Оглавление

Список сокращений

МК – микроконтроллер;

ПО – программное обеспечение;

ОСРВ – операционная система реального времени;

I2C - Inter-Integrated Circuit;

UART - Universal Asynchronous Receiver-Transmitter;

DMA - Direct Memory Access;

ШИМ – широтно-импульсная модуляция;

WiFi - Wireless Fidelity

TCP - Transmission Control Protocol;

IP - Internet Protocol Address;

SWD – Serial Wire Debug.

1. Введение

В современном мире важную роль в построении систем управления летательными аппаратами играют алгоритмы стабилизации: с их помощью БПЛА может сохранять заданную скорость, высоту, положение в пространстве без участия человека. Такие системы применяются в оборонной и гражданской областях. Системы управления позволяют исключить ошибку пилота, обезопасить жизнь и здоровье человека, а также использовать летательные аппараты в труднодоступных и опасных для жизни человека местах.

В данной дипломной работе рассматривается алгоритм стабилизации скорости полета микро-БПЛА, а также создание макета системы управления микро-БПЛА. Микро-БПЛА считается летательный аппарат с максимальной взлетной массой менее 5кг, имеющий малые габариты. Микро-БПЛА работают в малом радиусе действия, а также являются малозаметными ввиду их габаритов и низкого уровня издаваемого шума. В данной работе таким аппаратом является беспилотный вертолет.

БПЛА оснащен системой управления со стабилизацией по высотно-скоростным параметрам. Одним из основных высотно-скоростных параметров является скорость полета летательного аппарата. Система стабилизации скорости полета позволяет исключить участие человека в процессе поддержания необходимой скорости, а также исключает ошибку пилотирования летательным аппаратом. В связи с этим, к алгоритму стабилизации предъявляются требования по быстродействию, точности, компактности, результативности.

В данной работе макет системы управления микро-БПЛА представляет собой микроконтроллер, реализующий алгоритм стабилизации скорости полета летательного аппарата, набор периферийных устройств, представляющий собой датчики давления, угловой скорости, ускорения, магнитного поля Земли, макет автомата-перекоса, состоящий из сервоприводов и мотора бесколлекторного типа. Данная система должна выдавать значения высоты, заданной и текущей

скорости, углов тангажа, крена, курса, посредством сигнала WiFi. Данный сигнал принимается с компьютера и все полученные данные визуализируются в специальном приложении. С учетом всех перечисленных данных, система должна вырабатывать управляющие сигналы, которые подаются на сервоприводы и бесколлекторный мотор. Управляющие сигналы берутся из расчетов текущей скорости полета летательного аппарата. Высота, а также скорость полета, вычисляется с помощью давлений, принимаемых с установленных датчиков. Углы тангажа, крена и курса вычисляются с помощью показаний датчиков угловой скорости, ускорения, магнитного поля Земли. При заданной скорости, равной нулю, необходимо организовать алгоритм висения. Поскольку одним из основных требований к алгоритму стабилизации является быстродействие, программа выполнена на операционной системе реального времени FreeRTOS. Использование ОСРВ позволяет максимально эффективно использовать ресурсы МК, а также увеличивает скорость реакции контроллера на внешние прерывания.

С помощью датчиков давления, температуры, угловой скорости, ускорения и магнитного поля Земли, система получает информацию о текущих высотно-скоростных параметрах полета из окружающей среды, после чего передает эти данные в вычислитель. Вычислитель обрабатывает полученную информацию и вырабатывает управляющие команды для исполнительных устройств. Также вычислитель принимает данные с персонального компьютера о заданной скорости, которая учитывается при выработке управляющих команд.

Структурная схема работы системы управления микро-БПЛА представлена на рис.1.1.

Рис. 1.1 Структурная схема системы управления.

2. Специальная часть

2.1 Разработка макета системы управления микро-БПЛА и структуры ПО

2.1.1 Разработка макета системы управления

Макет системы управления состоит из нескольких частей: модель автомата-перекоса, платы управления, а также блока питания.

2.1.1.1 Автомат-перекос

Автомат-перекос был изобретен Б. Н. Юрьевым в 1911 г. и применяется во всех вертолетах. Управление вертолетом осуществляется несущим винтом путём изменения у него общего и циклического шага при помощи автомата-перекоса. Он размещен на втулке несущего винта и передает движение от невращающихся элементов на вращающиеся лопасти винта.

2.1.1.1.1 Принцип действия автомата-перекоса

На рис.2.1 изображен автомат-перекос несущего винта. Он состоит из неподвижного и подвижного колец, которые соединены подшипником. К подвижному кольцу крепятся тяги, соединенные с рычагами лопастей. К неподвижному кольцу крепятся тяги, соединенные с приводами управления.

При отклонении колец автомата-перекоса происходит циклическое изменение угла наклона лопастей. При прохождении конца тяги лопасти нижнего положения кольца, угол установки лопасти будет минимальным, а через 180° по азимуту он будет максимальным. На жестком винте циклическое изменение угла установки приводит к увеличению подъемной силы на одной стороне, где углы установки больше среднего, а на противоположной – к уменьшению подъемной силы. В результате общая тяга винта смещается параллельно оси вращения в сторону больших углов установки. Таким образом можно создать момент относительно любой оси, лежащей в плоскости вращения несущего винта, путём отклонения колец в соответствующую сторону.

Соответственно, создавая момент в любом направлении, можно осуществлять продольное и поперечное управление вертолетом.

Наиболее распространенным является автомат-перекос кольцевого типа, приводы управления которого расположены под 120° друг к другу.

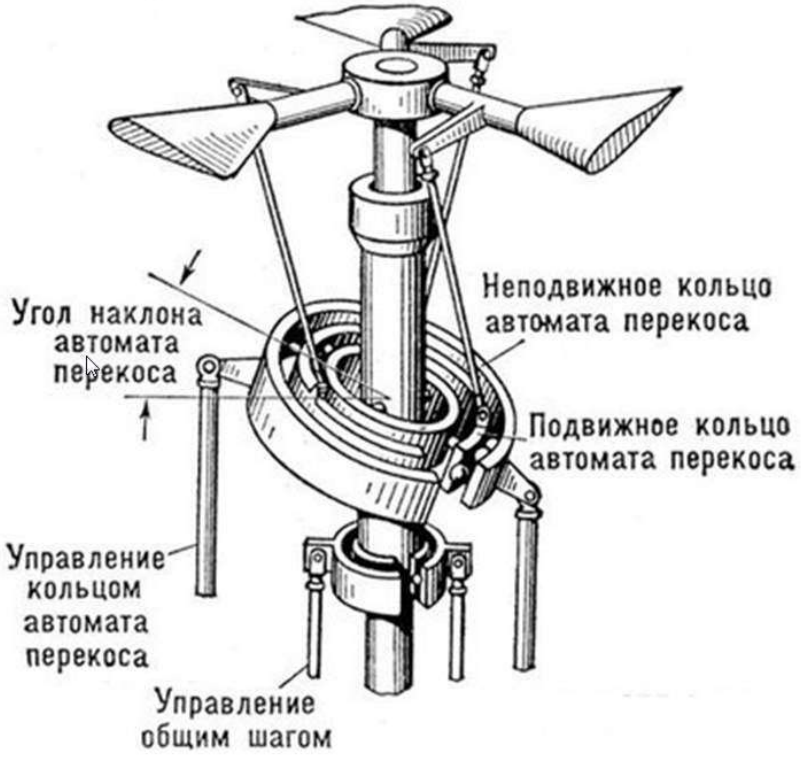


Рис.2.1 Автомат-перекос.

2.1.1.1.2 Приводы управления автомата-перекоса

Поскольку в данной работе рассматривается макет микро-БПЛА, в качестве управляющих приводов используются сервоприводы малой мощности Tower Pro SG90. Его изображение представлено на рис.2.2.

Характеристики сервопривода:

1. Материал редуктора: нейлон;
2. Напряжение питания: 4,8 – 6,0В;
3. Усилие на валу: 1,2 кг/см (4,8В); 1,6 кг/см (6,0В);

4. Ширина импульса: 2мс;
5. Размеры: 22мм х 11,5мм х 27мм;
6. Масса: 9г;
7. Угол поворота: 0 – 180°.

Рис.2.2 Сервопривод SG90.

Сервопривод имеет 3 контакта: земля, питание и контакт, на который подается ШИМ сигнал для управления углом поворота вала сервопривода.

2.1.1.1.3 Силовая установка

В качестве силовой установки в данном макете используется электромотор бесколлекторного типа A2212/13T. Его изображение представлено на рис.2.3.

Характеристики электромотора:

1. Оборот/В: 1000;
2. Максимальная эффективность: 80%;
3. Напряжение питания: 2 – 3S LiPo (7,4В – 12,6В);
4. Максимальная нагрузка по току: 12А/60с;
5. Максимальная мощность: 150Вт;
6. Размеры двигателя: 27,5мм х 30мм;
7. Диаметр вала: 3,2мм;
8. Масса: 52,7г.

Рис.2.3 Бесколлекторный мотор A2212/13T

Данный мотор является трехфазным, порядок подключения каждого из трех контактов к регулятору оборотов не имеет значения.

Поскольку данный мотор является трехфазным, для его управления необходимо применение регулятора оборотов. Для имеющейся модели мотора применяется регулятор ESC-30A.

Характеристики регулятора оборотов:

1. Размер: 57мм x 25мм x 8мм;
2. Входное напряжение: 6В – 13В;
3. Непрерывный выходной ток: 30А;
4. Максимальный выходной ток: 35А/12с;
5. Масса: 25г.

Регулятор оборотов имеет контакты на вход: земля, питание и контакт, на который подается ШИМ сигнал для регулировки оборотов мотора. Также у регулятора оборотов имеется два контакта на выход – контакт земли и контакт выходного напряжения 5В, однако они не будут использованы в данном макете. Помимо приведенных контактов, на выходе регулятора оборотов имеется 3 контакта для управления бесколлекторным мотором.

Из технических характеристик регулятора оборотов и мотора следует их полная совместимость, поскольку выходной ток регулятора превышает потребляемый ток мотором.

2.1.1.2 Микроконтроллер

Алгоритм стабилизации скорости полета микро-БПЛА в данной работе реализуется с помощью микроконтроллера STM32F103C8T6.

Основные технические характеристики микроконтроллера:

1. Диапазон рабочего напряжения: 2В – 3,6В;
2. Диапазон рабочей температуры: -40°С – 85°С;
3. Объем flash памяти: 64КБ;
4. Объем оперативной памяти: 20КБ;
5. Интерфейсы: 2 x SPI, 2 x I2C, 3 x USART, 1 x CAN;
6. 7-канальный DMA контроллер;
7. Количество портов ввода/вывода: 37;
8. Тактовая частота: 72Мгц.

Микроконтроллер распаян на отладочной плате, имеет необходимую обвязку, выполненную с помощью SMD компонентов. Программируется данный контроллер посредством интерфейса SWD. Изображение данной отладочной платы представлено на рис.2.4.

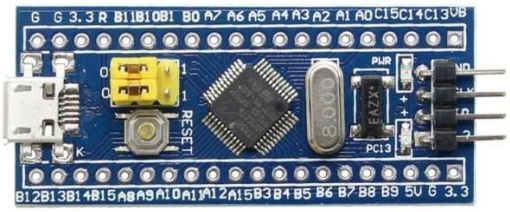


Рис.2.4 Отладочная плата на микроконтроллере STM32F103C8T6.

2.1.1.3 Датчик давления

Для нахождения воздушной скорости полета летательного аппарата и высоты необходимо применение датчиков давления. В данной работе используется датчик давления и температуры BMP280. Сам датчик распаян на

плате с необходимой обвязкой для него, а также выводом необходимых портов для передачи данных и питания всего модуля. Его изображение представлено на рис.2.5.

Характеристики датчика:

1. Напряжение питания: 1,7В – 3,6В;
2. Максимальный ток потребления: 2,7мкА;
3. Диапазон рабочей температуры: -40°С – 85°С;
4. Диапазон измерения давления: 300гПа – 1100гПа;
5. Погрешность барометра: 0,12гПа;
6. Интерфейсы: I2C, SPI;
7. Размеры модуля: 12мм х 16мм.

Поскольку для измерения воздушной скорости необходимо знать полное и статическое давления, в макете используется два датчика, что является минимальным количеством для построения данной системы.

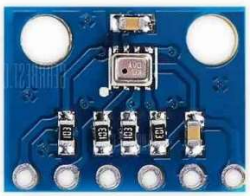


Рис.2.5 Датчик давления BMP280.

В таблице 2.1 представлено описание портов модуля BMP280 согласно технической документации производителя. Схема расположения портов изображена на рис. 2.6(вид сверху).

Таблица 2.1 описание портов модуля BMP280

Номер порта	Обозначение	Описание
1	GND	Земля
2	CSB	Перезапуск модуля
3	SDA	Линия данных I2C
4	SCL	Линия тактирования I2C
5	SDO	Порт, используемый для установки адреса модуля
6	V_{DDIO}	Питание модуля
7	GND	Земля
8	V_{DD}	Питание модуля

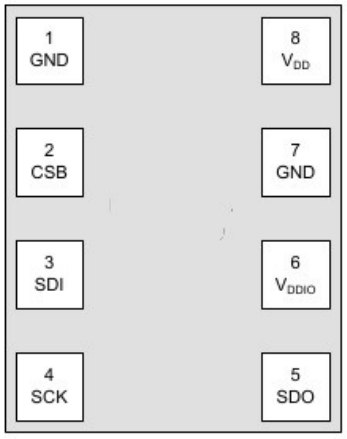


Рис.2.6 Схема портов датчика BMP280(вид сверху).

2.1.1.4 Акселерометр, гироскоп, магнитометр

Для того, чтобы организовать режим висения микро-БПЛА при введенной заданной скорости, равной нулю, необходимо вычисление углов крена и тангажа. Также техническое задание предполагает вывод углов тангажа, крена и

курса, поэтому необходимо использовать датчики угловой скорости, ускорения и магнитного поля Земли. Для реализации данной части системы управления решено использовать модуль MPU9250, который включает в себя акселерометр, гироскоп и магнитометр. Его изображение представлено на рис.2.7.

Характеристики модуля MPU9250:

1. Напряжение питания: 3В – 5В;
2. Интерфейсы: I2C, SPI;
3. Размеры: 25мм х 16мм х 3мм;
4. Масса: 2г.

Характеристики акселерометра:

1. Диапазон измерения: 2G, 4G, 8G, 16G;
2. Рабочий ток: 450мкА;

Характеристики гироскопа:

1. Диапазон измерения: 250, 500, 1000, 2000 °/с;
2. Рабочий ток: 3,2мА;

Характеристики магнитометра:

1. Диапазон измерения: 4800мкТл;
2. Рабочий ток: 280мкА.

Данный модуль отлично подходит для быстрого и точного определения углов тангажа и крена, однако магнитометр выдает плохие данные, поскольку в условиях помещения вектор магнитной напряженности сильно меняется из-за окружающего электрооборудования.

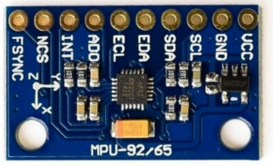


Рис.2.7 Модуль MPU9250.

В таблице 2.2 представлено описание портов модуля MPU9250 согласно технической документации производителя.

Схема расположения портов на модуле изображена на рис 2.8.

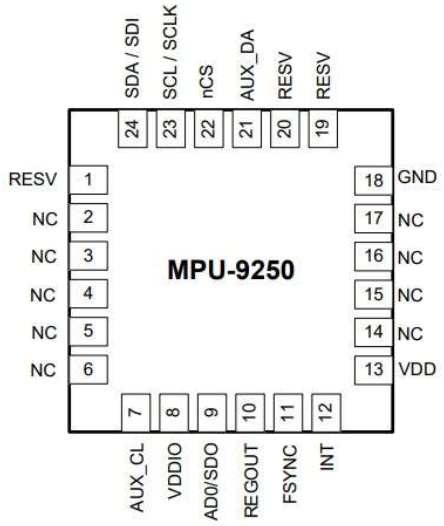


Рис. 2.8 Схема расположения портов модуля MPU9250

Таблица 2.2 описание портов модуля MPU9250

Номер порта	Обозначение	Описание
1	RESV	Зарезервирован
7	AUX_CL	Линия тактирования
8	VDDIO	I2C (внутри модуля) Питание модуля
9	AD0 / SDO	Порт, используемый для установки адреса модуля
10	REGOUT	Регулятор фильтра
11	FSYNC	Синхронизация
12	INT	Прерывание
13	VDD	Питание модуля
18	GND	Земля
19	RESV	Зарезервирован

20	RESV	Зарезервирован
21	AUX_DA	Линия данных I2C (внутри модуля)
22	nCS	Порт перезагрузки
23	SCL / SCLK	Линия тактирования I2C
24	SDA / SDI	Линия данных I2C
2 – 6, 14 – 17	NC	Не подключены

2.1.1.5 Модуль беспроводной связи

Техническое задание к данной дипломной работе предполагает визуализацию значений высоты, текущей и заданной скоростей полета, углов тангажа, крена и курса, а также изменение заданной скорости. Для реализации этой части системы управления решено использовать WiFi модуль, с помощью которого происходит обмен данными между компьютером и МК посредством приложения. Для экономии средств, а также ускорения процесса разработки системы был выбран WiFi модуль ESP8266 – 12E. Его изображение представлено на рис.2.9.

Характеристики WiFi модуля:

1. Рабочее напряжение: 3В – 3,6В;
2. Рабочий ток: 80мА;
3. Диапазон рабочих температур: -40°С – 125°С;
4. WiFi протоколы: 802.11 b/g/n;
5. Частотный диапазон: 2,4ГГц – 2,5ГГц;
6. Режимы: станция, точка доступа, точка доступа + станция;
7. Безопасность: WPA/WPA2;
8. Сетевые протоколы: IPv4, TCP/UDP/HTTP/FTP;
9. Размеры: 16мм х 24мм х 3мм.

Рис.2.9 WiFi модуль ESP8266 – 12E.

В таблице 2.3 представлено описание портов WiFi модуля ESP8266-12E согласно документации производителя.

Схема расположения портов модуля изображена на рис. 2.10.

Таблица 2.3 описание портов WiFi модуля ESP8266-12E

Номер порта	Обозначение	Описание
1	RST	Перезапуск модуля
2	ADC	АЦП
3	EN	Режим программирования
8	VCC	Питание модуля
9	CS0	Выбор ведомого (SPI)
10	MISO	Вход ведущего (SPI)
13	MOSI	Выход ведущего (SPI)
14	SCLK	Тактовый сигнал (SPI)
15	GND	Земля
21	RX	Линия приема (UART)
22	TX	Линия передачи (UART)
4 – 7, 11, 12, 16 – 20	GPIO	Порты ввода/вывода общего назначения

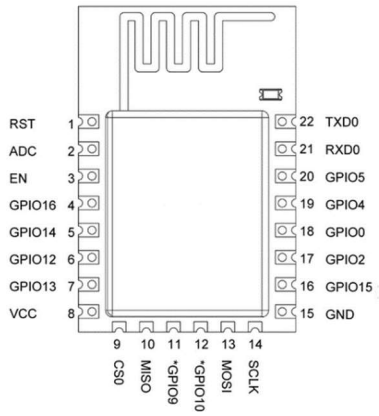


Рис. 2.10 Схема расположения портов WiFi модуля ESP8266 – 12E.

2.1.1.6 Питание макета системы управления

Питание макета осуществляется при помощи блока питания, выходное напряжение которого 12В. Исходя из технических характеристик используемых в макете устройств, напряжение питания всей системы варьируется от 3,3В до 12В.

Поскольку одним из требований к системе является компактность, необходимо развести на одной плате:

1. Питание МК, датчиков и WiFi модуля: 3,3В;
2. Питание сервоприводов: 4,8В – 6В;
3. Питание мотора: 12В.

Для того, чтобы организовать питание всей системы от одного источника, необходимо использование стабилизаторов напряжения. Исходя из технических характеристик сервоприводов, напряжением питания выбрано напряжение 5В. Для получения такого напряжения используется стабилизатор напряжения КР142ЕН5А.

Характеристики КР142ЕН5А:

1. Максимальное входное напряжение: 15В;

2. Выходное напряжение: 5В;
3. Выходной ток: 2А;
4. Точность выходного напряжения: 0,05В;
5. Диапазон рабочих температур: -40°С – 125°С.

Для организации напряжения питания МК и датчиков выбран стабилизатор AMS1117.

Характеристики AMS1117:

1. Максимальное входное напряжение: 15В;
2. Максимальный выходной ток: 1А;
3. Выходное напряжение: 3,3В;
4. Диапазон рабочих температур: -20°С – 125°С.

Исходя из технических характеристик всех используемых компонентов системы, стабилизаторы удовлетворяют требованиям по току.

Также используются конденсаторы различной емкости для снижения негативных воздействий питающих напряжений.

2.1.1.7 Схема подключения датчиков и исполнительных устройств к МК

Схема подключения датчиков, а также электрическая схема созданы в среде проектирования электроники EasyEDA.

Интерфейс данной среды представлен на рис. 2.11.

Слева в окне приложения находятся разделы, позволяющие просматривать библиотеки типовых элементов электрических схем, которые помещаются на основное поле. В последствии добавленные элементы могут быть соединены с помощью кнопок управления, расположенных в окне «Wiring Tools».

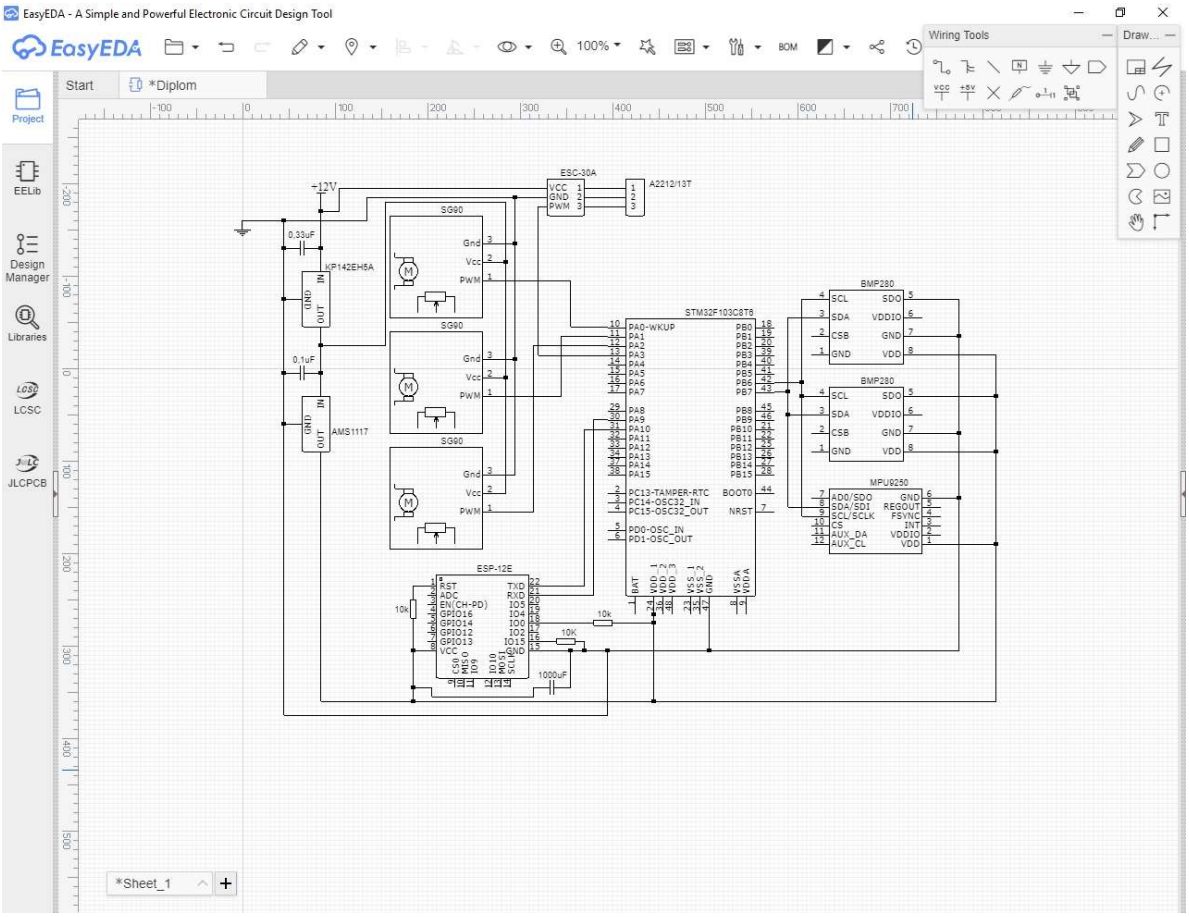


Рис. 2.11 Интерфейс EasyEDA

В данной системе управления датчики подключаются параллельно к МК посредством последовательного интерфейса I2C, а WiFi модуль с помощью интерфейса UART.

Однако, модули BMP280 изначально имеют одинаковый адрес, по которому МК обращается к устройствам. Для того, чтобы переназначить адрес датчиков, необходимо у одного из них подключить порт SDO к земле, а у другого – к питанию.

Сервоприводы и мотор управляются с помощью ШИМ сигнала. Сервоприводы управляются непосредственно с МК, мотор – посредством регулятора оборотов.

Для успешного запуска WiFi модуля ESP8266 – 12E необходимо притянуть контакт RST к питанию, GPIO15 – к земле, GPIO0 – к питанию. Все соединения производятся через резистор номиналом 10кОм.

На рис. 2.12 изображена схема подключения датчиков и исполнительных устройств к МК.

2.1.1.8 Электрическая схема платы управления

На рис. 2.13 изображена электрическая схема платы управления. На схеме отображено преобразование питания из исходных 12В в 5В и 3,3В с помощью стабилизаторов напряжения, а также подключение всей необходимой периферии к МК: общая шина I2C, а которой параллельно подключены используемые датчики, а также шина обмена данными МК с WiFi модулем посредством интерфейса UART.

Реализация данной схемы производится на макетной плате с помощью бытового электропаяльника.

На рис. 2.14 изображена собранная плата управления.

2.1.1.9 Сборка макета системы управления

Поскольку в зависимости от модели вертолета изменяется конфигурация сервоприводов и моторов, использованных в нем, решено поместить силовую установку отдельно от платы управления.

В макете автомата-перекоса располагается мотор и три сервопривода, все их контакты собраны в одну косу и выведены в сторону для дальнейшего подключения к плате управления.

Собранный воедино макет системы управления представляет собой три блока: макет автомата-перекоса, плата управления и блок питания.

На рис 2.15 изображен собранный макет системы управления.

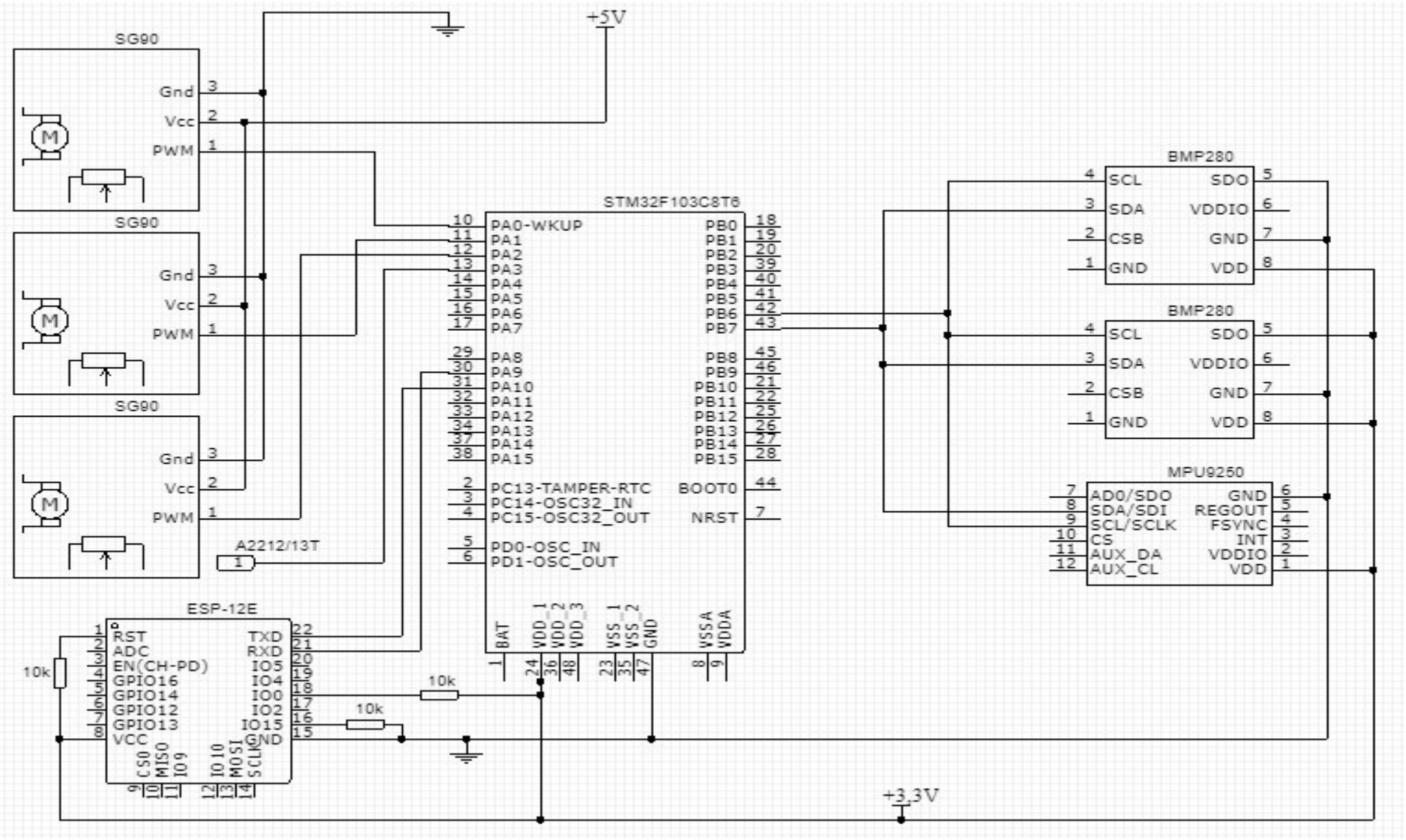


Рис. 2.12 Схема подключения датчиков и исполнительных устройств к МК.

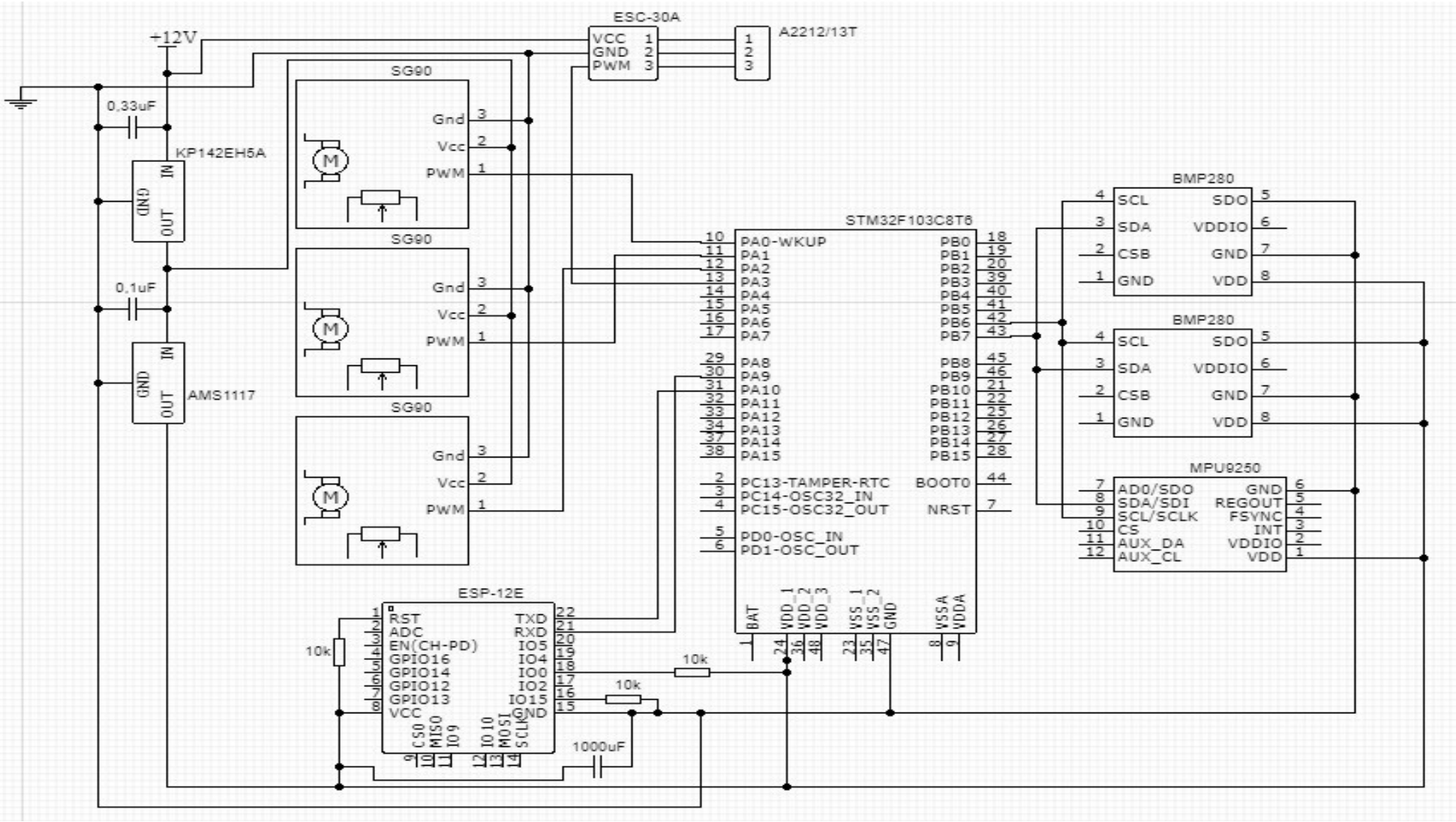


Рис. 2.13 Электрическая схема платы управления.

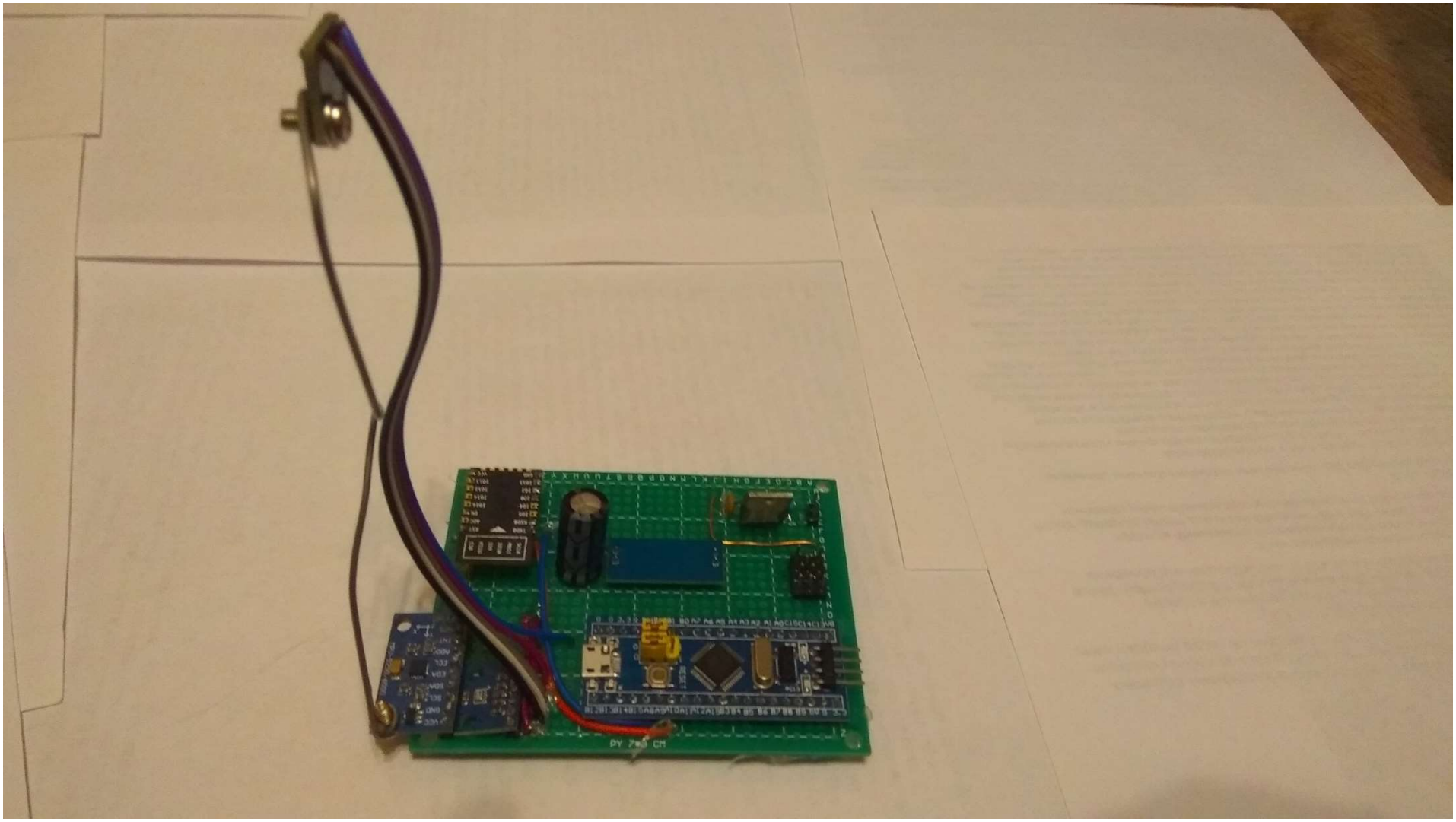


Рис 2.14 Плата управления

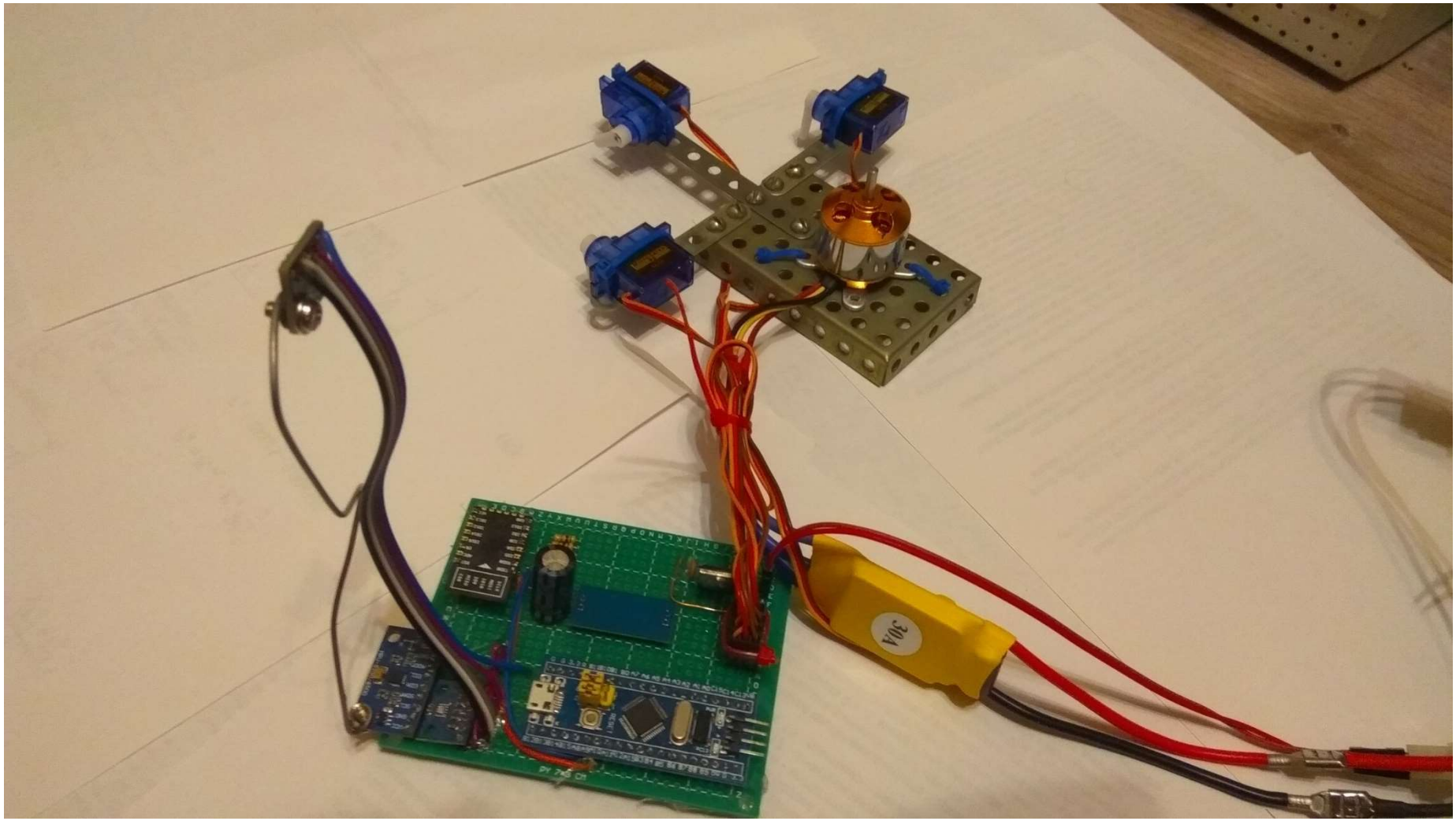


Рис 2.15 Собранный макет

2.1.2 Разработка структуры ПО

Поскольку алгоритм стабилизации скорости полета микро-БПЛА, рассматриваемый в данной дипломной работе реализован на микроконтроллере, а визуализация результатов работы алгоритма на персональном компьютере, программное обеспечение делится на две части – для МК и ПК отдельно.

2.1.2.1 Структура ПО для микроконтроллера

Алгоритм стабилизации скорости полета выполнен на микроконтроллере при использовании ОСРВ FreeRTOS. FreeRTOS является многозадачной системой жесткого реального времени, которая позволяет разрабатывать сложные устройства с большим функционалом. Поскольку алгоритм стабилизации скорости в ходе своей работы выполняет несколько задач (съем показаний с датчиков, выдача управляющего воздействия, прием данных с компьютера), необходимо использование многозадачности, что позволяет системе быстро реагировать на изменения и вырабатывать управляющие команды.

Для выполнения поставленной задачи, алгоритм делится на несколько частей:

1. Инициализация МК и периферии;
2. Прием данных от датчиков;
3. Функциональные вычисления;
4. Выработка управляющих команд;
5. Получение данных о заданной скорости с ПК.

Все части алгоритма, не считая инициализации, называются задачами, и выполняются в ОСРВ в соответствии с тем, какой из них передаст управление планировщик задач. Планировщик передает управление задачам в зависимости от типа выбранной многозадачности, а также исходя из приоритета каждой из них.

В данном алгоритме выбрана кооперативная многозадачность. При этом типе многозадачности планировщик передает управление следующей задаче только после того, как предыдущая сама передаст управление планировщику. Такой тип многозадачности выбран для достижения максимального быстродействия системы: когда текущая задача выполнится, начнет выполняться следующая и планировщик никогда не запустит пустую задачу, которая запускается в момент бездействия всех остальных. Также такой подход увеличит стабильность всей системы, поскольку может возникнуть конфликт между МК и периферией, когда МК за отведенное время на задачу не успевает принять данные с датчика, что может произойти при вытесняющей многозадачности.

Передача данных в ОСРВ FreeRTOS организована при помощи очередей, которые исключают потерю данных, нарушение последовательности при пересылке данных между задачами, а также является быстрым средством передачи информации внутри всей системы.

Получение данных с персонального компьютера реализовано при помощи прерываний. Такой метод позволяет не расходовать ресурсы МК в моменты, когда данные не поступают. Поскольку данные с ПК приходят посредством WiFi модуля в особом виде, для распознавания заданной скорости из всех полученных данных создается специальная задача. Она выполнится только тогда, когда получит данные из созданной для нее очереди, которая заполняется из прерывания.

В процессе инициализации МК и периферии происходит настройка используемых портов ввода/вывода МК, настройка интерфейсов I2C и UART, инициализация датчиков и WiFi модуля, в процессе которой происходят создание точки доступа TCP/IP соединения для связи МК с компьютером.

Структура программного обеспечения для микроконтроллера представлена на рис. 2.16.

Рис. 2.16 Структура ПО для микроконтроллера.

2.1.2.2 Структура ПО для персонального компьютера

Алгоритм визуализации результатов работы системы стабилизации реализован в приложении, созданном для ОС Windows. Его задача состоит в приеме данных с МК посредством сигнала WiFi через TCP/IP соединение. Данное приложение позволяет подключиться к созданному TCP/IP соединению, принимает данные, которые посылает МК, строит графики полученных данных, позволяет задавать необходимое пользователю значение скорости полета микро-БПЛА. Графики, построенные приложением, позволяют увидеть значения текущей и заданной скорости, текущей высоты и углов крена, курса, тангажа. При этом, все графики строятся параллельно друг другу, поскольку реализован функционал просмотра графиков во времени, поэтому каждая задача выполняется независимо от другой.

Структура ПО для персонального компьютера представлена на рис. 2.17.

Рис. 2.17 Структура ПО для персонального компьютера.

2.2 Разработка алгоритмов приема данных от датчиков, функциональных вычислений и выработки управляющих команд

2.2.1 Разработка алгоритма приема данных от датчиков

Алгоритм приема данных от датчиков представляет собой задачу в ОСРВ, реализующую последовательный опрос используемых датчиков с помощью функций, написанных для каждого датчика отдельно. Также эта задача передает полученные с датчиков данные в другую задачу при помощи очереди.

Поскольку в алгоритме стабилизации скорости полета использован кооперативный тип многозадачности, задача ожидает получение управления от планировщика, а также передает ему управление после выполнения своего кода.

Так как используется два одинаковых датчика давления и температуры, им были присвоены разные адреса при подключении к МК. Соответственно, необходимо опросить датчики с помощью одной и той же функции, но с разными адресами.

После чтения датчиков давления и температуры, происходит опрос модуля MPU9250, включающего в себя датчики угловой скорости, ускорения и магнитного поля Земли. После того, как данные были получены, рассчитываются углы тангажа, крена и курса.

Когда все необходимые данные получены, они отправляются в очередь. Из этой очереди следующая задача получит необходимые ей данные для дальнейших функциональных вычислений.

Функциональная схема алгоритма приема данных от датчиков представлена на рис. 2.18.

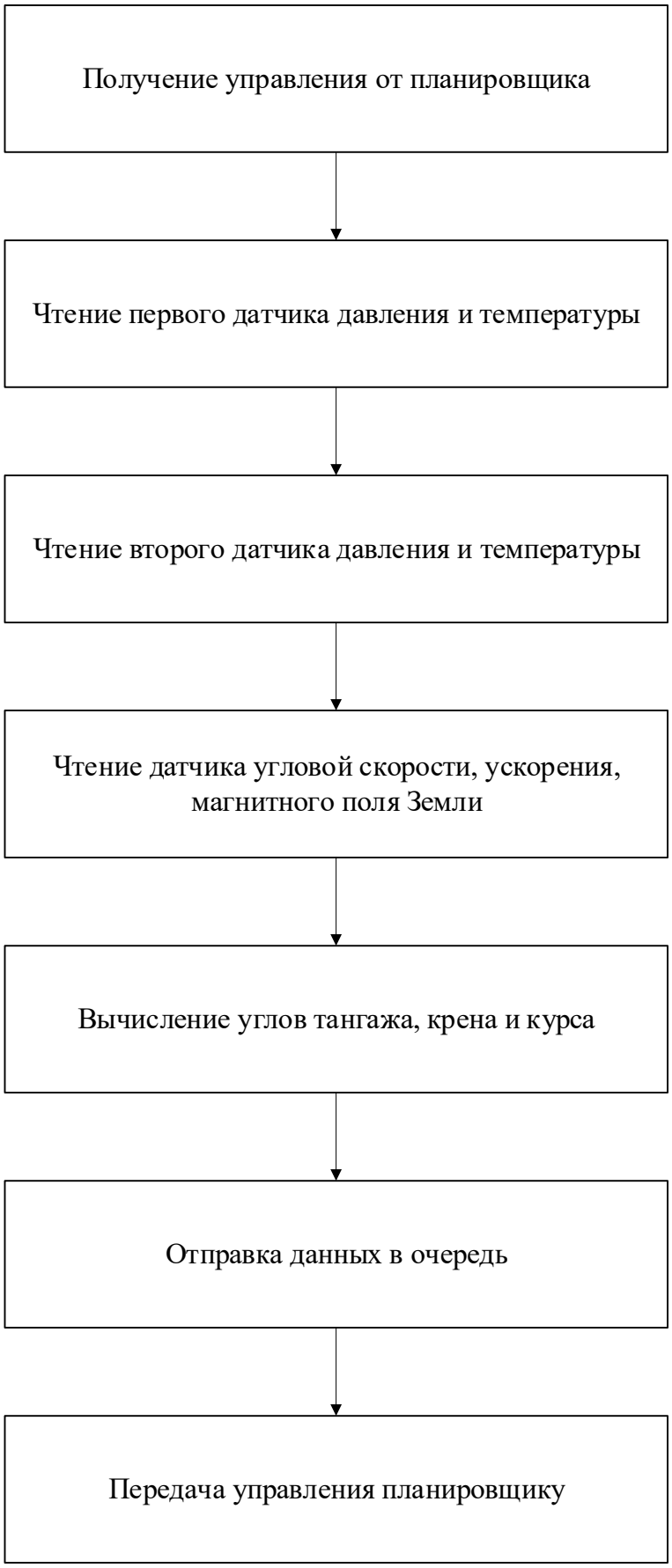


Рис. 2.18 Функциональная схема алгоритма приема данных от датчиков

2.2.2 Разработка алгоритма функциональных вычислений

Алгоритм функциональных вычислений представляет собой задачу в ОСРВ, которая производит вычисление необходимых углов положения сервоприводов и расчет необходимой тяги мотора для достижения заданной скорости полета микро-БПЛА. Все вычисления производятся исходя из полученных данных с датчиков с учетом введенной на компьютере заданной скорости полета БПЛА.

Поскольку получение данных с компьютера реализовано при помощи прерывания по приему, возникает необходимость в дополнительной задаче по поиску среди полученных данных необходимого значения заданной скорости. Соответственно, алгоритм функциональных вычислений состоит из двух задач: задача по вычислению необходимых значений высоты, углов положения сервоприводов и тяги с последующей передачей полученных значений на компьютер, а также задача по получению заданной скорости полета с компьютера.

2.2.2.1 Алгоритм получения заданной скорости полета с компьютера.

Значение заданной скорости полета вводится на компьютере, после чего отправляется на МК при помощи WiFi модуля. Когда МК принимает данные с WiFi модуля, возникает прерывание по приему, в котором происходит передача полученной информации в задачу по поиску значения заданной скорости в принятой строке посредством очереди.

В задаче по поиску происходит нахождение символов, обозначающих, что полученные данные приняты с компьютера, после чего в данной строке берутся символы, указывающие на введенную скорость с компьютера. Данные символы переводятся в численный тип данных и отправляются в очередь, данные из которое забирает задача, выполняющая необходимые системе вычисления.

Эта задача производит выполнение своего алгоритма только в случае получения данных из прерывания, в остальных случаях она сразу передает управление планировщику.

Функциональная схема алгоритма получения заданной скорости с компьютера представлена на рис. 2.19.

Рис. 2.19 Функциональная схема алгоритма приема заданной скорости с компьютера.

2.2.2.2 Алгоритм функциональных вычислений

Вычисления высоты, текущей и заданной скоростей, углов положения сервоприводов, а также значения тяги мотора производятся после приема информации из очереди, которая заполняется задачей по приему данных с датчиков. В момент первого выполнения вычислений, давление, полученное с датчика, измеряющего статическое, берется за начальное. Это значение сохраняется в оперативной памяти МК и удаляется в момент отключения устройства от питания.

В остальные моменты времени алгоритм вычисляет текущую высоту, текущую скорость и задает значения для углов положения сервоприводов и тяги мотора.

Для нахождения относительной высоты полета барометрическим способом используется формула 2.1:

$$H = 44330 * \left(1 - \left(\frac{P_{ст}}{P_0}\right)^{\frac{1}{5{,}255}}\right), \quad (2.1)$$

где:

$P_{ст}$ – статическое давление;

P_0 – давление на уровне плоскости, с которой начинается полет.

Вычисление воздушной скорости производится по формуле 2.2:

$$V = \sqrt{\frac{2 * (P_{полн} - P_{ст})}{\rho}}, \quad (2.2)$$

где:

$P_{полн}$ – полное давление;

$P_{ст}$ – статическое давление;

ρ – плотность воздуха.

Плотность воздуха рассчитывается по формуле 2.3:

$$\rho = \frac{P_{ст} * M}{R * T}, \quad (2.3)$$

где:

$P_{ст}$ – статическое давление;

M – молярная масса воздуха;

R – универсальная газовая постоянная;

T – температура окружающего воздуха.

Тангаж рассчитывается по формуле 2.4:

$$\theta = \frac{180}{\pi} * \left(\operatorname{arctg}\left(\frac{Ax}{\sqrt{Ay^2 + Az^2}} \right) \right), \quad (2.4)$$

где:

Ax – показания акселерометра по оси x;

Ay – показания акселерометра по оси y;

Az – показания акселерометра по оси z.

Крен рассчитывается по формуле 2.5:

$$\phi = \frac{180}{\pi} * \left(\operatorname{arctg}\left(\frac{Ay}{\sqrt{Ax^2 + Az^2}} \right) \right), \quad (2.5)$$

где:

Ax – показания акселерометра по оси x;

Ay – показания акселерометра по оси y;

Az – показания акселерометра по оси z.

Курс расчитывается по формуле:

$$\psi = \frac{180}{\pi} * \operatorname{arctg}(\frac{Yh}{Xh})^2 \quad (2.6)$$

где:

$$Xh = Mx * \cos(\theta) + Mz * \sin(\theta),$$

$$Yh = Mx * \sin(\phi) * \sin(\theta) + My * \cos(\theta) - Mz * \sin(\phi) * \cos(\theta),$$

Mx – показания магнитометра по оси x;

My – показания магнитометра по оси y;

Mz – показания магнитометра по оси z.

В ходе выполнения задачи функциональных вычислений происходит сравнение заданной скорости с текущей, а также с нулем.

При включении устройства, начальное значение заданной скорости равно нулю, следовательно, алгоритм стабилизации будет выполнять режим висения: берутся значения крена и тангажа и в соответствии с ними присваиваются значения углов отклонения сервоприводов.

Предположим, что угол тангажа равен 5°, соответственно, необходимо передний сервопривод отклонить на -5°, а правый и левый на 5°. При отрицательном угле тангажа происходит то же самое с изменением знака на противоположный.

При отклонении БПЛА по углу крена, правый и левый сервоприводы отклоняются в соответствии со значением угла крена: если он положительный, правый сервопривод отклоняется на отрицательный угол, левый – на положительный, и наоборот.

Тяга мотора в режиме висения задается равной 50%.

В случаях, когда заданная скорость полета не равна нулю, происходит её сравнение с текущей: в случае, если текущая скорость меньше заданной, передний сервопривод отклоняется вниз, а правый и левый – вверх; если текущая

скорость больше заданной, передний сервопривод отклоняется вверх, а правый и левый – вниз. В первом случае происходит нарастание тяги мотора, во втором – убывание.

Поскольку необходимые значения углов отклонения сервоприводов очень сильно зависят от установленного автомата-перекоса, каждое изменение углов производится на 1° в режиме стабилизации скорости, тяга мотора изменяет свое значение на 10%. Учитывая, что алгоритм выполняется с высокой скоростью, стабилизация проходит плавно и приводит к нужному результату.

После выполнения функциональных вычислений, значения углов тангажа, крена, курса, высоты и скоростей отправляются на компьютер посредством сигнала WiFi, а значения углов положения сервоприводов и тяги мотора – в очередь. Данные из этой очереди принимает задача по выдаче управляющих команд.

Функциональная схема алгоритма функциональных вычислений представлена на рис. 2.20.

Рис. 2.20 Функциональная схема алгоритма функциональных вычислений.

2.2.3 Алгоритм выработки управляющих команд

Алгоритм выработки управляющих команд представляет собой задачу ОСРВ, которая принимает из очереди значения углов положения сервоприводов и тяги мотора. После того, как данные будут приняты из очереди, МК выдает ШИМ сигналы на сервоприводы. Каждый ШИМ сигнал вычисляется отдельно исходя из диапазона ширины импульса для сервоприводов и мотора.

Функциональная схема алгоритма выработки управляющих команд представлена на рис. 2.21.

Рис. 2.21 Функциональная схема алгоритма выработки управляющих команд.

2.3 Разработка ПО

Разработка алгоритма стабилизации скорости полета производится в среде разработки Keil uVision 5 при помощи библиотеки HAL. Данная библиотека предоставляет широкий спектр возможностей при создании программного обеспечения для микроконтроллеров STM32.

Одним из основных требований к алгоритму является быстродействие. Для ускорения работы программы используется ОСРВ FreeRTOS, а также DMA – контроллер прямого доступа к памяти.

2.3.1 Инициализация

Для инициализации периферии МК используется приложение CubeMX. Он позволяет быстро и правильно настроить необходимые интерфейсы, таймеры, порты ввода/вывода, а также прерывания и DMA. Выбрав необходимые параметры, CubeMX генерирует проект, который в последствии открывается в Keil uVision 5 с заданными параметрами инициализации.

CubeMX представляет визуальный графический редактор, позволяющий настроить периферию и сгенерировать проект.

На рис. 2.22 представлен интерфейс приложения CubeMX при создании проекта.

На вкладке «Pinout» слева расположены доступные интерфейсы для выбранного микроконтроллера. Здесь можно выбрать используемые в проекте интерфейсы. На вкладке «Clock configuration» производится настройка схемы тактирования микроконтроллера. На вкладке «Configuration» настраиваются интерфейсы, выбранные для данного проекта.

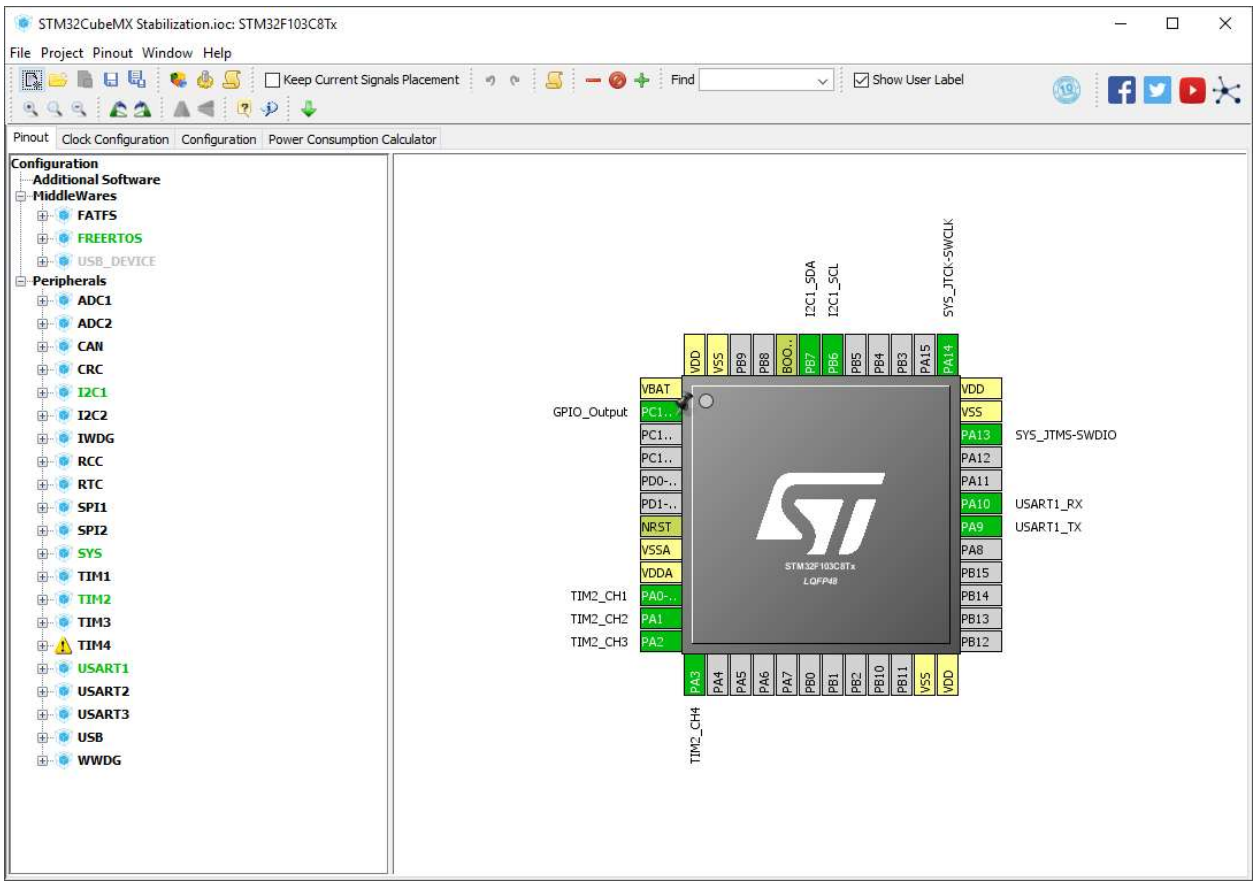


Рис. 2.22 Интерфейс CubeMX

Создание программного обеспечения для реализации алгоритма стабилизации производится в среде Keil uVision 5. В данной среде открывается проект, созданный в CubeMX с инициализированной периферией.

На рис. 2.23 представлен интерфейс приложения Keil uVision 5.

Окно среды состроит из нескольких частей: с левой стороны расположено дерево проекта, в котором можно выбрать необходимый для открытия или редактирования файл, по центру – область для редактирования кода, внизу – область для отображения ошибок/состояния компиляции при сборке проекта, сверху расположена панель элементов управления, позволяющих начать сборку проекта, запустить загрузку прошивки в микроконтроллер, войти в режим отладки и т.д.

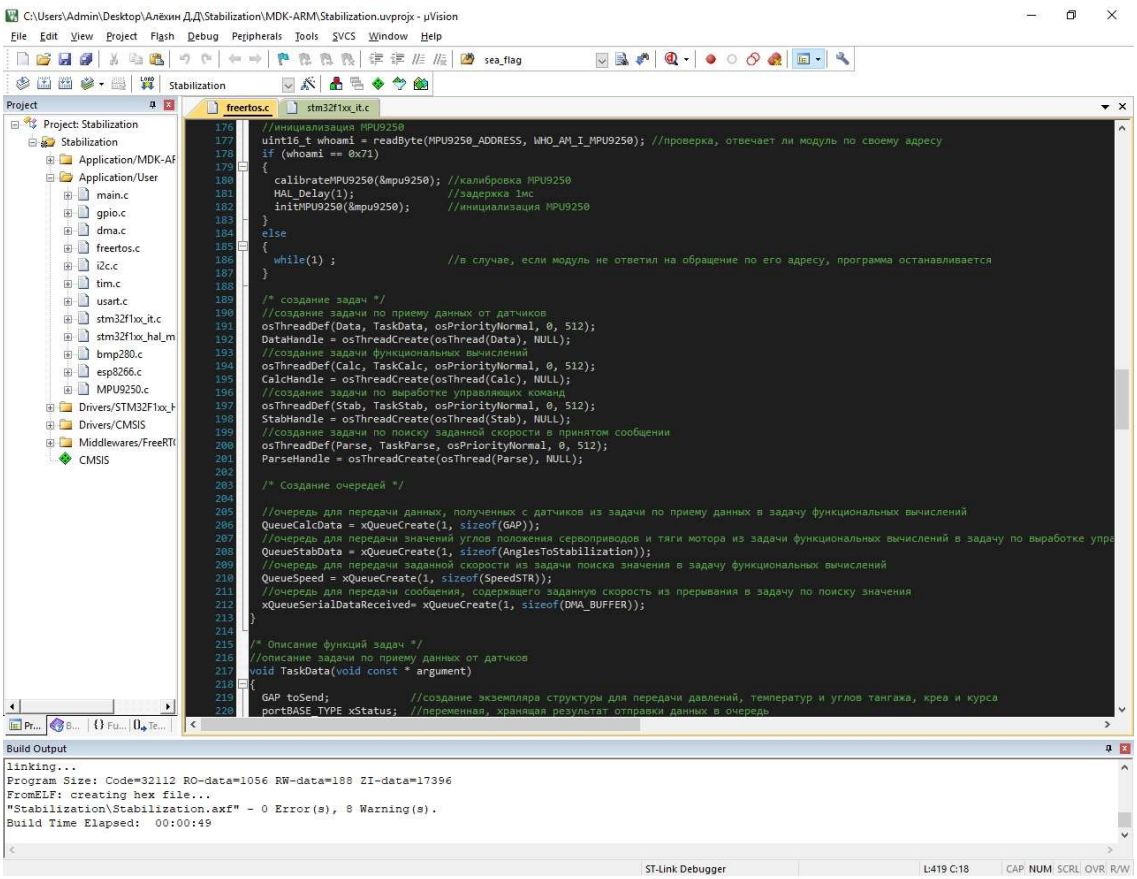


Рис 2.23 Интерфейс Keil uVision 5

2.3.1.1 Инициализация интерфейса I2C

Интерфейс I2C представляет собой последовательную ассиметричную шину, которая используется для связи между низкоскоростными периферийными устройствами и микроконтроллерами.

Для передачи данных шина I2C использует две двунаправленные линии: линию данных и линию тактирования. Всего по шине может быть подключено до 127 устройств.

Адресация включает в себя 7-битное адресное пространство с 16 зарезервированными адресами, следовательно, доступно до 112 адресов для подключения периферийных устройств на одну шину.

При обмене данными по шине I2C, одно из устройств является ведущим, а остальные – ведомыми. Генератором тактов на линии SCL выступает ведущий.

Началом процедуры обмена является формирование состояния старт: при высоком уровне на линии SCL генерируется переход сигнала из высокого состояния в низкое на линии SDA. Для всех устройств на шине этот переход обозначает начало передачи информации. Завершением процедуры обмена является формирование состояния стоп: при высоком уровне на линии SCL генерируется переход сигнала из низкого состояния в высокое на линии SDA. Состояния старт и стоп всегда вырабатываются ведущим устройством. Количество байт в сообщении не ограничено. Каждый байт, принятый ведомым-приёмником от ведущего-передатчика, должен быть подтвержден. Для этой цели служит специальный бит подтверждения, который выставляется на линию SDA после приёма одного байта данных. После успешного приема байта на линии SDA выставляется низкий уровень сигнала.

Ведущий не имеет права управлять переходом линии SCL из низкого состояния в высокое: в случаях, когда ведомому недостаточно времени на обработку принятого бита, он может удерживать линию SCL в низком состоянии до момента готовности к приему следующего бита.

Каждому устройству, подключенному к шине I2C, может быть программно присвоен уникальный адрес. Для выбора приемника ведущий использует данный адрес перед отправкой сообщения. В сообщении, первые 7 битов в первом байте занимает адрес устройства, 8 бит – бит, обозначающий направление передачи данных: 0 – запись информации ведущим в ведомого, 1 – считывание информации ведущим из ведомого.

После формирования состояния старт, каждый ведомый сравнивает первые семь бит со своим адресом. В случае совпадения, ведомый считается выбранным, и становится передатчиком, либо приемником, в зависимости от бита направления передачи данных.

В микроконтроллере STM32F103C8T6 шина I2C может работать на скоростях 100кГц и 400кГц, адресное пространство может быть как 7-битным,

так и 10-битным. Линия SCL в данном МК использует порт PB6, линия SDA – порт PB7. Тактирование шины производится от основной линии тактирования микроконтроллера.

Инициализация шины I2C заключается в установке скорости работы шины, адресного пространства, настройках скорости портов для линий SCL и SDA и т.д.

Поскольку используемые датчики BMP280 и MPU9250 могут работать на частотах в 400кГц и используют 7-битное адресное пространство, инициализация интерфейса I2C была сконфигурирована в соответствии с параметрами этих модулей.

В приложениях 2.1 и 2.2 представлен код инициализации шины I2C, который был сгенерирован с помощью CubeMX в соответствии с заданными настройками. Инициализация состоит из двух функций. В приложении А1 представлен код с конфигурацией настроек интерфейса I2C (установка адресного пространства, скорости работы шины и т.д.). В приложении А2 представлен код конфигурации портов PB6 и BP7, которые являются линиями SCL и SDA, а также включение прерываний по приему и ошибке на шине I2C.

2.3.1.2 Инициализация интерфейса UART

UART (УАПП, универсальный асинхронный приемник-передатчик, англ. Universal Asynchronous Receiver-Transmitter, UART) представляет собой последовательный интерфейс для приема и передачи данных между устройствами.

Передача данных между устройствами производится по одному биту в равные промежутки времени, которые определяются скоростью UART. Скорость UART измеряется в бодах – количестве бит в секунду. Существует ряд стандартных скоростей для данного интерфейса: 300; 600; 1200; 2400; 4800;

9600; 19200; 38400; 57600; 115200; 230400; 460800; 921600 бод. В отличие от интерфейса I2C, UART не имеет линии тактирования, и вместо тактового сигнала по линиям передачи данных добавляются биты старта и окончания передачи данных. Эти биты – синхронизирующие метки, которые автоматически вставляются в поток данных при передаче информации и удаляются при приеме.

В момент, когда не происходит информационного обмена между устройствами, на линиях приема и передачи установлен высокий уровень сигнала. Стартовый бит всегда является низким, поэтому устройство-приемник всегда ожидает перехода с высокого уровня на низкий на линии приема. Когда происходит такой переход, приемник запускает процесс приема минимальной посылки. Как правило, длина минимальной посылки имеет размер 8 бит. В процессе приема посылки приемник отсчитывает 9 битовых длительностей подряд, постоянно фиксируя состояние входа. Первые 8 принятых значений являются принятыми данными, а последнее – стоп бит, который всегда равен высокому значению. В случае, когда значение последнего бита не равно 1, интерфейсом UART фиксируется ошибка. Иногда, для повышения надежности синхронизации передачи данных используется два стоп бита, однако данный метод снижает скорость передачи.

Поскольку биты синхронизации занимают пространство в потоке передаваемых данных, реальная скорость передачи информации посредством UART получается меньше заданной скорости соединения. При длине минимальной посылки в 8 бит, синхронизирующие биты занимают 20% потока данных, что означает, что при скорости соединения 115200 бод скорость передачи информации составляет 11520 байт/с.

Также UART способен контролировать целостность переданных данных путем контроля битовой четности. При реализации данного функционала последний бит минимальной посылки контролируется логикой интерфейса и содержит информацию о четности количества единичных бит в переданной посылке. Различают контроль на четность и нечетность. При приеме посылки

UART автоматически контролирует бит четности и фиксирует признаки правильного или неправильного приема.

Инициализация интерфейса UART заключается в установке скорости соединения, определения длины минимальной посылки, а также выбора количества стоп битов. Также во время инициализации происходит настройка портов линий приема и передачи, установка скорости этих портов, а также включение прерываний и DMA для передачи по UART.

Поскольку в данной системе стабилизации скорости полета интерфейс UART используется для обмена данными между МК и WiFi модулем, настройки интерфейса производятся исходя из возможностей WiFi модуля. Соответственно, исходя из технических характеристик ESP8266-12E, были сконфигурированы следующие основные настройки для интерфейса UART:

1. Скорость соединения – 115200 бод;
2. Минимальная длина посылки – 8 бит;
3. Количество стоп битов – 1.

Порты микроконтроллера, отвечающие за передачу по линиям UART определены производителем микроконтроллера: PA9 для линии передачи и PA10 для линии приема.

В приложениях А3 и А4 представлены функции инициализации интерфейса UART, в которых производится настройка скорости соединения, минимальной длины посылки, количества стоп битов, а также конфигурация портов для линий приема и передачи, а также включение DMA и прерываний для этих линий.

2.3.1.3 Инициализация таймера

Поскольку управление сервоприводами и тягой мотора осуществляется при помощи ШИМ сигнала, необходимо использование таймера. Микроконтроллер STM32F103C8T6 имеет несколько различных видов таймеров, однако для реализации поставленной задачи в данной работе достаточно использования таймера общего назначения TIM2. Такой таймер способен генерировать ШИМ сигнал с заданными параметрами. Основным аспектом инициализации таймера является его тактирование, поскольку это прямо влияет на ширину импульса, выдаваемую при генерации ШИМ сигналов. Тактирование таймера производится от основной линии тактирования микроконтроллера, однако имеет свой настраиваемый предделитель частоты.

Из технических характеристик сервоприводов SG90 и регулятора оборотов ESC-30A следует, что данные устройства воспринимают ШИМ сигнал с шириной импульса от 1мс до 2мс и периодом в 20мс. Такому периоду соответствует частота 50Гц. Соответственно, при инициализации таймера необходимо настроить его тактирование на требуемый период.

В данном проекте тактирование микроконтроллера производится от внутреннего генератора частот, частота которого равна 8МГц. Для того, чтобы получить период ШИМ сигнала 20мс, необходимо задать значения для предделителя и периода. В данном случае, параметр периода выступает как своеобразный делитель, следовательно, задавая эти значения, необходимо получить частоту 50Гц. Для используемого таймера значение предделителя равно 8, а периода – 20000. С данными значениями получается тактирование таймера на частоте в 50Гц, из изначальных 8МГц.

Также, поскольку используется четыре устройства, управляемых посредством ШИМ сигнала, необходимо включить соответствующее количество каналов генерации ШИМ.

В приложении А5 представлена функция инициализации таймера, в которой настраивается его тактирование, а также включаются каналы генерации ШИМ сигналов. Параметры Prescaler (предделитель) и Period (период), равны 7 и 19999 соответственно, поскольку отсчет для данных параметров производится с 0.

В приложении А6 представлена функция настройки портов микроконтроллера, в которой определяются порты соответствующим каналам генерации ШИМ. Для МК STM32F103C8T6 для таймера TIM2 портами для генерации ШИМ сигнала являются PA0, PA1, PA2, PA3 соответственно.

2.3.1.4 Инициализация DMA

DMA (Direct Memory Access) – контроллер прямого доступа к памяти, который позволяет освободить процессор от операций с памятью.

В МК STM32F103C8T6 имеется два контроллера DMA, которые имеют 7 и 5 каналов соответственно. Каждый канал соответствует некоторому набору линий передачи данных для различных интерфейсов, которые могут быть использованы при работе с микроконтроллером.

В данном проекте DMA использован для увеличения быстродействия алгоритма стабилизации путем освобождения процессора от операций с большим потоком данных. Основным потоком большого количества данных является интерфейс UART, поскольку через него проходят как команды, так и данные, передаваемые от МК к WiFi модулю и наоборот.

Получение данных микроконтроллером от WiFi модуля происходит при помощи прерывания. Удобство такого метода заключается в том, что во время возникновения прерывания на обработку поступает сообщение целиком, а не один символ, что упрощает процесс нахождения необходимой информации, содержащейся в принятом пакете.

Настройка DMA для интерфейса UART производится при инициализации линий передачи и приема UART, в самой же функции запуска DMA производится только включение каналов DMA для работы с периферией. В данном случае линии передачи соответствует четвертый канал DMA, а линии приема – пятый. Также, для канала по приему данных использован циклический буфер, что позволяет избежать его переполнение.

2.3.1.5 Инициализация портов ввода/вывода

В инициализацию портов ввода/вывода входит настройка тактирования портов, определение их функций, назначение альтернативных функций, а также настройка их скорости работы. Также производится конфигурация портов для используемых интерфейсов. В данном проекте производится инициализация линий передачи данных для интерфейсов I2C и UART, включение портов для каналов таймера, а также подключение светодиода, размещенного на плате. Он управляется посредством порта PC13. Данный светодиод служит индикатором – он моргает во время выполнения программного кода и не моргает во время инициализации. Также, для оптимизации энергопотребления, все неиспользуемые порты сконфигурированы как аналоговые (рекомендация производителя МК).

В приложении А7 представлена функция, которая производит включение используемых портов с заданными настройками, описанными в коде инициализации соответствующей периферии.

2.3.1.6 Инициализация датчиков давления и температуры

Инициализация датчика давления и температуры BMP280 заключается в применении заданных настроек и чтении калибровочных коэффициентов: при выпуске изделия на производстве проводится калибровка датчика, в следствие

которой вычисляются калибровочные коэффициенты, которые записываются в память датчика. Соответственно, необходимо считать данные коэффициенты для точного измерения давления и температуры. Калибровочные коэффициенты считываются с устройства путем обращения к необходимому регистру по его адресу.

Чтение регистров и запись в них производится с помощью функций, которые содержит библиотека HAL.

Функция записи в регистр датчика BMP280 представлена в приложении А8.

Используется функция записи:

HAL_I2C_Mem_Write(&hi2c1, tx_buff, addr, 1, &value, 1, 10000);

В которой параметр &hi2c1- указатель на используемый интерфейс I2C, tx_buff – адрес периферийного устройства, addr – адрес регистра, в который производится запись, &value – указатель на значение, которое необходимо записать в регистр, 10000 – таймаут (время, в течение которого ожидается реакция от периферийного устройства), параметры, которые следуют за адресом регистра и значением для записи являются размерами в байтах этих значений соответственно.

Аналогичным образом работают функции чтения регистров.

Во время чтения калибровочных коэффициентов производится их запись в структуру, хранящую их значения. В последствии, эти коэффициенты будут использованы при вычислении давления и температуры.

В приложении А9 представлена функция чтения калибровочных коэффициентов.

Помимо чтения калибровочных коэффициентов, во время инициализации датчика BMP280, происходит настройка разрешения датчика, а также включение/отключение встроенного фильтра.

Согласно документации к датчику BMP280, выходное значение получается по формуле 2.7:

$$\text{data\_filtered} = \frac{\text{data\_filtered\_old} * (\text{filter\_coefficient} - 1) - \text{data\_ADC}}{\text{filter\_coefficient}} \quad (2.7)$$

где:

data_filtered – конечное значение текущего измерения;

data_filtered_old – значение предыдущего измерения;

filter_coefficient – коэффициент фильтра, зависящий от установленного разрешения датчика;

data_ADC – текущее измеренное значение.

В приложении А10 представлена функция инициализации датчика BMP280.

Поскольку в данной системе используется два таких датчика, при вызове функции в общей программе изменяется структура, хранящая адрес нужного устройства, соответственно, создается две структуры, каждая из которых хранит калибровочные коэффициенты датчика, адрес которого она хранит в себе.

2.3.1.7 Инициализация модуля MPU9250

Инициализация модуля MPU9250 заключается в конфигурации акселерометра, гироскопа, а также магнитометра. Также, перед чтением данных с датчика, необходимо произвести калибровку модуля. Калибровка производится с учетом поправочных коэффициентов и заданного разрешения датчиков. Чтение коэффициентов происходит так же, как и с датчиком BMP280.

Магнитометр в данном модуле является отдельным устройством, которое находится внутри кристалла MPU9250. Обращение к магнитометру происходит косвенно. Модуль, включающий в себя акселерометр и гироскоп, вступает в качестве ведущего и обменивается информацией с магнитометром. Несмотря на данную особенность, принцип обмена данными между МК и модулем MPU9250 остается неизменным.

В приложении А11 представлен полный код, реализующий функционал модуля MPU9250.

2.3.1.8 Инициализация WiFi модуля ESP8266-12E

Обмен данными, а также управление WiFi модулем ESP8266-12E производится с помощью AT команд, отправляемых по UART. AT команды описаны в документации к модулю и используются в прошивках, которые загружаются при производстве модулей на заводе. Между отправкой команд необходимо выдержать время минимум в 20мс.

Поскольку для выполнения поставленной задачи необходимо запустить TCP/IP сервер на модуле, необходимо применить ряд AT команд, реализующих данное соединение. Также была отключена функция эхо, которая дублирует отправленные сообщения на модуль обратно, для уменьшения объемов передаваемой информации между модулем и микроконтроллером, что способствует ускорению работы алгоритма.

Точка доступа всегда будет создаваться с IP адресом 192.168.1.4, однако порт соединения задается вручную с помощью команды «AT+CIPSERVER=1,88», в которой 1 – команда для создания точки доступа, 88 – номер порта. Согласно документации, каждая команда должна заканчиваться символами \r\n, для их правильного распознавания модулем.

В приложении А12 представлена функция инициализации модуля WiFi, которая состоит в создании TCP/IP точки доступа, к которой в дальнейшем станет возможно подключение с другого устройства в целях наблюдения результатов работы программы.

2.3.1.9 Инициализация ОСРВ FreeRTOS

В процессе инициализации ОСРВ задаются основные параметры системы, такие как:

1. Выбор типа многозадачности;
2. Определение системного кванта;
3. Выбор типа использования оперативной памяти;
4. Выбор схемы распределения оперативной памяти;
5. Определение количества возможных приоритетов;
6. Определение размера кучи;
7. Включение возможности использования мьютексов, семафоров и очередей;
8. Включение возможности использования системных функций.

Также во время инициализации системы происходит создание очередей, задач, семафоров и мьютексов.

Для рассматриваемой в данной работе системы стабилизации был выбран кооперативный тип многозадачности, системный квант, который используется как мера интервалов времени, определен в 1мс, тип использования оперативной памяти – динамический, а схема распределения – 4, которая автоматически выгружает из памяти неиспользуемые данные. Количество возможных приоритетов осталось в стандартном значении, равном семи. В данном проекте это не имеет значения, поскольку все реализованные задачи имеют одинаковый

приоритет в силу выбранного типа многозадачности. Размер кучи (динамически распределяемая память), выбрана равной 15360 битам, что является значением с запасом и используется для нужд используемых задач и очередей.

Поскольку функция инициализации ОСРВ вызывается в основной программе после инициализации всей периферии, удобно объявлять все переменные, использованные в реализации алгоритма, в этом же блоке. Также вызов инициализирующих датчики функций вызываются в процессе инициализации ОСРВ.

Перед описанием функции инициализации ОСРВ объявляются задачи и очереди, которые будут функционировать в системе. В самой функции инициализации происходит запуск каналов генерации ШИМ сигнала, включение прерываний по приему UART, вызов функций инициализации датчиков, а также создание задач и очередей.

В приложении А13 представлена функция инициализации ОСРВ, а также объявление задач и очередей, предшествующее её описанию.

2.3.1.10 Инициализация основной программы

В коде основной программы последовательно вызываются вышеописанные функции, в результате чего применяются все необходимые настройки, производится калибровка датчиков и запускаются необходимые каналы генерации ШИМ сигналов и DMA. После выполнения всех этих функций, производится запуск планировщика, который бесконечно распределяет процессорное время между созданными задачами.

2.3.2 Алгоритм получения данных с датчиков

2.3.2.1 Алгоритм получения данных с датчика давления и температуры BMP280

Получение давления и температуры с датчиков BMP280 производится с помощью чтения определенных регистров и компенсацией этих данных путем пересчета с учетом калибровочных коэффициентов, записанных в память МК во время инициализации датчиков.

В целом, процесс определения давления и температуры проходит три этапа:

1. Получение целочисленных данных путем считывания регистров;
2. Компенсация полученных данных с помощью калибровочных коэффициентов;
3. Перевод компенсированных целочисленных значений в числа с плавающей точкой.

Процесс получения давления и температуры построен таким образом, что одна функция внутри себя вызывает другую, поэтому, для снятия данных с датчиков необходимо вызвать только конечную функцию.

В приложении А14 представлены функции, реализующие полный алгоритм получения давления и температуры с датчиков BMP280.

2.3.2.2 Алгоритм получения данных с модуля MPU9250

Получение данных с датчика угловой скорости, ускорения и магнитного поля Земли производится аналогично тому, как это происходит с датчиком BMP280, однако из за конструктивных особенностей модуля не предоставляется возможным чтение регистров данных магнитометра напрямую: адресом устройства для шины I2C указывается не адрес модуля MPU9250, в составе которого акселерометр и гироскоп, а адрес магнитометра.

После прочтения регистров данных, полученные значения домножаются на коэффициенты, зависящие от установленного разрешения каждого датчика, после чего вычитаются значения, вычисленные после калибровки датчиков.

Когда данные по всем осям с каждого датчика получены, производятся вычисления углов тангажа, крена и курса. Эти расчеты производятся сразу для экономии памяти микроконтроллера, поскольку вычисление этих углов использует большое количество переменных, текущее значение которых можно сразу взять в теле этой функции.

2.3.3 Алгоритм обмена данными через WiFi модуль

Передача данных на устройство, подключенное к TCP/IP соединению, созданному в процессе инициализации системы, производится путем отправки AT команд WiFi модулю через UART. Необходимо использование команды AT+CIPSEND=X,Y, где X – номер подключенного устройства (0 – 3), Y – количество байт, которые будут переданы. После отправки этой команды следует сообщение, размер которого не превышает заданного значения в команде. Между командой и отправкой сообщения необходимо так же выдержать время, минимум 20мс, как и в процессе инициализации модуля.

Прием данных с модуля характерен тем, что от модуля микроконтроллеру приходит сообщение вида «+IPD, A, B : C», где A – номер подключенного устройства, с которого пришло сообщение, B – размер сообщения, C – текст сообщения.

Алгоритм получения заданной скорости описан далее в подпункте 2.3.5.

2.3.4 Алгоритм выработки ШИМ сигналов

Для управления углами положения сервоприводов, а также для регулирования оборотов мотора используется ШИМ сигнал. В процессе

инициализации был сконфигурирован таймер, каналы которого генерируют ШИМ, а также был произведен запуск каналов в работу.

Для того, чтобы задать ширину импульса ШИМ сигнала, от которого зависит положение сервоприводов, необходимо в регистр CCR занести значение ширины импульса. Значения предделителя выбраны такими, чтобы одна единица, записанная в этот регистр, была эквивалентна 1мкс. Следовательно, для того чтобы задать ширину импульса в 1мс, необходимо записать 1000 в регистр CCR созданного таймера TIM2. Поскольку используется четыре канала генерации, необходимо записывать значения ширины импульса в регистры CCR1, CCR2, CCR3, CCR4 соответственно.

Передний сервопривод соответствует первому каналу таймера, левый – второму, правый – третьему, мотор – четвертому.

Была создана функция, реализующая управление сервоприводами и мотором. Данная функция принимает в качестве аргументов номер канала и значение угла от 0 до 180 (градусов) для сервоприводов и от 0 до 100 (%) тяги для мотора.

В документации к сервоприводам SG90 и регулятору оборотов ESC-30A содержится информация о диапазонах ширины импульса, используемой для управления устройством. Функция управления, принимая на вход величину угла, либо тяги, пересчитывает её в значение ширины импульса с учетом допустимых диапазонов ширины импульса. После данного пересчета полученное значение записывается в соответствующий нужному каналу регистр, в результате чего вырабатывается управляющий ШИМ сигнал, получив который, устройство среагирует на это воздействие.

В приложении А15 представлен код функции генерации ШИМ сигналов.

2.3.5 Алгоритм выполнения прерывания по приему UART

В процессе инициализации ОСРВ FreeRTOS была вызвана стандартная функция, разрешающая прерывание. Также эта функция позволяет выбрать тип прерывания, в данном случае – прерывание по IDLE.

Данный тип прерывания характерен тем, что флаг прерывания USART_SR_IDLE фиксируется в регистре SR, когда по линии приема UART был получен первый бит. Поскольку в данном проекте для передачи по UART используется DMA, после разрешения прерывания была вызвана функция приема данных. Данная функция вызывается только 1 раз в том случае, если буфер данных циклический. Каждый раз, когда приходит сообщение, возникает прерывание, инициированное интерфейсом UART, в котором происходит проверка регистра SR на совпадение с флагом USART_SR_IDLE, и в случае их совпадения происходят следующие действия:

1. Отключение канала DMA;
2. Переназначение адреса буфера приема данных;
3. Переназначение длины буфера приема данных;
4. Проверка на условие – если полученное сообщение содержит знак «+», происходит копирование этого сообщения в буфер для дальнейшей отправки сообщения в задачу по поиску заданной скорости посредством очереди;
5. Очистка буфера приема данных;
6. Включение канала DMA.

Данный алгоритм позволяет очень быстро передать нужное сообщение в задачу, поскольку производится поиск лишь одного символа, который однозначно указывает на то, что данные приняты с компьютера. Также применение передачи данных с помощью очереди позволяет избежать возможных ошибок. Примером такой ошибки может послужить ситуация, когда задача по поиску заданной скорости в строке, в ходе выполнения своего

алгоритма, обратится к буферу, который в данный момент времени изменился из прерывания.

Функция обработки прерывания представлена в приложении А16

2.3.6 Реализация алгоритмов задач

2.3.6.1 Реализация алгоритма работы задачи по приему данных от датчиков

Прием данных от датчиков производится по линейному алгоритму, в котором происходит последовательный опрос датчиков, после которого следует отправка полученных значений в очередь.

Для успешного запуска бесколлекторного мотора необходимо перед его работой произвести следующие действия:

1. Установить значение тяги мотора в 100%;
2. Выдержать временной промежуток в 2с;
3. Установить значение тяги мотора в 0%;
4. Выдержать временной промежуток в 6с.

В целях повышения стабильности работы алгоритма всей системы, инициализация WiFi модуля и алгоритм, предшествующий запуску мотора, производятся при первом выполнении данной задачи. Такой метод позволяет избежать возможных ошибок, поскольку необходима приостановка системы на определенные интервалы времени, а при инициализации ОСРВ нежелательно применение задержек.

Для того, чтобы процесс инициализации WiFi модуля и мотора произошел один раз, используется флаг INIT_FLAG. Изначально данный флаг имеет значение «0». После выполнения необходимых действий, значение флага задается «1», и больше эта часть кода не выполняется.

Также, в ходе выполнения алгоритма приема данных от датчков предусмотрен вывод информации ошибок, если такие возникнут в процессе приема данных с датчиков.

В приложении А17 представлен код задачи по приему данных с датчиков.

2.3.6.2 Реализация алгоритма работы задачи функциональных вычислений

Поскольку алгоритм функциональных вычислений состоит из ряда отдельных задач, в данном пункте рассматривается только процесс вычисления необходимых параметров полета и отправка их в очередь. Реализация задачи по поиску значения заданной скорости в принятом сообщении с компьютера рассматривается в пункте 2.3.6.4.

Во время первого выполнения задачи происходит запись значения статического давления в переменную, которая используется в вычислениях высоты в качестве давления на нулевой высоте полета. Этот процесс выполняется только один раз, при запуске системы. Флаг INIT_PRESSURE_FLAG изначально имеет значение «0», однако после записи значения давления на начальной высоте полета, значение флага изменяется на «1».

При получении управления от планировщика, задача функциональных вычислений просматривает очередь, данные в которую отправляются в задаче поиска заданной скорости, на наличие новых данных. В случае, если значение заданной скорости изменилось, её значение перезаписывается в соответствующей переменной. Далее происходит прием данных из очереди, которая заполняется задачей по приему данных с датчиков. Принимается структура данных, которая содержит:

1. Статическое давление;
2. Полное давление;
3. Температуру окружающего воздуха;
4. Угол тангажа;
5. Угол крена;
6. Угол курса.

После завершения приема структуры данных производятся функциональные вычисления текущей высоты полета и текущей скорости полета при помощи формул, приведенных в пункте 2.2.2.2.

Когда вычисления выполнены. Выполняется алгоритм висения или стабилизации скорости полета в зависимости от значений, полученных в ходе выполнения программы. Алгоритм, которому следует программа, описан в пункте 2.2.2.2.

Значения углов положения сервоприводов и тяги мотора, полученные в ходе выполнения алгоритма, записываются в структуру, которая передается в очередь для отправки в задачу по выработке управляющих команд.

Также в данной задаче реализована передача значений высоты, текущей скорости, заданной скорости, а также углов тангажа, крена и курса на компьютер через WiFi модуль.

В приложении А18 представлен код задачи функциональных вычислений.

2.3.6.3 Реализация алгоритма выработки управляющих команд

При получении управления от планировщика, данная задача забирает структуру данных из очереди, после чего с помощью функции по выработке ШИМ сигналов задает значения углов положения сервоприводов, а также тяги бесколлекторного мотора. Алгоритм работы этой функции описан в пункте 2.3.4.

В приложении А19 представлен код задачи выработки управляющих команд.

2.3.6.4 Реализация алгоритма поиска заданной скорости в принятом сообщении

При получении управления от планировщика, данная задача просматривает очередь, заполняемую из прерывания, на наличие новых данных. В случае, когда новых данных нет, задача передает управление планировщику. Если же новые данные получены, производится проверка полученной строки на наличие символов +IPD, что однозначно указывает на то, что данные были получены WiFi модулем с компьютера. Далее, с помощью функции sscanf из библиотеки <string.h>, производится поиск численных символов, применяя шаблон: «+IPD,%d,%d:%d», в котором принимается последовательно три значения: номер подключенного устройства к ICP/IP соединению, с которого пришло сообщение, размер полученного сообщения, а также само сообщение. Поскольку заведомо известно, что сообщение будет содержать численное значение заданной скорости полета, данная функция сразу ожидает целочисленные данные. Полученное в результате работы данной функции значение заданной скорости полета передается в задачу функциональных вычислений посредством очереди, после чего передается управление планировщику.

В приложении А20 представлен код задачи поиска заданной скорости в сообщении.

2.3.7 Загрузка прошивки в микроконтроллер

Загрузка прошивки в микроконтроллер, а также отладка разработанного программного обеспечения проводится при помощи SWD интерфейса. SWD интерфейс для своей работы требует подключения двух сигнальных проводов: SWDIO и SWCLK.

Для того, чтобы загрузить собранный проект в микроконтроллер, необходим адаптер ST-LINK/V2. Для его правильной работы при создании проекта в CubeMX подключаются линии SWDIO и SWCLK, а в Keil uVision 5 в настройках проекта необходимо включить отладку по SWD интерфейсу.

Адаптер ST-LINK/V2 одной стороной подключается к компьютеру, с которого будет производиться загрузка проекта, другой подключается к микроконтроллеру, причем если используется внешнее питание МК, необходимо подключить только SWDIO, SWCLK, GND.

На рис. 2.24 изображен адаптер ST-LINK/V2.

Рис. 2.24 адаптер ST-LINK/V2

2.4 Разработка алгоритмов и ПО визуализации результатов работы разработанных средств

Разработка алгоритма и программного обеспечения визуализации результатов работы разработанных средств ставит задачу создания приложения, запуск которого производится с компьютера.

Данное приложение должно выполнять следующие функции:

1. Установка соединения с TCP/IP сервером, который был создан WiFi модулем;
2. Приём данных с микроконтроллера посредством использования TCP/IP соединения;
3. Визуализация полученных данных;
4. Отправка значения заданной скорости, заданного пользователем.
5. Отключение от сервера.

Разработка программного обеспечения, реализующего данный алгоритм, производится в среде разработки Visual Studio 2017.

Интерфейс приложения Visual Studio 2017 представлен на рис. 2.25.

Слева в окне среды расположены элементы управления, которые можно добавить на форму, разработка которой производится в центральном поле. Справа находится дерево проекта, позволяющее быстро выбрать необходимый компонент программы для просмотра или редактирования. Также, в правом нижнем углу расположено окно настроек элементов управления, в котором можно задать необходимые параметры выбранного элемента.

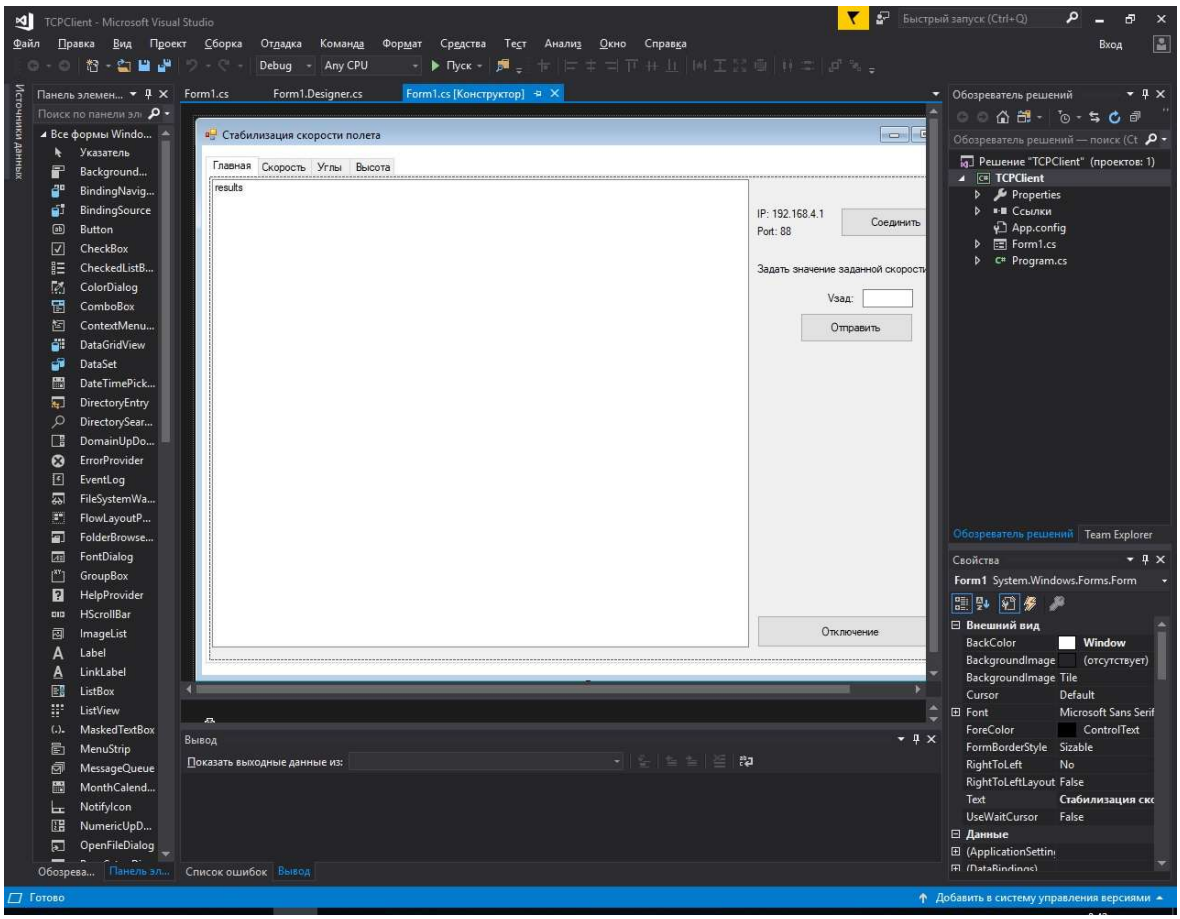


Рис. 2.25

2.4.1 Алгоритм визуализации результатов работы разработанных средств

Алгоритм визуализации заключается в реализации функционала, определенного в постановке задачи. Так как данные, принимаемые с МК, имеют разные размерности, а также диапазоны возможных значений, визуализация полученной информации разделена на три части:

1. Построение графиков текущей и заданной скоростей;
2. Построение графиков изменения углов ориентации БПЛА;
3. Построение графика изменения высоты полета БПЛА.

Поскольку необходимо наблюдать изменение параметров полета во времени, процессы построения графиков выполняются параллельно друг другу.

При запуске приложения производится инициализация приложения, в ходе которой выполняется первоначальная настройка всех элементов управления.

При условии, что компьютер подключен к точке доступа WiFi, созданной с помощью ESP8266-12E под управлением микроконтроллера, после нажатия кнопки «Соединить», начинается выполнение процесса приема данных с макета и параллельное построение графиков изменения параметров полета во времени. Для того, чтобы наблюдать построенные графики, необходимо выбрать соответствующую вкладку из доступных.

Для ввода значения заданной скорости полета используется специальное окно, в котором указывается скорость, после чего происходит отправка сообщения микроконтроллеру при нажатии кнопки «Отправить».

При нажатии кнопки «Отключение», произведется разрыв соединения и приложение будет автоматически закрыто.

2.4.2 Реализация алгоритма визуализации результатов работы разработанных средств

Приложение создано в среде разработки Microsoft Visual Studio 2017 при помощи платформы .NET framework. Реализация алгоритма визуализации результатов работы разработанных средств заключается в создании элементов управления, размещенных на форме, а также написании функций, реализующих данный алгоритм.

В приложении А21 представлен полный код приложения по визуализации результатов.

2.4.2.1 Формирование элементов управления

Для реализации возможности выбора графика, а также главного меню, в котором расположены кнопки для установки TCP/IP соединения и отправки заданной скорости, на форму приложения добавлен элемент tabControl, реализующий систему вкладок. Всего таких вкладки 4: «Главная», «Скорость», «Углы», «Высота». Данный элемент управления позволит пользователю быстро переходить с одного графика на другой.

На вкладке «Главная» расположены три кнопки: «Соединить», «Отправить» и «Отключение», а также область для отображения полученных данных в текстовом виде. Также на данной вкладке расположены надписи, позволяющие пользователю видеть, к какому IP адресу и порту будет производиться подключение. Также имеются надписи, показывающие пользователю, куда необходимо ввести значение заданной скорости полета.

На рис. 2.26 представлено изображение главной вкладки с расположенными на ней элементами управления.

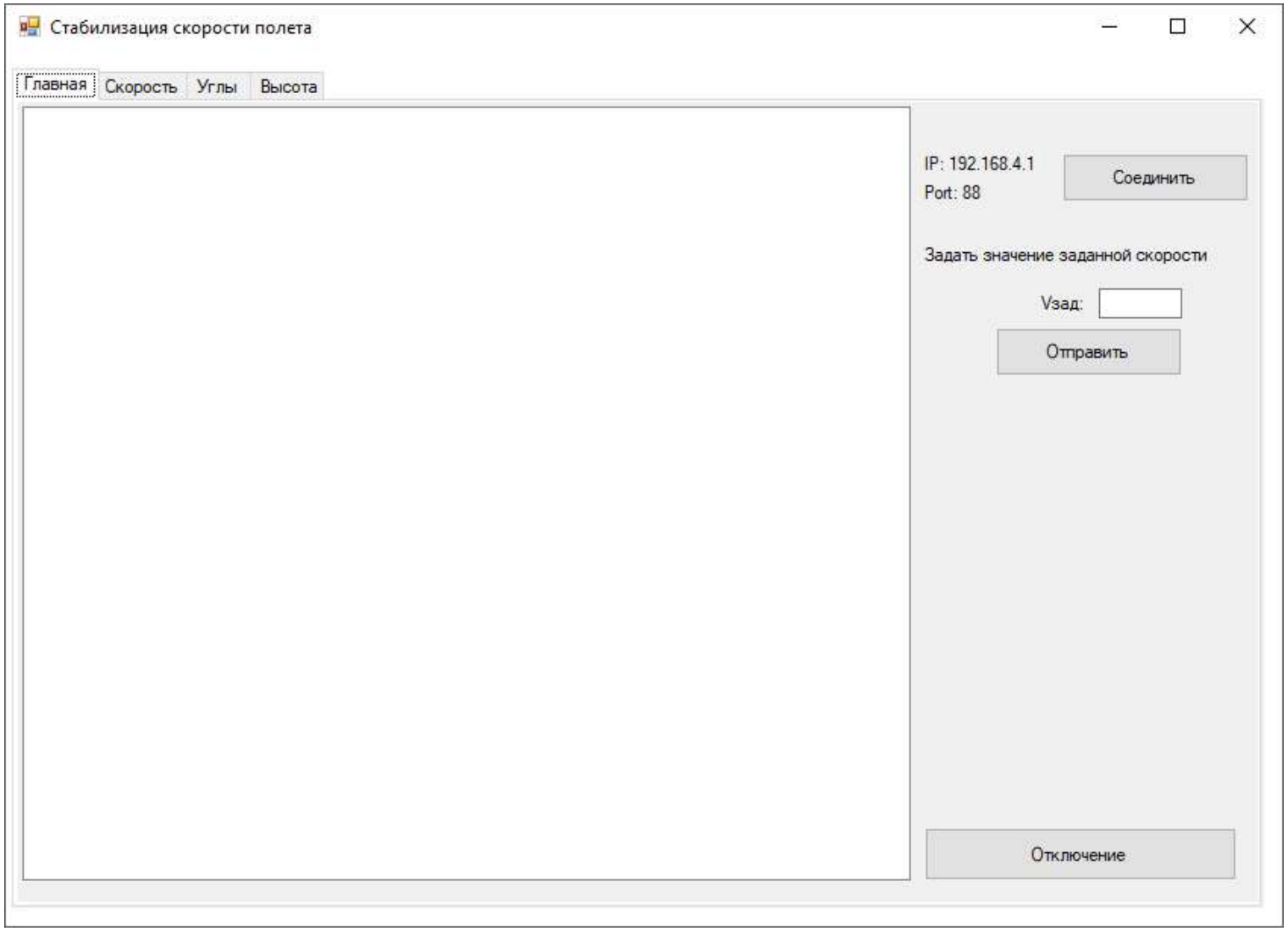


Рис. 2.26 Изображение главной вкладки

2.4.2.2 Формирование элементов отображения информации

На вкладках «Скорость», «Углы» и «Высота» расположен элемент chart, позволяющий отобразить информацию в виде графиков.

2.4.2.3 Реализация функционала приложения

2.4.2.3.1 Инициализация приложения

При запуске приложения выполняется его инициализация, которая включает настройку элементов управления и отображения информации. Также, для более удобного просмотра результатов пользователем, в программе реализована возможность масштабирования графиков, а также просмотр предыдущих показаний путем передвижения ползунка, расположенного под графиком. Реализация этого функционала производится путем настройки элемента chart в ходе инициализации приложения.

2.4.2.3.2 Реализация функции установки TCP/IP соединения

Поскольку WiFi модуль ESP8266-12E всегда будет иметь IP адрес 192.168.4.1, а порт задается в процессе создания точки доступа с МК, в данном приложении не реализована возможность самостоятельного выбора IP адреса и порта соединения в силу нецелесообразности.

TCP/IP соединение устанавливается по нажатию на кнопку «Соединить». Функция реализует подключение к серверу с помощью стандартных библиотек, а также выводит сообщения об ошибке, если подключение не удалось. В случае, если соединение установлено успешно, запускается прием данных.

2.4.2.3.3 Реализация функции приема и отображения данных

Функция реализует прием данных с помощью TCP/IP соединения. Принятая информация поступает в виде символьных данных, и её необходимо конвертировать в численный тип. Конвертация производится с помощью метода Parse.

Для того, чтобы графики отображали актуальную информацию одновременно, в данной функции производится построение этих графиков после каждого приема сообщения.

2.4.2.3.4 Реализация функции отправки сообщения

Для отправки значения заданной скорости используется окно для ввода информации и кнопка, по нажатию на которую производится передача сообщения. Имеется ряд ограничений на функцию отправки: введенное значение должно быть целым числом, имеющим размерность м/с.

2.4.2.3.5 Реализация функции завершения соединения

Функция завершения соединения запускается при нажатии кнопки «Отключение», выполняющая завершение TCP/IP коммуникации и закрытие приложения.

2.5 Верификация разработанного ПО

Верификация программного обеспечения – это тестирование, целью которого является достижение гарантии того, что верифицируемый объект соответствует требованиям, реализован без непредусмотренных функций и удовлетворяет проектным спецификациям и стандартам. Процесс верификации включает в себя инспекции, тестирование кода, анализ результатов тестирования, формирование и анализ отчетов о проблемах.

Для верификации программного обеспечения алгоритма стабилизации скорости полета микро-БПЛА был выбран динамический метод верификации ПО – тестирование.

В процессе данного тестирования необходимо выяснить, следует ли код разработанной программы заданному алгоритму и выполняет ли он поставленную задачу. Также во время тестирования происходит анализ кода на возникновение ошибок.

Для реализации тестирования разработанного программного обеспечения необходимо создание тестовой программы. Поскольку необходимо исследовать алгоритм стабилизации скорости, создание тестовой программы производится на основе кода, реализующего данный алгоритм. Так как данные передаются посредством сигнала WiFi, мониторинг передаваемой информации не обязательно осуществлять с помощью специализированной программы, а достаточно использовать терминал TCP с любого устройства. В ходе данного тестирования было использовано устройство под управлением ОС Android.

2.5.1 Тестовая программа

Поскольку алгоритм стабилизации скорости построен на ОСРВ с кооперативным типом многозадачности, необходимо выяснить, в каком порядке производится передача управления задачам от планировщика. Так как все задачи создавались с одинаковым приоритетом, планировщик должен передавать управление задачам в порядке их создания.

Список задач по порядку их создания:

1. Задача по приему данных от датчиков;
2. Задача функциональных вычислений;
3. Задача выработки управляющих команд;
4. Задача приема заданной скорости.

Для того, чтобы выяснить порядок выполнения этих задач, необходимо создать тестовую версию программы. В ней изменен код: из задачи функциональных вычислений вырезана передача данных на сервер, однако теперь в каждой задаче выдается информация о том, какая задача в данный момент запущена. Каждая задача в момент выполнения отправляет строку следующего содержания: «TASK N», где N – номер задачи.

После загрузки кода тестовой программы в микроконтроллер, ход выполнения алгоритма наблюдался с помощь TCP терминала.

Результат выполнения тестовой программы представлен на рис. 2.27, который отображает последовательность выполнения задач, реализованных в алгоритме стабилизации скорости полета.

Следующим этапом тестирования является процесс выявления программных ошибок, возникающих в ходе выполнения алгоритма.

Поскольку в коде задач изначально реализована выдача сообщений в случаях возникновения ошибок, необходимо скорректировать код в блоках обработки ошибок: добавлена переменная, служащая счетчиком ошибок. Счетчик ошибок инкрементируется каждый раз, когда будет возникать ошибка.

Ошибки, которые будут обрабатываться:

1. Ошибка инициализации датчиков;
2. Ошибка чтения данных с датчиков;
3. Ошибка отправки данных в очередь.

Проверка на наличие ошибок в ходе работы WiFi модуля нецелесообразна, поскольку в этом случае данные не будут передаваться на TCP сервер, откуда следует, что во время тестирования не представится возможности наблюдать какую-либо информацию.

Наблюдение за данной переменной производится в режиме отладки микроконтроллера в среде Keil uVision 5 при помощи SWD интерфейса. Время тестирования – 10 минут. Учитывая скорость выполнения программы, в частности, частоту считывания датчиков, данного времени достаточно для того, чтобы сделать вывод по проделанным наблюдениям.

Результат выполнения тестовой программы по выявлению количества ошибок представлен на рис. 2.28.

На рис.2.28 можно увидеть текущее значение счетчика ошибок, а также время выполнения программы в секундах.

Следующим этапом тестирования является проверка верного срабатывания прерывания, а также правильное распознавание введенного значения заданной скорости. Для этого в тестовой программе изменен код: теперь, вместо выдачи текста запущенными задачами, выдается только принятое значение заданной скорости полета. Заданная скорость вводится путем отправки значения через TCP терминал. В данной части тестирования производится ввод значений и наблюдение изменения переменной, отвечающей за хранение заданной скорости в программе.

Результат выполнения тестовой программы представлен на рис.2.29.

Полный код тестовой программы представлен в приложении А22

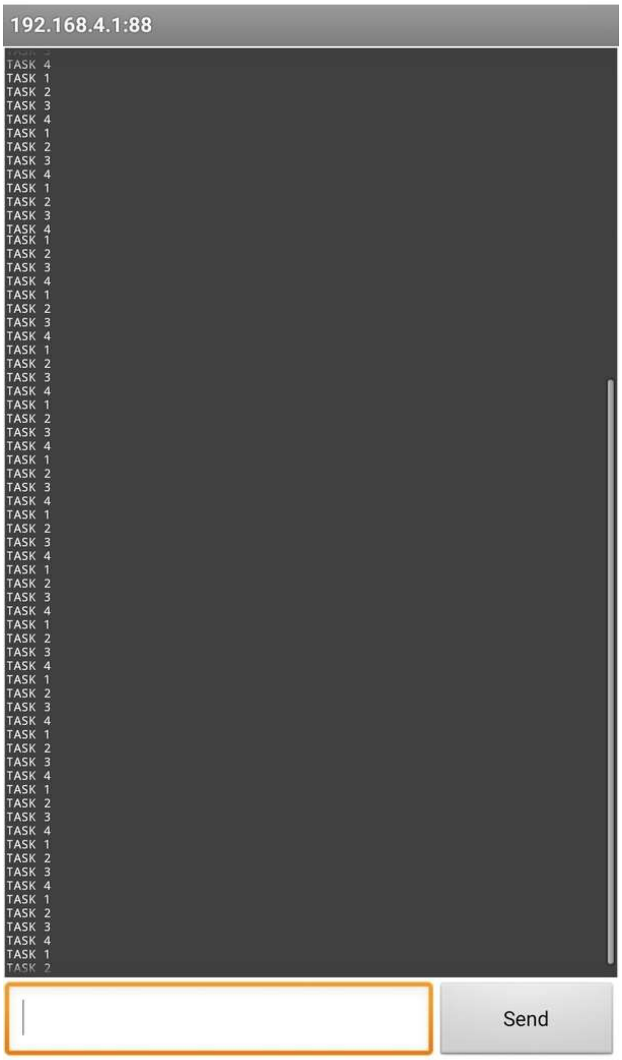


Рис. 2.27 Порядок выполнения алгоритма

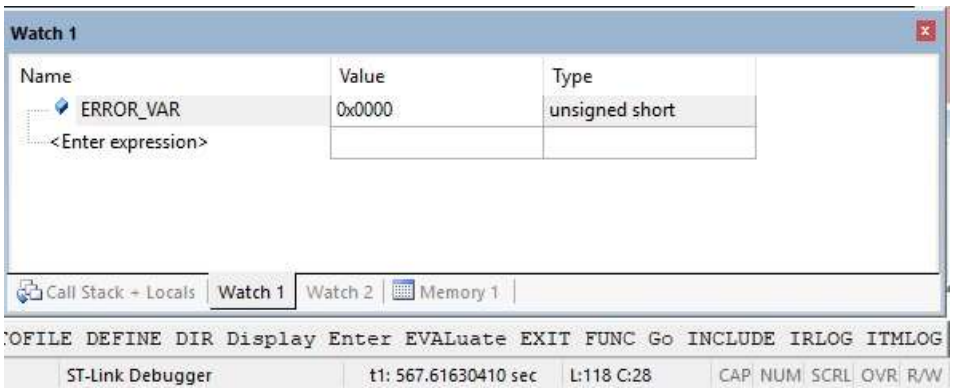


Рис. 2.28 Количество ошибок в ходе работы программы

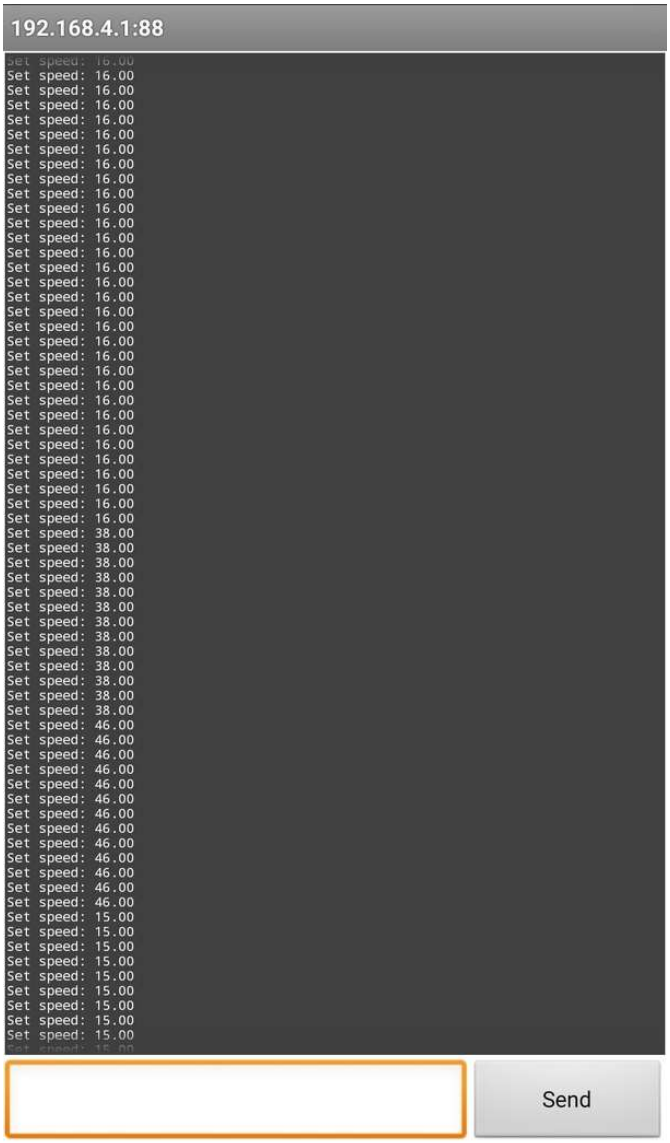


Рис. 2.29 Проверка отработки получения заданной скорости

2.5.2 Результаты выполнения разработанных средств

Также для верификации необходимо показать, удовлетворяет ли разработанный алгоритм техническому заданию, согласно которому должна производиться стабилизация скорости полета микро-БПЛА и визуализация высотно-скоростных параметров, вычисленных в процессе выполнения программы. Для этого необходимо произвести запуск созданного макета. Наблюдение за текущими значениями высоты, скорости полета, углов тангажа, крена и курса производится при помощи разработанного приложения под ОС Windows.

Поскольку алгоритм не выдает значений углов поворота сервоприводов и тяги мотора, результаты работы стабилизации скорости наблюдаются визуальным методом непосредственно на макете.

Также в ходе просмотра результатов плата управления, на которой расположены акселерометр, гироскоп и магнитометр, подвергалась различным воздействиям: легкой вибрации, поворотам в различных плоскостях и поднятие на высоту.

На рис. 2.30 представлено визуальное отображение изменения скорости в режиме стабилизации.

На рис. 2.31 представлено визуальное отображение изменения углов тангажа, крена и курса при поворотах платы управления.

На рис 2.32 представлено визуальное отображение изменения высоты при резком её изменении.

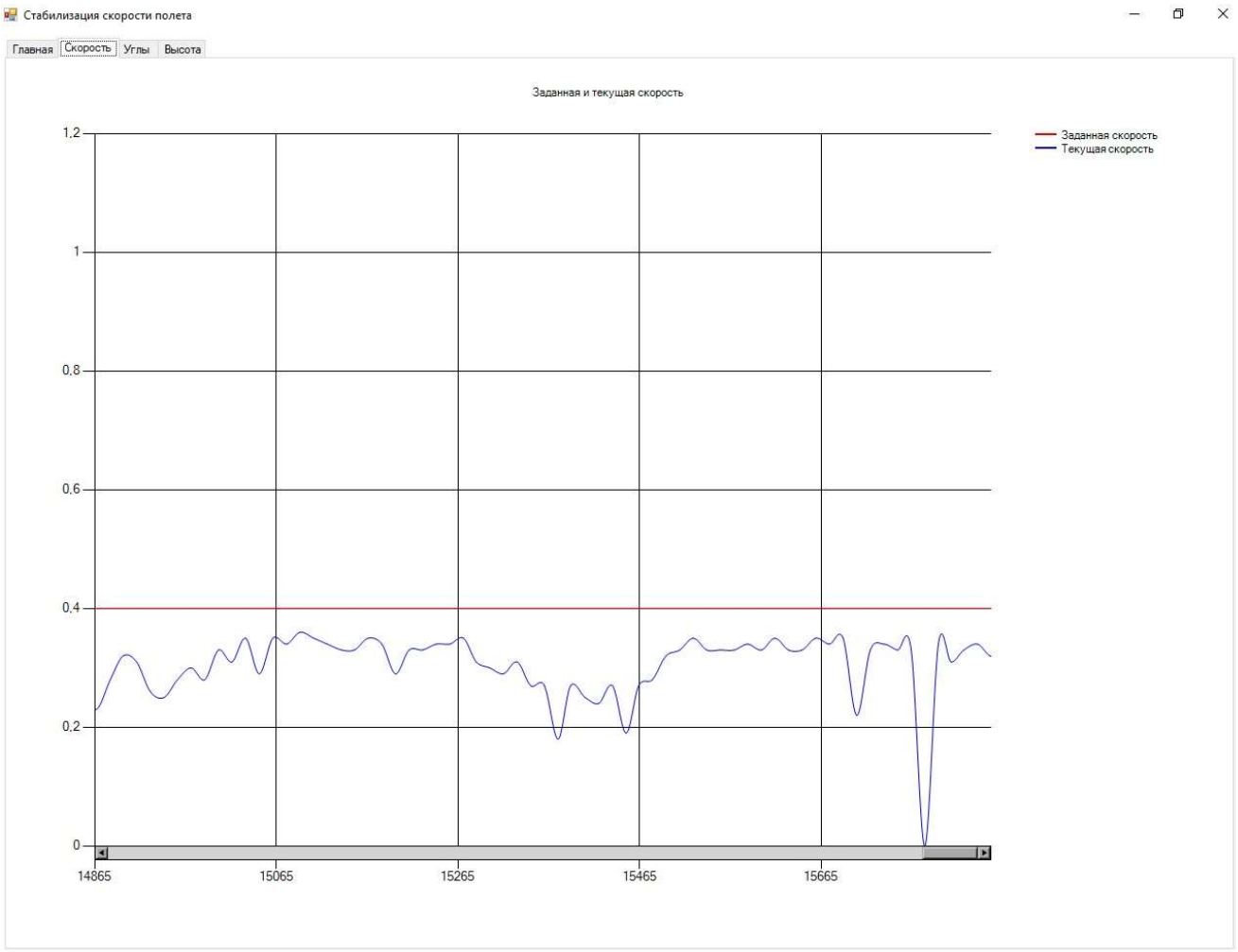


Рис. 2.30 График изменения скорости полета в режиме стабилизации

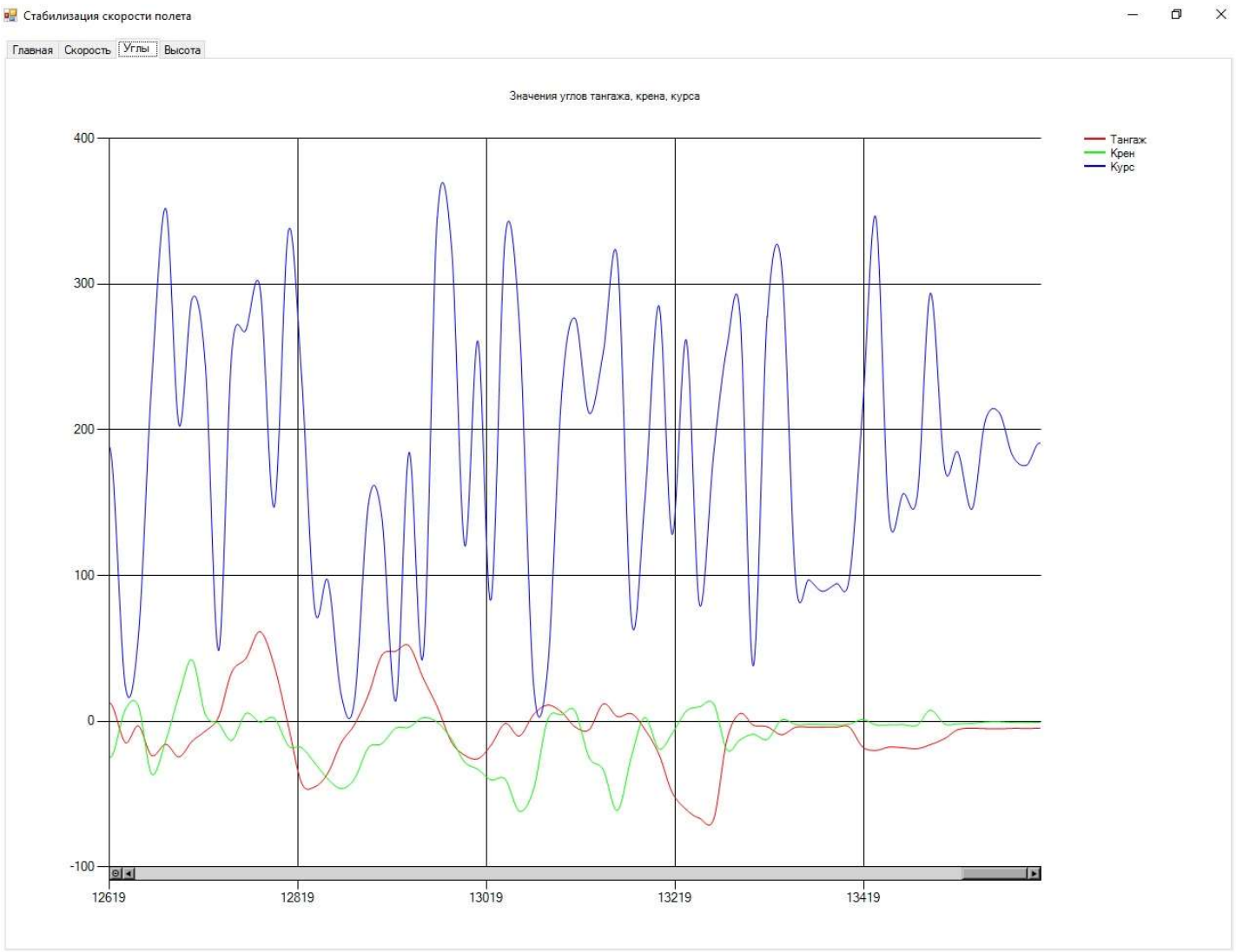


Рис. 2.31 График изменения углов тангажа, крена и курса

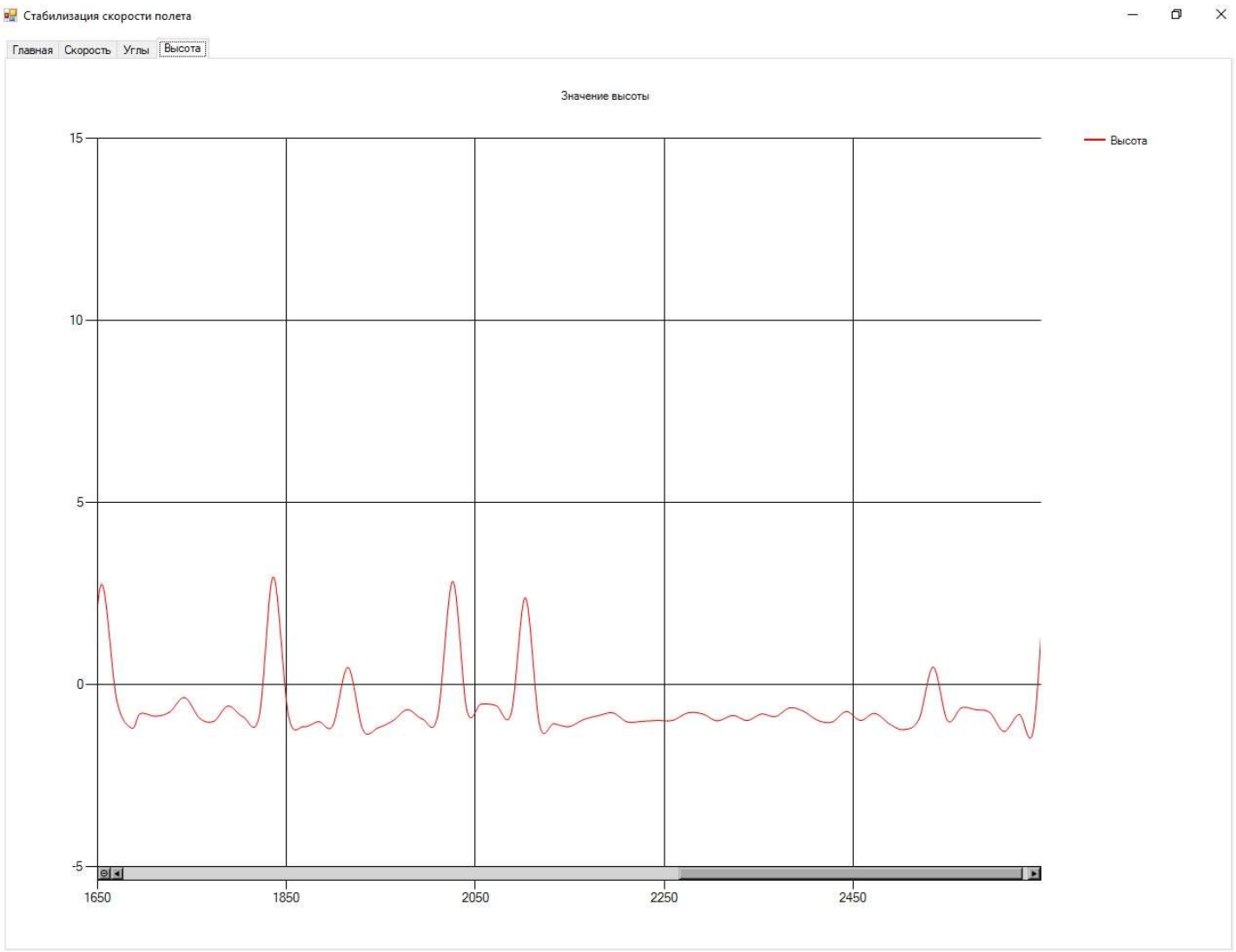


Рис. 2.32 График изменения высоты

2.5.3 Анализ результатов работы разработанных средств

Было произведено тестирование алгоритма стабилизации скорости полета микро-БПЛА, в ходе которого был проверен порядок выполнения задач, отслеживание ошибок, возникших в процессе работы алгоритма, а также проверка функционала по приему данных средствами беспроводной связи.

2.5.3.1 Анализ тестирования программного обеспечения

Первым этапом тестирования являлась проверка порядка выполнения задач. Рис.2.27 демонстрирует порядок, в котором задачи получают управление от планировщика. Данный порядок соответствует разрабатываемому алгоритму, в ходе работы нарушений не обнаружено.

Вторым этапом тестирование стало отслеживание количества ошибок во время выполнения алгоритма в течение 10 минут. Во время тестирования на этом этапе ошибок не возникло, что подтверждает рис.2.28, на котором изображен момент отслеживания значения счетчика ошибок.

На третьем этапе тестирования производилась проверка реакции системы на изменение значения заданной скорости посредством беспроводной связи. Рис.2.29 демонстрирует изменение значения заданной скорости полета, которое задавалось путем отправки данных через TCP терминал.

Обобщая результаты тестирования можно заключить, что алгоритм стабилизации скорости полета микро-БПЛА выполняется без ошибок, порядок выполнения задач соответствует разработанному алгоритму, а также работает заявленный функционал.

2.5.3.2 Анализ результатов работы разработанного алгоритма

После включения макета системы управления, было произведено подключение к точке доступа WiFi с ПК. Далее при помощи разработанного программного обеспечения было установлено TCP/IP соединение по заданному IP адресу и номеру порта. В ходе выполнения программ, были успешно приняты

данные от макета, а также построены графики необходимых параметров полета: заданной и текущей скоростей, высоты, углов тангажа, крена и курса.

Поскольку макет системы стабилизации скорости полета микро-БПЛА не оснащен полноценным автоматом-перекосом, а лишь его моделью, состоящей из трех сервоприводов и бесколлекторного мотора, не представляется возможности увидеть на графике стремление текущей скорости к заданной. Однако, алгоритм подразумевает два режима: режим стабилизации скорости и режим висения, который выполняется в случаях, когда значение заданной скорости равно нулю. В соответствии с этим можно наглядно видеть результаты работы системы по характеру изменения углов положения сервоприводов и частоты вращения мотора.

Анализируя графики, показывающие текущую скорость и заданную, можно утвердить: система верно реагирует на изменение заданной скорости пользователем, однако измерения текущей скорости весьма неточны. Причиной такой погрешности является низкое заданное разрешение показаний датчиков давления и встроенного фильтра. Выбраны такие параметры в целях увеличения быстродействия и исключения накопления ошибки. По этой же причине наблюдаются ошибки в измерениях высоты: значение текущей высоты периодически резко меняется, однако среднее значение остается близко нулю при условии неподвижности макета.

На рис.2.31 продемонстрированы графики изменения углов тангажа, крена и курса. Как видно по графикам, значения тангажа и крена весьма точны и отображают реальную картину изменения своих значений. Угол курса показывает неточное значение, поскольку магнитометр, установленный в модуле MPU9250 очень сильно подвержен шуму из окружающей среды, а так как запуск системы проводился в окружении электронных устройств, а также в местах с большим количеством модулей связи. Все эти устройства влияют на напряженность магнитного поля, что добавляет очень много шума в измерения магнитометра.

3. Экономическое обоснование разработки комплексного алгоритма для стабилизации скорости полета микро-БПЛА

3.1 Оценка целесообразности выполнения разработки на основе определения её технической прогрессивности

В данном разделе рассматривается экономическое обоснование разработки алгоритма для стабилизации скорости полета микро-БПЛА.

Для оценки научно-технической прогрессивности продукта необходимо использовать метод бальных оценок сравнения НТПр с аналогом.

Показатель научно-технической прогрессивности рассчитывается по формуле 3.1:

$$J_{\text{НТП}} = \frac{H_{\text{НТП Н}}}{H_{\text{НТП Б}}}, \tag{3.1}$$

где:

$J_{\text{НТП}}$ – индекс научно-технической прогрессивности НТПр;

$H_{\text{НТП Н}}$ – обобщённый количественный показатель научно-технического уровня разрабатываемого НТПр.

$H_{\text{НТП Б}}$ – обобщённый количественный показатель научно-технического уровня НТПр, выбранного за базу для сравнения.

$$H_{\text{НТП Н}} = \frac{\sum_i Q_{H_i} R_i}{\sum_i R_i}, \tag{3.2}$$

где:

Q_{H_i} – значение i –го признака НТПр, являющейся результатом разработки, выраженного в баллах

R_i – значение весового коэффициента i –го признака НТПр.

$$H_{\text{НТП Б}} = \frac{\sum_i Q_{\text{Б}_i} R_i}{\sum_i R_i}, \qquad (3.3)$$

где:

$Q_{\text{Б}_i}$ – значение i –го признака НТПр, выбранного за базу для сравнения.

В таблице 3.1 отражен научно-технический эффект НТПр с соответствующими бальными оценками.

Таблица 3.1

Научно-технический эффект НТПр с соответствующими бальными оценками.

Признак научно-технического эффекта НТПр	Значимость признаков научно-технического эффекта НТПр	Уровень свойств НТПр, выбранной за базу для сравнения	Уровень свойств НТПр, являющейся результатом дипломной работы
Научно-технический уровень (новизна) НТПр	0,4	4	6
Перспективность НТПр	0,3	4	5
Возможность применения результатов НТПр	0,3	5	6

Исходя из таблицы 3.1 значение обобщённых показателей:

$$H_{\text{НТП Н}} = \frac{6 * 0{,}4 + 5 * 0{,}3 + 6 * 0{,}3}{0{,}4 + 0{,}4 + 0{,}3} = 5{,}7$$

$$H_{\text{НТП Б}} == \frac{4 * 0{,}4 + 4 * 0{,}3 + 5 * 0{,}3}{0{,}4 + 0{,}4 + 0{,}3} = 4.3$$

$$J_{\text{НТП}} = \frac{5{,}7}{4{,}3} = 1{,}33$$

3.2 Планирование разработки

3.2.1 Определение трудоемкости разработки алгоритмов и программных продуктов

Трудоемкость определяется по каждой стадии работ. Структура трудовых затрат разработки алгоритмов и программных продуктов представлена в таблице 3.2. По данной таблице производится расчет этапов решения задач для разработки программных алгоритмов согласно формуле 3.3:

$$t_{\text{ПП}} = t_{\text{О}} + t_{\text{И}} + t_{\text{А}} + t_{\text{К}} + t_{\text{От}} + t_{\text{Д}}\,, \qquad (3.3)$$

где:

$t_{\text{о}}$ – затраты труда на подготовку описания задачи;

$t_{\text{и}}$ – затраты труда на изучение и постановку задачи;

$t_{\text{А}}$ – Затраты на разработку алгоритма решения задачи;

$t_{\text{к}}$ – затраты труда на программирование по блок схеме;

$t_{\text{от}}$ – затраты труда на отладку программы;

$t_{\text{Д}}$ – затраты труда на подготовку документации по программному продукту;

$$t_{\text{и}} = \frac{QB}{75K} \quad (3.4)$$

$$t_{\text{А}} = \frac{Q}{20K} \quad (3.5)$$

$$t_{K} = \frac{Q}{10K} \quad (3.6)$$

$$t_{\text{от}} = \frac{Q}{5K} \quad (3.7)$$

$$t_{\text{Д}} = \frac{1.75Q}{15K} \quad (3.8)$$

где:

$Q -$ условное количество операторов (строк) в машинной программе;

$B -$ увеличение затрат на изучение и постановку задачи в следствии её сложности и новизны;

$K -$ коэффициент квалификации разработчика;

$$Q = qK_c(1 + K_{k1} + .. + K_{kn}) , \quad (3.9)$$

где:

$q -$ количество этапов и элементарных процедур преобразования информации;

$K_c -$ коэффициент сложности программы;

$K_{k1} = \ldots = K_{kn} -$ коэффициенты коррекции при разработке, в среднем;

$n -$ количество коррекций.

Таблица 3.2

Структура трудовых затрат разработки алгоритмов и программных продуктов

№ стадии работ	Наименование стадии (этапа) работ	Удельный вес %	Трудоемкость чел.-дн.
1	Анализ предметной области	1	1
2	Изучение средств разработки	1	1
3	Изучение программируемой задачи	2	2
4	Анализ методов решения задачи	6	6
5	Составление структурной схемы алгоритма	3	3
6	Технико-экономическое обоснование выбранного варианта алгоритма	6	5
7	Уточнение и доработка выбранного варианта алгоритма	13	12
8	Составление программы	43	40
9	Отладка программы и составление документации	11	10
10	Анализ работы ПП	6	5
11	Испытание ПП в реальных условиях	8	7
Итого		100	92

$t_{\text{о}} = 1$, $q = 10$, $K_c = 2$, K_{ki}=0,06, $n = 10$, $B = 2{,}2$, $K = 1{,}3$.

Тогда:

$$Q = 32\ ;$$

$$t_{\text{и}} = \frac{32 * 2{,}2}{75 * 1{,}3} = 0{,}722\ ;$$

$$t_{\text{А}} = \frac{32}{20 * 1{,}3} = 1{,}23\ ;$$

$$t_K = \frac{32}{10 * 1{,}3} = 2{,}46\ ;$$

$$t_{\text{от}} = \frac{32}{5 * 1{,}3} = 4{,}92\ ;$$

$$t_{\text{Д}} = \frac{1{,}75 * 32}{15 * 1{,}3} = 2{,}87$$

Общее количество затраченного труда $t_{\text{ПП}} \approx 15$ чел.-час.

3.2.2 Построение и расчёт сетевого графика

Для оценки продолжительности и планирования НИР используется сетевой метод планирования. Определение продолжительности работ в сетевом графике отображено в таблице 3.3.

Таблица 3.3

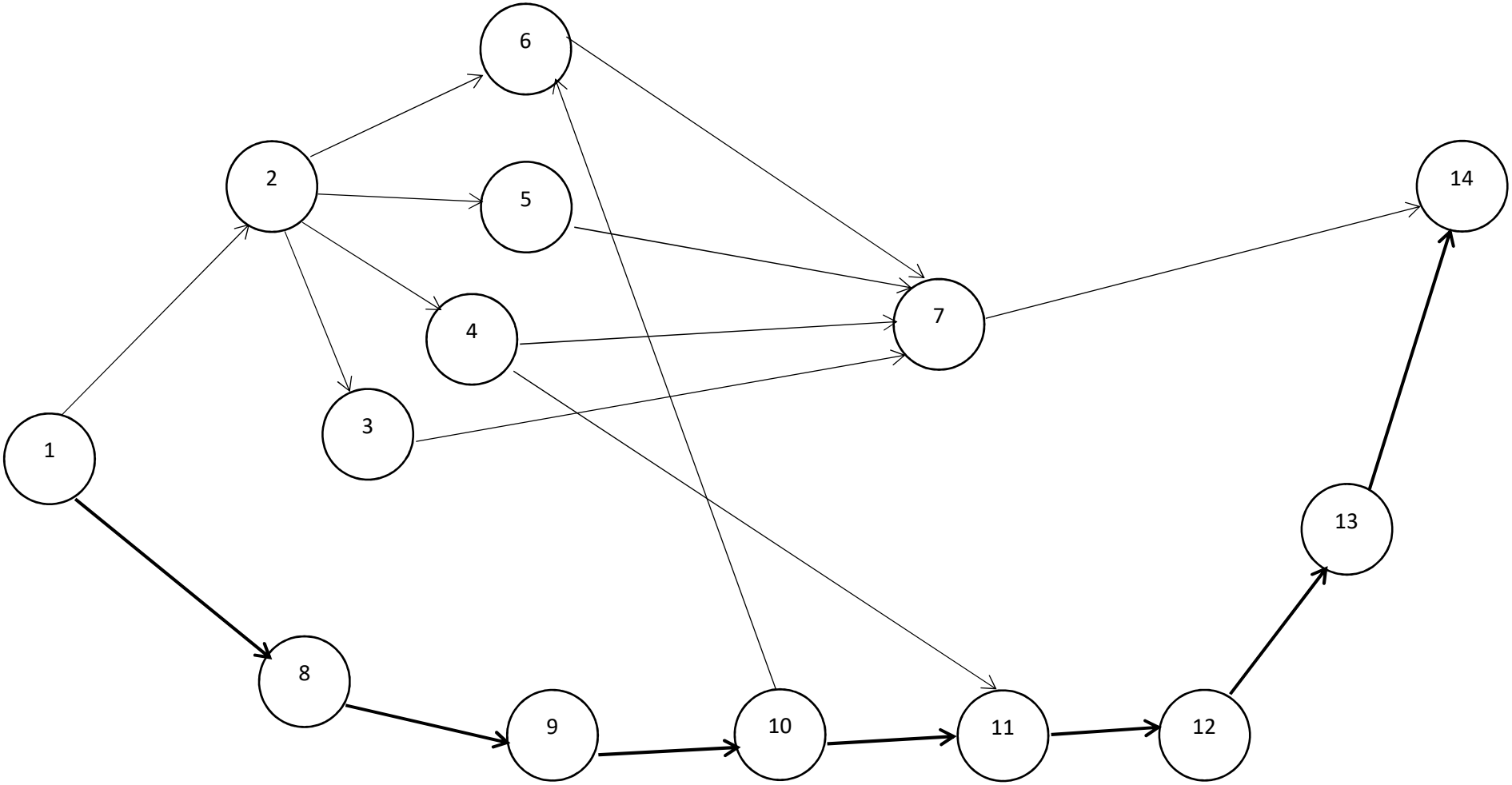
Определение продолжительности работ в сетевом графике.

№ п/п	Наименование работы	Номер события		Продолжительность работы		
		Начальное	Конечное	t_{min}	t_{max}	$t_{ож}$
1	Формирование проектов по стабилизации и отображению информации в Keil uVision5 и VisualStudio 2017.	1	2	1ч	2ч	1,4ч
2	Инициализация периферии, портов ввода-вывода, настройка ОСРВ в Keil uVision5.	1	3	1ч	2ч	1,4ч
3	Разработка алгоритма работы задачи ОСРВ по приему данных с датчиков в Keil uVision5.	2	7	3ч	6ч	4,2ч
4	Разработка алгоритма работы задачи ОСРВ по расчету углов отклонения автомата-перекоса и оборотов бесколлекторного мотора в Keil uVision5.	2	7	5ч	7ч	5,8ч
5	Разработка алгоритма работы задачи ОСРВ по управлению автоматом-перекосом, а также бесколлекторным мотором в Keil uVision5.	2	7	3ч	6ч	4,2ч

Продолжение таблицы 3.3

6	Разработка алгоритма работы задачи ОСРВ по приему необходимых данных, полученных от компьютера, посредством сигнала WiFi в Keil uVision5.	2	7	4ч	7ч	5,2ч
7	Отладка программы, созданной в Keil uVision5.	6	14	4ч	5ч	4,4ч
8	Формирование внешнего вида программы в VisualStudio 2017	1	8	1ч	3ч	1,8ч
9	Создание функций, реализующих TCP/IP подключение для WiFi сетей в VisualStudio 2017.	8	10	1ч	3ч	1,8ч
10	Создание функций, реализующий передачу информации посредством TCP/IP соединения в VisualStudio 2017.	9	11	3ч	4ч	3,4ч
11	Создание функций, реализующих прием данных, полученных с TCP/IP сервера последством WiFi в VisualStudio 2017.	10	12	3ч	5ч	3,8ч
12	Создание функций, реализующих отображение полученной информации в виде графиков в VisualStudio 2017.	11	13	3ч	5ч	3,8ч
13	Отладка программы, созданной в VisualStudio 2017	12	14	3ч	6ч	4,2ч
14	Отладка взаимодействия между программами, созданными для микроконтроллера и компьютера	6	14	5ч	10ч	7ч

Сетевой график



$t_{ож}$ −ожидаемое время работы:

$$t_{ож} = \frac{3t_{min} + 2t_{max}}{5} \quad (3.10)$$

$t_{кр}$ − путь наибольшей по времени исполнения продолжительности называется критическим путём.

$t_{кр} = 27{,}4$ ч.

Определение основных параметров сетевого графика.

$t_{р}(i)$ − ранний срок свершения i-го события, равный максимальному из путей ведущему к данному событию от исходного события, рассчитанный по формуле 3.11:

$$t_{р}(i) = max \sum t_{ож} \quad (3.11)$$

$t_{п}(i)$ − поздний срок свершения i-го события,

$$t_{п}(i) = t_{кр} - max \sum_{i}^{N} t_{ож} \quad (3.12)$$

$R(i)$ − резерв времени i-го события:

$$R(i) = t_{п}(i) - t_{р}(i) \quad (3.13)$$

Таблица 3.4

Расчет сроков свершения событий

№ состояния	$t_р(i)$	$t_п(i)$	R(i)
1	0	0	0
2	1,4	1,4	0
3	2,8	16,2	13,4
4	8,6	14,6	6
5	7	16,2	9,2
6	8	15,2	7,2
7	13	13	0
8	1,8	3,4	1,6
9	3,6	5,2	1,6
10	7	8,6	1,6
11	12,4	12,4	0
12	16	16,2	0,2
13	20,2	22,4	2,2
14	27,4	27,4	0

t_{PH} − самый ранний из возможных сроков начала работы:

$$t_{PH}(ij) = t_{\text{р}}(i) \tag{3.14}$$

$t_{\text{ПН}}$ − самый поздний срок начала работы:

$$t_{\text{ПН}}(ij) = t_{\text{п}}(j) - t(ij) \tag{3.15}$$

t_{PO} − самый ранний срок окончания работы:

$$t_{PO}(ij) = t_{\text{р}}(i) + t(ij) \tag{3.16}$$

$t_{ПО}$ − самый поздний срок окончания работы:

$$t_{ПО}(ij) = t_{П}(j) \quad (3.17)$$

Таблица 3.5

Расчет сроков окончания работ

Шифр работы (ij)	Продолжительность работы t(ij)	$t_{РН}$(ij)	$t_{ПН}$(ij)	$t_{РО}$(ij)	$t_{ПО}$(ij)
1-2	1,4	0	0	1,4	1,4
1-8	1,8	0	1,6	1,8	3,4
2-3	4,2	2,8	12	7	16,2
2-4	5,8	8,6	8,8	14,4	14,6
2-5	4,2	7	12	11,2	16,2
2-6	5,2	8	10	13,2	15,2
6-7	4,4	13	8,6	13	13
7-14	7	20,4	20,4	27,4	27,4
3-7	4,4	8,6	8,6	13	13
4-7	4,4	8,6	8,6	13	13
5-7	4,4	8,6	8,6	13	13
4-11	3,8	8,6	8,6	12,4	12,4
8-9	1,8	3,6	3,4	5,2	5,2
9-10	3,4	7	7,2	8	8,6
10-11	3,8	8,6	8,6	12,4	12,4
11-12	3,8	16	16	16,2	16,2
10-6	5,2	8	10	13,2	15,2
12-13	4,2	20,2	20,2	21	22,4
13-14	7	20,4	20,4	27,4	27,4

3.3 Определение затрат, себестоимости и цены

Для определения затрат на проектирование, необходимо произвести расчет затрат на оплату труда работников.

Заработная плата рассчитывается по следующей формуле 3.18:

$$З_{ЗП\ j} = T_j \tau_j\ , \qquad (3.18)$$

где:

$T_j -$ трудоемкость j-й стадии работ

$\tau_j -$ средняя дневная тарифная ставка оплаты работ j-ой стадии работы

В таблице 3.6 приведены данные для расчета заработной платы на оплату труда персонала.

Таблица 3.6

Расчёт заработной платы при оценке затрат на проектирование.

№ стадии работ	Трудоемкость стадии (этапа) (чел.-дн.)	Исполнители		Дневная ставка (руб)	Средняя дневная ставка (руб)	Заработная плата (руб)
		Должность	Численность			
1	1	Разработчик измерительных систем	1	500	600	600
2	1	Разработчик измерительных систем	1	600	500	500
3	2	Разработчик измерительных систем	1	600	600	1200
4	6	Разработчик измерительных систем	1	700	600	3600
5	3	Разработчик измерительных систем	1	400	500	1500

Продолжение таблицы 3.6

6	5	Разработчик измерительных систем	1	500	600	3000
7	12	Разработчик измерительных систем	1	600	800	9600
8	40	Разработчик измерительных систем	1	800	700	28000
9	10	Разработчик измерительных систем	1	700	600	6000
10	5	Разработчик измерительных систем	1	500	700	3500
11	7	Разработчик измерительных систем	1	500	600	4200
Итог						61700

Себестоимость НТПр определяются по следующей формуле:

$$З_{НТПр} = \frac{З_{ЗП}}{d_{зп}},$$

где:

$З_{НТПр}$ – затраты на НТПр;

$З_{ЗП}$ – оплата труда персонала в соответствии с действующими системами и формами оплаты труда.

$d_{зп}$ – удельный вес оплаты труда с начислением в общих затратах на создании НТПр.

$$d_{зп} = 0{,}4$$

Тогда:

$$З_{НТПр} = \frac{61700}{0{,}4} = 154250 \text{ руб}$$

Цена НТПр определяется исходя из принципа безубыточности деятельности предприятия, сущностью которого является получение прибыли, которая позволяет организации выплачивать обязательные платежи в бюджет, а также расширять собственную деятельность.

Цена НТПр считается по формуле 3.19:

$$Ц_{НТПр} = З_{НТПр} + \frac{З_{ЗП}\rho_{ЗП}}{100} \quad (3.19)$$

$\rho_{ЗП}$ – уровень рентабельности (прибыли по отношению к оплате труда персонала), обеспечивающий безубыточную деятельность:

$$\rho_{ЗП} = 300\%$$

Цена программного продукта:

$$Ц_{НТПр} = 154250 + \frac{61700 * 300}{100} = 339350 \text{ руб}$$

3.4 Определение и оценка показателей экономической эффективности разработки алгоритмов и программных продуктов.

Данная разработка позволяет решить поставленные задачи по стабилизации скорости полета микро-БПЛА с большей скоростью, поскольку применены алгоритмы, строящиеся на ОСРВ, что позволяет системе быстрее отзываться на воздействия внешних факторов.

Для расчёта годового экономического эффекта используется формула 3.20:

$$Э_{ппг} = \sum_{i=1}^{n} \Delta T_{мi} C_{ВТ}\,, \qquad (3.20)$$

где:

$\Delta T_{мi}$ – экономия машинного времени по i-й задаче;

$C_{ВТ}$ – стоимость одного машинного, руб;

n – количество задач, решаемых в год.

Разработка позволяет сократить время стабилизации скорости в 2 раза.

Средняя стоимость 1 машинного часа ЭВМ составляет: $C_{ВТ} = 213$ руб.

Предполагается, что данной продукцией будут пользоваться 2 раза в день

Следовательно:

$$n = 2 * 365$$

Таким образом:

$$Э_{ппг} = 730*2*213=310980 \text{ руб}$$

Величина экономической эффективности определяется как:

$$E = \frac{Э_{ппг}\gamma}{Ц_{НТП.Н}}$$

$\gamma = 1{,}5 -$ коэффициент долевого участия разработки в получении ожидаемой экономической эффективности

$$E = \frac{310980 * 2{,}5}{339350} = 1{,}37$$

Срок окупаемости вложений:

$$T_{ок} = \frac{1}{E} \approx 1 \text{ год.}$$

4. Охрана труда и окружающей среды

4.1 Охрана труда разработчика комплексного алгоритма для стабилизации скорости полета микро-БПЛА

В данной дипломной работе разработан макет комплексной системы управления микро-БПЛА и алгоритм стабилизации скорости полета микро-БПЛА.

В качестве микро-БПЛА рассматривается вертолет на радиоуправлении. В соответствии с этим, макет системы управления представляет собой автомат-перекос, в состав которого входит три сервопривода, а также бесколлекторный мотор постоянного тока. Управление автоматом-перекосом и мотором осуществляется автоматически микроконтроллером, исходя из показаний датчиков и применения алгоритма стабилизации скорости.

Разработка алгоритма проводилась с использованием персонального компьютера, сборка макета системы проводилась с использованием бытового электропаяльника.

4.2 Анализ условий труда разработчика алгоритма для стабилизации скорости полёта микро-БПЛА

Характеристика производственного помещения

Разработка проводилась в помещении размером 3,5м – 3м – 2,7м. Количество рабочих мест – 1, соответственно на рабочего приходится 28,35м^3, что удовлетворяет нормам СанПиН 2.2.2/2.4.1340-03.

Характеристика производственного процесса

Производственным процессом является разработка алгоритмов стабилизации скорости и макета комплексной системы микро-БПЛА.

Помещение оснащено персональным компьютером (мощность 500Вт), многофункциональным устройством (мощность 70Вт), электропаяльником (мощность 40Вт).

4.2.1 Микроклимат производственного помещения

Исходя из критериев теплового и функционального состояния человека установлены оптимальные и допустимые микроклиматические условия, согласно которым обеспечивается ощущение теплового комфорта во время рабочей смены, а также не возникают отклонения в состоянии здоровья работника.

Показатели, характеризующие микроклимат в производственных помещениях:

1. Температура воздуха;
2. Температура поверхностей;
3. Относительная влажность воздуха;
4. Скорость движения воздуха;
5. Интенсивность теплового облучения.

При обеспечении оптимальных величин микроклимата, перепады температур воздуха по высоте и по горизонтали, а также изменение температуры воздуха в течение смены не должны превышать 2° С, с учетом температуры поверхностей ограждающих конструкций, устройств и технологического оборудования. Оптимальные величины показателей микроклимата на рабочем месте должны соответствовать значениям, представленным в таблице 4.1.

Таблица 4.1

Оптимальные величины показателей микроклимата
на рабочих местах производственных помещений

Период года	Категория работ по уровню энергозатрат, Вт	Температура воздуха, °С	Температура поверхностей, °С	Относительная влажность воздуха, %	Скорость движения воздуха, м/с
Холодный	Ia (до 139) Iб (140-174) Iia(175-232) Iiб(233-290) III(более 290)	22-24 21-23 19-21 17-19 16-18	21-25 20-24 18-22 16-20 15-19	60-40 60-40 60-40 60-40 60-40	0,1 0,1 0,2 0,2 0,3
Теплый	Ia (до 139) Iб (140-174) Iia (175-232) Iiб (233-290) III (более 290)	23-25 22-24 20-22 19-21 18-20	22-26 21-25 19-23 18-22 17-21	60-40 60-40 60-40 60-40 60-40	0,1 0,1 0,2 0,2 0,3

При обеспечении допустимых величин микроклимата, с учетом температуры поверхностей ограждающих конструкций, устройств и технологического оборудования, перепад температуры воздуха по высоте не

должен превышать 3° С, по горизонтали – при категориях работ Iа и Iб – 4° С, при категориях работ IIа и IIб – 5° С, при категории работ III – 6° С, абсолютные значения температур не должны выходить за пределы величин, указанных в таблице 4.2.

Таблица 4.2

Допустимые величины показателей микроклимата
на рабочих местах производственных помещений

Период года	Категория работ по уровню энергозатрат, Вт	Температура воздуха, °С		Температура поверхностей, °С
		диапазон ниже оптимальных величин	диапазон выше оптимальных величин	
Холодный	Iа (до 139)	20,0-21,9	24,1-25,0	19,0-26,0
	Iб (140-174)	19,0-20,9	23,1-24,0	18,0-25,0
	Iiа (175-232)	17,0-18,9	21,1-23,0	16,0-24,0
	Iiб (233-290)	15,0-16,9	19,1-22,0	14,0-23,0
	III (более 290)	13,0-15,9	18,1-21,0	12,0-22,0
Теплый	Iа (до 139)	21,0-22,9	25,1-28,0	20,0-29,0
	Iб (140-174)	20,0-21,9	24,1-28,0	19,0-29,0
	Iiа (175-232)	18,0-19,9	22,1-27,0	17,0-28,0
	Iiб (233-290)	16,0-18,9	21,1-27,0	15,0-28,0
	III (более 290)	15,0-17,9	20,1-26,0	14,0-27,0

Допустимые величины интенсивности теплового облучения поверхности тела работающих от производственных источников: при менее 25% облучаемой

поверхности тела интенсивность теплового облучения не должна превышать 100 Вт/м^2, при 25-50% - 70 Вт/м^2, при более 50% - 35Вт/м^2.

Допустимые величины интенсивности теплового облучения работающих от источников излучения (расплавленный металл, стекло, пламя) не должны превышать 140 Вт\м^2 при менее 25% облучаемой поверхности тела. Также обязательно использование средств индивидуальной защиты.

Рассмотрим текущие показатели микроклимата рабочего места производственного помещения с учётом периодов года и энергозатрат. Необходимо обеспечение оптимальных или допустимых значений величин показателей микроклимата для сохранения теплового комфорта и сохранения здоровья работника.

По уровню энергозатрат работа соответствует Iа категории.

В холодный период года:

1. Температура воздуха в помещении – 23° С.
2. Температура поверхностей макета – 24° С.
3. Относительная влажность воздуха в помещении – 50%.
4. Скорость движения воздуха составляет 0,1м/с.

В теплый период года:

1. Температура воздуха в помещении – 24° С.
2. Температура поверхностей макета – 25° С.
3. Относительная влажность воздуха в помещении – 50%.
4. Скорость движения воздуха составляет 0,1м/с.

Показатели рабочего места, полученные в теплый и холодный периоды года, удовлетворяют оптимальным условиям микроклимата согласно СанПиН 2.2.4.548-96 «Гигиенические требования к микроклимату производственных помещений».

4.2.2 Производственное освещение

Согласно СниП 23-05-95, организовывать освещение производственного освещения следует с помощью разрядных ламп, ввиду их экономичности, однако допускается использование галогенных ламп в случаях, когда установка разрядных ламп невозможна, либо нецелесообразна. Для местного освещения рекомендуется использование как разрядных ламп, так и галогенных. Применение ксеноновых ламп не допускается внутри помещений.

Объектом различения является резистор, диаметр ножки которого укладывается в диапазон от 0,3мм до 0,5мм, соответственно работа – средней точности.

Освещенность рабочей поверхности, создаваемая светильниками общего освещения в системе комбинированного, должна составлять не менее 10% нормируемой для комбинированного освещения при тех источниках света, которые применяются для местного освещения. При использовании разрядных ламп освещенность должна быть не менее 200лк, при использовании ламп накаливания – не менее 75лк. Создавать освещенность от общего освещения в системе комбинированного более 500лк при разрядных лампах и более 150лк при лампах накаливания опускается только при наличии обоснований.

В рассматриваемом производственном помещении используется комбинированная система освещения. Источник освещения – 4 светодиодные лампы на потолке со световым потоком 490лм каждая, а также аналогичная лампа для местного освещения (при работе с паяльником). Освещенность производственного помещения составляет 450лк, рабочей поверхности – 140лк, что удовлетворяет нормам СниП 23-05-95.

4.2.3 Шум

На данном рабочем месте источниками шума являются вентиляторы охлаждения системного блока компьютера и бесколлекторный мотор A2212/13T, установленный на макете системы управления микро-БПЛА.

При помощи шумомера было установлено, что уровень шума при работе только персонального компьютера составляет 22дБ. Соответственно, такой уровень шума сохраняется практически на протяжении всего рабочего времени, так как мотор, установленный на макете, включается редко и кратковременно (во время испытаний).

Суммарный уровень шума на рабочем месте на максимальных оборотах мотора составляет 40дБ, что удовлетворяет нормам СН 2.2.4/2.1.8.562-96.

В таблице 4.3 приведены допустимые уровни звука и эквивалентный уровень звука для деятельности на рассматриваемом рабочем месте.

Таблица 4.3

Предельно допустимые уровни звукового давления, уровни звука и эквивалентные уровни звука для рассматриваемого рабочего места.

Вид трудовой деятельности, рабочее место.	Уровни звукового давления, дБ, в октавных полосах со среднегеометрическими частотами, Гц								Уровни звука и эквивалентные уровни звука, дБА
	63	125	250	500	100	2000	4000	8000	
Высококвалифицированная работа, требующая сосредоточенности, административно-управленческая деятельность, измерительные и аналитические работы в лаборатории; рабочие места в помещениях цехового управленческого аппарата, в рабочих комнатах конторских помещений, в лабораториях	79	70	68	58	55	52	52	49	60

4.2.4 Вибрация

Оборудование, использованное при проектировке, а также макет системы управления не создают вибраций, следовательно, помещение соответствует ГОСТ 12.1.012-2004.

4.2.5 Содержание вредных веществ в воздухе на рабочем месте

В связи с тем, что объем производства составляет 1 экземпляр, с учетом количества электронных компонентов и размеров макетной платы, целесообразно применение ручной пайки. Пайка платы осуществляется при помощи припоя ПОС-61 ГОСТ 21931-76, а также спирто-канифольного флюса.

Согласно ГОСТ 12.1.005-88, должны быть определены вещества, которые могут выделяться в воздух рабочей зоны. При наличии в воздухе нескольких вредных веществ контроль воздушной среды допускается проводить по наиболее опасным и характерным веществам.

Химический состав низкотемпературного припоя ПОС-61:

1. Олово – 59-61%;
2. Свинец – 38-40%;
3. Примеси – 0.37%.

Химический состав флюса ФКСП:

1. Сосновая канифоль – 60-70%;
2. Спирт этиловый – 30-40%.

Для промывки готового изделия от остатков флюса используется спирт этиловый.

Процесс пайки сопровождается загрязнением воздуха аэрозолями припоя, флюса, парами смывки. Наиболее опасными являются пары свинца, так как являются наиболее ядовитыми.

Пары свинца не должны превышать предельно допустимых концентраций.

Для предотвращения появления заболеваний, необходимо соблюдать среднесменную предельно допустимую концентрацию вредных веществ на рабочем месте.

Согласно ГОСТ 12.1.005-88 в таблице 4.4 приведены предельно допустимые концентрации вредных веществ.

Таблица 4.4

Предельно допустимая концентрация веществ в воздухе.

Наименование вещества	Величина ПДК, мг/м^3
Свинец	0,01
Олово	10
Спирт этиловый	1000

Поскольку свинец является самым опасным веществом, применяемым на данном рабочем месте, контроль воздушной среды можно провести исходя из содержания аэрозоля свинца в воздухе.

Количество аэрозоля свинца, выделяемого при пайке, составляет 0.04мг на 100 паек. Количество рабочих мест, на которых ведется пайка – 1. Количество паек в минуту – 10. Длительность смены – 8ч. Размеры помещения – 3,5м-3м-2,7м.

Концентрация аэрозоля свинца в воздухе находится по формуле 4.1:

$$\mathrm{C} = \frac{0{,}6 * \mathrm{A} * \mathrm{B} * N * t}{V}, \quad (4.1)$$

где:

A – удельное образование аэрозоля свинца;

B – количество паек в минуту;

N – количество рабочих мест в помещении;

t – длительность смены;

V – объем помещения. м^3

$$\frac{0{,}6 * 0{,}04 * 10 * 1 * 8}{28{,}35} = 0{,}07(\text{мг/м}^3)$$

Концентрация свинца в воздухе превышает предельно допустимую в 7 раз, соответственно в помещении необходима система вентиляции.

4.3 Организационно-технические мероприятия по охране труда

4.3.1 Расчет вентиляции производственного помещения

Поскольку концентрация аэрозоля свинца в воздухе превышает предельно допустимую норму, необходима вентиляция в производственном помещении.

На рабочем месте должны создаваться метеорологические условия в соответствии с ГОСТ 12.1 005-88.

Согласно постановлению МтиСР РФ от 17.07.2003 г. №55, вентиляционная установка должна быть включена до начала работ, а выключена после их проведения. Включенный электропаяльник находится в зоне действия местной вентиляции.

Местная вентиляция является наиболее эффективным и экономичным средством соблюдения санитарных норм на рабочем месте при пайке. Широкое применение в такой работе имеет местная вытяжная вентиляция.

В качестве воздухоприемника используется вытяжной зонт, имеющий форму конуса или пирамиды и расположенный на некотором расстоянии от источника выделения вредных веществ.

Требуемый диаметр вытяжного отверстия воздухоприемника, который будет удалять вредные пары свинца при работе, рассчитывается по формуле 4.2:

$$d = \sqrt{\frac{4 * L}{v * \pi * 3600}}, \quad (4.2)$$

где:

L – количество воздуха, удаляемое воздухоотсосом за 1 час;

v – скорость воздуха в отверстии воздухоприемника.

Количество воздуха, которое необходимо отвести от рабочего места за 1 час рассчитывается по формуле:

$$L = \frac{1000 * G}{\text{ПДК}_{\text{рз}}}, \quad (4.3)$$

где:

G – удельный показатель выделения аэрозоля свинца в воздух;

$\text{ПДК}_{\text{рз}}$ – предельно допустимые концентрации вещества в рабочей зоне.

Удельный показатель выделения аэрозоля свинца в воздух при пайке припоем ПОС-61 составляет 0.0011 г/ч. $\text{ПДК}_{\text{рз}}$ для свинца, согласно ГН2.2.5.1313-03 составляет 0,05мг/м^3. Скорость воздуха в отверстии воздухоприемника 0,8м/с.

Тогда:

$$L = \frac{1000 * 0{,}0011}{0{,}05} = 22(\text{м}^3/\text{ч})$$

Тогда диаметр вытяжного отверстия:

$$d = \sqrt{\frac{4 * 22}{0{,}8 * \pi * 3600}} = 0{,}099(\text{м})$$

Для данного рабочего места выбрано вытяжное устройство ММН-100-15, у которого диаметр вытяжного отверстия составляет 100мм.

Также, согласно постановлению МтиСР РФ от 17.07.2003 г. №55, необходимо применение средств индивидуальной защиты, излишки флюса и припоя удалять с помощью специальных материалов (хлопчатобумажная ткань, асбест и другие).

4.3.2 Пожарная безопасность

В процессе создания макета используются пожаровзрывоопасные материалы, характеристики которых приведены в таблице 4.5.

Необходимо определить категорию помещения по пожарной и взрывопожарной опасности в соответствии с ОНТП 24-86. Для определения категории помещения рассчитывается избыточное давление взрыва в помещении.

Таблица 4.5

Характеристики пожаровзрывоопасных материалов.

Вещество	Температура воспламенения	Температура самовоспламенения	Нижний предел взрываемости	Верхний предел взрываемости
Спирт этиловый	18	104	3,6%	19%
Канифоль	-	850	12,4г/м^3	-

Согласно ОНТП 24-86, избыточное давление взрыва для горючих веществ, состоящих из атомов C, H, O, N, Cl, Br, I, F, определяется по формуле 4.4:

$$\Delta P = (P_{max} - P_0) * \frac{m * z}{V_{св} * \rho_{г,п}} * \frac{100}{C_{ст}} * \frac{1}{K_н}, \quad (4.4)$$

где:

P_{max} – максимальное давление взрыва стехиометрической газовоздушной или паровоздушной смеси в замкнутом объеме;

P_0 – начальное давление(101кПа);

m – масса горючего вещества;

z – коэффициент участия горючего во взрыве, который можно определить по таблице 4.6;

$V_{св}$ – свободный объем помещения;

$\rho_{г,п}$ – плотность пара или газа;

$C_{ст}$ – стехиометрическая концентрация горючего газа или паров легковоспламеняющихся жидкостей;

$K_н$ – коэффициент, учитывающий негерметичность помещения и неадиабатичность процесса горения.

Таблица 4.6

Коэффициенты участия горючего во взрыве.

Вид горючего вещества	Значение
Горючие газы	0,5
Легковоспламеняющиеся и горючие жидкости, нагретые до температуры вспышки и выше	0,3
Легковоспламеняющиеся и горючие жидкости, нагретые ниже температуры вспышки, при наличии возможности образования аэрозоля	0,3
Легковоспламеняющиеся и горючие жидкости, нагретые ниже температуры вспышки, при отсутствии возможности образования аэрозоля	0

Расчет стехиометрической концентрации горючего газа или паров легковоспламеняющейся жидкости выполняется по формуле 4.5:

$$C_{ст} = \frac{100}{1 + 4{,}84 * \beta'}, \quad (4.5)$$

где β – стехиометрический коэффициент кислорода в реакции сгорания, который рассчитывается по формуле 4.6:

$$\beta = n_с + \frac{n_н - n_х}{4} - \frac{n_0}{2}, \quad (4.6)$$

где:

$n_с$, $n_н$, n_0, $n_х$ – число атомов С, Н, О и галоидов в молекуле горючего.

Для этилового спирта (как наиболее опасного):

Максимальное давление взрыва 750кПа;

Начальное давление 101кПа;

Масса горючего вещества 100мг;

Коэффициент участия горючего во взрыве 0,3;

Свободный объем помещения 28м^3;

Плотность пара 2,043 кг/м$^{-3}$;

Коэффициент негерметичности помещения принят равным 3 в соответствии с ОНТП 24-86.

Параметры для расчета стехиометрического коэффициента:

Число атомов углерода – 2, число атомов водорода – 6, число атомов кислорода – 1, число галоидов – 0.

Тогда:

$$\beta = 2 + \frac{6}{4} - \frac{1}{2} = 3$$

$$C_{ст} = \frac{100}{1 + 4{,}84 * 3} = 6{,}44$$

$$\Delta P = (750000 - 101000) * \frac{0{,}0001 * 0{,}3}{28 * 2{,}043} * \frac{100}{6{,}44} * \frac{1}{3} = 1{,}76(\text{Па})$$

В следствие того, что этиловый спирт применяется в данной работе исключительно для отмывки поверхностей от флюса после пайки, количество паров спирта очень мало, соответственно избыточное давление взрыва при таком объеме не представляет опасности, что подтверждается приведенными выше вычислениями.

5. Заключение

В данной работе требовалось разработать алгоритм стабилизации скорости полета микро-БПЛА. В ходе выполнения поставленной задачи были достигнуты следующие результаты:

1. Собран макет, состоящий из платы управления и модели автомата-перекоса;
2. Разработан алгоритм стабилизации скорости полета микро-БПЛА;
3. Разработан алгоритм визуализации результатов работы программы;
4. Разработано программное обеспечение для микроконтроллера;
5. Разработано программное обеспечение для персонального компьютера;
6. Произведена верификация разработанного программного обеспечения;
7. Произведен анализ полученных данных, вычисление которых производится в ходе выполнения разработанных алгоритмов.

В ходе верификации разработанного программного обеспечения было выявлено, что алгоритм выполняется верно, беспроводная связь между микроконтроллером и персональным компьютером устанавливается корректно, визуализация актуальных значений высоты, углов тангажа, крена и курса, а также скоростей производится правильно. Следовательно, система управления удовлетворяет заданным техническим требованиям, предъявленным к данной дипломной работе.

Анализ полученных данных показал, что для повышения точности измерений необходимо использовать более совершенные датчики.

В экономической части был произведен расчет следующих параметров:

1. Индекс научно-технической прогрессивности $J_{\text{НТП}} = 1{,}33$. Величина данного показателя говорит о том, что разработка превышает уровень научно-технической прогрессивности своего аналога.

2. В ходе планирования длительности работ с помощью сетевого метода, а также расчета трудозатрат, было выяснено, что максимальная длительность выполнения работы составляет 27,4 часа, а трудоемкость – 15 час.-чел
3. В процессе подсчета денежных затрат на оплату труда, а также на НТПр, были получены следующие значения: выплаты заработной платы – 61700 руб., расходы на НТПр – 154250 руб. С учетом рентабельности, цена программного продукта составила 339350 руб.
4. Расчет экономической эффективности был сделан с учетом коэффициента долевого участия разработки, который должен составлять не менее 1,5. Величина экономической эффективности составила 1,37, из чего следует срок окупаемости продукта – примерно 1 год.

В части охраны труда разработчика комплексного алгоритма для стабилизации скорости полета микро-БПЛА был проведен анализ труда, в ходе которого были исследованы: микроклимат помещения, освещенность, шум, вибрация, содержание вредных веществ в воздухе. В ходе анализа помещения выяснилось, что количество вредных веществ в воздухе превышает допустимую норму, в связи с чем был произведен расчет местной вытяжной вентиляции. Также в работе используется взрывопожароопасное вещество. Был произведен расчет по пожаробезопасности помещения.

Список использованных источников

1. Братухин И. П. Проектирование и конструкции вертолетов. – М.: Гос. Изд-во оборонной промышленности, 1955.
2. Тамре Л. Введение в тестирование программного обеспечения. – М.:Вильямс, 2003.
3. Калбертсон Р., Браун Л., Кобб Г. Быстрое тестирование. – М.:Вильямс, 2002.
4. Кирпичникова Л. Г., Кузнецова Е. Г. Лабораторные работы по курсу «Автоматические информационные устройства и системы летательных аппаратов». – М.:МАИ, 1982.
5. Юрьев Б.М. Аэродинамический расчёт вертолетов. – М.: Гос. Изд-во оборонной промышленности, 1956.
6. BMP280. Техническая документация.
7. MPU9250. Техническая документация.
8. SG90. Техническая документация.
9. STM32F103C8T6. Техническая документация.
10. A2212/13T. Техническая документация.
11. Курниц А. FreeRTOS.
12. Панагушин В. П., Ковалева Т.С., Малютина О. А., Михайловская Н. М., Прозорова В. С., Чайка Н. К. «Экономическое обоснование дипломных проектов (работ) по приборо- и радиоприборостроению». Методические указания. Издание шестое, исправленное и дополненное. Под редакцией д. э. н., профессора Панагушина В. П. – М.: Издательство ИВАКО Аналитик, 2008 – 44с.
13. СанПиН 2.2.2/2.4.1340-03 «Гигиенические требования к персональным электронно-вычислительным машинам и организации работы». — Москва: [б.и.], 2003. — 17 с.
14. СанПиН 2.2.4.548-96 «Гигиенические требования к микроклимату производственных помещений». — Москва: [б.и.], 1996. — 6 с

15. СниП 23-05-95 «Естественное и искусственное освещение». — Москва: [б.и.], 1995. — 90 с.
16. СН 2.2.4/2.1.8.562-96 «Шум на рабочих местах, в помещениях жилых, общественных зданий и на территории жилой застройки».
17. ГОСТ 12.1.012-2004 «Система стандартов безопасности труда. Вибрационная безопасность. Общие требования».
18. ГОСТ 21931-76 «Припои оловяно-свинцовые в изделиях. Технические условия (с изменениями N 1, 2, 3)».
19. ГОСТ 12.1.005-88 «Система стандартов безопасности труда. Общие санитарно-гигиенические требования к воздуху рабочей зоны».
20. Постановление МтиСР РФ от 17.07.2003 г. №55 «Об утверждении Межотраслевых типов инструкций по охране труда для работников, занятых проведением работ по пайке и лужению изделий».
21. ГН2.2.5.1313-03 «Предельно допустимые концентрации вредных веществ в воздухе рабочей зоны».
22. ММН-100-15. Техническая документация.
23. ОНТП 24-86 «Определение категорий помещений и зданий по взрывопожарной и пожарной опасности».

Приложение А1 Инициализация шины I2C

```
void MX_I2C1_Init(void)
{
	hi2c1.Instance = I2C1;
	hi2c1.Init.ClockSpeed = 400000;
	hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
	hi2c1.Init.OwnAddress1 = 0;
	hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
	hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
	hi2c1.Init.OwnAddress2 = 0;
	hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
	hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
	if (HAL_I2C_Init(&hi2c1) != HAL_OK)
	{
		_Error_Handler(__FILE__, __LINE__);
	 }
}
```

```
void HAL_I2C_MspInit(I2C_HandleTypeDef* i2cHandle)
{
	GPIO_InitTypeDef GPIO_InitStruct;
	if(i2cHandle->Instance==I2C1)
	{
		/*I2C1 GPIO Configuration
		PB6     ----→ I2C1_SCL
		PB7     ----→ I2C1_SDA
		*/
		 GPIO_InitStruct.Pin = GPIO_PIN_6|GPIO_PIN_7;
		GPIO_InitStruct.Mode = GPIO_MODE_AF_OD;
		GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
		HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
		__HAL_RCC_I2C1_CLK_ENABLE();
		HAL_NVIC_SetPriority(I2C1_EV_IRQn, 5, 0);
		HAL_NVIC_EnableIRQ(I2C1_EV_IRQn);
		HAL_NVIC_SetPriority(I2C1_ER_IRQn, 5, 0);
		HAL_NVIC_EnableIRQ(I2C1_ER_IRQn);
	 }
}
```

```
void MX_USART1_UART_Init(void)
{
        huart1.Instance = USART1;
         huart1.Init.BaudRate = 115200;
        huart1.Init.WordLength = UART_WORDLENGTH_8B;
        huart1.Init.StopBits = UART_STOPBITS_1;
        huart1.Init.Parity = UART_PARITY_NONE;
        huart1.Init.Mode = UART_MODE_TX_RX;
        huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
        huart1.Init.OverSampling = UART_OVERSAMPLING_16;
        if (HAL_UART_Init(&huart1) != HAL_OK)
        {
                _Error_Handler(__FILE__, __LINE__);
         }
}
```

Приложение А4 Инициализация линий приема и передачи UART

```
void HAL_UART_MspInit(UART_HandleTypeDef* uartHandle)
{


  GPIO_InitTypeDef GPIO_InitStruct;
  if(uartHandle->Instance==USART1)
  {

    __HAL_RCC_USART1_CLK_ENABLE();

    /**USART1 GPIO Configuration
    PA9     ----→ USART1_TX
    PA10    ----→ USART1_RX
    */

    GPIO_InitStruct.Pin = GPIO_PIN_9;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    GPIO_InitStruct.Pin = GPIO_PIN_10;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    /* USART1 DMA Init */
    /* USART1_RX Init */

    hdma_usart1_rx.Instance = DMA1_Channel5;
    hdma_usart1_rx.Init.Direction = DMA_PERIPH_TO_MEMORY;
    hdma_usart1_rx.Init.PeriphInc = DMA_PINC_DISABLE;
    hdma_usart1_rx.Init.MemInc = DMA_MINC_ENABLE;
    hdma_usart1_rx.Init.PeriphDataAlignment =
DMA_PDATAALIGN_BYTE;
    hdma_usart1_rx.Init.MemDataAlignment =
DMA_MDATAALIGN_BYTE;
    hdma_usart1_rx.Init.Mode = DMA_CIRCULAR;
    hdma_usart1_rx.Init.Priority = DMA_PRIORITY_LOW;

    if (HAL_DMA_Init(&hdma_usart1_rx) != HAL_OK)
    {
```

```
            _Error_Handler(__FILE__, __LINE__);

        }

        __HAL_LINKDMA(uartHandle,hdmarx,hdma_usart1_rx);

        /* USART1_TX Init */

        hdma_usart1_tx.Instance = DMA1_Channel4;
        hdma_usart1_tx.Init.Direction = DMA_MEMORY_TO_PERIPH;
        hdma_usart1_tx.Init.PeriphInc = DMA_PINC_DISABLE;
        hdma_usart1_tx.Init.MemInc = DMA_MINC_ENABLE;
        hdma_usart1_tx.Init.PeriphDataAlignment =
DMA_PDATAALIGN_BYTE;
        hdma_usart1_tx.Init.MemDataAlignment =
DMA_MDATAALIGN_BYTE;
        hdma_usart1_tx.Init.Mode = DMA_NORMAL;
        hdma_usart1_tx.Init.Priority = DMA_PRIORITY_LOW;
        if (HAL_DMA_Init(&hdma_usart1_tx) != HAL_OK)
        {

            _Error_Handler(__FILE__, __LINE__);

        }

        __HAL_LINKDMA(uartHandle,hdmatx,hdma_usart1_tx);

        /* USART1 interrupt Init */
        HAL_NVIC_SetPriority(USART1_IRQn, 5, 0);
        HAL_NVIC_EnableIRQ(USART1_IRQn);
    }

}
```

```
void MX_TIM2_Init(void)
{
	TIM_MasterConfigTypeDef sMasterConfig;
	TIM_OC_InitTypeDef sConfigOC;
	htim2.Instance = TIM2;
	htim2.Init.Prescaler = 7;
	htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
	htim2.Init.Period = 19999;
	htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
	htim2.Init.AutoReloadPreload =
TIM_AUTORELOAD_PRELOAD_DISABLE;
	if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
	{
		_Error_Handler(__FILE__, __LINE__);
	}

	 sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
	sMasterConfig.MasterSlaveMode =
TIM_MASTERSLAVEMODE_DISABLE;
	if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) !=
HAL_OK)
	{
		 _Error_Handler(__FILE__, __LINE__);
	}
	sConfigOC.OCMode = TIM_OCMODE_PWM1;
	sConfigOC.Pulse = 0;
	sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
	sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
```

```
        if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC,
TIM_CHANNEL_1) != HAL_OK)
        {
                _Error_Handler(__FILE__, __LINE__);
         }

         if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC,
TIM_CHANNEL_2) != HAL_OK)
        {
                _Error_Handler(__FILE__, __LINE__);
        }

        if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC,
TIM_CHANNEL_3) != HAL_OK)
        {
               _Error_Handler(__FILE__, __LINE__);
        }

        if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC,
TIM_CHANNEL_4) != HAL_OK)
        {
                _Error_Handler(__FILE__, __LINE__);
         }

        HAL_TIM_MspPostInit(&htim2);
}
```

Приложение А6 Конфигурация портов МК для каналов генерации ШИМ

```
void HAL_TIM_MspPostInit(TIM_HandleTypeDef* timHandle)
{
	GPIO_InitTypeDef GPIO_InitStruct;
	if(timHandle->Instance==TIM2)
	{	/**TIM2 GPIO Configuration
		PA0-WKUP	----→ TIM2_CH1
		PA1	----→ TIM2_CH2
		PA2	----→ TIM2_CH3
		PA3	----→ TIM2_CH4
		*/
		GPIO_InitStruct.Pin =
GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3;
		GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
		GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
		HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);



  }

}
```

```
void MX_GPIO_Init(void)
{
	GPIO_InitTypeDef GPIO_InitStruct;

	/* GPIO Ports Clock Enable */
	 __HAL_RCC_GPIOC_CLK_ENABLE();
	__HAL_RCC_GPIOD_CLK_ENABLE();
	__HAL_RCC_GPIOA_CLK_ENABLE();
	__HAL_RCC_GPIOB_CLK_ENABLE();

	/*Configure GPIO pin Output Level */
	HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);

	  /*Configure GPIO pin : PC13 */
	GPIO_InitStruct.Pin = GPIO_PIN_13;
	GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
	GPIO_InitStruct.Pull = GPIO_NOPULL;
	GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
	HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

	/*Configure GPIO pins : PC14 PC15 */
	GPIO_InitStruct.Pin = GPIO_PIN_14|GPIO_PIN_15;
	GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
	HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
	/*Configure GPIO pins : PA4 PA5 PA6 PA7
	                PA8 PA11 PA12 PA15 */
	GPIO_InitStruct.Pin = GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7
	            |GPIO_PIN_8|GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_15;
```

```
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    /*Configure GPIO pins : PB0 PB1 PB2 PB10
                PB11 PB12 PB13 PB14
                PB15 PB3 PB4 PB5
                PB8 PB9 */
    GPIO_InitStruct.Pin =
GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_10
                |GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14
                |GPIO_PIN_15|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5
                |GPIO_PIN_8|GPIO_PIN_9;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
}
```

Приложение А8 Функция записи в регистр датчика BMP280

```
static int write_register8(BMP280_HandleTypedef *dev, uint8_t addr, uint8_t value)
{
	uint16_t tx_buff;
	tx_buff = (dev->addr << 1);
	if (HAL_I2C_Mem_Write(&hi2c1, tx_buff, addr, 1, &value, 1, 10000) == HAL_OK)
		return false;
	else
		return true;
}
```

Приложение А9 Функция чтения калибровочных коэффициентов BMP280

```
static bool read_calibration_data(BMP280_HandleTypedef *dev)
{
	if (read_register16(dev, 0x88, &dev->dig_T1)
	&& read_register16(dev, 0x8a, (uint16_t *) &dev->dig_T2)
	&& read_register16(dev, 0x8c, (uint16_t *) &dev->dig_T3)
	&& read_register16(dev, 0x8e, &dev->dig_P1)
	&& read_register16(dev, 0x90, (uint16_t *) &dev->dig_P2)
	&& read_register16(dev, 0x92, (uint16_t *) &dev->dig_P3)
	&& read_register16(dev, 0x94, (uint16_t *) &dev->dig_P4)
	&& read_register16(dev, 0x96, (uint16_t *) &dev->dig_P5)
	&& read_register16(dev, 0x98, (uint16_t *) &dev->dig_P6)
	&& read_register16(dev, 0x9a, (uint16_t *) &dev->dig_P7)
	&& read_register16(dev, 0x9c, (uint16_t *) &dev->dig_P8)
	&& read_register16(dev, 0x9e, (uint16_t *) &dev->dig_P9))
{return true;}
		return false;
	}
```

```
bool bmp280_init(BMP280_HandleTypedef *dev)
{
	if (dev->addr != BMP280_I2C_ADDRESS_0
	&& dev->addr != BMP280_I2C_ADDRESS_1)
	{
		return false;
	}
	//мягкая перезагрузка
	if (write_register8(dev, BMP280_REG_RESET, BMP280_RESET_VALUE))
	{
		return false;
	}
		//ожидание копирования данных NVP
	while (1)
	{
		uint8_t status;
		if (!read_data(dev, BMP280_REG_STATUS, &status, 1)
		&& (status & 1) == 0)
			break;
	}
	//чтение калибровочных коэффициентов
	if (!read_calibration_data(dev))
	{
			return false;
	}
		//использование фильтра
		uint8_t config = (BMP280_STANDBY_250 << 5) |
(BMP280_FILTER_OFF << 2);
```

```
        if (write_register8(dev, BMP280_REG_CONFIG, config))
        {
                return false;
        }

        uint8_t ctrl = (BMP280_STANDARD << 5)
| (BMP280_STANDARD << 2) | (BMP280_MODE_NORMAL);
        if (write_register8(dev, BMP280_REG_CTRL, ctrl))
        {
                return false;
        }

        return true;

}
```

```
//функция записи байта
void writeByte(uint8_t address, uint8_t subAddress, uint8_t data)
{
uint8_t data_write[2];
data_write[0] = subAddress;
data_write[1] = data;
HAL_I2C_Master_Transmit(&hi2c1, address, data_write, 2, 1000);
}

//функция чтения байта
uint8_t readByte(uint8_t address, uint8_t subAddress)
{
uint8_t data[1];
uint8_t data_write[1];
data_write[0] = subAddress;
HAL_I2C_Master_Transmit(&hi2c1, address, data_write, 1, 1000);
HAL_I2C_Master_Receive(&hi2c1, address, data, 1, 1000);
return data[0];
}

//функция чтения нескольких байт
void readBytes(uint8_t address, uint8_t subAddress, uint8_t count, uint8_t *
dest)
{
uint8_t data[14];
uint8_t data_write[1];
data_write[0] = subAddress;
HAL_I2C_Master_Transmit(&hi2c1, address, data_write, 1, 1000);
HAL_I2C_Master_Receive(&hi2c1, address, data, count, 1000);
```

```
for(int ii = 0; ii < count; ii++) {
dest[ii] = data[ii];
}
}
//функция инициализации MPU9250
void initMPU9250(MPU9250_variables_HandleTypedef *dev)
{
writeByte(MPU9250_ADDRESS, PWR_MGMT_1, 0x80);
HAL_Delay(1);
writeByte(MPU9250_ADDRESS, PWR_MGMT_1, 0x00);
HAL_Delay(1);
writeByte(MPU9250_ADDRESS, PWR_MGMT_1, 0x01);
uint8_t c = readByte(MPU9250_ADDRESS, ACCEL_CONFIG);
c = c & ~0x18;
c = c | AFS_16G << 3;
writeByte(MPU9250_ADDRESS, ACCEL_CONFIG, c);
c = readByte(MPU9250_ADDRESS, ACCEL_CONFIG2);
c = c & ~0x0F;
c = c | 0x03;
writeByte(MPU9250_ADDRESS, ACCEL_CONFIG2, c);
writeByte(MPU9250_ADDRESS, INT_PIN_CFG, 0x22);
writeByte(MPU9250_ADDRESS, INT_ENABLE, 0x01);
uint8_t rawData[3];
//инициализация магнитометра
writeByte(AK8963_ADDRESS, AK8963_CNTL, 0x00);
HAL_Delay(1);
writeByte(AK8963_ADDRESS, AK8963_CNTL, 0x0F);
HAL_Delay(1);
readBytes(AK8963_ADDRESS, AK8963_ASAX, 3, &rawData[0]);
dev->magCalibrationX = (float)(rawData[0] - 128)/256.0f + 1.0f;
```

```
dev->magCalibrationY = (float)(rawData[1] - 128)/256.0f + 1.0f;
dev->magCalibrationZ = (float)(rawData[2] - 128)/256.0f + 1.0f;
writeByte(AK8963_ADDRESS, AK8963_CNTL, 0x00);
HAL_Delay(1);
writeByte(AK8963_ADDRESS, AK8963_CNTL, MFS_16BITS << 4 | 0x06);
HAL_Delay(1);
}

//функция калибровки MPU9250
void calibrateMPU9250(MPU9250_variables_HandleTypedef *dev)
{
uint8_t data[12];
uint16_t ii, packet_count, fifo_count;
int32_t accel_bias[3] = {0, 0, 0};
writeByte(MPU9250_ADDRESS, PWR_MGMT_1, 0x80);
HAL_Delay(1);
writeByte(MPU9250_ADDRESS, PWR_MGMT_1, 0x01);
writeByte(MPU9250_ADDRESS, PWR_MGMT_2, 0x00);
HAL_Delay(1);
writeByte(MPU9250_ADDRESS, INT_ENABLE, 0x00);
writeByte(MPU9250_ADDRESS, FIFO_EN, 0x00);
writeByte(MPU9250_ADDRESS, PWR_MGMT_1, 0x00);
writeByte(MPU9250_ADDRESS, I2C_MST_CTRL, 0x00);
writeByte(MPU9250_ADDRESS, USER_CTRL, 0x00);
writeByte(MPU9250_ADDRESS, USER_CTRL, 0x0C);
HAL_Delay(1);
writeByte(MPU9250_ADDRESS, CONFIG, 0x01);
writeByte(MPU9250_ADDRESS, SMPLRT_DIV, 0x00);
writeByte(MPU9250_ADDRESS, ACCEL_CONFIG, 0x00);
uint16_t  accelsensitivity = 16384;
```

```
writeByte(MPU9250_ADDRESS, USER_CTRL, 0x40);
writeByte(MPU9250_ADDRESS, FIFO_EN, 0x78);
HAL_Delay(1);
writeByte(MPU9250_ADDRESS, FIFO_EN, 0x00);
readBytes(MPU9250_ADDRESS, FIFO_COUNTH, 2, &data[0]);
fifo_count = ((uint16_t)data[0] << 8) | data[1];
packet_count = fifo_count/12;
for (ii = 0; ii < packet_count; ii++)
{
int16_t accel_temp[3] = {0, 0, 0};
readBytes(MPU9250_ADDRESS, FIFO_R_W, 12, &data[0]);
accel_temp[0] = (int16_t) (((int16_t)data[0] << 8) | data[1]  ) ;
accel_temp[1] = (int16_t) (((int16_t)data[2] << 8) | data[3]  ) ;
accel_temp[2] = (int16_t) (((int16_t)data[4] << 8) | data[5]  ) ;

accel_bias[0] += (int32_t) accel_temp[0];
accel_bias[1] += (int32_t) accel_temp[1];
accel_bias[2] += (int32_t) accel_temp[2];

}
accel_bias[0] /= (int32_t) packet_count;
accel_bias[1] /= (int32_t) packet_count;
accel_bias[2] /= (int32_t) packet_count;
if(accel_bias[2] > 0L) {accel_bias[2] -= (int32_t) accelsensitivity;}
else {accel_bias[2] += (int32_t) accelsensitivity;}
int32_t accel_bias_reg[3] = {0, 0, 0};
readBytes(MPU9250_ADDRESS, XA_OFFSET_H, 2, &data[0]);
accel_bias_reg[0] = (int16_t) ((int16_t)data[0] << 8) | data[1];
readBytes(MPU9250_ADDRESS, YA_OFFSET_H, 2, &data[0]);
accel_bias_reg[1] = (int16_t) ((int16_t)data[0] << 8) | data[1];
```

```
readBytes(MPU9250_ADDRESS, ZA_OFFSET_H, 2, &data[0]);
accel_bias_reg[2] = (int16_t) ((int16_t)data[0] << 8) | data[1];
uint32_t mask = 1uL;
uint8_t mask_bit[3] = {0, 0, 0};
for(ii = 0; ii < 3; ii++) {
if(accel_bias_reg[ii] & mask) mask_bit[ii] = 0x01;
}
accel_bias_reg[0] -= (accel_bias[0]/8);
accel_bias_reg[1] -= (accel_bias[1]/8);
accel_bias_reg[2] -= (accel_bias[2]/8);
data[0] = (accel_bias_reg[0] >> 8) & 0xFF;
data[1] = (accel_bias_reg[0])      & 0xFF;
data[1] = data[1] | mask_bit[0];
data[2] = (accel_bias_reg[1] >> 8) & 0xFF;
data[3] = (accel_bias_reg[1])      & 0xFF;
data[3] = data[3] | mask_bit[1];
data[4] = (accel_bias_reg[2] >> 8) & 0xFF;
data[5] = (accel_bias_reg[2])      & 0xFF;
data[5] = data[5] | mask_bit[2];
dev->accelBiasX = (float)accel_bias[0]/(float)accelsensitivity;
dev->accelBiasY = (float)accel_bias[1]/(float)accelsensitivity;
dev->accelBiasZ = (float)accel_bias[2]/(float)accelsensitivity;
}

void get_angles(MPU9250_variables_HandleTypedef *dev)
{
int16_t Ax, Ay, Az, Mx, My, Mz;
float ax, ay, az, mx, my, mz;
uint8_t rawData[6];
readBytes(MPU9250_ADDRESS, ACCEL_XOUT_H, 6, &rawData[0]);
```

```
Ax = (int16_t)(((int16_t)rawData[0] << 8) | rawData[1]) ;
Ay = (int16_t)(((int16_t)rawData[2] << 8) | rawData[3]) ;
Az = (int16_t)(((int16_t)rawData[4] << 8) | rawData[5]) ;
ax = (float) Ax * 0.00048828125f -  dev->accelBiasX;
ay = (float) Ay * 0.00048828125f -  dev->accelBiasY;
az = (float) Az * 0.00048828125f -  dev->accelBiasZ;
mx = (float) Mx * 1.4993895f * dev->magCalibrationX;
my = (float) My * 1.4993895f * dev->magCalibrationY;
mz = (float) Mz * 1.4993895f * dev->magCalibrationZ;
float Pitch = dev->pitch;
float Roll = dev->roll;
float Yaw = dev->yaw;
Pitch = (180/3.141592) * atan(ax / sqrt(pow(ay, 2) + pow(az,2)));
Roll = (180/3.141592) * atan(ay / sqrt(ax*ax + az*az));
float xh = mx * cos(Pitch) + mz * sin(Pitch);
float yh = mx * sin(Roll) * sin(Pitch) + my * cos(Roll) - mz * sin(Roll) *
cos(Pitch);
float zh = -(mx) * cos(Roll) * sin(Pitch) + my * sin(Roll) + mz * cos(Roll) *
cos(Pitch);
Yaw = (180/3.141592) * atan2(yh, xh);
if (yh < 0)    Yaw += 360;
dev->pitch = Pitch;
dev->roll = Roll;
dev->yaw = Yaw;
}
```

Приложение А12 Инициализация WiFi модуля ESP8266-12E

```
void ESP_Init(void){
	HAL_Delay(2500); //задержка 2.5с
	HAL_UART_Transmit_DMA(&huart1, "AT\r\n", 5);	//проверка, отвечает ли WiFi модуль
	HAL_Delay(25); //задержка 25мс
	HAL_UART_Transmit_DMA(&huart1, "ATE0\r\n", 9); //отключение эха
	HAL_Delay(25); //задержка 25мс
	HAL_UART_Transmit_DMA(&huart1, "AT+CWMODE=2\r\n", 14); //выбор режима точки доступа
	HAL_Delay(25); //задержка 25мс
	HAL_UART_Transmit_DMA(&huart1, "AT+CIPMODE=0\r\n", 15);
	//установка режима для множественного подключения
	HAL_Delay(25); //задержка 25мс
	HAL_UART_Transmit_DMA(&huart1, "AT+CIPMUX=1\r\n", 14);
//установка множественного подключения
	HAL_Delay(25); //задержка 25мс
	HAL_UART_Transmit_DMA(&huart1, "AT+CIPSERVER=1,88\r\n", 20);
//создание сервера с IP 192.168.4.1 и портом 88
	HAL_Delay(25); //задержка 25мс
	HAL_UART_Transmit_DMA(&huart1, "AT+CIPSTO=50\r\n", 15);
//установка таймаута 50с
	HAL_Delay(25); //задержка 25мс
}
```

```
//объявление очередей
//очередь для передачи заданной скорости из задачи поиска значения в задачу функциональных вычислений
xQueueHandle QueueSpeed;
//очередь для передачи данных, полученных с датчиков из задачи по приему данных в задачу функциональных вычислений
xQueueHandle QueueCalcData;
//очередь для передачи значений углов положения сервоприводов и тяги мотора из задачи функциональных вычислений в задачу по выработке управляющих команд
xQueueHandle QueueStabData;
//очередь для передачи сообщения, содержащего заданную скорость из прерывания в задачу по поиску значения
xQueueHandle xQueueSerialDataReceived;


//объявление задач
//задача по приему данных от датчиков
osThreadId DataHandle;
//задача функциональных вычислений
osThreadId CalcHandle;
//задача по выработке управляющих команд
osThreadId StabHandle;
//задача по поиску заданной скорости полета в принятом сообщении
osThreadId ParseHandle;
//создание прототипов функций задач
void TaskData(void const * argument); //задача по приему данных от датчиков
void TaskCalc(void const * argument); //задача функциональных вычислений
void TaskStab(void const * argument); //задача по выработке управляющих команд
```

```
void TaskParse(void const * argument); //задача по поиску заданной скорости полета в принятом сообщении

//прототип функции инициализации OCPB FreeRTOS

void MX_FREERTOS_Init(void);

//описание функции инициализации OCPB FreeRTOS

void MX_FREERTOS_Init(void)

{

	HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1); //запуск первого канала генерации ШИМ сигнала

	HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2); //запуск второго канала генерации ШИМ сигнала

	HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_3); //запуск третьего канала генерации ШИМ сигнала

	HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_4);  //запуск четвертого канала генерации ШИМ сигнала

	__HAL_UART_ENABLE_IT(&huart1, UART_IT_IDLE); //включение прерывания по UART

	HAL_UART_Receive_DMA(&huart1, DMA_BUFFER, DMA_BUFFER_SIZE); //запуск приема данных по UART с использованием DMA

	//инициализация датчика BMP280(статическое давление)

	bmp1.addr = BMP280_I2C_ADDRESS_1;

	bmp280_init(&bmp1);

	//инициализация датчика BMP280(полное давление)

	bmp0.addr = BMP280_I2C_ADDRESS_0;

	bmp280_init(&bmp0);

	//инициализация MPU9250

	uint16_t whoami = readByte(MPU9250_ADDRESS, WHO_AM_I_MPU9250); //проверка, отвечает ли модуль по своему адресу

	if (whoami == 0x71)

  {

		calibrateMPU9250(&mpu9250); //калибровка MPU9250
```

```
                HAL_Delay(1);
    //задержка 1мс
                initMPU9250(&mpu9250); //инициализация MPU9250
            }
            else
            {
                while(1) ; //в случае, если модуль не ответил на обращение по его адресу, программа останавливается
            }
            /* создание задач */
            //создание задачи по приему данных от датчиков
        osThreadDef(Data, TaskData, osPriorityNormal, 0, 512);
        DataHandle = osThreadCreate(osThread(Data), NULL);
            //создание задачи функциональных вычислений
        osThreadDef(Calc, TaskCalc, osPriorityNormal, 0, 512);
        CalcHandle = osThreadCreate(osThread(Calc), NULL);
            //создание задачи по выработке управляющих команд
        osThreadDef(Stab, TaskStab, osPriorityNormal, 0, 512);
        StabHandle = osThreadCreate(osThread(Stab), NULL);
            //создание задачи по поиску заданной скорости в принятом сообщении
        osThreadDef(Parse, TaskParse, osPriorityNormal, 0, 512);
        ParseHandle = osThreadCreate(osThread(Parse), NULL);
            /* Создание очередей */
            //очередь для передачи данных, полученных с датчиков из задачи по приему данных в задачу функциональных вычислений
            QueueCalcData = xQueueCreate(1, sizeof(GAP));
            //очередь для передачи значений углов положения сервоприводов и тяги мотора из задачи функциональных вычислений в задачу по выработке управляющих команд
            QueueStabData = xQueueCreate(1, sizeof(AnglesToStabilization));
```

```
        //очередь для передачи заданной скорости из задачи поиска
значения в задачу функциональных вычислений
        QueueSpeed = xQueueCreate(1, sizeof(SpeedSTR));
        //очередь для передачи сообщения, содержащего заданную скорость
из прерывания в задачу по поиску значения
        xQueueSerialDataReceived = xQueueCreate(1,
sizeof(DMA_BUFFER));
    }
```

```
//функция компенсанции температуры с помощью калибровочных коэффициентов
static inline int32_t compensate_temperature(BMP280_HandleTypedef *dev, int32_t adc_temp,    int32_t *fine_temp)
{
	int32_t var1, var2;
	var1 = ((((adc_temp >> 3) – ((int32_t) dev->dig_T1 << 1)))
			* (int32_t) dev->dig_T2) >> 11;
	var2 = (((((adc_temp >> 4) – (int32_t) dev->dig_T1)
			* ((adc_temp >> 4) – (int32_t) dev->dig_T1)) >> 12)
			* (int32_t) dev->dig_T3) >> 14;
	*fine_temp = var1 + var2;
	return (*fine_temp * 5 + 128) >> 8;
}
//функция компенсанции давления с помощью калибровочных коэффициентов
static inline uint32_t compensate_pressure(BMP280_HandleTypedef *dev, int32_t adc_press, int32_t fine_temp)
{
	int64_t var1, var2, p;
	var1 = (int64_t) fine_temp – 128000;
	var2 = var1 * var1 * (int64_t) dev->dig_P6;
	var2 = var2 + ((var1 * (int64_t) dev->dig_P5) << 17);
	var2 = var2 + (((int64_t) dev->dig_P4) << 35);
	var1 = ((var1 * var1 * (int64_t) dev->dig_P3) >> 8)
			+ ((var1 * (int64_t) dev->dig_P2) << 12);
	var1 = (((int64_t) 1 << 47) + var1) * ((int64_t) dev->dig_P1) >> 33;

	if (var1 == 0) {
		return 0;
	}
```

```
	p = 1048576 – adc_press;
	p = (((p << 31) – var2) * 3125) / var1;
	var1 = ((int64_t) dev->dig_P9 * (p >> 13) * (p >> 13)) >> 25;
	var2 = ((int64_t) dev->dig_P8 * p) >> 19;
	p = ((p + var1 + var2) >> 8) + ((int64_t) dev->dig_P7 << 4);
	return p;
}
//функция получения целочисленных данных с датчика
bool bmp280_read_fixed(BMP280_HandleTypedef *dev, int32_t *temperature,
uint32_t *pressure)
{
	int32_t adc_pressure;
	int32_t adc_temp;
	uint8_t data[8];
	if (read_data(dev, 0xf7, data, 6)) {
		return false;
	}
	adc_pressure = data[0] << 12 | data[1] << 4 | data[2] >> 4;
	adc_temp = data[3] << 12 | data[4] << 4 | data[5] >> 4;
	int32_t fine_temp;
	*temperature = compensate_temperature(dev, adc_temp, &fine_temp);
	*pressure = compensate_pressure(dev, adc_pressure, fine_temp);
	return true;
}
//функция получения данных в формате числа с плавающей точкой
bool bmp280_read_float(BMP280_HandleTypedef *dev, float *temperature, float
*pressure)
{
	int32_t fixed_temperature;
	uint32_t fixed_pressure;
```

```
    if (bmp280_read_fixed(dev, &fixed_temperature, &fixed_pressure))
    {
        *temperature = (float) fixed_temperature / 100;
        *pressure = (float) fixed_pressure / 256;
        return true;
    }
    return false;
}
```

Приложение А15 Функция генерации ШИМ сигналов

```
void set_pos(uint16_t value, uint8_t number)
{
	switch(number) //выбор ШИМ канала
	{
		case 1:	//канал для переднего сервопривода
		{
			if(value>180) value = 180;
			if(value<0) value = 0;
			uint16_t OnePercent = (SERVO_MAX - SERVO_MIN)/180;
			TIM2->CCR1 = SERVO_MIN + OnePercent*value;
//выработка ШИМ сигнала с заданной шириной импульса
			break;
		}
		case 2:	//канал для левого сервопривода
		{
			if(value>180) value = 180;
			if(value<0) value = 0;
			uint16_t OnePercent = (SERVO_MAX - SERVO_MIN)/180;
			TIM2->CCR2 = SERVO_MIN + OnePercent*value;
//выработка ШИМ сигнала с заданной шириной импульса
			break;
		}

		case 3:			//канал для правого сервопривода
		{
			if(value>180) value = 180;
			if(value<0) value = 0;
			uint16_t OnePercent = (SERVO_MAX - SERVO_MIN)/180;
			TIM2->CCR3 = SERVO_MIN + OnePercent*value;
//выработка ШИМ сигнала с заданной шириной импульса
```

```
                break;
            }

            case 4:  //канал для мотора
            {
                if(value > 100) value = 100;
                if(value < 0) value = 0;
                uint16_t OnePercent = (MOTOR_MAX - MOTOR_MIN)/100;
                TIM2->CCR4 = MOTOR_MIN + OnePercent*value;
//выработка ШИМ сигнала с заданной шириной импульса
                break;
            }
        }
}
```

Приложение А16 Функция обработки прерывания

```
void USART1_IRQHandler(void)
{
	static BaseType_t xHigherPriorityTaskWoken;	//переменная для отправки данных в очередь из прерывания
	//проверка, произошло ли прерывание
	if(USART1->SR&USART_SR_IDLE)
	{
		//чтение регистра DR
		(void)USART1->DR;
		DMA1_Channel5->CCR &= ~DMA_CCR_EN;  //отключение канала DMA
		DMA1_Channel5->CMAR = (uint32_t)DMA_BUFFER; //переустановка адреса буфера для приема сообщения
	  DMA1_Channel5->CNDTR = DMA_BUFFER_SIZE; //переустановка размера буфера
		//если сообщение начинается с символа "+", значит оно принято с компьютера
		if(DMA_BUFFER[2] == '+')
		{
			xHigherPriorityTaskWoken = pdFALSE; //установка значения pdFALSE для отправки данных в очередь из прерывания
			//копирование принятого сообщения в массив для отправки в очередь
			for(uint8_t i=0; i<25; i++)
				{
					DMA[i] = DMA_BUFFER[i+2];
				}
			//отправка данных в очередь
			xQueueSendFromISR(xQueueSerialDataReceived, &DMA, &xHigherPriorityTaskWoken);
		}
```

```
            //очистка буфера
            for(uint8_t i=0; i<25; i++)
                    {
                            DMA_BUFFER[i] = 0;
                    }
                    DMA1_Channel5->CCR |= DMA_CCR_EN; //включение
канала DMA
        }
  HAL_UART_IRQHandler(&huart1);
}
```

Приложение А17 Задача приема данных с датчиков

```
void TaskData(void const * argument)
{
	GAP toSend;		//создание экземпляра структуры для передачи давлений, температур и углов тангажа, крена и курса
	portBASE_TYPE xStatus;  //переменная, хранящая результат отправки данных в очередь
  for(;;)
  {
		// запуск WiFi модуля и мотора
		if(INIT_FLAG == 0)
		{
			ESP_Init();	//инициализация WiFi модуля, создание TCP/IP сервера
			set_pos(100,4);  //установка максимального значения тяги мотора
			HAL_Delay(2000); //задержка 2с
			set_pos(0,4);	//установка минимального значения тяги мотора
			HAL_Delay(6000); // задержка 6с
			INIT_FLAG = 1;	//установка значения INIT_FLAG = 1 для того, чтобы данный фрагмент кода больше не выполнялся
		}
		//чтение данных с BMP280 (полное давление) с проверкой на ошибку чтения
		// чтение данных с автоматической отправкой в структуру для отправки
    if(!bmp280_read_float(&bmp0, &toSend.Temperature0, &toSend.DynamicPressure))
		{
			size = sprintf((char *)Data,
				"Temperature/pressure reading from BMP280_0\n"); //сообщение об ошибке в случае её возникновения
```

```
                sprintf((char *)Data1,"AT+CIPSEND=0,%d\r\n", size);
                /* отправка сообщения посредством WiFi сигнала */
                HAL_UART_Transmit_DMA(&huart1, Data1, sizeof(Data1));
            //отправка команды на передачу данных
                HAL_Delay(100);
        //задержка
                HAL_UART_Transmit_DMA(&huart1, Data, sizeof(Data));
            //отправка сообщения
                HAL_Delay(100);
        //задержка
            }
            //чтение данных с BMP280 (статическое давление) с проверкой на ошибку чтения
            // чтение данных с автоматической отправкой в структуру для отправки
            if(!bmp280_read_float(&bmp1, &toSend.Temperature1, &toSend.StaticPressure))
            {
                size = sprintf((char *)Data,
                                "Temperature/pressure reading from BMP280_1\n");
//сообщение об ошибке в случае её возникновения
                sprintf((char *)Data1,"AT+CIPSEND=0,%d\r\n", size);
                /* отправка сообщения посредством WiFi сигнала */
                HAL_UART_Transmit_DMA(&huart1, Data1, sizeof(Data1));
            //отправка команды на передачу данных
                HAL_Delay(100);
        //задержка
                HAL_UART_Transmit_DMA(&huart1, Data, sizeof(Data));
            //отправка сообщения
                HAL_Delay(100);
        //задержка
```

```
        }
        //чтения данных с MPU9250
        if(readByte(MPU9250_ADDRESS, INT_STATUS) & 0x01) //проверка готовности данных
        {
            get_angles(&mpu9250);
                                //чтение данных с вычислением углов тангажа, крена и курса
        }

        /* заполнение структуры для отправки в очередь */
        toSend.Pitch = mpu9250.pitch;  //угол тангажа

        toSend.Roll = mpu9250.roll;          //угол крена
        toSend.Yaw = mpu9250.yaw;                //угол курса

        //отправка структуры данных в очередь
        xStatus = xQueueSendToBack(QueueCalcData, &toSend, 100 / portTICK_RATE_MS);
        if (xStatus == errQUEUE_FULL)
   {
                    size = sprintf((char *)Data,
                                "Queue from TaskData ERROR\n"); //сообщение об ошибке в случае её возникновения
            sprintf((char *)Data1,"AT+CIPSEND=0,%d\r\n", size);
            /* отправка сообщения посредством WiFi сигнала */
            HAL_UART_Transmit_DMA(&huart1, Data1, sizeof(Data1));
        //отправка команды на передачу данных
            HAL_Delay(100);

      //задержка
            HAL_UART_Transmit_DMA(&huart1, Data, sizeof(Data));
        //отправка сообщения
```

```
                HAL_Delay(100);
        //задержка
    }
            //передача управления планировщику
            taskYIELD();
  }
}
```

Приложение А18 задача функциональных вычислений

```
void TaskCalc(void const * argument)
{
  GAP toReceive;  // создание экземпляра структуры для приема данных с датчиков из очереди
    AnglesToStabilization toSend; //создание экземпляра структуры для передачи значений углов положения сервоприводов и тяги мотора
    portBASE_TYPE xStatus; //переменная, хранящая результат отправки данных в очередь
    SpeedSTR x; // создание экземпляра структуры для приема значения заданной скорости из очереди
    /* объявление переменных высоты, заданой скорости, текущей скорости, начального давления, полного давления, статического давления,
    плотности воздуха, температуры */
    static float Altitude, SetSpeed, CurSpeed, ZeroLevelPressure, FullPressure, StaticPressure, AirDensity, Temperature;
  for(;;)
  {
        // проверка, есть ли данные в очереди
        if(uxQueueMessagesWaiting(QueueCalcData)>0)
        {
            xQueueReceive(QueueCalcData, &toReceive, 0); //прием данных из очереди в случае когда они есть
        // получение давления на нулевой высоте
        if(INIT_PRESSURE_FLAG == 0)
        {
            ZeroLevelPressure = toReceive.StaticPressure; // первое значение давления, полученного с датчика статического давления
            INIT_PRESSURE_FLAG = 1; //установка значения INIT_PRESSURE_FLAG = 1 для того, чтобы данный фрагмент кода больше не выполнялся
        }
        }
```

```
//прием значения заданной скорости из очереди
xQueueReceive(QueueSpeed, &x, 0);
SetSpeed = x.speed;
// получение значений полного и статического давлений из очереди
FullPressure = toReceive.DynamicPressure;
StaticPressure = toReceive.StaticPressure;
//вычисление температуры окружающего воздуха в кельвинах
Temperature = toReceive.Temperature0 + 273.0f;
//вычисление плотности воздуха
AirDensity = (StaticPressure*29.0)/(8.314459848f*Temperature);
//вычисление высоты
Altitude = 44330.0f * (1.0f - pow(StaticPressure / ZeroLevelPressure,
0.1902949571836346f));
//вычисление текущей скорости
CurSpeed = sqrt((2*(FullPressure - StaticPressure))/(AirDensity));
//реализация режима висения
if(SetSpeed == 0)
{
    toSend.Pull = 50; //значение тяги двигателя
    toSend.StabF = toReceive.Pitch*(-1) + 90; //значение угла
переднего сервопривода
    toSend.StabR = toReceive.Roll*(-1) + 90; //значение угла
правого сервопривода
    toSend.StabL = toReceive.Roll + 90; //значение угла левого
сервопривода
}
//реализация алгоритма стабилизации скорости
if(CurSpeed<SetSpeed)
{
    toSend.StabL +=2; //значение угла левого сервопривода
    toSend.StabR +=2; //значение угла правого сервопривода
```

```
            toSend.StabF -=2; //значение угла переднего сервопривода
            toSend.Pull +=10; //значение тяги двигателя
        }
        else if(CurSpeed>SetSpeed)
        {
            toSend.StabL -=2; //значение угла левого сервопривода
            toSend.StabR -=2; //значение угла правого сервопривода
            toSend.StabF +=2; //значение угла переднего сервопривода
            toSend.Pull -=10; //значение тяги двигателя
        }
        //формирование сообщения, содержащего текущую высоту,
текущую и заданную скорости, углы тангажа, крена, курса
            size = sprintf((char *)Data,"%.2f %.2f %.2f %.2f %.2f %.2f\r\n",
Altitude, SetSpeed,
            CurSpeed, toReceive.Pitch, toReceive.Roll, toReceive.Yaw);

        sprintf((char *)Data1,"AT+CIPSEND=0,%d\r\n", size);
            // отправка сообщения по WiFi
        HAL_UART_Transmit_DMA(&huart1, Data1, sizeof(Data1));

        HAL_Delay(100);
        HAL_UART_Transmit_DMA(&huart1, Data, sizeof(Data));
        HAL_Delay(100);
        xStatus = xQueueSendToBack(QueueStabData, &toSend, 100 /
portTICK_RATE_MS);
        if (xStatus == errQUEUE_FULL)
    {
             size = sprintf((char *)Data,
                    "Queue from TaskCalc ERROR\n"); //сообщение об
ошибке в случае её возникновения
            sprintf((char *)Data1,"AT+CIPSEND=0,%d\r\n", size);
            /* отправка сообщения посредством WiFi сигнала */
```

```
                HAL_UART_Transmit_DMA(&huart1, Data1, sizeof(Data1));
            //отправка команды на передачу данных
                HAL_Delay(100);

        //задержка
                HAL_UART_Transmit_DMA(&huart1, Data, sizeof(Data));
            //отправка сообщения
                HAL_Delay(100);

        //задержка
    }
            //передача управления планировщику
            taskYIELD();
  }
}
```

Приложение А19 задача выработки управляющих команд

```
void TaskStab(void const * argument)
{
  AnglesToStabilization toReceive; //создание экземпляра структуры для приема значений углов поворота сервоприводов и тяги мотора
        uint8_t Pull;//значение тяги мотора
  for(;;)
  {
                //получение данных из очереди
    xQueueReceive(QueueStabData, &toReceive, 0);
                Pull = toReceive.Pull;
                //выработка ШИМ сигнала на сервоприводы и мотор
                set_pos(toReceive.StabF, 1);
                set_pos(toReceive.StabL, 2);
                set_pos(toReceive.StabR, 3);
                set_pos(toReceive.Pull, 4);
                //передача управления планировщику
                taskYIELD();
  }
}
```

Приложение А20 задача поиска заданной скорости в сообщении

```
void TaskParse(void const * argument)
{
  SpeedSTR toSend; //создание экземпляра структуры для передачи заданной скорости посредством очереди
	int fd, length;		//номер устройства, с которого пришло сообщение, длина сообщения
	float spd; //значение заданной скорости
  for(;;)
  {
		HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13); //мигание светодиодом
		//проверка, пришло ли сообщение из прерывание
		if(uxQueueMessagesWaitingFromISR(xQueueSerialDataReceived)>0)
		{
			//прием данных из очереди с прерывания
			xQueueReceive(xQueueSerialDataReceived,&(DMA),1);
			//проверка, является ли принятое сообщение сообщением с компьютера
			if(DMA[0] == '+' && DMA[1] == 'I' && DMA[2] == 'P' && DMA[3] == 'D')
			{
				//поиск значения заданной скорости
				if(sscanf((char *)DMA,"+IPD,%d,%d:%f", &fd, &length, &spd) > 0)
				{
					toSend.speed = spd;		//заполнение структуры для передачи в очередь
					//отправка данных в очередь
					xQueueSendToBack(QueueSpeed, &toSend, 100 / portTICK_RATE_MS);
				}
```

```
                }
            }
            //передача управления планировщику
            taskYIELD();
    }
}
```

Приложение А21 код приложения по визуализации результатов

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Globalization;

namespace TCPClient
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent(); //конфигурация приложения
         Milliseconds.Interval = 1;
         Milliseconds.Enabled = true;
         //настройки масштабирования графика скоростей
         chart1.ChartAreas[0].CursorX.IsUserEnabled = true;
         chart1.ChartAreas[0].CursorX.IsUserSelectionEnabled = true;
         chart1.ChartAreas[0].AxisX.ScaleView.Zoomable = true;
         chart1.ChartAreas[0].AxisX.ScrollBar.IsPositionedInside = true;


         chart1.ChartAreas[0].CursorY.IsUserEnabled = true;
         chart1.ChartAreas[0].CursorY.IsUserSelectionEnabled = true;
         chart1.ChartAreas[0].AxisY.ScaleView.Zoomable = true;
         chart1.ChartAreas[0].AxisY.ScrollBar.IsPositionedInside = true;
         //настройки масштабирования графика углов
         chart2.ChartAreas[0].CursorX.IsUserEnabled = true;
         chart2.ChartAreas[0].CursorX.IsUserSelectionEnabled = true;
         chart2.ChartAreas[0].AxisX.ScaleView.Zoomable = true;
         chart2.ChartAreas[0].AxisX.ScrollBar.IsPositionedInside = true;

         chart2.ChartAreas[0].CursorY.IsUserEnabled = true;
         chart2.ChartAreas[0].CursorY.IsUserSelectionEnabled = true;
         chart2.ChartAreas[0].AxisY.ScaleView.Zoomable = true;
```

```
            chart2.ChartAreas[0].AxisY.ScrollBar.IsPositionedInside = true;
            //настройки масштабирования графика высоты
            chart3.ChartAreas[0].CursorX.IsUserEnabled = true;
            chart3.ChartAreas[0].CursorX.IsUserSelectionEnabled = true;
            chart3.ChartAreas[0].AxisX.ScaleView.Zoomable = true;
            chart3.ChartAreas[0].AxisX.ScrollBar.IsPositionedInside = true;

            chart3.ChartAreas[0].CursorY.IsUserEnabled = true;
            chart3.ChartAreas[0].CursorY.IsUserSelectionEnabled = true;
            chart3.ChartAreas[0].AxisY.ScaleView.Zoomable = true;
            chart3.ChartAreas[0].AxisY.ScrollBar.IsPositionedInside = true;
        }
        int TimerCount = 0;
        private static Socket client;
        private static byte[] data = new byte[1024];

        //описание функции нажатия кнопки "Соединить"
        private void connect_Click(object sender, EventArgs e)
        {
            results.Items.Add("Connecting...");
            client = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
            IPEndPoint iep = new IPEndPoint(IPAddress.Parse("192.168.4.1"), 88);
            client.BeginConnect(iep, new AsyncCallback(Connected), client);
        }
        void Connected(IAsyncResult iar)
        {
            try
            {
                client.EndConnect(iar);
                results.Items.Add("Connected to:" + client.RemoteEndPoint.ToString());
                Thread receiver = new Thread(new ThreadStart(ReceiveData));
                receiver.Start();
            }
            catch (SocketException)
            {
                results.Items.Add("Error Connecting");
            }
        }
        //описание функции приема данных
        void ReceiveData()
        {
            //смена , на . при приеме данных, чтобы программа видела числа с
плавающей точкой
```

```
        IFormatProvider formatter = new NumberFormatInfo {
NumberDecimalSeparator = "." };
        //объявление переменных
        int recv;
        string StringData, s1, s2, s3, s4, s5, s6, s7;
        float Altitude, SetSpeed, CurSpeed, Pitch, Roll, Yaw;
        while (true)
        {
            //прием данных
            recv = client.Receive(data);
            StringData = Encoding.ASCII.GetString(data, 0, recv);
            //разделение полученного сообщения на несколько массивов по
признаку пробела
            string[] k = StringData.Split(' ');
            //получение численного значения высоты
            Altitude = float.Parse(k[0], formatter);
            //получение численного значения заданной скорости
            SetSpeed = float.Parse(k[1], formatter);
            //получение численного значения текущей скорости
            CurSpeed = float.Parse(k[2], formatter);
            //получение численного значения тангажа
            Pitch = float.Parse(k[3], formatter);
            //получение численного значения крена
            Roll = float.Parse(k[4], formatter);
            //получение численного значения курса
            Yaw = float.Parse(k[5], formatter);
            s1 = Altitude.ToString();
            s2 = SetSpeed.ToString();
            s3 = CurSpeed.ToString();
            s4 = Pitch.ToString();
            s5 = Roll.ToString();
            s6 = Yaw.ToString();
            //показ полученных значений на главной странице
            results.Items.Add("Высота: " + s1 + " Vзад: " + s2 + " Vтек: " + s3 + "
Тангаж: " + s4 + " Крен:" + s5 + " Курс:" + s6);

            //построение графика скоростей
            chart1.Series[0].Points.AddXY(TimerCount, SetSpeed);
            chart1.Series[1].Points.AddXY(TimerCount, CurSpeed);
            chart1.ChartAreas[0].AxisX.ScaleView.Zoom(TimerCount-1000,
TimerCount);
            //построение графика углов
            chart2.Series[0].Points.AddXY(TimerCount, Pitch);
            chart2.Series[1].Points.AddXY(TimerCount, Roll);
```

```
            chart2.Series[2].Points.AddXY(TimerCount, Yaw);
            chart2.ChartAreas[0].AxisX.ScaleView.Zoom(TimerCount-1000,
TimerCount);
            //построение графика высоты
            chart3.Series[0].Points.AddXY(TimerCount, Altitude);
            chart3.ChartAreas[0].AxisX.ScaleView.Zoom(TimerCount - 1000,
TimerCount);
            results.TopIndex = results.Items.Count - 1;
        }
    }

    //описание функции нажатия на кнопку "Отключение"
    private void exit_Click(object sender, EventArgs e)
    {
        client.Close();
        results.Items.Add("Connection stopped");
        Close();
    }

    //описание функции обновления дисплея
    private delegate void DlDisplay(string s);
    private void Display(string s)
    {
        if (results.InvokeRequired)
        {
            DlDisplay sd = new DlDisplay(Display);
            results.Invoke(sd, new object[] { s });
        }
        else
        {
            results.Items.Add(s);
        }
    }

    //описание функции нажатия на кнопку "Отправить"
    private void sendit_Click(object sender, EventArgs e)
    {
        byte[] message = Encoding.ASCII.GetBytes(newText.Text);
        newText.Clear();
        client.BeginSend(message, 0, message.Length, 0, new
AsyncCallback(SendData), client);
    }

    //функция отправки данных
```

```
        void SendData(IAsyncResult iar)
        {
            Socket remote = (Socket)iar.AsyncState;
            int sent = remote.EndSend(iar);
        }

        //функция подстчета количества тактов для получения времени
        private void timer1_Tick(object sender, EventArgs e)
        {
            TimerCount += 1;
        }

        //функция отмены масштабирования при двойном нажатии правой кнопки
мыши для графика скоростей
        private void chart1_DoubleClick(object sender, EventArgs e)
        {
            chart1.ChartAreas[0].AxisX.ScaleView.ZoomReset(0);
            chart1.ChartAreas[0].AxisY.ScaleView.ZoomReset(0);
        }

        //функция отмены масштабирования при двойном нажатии правой кнопки
мыши для графика углов
        private void chart2_DoubleClick(object sender, EventArgs e)
        {
            chart2.ChartAreas[0].AxisX.ScaleView.ZoomReset(0);
            chart2.ChartAreas[0].AxisY.ScaleView.ZoomReset(0);
        }

        //функция отмены масштабирования при двойном нажатии правой кнопки
мыши для графика высоты
        private void chart3_DoubleClick(object sender, EventArgs e)
        {
            chart3.ChartAreas[0].AxisX.ScaleView.ZoomReset(0);
            chart3.ChartAreas[0].AxisY.ScaleView.ZoomReset(0);
        }


    }


}
```

Приложение А22 Полный код тестовой программы

```
/* Подключение библиотек */
#include "FreeRTOS.h"
#include "task.h"
#include "main.h"
#include "cmsis_os.h"
#include "bmp280.h"
#include "mpu9250.h"
#include <math.h>
#include "esp8266.h"
#include "usart.h"
#include "dma.h"
#include <string.h>
#include "tim.h"

//_____________________________________________________________________//

/* создание структур для хранения данных, принятых с датчиков,
а также объявление структур для передачи данных в очередях */

MPU9250_variables_HandleTypedef mpu9250; // структура для хранения данных с MPU9250
BMP280_HandleTypedef bmp0; // структура для хранения данных с BMP280 (полное давление)
BMP280_HandleTypedef bmp1; // структура для хранения данных с BMP280 (статическое давление)

// структура для передачи заданной скорости посредством очереди
typedef struct
{
	float speed;
```

```
}SpeedSTR;

// структура для передачи полного и статического давлений, температуры, углок тангажа, круна и курса посредством очереди
typedef struct
{
	float DynamicPressure, Temperature1, StaticPressure, Temperature0;
	float Pitch, Roll, Yaw;
}GAP;

// структура для передачи тяги и углов положения сервоприводов
typedef struct
{
	uint8_t StabF, StabL, StabR, Pull;
}AnglesToStabilization;



#define DMA_BUFFER_SIZE 25 //размер буфера для приема данных с компьютера
#define SERVO_MIN 750	//минимальное значение ширины импульса для сервопривода в микросекундах
#define SERVO_MAX 2700		//максимальное значение ширины импульса для сервопривода в микросекундах
#define MOTOR_MIN 850		//минимальное значение ширины импульса для мотора в микросекундах
#define MOTOR_MAX 2250	//максимальное значение ширины импульса для мотора в микросекундах

//INIT_FLAG - флаг для запуска мотора и WiFi модуля, INIT_PRESSURE_FLAG - флаг для записи давления на нулевой высоте
volatile uint8_t INIT_FLAG, INIT_PRESSURE_FLAG;
//переменная - счетчик ошибок
```

```
volatile uint16_t ERROR_VAR;
//буфер для сообщений, полученных с компьютера
uint8_t DMA_BUFFER[DMA_BUFFER_SIZE];
//буфер для передачи сообщения с заданной скоростью посредством очереди
uint8_t DMA[DMA_BUFFER_SIZE];
/**

*Data1 - массив для хранения AT команды по отправке данных через WiFi модуль
*Data - массив, содержащий данные для передачи по WiFi
*size - размер сообщения для передачи по WiFi

**/
uint16_t size;
uint8_t Data[256];
uint8_t Data1[25];

//объявление очередей
//очередь для передачи заданной скорости из задачи поиска значения в задачу функциональных вычислений
xQueueHandle QueueSpeed;
//очередь для передачи данных, полученных с датчиков из задачи по приему данных в задачу функциональных вычислений
xQueueHandle QueueCalcData;
//очередь для передачи значений углов положения сервоприводов и тяги мотора из задачи функциональных вычислений в задачу по выработке управляющих команд
xQueueHandle QueueStabData;
//очередь для передачи сообщения, содержащего заданную скорость из прерывания в задачу по поиску значения
xQueueHandle xQueueSerialDataReceived;
```

```
//объявление задач
//задача по приему данных от датчиков
osThreadId DataHandle;
//задача функциональных вычислений
osThreadId CalcHandle;
//задача по выработке управляющих команд
osThreadId StabHandle;
//задача по поиску заданной скорости полета в принятом сообщении
osThreadId ParseHandle;

//функция выработки ШИМ сигнала
void set_pos(uint16_t value, uint8_t number)
{
	switch(number) //выбор ШИМ канала
	{
		case 1:	//канал для переднего сервопривода
		{
			if(value>180) value = 180;
			if(value<0) value = 0;
			uint16_t OnePercent = (SERVO_MAX - SERVO_MIN)/180;
			TIM2->CCR1 = SERVO_MIN + OnePercent*value;
//выработка ШИМ сигнала с заданной шириной импульса
			break;
		}
		case 2:	//канал для левого сервопривода
		{
			if(value>180) value = 180;
			if(value<0) value = 0;
			uint16_t OnePercent = (SERVO_MAX - SERVO_MIN)/180;
```

```
                TIM2->CCR2 = SERVO_MIN + OnePercent*value;
//выработка ШИМ сигнала с заданной шириной импульса
                break;
            }

            case 3:             //канал для правого сервопривода
            {
                if(value>180) value = 180;
                if(value<0) value = 0;
                uint16_t OnePercent = (SERVO_MAX - SERVO_MIN)/180;
                TIM2->CCR3 = SERVO_MIN + OnePercent*value;
//выработка ШИМ сигнала с заданной шириной импульса
                break;
            }

            case 4:  //канал для мотора
            {
                if(value > 100) value = 100;
                if(value < 0) value = 0;
                uint16_t OnePercent = (MOTOR_MAX - MOTOR_MIN)/100;
                TIM2->CCR4 = MOTOR_MIN + OnePercent*value;
//выработка ШИМ сигнала с заданной шириной импульса
                break;
            }
        }
}

//функция обработки прерывания по ошибке UART
void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart)
{
    if(huart == &huart1)
```

```
        {
                HAL_UART_Receive_DMA(&huart1, DMA_BUFFER,
DMA_BUFFER_SIZE);          //перезапуск приема данных
        }
}

//создание прототипов функций задач
void TaskData(void const * argument); //задача по приему данных от датчиков
void TaskCalc(void const * argument); //задача функциональных вычислений
void TaskStab(void const * argument); //задача по выработке управляющих
команд
void TaskParse(void const * argument); //задача по поиску заданной скорости
полета в принятом сообщении

//прототип функции инициализации ОСРВ FreeRTOS
void MX_FREERTOS_Init(void);

//описание функции инициализации ОСРВ FreeRTOS
void MX_FREERTOS_Init(void)
{
        HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1); //запуск первого
канала генерации ШИМ сигнала
        HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2); //запуск второго
канала генерации ШИМ сигнала
        HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_3); //запуск третьего
канала генерации ШИМ сигнала
        HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_4);  //запуск четвертого
канала генерации ШИМ сигнала
        __HAL_UART_ENABLE_IT(&huart1, UART_IT_IDLE); //включение
прерывания по UART
        HAL_UART_Receive_DMA(&huart1, DMA_BUFFER,
DMA_BUFFER_SIZE); //запуск приема данных по UART с использованием
DMA
```

```
	//инициализация датчика BMP280(статическое давление)
	bmp1.addr = BMP280_I2C_ADDRESS_1;
	//инициализация датчика BMP280(полное давление)
	if(!bmp280_init(&bmp1))
	{
		ERROR_VAR++; //инкрементация счетчика ошибок
	}
	bmp0.addr = BMP280_I2C_ADDRESS_0;
	if(!bmp280_init(&bmp0))
	{
		ERROR_VAR++; //инкрементация счетчика ошибок
	}
	//инициализация MPU9250
	uint16_t whoami = readByte(MPU9250_ADDRESS,
WHO_AM_I_MPU9250); //проверка, отвечает ли модуль по своему адресу
	if (whoami == 0x71)
 {
		calibrateMPU9250(&mpu9250); //калибровка MPU9250
		HAL_Delay(1);
	//задержка 1мс
		initMPU9250(&mpu9250);		//инициализация MPU9250
	}
	else
	{
		ERROR_VAR++; //инкрементация счетчика ошибок
		while(1);		//в случае, если модуль не ответил на обращение
по его адресу, программа останавливается
	}

	/* создание задач */
	//создание задачи по приему данных от датчиков
```

```
  osThreadDef(Data, TaskData, osPriorityNormal, 0, 512);
  DataHandle = osThreadCreate(osThread(Data), NULL);
        //создание задачи функциональных вычислений
  osThreadDef(Calc, TaskCalc, osPriorityNormal, 0, 512);
  CalcHandle = osThreadCreate(osThread(Calc), NULL);
        //создание задачи по выработке управляющих команд
  osThreadDef(Stab, TaskStab, osPriorityNormal, 0, 512);
  StabHandle = osThreadCreate(osThread(Stab), NULL);
        //создание задачи по поиску заданной скорости в принятом сообщении
  osThreadDef(Parse, TaskParse, osPriorityNormal, 0, 512);
  ParseHandle = osThreadCreate(osThread(Parse), NULL);


        /* Создание очередей */


        //очередь для передачи данных, полученных с датчиков из задачи по приему данных в задачу функциональных вычислений
        QueueCalcData = xQueueCreate(1, sizeof(GAP));
        //очередь для передачи значений углов положения сервоприводов и тяги мотора из задачи функциональных вычислений в задачу по выработке управляющих команд
        QueueStabData = xQueueCreate(1, sizeof(AnglesToStabilization));
        //очередь для передачи заданной скорости из задачи поиска значения в задачу функциональных вычислений
        QueueSpeed = xQueueCreate(1, sizeof(SpeedSTR));
        //очередь для передачи сообщения, содержащего заданную скорость из прерывания в задачу по поиску значения
        xQueueSerialDataReceived= xQueueCreate(1, sizeof(DMA_BUFFER));
}


/* Описание функций задач */
//описание задачи по приему данных от датчков
```

```
void TaskData(void const * argument)
{
	GAP toSend;		//создание экземпляра структуры для передачи давлений, температур и углов тангажа, креа и курса
	portBASE_TYPE xStatus;  //переменная, хранящая результат отправки данных в очередь
  for(;;)
  {
		// запуск WiFi модуля и мотора
		if(INIT_FLAG == 0)
		{
			ESP_Init();	//инициализция WiFi модуля, создание TCP/IP сервера
			set_pos(100,4);  //установка максимального значения тяги мотора
			HAL_Delay(2000); //задержка 2с
			set_pos(0,4);	//установка минимального значения тяги мотора
			HAL_Delay(6000); // задержка 6с
			INIT_FLAG = 1;	//установка значения INIT_FLAG = 1 для того, чтобы данный фрагмент кода больше не выполнялся
		}
		//чтение данных с BMP280 (полное давление) с проверкой на ошибку чтения
		// чтение данных с автоматической отправкой в структуру для отправки
    if(!bmp280_read_float(&bmp0, &toSend.Temperature0, &toSend.DynamicPressure))
		{
			ERROR_VAR++; //инкрементация счетчика ошибок
			size = sprintf((char *)Data,
				"Temperature/pressure reading from BMP280_0\n"); //сообщение об ошибке в случае её возникновения
```

```
                sprintf((char *)Data1,"AT+CIPSEND=0,%d\r\n", size);
                /* отправка сообщения посредством WiFi сигнала */
                HAL_UART_Transmit_DMA(&huart1, Data1, sizeof(Data1));
            //отправка команды на передачу данных
                HAL_Delay(100);

        //задержка
                HAL_UART_Transmit_DMA(&huart1, Data, sizeof(Data));
            //отправка сообщения
                HAL_Delay(100);

        //задержка
            }
            //чтение данных с BMP280 (статическое давление) с проверкой на ошибку чтения
            // чтение данных с автоматической отправкой в структуру для отправки
            if(!bmp280_read_float(&bmp1, &toSend.Temperature1, &toSend.StaticPressure))
            {
                ERROR_VAR++; //инкрементация счетчика ошибок
                size = sprintf((char *)Data,
                                "Temperature/pressure reading from BMP280_1\n"); //сообщение об ошибке в случае её возникновения
                sprintf((char *)Data1,"AT+CIPSEND=0,%d\r\n", size);
                /* отправка сообщения посредством WiFi сигнала */
                HAL_UART_Transmit_DMA(&huart1, Data1, sizeof(Data1));
            //отправка команды на передачу данных
                HAL_Delay(100);

        //задержка
                HAL_UART_Transmit_DMA(&huart1, Data, sizeof(Data));
            //отправка сообщения
```

```
                HAL_Delay(100);

        //задержка

                }

                //чтения данных с MPU9250

                if(readByte(MPU9250_ADDRESS, INT_STATUS) & 0x01) //проверка
готовности данных

                {

                        get_angles(&mpu9250);
                                        //чтение данных с вычислением углов
тангажа, крена и курса

                }


                /* заполнение структуры для отправки в очередь */

                toSend.Pitch = mpu9250.pitch; //угол тангажа

                toSend.Roll = mpu9250.roll; //угол крена

                toSend.Yaw = mpu9250.yaw; //угол курса

                //отправка структуры данных в очередь

                xStatus = xQueueSendToBack(QueueCalcData, &toSend, 100 /
portTICK_RATE_MS);

                if (xStatus == errQUEUE_FULL)

   {

                        ERROR_VAR++; //инкрементация счетчика ошибок

                        size = sprintf((char *)Data, "Queue from TaskData ERROR\n");
//сообщение об ошибке в случае её возникновения

                        sprintf((char *)Data1,"AT+CIPSEND=0,%d\r\n", size);

                        /* отправка сообщения посредством WiFi сигнала */

                        HAL_UART_Transmit_DMA(&huart1, Data1, sizeof(Data1));
                //отправка команды на передачу данных

                        HAL_Delay(100); //задержка

                        HAL_UART_Transmit_DMA(&huart1, Data, sizeof(Data));
                //отправка сообщения

                        HAL_Delay(100); //задержка
```

```
    }

//раскомментировать в процессе выполнения теста 1
//              size = sprintf((char *)Data,"TASK 1\r\n");
//              sprintf((char *)Data1,"AT+CIPSEND=0,%d\r\n", size);
//              HAL_UART_Transmit(&huart1, Data1, sizeof(Data1), 100);
//              HAL_Delay(20);
//              HAL_UART_Transmit(&huart1, Data, sizeof(Data), 100);
//              HAL_Delay(20);
                //передача управления планировщику
                taskYIELD();
  }
}

// описание функции задачи функциональных вычислений
void TaskCalc(void const * argument)
{
  GAP toReceive;  // создание экземпляра структуры для приема данных с датчиков из очереди
        AnglesToStabilization toSend; //создание экземпляра структуры для передачи значений углов положения сервоприводов и тяги мотора
        portBASE_TYPE xStatus; //переменная, хранящая результат отправки данных в очередь
        SpeedSTR x; // создание экземпляра структуры для приема значения заданной скорости из очереди
        /* объявление переменных высоты, заданой скорости, текущей скорости, начального давления, полного давления, статического давления,
        плотности воздуха, температуры */
        static float Altitude, SetSpeed, CurSpeed, ZeroLevelPressure, FullPressure, StaticPressure, AirDensity, Temperature;
  for(;;)
  {
```

```
// проверка, есть ли данные в очереди
if(uxQueueMessagesWaiting(QueueCalcData)>0)
{
    xQueueReceive(QueueCalcData, &toReceive, 0); //прием данных из очереди в случае когда они есть
// получение давления на нулевой высоте
if(INIT_PRESSURE_FLAG == 0)
{
    ZeroLevelPressure = toReceive.StaticPressure; // первое значение давления, полученного с датчика статического давления
    INIT_PRESSURE_FLAG = 1; //установка значения INIT_PRESSURE_FLAG = 1 для того, чтобы данный фрагмент кода больше не выполнялся
}
}
//прием значения заданной скорости из очереди
xQueueReceive(QueueSpeed, &x, 0);
SetSpeed = x.speed;
// получение значений полного и статического давлений из очереди
FullPressure = toReceive.DynamicPressure;
StaticPressure = toReceive.StaticPressure;
//вычисление температуры окружающего воздуха в кельвинах
Temperature = toReceive.Temperature0 + 273.0f;
//вычисление плотности воздуха
AirDensity = (StaticPressure*29.0)/(8.314459848f*Temperature);
//вычисление высоты
Altitude = 44330.0f * (1.0f - pow(StaticPressure / ZeroLevelPressure, 0.1902949571836346f));
//вычисление текущей скорости
CurSpeed = sqrt((2*(FullPressure - StaticPressure))/(AirDensity));
//реализация режима висения
```

```
if(SetSpeed == 0)
{
	toSend.Pull = 50; //значение тяги двигателя
	toSend.StabF = toReceive.Pitch*(-1) + 90; //значение угла переднего сервопривода
	toSend.StabR = toReceive.Roll*(-1) + 90; //значение угла правого сервопривода
	toSend.StabL = toReceive.Roll + 90; //значение угла левого сервопривода
}
//реализация алгоритма стабилизации скорости
if(CurSpeed<SetSpeed)
{
	toSend.StabL +=2; //значение угла левого сервопривода
	toSend.StabR +=2; //значение угла правого сервопривода
	toSend.StabF -=2; //значение угла переднего сервопривода
	toSend.Pull +=10; //значение тяги двигателя
}
else if(CurSpeed>SetSpeed)
{
	toSend.StabL -=2; //значение угла левого сервопривода
	toSend.StabR -=2; //значение угла правого сервопривода
	toSend.StabF +=2; //значение угла переднего сервопривода
	toSend.Pull -=10; //значение тяги двигателя
}
//формирование сообщения, содержащего текущую высоту, текущую и заданную скорости, углы тангажа, крена, курса
	size = sprintf((char *)Data,"%.2f %.2f %.2f %.2f %.2f %.2f\r\n", Altitude, SetSpeed,
	CurSpeed, toReceive.Pitch, toReceive.Roll, toReceive.Yaw);

sprintf((char *)Data1,"AT+CIPSEND=0,%d\r\n", size);
```

```
                // отправка сообщения по WiFi
            HAL_UART_Transmit_DMA(&huart1, Data1, sizeof(Data1));

            HAL_Delay(100);
            HAL_UART_Transmit_DMA(&huart1, Data, sizeof(Data));
            HAL_Delay(100);
            xStatus = xQueueSendToBack(QueueStabData, &toSend, 100 /
portTICK_RATE_MS);
            if (xStatus == errQUEUE_FULL)
    {
                ERROR_VAR++; //инкрементация счетчика ошибок
                size = sprintf((char *)Data,
                        "Queue from TaskCalc ERROR\n"); //сообщение об
ошибке в случае её возникновения
                sprintf((char *)Data1,"AT+CIPSEND=0,%d\r\n", size);
                /* отправка сообщения посредством WiFi сигнала */
                HAL_UART_Transmit_DMA(&huart1, Data1, sizeof(Data1));
            //отправка команды на передачу данных
                HAL_Delay(100); //задержка
                HAL_UART_Transmit_DMA(&huart1, Data, sizeof(Data));
            //отправка сообщения
                HAL_Delay(100); //задержка
    }

//раскомментировать в процессе выполнения теста 1
//          size = sprintf((char *)Data,"TASK 2\r\n");
//          sprintf((char *)Data1,"AT+CIPSEND=0,%d\r\n", size);
//          HAL_UART_Transmit(&huart1, Data1, sizeof(Data1), 100);
//          HAL_Delay(20);
//          HAL_UART_Transmit(&huart1, Data, sizeof(Data), 100);
//          HAL_Delay(20);
```

```
                //передача управления планировщику
                taskYIELD();
  }
}
//описание функции задачи выработки управляющих команд
void TaskStab(void const * argument)
{
  AnglesToStabilization toReceive; //создание экземпляра структуры для приема значений углов поворота сервоприводов и тяги мотора
        uint8_t Pull;//значение тяги мотора
  for(;;)
  {
                //получение данных из очереди
    xQueueReceive(QueueStabData, &toReceive, 0);
                Pull = toReceive.Pull;
                //выработка ШИМ сигнала на сервоприводы и мотор
                set_pos(toReceive.StabF, 1);
                set_pos(toReceive.StabL, 2);
                set_pos(toReceive.StabR, 3);
                set_pos(toReceive.Pull, 4);
//раскомментировать в процессе выполнения теста 1
//              size = sprintf((char *)Data,"TASK 3\r\n");
//              sprintf((char *)Data1,"AT+CIPSEND=0,%d\r\n", size);
//              HAL_UART_Transmit(&huart1, Data1, sizeof(Data1), 100);
//              HAL_Delay(20);
//              HAL_UART_Transmit(&huart1, Data, sizeof(Data), 100);
//              HAL_Delay(20);
                //передача управления планировщику
                taskYIELD();
  }
```

```
}
//описани функции задачи поиска заданного значения скорости в сообщении
void TaskParse(void const * argument)
{
  SpeedSTR toSend; //создание экземпляра структуры для передачи заданной скорости посредством очереди
        int fd, length;         //номер устройства, с которого пришло сообщение, длина сообщения
        float spd; //значение заданной скорости
  for(;;)
  {
                HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13); //мигание светодиодом
                //проверка, пришло ли сообщение из прерывание
                if(uxQueueMessagesWaitingFromISR(xQueueSerialDataReceived)>0)
                {
                        //прием данных из очереди с прерывания
                        xQueueReceive(xQueueSerialDataReceived,&(DMA),1);
                        //проверка, является ли принятое сообщение сообщением с компьютера
                        if(DMA[0] == '+' && DMA[1] == 'I' && DMA[2] == 'P' && DMA[3] == 'D')
                        {
                                //поиск значения заданной скорости
                                if(sscanf((char *)DMA,"+IPD,%d,%d:%f", &fd, &length, &spd) > 0)
                                {
                                        toSend.speed = spd;         //заполнение структуры для передачи в очередь
                                        //отправка данных в очередь
                                        xQueueSendToBack(QueueSpeed, &toSend, 100 / portTICK_RATE_MS);
```

```
                    }
                }
            }
                    //раскомментировать в процессе выполнения теста 1
//          size = sprintf((char *)Data,"TASK 4\r\n");
//          sprintf((char *)Data1,"AT+CIPSEND=0,%d\r\n", size);
//          HAL_UART_Transmit(&huart1, Data1, sizeof(Data1), 100);
//          HAL_Delay(20);
//          HAL_UART_Transmit(&huart1, Data, sizeof(Data), 100);
//          HAL_Delay(20);
            //передача управления планировщику
            taskYIELD();
  }
}
```