Самостоятельная работа. Модуль 1.

Швецов Леонид Сергеевич

«Разработка и отладка программного обеспечения для беспилотных летательных аппаратов (БПЛА). Методы тестирования и верификации».

Тема 8.3. Методы тестирования и верификации программного обеспечения БПЛА.

№ п/п	Наименование задания	Формат ответа
1	**Практическая работа по тестированию и верификации программного обеспечения для БПЛА:** В рамках данной работы необходимо применить методы тестирования и верификации, изученные на семинаре, для обнаружения и исправления ошибок в предоставленном коде управления БПЛА.	1. Изучите предоставленный код модулей uav_control.py и mission_planner.py. Опишите их предназначение и функционал. Проанализируйте код на наличие ошибок, используя методы статического анализа кода (например, pylint, flake8, mypy). **1.1. UAVControl**: Этот класс предоставляет методы для управления БПЛА через MAVLink. Он включает: • Подключение к БПЛА. • Взведение (**arm**) и разоружение (**disarm**) БПЛА. • Установка режима полета. • Отправка команды на взлет (**takeoff**) и перелет на заданные координаты (**goto**). • Получение телеметрических данных (шаг по более сложным показателям типа крена, тангажа и рысканье). • Ожидание подтверждения выполнения команд. **1.2. MissionPlanner**: Этот класс использует возможности **UAVControl** для планирования и выполнения заданной миссии по последовательности координат. Он включает: • Выполнение миссии, начиная с взвода, взлета, перелета по waypoints и возвращения в исходную точку (**RTL**). • Логирование информации и ошибок в процессе выполнения миссии. • Интеграция проверок допусков по достижении заданных точек. Я провёл статический анализ кода с использованием инструментов pylint, flake8 и mypy.

		Результаты анализа: • **pylint**: o Выявил несколько предупреждений о длине строк, несоответствии стиля именования и отсутствии Docstrings в некоторых местах, порядка импортов. o Рекомендуется использование lazy-форматирования в логах o Рекомендуется использовать более конкретные ошибки. • **flake8**: o Обнаружил несколько проблем с длинными строками o Предупреждения о неиспользуемых импортированных модулях. • **mypy**: o Выявил некоторые несоответствия типов, особенно в возвращаемых значениях методов. o Указал на отсутствие аннотаций типов в некоторых аргументах функций.
		2. Создайте тесты test_uav_control.py и test_mission_planner.py для проверки функционала существующих классов. Разработайте дополнительные модульные и интеграционные тесты для проверки функциональности, в частности, критически важных функций (например, goto) **2.1. Тестирование uav_control.py** Я разработал модуль test_uav_control.py для модульного тестирования класса UAVControl. **Основные шаги:** • Использовал библиотеку unittest для организации тестов. • Применил unittest.mock для создания mock-объектов и эмуляции поведения MAVLink-соединения. **Разработанные тесты:** 1. **test_connection**: проверка установления соединения с БПЛА. 2. **test_arm_disarm**: проверка методов arm и disarm. 3. **test_set_mode_valid**: проверка установки корректного

		режима полёта. 4. **test_set_mode_invalid**: проверка реакции на установку некорректного режима. 5. **test_takeoff_positive_altitude**: проверка взлёта на положительную высоту. 6. **test_takeoff_negative_altitude**: проверка реакции на отрицательную высоту. 7. **test_goto**: проверка команды полёта к заданной точке. 8. **test_get_telemetry**: проверка получения телеметрических данных. 9. **test_wait_command_ack**: проверка ожидания подтверждения команды. **2.2. Тестирование mission_planner.py** Я разработал модуль test_mission_planner.py для интеграционного тестирования класса MissionPlanner. **Основные шаги:** • Использовал unittest и unittest.mock для эмуляции поведения UAVControl. • Проверил выполнение миссии с последовательным достижением точек. **Разработанные тесты:** 1. **test_execute_mission_success**: проверка успешного выполнения миссии. 2. **test_execute_mission_failure**: проверка реакции на недостижение точки миссии.
		3. Используя методы тестирования из семинара, выявите скрытые ошибки в коде (например, намеренно внесённые ошибки в методах управления). **Выявление скрытых ошибок в коде** При разработке тестов и их запуске были обнаружены следующие скрытые ошибки: **3.1. Ошибка в методе goto** **Описание ошибки:** • В методе goto используется неправильный фрейм координат MAV_FRAME_GLOBAL_INT вместо

		MAV_FRAME_GLOBAL_RELATIVE_ALT. • Это приводит к тому, что высота интерпретируется относительно уровня моря, а не относительно точки взлёта, что может вызвать проблемы в управлении высотой БПЛА. **Как была обнаружена:** • Тест test_goto проверяет параметр frame в вызове mission_item_send. • Тест выявил несоответствие ожидаемого фрейма координат. **3.2. Использование assert в методе get_telemetry** **Описание ошибки:** • Использование assert для проверки диапазонов значений телеметрии не рекомендуется в продакшн-коде, так как при запуске с оптимизациями (-O) assert отключаются. • Это может привести к пропуску важных проверок. **Как была обнаружена:** • Инструмент pylint выдал предупреждение об использовании assert в продакшн-коде.
		4. Исправьте обнаруженные ошибки, документируя внесённые изменения и обоснования. - mavutil.mavlink.MAV_FRAME_GLOBAL_INT -> mavutil.mavlink.MAV_FRAME_GLOBAL_RELATIVE_ALT - removing assert из get_telemetry()
		5. Подготовьте отчёт о проведённой работе, включающий результаты тестирования, обнаруженные ошибки, внесённые исправления и выводы. **5.1. Результаты тестирования** • После внесения исправлений все тесты проходят успешно. • Статический анализ кода не выявляет критических замечаний. **5.2. Обнаруженные ошибки** • Неправильный фрейм координат в методе goto. • Использование assert в методе get_telemetry. • Несоответствия стиля кодирования и отсутствующие аннотации типов. **5.3. Внесённые исправления** • Исправлен фрейм координат в методе goto.

		• Заменены assert на явные проверки с выбросом исключений. • Улучшен стиль кода в соответствии с PEP 8. • Добавлены аннотации типов и Docstrings. **5.4. Выводы** • Тщательное тестирование и использование статического анализа позволяют выявить как очевидные, так и скрытые ошибки в коде. • Разработка тестов с учётом критически важных функций обеспечивает надёжность и безопасность программного обеспечения для БПЛА. • Соблюдение стандартов кодирования и документирования облегчает поддержку и развитие кода.
		6. Предоставьте обновлённый код модулей и тестов, соответствующий требованиям качества и безопасности. Код модулей и тестов обновлён в соответствии с выявленными ошибками и улучшениями. • **uav_control.py**: исправлены ошибки, улучшен стиль кода, добавлены аннотации типов. • **mission_planner.py**: улучшен стиль кода, добавлены аннотации типов. • **test_uav_control.py**: разработаны и актуализированы тесты для класса UAVControl. • **test_mission_planner.py**: разработаны и актуализированы тесты для класса MissionPlanner.
		7. Перечислите используемые методы и инструменты тестирования, обоснуйте их выбор и эффективность. **7.1. Методы тестирования** • **Модульное тестирование**: тестирование отдельных методов классов. • **Интеграционное тестирование**: проверка взаимодействия между классами UAVControl и MissionPlanner. • **Статический анализ кода**: выявление ошибок и несоответствий без запуска программы. • **Тестирование на основе граничных значений**: проверка

		реакции на предельные и некорректные значения входных данных. **7.2. Инструменты тестирования** • **unittest**: стандартный модуль для модульного тестирования в Python. • **unittest.mock**: позволяет создавать mock-объекты для эмуляции поведения внешних систем. • **pylint**: инструмент для статического анализа кода и проверки соответствия стилю. • **flake8**: инструмент для статического анализа кода, объединяющий pycodestyle, pyflakes и McCabe complexity checker. • **mypy**: инструмент для проверки статической типизации в Python. **7.3. Обоснование выбора инструментов** • **unittest** и **unittest.mock** позволяют эффективно разрабатывать модульные и интеграционные тесты, эмулируя поведение внешних зависимостей. • **pylint**, **flake8**, **mypy** предоставляют комплексный подход к статическому анализу кода, выявляя различные типы ошибок и несоответствий. **7.4. Эффективность методов** • Использованные методы и инструменты позволили обнаружить критические ошибки, которые могли привести к неправильной работе БПЛА. • Комбинация статического анализа и тестирования обеспечивает высокое качество и безопасность программного обеспечения.
		8. Загрузите свой код на GitHub или GitLab. Предоставьте публичный доступ для репозитория с вашим кодом. Код проекта размещён в публичном репозитории на GitHub.
		9. Ссылку на репозиторий приложите в файл домашнего задания. https://github.com/lshvetsov/innopolis_uav/tree/master/module_8/8_3