

# Дипломна работа

---

КАТЕДРА СИСТЕМИ И УПРАВЛЕНИЕ

---

НА ТЕМА:

## Платформа за безпилотен летателен апарат с четири ротора

*Ръководител:*

*Автор:* гл. ас. д-р Александър

РАФАЕЛ КАЛЪЧЕВ, Хотмар

IV курс, № 011217071 *Ръководител на кат. СУ:*

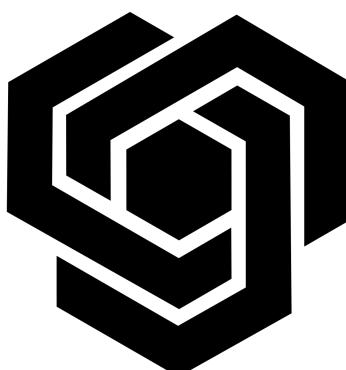
доц. д-р Теофана Пулева

ТЕХНИЧЕСКИ УНИВЕРСИТЕТ

София

ФАКУЛТЕТ АВТОМАТИКА

АИУТ



ФЕВРУАРИ, 2022

София

## Съдържание

<b>1 Въведение</b>	<b>6</b>
<b>2 Използвана среда и инструменти за разработка на платформата</b>	<b>7</b>
2.1 Среда за разработка на софтуера . . . . .	7
2.2 Инструменти . . . . .	8
2.2.1 Логически анализатор и генератор на (дигитални) сигнали . . . . .	8
2.2.2 Балансъор за витла . . . . .	10
2.2.3 3D Принтер . . . . .	10
<b>3 Използван хардуер</b>	<b>12</b>
3.1 Микроконтролер . . . . .	12
3.1.1 Характеристики . . . . .	12
3.1.2 Комплект 32F429IDISCOVERY . . . . .	14
3.2 Жироскоп и акселерометър . . . . .	14
3.3 Магнитометър . . . . .	15
3.4 Електронен контролер на скоростта за безчеткови постояннотокови мотори (ESC) . . . . .	16
3.5 Безчеткови постояннотокови мотори (BLDC) . . . . .	17
3.6 Li-Po батерия и зарядно . . . . .	18
3.7 Джойстик и приемник . . . . .	18
<b>4 Архитектура на системата</b>	<b>20</b>

4.1 Конструкция на платформата . . . . .	20
4.2 Платформа с четири ротора . . . . .	20
4.3 Платформа за управление на ъгъл на завъртане . . . . .	21
4.4 Софтуерна част . . . . .	22
4.4.1 Модул stdperiph . . . . .	22
4.4.2 Модул CMSIS . . . . .	23
4.4.3 Модул math . . . . .	23
4.4.4 Модул control . . . . .	23
4.4.5 Модул специфични за проекта функции . . . . .	23
<b>5 Изграждане на системата и решени проблеми</b> . . . . .	<b>24</b>
5.1 Изграждане на работната среда . . . . .	24
5.1.1 Система за изграждане (Make) . . . . .	24
5.1.2 Компилатор . . . . .	24
5.1.3 Програматор . . . . .	25
5.1.4 Дебъгер . . . . .	25
5.1.5 Получаване и обработка на данни . . . . .	25
5.1.6 Линкерен скрипт . . . . .	25
5.2 Конфигурация на микроконтролера . . . . .	26
5.2.1 Стартране . . . . .	26
5.2.2 Системна инициализация . . . . .	26

5.2.3 Инициализация на периферията . . . . .	27
5.3 Моделиране . . . . .	29
5.3.1 Моделиране на тяга от витло и мотор . . . . .	29
5.3.2 Моделиране на платформа за управление на ъгъл на завъртане	29
5.3.3 Моделиране на платформа с 4 ротора . . . . .	31
5.4 Компенсация на сензорни отмествания . . . . .	33
5.4.1 Компенсация на жироскопен дрейф . . . . .	33
5.4.2 Компенсация на магнитни отмествания от околната среда . . .	35
5.4.3 Баланс на витлата . . . . .	36
5.5 Калманов филтър . . . . .	37
5.6 Експериментални задачи . . . . .	41
5.6.1 Снемане на статичните характеристики за тягата на мотор с витло	41
5.6.2 Управление на ъгъл на завъртане . . . . .	42
<b>6 Предложения за надграждане</b>	<b>44</b>
<b>7 Литература</b>	<b>45</b>
<b>A Системна конфигурация</b>	<b>47</b>
<b>Б Процедура за изчисляване на магнитометърна компенсация</b>	<b>48</b>

## Списък на фигурите

1	Логически анализатор SQ50 . . . . .	9
2	Прозорец Scana Studio . . . . .	9
3	Балансър за витла . . . . .	10
4	3D принтер Creality Ender 3 V2 . . . . .	11
5	Прозорец на Ultimaker Cura . . . . .	11
6	Прозорец на FreeCad . . . . .	12
7	Комплект 32F429IDISCOVERY . . . . .	13
8	Наименования на изведените пинове през използвания пакет . . . . .	14
9	<i>LSM6DS33</i> Ориентация на осите и наименованията на пиновете в пакета . . . . .	15
10	Магнитометър – оси на измерване и наименования на пиновете . . . . .	16
11	ESC с анотация на изходите и входовете . . . . .	17
12	Безчетков постояннотоков мотор AX-4008D KV620 . . . . .	17
13	Li-Po батерия и зарядно . . . . .	18
14	Джойстик Spektrum Dx6i . . . . .	19
15	Приемник AR6200 . . . . .	19
16	Диаграма за свързване на изходните пинове на приемника . . . . .	20
17	Конструкция на платформата с четири ротора . . . . .	21
18	Конструкция на платформата за управление на ъгъл . . . . .	22
19	Основни събития касаещи управлението . . . . .	22

20	Честоти на системните шини и часовници след конфигурация . . . . .	27
21	Опростен модел на витло и мотор . . . . .	29
22	Диаграма на платформа за управление на ъгъл на завъртане . . . . .	30
23	Основни типове на конфигурация . . . . .	31
24	Сили, моменти и отправни системи на квадракоптера . . . . .	32
25	SIMULINK схема на динамиката на платформа с 4 ротора. . . . .	33
26	Жироскопен дрейф спрямо температура и полиномиална компенсация спрямо температура . . . . .	34
27	Компенсирани данни получени от магнитометър . . . . .	35
28	Небалансирано витло . . . . .	37
29	Балансирано витло . . . . .	37
30	Проверка на Калман филтъра за ос $\phi$ . . . . .	40
31	Проверка на Калман филтъра за ос $\phi$ , приближен . . . . .	41
32	Управляващ сигнал / тяга . . . . .	42
33	Упростена сема на управлението . . . . .	42
34	Схема ПИД регулатора . . . . .	43
35	Преходен процес на затворената система . . . . .	43

## 1 Въведение

През последните години се наблюдава засилване на интереса към създаване и управление на т.нар. квадрокоптери или четирироторни хеликоптери. Квадрокоптерът представлява малък до среден по размер безпилотен летателен апарат (UAV), който има четири симетрично разположени ротора, обикновено прикрепени в краищата на рамената на платформата.

В сравнение с други подобни платформи четирироторният апарат притежава здравина, механична простота, стабилност и относително ниска цена, което го прави обект на внимание както по военни, така и по търговски причини.

По-голямата част от четирироторните хеликоптери са конструирани от компоненти за играчки с дистанционно управление и в резултат на това тези устройства нямат необходимата надеждност и производителност, за да бъдат практически експериментални платформи.

Поради предимствата пред обикновените летателни апарати, потенциала по отношение употребата в открита среда и поради сложната динамика управлението на четирироторен летателен апарат е фундаментално труден и интересен проблем.

Този труд се концентрира върху цялостното изграждане на система за управление като предоставя за пример създаване на „Безпилотна платформа за летателен апарат с четири ротора“. Ще се наблюде върху направата на софтуер из основи за безпилотния летателен апарат. По този начин ще бъде демонстрирано как може да се изгради основа за софтуер за управление на непознат, иновативен микроконтролер, за който не съществуват библиотеки.

С цел да се подобри разбирането за работата на софтуера на системата за управление, ще се елиминира интеграцията с MATLAB (за управление) и използването на специфични много популярни модули, за които има налични множество библиотеки. Ще бъде разгледан начин за инициализиране и управление на периферията, както методи за обработка на данните, постъпващи от периферията за сформиране на управляващи въздействия.

Избраната система е многомерна и има състояния, които не могат да бъдат измервани директно. Този труд ще демонстрира изграждането на наблюдателя на състоянията на системата, както и неговата имплементация като част от алгоритъ-

ма за управление.

Този труд няма да разглежда изграждането на система за управление с помощта на Операционна система за реално време. Изграждането на ОС за реално време ще бъде плот на отделен бъдещ труд. Работата в реално време ще бъде осигурена от софтуера на контролера, но тя няма да бъде разпределена на отделни „задачи“, а ще се управлява от регулярните прекъсвания на таймера, съпътствани от приста логика и функции, имплементирани по начин, който ще гарантира изпълнение за определеното време.

Като част от този труд е изградена платформа за управление на ъгъл на завъртане на рамо с два ротора. Тази задача е идентична с една от подзадачите за управление на четирироторна платформа.

## 2 Използвана среда и инструменти за разработка на платформата

### 2.1 Среда за разработка на софтуера

Изградената среда за разработка на софтуера е конфигурируема и поддръжка базата микроконтролери от семейство *STM32M4xxx*. като основа е използвана автоматичната система за изграждане *GNU make*, която позволява насочена обработка на файловете, изграждащи софтуера и документацията, с цел намаляване времето за обработка. *GNU make* свързва всички елементи от средата и последователно изпълнява само нужните команди с цел намаляване на използваните ресурси. Интерфейсът е команден, което позволява допълнителни нива на автоматизиране на процесите по разработка.

За компилация е използван свободният компилатор на *GNU GCC (GNU Compiler Collection) ARM NON-EABI (No Embedded-Application Binary Interface)*. Тази разновидност на компилатора е неспецифична към целева операционна система, което е нужно, тъй като разработваният софтуер няма да работи под операционна система. Тази разновидност на компилатора също е неспецифична към производителя на процесора. *GCC ARM NON-EABI* е колекция от свързани инструменти, за разработка на софтуер за системи с ядро *ARM*. Състои се от компилатор на езика C, асемблатор,

линкер, инструменти за преглед и конверсия между стандартни формати двоични файлове.

За връзка с контролера е използван командният пакет за *ST-LINK* на *STMicroelectronics*. Пакетът предоставя команди за връзка с програматора *ST-LINK V2*, чрез който се програмира микроконтролерът. Пакетът се използва също за управление на порта за дебъг, през който се осъществява дебъг комуникацията.

Като дебъгер е използван свободният дебъгер на *GNU GDB*. Той може да се използва както за локално така и за отдалечно дебъгване. Тъй като микроконтролерът не е локален, използваме командния пакет за *ST-LINK* да конфигурираме порт за дебъг, към който се свързваме чрез *GDB*.

За писането на софтуера е използван терминалният текстов редактор *VIM* поради удобния си команден интерфейс. За генериране на всички софтуерни тагове, които *VIM* ще използва за подпомагане на процеса за разработка, е използван софтуерът *ctags*, който обработва релевантните файлове и поддържа опростена база данни за всички идентификатори, имена и тагове, които *VIM* изпозва спрямо контекста.

Основата за средата за разработка съм публикувал в публично хранилище на GitHub [7].

## 2.2 Инструменти

За хардуерната част са използвани стандартни инструменти, като отвертки, шестограми, винтоверт, нивелир, макетен нож, рулетка, шублер, трион, шкурка, поясник, пистолет за топъл силикон и т.н.

### 2.2.1 Логически анализатор и генератор на (дигитални) сигнали

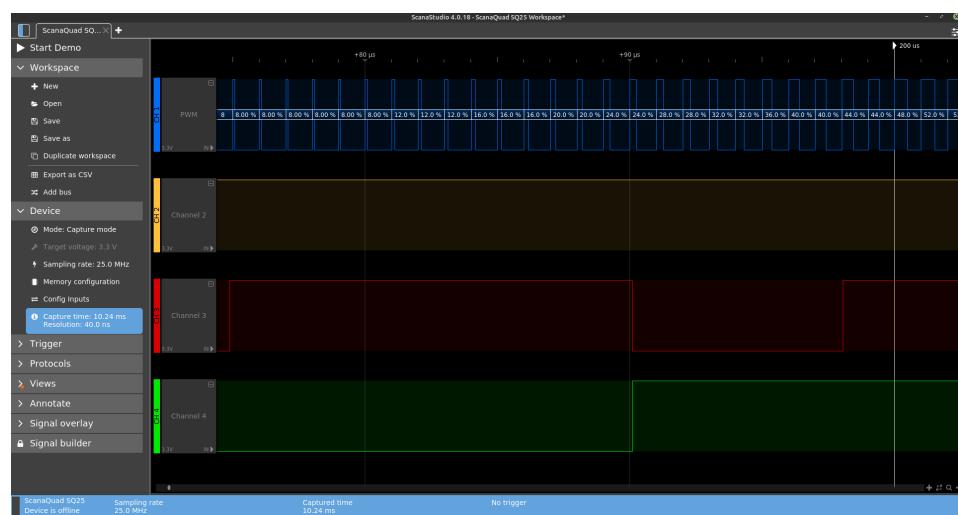
Тъй-като този труд има за цел управление и работа с хардуер и създаване на хардуерни драйвери, е нужен начин за анализ и диагностика на крайната комуникация. Затова е избран модулът *SQ50 - logic analyzer* от семейство *SQ series (ScanaQuad)* на *IKALOGIC S.A.S.* ([Фигура 1](#)). Модулът разполага с 4 канала с възможност за използване за вход и/или изход, максималана честота на дискретизация  $50MHz$ , максимална честота на измерим/генерируем дигитален сигнал  $12MHz$ , входно напрежение  $\pm 5V$ ,

входен импеданс  $1M\Omega/4pF$ , конфигурируем изходен драйвър (Push-Pull/open-drain), и максималнен изходен ток  $20mA$  [6].



Фигура 1: Логически анализатор SQ50

*SQ50 - logic analyzer* е използван със софтуерния продукт *ScanaStudio* (Фигура 2) на *IKALOGIC S.A.S.*. *ScanaStudio* е достъпен за платформите *Windows 7, 8 и 10, LINUX UBUNTU 16.04 и по-нови (x64), MACOS 10.9 и по-нови*. Софтуерният продукт позволява възможности за конфигурация на хардуера от семейство *SQ series (ScanaQuad)*, както и четене, експорт и представяне на данните, постъпили от модула. Към продукта *IKALOGIC S.A.S.* предоставят набор от готови разширения, които са свободно достъпни и позволяват: параметрично генериране на основни типове сигнали (ШИМ, честотна модулация, I2C), декодиране за сигнали (USART, I2C, I2S, SPI, PWM, SWD, 1-Wire и т.н.), конфигурируем източник на задействане (trigger) при определени условия – получена специфична последователност, логическа промяна, специфичен/случаен валиден кадър от комуникационен протокол и т.н.



Фигура 2: Прозорец Scana Studio

### 2.2.2 Балансър за витла

Използван е балансър за витла ([Фигура 3](#)) с цел балансиране на витлата. Балансиорът сам по себе си е тънка прецизно изработена права ос с тънка резба, в комбинация с две прецизно изработени конични гайки с успоредна на оста задна част, за да фиксират витлото успоредно спрямо оста, както и да центрират средата на отвора.

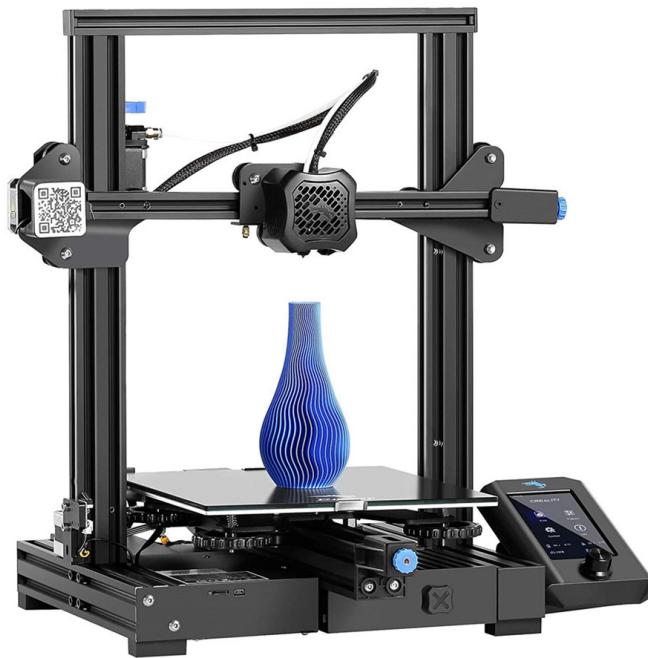


Фигура 3: Балансър за витла

### 2.2.3 3D Принтер

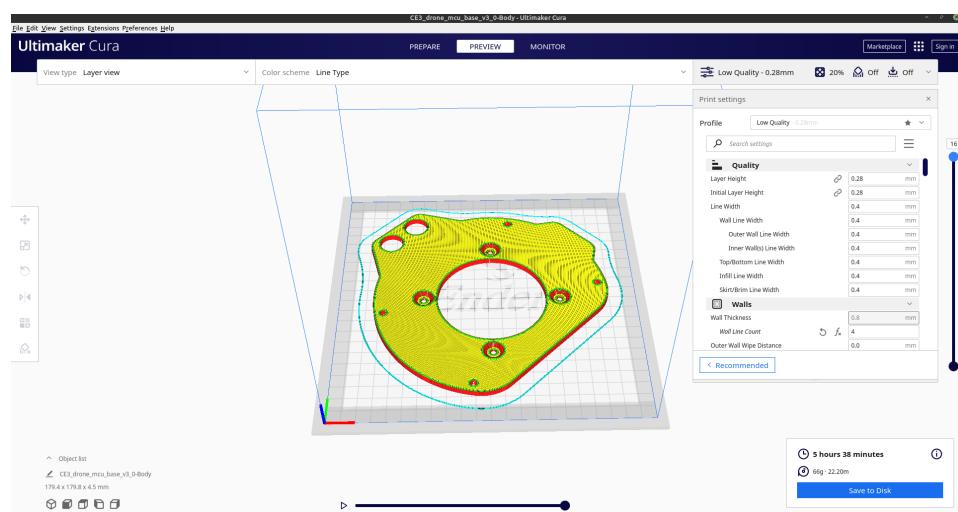
За изработване на платформата е използван 3D принтера *Ender 3 V2* ([Фигура 4](#)) на *Creatly*. Неговата технология на работа е FFF (Fused Filament Fabrication – производство чрез стопени нишки), която е форма на адитивно производство. Принтерът е оборудван с  $0.4mm$  дюза, легло със загравяне и стъклена повърхност и има работен обем  $220x220x250mm$ , и диаметър на нишката  $1.75mm$  [[11](#)].

За целите на този труд е използвана нишка от ECO PLA (рециклирана полимлечна киселина), произведена от *3D Jake* с диаметър  $1.75mm$ , температура на принтиране  $195 \rightarrow 215^{\circ}C$ , максимална работна температура  $60^{\circ}C$ , якост на опън  $70MPa$ ) и деформация при опън  $5\%$  [[1](#)].



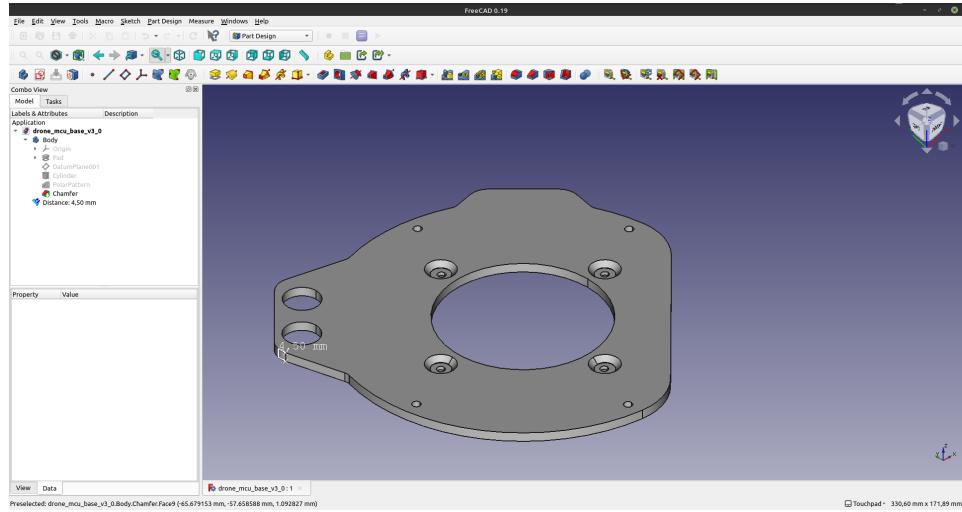
Фигура 4: 3D принтер Creality Ender 3 V2

За генериране на G-code, който ще бъде изпълнен от принтера, е използван софтуерният продукт *Ultimaker Cura* ([Фигура 5](#)) на *Ultimaker*. Продуктът е конфигуриран ръчно за устройството *Ender 3 V2*, тъй като не бе налично в базата данни с готови конфигурации. Софтуерът позволява „нарязване“ на триизмерни обекти на слоеве и конвертирането им в G-code програма за изпълнение от устройството, както и конфигуриране на множество параметри относно процеса на принтиране.



Фигура 5: Прозорец на Ultimaker Cura

За създаване на 3D модел е използван софтуерният продукт *FreeCad* **Фигура 6**, тъй като работи под *Linux* платформа и е добре документиран, свободен и безплатен софтуер.



Фигура 6: Прозорец на FreeCad

### 3 Използван хардуер

#### 3.1 Микроконтролер

Микроконтролерът, използван за проекта, е с ядро с архитектура *ARM Cortex-M4*, произведен от *STMicroelectronics*. Модел *stm32f429ZIT6U*.

##### 3.1.1 Характеристики

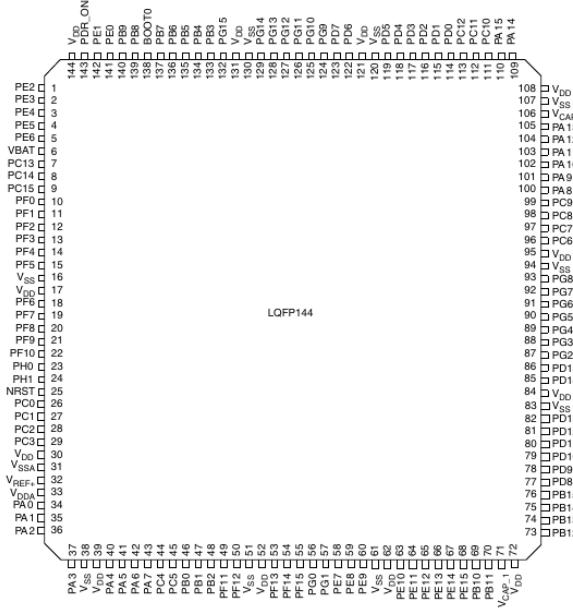
В следния списък са поместени основните характеристики на микроконтролера [18].

- Ядро: *32b Arm Cortex-M4* с FPU
- Максимална честота на процесора: 180MHz
- Флаш памет: 2048 Kbytes
- SRAM: Системна : 256 ( 112 + 16 + 64 + 64 ) Kbytes



Фигура 7: Комплект 32F429IDISCOVERY

- Таймери:
  - General Purpose: 10бр. (TIM2-5, TIM9-14)
  - Advanced control: 2бр. (TIM1, TIM8)
  - Basic: 2бр. (TIM6-7)
- Комуникационни интерфейси:
  - SPI/I2S : 6/2 (пълен дуплекс)
  - I2C: 3
  - USART/UART: 4/4
- GPIO: 114бр.
- Интерфейс за програмиране: *ST-LINK*
- Опаковка: LQFP144 (**Фигура 8**)



Фигура 8: Наименования на изведените пинове през използвания пакет

### 3.1.2 Комплект 32F429IDISCOVERY

Микроконтролерът е част от платка 32F429IDISCOVERY (Фигура 7) (комплект за оценка на функционалностите на *STMicroelectronics*). Комплектът предоставя множество функционалности, вградени в платката като 8MHz външен осцилатор, вграден ST-LINK програматор, LCD дисплей, 2 броя LED, както и удобна връзка към пиновете на микроконтролера в THT формат.

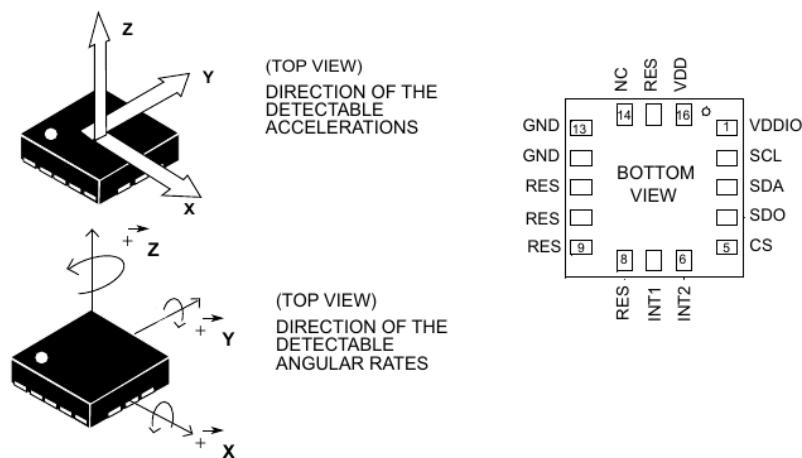
Важно е да се отбележи, че в комплекта 32F429IDISCOVERY има познат проблем с вградения ST-LINK програматор. Докато устройството не е свързано чрез USB, програматорът поддържа процесора в RESET през дебъгера. Проблемът се решава, чрез ъпдейт на фърмуера на дебъгера или декуплиране чрез физическо разединяване на каналите на дебъгера върху платката (CN4).

## 3.2 Жироскоп и акселерометър

Изпозван е чип LSM6DS33 (Фигура 9) [16], който е интегрирано пакетно решение, предоставящо 3D дигитален Жироскоп и 3D дигитален акселерометър. Чипът използва  $1.7 \rightarrow 3.6V$  захранване като в нужния ни решим консумира  $0.9mA$ . Чипът е част от сензорна платка, позволяваща директна връзка чрез протокол I<sup>2</sup>C (адрес:

0xD6).

За целите на настоящия проект е използвана следната конфигурация: директен достъп (без буфериране) до измерените стойности; обхват на акселерометъра  $\pm 8g$ ; обхват на жироскопа  $\pm 1000dps$ .

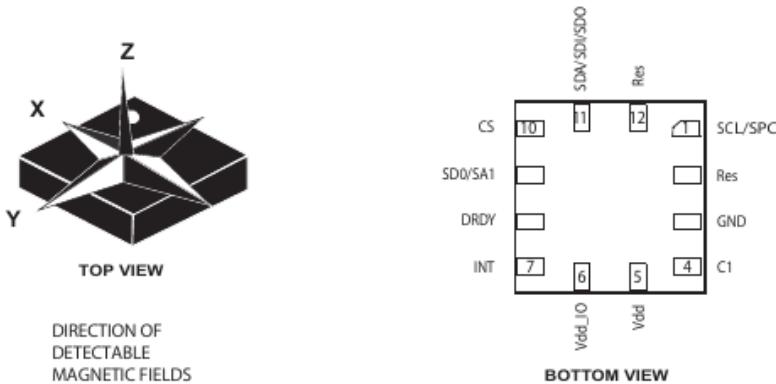


Фигура 9: *LSM6DS33* Ориентация на осите и наименованията на пиновете в пакета

### 3.3 Магнитометър

Изпозван е чип *LIS3MDL* ([Фигура 10](#)) [14], който е интегрирано пакетно решение, предоставящо магнитометър по 3 оси. Чипът използва  $1.9 \rightarrow 3.6V$  захранване като в нужния режим консумира не повече от  $270\mu A$ . Чипът е част от сензорна платка, позволяваща директна връзка чрез протокол *I2C* (адрес: 0x3C).

За целите на настоящия проект е използвана следната конфигурация: директен достъп (без буфериране) до измерените стойности; обхват на всички оси  $\pm 8gaus$ .



Фигура 10: Магнитометър – оси на измерване и наименования на пиновете

### 3.4 Електронен контролер на скоростта за безчеткови постояннотокови мотори (ESC)

Използван е контролер на скоростта за безчеткови постояннотокови мотори *Plush 40amp Speed Controller* (Фигура 11) на *TURNIGY*. Контролерът може да черпи максимум 40A ток за продължително време, както и максимален моментен ток 55A ( $> 10s$ ) [20].

Използвана е настройка за Li-Po батерия, спирачка-включена, опция за намаляване на мощността, средна граница на намаляване на мощността, нормално стартиране, средно време за реакция. Посоката на въртене не е софтуерно настройваема. За смяна на посоката на въртене се разменят които и да е два от изходите към безчетковия мотор. Това променя последователността на сигналите към отделните намотки и посоката на въртене се обръща.

Опциите за настройка и процедурата са описани в [20]. Важно е да се отбележи, че моторите се инициализират след около 6 секунди, при възможно най-ниска стойност на управляващия сигнал. Управляващият сигнал е с честота 50Hz и е ШИМ със запълване между  $(1000 \pm 50 \rightarrow 2000 \pm 50\mu s)$ .



Фигура 11: ESC с анотация на изходите и входовете

### 3.5 Безчеткови постояннотокови мотори (BLDC)

Използвани са безчетковите постояннотокови мотори *AX-4008D KV620*. С KV константа  $620 rpm/V$ , тегло  $76g$  и максимална мощност  $250W$ .

Върху оста на мотора е закрепен затягащ конус с резба за монтиране на витлата. За основата им са предварително изработени поставки за монтиране.



Фигура 12: Безчетков постояннотоков мотор AX-4008D KV620

### 3.6 Li-Po батерия и зарядно

Използвана е Li-Po батерия *TURNIGY NANO 4.5Ah, 14.8V(4S)*, която е четири клетъчна с капацитет 4.5Ah, напрежение 14.8V и максимална скорост на разреждане 50C ( $4.5Ah * 50C = 225A$ ), както и зарядно за батерията *SIGMA 2 EQ* с опции за зареждане и баланс на повечето стандартни типове батерии.



Фигура 13: Li-Po батерия и зарядно

### 3.7 Джойстик и приемник

Използван е джойстик за управление *Dx6i* на *Spektrum* (Фигура 14). Джойстика има 6 канала, и се свързва към приемника чрез радиовълни с честота  $2.4GHz$  използвайки технология разработена от производителя, наречена *DSM Spread Spektrum Technology*. За това се налага използването на приемник, поддържащ специфичната технология.

Използван е приемник *AR6200* на *Spektrum* (Фигура 15). Който поддържа *DSM Spread Spektrum Technology* на джойстика. Каланите на изходите предоставят ШИМ сигнал с честота  $50Hz$  и запълване в диапазона  $9500 \rightarrow 1950\mu s$ .

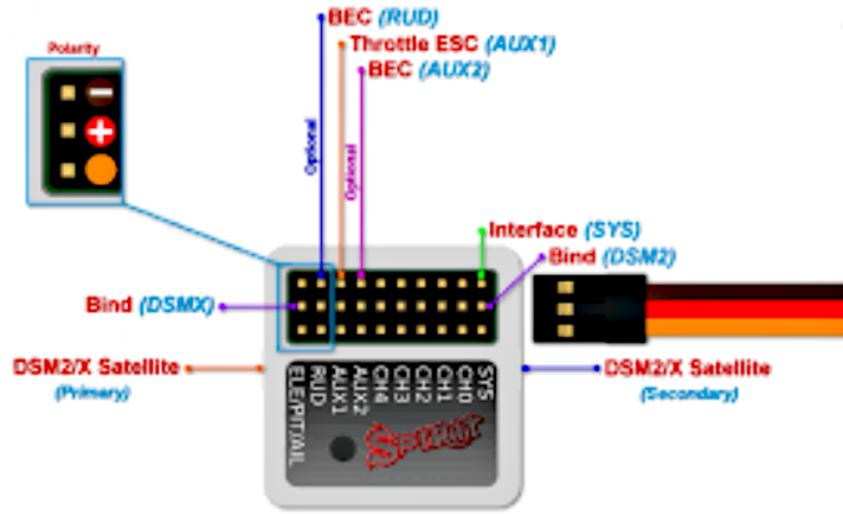
Като конфигурацията на изходните пинове на приемника е описана в Фигура 16.



Фигура 14: Джойстик Spektrum Dx6i



Фигура 15: Приемник AR6200



Фигура 16: Диаграма за свързване на изходните пинове на приемника

## 4 Архитектура на системата

### 4.1 Конструкция на платформата

### 4.2 Платформа с четири ротора

Платформата [Фигура 17](#) е конструирана от два метални П-образни профила, сключващи прав ъгъл помежду си и имащи пресечна точка в средата. В края на профилите се намира по един безчетков постояннотоков мотор (без обратна връзка). Перките са свързани директно (без трансмисия) за въртящата ос на моторите. Батерията и контролният модул са позиционирани в средата на платформата. Батерията се намира под пресечната точка на профилите. Управляващото устройство е над пресечната точка на профилите върху изработена, като част от проекта, платформа.

При този начин на организиране на хардуера центърът на тежестта лежи под пресечната точка на профилите.



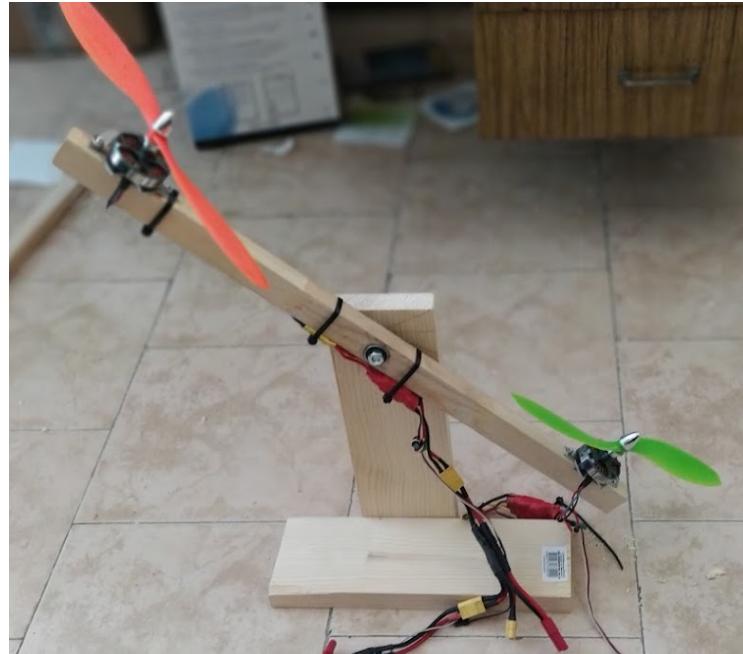
Фигура 17: Конструкция на платформата с четири ротора

### 4.3 Платформа за управление на ъгъл на завъртане

Платформата за управление на ъгъл на завъртане има за цел да ни позволи лесно и безаварийно да изprobваме алгоритмите за управление на ъгъл на завъртане.

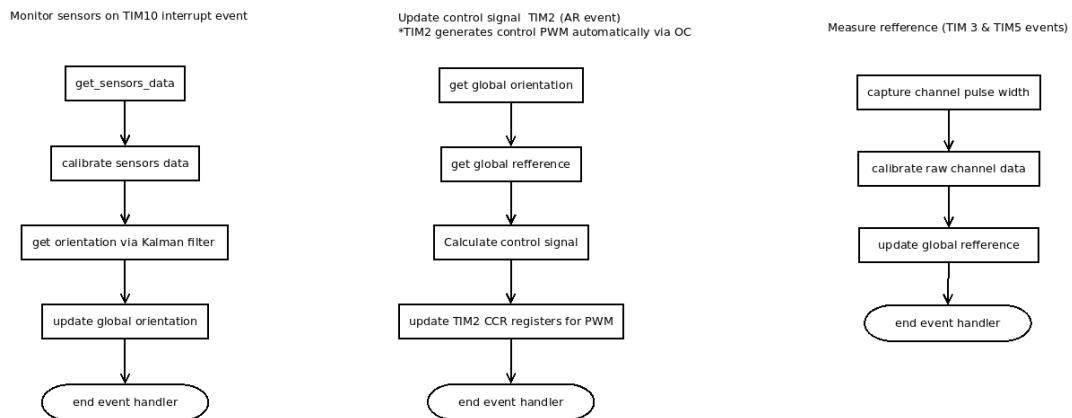
Платформата [Фигура 18](#) е изградена от дърво. Състои се от Т-образна основа с ограничители и въртяща част. Основата е висока  $30cm$  и е съставена от два правоъгълни дървени профила ( $30x10x2.5cm$ ), съединени с винтове. Ограничителите ограничават максималния ъгъл, който въртящата част може да сключва с хоризонта в диапазона ( $\pm 40^\circ$ ). Оста на въртене представлява M8 болт. Оста образува болтово съединение с основата, както и с лагерите на въртящата част. Въртящата част е съставена от правоъгълен дървен профил ( $60x2.5x3cm$ ) с вложени два лагера (сачмен с дълбок канал  $8x22mm$ , максимално статично натоварване  $138kg$  [9]), като са пробити отвори за болтово съединяване на основите за безчетковите постояннотокови мотори, както и отвори за болтово съединяване на основата на жироскопа и акселерометъра. Останалите нужни компоненти като *ESC (Electronic Speed Controller)* за моторите са прикрепени към рамената на въртящата част чрез кабелни превръзки (т.нар свински опашки), като е взето предвид балансиране на

платформата чрез разпределение на тежестта на допълнителните елементи.



Фигура 18: Конструкция на платформата за управление на ъгъл

#### 4.4 Софтуерна част



Фигура 19: Основни събития касаещи управлението

##### 4.4.1 Модул stdperiph

За управление на периферията са имплементирани отделни подмодули, всеки от които е опростен API (Application Programming Interface) за управление на интегрираната периферия в микроконтролера. всеки подмодул е отговорен само и единств-

твено за конкретното интегрирано периферно устройство за което е създаден. Интеграцията между периферните подмодули се осъществява отвън.

Подмодулите от този модул са отговорни за инициализиране и управление на регистрите на интегрираната периферия.

#### 4.4.2 Модул CMSIS

CMSIS е стандарт на ARM за стандарни математически операции. Стандартът дефинира прототипите на функциите, както и множество структури от данни за работа с матрици, вектори, тригонометрични и други математически функции.

В проекта е интегрирана модифицирана CMSIS библиотека, като част от модулът за математика.

#### 4.4.3 Модул math

Този модул предоставя функции за работа с числа, матрици и полиноми, като част от този модул е съставена опростена библиотека за конверсия, която ще бъде използвана за интегриране с останалите модули.

#### 4.4.4 Модул control

Този модул предоставя възможността за лесна инициализация, конфигуриране и използване на средства за управление. За момента са в модула е имплементиран ПИД регулатор както и ограничител.

#### 4.4.5 Модул специфични за проекта функции

В този модул са изнесени подмодули и логически смислени наименования на изпълнявани процедури за удобство при писане на софтуера, както и нагледност. Някои от подмодулите модулите включват подмодул модул за филтър на Калман, подмодул за калибриране и компенсация на сензорни отмествания и други.

## 5 Изграждане на системата и решени проблеми

### 5.1 Изграждане на работната среда

За изграждането на работната среда бяха положени много усилия, вследствие на избора да не се използват готови решения, както и поради нуждата от работа под Linux, тъй като множеството от решения имат целева система Windows.

#### 5.1.1 Система за изграждане (Make)

Работната среда е базирана на *Make*, което ни позволява да изградим собствен инструментариум за целите на проекта. *Make* използва „цели“ (targets) и взаимовръзките „ зависимости“ (Dependencies), които изграждат „целите“. За всяка цел се съставя дърво на зависимостите, като се проверява кои зависимости имат нужда от преправяне (rebuild). *Make* е платформно независима и може лесно да бъде пренесена на почти всяка система. *Make* се нуждае единствено от статично или динамично строго описание на взаимовръзките и начините да се премине от зависимостите към целите. Това става чрез дефиниране на командни скриптове „рецепти“ (recipes), което налага ограничението използваният инструментариум да бъде достъпен чрез команден интерфейс.

#### 5.1.2 Компилатор

Подбраният компилатор е *GCC ARM NON-EABI*, който е платформно независим, с целева крайна система ARM. Компилаторът е конфигуриран през командният интерфейс при извикването си от *Make*, като конфигурацията е публикувана Listing 5. Конфигурацията включва използването на хардуерния модул за обработка на числа с плаваща запетая, изключването на всички оптимизации, които компилаторът прави за подобряване на скоростта и използване на паметта с цел гаранция на правилното изпълнение.

### 5.1.3 Програматор

Подбран е командният пакет за *ST-LINK*, тъй като е платформно независим и позволява командно управление на *ST-LINK V2* устройството, кето е част от комплекта *32F429IDISCOVERY*. Този пакет позволява презаписване на вътрешната флаш памет на контролера, както и стартиране и управление на порт за дебъг, който да бъде използван за дебъгера.

### 5.1.4 Дебъгер

Подбран е дебъгер GDB, тъй като е платформно независим и позволява командно базирано дебъгване през дебъг порта отворен с помощта на *ST-LINK*. Дебъгерът позволява условно и безусловно поставяне на точки за прекъсване (breakpoints) на отделни моменти от изпълнението, като се използва генерираният \*.elf обект, който е записан във флаш паметта на процесора.

### 5.1.5 Получаване и обработка на данни

Денните се изпращат от микроконтролера, през USART порта и се получават на локалната машина (Linux) чрез TTL четец, *picocom*. Данните се записват в \*.log файл, който се обработва автоматично през дефинираната поточна линия от процеси, като създава \*.csv файл, който ще бъде използван от *MATLAB (for Linux)* за обработка на данните и генериране на графики.

### 5.1.6 Линкерен скрипт

Написан е линкерен скрипт, който дефинира правилното разположение на паметта на контролера и управлява разположението и подравняването на символите и секциите в паметта на микроконтролера. Линкерният скрипт поставя всички константни символи във флаш паметта на контролера, което гарантира, че те няма да бъдат променяни. Линкерният скрипт също гарантира правилното разположение на вектора на прекъсванията в паметта. Линкерният скрипт отстранява всички символи, свързани със стандартните библиотеки, тъй като стандартните библиотеки предполагат наличието на операционна система, която да имплементира

подфункциите. По този начин софтуерът, който пишем, е зависим единствено и само от източниците, които ние сме написали или добавили. Крайният резултат е независим статично свързан обект, който ще бъде поставен в микроконтролера.

## 5.2 Конфигурация на микроконтролера

### 5.2.1 Стартране

Написан е прост стартращ скрипт на ARM assembly за целите на проекта, който инициализира SP на процесора и инициализира PC=ResetHandler. Скриптът дефинира символите от таблицата за прекъсванията, като .weak референции, към DefaultHandler, който изпълнява безкраен празен цикъл. Този скрипт има за цел да инициализира .bss секцията на контролера, както и всички статични променливи, като използва позициите, генериирани от линкерния скрипт, както и да извика функцията за системна инициализация, след което се извиква main функцията.

### 5.2.2 Системна инициализация

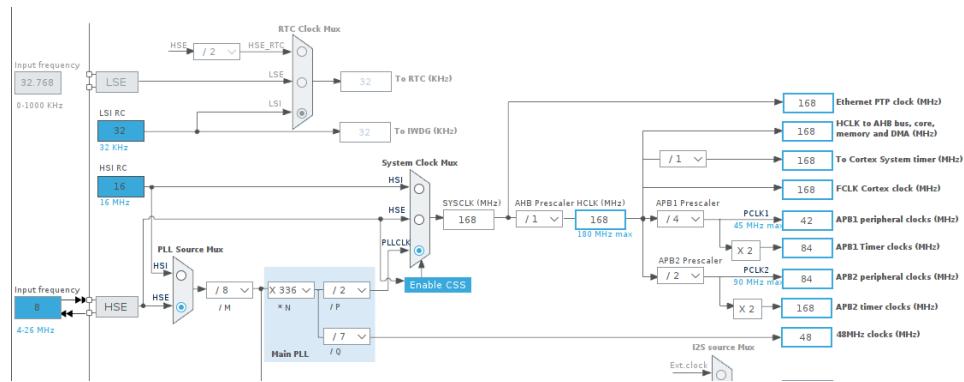
В тази фаза се налага да бъде инициализиран системният часовник. В нашия случай ще инициализираме системния часовник от високочестотният външен осцилатор (HSE). Тъй като при стартиране се използва високочестотният вътрешен осцилатор (HSI), който е неточен и може да доведе до синхронизационни проблеми при асинхронни комуникации с висок baud-rate като UART. Поради тази причина се налага инициализирането на системния часовник с времева основа високочестотният външен осцилатор (HSE), тъй като е много по прецизен. Високочестотният външен осцилатор (HSE) на платката е с честота 8MHz, затова използваме хардуерният PLL модул за увеличаване на честотата със следните настройки:

(PLL\_M=8, PLL\_N=336, PLL\_P=2, PLL\_Q=7)

, като така получаваме 168MHz честота на системния часовник. След това инициализираме делителите на честота за отделните шини:

(AHB\_Prescaler=1, APB1\_Prescaler=4, APB2\_Prescaler=2).

След настройката на часовниците получаваме следните честоти за отделните шини (Фигура 20)



Фигура 20: Честоти на системните шини и часовници след конфигурация

Дуга основна част на системата инициализация е инициализирането на хардуерния модул за числа с плаваща запетая (FPU). Тъй като от конфигурацията (Listing 5) настройваме компилатора за работа с модула за числа с плаваща запетая, е важно преди която и да е операция с числа с плаваща запетая този модул да бъде инициализиран. Необходимо е инициализацията на FPU да се случи преди преминването в ограничен режим. Инициализацията се случва, чрез блокът от код (Listing 1).

```

1  /* Enable The FPU*/
2  uint32_t* CPACR = (uint32_t*)0xE000ED88;
3  *CPACR |= 0b00000000111100000000000000000000;
```

Listing 1: Инициализация на модула за числа с плаваща запетая

### 5.2.3 Инициализация на периферията

В тази фаза се инициализират периферните устройства като таймери, драйвъри за комуникационни протоколи и т.н. Важно е да се отбележи, че преди инициализирането на който и да е периферен драйвър е нужно трябва неговият часовник да бъде разрешен, използвайки rcc интерфейса на шината, на която е закачен.

#### USART :

Инициализиран е USART1, който се използва за извеждане на информация от контролера и получаване на команди. USART1 е инициализиран с конфигурация –

baud-rate 115200; дължина на думата 8 бита; брой стоп битове 1; Режим приемник / предавател.

За вход и изход на модула са инициализирани пинове PB6 и PB7.

За обработка на постъпващите команди е разрешено прекъсване (USART1\_IRQHandler), което обработва прекъсванията свързани с получаване на команди.

За целите на обработка на команди е съставен прост краен автомат, който да обработва постъпващите команди. За момента командният интерфейс поддържа команди за задаване на управляващо въздействие на избран канал, управляващ ESC.

**I2C:** Инициализиран е I2C3, който се използва за комуникация с сензорите (жироскоп, акселерометър). Използвана е следната конфигурация – честота на I2C часовник 100000 Hz; Режим I2C; Адресиране 7 бита;

Поради факта, че периферните устройства използващи I2C изискват запазване на състояние, което не можде изцяло да бъде възстановено от регистрите на I2C периферията е съставен стеков автомат, който да управлява четенето и писането на блок от адреси само през прекъсването. Така избягваме нуждата да изчакваме комуникацията да завърши всеки отделен стадий а се използва прекъсването в комбинация с краен автомат, който използваме за вземане на решенията.

Тъй-като връзката (хардуерно) не е високо надеждна (jumper кабели), се случва периферията да се окаже в състояние на грешка, поради загуба на арбитрация или друг тип грешка. Проблемът е решен, чрез използването на (I2C3\_ER\_IRQHandler), като при възникване на проблем рестартираме или реинициализираме периферията за да продължим комуникацията.

**Таймери:** Инициализирани са таймери TIM2, TIM3, TIM5 и TIM7.

TIM2 е с период 50Hz и се използва в режим Output-compare този таймер генерира сигналите за управление и има резолюция  $1\mu s$ .

TIM3 и TIM5 са конфигурирани с резолюция  $0.2\mu s$  и се използват за приемане на сигналите подадени от оператора, през приемника.

TIM10 е конфигуриран с честота 52Hz, като се използва за прочитане на сензорите и обработка на получените сирови данни.

### 5.3 Моделиране

#### 5.3.1 Моделиране на тяга от витло и мотор

За моделиране на мотора и витлото е приет опростеният модел, който моделира системата като коефициент на усилване и нискочестотен филтър.



Фигура 21: Опростен модел на витло и мотор

$$G_{motor}(s) = \frac{k_{motor}}{s\tau_{motor} + 1} \quad (1)$$

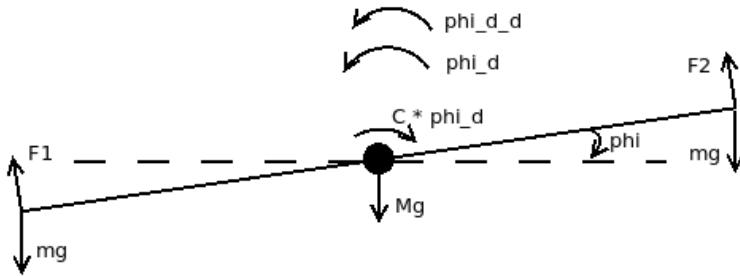
Този модел не е добра апроксимация на тягата на моторите, защото реалната характеристика е нелинейна и се променя спрямо множество параметри (напрежение на батерията, температура, налягане, вляжност, скорост), но за целите на проекта, т.е. в неголеми времеви интервали и неголеми промени във входния сигнал статичната характеристика може да бъде описана достатъчно добре, чрез линейната апроксимация ([Уравнение 1](#)).

#### 5.3.2 Моделиране на платформа за управление на ъгъл на завъртане

За моделиране на платформата за управление на ъгъл на завъртане ([Фигура 22](#)) нека приемем, че въртящата час е пренебрежимо тънка, с дължина  $l$ , и има маса  $M$ . Ъгълът, който въртящата част сключва с хоризонта е означен с  $\phi$ . Във края на рамената се намират безчетковите мотори и витлата, моделирани, като концентрирани маси с големина  $m$ , като всеки от тях има сила на тягата означена с  $F_1, F_2$  респективно. Триенето с въртящата ос е моделирано с коефициентът на триене  $C$ .

От втори закон на Нютон [Уравнение 2](#),

$$I\ddot{\phi} = \sum_i \tau_i \quad (2)$$



Фигура 22: Диаграма на платформа за управление на ъгъл на завъртане

От където следва:

$$I\ddot{\phi} = F_2 * \frac{l}{2} - F_1 * \frac{l}{2} - C\dot{\phi} \quad (3)$$

Нека:

$$F_2 = F_0 + \delta f, F_1 = F_0 - \delta f \quad (4)$$

След като заместим в Уравнение 3

$$I\ddot{\phi} = (F_0 + \delta f) * \frac{l}{2} - (F_0 - \delta f) * \frac{l}{2} - C\dot{\phi} \quad (5)$$

Получаваме:

$$I\ddot{\phi} + C\dot{\phi} = l * \delta f \quad (6)$$

Което опростява системата до линейна SISO система.

Апроксимираме инерционният момент  $I$  като :

$$I = I_{rod} + 2 * I_{motor} \quad (7)$$

$$I = \frac{1}{12}Ml^2 + 2m\left(\frac{l}{2}\right)^2 \quad (8)$$

$$I = \frac{l^2}{12}(M + 6m) \quad (9)$$

При нулеви начални условия след трансформация на Лаплас получаваме:

$$s^2 I \bar{\Phi}(s) + sC\bar{\Phi}(s) = l * \bar{\delta f}(s) \quad (10)$$

От където получаваме предавателната функция:

$$G_{platform}(s) = \frac{\bar{\Phi}(s)}{\bar{\delta f}(s)} = \frac{l}{s(sI + C)} \quad (11)$$

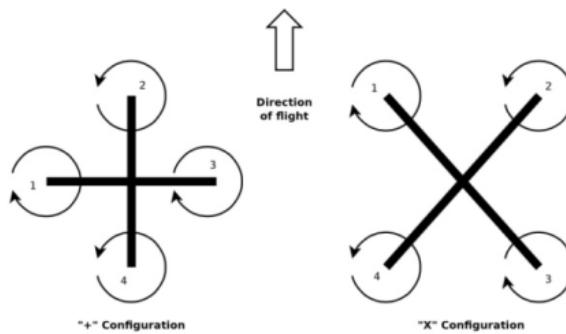
За получаване на предавателната функция на отворената система:

$$G_{sys} = G_{motor}(s)G_{platform}(s) = \frac{k_{motor}l}{s(sI + C)(s\tau_{motor} + 1)} \quad (12)$$

### 5.3.3 Моделиране на платформа с 4 ротора

Безполитните летателни платформи с 4 ротора се разделят на 2 основни дизайна "+" и "x". X-конфигурацията се счита за по-стабилна, отколкото + конфигурацията.

Рамката на платформата се състои от 2 кръстосани рамена, като роторите са монтирани в краишата им. Роторите на всяко рамо се въртят в противоположни посоки както е показано на фигура23 Важно е да се отбележи, че перките на ротори 1 и 3 имат обратен наклон спрямо перките на роторите 2 и 4. Така тягата на всички ротори е в еднаква посока. Респективно различната посока на въртене на роторите е нужна за да може приведеният въртящ момент да бъде равен на нула, за да може платформата да запази своята ориентация.



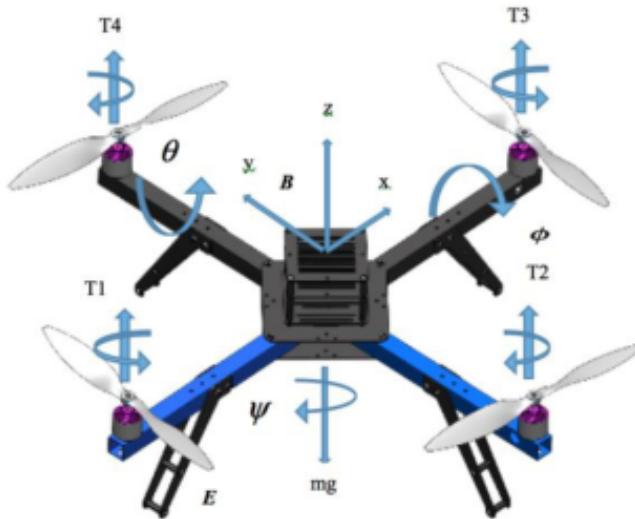
Фигура 23: Основни типове на конфигурация .

Тук е важно да дефинираме 2 основни координатни системи, които ще използваме. Първата е спрямо земята ( $X_E, Y_E, Z_E$ ) а втората е спрямо тялото на квадракоптера ( $X_B, Y_B, Z_B$ ). Ориентацията на квадракоптера може да се дефинира чрез Ойлеровите ъгли на тялото на квадракоптера. На фигура24 са показани коростите на въртене на роторите  $\omega_1, \omega_2, \omega_3, \omega_4$ , силите на тягата генерирана от перките  $T_1, T_2, T_3, T_4$ .

Позицията на квадракоптера е дефинирана, в координатната система спрямо земята по осите  $x, y, z$  с вектора  $\xi = [x, y, z]^T$ . А ориентацията се дава, чрез Ойлеровите ъгли  $\eta = [\phi, \theta, \psi]^T$ . А векторът  $q = [\xi, \eta]^T$  съдържа както се вижда и линейните и ъгловите координати.

Центъра на масите на дрона се смята за основа на координатната система на тялото на дрона. Спрямо тялото на дрона линейните скорости се определят от  $JB = \begin{bmatrix} J_x, B \\ J_y, B \\ J_z, B \end{bmatrix}$

а тъгловите скорости се определят от  $\omega = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$



Фигура 24: Сили, моменти и отправни системи на квадракоптера

За да може квадракоптерът да се носи във въздуха е нужно да са спазени следните условия:

$$\sum_{i=1}^4 T_i = -mg \quad (13)$$

$$\sum_{i=1}^4 M_i = 0 \quad (14)$$

$$T_{1,2,3,4} \parallel g \quad (15)$$

$$(\omega_1 + \omega_3) - (\omega_2 + \omega_4) = 0 \quad (16)$$

от което и следва, че  $\phi = 0, \theta = 0, \psi = 0$ .

При увеличаване/намаляване на скоростта на роторите дронът може да се движи, нагоре и надолу. За да се движим нагоре:  $\sum_{i=1}^4 T_i > -mg$ . И респективно да се двишим надолу:  $\sum_{i=1}^4 T_i < -mg$ . Като не забравяме, че и в двета случая другите параметри остават незасегнати.

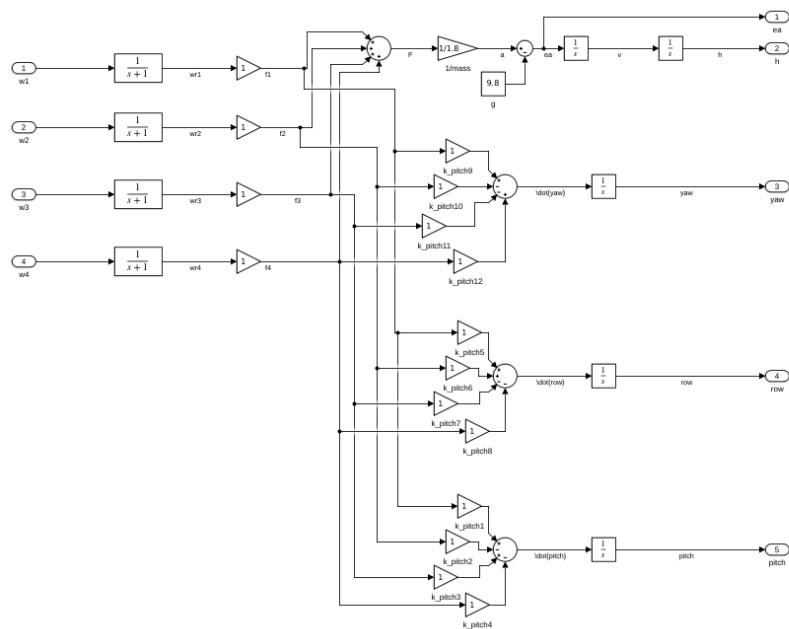
За промяна на ориентацията се налага да:

$$\dot{\psi} = k_\psi((\omega_1 + \omega_3) - (\omega_2 + \omega_4)), \psi = \int \dot{\psi} dt \quad (17)$$

$$\dot{\phi} = k_\phi((\omega_1 + \omega_4) - (\omega_2 + \omega_3)), \phi = \int \dot{\phi} dt \quad (18)$$

$$\dot{\theta} = k_\theta((\omega_1 + \omega_2) - (\omega_3 + \omega_4)), \theta = \int \dot{\theta} dt \quad (19)$$

От което следва, че при намаляне на скоростта на ротор 2 и увеличаване на скоростта на ротор 4 постигаме завъртане по  $\phi$ . Съответно намаляване на скоростта на ротор 1 и увеличаване на скоростта на ротор 3 постигаме завъртане по  $\theta$  и съответно намаляване на скоростта на срещуположни ротори и съответно увеличаване на другите два постигаме завъртане по  $\psi$ .



Фигура 25: SIMULINK схема на динамиката на платформа с 4 ротора.

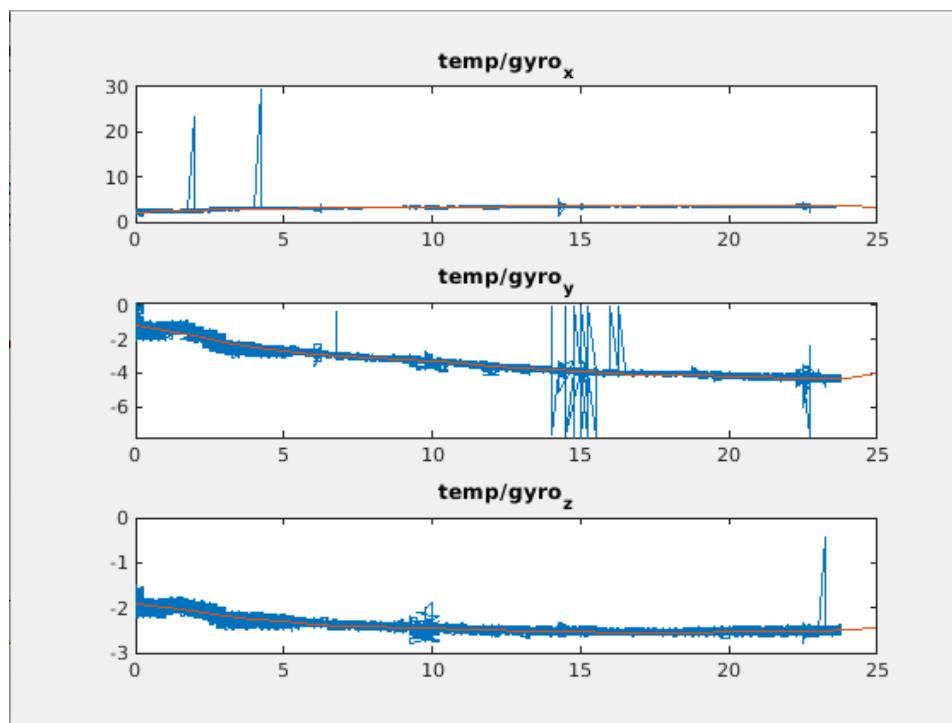
## 5.4 Компенсация на сензорни отмествания

### 5.4.1 Компенсация на жироскопен дрейф

Жироскопите обикновено не са силно зашумени за разлика от акселерометрите, но имат своите недостатъци. Основен проблем при жироскопите е т. нар. жироскопен дрейф. В неподвижно състояние жироскопът показва малко постоянно завъртане в някоя посока. Жироскопният дрейф зависи основно от температурата.

За да калибрираме и занулим стойностите на жироскопа се налага да направим полиномиална компенсация по температура.

За целта жироскопа е ухладен до  $0^{\circ}C$  след което е поставен неподвижно бавно да се загрее в продължение на 45мин до стайна температура. През цялото време събираме данните от жироскопа и измерената от бордовия термометър температура. След приключване на експеримента използваме matlab за намиране на полином, който точно описва получените данни за всяка от осите.



Фигура 26: Жироскопен дрейф спрямо температура и полиномиална компенсация спрямо температура

```

1
2 p_x =
3
4 -42.6978e-9 3.7797e-6 -131.8066e-006 2.2717e-3 -19.5918e-3 67.9744e-3
5 95.1384e-3 2.2440e+0
6
7 p_y =
8 87.1716e-9 -7.6384e-6 263.1019e-006 -4.4660e-3 37.8030e-3 -129.2365e-3
9 -191.6577e-3 -1.1695e+0
10
11 p_z =
12 15.9591e-9 -1.4613e-6 53.0721e-006 -961.5439e-6 8.8365e-3 -33.3197e-3

```

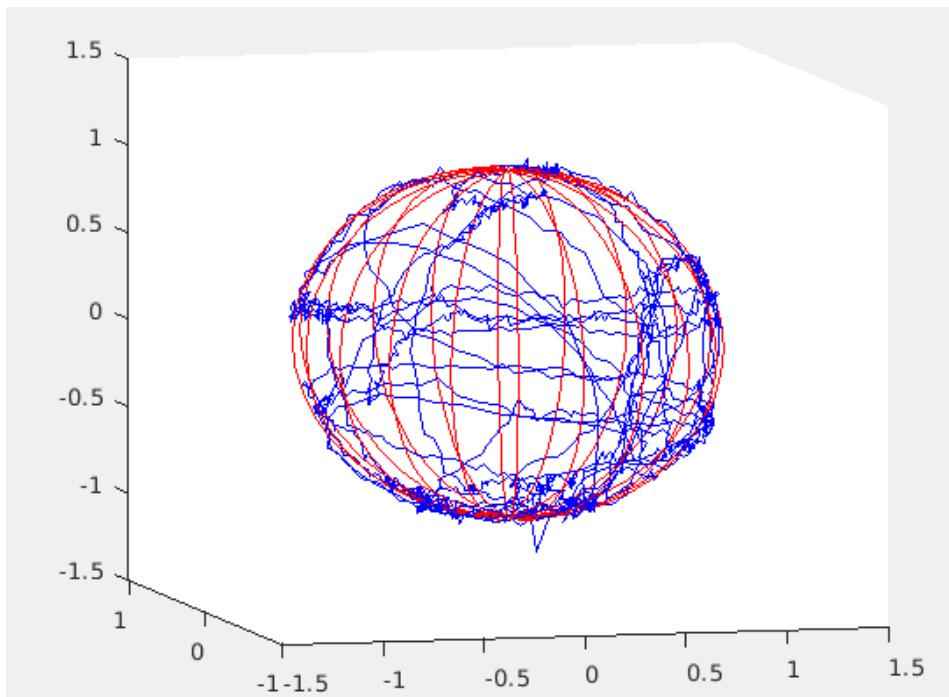
```
-44.0584e-003 -1.9076e+0
```

Listing 2: Получени полиноми за компенсация на жироскопният дрейф

#### 5.4.2 Компенсация на магнитни отмествания от околната среда

Магнитометрите имат постоянно отместване поради средата в която се намират. Това отместване се дължи на металните и магнитни обекти, както и всички електромагнитни смущения причинени от заобикалящата ни среда. Поради факта, че в близката ни околнна среда тези смущения са постоянни, което означава, че ние можем да ги компенсираме статично.

За целта поставяме сензора в нормална позиция и бавно го завъртаме, като се стараем да покрием възможно най-много ъглови комбинации. След получаване на данните виждаме, че те образуват Елипсоид с отместен център. Използваме техника на Мерайо. Идентифицираме елипсоида и отместването на центъра, след което правим трансформация елипсоид-сфера, след което изваждаме отместването на центъра. Верифицираме резултатът, като виждаме, че всички измерени точки са в допустим диапазон от повърхността на сфера с център 0 и радиус 1.



Фигура 27: Компенсирани данни получени от магнитометър

```
1
2 U =
```

```

3
4      21.4229e+000   837.9000e-003   3.0861e+000
5      0.0000e+000    21.6201e+000   2.0207e+000
6      0.0000e+000    0.0000e+000    27.2441e+000
7
8  C  =
9
10     -1.6000e-003
11     19.9000e-003
12     45.7000e-003

```

Listing 3: Получени трансформация елипсоид-сфера и отместване на център

#### 5.4.3 Баланс на витлата

След инспекция, използвайки балансьора за витла бе установено, че използвани пластиносови битла не са балансираны. Не балансираните витла имат отместен център на тежестта, който има неблагоприятен ефект поради голямата скорост на въртене на витлата по време на работа. Въртящият се изместен център създава силни високочестотни вибрации, които биват измервани от акселерометъра, кето зашумява и измерването на ъглите на завъртане. Неблагоприятният ефект от небалансирана витла, както и количествено оценяване на големината и характера на този ефект са разгледани в този труд [8]

За максимално компенсиране на този ефект витлата са балансираны по следната процедура за статично балансиране:

Витлата са захванати и центрирани върху оста от двата алюминиеви конуса. Подвижният конус се задържа здраво към отвора на витлото чрез затягане.

Оста се поставя на две нивелирани повърхности с ниско триене. Между повърхностите се намира витлото пристегнато от конусите [Фигура 28](#).

Има два основни начина за балансиране – адитивен и изваден. При адитивният метод добавяме тежест към по-тежката част на витлото, което много намират за по-лесният начин да се балансираят витлата. За адитивният метод може да се използва Гел базирано супер лепило, което се поставя върху по-леката страна на витлото. Друг метод е поставянето на тиксо върху по-леката страна, което го прави удобно за отстраняване при нужда. За извадният метод се използва шкурка за отстраняване на малки количества от пластиносата на витлото. Важно е да се отбележи, че при



Фигура 28: Небалансирано витло

извадният метод не трябва да се отстраняват големи количества материал, затова на всяка итерация се премахва малко количество макетиал от по-тежката страна, като постоянно витлото бива проверявано. Извадният метод запазва аеродинамичната форма на витлото (ако бъде изпълнен коректно).



Фигура 29: Балансирано витло

## 5.5 Калманов филтър

Калмановият наблюдател е рекурсивен естиматор на състоянието на системата, който дава статистически оптimalна естимация на състоянието дори и при използване на зашумени входни данни. Калмановият филтър често се използва при т. нар сензорно сливане, и е най-честият метод, който се използва за обединяване

на жироскоп и акселерометър за намиране на ориентация. Калмановият филтър може да бъде разделен на 2 основни стъпки: предсказване и обновяване.

В стъпка предсказване алгоритъма продуцира оценка за състоянието, както и за неопределеностите в състоянието, в резултат на шум. Тази стъпка може да бъде изпълнена множество пъти, като предсказанието е все по-неопределеност с всяка итерация. В стъпка обновяване алгоритъма сравнява предсказанията спрямо зашумените измервания и обновява оценката използвайки претеглена комбинация от предсказание и измерване, като се дава по-високо тегло на оценките с по-ниска неопределеност.

Алгоритъма може да бъде изпълняван на всяка итерация, без нужда от стари стойности на използваните променливи.

Крайната естимация може да бъде описана като оптимално решение, което системата е линейна и е зашумена от шум в състоянията и шумове от измерванията с Бял Гаусов Шум. Можем да представим това математически чрез модел на предсказанието и модел на измерването.

$$x_k = Fx_{k-1} + Bu_k + N(0, Q_k) \quad (20)$$

$$z_k = Hx + N(0, R_k) \quad (21)$$

Филтърът ще бъде използван за оценка на ойлеровите ъгли, на завъртане спрямо оста X и оста Y. Така дефинираме вектор на състоянията във форма:

$$x = \begin{bmatrix} \phi \\ \psi \\ \omega_{xb} \\ \omega_{yb} \end{bmatrix}_k$$

Моделите на измерване и предсказване са дефинирани:

$$\begin{aligned} \begin{bmatrix} \phi \\ \psi \\ \omega_{xb} \\ \omega_{yb} \end{bmatrix}_k &= \begin{bmatrix} 1 & 0 & -d_t & 0 \\ 0 & 1 & 0 & -d_t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}_k * \begin{bmatrix} \phi \\ \psi \\ \omega_{xb} \\ \omega_{yb} \end{bmatrix}_{k-1} + \begin{bmatrix} d_t & 0 & 0 & 0 \\ 0 & d_t & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_k * \begin{bmatrix} \omega_x \\ \omega_y \\ 0 \\ 0 \end{bmatrix}_k + N(0, Q_k) \quad (22) \\ \begin{bmatrix} A_\phi \\ A_\psi \\ 0 \\ 0 \end{bmatrix}_k &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_k * \begin{bmatrix} \phi \\ \psi \\ \omega_{xb} \\ \omega_{yb} \end{bmatrix}_k + N(0, R_k) \quad (23) \end{aligned}$$

След снемане на експериментални данни са подбрани следните ковариантни матрици:

$$Q = \begin{bmatrix} 33.8 & 0 & 0 & 0 \\ 0 & 33.8 & 0 & 0 \\ 0 & 0 & 0.4 & 0 \\ 0 & 0 & 0 & 0.4 \end{bmatrix}, \quad R = \begin{bmatrix} 32 & 0 & 0 & 0 \\ 0 & 32 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (24)$$

Имайки дефинирани матриците изграждащи филтъра Заместваме в следните уравнения за получаване на оценките:

**Стъпка предсказване:**

$$x_{k|k-1} = Fx_{k-1|k-1} + Bu \quad (25)$$

$$P_{k|k-1} = FP_{k-1|k-1}F^T + Q \quad (26)$$

**Стъпка обновяване:**

$$y = z - Hx_{k|k-1} \quad (27)$$

$$S = HP_{k|k-1}H^T + R \quad (28)$$

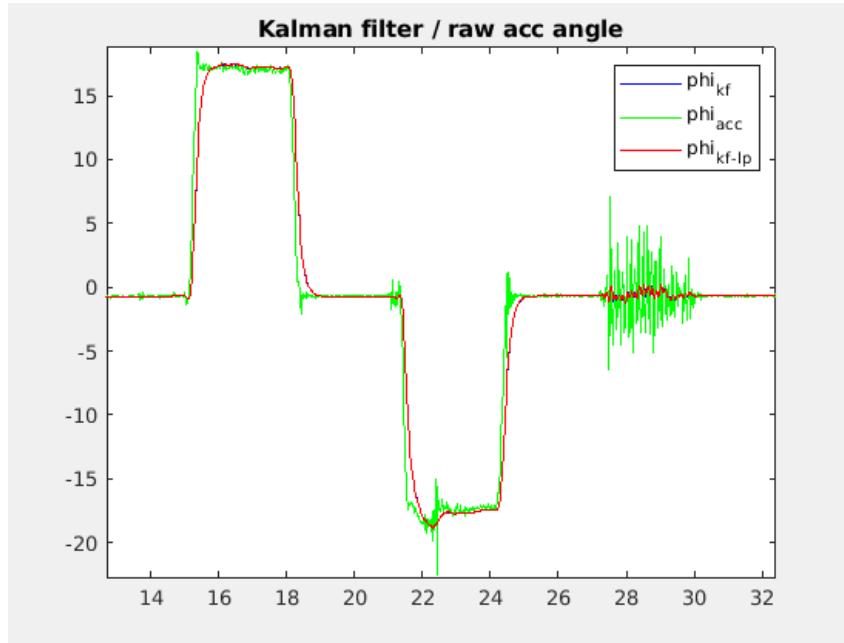
$$K = P_{k|k-1} H^T S^{-1} \quad (29)$$

$$x_{k|k} = x_{k|k-1} + Ky \quad (30)$$

$$P_{k|k} = (I - KH)P_{k|k-1} \quad (31)$$

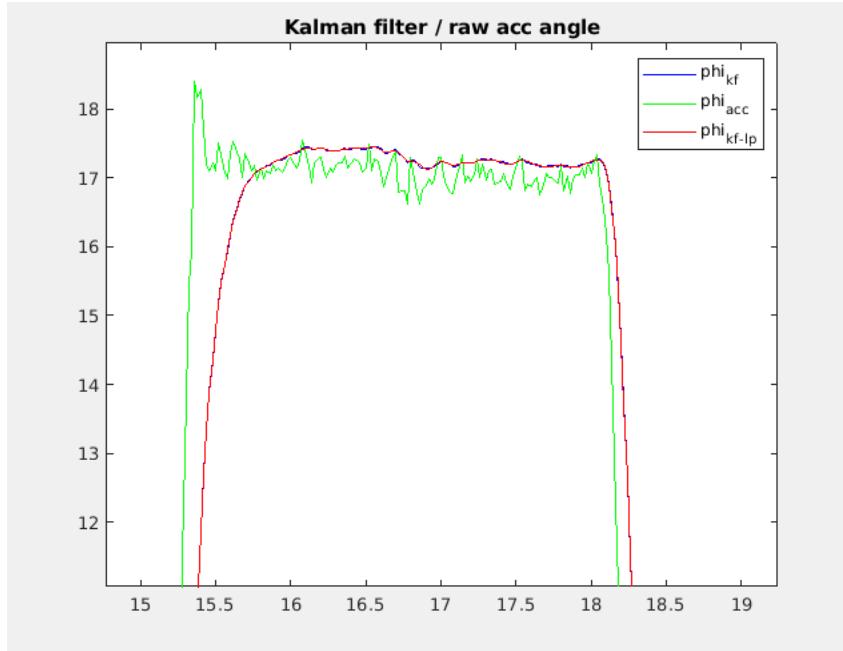
Така синтезириания филтър на Калман е имплементиран в контролера. За проверка на филтъра е изпълнен следният експеримент. След стартиране изчакваме, за да сме сигурни, че филтърът се е устнановил. След което накланяме ръчно дрона под ъгъл около  $20^\circ$ , като задържаме ъгъла за няколко секунди поставяме в изходно положение отново за няколко секунди, след което повтаряме наклона в обратната посока и отново задържаме. Връщаме в изходна позиция изчакваме, след което внасяме допълнителни смущения в системата, чрез относително силни почуквания в успоредник на осите на въртене.

Резултатите от експеримента могат да бъдат проверени в ([Фигура 30](#), [Фигура 31](#)).



Фигура 30: Проверка на Калман филтъра за ос  $\phi$

Забелязваме, че филтъра успешно потиска смущенията и крайният резултат се установява за около  $0.1s$ .



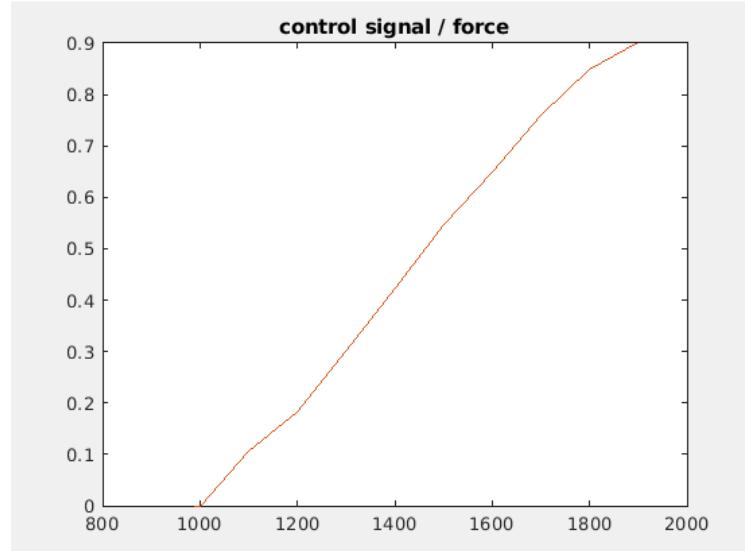
Фигура 31: Проверка на Калман филтъра за ос  $\phi$ , приближен

## 5.6 Експериментални задачи

### 5.6.1 Снемане на статичните характеристики за тягата на мотор с витло

Характеристиките са снети при използване на напълно заредена батерия. Процедурата по снемане е както следва: Използвана е платформата за управление на ъгъл на завъртане, като под рамо едно е поставена везна. Върху везната се поставя право парче дърво, което да се допира във въртящата част на платформата, на разстояние от центъра равно на разстоянието на срещуположният двигател от центъра. По този начин предаването на сила е 1:1. Кантара се занулява и се задават управляващи сигнали в диапазона  $950 \rightarrow 1950ms$  запълване, на ШИМ  $50Hz$ . при подаването на всеки сигнал се изчаква преходните процеси да преминат и се снема стойността от везната.

След снемане на характеристиките е изчислен предавателен коефициент за тягата спрямо отместване от опорната точка  $1500\mu s$   $k_{motor} = 12.2e - 3$  в диапазона на управление ( $1300 \rightarrow 1700$ ). Важно е, че характеристиката е почти линейна в диапазона на управление. От тук можем да заключим, че модела на системата описан в ?? е достатъчно добра апроксимация на системата в диапазона на управление, тъй като нелинейностите на модела се проявяват извън този диапазон. В опорната точка е снета тяга  $424g \approx 4.15N$

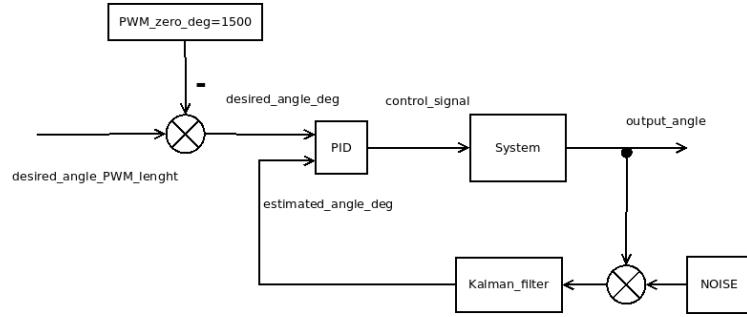


Фигура 32: Управляващ сигнал / тяга

### 5.6.2 Управление на ъгъл на завъртане

Използвайки сглобената платформа и модела от [Събсъбсекция 5.3.2](#). Е синтезирано управление използващо PID регулатор.

Схемата на управление е описана в [Фигура 33](#).

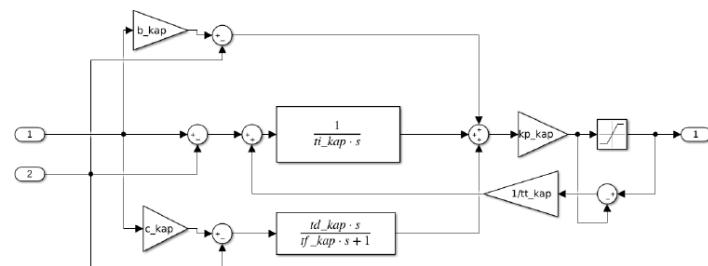


Фигура 33: Упростена схема на управлението

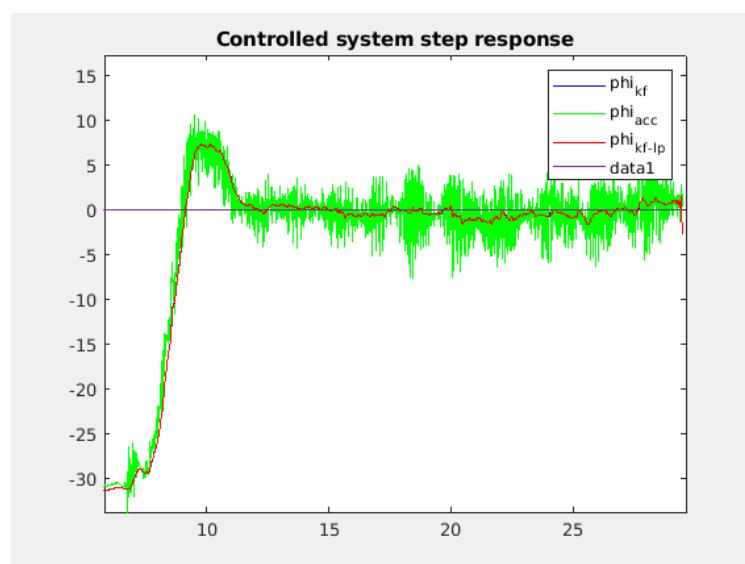
Използваният ПИД регулатор е имплементация на универсален ПИД ([Фигура 34](#)), с настройка:

$$K_p = 0.4, K_i = 0.003, K_d = 2, \tau_d = 0.26, saturation = \pm 150$$

След затваряне с ПИД регулатора от системата е снет преходният процес показан на ([Фигура 35](#))



Фигура 34: Схема ПИД регулатора



Фигура 35: Преходен процес на затворената система

## 6 Предложения за надграждане

При експериментирането с платформата бе забелязано, че металните профили не поддържат добре вибрациите, което създава силни зашумявания в акселерометъра. Това не се наблюдава в повечето готови решения от подобен размер, поради използвамето на карбонови профили.

За подобряване на стабилността, както и качеството на управлението се предлага използване на по-високи честоти за измерванията, както и използване на контролери за управление с дигитален вход за скоростта на моторите, тъй като използването на ШИМ не е надеждно за точно задаване на скоростта.

За момента този труд се фокусира върху управлението на ъгъл на завъртане, като един от основните проблеми за цялостно изграждане на беезпилотна платформа с 4 ротора. Като надграждане на този труд се предлага интегрирането на управление по височина използвайки сензорно сливане на сензор за разстояние, и барометър. Както и използването на GPS и камера за управление, чрез позициониране в пространството.

## 7 Литература

- [1] 3D Jake, *Technical Data Sheet, 3DJAKE ecoPLA*, 3D Jake, 2018. url: [https://cdn-3d.niceshops.com/upload/file/Technical\\_Data\\_Sheet\[0\].pdf](https://cdn-3d.niceshops.com/upload/file/Technical_Data_Sheet[0].pdf).
- [2] ARM Limited, *ARM v7-M Architecture Reference Manual*, ARM, 2018. url: <https://developer.arm.com/documentation/ddi0403/ed>.
- [3] ARM Limited, *Cortex-M4 Technical Reference Manual (Revision r0p1)*, ARM, 2020. url: <https://developer.arm.com/documentation/100166/0001/>.
- [4] T. Bresciani, „Modelling, identification and control of a quadrotor helicopter“, *MSc theses*, 2008.
- [5] GNU, *GCC Manual, Arm Options*, GNU, 2021. url: <https://gcc.gnu.org/onlinedocs/gcc/ARM-Options.html>.
- [6] IKALOGIC S.A.S., *SQ Series User Manual SQ25/SQ50/SQ100/SQ200 4 channels, 200 MSPS logic analyzer and pattern generator*, IKALOGIC S.A.S., -. url: <https://cdn.ikalogic.com/docs/ds/sq-series-manual-EN.pdf>.
- [7] R. Kalachev. (2022). GitHub User: Rafael Kalachev, url: [https://github.com/Rafael-Kalachev/thesis\\_stm32\\_base](https://github.com/Rafael-Kalachev/thesis_stm32_base) (дата на посещ. 12.02.2012).
- [8] E. Kuantama, O. G. Moldovan, I. Țarcă, T. Vesselényi и R. Țarcă, „Analysis of quadcopter propeller vibration based on laser vibrometer“, *Journal of Low Frequency Noise, Vibration and Active Control*, т. 40, № 1, с. 239—251, 2021.
- [9] Mechatronics Bearing Group, *608-2RS BEARING DATASHEET*, Mechatronics Bearing Group, -. url: <https://www.mechatronicsbearing.com/data-sheets/608-2RS-Extra-Small-Ball-Bearings.pdf>.
- [10] L. X. Minh Quan Huynh Weihua Zhao, *L1 adaptive control for quadcopter: Design and implementation*, пор. -. IEEE Control Automation Robotics и Vision (ICARV), 2014, ISBN: -.
- [11] Sain samrt, *User Manual: CREALITY ENDER-3 V2 3D PRINTER*, Sain samrt, 2020. url: <https://m.media-amazon.com/images/I/B1f9eP6H3OS.pdf>.
- [12] J. Solà, „Quaternion kinematics for the error-state Kalman filter“, техн. докл., 2017.
- [13] ST Microelectronics, *Errata sheet STM32F427/437 and STM32F429/439 line limitations*, ST Microelectronics, 2021. url: [https://www.st.com/resource/en/errata\\_sheet/es0206-stm32f427437-and-stm32f429439-line-limitations-stmicroelectronics.pdf](https://www.st.com/resource/en/errata_sheet/es0206-stm32f427437-and-stm32f429439-line-limitations-stmicroelectronics.pdf).
- [14] ST Microelectronics, *LIS3MDL Digital output magnetic sensor: ultra-low-power, high-performance 3-axis magnetometer*, ST Microelectronics, 2015. url: <https://www.st.com/resource/en/datasheet/lis3mdl.pdf>.

- [15] ST Microelectronics, *LPS25H MEMS pressure sensor: 260-1260 hPa absolute digital output barometer*, ST Microelectronics, 2014. url: <https://www.st.com/resource/en/datasheet/lps25h.pdf>.
- [16] ST Microelectronics, *LSM6DS33 iNEMO inertial module: accelerometer and 3D gyroscope*, ST Microelectronics, 2015. url: <https://gcc.gnu.org/onlinedocs/gcc/ARM-Options.html>.
- [17] ST Microelectronics, *RM0090 Reference manual STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced ARM-based 32-bit MCUs*, ST Microelectronics, 2020. url: [https://www.st.com/resource/en/reference\\_manual/dm00031020-stm32f405-415-stm32f407-417-stm32f427-437-and-stm32f429-439-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf](https://www.st.com/resource/en/reference_manual/dm00031020-stm32f405-415-stm32f407-417-stm32f427-437-and-stm32f429-439-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf).
- [18] ST Microelectronics, *STM32F427xx STM32F429xx 32b Arm Cortex-M4 MCU+FPU, 225DMIPS, up to 2MB Flash/256+4KB RAM, USB OTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 20 com. interfaces, camera and LCD-TFT*, ST Microelectronics, 2021.
- [19] ST Microelectronics, *UM1061 Description of STM32F2xx Standard Peripheral Library*, ST Microelectronics, 2011. url: [https://www.st.com/resource/en/user\\_manual/um1061-description-of-stm32f2xx-standard-peripheral-library-stmicroelectronics.pdf](https://www.st.com/resource/en/user_manual/um1061-description-of-stm32f2xx-standard-peripheral-library-stmicroelectronics.pdf).
- [20] TURNIGY, *TURNIGY Manual for Brushless motor Speed Controller*, TURNIGY, -. url: [https://cdn-global-hk.hobbyking.com/media/file/t/u/turnigy\\_plush\\_manual.pdf](https://cdn-global-hk.hobbyking.com/media/file/t/u/turnigy_plush_manual.pdf).
- [21] M. Watson, „The design and implementation of a robust ahrs for integration into a quadrotor platform“, *Meng electronic engineering, Department of electronic and electrical engineering*, 2013.
- [22] Q. H. ( Wingo), *Mathematical Modeling of Quadcopter Dynamics*, пор. -, ISBN: -.
- [23] З. М. С. В. Б. Заманов Д. Н. Каастоянов, *Механика и управление на роботите*, пор. -. Литера прнт, 1993, ISBN: -.
- [24] а. и. О. С. М. гл. ас. д-р инж. Иван Евгениев Иванов, *Микропроцесорна техника*, пор. -. ТУ-София, 2008, ISBN: -.

## Приложения

### A Системна конфигурация

```

1
2 ## 
3 # LOGGING
4 ##
5
6 LOGFILE := build/buildlog.log
7
8 ## VERBOSITY
9 # 0 : Do NOT log
10 # 1 : Log only the supplied message
11
12 VERBOSITY := 1
13 VERBOSITY_ECHO := $(VERBOSITY)
14 VERBOSITY_LOG := $(VERBOSITY)
15
16 LOG_FLAGS :=
17 LOG_FLAGS += --vecho=$(VERBOSITY_ECHO)
18 LOG_FLAGS += --vlog=$(VERBOSITY_LOG)
19 LOG_FLAGS += --logfile=$(LOGFILE)
20
21 ##
22 # CTAGS
23 ##
24
25 CTAGS := ctags
26 CTAGS_FLAGS := --recurse=yes
27 CTAGS_FLAGS += --exclude=.git
28 CTAGS_FLAGS += --exclude=BUILD
29 CTAGS_FLAGS += --exclude=*.swp
30 CTAGS_FLAGS += --exclude=*.o
31 CTAGS_FLAGS += --exclude=*.a
32
33
34
35
36 ##
37 # CONTROLLER
38 ##
39
40 MCU_FLASH_MEMORY_START_ADDRESS := 0x08000000
41 MCU_MODEL := STM32F429_439xx

```

```

42
43 ## 
44 # COMPIILATION
45 ##
46
47
48 C_FLAGS := 
49
50 # for debugging
51 C_FLAGS += -g
52 # optimization level
53 C_FLAGS += -O0
54 # configure for the cortex m4
55 C_FLAGS += -mlittle-endian -mthumb -mcpu=cortex-m4 -mthumb-interwork
56 # configure FPU
57 C_FLAGS += -mfloating-abi=hard -mfpu=fpv4-sp-d16
58 # configure Warnings
59 C_FLAGS += -Wall -Wextra
60 # system includes
61 C_FLAGS += -I./ startup
62 C_FLAGS += -I./ std_perif/ inc
63 C_FLAGS += -I./ system
64 C_FLAGS += -I./ cmsis/ inc
65 # home includes
66 C_FLAGS += -I.
67 # controller model
68 C_FLAGS += -D$(MCU_MODEL)
69 C_FLAGS += -DUSE_STDPERIPH_DRIVER
70 C_FLAGS += -DUSE_FULL_ASSERT
71 C_FLAGS += -DARM_MATH_CM4
72
73
74
75 LD_FLAGS :=
76 # add the linker script
77 LD_FLAGS += -Tlinker/stm32_flash.ld

```

Listing 4: config.mk

## Б Процедура за изчисляване на магнитометърна компенсация

```

1 function [U, c] = MgnCalibration(X)
2 % performs magnetometer calibration from a set of data
3 % using Merayo technique with a non iterative algoritm

```

```

4 % J.Merayo et al. "Scalar calibration of vector magnetometers"
5 % Meas. Sci. Technol. 11 (2000) 120–132.
6 %
7 % X      : a Nx3 (or 3xN) data matrix
8 %           each row (columns) contains x, y, z measurements
9 %           N must be such that the data set describes
10 %           as completely as possible the 3D space
11 %           In any case N > 10
12 %
13 % The calibration tries to find the best 3D ellipsoid that fits the data set
14 % and returns the parameters of this ellipsoid
15 %
16 % U      : shape ellipsoid parameter, (3x3) upper triangular matrix
17 % c      : ellipsoid center, (3x1) vector
18 %
19 % Ellipsoid equation : (v-c)'*(U'*U)(v-c) = 1
20 % with v a rough triaxes magnetometer measurement
21 %
22 % calibrated measurement w = U*(v-c)
23 %
24 % author : Alain Barraud, Suzanne Lesecq 2008
25 %
26 %%%%%%%%%%%%%%
27 [N,m] = size(X);
28 if m>3&&N==3,X = X';N = m;m = 3;end;%check that X is not transposed
29 if N<=10,U = [];c = [];%return;%not enough data no calibration !!
30 % write the ellipsoid equation as D*p=0
31 % the best parameter is the solution of min||D*p|| with ||p||=1;
32 % form D matrix from X measurements
33 x = X(:,1); y = X(:,2); z = X(:,3);
34 D = [x.^2, y.^2, z.^2, x.*y, x.*z, y.*z, x, y, z, ones(N,1)];
35 D=triu(qr(D));%avoids to compute the svd of a large matrix
36 [U,S,V] = svd(D);%because usually N may be very large
37 p = V(:,end);if p(1)<0,p=-p;end;
38 % the following matrix A(p) must be positive definite
39 % The optimization done by svd does not include such a constraint
40 % With "good" data the constraint is always satisfied
41 % With too poor data A may fail to be positive definite
42 % In this case the calibration fails
43 %
44 A = [p(1) p(4)/2 p(5)/2;
45     p(4)/2 p(2) p(6)/2;
46     p(5)/2 p(6)/2 p(3)];
47 [U,ok] = fchol(m,A);
48 if ~ok,U = [];c = [];%calibration fails too poor data!!
49 b = [p(7);p(8);p(9)];
50 v = Utsolve(U,b/2,m);

```

```

51 d = p(10);
52 s = 1/sqrt(v*v'-d);
53 c =-Usolve(U,v,m)';%ellipsoid center
54 U = s*U;%shape ellipsoid parameter
55 %%%%%%%%%%%%%%
56 function [A,ok] = fchol(n,A)
57 % performs Cholesky factorization
58 A(1,1:n) = A(1,1:n)/sqrt(A(1,1));
59 A(2:n,1) = 0;
60 for j=2:n
61     A(j,j:n) = A(j,j:n) - A(1:j-1,j)'*A(1:j-1,j:n);
62     if A(j,j)<=0,ok=0;break;end%A is not positive definite
63     A(j,j:n) = A(j,j:n)/sqrt(A(j,j));
64     A(j+1:n,j) = 0;
65 end
66 ok=1;
67 function x=Utsolve(U,b,n)
68 % solves U'*x=b
69 x(1) = b(1)/U(1,1);
70 for k=2:n
71     x(k) = (b(k)-x(1:k-1)*U(1:k-1,k))/U(k,k);
72 end
73 function x=Usolve(U,b,n)
74 % solves U*x=b
75 x(n) = b(n)/U(n,n);
76 for k=n-1:-1:1
77     x(k) = (b(k)-U(k,k+1:n)*x(k+1:n)')/U(k,k);
78 end
79 %%%%%%%%%%%%%%

```

Listing 5: MgnCalibration.mk