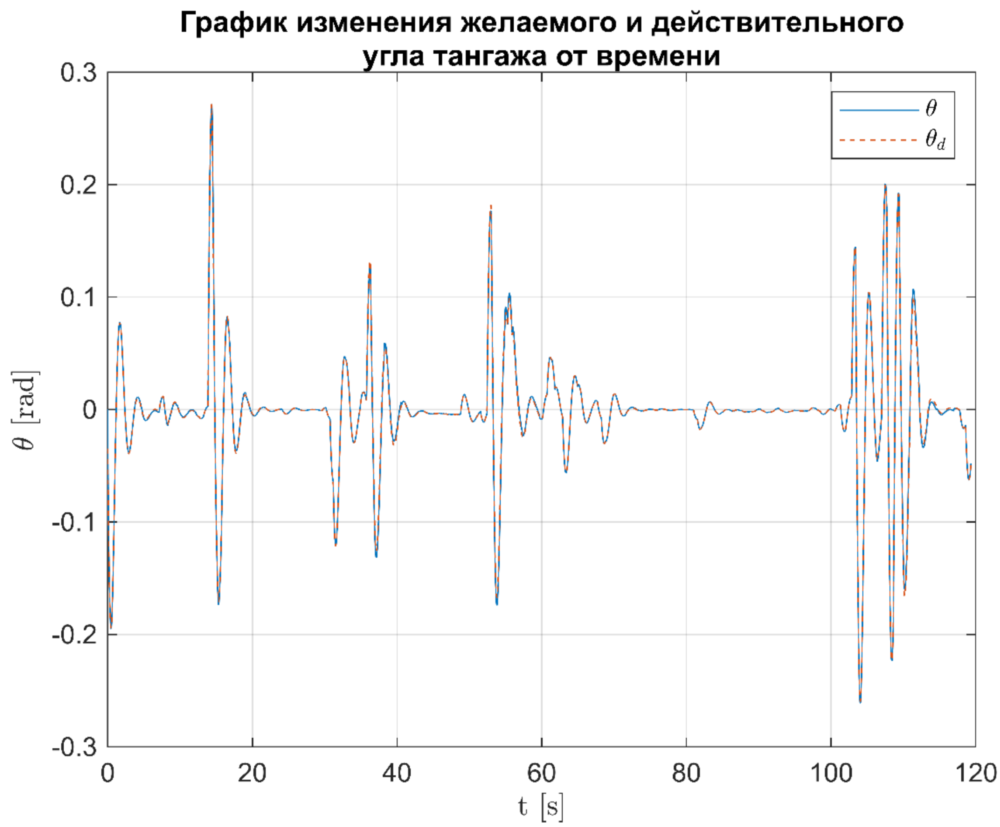

Рисунок 9. Изменение угла тангажа θ

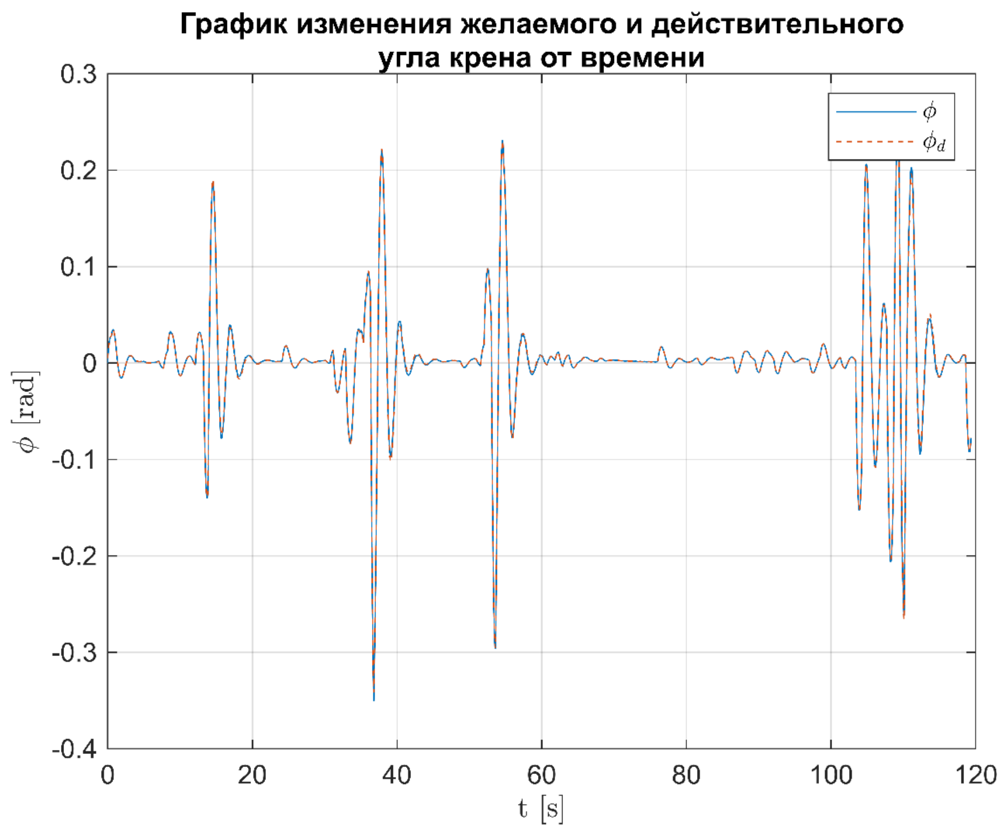

Рисунок 10. Изменение угла крена φ

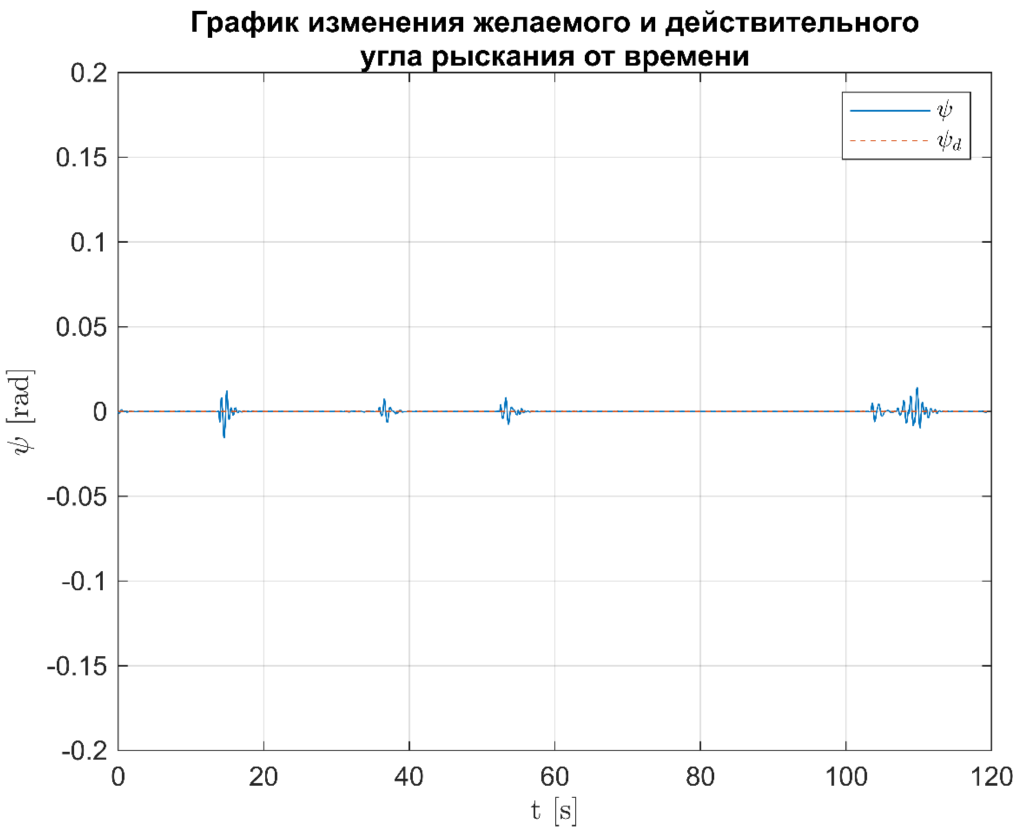

Рисунок 11. Изменение угла рыскания ψ

На графиках изменения углов видно, что и здесь предлагаемый алгоритм справляется крайне хорошо. Однако, необходимо провести сравнительный анализ с другим алгоритмом. Например, возьмём ПД-регулятор. Он был также встроен в исходную модель с квадрокоптером из Robotic Toolbox. Для сравнения на одном графике были одновременно выведены суммарные ошибки по координатам и по углам для алгоритма бэкстеппинга и для ПД-регулятора.

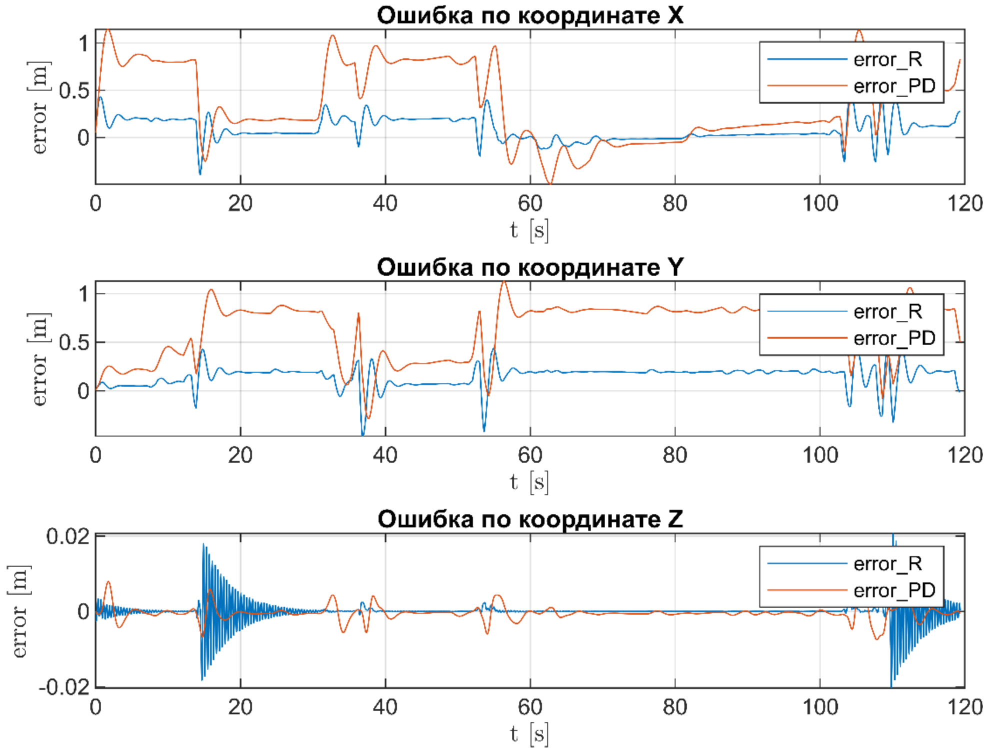

Рисунок 12. Сравнение ошибок по координатам для алгоритма бэкстеппинга и для изначальной системы управления с ПД-регулятором

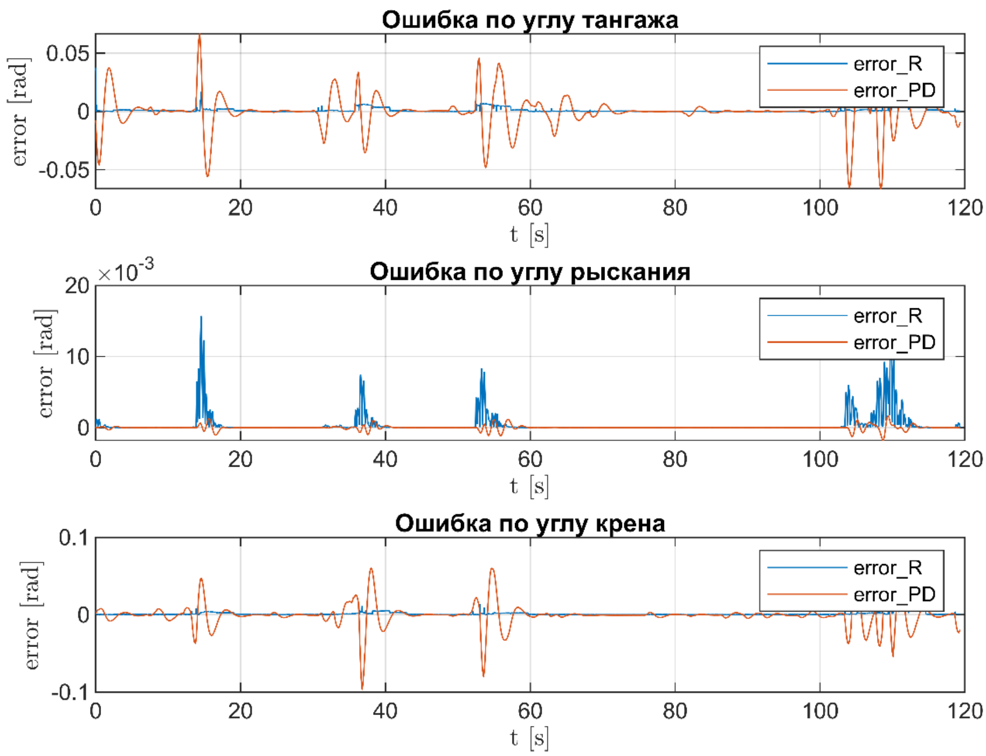

Рисунок 13. Сравнение ошибок по углам Эйлера для алгоритма бэкстеппинга и для изначальной системы управления с ПД-регулятором

На графиках ошибки для координат и углов Эйлера можно наблюдать, что для координат X, Y, угла тангажа и угла крена предлагаемый алгоритм бэкстеппинга показал себя значительно лучше – ошибка всё же присутствует, но она значительно меньше, чем для регулирования ПД-регулятором. Однако, для угла рыскания ошибка у метода бэкстеппинга будет больше в некоторые моменты, чем у ПД-регулятора, а в некоторые моменты будет меньше, практически нулевой.

Заключение

В ходе выполнения курсовой работы были решены следующие задачи:

1. Выполнен кинематический анализ робототехнической системы типа квадрокоптер и построена динамическая модель данного робота;
2. С помощью алгоритма быстроисследующего случайного дерева была спланирована траектория на карте с препятствиями, также данная траектория была оптимизирована таким образом, чтобы квадрокоптер поддерживал расстоянием минимум в 1 метр от всех препятствий;
3. Был построен алгоритм управления, базирующийся на методе бэкстеппинга;
4. Было проведено численное моделирование движения квадрокоптера по спланированной траектории и результаты моделирования сравнены с ПД-регулятором, предложенным в базовой модели из Robotic Toolbox;

В итоге было получено, что синтезированный регулятор обеспечивает управление по координатам X, Y, углу тангажа и углу крена лучше, чем ПД-регулятор, а по углу рыскания регулятор с алгоритмом бэкстеппинга обеспечивает управление не хуже, чем ПД-регулятор.

Список литературы

1. Варламова Л.П. Применение беспилотных летательных аппаратов в обеспечении технологической безопасности // Journal of Technical and Natural Sciences 5(14), 2019. – 54 – 90.;

2. Peter Corke. Robotics, Vision and Control fundamental algorithms in MATLAB. 2nd edition, 2017. – 697.;

3. Shirsat A., Modeling and control of a Quadrotor, aircraft, UAV. Arizona State University, 2015. – p. 13 – 16.;

4. Hassani H., Mansouri A. Control system of a quadrotor UAV with an optimized backstepping controller. 2019 International Conference on ISACS, 2020. – 7.;

5. He Z, Zhao L. A simple attitude control of quadrotor helicopter based on Ziegler-Nichols rules for tuning PD parameters. ScientificWorldJournal. 2014;2014:280180. doi: 10.1155/2014/280180. Epub 2014 Dec 29. PMID: 25614879; PMCID: PMC4295143.

6. Борисов О.И., Громов В.С., Пыркин А.А., Методы управления робототехническими приложениями. Учебное пособие. — СПб.: Университет ИТМО, 2016. — 108 с;

7. Довгополик И. С., Артемов К., Борисов О. И., Забихифар С., Семочкин А. Н. АЛГОРИТМ МОДИФИЦИРОВАННОГО ИНТЕЛЛЕКТУАЛЬНОГО ДВУНАПРАВЛЕННОГО СЛУЧАЙНОГО ДЕРЕВА ДЛЯ ПЛАНИРОВАНИЯ ДВИЖЕНИЯ АНТРОПОМОРФНЫХ МАНИПУЛЯТОРОВ // Приборостроение. 2022. №3. URL: https://cyberleninka.ru/article/n/algoritm-modifitsirovannogo-intellektualnogo-dvunapravlennogo-sluchaynogo-dereva-dlya-planirovaniya-dvizheniya-antropomorfnyh (дата обращения 25.03.2023);

8. Как работает метод Левенберга-Марквардта, 2019. По адресу URL: https://habr.com/ru/post/470181/ (дата обращения 25.03.2023).

Приложение

Ниже приведён программный листинг для курсовой работы, использовалась версия MATLAB 2022b. Возможно, некоторые функции по планированию траектории не будут работать в более ранних версиях.

Листинг программы:

```
clc; clear all;

warning('off','all')
% создаем карту
% map = makemap(100)
% save map_try.mat

%% планирование трпектории методом быстроисследующего случайного дерева
load map_try.mat

% создает пространство состояний для планирования пути робота
ss = stateSpaceSE2;
% создает валидатор (проверяющий) пути на основе заданной карты препятствий
sv = validatorOccupancyMap(ss);
% загружаем карту препятствий
map = occupancyMap(map);
sv.Map = map;
% задаем минимальное расстояние между двумя состояниями, которые должны быть проверены на достижимость
sv.ValidationDistance = 0.1;
% задаем границы пространства состояний
ss.StateBounds = [map.XWorldLimits;map.YWorldLimits; [-pi pi]];
% создаем планировщик на основе пространства состояний и валидатора пути
planner = plannerRRT(ss,sv);
% задаем максимальную дистанцию между двумя состояниями, которые могут быть соединены в дереве RRT
planner.MaxConnectionDistance = 3;

% задаем начальную точку [x, y, yaw]
% угол рыскания на траектории мы держим постоянным yaw = 0
start = [40,2,0];
% задаем конечную точку [x, y, yaw]
goal = [90,90,0];
% задаем высоту
% высоту на тректории мы держим постоянной
z = 5;
% задаем генератор случайных чисел
rng(100,'twister');
% выполняем планирование пути робота на основе заданного планировщика, начального и конечного состояний
```

```
[pthObj,solnInfo] = plan(planner,start,goal);

figure(1)
show(map)
hold on
plot(solnInfo.TreeData(:,1),solnInfo.TreeData(:,2),'.-','MarkerSize',10);
p1 = plot(pthObj.States(:,1),pthObj.States(:,2),'r-','LineWidth',2);

plot(start(1),start(2),'*g')
plot(goal(1),goal(2),'*r')

% оптимизируем траекторию
options = optimizePathOptions;
options.ObstacleSafetyMargin = 1;
optPath = optimizePath(pthObj.States,map,options);
hold on
grid minor
p2 = plot(optPath(:,1),optPath(:,2),"m-",LineWidth=1);
legend([p1,p2],"Planned Path","Optimized Path",Location="northwest")
hold off

% выводим параметры траектории для simulink модели
x_des = optPath(:,1);
y_des = optPath(:,2);

simout_ = sim('quadrotor_project_2021a');
simout_pd = sim('quadrotor_default_2021a');

figure(2)
plot(simout_.state.X.Time, simout_.state.X.Data)
hold on
plot(simout_.state_des.x__y_.Time, simout_.state_des.x__y_.Data(:,1), ...
        "LineStyle","--")
grid on
label2 = '${X_d}$';
label1 = 'X';
legend(label1,label2,'Interpreter',"latex",Location="northwest")
xlabel('t [s]','Interpreter',"latex")
ylabel('X [m]','Interpreter',"latex")
title({'График изменения желаемой и действительной', ...
   'координаты X от времени'})

figure(3)
plot(simout_.state.Y.Time, simout_.state.Y.Data)
hold on
plot(simout_.state_des.x__y_.Time, simout_.state_des.x__y_.Data(:,2), ...
```

```
        "LineStyle","--")
grid on
legend('Y','${Y_d}$','Interpreter',"latex",Location="northwest")
xlabel('t [s]','Interpreter',"latex")
ylabel('Y [m]','Interpreter',"latex")
title({'График изменения желаемой и действительной', ...
    'координаты Y от времени'})


figure(4)
plot(simout_.state.Z.Time, (simout_.state.Z.Data) * (-1))
hold on
plot(simout_.state.Z.Time, ones(1,length(simout_.state.Z.Time)) * ...
    simout_.state_des.z_des.Data,"LineStyle","--")
grid on
label2 = '${Z_d}$';
label1 = 'Z';
legend(label1,label2,'Interpreter',"latex")
xlabel('t [s]','Interpreter',"latex")
ylabel('Z [m]','Interpreter',"latex")
ylim([0,10])
title({'График изменения желаемой и действительной', ...
    'координаты Z от времени'})

figure(5)
plot3(simout_.state.X.Data, simout_.state.Y.Data, (simout_.state.Z.Data) * (-1))
hold on
plot3(simout_.state_des.x__y_.Data(:,1), simout_.state_des.x__y_.Data(:,2), ...
    ones(1,length(simout_.state.Z.Time)) * ...
    simout_.state_des.z_des.Data,"LineStyle","--")
hold on
plot3(start(1),start(2),z,'g.','MarkerSize', 20)
hold on
plot3(goal(1),goal(2),z,'r.','MarkerSize', 20)
grid on
label1 = 'traj';
label2 = '${traj_d}$';
label3 = 'Start';
label4 = 'Finish';
legend(label1,label2,label3,label4,'Interpreter',"latex")
xlabel('X [m]','Interpreter',"latex")
ylabel('Y [m]','Interpreter',"latex")
zlabel('Z [m]','Interpreter',"latex")
zlim([0,10])
title('График желаемой и действительной траектории')
```

```
figure(6)
plot(simout_.state.pitch.Time, simout_.state.pitch.Data)
hold on
plot(simout_.state_des.roll__pitch_.Time, simout_.state_des.roll__pitch_.Data(:,2), ...
        "LineStyle","--")
grid on
label2 = '${\theta_d}$';
label1 = '${\theta}$';
legend(label1,label2,'Interpreter',"latex")
xlabel('t [s]','Interpreter',"latex")
ylabel('${\theta}$ [rad]','Interpreter',"latex")
title({'График изменения желаемого и действительного', ...
   'угла тангажа от времени'})


figure(7)
plot(simout_.state.roll.Time, simout_.state.roll.Data)
hold on
plot(simout_.state_des.roll__pitch_.Time, simout_.state_des.roll__pitch_.Data(:,1), ...
        "LineStyle","--")
grid on
label2 = '${\phi_d}$';
label1 = '${\phi}$';
legend(label1,label2,'Interpreter',"latex")
xlabel('t [s]','Interpreter',"latex")
ylabel('${\phi}$ [rad]','Interpreter',"latex")
title({'График изменения желаемого и действительного', ...
   'угла крена от времени'})

figure(8)
plot(simout_.state.yaw.Time, simout_.state.yaw.Data)
hold on
plot(simout_.state.yaw.Time, zeros(1,length(simout_.state.yaw.Time)), ...
        "LineStyle","--")
grid on
label2 = '${\psi_d}$';
label1 = '${\psi}$';
legend(label1,label2,'Interpreter',"latex")
xlabel('t [s]','Interpreter',"latex")
ylabel('${\psi}$ [rad]','Interpreter',"latex")
ylim([-0.2,0.2])
title({'График изменения желаемого и действительного', ...
   'угла рыскания от времени'})

% графики для ПД
```

```
figure(9)
plot(simout_pd.state.X.Time, simout_pd.state.X.Data)
hold on
plot(simout_pd.state_des.x__y_.Time, simout_pd.state_des.x__y_.Data(:,1), ...
        "LineStyle","--")
grid on
label1 = 'X';
label2 = '${X_d}$';
legend(label1,label2,'Interpreter',"latex",Location="northwest")
xlabel('t [s]','Interpreter',"latex")
ylabel('X [m]','Interpreter',"latex")
title({'График изменения желаемой и действительной', ...
    'координаты X от времени для ПД'})

figure(10)
plot(simout_pd.state.Y.Time, simout_pd.state.Y.Data)
hold on
plot(simout_pd.state_des.x__y_.Time, simout_pd.state_des.x__y_.Data(:,2), ...
        "LineStyle","--")
grid on
label1 = 'Y';
label2 = '${Y_d}$';
legend(label1,label2,'Interpreter',"latex",Location="northwest")
xlabel('t [s]','Interpreter',"latex")
ylabel('Y [m]','Interpreter',"latex")
title({'График изменения желаемой и действительной', ...
    'координаты Y от времени для ПД'})

figure(11)
plot(simout_pd.state.Z.Time, (simout_pd.state.Z.Data) * (-1))
hold on
plot(simout_pd.state_des.z_des.Time, (simout_pd.state_des.z_des.Data) * (-1),"LineStyle","--")
grid on
label2 = '${Z_d}$';
label1 = 'Z';
legend(label1,label2,'Interpreter',"latex")
xlabel('t [s]','Interpreter',"latex")
ylabel('Z [m]','Interpreter',"latex")
ylim([0,10])
title({'График изменения желаемой и действительной', ...
    'координаты Z от времени для ПД'})

figure(12)
plot3(simout_pd.state.X.Data, simout_pd.state.Y.Data, (simout_pd.state.Z.Data) * (-1))
hold on
plot3(simout_pd.state_des.x__y_.Data(:,1), simout_pd.state_des.x__y_.Data(:,2), ...
```

```
    (simout_pd.state_des.z_des.Data) * (-1),"LineStyle","--")
hold on
plot3(start(1),start(2),z,'g.','MarkerSize', 20)
hold on
plot3(goal(1),goal(2),z,'r.','MarkerSize', 20)
grid on
label1 = 'traj';
label2 = '${traj_d}$';
label3 = 'Start';
label4 = 'Finish';
legend(label1,label2,label3,label4,'Interpreter',"latex")
xlabel('X [m]','Interpreter',"latex")
ylabel('Y [m]','Interpreter',"latex")
zlabel('Z [m]','Interpreter',"latex")
zlim([0,10])
title('График желаемой и действительной траектории для ПД')


figure(13)
plot(simout_pd.state.pitch.Time, simout_pd.state.pitch.Data)
hold on
plot(simout_pd.state_des.roll__pitch_.Time, simout_pd.state_des.roll__pitch_.Data(:,2), ...
        "LineStyle","--")
grid on
label2 = '${\theta_d}$';
label1 = '${\theta}$';
legend(label1,label2,'Interpreter',"latex")
xlabel('t [s]','Interpreter',"latex")
ylabel('${\theta}$ [rad]','Interpreter',"latex")
title({'График изменения желаемого и действительного', ...
    'угла тангажа от времени для ПД'})


figure(14)
plot(simout_pd.state.roll.Time, simout_pd.state.roll.Data)
hold on
plot(simout_pd.state_des.roll__pitch_.Time, simout_pd.state_des.roll__pitch_.Data(:,1), ...
        "LineStyle","--")
grid on
label2 = '${\phi_d}$';
label1 = '${\phi}$';
legend(label1,label2,'Interpreter',"latex")
xlabel('t [s]','Interpreter',"latex")
ylabel('${\phi}$ [rad]','Interpreter',"latex")
title({'График изменения желаемого и действительного', ...
    'угла крена от времени для ПД'})
```

```
figure(15)
plot(simout_pd.state.yaw.Time, simout_pd.state.yaw.Data)
hold on
plot(simout_pd.state.yaw.Time, zeros(1,length(simout_pd.state.yaw.Time)), ...
        "LineStyle","--")
grid on
label2 = '${\psi_d}$';
label1 = '${\psi}$';
legend(label1,label2,'Interpreter',"latex")
xlabel('t [s]','Interpreter',"latex")
ylabel('${\psi}$ [rad]','Interpreter',"latex")
ylim([-0.2,0.2])
title({'График изменения желаемого и действительного', ...
    'угла рыскания от времени для ПД'})
%%%%%%%%%%%%%%%%%%

% считаем ошибку по координатам для навороченного
error_x_R = (simout_.state_des.x__y_.Data(:,1) - simout_.state.X.Data);
error_y_R = (simout_.state_des.x__y_.Data(:,2) - simout_.state.Y.Data);
error_z_R = (ones(length(simout_.state.Z.Time),1) * ...
    simout_.state_des.z_des.Data - (simout_.state.Z.Data) * (-1));

% считаем ошибку по координатам для ПД
error_x_PD = (simout_pd.state_des.x__y_.Data(:,1) - simout_pd.state.X.Data);
error_y_PD = (simout_pd.state_des.x__y_.Data(:,2) - simout_pd.state.Y.Data);
error_z_PD = ((simout_pd.state_des.z_des.Data) * (-1) - (simout_pd.state.Z.Data) * (-1));

% считаем ошибку по углам для навороченного
error_pitch_R = abs(simout_.state_des.roll__pitch_.Data(:,2) - simout_.state.pitch.Data);
error_yaw_R = abs(zeros(length(simout_.state.yaw.Time), 1) - simout_.state.yaw.Data);
error_roll_R = abs(simout_.state_des.roll__pitch_.Data(:,1) - simout_.state.roll.Data);

% считаем ошибку по углам для ПД
error_pitch_PD = (simout_pd.state_des.roll__pitch_.Data(:,2) - simout_pd.state.pitch.Data);
error_yaw_PD = (zeros(length(simout_pd.state.yaw.Time),1) - simout_pd.state.yaw.Data);
error_roll_PD = (simout_pd.state_des.roll__pitch_.Data(:,1) - simout_pd.state.roll.Data);

error_Rc = [error_x_R error_y_R error_z_R];
error_Ra = [error_pitch_R error_yaw_R error_roll_R];
error_PDc = [error_x_PD error_y_PD error_z_PD];
error_PDa = [error_pitch_PD error_yaw_PD error_roll_PD];
ttlec = {'Ошибка по координате X', 'Ошибка по координате Y', 'Ошибка по координате Z'};
ttlea = {'Ошибка по углу тангажа', 'Ошибка по углу рыскания', 'Ошибка по углу крена'};

figure(16)
```

```
for i = 1 : size(error_Rc,2)
    subplot(3,1,i)
    plot(simout_.state.X.Time, error_Rc(:,i))
    hold on
    plot(simout_pd.state.X.Time, error_PDc(:,i))
    grid on
    label1 = 'error_R';
    label2 = 'error_PD';
    legend(label1,label2,'Interpreter',"latex")
    xlabel('t [s]','Interpreter',"latex")
    ylabel('error [m]','Interpreter',"latex")
    title(ttlec{i})
end

figure(17)
for i = 1 : size(error_Ra,2)
    subplot(3,1,i)
    plot(simout_.state.X.Time, error_Ra(:,i))
    hold on
    plot(simout_pd.state.X.Time, error_PDa(:,i))
    grid on
    label1 = 'error_R';
    label2 = 'error_PD';
    legend(label1,label2,'Interpreter',"latex")
    xlabel('t [s]','Interpreter',"latex")
    ylabel('error [rad]','Interpreter',"latex")
    title(ttlea{i})
end

mkdir images
str = [pwd,"images"];
pathdir =  join(str,"\");
cd(pathdir)
for i = 1 : 17
    exportgraphics(figure(i),[num2str(i),'.png'],'Resolution',1200);
end
```