Оглавление

[**Квадрокоптер на raspberry pi и Arduino** 2](#_Toc38623765)

[**Схема подключения** 2](#_Toc38623766)

[**Элементы коптера и их алгоритм решения задач** 3](#_Toc38623767)

[**Arduino** 4](#_Toc38623768)

[**Ультразвуковой дальномер** 4](#_Toc38623769)

[**БИНС** 5](#_Toc38623770)

[**Контроллер бесколлекторных двигателей** 5](#_Toc38623771)

[**Raspberry** 6](#_Toc38623772)

[**Удаленное управление на компьютере через Wi-Fi** 6](#_Toc38623773)

[**Заключение** 6](#_Toc38623774)

[**Программное обеспечение** 8](#_Toc38623775)

[**Структура программного обеспечения** 8](#_Toc38623776)

[**Автопилот** 9](#_Toc38623777)

[**Основные параметры и принципы работы** 9](#_Toc38623778)

[**Расчет основных параметров** 9](#_Toc38623779)

[**Режимы автопилота** 9](#_Toc38623780)

[**ARMING** 9](#_Toc38623781)

[**Взлет на заданную высоту** 9](#_Toc38623782)

[**Удержание высоты** 9](#_Toc38623783)

[**Движение вперед** 9](#_Toc38623784)

[**Поворот на заданный курс** 9](#_Toc38623785)

[**Посадка** 9](#_Toc38623786)

# **Квадрокоптер на raspberry pi и Arduino**

Для осуществления определенных задач с помощью квадрокоптера (поиск, идентификация, слежение, нахождение оптимального маршрута и т.д.), которые требуют больших вычислительных возможностей, требуется использовать микрокомпьютер (контроллер) на борту летательного аппарата (ЛА). Для быстрой модификации или замены необходимых элементов на борту ЛА будет практична связка из нескольких контроллеров, которые, в свою очередь, будут выполнять определенные функции самостоятельно. В нашем случае будет использована связка контроллера Arduino и raspberry pi. Необходимо добавить, что контроллер raspberry позволяет осуществлять связь между персональным компьютером через беспроводную сеть WI-Fi. Рассмотрим описанное выше подключение подробнее.

Схема подключения (рис.1) работает следующим образом:

## **Схема подключения**

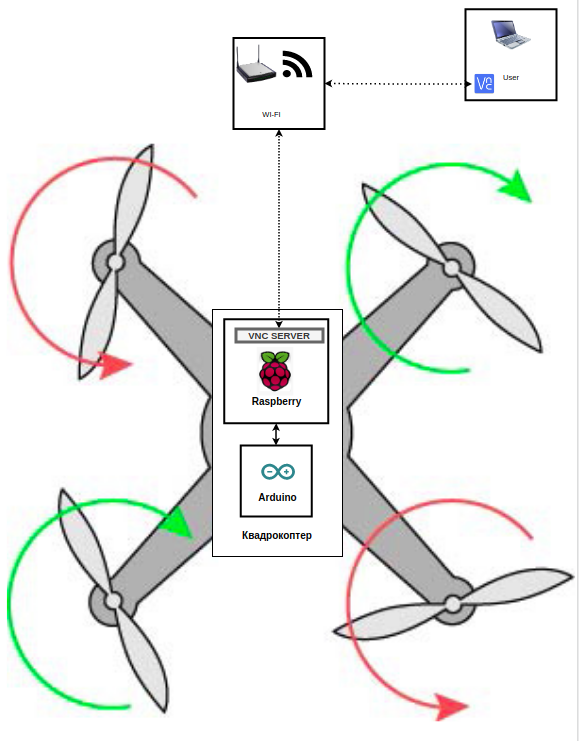
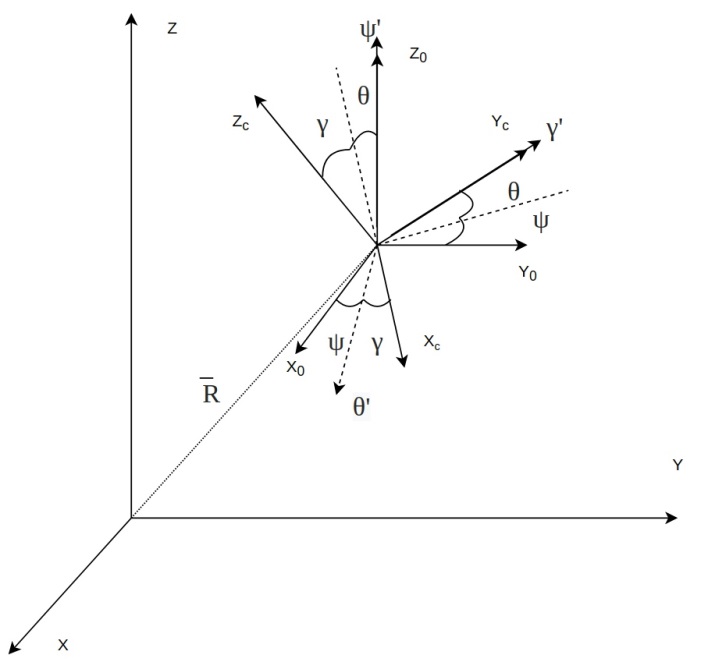


Рисунок 1

1. На квадрокоптере установлен Raspberry pi, который выполняет функцию блока вычисления автономного пилотирования, а также функции компьютера дистанционного управления через VNC сервер. Передает на блок Arduino вычисленные значения скоростей двигателей, необходимые для решения текущей задачи.
2. Блок Arduino включает в себя бесплатформенную инерциальную навигационную систему (БИНС), дальномеры, контроллеры двигателей. Данный блок передает в блок вычисления автономного пилотирования информацию с дальномера и вектор состояния летательного аппарата, о котором будет сказано чуть дальше.

## **Системы координат и характеристики квадроптера**

Поскольку летательному аппарату для навигации необходимо измерять свое местоположение в пространстве, то будем использовать следующие обозначения систем отчета, которые нам необходимы.



На рисунке … изображены 3 необходимые нам системы координат:

1. Стартовая система координат XYZ.

Точка отчета находится в точке запуска квадрокоптера

X – ось, направленная на север;

Z – ось, направленная вертикально вверх

Y – дополняет до правой тройки векторов

1. Сопровождающая система координат X0 Y0 Z0.

Оси паралельны осям стартовой системы координат, но точка отсчета в центре масс квадрокоптера.

1. Связанная система координат Xс Yс Zс.

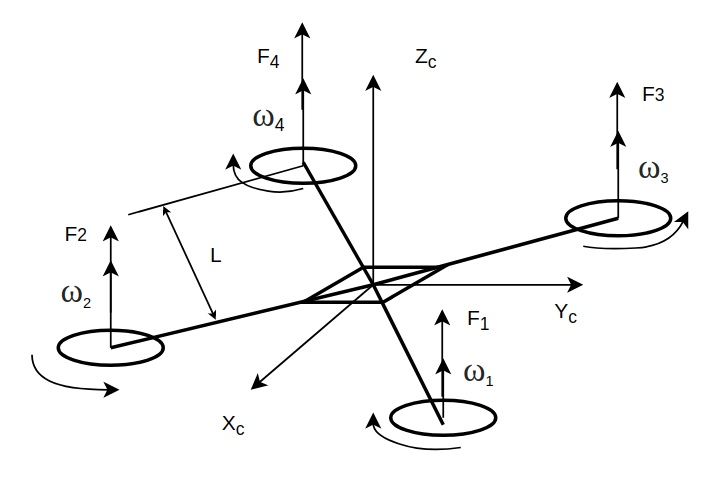
Xс – ось, направленная вдоль строи;

Zс – ось, перпендикулярна плоскости, в которой находятся 4 двигателя, и направлена вертикально вверх;

Yс – дополняет до правой тройки векторов;

Оси повернуты на углы тангажа, крена и рыскания относительно осей земной системы координат.

В нашем случае летательный аппарат оснащен 4 двигателями, расположенными крестообразно на 4 стойках.



Достоинства такой схемы:

+ Устойчивость при внешних нагрузках

+ Возможность набрать большую мощность

+ Маневренность в плоскости в любые стороны одинакова

+ Возможность при поломке одного из ротеров, изменяя мощность на остальных двигателях, безопасно завершить полет

Недостатки данной схемы:

- более тяжелый вес

- большие энергозатраты

На квадрокоптер действуют аэродинамические силы тяги несущих винтов *F1, F2, F3, F4*, приложенные к их центрам масс роторов соответственно и силы тяжести корпуса *mк* и винтов *mвi*. Ротеры расположены на расстоянии *L* от центра масс и повернуты на угол η. Необходимо добавить, что *Fi* пропорционален . Где: : скорости вращения пропеллеров;

Для расчета момента инерции необходимо знать габариты всех элементов, которые будут вычислены заранее. В список этих элементов входят:

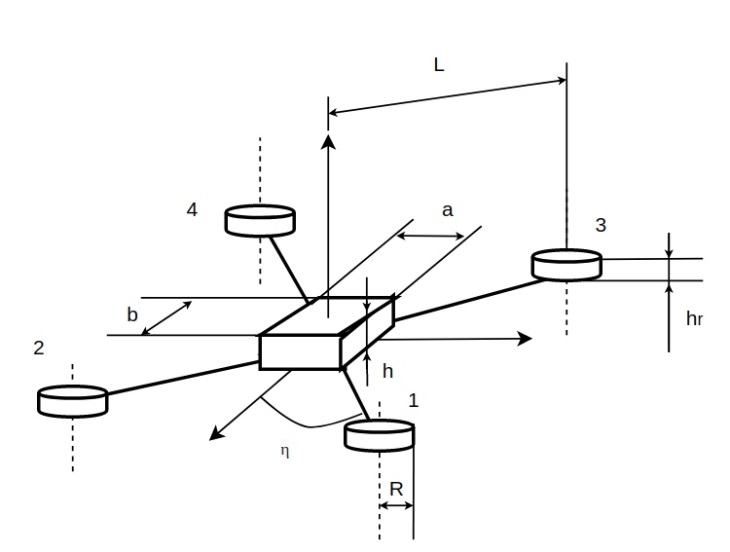
а – ширина корпуса;

b – длина корпуса;

h – высота корпуса;

R – радиус винта;

hc – высота винта;



## **Элементы коптера и их алгоритм решения задач**

Основные элементы квадрокоптера (рис.2) подключены между собой следующим образом:

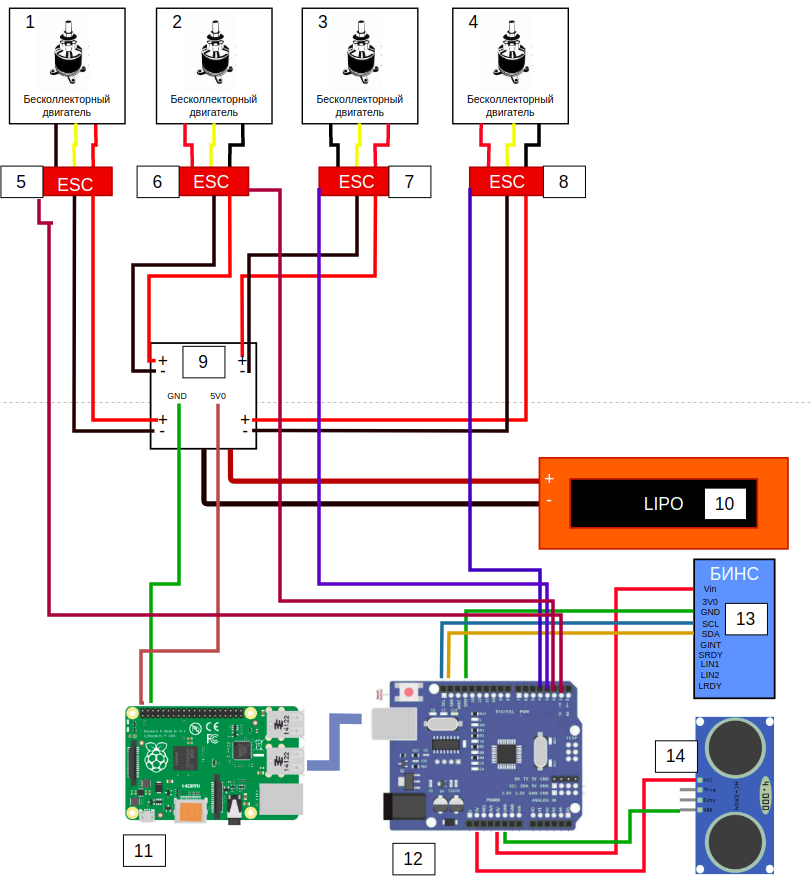


Рисунок 2

Бесколлекторные двигатели(1,2,3,4) соединены трехфазно с контроллерами бесколлекторных двигателей (5,6,7,8), которые соединены в распределительной плате (9). Распределительная плата запитана аккумулятором (10), напряжение (11-12 В) с блока питания переходит на контроллеры двигателей (5,6,7,8). Один из контроллеров питания преобразует входное напряжение с аккумулятора в выходное 5 В и запитывает Raspberry pi (11). Raspberry pi через COM port подключена к плате Arduino, к которой подключены БИНС (13), ультразвуковой дальномер (14) и провода управления ESC (5,6,7,8).

### **Arduino**

Каждый летательный аппарат обладает определенным списком значений параметров, который называется вектором состояния. Вектор состояния описывает состояние системы в фиксированный момент времени. Для определения вектора состояния квадрокоптера используется отдельная плата на базе Arduino, к которому подключены все необходимые элементы, которые будут описаны ниже. Вектор состояния передается на основную плату в составе сообщения, в котором находится вектор информации.

Вектор информации имеет следующие элементы:

|  |  |
| --- | --- |
|  | (1.1) |

D (см)-значение дальномера; X,Y,Z (м) – проекции перемещения квадрокоптера относительно стартовой системы координат; Vx, Vy, Vz (м/с) – скорость квадрокоптера в системе координат, связанной со стартом; Ψ(град.)– курс;ϕ θ (град.) – тангаж; ƴ (град.) – крен; ωx, ωy, ωz (град./сек.) – угловая скорость относительно бортовой системы координат.

На выход с Arduino идет вектор информации приведенный выше.

На вход в Arduino с основной платы идут следующие элементы:

1. Массив двигателей
2. Высота h (метры)
3. Географическая широта положения 

### **Ультразвуковой дальномер**

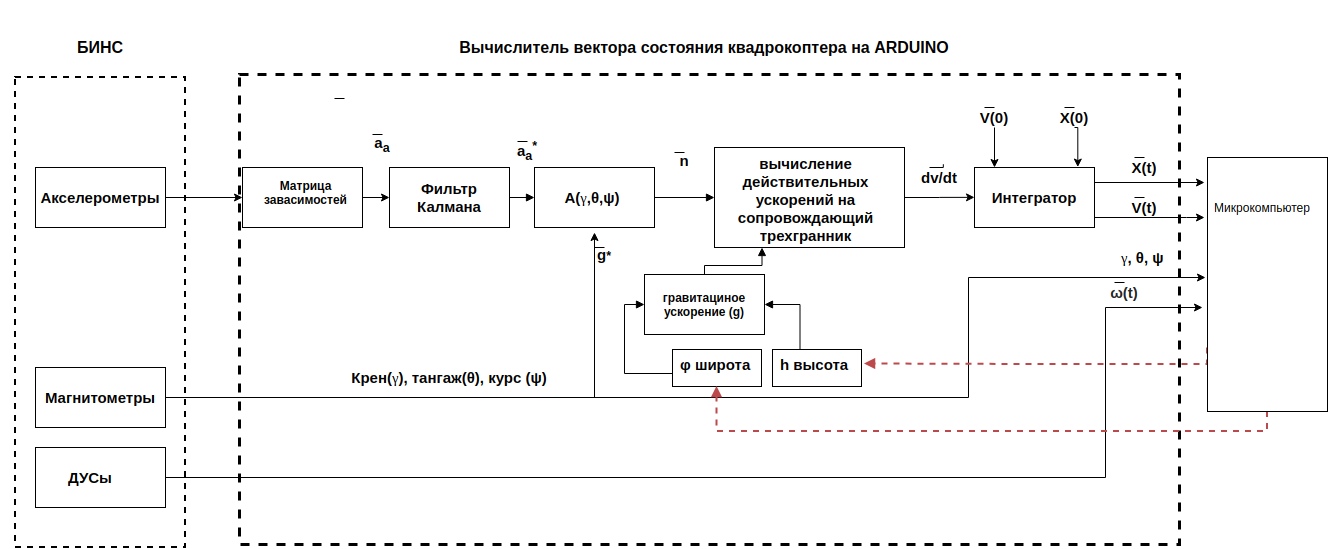
Для одного из вариантов навигации в пространстве можно использовать дальномер, в нашем случае используется ультразвуковой дальномер. Действие ультразвукового дальномера основано на принципе эхолокации. Он излучает звуковые импульсы в пространство и принимает отражённый от препятствия сигнал. По времени распространения звуковой волны к препятствию и обратно определяется расстояние до объекта.

Запуск звуковой волны начинается с подачи положительного импульса длительностью не менее 10 микросекунд в канал TRIG дальномера. Как только импульс заканчивается, дальномер излучает в пространство перед собой пачку звуковых импульсов с частотой 40 кГц. В это время запускается алгоритм определения времени задержки отражённого сигнала, а в канале ECHO дальномера появляется логическая единица. Как только датчик улавливает отражённый сигнал, на выводе ECHO появляется логический ноль. По длительности этого сигнала определяется расстояние до объекта.

Диапазон измерения расстояния дальномера - до 4 метров с разрешением 0,3 см. Угол наблюдения - 30 градусов, эффективный угол - 15 градусов.

### **БИНС**

Для корректной ориентации в пространстве необходимо использовать навигационные системы, которые определяют ориентацию ЛА в бортовой системе координат. Для получения вектора состояния квадрокоптера можно использовать БИНС, функциональная схема которого изображена на рис. ..



Алгоритм вычисления вектора состояния происходит в следующем порядке:

1. Считывание значений с ШИМ-сигналов акселерометров, ДУСов и магнитометров
2. С помощью умножения матрицы зависимостей между показаниями акселерометров на вектор линейных ускорений, получаем откалиброванные значения вектора линейных ускорений ;
3. С помощью умножения матрицы зависимостей между показаниями приборов ДУСов на вектор угловых скоростей, получаем откалиброванные значения вектора угловых скоростей
4. Используя фильтр Калмана, избавляемся от шумов в ускорениях .
5. С помощью магнитометра считываем 3 угла ориентации: рысканье , тангаж, крен
6. С помощью матрицы ориентации A получаем истинное ускорение, вычитая из вектора кажущегося ускорения вектор гравитационного ускорения.

Матрица А будет выглядеть следующим образом:

Где: Ψ(град.)– курс; θ (град.) – тангаж; ƴ (град.) – крен

1. Вектор кажущегося ускорения в осях сопровождающего трехгранника будет вычислен с помощью следующего выражения:

=A\*;

1. Используя векторное уравнение , производится описание проекций действительных ускорений на оси сопровождающего трехгранника.

Три скалярных уравнения для вычисления проекций текущих скоростей.

где  – длина радиуса-вектора точки расположения БИНС,  – текущая широта этой точки

, ,  – проекции кажущегося ускорения на сопровождающий трёхгранник.

Для определения величины ускорения силы тяжести и длины радиуса-вектора точки расположения используется следующее выражение:

 м/с2,

где:

 м/с2 – ускорение силы тяжести на экваторе;

 м – средний экваториальный радиус Земли;

 – широта места;

 – высота над уровнем моря, м.

 – длина радиус-вектора точки расположения блока БИНС на широте  и высоте  над уровнем моря, м, где:

 рад/с – угловая скорость вращения Земли;

 – знаменатель сжатия Земного эллипсоида;

 – эксцентриситет кривизны Земной поверхности.

Перемещения по координатным осям получаются интегрированием проекций скоростей

1. Вычисление проекций относительной линейной скорости
2. Вычисление проекций относительной линейной скорости
3. Передача на борт следующий параметров:

Значение дальномера, проекции перемещения квадрокоптера, скорость квадрокоптера , углов ориентации и угловых скоростей относительно бортовой системы координат.

1. Вычисление dt - разность времени между предыдущим циклом программы

А) dt=текущее время работы – ti-1

Б) запись текущего времени в переменную ti.

1. Считывание из COM-порта высоту h и широту положения .
2. Повторение цикла с пункта 1.

### **Контроллер бесколлекторных двигателей**

Для управления бесколлекторными двигателями необходимо использовать контроллеры скоростей вращения двигателей, которые получают напряжение (12В) на распределительной плате, подключенной к аккумулятору. Контроллер двигателей преобразует на выход ток, с помощью которого можно подключить в сеть контроллер Raspberry с помощью двух проводов (плюс и минус). Третий провод является проводом управления, по которому передается ШИМ сигнал с контроллера Arduino. Контроллеры бесколлекторных двигателей преобразуют полученный сигнал и подают напряжение на двигатели.

Для корректного управления двигателями необходимо откалибровать двигатели. Для калибровки необходимо подать на вход максимальное значение сигнала (2300) и после небольшой паузы подать минимальное значение сигнала (800).

### **Raspberry**

Поскольку основной алгоритм автоматического управления и алгоритмы определенных задач требует большего количества ресурсов чем может предоставить Arduino, требуется использовать более мощный процессор, для этого будем использовать микрокомпьютер raspberry pi. Для этого следует создать программное окружение, состоящее из следующих элементов:

1. Установка операционной системы Raspbian
2. Установка VNC server
3. Открытие портов VNC
4. Установка программы Arduino
5. Установка языка программирования Python 3

Как было сказано ранее, взаимосвязь с Arduino осуществляется через COM-port, который в свою очередь несет информацию в битовых строках

Алгоритм считывания информации и передачи на плату Arduino вектора, состоящего из значений угловых скоростей двигателей, выглядит следующим образом:

1. Идентификация COM-порта по имени
2. Создание объекта типа *“Пользователь”*, в котором хранятся все необходимые поля и методы для решения задач пилотирования, улучшения навигации, поиск объекта и других поставленных задач.
3. Считывание из порта и перевод и массив значений сообщения с переферийных устройств (в нашем случае Arduino).
4. Выполнение пользовательского основного метода. В зависимости от его реализации, на выход будет получен массив мощностей двигателей
5. Перевод массив значений в сообщение для отправки в COM-порт
6. Запись в COM-порт
7. Вычисление времени выполнения цикла
8. Начало нового цикла с пункта 3

## **Удаленное управление на компьютере через Wi-Fi**

Для дистанционного управления квадрокоптером с возможностью получать информацию о векторе состояния и остальных устройств на raspberry pi будет использована программа VNC Viewer. Подключение состоит из следующих этапов:

1. Включение VNC-сервера на контроллере raspberry
2. Определение IP-адреса Raspberry
3. Подключение по IP-адресу к raspberry через приложение VNC Viewer на ПК

После этого через VNC можно подключаться к рабочему столу Raspberry Pi

Возможности дистанционного подключения к микроконтроллеру по VNC-серверу:

1. Управление коптером на расстоянии
2. Возможность включения ручного управления коптером через любой персональный копьютер
3. Наблюдение и анализ данных сразу же после их получения

## **Заключение**

Подведем итоги и распишем достоинства и недостатки данной схемы подключения

Достоинства системы:

+ увеличение количества модулей позволит более тщательно подобрать элементы для определенной задачи

+разнесение задач по разным модулям позволит модернизировать определенные компоненты без зависимости от всей системы

+быстрая замена определенных модулей

+можно создать определенные маневры, которые не могут осуществиться на обычном автопилоте, так как появилась возможность управлять каждым двигателем в отдельности

+ возможность дублирования систем для повышения точности и надежности

+ решение различных задач на микрокомпьютере, расположенном на борту квадрокоптера

Недостатками данной схемы являются:

- увеличение массы и габаритов за счёт увеличения соединительных устройств между компонентами

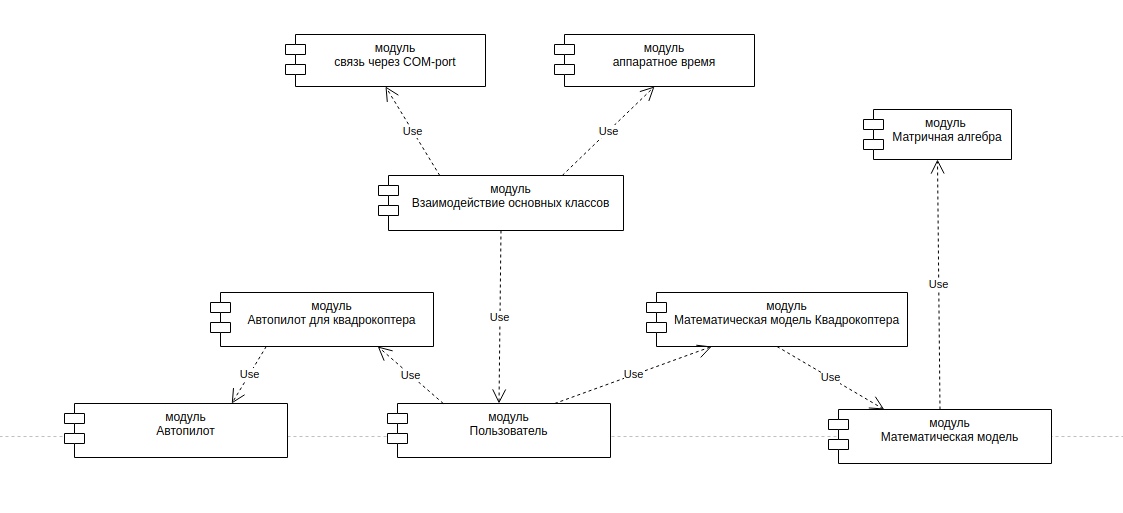
-время на разработку и тестирование программного обеспечения

- увеличение энергозатрат

# **Программное обеспечение**

## **Структура программного обеспечения**

На борту летательного аппарата, в нашем случае расположен микрокомпьютер Rasspbery pi3, на котором можно создать нужное для задачи программное обеспечение (ПО). Данная структура будет выглядеть следующим образом



Основной и отвечающий за связь с дополнительными контролерами является модуль *“Взаимодействие основных классов”*, который подключает для себя: модуль *“Cвязь через COM-port*”. В нем хранится класс, с помощью функций которого, осуществляется связь с остальными модулями. За предоставление системного времени следует подключить модуль *“Аппаратное время”*, отвечающий за определенную систему времени. Оба этих модуля являются системными в Python 3.

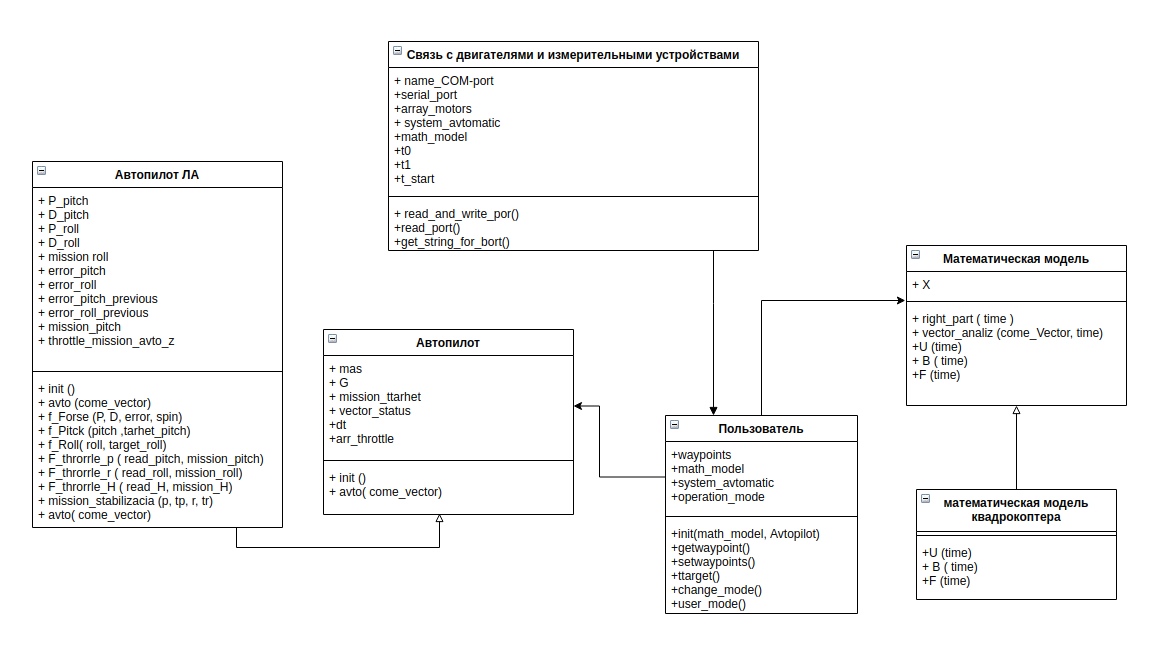
Третий модуль, который подключен к основному это *“Пользователь”*. В нем хранятся классы для выполнения всех поставленных миссий, заложенным в класс, а также и подключены дополнительные модули, которыми необходимы для класс пользователя.

Класс *“Пользователь”-* индивидуальный класс, позволяющий подключить любой нужный элемент в зависимости от поставленных целей. В нашем случае, для поставленных задач подключены модуль “ Автопилот для квадрокоптера” и модуль” Математическая модель Квадрокоптера”. При изменении конструкции, на которой поставлен микрокомпьютер, можно изменить модуль автопилота на другой, например на “Автопилот Машины”. Аналогично можно сделать и с математической моделью аппарата.

Также гибкость такой структуры можно наблюдать и на диаграмме классов, описанных ниже.

## **Диаграмма классов, используемых в ПО на Raspberry**

Для того, чтобы программное обеспечение позволяло универсально переключать необходимые функции и переменные, без переписывания большого количество одинокого кода для разных вещей, необходимо использовать объектно-ориентированное программирование (ООП), диаграмму которого описана на рисунке …



### **Класс “Связь с двигателями и измерительными устройствами” и класс “Пользователь”**

Класс *“Связь с двигателями и измерительными устройствами”* является статичным и вызывается сразу же после запуска модуля *“Взаимодействие основных классов”*.Содержит в себе объект типа *“Пользователь”*, который, в свою очередь имеет:

1) Поля объектов абстрактных классов *“Автопилот”* и *“Математическая модель”*, *system\_avtomati*c и *math\_model* соответственно.

2) Переменную, содержащую целевые точки передвижения *waypoints.*

3) Поле *operation\_mode*, отвечающее за текущий режим работы.

4) Конструктор *init*, который при инициализации создает необходимые объекты и переменные.

5) Метод*, change\_mode,* вызываемый в каждый промежуток времени и выполняющий создание (в зависимости от условий) и запись новых точек, а также переключение режима в зависимости от значения *operation\_mode.*

6) Метод *user\_mode,* активируется при определенном значении *operation\_mode* в методе *ttargets*.

7)Пользовательский метод *ttarget,*который выполняется в *change\_mode,* для решения иных задач, не связанных с пилотированием ( например, алгоритм обработки изображения с камер).

8) функции задания и считывания целевой точки движения *getwaypoint* и *setwaypoint*.

Именно класс *“Связь с двигателями и измерительными устройствами”* при активации производит связь между периферийными устройствами (в нашем случае Arduino) с помощью порта, объявленного в переменной *com\_port*. И передает команду на двигатели через функцию *read\_port* преобразованные значения вектора двигателей array\_motors. Вектор двигателей преобразуется в строчку через процедуру *get\_string\_for\_bor*t и отправляется на плату Arduino через *read\_and\_write\_port*.

А также этот статичный класс измеряет время dt между циклом работы решения всех задач.

### **Класс “*Математическая модель*”**

Поскольку в некоторых приборах существуют свои недостатки, приходится использовать дублирующие элементы, которые избавляют от данных нежелательных эффектов.

Так, для определения местоположения используется БИНС, записывающий в вектор ускорений колебания от двигателей, что нежелательно. Необходимо знать, когда именно двигатель работает, чтобы обрезать ошибочные скачки ускорений, внесенные вибрацией. Так же желательно использовать второй способ для навигации, позволяющий определять состояние в случае поломки одного из приборов. С этой задачей справляется общий класс *“Математическая модель”*.

Класс предназначен для повышения точности и определения вектора состояния в зависимости от посланных (в текущее время) значений скоростей для двигателей. Рассмотрим общий алгоритм данного класса.

В методе *rightpart* решаются правые части дифференциальных уравнений. Является абстрактным. После решения всех элементов данной функции, полученные значения интегрируются с предыдущими значениями вектора состояния:

Абстрактный метод *vector\_analiz* используется для обработки и анализа всех полученных данных, переопределяется потомками.

### **Класс “*Математическая модель*** ***квадрокоптера*”**

Класс предназначен для повышения точности навигации

Модель описывает дифференциальные уравнения системы. Рекомендуется упрощать модель системы управления с целью соответствия ограничениям в реальном времени, предъявляемых к контуру встроенной системы. Следовательно, можем пренебречь моментом вращения и силой действующей на винт, а коэффициенты тяги и лобового сопротивления примем постоянными. Система может быть переписана в форме пространства состояния , где U вектор входных сигналов и X вектор состояния который задается следующим образом: берем вектор состояния (1.1) и оставляем элементы, которые необходимо использовать для моделирования полета.

Получим следующий вектор состояния квадрокоптера:

Поскольку наш объект является ЛА, то будет использоваться вектор управления, составленный из 4 сигналов управления:

1) управление в вертикальном канале.

2) управление в канале рысканья

3) управление в канале тангажа

4) управление в канале крена

Система из данных входных сигналов будут иметь вид:

Функция f(**X**,U) будем иметь вид столбчатого вектора из N элементов, где N – количество элементов в векторе состояния **Х**.

Где:

m – масса, вводимая в расчетную модель самим пользователем

### **Класс “*Автопилот*”**

Класс “*Автопилот*” является абстрактным и предком классов конкретных автопилотов. Общими элементами для всех автопилотов являются:

1. Mass – масса пилотируемого объекта. Задается и вычисляется через математическую модель или считывается с датчиков.
2. G – гравитационная постоянная. Задается или вычисляется необходимыми способами, а также может быть зафиксирована через интернет.
3. Mission\_target – вектор доступных режимов пилотирования в автоматическом режиме (вектор индикаторов активации режима).
4. Vector\_status – вектор состояния управляемого объекта, полученного через пользователя
5. dt - разница во времени между текущим и предыдущим значением времени.
6. arr\_throttle - массив вычисленных параметров двигателей.
7. Init() – конструктор класс, инициализация всех необходимых параметров.
8. Avto() – процедура, в которой происходит решение поставленной задачи, вычисление необходимых значений *arr\_throttle* в зависимости от mission\_target.

Класс “*Автопилот* ЛА” является потомком класса “Автопилот”. Наследует и переопределяет методы. Далее о работе автопилота в следующей главе.

# **Автопилот**

## **Основные параметры и принципы работы**

Автопилот – это программно-аппаратная система, которая имеет возможность вести транспортное средство по заданному маршруту. В нашем случае будем рассматривать автопилот на основе кавдрокоптера.

Суть управления параметрами полета квадокоптера заключается в поддержании такого углового положения коптера, при котором вектор скорости имеет направление и величину, обеспечивающие движение (полет) квадрокоптера по заданной траектории (а также обеспечивает «зависание»).

Все моменты для формирования вращательного движения коптера вокруг его центра масс и силы для формирования поступательного перемещения формируются за счет установки заданных скоростей вращения роторов каждого из двигателей, а величины моментов сил – за счет взаимных соотношений между этими тяговыми силами.

### **Расчет основных параметров**

Для расчета компенсирующей скорости вращения двигателя ***Force*** в данном случае используется регулятор типа PD. Данная функция аналогична для расчета реакции на отклонение от заданного значения для всех каналов управления, а именно:

* Для расчета компенсирующей скорости двигателей для набора/уменьшения высоты;
* Для расчета компенсирующей скорости двигателей в канале тангажа;
* Для расчета компенсирующей скорости двигателей в канале рысканья;

В данную функцию поступают значения *P, D, error, spin,* . Далее представлена формула, по которой рассчитывается ответная реакция на разницу между заданным и текущем значением параметра.

*Force = P\*error + D\*spin*

Где:

*spin* - скорость изменения ошибки;

*spin* =;  
*error* – ошибка, разница между заданным значением *target* и текущим значением;

– значение ошибки на предыдущем шаге;

*dt* – разница между текущим значением времени и предыдущем;

P — коэффициент пропорциональности, чем он больше, тем сильнее будет реакция на отклонение от заданного значения;

D — настраиваемый коэффициент, чем он больше, тем сильнее останавливающее усилие.

Необходимо добавить, что значения *P* и *D* для каждого канала управления, через PD регулятор, разные

Также, можно рассчитать значение двигателей при зависании на постоянной высоте, для использования в следующих операциях. Для этого, сумма сил от всех двигателей должна равняться силе тяжести квадрокоптера:

;

i=1..4, - количество двигателей;

Приняты следующие допущения =K\* *Throttle\_mission\_avto\_z*;

где *Throttle\_mission\_avto\_z* – средняя скорость вращения всех двигателей для удержания постоянной высоты.

K – Коэффициент пропорциональности;

Из данного равенства можно рассчитать значение средней угловой скорости винта, при условии зависания на одной высоте.

*Throttle\_mission\_avto\_z* =

## **Алгоритм работы автопилота квадрокоптера**

Алгоритм работы автопилота квадрокоптера заключается в следующем:

1. Проверка в методе  *change\_mode* (который относится к классу *Пользователь*) переведено ли управление на автопилотирование. Если *operation\_mode ==avto*, выполняется следующий пункт;
2. Запускается метод *avto*, в который поступает вектор текущего состояния квадрокоптера
3. В методе *avto* проверяются условия на активность каждого режима из вектора mission\_target. То есть если mission\_targeti =1, то выполняется режим, имеющий номер i.

i=0..N-1, где N – количество режимов в автопиолоте. Если mission\_targeti =0, то значит, что автопилот не переведен в данный режим.

1. После выполнения метода, принадлежащему i-му индикатору активации режима, полученные значения записываются в вектор *throttle*, который хранит в себе вычисленные значения двигателей.

Далее рассмотрим все возможные режимы автопилота, в которые можно переводить квадрокоптер под управлением микрокопьютера.

## **Режимы автопилота квадрокоптера**

### **ARMING**

Данный режим производит включение всех основных модулей и калибровку двигателей после включения квадрокоптера через аккумулятор или другие периферийные устройства.

Алгорит калибровки был вынесен отдельно от остальных и находится на плате Arduino. Данное преобразование дает возможность на микрокомпьютере Raspberry в данный момент решать другие задачи.

Алгоритм калибровки двигателей заключается в следующем:

1. Идентификация двигателей
2. Пауза в 2 секунды
3. Подача максимального напряжения на двигатели
4. Пауза в 4 секунды
5. Подача минимального напряжения на двигатель

### **Удержание горизонтального положения**

Режим удержания положения (режим стабилизации) заключается в поддержании заданного углов ориентации квадрокоптера.

Поддержание заданных углов осуществляется в методе *mission\_stabilization* следующим образом

* Задание требуемых значений углов по крену и тангажу, соответствующих горизонтальному значению.

Mission\_roll=0

Mission\_pitch=0

* Вычисление в методах F\_throttle\_P и F\_throttle\_R необходимую скорость вращения винтов *Force\_p* и *Force\_r*. Алгоритм расчета *Force* идентичен в обоих случаях.
* Вычисление окончательных значений на каждый из двигателей в выходной вектор *throttle*.

*throttle[3]=throttle\_mission\_avto\_z+throttle\_p+throttle\_r  
throttle[2]=throttle\_mission\_avto\_z+throttle\_p-throttle\_r  
throttle[1]=hrottle\_mission\_avto\_z-throttle\_p+throttle\_r  
throttle[0]= throttle\_mission\_avto\_z-throttle\_p-throttle\_r*

где:

*throttle[i]* окончательное значение i-ого двигателя.

* Передача вектора вычисленных значений двигателей в функцию *avto*

### **Взлет на заданную высоту**

Взлет на заданную высоту производится при вычислении *Force\_h* аналогичным способом, как и функция *Force*. Во входные параметры поступают заданное, текущее значения высоты, а также коэффициенты , для PD регулятора. Далее идет расчет *Force\_p* и *Force\_r*. После этого вычислется вектор *throttle*.

*throttle[3]=Force\_h+throttle\_p+throttle\_r  
throttle[2]=Force\_h+throttle\_p-throttle\_r  
throttle[1]=Force\_h-throttle\_p+throttle\_r  
throttle[0]=Force\_h-throttle\_p-throttle\_r*

Данный метод схож с режимом удержания горизонтального положения, но имеет следующие особенности:

* Требует дополнительного вычисления вычисления для корректировки по высоте на каждом шаге
* Имеет возможность более точного автозависания на заданной высоте
* Возможность изменять скорость через PD регулятор

### **Движение вперед**

### **Поворот на заданный курс**

### **Посадка**