Промежуточная аттестация. Модуль 8.

ФИО

РЕШЕНИЕ КЕЙСОВ. Разработка и отладка программного обеспечения.

1. **Название кейса/задания/проекта:**

«Разработка и отладка программного обеспечения для управления беспилотным летательным аппаратом (БЛА)»..

1. **Подробное описание объекта (БАС, компоненты, механизмы, узлы и т.д.) и его характеристики в рамках работы над кейсом/заданием/проектом:**

Беспилотный летательный аппарат (БЛА):

Компоненты:

- Микроконтроллер (например, STM32, Arduino).

- Сенсоры (гироскопы, акселерометры, GPS, барометры).

- Актюаторы (электронные регуляторы скорости (ESC), моторы).

- Коммуникационные модули (Wi-Fi, Bluetoooth, радиосвязь).

Характеристики:

- Максимальная взлётная масса БЛА = 1,5 кг.

- Время полёта = до 30 минут.

- Дальность управления = до 1 км.

- Точность позиционирования = до 1 метра.

1. **Подробное описание задач, выполняемых в рамках кейса/задания/проекта:**

В рамках данного кейса/задания/проекта студенту предстоит разработать и выполнить отладку программного обеспечения для управления беспилотным летательным аппаратом (БЛА).

|  |  |  |
| --- | --- | --- |
| № п/п | Наименование задачи | Решение |
| 1 | Изучение и настройка среды разработки (IDE) | * 1. Опишите процесс установки и настройки интегрированной среды разработки (например, Visual Studio Code, Eclipse, PyCharm). |
| 1.2. Создайте новый проект (например, для гироскопа, акселерометра GPS или барометра) и выполните настройку необходимых параметров для реализации кода. Для следующего сценария:  - Максимальная взлётная масса БЛА = 1,5 кг.  - Время полёта = до 30 минут.  - Дальность управления = до 1 км.  - Точность позиционирования = до 1 метра. |
| 2 | Разработка программного обеспечения для управления БЛА.. | 2.1. Напишите код для основных функций управления БЛА (взлёт, посадка, стабилизация, навигация).  Для следующего сценария:  - максимальная взлётная масса БЛА = 1,5 кг.  - Время полёта = до 30 минут.  - Дальность управления = до 1 км.  - Точность позиционирования = до 1 метра. |
| 2.2. Выполните интеграцию сенсоров (например, гироскоп, акселерометр, GPS или барометр) и актюаторов (например, электронные регуляторы скорости (ESC), моторы) в программное обеспечение.  Для следующего сценария:  - максимальная взлётная масса БЛА = 1,5 кг.  - Время полёта = до 30 минут.  - Дальность управления = до 1 км.  - Точность позиционирования = до 1 метра. |
| 2.3. Выполните интеграцию коммуникационных модулей (например, Wi-Fi, Bluetooth или радиосвязь) в программное обеспечение (например, FreeRTOS для управления задачами в реальном времени).  Для следующего сценария:  - максимальная взлётная масса БЛА = 1,5 кг.  - Время полёта = до 30 минут.  - Дальность управления = до 1 км.  - Точность позиционирования = до 1 метра. |
| 3 | Использование инструментов отладки и тестирования. | 3.1. Выполните настройку и использование инструментов отладки (например, GDB, LLDB). |
| 3.2. Выполните пошаговую отладку кода для выявления и исправления ошибок. |
| 4 | Применение методов тестирования и верификации. | 4.1. Разработайте и выполните модульные тесты для проверки отдельных компонентов системы (например, PyTest для модульного тестирования). |
| 4.2. Выполните интеграционное тестирование для проверки взаимодействия между компонентами. |

1. **Перечень инструментов, материалов и оборудования, используемых для выполнения задач в рамках кейса/задания/проекта:**

Компьютер с установленной средой разработки (IDE).

Набор сенсоров (гироскопы, акселерометры, GPS, барометры).

Актюаторы (ESC, моторы).

Коммуникационные модули (Wi-Fi, Bluetooth, радиосвязь).

1. **Перечень программного обеспечения, языков программирования, их фреймворков и библиотек, используемых для выполнения задач в рамках кейса/задания/проекта:**

Среда разработки (IDE): Visual Studio Code, Eclipse, PyCharm.

Языки программирования: C/C++, Python.

Фреймворки и библиотеки: ArduPilot, HAL (Hardware Abstraction Layer), Matplotlib, Pandas, PyTest и др.

Протокол Mavlink для обмена телеметрической информацией между полетным контроллером и наземной станцией.

ПРИМЕР РЕШЕНИЯ.

|  |  |  |
| --- | --- | --- |
| № п/п | Наименование задачи | Решение |
| 1 | Изучение и настройка среды разработки (IDE) | * 1. Опишите процесс установки и настройки интегрированной среды разработки (например, Visual Studio Code, Eclipse, PyCharm).   Рассмотрим пример процесса установки и настройки интегрированной среды разработки Visual Studio Code.  Интегрированная среда разработки Visual Studio Code поддерживает язык программирования Python, имеет расширенный поиск и статический анализ кода, поддерживает Git, различные фреймворки, библиотеки, предоставляет возможность выполнять модульное, интеграционное и системное тестирование и отладку, поддерживает расширенную подсветку синтаксиса и имеет функцию автозавершения.  Visual Studio Code позволяет разработчикам тестировать и разрабатывать ПО для БЛА на удалённых серверах.  Для установки Visual Studio Code потребовалось скачать установочный файл на официальном сайте URL: https://code.visualstudio.com для нужной версии операционной системы – Windows. Далее была выполнена настройка интегрированной среды разработки и установлены дополнительные расширения Python, Git, Pip Manager, Jenkins, Jupiter, Pip installer, Sourcery и др.  Ниже показана скринкопия установленной интегрированной среды разработки. |
| 1.2. Создайте новый проект (например, для гироскопа, акселерометра GPS или барометра) и выполните настройку необходимых параметров для реализации кода. Для следующего сценария:  - Максимальная взлётная масса БЛА = 1,5 кг.  - Время полёта = до 30 минут.  - Дальность управления = до 1 км.  - Точность позиционирования = до 1 метра.  Рассмотрим создание нового проекта для акселерометра в Visual Studio Code. В качестве исходных данных рассматриваются параметры, указанные в сценарии для промежуточной аттестации.  1) В настроенном репозитории создадим папку проекта ACCELEROMETER4.  2) Откроем эту папку в интегрированной среде разработки.  3) В интегрированной среде разработки создадим файл drone\_accelerometer.py, в котором напишем код для акселерометра.  **Ниже представлен код для акселерометра:**  import numpy as np  import unittest  # Константы для параметров БЛА  MAX\_PAYLOAD = 1500  # максимальная взлетная масса в граммах  MAX\_FLIGHT\_TIME = 30 \* 60  # максимальное время полета в секундах  MAX\_CONTROL\_DISTANCE = 1000  # максимальная дальность управления в метрах  POSITIONING\_ACCURACY = 1  # точность позиционирования в метрах  def data\_acs(accel\_data, weight, flight\_time, control\_distance):      """      Обработка данных с акселерометра с учетом параметров БЛА.      Args:          accel\_data: Список значений ускорения по осям X, Y, Z (g).          weight: Масса БЛА в граммах.          flight\_time: Время полета в секундах.          control\_distance: Дальность управления в метрах.      Returns:          Словарь с обработанными данными:          - acceleration: Список значений ускорения (g).          - inclination: Список углов наклона по осям X, Y (град).      """      if weight > MAX\_PAYLOAD:          raise ValueError("Превышена максимальная взлетная масса БЛА.")      if flight\_time > MAX\_FLIGHT\_TIME:          raise ValueError("Превышено максимальное время полета.")      if control\_distance > MAX\_CONTROL\_DISTANCE:          raise ValueError("Превышена максимальная дальность управления.")      acceleration\_x = np.mean(accel\_data[0])      acceleration\_y = np.mean(accel\_data[1])      acceleration\_z = np.mean(accel\_data[2])      inclination\_x = np.arctan2(acceleration\_x, np.sqrt(np.square(acceleration\_y) + np.square(acceleration\_z))) \* 180 / np.pi      inclination\_y = np.arctan2(acceleration\_y, np.sqrt(np.square(acceleration\_x) + np.square(acceleration\_z))) \* 180 / np.pi      return {          "acceleration": [acceleration\_x, acceleration\_y, acceleration\_z],          "inclination": [inclination\_x, inclination\_y]      }  if \_\_name\_\_ == "\_\_main\_\_":      accel\_data = [[2, 2, 2], [5, 5, 5], [8, 8, 8]]      weight = 1200  # масса БЛА в граммах      flight\_time = 1600  # время полета в секундах      control\_distance = 800  # дальность управления в метрах      try:          data\_acs\_obj = data\_acs(accel\_data, weight, flight\_time, control\_distance)          print("Обработка данных:")          print(f"Ускорение: {data\_acs\_obj['acceleration']}")          print(f"Углы наклона: {data\_acs\_obj['inclination']}")      except ValueError as e:          print(e)  class TestDataAcs(unittest.TestCase):      def test\_data\_processing(self):          accel\_data = [[2, 2, 2], [5, 5, 5], [8, 8, 8]]          weight = 1200  # допустимая масса БЛА          flight\_time = 1600  # допустимое время полета          control\_distance = 800  # допустимая дальность управления          result = data\_acs(accel\_data, weight, flight\_time, control\_distance)          self.assertAlmostEqual(result['acceleration'][0], 2.0)          self.assertAlmostEqual(result['acceleration'][1], 5.0)          self.assertAlmostEqual(result['acceleration'][2], 8.0)          self.assertAlmostEqual(result['inclination'][0], 11.96946312460731, places=2)          self.assertAlmostEqual(result['inclination'][1], 31.23006977557103, places=2)  if \_\_name\_\_ == "\_\_main\_\_":      unittest.main(verbosity=2)  Теперь необходимо выполнить код и его протестировать:  Команда для выполнения кода: python drone\_accelerometer.py    Далее запустим тесты с помощью команды в терминале: python –m unittest drone\_accelerometer.py |
| 2 | Разработка программного обеспечения для управления БЛА.. | 2.1. Напишите код для основных функций управления БЛА (взлёт, посадка, стабилизация, навигация).  Для следующего сценария:  - максимальная взлётная масса БЛА = 1,5 кг.  - Время полёта = до 30 минут.  - Дальность управления = до 1 км.  - Точность позиционирования = до 1 метра.  Рассмотрим создание нового проекта для функции управления БЛА – взлёт, в Visual Studio Code. В качестве исходных данных рассматриваются параметры, указанные в сценарии для промежуточной аттестации.  1) В папке проекта ACCELEROMETER4 создадим файл drone\_takeoff.py, в котором напишем код для параметров взлёта.  **Ниже представлен код для параметров взлёта:**  import numpy as np  import unittest  # Константы для параметров БЛА  MAX\_PAYLOAD = 1500  # максимальная взлетная масса в граммах  MAX\_FLIGHT\_TIME = 30 \* 60  # максимальное время полета в секундах  MAX\_CONTROL\_DISTANCE = 1000  # максимальная дальность управления в метрах  def data\_acs(accel\_data, weight, flight\_time, control\_distance, is\_taking\_off):      """Обработка данных с акселерометра с учетом параметров БЛА."""        if weight > MAX\_PAYLOAD:          raise ValueError("Превышена максимальная взлетная масса БЛА.")        if flight\_time > MAX\_FLIGHT\_TIME:          raise ValueError("Превышено максимальное время полета.")        if control\_distance > MAX\_CONTROL\_DISTANCE:          raise ValueError("Превышена максимальная дальность управления.")        # Расчет средних значений      acceleration\_x = np.mean(accel\_data[0])      acceleration\_y = np.mean(accel\_data[1])      acceleration\_z = np.mean(accel\_data[2])        if is\_taking\_off:          # Ожидается увеличение ускорения по Z при взлете.          acceleration\_z += 9.81  # Добавляется ускорение, учитывая гравитацию      # Теперь, если угол наклона все еще растет из-за взлета, можно вычислить наклон      inclination\_x = np.arctan2(acceleration\_y, np.sqrt(np.square(acceleration\_x) + np.square(acceleration\_z))) \* 180 / np.pi      inclination\_y = np.arctan2(acceleration\_x, np.sqrt(np.square(acceleration\_y) + np.square(acceleration\_z))) \* 180 / np.pi      # Углы наклона должны быть равны 0, если БЛА находится в равновесии      if is\_taking\_off and acceleration\_z > 9.81:          inclination\_x = 0.0          inclination\_y = 0.0      return {          "acceleration": [acceleration\_x, acceleration\_y, acceleration\_z],          "inclination": [inclination\_x, inclination\_y]      }  def drone\_takeoff(weight):      """Функция для управления взлетом БЛА."""      if weight <= MAX\_PAYLOAD:          return "БЛА готов к взлету."      else:          raise ValueError("Не удается взлететь. Превышена максимальная взлетная масса.")  # Выполнение кода  if \_\_name\_\_ == "\_\_main\_\_":      accel\_data = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]  # Данные акселерометра      weight = 1200  # масса БЛА в граммах      flight\_time = 1600  # время полета в секундах      control\_distance = 800  # дальность управления в метрах      is\_taking\_off = True  # Обозначение взлета      try:          data\_acs\_obj = data\_acs(accel\_data, weight, flight\_time, control\_distance, is\_taking\_off)          print("Обработка данных:")          print(f"Ускорение: {data\_acs\_obj['acceleration']}")          print(f"Углы наклона: {data\_acs\_obj['inclination']}")          print(drone\_takeoff(weight))  # Вызов функции взлета      except ValueError as e:          print(e)  class TestDroneControl(unittest.TestCase):      def test\_takeoff\_success(self):          weight = 1200  # допустимая масса БЛА          result = drone\_takeoff(weight)          self.assertEqual(result, "БЛА готов к взлету.")      def test\_takeoff\_failure(self):          weight = 1600  # превышение массы БЛА          with self.assertRaises(ValueError):              drone\_takeoff(weight)  class TestDataAcs(unittest.TestCase):      def test\_data\_processing\_takeoff(self):          # Данные акселерометра предполагаются как равные нулю          accel\_data = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]          weight = 1200  # допустимая масса БЛА          flight\_time = 1600  # допустимое время полета          control\_distance = 800  # допустимая дальность управления          is\_taking\_off = True  # Симуляция взлета          result = data\_acs(accel\_data, weight, flight\_time, control\_distance, is\_taking\_off)          # Ожидается, что ускорение по Z будет равно 9.81, отразить влияние гравитации.          self.assertAlmostEqual(result['acceleration'][0], 0.0)  # Ускорение по X          self.assertAlmostEqual(result['acceleration'][1], 0.0)  # Ускорение по Y          self.assertAlmostEqual(result['acceleration'][2], 9.81)  # Ускорение по Z          # Проверка углов наклона          self.assertAlmostEqual(result['inclination'][0], 0.0, places=2)  # Угол наклона по X должен быть 0          self.assertAlmostEqual(result['inclination'][1], 0.0, places=2)  # Угол наклона по Y должен быть 0  if \_\_name\_\_ == "\_\_main\_\_":      unittest.main(verbosity=2)  Теперь необходимо выполнить код и его протестировать:  Команда для выполнения кода: python drone\_takeoff.py    Далее запустим тесты с помощью команды в терминале: python –m unittest drone\_takeoff.py |
| 2.2. Выполните интеграцию сенсоров (например, гироскоп, акселерометр, GPS или барометр) и актюаторов (например, электронные регуляторы скорости (ESC), моторы) в программное обеспечение.  Для следующего сценария:  - максимальная взлётная масса БЛА = 1,5 кг.  - Время полёта = до 30 минут.  - Дальность управления = до 1 км.  - Точность позиционирования = до 1 метра.  Рассмотрим интеграцию нового сенсора, например гироскопа, в рассматриваемый сценарий взлёта БЛА, в Visual Studio Code. В качестве исходных данных рассматриваются параметры, указанные в сценарии для промежуточной аттестации.  1) В папке проекта ACCELEROMETER4 создадим файл drone\_gyro\_takeoff.py, в котором напишем код для параметров взлёта.  **Ниже представлен код для параметров взлёта:**  import numpy as np  import random  # Импорт для генерации случайных данных гироскопа  import unittest  # Константы для параметров БЛА  MAX\_PAYLOAD = 1500  # максимальная взлетная масса в граммах  MAX\_FLIGHT\_TIME = 30 \* 60  # максимальное время полета в секундах  MAX\_CONTROL\_DISTANCE = 1000  # максимальная дальность управления в метрах  def data\_acs(accel\_data, weight, flight\_time, control\_distance, is\_taking\_off):      """Обработка данных с акселерометра с учетом параметров БЛА."""        if weight > MAX\_PAYLOAD:          raise ValueError("Превышена максимальная взлетная масса БЛА.")        if flight\_time > MAX\_FLIGHT\_TIME:          raise ValueError("Превышено максимальное время полета.")        if control\_distance > MAX\_CONTROL\_DISTANCE:          raise ValueError("Превышена максимальная дальность управления.")        # Рассчитываем средние значения      acceleration\_x = np.mean(accel\_data[0])      acceleration\_y = np.mean(accel\_data[1])      acceleration\_z = np.mean(accel\_data[2])        if is\_taking\_off:          # При взлете мы ожидаем увеличение ускорения по Z.          acceleration\_z += 9.81  # Добавляем ускорение, учитывая гравитацию      # Мы можем вычислить наклон      inclination\_x = np.arctan2(acceleration\_y, np.sqrt(np.square(acceleration\_x) + np.square(acceleration\_z))) \* 180 / np.pi      inclination\_y = np.arctan2(acceleration\_x, np.sqrt(np.square(acceleration\_y) + np.square(acceleration\_z))) \* 180 / np.pi      return {          "acceleration": [acceleration\_x, acceleration\_y, acceleration\_z],          "inclination": [inclination\_x, inclination\_y]      }  def drone\_takeoff(weight):      """Функция для управления взлетом БЛА."""      if weight <= MAX\_PAYLOAD:          return "БЛА готов к взлету."      else:          raise ValueError("Не удается взлететь. Превышена максимальная взлетная масса.")  def read\_gyro():      """Чтение данных гироскопа (симуляция)."""      # Симуляция случайных данных гироскопа      x\_gyro = random.uniform(-100, 100)  # Угловая скорость по X      y\_gyro = random.uniform(-100, 100)  # Угловая скорость по Y      z\_gyro = random.uniform(-100, 100)  # Угловая скорость по Z      return x\_gyro, y\_gyro, z\_gyro  if \_\_name\_\_ == "\_\_main\_\_":      accel\_data = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]  # Данные акселерометра      weight = 1200  # масса БЛА в граммах      flight\_time = 1600  # время полета в секундах      control\_distance = 800  # дальность управления в метрах      is\_taking\_off = True  # Флаг для обозначения взлета      try:          data\_acs\_obj = data\_acs(accel\_data, weight, flight\_time, control\_distance, is\_taking\_off)          print("Обработка данных:")          print(f"Ускорение: {data\_acs\_obj['acceleration']}")          print(f"Углы наклона: {data\_acs\_obj['inclination']}")          print(drone\_takeoff(weight))  # Вызов функции взлета          # Чтение данных гироскопа          gyro\_data = read\_gyro()          print(f"Данные гироскопа (X, Y, Z): {gyro\_data}")      except ValueError as e:          print(e)  class TestDroneControl(unittest.TestCase):      def test\_takeoff\_success(self):          weight = 1200  # допустимая масса БЛА          result = drone\_takeoff(weight)          self.assertEqual(result, "БЛА готов к взлету.")      def test\_takeoff\_failure(self):          weight = 1600  # превышение массы БЛА          with self.assertRaises(ValueError):              drone\_takeoff(weight)  class TestDataAcs(unittest.TestCase):      def test\_data\_processing\_takeoff(self):          # Данные акселерометра предполагаются как равные нулю          accel\_data = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]  # Настроенный на актуальные данные          weight = 1200  # допустимая масса БЛА          flight\_time = 1600  # допустимое время полета          control\_distance = 800  # допустимая дальность управления          is\_taking\_off = True  # Симуляция взлета          result = data\_acs(accel\_data, weight, flight\_time, control\_distance, is\_taking\_off)          # Ожидаем, что ускорение по Z будет равно 9.81, отразить влияние гравитации.          self.assertAlmostEqual(result['acceleration'][0], 0.0)  # Ускорение по X          self.assertAlmostEqual(result['acceleration'][1], 0.0)  # Ускорение по Y          self.assertAlmostEqual(result['acceleration'][2], 9.81)  # Ускорение по Z          # Проверка углов наклона          self.assertAlmostEqual(result['inclination'][0], 0.0, places=2)  # Угол наклона по X должен быть 0          self.assertAlmostEqual(result['inclination'][1], 0.0, places=2)  # Угол наклона по Y должен быть 0  class TestGyro(unittest.TestCase):      def test\_read\_gyro(self):          """Тестируем чтение данных гироскопа."""          gyro\_data = read\_gyro()          # Проверяем, что данные гироскопа находятся в допустимых пределах          for value in gyro\_data:              self.assertTrue(-100 <= value <= 100, "Данные гироскопа вне допустимого диапазона.")  if \_\_name\_\_ == "\_\_main\_\_":      unittest.main(verbosity=2)  Теперь необходимо выполнить код и его протестировать:  Команда для выполнения кода: python drone\_gyro\_takeoff.py    Далее запустим тесты с помощью команды в терминале: python –m unittest drone\_gyro\_takeoff.py |
| 2.3. Выполните интеграцию коммуникационных модулей (например, Wi-Fi, Bluetooth или радиосвязь) в программное обеспечение (например, FreeRTOS для управления задачами в реальном времени).  Для следующего сценария:  - максимальная взлётная масса БЛА = 1,5 кг.  - Время полёта = до 30 минут.  - Дальность управления = до 1 км.  - Точность позиционирования = до 1 метра.  Рассмотрим интеграцию нового коммуникационного модуля, например Wi-Fi, в рассматриваемый сценарий взлёта БЛА, в Visual Studio Code. В качестве исходных данных рассматриваются параметры, указанные в сценарии для промежуточной аттестации.  1) В папке проекта ACCELEROMETER4 создадим файл drone\_wi-fi\_takeoff.py, в котором напишем код для параметров взлёта.  **Ниже представлен код c интеграцией нового коммуникационного модуля:**  import numpy as np  import random  # Импорт для генерации случайных данных гироскопа  import unittest  import socket  # Для проверки подключения к Wi-Fi  # Константы для параметров БЛА  MAX\_PAYLOAD = 1500  # максимальная взлетная масса в граммах  MAX\_FLIGHT\_TIME = 30 \* 60  # максимальное время полета в секундах  MAX\_CONTROL\_DISTANCE = 1000  # максимальная дальность управления в метрах  def data\_acs(accel\_data, weight, flight\_time, control\_distance, is\_taking\_off):      """Обработка данных с акселерометра с учетом параметров БЛА."""        if weight > MAX\_PAYLOAD:          raise ValueError("Превышена максимальная взлетная масса БЛА.")        if flight\_time > MAX\_FLIGHT\_TIME:          raise ValueError("Превышено максимальное время полета.")        if control\_distance > MAX\_CONTROL\_DISTANCE:          raise ValueError("Превышена максимальная дальность управления.")        # Рассчитываем средние значения      acceleration\_x = np.mean(accel\_data[0])      acceleration\_y = np.mean(accel\_data[1])      acceleration\_z = np.mean(accel\_data[2])        if is\_taking\_off:          # При взлете мы ожидаем увеличение ускорения по Z.          acceleration\_z += 9.81  # Добавляем ускорение, учитывая гравитацию      # Теперь, если угол наклона все еще растет из-за взлета, мы можем вычислить наклон      inclination\_x = np.arctan2(acceleration\_y, np.sqrt(np.square(acceleration\_x) + np.square(acceleration\_z))) \* 180 / np.pi      inclination\_y = np.arctan2(acceleration\_x, np.sqrt(np.square(acceleration\_y) + np.square(acceleration\_z))) \* 180 / np.pi      return {          "acceleration": [acceleration\_x, acceleration\_y, acceleration\_z],          "inclination": [inclination\_x, inclination\_y]      }  def drone\_takeoff(weight):      """Функция для управления взлетом БЛА."""      if weight <= MAX\_PAYLOAD:          return "БЛА готов к взлету."      else:          raise ValueError("Не удается взлететь. Превышена максимальная взлетная масса.")  def read\_gyro():      """Чтение данных гироскопа (симуляция)."""      # Симуляция случайных данных гироскопа      x\_gyro = random.uniform(-100, 100)  # Угловая скорость по X      y\_gyro = random.uniform(-100, 100)  # Угловая скорость по Y      z\_gyro = random.uniform(-100, 100)  # Угловая скорость по Z      return x\_gyro, y\_gyro, z\_gyro  def check\_wifi\_connection(host='8.8.8.8', port=53, timeout=3):      """Проверка подключения к Wi-Fi, с помощью попытки подключения к интернету."""      try:          socket.create\_connection((host, port), timeout)          return True      except OSError:          return False  # Начиная с основного выполнения кода  if \_\_name\_\_ == "\_\_main\_\_":      accel\_data = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]  # Данные акселерометра      weight = 1200  # масса БЛА в граммах      flight\_time = 1600  # время полета в секундах      control\_distance = 800  # дальность управления в метрах      is\_taking\_off = True  # Флаг для обозначения взлета      try:          data\_acs\_obj = data\_acs(accel\_data, weight, flight\_time, control\_distance, is\_taking\_off)          print("Обработка данных:")          print(f"Ускорение: {data\_acs\_obj['acceleration']}")          print(f"Углы наклона: {data\_acs\_obj['inclination']}")          print(drone\_takeoff(weight))  # Вызов функции взлета          # Чтение данных гироскопа          gyro\_data = read\_gyro()          print(f"Данные гироскопа (X, Y, Z): {gyro\_data}")          # Проверка Wi-Fi соединения          if check\_wifi\_connection():              print("Wi-Fi соединение активно.")          else:              print("Нет подключения к Wi-Fi.")      except ValueError as e:          print(e)  class TestDroneControl(unittest.TestCase):      def test\_takeoff\_success(self):          weight = 1200  # допустимая масса БЛА          result = drone\_takeoff(weight)          self.assertEqual(result, "БЛА готов к взлету.")      def test\_takeoff\_failure(self):          weight = 1600  # превышение массы БЛА          with self.assertRaises(ValueError):              drone\_takeoff(weight)  class TestDataAcs(unittest.TestCase):      def test\_data\_processing\_takeoff(self):          # Данные акселерометра предполагаются как равные нулю          accel\_data = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]  # Настроенный на актуальные данные          weight = 1200  # допустимая масса БЛА          flight\_time = 1600  # допустимое время полета          control\_distance = 800  # допустимая дальность управления          is\_taking\_off = True  # Симуляция взлета          result = data\_acs(accel\_data, weight, flight\_time, control\_distance, is\_taking\_off)          # Ожидаем, что ускорение по Z будет равно 9.81, отразить влияние гравитации.          self.assertAlmostEqual(result['acceleration'][0], 0.0)  # Ускорение по X          self.assertAlmostEqual(result['acceleration'][1], 0.0)  # Ускорение по Y          self.assertAlmostEqual(result['acceleration'][2], 9.81)  # Ускорение по Z          # Проверка углов наклона          self.assertAlmostEqual(result['inclination'][0], 0.0, places=2)  # Угол наклона по X должен быть 0          self.assertAlmostEqual(result['inclination'][1], 0.0, places=2)  # Угол наклона по Y должен быть 0  class TestWiFi(unittest.TestCase):      def test\_wifi\_connection(self):          """Тестирование Wi-Fi соединения."""          # Здесь мы просто проверяем функцию, в реальной системе это будет зависеть от наличия сети          self.assertTrue(check\_wifi\_connection() or not check\_wifi\_connection(), "Проблема с подключением к Wi-Fi или отсутствием сети.")  if \_\_name\_\_ == "\_\_main\_\_":      unittest.main(verbosity=2)  Теперь необходимо выполнить код и его протестировать:  Команда для выполнения кода: python drone\_wi-fi\_takeoff.py    Далее запустим тесты с помощью команды в терминале: python –m unittest drone\_wi-fi\_takeoff.py |
| 3 | Использование инструментов отладки и тестирования. | 3.1. Выполните настройку и использование инструментов отладки (например, GDB, LLDB).  1) Выполним настройку инструмента отладки LLDB.  2) Для Windows установим с официального сайта LLDB.  URL: https://llvm.org/releases/download.html.  3) Выполним установку расширения CodeLLDB в Visual Studio Code.  4) Создадим проект на Python для отладки и далее выполним пошаговую отладку кода для выявления и исправления возможных ошибок. |
| 3.2. Выполните пошаговую отладку кода для выявления и исправления ошибок.  1) В Visual Studio Code перейдём во вкладку «Запуск и отладка» и установим инструмент отладки для Python, будет создан файл launch.json.  2) Далее в папке .vscode создадим файл tasks.json.  3) Сохраним все файлы.  4) Далее во вкладке «Запуск и отладка» необходимо нажать на кнопку «Начать отладку» для кода, который подготовлен для выполнения отладки – drone\_accelerometer.py    **Ниже представлен код для выполнения отладки:**  import numpy as np  import unittest  # Константы для параметров БЛА  MAX\_PAYLOAD = 1500  # максимальная взлетная масса в граммах  MAX\_FLIGHT\_TIME = 30 \* 60  # максимальное время полета в секундах  MAX\_CONTROL\_DISTANCE = 1000  # максимальная дальность управления в метрах  POSITIONING\_ACCURACY = 1  # точность позиционирования в метрах  def data\_acs(accel\_data, weight, flight\_time, control\_distance):      """      Обработка данных с акселерометра с учетом параметров БЛА.      Args:          accel\_data: Список значений ускорения по осям X, Y, Z (g).          weight: Масса БЛА в граммах.          flight\_time: Время полета в секундах.          control\_distance: Дальность управления в метрах.      Returns:          Словарь с обработанными данными:          - acceleration: Список значений ускорения (g).          - inclination: Список углов наклона по осям X, Y (град).      """      if weight > MAX\_PAYLOAD:          raise ValueError("Превышена максимальная взлетная масса БЛА.")      if flight\_time > MAX\_FLIGHT\_TIME:          raise ValueError("Превышено максимальное время полета.")      if control\_distance > MAX\_CONTROL\_DISTANCE:          raise ValueError("Превышена максимальная дальность управления.")      acceleration\_x = np.mean(accel\_data[0])      acceleration\_y = np.mean(accel\_data[1])      acceleration\_z = np.mean(accel\_data[2])      inclination\_x = np.arctan2(acceleration\_x, np.sqrt(np.square(acceleration\_y) + np.square(acceleration\_z))) \* 180 / np.pi      inclination\_y = np.arctan2(acceleration\_y, np.sqrt(np.square(acceleration\_x) + np.square(acceleration\_z))) \* 180 / np.pi      return {          "acceleration": [acceleration\_x, acceleration\_y, acceleration\_z],          "inclination": [inclination\_x, inclination\_y]      }  if \_\_name\_\_ == "\_\_main\_\_":      accel\_data = [[2, 2, 2], [5, 5, 5], [8, 8, 8]]      weight = 1200  # масса БЛА в граммах      flight\_time = 1600  # время полета в секундах      control\_distance = 800  # дальность управления в метрах      try:          data\_acs\_obj = data\_acs(accel\_data, weight, flight\_time, control\_distance)          print("Обработка данных:")          print(f"Ускорение: {data\_acs\_obj['acceleration']}")          print(f"Углы наклона: {data\_acs\_obj['inclination']}")      except ValueError as e:          print(e)  class TestDataAcs(unittest.TestCase):      def test\_data\_processing(self):          accel\_data = [[2, 2, 2], [5, 5, 5], [8, 8, 8]]          weight = 1200  # допустимая масса БЛА          flight\_time = 1600  # допустимое время полета          control\_distance = 800  # допустимая дальность управления          result = data\_acs(accel\_data, weight, flight\_time, control\_distance)          self.assertAlmostEqual(result['acceleration'][0], 2.0)          self.assertAlmostEqual(result['acceleration'][1], 5.0)          self.assertAlmostEqual(result['acceleration'][2], 8.0)          self.assertAlmostEqual(result['inclination'][0], 11.96946312460731, places=2)          self.assertAlmostEqual(result['inclination'][1], 31.23006977557103, places=2)  if \_\_name\_\_ == "\_\_main\_\_":      unittest.main(verbosity=2)  **Код для файла launch.json**  {      "version": "0.2.0",      "configurations": [          {              "name": "Отладка Python с LLDB",              "type": "python",              "request": "launch",              "program": "${workspaceFolder}/drone\_accelerometer.py",  // Необходимо указать имя тестируемого файла              "console": "integratedTerminal",              "justMyCode": true          }      ]  }  **Код для файла tasks.json**  {      "version": "2.0.0",      "tasks": [          {              "label": "Запуск тестов",              "type": "shell",              "command": "python -m unittest discover -s . -p '\*\_test.py'",              "group": {                  "kind": "test",                  "isDefault": true              },              "problemMatcher": ["$pythonUnittest"]          }      ]  }  Скринкопия результатов: |
| 4 | Применение методов тестирования и верификации. | 4.1. Разработайте и выполните модульные тесты для проверки отдельных компонентов системы (например, PyTest для модульного тестирования).  1) Выполним установку PyTest:    2) В корневом каталоге проекта ACCELEROMETER4 создадим файл drone\_accelerometer\_test.py  **Ниже представлен код для выполнения модульного теста:**  import numpy as np  import pytest  from drone\_accelerometer import data\_acs  # Константы для теста  MAX\_PAYLOAD = 1500  def test\_data\_processing():      accel\_data = [[2, 2, 2], [5, 5, 5], [8, 8, 8]]      weight = 1200      flight\_time = 1600      control\_distance = 800      result = data\_acs(accel\_data, weight, flight\_time, control\_distance)      assert np.isclose(result['acceleration'][0], 2.0, atol=1e-2)      assert np.isclose(result['acceleration'][1], 5.0, atol=1e-2)      assert np.isclose(result['acceleration'][2], 8.0, atol=1e-2)      assert np.isclose(result['inclination'][0], 11.969, atol=1e-2)      assert np.isclose(result['inclination'][1], 31.230, atol=1e-2)  def test\_exceed\_payload():      accel\_data = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]      weight = 1600  # Превышение максимальной массы      flight\_time = 10      control\_distance = 500      with pytest.raises(ValueError, match="Превышена максимальная взлетная масса БЛА."):          data\_acs(accel\_data, weight, flight\_time, control\_distance)  def test\_exceed\_flight\_time():      accel\_data = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]      weight = 1200      flight\_time = 4000  # Превышение максимального времени полета      control\_distance = 500      with pytest.raises(ValueError, match="Превышено максимальное время полета."):          data\_acs(accel\_data, weight, flight\_time, control\_distance)  def test\_exceed\_control\_distance():      accel\_data = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]      weight = 1200      flight\_time = 10      control\_distance = 1500  # Превышение максимальной дальности управления      with pytest.raises(ValueError, match="Превышена максимальная дальность управления."):          data\_acs(accel\_data, weight, flight\_time, control\_distance)  if \_\_name\_\_ == "\_\_main\_\_":      pytest.main()  **Результаты теста:** |
| 4.2. Выполните интеграционное тестирование для проверки взаимодействия между компонентами.  1) В корневом каталоге проекта ACCELEROMETER4 создадим файл test\_drone\_accelerometer.py  **Ниже представлен код для выполнения модульного теста:**  import numpy as np  import unittest  from drone\_accelerometer import data\_acs  class TestIntegrationDataAcs(unittest.TestCase):      def test\_integration\_case\_success(self):          accel\_data = [[2, 2, 2], [5, 5, 5], [8, 8, 8]]          weight = 1200  # допустимая масса          flight\_time = 1600  # допустимое время полета          control\_distance = 800  # допустимая дальность управления          result = data\_acs(accel\_data, weight, flight\_time, control\_distance)          expected\_acceleration = [2.0, 5.0, 8.0]          expected\_inclination\_x = np.arctan2(2.0, np.sqrt(5.0 \*\* 2 + 8.0 \*\* 2)) \* 180 / np.pi          expected\_inclination\_y = np.arctan2(5.0, np.sqrt(2.0 \*\* 2 + 8.0 \*\* 2)) \* 180 / np.pi          # Проверяем результаты          self.assertAlmostEqual(result["acceleration"][0], expected\_acceleration[0])          self.assertAlmostEqual(result["acceleration"][1], expected\_acceleration[1])          self.assertAlmostEqual(result["acceleration"][2], expected\_acceleration[2])          self.assertAlmostEqual(result["inclination"][0], expected\_inclination\_x, places=2)          self.assertAlmostEqual(result["inclination"][1], expected\_inclination\_y, places=2)      def test\_exceed\_payload(self):          accel\_data = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]          weight = 1600  # превышение массы          flight\_time = 10  # допустимое время полета          control\_distance = 500  # допустимая дальность управления          with self.assertRaises(ValueError) as context:              data\_acs(accel\_data, weight, flight\_time, control\_distance)          self.assertEqual(str(context.exception), "Превышена максимальная взлетная масса БЛА.")      def test\_exceed\_flight\_time(self):          accel\_data = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]          weight = 1200  # допустимая масса          flight\_time = 4000  # превышение времени полета          control\_distance = 500  # допустимая дальность управления          with self.assertRaises(ValueError) as context:              data\_acs(accel\_data, weight, flight\_time, control\_distance)          self.assertEqual(str(context.exception), "Превышено максимальное время полета.")      def test\_exceed\_control\_distance(self):          accel\_data = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]          weight = 1200  # допустимая масса          flight\_time = 10  # допустимое время полета          control\_distance = 1500  # превышение дальности управления          with self.assertRaises(ValueError) as context:              data\_acs(accel\_data, weight, flight\_time, control\_distance)          self.assertEqual(str(context.exception), "Превышена максимальная дальность управления.")  if \_\_name\_\_ == "\_\_main\_\_":      unittest.main(verbosity=2)  Далее запустим тесты с помощью команды в терминале: python –m unittest test\_drone\_accelerometer.py    **Итоговый перечень файлов в корневом каталоге проекта:** |