МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)»

Институт (Филиал) **№1 Авиационная техника** **Кафедра 106**

Группа **М1О-204М-22** **Направление подготовки** **24.04.03 - Баллистика и гидроаэродинамика**

Магистерская программа **Динамика полёта и управление движением летательных аппаратов**

Квалификация **магистр**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

На тему: Использование методов обучения с подкреплением в задаче продольного управления ЛА

Автор ВКРМ (магистерской диссертации) Москвитин Андрей Семенович (_____)
(фамилия, имя, отчество полностью)

Научный руководитель Тюменцев Юрий Владимирович (_____)
(фамилия, имя, отчество полностью)

Консультант _____ (_____)
(фамилия, имя, отчество полностью)

Консультант _____ (_____)
(фамилия, имя, отчество полностью)

Рецензент Каганов Юрий Тихонович (_____)

К защите допустить

Заведующий кафедрой 106 Ефремов Александр Викторович (_____)
(№ каф) (фамилия, имя, отчество полностью)

30 мая 2024 г.

Москва 2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)»

Институт (Филиал) №1 Авиационная техника **Кафедра** 106

Группа М1О-204М-22 **Направление подготовки** 24.04.03 - Баллистика и гидроаэродинамика

Магистерская программа Динамика полёта и управление движением летательных аппаратов

Квалификация магистр

УТВЕРЖДАЮ

Заведующий кафедрой 106 Ефремов А.В.
(№ каф.) (подпись) (инициалы, фамилия)
26 сентября 2022 г.

ПЛАН РАБОТЫ
над выпускной квалификационной работой магистра (магистерской диссертацией)

Обучающийся Москвитин Андрей Семенович
(фамилия, имя, отчество полностью)

Руководитель Тюменцев Юрий Владимирович
(фамилия, имя, отчество полностью)
д.т.н, профессор кафедры 106
ученая степень, ученое звание, должность и место работы)

1. Наименование предварительной темы (тематики) Использование методов обучения с подкреплением в задаче продольного управления ЛА

2. Срок сдачи обучающимся законченной работы 30.05.2024

3. Цель Разработка адаптивного контроллера управления в продольном канале

Перечень иллюстративно-графических материалов:

№ п/п	Наименование	Количество листов
1	Презентация «Использование методов обучения с подкреплением в задаче продольного управления ЛА»	26

4. Перечень задач, решаемых для достижения поставленной цели

№	Наименование задачи	Срок выполнения	Примечание
1	Проведение анализа теоретического материала	06.02.2023	
2	Разработка и отладка математической модели движения ЛА в продольном канале	08.05.2023	
3	Синтез адаптивного контроллера с помощью метода обучения с подкреплением	23.10.2023	
4	Выполнение серии вычислительных экспериментов с разработанным контроллером для модели продольного движения ЛА	04.03.2024	

5. Исходные материалы и пособия

1. Nguyen Luat T., Ogburn Marilyn E., Gilbert William P., Kibler Kemper S., Brown Phillip W., Deal Perry L. Simulator study of stall/post-stall characteristics of a fighter airplane with relaxed longitudinal static stability // NASA TP-1538. 1979, с. 223.

2. Sutton Richard S, Barto Andrew G. Reinforcement learning: An introduction // MIT press, 2018.

3. Schulman John, Wolski Filip, Dhariwal Prafulla, Radford Alec, Klimov Oleg. Proximal Policy Optimization Algorithms. arXiv: 1707.06347, 2017.

4. EASA. Easy Access Rules for the Certification Specifications for All-Weather Operations (EAR for CS-AWO Issue 2). 2022.

5. Raffin Antonin, Kober Jens, Stulp Freek. Smooth Exploration for Robotic Reinforcement Learning. // arXiv: 2005.05719, 2021.

6. John Schulman Sergey Levine, Philipp Moritz Michael I. Jordan Pieter Abbeel. Trust Region Policy Optimization. // arXiv: 1502.05477, 2017.

7. Вересников Г.С., Скрябин А.В. “Методы искусственного интеллекта в системах автоматизированного управления беспилотными летательными аппаратами”. // Информационные технологии №3 том 30, 2024, с. 115—123.

6. Дата составления плана 26.09.2022г.

Руководитель_____

(подпись)

Обучающийся_____

(подпись)

Реферат

72 страниц, 26 рисунков, 6 таблиц, 12 источников, 1 приложение.

Ключевые слова: динамика полета, управление полетом, адаптивное управление, обучение с подкреплением, моделирование движения.

В данной работе рассматривается задача формирования адаптивного контроллера методом обучения с подкреплением. Используется метод непосредственной оптимизации стратегии (Proximal Policy Optimization) как алгоритм для формирования контроллера управления. Проведена адаптация метода для задачи управления продольным движением летательного аппарата, выполнено тестирование полученного контроллера в ряде тестовых сценариев. Полученные результаты показывают работоспособность данного подхода в формировании адаптивного контроллера.

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 7

ВВЕДЕНИЕ 9

1 ОПИСАНИЕ ОБЪЕКТА УПРАВЛЕНИЯ 11
1.1 Уравнения движения 11
1.2 Органы управления 12
1.3 Использованные упрощения 14
1.4 Реакция объекта управления без системы управления 15
1.5 Выводы по разделу 16

2 ОПИСАНИЕ МЕТОДОВ ОБУЧЕНИЯ С ПОДКРЕПЛЕНИЕМ ДЛЯ РЕШЕНИЯ ЗАДАЧИ СЛЕЖЕНИЯ 17
2.1 Марковский процесс принятия решений 17
2.2 Уравнение Беллмана 20
2.3 Методы градиента стратегии 22
2.4 Актор-критик 23
2.5 Алгоритм оптимизации стратегии на основе доверительного региона 25
2.6 Непосредственная оптимизация стратегии 26

3 ПРИМЕНЕНИЯ МЕТОДА НЕПОСРЕДСТВЕННОЙ ОПТИМИЗАЦИИ СТРАТЕГИИ К ЗАДАЧЕ ОТСЛЕЖИВАНИЯ 29
3.1 Описание задачи управления 29
3.2 Описание задачи в терминах обучения с подкреплением . . . 29
3.3 Функция вознаграждения 31
3.4 Критерий для оценки эффективности управления 32
3.5 Формирование действий 32
3.5.1 Выбор действия на основе состояния 33
3.6 Описание тренировочного цикла 35
3.7 Обучающий сигнал 40

4 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ СФОРМИРОВАННОГО КОНТРОЛЛЕРА УГЛОВОЙ СКОРОСТИ ТАНГАЖА 42
4.1 Тестовые сценарии 43
4.2 Адаптация вне обучающей выборки 47
4.3 Особые случаи 49
4.3.1 Влияние шума датчика угловых скоростей 50
4.3.2 Влияние ветровых возмущений 52
4.3.3 Анализ результатов 54
4.4 Сравнение сформированного PPO-контроллера с PI-контроллером 54
4.5 Анализ результатов 57

ЗАКЛЮЧЕНИЕ 59

СПИСОК ЛИТЕРАТУРЫ 61

ПРИЛОЖЕНИЕ А 63

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ЛА – летательный аппарат

МППР – марковский процесс принятия решений

НС – нейронная сеть

ОсП – обучение с подкреплением

РУД – ручка управления двигателем

САО – средняя абсолютная ошибка

СКО – среднеквадратическая ошибка

EASA – European Union Aviation Safety Agency

PPO – Proximal Policy Optimization

По части объекта управления:

x – вектор состояние объекта управления

H – высота полета

V_x – проекция воздушной скорости на ось O_x

V_y – проекция воздушной скорости на ось O_y

ϑ – угол тангажа

ω_z – угловая скорость тангажа

$\omega_{z_{err}}$ – ошибка слежения угловой скорости тангажа

α – угол атаки

q – скоростной напор

n_{xa} – тангенциальная перегрузка

n_{ya} – нормальная перегрузка

Y_a – подъемная сила

X_a – сила лобового сопротивления

φ_{act} – угол отклонения стабилизатора

φ_{ref} – командное значение угла отклонения стабилизатора

По части обучения с подкреплением:

π – стратегия

a_t – управляющий сигнал формируемый агентом в момент времени t

θ – вектор параметра стратегии

s_t – состояние среды в момент времени t

$\bar{s}_t$ – нормированное состояние среды в момент времени t

R_t – награда от среды в момент времени t

$G_t(s)$ – функция возврата

$v_t(s)$ – функция ценности состояния

$q(s, a)$ – функция ценности действия

$V_t(s)$ – приближенная функция ценности состояния

$Q(s, a)$ – приближенная функция ценности действия

$J(\theta)$ – целевая функция

$A(s, a)$ – функция превосходства

ВВЕДЕНИЕ

В последние годы наблюдается бурное развитие инструментов машинного обучения в различных областях, где с переменным успехом решались различные задачи. Область автоматического управления не осталась в стороне.

Основная цель системы управления для различных динамических систем (объектов управления) — это достижение желаемой реакции объекта управления и обеспечение устойчивости в определенном эксплуатационном диапазоне.

Существуют различные традиционные подходы к разработке методов управления, такие как: ПИД-контроллер, LQR/LQG-контроллер, H_∞ управление и т.д. Все эти методы используют линейный контроллер и требуют подбора коэффициентов и/или оптимизации функции невязки, которая обеспечат желаемую реакцию объекта нелинейного в определенной области и для его синтеза необходимо иметь как можно более точную модель объекта, однако как известно все реальные объекты нелинейны, в том числе маневренные самолеты, и иметь модель, в точности описывающую реальный объект, в большинстве задач не представляется возможным. Методы адаптивного управления могут справляться с неопределенностями в динамике управляемого объекта, однако формируемое решение не всегда удовлетворяет критериям оптимальности.

В данной работе рассматривается универсальный подход к синтезу контроллера, решающий широкий спектр задач управления. Обладающий преимуществами линейных контроллеров — их эффективность в определенном диапазоне, а также адаптивных контроллеров, которые позволяют подстраиваться под изменение параметров объекта.

Совершенно иной подход к синтезу закона управления, основанный на подстройке контроллера методом проб и ошибок, который объединяет традиционные методы управления и адаптивные, путем поиска решения

уравнения Гамильтона-Якоби-Беллмана. Одна из областей машинного обучения — обучение с подкреплением (ОсП) использует данный принцип и может быть применена для синтеза адаптивного контроллера.

1 ОПИСАНИЕ ОБЪЕКТА УПРАВЛЕНИЯ

В данном разделе будет дано математическое описание объекта управления, к которому будут применен метод ОсП для решения задачи отслеживания задающего сигнала угловой скорости тангажа.

1.1 Уравнения движения

Рассматривается модель изолированного продольного движения, где вектор состояния ЛА $x = (V_x, V_y, H, \vartheta, V, \alpha, \omega_z)$ описывает движение ЛА в продольном канале, которая определяется системой нелинейных дифференциальных уравнений:

$$\dot{x} = \begin{cases} \dot{V}_y = V\cos\alpha, \\ \dot{V}_x = V\sin\alpha, \\ \dot{H} = V_x\sin\vartheta + V_y\cos\vartheta, \\ \dot{\vartheta} = \omega_z, \\ \dot{V} = g\left[n_{xa} - \sin\left(\vartheta - \alpha\right)\right], \\ \dot{\alpha} = -\frac{g}{V}\left[n_{ya} - \cos\left(\vartheta - \alpha\right)\right] + \omega_z, \\ \dot{\omega}_z = \frac{M_{R_{za}}}{J_z}, \end{cases} \tag{1.1}$$

где:

H – высота полета;

V_x – проекция воздушной скорости на ось O_x связанной системы координат;

V_y – проекция воздушной скорости на ось O_y связанной системы координат;

ϑ – угол тангажа;

α – угол атаки;

ω_z – угловая скорость тангажа;

$g = 9.81\frac{\text{м}}{\text{с}^2}$ – ускорение свободного падения;

$n_{xa} = \frac{X_a + P_{xa}}{mg}$ – тангенциальная перегрузка в скоростной системе координат.;

$n_{ya} = \frac{Y_a}{mg}$ – нормальная перегрузка в скоростной системе координат;

$X_a = -C_{xa} q S$ – сила лобового сопротивления;

$q = \frac{\rho V^2}{2}$ – скоростной напор (динамическое давление);

$Y_a = C_{ya} q S$ – подъемная сила;

$M_{R_{za}} = M_{za} + r_{\text{ц.м}} Y_a = m_z q S b_a + r_{\text{ц.м}} Y_a$ – полный момент тангажа.

В данных уравнениях аэродинамические силы и моменты зависящие от безразмерных коэффициентов $m_z(\alpha, \omega_z, V, \varphi)$, $C_{xa}(\alpha, \omega_z, V, \varphi)$, $C_{ya}(\alpha, \omega_z, V, \varphi)$, а также от силы тяги $P_x(H, M, P_a)$, которые являются нелинейными функциями, заданными таблично в работе [4] для самолета F-16.

Коэффициенты C_{xa}, C_{ya}, m_z определены в диапазоне углов атаки $-20 \leqslant \alpha \leqslant 90°$, P_x определен в диапазоне чисел Маха $0.1 \leqslant M \leq 0.6$. При вычисления состояния с фиксированным шагом $\Delta t = 0.01$ с интерполирование табличных значений производилось линейным методом.

Общая структура расчета состояния показана на рис. 1.1.

Рисунок 1.1 – Структурная схема расчета состояния x

1.2 Органы управления

Для описания привода органа управления, вводящего задержку в тракт управления между командным сигналом от контроллера и реальным отклонением стабилизатора, будем использовать передаточную функцию второго порядка:

$$W\left\{\frac{\varphi_{act}}{\varphi_{ref}}\right\} = \frac{1}{T^2_{\text{стаб}} p^2 + 2 T_{\text{стаб}} \xi_{\text{стаб}} p + 1},$$

где $T_{\text{стаб}} = 0.03$ с, $\xi_{\text{стаб}} = 0.707$, φ_{act} — реальное положение стабилизатора, φ_{ref} — командное значение стабилизатора. При приведении графиков отклонения за φ будет приниматься реальное положение стабилизатора φ_{act}, если на графике не обозначено иное.

Реакция на ступенчатый сигнал имеет вид, представленный на рис. 1.2.

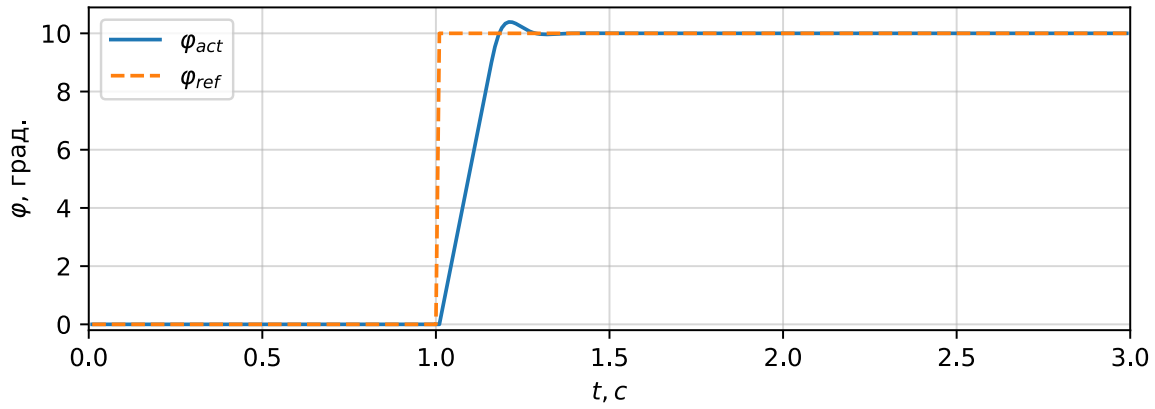


Рисунок 1.2 – Динамика привода органа управления стабилизатором

Динамика управляющего сигнала двигателя описывается апериодическим звеном 1-го порядка:

$$\dot{P}_a = \frac{1}{\tau}(P_c - P_a),$$

где P_a — уровень тяги, создаваемый двигателем [%], P_c — командный сигнал уровня тяги, зависящий от положения РУД $\delta_{\text{РУД}}$, определяется как:

$$P_c(\delta_{\text{РУД}}) = \begin{cases} 64.94\,\delta_{\text{РУД}}, & \text{если } \delta_{\text{РУД}} \leqslant 0.77 \\ 217.38\,\delta_{\text{РУД}} - 117.398, & \text{если } \delta_{\text{РУД}} > 0.77 \end{cases}$$

$$P_c = \begin{cases} P_c, & \text{если } P_c \geqslant 50 \text{ и } P_a \geqslant 50 \\ 60, & \text{если } P_c \leqslant 50 \text{ и } P_a < 50 \\ 40, & \text{если } P_c < 50 \text{ и } P_a \geqslant 50 \\ P_c, & \text{если } P_c < 50 \text{ и } P_a < 50 \end{cases}$$

Постоянная времени $\frac{1}{\tau}$ определяется условиями, зависящими от P_c и P_a:

$$\frac{1}{\tau} = \begin{cases} 5, & \text{если } P_c \geq 50 \text{ и } P_a \geqslant 50 \\ \omega_\tau, & \text{если } P_c \leqslant 50 \text{ и } P_a < 50 \\ 5, & \text{если } P_c < 50 \text{ и } P_a \geqslant 50 \\ \omega_\tau, & \text{если } P_c < 50 \text{ и } P_a < 50 \end{cases}$$

$$\omega_\tau = \begin{cases} 1, & \text{если } (P_c - P_a) \leqslant 25 \\ 0.1, & \text{если } (P_c - P_a) \geqslant 50 \\ 1.9 - 0.036(P_c - P_a), & \text{если } 25 < (P_c - P_a) < 50 \end{cases}$$

1.3 Использованные упрощения

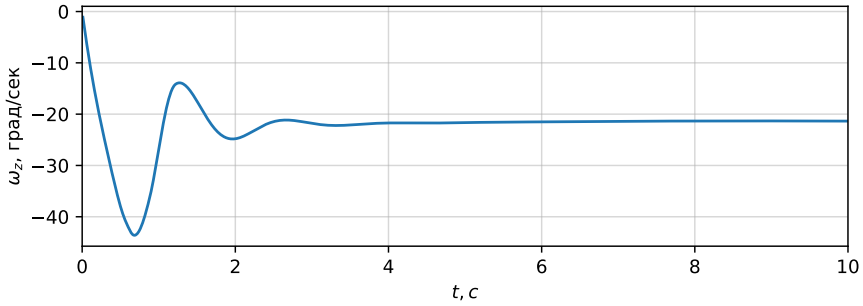
Для дальнейшего рассмотрения задачи отслеживания задающего сигнала угловой скорости тангажа ω_z будут приняты следующие упрощения в модели:

1. Скорость на всем протяжении эксперимента замораживается начальным значением, так как управление тягой $\delta_{\text{РУД}}$ не будет рассматриваться ($P_a = 0$).
2. Введено ограничение на максимальное значение угловой скорости тангажа:

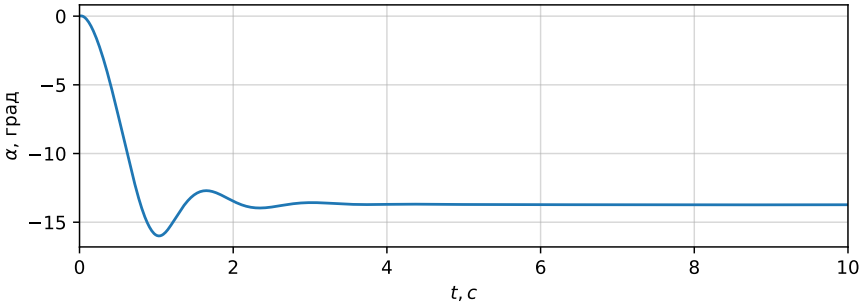
$$\omega_z = \begin{cases} -60 \text{ град/с} & \text{при } \omega_z \leqslant -60 \text{ град/с}, \\ \omega_z \text{ град/с} & \text{при } -60 \text{ град/с} < \omega_z < 60 \text{ град/с}, \\ 60 \text{ град/с} & \text{при } \omega_z \geqslant 60 \text{ град/с}. \end{cases}$$

1.4 Реакция объекта управления без системы управления

Рассмотрим динамику объекта с фиксированными органами управления $u_0 = (\varphi = 0, \delta_{\text{РУД}} = 0)$ из начального состояния $x_0 = (H = 2500\,\text{м}, V = 275\,\text{м/с}, \vartheta = 0, \omega_z = 0, \alpha = 0)$, реакция объекта приведена на рис. 1.3.



(a) Изменение угловой скорости тангажа ω_z

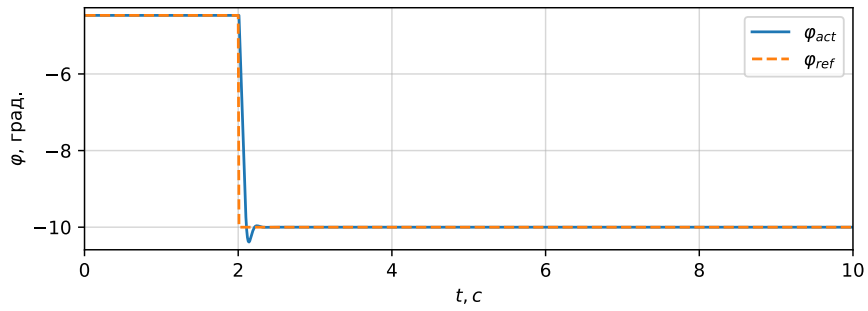


(b) Изменение угла атаки α

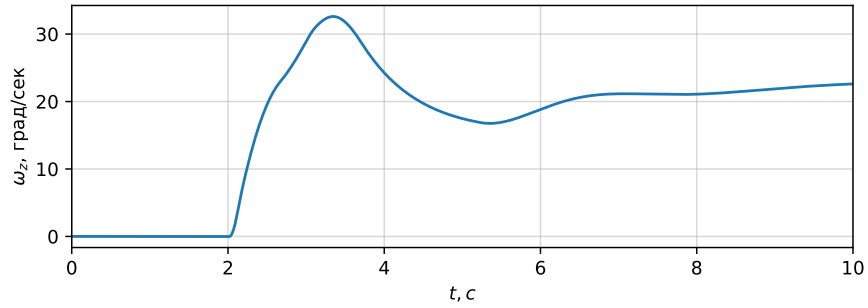
Рисунок 1.3 – Изменение параметров модели при $u_0 = (\varphi = 0, \delta_{\text{РУД}} = 0)$

Управляющее воздействие u_0 являются не балансировочными и объект достаточно быстро ($\approx$ 3 сек.) выходит на установившееся значение угловой скорости $\omega_z \approx 22$ град/с.

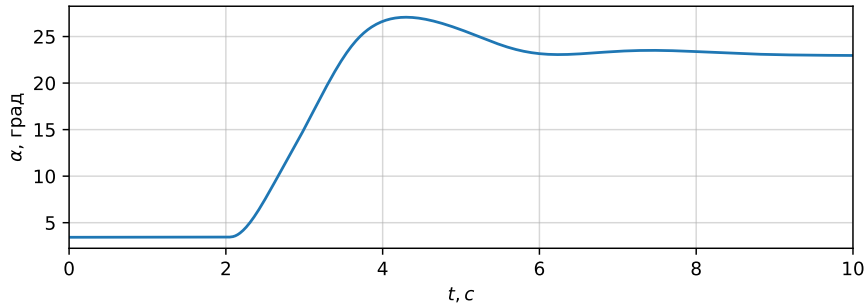
Также рассмотрим реакцию объекта на ступенчатое отклонение φ из балансировочного состояния $x_{\text{бал.}} = (H = 2500\,\text{м}, V = 175\,\text{м/с}, \vartheta = 3.4334\,\text{град}, \omega_z = 0, \alpha = 3.4334\,\text{град})$, $u_{\text{бал}} = (\varphi = -4.47\,\text{град}, \delta_{\text{РУД}} = 0.28\%)$, реакция объекта приведена на рис. 1.4. Изменения параметров ω_z, α при величине ступенчатого сигнала $\Delta\varphi_{ref} = -5.52$ град характерны для маневренного самолета.



(a) Изменение командного сигнала φ_{ref} и φ_{act}



(b) Изменение угловой скорости тангажа ω_z



(c) Изменение угла атаки α

Рисунок 1.4 – Изменение параметров модели при отработке ступенчатого сигнала на кабрирование

1.5 Выводы по разделу

Приведенная модель нелинейного продольного движения показывает адекватную реакцию, характерную для маневренного самолета в продольном канале, основанная на исходных данных из [4] и может быть использована для синтеза системы управления стабилизации углового движения тангажа.

2 ОПИСАНИЕ МЕТОДОВ ОБУЧЕНИЯ С ПОДКРЕПЛЕНИЕМ ДЛЯ РЕШЕНИЯ ЗАДАЧИ СЛЕЖЕНИЯ

Для решения задачи оптимального управления применяют различные методы (LQR/LQG-контроллер, MPC-управление, H_∞-контроллер и т.д.), один из альтернативных подходов для решения задачи оптимального управления это методы ОсП.

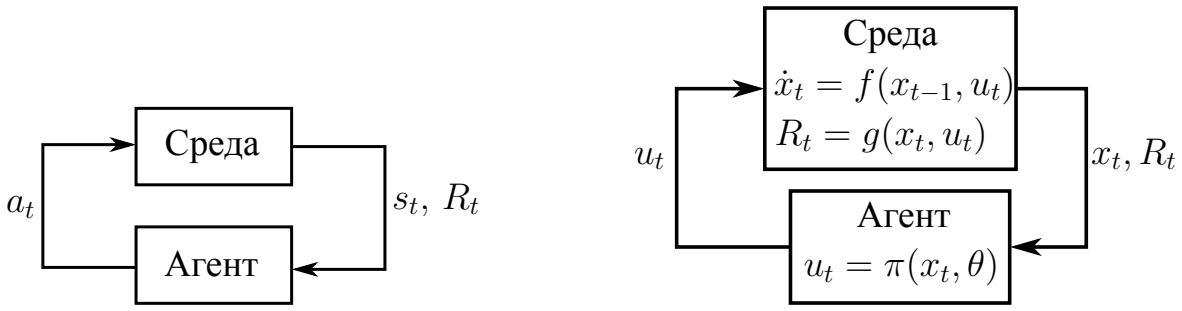
Такие классы методов показывает неплохие результаты в решении реальных задач по сравнению с классическими подходами без применения методов ОсП, а также в некоторых задачах такой подход дает лучший результат [9], чем классический метод решения. В данных методах задается определенная цель, которую нужно достичь и алгоритмы позволяют с определенным успехом добиться поставленной цели путем проб и ошибок.

Методы ОсП работают в режиме реального времени (имеется в виду нужно непрерывное взаимодействие с окружающей средой) с постоянной обратной связью от окружающей среды (в дальнейшем будем именовать среда) в виде скалярного значения, именуемой наградой R_t, которую агент получает в момент времени t. Величина награды определяет успешность выполнения задачи. Среда — модель какого-либо объекта или реальный объект управления, с которым взаимодействует агент. Агент — абстракция для функции, которую нужно определить, чтобы на основе состояния среды s_t выбиралось оптимальное действие a_t. Упрощенная структурная схема взаимодействия представлена на рис. 2.1.

2.1 Марковский процесс принятия решений

Марковский процесс принятия решений (Markov Decision Process) (МППР) определяет основные термины, связанные со средой для методов обучения с подкреплением. МППР определяется как кортеж, состоящий из $(S, D, A, \{P_a(s, s')\}, \gamma, G_t)$, где:

- S – множество состояний среды, включая начальное состояние S_0;



(a) Взаимодействие агента со средой в терминах ОсП; s_t – состояние среды, R_t – награда на текущем шаге, a_t – действие агента на основе состояния s_t

(b) Взаимодействие агента со средой в терминах управления движением ЛА; x – вектор состояния ЛА, u – вектор управления

Рисунок 2.1 – Упрощенная структурная схема взаимодействия агента со средой

- D – распределение начальных состояний среды;
- A – множество возможных действий над средой, которые агент выбирает в момент времени t;
- $P_{sa}(\cdot)$ – вероятность перехода состояния. Определяет для состояния $s \in S$ и действия $a \in A$ в момент времени t, *распределение вероятности* перехода в следующее состояние s' для следующего момента времени $t+1$;
- γ – число в диапазоне $[0, 1]$, называемое фактором дисконтирования;
- R – функция награды, ограниченная максимальной наградой $R_{\max}$ ($|R(s)| \leqslant R_{\max} \forall s$);

Последовательность происходящих событий в терминах МППР определяется следующим образом. Агент из начального состояния s_0, определенного из распределения начальных состояний D на каждом временно́м шаге t, выбирает действие a_t, в результате которого среда переходит в следующее состояние s_{t+1}. Распределение вероятности перехода среды в следующее состояние определяется функцией $P_{sa}(\cdot)$. Путем последовательного выбора различных действий a формируется последовательность состояний, наград и действий $\tau = (s_0, a_0, r_0, s_1, \ldots)$, такая последовательность называется *траекторией*. *Эпизодом* называется по-

следовательность взаимодействий агента из начального состояния среды s_0 до конечного состояния s_T. Для определения эффективности агента на траектории τ вводят *функцию возврата*, которая вычисляет полученную суммарную дисконтированную награду и определяется как:

$$G_t(\tau) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots . \tag{2.1}$$

Фактор дисконтирования γ, строго говоря, всегда меньше 1, чтобы не возникало ситуаций, когда на бесконечно длинном эпизоде возникала бесконечная награда. Интуитивно это объясняется тем, что награда, полученная в будущем, вносит меньший вклад по сравнению с наградой, получаемой в начале эпизода, чтобы стимулировать агента определять действия, приносящих кратковременную выгоду. Если же коэффициент дисконтирования близок к 1, то рассматривается полная полученная награда на конечном эпизоде и агент будет учитывать долгосрочную выгоду в своих действиях.

Функция награды также может быть стохастической функцией $R(s_t, a_t)$, зависящей от действия a_t, нежели детерминированной и зависящей только от состояния $R(s_t)$.

Для ситуации, где начальные состояния не играют роли, последовательность МППР может иметь вид $(S, A, \{P_a(s, s')\}, \gamma, G_t)$.

Цель в ОсП — это найти способ выбора действий $a_0, a_1 \dots, a_t$, которые бы максимизировали функцию возврата (2.1).

В МППР агент от среды получает наблюдение состояния s_t на каждом временном шаге, что позволяет выбрать действие a_t как функцию, зависящую от текущего состояния и предыдущих состояний $s_0, \dots, s_t$. Однако в связи с марковсим свойством для МППР, которое состоит в том, что выбор следующего состояния зависит только от текущего и никак не связано с предшествующими состояниями. Тогда для оптимальности функции ценности (2.1) достаточно выбирать действие a_t на основе только текущего состояния s_t [10]. В этом случае **цель ОсП** можно сформулировать как поиск такой стратегия $\pi : S \mapsto A$, которая для каждого состояния

состояния s максимизирует ожидаемую награду:

$$\mathbb{E}_\pi \left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots \right].$$

Здесь индекс π в $\mathbb{E}_\pi$ означает математическое ожидание награды по отношению к выбранному действию согласно стратегии π.

Функция награды R — это интерпретация «описания задачи» агенту и данная функция входит в целевую функцию, которую нужно минимизировать. Обычно данная функция зависит от задачи и определяется эмпирически, исходя из цели задачи. В зависимости от выбора функции награды агент может обучиться оптимальной стратегии или же уйти в сторону субоптимального решения при неудачно выбранной функции награды. Выбор данной функции оказывает влияние на результативность агента.

2.2 Уравнение Беллмана

Имея стратегию π, можно определить функцию ценности состояния $v^\pi : S \mapsto \mathbb{R}^1$ (state-value function), как математическое ожидание будущей дисконтированной награды с момента времени t, начиная со состояния s, или можно понимать как выгодность нахождения в определенном состоянии s. Формально функцию ценности состояния можно записать как:

$$v^\pi(s) = \mathbb{E}_\pi \left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots | s_t = s \right],$$

где верхний индекс π в v^π означает, что функция ценности связана с определенной стратегией π на основе которого происходило получение состояния среды s_t. В дальнейшем верхний индекс будет опускаться.

Аналогично можно определить функцию ценности для действия (action-value function), также по другому функцию называемую Q-функция:

$$q^\pi(s, a) = \mathbb{E}_\pi \left[G_t | s_t = s, a_t = a \right] =$$

$$= \mathbb{E}_\pi [R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots | s_t = s, a_t = a, \forall t > 0 \; a_t = \pi(s_t)].$$

На практике функции $q(s,a)$ и $v(s)$ принимают аппроксимированное значение с помощью нейронных сетей, их принято обозначать как $Q^{\pi}(s,a)$, $V^{\pi}(s)$.

Оптимальная функция ценности $v^* : S \mapsto \mathbb{R}^1$ есть максимальная ожидаемая награда, начиная с состояния s:

$$v^*(s) = \max_{\pi} v^{\pi}(s).$$

Аналогично оптимальная стратегия π также будет определять оптимальную функцию ценности действия q^*:

$$q^*(s,a) = \max_{\pi} q^{\pi}(s,a).$$

Тогда выражение для v^* удовлетворяет следующему рекурсивному соотношению:

$$v^*(s) = \max_{a \in A} q^*(s,a) = \max_{a} \left(R(s,a) + \gamma \sum_{s' \in S} P_{sa}(s') v^*(s')\right). \tag{2.2}$$

Данное уравнение оптимальности Беллмана (2.2) дает рекурсивное определение для v^*. Интуитивно уравнение оптимальности Беллмана для v^* выражает тот факт, что ценность для текущего состояния, если агент следует оптимальной стратегии π^*, должна равняться ожидаемой будущей награде, получаемой при оптимальных действиях, начиная с текущего состояния. Согласно этому нужно действовать оптимально для первого и последующих шагов.

Также именно оптимальная стратегия $\pi^* : S \mapsto A$ определяет оптимальную функцию ценности состояния и функцию ценности действия, так что начиная с любого состояния и следуя оптимальной стратегии π_* формируется оптимальная функция ценности состояния:

$$v^{\pi_*}(s) = v^*(s) \forall s.$$

Тогда оптимальная стратегия определяется как:

$$\pi^*(s) = \arg\max_{a \in A} Q^*(s,a).$$

Имея решение для уравнения оптимальности Беллмана (2.2), можно выбирать действия согласно оптимальной стратегии π^*, которое будет удовлетворять условию оптимальности. Однако на практике найти оптимальное решения для уравнения (2.2) в аналитическом виде не представляется возможным, лишь приближенно можно найти его оптимальное значение. По этой причине основные итеративные методы ОсП нацелены на нахождении удовлетворительного приближенного значения решения уравнения оптимальности Беллмана.

2.3 Методы градиента стратегии

Методы, основанные на вычислении градиента стратегии, являются традиционными подходами к обучению агента.

Цель данных методов состоит в том, чтобы найти такое значение набора параметров θ (обычно подразумевают веса нейронной сети) для стохастической функции стратегии $\pi_\theta(a_t|s_t,\theta) = P\left[a_t = a|s_t = s, \theta_t = \theta\right]$, чтобы она максимизировала целевую функцию $J(\pi_\theta)$, зависящую от стратегии и для этого применяются различные алгоритмы оптимизации θ. Для оптимизации параметров θ используется метод градиентного подъема:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta), \tag{2.3}$$

Градиент целевой функции $\nabla_\theta J(\pi_\theta)$ — называют *градиентом стратегии*, а алгоритмы которые оптимизируют параметр θ для нахождения оптимальной стратегии таким образом называют *алгоритмы градиента стратегии* (policy gradient algorithms). Для того, чтобы применять данные алгоритмы, следует определить выражения для градиента стратегии, результат вычисления которого будет градиентом целевой функции $J(\pi_\theta)$.

Целевая функция для непрерывной среды имеет вид:

$$J(\theta) = \sum_{s_t \in S} d^\pi(s_t) V^\pi(s_t) = \sum_{s_t \in S} d^\pi(s_t) \sum_{a_t \in A} \pi_\theta(a_t|s_t) Q^\pi(s_t, a_t), \tag{2.4}$$

где $d^{\pi}(s)$ — стационарное распределение вероятности перехода состояния цепи Маркова следуя стратегии π_θ. Детальный вывод приведен в работе [10]

Тогда, используя метод градиентного подъема, можно найти значение θ, которое максимизирует (2.4). Градиент целевой функции по отношению к θ будет иметь вид:

$$\nabla J(\theta) = \sum_{s_t \in S} d(s_t) \sum_{a_t \in A} \nabla \pi(a_t|s_t) Q^{\pi}(s_t, a_t) =$$

$$= \sum_{s_t \in S} d(s_t) \sum_{a_t \in A} \pi(a_t|s_t) \frac{\nabla \pi(a_t|s_t)}{\pi(a_t|s_t)} Q^{\pi}(s_t, a_t) =$$

$$= \sum_{s_t \in S} d(s_t) \sum_{a_t \in A} \pi(a_t|s_t) \nabla \ln \pi(a_t|s_t) Q^{\pi}(s_t, a_t) =$$

$$= \mathbb{E}_{\pi_\theta} \left[\nabla \ln \pi(a_t|s_t) Q^{\pi}(s_t, a_t) \right]. \tag{2.5}$$

Полученное выражение (2.5) называют Теоремой градиента стратегии, на которой базируются основные алгоритмы градиента стратегии. Однако использовать ее на практике в таком виде не очень эффективно, так как при вычислении градиента наблюдается высокая дисперсия (результаты вычисленного градиента для близких траекторий имеют различное значение), которая замедляет процесс сходимости методов. По этой причине прибегают к методам уменьшения дисперсии вычисляемого градиента. Данная проблема возникает не только в области ОсП, аналогичные задачи по уменьшению дисперсии выборки случайной величины возникают в области математической статистики.

2.4 Актор-критик

Один из способов уменьшения дисперсии, это выбор базовой функции $b(s_t)$ и подстановка его вместо функции ценности действия Q^{π}, тогда целевая функция будет имеет вид:

$$\nabla J(\theta) \propto \mathbb{E} \left[\nabla \log \pi_\theta(a_t|s_t) (Q^{\pi}(s_t, a_t) - b(s_t)) \right].$$

Значение базовой функции может быть любым, даже Произвольной константой, главное чтобы была независимость от a_t. Добавление базовой функция в целевую функцию не дает смещения при вычислении градиента целевой функции, так как в соответствии с линейностью математического ожидания можно показать, что:

$$\mathbb{E}\left[\nabla \log \pi_\theta(a_t|s_t) b(s_t)\right] = b(s_t)\mathbb{E}\left[\nabla \log \pi(a_t|s_t)\right] =$$

$$= b(s_t) \int \frac{\nabla \pi(a_t|s_t)}{\pi(a_t|s_t)} \pi(a_t|s_t)\, da_t = b(s_t)\nabla \int \pi(a_t|s_t)\, da_t = b(s_t)\nabla 1 = 0.$$

Доказательство для бесконечного эпизода приведено в работе [7]. Уменьшение дисперсии можно показать простыми выкладками для одного эпизода:

$$\begin{aligned} &\text{Var}(\sum_{t=0}^{T-1} \nabla \log \pi(a_t|s_t)(G_t(\tau) - b(s_t))) \approx \\ &\overset{(i)}{\approx} \sum_{t=0}^{T-1} \mathbb{E}\left[(\nabla \log \pi(a_t|s_t)(G_t(\tau) - b(s)))^2\right] \approx \\ &\overset{(ii)}{\approx} \sum_{t=0}^{T-1} \mathbb{E}\left[(\nabla \log \pi(a_t|s_t))^2\right] \mathbb{E}\left[(G_t(\tau) - b(s_t))^2\right]. \end{aligned} \tag{2.6}$$

(i) В данном случае можно приближенно вычислить дисперсию суммы, вычисляя сумму дисперсий. Тогда по определению дисперсии $\text{Var}(X) := \mathbb{E}\left[X^2\right] - \left(\mathbb{E}\left[X\right]\right)^2$ и остается левая часть уравнения, так как для правой части было показано, что базовая функция не добавляет смещения.

(ii) В силу независимости значений входящих в математическое ожидание, можно разделить математическое ожидание. Как видно уменьшение значения $\mathbb{E}\left[(G_t(\tau) - b(s_t))^2\right]$ снижает значение дисперсии, путем выбора базовой функции $b(s)$.

На практике за базовую функцию принимают значение $b(s_t) = V^\pi(s_t)$ и можно определить *функцию превосходства*, которая показывает разницу между функцией ценности действия и функцией ценности состояния:

$$A_\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t).$$

Интуитивно данная функция определяет насколько хорошим будет действие a по сравнению со средним значением награды. Таким образом градиент целевой функции для метода актор-критик основанный на функции превосходства (Advantage Actor Critic, A2C) будет иметь вид:

$$\nabla J(\theta) \approx \mathbb{E}\left[\nabla \log \pi(a_t|s_t) \cdot A_\pi(s_t, a_t)\right].$$

Данный класс методов имеет название «методы актор-критик», где для вычисления целевой функции и нахождения его градиента используют приближенную функцию ценности состояния (или действия) именуемую как *критик* и приближенное значение стратегии π называемое как *актор*.

2.5 Алгоритм оптимизации стратегии на основе доверительного региона

В алгоритме «оптимизация стратегии на основе доверительного региона» (Trust Region Policy Optimization) используется идея обновления стратегии на основе максимального допустимого рассогласования двух стратегий. Развитие данной идеи используется в работе [8] для формирования метода под названием *«непосредственная оптимизация стратегии»* (Proximal Policy Optimization, PPO), который будет рассмотрен в следующем подразделе.

Целевая функции, которую нужно максимизировать, имеет вид:

$$J(\theta_k, \theta) = \mathbb{E}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} \hat{A}_{\pi_{\theta_k}}(s, a)\right],$$

где θ_k – параметры прошлой стратегии, θ – параметры текущей стратегии, $\hat{A}_{\pi_{\theta_k}}(s, a)$ – аппроксимация функции превосходства на основе прошлой стратегии.

Критерий на основе которого задается ограничение на обновление функции — это расстояние Кульбака-Лейблера (KL-divergence) или по другому называемое *относительной энтропией*. Ее цель дать скаляр, который бы описывал различие двух вероятностных распределений друг от друга.

Для распределений P и Q с плотностями вероятности p и q определяется как:

$$D_{\mathrm{KL}}(P||Q) = \mathbb{E}_{x\sim p(x)}\left[\log\frac{p(x)}{q(x)}\right].$$

Задаваясь ограничением δ на максимальную относительную энтропию для двух стратегий текущей и прошлой, можно определить максимальное рассогласование текущей стратегии относительно старой:

$$\mathbb{E}\left[D_{\mathrm{KL}}(\pi_{\theta_k}||\pi_\theta)\right] \leqslant \delta.$$

Так при итеративном обновлении весов текущая стратегия гарантированно не будет выполнять большие обновления по отношению к старой стратегии. Таким образом алгоритм оптимизации стратегии на основе доверительного региона может гарантировать монотонное улучшение стратегии. Детальное доказательство алгоритма приведено в работе [6].

2.6 Непосредственная оптимизация стратегии

В работе [8] представляется один из популярных методов обучения с подкреплением, основанный на методах градиента стратегии — непосредственная оптимизация стратегии. Данный алгоритм будет использоваться для решения поставленной задачи и далее будет называться «PPO-алгоритм».

Алгоритмы градиента стратегии вычисляют градиент целевой функции с помощью упрощающей функции градиента (gradient estimator):

$$\nabla\hat{J}^{PG}(\theta) = \hat{\mathbb{E}}_t\left[\nabla_\theta\log\pi_\theta(a_t|s_t)\hat{A}_t\right],$$

где $\hat{A}_t$ — приближенное нахождение функции превосходства в момент времени t, π_θ — стохастическая стратегия. Значение данного градиент используется для оптимизации целевой функции вида:

$$J^{PG}(\theta) = \hat{\mathbb{E}}_t\left[\log\pi_\theta(a_t|s_t)\hat{A}_t\right],$$

Как было отмечено, для методов, использующих доверительный регион (Trust Region Methods), определяют «суррогатную» целевую функцию для того, чтобы ограничить величину обновления параметров стратегии:

$$J^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right]$$

В PPO-алгоритме модифицируют целевую функцию метода доверительных регионов, введя ограничение на изменения отношения стратегии $r_t(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$ в диапазоне $1 - \varepsilon \leqslant r_t(\theta) \leqslant 1 + \varepsilon$:

$$J^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \varepsilon, 1 + \varepsilon) \right) \right],$$

где ε — гиперпараметр ограничения области, обычно принимается как 0.2. Вводится дополнительная регуляризация целевой функции добавлением энтропийного штрафа стратегии — $S\left[\pi_\theta\right](s)$, добавляющего случайность в действия, чтобы производить исследование среды эффективнее и не эксплуатировать схожие действия. Также вводится квадратичная ошибка аппроксимации функции ценности $J^{\text{VF}}(\theta) = \left(V_\theta(s_t) - V_t^{\text{targ}}\right)^2$. Тогда окончательно целевая функция для PPO-алгоритма имеет вид:

$$J_t^{\text{CLIP + VF + S}}(\theta) = \hat{\mathbb{E}}_t \left[J_t^{\text{CLIP}}(\theta) - c_1 J^{\text{VF}}(\theta) + c_2 S\left[\pi_\theta\right](s) \right], \qquad (2.7)$$

где c_1, c_2 — константы (гиперпараметры). Также для приближенного вычисления $\hat{A}_t$ в «суррогатной» целевой функции $J_t^{\text{CLIP}}(\theta)$ будем использовать упрощенное вычисление обобщенной функции превосходства (Generalized Advantage Estimator), определенной в работе [8]:

$$\hat{A}_t = -V(s_t) + R_t + \gamma R_{t+1} + \cdots + \gamma^{T-t+1} R_{T-1} + \gamma^{T-t} V(s_T). \qquad (2.8)$$

Псевдокод, описывающий PPO-алгоритм, представлен ниже.

Алгоритм 1. Псевдокод PPO-алгоритма

Инициализация сети стратегии, функции ценности с параметрами θ, M — число итераций, N — число эпох
Цикл каждой итерации = 1 до M **выполнять**
    Инициализировать буфер $\mathcal{B}$ для хранения состояний эпизода
    До тех пока не закончился эпизод **выполнять**
        Выбрать действие a_t на основе $\pi_\theta(a_t|s_t)$
        Сохранять s_t, a_t, r_t в буфер $\mathcal{B}$
        Переход среды в состояние s_{t+1}
    Конец цикла
    Вычислить приближенную функцию превосходства $\hat{A}$
    Цикл каждой эпохи = 1 до N **выполнять**
        Разделить буфер $\mathcal{B}$ на мини-сеты K
        Цикл каждого мини-сета = 1 до K **выполнять**
            Вычислить целевую функцию $J^{\text{CLIP + VF + S}}(\theta)$
            Обновить веса θ стратегии и функции ценности
        Конец цикла
    Конец цикла
Конец цикла

3 ПРИМЕНЕНИЯ МЕТОДА НЕПОСРЕДСТВЕННОЙ ОПТИМИЗАЦИИ СТРАТЕГИИ К ЗАДАЧЕ ОТСЛЕЖИВАНИЯ

В предыдущем разделе была изложена основная теория, необходимая для реализации рассматриваемого алгоритма. Далее будут приведены особенности реализации PPO-алгоритма для задачи отслеживания продольного углового движения самолета.

3.1 Описание задачи управления

Необходимо реализовать управление стабилизатором φ_{act}, для того чтобы свести к минимуму ошибку для каждого момента времени t:

$$\omega_{z_{err}} = \omega_z - \omega_{z_{ref}},$$

где $\omega_{z_{ref}}$ — задающий сигнал по угловой скорости тангажа.

3.2 Описание задачи в терминах обучения с подкреплением

Для корректного обучения агента нужно определиться, как будет устроена среда. В нашем случае в качестве среды выступает объект управления, определенный в разделе 1. В вектор состояния среды s_t будут входить такие параметры как:

$$s_t = (H, \omega_z, V, \omega_{z_{ref}}, \omega_{z_{err}})_t,$$

где t — момент времени. Параметры вектора среды определяются на основе интегрирования уравнения (1.1), откуда вектор состояния ЛА имеет вид:

$$x_t = (V_x, V_y, H, \vartheta, V, \alpha, \omega_z)_t.$$

Действие, которые может выполнять агент над средой — это выдача командного сигнала привода стабилизатора $a = \varphi_{ref}$, который изменяется в диапазоне $-25 \ldots 25$ град.

Для улучшения обучения параметры H, ω_z, V из вектора состояния среды s_t нормируется в диапазоне от 0 до 1, в соответствии с реальным диапазоном, определенным в таблице 3.1. Вектор состояния, содержащий нормированные параметры, будет обозначаться как $\bar{s}$.

Таблица 3.1 – Физический диапазон переменных состояния

Состояние	Мин.	Макс.
H, м	0	35000
ω_z, град/с	-150	150
V, м/с	0	500

Последовательный процесс взаимодействия агента со средой от начального состояния s_0 до конечного состояния среды s_T называют эпизодом. В нашем случае эпизод состоит из 1000 взаимодействий агента со средой до достижения $t_{\text{кон.}} = 10$ с, с шагом дискретности $\Delta t = 0.01$ с. Данный промежуток времени выбран с целью репрезентативной демонстрации динамических свойств самолета.

Инициализация начального состояния объекта x_0 в начале каждого эпизода производится равновероятно в диапазоне в соответствии с таблицей 3.2, где $x_{\text{НГ}}$ — нижняя граница случайного выбора начального состояния, $x_{\text{ВГ}}$ — верхняя граница случайного выбора начального состояния. Состояния, не указанные в таблице 3.2, принимают значение 0.

Таблица 3.2 – Диапазон стартовых состояний при начале каждого эпизода

Состояние	$x_{\text{НГ}}$	$x_{\text{ВГ}}$
Oy, м	2000	4000
ω_z, град/с	-5	5
V, м/с	90	250
α, град	-5	5

Завершается эпизод раньше времени $t_{\text{кон.}}$, если параметры вектора состояния объекта управления H, ω_z будут находится вне диапазона:

$$s_{\text{доп.}} = \begin{cases} 300 \text{ м} \leqslant H \leqslant 30000 \text{ м}, \\ |\omega_z| \geqslant 50 \text{ град/с}. \end{cases} \tag{3.1}$$

Данные значения приняты для повышения числа встречаемых информативных эпизодов и исключения выхода на диапазон больших угловых скоростей.

3.3 Функция вознаграждения

На каждом временном шаге t вычисляется функция вознаграждения, зависящая от ошибки слежения $\omega_{z_{err}}$:

$$R_t = 1 - \frac{\frac{|\omega_{z_{err}}|}{k}}{1 + \frac{|\omega_{z_{err}}|}{k}}, \tag{3.2}$$

где коэффициент k определяет величину ошибки $|\omega_{z_{err}}|$. При этом агент будет получать половинную от максимальной награды $R_t = 1$ (см. рис. 3.1). Данная функция вознаграждения, есть репрезентация нашей цели для достижения минимальной ошибки слежения, её агенту нужно максимизировать. Если отслеживать сигнал с ошибкой 0 для каждого момента времени, то максимальная суммарная награда за эпизод будет равняться 1000 и в таком случае достигается оптимальность.

Дополнительно агент получает вознаграждение, если решение станет неустойчивым:

$$R_{\text{огр}_t} = \begin{cases} -1000, \text{ если } H \leqslant 300 \text{ м или } H \geqslant 30000 \text{ м}, \\ -1000, \text{ если} |\omega_z| \geq 50 \text{ град/с}. \end{cases} \tag{3.3}$$

Вознаграждение (3.3) агент получает в случае завершения эпизода по причине выхода из допустимой области $s_{\text{доп.}}$ (3.1), вследствие которого эпизод завершается раньше конечного времени $t_{\text{кон.}}$.

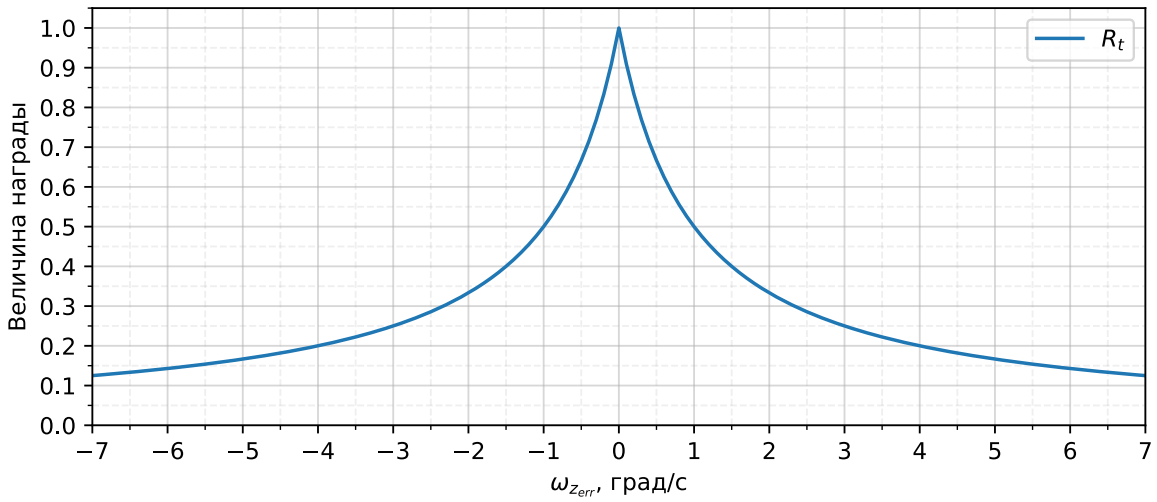


Рисунок 3.1 – Использованная функция вознаграждения

3.4 Критерий для оценки эффективности управления

Для количественной оценки синтезированного контроллера будем использовать следующие критерии:

– Средняя абсолютная ошибка (САО) — усредненное значение абсолютной ошибки между значением y и приближенным значением $\hat{y}$:

$$\text{САО}(y, \hat{y}) = \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{n}.$$

– Среднеквадратическая ошибка (СКО) — усредненный квадрат разности между реальным значением y и приближенным значением $\hat{y}$:

$$\text{СКО} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2.$$

В нашей задаче $y = \omega_{z_{\text{ref}}}$ и $\hat{y} = \omega_z$.

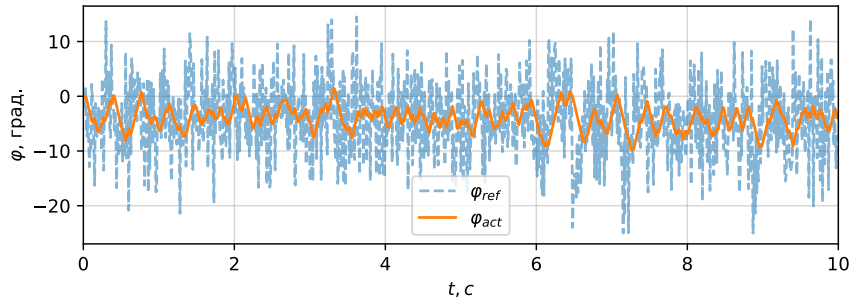
3.5 Формирование действий

В исходной версии алгоритма в процессе обучения агента со стохастической стратегией действие выбирается из нормального распределения $\mathcal{N}(\mu, \sigma)$:

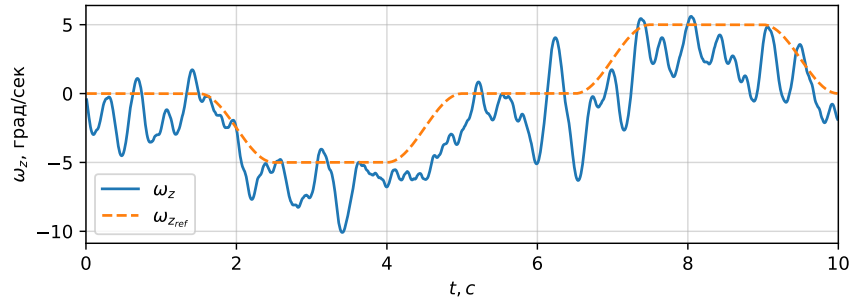
$$a_t \sim \mathcal{N}(\mu_t, \sigma_t),$$

где μ — среднее значение, σ — среднеквадратическое отклонение. Данные параметры нормального распределения определяются нейронной сетью на основе состояния среды s_t.

Используя такой подход в выборе действий, агент в процессе обучения не может достичь желаемой стратегии, так как управляющие действия от агента имеют стохастический характер, то есть для определенного состояния среды s_t возможны разные управляющие действия, не позволяющие должным образом обучить агента. Иллюстрация стохастических управляющих действий для одного эпизода приведена на рис. 3.2.



(a) Характер стохастических управляющих действий



(b) Изменение угловой скорости тангажа ω_z

Рисунок 3.2 – Пример обученного агента действия которого, выбираются на основе нормального распределения

3.5.1 Выбор действия на основе состояния

Подойти к решению проблемы стохастических действий можно разными способами, например, вводом фильтра для управляющего сигнала или же подбором функции вознаграждения. Успешность применения таких подходов сильно зависит от решаемой задачи. Хотелось бы использовать

метод, обеспечивающий плавность и постоянство управляющих действий вне зависимости от рассматриваемой задачи. Метод под названием обобщенное исследование на основе состояния (Generalized State-Dependent Exploration, gSDE) [5] позволяет добиться таких целей.

Идея метода заключается в добавлении к детерминированному действию $\mu(s_t,\, \theta_\mu)$ шума ε, как функции, зависящей от текущего состояния s_t при обучении агента, тогда действие определяется как:

$$a_t = \mu(s_t;\, \theta_\mu) + \varepsilon(s_t; \theta_\varepsilon),$$

в данном случае $\varepsilon(s_t; \theta_\varepsilon) = \theta_\varepsilon z_\mu(s_t)$, $\mu(s) = \theta_\mu z_\mu(s; \theta_{z_\mu})$, где z_μ – выход последнего слоя нейронной сети (НС) θ_μ, который определяет детерминированную составляющую μ, а стохастическая составляющая шума ε вводится с помощью параметра θ_ε, который в свою очередь выбирается из нормального распределения $\theta_\varepsilon \sim \mathcal{N}(0, \sigma^2)$. Обновление параметра θ_ε выполняется в начале каждого шага эпизода [5].

В данной работе будет использована реализация gSDE, где действие будет определяться как:

$$a_t = \text{mean}(\mathcal{N}(\mu(s_t, \theta_\mu);\, \sigma(s_t, \theta_\sigma))) + \mathcal{N}(0, \varepsilon(s_t; \theta_\varepsilon)) = \mu(s_t, \theta_\mu) + \mathcal{N}(0, \varepsilon(s_t; \theta_\varepsilon)),$$

где параметры нормального распределения μ, ε, определяются двумя нейронными сетями θ_μ, θ_σ на основе состояния s_t. Определение шумовой составляющей $\mathcal{N}(0, \varepsilon(s_t; \theta_\varepsilon))$ будет производиться в начале каждого эпизода.

Использование такого управляющего действия позволяет избавиться от «зашумленности» в действиях агента и ускорить процесс поиска оптимальной стратегии в процессе обучения агента.

3.6 Описание тренировочного цикла

Чтобы обучить агента, требуется реализовать цикл обучения агента, где происходит обновление весов нейронных сетей критика и актора. Данный процесс непосредственно влияет на эффективность обученного агента. Чтобы пройти процесс обучения нейронных сетей, вначале нужно определить значения гиперпараметров — величины которые нужно задать непосредственно до обучения. Основные гиперпараметры определены в таблице 3.3.

Таблица 3.3 – Описание основных гиперпараметров

Название	Описание
Темп обучения	Параметр α в уравнении (2.3), определяет величину шага на каждой итерации при обновлении весов сети.
Количество шагов обучения	Продолжительность обучения, где каждый шаг — это одно взаимодействие агента со средой (рис. 2.1), равен 200000.
Размер буфера $\mathcal{B}$	Определяет количество сохраняемых взаимодействий, необходимых для цикла обновления весов, равен 2048.
Количество эпох N	Количество прогонов буфера для обновления весов сети, равно 10.
Размер мини-выборки K	Определяет количество подаваемых взаимодействий для обновления весов, равен 256.
γ	Коэффициент дисконтирования получаемой награды в уравнении (2.8), равен 0.99.

Коэффициенты $c_1 = 0.5$, $c_2 = 0$	Входят в целевую функцию (2.7). В нашей задаче энтропия стратегии не будет вносить вклад в целевую функцию, так как среда от эпизода к эпизоду имеет случайное распределение инициализируемых состояний (см. таблицу 3.2).

Значения данных гиперпараметров напрямую влияют на характер процесса обучения. Подбор числовых значений гиперпараметров – задача нетривиальная. Эти значения зависят от решаемой задачи и сложности сети, они подбираются эмпирически. Однако существуют различные методы оптимизации значений гиперпараметров, но все они должны пройти минимальное количество шагов обучения, чтобы сделать вывод о подобранных гиперпараметрах, что требует вычислительной мощности и времени для подбора значений гиперпараметров. Так как методы вычисления градиента для выполнения обратного распространения ошибки нейронной сети в нашем случае оказались чувствительны к ошибке с плавающей запятой, то гиперпараметры, подобранные для одной ЭВМ с одной операционной системой, могут определять совсем иной характер обучения агента на другой ЭВМ. Данная проблема воспроизведения обучения широко известна в области машинного обучения [2]. Таким образом выбор конкретных значений гиперпараметров тесно связан с решаемой задачей и ЭВМ, на которой производится цикл обучения.

Представление программной реализации алгоритма в виде функциональной схемы показано на рис. 3.3.

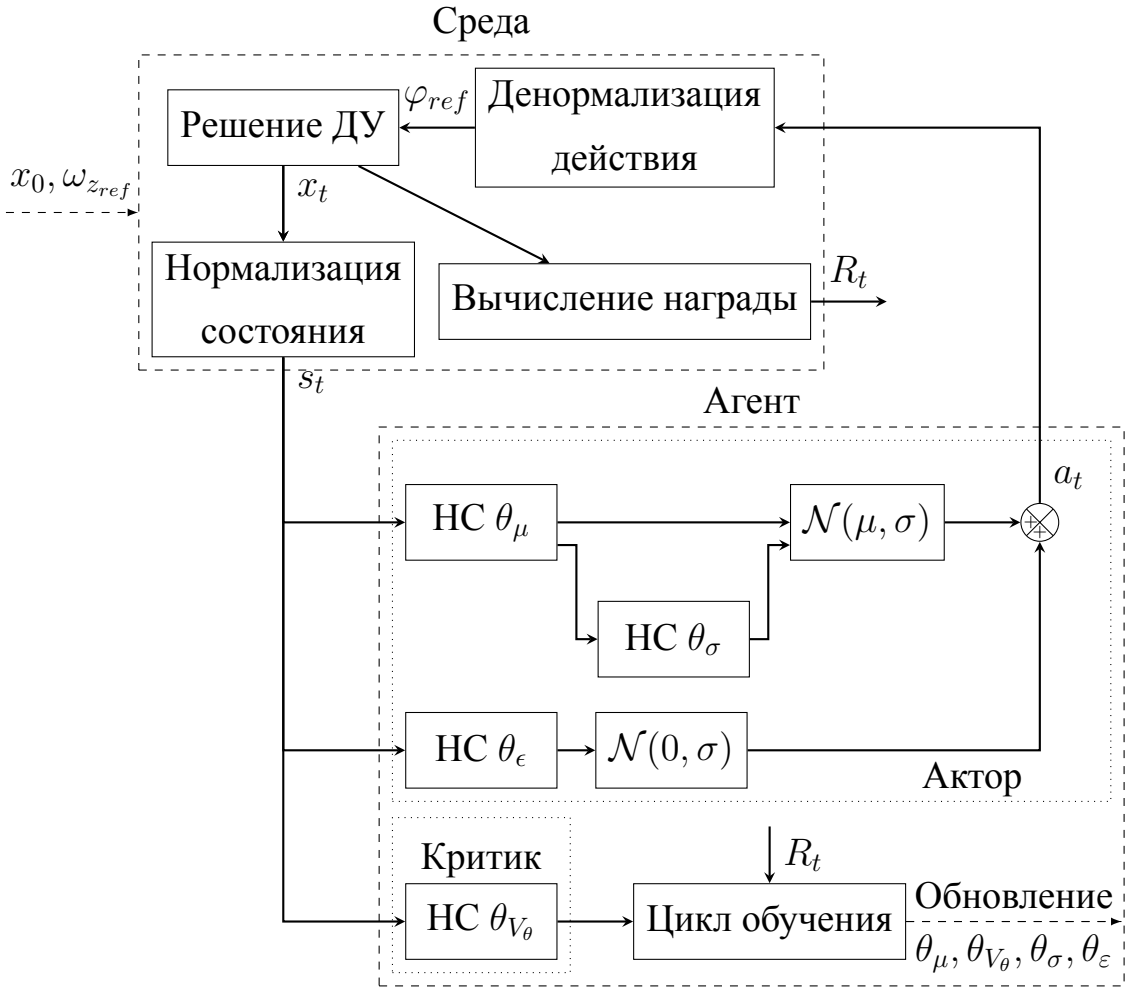


Рисунок 3.3 – Функциональная схема взаимодействия PPO-алгоритма со средой

В состав среды входят такие процессы как:

– *Денормализация.* На базе нормированного действия агента a_t, которое определено в диапазоне $-1 \ldots 1$, производится масштабирование в командный сигнал отклонения стабилизатора φ_{ref}, определенный в диапазоне $-25 \ldots 25$ град. Данный способ нормирования управляющих действий, получаемых от агента, позволяет улучшить стабильность обучения.

– *Решение дифференциальных уравнений продольного движения.* Производится вычисление следующего состояния ЛА x_{t+1}, на основе управляющего действия φ, начиная с начального состояния x_0 методом Эйлера.

– *Вычисление награды.* Рассчитывается награда, определяемая уравнениями (3.2), (3.3) на основе состояния x_{t+1}. Награда используется агентом исключительно для обучения сети.

– *Нормализация состояния.* Происходит преобразование вектора состояния ЛА x в вектор состояния среды s:

$$x = (V_x, V_y, H, \vartheta, V, \alpha, \omega_z) \mapsto s = (H, \omega_z, V, \omega_{z_{ref}}, \omega_{z_{err}}).$$

Нормирование вектора состояния среды s производится в соответствии с таблицей 3.1 и передача $\bar{s}$ агенту. Нормировка данных – это стандартная практика при обучении нейронных сетей методом обратного распространения ошибки, которая позволяет улучшить стабильность обучения.

Агент отвечает за формирование действий на основе полученного нормированного состояния $\bar{s}_t$ от среды. Для реализации агента нужны:

– *Нейронные сети актора.* Сеть θ_μ, имеющая 2 скрытых слоя с 64 нейронами в каждом слое, активационная функция – гиперболический тангенс. Данная сеть отвечает за побор коэффициента μ для функции нормального распределения $\mathcal{N}(\mu, \sigma)$, которая определяет действия агента. Упрощенная структурная схема нейронной сети представлена на рис. 3.4.

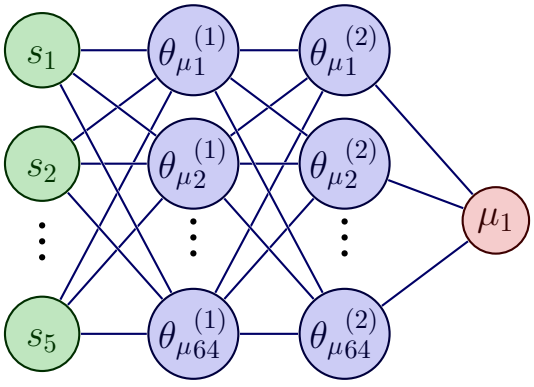


Рисунок 3.4 – Структурная схема сети θ_μ

Сеть θ_σ является линейным нейроном с размерностью входного и выходного вектора равной единице, который записывается как:

$$\sigma = w_1\mu + b_1,$$

а схематическое представление показано на рис. 3.5.

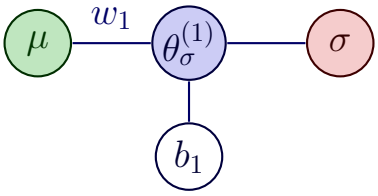


Рисунок 3.5 – Структурная схема сети θ_σ

Данная сеть отвечает за подбор коэффициента среднеквадратического отклонения σ для нормального распределения $\mathcal{N}$ на базе полученного значения μ.

На базе первого состояния $\bar{s}_0$ в начале каждого эпизода производится выбор коэффициента σ_{SDE} с помощью другой нейронной сети θ_ε аналогичной по архитектуре θ_μ. Шум ε в данном случае выбирается из нормального распределения с математическим ожиданием 0 и среднеквадратическим отклонением σ_{SDE}.

Сумма шума ε и математического ожидания распределения $\mathcal{N}$ определяют действие a_{t+1}.

– *Нейронная сеть критика.* По архитектуре аналогична сети θ_μ. Данная сеть отвечает за аппроксимацию функции ценности $V(s)$ для состояния $\bar{s}_t$, необходимой для определения целевой функции (2.7).
– *Цикл обучения.* По заполнении буфера $\mathcal{B}$ выполняется цикл обновления весов нейронных сетей актора и критика в соответствии с PPO-алгоритм (см. алгоритм 1). Используемый метод для оптимизации весов нейронной сети — Adam [3].

Реализация алгоритма на языке программирования Python представлена в приложении А.

3.7 Обучающий сигнал

Чтобы обучить агента, способного к адаптации к различным вариациям командного сигнала, нужно их представить в процессе тренировки для каждого эпизода. В каждом эпизоде агент будет обучаться на отслеживании следующих тестовых сигналов:

а) ступенчатый сигнал со случайной амплитудой в диапазоне $-20 \ldots 20$ град/с;

б) синусоидальный сигнал со случайной частотой от $0.125 \ldots 0.75$ Гц и амплитудой $-20 \ldots 20$ град/с;

в) плавный ступенчатый сигнал, определенный кусочно-линейной функцией вида:

$$y(t) = \begin{cases} -\frac{1}{2}(\cos{(\frac{1}{w_r}\pi(t - t_0))} - 1), & t_0 < t < t_0 + w_r \\ A, & t_0 + w_r \leqslant t \leqslant t_0 + w_r + w \\ -\frac{1}{2}(\cos{(\frac{1}{w_r}\pi(t - t_0 - w))} - 1), & t_0 + w_r + w < t < t_0 + 2w_r + w \end{cases}$$

где $w_r = 1$ с — время участка подъема, $t_0 = 0$ с — время начала сигнала, A — амплитуда ступенчатого сигнала определяется равномерным распределением в диапазоне $-20 \ldots 20$ град/с, $w = 1$ с — длительность установившегося участка. Визуализация данной функции с её параметрами представлена на рис. 3.6.

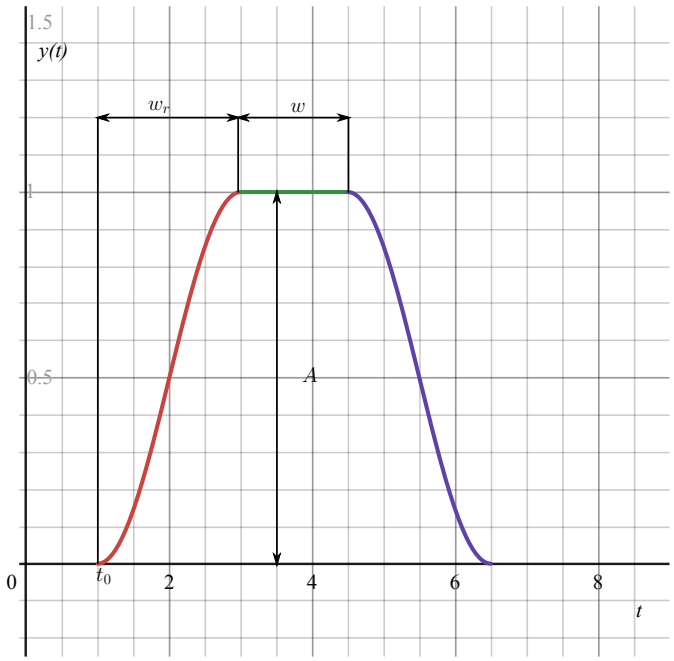


Рисунок 3.6 – Определение параметров плавного ступенчатого сигнала

Случайно комбинируя определенные выше сигналы в один, получаем обучающий сигнал, представленный на рис. 3.7, с длительностью сигнала – один эпизод.

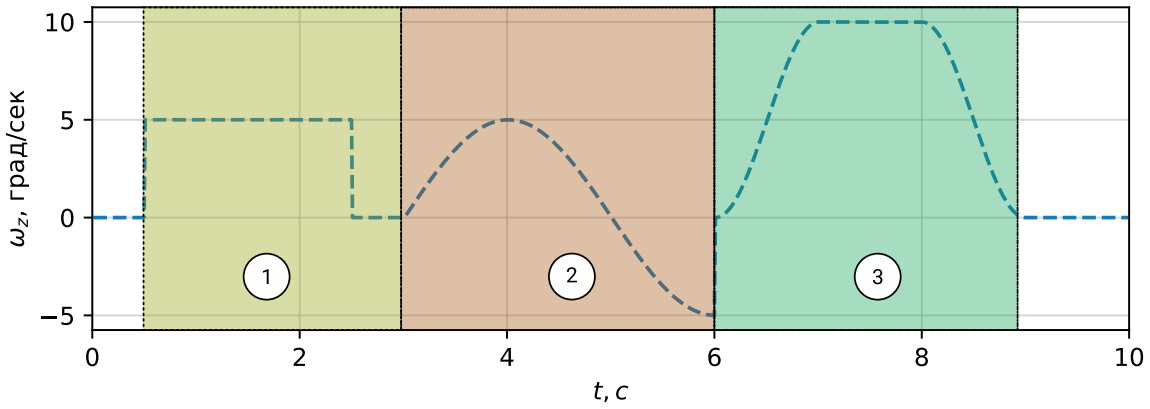


Рисунок 3.7 – Пример обучающего сигнала, где в зонах 1, 2, 3 располагаются тестовые сигналы

4 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ СФОРМИРОВАННОГО КОНТРОЛЛЕРА УГЛОВОЙ СКОРОСТИ ТАНГАЖА

По завершении процесса обучения агент спустя 200000 взаимодействий со средой набирает в среднем около 650 единиц суммарной награды за эпизод, что означает значение среднеквадратической ошибки отслеживания меньше 1 град/сек при отслеживании обучающего сигнала (см. рис.3.7). Изменение суммарной награды в процессе обучения представлено на рис. 4.1.

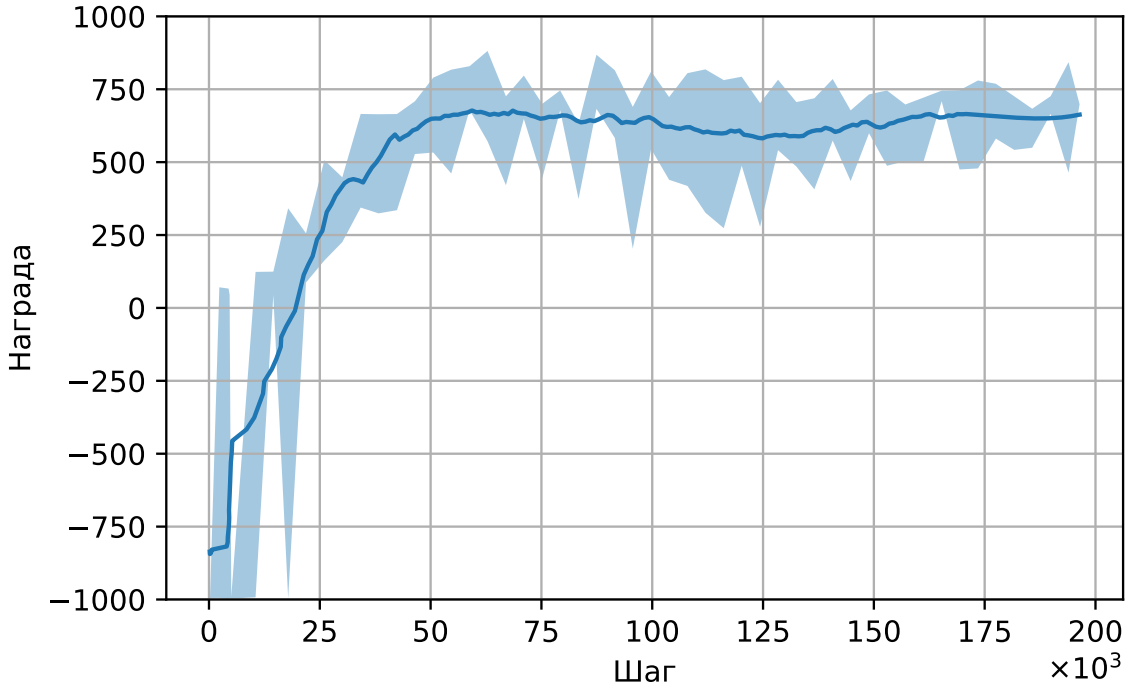


Рисунок 4.1 – Синяя линия – сглаженное значение суммарной награды за эпизод в процессе тренировки агента; Синяя область – разброс значений суммарной награды за эпизод

Обучение на большем числе шагов с текущим значением гиперпараметров не дают значительных улучшений в получаемой награде.

Результаты работы обученного агента после цикла обучения будут представлены в подпунктах данного раздела. Для всех экспериментов начальное состояние было выбрано $x_0 = (H = 2500$ м$, V = 175$ м/с$, \vartheta = 0$ град$, \omega_z = 0$ град/с$, \alpha = 0$ град$)$, $u_0 = (\varphi_{act} = 0$ град$)$ и длительность

сценария $t = 10$ с. Данные начальные параметры находятся вне балансировочного состояния, поэтому в начале каждого эксперимента возникает возмущение, которое нужно устранить агенту.

4.1 Тестовые сценарии

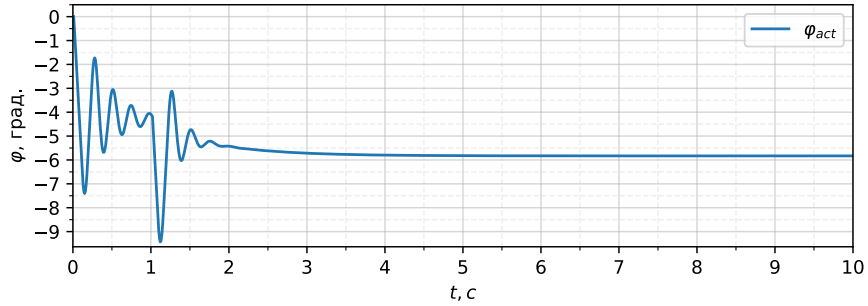
Обученный агент был протестирован для следующих сценариев:

- отслеживание задающего ступенчатого сигнала с амплитудой 5 град/сек (рис. 4.2);
- отслеживание ряда задающих плавных ступенчатых сигналов с амплитудой 5 град/сек (рис. 4.3);
- отслеживание задающего синусоидального сигнала с частотой 0.25 Гц и амплитудой 5 град/сек (рис. 4.4);
- отслеживание одной комбинации задающих сигналов, использованных для тренировки (рис. 4.5).

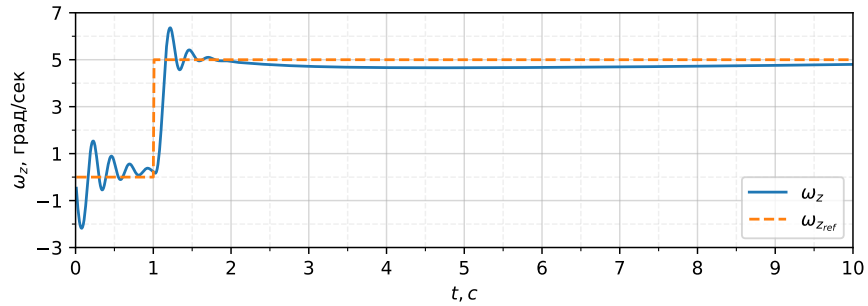
Определение данных сигналов было представлено в разделе 3.7. По результатам отработок в таблице 4.4 определены основные критерии для оценки качества отслеживания сигнала.

Таблица 4.4 – Показатели качества отслеживания для тестовых сценариев

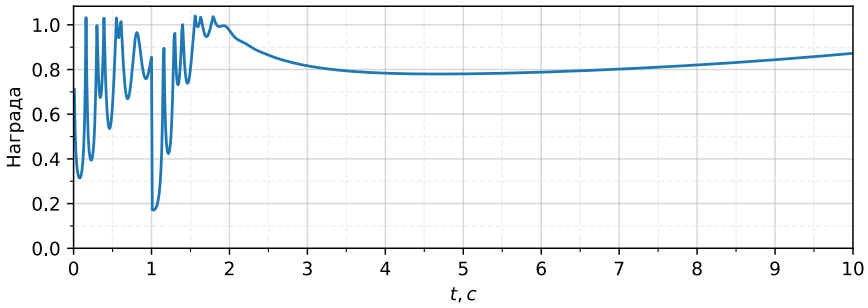
Сценарий	САО	СКО
Рис. 4.2	0.328	0.324
Рис. 4.3	0.258	0.122
Рис. 4.4	0.212	0.125
Рис. 4.5	0.526	0.959



(a) Управляющие действия

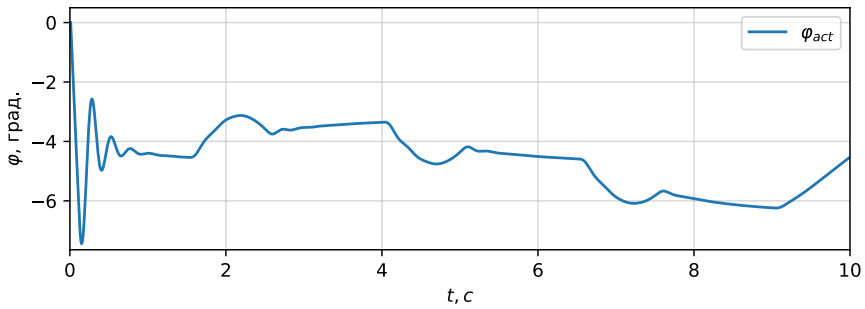


(b) Изменение угловой скорости тангажа ω_z

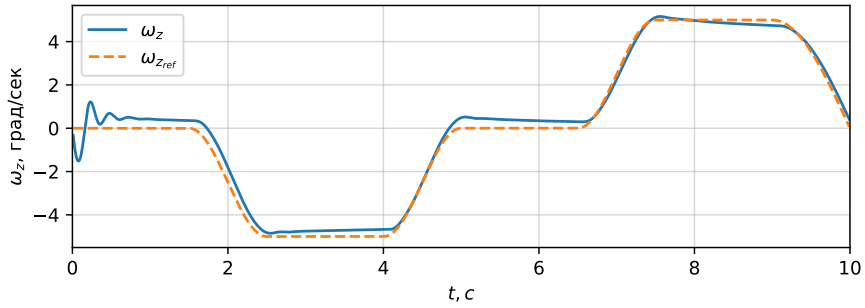


(c) График награды

Рисунок 4.2 – Отслеживание задающего ступенчатого сигнала по угловой скорости тангажа

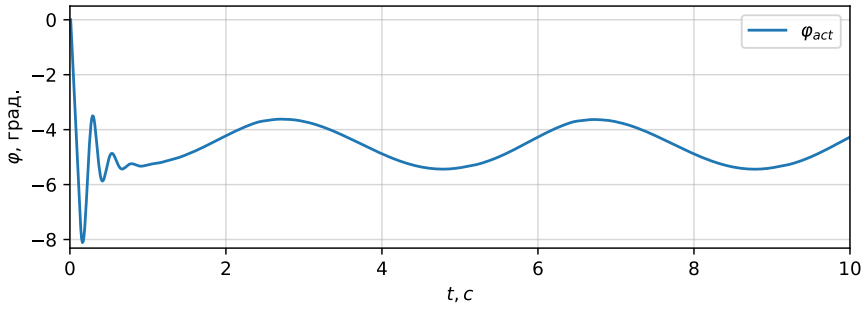


(a) Управляющие действия

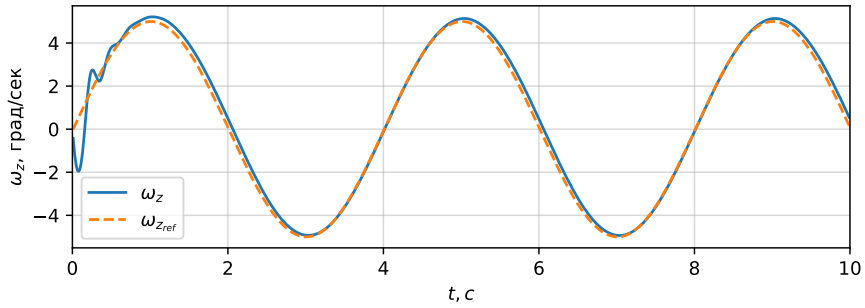


(b) Изменение угловой скорости тангажа ω_z

Рисунок 4.3 – Отслеживание серии задающих ступенчатых сигналов по угловой скорости тангажа

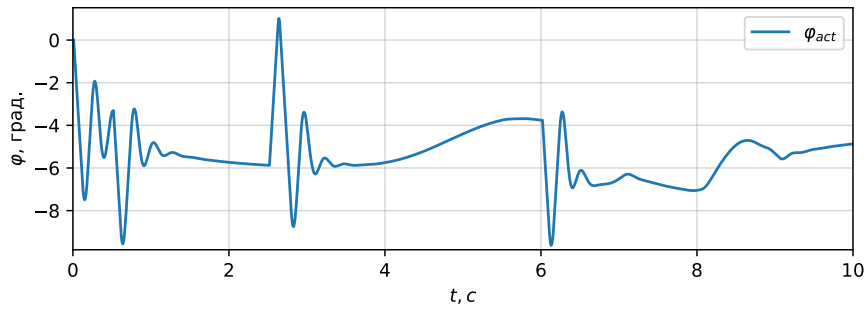


(a) Управляющие действия

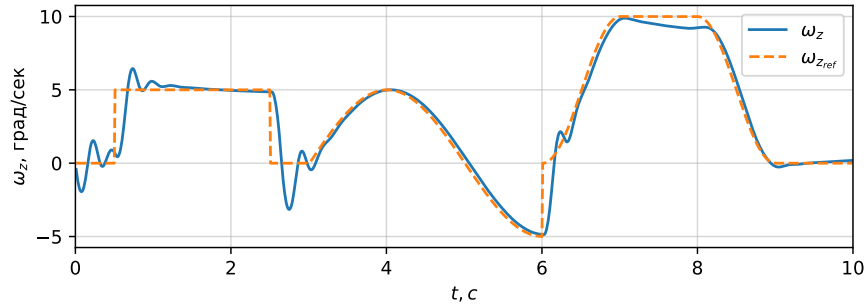


(b) Изменение угловой скорости тангажа ω_z

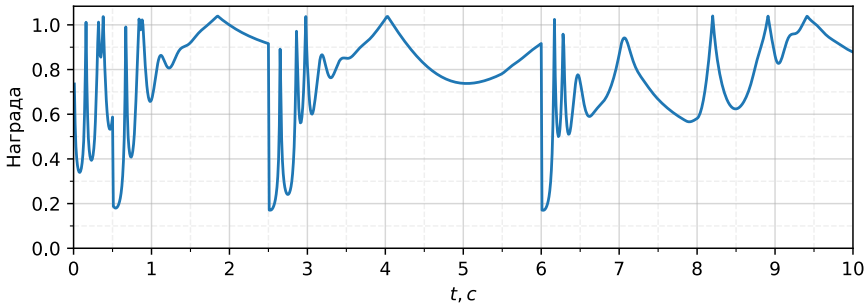
Рисунок 4.4 – Отслеживание задающего синусоидального сигнала по угловой скорости тангажа с частотой 0.25 Гц



(a) Управляющие действия



(b) Изменение угловой скорости тангажа ω_z



(c) График награды

Рисунок 4.5 – Отслеживание задающего сигнала по угловой скорости тангажа, использованного для тренировки

По результату отработки для отслеживания ступенчатого сигнала (см. рис. 4.2) установившаяся статическая ошибка слежения составляет порядка 0.3 град/с и время регулирования 0.5 с, что допустимо. Аналогичная статическая ошибка возникает при отслеживании ряда ступенчатых сигналов (см. рис. 4.3). При отработке тренировочного сигнала (см. рис. 4.5) обученный набрал 775 единиц суммарной награды, что удовлетворительно в данном случае. Обученный агент не в состоянии полностью устранить статическую ошибку для слежения за установившимся сигналом, так как не обладает информацией об интегральной составляющей ошибки слежения.

4.2 Адаптация вне обучающей выборки

Проведем серию экспериментов для диапазона начальных состояний $H = (2000 \dots 10000)$ м, $V = (90 \dots 340)$ м/с, чтобы получить график критерия САО зависящий от начального состояния H, V, в задаче отслеживании задающего синусоидального сигнала, приведенного на рис. 4.4. Полученный график представлен на рис. 4.6.

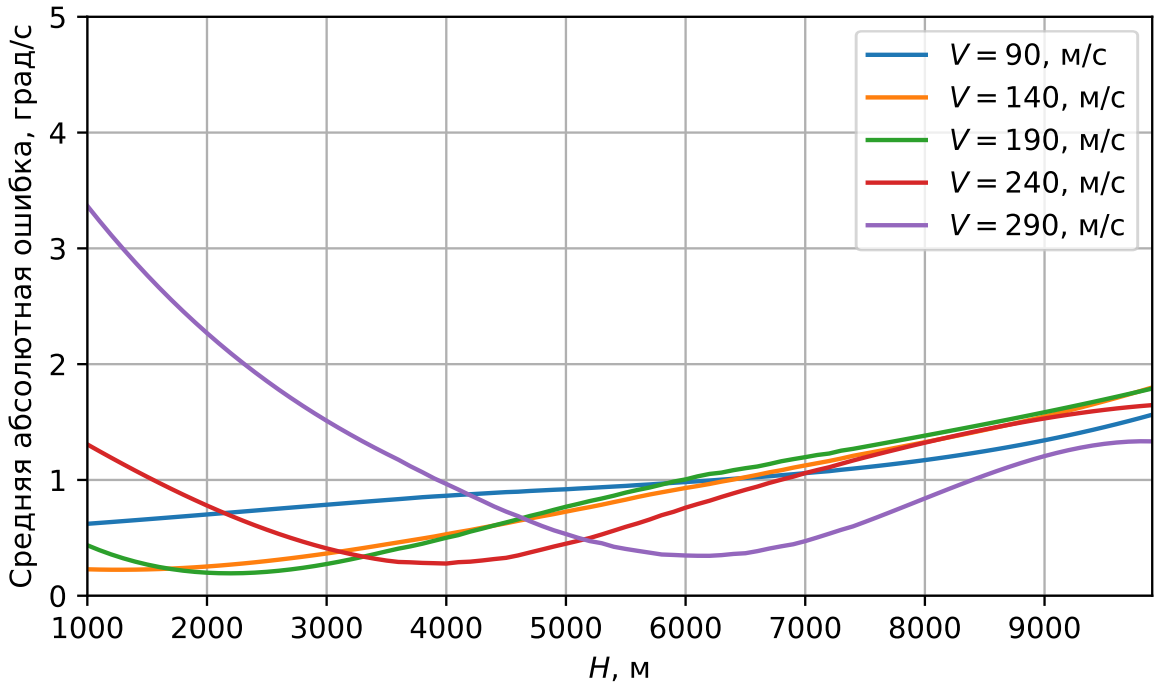
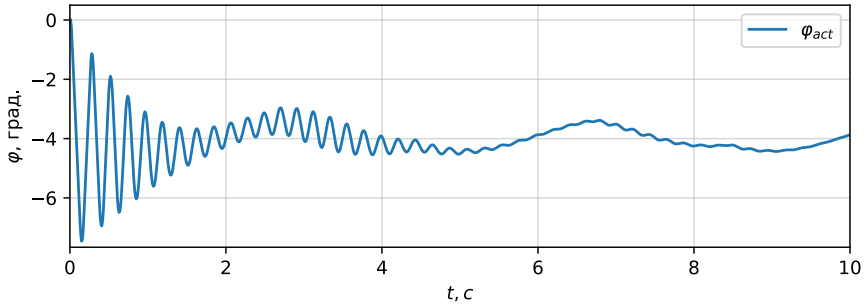


Рисунок 4.6 – График критерия САО зависящего от H при вариации V

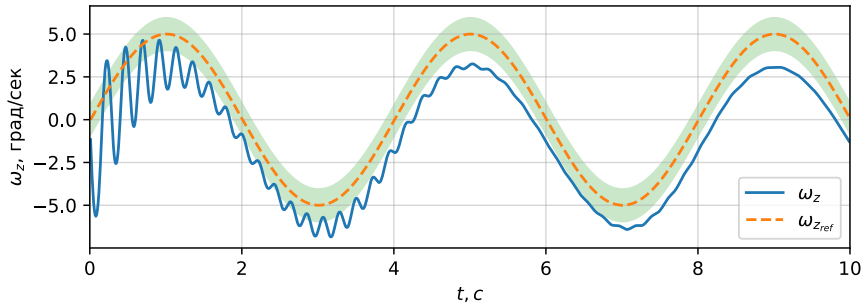
Как видно из графика на рис. 4.6, средняя абсолютная ошибка составляет не более 0.6 град/с для значений высот, находящихся в обучающей выборке в соответствии с таблицей 3.2. Для состояний вне тренировочного диапазона по значению высоты, наблюдается линейное увеличение САО вплоть до 1.8 град/с для высоты 10000 м. Для скоростей находящихся вне тренировочного диапазона ($\geqslant$ 250 м/с), проявляется неустойчивость в управлении на малых высотах.

Объясняется это отсутствием взаимодействием агента с динамикой в данном диапазоне, так как обучение проводилось для состояний в диапазоне, указанном в таблице 3.2. Для состояний со значением такого

критерия для больших высот, характерны затухающие колебания в управляющих действиях и значительная статическая ошибка, как например, на рис. 4.7. Также увеличивается статическая ошибка слежения для значения высот больше $H > 6000$ м рис. 4.8 порядка 2 град/с. Данная статическая ошибка сохраняется и при слежении за сигналом большей амплитуды, тогда при увеличении амплитуды в два раза (до значения 10 град/сек) критерий САО остается постоянной, пример на рис. 4.9.



(a) Управляющие действия



(b) Изменение угловой скорости тангажа ω_z. Зеленая область обозначает максимально допустимое отклонение от командного сигнала $\omega_{z_{ref}}$ равное ± 1 град/с

Рисунок 4.7 – Возникновение затухающего колебания для начального состояния $H = 2000$ м, $V = 275$ м/с (САО $= 1.323$)

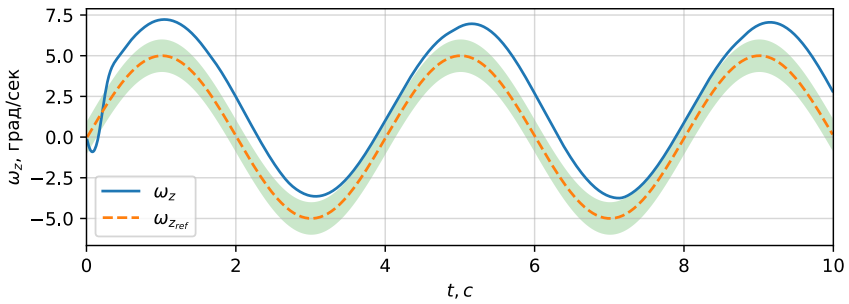


Рисунок 4.8 – Увеличение статической ошибки для состояния $H = 10000$ м., $V = 175$ м/с (САО=1.786)

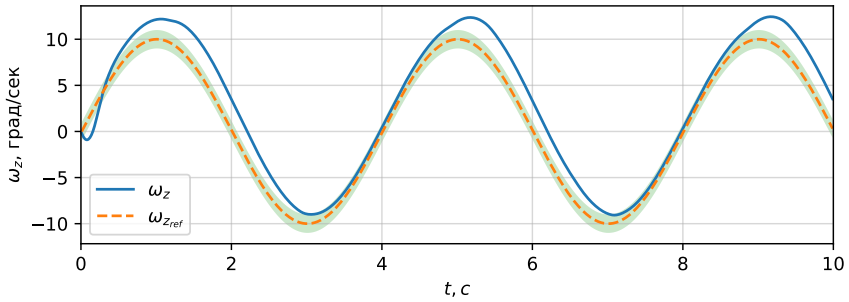


Рисунок 4.9 – Отслеживание задающего сигнала с увеличенной амплитудой для состояния $H = 10000$ м, $V = 175$ м/с (САО=1.758)

Данную особенность можно обойти, расширив диапазон начальных состояний, используемых для обучения, где в нашем случае верхняя граница была $V = 250$ м/с, $H = 4000$ м, после чего провести цикл обучения заново.

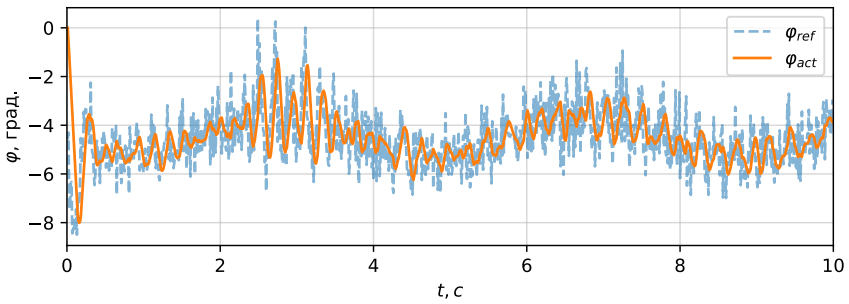
4.3 Особые случаи

В данном разделе будут рассмотрены случаи воздействия различных возмущающих факторов на состояния, наблюдаемые агентом s и на состояние модели x.

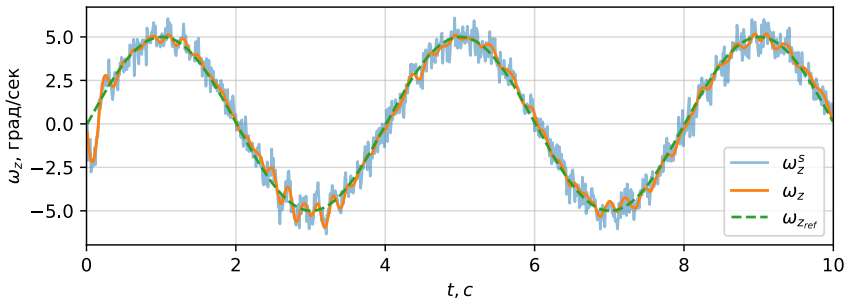
4.3.1 Влияние шума датчика угловых скоростей

Рассмотрим реакцию агента при наличии шумовой составляющей в сигнале ω_z. Данный шум будет вводиться исключительно в вектор состояния s, на основе которого агент формирует свои действия. Расчет состояния модели x будет производится без учета шума. Шум будет формироваться на основе нормального распределения $\mathcal{N}(0, 0.5)$ и добавляться к $\omega_z \in s$. Результат тестирования агента для этого случая представлен на рис. 4.10.

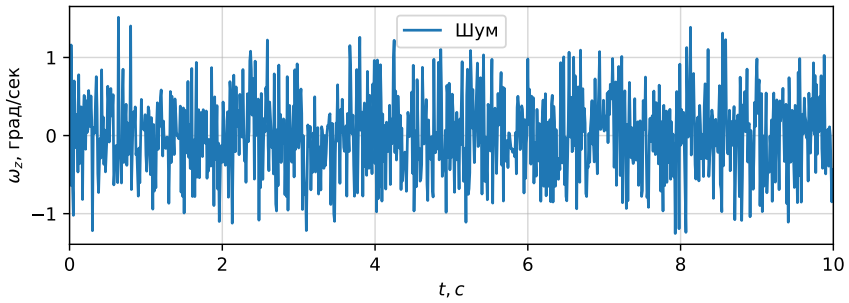
Современные датчики имеют уровень шумовой составляющей порядка $3\sigma = 0.1$град/с для измерения угловой скорости. Данный параметр выбран из характеристик датчика ИПДММ-2-1 производства ПАО МИЭА, где дисперсия шума ниже в 15 раз, чем в эксперименте на рис. 4.10. Из этих данных следует, что уровень шума $3\sigma = 0.1$ град/с ($\mathcal{N}(0, \frac{1}{30})$) влияет на динамику незначительно.



(a) Управляющие действия



(b) Изменение угловой скорости тангажа ω_z



(c) Величина добавленного шума ($\omega_z^s - \omega_z^x$)

Рисунок 4.10 – Отслеживание сигнала, использованного для тренировки с шумовой составляющей

Введённый шум непосредственно влияет на формирование управляющих действий контроллера, аналогичный шум, но меньший по величине, возникает в управляющих действиях по сравнению с экспериментом без шума (рис. 4.4). Аналогично шумовая составляющая влияет и на изменение состояния $\omega_z \in x$ модели. Сравнение качества отслеживания с экспериментом без шума представлено в таблице 4.5.

Таблица 4.5 – Сравнение показателей качества отслеживания

Случай	САО	СКО
Без шума	0.212	0.125
С шумом	0.271	0.167

Исходя из результатов представленных в таблице 4.5, введение шума на измерение угловой скорости тангажа практически не влияет на работу обученного агента, изменение средней абсолютной ошибки составляет величину порядка 0.06 град/с.

4.3.2 Влияние ветровых возмущений

Рассмотрим сценарий влияния турбулентных порывов на работу обученного агента. Была выбрана модель турбулентности фон Кармана. Данная модель рекомендована Европейским агентством безопасности полетов (EASA) для использования при определении соответствия характеристик самолета на заходе и при посадке методами моделировании при выполнении сертификации [1].

В данной модели определяются спектральные плотности компонент турбулентности. В нашем случае будет достаточно использовать вертикальную компоненту турбулентности применительно к $V \in x$:

$$\Phi_u(\Omega) = \frac{\sigma_u^2 L_u}{\pi} \frac{1}{\left(1 + (1.339 L_u \Omega)^2\right)^{\frac{5}{6}}},$$

где $\Phi_u(\Omega)$ – спектральная плотность турбулентности, [фут/с]^2;

$\sigma = 0.1061 V_{20}$ – среднеквадратическая интенсивность турбулентности, [фут/с];

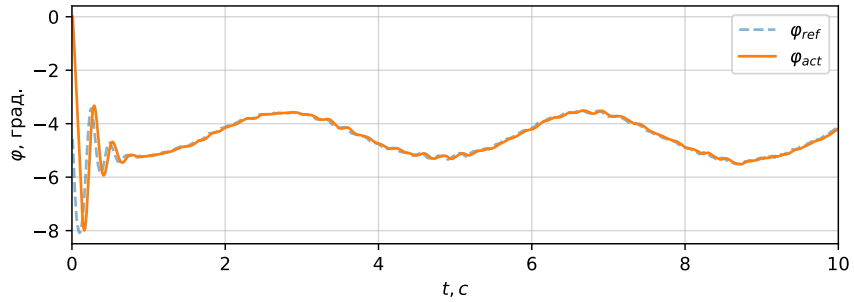
V_{20} – средняя скорость ветра, [узлы];

$L = 1000$ – масштабный коэффициент, фут;

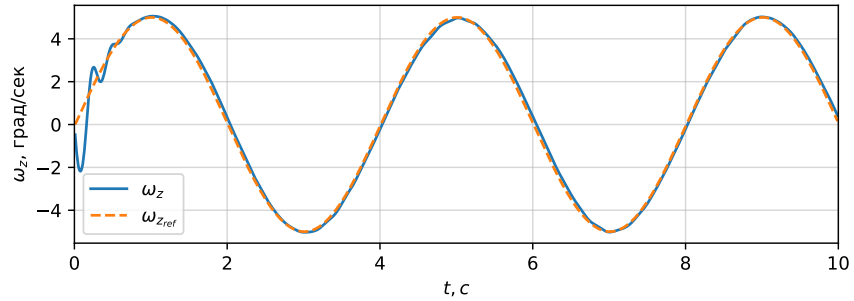
$\Omega = \frac{\omega}{V_T}$ – круговая частота, [рад/фут];

V_T – скорость ЛА, [фут/с].

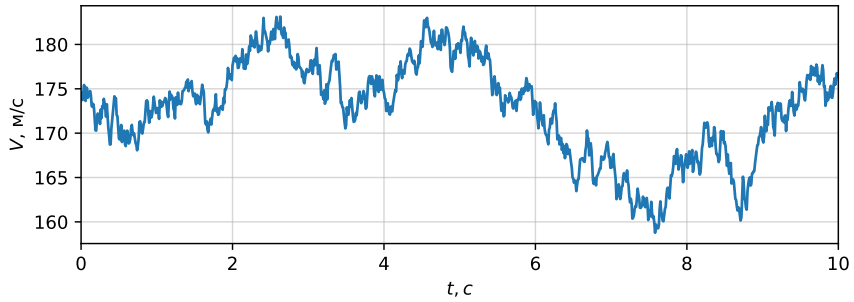
Результаты отработки при турбулентности со средней интенсивностью $V_{20} = 30$ узлов (10.3 м/с) представлены на рис. 4.11.



(a) Управляющие действия



(b) Изменение угловой скорости тангажа ω_z



(c) Изменение скорости V

Рисунок 4.11 – Отслеживание задающего синусоидального сигнала при турбулентных возмущениях

По результату отработки турбулентности со средней интенсивностью обученный агент справился не хуже, чем без возмущающих воздействий, при том что во время обучения агента не предполагалось изменение скорости V, однако в управляющих действиях характерна реакция на изменение величины V (см. рис. 4.11a).

4.3.3 Анализ результатов

Критерии качества управления для рассмотренных случаев сведены в таблицу 4.6.

Таблица 4.6 – Сравнение критериев для особых случаев

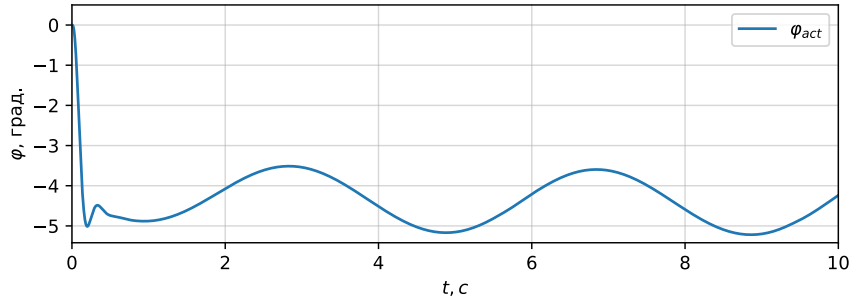
Случай	САО	СКО
Турбулентное воздействие	0.169	0.101
Шумовое воздействие	0.271	0.167
Без внешних воздействующих факторов	0.172	0.102

Как видно из данных результатов агент в состоянии справляться с изменению значения состояния V, ω_z объекта, вызванных внешними возмущениями. Существенной деградации в качестве управления в данных случаях не наблюдалось. Только введение шума $\mathcal{N}(0, 0.5)$ (с дисперсией 0.5 град/с) увеличили ошибку слежения на 0.1 град/с, что удовлетворительно, так как величина шумовой составляющей в разы больше чем на реальных датчиках угловых скоростей. Для турбулентного случая незначительное улучшения в качестве управления объясняется, тем что для иза вариации скорости V для некоторых значений V управляющие действия могут быть более близкими к оптимальным, чем для случая с постоянным $V = 175$ м/с (см. рис. 4.4).

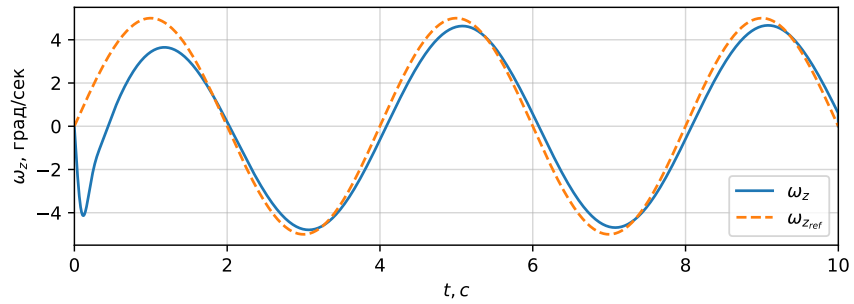
4.4 Сравнение сформированного PPO-контроллера с PI-контроллером

В данном пункте проведем сравнение PPO-контроллера с PI-контроллером по части критериев САО и СКО. Подбор коэффициентов PI-контроллера исходил из условия достижения удовлетворительных характеристик управляемости, а именно значение абсолютная статическая ошибка угловой скорости тангажа менее 0.5 град/с для отслеживания синусоидаль-

ного сигнала с частотой 0.25 Гц и амплитудой 5 град. Результат работы PI-контроллера для отслеживания синусоидального сигнала представлен на рис. 4.12.



(a) Управляющие действия



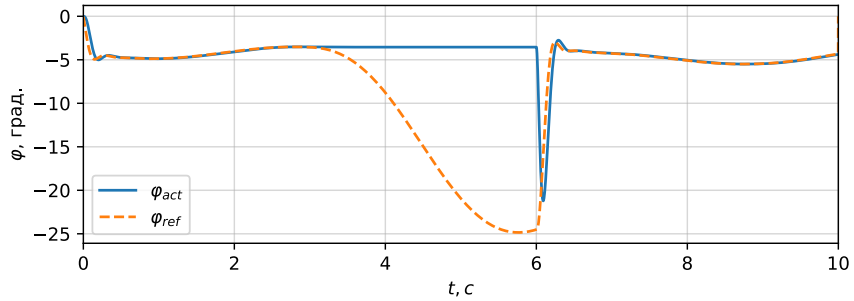
(b) Изменение угловой скорости тангажа ω_z

Рисунок 4.12 – Отслеживание синусоидального сигнала PI-контроллером, САО $= 0.216$

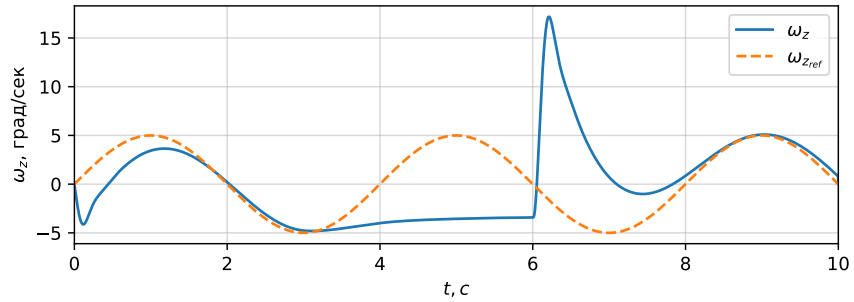
Для сравнения, обученный агент при отслеживании аналогичного сигнала (см. рис. 4.4) при одинаковых начальных условиях имеет значение САО = 0.172. Однако классический подход превосходит обученного агента в задаче отслеживания установившегося сигнала, так как имеющаяся интегральная составляющая PI-контроллера устраняет статическую ошибку, чего нельзя сказать об обученном агенте, который не обладает информацией о предыдущих состояниях. По этой причине PPO-контроллер способен только удерживать сигнал в определенной области. Однако в некоторых ситуациях данная особенность является положительной характеристикой.

Рассмотрим ситуацию кратковременного заклинения цельноповоротного управляемого стабилизатора, что вызывает невозможность изменения φ_{act}. В данной ситуации интегральный коэффициент PI-контроллера будет накапливать ошибку, что вызывает большие потребные отклонения

в момент устранения заклинения управляющей поверхности. Результаты данного эксперимента для PI-контроллера и обученного агента приведены на рис. 4.13 и 4.14.



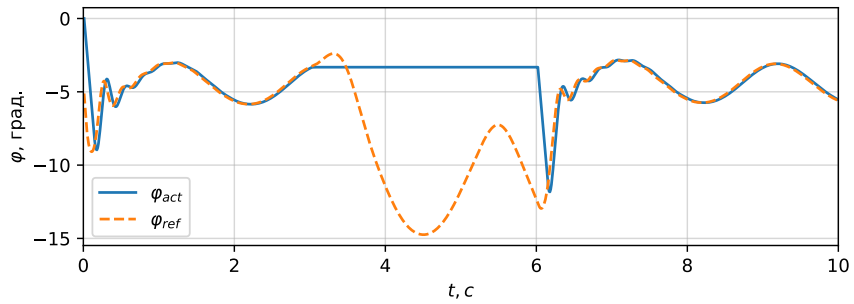
(a) Управляющие действия



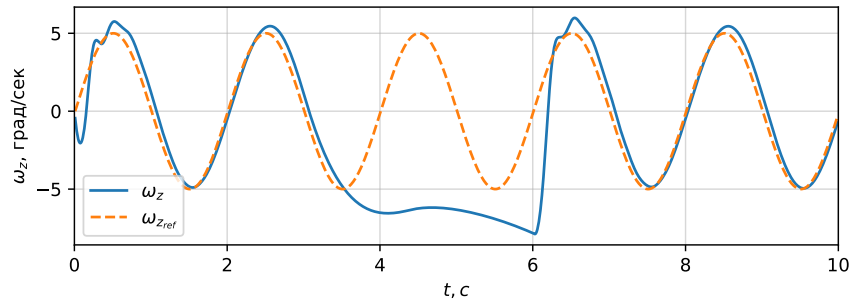
(b) Изменение угловой скорости тангажа ω_z

Рисунок 4.13 – Реакция PI-контроллера на заклинение стабилизатора, САО $= 1.035$

По результатам, приведенным на рис. 4.13, 4.14, видно, что для PI-контроллера при восстановлении управлением стабилизатором возникают угловые скорости порядка 15 град/с. В аналогичной ситуации синтезированный агент после возвращения управления отрабатывает возмущения с минимальным перерегулированием и реакция схожа с P-контроллером. В задаче отслеживания синусоидального сигнала значение средней абсолютной ошибки меньше на 0.04 град/с для обученного агента по сравнению с PI-контроллером.



(a) Управляющие действия



(b) Изменение угловой скорости тангажа ω_z

Рисунок 4.14 – Реакция обученного агента на заклинение стабилизатора, САО $= 0.644$

4.5 Анализ результатов

По результатам тестирования обученного агента можно сделать вывод, что формирование действий в диапазоне обучающей выборки (см. таблицу 3.2) близко к оптимальному (см. рис. 4.1). Обученный агент успешно справился с различными тестовыми сигналами, не входящими в обучающую выборку, чего нельзя сказать о состояниях, находящихся вне обучающего диапазона. Это связано со значительным изменением динамики объекта управления, с которыми обученный агент никогда не сталкивался. Что примечательно, потери устойчивости в данных случаях не наблюдалось, было лишь значительное ухудшение качества управлением объекта.

Также полученный агент устойчив к воздействию нормального шума с дисперсией 0.5 град/с при измерении угловой скорости тангажа ω_z и к изменению скорости V вследствие умеренных турбулентных возмущений. Приведенные выше ситуации не были рассмотрены в процессе обучения

агента и по результатам тестирования такие воздействия несущественно влияют на управление самолетом.

При сравнении с PI-контроллером, обученный агент не был хуже, а в ряде случаев значительно превосходит PI-контроллер в задаче отслеживания задающего сигнала.

ЗАКЛЮЧЕНИЕ

В данной работе был применен метод обучения с подкреплением под названием «непосредственная оптимизация стратегии» (Proximal Policy Optimization) для формирования контроллера угловой скорости устойчивого к измерительным шумам и атмосферным турбулентностям. Разобран один из подходов к обучению агента, который выступает в роли контроллера угловой скорости тангажа, также приведены особенности процесса формирования контроллеров на базе методов ОсП применительно к управлению ЛА. Аналогичный подход может использоваться для подобных задач, использующих другие алгоритмы ОсП в формировании контроллера.

Полученные результаты показывают возможность формирования адаптивного контроллера для решения данной задачи таким подходом и устойчивости сформированного контроллера в рассмотренных тестовых ситуациях. Также в работе приведено сравнение с традиционным подходом, для которого результат отработки в ряде тестовых сценариев оказался хуже, чем для сформированного контроллера.

Дальнейшие развитие работы может рассматривать:

- использование данного метода при формирования контроллера для систем с множественным входом-множественным выходом (MIMO);
- формирование подхода для устранения статической ошибки слежения. В процессе написания данной работы были попытки реализовать метод описанный в [11], но удовлетворительный результат не был достигнут;
- разработка тестовой среды в задаче управлением движения ЛА для оценки алгоритмов ОсП, обеспечивающего вычисление параметров движения во много раз быстрее чем течение реального времени;
- определение методологии к переносу разработанных алгоритмов ОсП на вычислители системы управления объектом;

На данном этапе такой подход показывает неплохие результаты по сравнению с традиционными методом управления. Однако на текущем этапе развития авиационной техники использование недетерминированных алгоритмов ограничено требованиям безопасности полетов, которые не позволяют пройти сертификацию критически важных систем [12].

Такие методы могут быть эффективны при синтезе контроллера, где высокий порядок управляющих действий над объектом управления и/или возникает сложность синтеза контроллера использующего традиционные методы.

СПИСОК ЛИТЕРАТУРЫ

[1] EASA. *Easy Access Rules for the Certification Specifications for All-Weather Operations (EAR for CS-AWO Issue 2)*. 2022. URL: https://www.easa.europa.eu/en/document-library/easy-access-rules/easy-access-rules-all-weather-operations-cs-awo.

[2] Hertel Lars, Baldi Pierre, Gillen Daniel L. “Reproducible Hyperparameter Optimization”. B: *Journal of Computational and Graphical Statistics* 31.1 (2022), с. 84—99. DOI: 10.1080/10618600.2021.1950004. URL: https://doi.org/10.1080/10618600.2021.1950004.

[3] Kingma Diederik P., Ba Jimmy. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980.

[4] Nguyen Luat T., Ogburn Marilyn E., Gilbert William P., Kibler Kemper S., Brown Phillip W., Deal Perry L. *Simulator study of stall/post-stall characteristics of a fighter airplane with relaxed longitudinal static stability*. NASA TP-1538. 1979, с. 223.

[5] Raffin Antonin, Kober Jens, Stulp Freek. *Smooth Exploration for Robotic Reinforcement Learning*. 2021. arXiv: 2005.05719.

[6] Schulman John, Levine Sergey, Moritz Philipp, Jordan Michael I., Abbeel Pieter. *Trust Region Policy Optimization*. 2017. arXiv: 1502.05477.

[7] Schulman John, Moritz Philipp, Levine Sergey, Jordan Michael, Abbeel Pieter. “High-Dimensional Continuous Control Using Generalized Advantage Estimation”. B: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2016.

[8] Schulman John, Wolski Filip, Dhariwal Prafulla, Radford Alec, Klimov Oleg. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347.

[9] Silver David, Huang Aja, Maddison Christopher J., Guez Arthur, Sifre Laurent, Driessche George van den, Schrittwieser Julian, Antonoglou Ioannis, Panneershelvam Veda, Lanctot Marc, Dieleman Sander, Grewe Dominik, Nham John, Kalchbrenner Nal, Sutskever Ilya, Lillicrap Timothy, Leach Madeleine, Kavukcuoglu Koray, Graepel Thore, Hassabis Demis. “Mastering the game of Go with deep neural networks and tree search”. В: *Nature* 529 (2016), с. 484—503. URL: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

[10] Sutton Richard S, Barto Andrew G. *Reinforcement learning: An introduction*. MIT press, 2018.

[11] Weber Daniel, Schenke Maximilian, Wallscheid Oliver. *Steady-State Error Compensation in Reference Tracking and Disturbance Rejection Problems for Reinforcement Learning-Based Control*. 2022. arXiv: 2201.13331 [eess.SY].

[12] Вересников Г.С., Скрябин А.В. “Методы искусственного интеллекта в системах автоматизированного управления беспилотными летательными аппаратами”. В: *Информационные технологии №3* 30 (2024), с. 115—123.

ПРИЛОЖЕНИЕ А

В листинге 1 приведен код для PPO-алгоритма, в листинге 2 приведен интерфейс запуска процесса обучения, в лиситнге 3 представлена реализация модели F-16 как среды для обучения агента. Также весь код на языке Python размещен в репозитории `https://github.com/lalapopa/F16-model/tree/distributed_controller/`.

Листинг 1: Реализация алгоритма PPO на языке Python

```python
import os
import time
import random
import torch
import numpy as np
from torch import nn, optim
from torch.distributions.normal import Normal


def layer_init(layer, std=np.sqrt(2), bias_const=0.0):
    torch.nn.init.orthogonal_(layer.weight, std)
    torch.nn.init.constant_(layer.bias, bias_const)
    return layer


class Agent(nn.Module):
    """Agent for PPO algo"""

    def __init__(self, env, config):
        super().__init__()
        self.config = config
        self._setup_seed()
        self.env = env
        self.action_shape = np.array(env.single_action_space.shape).prod()
        self.obs_shape = np.array(env.single_observation_space.shape).prod
            ()
        self.critic = nn.Sequential(
            layer_init(nn.Linear(self.obs_shape, 64)),
            nn.Tanh(),
            layer_init(nn.Linear(64, 64)),
            nn.Tanh(),
            layer_init(nn.Linear(64, 1), std=1.0),
        )
        self.actor_mean = nn.Sequential(
            layer_init(nn.Linear(self.obs_shape, 64)),
            nn.Tanh(),
            layer_init(nn.Linear(64, 64)),
            nn.Tanh(),
            layer_init(nn.Linear(64, self.action_shape), std=0.01),
        )  # aka Policy
        self.actor_logstd = nn.Parameter(torch.zeros(1, self.action_shape))

        self.gsde_mean = nn.Sequential(
            layer_init(nn.Linear(self.obs_shape, 64)),
            nn.Tanh(),
```

```
            layer_init(nn.Linear(64, 64)),
            nn.Tanh(),
            layer_init(nn.Linear(64, self.action_shape), std=0.01),
        )
        self.gsde_logstd = nn.Parameter(torch.zeros(1, self.action_shape))

    def get_value(self, x):
        return self.critic(x)

    def sample_theta_gsde(self, x, sample_idx=[]):
        action_mean = self.gsde_mean(x)
        gsde_std = torch.exp(self.gsde_logstd.expand_as(action_mean))
        theta_gsde_temp = Normal(0, gsde_std).rsample()
        if sample_idx:
            for idx_env in sample_idx:
                self.theta_gsde[idx_env] = theta_gsde_temp[idx_env]
        else:
            self.theta_gsde = theta_gsde_temp

    def get_action_and_value(self, x, action=None, learn_feature=False):
        action_mean = (
            self.actor_mean(x) if learn_feature else self.actor_mean(x).
                detach()
        )
        action_std = torch.exp(self.actor_logstd.expand_as(action_mean))
        probs = Normal(action_mean, action_std)

        if action is None:
            noise = self.theta_gsde * action_std
            action = probs.mean + noise
        log_prob = (
            probs.log_prob(action) if learn_feature else probs.log_prob(
                action.detach())
        )
        return (
            action,
            log_prob.sum(1),
            probs.entropy().sum(1),
            self.critic(x),
        )

    def save(self):
        try:
            os.makedirs(f"{self.config.save_dir}/{self.run_name}")
        except FileExistsError:
            pass
        torch.save(
            self.actor_logstd, f"{self.config.save_dir}/{self.run_name}/
                actor_logstd"
        )
        torch.save(
            self.actor_mean, f"{self.config.save_dir}/{self.run_name}/
                actor_mean"
        )
        torch.save(self.critic, f"{self.config.save_dir}/{self.run_name}/
            critic")
        torch.save(
            self.gsde_logstd, f"{self.config.save_dir}/{self.run_name}/
                gsde_logstd"
        )
        torch.save(self.gsde_mean, f"{self.config.save_dir}/{self.run_name
            }/gsde_mean")
        print(f"Saving model in: {self.config.save_dir}/{self.run_name}")
```

```
101
102     def load(self, path_name):
103         self.actor_logstd = torch.load(f"{path_name}/actor_logstd")
104         self.actor_mean = torch.load(f"{path_name}/actor_mean")
105         self.critic = torch.load(f"{path_name}/critic")
106         self.gsde_logstd = torch.load(f"{path_name}/gsde_logstd")
107         self.gsde_mean = torch.load(f"{path_name}/gsde_mean")
108
109     def _setup_seed(self):
110         random.seed(self.config.seed)
111         np.random.seed(self.config.seed)
112         torch.manual_seed(self.config.seed)
113         torch.backends.cudnn.deterministic = self.config.
                torch_deterministic
114
115     def _get_device(self):
116         return torch.device(
117             "cuda" if torch.cuda.is_available() and self.config.cuda else "
                    cpu"
118         )
119
120     def _init_episode_buffer(self):
121         obs = torch.zeros(
122             (self.config.num_steps, self.config.num_envs) + (self.obs_shape
                    ,),
123             dtype=torch.float32,
124         )
125         actions = torch.zeros(
126             (self.config.num_steps, self.config.num_envs) + (self.
                    action_shape,),
127             dtype=torch.float32,
128         )
129         logprobs = torch.zeros(
130             (self.config.num_steps, self.config.num_envs), dtype=torch.
                    float32
131         )
132         rewards = torch.zeros(
133             (self.config.num_steps, self.config.num_envs), dtype=torch.
                    float32
134         )
135         dones = torch.zeros(
136             (self.config.num_steps, self.config.num_envs), dtype=torch.
                    float32
137         )
138         values = torch.zeros(
139             (self.config.num_steps, self.config.num_envs), dtype=torch.
                    float32
140         )
141         return obs, actions, logprobs, rewards, dones, values
142
143     def train(self, run_name):
144         self.run_name = run_name
145         optimizer = optim.Adam(
146             self.parameters(),
147             lr=self.config.learning_rate,
148             amsgrad=True,
149         )
150         global_step = 0
151         num_updates = self.config.total_timesteps // self.config.batch_size
152         for update in range(1, num_updates + 1):
153             if self.config.anneal_lr:
154                 frac = 1.0 - (update - 1.0) / num_updates
155                 lrnow = frac * self.config.learning_rate
```

```
156                 optimizer.param_groups[0]["lr"] = lrnow
157
158             obs, actions, logprobs, rewards, dones, values = self.
                    _init_episode_buffer()
159             obs_init, _ = self.env.reset(seed=self.config.seed + update)
160             next_obs = torch.Tensor(obs_init)
161             next_done = torch.zeros(self.config.num_envs)
162             with torch.no_grad():
163                 self.sample_theta_gsde(next_obs)
164                 init_gsde_state = next_obs
165             for step in range(0, self.config.num_steps):
166                 global_step += 1 * self.config.num_envs
167                 obs[step] = next_obs
168                 dones[step] = next_done
169                 with torch.no_grad():
170                     action, logprob, _, value = self.get_action_and_value(
                            next_obs)
171                     values[step] = value.flatten()
172                 actions[step] = action
173                 logprobs[step] = logprob
174                 next_obs, reward, done, _, _ = self.env.step(action.cpu().
                        numpy())
175                 rewards[step] = torch.tensor(reward).view(-1)
176                 next_obs, next_done = torch.Tensor(next_obs).to(
177                     self.device
178                 ), torch.Tensor(done)
179                 if done.any():  # ONLY for perfomance monitor & resample
                        theta_gsde
180                     done_envs = []  # EXAMPLE: [0, 1, 2, 3] pick all
                            paralel env
181                     for idx_done_env in done_envs:
182                         init_gsde_state[idx_done_env] = next_obs[
                                idx_done_env]
183                     with torch.no_grad():
184                         self.sample_theta_gsde(init_gsde_state, sample_idx=
                                done_envs)
185             with torch.no_grad():
186                 next_value = self.get_value(next_obs).reshape(1, -1)
187                 if self.config.gae:
188                     advantages = torch.zeros_like(rewards)
189                     lastgaelam = 0
190                     for t in reversed(range(self.config.num_steps)):
191                         if t == self.config.num_steps - 1:
192                             nextnonterminal = 1.0 - next_done
193                             nextvalues = next_value
194                         else:
195                             nextnonterminal = 1.0 - dones[t + 1]
196                             nextvalues = values[t + 1]
197                         delta = (
198                             rewards[t]
199                             + self.config.gamma * nextvalues *
                                    nextnonterminal
200                             - values[t]
201                         )
202                         advantages[t] = lastgaelam = (
203                             delta
204                             + self.config.gamma
205                             * self.config.gae_lambda
206                             * nextnonterminal
207                             * lastgaelam
208                         )
209                     returns = advantages + values
210                 else:
```

```
                    returns = torch.zeros_like(rewards)
                    for t in reversed(range(self.config.num_steps)):
                        if t == self.config.num_steps - 1:
                            nextnonterminal = 1.0 - next_done
                            next_return = next_value
                        else:
                            nextnonterminal = 1.0 - dones[t + 1]
                            next_return = returns[t + 1]
                        returns[t] = (
                            rewards[t]
                            + self.config.gamma * nextnonterminal *
                                next_return
                        )
                    advantages = returns - values

                b_obs = obs.reshape((-1,) + (self.obs_shape,))
                b_logprobs = logprobs.reshape(-1)
                b_actions = actions.reshape((-1,) + (self.action_shape,))
                b_advantages = advantages.reshape(-1)
                b_returns = returns.reshape(-1)
                b_values = values.reshape(-1)

                b_inds = np.arange(self.config.batch_size)
                for _ in range(self.config.update_epochs):
                    np.random.shuffle(b_inds)
                    for start in range(
                        0, self.config.batch_size, self.config.minibatch_size
                    ):
                        end = start + self.config.minibatch_size
                        mb_inds = b_inds[start:end]
                        self.sample_theta_gsde(b_obs[mb_inds])
                        _, newlogprob, entropy, newvalue = self.
                            get_action_and_value(
                            b_obs[mb_inds], b_actions[mb_inds], learn_feature=
                                True
                        )
                        logratio = newlogprob - b_logprobs[mb_inds]
                        ratio = logratio.exp()  # exp(log(p) - log(q)) = p/q

                        mb_advantages = b_advantages[mb_inds]
                        if self.config.norm_adv:
                            mb_advantages = (mb_advantages - mb_advantages.mean
                                ()) / (
                                mb_advantages.std() + 1e-8
                            )
                        # Policy loss
                        pg_loss1 = (
                            -mb_advantages * ratio
                        )  # - A(s,a) * policy_new/ policy_old
                        pg_loss2 = -mb_advantages * torch.clamp(
                            ratio, 1 - self.config.clip_coef, 1 + self.config.
                                clip_coef
                        )  # part of Objective function
                        pg_loss = torch.max(pg_loss1, pg_loss2).mean()  # L^{
                            CLIP}(theta)
                        # Value loss
                        newvalue = newvalue.view(-1)
                        v_loss_unclipped = (newvalue - b_returns[mb_inds]) ** 2
                        v_clipped = b_values[mb_inds] + torch.clamp(
                            newvalue - b_values[mb_inds],
                            -self.config.clip_coef,
                            self.config.clip_coef,
                        )
```

```
                    v_loss_clipped = (v_clipped - b_returns[mb_inds]) ** 2
                    v_loss_max = torch.max(v_loss_unclipped, v_loss_clipped
                        )
                    v_loss = 0.5 * v_loss_max.mean()
                    entropy_loss = entropy.mean()
                    loss = (
                        pg_loss
                        - self.config.ent_coef * entropy_loss
                        + v_loss * self.config.vf_coef
                    )
                    optimizer.zero_grad()
                    loss.backward()
                    nn.utils.clip_grad_norm_(
                        self.parameters(), self.config.max_grad_norm
                    )
                    optimizer.step()
            if round(global_step, -3) % 10000 == 0:
                print(f"Saving model after {global_step}")
                self.save()
        print(f"Saving model after {global_step}")
        self.save()
```

Листинг 2: Интерфейс для запуска процесса обучения

```
import os
import time
import random
import gymnasium as gym

from utils import parse_args
from agent import Agent
from F16model.env import F16


def make_env(seed, env_config):
    def wrap_env():
        env = F16(env_config)
        env.action_space.seed(seed)
        env.observation_space.seed(seed)
        return env

    return wrap_env


if __name__ == "__main__":
    ENV_CONFIG = {
        "dt": 0.01,
        "tn": 10,
        "debug_state": False,
        "determenistic_ref": False,
        "scenario": "combo",
    }
    args = parse_args()
    envs = gym.vector.SyncVectorEnv(
        [make_env(args.seed + i, ENV_CONFIG) for i in range(args.num_envs)]
    )
    model = Agent(envs, args)

    run_name = f"F16__{args.seed}__{str(int(time.time()))}__{('%032x' %
        random.getrandbits(128))[:4]}"
    start_time = time.time()
    model.train(run_name)
```

Листинг 3: Реализация интерфейса среды

```python
import copy
import random
import numpy as np
import gymnasium as gym

from F16model.model import States, Control, F16model
from F16model.utils.calc import normalize_value, minmaxscaler
from F16model.env.task import ReferenceSignal
from F16model.model.engine import find_correct_thrust_position
from F16model.data import plane


class F16(gym.Env):
    """
    Gym like enviroment for custom F16model.
    """

    def __init__(self, config):
        self.config = config
        self.dt = float(self.config["dt"])
        self.tn = float(self.config["tn"])
        self.init_state = self._get_init_state()
        self.model = self._get_init_model()

        self.ref_signal = ReferenceSignal(
            self.dt,
            self.tn,
            determenistic=self.config["determenistic_ref"],
            scenario=self.config["scenario"],
        )
        self._destroy()
        self.action_space = gym.spaces.Box(-1, 1, (1,), dtype=np.float32)
        low = -np.ones(self._state_size())
        high = -low
        self.observation_space = gym.spaces.Box(low, high)

    def _destroy(self):
        self.clock = 0
        self.total_return = 0
        self.episode_length = 0
        self.done = False
        self.prev_state = self.init_state
        self.error_integral = 0

    def step(self, action):
        if isinstance(action, np.ndarray):
            action = F16.rescale_action(action)
            self.action = Control(action, 0)
        else:
            raise TypeError(f"Action has type '{type(action)}' should be np
                .array")
        state = self.model.step(self.action)
        self.prev_state = state
        self.clock += self.dt
        reward = self.check_state(state)
        reward += self.compute_reward(state)
        noized_state = self.add_noise(state)

        out_state = self.state_transform(noized_state)
        self.episode_length += 1
        self.total_return += reward
        info = {
```

```
            "episode_length": self.episode_length,
            "total_return": self.total_return,
            "clock": self.clock,
            "state": self.prev_state,
        }
        return out_state, reward, self.done, False, info

    def compute_reward(self, state):
        tracking_ref = (self.ref_signal.wz_ref[self.episode_length],)
        e = np.degrees(tracking_ref - state.wz)
        k = 1
        asymptotic_error = np.clip(
            1 - ((np.abs(e) / k) / (1 + (np.abs(e) / k))), a_min=0, a_max=1
        )
        linear_error = np.clip(1 - (1 / k) * e**2, a_min=0, a_max=1)
        reward = asymptotic_error + 0.04 * linear_error
        return reward.item()

    def state_transform(self, state):
        """
        Return states are :
        Oy
        wz
        V
        wz_ref
        0.5 * (wz_ref - wz)
        """
        state_short = {
            k: vars(state)[k] for k, _ in plane.state_bound.items() if k in
                vars(state)
        }  # take keys that defines in state_boundfrom 'States' class
        state_short = list(state_short.values())
        state_short = F16.normalize(state_short)  # Always output
            normalized states
        state_short.append(self.ref_signal.wz_ref[self.episode_length])
        state_short.append(
            0.5 * (self.ref_signal.wz_ref[self.episode_length] - state.wz)
        )
        return np.array(state_short)

    def check_state(self, state):
        reward = 0
        if state.Oy <= 300:
            reward += -1000
            self.done = True
        if state.Oy >= 30000:
            reward += -1000
            self.done = True
        if abs(state.wz) >= np.radians(50):
            reward += -((self.tn / self.dt) - self.episode_length)
            self.done = True
            print(f"Early done in step #{self.episode_length} | R: {reward}
                ")
        if self.episode_length == (self.tn / self.dt) - 1:
            self.done = True
        return reward

    def normalize(truncated_state):
        norm_values = []
        for i, values in enumerate(plane.state_bound.values()):
            norm_values.append(minmaxscaler(truncated_state[i], values[0],
                values[1]))
        return norm_values
```

```
121
122     def denormalize(state_norm):
123         norm_values = []
124         for i, values in enumerate(plane.state_bound.values()):
125             norm_values.append(
126                 minmaxscaler(
127                     state_norm[i], values[0], values[1], inverse_transform=
                            True
128                 )
129             )
130         norm_values = np.array(norm_values)
131         if i < len(state_norm) - 1:
132             additional_states = state_norm[i + 1 :]
133             norm_values = np.concatenate((norm_values, additional_states))
134         return norm_values
135
136     def add_noise(self, state):
137         if "noise" in self.config:
138             noise_state = copy.deepcopy(state)
139             if self.config["noise"]:
140                 noise_part_wz = np.random.normal(0, 0.5)
141                 noise_state.wz = noise_state.wz + np.radians(noise_part_wz)
142             return noise_state
143         return state
144
145     def reset(self, seed=None, options=None):
146         super().reset(seed=seed)
147         if seed:
148             random.seed(seed)
149             np.random.seed(seed)
150         self.init_state = self._get_init_state()
151         self.model = self._get_init_model()
152         self.ref_signal = ReferenceSignal(
153             self.dt,
154             self.tn,
155             determenistic=self.config["determenistic_ref"],
156             scenario=self.config["scenario"],
157         )
158         self._destroy()
159         out_state = self.state_transform(self.init_state)
160         return out_state, {}
161
162     def _get_init_state(self):
163         try:
164             if isinstance(self.config["init_state"], np.ndarray) and
                    isinstance(
165                     self.config["init_control"], np.ndarray
166                 ):
167                     init_state = self.config["init_state"]
168                     init_control = self.config["init_control"]
169                     state = States(
170                         Ox=init_state[0],
171                         Oy=init_state[1],
172                         wz=init_state[2],
173                         theta=init_state[3],
174                         V=init_state[4],
175                         alpha=init_state[5],
176                         stab=init_control[0],
177                         dstab=0,
178                         Pa=find_correct_thrust_position(init_control[1]),
179                     )
180             else:
181                 raise TypeError(
```

```
182                     f"INIT_STATE has type {type(self.config['init_state'])}
                            should be np.array"
183                 )
184             except KeyError:
185                 state = get_random_state()
186             return state
187
188         def _get_init_model(self):
189             model = F16model(self.init_state, dt=self.dt, config=self.config)
190             return model
191
192         def rescale_action(action):
193             """
194             Rescale action [stab]
195             """
196             action = np.clip(action[0], -1, 1)
197             stab_rescale = normalize_value(
198                 action,
199                 -plane.maxabsstab,
200                 plane.maxabsstab,
201                 inverse_transform=True,
202             )
203             return stab_rescale
204
205         def _state_size(self):
206             return self.state_transform(self.init_state).size
```