МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Некоммерческое акционерное общество
АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ

кафедра Компьютерных технологий

«Допущен к защите»
Заведующий кафедрой Куралбаев З.К
профессор, д.ф-м.н
(Ф.И.О., ученая степень, звание)
_____ «_____»_____ **20__ г.**
(подпись)

ДИПЛОМНЫЙ ПРОЕКТ

На тему: Разработка блока управления летательного аппарата на основе микроконтроллера Arducopter

Специальность Вычислительная техника и программное обеспечение

Выполнил (а) Осипов Д.Б ВТ-12-4
(Фамилия и инициалы) группа

Научный руководитель Куралбаев З.К. профессор, д.ф-м.н
(Фамилия и инициалы, ученая степень, звание)

Консультанты:

по экономической части:
Бекишева Анна Ивановна к.э.н, доцент
(Фамилия и инициалы, ученая степень, звание)
_____ «29» 04 2016 г.
(подпись)

по безопасности жизнедеятельности:
Назаров Иван Федорович к.х.н, доцент
(Фамилия и инициалы, ученая степень, звание)
_____ «4» 05 2016 г.
(подпись)

по применению вычислительной техники:
Куралбаев З.К профессор, д.ф-м.н
(Фамилия и инициалы, ученая степень, звание)
_____ «19» 05 2016 г.
(подпись)

Нормоконтролер: Куралбаев З.К. профессор, д.ф-м.н
(Фамилия и инициалы, ученая степень, звание)
_____ «19» 05 2016 г.
(подпись)

Рецензент: _____
(Фамилия и инициалы, ученая степень, звание)
_____ «_____»_____ 20___ г.
(подпись)

Алматы 2016 г.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Некоммерческое акционерное общество
АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ

Факультет Аэрокосмических и информационных технологий
Специальность Вычислительная техника и программное обеспечение
Кафедра Компьютерных технологий

ЗАДАНИЕ

на выполнение дипломного проекта

Студент Осипов Дмитрий Борисович
(фамилия, имя, отчество)

Тема проекта Разработка блока управления летательного аппарата на основе микроконтролера tidecopter

утверждена приказом ректора № 148 от «19» октября 2015 г.
Срок сдачи законченной работы «___» _____ 20___ г.
Исходные данные к проекту требуемые параметры результатов проектирования (исследования) и исходные данные объекта
разработать блок управления летательным аппаратом на основе микроконтролера tidecopter

Перечень подлежащих разработке дипломного проекта вопросов или краткое содержание дипломного проекта:
1) Разобрать принцип работы компонентов летательного аппарата.
2) Выполнить анализ требований к летательному аппарату и блоку управления.
3) Рассчитать параметры летательного аппарата и блока управления
4) Подобрать комплектующие к квадрокоптеру.
5) Разработать алгоритм стабилизации квадрокоптера
6) Написать прошивку для квадрокоптера.
7) Собрать летательный аппарат.
8) Провести тестирование и анализ тестовых полетов квадрокоптера.
9) Представить финальную сборку квадрокоптера на основании проведенной доработки.

Перечень графического материала (с точным указанием обязательных чертежей)

8 рисунков в главе 1
9 рисунков в главе 2
51 рисунок в главе 3
3 рисунка в разделе БЖД
0 рисунков в разделе Экономика

Рекомендуемая основная литература

K.N. King. C Programming: t modern tpproach
Stephen Kochan. Programming in C
John Boxall. tduino workshop
John-David Warren. tduino Robotics

Консультанты по проекту с указанием относящихся к ним разделов

Раздел	Консультант	Сроки	Подпись
Экономика	Бекмиева Ж.М.	1.03-7.05.2016	
БЖД	Мозолов И.Ф.	1.03-7.05.2016	
Глава 1	Курмамбаев З.К	3.01-29.02.2016	
Глава 2	Курмамбаев З.К	1.03-30.03.2016	
Глава 3	Курмамбаев З.К	1.04-30.04.2016	
Нормоконтроль	Курмамбаев З.К	19.05.2016	

Г Р А Ф И К
подготовки дипломного проекта

№ п/п	Наименование разделов, перечень разрабатываемых вопросов	Сроки представления руководителю	Примечание
1	Глава 1	3.01 – 29.02.2016	
2	Глава 2	1.03 – 30.03.2016	
3	Глава 3	1.04 – 30.04.2016	
4	БЖД	1.03 – 7.05.2016	
5	Экономика	1.03 – 7.05.2016	

Дата выдачи задания «19» октября 2016 г.

Заведующий кафедрой _____ Курамбаев З.К.
(подпись) (Фамилия и инициалы)

Руководитель _____ Курамбаев З.К.
(подпись) (Фамилия и инициалы)

Задание принял к исполнению
студент _____ Осипов Д.Б.
(подпись) (Фамилия и инициалы)

Аннотация

В данной дипломной работе представлено описание принципа работы компонентов летательного аппарата, выполнен анализ требований к летательному аппарату и блоку управления, рассчитаны параметры летательного аппарата и блока управления, подобраны комплектующие к квадрокоптеру, разработан алгоритм стабилизации квадрокоптера, написана прошивка для квадрокоптера, осуществлена сборка летательного аппарата, проведено тестирование и анализ тестовых полетов квадрокоптера, на основе анализа произведена доработка и представлена финальная сборка квадрокоптера, включающая передатчик видео, подвес, камеру, OSD, GPS–модуль и телеметрию.

В разделе безопасности жизнедеятельности произведён анализ условий труда, а также произведён расчёт заземлителя и молниезащиты.

В экономической части произведён расчёт затрат на разработку блока управления летательным аппаратом, которые составили 284614.64 тг, а также я рассчитал конечную стоимость летательного аппарата, которая составила 388897.44 тг.

Аңдатпа

Осы дипломдық жұмыста ұшу аппараты құрауыштарының жұмыс істеу қағидаты сипатталып, ұшу аппараты мен басқару блогына қойылатын талаптарға талдау жасалған, ұшу аппараты мен басқару блогының параметрлері есептеліп, квадрокоптердің құрауыштары іріктелген, квадрокоптерді тұрақтандыру алгоритмі әзірленіп, квадрокоптерге арналған жасақтама жазылған, квадрокоптерді бақылау үшін ұшыруды тестілеп, оларға талдау жасалған, квадрокоптердің соңғы құрастырылымы берілген.

Тіршіліктің қауіпсіздігі тарауында еңбек шарттарына талдау жасалып, сонымен бірге жерлендіргіш пен жайқорғанның есебі жүргізілді.

Экономика бөлімінде ұшу аппаратын басқару блогын әзірлеуге жұмсалатын шығындар есептелді, олар 284614.64 теңгені құрады, сондай–ақ ұшу аппаратының соңғы құны есептелді, ол 388897.44 теңгені құрады.

Annotation

In this thesis work the description of the principle of operation of the aircraft components were done, the analysis of the requirements for the aircraft and the control unit were made, the parameters of the aircraft and the control unit were calculated, components for quadcopter were chosen, quadcopter stabilization algorithm was designed, firmware for quadcopter was written, quadcopter was assembled, testing and analysis of quadcopter test flights were conducted, based on the analysis rework were performed and final version of quadcopter was presented,

including video transmitter, suspension, camera, OSD, GPS–module and the telemetry.

In the life safety section the analysis of working conditions was promoted, as well as the calculation of earthing and lightning protection were made.

In the economic part the calculation of development cost of the aircraft control unit, which amounted 284614.64 KZT, were made, and I calculated the final cost of the aircraft, which amounted 388897.44 KZT.

Содержание

Введение 10
1 Обзор предметной области 11
1.1 Область применения летательных аппаратов 11
1.2 Способы управлением летательными аппаратами 16
1.3 Классификация беспилотных летательных аппаратов 17
2 Принципы работы элементов летательного аппарата 20
2.1 Полётные контроллеры 20
2.2 Принцип работы бесколлекторного мотора 22
2.3 Принцип работы регулятора оборотов мотора 25
2.4 Принцип работы GPS–модуля 27
2.5 Общая информация о LiPo–аккумуляторе 28
3 Разработка блока управления летательного аппарата и сборка летательного аппарата 31
3.1 Анализ требований к блоку управления летательного аппарата 31
3.2 Выбор комплектующих и расчёт параметров летательного аппарата 32
3.2.1 Рама летательного аппарата 32
3.2.2 GPS–модуль Ublox Neo–6M 34
3.2.3 Полётный контроллер 35
3.2.4 Моторы летательного аппарата и расчёт регуляторов их скорости 36
3.2.5 Подвес и камера 39
3.2.6 Приёмник Boscam RC832 и передатчик TS832. Расчёт дальности сигнала 40
3.2.7 Расчёт электрических параметров летательного аппарата 43
3.3 Написание прошивки для полётного контроллера 45
3.3.1 Математическая модель стабилизации 45
3.3.2 Написание функций стабилизации летательного аппарата на основе математической модели 48
3.4 Сборка и настройка летательного аппарата 52
3.4.1 Прошивка GPS–модуля и полётного контроллера 52
3.4.2 Сборка летательного аппарата 61
3.4.3 Настройка полётного контроллера 68
3.5 Тестирование собранного летательного аппарата 71
3.6 Анализ тестовых полётов и доработка летательного аппарата 73
3.7 Окончательный вариант сборки 75
4 Безопасность жизнедеятельности 78
4.1 Анализ условий труда в офисе 78
4.2 Расчёт заземлителя 80
4.3 Расчёт молниезащиты 84
5 Технико–экономическое обоснование 87
5.1 Цели и задачи проекта 87
5.2 Расчет трудоемкости для разработки 87

5.3 Расчет затрат на выполнение НИР 90
5.4 Расчёт рентабельности НИР.....95
5.5 Оценка научно–технической результативности в социальной эффективности НИР.....96
Заключение 97
Список используемой литературы 99
Список используемых определений.....101
Приложение А 103
Приложение Б.....104

Введение

Причиной развития летательных аппаратов послужила потребность в лёгких, небольших размеров летательных аппаратах, обладающих высокой манёвренностью и способностью выполнять большое количество задач. Летательные аппараты успешно применяются в ходе военных операций по всему миру, при этом также успешно выполняются задачи гражданского назначения.

Область применения квадрокоптеров достаточно широка. Он может быть использован как недорогое и эффективное средство для получения фото, изображений и видео с воздуха, даже при плохих погодных условиях. Благодаря тому, что квадрокоптер – дистанционно управляемый летательный аппарат, он хорошо подходит для наблюдения и контроля объектов и зон, доступ к которым затруднен (например, в случае естественных или техногенных катастроф) или в условиях, непригодных для человека, таких как повышенный уровень радиации или сильное загрязнение воздуха.

На сегодняшний день большинство летательных аппаратов пилотируются вручную, с помощью пульта управления или наземной станции.

В дипломной работе я разрабатываю блок управления летательным аппаратом на основе микроконтроллера Arducopter. Разрабатываемый блок управления не имеет аналогов, т.к. в гражданских целях используются простые блоки управления, а в военных – закрытого типа, что не позволяет узнать его конструкцию. Я провожу сборку квадрокоптера и тесты.

Актуальность работы заключается в разработке open source продукта для устройства, которое может заменить человека в некоторых сферах деятельности.

Цель работы: разработать блок управления летательным аппаратом на основе микроконтроллера Arducopter.

Задачи проекта:

- разобрать принцип работы компонентов летательного аппарата;
- выполнить анализ требований к летательному аппарату и блоку управления;
- рассчитать параметры летательного аппарата и блока управления;
- подобрать комплектующие к квадрокоптеру;
- разработать алгоритм стабилизации квадрокоптера;
- написать прошивку для квадрокоптера;
- собрать летательный аппарат;
- провести тестирование и анализ тестовых полетов квадрокоптера;
- представить финальную сборку квадрокоптера на основании анализа тестовых полётов.

1 Обзор предметной области

1.1 Область применения летательных аппаратов

В настоящее время летательные аппараты начинают оказывать влияние на нашу жизнь, упрощая её. Летательные аппараты в скором времени будут использоваться повсеместно. Пока применение летательных аппаратов в большинстве случаев несёт только научный характер, т.к. только в некоторых странах, таких как США, Франция и Япония, существует свод законов, который регулирует деятельность летательных аппаратов. Благодаря этим законам выделяется летательное пространство для летательных аппаратов, т.е. высоты, на которых летательный аппарат может летать. К сожалению, в Казахстане таких законов нет, что замедляет рост популярности летательных аппаратов и использование их в качестве помощника человека. Летательные аппараты можно использовать в любой сфере деятельности. Но пока основная сфера использования летательных аппаратов военная, т.к. летательные аппараты позволяют вести исследование территорий с большой высоты, при этом развивать большую скорость.

Военные используют летательные аппараты для разведывательных операций, позволяющих произвести разведку местности и съёмку расположения противника. Также летательные аппараты используют для наведения ракет на объекты поражения, что позволяет свести к минимуму риск быть обнаруженным и нанести неожиданный удар. В недавнем конфликте в Сирии российскими военными были использованы летательные аппараты для разведывательных операций и съёмки зоны поражения после обстрела артиллерией. Недавние разработки позволили поместить тяжёлое огнестрельное оружие на летательных аппаратах, позволяя отправлять летательный аппарат в зону боевых действий для зачистки местности. Несмотря на небольшие размеры, летательные аппараты могут перевозить ракеты для ведения боевых действий. Благодаря летательным аппаратам снижается количество жертв среди военных, позволяя вести бой без участия человека непосредственно в зоне боевых действий. Летательные аппараты позволяют вести войну против террористов, уменьшая потери среди заложников и мирного населения. Из–за небольшого размера летательных аппаратов их трудно обнаружить. Лидером в производстве летательных аппаратов для военных целей является Израиль. На рисунке 1.1 показан военный израильский летательный аппарат, использующийся в разведывательных операциях. Но с каждым годом использование летательных аппаратов становится опаснее: данные GPS можно фальсифицировать, что позволяет поменять маршрут полёта летательного аппарата; также стоит учесть то, что для управления необходимы каналы связи с большой пропускной способностью, т.к. количество данных, которые необходимо передавать между летательным аппаратом и наземной станцией велико.

Несмотря на систему защиты, хакерам удаётся перехватить не только сигнал, но и перехватить управление летательным аппаратом.

Рисунок 1.1 – Военный израильский летательный аппарат

Самым известным взломом считается в 2011 году американского летательного аппарата иранскими военными, которые использовали GPS–спуфинг. Спуфинг означает, что сигналы GPS были подменены и впоследствии возникла внештатная ситуация, не поддерживаемая программным кодом, из–за чего летательный аппарат вернулся на домашнюю базу по подменённым данным.

Но использование летательных аппаратов не ограничивается военными, они также нашли применение в гражданской области. Область применения летательных аппаратов в гражданских целях обширна, начиная от защиты объектов и заканчивая развлечениями.

Летательные аппараты можно будет использовать для охраны объектов, что позволит охватить большую площадь за небольшое время, к тому же позволит оперативно реагировать на проникновение в зону постороннего объекта. Российские пограничники используют летательные аппараты для контроля государственной границы, позволяя автоматически определить человека, находящегося в запрещенной зоне. Такие летательные аппараты, как и военные, оборудованы видеокамерой и тепловизором для съёмки в ночное время. Также летательные аппараты осуществляют охрану не только объектов, но съемку и сопровождение различных мероприятий, таких как Олимпийские игры, футбольные матчи, праздничные мероприятия городского масштаба и т.д.

Также летательные аппараты удобнее использовать для поиска людей, т.к. для него не требуется аэродром для взлёта, что позволяет начать поиски людей мгновенно, что ускоряет время поиска. Т.к. летательный аппарат имеет небольшие размеры, то им проще осуществлять поиск в ущельях гор, а также

в лесах, в то время когда сейчас используются людей для поиска. Летательные аппараты используются при чрезвычайных ситуациях, позволяя охватывать большую площадь за небольшое время. Летательные аппараты используют для поиска пострадавших во время наводнения. Летательные аппараты для таких целей используются МЧС России. На рисунке 1.2 показана группа МЧС России с летательными аппаратами, используемые во время ЧС.

Рисунок 1.2 – МЧС России с летательными аппаратами

Также в 2011 году после аварии на АЭС Фукусиме–1 были использованы летательные аппараты для оценки повреждения АЭС, не подвергая людей опасности.

В 2015 году во время селя в городе Алматы, прошедшем по реке Каргалинка, были использованы летательные аппараты для мониторинга русла реки, а также для мониторинга мореных озёр, что позволило выиграть время, если вдруг прорвало бы ещё одно озеро.

Также их задействовали для поиска людей во время селя, прошедшего по реке Талгар, когда искали несколько групп детей, т.к. летательные аппараты благодаря тепловизорам могут производить поиск людей в тёмное время суток.

Летательные аппараты также хорошо подходят для доставки товаров, что позволяет отказаться от доставки курьером на машине. Благодаря этому уменьшаются затраты на бензин и его использование, т.к. летательные аппараты работают от аккумуляторов LiPo. С уменьшением количества машин улучшится и экологическая ситуация в городах, т.к. с уменьшением количества машин уменьшается количество выбросов вредных газов. В 2014 году в России пиццерия осуществляла доставку пиццы с помощью летательных аппаратов непродолжительное время, после чего пришлось выплатить штраф фирме за отсутствие лицензии на перевозку воздушным транспортом. На рисунке 1.3 летательный аппарат доставляет пиццу.

Рисунок 1.3 – Доставка пиццы летательным аппаратом

Первым, кто осуществил доставку товара летательным аппаратом, был Amazon, который дорабатывает технологии, чтобы они были безопасны и соответствовали законам США. На рисунке 1.4 показан летательный аппарат компании Amazon.

Рисунок 1.4 – Летательный аппарат компании Amazon

Летательные аппараты можно использовать для обнаружения пожаров в степной, горной и лестной местности. Они также позволят в кратчайшие сроки оповестить о пожаре, ведь в настоящее время между началом пожара и оповещением проходит много времени, что уменьшает эффективность борьбы с пожаром.

Одна из частых опасностей – это дорожное движение. Летательные аппараты позволят наблюдать за дорожной обстановкой, выявлять нарушителей, таким образом уменьшить участие дорожной полиции, что приведёт к автоматизации деятельности полиции и уменьшит количество взяток. Например, в Китае с помощью летательных аппаратов выявляют нарушителей дорожного движения.

Летательные аппараты пользуются большой популярностью в сфере развлечения, фото– и видеосъёмки. Так, например, во время Олимпийских игр в Сочи в 2015 году использовались летательные аппараты для съёмки соревнований по сноуборду и лыжам. На рисунке 1.5 показан летательный аппарат, производящий съёмку Зимних Олимпийских игр.

Рисунок 1.5 – Летательный аппарат, производящий съёмку Зимних Олимпийских игр

В сфере развлечений используются от небольших летательных аппаратов, размером с кулак, до больших устройств. В этом сегменте в основном представлены летательные аппараты фирм DJI и Syma, а также множество небрендовых фирм. Цены варьируются от 15$ за небольшой летательный аппарат и доходят до нескольких тысяч долларов. На рисунке 1.6 показан летательный аппарат DJI Inspire 1 Pro, который предназначен для профессиональной съёмки.

Рисунок 1.6 – Летательный аппарат DJI Inspire 1 Pro

Также стоит учесть, что летательные аппараты стали частью выставок. Например, на выставке CES, проходящей в Лас-Вегасе компании представляют свои версии летательных аппаратов, кроме прототипов

показываются работающие модели. На выставке CES-2016 была показана интересная модель летательного аппарата, выполненная в форме мяча (рисунок 1.7).

Рисунок 1.7 – Летательный аппарат, представленный на выставке CES-2016

В Казахстане, к сожалению, не проводятся такие выставки, но в России ежегодно проводится выставка HelliRussia, которая направлена на популяризацию летательных аппаратов, в рамках которой проходит не только демонстрация летательных аппаратов и конференции, но и разные соревнования, например, гонка на летательных аппаратах на специализированной трассе.

Также в России ежегодно проводятся соревнования по разработке летательных аппаратов, которые должны автоматически выполнять разные действия. Эти действия меняются каждый год. Например, в одном из соревнований требовалось, чтобы летательный аппарат заданных размеров пролетел по созданным коридорам, не пересекая заданную границу, при этом должен был приземлиться в конце полосы испытаний на специально отведенное для этого место, при этом учитывается время прохождения дистанции.

1.2 Способы управлением летательными аппаратами

Способ управления летательным аппаратом определяется выполняемой задачей. Выделю пять основных способов управления летательными аппаратами:

1) Дистанционное пилотирование. Оператор непрерывно управляет всеми функциями управления, устройствами и поведением летательного аппарата. Управление осуществляется с подвижного или со стационарного пункта управления.

2) Автоматическое пилотирование. Оператор не осуществляет управление летательным аппаратом. Летательный аппарат реализует своё функциональное назначение в автоматическом режиме в соответствии с

написанной программой его поведения, зашитой в него. Все функции управления выполняются на основе алгоритма программы. Наземная станция, база, принимает данные с летательного аппарата и обрабатывает их. Широко применяются в боевых операциях.

3) Дистанционное управления авиационной системой. Летательный аппарат производит непрерывное управление функциональным поведением, которое осуществляется с подвижного или неподвижного пункта управления. Этот тип управления сочетает преимущества двух предшествующих способов. Оператор влияет непосредственно на использование заложенных функций, не отвлекаясь при этом на выполнение задач пилотирования летательным аппаратом.

4) Дистанционно-автономное управление. Летательный аппарат, автономно реализующий свое функциональное предназначение путем формирования и выполнения внутренних динамических алгоритмов функционального поведения при эпизодическом вмешательстве оператора боевого управления для переопределения цели или постановки новой боевой задачи. Используется в основном для управления задачами групп летательных аппаратов и контролем их поведения.

5) Управление нейронной сетью. Каждый летательный аппарат управляется системой, состоящей из нейронной сети и конечного автомата. Каждый летательный аппарат представляет собой агента, взаимодействующего с внешней средой и другими агентами. Нейронная сеть используется для обработки входных параметров и передачи их на конечный автомат, который вырабатывает управляющий выходной сигнал и передаёт летательным аппаратам. На рисунке 1.8 показана система управления летательным аппаратом на основе нейронной сети.

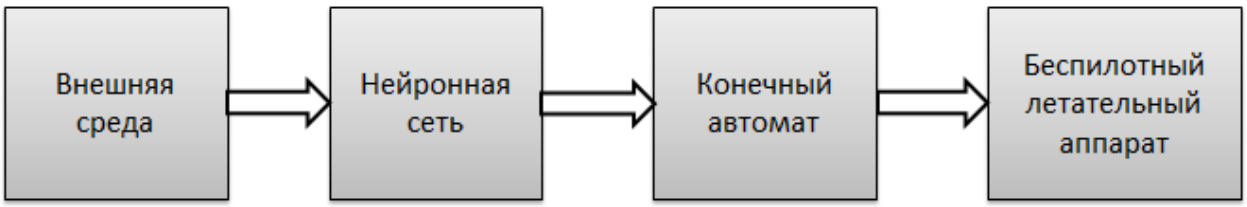

Рисунок 1.8 – Структурная схема системы управления летательным аппаратом на основе нейронной сети.

1.3 Классификация беспилотных летательных аппаратов

В зависимости от целей, которые выполняет летательный аппарат, разделяют на несколько классов, которые отличаются взлётной массой и дальностью полёта.

Выделю следующие классы летательных аппаратов:

1) Микро- и мини-летательные аппараты ближнего радиуса действия. К этому классу относятся маленькие и сверхлёгкие аппараты и комплексы с

взлётной массой до 5 кг. В данное время большинство летательных аппаратов относятся к этому классу. Такие летательные аппараты предназначены для индивидуального использования до 25–40 км. Они просты в эксплуатации и транспортировке. Запуск осуществляется с небольших неспециализированных площадок. Область применения: охрана объектов, аэросъёмка и развлекательная. К данному классу относятся самолёты, вертолёты и квадрокоптеры.

2) Лёгкие летательные аппараты малого радиуса действия. К этому классу относятся более крупные летательные аппараты: взлётной массой от 5 до 50 кг, а дальность действия до 120 км. К данному типу можно отнести как специализированные, так и самодельные летательные аппараты. Для взлёта также не требуется специализированная взлётная площадка. Область применения разведка на небольшие расстояния, охрана объектов. К данному классу уже относятся только самолёты, т.к. квадрокоптеры не могут летать на расстояния до 120 км.

3) Лёгкие летательные аппараты среднего радиуса действия. От предыдущего класса отличаются большим радиусом действия (до 250 км) и взлётная масса от 100–300 км. Лёгкие летательные аппараты данного класса позволяют производить разведывательные операции на средние расстояния, что позволяет использовать их непосредственно в зоне боевых действий.

4) Средние летательные аппараты. Дальность действия увеличена до 1000 км, а полётный вес составляет до 300 кг. Это позволяет вести разведывательные операции на средних дистанциях. Благодаря такой дистанции, штаб может находиться на территории соседа–союзника и выполнять управление оттуда.

5) Тяжелые летательные аппараты. Данным летательным аппаратам свойственна большая продолжительность работы, поэтому они используются для разведывательных целей на больших расстояниях до 1500 км. Могут быть укомплектованы оружием для самообороны. К данному классу относятся Predator, Reaper, Global Hawk, Heron, Heron TP. Летательные аппараты такого класса есть только у ВВС США и Израиля.

6) Беспилотные боевые самолёты. Дальность действия – 1500 км. Несут на борту боевое оружие. Данный класс предназначен для ракетных ударов по наземным и надводным целям, также хорошо показывают себя против подвижных целей. Взлётная масса свыше 1500 кг, что позволяет хорошо укомплектовать летательный аппарат.

Вывод

Область применения летательных аппаратов разнообразна: военная, спасательная, доставка товаров, развлечения и т.д. С каждым годом летательным аппаратам находят новые области применения. Поэтому важно развивать эту область. В скором времени летательные аппараты будут полностью автоматизированы и автономны, что позволит использовать их длительное время и на больших расстояниях без участия человека, позволив исключить человеческих фактор. Стоит увеличивать количество мероприятий, которые направлены на популяризацию летательных аппаратов. Все эти выставки и соревнования оказывают влияние не только на развитие технического прогресса, но и позволяют использовать летательные аппараты в учебных целях, начиная от простых модульных аппаратов для школьников и заканчивая отдельным курсом написания программ для летательных аппаратов. Кроме того, это всё закладывает основы знаний из области физики, прикладной механики, программирования. Это позволяет внести знания в учебный процесс, разбавляя серые будни школьников и студентов, позволяя заинтересовать технической наукой с раннего возраста.

2 Принципы работы элементов летательного аппарата

Летательный аппарат состоит из двух основных компонентов: блока управления и электроники. В блок управления входят такие элементы, как полётный контроллер, приёмник сигнала, GPS–модуль, телеметрия, также передатчик видеосигнала и видеокамера. К электронике относятся другие элементы, которые не входят в блок управления: моторы, аккумулятор, регуляторы оборотов моторов.

Необходимо иметь представление об отдельных компонентах летательного аппарата, о его характеристиках, внутреннем строении и принципе работы. В ходе выполнения работы важен не только конечный результат, но и принципы работы, как отдельных компонентов, так и всей системы в целом. Поэтому важно иметь представление:

– какие бывают полётные контроллеры;

– принцип работы бесколлекторного мотора;

– из каких блоков состоит регулятор скорости оборотов и за что каждый блок отвечает;

– как определяются координаты GPS–модуля;

– знать общую информацию о LiPo–аккумуляторах.

2.1 Полётные контроллеры

Полётный контроллер является мозгом летательного аппарата, который позволяет управлять им, основываясь на показаниях датчиков и входных сигналов с приёмника. Контроллер отвечает за выполнение таких важных функций, как стабилизация квадрокоптера в воздухе, удержание высоты, полёт по заданному маршруту, обеспечение безопасности полёта, т.е. отвечает за поведение квадрокоптера во время полёта. Существует множество полётных контроллеров, но я выделю самые популярные, которые имеют лучшие характеристики по сравнению с другими.

Начну сравнение с контроллера Naza–M, который используется в квадрокоптерах фирмы DJI. Контроллер имеет барометр, гироскоп, альтметр и акселерометр. Но нет компаса, поэтому придётся устанавливать внешний компас. На рисунке 2.1 показан контроллер Naza–M.

Рисунок 2.1 – Полётный контроллер Naza–M

Сразу стоит отметить, что проект является закрытым и самому изменять прошивку не будет возможно, также нет схем разводки датчиков. Контроллер идёт из «коробки», готовый к полёту, что является плюсом для тех, кто хочет сразу летать, а не писать прошивку и настраивать. Также к плюсам отнесу 32-битный процессор STM32, который позволяет быстро обрабатывать входные сигналы и данные. Но есть большие минусы: не совместим с большинством GPS–модулей, а только с GPS–модулями DJI, а также цена. Цена комплекта с GPS–модулем и контроллером стоит 300$, а без GPS–модуля 250$, что позволяет использовать его в дорогих летательных аппаратах.

Самым популярным полётным контроллером является Arducopter, т.к. он является проектом с открытым исходным кодом, т.е. в свободном доступе есть все чертежи и прошивки, на основе которых можно написать свою прошивку. Из–за того, что чертежи схемы находятся в открытом доступе, это позволяет осуществить сборку самому или заказать аналог, который стоит от 35$, а с GPS–модулем от 45$. Arducopter представлен на рисунке 2.2.

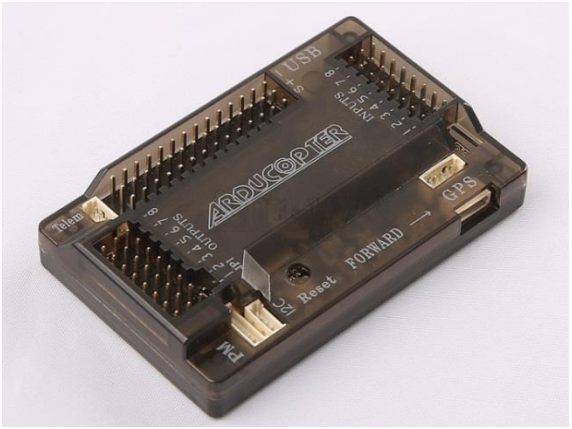

Рисунок 2.2 – Полётный контроллер Arducopter

Как и Naza–M, Arducopter также имеет барометр, гироскоп, альтметр, акселерометр и встроенный компас. Т.к. Arducopter совместим с Arduino, то можно использовать датчики Arduino: датчик определение скорости, датчик определения расстояния и т.д. Также Arducopter совместим с большинством GPS–модулей, что позволяет не тратить большие деньги и легко найти совместимый модуль. К минусам контроллера можно отнести процессор AtMega2560, т.к. он является 8-битным, из–за этого при усложнении прошивки появится вероятность того, что процессор окажется узким местом полётного контроллера. Разработчики учли, что 8-битный процессор в будущем станет узким местом, и разработали усовершенствованную версию Arducopter – Pixhawk. На рисунке 2.3 показан полётный контроллер Pixhawk.

Рисунок 2.3 – Полётный контроллер Pixhawk

Как и на всех контроллерах, имеется в наличии барометр, гироскоп, альтметр, акселерометр и встроенный компас. Но большим плюсом контроллера является процессор и датчики. В качестве процессора используется Arm Cortex–M4, 32-битный процессор, который совмещает высокую производительность и энергоэффективность. В качестве гироскопа использует MPU6000 шестиосевой гироскоп и акселерометр, что увеличивает скорость и точность данных. Контроллер не позволяет подключать такие же датчики, как к Arducopter. Также стоит учесть, что Pixhawk является проектом с закрытом кодом, таким образом, усложняется разработка под данный контроллер. Также стоит учесть высокую цену – от 350$.

На основе проанализированных данных составлю сводную таблицу 2.1.

Таблица 2.1 – Сводная таблица данных о контроллерах

Наименование	Naza–M	Arducopter	Pixhawk
Наличие гироскопа, акселерометра	+	+	+
Наличие компаса	–	+	+
Возможность подключать датчиков	–	+	–
Процессор	STM32	AtMega2560	ARM Cortex–M4
Разрядность процессора, бит	32	8	32
Открытый проект	–	+	–
Цена, $	250	35	350

2.2 Принцип работы бесколлекторного мотора

В качестве моторов используются бесколлекторные моторы, т.к. они имеют большой КПД по сравнению с коллекторными, а также меньше греются. Конструктивно бесколлекторный мотор состоит из ротора с постоянными магнитами и статора с обмотками. В коллекторном моторе в роторе находятся обмотки, а в статоре магниты. Для управления мотором

применяется электронный регулятор, регулятор оборотов мотора. На рисунке 2.4 показано внутреннее строение бесколлекторного мотора.

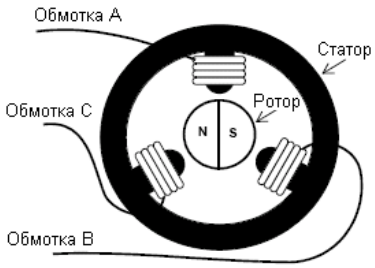

Рисунок 2.4 – Внутреннее строение бесколлекторного мотора

По сравнению с коллекторными двигателями тут отсутствует коллектор, который имеет большой вес и усложняет конструкцию мотора. С его удалением конструкция упростилась. Это повлияло также на уменьшение веса, тепловыделения и размеров мотора, при этом увеличив КПД. Бессколлекторные моторы позволяют изменять скорость вращения в большом диапазоне. Бессколлеторные моторы не создают радиопомех. Единственный недостаток – это необходимость использование регуляторов скорости, что увеличивает стоимость летательного аппарата. При подаче напряжения на мотор магнитное поле начинает действовать на обмотки двигателя. Ротор вращается в магнитном поле не постоянно, а поворачивается до определённого положения. Чтобы он вращался постоянно, необходимо подавать на нужные обмотки статора постоянное напряжение, в зависимости от положения ротора. Для определения этого положения используются датчики положения, которые помогают регуляторам оборотов знать, в каком месте находится ротор и подать на нужные обмотки статора напряжение. В основном используются трёхфазные моторы. С увеличением фаз увеличивается плавность хода мотора. Фазы соединяются по схеме “звезда” или “треугольник”. На рисунке 2.5 показаны схемы соединения обмоток.

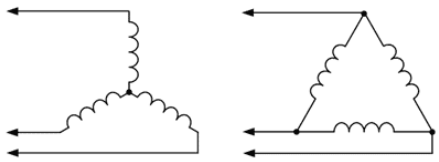

Рисунок 2.5 – Схемы соединения обмоток мотора

В трёхфазной системе в каждый момент времени напряжение подаётся на две из трёх обмоток. Таким образом, существует шесть вариантов подачи постоянного напряжения на обмотки двигателя, как показано на рисунке 2.6. Это позволяет создавать магнитное поле, которое будет поворачивать ротор на 60° при каждом переключении.

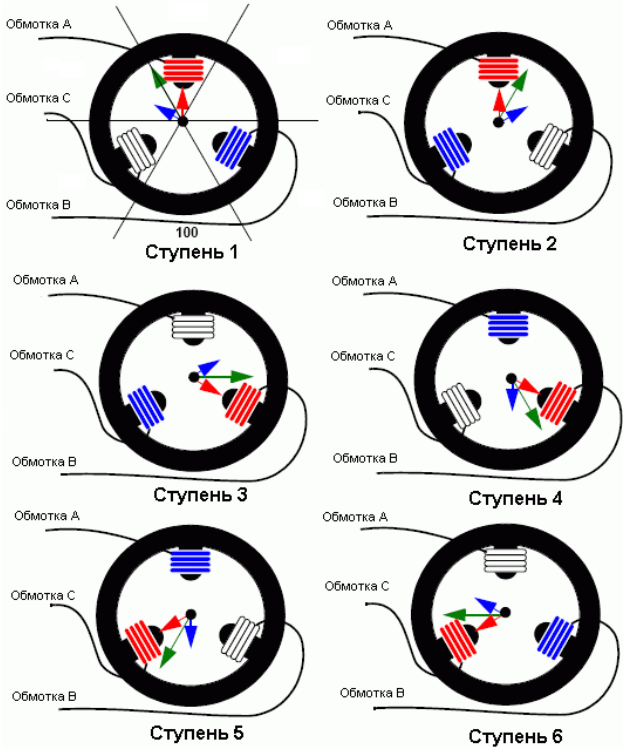

Рисунок 2.6 – Варианты подачи постоянного напряжения

В большинстве моторов используются датчики Холла для определения магнитного поля, которые позволяют определить положение ротора. На основе датчиков замыкаются необходимые ключи в мостовой схеме управления регуляторов скорости моторов. На рисунке 2.7 показана мостовая схема ключей, которая управляет ротором.

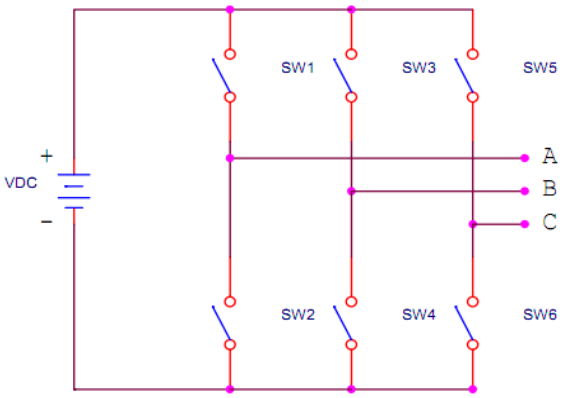

Рисунок 2.7 – Мостовая схема ключей

Двигатель имеет три фазы, которым передаются в разные моменты времени “+” или “–“. Это реализуется с помощью электронных ключей. Замыкая ключ SW1, подаем “+” на фазу A, а замыкая SW6, подаем “–” на фазу C. Таким образом, ток потечет от “+” батареи через фазы A и C. Для обеспечения обратного направления, открываем SW5 и SW2. В этом случае ток потечет от “+” батареи через фазы C и A в обратном направлении. При

работе двигателя одновременно должен быть открыт только один верхний ключ и один нижний ключ. При смене состояния нужно сразу выключить пару ключей, выждать время, необходимое для закрытия ключей, и только после этого включить другую пару ключей. Ключи открываются согласно значениям, полученным с датчиков Холла. В таблице 2.2 представлена комбинация ключей в зависимости от значения датчиков Холла.

Таблица 2.2 – Таблица включения ключей в зависимости от значений датчиков Холла

Значение датчиков Холла	Фаза	Переключатели
101	A–B	SW1;SW4
001	A–C	SW1;SW6
011	B–C	SW3;SW6
010	B–A	SW3;SW2
110	C–A	SW5;SW2
100	C–B	SW5;SW4

2.3 Принцип работы регулятора оборотов мотора

Регулятор оборотов мотора контролирует скорость мотора на основе входных сигналов. Регулятор состоит из контроллера, ключей и датчиков. Схема регулятора оборотов мотора показана на рисунке 2.8.

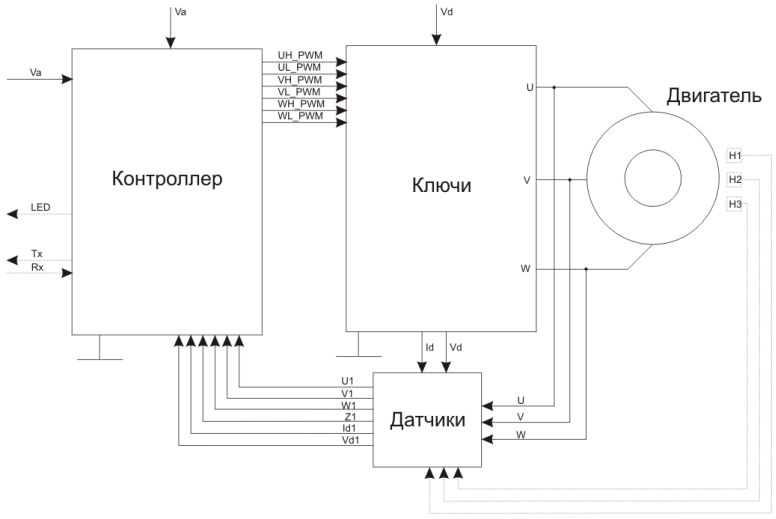

Рисунок 2.8 – Схема регулятора оборотов мотора

Регулятор состоит из трёх блоков:

1) блок контроллера – принимает сигналы с датчиков и полётного контроллера, при этом отправляет управляющие сигналы на ключи;
2) блок силовых ключей – управляет силовыми ключами;
3) блок датчиков – набор различных датчиков и схем согласования.

Блок силовых ключей состоит из шести ключей, включенных по мостовой схеме, как показано на рисунке 2.7. Блок состоит из транзисторных ключей и драйверов ключей, которые преобразовывают логические уровни микроконтроллера в сигналы, управляющие силовыми транзисторами. Драйверы должны обеспечивать временную задержку между закрытием одного ключа и открытием другого. При выходе из строя ключа драйвер ключа должен препятствовать прохождению напряжения, питающего двигатель. В регуляторах, рассчитанных на работу с большим током, используются два полевых транзистора, подключенных параллельно, а к затворам транзисторов подключить сопротивление. Блок ключей можно представить в виде логической схемы, где элемент «И» выполняет функцию транзистора. При поступлении двух единичных сигналов сигнал проходит дальше. Логическая схема представлена на рисунке 2.9.

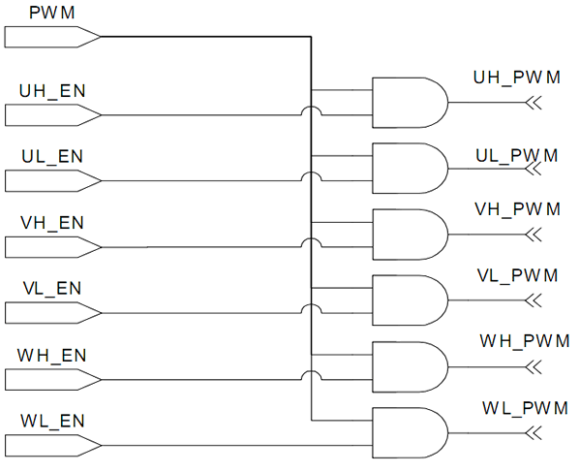

Рисунок 2.9 – Логическая схема представления транзисторных ключей

Блок датчиков занимается непосредственно измерением и формированием сигналов, преобразуя их к виду, удобному для контроллера. Например, блок датчиков считывает данные с датчиков Холла, расположенных в моторе, и передают их в контроллер для дальнейшей обработки.

Блок контроллера должен выполнять следующие функции:

- генерировать ШИМ–сигнал для открытия и закрытия шести ключей;
- регулировать силу тока, которая необходима мотору;
- регулировать напряжение мотора;
- принимать управляющий сигнал, который задаёт обороты двигателя.

Регулятор должен препятствовать возникновению нештатных ситуаций и в критических случаях отключать двигатель:

- превышение допустимого тока;
- заклинивание двигателя;
- превышение двигателем допустимых оборотов;

– снижение питающего напряжения ниже допустимого уровня (предотвращение глубокого разряда батареи).

Работа начинается с входного управляющего сигнала с полётного контроллера. После чего контроллер считывает значение блока датчиков, чтобы узнать положение статора. На основе полученных данных контроллер отправляет сигнал в блок силовых ключей, где закрываются два транзисторных ключа. Блок датчиков в это время получает данные с датчиков мотора и передаёт в контроллер, который изменяет состояние ключей.

2.4 Принцип работы GPS–модуля

В начале работы GPS–модуль проводит проверку альманаха и эфемерид. На основе результата проверки будет выбран один из режимов инициализации. Альманах содержит параметры орбит всех спутников, а эфемерид – точные данные об орбите спутника. Каждый спутник передаёт альманах для всех спутников. Таким образом, альманах содержит примерные координаты спутников и не требует частого обновления. Каждый спутник передаёт свой собственную эфемериду, т.е. свои точные данные о спутнике. Эфемерида для каждого спутника достоверна в течение нескольких часов. Каждый эфемерид имеет метку, на основе которой определяется достоверность информации.

У GPS–модуля существует три режима инициализации:

– холодный режим – данные альманаха устарели и, следовательно, устарела эфемерида. GPS–модуль получает альманах, а затем эфемериду;

– тёплый старт – данные альманаха актуальны, а данные эфемерида устарели. GPS–модуль принимает только данные эфемерида спутников;

– горячий старт – данные альманаха и эфемерида актуальны и GPS–модуль готов к работе.

Когда при запуске данные альманаха отсутствуют или проходят проверку, то GPS–модуль сканирует используемые частоты и все спутники. Если GPS–модуль нашёл спутник, то спутник передаёт ему свои точные данные, эфемериду, что занимает примерно 30 с. Пока спутник передаёт эфемериду, другие каналы GPS–модуля продолжают искать спутники. После нахождения необходимого количества спутников, GPS–модуль может рассчитать свои координаты. Для расчёта координат расположения GPS–модуля необходимо знать расстояние от GPS–модуля до трёх спутников. Для этого приемник генерирует свой внутренний код в то же самое время, чтобы он точно дублировал код спутников. Приемник сравнивает разницу во времени между приемом соответствующей части спутникового кода с такой же частью своего кода. Зная сдвиг по времени и скорость распространения радиоволн, приемник получает расстояние до спутника, называемое псевдодальностью, и по двум расстояниям может определить свое точное положение. Псевдодальность включает в себя ошибки, вызванные задержками сигналов в топосфере, ионосфере и т.д., а также ошибки часов приёмника.

Часы спутника являются точными и корректируются с Земли, которые создают суммарную ошибку, которая корректируется третьим спутником. С увеличением количества спутников, которым GPS–модуль отправляет сигнал, увеличивается точность координат.

2.5 Общая информация о LiPo–аккумуляторе

Аккумулятор является поставщиком энергии летательному аппарату, поэтому важно уметь разбираться в их характеристиках, а также правильно эксплуатировать и хранить их. LiPo–аккумуляторы – усовершенствованная конструкция литий–ионного аккумулятора, в качестве электролита используется полимерный материал с включениями гелеобразного литий–проводящего наполнителя.

Преимущества LiPo–аккумуляторов, по сравнению с Li–Ion:

– способность отдавать ток, в десятки раз превышающего численное значение ёмкости;

– большая плотность энергии на единицу объёма и массы;

– низкий ток саморазряда;

– вследствие возможности большей токоотдачи увеличивается выходная мощность аккумулятора.

Для правильного выбора аккумулятора необходимо правильно расшифровать обозначения аккумулятора. Например, есть аккумулятор с характеристиками 5000 мА · ч 4S2P 35C;

1) 5000 мА · ч является ёмкостью аккумулятора, равная 5000. С увеличением ёмкости увеличивается время, в течение которого аккумулятор может обеспечивать необходимым ток и напряжение во время нагрузки на него. От ёмкости также зависит максимальная токоотдача;

2) аккумулятор состоит из нескольких элементов, которые соединяются последовательно или параллельно в зависимости от необходимых выходных характеристик. Каждый элемент имеет максимальное напряжение 4.2 В. 4S означает, что четыре элементы соединены последовательно, при этом увеличивается выходное напряжение. 2P означает, что это два последовательно соединённых элемента подключаются друг к другу параллельно, при этом увеличивается выходной ток;

3) одной из важных характеристик является параметр C, который характеризует величину токоотдачи, т.е. максимальный ток аккумулятор, который он может отдавать. Эта величина в примере равна 35C. Для этого необходимо ёмкость умножить на величину токоотдачи, для примера это 175 А. 175 А может отдавать аккумулятор.

Для обеспечения продолжительной работы необходимо правильно эксплуатировать аккумулятор. Для этого надо придерживаться следующих пунктов:

1) критическое напряжение для одного элемента аккумулятора является 3.3 В. Эксплуатация или хранение аккумулятора с более низким

напряжением приводит к снижению ёмкости. При напряжении 2.8 В происходит выделение соли лития, которая осаждается на электродах, уменьшая плотность электролита (снижая ёмкость) и токоотдачу. При напряжении меньше 2.5 В происходит металлизация лития, образование токопроводящих участков внутри аккумулятора, при заряде такого аккумулятора увеличивается температура аккумулятора в результате короткого замыкания. В конечном итоге возможен взрыв;

2) ток заряда должен соответствовать 1С, т.е. для аккумулятора 5000 мА · ч ток заряда должен быть 5 А;

3) в холодное время аккумуляторы стоит держать в тёплом месте;

4) температура во время эксплуатации не должна превышать 60 °С.

При повреждении аккумулятора необходимо его утилизировать. Но сначала выполнить следующее:

- полностью разрядить аккумулятор, используя нагрузку;
- подержать в соленой воде до прекращения газообразования;
- не надрезать сам аккумулятор, т.к. это может привести к взрыву.

Вывод

Перед анализом требований к блоку летательного аппарата необходимо было произвести обзор его компонентов. Без знаний характеристик и принципов работы будет сложно составить требования к блоку управления летательным аппаратом, на основе которых будет производиться расчёт и сборка. Поэтому я разобрал основные компоненты, представив схемы внутреннего устройства компонентов, а также произвёл описание их принципа работы. Также представил информацию о LiPo–аккумуляторах. Эта информация позволит на основе расчётных параметров правильно подобрать аккумулятор для полёта.

3 Разработка блока управления летательного аппарата и сборка летательного аппарата

Необходимо обозначить этапы разработки, чтобы разработка всей системы не была хаотичной, приводящей к нежелательным результатам. Поэтому выделяю этапы разработки:

1) анализ требований к летательному аппарату;
2) выбор комплектующих и расчёт параметров летательного аппарата;
3) программирование полётного контроллера;
4) сборка летательного аппарата;
5) тестирование собранной системы;
6) анализ необходимых доработок;
7) окончательный вариант сборки после доработки.

Следуя вышеперечисленным этапам, получится избежать проблем, когда часть требований не выполняется и приходится это дорабатывать в дальнейшем, после выпуска финальной версии.

3.1 Анализ требований к блоку управления летательного аппарата

Перед началом разработки блока управления летательного аппарата, необходимо проанализировать и составить список требований, которым блок управления должен соответствовать. На основе требований к блоку управления будет написана прошивка для полётного контроллера, которая будет связывать компоненты блока управления, обрабатывать полученные данные с датчиков и впоследствии принимать решения во время полёта. Также необходимо составить список требований к самому летательному аппарату, без этих требований будет невозможно собрать необходимый летательный аппарат.

Требования к блоку управления летательным аппаратом:

– наличие полётного контроллера с гироскопом, компасом, барометром, акселерометром и возможностью подключения других датчиков;

– наличие приёмника сигнала, от 5–ти каналов и выше;

– наличие передатчика видеосигнала с диапазоном действия не менее 1 км;

– наличие модуля телеметрии для передачи данных с контроллера на расстояние свыше 10 км;

– наличие подвеса для камеры из алюминия, т.к. пластик ломкий;

– наличие контроллера для подвеса, необходимого для управления подвесом и регулирования уровня камеры;

– камера должна обеспечивать хороший обзор, небольшое фокусное расстояние, и небольшой вес;

– наличие GPS–модуля для удобной навигации на больших расстояниях полёта.

Каждый из элементов блока управления помогает ориентироваться на местности во время полёта визуально, также на основе полученных данных геолокации, что позволит управлять летательным аппаратом на больших расстояниях.

Требования к летательному аппарату:

– вес летательного аппарата не должен превышать 1.5 кг;

– мощность моторов должна быть достаточна, чтобы летательный аппарат зависал в воздухе, не терял высоту, при уровне газа не больше, чем 50%;

– время полёта должно быть больше 20 минут;

– рама должна быть сделана из лёгких и прочных материалов;

– ремонтопригодность, т.е. летательный аппарат можно легко починить;

– модульность. Модульность является самым важным критерием, т.к. от этого зависит ремонтопригодность и затраты на ремонт. Благодаря модульности нет необходимости при отказе одного из компонентов ремонтировать весь летательный аппарат в мастерской вдали от места полёта и перепаивать. Можно достать привезённую с собой запасную часть и произвести замену, после чего продолжить полёт. Также модульность помогает упростить систему и в дальнейшем проще производить модернизацию аппарата.

Написание прошивки заключается в получении данных с датчиков контроллера, обработке их и в дальнейшем на основе обработанных данных микропроцессор принимает решения по управлению летательным аппаратом.

3.2 Выбор комплектующих и расчёт параметров летательного аппарата

Перед написанием прошивки необходимо выбрать комплектующие для летательного аппарата, которые будут удовлетворять требованиям, которые были описаны в 3.1. Следуя этим требованиям, можно будет точно подобрать комплектующие для данного летательного аппарата и в дальнейшем произвести расчёт его параметров. Расчёт поможет определить такие параметры, как максимальный полётный вес, время полёта, а также электрические параметры летательного аппарата, на основе которых будут выбраны регуляторы моторов, аккумулятор, и в дальнейшем сборки импульсный стабилизатора напряжения. Полётный контроллер может не обеспечивать достаточный уровень напряжение на вспомогательные элементы и в дальнейшем им может понадобиться отдельный преобразователь питания.

3.2.1 Рама летательного аппарата

Первым комплектующим квадрокоптера, который необходимо выбрать – это рама. Рама является важной составляющей квадрокоптера. Рама определяет, каким будет квадрокоптер: гоночным, для съёмки или для выполнения акробатических трюков. Также от рамы зависит, как

квадрокоптер будет вести себя в полёте. Основная задача квадрокоптера - видеосъёмка на небольшую экшн–камеру. Особенностью квадрокоптера должен быть небольшой вес (не более 1.5 кг), поэтому размер рамы не должен быть большим, чтобы было удобнее маневрировать. Поэтому я выбрал раму с маркировкой S500. Это означает, что диаметр рамы равен 500 мм. Высота до нижней пластины 15 см. Рама квадрокоптера с размерами показана на рисунке 3.1.

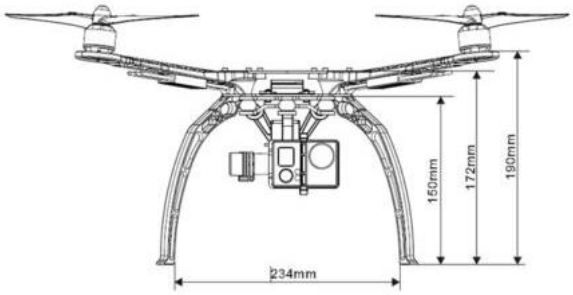

Рисунок 3.1 – Рама квадрокоптера

Ножки изготовлены из пластика, а две пластины – из стекловолокна. Лучи сделаны из пластика с трубкой из стекловолокна посередине для большей прочности. Одна из пластин является платой распределения питания, т.е. внутри разведены дорожки для питания элементов. Это является плюсом, потому что не надо дополнительно покупать или делать распайку платы для распределения тока между элементами, и это к тому же экономит место. Для работы с такой рамой подходят пропеллеры размером 9″ и 10″. Также в комплекте идёт нижняя рама, крепящаяся с помощью трубок из стекловолокна к раме квадрокоптера, на которую можно установить подвес для камеры. На концах лучей расположены отверстия для крепления моторов с отверстиями 19 и 25 мм от центра, поэтому рама универсальна, т.е. можно установить большинство моторов. Вес собранной рамы квадрокоптера равен 380 гр. На рисунке 3.2 представлена рама квадрокоптера в собранном виде.

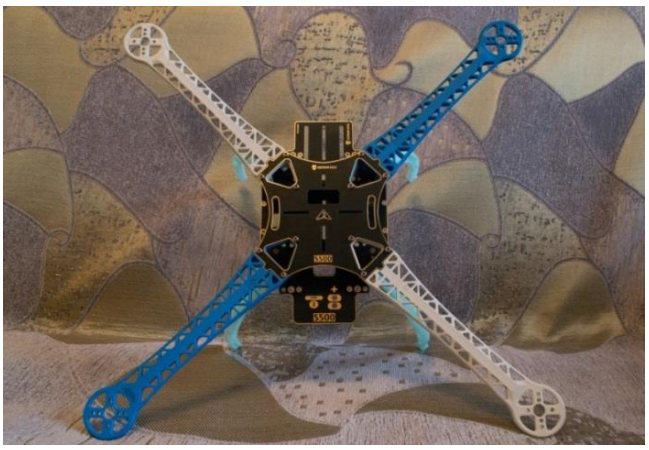

Рисунок 3.2 – Рама квадрокоптера в собранном виде

При выборе рамы важно сразу подсчитать примерный вес квадрокоптера, что упростит в дальнейшем расчёт. Вес квадрокоптера согласно таблице 3.1 равен 1310 гр.

Таблица 3.1 – Таблица весов комплектующих квадрокоптера

Наименование	Количество, шт	Вес одной единицы, гр	Вес, гр
Рама	1	380	380
Моторы	4	55	220
Регулятор скорости мотора	4	10	40
Полётный контроллер	1	50	50
Подвес	1	170	150
Камера	1	50	50
Приёмники и передатчики	1	60	60
Аккумулятор	1	360	360
Всего:			1310

3.2.2 GPS–модуль Ublox Neo–6M

GPS–модуль является важным компонентом квадракоптера, т.к. благодаря GPS–модулю появляются возможности избежать крушения и найти квадракоптер, если он улетит или упадёт. GPS–модуль позволяет обрабатывать режим обработки экстренных ситуаций. Если вдруг напряжение на аккумуляторе достигнет предельных значений, то квадрокоптер автоматически вернётся в точку запуска. Также бывают моменты, когда квадракоптер вылетает за пределы видимости и трансляция с камеры невозможно по причине отдалённости, то можно задать максимальную дистанцию, на которую может улетать квадракоптер. Если он пересечёт её, то квадракоптер возвращается в точку взлёта. Всё это возможно благодаря GPS–модулю. Также появляется возможность использовать следующие режимы полёта:

1) loiter – режим удержания точки (по координате и высоте);
2) rtl – возврат домой, в точку взлета;
3) failsafe – режим спасения, который отправит квадрокоптер лететь домой;
4) geofence – режим, при включении которого можно ограничить радиус и высоту полета.

Я выбрал GPS–модуль фирмы Ublox с маркировкой Neo–6M. Одной из причин выбора данного модуля было наличие в нём компаса, т.к. работа компаса полётного контроллера не устраивала, к тому же компас находился близко к элементам питания, которые создают помехи, поэтому наличие

компаса является большим плюсом. Стоит отметить, что модуль имеет пятидесятиканальное ядро с поддержкой большого количества систем позиционирования: GPS L1 C/A, GLONASS L1 FDMA, QZSS L1 C/A, SBAS WAAS, EGNOS, MSAS, Galileo E1B/C. Также частота обновления спутников варьируется от 1 до 5 Гц, которую можно изменять во время работы модуля. GPS–модуль обладает хорошей точностью: при использовании ГЛОНАСС до 2.5 м, а GPS точность увеличивается до 2 м. Также GPS–модуль показывает хорошее время холодного и тёплого старта – 26 с, а горячий старт занимает 1 с. Также модуль имеет собственную EEprom для хранения настроек и батарейку для поддержания EEprom в рабочем состоянии. Neo–6M имеет интерфейсы UART и USB, что позволяет легко подключить модуль к компьютеру и настраивать его. После получения GPS–модуля нужно его прошить, т.е. записать конфигурационный файл с настройками спутников, чтобы модуль мог принимать от них данные, т.к. модуль идёт «пустым», без прошивки.

3.2.3 Полётный контроллер

Полётный контроллер является мозгом квадрокоптера и отвечает за выполнение таких важных функций, как стабилизация квадрокоптера во время полёта, удержание высоты, полёт по заданному маршруту, обеспечение безопасности полёта, т.е. отвечает за поведение квадрокоптера во время полёта. Поэтому важно выбрать полётный контроллер, который отвечает следующим требованиям:

1) наличие гироскопа, акселерометра, барометра;
2) наличие портов для подключения внешнего GPS–модуля, компаса;
3) хороший процессор;
4) наличие порта для подключения телеметрии, чтобы иметь возможность летать по заданному маршруту;
5) проект должен быть открытым, т.к. существенно влияет на конечную цену контроллера и впоследствии можно самому модифицировать контроллер.

Вышеуказанным требованиям удовлетворял только один полётный контроллер и это Arducopter v2.8, поэтому я и выбрал его. Особенности Arducopter v2.8:

1) наличие трёх осевого гироскопа, акселерометра и барометра;
2) совместимость с большим количеством GPS–модулей;
3) возможность подключения большого количества различных датчиков, которые совместимы с контроллерами Arduino;
4) поддержка датчика уровня заряда батареи;
5) поддержка совместимости с большинством радиоуправляемых приемников PWM– и PPM–сигналов;
6) наличие поддержки OSD–телеметрии (наложение на поток видео–телемерических данных);

7) наличие бортовой флэш–памяти размерностью 16 МБит для автоматической регистрации данных (ведение логов);

8) наличие микропроцессора Atmel ATMega2560;

9) является открытым проектом, т.е. все чертежи контроллера находятся в открытом доступе и можно самому сделать аналогичный полётный контроллер или произвести модификацию контроллера. Модификация заключается в перепайке датчиков или процессора на новые версии или усовершенствованные аналоги.

Полётный контроллер поставляется без прошивки, поэтому нужно написать прошивку и затем прошить контроллер.

3.2.4 Моторы летательного аппарата и расчёт регуляторов их скорости

Очень важно правильно подобрать моторы, которые будут создавать необходимую тягу во время взлёта и полёта, если ошибиться в выборе моторов, то может, будет не хватать тяги, чтобы поднять квадрокоптер в воздух, или моторы слишком мощными и не будут рассчитаны для работы с регулятором скорости мотора и аккумулятором. Поэтому важно правильно рассчитать и проанализировать полученные данные. Также моторы должны поднимать квадрокоптер в воздух при уровне газа не более 50%. Выше я подсчитал примерный вес квадрокоптера, определил, что квадрокоптер нужен для видеосъёмки. Примерный вес квадрокоптера, согласно таблице 3.1 получился 1310 гр. Это важно, т.к. если квадрокоптер взлетает, когда моторы работают в полную силу, то моторы проживут не так долго и могут сгореть во время полёта, что приведёт к поломке квадрокоптера или к несчастному случаю. Т.к. я собираю квадрокоптер для долгого полёта, поэтому тяга должна быть в два раза больше веса. Я выбрал моторы фирмы EMAX с маркировкой mt2213, один из которых представлен на рисунке 3.3.

Рисунок 3.3 – Мотор mt2213

Данные моторы могут работать с 3S–4S аккумуляторами LiPo. Это означает, что можно будет впоследствии использовать 4S аккумуляторы для увеличения мощности квадрокоптера. Данные моторы потребляют в пике 15 А, что важно при выборе регуляторов мотора. Также мощность составляет 200 Вт, а значение Kv равно 935, что важно далее для расчёта. Сначала рассчитаю максимальную тягу, которая создаётся моторами. Рассчитаю эту величину по формуле:

$$K = \frac{T_m}{M_k}, \tag{3.1}$$

где K – коэффициент тяги;

T_m – тяга одного мотора, кг;

$M_к$ – вес квадрокоптера, кг.

Подставлю значения в формулу (3.1) и получу:

$$K = \frac{4 \cdot 850}{1310} = 2.596$$

Получается, что коэффициент тяги больше двух, квадрокоптер взлетит в воздух при уровне газа меньше, чем 50%. Т.е. моторы хорошо подходят для моего квадрокоптера и тяга намного больше требуемого. Тяга зависит не только от мотора, но и от пропеллеров, которые используются. Пропеллеры большего диаметра генерируют большую тягу.

Рассчитаю максимальное количество оборотов мотора при использовании 3S аккумулятора:

$$N = Z \cdot U, \tag{3.2}$$

где U – напряжение аккулуятора, В;

Z – Kv мотора, В/Об;

N – количество оборотов, Об.

Подставлю в формулу (3.2):

$$N = 935 \cdot 12.6 = 11781 \text{ Об}$$

После удостоверюсь, что максимальный ток соответствует документации. Найду его по формуле:

$$I = \frac{P}{U}, \tag{3.3}$$

где I – сила тока, А;

U – напряжение аккумулятора, В;

P – мощность мотора, Вт.

Подставлю значения в формулу (3.3):

$$I = \frac{200}{12.6} = 15.87 \text{ A}$$

Расчётный ток равен 15.87 А, что примерно равно заявленному току. Для использования с 3S аккумуляторами необходимы регуляторы скорости моторов, которые рассчитаны на ток, рассчитываемый по формуле:

$$I = I_m \cdot 125\%, \quad (3.4)$$

где I_m – максимальный ток мотора, А;

I – расчётный ток регулятора скорости мотора, А.

Подставлю в формулу (3.4) значения:

$$I = 15.87 \cdot 125\% = 19.84 \text{ A}$$

Нужно выбирать регуляторы скорости мотора с запасом, чтобы они не перегревались и вероятность аварии была бы низкой. Поэтому я выбрал регуляторы скорости Turnigy Plush 30A, которые рассчитаны на ток 30 А и имеют прошивку Simonk. Регулятор Turnigy Plush 30A показан на рисунке 3.4.

Рисунок 3.4 – Регулятор скорости мотора Turnigy Plush 30A

Преимуществом прошивки является то, что регуляторы имеют более быстрый отклик и моторы развивают большую тягу при взлёте. Регуляторы скорости мотора рассчитаны на работу с 3S–4S аккумуляторами. Таким образом, моторы и регуляторы выбраны с запасом по току и напряжению, что позволяет впоследствии модернизировать квадрокоптер. Если неправильно подобрать регуляторы скорости мотора, то можно случайно сжечь моторы и регуляторы.

3.2.5 Подвес и камера

В качестве камеры я выбрал камеру Mobius, которая произведена в Китае. Основными характеристиками камерами являются:

1) возможность снимать видео с разрешением 1920x1080 с частотой 30 кадр/с. Также может снимать с разрешением 1280x720, но с частотой 60 кадр/с. Видео записывается с расширением mov с использованием кодека компрессии H.264. Запись осуществляется с битрейтом 18000 Мбит · с;

2) встроенный аккумулятор позволяет работать до 8 мин;

3) позволяет не только делать видео, но и ещё снимки. Фотографии снимаются с разрешением 2048x1536, 1920x1080, 1280x720 пикселей. Также имеется возможность снимать фотографии с задержкой от 1 с до 1 мин;

4) имеется слот для microSD. Камера поддерживает карты памяти до тридцати двух гигабайт;

5) камера лёгкая. Вес её составляет 50 гр;

6) камера имеет широкоугольный объектив, т.е. имеет большой угол обзора, что положительно скажется на управлении квадрокоптером.

На рисунке 3.5 показанная камера Mobius для съёмки.

Рисунок 3.5 – Камера Mobius

Подвес необходим для управления положением камеры и стабилизации её во время съёмки не зависимо от наклона квадрокоптера во время полёта. В качестве подвеса я выбрал подвес из алюминия, т.к. алюминий является лёгким материалом и соответствует требованиям по весу. Подвес является двухосевым, т.е. можно изменять положение камеры в двух плоскостях – по вертикали и горизонтали. Двухосевого подвеса достаточно для использования в квадрокоптере среднего размера и для любительской съёмки. Двухосевой подвес показан на рисунке 3.6.

Рисунок 3.6 – Собранный подвес с моторами

В качестве моторов для подвеса я выбрал два мотора D2208 с Kv равным 80, что позволяет плавно перемещать камеру в двух осях. Моторы рассчитаны на вес до 200 гр, таким образом, они спокойно могут удержать прикреплённую камеру. Также я выбрал плату для стабилизации и управления моторами подвеса – это Alexmos BGC. В комплекте с платой идёт датчик уровня, который крепится к нижней подвижной части подвеса, чтобы контроллер мог знать уровень наклона камеры и исправить его. Проект платы не является свободным, поэтому в свободном доступе их нет. Плата может питаться от напряжения 7–12 В, таким образом можно подключить в общую цепь питания. Плата поставляется сразу с прошивкой. Необходимо настроить подвес, чтобы он мог распознавать положения датчика уровня и значения ПИД, от которых зависит скорость и плавность стабилизации.

3.2.6 Приёмник Boscam RC832 и передатчик TS832. Расчёт дальности сигнала

Для удобства управлением и отслеживанием параметров квадрокоптера необходимо осуществлять видеотрансляцию на монитор с камеры, которая находится на квадрокоптере. Для трансляции я буду использовать приёмник Boscam RC832 и передатчик TS832. Я выбрал приёмник Boscam RC832, т.к. приёмник может быть настроен на тридцать два канала, что даёт меньшую вероятность, что выбранный канал при передаче будет использоваться кем–то другим. Также приёмник может питаться от аккумулятора LiPo с напряжением 12 В, таким образом, я подключу к аккумулятору, который я использую для квадрокоптера. Формат выхода видео NTSC/PAL и используется коннектор типа «тюльпан». Получается, что он совместим с любыми мониторами. Также отличием является рабочая частота приёмника – это 5.8 ГГц. Благодаря такой частоте приёмник и передатчик не попадут ни на одну из гармоник пульта, который работает на частоте 2.4 ГГц, а также телеметрии, которая работает на частоте 915 МГц. Передатчик TS832 имеет большую выходную мощность – 0.6 Вт. Передатчик питается также от аккумулятора, который обеспечивает электричеством весь квадрокоптер,

напряжением 12 В. Передатчик рассчитан на работу и с большим напряжением вплоть до 16 В. Таким образом, при замене аккумулятора на более мощный аккумулятор не придётся менять схему питания. Приёмник поддерживает работу с тридцати двумя каналами. Приёмник и передатчик имеют разъём для подключения антенны типа RP–SMA, который совместим с большинством антенн. В комплекте приёмника и передатчика идут антенны длиной одиннадцать сантиметров. На рисунке 3.6 показаны приёмник и передатчик.

Рисунок 3.6 – Приёмник RC832 и передатчик TS832

Для успешного полёта необходимо знать теоретическое максимальное расстояние между приёмником и передатчиком. Очень важно знать эту границу, если не знать её, то можно улететь за неё и потерять сигнал и на большом расстоянии будет сложно, даже невозможно, управлять квадрокоптером, если только не установлена телеметрия, которая позволит определить расположение квадрокоптера по данным GPS–модуля. На дальность действия сигнала оказывают влияние: антенны, мощность передатчика, чувствительность приёмника, электромагнитные помехи, естественные и искусственные препятствия, которые негативно сказываются на дальности сигнала. Для увеличения дальности сигнала необходимо увеличить мощность передатчика, но это не единственный способ. Также с увеличение коэффициента усиления антенны увеличивается и мощность сигнала. Но на мощность влияет ещё и чувствительность приёмника, т.е. его мощность.

Для того, чтобы подсчитать максимальное расстояние сигнала, сначала необходимо рассчитать затухание сигнала. Затухание сигнала – это максимальная потеря мощности сигнала при передаче между приёмником и передатчиком. Затухание сигнала рассчитывается по формуле:

$$L = P_t - S_r + G - L - F, \qquad (3.5)$$

где L – затухание сигнала, дБ;

P_t – мощность передатчика, дБ;

S_r – чувствительность приёмника, дБ;

G – коэффициент усиления антенн, дБ;

L – потери кабеля, дБ;

F – другие помехи, дБ.

Мощность моего передатчика равняется 27.5 дБ, чувствительность приёмника 85 дБ, коэффициент усиления антенны 2 дБ (для двух антенн равен 4 дБ). Потери кабеля равны нулю, т.к. я не использую кабели, а другие помехи я не учитываю, т.к. в разных частях города будут сильно варьироваться, поэтому получаются идеальные условия для передачи данных. Значения мощности передатчика и чувствительности приёмника взяты не максимальные, т.к. характеристики, указанные в документации заведомо завышены. Таким образом, подставив значения в формулу (3.5), получаю следующее:

$$L = 27.5 – (–85) + 4 = 116.5 \text{ дБ}$$

Получается, что максимальная мощность сигнала равняется 116.5 дБ. Теперь можно рассчитать максимальную дальность сигнала. Для этого воспользуюсь формулой:

$$L = 32.44 + 20 \cdot \log f + 20 \cdot \log D, \quad (3.6)$$

где L – затухание сигнала, дБ;

f – частота сигнала, МГц;

D – дальность сигнала, м.

Мне нужно вычислить дальность сигнала, поэтому переносим logD в левую часть, а остальное в правую и умножаем на минус один, чтобы слева D была с плюсом. После чего делю на 20, чтобы избавиться от множителя и затем из определения логарифма раскрою его и получу конечный вид формулы:

$$D = 10^{\frac{(L – 20 \cdot \log f – 32.44)}{20}} \quad (3.7)$$

Подставив значения в формулу (3.7), получу следующее:

$$D = 10^{\frac{(116.5 – 20 \cdot \log 5800 – 32.44)}{20}} = 2.75 \text{ км}$$

Максимальная дистанция сигнала по расчётам составляет 2,75 км, но в реальных условиях она будет меньше.

Для отображения видео используется машинный экран с диагональю семь дюймов. Экран имеет разрешение 800x480 пикселей, а напряжение питания 11–32 В. Подключение осуществляется с помощью коннектора типа «тюльпан». Экран и приёмник питаются от аккумулятора LiPo ёмкостью 2500 мА · ч и напряжением 12.6 вольт.

3.2.7 Расчёт электрических параметров летательного аппарата

Аккумулятор является сердцем квадрокоптера, обеспечивая работоспособность квадрокоптера, давая ему необходимую энергию для полёта. От выбора аккумулятора зависит не только время полёта, но и работа моторов. Если аккумулятор подобран неправильно, то квадрокоптер может не взлететь, а переизбыток максимального отдаваемого тока плохо скажется на жизни моторов и регуляторов скорости мотора. Также стоит правильно выбрать напряжение аккумулятора, если оно будет меньше, то мощности аккумулятора не хватит на взлёт квадрокоптера. Если напряжение выше, то надо смотреть, рассчитаны ли регуляторы скорости моторов и моторы на работу с таким напряжением. Если регуляторы скорости моторов рассчитаны на работу с таким напряжением, то при высоком напряжении будет требоваться меньший ток для взлёта, т.к. мощность останется на том же уровне.

Я выбрал аккумуляторы LiPo Zippy 5000 мА · ч с отдачей тока в 20 С, аккумулятор является 3S и рассчитан на напряжение 12.6 В. На рисунке 3.7 показан аккумулятор Zippy с ёмкостью 5000 мА · ч.

Рисунок 3.7 – LiPo аккумулятор Zippy 5000mAh

LiPo аккумуляторы, по сравнению с LiOn, имеют большую мощность, т.е. могут хранить большую энергию. По документации регуляторы скорости моторов и сами моторы поддерживают работы с аккумуляторами 4S, но для работы с данными моторами необходимы меньшие пропеллеры, т.к. при использовании пропеллеров размера в 12″ и более, то моторы будут греться. Произведу расчёт потребления тока квадрокоптера. Для расчёта понадобятся данные моторов и всей системы. Каждый мотор потребляет по расчётам 16 А.

Остальная электроника, включая контроллер, потребляет около 3 А. По формуле рассчитаю величину максимального тока системы:

$$I = I_m \cdot 4 + I_c, \quad (3.8)$$

где I – максимальный ток потребления всей системы, А;
I_m – потребляемый ток одного мотора, А;
I_c – ток остальных компонентов системы, А.

Подставлю значения в формулу (3.8) и получу:

$$I = 16 \cdot 4 + 3 = 64 + 3 = 67 \text{ А}$$

Рассчитаем максимальный ток, который может выдать батарея по формуле:

$$I_o = C \cdot I_c, \quad (3.9)$$

где I_o –токоотдача аккумулятора, А;
C – коэффициент токоотдачи аккумулятора;
I_c – ёмкость аккумулятора, А.

Подставлю значения в формулу (3.9):

$$I_a = 5 \cdot 20 = 100 \text{ А}$$

Сравню значения I_a и I. Получается, что ток, который может отдавать аккумулятор на много больше тока, который требуется для работы квадрокоптера. Таким образом, аккумулятор подходит для полётов. Время полёта зависит не только от веса, но и от манеры полёта. С увеличением веса моторам необходима большая мощность, чтобы поднять квадрокоптер в воздух, таким образом, увеличивается ток, т.к. напряжение постоянная величина и может только уменьшаться при разряде аккумулятора. Также время полёта зависит не только от параметров квадрокоптера, но и условий полёта. Таким образом, получается, что на время полёта влияет температура окружающей среды, влажность воздуха. Благодаря чему можно только подсчитать только время полёта для конкретных условий окружающей среды. Но можно подсчитать максимальное время полёта, если опустить все эти параметры. Для этого понадобится знать ток висения квадрокоптера, т.е. минимальное значение силы тока, которое необходимо, чтобы квадрокоптер не терял высоту. Тогда время рассчитаю по формуле:

$$t = \frac{I_c \cdot 60}{I_p}, \quad (3.10)$$

где t–время полёта, с;

I_c–ёмкость аккумулятора, А;

I_p – ток висению, А.

Я знаю, что ёмкость аккумулятора равна 5000 мА · ч, а значение тока, при котором квадрокоптер удерживает высоту, находится экспериментально с помощью амперметра и равен 12.3 А. Подставлю значения в формулу (3.10):

$$t = \frac{5 \cdot 60}{12.3} = 1467 \text{ с}$$

Таким образом, теоретическое максимальное время полёта составляет 24 мин 27 с. На практике это время уменьшится в зависимости от условий окружающей среды и манеры полёта.

3.3 Написание прошивки для полётного контроллера

Для написания прошивки я использовал среду Arduino IDE, в которую встроен компилятор со встроенным GCC–компилятором и Atmel AVR, который содержит библиотеки для разработки на C и C++, а также кроссплатформенный компилятор.

Прошивка была написана на языке C, т.к. для микроконтроллера важна скорость выполнения кода, а язык C обеспечивает высокую скорость работы по сравнению с C++.

Написанная прошивка получилась большой, поэтому разберу самую важную функцию прошивки полётного контроллера – стабилизация полёта.

Для реализации стабилизации полёта необходимо сначала разобрать математическую модель стабилизации, на основе которой будут написаны функции стабилизации летательного аппарата.

3.3.1 Математическая модель стабилизации

Построение математической модели начну с разбора, как полётный контроллер обрабатывает команды. Для этого рассмотрю простую систему летательного аппарата – летательный аппарат в двумерном пространстве, где он имеет только один угол движения – крен, и два мотора. Пульт, расположенный на земле, непрерывно передаёт команды, а контроллер их получает и обрабатывает. Задача контроллера – выполнить команды с максимальной для него скоростью, точность регулируется с помощью моторов, учитывая такие факторы, как ветер, смещённый центр тяжести, инерция квадрокоптера и т.д. Контроллер обрабатывает каждую итерацию за определённое время, т.е. раз в несколько миллисекунд контроллер считывает показания датчиков и на основе вычислений отправляет сигнал управляющий сигнал на регуляторы скорости. Приёмник передаёт контроллеру уровень газа, который обозначу как throttle. Приняв за left скорость левого мотора то, следовательно, его скорость:

$$\text{left} = \text{throttle} + \text{force}, \qquad (3.11)$$

где force – реакция квадрокоптера;

throttle – уровень газа, передаваемый приёмником.

Реакция квадрокоптера создаёт момент вращения за счёт того, что левый мотор крутится быстрее на величину force, а правый, следовательно, на величину force медленнее. Величина force может принимать отрицательные значения, тогда будет всё наоборот: левый будет крутиться медленнее, а правый – быстрее. Необходимо вычислить force. Force зависит от текущего угла крена и необходимого угла крена, который поступает с пульта.

Рассмотрю ситуацию, когда с пульта поступает угол крена, равный нулю, а квадрокоптер находится в положении, показанный на рисунке 3.8.

Рисунок 3.8 – Положение квадрокоптера во время удержания горизонта

Таким образом, появляется величина error, которая является разностью между заданным углом и текущим углом крена:

$$error = target\_roll - roll, \tag{3.12}$$

где roll – текущий угол крена;

target_roll – заданный угол крена.

Чем больше разность между желаемым и текущим углами крена, тем реакция должна быть сильнее. То согласно рисунку 3.8 необходимо, чтобы левый мотор крутился сильнее для выравнивания положения квадрокоптера. Таким образом, получу значение force:

$$force = P \cdot error, \tag{3.13}$$

где P – коэффициент усиления.

Если необходимо, чтобы квадрокоптер реагировал сильнее на отклонение от необходимого угла крена, то коэффициент усиления будет больше.

Через несколько итераций квадрокоптер вернётся в необходимое положение. Всё время усилие force и ошибка error имеют одинаковый знак, хоть и уменьшаются по модулю.

Набрав некоторую скорость, квадрокоптер завалится на противоположный бок, т.к. нет силы, которая помешает совершить это. Поэтому необходимо добавить слагаемое, которое будет препятствовать переваливанию в противоположную сторону. Эта величина напрямую зависит от коэффициента усиления, чем выше коэффициент усиления, тем больше должна быть сила, которая будет пытаться остановить переворачивающийся квадрокоптер. Теперь обозначу скорость изменения ошибки как spin. Следовательно, получу:

$$\text{force} = P \cdot \text{error} + D \cdot \text{spin}, \tag{3.14}$$

где D – настраиваемый коэффициент, позволяющий остановить усилие P.

Приму во внимание, что скорость изменения ошибки величина, изменяющая во времени, таким образом, получу:

$$\text{spin} = \frac{\text{d error}}{\text{dt}} \tag{3.15}$$

Подставлю формулу (3.15) в формулу (3.14) и получу пропорционально–дифференциальный вид (дифференциальное слагаемое и пропорциональное):

$$\text{force} = P \cdot \text{error} + D \cdot \frac{\text{d error}}{\text{dt}} \tag{3.16}$$

Ошибка error вычисляется на каждой итерации по формуле (3.12), значения P и D являются настраиваемыми параметрами. Для вычисления производной error необходимо сохранить предыдущее значение error, также знать значение error в текущий момент времени и время между двумя итерациями.

Осталось учесть последний коэффициент. Допустим, что левый край весит больше правого, таким образом, квадракоптер будет немного наклонен в левую сторону. Дифференциальное слагаемое равно нулю, а пропорциональное слагаемое имеет положительное значение, но его не хватает, чтобы вернуть квадрокоптер в горизонтальное положение, ведь левый край весит чуть больше правого. Вследствие чего квадрокоптер будет все время тянуть влево.

Необходимо отследить такие отклонения и исправить их. Особенностью таких ошибок является то, что проявляют они себя со временем. Необходимо ввести третье слагаемое, которое будет хранить сумму всех ошибок error по всем итерациям. Если пропорционального слагаемого не достаточно, чтобы исправить маленькую ошибку, но она все равно есть – постепенно, со

временем, третье слагаемое увеличивается, увеличивая силу force, и квадрокоптер принимает необходимый угол.

Чтобы за одно и тоже время третье слагаемое набирало одно и тоже значение при разных временах итераций, полученную сумму умножу на само время между итерациями. На основе этого получу интегральное слагаемое:

$$Z = \Delta t \cdot \sum \text{error} = \int_0^T \text{error}\, dt, \tag{3.17}$$

где T – текущий момент времени.

Теперь подставлю формулу (3.17) в формулу (3.16) и добавлю коэффициент перед интегральным слагаемым. Получу формулу:

$$\text{force} = P \cdot \text{error} + D \cdot \frac{d\,\text{error}}{dt} + I \cdot \int_0^T \text{error}\, dt \tag{3.18}$$

На основе формулы 3.18 необходимо написать код, который позволяет стабилизировать полёт летательного аппарата. Таким образом, необходимо рассчитать для каждого из трёх направлений движения (крена, тангажа и рысканья) рассчитать силу реакции квадрокоптера.

3.3.2 Написание функций стабилизации летательного аппарата на основе математической модели

Для осуществления стабилизации летательного аппарата выделю последовательность действий:

1) необходимо считать входные параметры с приёмника для трёх каналов;
2) получить данные о текущих значениях крена, тангажа и рысканья;
3) рассчитать отклонения текущих значений от заданных;

Первым делом напишу функцию stabilize_run(), которая позволяет считать и конвертировать входные параметры с приёмника в более удобный вид для дальнейшей работы. Реализация функции показана на рисунке 3.9. Сразу же определю переменные, в которых будут храниться значения с приёмника для каждого из каналов. Функция get_pilot_desired_lean_angles() считывает данные с приёмника по заданному каналу и возвращает значения с него. Функция angle_ef_roll_pitch_rate_ef_yaw() конвертирует значения тангажа, крена и рысканья в радианы, после чего проверяет, не превышен ли максимальный допустимый угол наклона, если да, то задаёт максимальное значение.

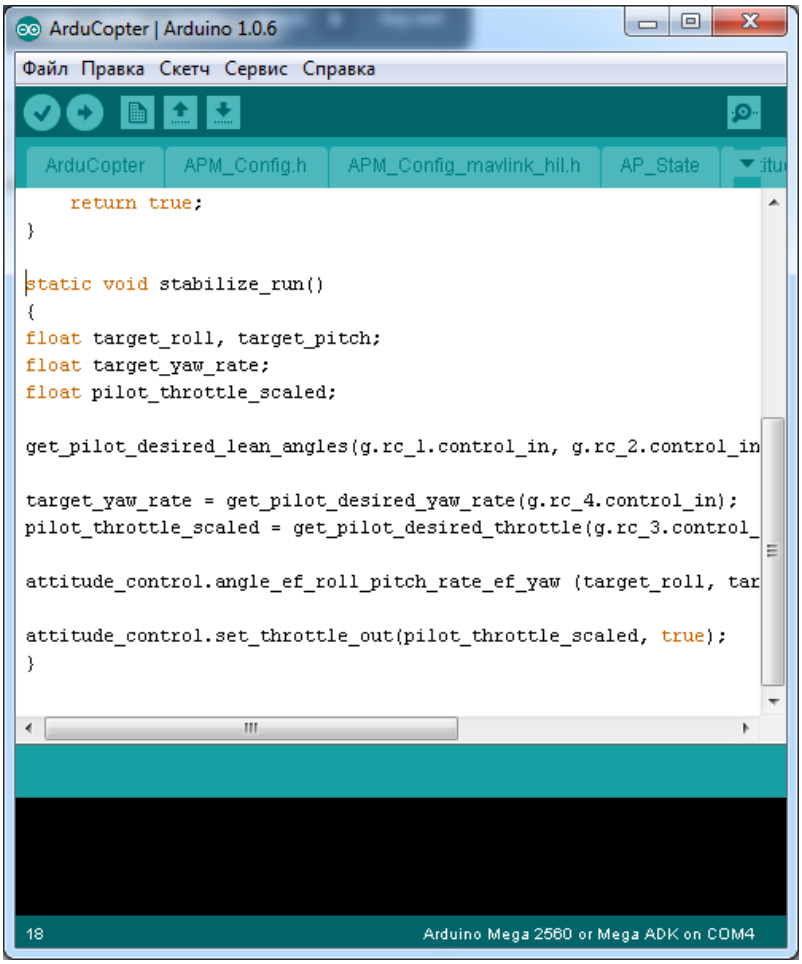

Рисунок 3.9 – Реализация функции stabilize_run()

Функция rate_controller_run() позволяет рассчитать реакцию квадрокоптера на отклонения от заданного уровня произвести стабилизацию, также если отклонения нет, то выполнять поступающую команду. Разберу функцию rate_bf_to_motor_roll(). Функция принимает значение текущего уровня наклона. В переменную считывается значение гироскопа и вычисляется отклонение заданного значения от текущего значения. После чего в переменную output записывается рассчитанная реакция квадракоптера, согласно формуле 3.18 и передаётся дальше в регулятор оборотов мотора. На рисунке 3.10 показана реализация функций rate_controller_run() и rate_bf_to_motor_roll().

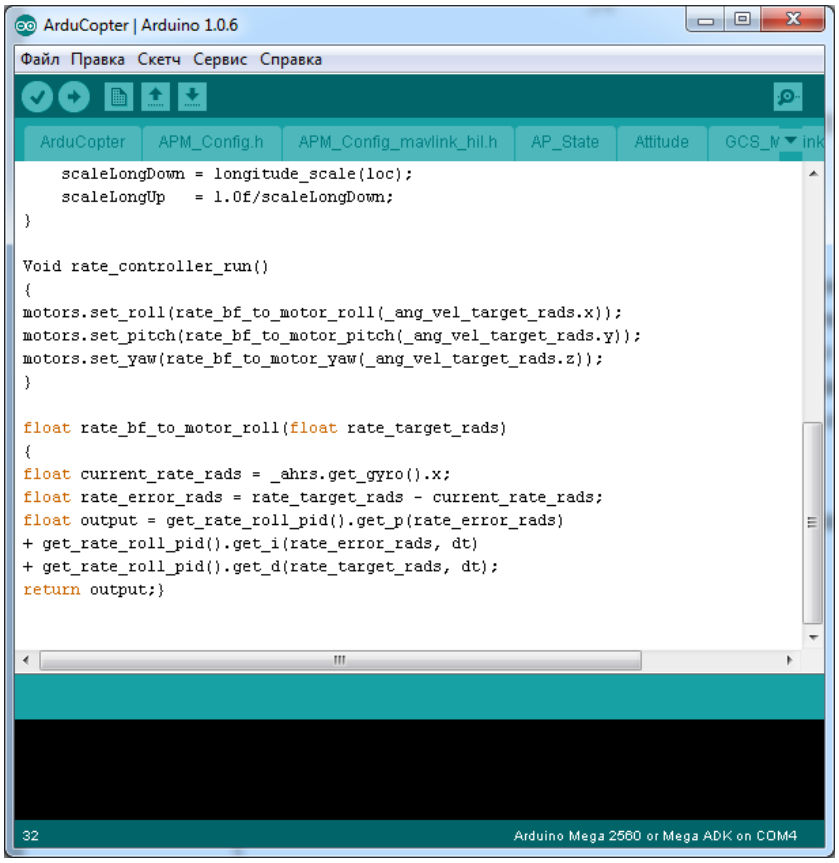

Рисунок 3.10 – Реализация функций rate_controller_run() и rate_bf_to_motor_roll()

На основе математической модели стабилизации найду три составляющие для параметров P, I и D. Для каждого из трёх каналов эти значения находятся отдельно. На рисунке 3.11 показана реализация функций для расчёта параметров P, I и D.

Также прошивка состоит из примерно трёх тысяч строк кода, которые находятся в .cp и .h файлах. В .h определяются переменные и функции, а в .cp уже написана реализация функций, которая соответствует выполняемой функции. В основе прошивки лежит получение данных с датчиков, обработка и принятие решений полётным контроллером на основе этих данных.

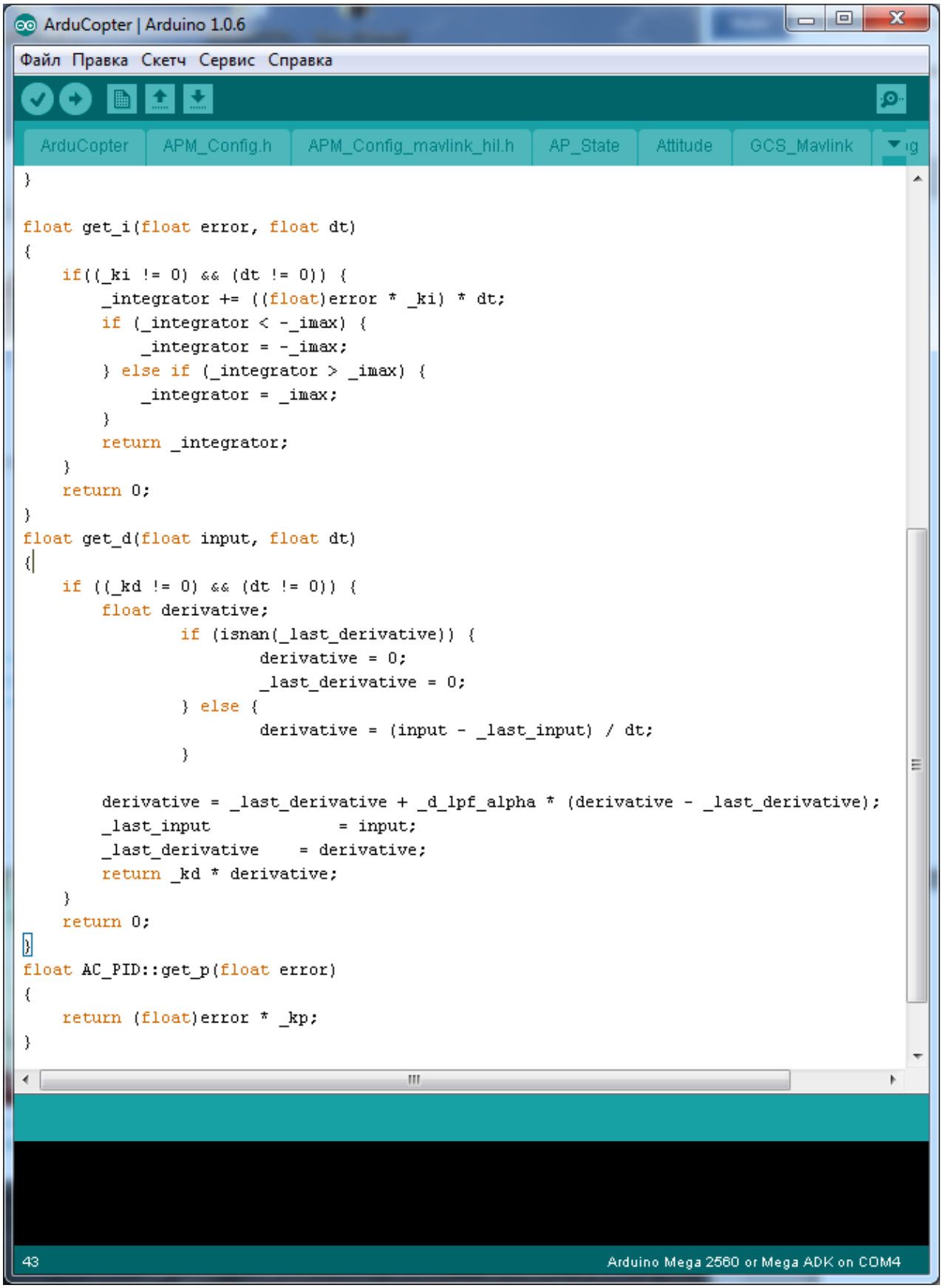

Рисунок 3.11 – Реализация функций для расчёта параметров P, I и D

3.4 Сборка и настройка летательного аппарата

Процесс состоит из трёх частей:

- прошивка GPS–модуля и контроллера;
- сборка квадрокоптера;
- настройка полётного контроллера.

В сборку не входит установка подвеса с камерой, GPS–модуля, телеметрии и видео передатчика. Это всё устанавливается после тестирования. После выполнения всех шагов необходимо провести тестирование квадрокоптера на открытом воздухе. После тестирования производится анализ тестов и происходит доработка.

Сборка квадрокоптера осуществляется на основе принципиальной схемы (рисунок 3.12).

Но перед началом сборки необходимо произвести прошивку GPS–модуля и полётного контроллера.

3.4.1 Прошивка GPS–модуля и полётного контроллера

Перед прошивкой полётного контроллера необходимо прошить GPS–модуль, т.к. я буду использовать полётный контроллер в качестве моста между компьютером и GPS–модулем.

Для прошивки GPS–модуля понадобится:

1) Arduino IDE для написания прошивки и программирования микроконтроллера;
2) программа U–Center для прошивки и настройки GPS–модуля;
3) конфигурационный файл с настройкой GPS–модуля.

В первую очередь необходимо прошить контроллер, чтобы он работал в режиме моста. Для этого открываю программу Arduino IDE. В ней выбираю процессор (мой микропроцессор AtMega2560). Захожу в «Сервис» – «Плата» и выбираю «Arduino Mega 2560». После чего копирую ранее написанный код для переадресации с одного COM–порта на другой, что реализуется проверкой, какой из портов в данный момент передаёт данные и затем перенаправляет вывод на другой порт (рисунок 3.13).

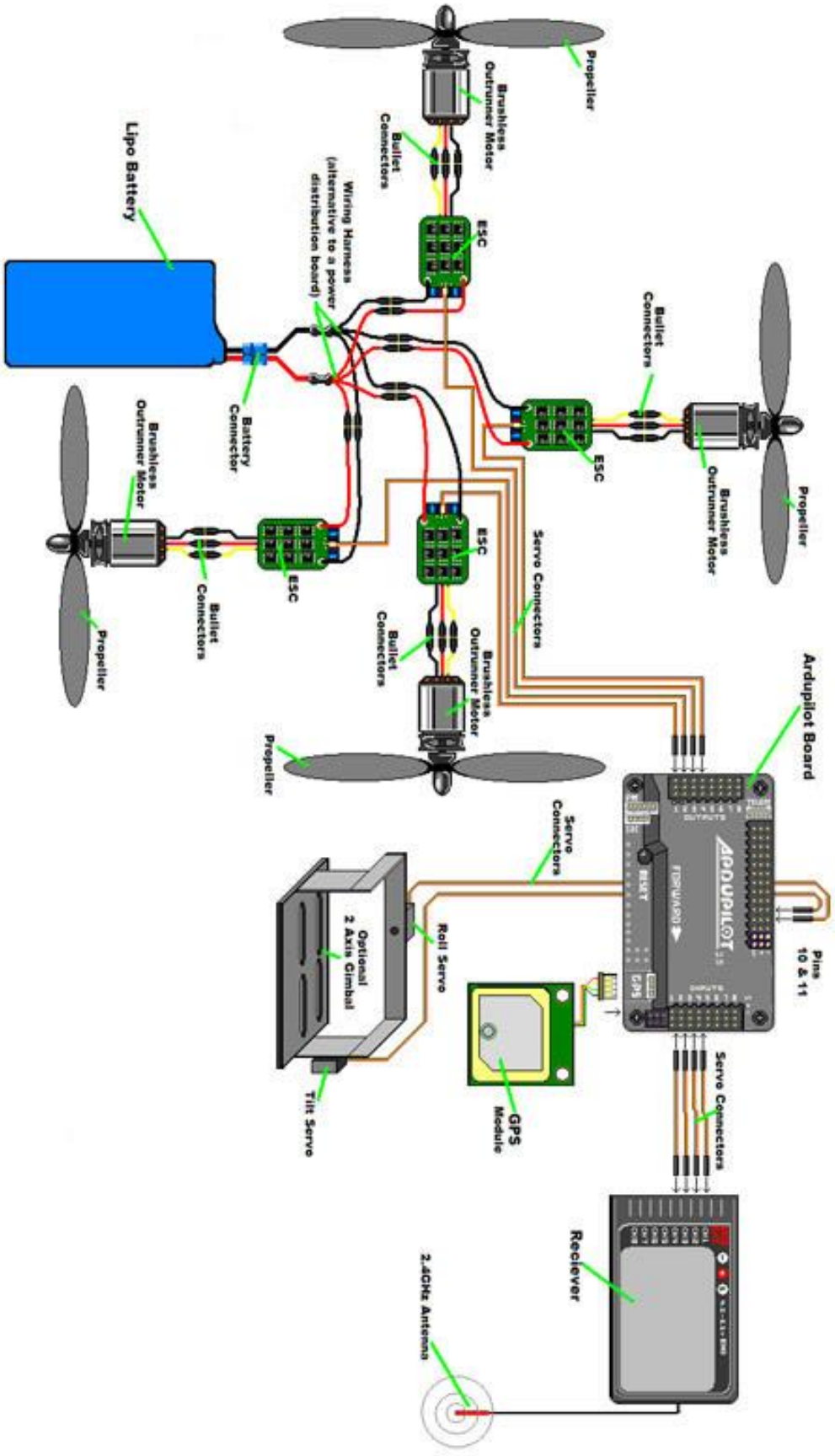

Рисунок 3.12 – Принципиальная схема сборки квадрокоптера

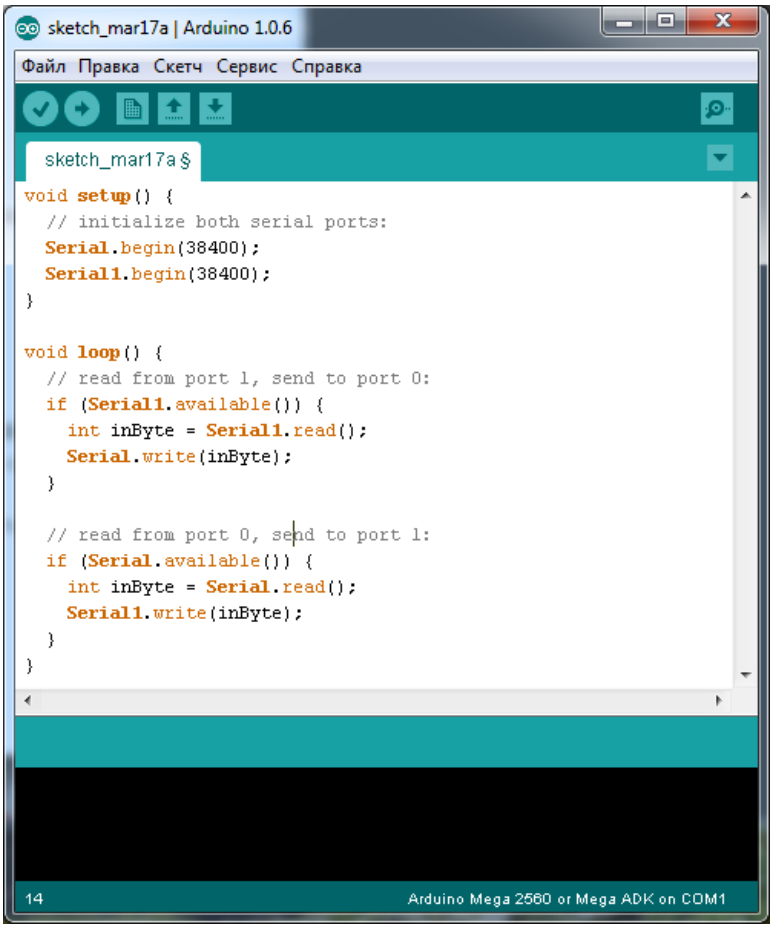

Рисунок 3.13 – Код переадресации с одного COM–порта на другой

Важно, чтобы скорость порта была 38400, т.к. UART GPS–модуля работает на скорости 38400. После прошивки контроллера необходимо скачать программу U–Center, в которой уже буду осуществлять манипуляции по настройке и прошивке GPS–модуля. Запускаю и выбираю скорость порта 38400, что показано на рисунке 3.14. После чего выбираю COM–порт, который буду использовать для работы с GPS–модулем, и подключаюсь к GPS–модулю.

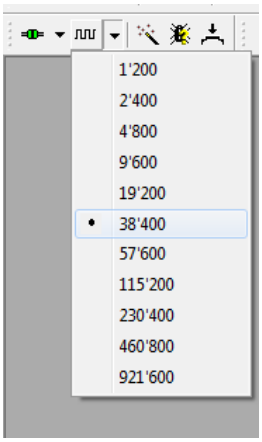

Рисунок 3.14 – Выбор скорости COM–порта

При удачном подключении внизу экрана рядом с номером порта будет мигать зелёным, как показано на рисунке 3.15. Но скорость показана неправильно, поэтому надо настроить UART в GPS–модуле.

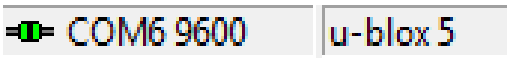

Рисунок 3.15 – Подключение к модулю удалось

Для настройки необходимо пройти в настройки и задать скорость порта и частоту обновления спутников, т.к. нужно, чтобы GPS–модуль как можно быстрее получал свежие данные о местоположении. По умолчанию период опроса равен одной секунде, но это меня не устраивает, поэтому меняю на 200 мс. Настройка частоты обновления показана на рисунке 3.16.

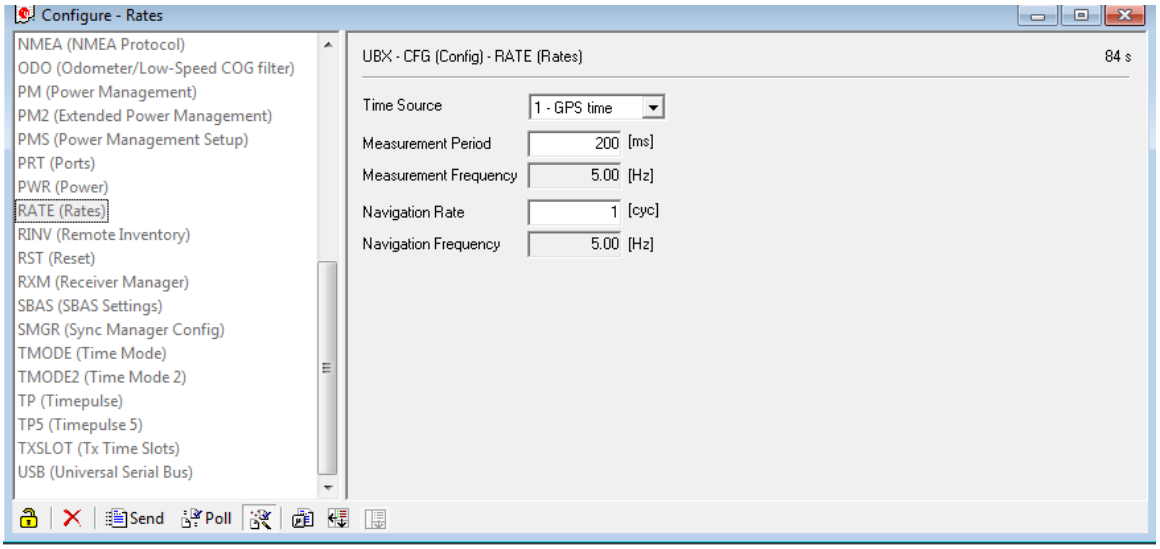

Рисунок 3.16 – Настройка частоты обновления получения сигнала с устройства

После этого настраиваю скорость порта UART. По умолчанию значение равно 57600, но оно должно быть равным 38400, поэтому выставляю 38400, т.к. с другими скоростями прошить модуль будет невозможно. Параметры настройки UART показаны на рисунке 3.17.

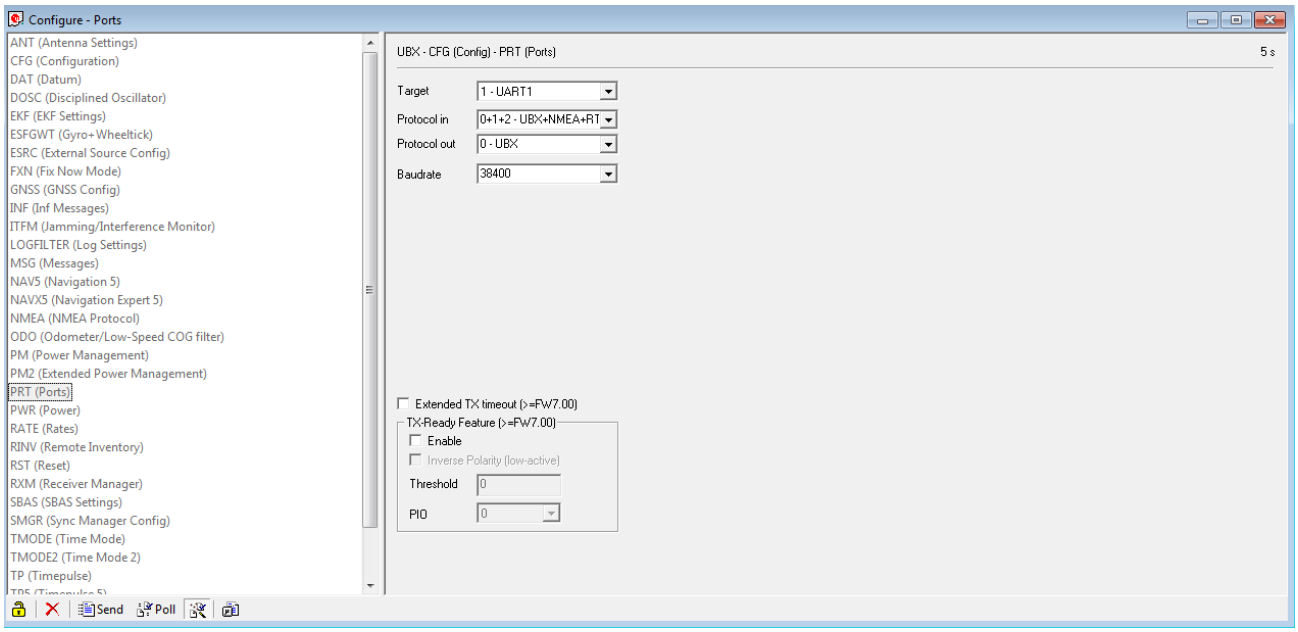

Рисунок 3.17 – Параметры настройки UART

После такой настройки модуля необходимо сохранить настройки, т.к. они должны храниться в EEprom для дальнейшей корекктной работы модуля. Сохранив парметры UART, перепрошью модуль, т.е. запишу конфигурационный файл с настройками спутников. Для этого необходимо нажать меню «Tools» – «GPS Configuration». После чего выберу конфигурационный файл и после чего нажму «File–>GNSS». Окно для выбора прошивки и загрузки файла конфигурации показано на рисунке 3.18.

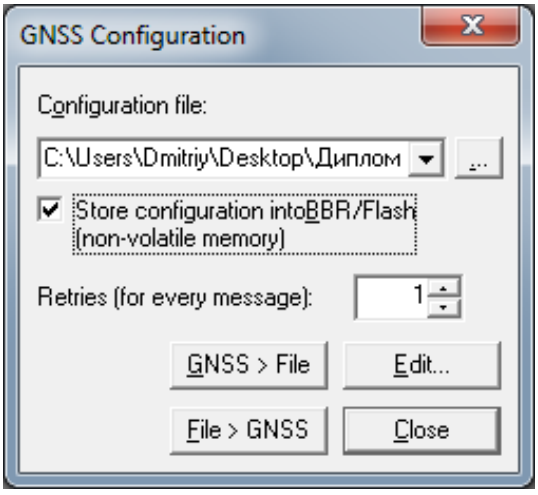

Рисунок 3.18 – Окно выбора прошивки GPS–модуля

После нажатия «File–>GNSS» появится окно с предупреждением о несоответствии версий конфигураций, где надо нажать «Yes». Окно с предупреждением о несоответствии версий конфигураций показано на рисунке 3.19. Если слева в колонке версия не отображается, то необходимо сбросить настройки модуля, т.е. поставить настройки по умолчанию. Для

этого нажимаем меню «Receiver – Action – Revert Config», после чего пробую загрузить настройки снова.

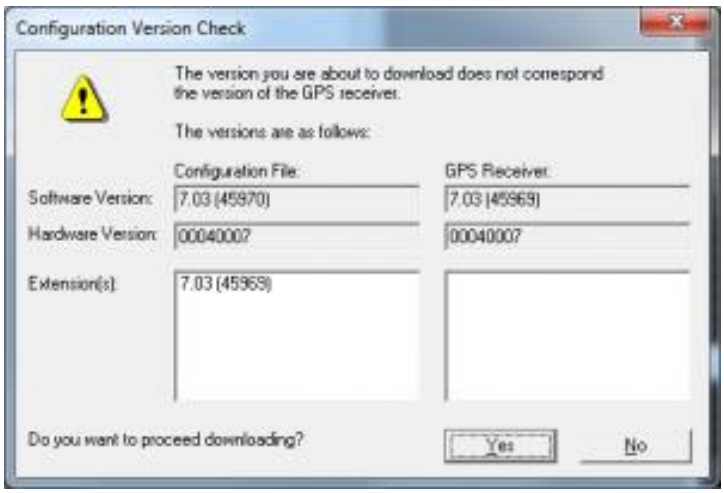

Рисунок 3.19 – Окно с предупреждением о несоответствии версий

Далее начнется процесс загрузки конфигурационного файла в модуль, о чем будет свидетельствовать движение индикатора прогресса, который показан на рисунке 3.20. Модуль успешно прошит, когда окно с индикатором закроется. Если во время загрузки прошивки возникнут ошибки, то задана неправильная скорость COM–порта или есть ошибка в конфигурационном файле.

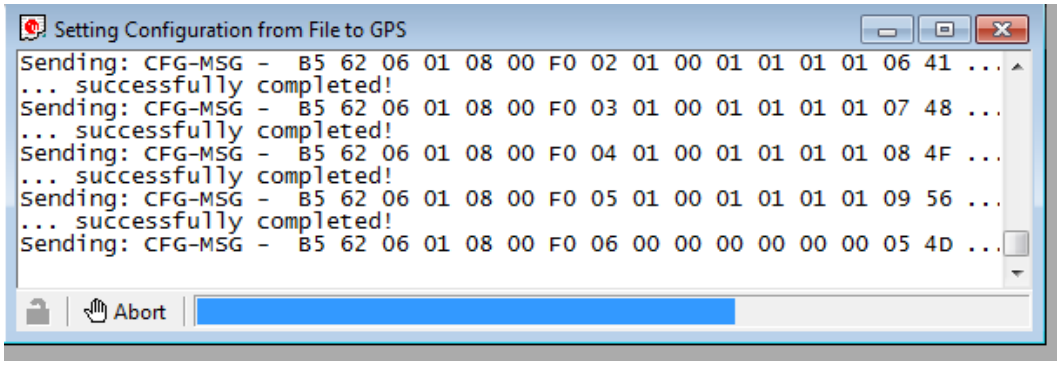

Рисунок 3.20 – Окно прошивки GPS–модуля

После прошивки необходимо проверить работоспособность модуля и точность определения местоположения. Из–за того, что в самом начале осуществляется холодный старт модуля, то модуль должен впервые получить данные от спутников, чему могут мешать естественные и искуственные объекты. Необходимо подождать примерно 30 с. После этого модуль найдёт спутники и определит своё местонахождение. На рисунке 3.21 показана карта с отмеченным местом положения GPS–модуля. На графиках видно количество пойманных спутников, их расположение, уровень сигнала и высота GPS–модуля над уровнем моря. Настройка GPS–модуля закончена.

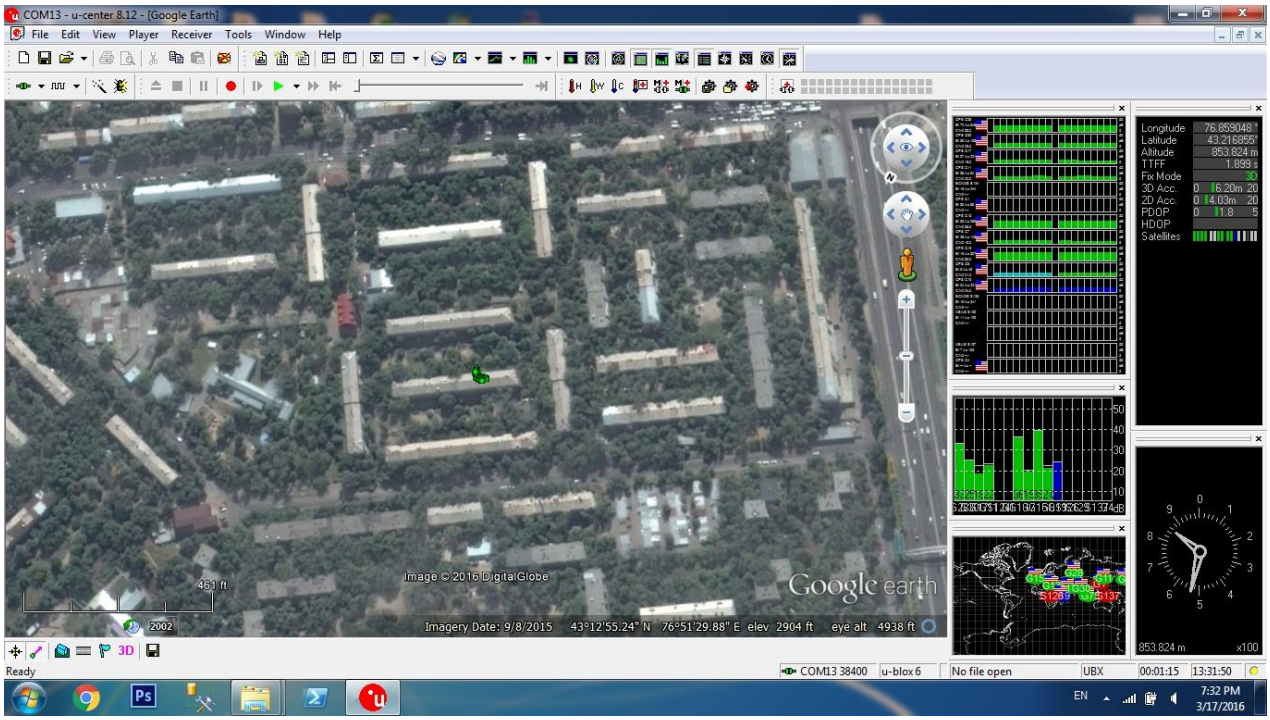

Рисунок 3.21 – Карта с местоположением модуля

После прошивки GPS–модуля необходимо запрограммировать полётный контроллер.

Для прошивки мне понадобятся среда Arduino IDE со встроенным GCC компилятором и Atmel AVR, который содержит библиотеки для разработки C и C++, а также кроссплатформенный компилятор.

В первую очередь необходимо подключить полётный контроллер с помощью USB–кабеля к компьютеру. После установки драйверов начну прошивку полётного контроллера. Надо открыть Arduino IDE и настроить его для работы с полётным контроллером. Для этого необходимо выбрать версию микропроцессора, на который буду записывать прошивку. Мне необходимо выбрать микропроцессор ATMega2560. Выбор микропроцессора показан на рисунке 3.22.

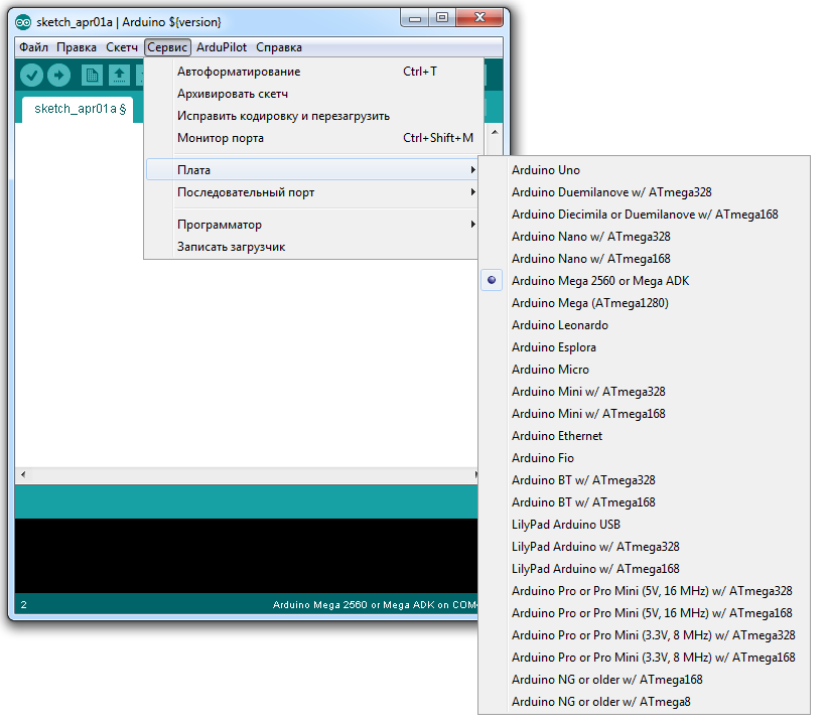

Рисунок 3.22 – Выбор микропроцессора платы

Также необходимо проверить используемый COM–порт. Для этого необходимо открыть «Мой компьютер» – «Управление» – «Диспетчер устройств» и развернуть вкладку «Порты (COM и LPT)». На рисунке 3.23 показана информация о портах и из неё видно, что плата определилась на порту COM4.

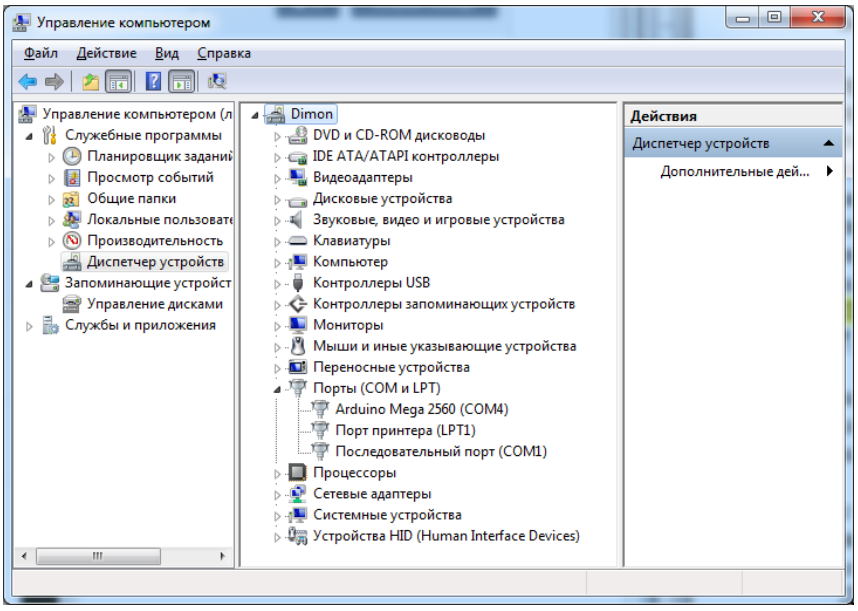

Рисунок 3.23 – Информация о портах

После того, как я убедился, что контроллер подключен к порту COM4, необходимо выбрать данный порт в Arduino IDE, для этого надо пройти «Сервис»–«Последовательный порт» и выбрать COM4. На рисунке 3.24 показан выбор COM–порта.

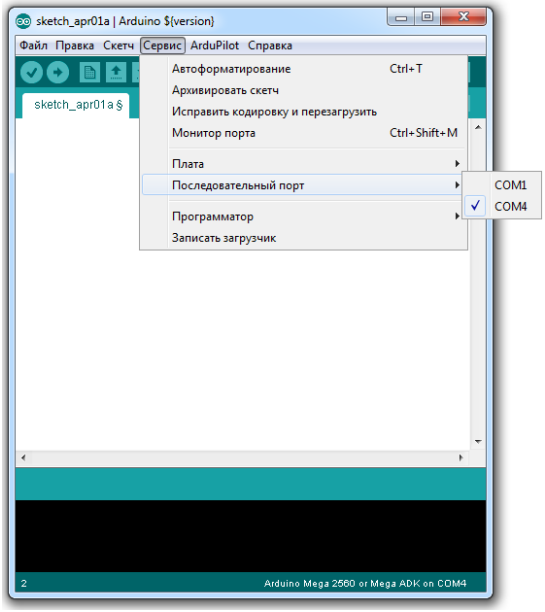

Рисунок 3.24 – Выбор COM–порта

После этих настроек можно открыть проект с готовой прошивкой для контроллера, написанную ранее. На картинке 3.25 показан листинг проекта.

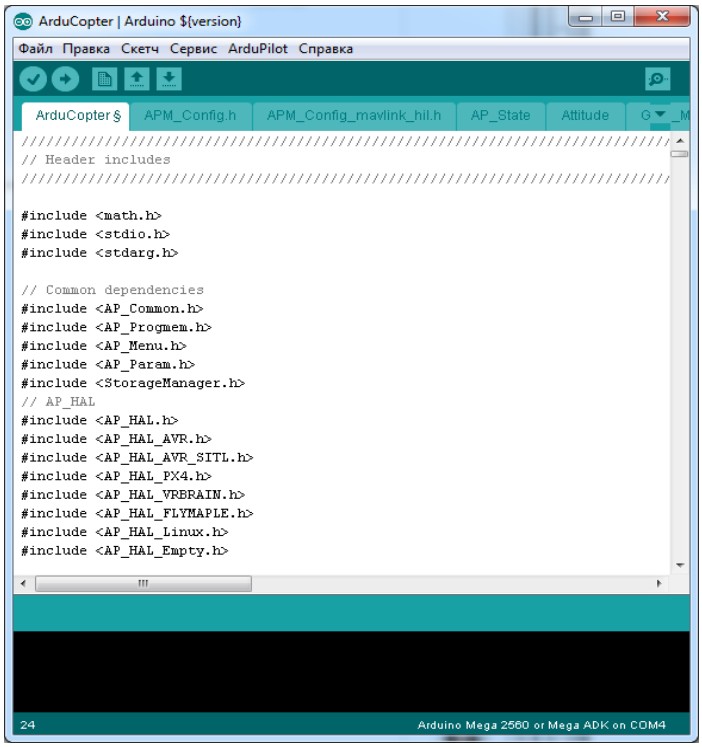

Рисунок 3.25 – Открытый проект с заголовками файлов

Чтобы удостовериться, что не будет проблем при прошивке, надо код сначала скомпилировать, а уже после этого загружать в микропроцессор. Для этого необходимо нажать «Скетч» – «Проверить/Компилировать». При удачном компилировании внизу будет выведено сообщение, которое показано на рисунке 3.26. При неудачной компиляции будет выведена ошибка об этом, сама ошибка, файл, где найдена ошибка и номер строки с номером символа в строке.

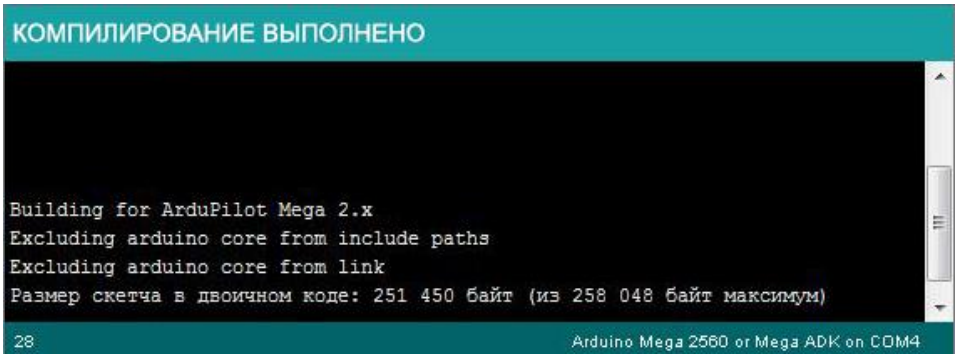

Рисунок 3.26 – Удачная компиляция кода

После удачной компиляции можно загружать прошивку в микропроцессор. Для этого необходимо нажать «Файл» – «Загрузить с помощью программатора». При удачной загрузке появится сообщение, что загрузка выполнена. В других случаях появятся сообщения об ошибке. На рисунке 3.27 показана успешная прошивка микроконтроллера.

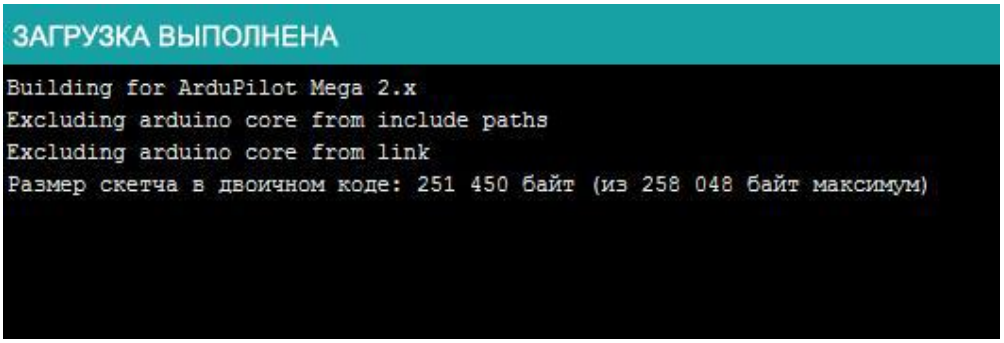

Рисунок 3.27 – Успешная прошивка микроконтроллера

Прошивка микроконтроллера прошла удачно, после чего контроллер готов к настройке для полёта квадрокоптера.

3.4.2 Сборка летательного аппарата

Сборка осуществляется согласно рисунку 3.12, который показан в главе 3.4.

Сборку квадрокоптера я начал не со сборки рамы, т.к. она будет собираться последовательно, а с укорачивания проводов регуляторов скорости моторов и пайки коннекторов. Пайка осуществляется паяльной станцией. Я

использовал коннекторы типа «банан» для соединения регуляторов скорости с моторами. После обрезки лишней длины проводов и пайки коннекторов, которые отсутствовали. Я укомплектовал коннекторы в термотрубки, потому что коннекторы типа «банан» представляют собой цилиндр, выполненный из металла. В коннектор с одной стороны впаивается провод, а с другой – вставляется второй коннектор. На рисунке 3.28 показан регулятор оборотов после всех манипуляций с ним.

Рисунок 3.28 – Готовый к монтажу коннектор

Эти все действия повторяю для остальных трёх регуляторов скорости. После манипуляций с регуляторами скорости, необходимо их припаять к нижней пластине квадрокоптера, которая также является и платой распределения питания. Для начала необходимо взять мультиметр и «прозвонить» выходы, чтобы быть точно уверенным, что смогу их использовать. «Прозвоном» удалось установить, что выходы не глухие и теперь можно паять. Также необходимо припаять модуль питания контроллера, который показан на рисунке 3.29.

Рисунок 3.29 – Модуль питания котроллера

При пайке я учитывал, что регуляторы оборотов будут находиться в нижней части лучей, что позволяет лучше охлаждаться, и если вдруг хомут не выдержит и будет срыв, то он не попадёт в лопасть и не будет падения. Также важно было расположить провода ближе к центру, потому что провода будут

проходить через небольшое отверстие, которое сформируется между двумя пластинами квадрокоптера, и это пространство позволит провести через него провода. При пайке модуля питания я использовал много припоя, т.к. провода, которые используются, толстые и рассчитаны на большой ток, являются мостом между электроникой квадрокоптера и аккумулятором, вследствие чего нельзя допускать, чтобы провод под нагрузкой отсоединился. Спаянная нижняя пластина показана на рисунке 3.30.

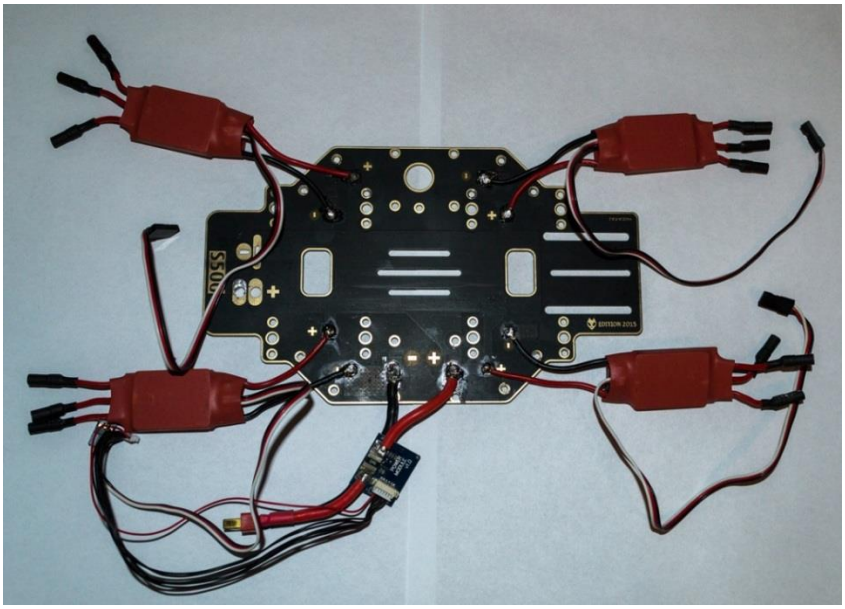

Рисунок 3.30 – Спаянная нижняя пластина с регуляторами скорости

Также важно не забыть прикрутить в этот момент крепления для трубок, на которых будет держаться подвес. Если этого не сделать, то в собранном состоянии качественно это сделать будет трудно, поэтому не буду усложнять и прикручу сразу. Теперь необходимо достать лучи квадрокоптера и к ним с помощью болтиков прикрутить моторы. Луч показан на рисунке 3.31.

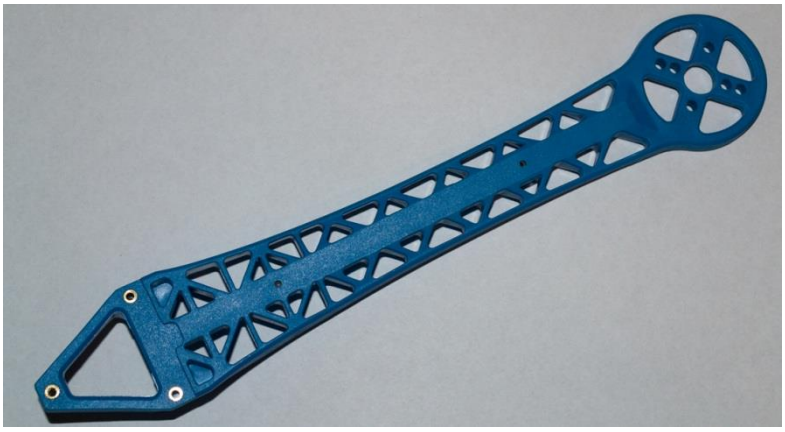

Рисунок 3.31 – Луч квадрокптера

Как и было сказано раньше, что моторы крепятся болтиками диаметром 3 мм. Мотор прикручивается сзади луча. Обязательно нужно пропустить через отверстие перед мотором провода, т.к. они слишком длинные и обрезать не стоит, поэтому их удобнее пропустить между отверстиями в лучах. На рисунке 3.32 показан прикрученный мотор.

Рисунок 3.32 – Мотор, прикрученный к лучу квадрокоптера

Следующим шагом будет крепление лучей к верхней пластине с помощью болтов диаметром 5 мм. Каждый луч с верхней пластиной соединяется тремя болтами. Соединённые лучи показаны на рисунке 3.33.

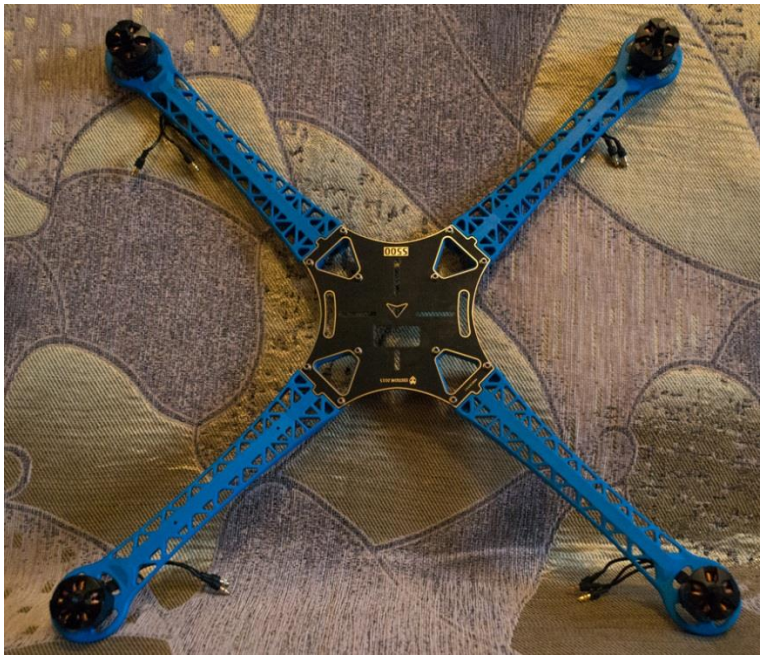

Рисунок 3.33 – Верхняя часть квадрокоптера

Теперь необходимо соединить верхнюю и нижнюю части квадрокоптера. Временно соединяю их болтами, т.к. впоследствии туда я прикреплю ножки. После временной сцепки необходимо соединить провода моторов с регуляторами скорости. Необходимо учитывать, в какую сторону будет крутиться мотор. На рисунке 3.34 показано направление вращения моторов квадрокоптера.

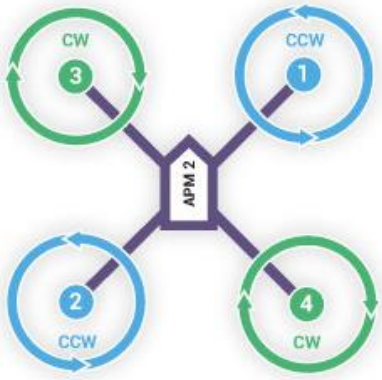

Рисунок 3.34 – Направление вращения моторов квадрокоптера

Согласно данной схеме необходимо, чтобы первый и второй моторы вращались против часовой стрелки, а третий и четвертый – по часовой стрелке. Т.к. я использую бесколлекторные моторы, то для изменения направления вращения необходимо поменять два крайних провода местами, как показано на рисунке 3.35.

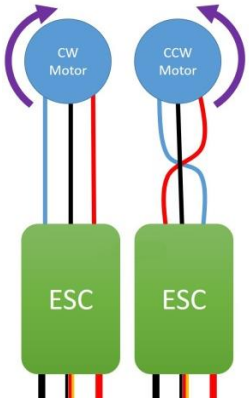

Рисунок 3.35 – Направление вращения мотора в зависимости от подключения к регулятору скорости

Таким образом, первый и второй моторы подключаются в прямой последовательности к регуляторам скорости моторов, а моторы три и четыре – в обратной последовательности. Также прикрепляю регуляторы скорости моторов к лучам с помощью хомутов. На рисунке 3.36 показан прикреплённый к лучу регулятор скорости, который подключен к мотору.

Рисунок 3.36 – Установленный регулятор оборов

После соединения двух частей квадрокоптера необходимо установить ножки, т.к. без ножек квадрокоптер заваливается на один бок и с ним неудобно работать. Поэтому необходимо их установить. Высота ножек 15 см, что позволяет установить подвес без проблем. Каждая ножка крепится на два болта. После установки ножек необходимо приклеить на двухсторонний скотч плату, которая поглощает вибрации. Это необходимо, чтобы при вибрации, создаваемой моторами, не фиксировались ненужные колебания, которые контроллер захочет исправить, что приведёт к неустойчивому состоянию. На эту плату также на двусторонний скотч креплю контроллер. Важно прикрепить контроллер в правильном положении, чтобы передняя часть контроллера смотрела в сторону передней части квадрокоптера. Это обусловлено тем, что на контроллере установлен гироскоп и благодаря ему контроллер понимает, где передняя часть, а где задняя. На рисунке 3.37 показан установленный полётный контроллер, установленный на квадрокптере.

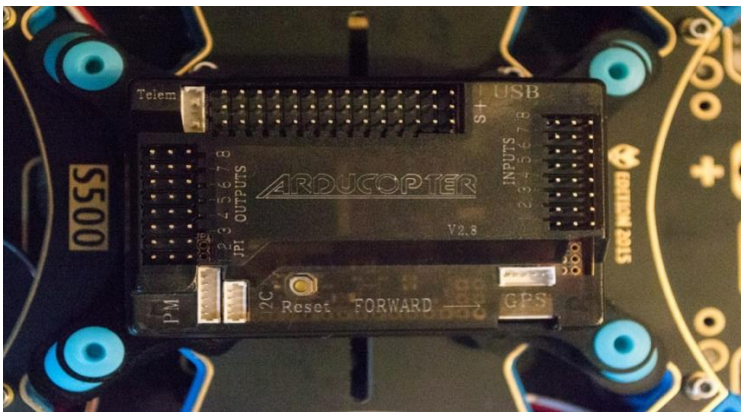

Рисунок 3.37 – Установленный полётный контроллер

Далее необходимо приклеить на двусторонний скотч приёмник сигнала. Затем приёмник необходимо подключить к полётному контроллеру. Чтобы рационально использовать место на квадрокоптере, приёмник было решено приклеить его справа от контроллера, он как раз помещался на небольшой

свободной части площадки. На рисунке 3.38 показан установленный приёмник.

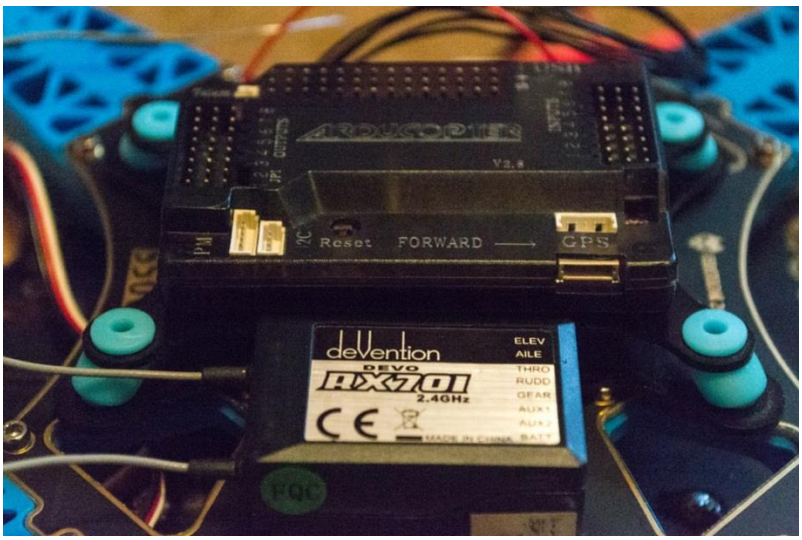

Рисунок 3.38 – Установленный приёмник сигнала

Приёмник установлен, и теперь необходимо правильно подключить каналы к контактам контроллера полёта, которые помечены как «Inputs». На рисунке 3.39 показано описание контактов полётного контроллера.

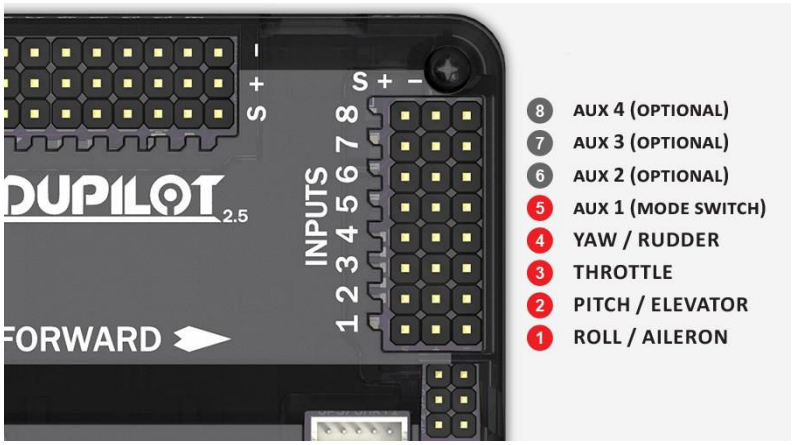

Рисунок 3.39 – Описание входных контактов контроллера

После этого необходимо подключить моторы к контроллеру. На рисунке 3.40 показано, как необходимо подключить моторы к полётному контроллеру.

Рисунок 3.40 – Соответствие контактов контроллера расположению моторов

Подключение осуществляется с помощью проводов небольшой длины, чтобы лишнее не мешало при полёте. Провода с двух концов заканчиваются коннекторами, что упрощает подключение к контроллеру.

3.4.3 Настройка полётного контроллера

После сборки квадрокоптера необходимо подключить контроллер с помощью USB–порта к компьютеру для дальнейшей настройки с помощью программы Mission Planner. Для этого необходимо открыть окно с программой и в правом углу нажать «Connect», выбрав скорость подключения 115200 и COM–порт, у меня COM6. На рисунке 3.41 показана часть окна программы Mission Planner, которая отвечает за подключение.

Рисунок 3.41 – Параметры подключения к контроллеру

Во время подключения программа считывает параметры квадрокоптера и впоследствии их можно изменять. На рисунке 3.42 показано окно программы.

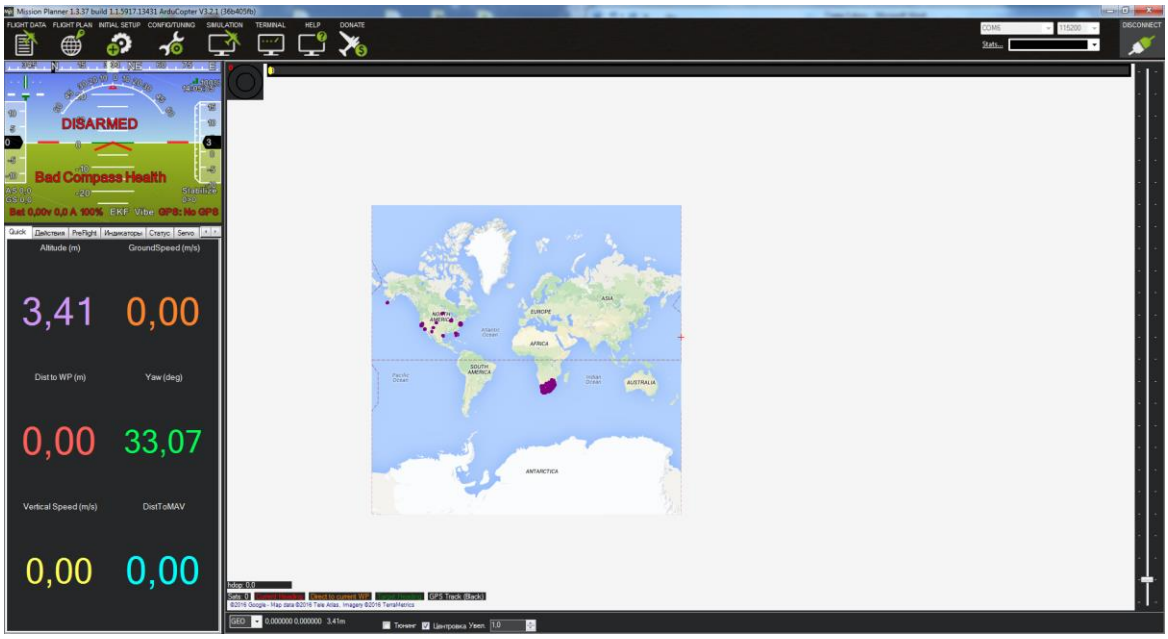

Рисунок 3.42 – Интерфейс программы Mission Planner

Для настройки мне понадобятся две вкладки – это «Initial Setup» для настройки железа и «Config/Tuning» для настройки поведения квадрокптера. Для начала необходимо настроить уровень акселерометра и максимальный угол наклона квадрокоптера. Для этого перехожу в «Initial Setup» – «Accel Calibration» и нажимаю «Calibrate Level», что позволит задать уровень, который будет держать квадрокоптер при полёте. Теперь необходимо откалибровать акселерометр. Для этого надо нажать «Calibrate Accel» и поставить квадрокоптер в шесть разных положений и после изменения нажимать любую клавишу. На рисунке 3.43 показаны положения квадрокоптера во время калибровки акселерометра.

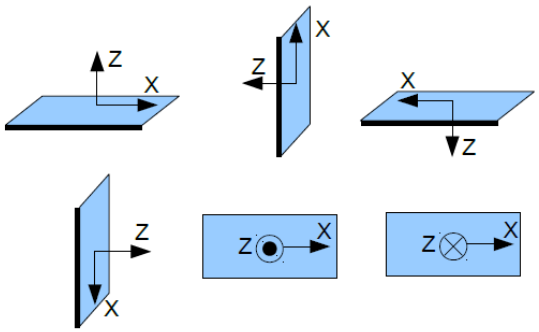

Рисунок 3.43 – Расположение осей квадрокоптера во время калибровки акселерометра

Теперь необходимо откалибровать радиоканалы для приёма сигнала, чтобы контроллер знал, какие параметры являются минимальными и максимальными. Для этого необходимо перейти «Radio Calibration», включить пульт и нажать «Calibrate Radio». После чего перевожу все стики в

минимальное значение, а затем в максимальное. После чего нажимаю «Calibrate» и значения записываются в контроллер. На рисунке 3.44 показано окно калибровки радио.

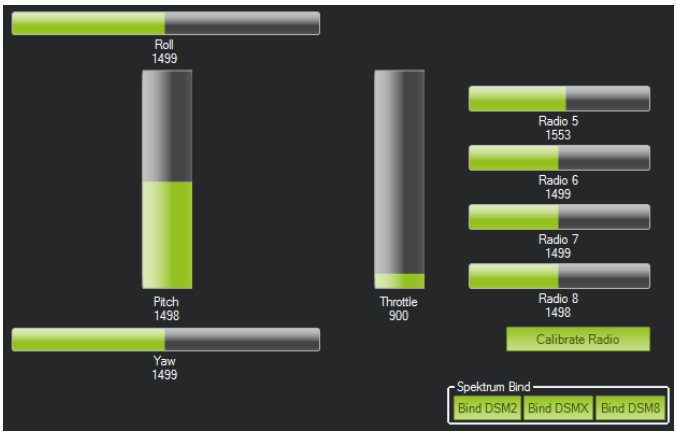

Рисунок 3.44 – Откалиброванные значения радиоприёмника

После калибровки радио необходимо настроить ПИД контроллера. Для этого открываю «Config/Tuning» – «Extended Tuning». Данные параметры настраиваются примерно, после чего значения после полёта будут меняться, пока не будет найден баланс, при котором квадрокоптер будет плавно передвигаться и возвращаться в первоначальный уровень. На рисунке 3.45 показаны настройки ПИД, но они слишком агрессивны для полёта, поэтому они будут впоследствии изменены.

Рисунок 3.45 – Настройка ПИД контроллера полётов

Перед полётом необходимо откалибровать регуляторы скорости. Регуляторы скорости мотора идут ненастроенными, т.е. перед полётом их необходимо настроить. Настройка проходит в автоматическом режиме, но для

этого нужно войти в режим программирования. Настроить регуляторы скорости моторов можно двумя способами: непосредственно подключить их к полётному контроллеру (тогда настраиваются все регуляторы скорости одновременно и необходим настроенный квадрокоптер) или каждый регулятор подключать к приёмнику и настраивать. Я использую первый способ, т.к. он быстрее и можно сразу проверить правильно ли настроились регуляторы скорости или придётся их перенастраивать. Всё ниже написанное необходимо выполнять со снятыми пропеллерами. Сначала необходимо включить пульт и стик газа выставить на максимум, после чего подключить батарею к квадрокоптеру. Затем на полётном контроллере начнут мигать синий и красный диоды в разнобой, символизируя, что при следующей подаче напряжения контроллеры будут в режим программирования. После этого отключаем питание и снова подаём, при этом не опускаем стик газа вниз. После включения регуляторы скорости издадут сигналы и нужно ждать, пока не будет три пика, которые будут звучать как ди–да–ди. После этого необходимо перевести стик в нижнее положение, прозвучат сначала два сигнала, которые обозначают, что задаётся минимальное значение газа, и уже затем прозвучит один длинный сигнал, который обозначает, что регуляторы настроены и готовы к полёту. Но перед этим необходимо проверить это. Для начала необходимо подать газ, т.е. постепенно переводить стик в положение, и моторы будут реагировать, начиная движение. Всё хорошо, но нужна последняя проверка. Отключаем питание квадрокоптера, переводим стик газа в нижнее положение и после этого включаем питание квадрокоптера. Подожду 10 с и нужно разблокировать полётный контроллер. Очень важно, чтобы действия выполнялись с снятыми пропеллерами. После разблокировки контроллер проверит важные датчики: гироскоп, барометр, акселерометр и GPS–модуль. Если все прошло успешно, то двигатели должны одновременно запуститься. Если двигатели запустились неодновременно или один из двигателей не запустился, то необходимо снова перенастроить регуляторы скорости по вышенаписанному алгоритму. Также важно, чтобы батарея была максимально заряжена, чтобы регуляторы запомнили максимальное значение тока, которое необходимо мотору.

После описанных действий можно приступить к тестированию собранного аппарата на открытой местности.

3.5 Тестирование собранного летательного аппарата

Тестирование проводилось в открытом поле вдалеке от людей, чтобы если вдруг пойдёт что–то не так, то никто из людей не пострадал. Тестирование проводилось три раза, т.е. три полёта, пока все три аккумулятора не разрядились. Во время тестирования проводилась отладка настроек полётного контроллера.

Необходимо было произвести три теста:

- взлёт, посадка и удержание высоты;
- маневрирование на небольшой высоте;

– маневрирование на высоте около 50 м и расстоянии 40 м.

Первый тест это взлёт, посадка и удержание квадрокоптера на одной высоте при газе висения. Во время этого теста была солнечная и безветренная погода. Таким образом, получаются хорошие условия для тестирования, т.к. легче определить правильно ли квадрокоптер держит высоту. В ветреную погоду такое тестирование не возможно, т.к. ветер будет сносить квадрокоптер. Во время взлёта важно, чтобы квадрокоптер хорошо реагировал на стик газа, если он будет плохо реагировать, то нужно увеличить значение П в настройках ПИД, Но мой квадрокоптер хорошо реагирует на значение П, поэтому изменять его не буду. От реагирования стика газа также зависит и посадка, будет она мягкой или нет. После взлёта я поднял квадрокоптер на высоту примерно 2 м, чтобы если вдруг случится падение, то он бы упал с небольшой высоты, получив малые повреждения. С такой высоты удобно наблюдать за поведением квадрокоптера. Квадрокоптер немного ведёт назад, т.е. надо снова откалибровать уровни гироскопа или это из–за того, что сместился центр тяжести, т.к. аккумулятор находится сзади. Эта проблема впоследствии решится после установки подвеса с камерой, и расположение центра тяжести придёт в норму. Также это этап тестирования позволял определить расчётное время висения квадрокоптера, которое было рассчитано ранее. Также важно провести этот этап как подготовительный к полёту. Моторы квадрокоптера новые и не использовались ни разу, поэтому важно, чтобы они могли поработать при небольшой нагрузке и впоследствии вероятность аварии будет меньше.

Второе тестирование – маневрирование на небольшой высоте, которое позволит настроить ПИД для трёх каналов: крен, тангаж и рыскание. Тестирование проводилось на высоте около 3 м и на расстоянии от пилота примерно 5 м. В данном тесте ветер играет положительную роль, т.к. при ветре на небольшой высоте можно представить, как себя будет вести квадрокоптер на больших высотах при ветре. Во время зависания квадрокоптера необходимо перемещать стики вперёд или назад и смотреть, как реагирует на них квадрокоптер. Если квадрокоптер реагирует резко на изменение положения стика, то нужно уменьшить значение П, а если возвращается резко, то уменьшить значение Д по необходимому каналу. Мой квадрокоптер резко реагировал на изменение положения стика и возвращался в исходное положение после манёвра резко. Поэтому было решено чуть уменьшить значения П и Д, т.к. на большей высоте, если квадрокоптер будет медленно реагировать, то при сильном ветре его может снести и произойти несчастный случай.

Третье тестирование проводилось на высоте примерно 50 м, чтобы определить, как ведёт себя квадрокоптер на такой высоте при ветре, т.к. на небольших высотах поле было обнесено деревьями, что мешало проникновению ветра. Квадрокоптер при среднем ветре не сносило в бок, что очень хорошо, но сажать было сложнее, т.к. ветер задувал под квадрокоптер и

старался поднять его, что создавало опасность падения. Но квадрокоптер хорошо показал себя на такой высоте. Скорости полёта хватало, чтобы противостоять ветру и лететь против него.

Во время всех этапов теста все показания датчиков записывались в флэш–память для дальнейшего анализа.

3.6 Анализ тестовых полётов и доработка летательного аппарата

После полётов важно проанализировать результаты тестирования. Первое на что обращу внимание – это смещенный центр тяжести. Из–за чего квадрокоптер ведёт назад, т.к. в сторону хвоста смещён центр тяжести. Путь решения – это установка подвеса, которая позволит уравновесить квадрокоптер и сместить центр тяжести в нужную сторону. Вес подвеса вместе с камерой 200 гр и подвес будет смещён вперёд, что позволит уравновесить квадрокоптер.

Также необходимо просмотреть файлы, в которых записаны показания датчиков. Меня больше волнует напряжение на контроллере, т.к. контроллеру необходимо напряжение не выше 5 В, а при напряжении меньше, чем 4.9 В будет вести себя нестабильно и будет велика вероятность отключения контроллера и падение его. Поэтому проанализирую данные с датчика замера напряжения. График показан на рисунке 3.46.

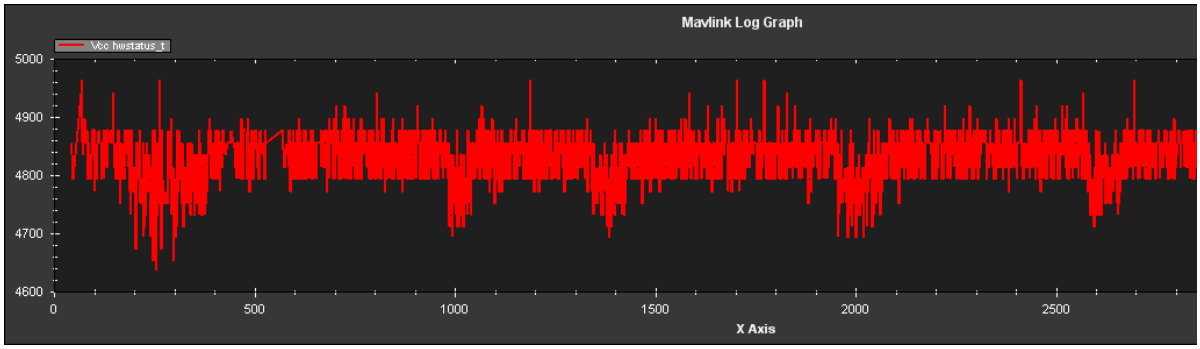

Рисунок 3.46 – Напряжение на контроллере

Как видно из рисунка выше, что напряжение варьируется в большом диапазоне, что недопустимо для полётного контроллера. Причиной является установленный модуль телеметрии, которому тоже необходимо 5 В и он питается от контроллера, что в большой вероятности является причиной такого напряжения. После тестового полёта без подключенной телеметрии проанализирую данные с напряжением. Без подключенной телеметрии напряжение стабилизируется и находится на уровне 4.984 В. На рисунке 3.47 показано напряжение без подключенной телеметрии.

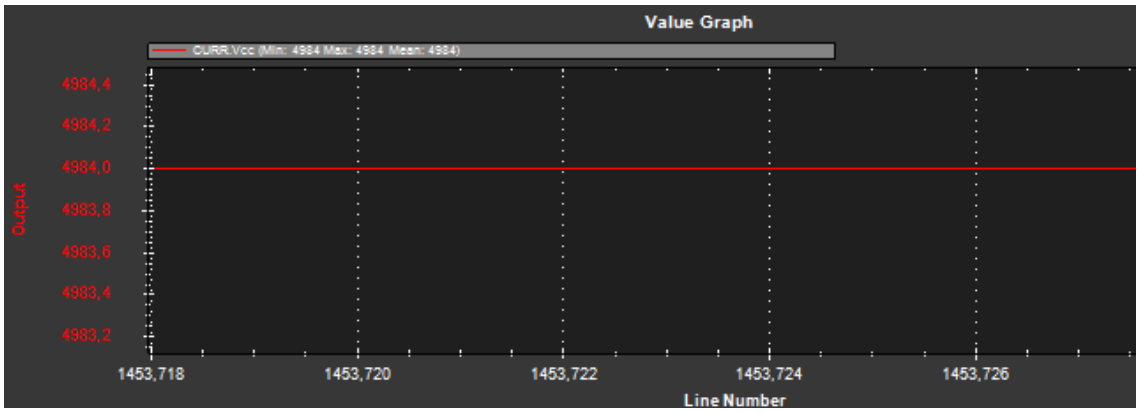

Рисунок 3.47 – График напряжения на контроллере

Чтобы решить эту проблему, сделаю отдельный преобразователь питания для телеметрии, от которой также буду питать OSD. Преобразователь питания представляет собой соединённые вместе импульсный и линейный стабилизаторы. С импульсного напряжения на выходе должно быть больше 6 В для корректной работы, далее последовательно подключается линейный стабилизатор. Плюсом данного решения будет не только возможность подключать аккумуляторы с большим входным напряжением, но и также стабильное выходное напряжение. На рисунке 3.48 показан импульсный стабилизатор.

Рисунок 3.48 – Импульсный стабилизатор

В качестве стабилизатора был выбрал LM7805, а в качестве конденсатора можно выбрать любой конденсатор с напряжением 5 – 10 В и ёмкостью не менее 500 µФ, я выбрал конденсатор напряжением 10 В и ёмкостью 1200 µФ. Вход стабилизатора соединяется с выходом импульсного стабилизатора, а выход стабилизатора «GND» соединяется с минусом импульсного стабилизатора. Конденсатор подключается параллельно после стабилизатора LM7805. Теперь необходимо собрать схему с линейным стабилизатором на основе LM7805. Схема представлена на рисунке 3.49.

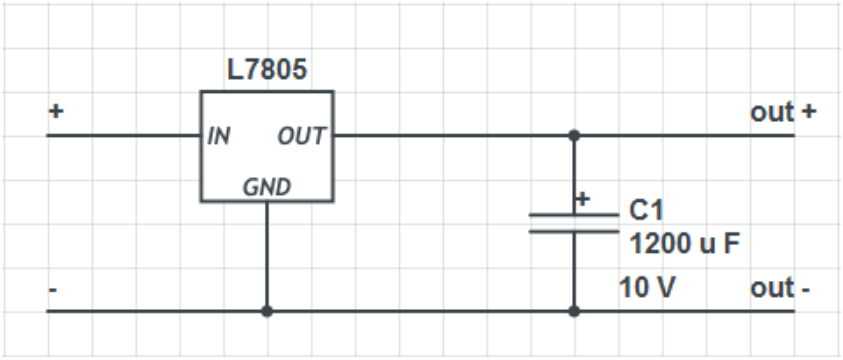

Рисунок 3.49 – Схема линейного стабилизатора

Также на выходе обязательно надо пропустить провода через ферритовое кольцо, чтобы погасить высокочастотные колебания, которые могут привести к неполадке телеметрии и OSD. Собранный стабилизатор показан на рисунке 3.50.

Рисунок 3.50 – Собранный преобразователь питания

Готовый стабилизатор укомплектую в термотрубку, чтобы при случайном контакте с оголённым проводом стабилизатор напряжения или другой элемент не вышли из строя.

3.7 Окончательный вариант сборки

Финальная версия летательного аппарата включает установленный подвес с камерой. OSD с телеметрией соединяются параллельно. Т.к. к ним нет входных сигналов, кроме питания, а контроллер посылает сигналы, поэтому телеметрия с OSD соединены параллельно. OSD вместе с преобразователем питания укомплектованы в кейс, чтобы во время полёта не повредить их. Также были заменены штатные гайки, которые крепят пропеллеры к моторам, на шестигранную, т.к. прошлая гайка ненадёжно держала пропеллер и была вероятность, что она открутится. Также был присоединён GPS–модуль, для геолокации.

Необходимо также использовать собранный стабилизатор напряжения. Соединить стабилизатор по схеме, показанной на схеме 3.51.

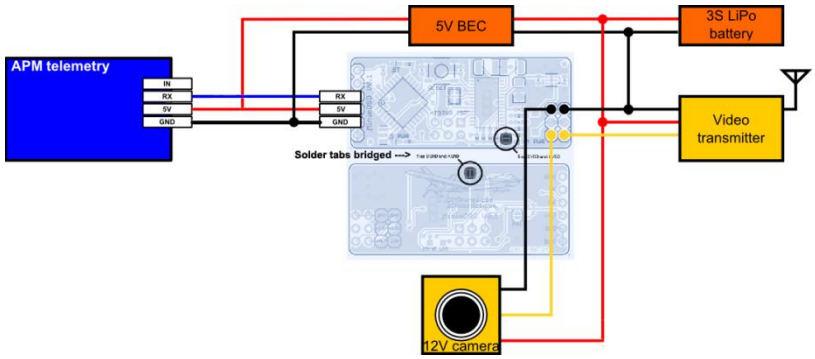

Рисунок 3.51 – Схема соединения преобразователя питания с OSD и телеметрией

Выводы

В ходе выполнения главы составил требования к блоку управления летательного аппарата и летательному аппарату, на основе, которых производился выбор комплектующих, обоснование выбора и расчёт параметров. Была написана прошивка для полётного контроллера на языке C, которая позволила управлять летательным аппаратом и осуществлять его стабилизацию и обработку исключающих ситуаций, таких как потеря сигнала передатчика, малый заряд аккумулятора и т.д. После сборки произвёл тестовые полёты для выявления недостатков конструкции. В ходе анализа тестовых полётов была замечена сильная просадка напряжения на контроллере питания, вследствие чего собрал преобразователь питания для стабильного выходного напряжения. В тестовой версии из составляющих блока управления была установлена телеметрия для удалённого управления квадрокоптером. После доработки были установлены остальные компоненты: передатчик видео, подвес, камера, OSD, GPS–модуль и телеметрия. Таким образом, квадрокоптер откалиброван, на основе проведённого анализа решена была проблема с питание телеметрии и OSD, также заданы параметры failsafe, которые помогают спасти квадрокоптер при происшествии событий, которые угрожают потерять квадрокоптер или приведут к аварии.

4 Безопасность жизнедеятельности

4.1 Анализ условий труда в офисе

Программирование – это основная часть работы, поэтому для данной работы используется компьютер. Здание,, как и рабочее помещение, должны соответствовать СанПиН, ГОСТ и СНиП.

Организация рабочего места определяется ГОСТ 1.2.032–78. Располагается рабочее место так, чтобы согласно санитарным нормам, свет падал сбоку. Оконные проёмы оборудованы для комфортной работы жалюзи. Уборка влажная проводится в понедельник, среда и пятница. Вся мебель соответствует ГОСТ 1.2.032–78.

Также расположен стол шириной 0.77 м и длиной 2.4 м, что соответствует требованиям санитарным нормам. Стол имеет отдельную поверхность снизу для клавиатуры, позволяющая освободить пространство перед монитором для разных вещей или документов, необходимых для работы. Помещение должно отвечать СН 512–78. В одном из бизнес–центров города Алматы располагается помещение, план помещения представлен на рисунке 4.1. В соответствии СН 512–78, офис с компьютерами не должен располагаться в подвале. Офис расположен на 2–ом этаже.

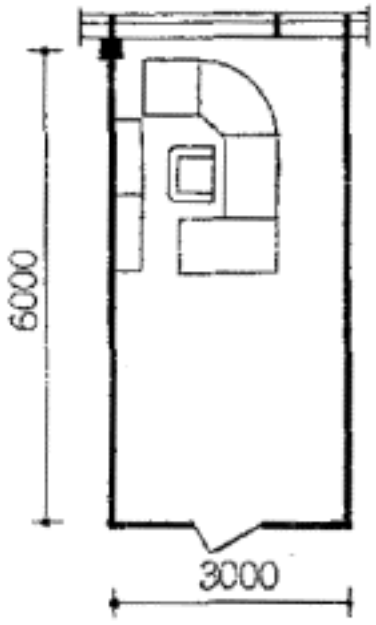

Рисунок 4.1 – Схема размещения офиса

Согласно рисунку 4.1 площадь офиса 18 м^2. В помещении работает один работник. Высота потолка 4 м, что удовлетворяет СН 512–78. Согласно «Санитарно–эпидемиологические требования к условиям работы с источниками физических факторов (компьютеры и видеотерминалы), оказывающих воздействие на человека» площадь необходимая на одного человека должна быть больше 6 м^2, а объём согласно

СНиП РК 3.02–04–2009 больше 20 $м^3$. На сотрудника в помещении выделяется 17 $м^2$ и объём 70 $м^3$, что соответствует санитарным правилам.

Чтобы удостовериться в правильных параметрах, для замера офиса использовался лазерный дальномер Bosch DLE 40. Проверка уровня пола производилась уровнем лазерным BOSCH PLL 5, который позволил удостовериться в ровной поверхностей офиса. В офисе присутствуют шкаф , стул и стол. Из техники – компьютер, паяльная станция, ИБП. В соответствии СН 512–78 помещение оборудовано отоплением, кондиционированием, кабель–каналом, в котором произведена проводка UTP кабеля, присутствуют сетевые розетки. В офисе также присутствует городской телефон.

В офисе имеется автоматическая пожарная сигнализация, также в помещении находятся 3–мя газовыми огнетушителями. В офисе находится аптечка первой помощи. Согласно СН 512–78, в помещениях, оборудованными компьютерами, необходимо более одного огнетушителя на каждые 20 $м^2$ помещения. Офис оборудован двумя датчиками пожарной сигнализации.

Компьютер – основной источник шума. Уровень шума, согласно санитарным нормам, при работе компьютера не должен превышать 50 дБ. Измерение провожу у рабочего места, так как это требуется санитарными нормами. С помощью шумомера Mastech MS6708 производился замер. Шумомер показал значения в 27 дБ, что удовлетворяет требованиям «Санитарно–эпидемиологические требования к условиям работы с источниками физических факторов, оказывающих воздействие на человека». Небольшой такой уровень шума связан с тем, что в компьютере используются вентиляторы системы охлаждения большого диаметра, около 14 дюймов, и вращаются с невысокой скоростью.

Офис попадает под категорию 3а, согласно СНиП РК 4.02–42–2006, где сотрудник трудится умственно в положении сидя без верхней одежды. Лампы светодиодные используются как источник света. В офисе располагаются два светильника, в которых находятся лампы светодиодные. Свет, испускаемый светодиодными лампами является белым, что сказывается хорошо на здоровье глаз, глаза дольше не утомляются.

Согласно СН РК 2.04–02–2011 «Естественное и искусственное освещение» освещенность рабочего стола при работе должна быть в диапазоне 300–500 лк. С помощью люксмера измерил освещенность рабочего места, которая равна 346 лк, что соответствует требованиям СН РК 2.04–02–2011. На рабочем столе также имеется лампа с светодиодным освещением, которая используется при пайки деталей. С включенной лампой освещенность рабочего места равна 423 лк.

Чтобы защититься от травмы током нужно работать согласно техническим правилам эксплуатации электрооборудования. Необходимое условие – придерживаться трудовой дисциплины, рабочее место следует организовывать правильно, необходимо проходить постоянно медецинское

обследование. Для защиты от соприкосновения к токоведущим элементам оборудования и приборов используется изоляция. Защитное заземление производится для защиты от нанесения травм током при контакте с металлической частью оборудования, оказавшейся самопроизвольно под напряжением.

В офисе есть предохранители для защиты сети от перегрузки. Также полы являются не токопроводящими. Шины заземления располагаются в легкодоступных местах. Причиной пожара может быть короткое замыкание, также пробитый кабель электропитания. В офисе в случае несчастного случая расположена аптечка для оказания помощи.

В офисе поддерживается микроклимат, т.к. влияет на эксплуатационный срок техники и здоровье работника. Температура поддерживается не более 28 °С в тёплое время года, т.к. при большой разнице температур внутри офиса и на улице можно простудиться или получить переохлаждение, и влажность не более 60%. Во время рабочего дня, Каждые 2 часа, и до начала рабочего дня проветривается помещение. Кондиционер используется для поддержания необходимой температуры в тёплое время. А холодный период работает отопление, и температура держится в диапазоне 21–22 °С и влажность не более 75%. В холодной время также осуществляется проветривание помещения.

Поскольку сотрудник, работающий на компьютере, выполняет различные задачи, которые требуют повышенной концентрации внимания, то после каждого часа необходим перерыв в 10 минут. Запрещена работа свыше 6 часов на компьютере согласно трудовому кодексу РК.

Применяется чередование работы, т.е. изменяется содержание работы. Невыполнение условий отдыха и труда приводит к переутомлению сотрудника, что может стать причиной допущения ошибок при решении задач разработки программного и аппаратного комплекса. Рабочий день сотрудника длится 9 часов с учётом перерыва на обед, в неделю 5 дней, что составляет 40 часов в неделю в соответствии с трудовым законодательством Республики Казахстан.

4.2 Расчёт заземлителя

Заземление производится для избегания поражения током при соприкосновении с телом частей прибора, которые не должны быть под высоким напряжением. Контур заземлителя состоит из горизонтальных и вертикальных заземлителей. Вертикальные заземлители проводятся в почву на некую глубину. С вертикальными соединяются горизонтальные заземлители. Провод заземления соединяет контур непосредственно из щитка. Расчётное количество и параметры заземлителей напрямую зависят от сопротивления почвы. Заземлители устанавливаются по контуру или в ряд на глубину, когда расстояние между заземлителем и верхней границей земли должно составлять 0.4–0.9 м. Протяжённость между вертикальными заземлителями составляет более 2.5–3 м. Заземлители между собой

соединяются полосками из стали толщиной более 5 мм или проводом из стали радиусом более 4 мм.

Из–за меньшего сопротивления заземления ток устремляется в землю, таким образом, защищая от повреждения им. Защитное заземление производится для защиты от нанесения травм током при контакте с металлической частью оборудования, оказавшейся самопроизвольно под напряжением. Для этого выполняется заземление, сопротивление которого меньше или равно 4 Ом.

Вертикальные электроды выполнено электродов из труб диаметром 0.15 м и длиной 9 м. Горизонтальное соединительная полоса выполнена из стали 0.08 м, а глубина места заложения – 0.8 м. Грунт в месте установки защитного заземления – суглинок, удельное сопротивление которого равна 100 Ом · м. Здание расположено в Алматы, то возьму коэффициент сезонности равеным для заземлителей вертикальных – 1.7, а для горизонтальных – 4. Трубы расположены по контуру.

Сначала необходимо рассчитать удельное сопротивление грунта:

$$\rho_{гр} = \rho \cdot к_{в}, \tag{4.1}$$

где ρ – удельное сопротивление грунта, Ом · м;

$к_{в}$ – коэффициент сезонности.

Рассчитаю для вертикальных заземлителей расчётное удельное сопротивление по формуле (4.1):

$$\rho_{в} = 1.7 \cdot 100 = 170 \text{ Ом} \cdot \text{м}$$

Рассчитаю для горизонтальных заземлителей расчётное удельное сопротивление по формуле (4.1):

$$\rho_{гр} = 4 \cdot 100 = 400 \text{ Ом} \cdot \text{м}$$

Сопротивление одиночного заземлителя определяется по формуле:

$$R = 0.366 \cdot \frac{\rho}{L} \cdot \left(\lg\frac{2 \cdot L}{d} + 0.5 \cdot \lg\frac{4 \cdot t + L}{4 \cdot t - L}\right), \tag{4.2}$$

где R – сопротивление одиночного заземлителя, Ом;

ρ – удельное сопротивление грунта, Ом · м;

L – длина заземлителя, м;

t – глубина заложения заземлителя, м;

d – диаметр заземлителя, м.

При этом глубина заложения вертикального заземления рассчитывается по формуле:

$$t = \frac{L}{2} + h, \qquad (4.3)$$

Подставлю значения в формулу (4.3):

$$t = \frac{9}{2} + 0.5 = 5 \text{ м}$$

Теперь найду сопротивление одиночного вертикального заземлителя по формуле (4.2):

$$R = \frac{0.366 \cdot 170}{9} \cdot \left(\lg \frac{2 \cdot 9}{0.15} + 0.5 \cdot \lg \frac{4 \cdot 5 + 9}{4 \cdot 5 - 9} \right) = 6.91 \cdot (2.08 + 0.42) =$$
$$= 17.28 \text{ Ом} \cdot \text{м}$$

Необходимое количество вертикальных электродов рассчитывается по формуле:

$$n = \frac{R}{R_y \cdot \mu_0}, \qquad (4.4)$$

где μ_0 – коэффициент использования заземлителя.
Найду необходимое количество труб при $\mu_0 = 1$ по формуле (4.4):

$$n = \frac{17.28}{4} = 4.32$$

Округлю число n = 4. Принимая отношение a/l = 2 и контурное расположение заземлителей для количества труб равным 4, с учётом интерполяции, по таблице 4.1 получу $\mu_0 = 0.8$.

Таблица 4.1 – Коэффициент использования μ вертикальных электродов в контуре

Отношение a/l	Число электродов	μ
2	4	0.76–0.80
	6	0.71–0.75
	10	0.66–0.71
	20	0.61–0.66

Тогда подставив значения в формулу (4.4):

$$n = \frac{17.28}{0.8 \cdot 4} = 5.4$$

Округляю до целого числа, т.е. n=6. Для заземлителей, расположенных в контуре, рассчитаю длину полосы:

$$l = 1.05 \cdot n \cdot a, \tag{4.5}$$

где n – количество труб,

a – расстояние между вертикальными заземлителями.

Подставлю значения в формулу (4.5) и получу:

$$l = 1.05 \cdot 6 \cdot 18 = 113.4 \text{ м}$$

Вычислю сопротивление растекания горизонтальной соединительной полосы, расположенной в земле:

$$R_n = 0.366 \cdot \frac{\rho}{l} \cdot \lg\frac{2 \cdot l^2}{b \cdot h} \tag{4.6}$$

Подставлю значения в формулу (4.6):

$$R_n = 0.366 \cdot \frac{400}{113.4} \cdot \lg\frac{2 \cdot 113.4^2}{0.08 \cdot 0.5} = 0.77 \cdot 5.81 = 7.5 \text{ Ом}$$

При n = 6, a/l = 2 и расположению труб в групповом заземлителе по контуру коэффициент полосы μ = 0.46 согласно таблице 4.2

Таблица 4.2 – Коэффициент использования μ горизонтального полосового электрода при размещении вертикальных электродов по контуру

Отношение a/l	Число электродов	μ
2	4	0.55
	5	0.48
	8	0.43
	10	0.40

Рассчитаю сопротивление растеканию заземлителей по формуле:

$$R_{гр} = \frac{R \cdot R_n}{R \cdot \mu + R_n \cdot \mu_0 \cdot n} \tag{4.7}$$

Подставлю значения в формулу (4.7):

$$R_{гр} = \frac{17.28 \cdot 7.5}{17.28 \cdot 0.46 + 7.5 \cdot 0.8 \cdot 6} = \frac{129.6}{33.95} = 2.95 \text{ Ом}$$

Согласно расчёту, сопротивление заземлителя равно 2.95 Ом, что меньше 4 Ом, и является допустимым значением. Таким образом, рассчитанный заземлитель удовлетворяет необходимым параметрам для использования. Схема заземлителя показана на рисунке 4.2

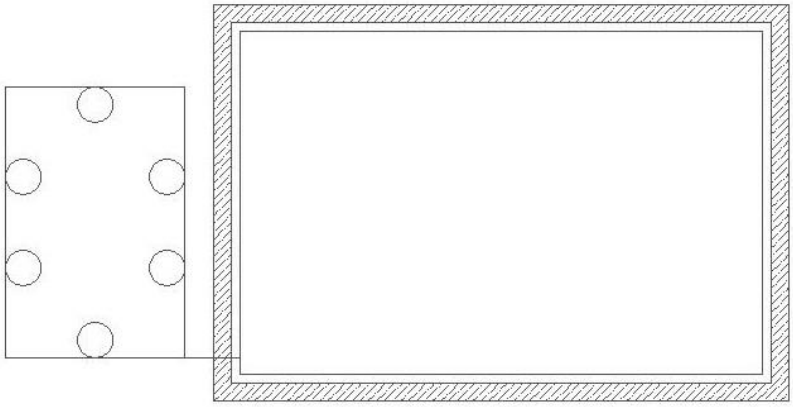

Рисунок 4.2 – Схема заземлителя

4.3 Расчёт молниезащиты

Молниезащита необходима для безопасности здания, техники и людей. Попадание молний в здание приводит к:

- его повреждению;
- выходу из строя электрических частей;
- травмированию и летальному исходу живых существ, находящемся в нём..

Молниезащита бывает внешней и внутренней. Внешняя молниезащита – система, обеспечивающая перехват молний и отвод их в землю, таким образом, защищая от повреждений и пожаров. При ударе молнии в здание молниезащита принимает на себя удар и производит отвод мощности в заземление, в земле энергия рассеивается. Ток должен пройти, не повредив здание и не нанеся ущерб людям внутри.

Молниеприемник принимает разряд молнии и отводит его в землю. Молниеприёмник может быть:

- металлическим штырем;
- сетью из проводящего материала;
- металлическим тросом, натянутым над защищаемым домом.

Молниезащиту необходимо выполнять согласно стандарту СН РК 2.04–29–2005 «Инструкция по устройству молниезащиты зданий и сооружений», в котором описываются нормы по оборудованию зданий и сооружений молниезащитой.

Я провожу расчёт для здания, в котором находится офис, в котором производятся работы. Здание имеет длину 40 м, высоту 20 м и ширину 25 м.

Т.к. в здании находится компьютерная техника, то здание относится к 3–ей категории объекта по молниезащите.

Для города Алматы интенсивность грозовой деятельности n = 20…40 ч/год, а среднее число ударов молнии в год на 1 км^2 равна 4.

Молниеотвод устанавливается в центре крыши здания. Схема зоны защиты здания одиночным стержневым молнеотводом приведена на рисунке 4.3.

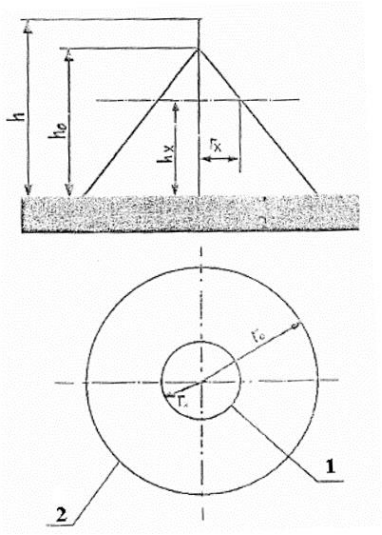

Рисунок 4.3 – Зона защитного стержневого молниеотвода:
1 – граница зоны защиты на уровне h_x; 2 – граница зоны защиты на уровне земли

Определю вероятность количества ударов молнии в год в здание:

$$N = (A + 6 \cdot h_x) \cdot (B + 6 \cdot h_x) \cdot n \cdot 10^{-6}\ , \qquad (4.8)$$

где A – длина здания, м;

B – ширина здания, м;

h – высота здания, м;

n – среднее число ударов молнии в год на 1 км^2, где n = 4.

По схеме защиты сечение зоны защиты R_x представляет половину диагонали защищаемого здания:

$$R_x = \sqrt{A^2 + B^2} \qquad (4.9)$$

Подставлю значения в формулу (4.9):

$$R_x = \sqrt{(40)^2 + (20)^2} = 44.72 \text{ м}$$

Принимаем значение R_x равным 45, округляя в большую сторону. Следовательно, число ударов вычислю, подставив значения в формулу (4.8):

$$N = (40 + 6 \cdot 20) \cdot (25 + 6 \cdot 20) \cdot 4 \cdot 10^{-6} = 0.09$$

Рассчитаю высоту громоотвода:

$$h = \frac{R_x + 1.63 \cdot h_x}{1.5} \quad (4.10)$$

Подставлю значения в формулу (4.10):

$$h = \frac{45 + 1.63 \cdot 20}{1.5} = 51.73 \text{ м}$$

Найду область зоны защиты:

$$R_0 = 1.5 \cdot h \quad (4.11)$$

Подставлю значения в формулу (4.11), чтобы определить радиус зоны защиты:

$$R_0 = 1.5 \cdot 45 = 67.5 \text{ м}$$

Расчёты показывают, что в центре крыши здания необходимо установить стержневой молниеотвод на опоре высотой примерно 32 м, радиус горизонтального сечения зоны защиты здания составит 44.72 м, а на уровне земли защита формирует круг радиусом 67,5 м, общая высота молниеотвода – 52 м.

5 Технико–экономическое обоснование

5.1 Цели и задачи проекта

Целью является доказать целесообразность разработки разработка блока управления летательного аппарата, т.е. дрона, м экономической точки зрения. Дрон можно использовать разведывательных операций, быстрой доставки по воздуху товаров, ведение боевых действий без необходимости подключения людей, охраны объектов, ведения поисковых работ, видео съёмки различных соревнований с воздуха и других задач. Это уменьшает затраты времени, денег и уменьшает влияние человека на проводимые работы. Разработка хоть и является затратной, но быстро окупается за счёт большого спроса на летательные аппараты.

Задачами экономической части дипломной работы являются определить трудоемкость разработки, социальные отчисления, расходы на техническое обеспечение, разработки, затраты на оплату труда работников, и затраты на потребляемую электроэнергию, а также цену программного обеспечения с учетом НДС.

5.2 Расчет трудоемкости для разработки

Разработка блока управления летательного аппарата состоит из шести этапов:

- создание технического задания;
- разработка алгоритма;
- написание программы;
- сборка летательного аппарата;
- написание документации;
- отладка и тестирование.

На основании шести этапов разработки необходимо рассчитать трудоёмкость работ на каждом этапе. Совместное выполнение двух и более этапов не возможно, т.к. результат предыдущего этап. Для начала необходимо рассчитать часовую ставку работника по формуле:

$$ЧС = \frac{ЗП}{ФРВ}, \quad (5.1)$$

где ЗП – заработная плата работника, тг;

ФРВ – месячный фонд рабочего времени работника, тг;

Затем вычислю общую сумму затрат на оплату труда на время проекта:

$$З_{тр} = \sum_{i=1}^{m} ЧС_i \cdot Т_i, \quad (5.2)$$

где ЧС – часовая ставка работника, тг/час;

Т – трудоёмкость выполнения НИР, час.

Зарплата инженера–программиста составляет 120000 тг, а руководителя – 150000 тг.

Месячная норма выработки составляет 160 часов. Рассчитаю часовую ставку, подставив значения из таблицы 5.1 в формулу (5.1):

$$ЧС_{пр} = \frac{120000}{160} = 750 \text{ тг}$$

Таким образом, для инженера–программиста равной 750 тг, а для руководителя – 938 тг.

По расчётам общее наработанное время составляет для инженера–программиста 111 часов, а для руководителя – 34 часа. Подставив значения в формулу (5.2) для расчёта заработной платы за выполненный проект:

$$З_{тр} = 111 \cdot 750 = 83250 \text{ тг}$$

Таким образом, зарплата инженера–программиста без вычета налога и пенсионных выплат составляет 83250 тг, руководителя – 31892 тг. А общие затраты – 115142 тг. Затраты на заработную плату представлены в таблице 5.1

Таблица 5.1– Заработная плата

Наименование и содержание работ	Исполнитель	Трудоемкость, норма–час	Заработная плата за час работы	Сумма заработной платы
Создание ТЗ	Руководитель	34	938	31892
Разработка алгоритма	Инженер–программист	21	750	15750
Написание программы	Инженер–программист	50	750	37500
Сборка летательного аппарата	Инженер–программист	5	750	3750
Написание документации	Инженер–программист	15	750	11250
Отладка и тестирование	Инженер–программист	20	750	15000
Всего:		145		115142

Необходимо вычислить сумму социальных отчислений. От заработной платы отнимаются 10% пенсионных отчислений, а также социальный налог 11%. Для начала необходимо вычислить пенсионные выплаты по формуле:

$$О_{п} = З_{тр} \cdot 10\%, \quad (5.3)$$

где $О_п$ – размер пенсионных отчислений, тг;
$З_{тр}$ – заработная плата работника, тг.
Подставлю значения в формулу (5.3):

$$О_п = 83250 \cdot 10\% = 8325 \text{ тг}$$

Пенсионные отчисления инженера–программиста составляют 8325 тг, для руководителя – 3189.2 тг. После чего определю размер социального налога по формуле:

$$Н_{сц} = (З_{тр} – В_п) \cdot 11\%, \quad (5.4)$$

где $Н_{сц}$ – размер социального налога, тг;
$В_п$ – размер пенсионных отчислений, тг;
$З_{тр}$ – заработная плата работника, тг.
Подставлю в формулу (5.4) и получу:

$$Н_{сц} = (83250 – 8325) \cdot 11\% = 8241.75 \text{ тг}$$

Для инженера–программиста размер социального налога составляет 8241.75 тг, а для руководителя – 3157.3 тг. Таким образом, отчисления на социальные нужды рассчитываются по формуле:

$$СН = \sum_{i=1}^{m} (Н_{сці} + О_{пі}), \quad (5.5)$$

где $Н_{сці}$ – размер социального налога, тг;
СН – отчисления на социальные нужды, тг;
$О_{пі}$ – размер пенсионных отчислений, тг.
Подставлю значения в формулу (5.5):

$$СН = 8241.75 + 3157.3 + 8325 + 3189.2 = 22913.25 \text{ тг}$$

Отчисления на социальные нужды составляют 22913.25 тг. Найду заработную плату после вычета пенсионных отчисления и социального налога по формуле:

$$З = З_{тр} – Н_{сц} – О_п, \quad (5.6)$$

где $Н_{сц}$ – размер социального налога, тг;
$З_{тр}$ – заработная плата работника, тг;
$О_п$ – размер пенсионных отчислений, тг.
Подставлю значения в формулу (5.6) и получу:

$$З = 83250 – 8325 – 8241.75 = 66683.25 \text{ тг}$$

Таким образом, заработная плата после вычета пенсионных отчисления и социального налога инженера–программиста составляет 66683.25 тг, а руководителя – 25545.5 тг.

Таблица 5.2 – Длительность цикла в днях по каждому виду работы

Наименование и содержание работ	Длительность цикла в днях
Создание ТЗ	5
Разработка алгоритма	3
Написание программы	7
Написание документации	2
Сборка летательного аппарата	1
Отладка и тестирование	3
Всего:	20

5.3 Расчет затрат на выполнение НИР

Для выполнения проекта подсчитаю следующие виды затрат: на материальные ресурсы, социальные отчисления, на оплату труда, амортизацию основных фондов. В материальные затраты делятся на вспомогательные (канцелярские товары, картридж) и основные, энергия, необходимая при разработке проекта. Рассчитаю стоимость материальных ресурсов по формуле:

$$З_м = \sum_{i=1}^{m} К_{Mi} \cdot Ц_{Mi}, \quad (5.7)$$

где $К_{Mi}$ – количество материалов, шт;

$Ц_{Mi}$ – цена за единицу материала, тг.

Рассчитаю вспомогательные материальные ресурсы, подставив данные в формулу (5.7):

$$З_м = 1 \cdot 500 + 1 \cdot 5000 + 50 \cdot 6 + 1 \cdot 1600 + 1 \cdot 1000 + 10 \cdot 200 + 1 \cdot 450 = \\ = 10850 \text{ тг}$$

Объём затрат на материальные вспомогательные ресурсы составил 10850 тг. В таблице 5.3 показаны затраты на вспомогательные материальные ресурсы.

Таблица 5.3 – Затраты на вспомогательные материальные ресурсы

Наименование материала	Марка	Единицы измерения	Количество	Цена за единицу, тг	Сумма, тг
Бумага	SvetoCopy	уп	1	500	500
Картридж	PG 513	шт	1	7000	10050
Ручка	ErichCrause	шт	6	50	300
Всего:					10850

К основным единичным материальным затратам отнесу паяльную станцию, ноутбук, монитор и источник бесперебойного питания, которые представлены в таблице 5.4. В них не включаю программное обеспечение и операционную систему, т.к. в качестве операционной системы используется бесплатная Debian 8. В качестве среды разработки использую Arduino IDE, которая также является бесплатной.

Таблица 5.4 – Основные материальные затраты

Наименование материала	Количество, шт	Цена за единицу, тг	Сумма, тг
Ноутбук Acer	1	70000	70000
Паяльная станция	1	10500	10500
Монитор Asus 24 дюйма	1	24000	24000
Источник бесперебойного питания	1	8000	8000
Струйный принтер HP	1	7500	7500
Arduino IDE	1	Бесплатно	0
ОС Debian 8	1	Бесплатно	0
Всего:			120000

В таблице 5.5 представлены затраты на комплектующие для сборки одного дрона, затраты которого составляют 102816 тг. В стоимость комплектующих уже включена доставка.

Таблица 5.5 – Затраты на комплектующие для сборки

Наименование изделий	Количество единиц	Цена за единицу в тенге	Общая сумма в тенге
Рама квадрокоптера	1	5481	5481
Моторы Emax mt2213	4	2646	10584
Пропеллеры	4	189	756
Контроллер Arducopter	1	6426	6426
Аккумулятор Turnigy 3S 5000mAh	3	7182	21546
Приёмник и передатчик видео сигнала	1	10584	10584
Пульт и приёмник сигнала	1	13608	13608
Регуляторы скорости	4	13608	13608
OSD	1	378	378
Подвес для камеры	1	8505	8505
Камера	1	8505	8505
GPS–модуль	1	2835	2835
Всего:			102816

После расчёта затрат на вспомогательные материалы и комплектующие, рассчитаю затраты на электроэнергию. Сумма затрат ($З_Э$) рассчитывается по формуле:

$$З_э = \sum_{i=1}^{m} K_i \cdot M_i \cdot T_i \cdot Ц, \quad (5.8)$$

где M_i – паспортная мощность i–го электрооборудования, кВт;

K_i – коэффициент использования мощности i–го электрооборудования (принимается K_i=0.7–0.9);

T_i – время работы i–го оборудования за весь период разработки ПП, ч;

Ц – цена электроэнергии, тг/кВт · ч;

i – вид электрооборудования;

n – количество электрооборудования.

Рассчитаю общую стоимость затрат на электроэнергию согласно таблице 5.6, подставив значения в формулу (5.8):

$$З_э = 0.7 \cdot 0.2 \cdot 5 \cdot 21.64 = 15.148 \text{ тг}$$

Таблица 5.6 – Затраты на электроэнергию

Наименование оборудования	Паспортная мощность, кВт	Коэффициент использования мощности	Время работы оборудования для разработки ПП, ч	Цена электроэнергии, Описание: тг/кВт	Сумма, тг
Паяльная станция	0,2	0,7	5	21,64	15.148
Принтер	0,06	0,7	2	21,64	1.82
Монитор	0,036	0,9	115	21,64	80.63
Ноутбук	0,35	0,8	115	21,64	696.808
Освещение	0,15	0,9	120	21,64	408.996
Всего:					1203.4

Затраты на электроэнергию составили 1203.4 тг. В амортизацию основных фондов включается сумма амортизационных отчислений от стоимости оборудования, используемых при разработке ПП. Общая сумма амортизационных отчислений определю по формуле:

$$З_{am} = \sum_{i=1}^{n} \frac{Ф_i \cdot Н_{Ai} \cdot Т_{нир}}{100 \cdot Т_{эфi}}, \quad (5.9)$$

где $Ф_i$ – стоимость i–го ОФ, тг;

$Н_{Ai}$ – годовая норма амортизации i–го ОФ, %;

$Т_{НИРi}$ – время работы i–го ОФ за весь период разработки ПП, ч;

$Т_{Эфi}$ – эффективный фонд времени работы i–го ОФ за год, ч/год;

i – вид ОФ;

n – количество ОФ.

Годовые нормы амортизации ОФ принимаются по налоговому кодексу РК или определяются, исходя из возможного срока полезного использования ОФ:

$$Н_{Ai} = \frac{100\%}{Т_{Ni}} \quad (5.10)$$

где $Т_{Ni}$ – возможный срок использования i–го ОФ, год;

Например, для монитора средний срок службы 5 лет, следовательно, подставив в формулу (5.10) получу:

$$Н_а = \frac{100\%}{5} = 20\%$$

Подставлю значения в формулу (5.9), чтобы вычислить амортизационные отчисления:

$$З_{ам} = \frac{20 \cdot 70000 \cdot 115}{100 \cdot 1848} = 871.21 \text{ тг}$$

Амортизационные отчисления ноутбука за время разработки составляют 871.21 тг. Амортизационные отчисления приведены в таблице 5.7.

Таблица 5.7 – Амортизация основных фондов (ОФ)

Наименование оборудования и ПО	Стоимость оборудования, тг	Годовая норма амортизации, %	Эффективный фонд времени работы оборудования, ч/год	Время работы оборудования для разработки ПП, ч	Сумма, тг
Ноутбук Acer	70000	20	1848	115	871.21
Паяльная станция	10500	10	1848	5	2.841
Монитор Asus 24 дюйма	24000	20	1848	115	298.701
Источник бесперебойного питания	8000	15	1848	115	74.675
Струйный принтер HP	7500	10	1848	2	0.812
Итого:					1248.24

Также необходимо подсчитать расходы на интернет. Месячная плата составляет 3830 тг.

Также рассчитаю стоимость аренды офисного помещения. Площадь офисного помещения составляет 20 м^2, а стоимость квадратного метра составляет 1490 тг/м^2 за месяц аренды. Полная аренда офисного помещения составляет 29800 тг в месяц.

После чего составлю смету затрат, которая представлена в таблице 5.8.

Таблица 5.8 – Смета затрат на выполнение НИР

Статьи затрат	Сумма, тг
1. Материальные затраты, в том числе:	
– материалы	113666
– электроэнергия	1203.4
2. Затраты на оплату труда.	115142
3. Отчисления на социальные нужды.	22913.25
4. Амортизация основных фондов.	1248.24
5. Прочие затраты.	
– аренда	29800
– плата за интернет	3830
– прочее	20725
ИТОГО по смете	284614.64

5.4 Расчёт рентабельности НИР

Величина цены НИР устанавливается с учётом эффективности, качества и сроков её выполнения на уровне, отвечающим экономическим интересам потребителя и исполнителя.

Договорная цена рассчитывается по формуле.

$$Ц_д = З \cdot (1 + \frac{Р}{100}), \quad (5.11)$$

где Р – рентабельность продукта, принимающаяся в размере 22%, %;

З – затраты на производство, тг;

$Ц_д$ – договорная цена продукта, тг;

Подставлю значения в формулу (5.11) и получу:

$$Ц = 284614.64 \cdot \left(1 + \frac{22}{100}\right) = 347229.86 \text{ тг}$$

Договорная цена составляет 347229.86 тг. Далее необходимо определить цену реализации с учётом налога на добавочную стоимость, ставка НДС устанавливается законодательно Налоговым Кодексом РК и равно 12%.

Таким образом, цена реализации с учётом НДС рассчитывается по формуле:

$$Ц_р = Ц_д + Ц_д \cdot НДС, \quad (5.12)$$

где $Ц_р$ – цена реализации, тг;

$Ц_д$ – договорная цена, тг;

НДС – налог на добавочную стоимость, %.

Подставлю значения в формулу (5.12):

$$Ц_р = 347229.86 + 347229.86 \cdot 12\% = 388897.44 \text{ тг}$$

Таким образом, цена реализации составляет 388897.44 тг.

5.5 Оценка научно–технической результативности в социальной эффективности НИР

Результат работы – оценить затраты на создание дрона и оценить рентабельность проекта. Дроны могут найти применение в любой сфере деятельности: начиная доставкой товаров на дом и заканчивая военными целями. Таким образом, дроны в скором времени будут использоваться повсеместно, хотя нет ещё существующих законов, которые контролируют их, но в скором времени будут. Аналогов проекта в продаже нет, только есть с меньшим функционалом, но с большей стоимостью. Проект имеет плюсы и минусы. Один из плюсов – цена и большие возможности эксплуатации. Минус проекта – это стоимость комплектующих. Стоимость проекта можно уменьшить, если разрабатывать моторы, раму, полётный контроллер собственными силами. Конечно, начальные затраты на проектирование рамы будут стоить дорого, но в дальнейшем это окупится тем, что Вы сами выбираете материал рамы и вместо пластика можно использовать углеродное волокно, которое прочнее и легче пластика, что скажется на характеристиках дрона, увеличит полётный вес и уменьшит ремонтные затраты. Полётный контроллер – это плата с распаянными микросхемами. Затраты на проектирование контроллера равно нулю, т.к. чертежи платы и спецификация микросхем находится в свободном доступе, т.к. проект контроллера открытый и можно сделать аналог, что удешевит впоследствии стоимость и можно заменить многие датчики, если вдруг хочется улучшить сам контроллер. Но при этом главное выдержать модульность, т.к. нужно учитывать ремонтопригодность. Если всю электронику распаять на одной плате, то это может уменьшить стоимость готового изделия, но тогда ремонт будет стоить дорого, т.к. при сгорании одного из элементов необходимо или менять всю плату, что дорого, или делать ремонт, который сложный. Поэтому все компоненты системы являются модульными, т.е. их легко заменить.

Социальный эффект по отношения к дронам может быть различным, т.к. они полезны и это оказывает положительный эффект на общество, но из–за того, что это новое и люди боятся перемен, что может отрицательно сказаться, т.к. дроны не контролируются законом, и социум будет опасаться их. Из–за того, что дроны будут выполнять часть работы, где нужны работники без высшего образования, то это должно послужить стимулом для повышения своих навыков, окончив высшее учебное заведение.

Заключение

В первом разделе дипломного проекта я сделал обзор предметной области, для чего используются летательные аппараты, какие проводятся выставки на территории СНГ и всего мира, что это оказывает влияние не только на развитие технического прогресса, но и позволяют использовать летательные аппараты в учебных целях, начиная от простых модульных аппаратов для школьников и заканчивая отдельным курсом написания программ для летательных аппаратов. Кроме того, это всё закладывает основы знаний из области физики, прикладной механики, программирования. Это позволяет внести знания в учебный процесс, разбавляя серые будни школьников и студентов, позволяя заинтересовать технической наукой с раннего возраста. Рассмотрел классы летательных аппаратов и способы управления летательных аппаратов, начиная с полного управления летательным аппаратом оператором и заканчивая нейронной сетью.

Во второй главе определил, из каких основных компонентов состоит летательный аппарат, дал информацию о полётных контроллерах, разобрал принципы работы бесколлекторного двигателя, регулятора оборотов двигателя, как GPS определяет координаты и дал информацию характеристиках LiPo–аккумуляторов, правила заряда и хранения.

В третьей главе составил требования к блоку управления летательного аппарата и летательному аппарату, но основе которых производился выбор комплектующих и расчёт параметров (электрических параметров летательного аппарата, дальности действия видео передатчика, вес летательного аппарата, максимальная тяга моторов, максимальный ток регуляторов оборотов моторов), произвёл описание GPS–модуля и этапы прошивки, обосновал выбор полётного контроллера. Была написана прошивка для полётного контроллера на языке C, которая позволила управлять летательным аппаратом и осуществлять его стабилизацию и обработку исключающих ситуаций, таких как потеря сигнала передатчика, малый заряд аккумулятора и т.д. После сборки произвёл тестовые полёты, на основе которых провёл анализ полученных данных контроллера, выявив слабые места конструкции. В ходе анализа тестовых полётов было замечено сильное падение напряжения на полётном контроллере, вследствие чего собрал преобразователь питания для стабильного выходного напряжения. После доработки была установлены остальные компоненты: передатчик видео, подвес, камера, OSD, GPS–модуль и модуль телеметрии. Таким образом, квадрокоптер откалиброван, на основе проведённого анализа решена была проблема с питанием телеметрии и OSD, также заданы параметры failsafe, которые помогают спасти квадрокоптер при происшествии событий, которые угрожают потерять квадрокоптер или приведут к аварии. Эти все изменения вошли в финальную версию.

В четвертой главе проанализировал условия труда в арендованном офисе на основе ГОСТ–ов, СанПиН–ов и СНиП–ов. Произвёл расчёт заземлителя и молниеотвода.

В пятой главе рассчитал затраты разработки блока управления летательного аппарата. Стоимость затрат составляет 284614.64 тг, а стоимость реализации 388897.44 тг. Произвёл социальный анализ, как летательные аппараты повлияют на общественную жизнь граждан.

Результаты дипломного проекта показывают, что поставленные задачи были выполнены успешно и были достигнуты поставленные цели.

Список используемой литературы

1) Зинченко О.Н. Беспилотный летательный аппарат: применение в целях аэрофотосъемки для картографирования. – М.: Ракурс, 2011.
2) Г. Шилдт. Самоучитель по C. – 2013.
3) Gary J. Bronson. C for Engineers and Scientists. – 2015.
4) Michael Margolis. Arduino Cookbook. – 2014.
5) John Boxall. Arduino Workshop: A Hands–On Introduction with 65 Projects – 2013.
6) Josh Adams. Arduino Robotics – 2011.
7) Jeremy Blum. Exploring Arduino: Tools and Techniques for Engineering Wizardry – 2012.
8) Ulli Sommer. Arduino: Mikrocontroller–Programmiering mit Arduino/Freeduino – 2010.
9) Угрюмов Е. Цифровая схемотехника. – 2000.
10) Бессонов Л. А. Теоретические основы электротехники. Электрические цепи. – 1996.
11) Алешкевич В.А., Деденко Л.Г., Караваев В.А. Колебания и волны. Лекции. М.: Физфак МГУ, 2001.
12) Калитеевский Н.И. Волновая оптика. (2–е изд.) М.: Высш. школа, 1978.
13) Никольский В.В. Электродинамика и распространение радиоволн. М.: Наука, 1973.
14) Фок В.А. Проблемы дифракции и распространения электромагнитных волн. М.: Сов. радио, 1970.
15) Горелик Г.С. Колебания и волны. Введение в акустику, радиофизику и оптику (2–е издание). М.: Физматлит, 1959.
16) Дьяков В.И. Типовые расчёты по электрооборудованию. М.: Высшая школа. 1990.
17) ГОСТ 1.2.032–78 «ССБТ. Рабочее место при выполнении работ сидя. Общие эргономические требования».
18) СН 512–78. Технические требования к зданиям и помещениям для установки средств вычислительной техники.
19) СНиП_РК_2.02–05–2009. Пожарная безопасность зданий и сооружений.
20) СНиП_РК_3.02–04–2009. Административные и бытовые здания.
21) СНиП_РК_4.02–42–2006. Отопление вентиляция и кондиционирование.
22) Санитарные нормы «Санитарно–эпидемиологические требования к условиям работы с источниками физических факторов (компьютеры и видеотерминалы), оказывающих воздействие на человека».
23) СН РК 2.04–02–2011 «Естественное и искусственное освещение».
24) Дюсебаев М.К. Безопасность жизнедеятельности. Методические указания к выполнению раздела в дипломных проектах. – Алматы, 2003.

25) Базылов К. Б., Алибаева С.А., Бабич А.А. Выпускная работа бакалавров. Экономический раздел. Методичекие указания для студентов всех форм обучения. АИЭС, 2008.

26) Методические указания к выполнению экономической части дипломных работ для студентов специальности 5В070400 – Вычислительная техника и программное обеспечение / Еркешева З.Д, Боканова Г.Ш. – Алматы: АУЭС, 2013 – 40 с.

Список используемых определений

Квадрокоптер – летательный аппарат, оснащенный четырьмя двигателями.

Arducopter – полётный контроллер с открытым исходным кодом.

Arduino – аппаратно-программное средство для построения систем автоматики и робототехники.

GPS – спутниковая система навигации.

Спуфинг – метод подмены данных.

Нейронная сеть – математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей – сетей нервных клеток живого организма.

Барометр – прибор для измерения атмосферного давления.

Гироскоп – устройство, способное реагировать на изменение углов ориентации тела, на котором оно установлено, относительно инерциальной системы отсчета.

Arm – микропроцессорная архитектура с сокращённым набором команд (RISC).

Ротор – подвижная часть мотора.

Статор – неподвижная часть мотора.

Альманах – совокупность данных об основных параметрах орбит спутников в навигационной системе.

Эфемерида – таблица координат спутников.

Loiter – режим удержания точки (по координате и высоте).

RTL – возврат домой, в точку взлета.

Failsafe – режим спасения, который отправит квадрокоптер лететь домой.

Geofence – режим, при включении которого можно ограничить радиус и высоту полета.

Arduino IDE – среда разработки под микроконтроллеры Arduino.

UART – узел вычислительных устройств, предназначенный для организации связи с другими цифровыми устройствами.

COM-порт – последовательный порт, через который данные передаются последовательно.

Телеметрия – получение информации о значениях измеряемых параметров (напряжения, тока, давления, температуры и т. п.) контролируемых и управляемых объектов методами.

OSD – устройство, позволяющее добавлять в поток видео заданные данные, которые будут отображаться на экране.

Импульсный стабилизатор - это стабилизатор напряжения, в котором регулирующий элемент работает в ключевом режиме, то есть большую часть времени он находится либо в режиме отсечки, когда его сопротивление

максимально, либо в режиме насыщения – с минимальным сопротивлением, а значит, может рассматриваться как ключ.

Линейный стабилизатор – делитель напряжения, на вход которого подаётся входное (нестабильное) напряжение, а выходное (стабилизированное) напряжение снимается с нижнего плеча делителя.

Приложение А

Схема сборки квадрокоптера

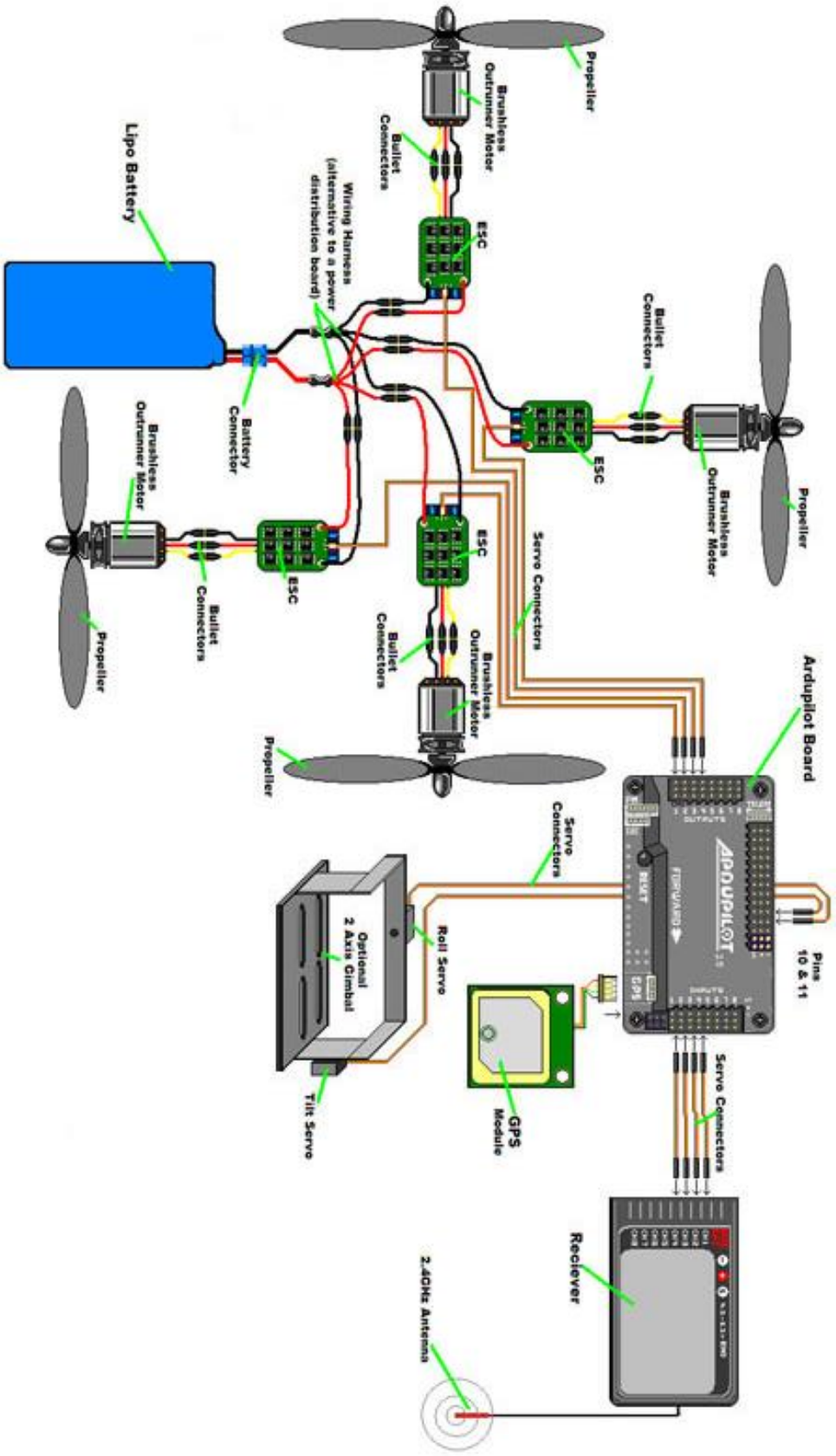

Рисунок А.1 – Принципиальная схема сборки квадрокоптера

Приложение Б

Листинг прошивки

```
#include <math.h>
#include <stdio.h>
#include <stdarg.h>

// Common dependencies
#include <AP_Common.h>
#include <AP_Progmem.h>
#include <AP_Menu.h>
#include <AP_Param.h>
#include <StorageManager.h>
// AP_HAL
#include <AP_HAL.h>
#include <AP_HAL_AVR.h>
#include <AP_HAL_AVR_SITL.h>
#include <AP_HAL_PX4.h>
#include <AP_HAL_VRBRAIN.h>
#include <AP_HAL_FLYMAPLE.h>
#include <AP_HAL_Linux.h>
#include <AP_HAL_Empty.h>

// Application dependencies
#include <GCS.h>
#include <GCS_MAVLink.h>        // MAVLink GCS definitions
#include <AP_GPS.h>             // ArduPilot GPS library
#include <AP_GPS_Glitch.h>      // GPS glitch protection library
#include <DataFlash.h>          // ArduPilot Mega Flash Memory Library
#include <AP_ADC.h>             // ArduPilot Mega Analog to Digital
Converter Library
#include <AP_ADC_AnalogSource.h>
#include <AP_Baro.h>
#include <AP_Baro_Glitch.h>     // Baro glitch protection library
#include <AP_Compass.h>         // ArduPilot Mega Magnetometer Library
#include <AP_Math.h>            // ArduPilot Mega Vector/Matrix math Library
#include <AP_Curve.h>           // Curve used to linearlise throttle pwm to
thrust
#include <AP_InertialSensor.h>  // ArduPilot Mega Inertial Sensor (accel &
gyro) Library
#include <AP_AHRS.h>
#include <AP_NavEKF.h>
```

```
#include <AP_Mission.h>         // Mission command library
#include <AP_Rally.h>           // Rally point library
#include <AC_PID.h>             // PID library
#include <AC_HELI_PID.h>        // Heli specific Rate PID library
#include <AC_P.h>               // P library
#include <AC_AttitudeControl.h> // Attitude control library
#include <AC_AttitudeControl_Heli.h> // Attitude control library for
traditional helicopter
#include <AC_PosControl.h>      // Position control library
#include <RC_Channel.h>         // RC Channel Library
#include <AP_Motors.h>          // AP Motors library
#include <AP_RangeFinder.h>     // Range finder library
#include <AP_OpticalFlow.h>     // Optical Flow library
#include <Filter.h>             // Filter library
#include <AP_Buffer.h>          // APM FIFO Buffer
#include <AP_Relay.h>           // APM relay
#include <AP_ServoRelayEvents.h>
#include <AP_Camera.h>          // Photo or video camera
#include <AP_Mount.h>           // Camera/Antenna mount
#include <AP_Airspeed.h>        // needed for AHRS build
#include <AP_Vehicle.h>         // needed for AHRS build
#include <AP_InertialNav.h>     // ArduPilot Mega inertial navigation library
#include <AC_WPNav.h>     // ArduCopter waypoint navigation library
#include <AC_Circle.h>          // circle navigation library
#include <AP_Declination.h>     // ArduPilot Mega Declination Helper
Library
#include <AC_Fence.h>           // Arducopter Fence library
#include <SITL.h>               // software in the loop support
#include <AP_Scheduler.h>       // main loop scheduler
#include <AP_RCMapper.h>        // RC input mapping library
#include <AP_Notify.h>          // Notify library
#include <AP_BattMonitor.h>     // Battery monitor library
#include <AP_BoardConfig.h>     // board configuration library
#include <AP_Frsky_Telem.h>
#if SPRAYER == ENABLED
#include <AC_Sprayer.h>         // crop sprayer library
#endif
#if EPM_ENABLED == ENABLED
#include <AP_EPM.h>// EPM cargo gripper stuff
#endif
#if PARACHUTE == ENABLED
#include <AP_Parachute.h>// Parachute release library
#endif
```

```
#include <AP_Terrain.h>

// AP_HAL to Arduino compatibility layer
#include "compat.h"
// Configuration
#include "defines.h"
#include "config.h"
#include "config_channels.h"

// key aircraft parameters passed to multiple libraries
static AP_Vehicle::MultiCopter aparm;

// Local modules
#include "Parameters.h"

static AP_HAL::BetterStream* cliSerial;

const AP_HAL::HAL& hal = AP_HAL_BOARD_DRIVER;
static Parameters g;

// main loop scheduler
static AP_Scheduler scheduler;

// AP_Notify instance
static AP_Notify notify;

// used to detect MAVLink acks from GCS to stop compassmot
static uint8_t command_ack_counter;

// has a log download started?
static bool in_log_download;

////////////////////////////////////////////////////////////////////////////////
// prototypes
////////////////////////////////////////////////////////////////////////////////
static void update_events(void);
static void print_flight_mode(AP_HAL::BetterStream *port, uint8_t mode);

////////////////////////////////////////////////////////////////////////////////
// Dataflash
////////////////////////////////////////////////////////////////////////////////
#if CONFIG_HAL_BOARD == HAL_BOARD_APM2
static DataFlash_APM2 DataFlash;
```

```
#elif CONFIG_HAL_BOARD == HAL_BOARD_APM1
static DataFlash_APM1 DataFlash;
#elif defined(HAL_BOARD_LOG_DIRECTORY)
static DataFlash_File DataFlash(HAL_BOARD_LOG_DIRECTORY);
#else
static DataFlash_Empty DataFlash;
#endif

////////////////////////////////////////////////////////////////////////////
// the rate we run the main loop at
////////////////////////////////////////////////////////////////////////////
#if MAIN_LOOP_RATE == 400
static const AP_InertialSensor::Sample_rate ins_sample_rate =
AP_InertialSensor::RATE_400HZ;
#else
static const AP_InertialSensor::Sample_rate ins_sample_rate =
AP_InertialSensor::RATE_100HZ;
#endif


static AP_GPS  gps;

static GPS_Glitch gps_glitch(gps);

// flight modes convenience array
static AP_Int8 *flight_modes = &g.flight_mode1;

#if CONFIG_BARO == HAL_BARO_BMP085
static AP_Baro_BMP085 barometer;
#elif CONFIG_BARO == HAL_BARO_PX4
static AP_Baro_PX4 barometer;
#elif CONFIG_BARO == HAL_BARO_VRBRAIN
static AP_Baro_VRBRAIN barometer;
#elif CONFIG_BARO == HAL_BARO_HIL
static AP_Baro_HIL barometer;
#elif CONFIG_BARO == HAL_BARO_MS5611
static AP_Baro_MS5611 barometer(&AP_Baro_MS5611::i2c);
#elif CONFIG_BARO == HAL_BARO_MS5611_SPI
static AP_Baro_MS5611 barometer(&AP_Baro_MS5611::spi);
#else
 #error Unrecognized CONFIG_BARO setting
#endif
static Baro_Glitch baro_glitch(barometer);
```

```
#if CONFIG_COMPASS == HAL_COMPASS_PX4
static AP_Compass_PX4 compass;
#elif CONFIG_COMPASS == HAL_COMPASS_VRBRAIN
static AP_Compass_VRBRAIN compass;
#elif CONFIG_COMPASS == HAL_COMPASS_HMC5843
static AP_Compass_HMC5843 compass;
#elif CONFIG_COMPASS == HAL_COMPASS_HIL
static AP_Compass_HIL compass;
#else
 #error Unrecognized CONFIG_COMPASS setting
#endif

#if CONFIG_INS_TYPE == HAL_INS_OILPAN ||
CONFIG_HAL_BOARD == HAL_BOARD_APM1
AP_ADC_ADS7844 apm1_adc;
#endif

AP_InertialSensor ins;

// Inertial Navigation EKF
#if AP_AHRS_NAVEKF_AVAILABLE
AP_AHRS_NavEKF ahrs(ins, barometer, gps);
#else
AP_AHRS_DCM ahrs(ins, barometer, gps);
#endif

#if CONFIG_HAL_BOARD == HAL_BOARD_AVR_SITL
SITL sitl;
#endif

// Mission library
// forward declaration to keep compiler happy
static bool start_command(const AP_Mission::Mission_Command& cmd);
static bool verify_command(const AP_Mission::Mission_Command& cmd);
static void exit_mission();
AP_Mission mission(ahrs, &start_command, &verify_command,
&exit_mission);

////////////////////////////////////////////////////////////////////////////
// Optical flow sensor
////////////////////////////////////////////////////////////////////////////
 #if OPTFLOW == ENABLED
```

```
static AP_OpticalFlow_ADNS3080 optflow;
 #endif

////////////////////////////////////////////////////////////////////////////
// GCS selection
////////////////////////////////////////////////////////////////////////////
static const uint8_t num_gcs = MAVLINK_COMM_NUM_BUFFERS;
static GCS_MAVLINK gcs[MAVLINK_COMM_NUM_BUFFERS];

////////////////////////////////////////////////////////////////////////////
// SONAR
#if CONFIG_SONAR == ENABLED
static RangeFinder sonar;
static bool sonar_enabled = true; // enable user switch for sonar
#endif

////////////////////////////////////////////////////////////////////////////
// User variables
////////////////////////////////////////////////////////////////////////////
#ifdef USERHOOK_VARIABLES
 #include USERHOOK_VARIABLES
#endif

static union {
    struct {
        uint8_t home_is_set         : 1; // 0
        uint8_t simple_mode         : 2; // 1,2 // This is the state of simple mode :
0 = disabled ; 1 = SIMPLE ; 2 = SUPERSIMPLE
        uint8_t pre_arm_rc_check    : 1; // 3   // true if rc input pre-arm checks
have been completed successfully
        uint8_t pre_arm_check       : 1; // 4   // true if all pre-arm checks (rc,
accel calibration, gps lock) have been performed
        uint8_t auto_armed          : 1; // 5   // stops auto missions from beginning
until throttle is raised
        uint8_t logging_started     : 1; // 6   // true if dataflash logging has started
        uint8_t land_complete       : 1; // 7   // true if we have detected a landing
        uint8_t new_radio_frame     : 1; // 8       // Set true if we have new PWM
data to act on from the Radio
        uint8_t CH7_flag            : 2; // 9,10   // ch7 aux switch : 0 is low or false,
1 is center or true, 2 is high
        uint8_t CH8_flag            : 2; // 11,12   // ch8 aux switch : 0 is low or
false, 1 is center or true, 2 is high
```

```
        uint8_t usb_connected       : 1; // 13      // true if APM is powered from
USB connection
        uint8_t rc_receiver_present : 1; // 14  // true if we have an rc receiver
present (i.e. if we've ever received an update
        uint8_t compass_mot         : 1; // 15  // true if we are currently
performing compassmot calibration
        uint8_t motor_test          : 1; // 16  // true if we are currently performing
the motors test
        uint8_t initialised         : 1; // 17  // true once the init_ardupilot function
has completed.  Extended status to GCS is not sent until this completes
        uint8_t land_complete_maybe : 1; // 18  // true if we may have landed
(less strict version of land_complete)
        uint8_t throttle_zero       : 1; // 19  // true if the throttle stick is at zero,
debounced
    };
    uint32_t value;
} ap;

static int8_t control_mode = STABILIZE;
static uint8_t oldSwitchPosition;
static RCMapper rcmap;

// board specific config
static AP_BoardConfig BoardConfig;

// receiver RSSI
static uint8_t receiver_rssi;

////////////////////////////////////////////////////////////////////////////////
// Failsafe
////////////////////////////////////////////////////////////////////////////////
static struct {
    uint8_t rc_override_active  : 1; // 0   // true if rc control are overwritten by
ground station
    uint8_t radio               : 1; // 1   // A status flag for the radio failsafe
    uint8_t battery             : 1; // 2   // A status flag for the battery failsafe
    uint8_t gps                 : 1; // 3   // A status flag for the gps failsafe
    uint8_t gcs                 : 1; // 4   // A status flag for the ground station failsafe
    uint8_t ekf                 : 1; // 5   // true if ekf failsafe has occurred

    int8_t radio_counter;               // number of iterations with throttle below
throttle_fs_value
```

```
    uint32_t last_heartbeat_ms;             // the time when the last HEARTBEAT
message arrived from a GCS - used for triggering gcs failsafe
} failsafe;

////////////////////////////////////////////////////////////////////////////////
// Motor Output
////////////////////////////////////////////////////////////////////////////////
#if FRAME_CONFIG == QUAD_FRAME
 #define MOTOR_CLASS AP_MotorsQuad
#elif FRAME_CONFIG == TRI_FRAME
 #define MOTOR_CLASS AP_MotorsTri
#elif FRAME_CONFIG == HEXA_FRAME
 #define MOTOR_CLASS AP_MotorsHexa
#elif FRAME_CONFIG == Y6_FRAME
 #define MOTOR_CLASS AP_MotorsY6
#elif FRAME_CONFIG == OCTA_FRAME
 #define MOTOR_CLASS AP_MotorsOcta
#elif FRAME_CONFIG == OCTA_QUAD_FRAME
 #define MOTOR_CLASS AP_MotorsOctaQuad
#elif FRAME_CONFIG == HELI_FRAME
 #define MOTOR_CLASS AP_MotorsHeli
#elif FRAME_CONFIG == SINGLE_FRAME
 #define MOTOR_CLASS AP_MotorsSingle
#elif FRAME_CONFIG == COAX_FRAME
 #define MOTOR_CLASS AP_MotorsCoax
#else
 #error Unrecognised frame type
#endif

#if FRAME_CONFIG == HELI_FRAME  // helicopter constructor requires
more arguments
static MOTOR_CLASS motors(g.rc_1, g.rc_2, g.rc_3, g.rc_4, g.rc_7, g.rc_8,
g.heli_servo_1, g.heli_servo_2, g.heli_servo_3, g.heli_servo_4);
#elif FRAME_CONFIG == TRI_FRAME  // tri constructor requires
additional rc_7 argument to allow tail servo reversing
static MOTOR_CLASS motors(g.rc_1, g.rc_2, g.rc_3, g.rc_4, g.rc_7);
#elif FRAME_CONFIG == SINGLE_FRAME  // single constructor requires
extra servos for flaps
static MOTOR_CLASS motors(g.rc_1, g.rc_2, g.rc_3, g.rc_4,
g.single_servo_1, g.single_servo_2, g.single_servo_3, g.single_servo_4);
#elif FRAME_CONFIG == COAX_FRAME  // single constructor requires
extra servos for flaps
```

```
static MOTOR_CLASS motors(g.rc_1, g.rc_2, g.rc_3, g.rc_4,
g.single_servo_1, g.single_servo_2);
#else
static MOTOR_CLASS motors(g.rc_1, g.rc_2, g.rc_3, g.rc_4);
#endif

////////////////////////////////////////////////////////////////////////////
// PIDs
////////////////////////////////////////////////////////////////////////////
// This is used to hold radio tuning values for in-flight CH6 tuning
float tuning_value;


////////////////////////////////////////////////////////////////////////////
// GPS variables
////////////////////////////////////////////////////////////////////////////
// We use atan2 and other trig techniques to calaculate angles
// We need to scale the longitude up to make these calcs work
// to account for decreasing distance between lines of longitude away from
the equator
static float scaleLongUp = 1;
// Sometimes we need to remove the scaling for distance calcs
static float scaleLongDown = 1;

////////////////////////////////////////////////////////////////////////////
// Location & Navigation
////////////////////////////////////////////////////////////////////////////
// This is the angle from the copter to the next waypoint in centi-degrees
static int32_t wp_bearing;
// The location of home in relation to the copter in centi-degrees
static int32_t home_bearing;
// distance between plane and home in cm
static int32_t home_distance;
// distance between plane and next waypoint in cm.
static uint32_t wp_distance;
static uint8_t land_state;          // records state of land (flying to location,
descending)

////////////////////////////////////////////////////////////////////////////
// Auto
////////////////////////////////////////////////////////////////////////////
static AutoMode auto_mode;   // controls which auto controller is run
```

```
////////////////////////////////////////////////////////////////////////////
// Guided
////////////////////////////////////////////////////////////////////////////
static GuidedMode guided_mode;  // controls which controller is run (pos or vel)

////////////////////////////////////////////////////////////////////////////
// RTL
////////////////////////////////////////////////////////////////////////////
RTLState rtl_state;  // records state of rtl (initial climb, returning home, etc)
bool rtl_state_complete; // set to true if the current state is completed

////////////////////////////////////////////////////////////////////////////
// Circle
////////////////////////////////////////////////////////////////////////////
bool circle_pilot_yaw_override; // true if pilot is overriding yaw

////////////////////////////////////////////////////////////////////////////
// SIMPLE Mode
////////////////////////////////////////////////////////////////////////////
// Used to track the orientation of the copter for Simple mode. This value is reset at each arming
// or in SuperSimple mode when the copter leaves a 20m radius from home.
static float simple_cos_yaw = 1.0;
static float simple_sin_yaw;
static int32_t super_simple_last_bearing;
static float super_simple_cos_yaw = 1.0;
static float super_simple_sin_yaw;


// Stores initial bearing when armed - initial simple bearing is modified in super simple mode so not suitable
static int32_t initial_armed_bearing;

////////////////////////////////////////////////////////////////////////////
// Throttle variables
////////////////////////////////////////////////////////////////////////////
static float throttle_avg;              // g.throttle_cruise as a float
static int16_t desired_climb_rate;      // pilot desired climb rate - for logging purposes only


////////////////////////////////////////////////////////////////////////////
```

```
// ACRO Mode
////////////////////////////////////////////////////////////////////////////
static float acro_level_mix;              // scales back roll, pitch and yaw
inversely proportional to input from pilot

////////////////////////////////////////////////////////////////////////////
// Loiter control
////////////////////////////////////////////////////////////////////////////
static uint16_t loiter_time_max;              // How long we should stay in
Loiter Mode for mission scripting (time in seconds)
static uint32_t loiter_time;                  // How long have we been loitering -
The start time in millis

////////////////////////////////////////////////////////////////////////////
// Flip
////////////////////////////////////////////////////////////////////////////
static Vector3f flip_orig_attitude;       // original copter attitude before flip

////////////////////////////////////////////////////////////////////////////
// Battery Sensors
////////////////////////////////////////////////////////////////////////////
static AP_BattMonitor battery;

////////////////////////////////////////////////////////////////////////////
// FrSky telemetry support
#if FRSKY_TELEM_ENABLED == ENABLED
static AP_Frsky_Telem frsky_telemetry(ahrs, battery);
#endif

////////////////////////////////////////////////////////////////////////////
// Altitude
////////////////////////////////////////////////////////////////////////////
// The cm/s we are moving up or down based on filtered data - Positive = UP
static int16_t climb_rate;
// The altitude as reported by Sonar in cm - Values are 20 to 700 generally.
static int16_t sonar_alt;
static uint8_t sonar_alt_health;   // true if we can trust the altitude from the
sonar
static float target_sonar_alt;     // desired altitude in cm above the ground
static int32_t baro_alt;           // barometer altitude in cm above home
static float baro_climbrate;       // barometer climbrate in cm/s
```

```
////////////////////////////////////////////////////////////////////////////
// 3D Location vectors
////////////////////////////////////////////////////////////////////////////
// Current location of the copter
static struct   Location current_loc;


////////////////////////////////////////////////////////////////////////////
// Navigation Roll/Pitch functions
////////////////////////////////////////////////////////////////////////////
#if OPTFLOW == ENABLED
// The Commanded ROll from the autopilot based on optical flow sensor.
static int32_t of_roll;
// The Commanded pitch from the autopilot based on optical flow sensor.
negative Pitch means go forward.
static int32_t of_pitch;
#endif // OPTFLOW == ENABLED


////////////////////////////////////////////////////////////////////////////
// Throttle integrator
////////////////////////////////////////////////////////////////////////////
// This is a simple counter to track the amount of throttle used during flight
// This could be useful later in determining and debuging current usage and
predicting battery life
static uint32_t throttle_integrator;


////////////////////////////////////////////////////////////////////////////
// Navigation Yaw control
////////////////////////////////////////////////////////////////////////////
// auto flight mode's yaw mode
static uint8_t auto_yaw_mode = AUTO_YAW_LOOK_AT_NEXT_WP;
// Yaw will point at this location if auto_yaw_mode is set to
AUTO_YAW_ROI
static Vector3f roi_WP;
// bearing from current location to the yaw_look_at_WP
static float yaw_look_at_WP_bearing;
// yaw used for YAW_LOOK_AT_HEADING yaw_mode
static int32_t yaw_look_at_heading;
// Deg/s we should turn
static int16_t yaw_look_at_heading_slew;
// heading when in yaw_look_ahead_bearing
```

```
static float yaw_look_ahead_bearing;


////////////////////////////////////////////////////////////////////////////
// Delay Mission Scripting Command
////////////////////////////////////////////////////////////////////////////
static int32_t condition_value;  // used in condition commands (eg delay, change alt, etc.)
static uint32_t condition_start;


////////////////////////////////////////////////////////////////////////////
// IMU variables
////////////////////////////////////////////////////////////////////////////
// Integration time (in seconds) for the gyros (DCM algorithm)
// Updated with the fast loop
static float G_Dt = 0.02;

////////////////////////////////////////////////////////////////////////////
// Inertial Navigation
////////////////////////////////////////////////////////////////////////////
#if AP_AHRS_NAVEKF_AVAILABLE
static AP_InertialNav_NavEKF inertial_nav(ahrs, barometer, gps_glitch, baro_glitch);
#else
static AP_InertialNav inertial_nav(ahrs, barometer, gps_glitch, baro_glitch);
#endif

////////////////////////////////////////////////////////////////////////////
// Attitude, Position and Waypoint navigation objects
// To-Do: move inertial nav up or other navigation variables down here
////////////////////////////////////////////////////////////////////////////
#if FRAME_CONFIG == HELI_FRAME
AC_AttitudeControl_Heli attitude_control(ahrs, aparm, motors, g.p_stabilize_roll, g.p_stabilize_pitch, g.p_stabilize_yaw,
                  g.pid_rate_roll, g.pid_rate_pitch, g.pid_rate_yaw);
#else
AC_AttitudeControl attitude_control(ahrs, aparm, motors, g.p_stabilize_roll, g.p_stabilize_pitch, g.p_stabilize_yaw,
                  g.pid_rate_roll, g.pid_rate_pitch, g.pid_rate_yaw);
#endif
AC_PosControl pos_control(ahrs, inertial_nav, motors, attitude_control,
                  g.p_alt_hold, g.p_throttle_rate, g.pid_throttle_accel,
```

```
                    g.p_loiter_pos, g.pid_loiter_rate_lat, g.pid_loiter_rate_lon);
static AC_WPNav wp_nav(inertial_nav, ahrs, pos_control);
static AC_Circle circle_nav(inertial_nav, ahrs, pos_control);

////////////////////////////////////////////////////////////////////////////
// Performance monitoring
////////////////////////////////////////////////////////////////////////////
static int16_t pmTest1;

// System Timers
// --------------
// Time in microseconds of main control loop
static uint32_t fast_loopTimer;
// Counter of main loop executions.  Used for performance monitoring and
failsafe processing
static uint16_t mainLoop_count;
// Loiter timer - Records how long we have been in loiter
static uint32_t rtl_loiter_start_time;

// Used to exit the roll and pitch auto trim function
static uint8_t auto_trim_counter;

// Reference to the relay object (APM1 -> PORTL 2) (APM2 -> PORTB 7)
static AP_Relay relay;

// handle repeated servo and relay events
static AP_ServoRelayEvents ServoRelayEvents(relay);

//Reference to the camera object (it uses the relay object inside it)
#if CAMERA == ENABLED
  static AP_Camera camera(&relay);
#endif

// a pin for reading the receiver RSSI voltage.
static AP_HAL::AnalogSource* rssi_analog_source;

#if CLI_ENABLED == ENABLED
    static int8_t   setup_show (uint8_t argc, const Menu::arg *argv);
#endif

// Camera/Antenna mount tracking and stabilisation stuff
// --------------------------------------
#if MOUNT == ENABLED
```

```
// current_loc uses the baro/gps soloution for altitude rather than gps only.
static AP_Mount camera_mount(&current_loc, ahrs, 0);
#endif

#if MOUNT2 == ENABLED
// current_loc uses the baro/gps soloution for altitude rather than gps only.
static AP_Mount camera_mount2(&current_loc, ahrs, 1);
#endif

////////////////////////////////////////////////////////////////////////////////
// AC_Fence library to reduce fly-aways
////////////////////////////////////////////////////////////////////////////////
#if AC_FENCE == ENABLED
AC_Fence    fence(&inertial_nav);
#endif

////////////////////////////////////////////////////////////////////////////////
// Rally library
////////////////////////////////////////////////////////////////////////////////
#if AC_RALLY == ENABLED
AP_Rally rally(ahrs);
#endif

////////////////////////////////////////////////////////////////////////////////
// Crop Sprayer
////////////////////////////////////////////////////////////////////////////////
#if SPRAYER == ENABLED
static AC_Sprayer sprayer(&inertial_nav);
#endif

////////////////////////////////////////////////////////////////////////////////
// EPM Cargo Griper
////////////////////////////////////////////////////////////////////////////////
#if EPM_ENABLED == ENABLED
static AP_EPM epm;
#endif

////////////////////////////////////////////////////////////////////////////////
// Parachute release
////////////////////////////////////////////////////////////////////////////////
#if PARACHUTE == ENABLED
static AP_Parachute parachute(relay);
#endif
```

```
////////////////////////////////////////////////////////////////////////////
// terrain handling
#if AP_TERRAIN_AVAILABLE
AP_Terrain terrain(ahrs, mission, rally);
#endif

////////////////////////////////////////////////////////////////////////////
// Nav Guided - allows external computer to control the vehicle during missions
////////////////////////////////////////////////////////////////////////////
#if NAV_GUIDED == ENABLED
static struct {
   uint32_t start_time;       // system time in milliseconds that control was handed to the external computer
   Vector3f start_position;   // vehicle position when control was ahnded to the external computer
} nav_guided;
#endif

////////////////////////////////////////////////////////////////////////////
// function definitions to keep compiler from complaining about undeclared functions
////////////////////////////////////////////////////////////////////////////
static void pre_arm_checks(bool display_failure);

////////////////////////////////////////////////////////////////////////////
// Top-level logic
////////////////////////////////////////////////////////////////////////////

// setup the var_info table
AP_Param param_loader(var_info);

#if MAIN_LOOP_RATE == 400
static const AP_Scheduler::Task scheduler_tasks[] PROGMEM = {
   { rc_loop,              4,     10 },
   { throttle_loop,        8,     45 },
   { update_GPS,           8,     90 },
   { update_batt_compass,  40,    72 },
   { read_aux_switches,    40,    5 },
   { arm_motors_check,     40,    1 },
   { auto_trim,            40,    14 },
   { update_altitude,      40,    100 },
```

```
    { run_nav_updates,       8,     80 },
    { update_thr_cruise,    40,     10 },
    { three_hz_loop,       133,      9 },
    { compass_accumulate,    8,     42 },
    { barometer_accumulate,  8,     25 },
#if FRAME_CONFIG == HELI_FRAME
    { check_dynamic_flight,  8,     10 },
#endif
    { update_notify,         8,     10 },
    { one_hz_loop,         400,     42 },
    { ekf_dcm_check,        40,      2 },
    { crash_check,          40,      2 },
    { gcs_check_input,       8,    550 },
    { gcs_send_heartbeat,  400,    150 },
    { gcs_send_deferred,     8,    720 },
    { gcs_data_stream_send,  8,    950 },
#if COPTER_LEDS == ENABLED
    { update_copter_leds,   40,      5 },
#endif
    { update_mount,          8,     45 },
    { ten_hz_logging_loop,  40,     30 },
    { fifty_hz_logging_loop, 8,     22 },
    { perf_update,        4000,     20 },
    { read_receiver_rssi,   40,      5 },
#if FRSKY_TELEM_ENABLED == ENABLED
    { telemetry_send,       80,     10 },
#endif
#ifdef USERHOOK_FASTLOOP
    { userhook_FastLoop,     4,     10 },
#endif
#ifdef USERHOOK_50HZLOOP
    { userhook_50Hz,         8,     10 },
#endif
#ifdef USERHOOK_MEDIUMLOOP
    { userhook_MediumLoop,  40,     10 },
#endif
#ifdef USERHOOK_SLOWLOOP
    { userhook_SlowLoop,    120,    10 },
#endif
#ifdef USERHOOK_SUPERSLOWLOOP
    { userhook_SuperSlowLoop,400,   10 },
#endif
};
```

```
#else
static const AP_Scheduler::Task scheduler_tasks[] PROGMEM = {
    { rc_loop,              1,     100 },
    { throttle_loop,        2,     450 },
    { update_GPS,           2,     900 },
    { update_batt_compass,  10,    720 },
    { read_aux_switches,    10,    50 },
    { arm_motors_check,     10,    10 },
    { auto_trim,            10,    140 },
    { update_altitude,      10,    1000 },
    { run_nav_updates,      4,     800 },
    { update_thr_cruise,    1,     50 },
    { three_hz_loop,        33,    90 },
    { compass_accumulate,   2,     420 },
    { barometer_accumulate, 2,     250 },
#if FRAME_CONFIG == HELI_FRAME
    { check_dynamic_flight, 2,     100 },
#endif
    { update_notify,        2,     100 },
    { one_hz_loop,          100,   420 },
    { ekf_dcm_check,        10,    20 },
    { crash_check,          10,    20 },
    { gcs_check_input,      2,     550 },
    { gcs_send_heartbeat,   100,   150 },
    { gcs_send_deferred,    2,     720 },
    { gcs_data_stream_send, 2,     950 },
    { update_mount,         2,     450 },
    { ten_hz_logging_loop,  10,    300 },
    { fifty_hz_logging_loop, 2,    220 },
    { perf_update,          1000,  200 },
    { read_receiver_rssi,   10,    50 },
#if FRSKY_TELEM_ENABLED == ENABLED
    { telemetry_send,       20,    100 },
#endif
#ifdef USERHOOK_FASTLOOP
    { userhook_FastLoop,    1,     100  },
#endif
#ifdef USERHOOK_50HZLOOP
    { userhook_50Hz,        2,     100  },
#endif
#ifdef USERHOOK_MEDIUMLOOP
    { userhook_MediumLoop,  10,    100 },
#endif
```

```
#ifdef USERHOOK_SLOWLOOP
   { userhook_SlowLoop,     30,    100 },
#endif
#ifdef USERHOOK_SUPERSLOWLOOP
   { userhook_SuperSlowLoop,100,   100 },
#endif
};
#endif


void setup()
{
   cliSerial = hal.console;

   // Load the default values of variables listed in var_info[]s
   AP_Param::setup_sketch_defaults();

   // setup storage layout for copter
   StorageManager::set_layout_copter();

   init_ardupilot();

   // initialise the main loop scheduler
   scheduler.init(&scheduler_tasks[0],
sizeof(scheduler_tasks)/sizeof(scheduler_tasks[0]));
}

/*
  if the compass is enabled then try to accumulate a reading
 */
static void compass_accumulate(void)
{
   if (g.compass_enabled) {
      compass.accumulate();
   }
}

/*
  try to accumulate a baro reading
 */
static void barometer_accumulate(void)
{
   barometer.accumulate();
```

```
}

static void perf_update(void)
{
    if (should_log(MASK_LOG_PM))
        Log_Write_Performance();
    if (scheduler.debug()) {
        cliSerial->printf_P(PSTR("PERF: %u/%u %lu\n"),
                            (unsigned)perf_info_get_num_long_running(),
                            (unsigned)perf_info_get_num_loops(),
                            (unsigned long)perf_info_get_max_time());
    }
    perf_info_reset();
    pmTest1 = 0;
}

void loop()
{
    // wait for an INS sample
    ins.wait_for_sample();
    uint32_t timer = micros();

    // check loop time
    perf_info_check_loop_time(timer - fast_loopTimer);

    // used by PI Loops
    G_Dt                    = (float)(timer - fast_loopTimer) / 1000000.f;
    fast_loopTimer          = timer;

    // for mainloop failure monitoring
    mainLoop_count++;

    // Execute the fast loop
    // ---------------------
    fast_loop();

    // tell the scheduler one tick has passed
    scheduler.tick();

    // run all the tasks that are due to run. Note that we only
    // have to call this once per loop, as the tasks are scheduled
    // in multiples of the main loop tick. So if they don't run on
    // the first call to the scheduler they won't run on a later
```

```
    // call until scheduler.tick() is called again
    uint32_t time_available = (timer + MAIN_LOOP_MICROS) - micros();
    scheduler.run(time_available);
}


// Main loop - 100hz
static void fast_loop()
{

    // IMU DCM Algorithm
    // --------------------
    read_AHRS();

    // run low level rate controllers that only require IMU data
    attitude_control.rate_controller_run();

#if FRAME_CONFIG == HELI_FRAME
    update_heli_control_dynamics();
#endif //HELI_FRAME

    // write out the servo PWM values
    // ------------------------------
    set_servos_4();

    // Inertial Nav
    // --------------------
    read_inertia();

    // run the attitude controllers
    update_flight_mode();

    // optical flow
    // --------------------
#if OPTFLOW == ENABLED
    if(g.optflow_enabled) {
        update_optical_flow();
    }
#endif  // OPTFLOW == ENABLED

}

// rc_loops - reads user input from transmitter/receiver
```

```
// called at 100hz
static void rc_loop()
{
    // Read radio and 3-position switch on radio
    // ----------------------------------------
    read_radio();
    read_control_switch();
}

// throttle_loop - should be run at 50 hz
// ---------------------------
static void throttle_loop()
{
    // get altitude and climb rate from inertial lib
    read_inertial_altitude();

    // check if we've landed
    update_land_detector();

    // check auto_armed status
    update_auto_armed();

#if FRAME_CONFIG == HELI_FRAME
    // update rotor speed
    heli_update_rotor_speed_targets();

    // update trad heli swash plate movement
    heli_update_landing_swash();
#endif
}

// update_mount - update camera mount position
// should be run at 50hz
static void update_mount()
{
#if MOUNT == ENABLED
    // update camera mount's position
    camera_mount.update_mount_position();
#endif

#if MOUNT2 == ENABLED
    // update camera mount's position
    camera_mount2.update_mount_position();
```

```
#endif

#if CAMERA == ENABLED
   camera.trigger_pic_cleanup();
#endif
}

// update_batt_compass - read battery and compass
// should be called at 10hz
static void update_batt_compass(void)
{
   // read battery before compass because it may be used for motor
interference compensation
   read_battery();

   if(g.compass_enabled) {
      // update compass with throttle value - used for compassmot
      compass.set_throttle((float)g.rc_3.servo_out/1000.0f);
      compass.read();
      // log compass information
      if (should_log(MASK_LOG_COMPASS)) {
         Log_Write_Compass();
      }
   }

   // record throttle output
   throttle_integrator += g.rc_3.servo_out;
}

// ten_hz_logging_loop
// should be run at 10hz
static void ten_hz_logging_loop()
{
   if (should_log(MASK_LOG_ATTITUDE_MED)) {
      Log_Write_Attitude();
   }
   if (should_log(MASK_LOG_RCIN)) {
      DataFlash.Log_Write_RCIN();
   }
   if (should_log(MASK_LOG_RCOUT)) {
      DataFlash.Log_Write_RCOUT();
   }
```

```
    if (should_log(MASK_LOG_NTUN) &&
(mode_requires_GPS(control_mode) || landing_with_GPS())) {
        Log_Write_Nav_Tuning();
    }
}

// fifty_hz_logging_loop
// should be run at 50hz
static void fifty_hz_logging_loop()
{
#if HIL_MODE != HIL_MODE_DISABLED
    // HIL for a copter needs very fast update of the servo values
    gcs_send_message(MSG_RADIO_OUT);
#endif

#if HIL_MODE == HIL_MODE_DISABLED
    if (should_log(MASK_LOG_ATTITUDE_FAST)) {
        Log_Write_Attitude();
    }

    if (should_log(MASK_LOG_IMU)) {
        DataFlash.Log_Write_IMU(ins);
    }
#endif
}

// three_hz_loop - 3.3hz loop
static void three_hz_loop()
{
    // check if we've lost contact with the ground station
    failsafe_gcs_check();

#if AC_FENCE == ENABLED
    // check if we have breached a fence
    fence_check();
#endif // AC_FENCE_ENABLED

#if SPRAYER == ENABLED
    sprayer.update();
#endif

    update_events();
```

```
    if(g.radio_tuning > 0)
        tuning();
}

// one_hz_loop - runs at 1Hz
static void one_hz_loop()
{
    if (should_log(MASK_LOG_ANY)) {
        Log_Write_Data(DATA_AP_STATE, ap.value);
    }

    // log battery info to the dataflash
    if (should_log(MASK_LOG_CURRENT)) {
        Log_Write_Current();
    }

    // perform pre-arm checks & display failures every 30 seconds
    static uint8_t pre_arm_display_counter = 15;
    pre_arm_display_counter++;
    if (pre_arm_display_counter >= 30) {
        pre_arm_checks(true);
        pre_arm_display_counter = 0;
    }else{
        pre_arm_checks(false);
    }

    // auto disarm checks
    auto_disarm_check();

    if (!motors.armed()) {
        // make it possible to change ahrs orientation at runtime during initial
config
        ahrs.set_orientation();

        // check the user hasn't updated the frame orientation
        motors.set_frame_orientation(g.frame_orientation);
    }

    // update assigned functions and enable auxiliar servos
    RC_Channel_aux::enable_aux_servos();

#if MOUNT == ENABLED
    camera_mount.update_mount_type();
```

```
#endif

#if MOUNT2 == ENABLED
   camera_mount2.update_mount_type();
#endif

   check_usb_mux();

#if AP_TERRAIN_AVAILABLE
   terrain.update();
#endif

#if AC_FENCE == ENABLED
   // set fence altitude limit in position controller
   if ((fence.get_enabled_fences() & AC_FENCE_TYPE_ALT_MAX) != 0)
{
      pos_control.set_alt_max(fence.get_safe_alt()*100.0f);
   }
#endif
}

// called at 100hz but data from sensor only arrives at 20 Hz
#if OPTFLOW == ENABLED
static void update_optical_flow(void)
{
   static uint32_t last_of_update = 0;
   static uint8_t of_log_counter = 0;

   // if new data has arrived, process it
   if( optflow.last_update != last_of_update ) {
      last_of_update = optflow.last_update;
      optflow.update_position(ahrs.roll, ahrs.pitch, ahrs.sin_yaw(),
ahrs.cos_yaw(), current_loc.alt);      // updates internal lon and lat with
estimation based on optical flow

      // write to log at 5hz
      of_log_counter++;
      if( of_log_counter >= 4 ) {
         of_log_counter = 0;
         if (should_log(MASK_LOG_OPTFLOW)) {
            Log_Write_Optflow();
         }
      }
```

```
    }
}
#endif  // OPTFLOW == ENABLED

// called at 50hz
static void update_GPS(void)
{
    static uint32_t last_gps_reading[GPS_MAX_INSTANCES];   // time of
last gps message
    static uint8_t ground_start_count = 10;     // counter used to grab at least 10
reads before commiting the Home location
    bool report_gps_glitch;
    bool gps_updated = false;

    gps.update();

    // logging and glitch protection run after every gps message
    for (uint8_t i=0; i<gps.num_sensors(); i++) {
        if (gps.last_message_time_ms(i) != last_gps_reading[i]) {
            last_gps_reading[i] = gps.last_message_time_ms(i);

            // log GPS message
            if (should_log(MASK_LOG_GPS)) {
                DataFlash.Log_Write_GPS(gps, i, current_loc.alt);
            }

            gps_updated = true;
        }
    }

    if (gps_updated) {
        // run glitch protection and update AP_Notify if home has been
initialised
        if (ap.home_is_set) {
            gps_glitch.check_position();
            report_gps_glitch = (gps_glitch.glitching() && !ap.usb_connected
&& hal.util->safety_switch_state() !=
AP_HAL::Util::SAFETY_DISARMED);
            if (AP_Notify::flags.gps_glitching != report_gps_glitch) {
                if (gps_glitch.glitching()) {
                    Log_Write_Error(ERROR_SUBSYSTEM_GPS,
ERROR_CODE_GPS_GLITCH);
                }else{
```

```
                Log_Write_Error(ERROR_SUBSYSTEM_GPS,
ERROR_CODE_ERROR_RESOLVED);
            }
            AP_Notify::flags.gps_glitching = report_gps_glitch;
        }
    }

    // checks to initialise home and take location based pictures
    if (gps.status() >= AP_GPS::GPS_OK_FIX_3D) {

        // check if we can initialise home yet
        if (!ap.home_is_set) {
            // if we have a 3d lock and valid location
            if(gps.status() >= AP_GPS::GPS_OK_FIX_3D &&
gps.location().lat != 0) {
                if (ground_start_count > 0 ) {
                    ground_start_count--;
                } else {
                    // after 10 successful reads store home location
                    // ap.home_is_set will be true so this will only happen once
                    ground_start_count = 0;
                    init_home();

                    // set system clock for log timestamps
                    hal.util->set_system_clock(gps.time_epoch_usec());

                    if (g.compass_enabled) {
                        // Set compass declination automatically
                        compass.set_initial_location(gps.location().lat,
gps.location().lng);
                    }
                }
            } else {
                // start again if we lose 3d lock
                ground_start_count = 10;
            }
        }

        //If we are not currently armed, and we're ready to
        //enter RTK mode, then capture current state as home,
        //and enter RTK fixes!
        if (!motors.armed() && gps.can_calculate_base_pos()) {
```

```
            gps.calculate_base_pos();

        }

#if CAMERA == ENABLED
        if (camera.update_location(current_loc) == true) {
            do_take_picture();
        }
#endif
    }
}

    // check for loss of gps
    failsafe_gps_check();
}

static void
init_simple_bearing()
{
    // capture current cos_yaw and sin_yaw values
    simple_cos_yaw = ahrs.cos_yaw();
    simple_sin_yaw = ahrs.sin_yaw();

    // initialise super simple heading (i.e. heading towards home) to be 180 deg
from simple mode heading
    super_simple_last_bearing = wrap_360_cd(ahrs.yaw_sensor+18000);
    super_simple_cos_yaw = simple_cos_yaw;
    super_simple_sin_yaw = simple_sin_yaw;

    // log the simple bearing to dataflash
    if (should_log(MASK_LOG_ANY)) {
        Log_Write_Data(DATA_INIT_SIMPLE_BEARING, ahrs.yaw_sensor);
    }
}

// update_simple_mode - rotates pilot input if we are in simple mode
void update_simple_mode(void)
{
    float rollx, pitchx;

    // exit immediately if no new radio frame or not in simple mode
    if (ap.simple_mode == 0 || !ap.new_radio_frame) {
        return;
```

```
    }

    // mark radio frame as consumed
    ap.new_radio_frame = false;

    if (ap.simple_mode == 1) {
        // rotate roll, pitch input by -initial simple heading (i.e. north facing)
        rollx = g.rc_1.control_in*simple_cos_yaw -
g.rc_2.control_in*simple_sin_yaw;
        pitchx = g.rc_1.control_in*simple_sin_yaw +
g.rc_2.control_in*simple_cos_yaw;
    }else{
        // rotate roll, pitch input by -super simple heading (reverse of heading to
home)
        rollx = g.rc_1.control_in*super_simple_cos_yaw -
g.rc_2.control_in*super_simple_sin_yaw;
        pitchx = g.rc_1.control_in*super_simple_sin_yaw +
g.rc_2.control_in*super_simple_cos_yaw;
    }

    // rotate roll, pitch input from north facing to vehicle's perspective
    g.rc_1.control_in = rollx*ahrs.cos_yaw() + pitchx*ahrs.sin_yaw();
    g.rc_2.control_in = -rollx*ahrs.sin_yaw() + pitchx*ahrs.cos_yaw();
}

// update_super_simple_bearing - adjusts simple bearing based on location
// should be called after home_bearing has been updated
void update_super_simple_bearing(bool force_update)
{
    // check if we are in super simple mode and at least 10m from home
    if(force_update || (ap.simple_mode == 2 && home_distance >
SUPER_SIMPLE_RADIUS)) {
        // check the bearing to home has changed by at least 5 degrees
        if (labs(super_simple_last_bearing - home_bearing) > 500) {
            super_simple_last_bearing = home_bearing;
            float angle_rad = radians((super_simple_last_bearing+18000)/100);
            super_simple_cos_yaw = cosf(angle_rad);
            super_simple_sin_yaw = sinf(angle_rad);
        }
    }
}

static void read_AHRS(void)
```

```
{
    // Perform IMU calculations and get attitude info
    //-----------------------------------------------
#if HIL_MODE != HIL_MODE_DISABLED
    // update hil before ahrs update
    gcs_check_input();
#endif

    ahrs.update();
}

// read baro and sonar altitude at 10hz
static void update_altitude()
{
    // read in baro altitude
    read_barometer();

    // read in sonar altitude
    sonar_alt           = read_sonar();

    // write altitude info to dataflash logs
    if (should_log(MASK_LOG_CTUN)) {
        Log_Write_Control_Tuning();
    }
}

static void tuning(){

    // exit immediately when radio failsafe is invoked so tuning values are not
set to zero
    if (failsafe.radio || failsafe.radio_counter != 0) {
        return;
    }

    tuning_value = (float)g.rc_6.control_in / 1000.0f;
    g.rc_6.set_range(g.radio_tuning_low,g.radio_tuning_high);                 // 0
to 1

    switch(g.radio_tuning) {

    // Roll, Pitch tuning
    case CH6_STABILIZE_ROLL_PITCH_KP:
        g.p_stabilize_roll.kP(tuning_value);
```

```
    g.p_stabilize_pitch.kP(tuning_value);
    break;

case CH6_RATE_ROLL_PITCH_KP:
    g.pid_rate_roll.kP(tuning_value);
    g.pid_rate_pitch.kP(tuning_value);
    break;

case CH6_RATE_ROLL_PITCH_KI:
    g.pid_rate_roll.kI(tuning_value);
    g.pid_rate_pitch.kI(tuning_value);
    break;

case CH6_RATE_ROLL_PITCH_KD:
    g.pid_rate_roll.kD(tuning_value);
    g.pid_rate_pitch.kD(tuning_value);
    break;

// Yaw tuning
case CH6_STABILIZE_YAW_KP:
    g.p_stabilize_yaw.kP(tuning_value);
    break;

case CH6_YAW_RATE_KP:
    g.pid_rate_yaw.kP(tuning_value);
    break;

case CH6_YAW_RATE_KD:
    g.pid_rate_yaw.kD(tuning_value);
    break;

// Altitude and throttle tuning
case CH6_ALTITUDE_HOLD_KP:
    g.p_alt_hold.kP(tuning_value);
    break;

case CH6_THROTTLE_RATE_KP:
    g.p_throttle_rate.kP(tuning_value);
    break;

case CH6_THROTTLE_ACCEL_KP:
    g.pid_throttle_accel.kP(tuning_value);
    break;
```

```
case CH6_THROTTLE_ACCEL_KI:
    g.pid_throttle_accel.kI(tuning_value);
    break;

case CH6_THROTTLE_ACCEL_KD:
    g.pid_throttle_accel.kD(tuning_value);
    break;

// Loiter and navigation tuning
case CH6_LOITER_POSITION_KP:
    g.p_loiter_pos.kP(tuning_value);
    break;

case CH6_LOITER_RATE_KP:
    g.pid_loiter_rate_lon.kP(tuning_value);
    g.pid_loiter_rate_lat.kP(tuning_value);
    break;

case CH6_LOITER_RATE_KI:
    g.pid_loiter_rate_lon.kI(tuning_value);
    g.pid_loiter_rate_lat.kI(tuning_value);
    break;

case CH6_LOITER_RATE_KD:
    g.pid_loiter_rate_lon.kD(tuning_value);
    g.pid_loiter_rate_lat.kD(tuning_value);
    break;

case CH6_WP_SPEED:
    // set waypoint navigation horizontal speed to 0 ~ 1000 cm/s
    wp_nav.set_speed_xy(g.rc_6.control_in);
    break;

// Acro roll pitch gain
case CH6_ACRO_RP_KP:
    g.acro_rp_p = tuning_value;
    break;

// Acro yaw gain
case CH6_ACRO_YAW_KP:
    g.acro_yaw_p = tuning_value;
    break;
```

```
#if FRAME_CONFIG == HELI_FRAME
    case CH6_HELI_EXTERNAL_GYRO:
        motors.ext_gyro_gain(g.rc_6.control_in);
        break;

    case CH6_RATE_PITCH_FF:
        g.pid_rate_pitch.ff(tuning_value);
        break;

    case CH6_RATE_ROLL_FF:
        g.pid_rate_roll.ff(tuning_value);
        break;

    case CH6_RATE_YAW_FF:
        g.pid_rate_yaw.ff(tuning_value);
        break;
#endif

    case CH6_OPTFLOW_KP:
        g.pid_optflow_roll.kP(tuning_value);
        g.pid_optflow_pitch.kP(tuning_value);
        break;

    case CH6_OPTFLOW_KI:
        g.pid_optflow_roll.kI(tuning_value);
        g.pid_optflow_pitch.kI(tuning_value);
        break;

    case CH6_OPTFLOW_KD:
        g.pid_optflow_roll.kD(tuning_value);
        g.pid_optflow_pitch.kD(tuning_value);
        break;

    case CH6_AHRS_YAW_KP:
        ahrs._kp_yaw.set(tuning_value);
        break;

    case CH6_AHRS_KP:
        ahrs._kp.set(tuning_value);
        break;

    case CH6_DECLINATION:
```

```
        // set declination to +-20degrees
        compass.set_declination(ToRad((2.0f * g.rc_6.control_in -
g.radio_tuning_high)/100.0f), false);     // 2nd parameter is false because we
do not want to save to eeprom because this would have a performance impact
        break;

    case CH6_CIRCLE_RATE:
        // set circle rate
        circle_nav.set_rate(g.rc_6.control_in/25-20);   // allow approximately 45
degree turn rate in either direction
        break;

    case CH6_SONAR_GAIN:
        // set sonar gain
        g.sonar_gain.set(tuning_value);
        break;

#if 0
        // disabled for now - we need accessor functions
    case CH6_EKF_VERTICAL_POS:
        // EKF's baro vs accel (higher rely on accels more, baro impact is
reduced)
        ahrs.get_NavEKF()._gpsVertPosNoise = tuning_value;
        break;

    case CH6_EKF_HORIZONTAL_POS:
        // EKF's gps vs accel (higher rely on accels more, gps impact is reduced)
        ahrs.get_NavEKF()._gpsHorizPosNoise = tuning_value;
        break;

    case CH6_EKF_ACCEL_NOISE:
        // EKF's accel noise (lower means trust accels more, gps & baro less)
        ahrs.get_NavEKF()._accNoise = tuning_value;
        break;
#endif

    case CH6_RC_FEEL_RP:
        // roll-pitch input smoothing
        g.rc_feel_rp = g.rc_6.control_in / 10;
        break;

    case CH6_RATE_PITCH_KP:
        g.pid_rate_pitch.kP(tuning_value);
```

```
            break;

        case CH6_RATE_PITCH_KI:
            g.pid_rate_pitch.kI(tuning_value);
            break;

        case CH6_RATE_PITCH_KD:
            g.pid_rate_pitch.kD(tuning_value);
            break;

        case CH6_RATE_ROLL_KP:
            g.pid_rate_roll.kP(tuning_value);
            break;

        case CH6_RATE_ROLL_KI:
            g.pid_rate_roll.kI(tuning_value);
            break;

        case CH6_RATE_ROLL_KD:
            g.pid_rate_roll.kD(tuning_value);
            break;
        }
    }

AP_HAL_MAIN();
```