МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Автоматики   
 (полное название кафедры)

Ивойлов А.Ю.

(И., О., фамилия студента – автора работы)

Разработка и исследование алгоритма работы системы стабилизации двухколесной

(полное название темы магистерской диссертации)

платформы на базе микроконтроллера STM32

**МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ**

по направлению высшего образования

27.04.04 Управление в технических системах

( код и наименование направления подготовки магистра)

Автоматики и вычислительной техники

(факультет)

Тема диссертации утверждена приказом по НГТУ № \_\_\_\_\_\_ от «\_\_\_\_» \_\_\_\_\_\_\_\_\_201\_\_ г.

***Примечание:*** *данные приказа об утверждении темы магистерской диссертации проставляются секрета-рём экзаменационной комиссии при приёме диссерта-ции к защите.*

##### Руководитель

Жмудь В.А.

(фамилия, И., О.)

зав.каф. АВТ, д.т.н., доцент

(уч. степень, уч. звание)

Новосибирск, 2015 г.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Автоматики   
 (полное название кафедры)

##### УТВЕРЖДАЮ

# Зав. кафедрой Жмудь В.А.

(фамилия, И., О.)

(подпись, дата)

**ЗАДАНИЕ**

**на магистерскую диссертацию**

студент *у(ке)* Ивойлову Андрею Юрьевичу   
 (фамилия, имя, отчество)

факультета автоматики и вычислительной техники

Направление подготовки 27.04.04 Управление в технических системах

(код и наименование направления подготовки магистра)

Магистерская программа Комплексные системы автоматизации

(наименование программы)

Тема Разработка и исследование алгоритма работы системы стабилизации

(полное название темы)

двухколесной платформы на базе микроконтроллера STM32

Цели работы 1. Разработать алгоритм работы системы стабилизации двухколесной

платформы

2. Получить и исследовать свойства математической модели платформы

3. Обеспечить отработку возмущений и начальных условий угла наклона в диапазоне ± 5о

4. Реализовать режимы работы платформы: балансировка, движение, вращение

# Руководитель

Жмудь В.А.

(фамилия, И., О.)

зав. каф. АВТ, д.т.н., доц.

(уч. степень, уч. звание)



(подпись, дата)

# Содержание

[Содержание 3](#_Toc422351803)

[Введение 5](#_Toc422351804)

[1. Обзор существующих решений 6](#_Toc422351805)

[2. Описание системы 9](#_Toc422351806)

[3. Постановка задачи 13](#_Toc422351807)

[4. Математическая модель системы 15](#_Toc422351808)

[4.1. Получение математической модели системы 15](#_Toc422351809)

[4.2. Анализ математической модели системы 19](#_Toc422351810)

[4.3. Описание процедуры синтеза регулятора 23](#_Toc422351811)

[5. Описание Электрической схемы устройства 26](#_Toc422351812)

[5.1. Описание работы электрической схемы платформы 26](#_Toc422351814)

[5.2. Описание компонентов схемы 28](#_Toc422351815)

[6. Управление сервомоторами 30](#_Toc422351816)

[6.1. Алгоритм получения данных от энкодера 30](#_Toc422351817)

[6.2. Управление напряжением якорной цепи сервомотора 31](#_Toc422351818)

[6.3. Получение значений параметров математической модели 32](#_Toc422351819)

[6.4. Оценка скорости вращения ротора сервомотора 33](#_Toc422351820)

[6.5. Получение значений конструктивных коэффициентов сервомотора 39](#_Toc422351821)

[6.6. Уточнение модели сервомотора 42](#_Toc422351822)

[7. Определение угла наклона 46](#_Toc422351823)

[7.1. Измерение угла наклона с помощью акселерометра 46](#_Toc422351824)

[7.2. Описание тестовой программы акселерометра ADXL335 47](#_Toc422351825)

[7.3. Анализ данных акселерометра 48](#_Toc422351826)

[7.4. Применение гироскопа для оценки угла наклона 50](#_Toc422351827)

[7.5. Подключение MPU6050 50](#_Toc422351828)

[7.6. Анализ данных гироскопа 54](#_Toc422351829)

[7.7. Описание принципа работы комплементарного фильтра 55](#_Toc422351830)

[8. Получение параметров математической модели 58](#_Toc422351831)

[8.1. Определение координат центра масс платформы 58](#_Toc422351832)

[8.2. Определение радиуса инерции платформы 59](#_Toc422351833)

[8.3. Проверка параметров математической модели 62](#_Toc422351834)

[9. Расчет и реализация регулятора 65](#_Toc422351835)

[9.1. Расчет коэффициентов регулятора 65](#_Toc422351836)

[9.2. Реализация регулятора 70](#_Toc422351837)

[9.3. Настройка коэффициентов регулятора 72](#_Toc422351838)

[9.4. Возможности системы управления в режиме балансировки 77](#_Toc422351839)

[9.5. Реализация режимов перемещения и вращения 81](#_Toc422351840)

[Заключение 85](#_Toc422351841)

[Список литературы 87](#_Toc422351842)

[Приложение 88](#_Toc422351843)

# Введение

Данная работа посвящена разработке двухколесной балансирующей платформы. Это устройство, представляющее собой платформу, на которой размещена управляющая электроника и датчики. К платформе крепятся статоры двух двигателей постоянного ток, валы которых приводят во вращения колеса. В процессе работы платформа поддерживает угол отклонения от вертикали близким к нулю путем балансировки, осуществляемой вращением колес. В рамках данной работы рассматриваются вопросы, связанный с разработкой алгоритма работы системы автоматической стабилизации платформы в вертикальном положении. Вопросы, связанные с разработкой аппаратной части и программного обеспечения платформы рассматриваются в работе Федорова Д.С. [1]. В данной работе используется часть результатов, представленных в работе [1]. В свою очередь, результаты, полученные в данной работе, применяются в работе [1].

С математической точки зрения двухколесная платформа является объектом управления, относящимся к классу объектов «Перевернутый маятник». В настоящее время задача стабилизации объектов этого типа обретает все большую актуальность, поскольку появляется все больше технических средств, математическая модель которых сводится к математической модели объекта «Перевернутый маятник». Среди них можно назвать транспортное средство «Segway», многоступенчатые антенны, а так же шагающие антропоморфные механизмы.

За последние годы развитие электроники привело к тому, что достаточно сложные системы могут быть реализованы на небольшом количестве микросхем. Наличие на рынке дешевых, но в то же время достаточно мощных в вычислительном аспекте микроконтроллеров позволяет реализовать управление системой на одной микросхеме. А появление технологии МЭМС-датчиков позволяет сравнительно просто оценивать такие параметры, как угол наклона объекта по отношению к вертикали с хорошей точностью. Данные обстоятельства упрощают реализацию систем автоматического управления.

# Обзор существующих решений

С точки зрения математики, рассматриваемая в данной работе система «Балансирующая двухколесная платформа» является объектом, относящимся к классу объектов «Перевернутый маятник». Задача стабилизации платформы в вертикальном положении является задачей стабилизации перевернутого маятника. Объект этого типа является нелинейным. В общем случае он имеет два положения равновесия – устойчивое нижнее и неустойчивое верхнее. В случае с двухколесной платформой существует только верхнее положение равновесия. К настоящему времени опубликовано множество работ, посвященных вопросу стабилизации перевернутого маятника.

Объекты типа «Перевернутый маятник» можно рассматривать как успешные лабораторные идеализации неустойчивых электромеханических систем [2]. К объектам данного типа относятся различные технические устройства. Среди них:

* Ракета при взлете;
* Солнечные батареи искусственных спутников [2];
* Многоступенчатые антенны;
* Звенья двуногих антропоморфных механизмов;
* Индивидуальное транспортное средство «Segway»;
* Летательные аппараты с центром давления, расположенным впереди центра масс [3].

Существуют различные виды объекта «Перевернутый маятник». По структуре их можно разделить на следующие типы:

* Перевернутый маятник с неподвижной точкой подвеса;
* Перевернутый маятник с точкой подвеса на каретке или тележке;
* Перевернутый маятник с точкой подвеса на колесе.

По количеству звеньев перевернутые маятники можно разделить на однозвенные и многозвенные. Рассматриваемый объект относится к типу «Однозвенный перевернутый маятник с точкой подвеса на колесе».

Управление объектами такого типа может осуществляться по различными алгоритмам. Различны и подходы к синтезу систем управления перевернутого маятника. Один из вариантов заключается в линеаризации математической модели вблизи положения равновесия и организации закона управления как для линейного объекта. Однако результаты в этом случае не всегда достаточно хороши. Среди систем управления такого типа наиболее распространены ПИД-регуляторы и линейно-квадратичные регуляторы. Другой подход заключается в синтезе системы стабилизации на основе методов синтеза нелинейных систем. Примером такой системы может служить система стабилизации объекта «маятник-каретка» на основе скользящего режима, описанная в работе [4].

Математические модели объекта «маятник-каретка» и объекта «перевернутый маятник на колесе», имеют сходный вид. Однако есть существенное отличие. В работе [4] показано, что в модели объекта «маятник-каретка» можно выделить две подсистемы – подсистема «маятник» и подсистема «каретка». При этом, управляющим воздействием в подсистеме «маятник» является ускорение каретки. Так же, в системе присутствуют вырожденные движения, и подсистема этих движений соответствует подсистеме «каретка», которая находится на границе устойчивости. Это обстоятельство ограничивает применения методов синтеза, основанных на линейной модели для объектов этого типа. Для синтеза системы управления предложена специальная методика, основанная на методе скользящих режимов.

В отличие от объекта «маятник-каретка», в объекте «перевернутый маятник на колесе» в случае, когда в качестве исполнительного механизма в системе используется двигатель постоянного тока, нельзя в явном виде выделить уравнение, описывающее только колебания маятника [3]. То есть, в системе нельзя явно выделить подсистему, связанную с маятником, и подсистему, связанную с колесом. Связано это с тем, что статоры электродвигателя соединены с платформой, а значит, помимо ускорения колеса на маятник действует крутящий момент двигателя. С одной стороны это несколько усложняет анализ свойств модели, поскольку нельзя отдельно рассматривать подсистемы «маятник» и «колесо». С другой стороны, с точки зрения управления ситуация упрощается, поскольку ресурс управления будет выше, за счет прямого действия крутящего момента исполнительного механизма на маятник, а не только через ускорение точки подвеса, как в случае с объектом «каретка-маятник». Это дает возможность получить приемлемые результаты работы системы стабилизации, реализованной на основе синтеза системы управления одним из линейных методов. Наиболее перспективной выглядит система управления на основе модального метода синтеза, поскольку существует возможность измерять напрямую часть переменных состояния с помощью специальных датчиков. Это не отменяет возможности синтеза системы на основе нелинейного метода, и в частности, на основе скользящего режима, однако методика синтеза, предложенная в работе [4] не может быть применена напрямую для объекта такого типа.

# Описание системы

Устройство представляет собой платформу, на которой размещена плата управления с датчиками и процессорным устройством на базе микроконтроллера. К платформе жестко крепятся статоры сервомоторов, на валах которых закреплены колеса. Внешний вид платформы представлен на рисунке 2.1. Управляемыми параметрами системы являются угол наклона платформы по отношению к вертикали и угол поворота колеса относительно начального положения. Управляющим воздействием является напряжение, прикладываемое к якорной цепи сервомоторов.

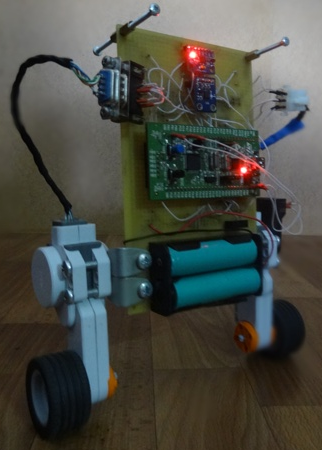


Рис.2.1 Внешний вид двухколесной платформы.

С математической точки зрения двухколесная платформа представляет собой объект вида «перевернутый маятник с точкой подвеса на оси колеса». Задачей работы системы стабилизации платформы является удержание её в неустойчивом верхнем положении равновесия. При этом диапазон изменения угла поворота колеса, а значит и расстояния, которое проходит платформа в процессе работы, ограничен.

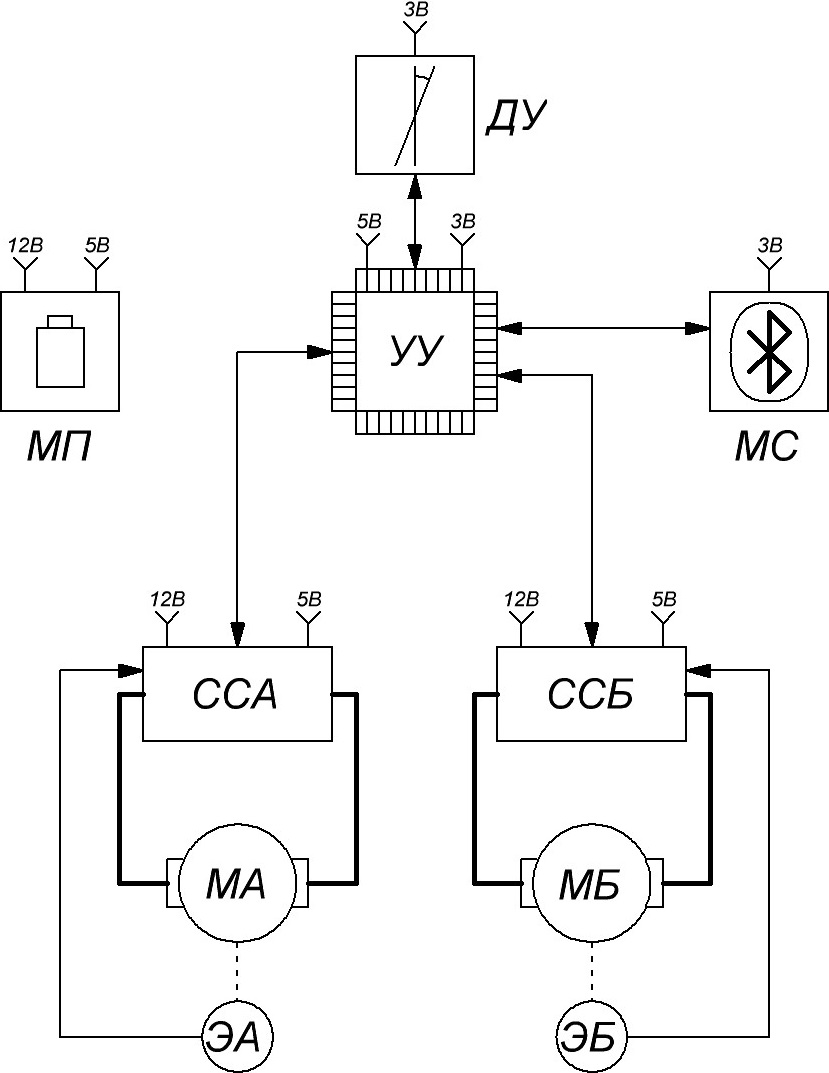


Рис.2.2 Структурная схема двухколесной платформы. ДУ – датчик угла наклона, МП – модуль питания, УУ – устройство управления, МС – модуль связи, ССА – схема сопряжения с сервомотором А, ССБ – схема сопряжения с сервомотором Б, МА – сервомотор А, МБ – сервомотор Б, ЭА – энкодер сервомотора А, ЭБ – энкодер сервомотора Б.

Структурная схема устройства представлена на рисунке 2.2. Платформа включает в себя следующие основные составляющие:

* Датчик угла наклона платформы;
* Исполнительные механизмы;
* Датчики угла поворота колес;
* Устройство связи;
* Устройство управления;
* Схема питания.

В качестве датчика угла наклона могут быть использованы МЭМС-датчики, такие как гироскоп или акселерометр. Каждый их этих датчиков имеет свои преимущества и недостатки. Акселерометр подвержен влиянию сил, действующих в горизонтальном направлении и искажающих данные. Так же сигнал акселерометра содержит высокочастотную помеху. Гироскоп имеет характерную особенность, называемую дрейфом нуля. Дрейф нуля приводит к тому, что с течением времени значение угла, восстанавливаемого по показаниям гироскопа, увеличивается. Поэтому, для получения качественной оценки угла наклона необходимо применять специальные методы, которые снижают влияния недостатков датчика на итоговый результат оценки. В частности, может быть использована связка из двух датчиков – гироскопа и акселерометра.

Исполнительными механизмами системы являются сервомоторы. Для управления напряжением якорной цепи сервомотора, используется схема сопряжения на основе специальных микросхем драйверов. Входными сигналами схемы являются сигналы от микроконтроллера, на выходе схемы генерируется напряжение нужной формы и амплитуды, которое прикладывается к якорной цепи сервомотора.

В качестве датчиков угла поворота используются энкодеры, встроенные в сервомоторы. Сигналы энкодера поступают через схему сопряжения с двигателем на вход микроконтроллера, который осуществляет обработку этих сигналов.

Устройство связи необходимо для получения информации о работе системы для ее отладки, а также отправки команд устройству. В работе используется интерфейс UART. К устройству связи может быть подключен Bluetooth-модуль для беспроводной передачи данных, например HC05, или преобразователь USB-TTL для проводного соединения с компьютером. Так же, устройство связи содержит устройство, осуществляющее прием сигнала по радиоканалу на частоте 2.4ГГц от пульта управления.

Ядром системы является отладочная плата STM32VLDISCOVERY с установленным на ней микроконтроллером STM32F100RBT6B. Программа микроконтроллера осуществляет прием и обработку данных от датчиков, расчет управляющего воздействия в соответствии с алгоритмом управления, а так же прием и передачу данных.

Схема питания позволяет получить необходимые уровни напряжения питания для силовой и цифровой части платформы. Входное напряжение от двух последовательно соединенных аккумуляторов 8.4В преобразуется в напряжение 12В, от которого питается силовая часть схемы (сервомоторы и схема сопряжения). Так же входное напряжение понижается до уровня 5В, необходимого для питания платы STM32VLDISCOVERY. На самой плате установлен линейный регулятор напряжения, позволяющий получить напряжение питания 3В. Питание от аккумуляторов обеспечивает автономность работы платформы.

Платформа может функционировать в нескольких режимах. В режиме балансировки, платформа поддерживает угол отклонения от вертикали близким к нулю путем балансировки. В режиме перемещения платформа поддерживает близкое к нулю отклонение угла наклона от вертикали в процессе движения вперед или назад. В режиме вращения близкое к нулю значение угла наклона поддерживается при повороте влево или вправо. Во всех режимах работы система стабилизации должна отрабатывать возмущающие воздействия, вызванные действием внешних сил или неровностями поверхности, по которой перемещается платформа.

# Постановка задачи

В рамках данной работы требуется разработать алгоритм работы системы стабилизации двухколесной платформы, который должен обеспечивать параметры работы устройства, описанные ниже.

Платформа должна функционировать в трех режимах работы:

* Режим балансировки
* Режим перемещения
* Режим вращения

Во всех режимах требуется обеспечить устойчивость платформы в верхнем положении равновесия, поддерживая угол отклонения платформы от вертикали близким к нулю. Система стабилизации должна обеспечивать отработку начальных условий и возмущений по углу отклонения от вертикали в диапазоне ± 5о. При этом максимальное расстояние, проходимое платформой в процессе отработки возмущений, не должно превышать 0.2м. Так же, система должна работать от автономного источника питания и обеспечивать передачу данных на ПК по беспроводному интерфейсу.

В рамках данной работы требуется решить следующие задачи:

1. Получить математическую модель системы и исследовать ее свойства;
2. Разработать алгоритм оценки скорости вращения вала сервомотора;
3. Получить численные значения параметров модели сервомотора, уточнить его математическую модель;
4. Провести анализ показаний датчиков, выбрать метод оценки угла наклона;
5. Получить численные значения параметров математической модели системы;
6. Рассчитать и реализовать систему автоматической стабилизации на микроконтроллере;
7. Реализовать алгоритм работы системы в режимах перемещения и вращения.

Требования к аппаратной части устройства:

* Для разработки использовать отладочную плату STM32VLDISCOVERY с микроконтроллером STM32F100RBT6B;
* В качестве электродвигателей использовать сервомоторы Lego NXT, включающие в себя энкодеры для определения угла поворота колеса;
* Для дистанционного управления использовать систему **HobbyKing 2.4Ghz 6Ch Tx & Rx V2;**
* В качестве устройства, обеспечивающего связь с ПК, использовать Bluetooth модуль HC-05;
* Схема питания должна работать на основе литиево-ионных аккумуляторов и обеспечивать на выходе стабильное напряжение 5В и 12В.

Требования к программной части устройства:

* Использовать язык программирования С++ и среду разработки Eclipse;
* Не использовать операции с числами с плавающей запятой.

# Математическая модель системы

## Получение математической модели системы

Необходимым этапом процесса разработки системы стабилизации любого объекта является получение и исследование его математической модели. Важно иметь представление о том, каков порядок системы, какие параметры влияют на выходную величину и каким образом. Так же, необходимо убедиться в том, что задача синтеза регулятора разрешима.

С математической точки зрения двухколесная платформа является объектом вида «перевернутый маятник» с точкой подвеса на оси колес. Схема данного вида объекта приведена на рисунке 4.1.

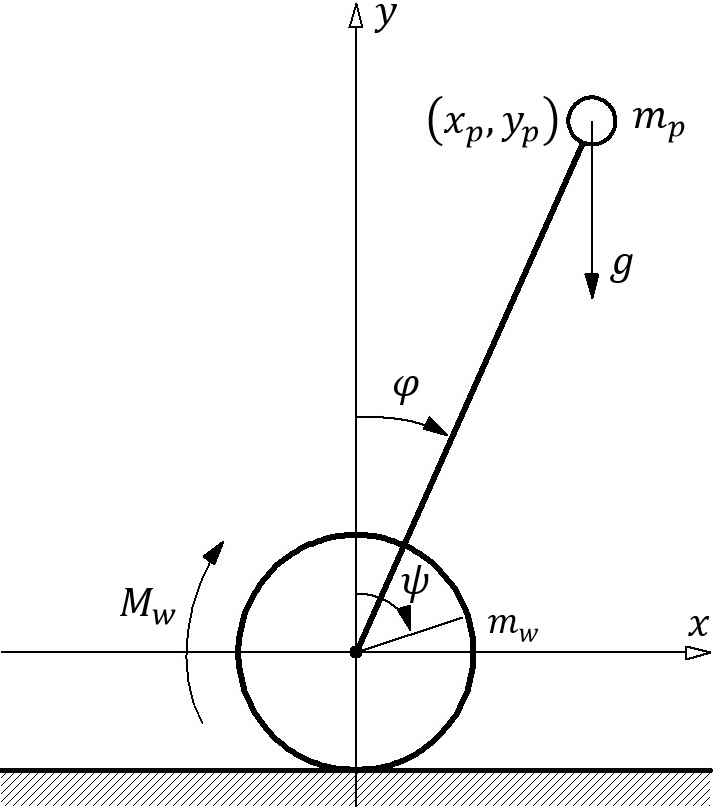


Рис.4.1 Схема объекта вида «перевернутый маятник» с точкой подвеса на колесе.

Здесь – угол отклонения стержня маятника относительно вертикали, – угол поворота колеса относительно начального положения, – масса стержня маятника, – масса колеса, – момент, действующий на колесо со стороны исполнительного механизма (например двигателя постоянного тока), – координаты центра масс стержня маятника.

Математическая модель такого объекта может быть получена на основе уравнения Лагранжа второго рода [3, c.27]. Запишем уравнение Лагранжа второго рода:

|  |  |
| --- | --- |
|  | (4.1) |

Здесь *T* – кинетическая энергия системы, – обобщенные координаты, – обобщенные моменты. В качестве обобщенных координат выберем угол наклона стержня маятника относительно вертикали и угол поворота колеса относительно начального положения:

Тогда соотношения для обобщенных моментов будут иметь следующий вид:

|  |  |
| --- | --- |
|  | (4.2) |
|  | (4.3) |

Здесь *L* – расстояние от центра масс стержня до точки подвеса маятника.

Кинетическая энергия системы является суммой кинетической энергии стержня маятника и колес. Энергия стержня маятника включает в себя энергию поступательного движения центра масс системы и энергию вращательного движения его относительно точки подвеса:

|  |  |
| --- | --- |
|  | (4.4) |

где – скорость центра масс стержня, – угловая скорость стержня, – радиус инерции стержня маятника относительно точки подвеса. Линейная скорость центра масс стержня определяется выражением:

|  |  |
| --- | --- |
|  | (4.5) |

Координаты центра масс стержня и точки подвеса связаны соотношениями:

Поскольку в выбранной системе отсчета (см. рис.4.1), выражения для производных координат центра масс стержня примут вид:

Подставим данные выражения в (4.5):

C учетом того, что и, где *R* – радиус колеса:

|  |  |
| --- | --- |
|  | (4.6) |

Подставив (4.6) в (4.4) с учетом того, что, получим окончательное выражение для кинетической энергии стержня маятника:

|  |  |
| --- | --- |
|  | (4.7) |

Кинетическая энергия колеса включает энергию поступательного и вращательного движения:

где – радиус инерции колеса. С учетом того, что,, получим:

|  |  |
| --- | --- |
|  | (4.8) |

С учетом (4.7) и (4.8) выражение для кинетической энергии системы примет вид:

|  |  |
| --- | --- |
|  | (4.9) |

Для получения дифференциальных уравнений, описывающих поведение системы, необходимо продифференцировать выражение (4.9) по обобщенным координатам и их производным в соответствии с (4.1). Для получения первого дифференциального уравнения найдем производные и.

|  |  |
| --- | --- |
|  | (4.10) |
|  | (4.11) |

Для получения второго дифференциального найдем производные и.

|  |  |
| --- | --- |
|  | (4.12) |
|  | (4.13) |

Подставив (4.2), (4.3), (4.10), (4.11), (4.12) и (4.13) в (4.1), получим систему дифференциальных уравнений, описывающую поведение платформы:

|  |  |
| --- | --- |
|  | (4.14) |

После того, как получены соотношения, описывающие систему, необходимо получить выражение для момента, действующего со стороны исполнительных механизмов. В нашем случае таковыми являются двигатели постоянного тока. Поведение двигателя постоянного тока приближенно описывают следующие соотношения:

|  |  |
| --- | --- |
|  | (4.15) |
|  | (4.16) |
|  | (4.17) |

Здесь, – угловые скорость и ускорение ротора двигателя относительно статора,,, – напряжение, ток и сопротивление якорной цепи двигателя, , – конструктивные коэффициенты, – момент инерции вала двигателя, – момент, создаваемый двигателем, – нагрузочный момент, приложенный к валу. Подставим (4.15) и (4.16) в (4.17) и разрешим полученное выражение относительно нагрузочного момента:

|  |  |
| --- | --- |
|  | (4.18) |

Поскольку ротор двигателя соединен с колесом, а статор – со стержнем маятника (платформой), справедливы следующие соотношения:

|  |  |
| --- | --- |
|  | (4.19) |
|  | (4.20) |

Момент, действующий со стороны двигателя на стержень маятника и колесо является нагрузочным моментом для двигателя. Подставив (4.19) и (4.20) в (4.18), и затем (4.18) в (4.14) получим итоговые соотношения, описывающие поведение системы:

|  |  |
| --- | --- |
|  | (4.21) |

## Анализ математической модели системы

Для упрощения анализа поведения системы выполним линеаризацию математической модели при углах отклонения от вертикали близких к нулю. Данное действие допустимо, поскольку выполнение условия является обязательным во всех режимах работы системы, а рабочий диапазон угла наклона составляет ± 5о. Воспользуемся следующими соотношениями:

Тогда система (4.21) примет вид:

|  |  |
| --- | --- |
|  | (4.22) |

Запишем систему (4.22) в векторно-матричной форме. Для этого расширим систему двумя уравнениями:

В качестве переменных состояния выберем следующие величины:

Здесь – угловая скорость ротора сервомотора относительно статора, – угол поворота ротора относительно статора. Управляющим воздействием является напряжение якорной цепи двигателя:

Тогда система (4.22) будет иметь вид:

или кратко:

|  |  |
| --- | --- |
|  | (4.23) |

Здесь , , , , ,,, , – коэффициенты, выражения для которых приведены в таблице 4.1.

Таблица 4.1

Коэффициенты системы (4.23)

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

Чтобы перейти к стандартному описанию, умножим систему (4.23) слева на . Получим следующие соотношения:

|  |  |
| --- | --- |
|  | (4.24) |

Соотношения для коэффициентов матриц *A* и *B* приведены в таблице 4.2.

Таблица 4.2

Коэффициенты системы (4.24)

|  |  |
| --- | --- |
| *A* | *B* |
|  |  |

Для определения устойчивости системы запишем ее характеристическое уравнение.

где

Поскольку коэффициенты положительны, часть коэффициентов полинома отрицательны. Следовательно, система является неустойчивой (как и предполагалось).

Определим множество равновесных состояний системы. Для этого приравняем производные в системе (4.24) к нулю. Это соответствует состоянию системы после завершения переходных процессов. Тогда:

– произвольное.

Таким образом, равновесное состояние системы с нулевым углом отклонения от вертикали может быть достигнуто при любом угле поворота колес.

Проведем анализ управляемости системы. Для этого составим матрицу управляемости. Выражение для нее имеет следующий вид:

Критерием проверки управляемости для одноканального объекта является выполнение условия [5]. Матрица управляемости не приведена здесь из-за громоздкости выражений ее коэффициентов, однако расчетным путем установлено, что ее определитель не равен нулю, а значит, объект управления является полностью управляемым.

Выполним анализ наблюдаемости системы. Для этого составим матрицу наблюдаемости. Выражение для нее имеет следующий вид:

Поскольку прямому измерению доступны параметры *φ* и *α*, матрица выхода *C* будет иметь вид:

Критерием проверки наблюдаемости для объекта является выполнение условия [5]. Матрица наблюдаемости так же на приведена здесь из-за громоздкости выражений ее коэффициентов. Расчетным путем установлено, что. Следовательно, объект управления является полностью наблюдаемым.

## Описание процедуры синтеза регулятора

Для расчета регулятора был применен модальный метод синтеза. Целью работы системы стабилизации являются поддержание значения угла наклона платформы близким к нулю. При этом, угол поворота колеса не может неограниченно возрастать. Таким образом, постановка задачи синтеза выглядит следующим образом. Для объекта (4.24) разработать систему автоматического управления, реализующую выполнение требования

при условиях:

Поскольку состояния системы с углом отклонения от вертикали, не равным нулю, не входят в множество равновесных состояний системы, система должна быть астатической:

Условиями разрешимости задачи синтеза являются:

* Полная управляемость или устойчивость неуправляемой части объекта;
* Полная наблюдаемость или устойчивость ненаблюдаемой части объекта;
* Наличие достаточного ресурса управления.

Анализ управляемости и наблюдаемости, проведенный в п.4.2 показал, что объект является полностью управляемым и полностью наблюдаемым, следовательно, первые два условия разрешимости задачи синтеза выполняются. Проверка ресурсного ограничения может быть осуществлена только после получения численных значений параметров математической модели. В разделе 8 показано, что система имеет достаточный ресурс управления. Таким образом, задача синтеза является разрешимой.

В соответствие с выбранным методом закон управления имеет следующий вид:

где *K* – матрица коэффициентов регулятора. Здесь параметры матрицы *K* соответствуют коэффициентам в обратной связи при следующих величинах:

* – коэффициент при производной угла наклона
* – коэффициент при угле наклона
* – коэффициент при угловой скорости вала сервомотора
* – коэффициент при угле поворота вала сервомотора

Определим характеристическое уравнение системы с регулятором. Для этого воспользуемся следующим соотношением:

Выражения для коэффициентов при степенях оператора *p* приведены в таблице 4.3.

Таблица 4.3

Коэффициенты характеристического уравнения замкнутой системы

|  |  |
| --- | --- |
|  | . |
|  | ;  ;  . |
|  | ; ;  ; . |
|  | ; ; . |

Желаемое уравнение динамики задается в следующем виде:

Значения коэффициентов определяется распределением корней системы. Для нахождения коэффициентов регулятора необходимо приравнять коэффициенты при степенях оператора p характеристического уравнения к соответствующим коэффициентам желаемого уравнения. Итоговые расчетные соотношения для коэффициентов регулятора приведены ниже.

|  |  |
| --- | --- |
| ;  ;  ;  ; | (4.25) |

# Описание Электрической схемы устройства

## Подробно разработка и расчет принципиальной схемы устройства приведен в работе [1]. В данной работе приведено лишь описание работы схемы и некоторые особенности, связанные с ее реализацией.

## Описание работы электрической схемы платформы

Принципиальная схема устройства приведена в приложении А. В соответствии со структурной схемой устройства электрическая схема включает следующие составляющие:

* Устройство управления;
* Схема подключения датчика угла наклона платформы;
* Схема сопряжения с сервомоторами;
* Схема подключения устройства связи;
* Схема питания.

Управляющим устройством в схеме является отладочная плата STM32VLDISCOVERY с микроконтроллером STM32F100RBT6B. Данная плата питается от напряжения 5В и имеет встроенный программатор-отладчик для загрузки программы в микроконтроллер и ее отладки. Так же, плата содержит линейный регулятор напряжения 3В с током нагрузки до 1.1А. Выводы микроконтроллера выведены на штыревые линейки, установленные на плате, что позволяет легко подключать устройства к данной плате. Рабочая частота микроконтроллера составляет 24МГц. Технические характеристики платы и микроконтроллера приведены в п.5.2.

В качестве датчика угла наклона используется измерительный модуль MPU6050. Данный модуль включает в себя трехосевые акселерометр, гироскоп, а так же датчик температуры. Модуль имеет цифровой выход. Связь с ним производится по интерфейсу I2C. Данный интерфейс требует внешних подтягивающих к питанию резисторов. Данные резисторы уже установлены на плате MPU6050. Таким образом, никаких специальных схем для подключения модуля к микроконтроллеру не требуется. В схеме так же заложена возможность подключения трехосевого акселерометра ADXL335, имеющего аналоговый выход. Данный датчик так же не требует специальной схемы подключения.

В качестве исполнительных механизмов в системе используются два сервомотора NXT Lego Mindstorm. Данный сервомотор конструктивно включает так же датчик угла поворота вала – квадратурный энкодер. Для управления напряжением якорной цепи сервомотора был выбран способ широтно-импульсной модуляции. Управление реализуется с помощью микросхем драйверов BA6222. Логические входы драйвера по уровню напряжения на совпадают с уровнем микроконтроллера. Поэтому в схему сопряжения с сервомотором включена схема согласования, реализованная на транзисторах ВС847A. Уровни сигналов энкодера так же не совпадают с уровнем микроконтроллера, для согласования уровней в схему сопряжения с сервомотором был включен резистивный делитель. Для генерирования сигналов, подаваемых на драйвер, и обработки сигналов, принимаемых от энкодера, был задействован встроенный в микроконтроллер четырехканальный таймер с возможностью захвата фронтов сигналов на входе и генерирования ШИМ-сигнала на выходе.

В качестве устройства связи в схему может быть включен Bluetooth-модуль HC-05 или преобразователь напряжения USB-TTL для связи с ПК. Передача данных осуществляется по интерфейсу UART, частота передачи – до 115200 бит/с. Для подключения этих устройств не требуется специальной схемы сопряжения. Так же схема содержит приемный модуль пульта дистанционного управления HK-TR6A. Передача данных осуществляется по радиоканалу на частоте 2.4ГГц.

Схема питания устройства преобразует входное напряжения литий-ионных аккумуляторов 8.4В в напряжения питания цифровой и силовой части схем 5В и 12В соответственно. Для понижения напряжения до 5В применяется линейный регулятор напряжения L7805AB. Для питания устройств, требующих напряжения 3В используется выход линейного регулятора, установленного на плате STM32VLDISCOVERY. Для повышения входного напряжения до 12В используется модуль питания на базе микросхемы повышающего DC/DC преобразователя LM2577S. Данный уровень необходим для работы микросхем драйверов BA6222.

## Описание компонентов схемы

* + 1. Плата STM32VLDISCOVERY

Плата включает следующие компоненты:

* Микросхема загрузки и отладки ST-Link;
* USB-разъем;
* Вывод для внешнего источника питания 5В;
* Кнопка сброса;
* Пользовательская кнопка;
* Пользовательские светодиоды (зеленый и синий);
* Индикационные светодиоды (индикатор питания, индикатор связи с ПК;
* Кварцевый резонатор.
  + 1. Микроконтроллер STM32F100RBT6B

32х-разрядный микроконтроллер фирмы STMicroelectronics.

Основные характеристики:

* Ядро: ARM 32-bit Cortex-M3 CPU;
* Максимальная частота: 24МГц;
* Память: 128 кБ – flash, 8 кБ – SRAM;
* Периферия: таймеры, АЦП, ЦАП, интерфейсы ввода-вывода SPI, USART, I2C;
* 3 16-выводных порта ввода-вывода;
* Разрядность АЦП и ЦАП: 12 разрядов;
* Встроенный датчик температуры.

Электрические параметры:

* Напряжение питания: 2,0 – 3,6В;
* Максимальный потребляемый ток: 150мА;
* Температурный диапазон: -40 – +85 оС;
* Максимальный ток вывода порта: 25мА;
* Максимальный суммарный ток всех выводов контроллера: 25мА;
* Максимальная потребляемая мощность: 444мВт.
  + 1. Измерительный модуль MPU6050

Измерительная система включает 2 МЭМС-датчика – акселерометр и гироскоп. Технические характеристики модуля:

Электрические характеристики:

* Напряжение питания 2.375V-3.46V;
* Потребление тока в режиме работы – 3,6 мА;
* Потребление тока в режиме ожидания – 5 мкА

Типы сенсоров:

* Трехосевой гироскоп;
* Трехосевой акселерометр;
* Термодатчик;

Прочие параметры

* Интерфейс I2C для записи и чтения регистров, работающий на частоте до 400кГц;
* FIFO-буфер объемом 1024 байт;
* Программируемые пользователем цифровые фильтры для гироскопа, акселерометра и термодатчика;
* Программируемый пользователем диапазон измерений: ±250, ±500, ±1000, и ±2000 °/с;
* Встроенный 16-разрядный АЦП.

# Управление сервомоторами

Данный раздел работы посвящен вопросам, связанным с использованием сервомоторов NXT Lego Mindstorm. Данные сервомоторы применяются в качестве исполнительных механизмов системы. Внешний вид сервомотора приведен на рисунке 6.1. В сервомотор встроены квадратурный энкодер и редуктор. Энкодер служит для оценки угла поворота ротора двигателя.



Рис.6.1 Сервомотор NXT Lego Mindstorm.

Для реализации управления сервомоторами необходимо решить следующие задачи:

1. Получение данных от энкодера двигателя;
2. Оценка скорости вращения ротора сервомотора по данным энкодера;
3. Управление напряжением якорной цепи сервомотора;
4. Получение параметров математической модели сервомотора;
5. Разработка тестовой программы управления сервомотором.

В рамках данной работы решались задачи 2, 4 и 5. Задачи 1 и 3 рассмотрены в работе [1], в данной работе приведено лишь краткое описание результатов по этим задачам.

## Алгоритм получения данных от энкодера

Для получения данных об угле поворота ротора и оценки скорости его вращения применяется встроенный в сервомотор квадратурный энкодер. Энкодер имеет два выходных канала: канал A и канал B. Диаграммы сигналов в каналах приведены на рисунке 6.2.

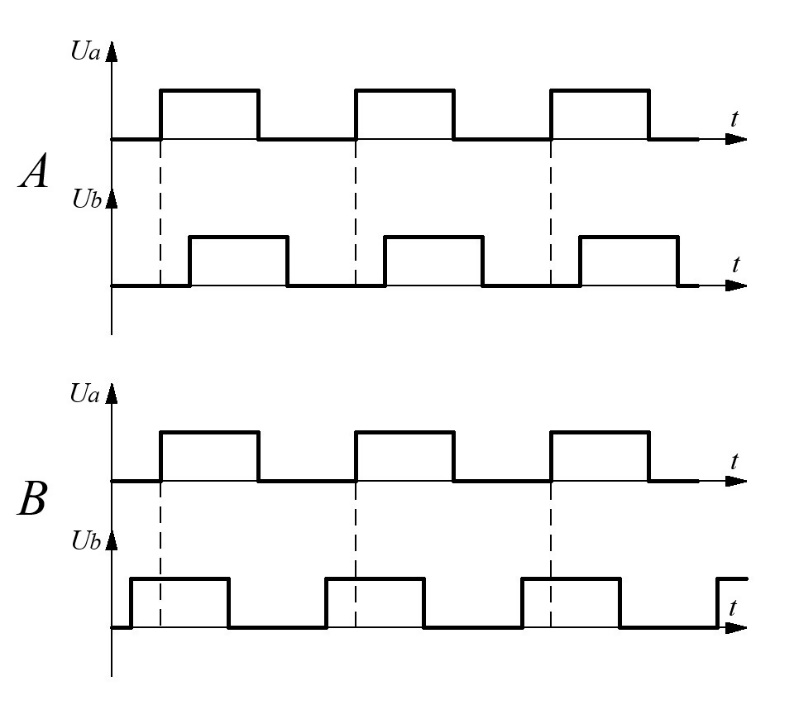


Рис.6.2 Диаграмма сигналов энкодера. A – вращение по часовой стрелке, B – вращение против часовой стрелки

Для определения угла поворота ротора достаточно считать количество фронтов сигналов с канала. Поскольку время импульса равно времени паузы, а общее число импульсов на один оборот равно 180, каждому фронту сигнала соответствует поворот на 1 градус. Анализируя, сигнал какого канала в данный момент времени опережает сигнал другого, можно определить направление вращения ротора двигателя (см. рис.6.2).

## Управление напряжением якорной цепи сервомотора

Управление сервомотором осуществляется с помощью широтно-импульсной модуляции напряжения, подаваемого на двигатель. На вход двигателя подаются прямоугольные импульсы постоянной частоты, но различной скважности. При достаточно высокой частоте импульсов ток в якорной цепи и, следовательно, момент будут меняться плавно благодаря индуктивности обмоток электродвигателя. При этом изменение скважности входного сигнала, имеющего постоянную амплитуду, будет эквивалентно изменению амплитуды входного непрерывного сигнала. Управление осуществляется с помощью микросхемы драйверов BA6222.

## Получение значений параметров математической модели

Для работы двигателя в соответствии с алгоритмом управления системы необходимо получить его математическую модель. Приближенно поведение двигателя постоянного тока описывается уравнением

|  |  |
| --- | --- |
|  | (6.1) |

Здесь – угловая скорость ротора двигателя относительно статора,, – напряжение и сопротивление якорной цепи двигателя, , – конструктивные коэффициенты, – момент инерции вала двигателя, – нагрузочный момент, приложенный к валу двигателя. В режиме холостого хода при напряжении на входе двигателя решением дифференциального уравнения (6.1) является функция:

|  |  |
| --- | --- |
|  | (6.2) |

где, . Если проинтегрировать это уравнение на интервале от 0 до t, получим выражение для угла поворота ротора двигателя:

|  |  |
| --- | --- |
|  | (6.3) |

Как видно из уравнения (6.3), с течением времени угол поворота приближается к прямой, описываемой уравнением

|  |  |
| --- | --- |
|  | (6.4) |

Для получения математической модели двигателя были проведены измерения угла поворота ротора при напряжении *U0 = 5В*. Результаты измерения угла поворота двигателя представлены на рисунке 6.3. Точки на линейном участке графика были аппроксимированы прямой, по коэффициентам которой были определены параметры модели:

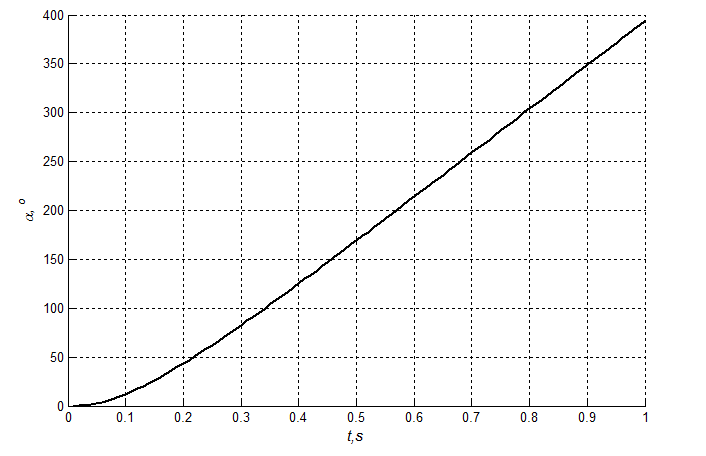


Рис.6.3 Переходный процесс угла поворота вала сервомотора при приложении напряжения *U0 = 5В.*

## Оценка скорости вращения ротора сервомотора

Для работы системы в соответствии с алгоритмом, описанным в разделе 4, необходимо иметь оценку угла поворота и скорости вращения вала сервомотора в текущий момент времени. Задача определения угла поворота решается сравнительно просто (см.п.6.1). Однако скорость вращения недоступна для прямых измерений. Для оценки ее значения может быть применен дифференцирующий фильтр.

Дифференцирующий фильтр – это устройство, позволяющее получить оценку фильтруемой величины и n её производных. Помимо оценки производных входного сигнала, в процессе работы фильтр должен так же подавлять высокочастотную помеху, которую сдержит входной сигнал, поскольку при дифференцировании высокочастотная составляющая сигнала усиливается. Невыполнение данного условия может привести к тому, что соотношение сигнал/шум оценки производных будет неудовлетворительным. Исходя из данного требования для получения оценки скорости вращения ротора сервомотора, то есть первой производной угла поворота ротора, был применен фильтр второго порядка. Структурная схема фильтра приведена на рисунке 6.4.

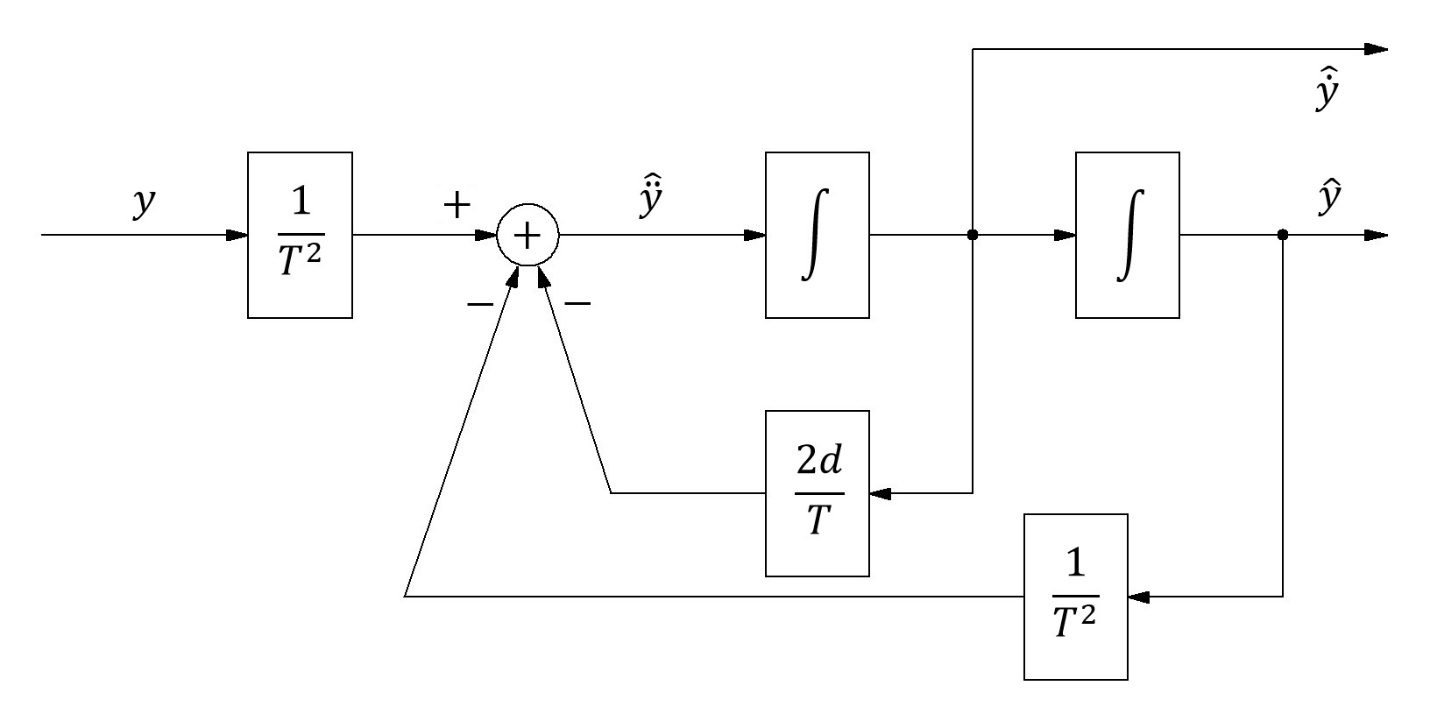


Рис.6.4 Структурная схема дифференцирующего фильтра второго порядка.

Данной структурной схеме соответствует дифференциальное уравнение:

Здесь – входная величина, – оценка входной величины, – оценка первой производной, – оценка второй производной. Характер процессов в фильтре определяется параметрами *T* и *d*. Коэффициент демпфирования *d* определяет степень колебательности процессов в фильтре, постоянная времени *T* – их длительность. Для корректной работы фильтра необходимо правильно выбрать данные значения. Примем значение параметра. При таком значении колебательность процессов в фильтре отсутствует. Для выбора значения постоянной времени воспользуемся результатами, полученными в п.6.3. Полученное значение постоянной времени переходного процесса двигателя составляет 0.12 с. Для корректной работы фильтра необходимо, чтобы его постоянная времени была много меньше постоянной времени объекта. Однако, следует помнить, что чем меньше значение постоянной времени фильтра, тем хуже эффект от его применения. Таким образом, значение постоянной времени должно быть достаточно малым, чтобы задержка выходного сигнала фильтра была приемлемой, но в то же время достаточно большим, чтобы помехи выходного сигнала успешно сглаживались. На основании вышеизложенного примем значение постоянной времени c. Теперь необходимо рассмотреть особенности, связанные с реализацией дифференцирующего фильтра на микроконтроллере.

Наиболее простой способ реализации дифференцирующего фильтра второго порядка со структурой, описанной выше, состоит в замене операции интегрирования операцией суммирования:

Здесь– значение фильтруемого параметра на предыдущем шаге, – значение на текущем, – входной параметр интегратора (производная входного сигнала) на текущем шаге, – шаг интегрирования. Фильтр второго порядка может быть реализован последовательным выполнением следующих операций:

Здесь – угол поворота ротора, – оценка угла поворота ротора, – оценка скорости вращения ротора, – оценка углового ускорения. В отличие от теоретической модели фильтра, в фильтре, реализованном на микроконтроллере, появляется еще один параметр, значение которого необходимо выбрать – шаг интегрирования. Для корректной работы описанного алгоритма необходимо, чтобы шаг интегрирования был меньше наименьшего параметр фильтра, то есть – постоянной времени. Выберем значение шага интегрирования с.

При реализации фильтра на многих микроконтроллерах существует еще одна особенность. Связана она с тем, что во многих случаях микроконтроллер может работать только с целыми числами. Поэтому во избежание потерь данных при делении, необходимо ввести в алгоритм дополнительный параметр, который позволит сохранить точность. Скорректированный алгоритм приведен ниже:

Здесь – коэффициент точности. Следует помнить, что при таком варианте алгоритма значения оценок входной величины и производных будут больше истинных в раз, что необходимо учитывать при использовании данных величин в алгоритме управления. Значение параметра удобно выбирать кратным степени двойки. В этом случае операции умножения и деления можно заменить операциями сдвига, что увеличит быстродействие.

При работе с целочисленной арифметикой появляется такой параметр, как шаг квантования оценки входного сигнала. Причина появления этого параметра состоит в том, что при операциях с целыми числами приращение оценки при интегрировании ограниченно снизу значением 1. При значениях входной величины цифрового интегратора меньше 1 приращение выходной величины равно 0. Эта особенность приводит к тому, что при постоянном значении входной величины, начиная с некоторого момента времени, приращение оценки входного сигнала становится равным нулю, и дальнейшее приближение значений оценки к значениям входной величины невозможно. На рисунках 6.5 и 6.6 приведены графики процессов в фильтре при оценке ступенчатого сигнала при различных значениях параметра. Как видно из рисунка 6.6, при малом значении оценка входного сигнала приближается к истинному значению на величину не более 6%. Аналогичен результат и для оценки производной входного сигнала.

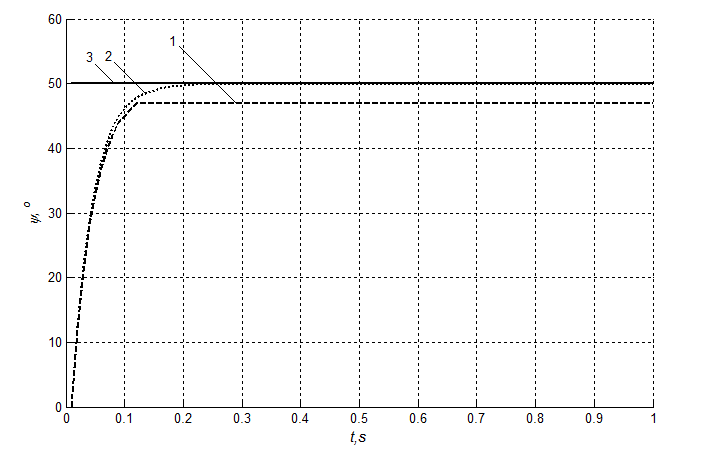


Рис.6.5 Влияние шага квантования на процессы в фильтре. Входная величина и ее оценка. 1 – оценка при, 2 – оценка при, 3 – входной сигнал.

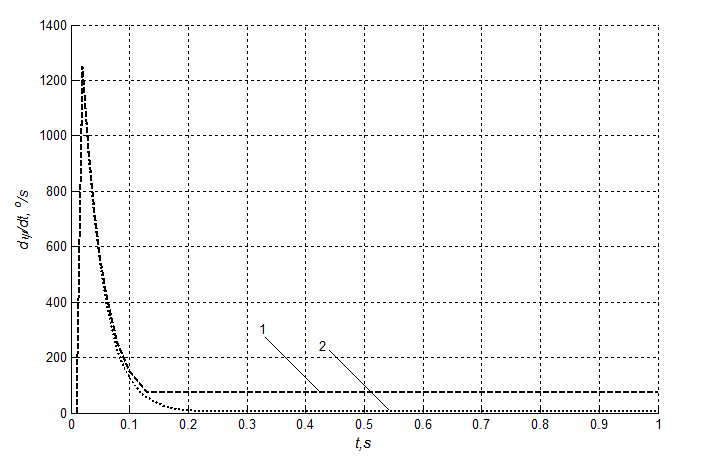


Рис.6.6 Влияние шага квантования на процессы в фильтре. Оценка первой производной входной величины. 1 – оценка при, 2 – оценка при

Величина шага квантования определяется параметром и параметрами фильтра. Исходя из экспериментальных данных было выбрано значение.

Описанный выше алгоритм был реализован на микроконтроллере. Результат работы алгоритма оценки скорости сервомотора при подаче напряжения *U = 5В* приведен на рисунке 6.7. На рисунке также приведен результат моделирования фильтра с принятыми значениями параметров для модели сервомотора, полученной в п.6.3. Как видно из рисунка, результат моделирования и результат оценки реального сигнала совпадают. Колебания величины оценки скорости вращения ротора реального сервомотора связаны с неточностью энкодера (колебания периодические), а так же с наличием люфтов.

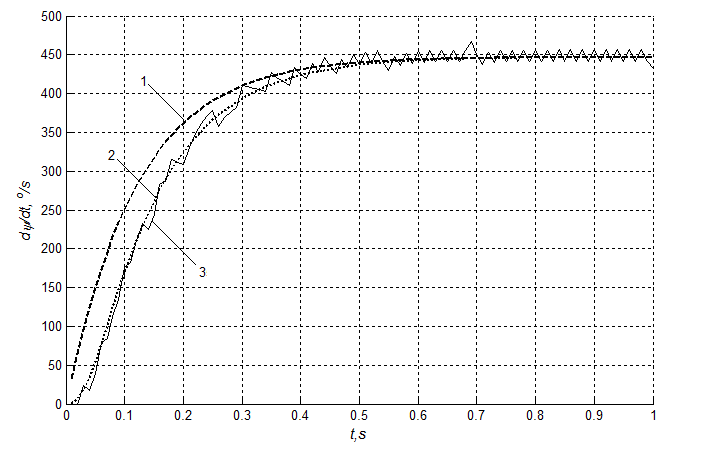


Рис.6.7 Оценка производной входного сигнала модели и реального сигнала. 1 – модель без фильтра, 2 – модель с фильтром, 3 – оценка производной реального сигнала с фильтром на микроконтроллере.

## Получение значений конструктивных коэффициентов сервомотора

Далее необходимо определить конструктивные коэффициенты двигателя. Значение полностью определяется параметром *K*:

Однако данный параметр обладает существенной нелинейностью. Причинами наличия нелинейности являются конструктивные особенности двигателя, а так же микросхем драйверов, используемых для его управления. На рисунке 6.8 представлены переходные характеристики двигателя при значениях входного напряжения в диапазоне 1В – 10В с шагом 1В. На рисунке 6.9 представлена зависимость параметраот входного напряжения.

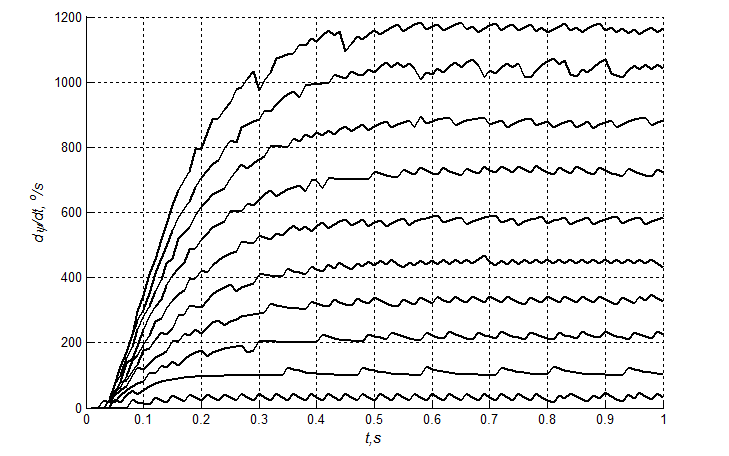


Рис.6.8 Переходные характеристики двигателя при различных значениях входного напряжения*.*

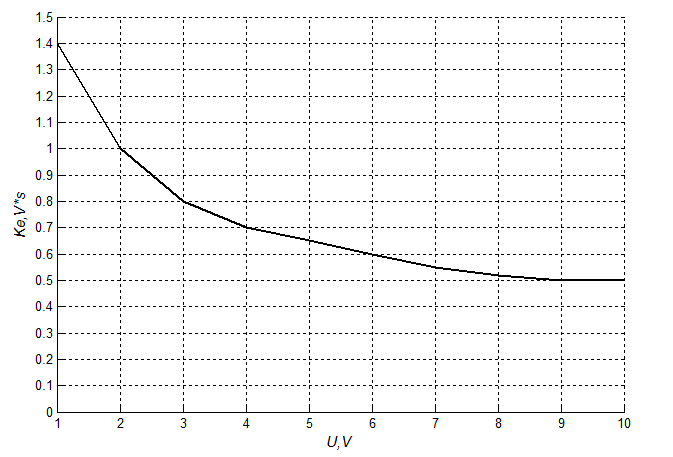


Рис.6.9 Зависимость величины параметра от входного напряжения.

Для определения параметра был проведен эксперимент, в ходе которого измерялось значение пускового тока при известном нагрузочном моменте на валу двигателя. Схема опыта приведена на рисунке 6.10.

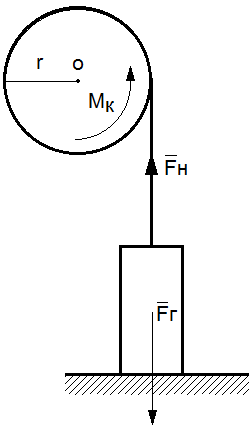
**

Рис.6.10Схема эксперимента по определению конструктивного коэффициента. – момент, создаваемый двигателем; O – ось колеса; r – радиус колеса; сила натяжения нити; сила давления груза на опору.

На колесе, соединенном с валом двигателя, был закреплен груз с помощью нити. Перед подачей напряжения на сервомотор измерялась сила давления этого груза на опору. Далее на двигатель подавалось напряжение, при этом измерялся якорный ток. Сила давления груза уменьшалась на величину силы натяжения нити. Зная эту силу и радиус колеса, несложно определить создаваемый колесом момент.

,

где сила давления груза на опору при отключенном электродвигателе, при включенном. На рисунке 6.11 изображена полученная зависимость момента колеса от тока в якорной цепи.

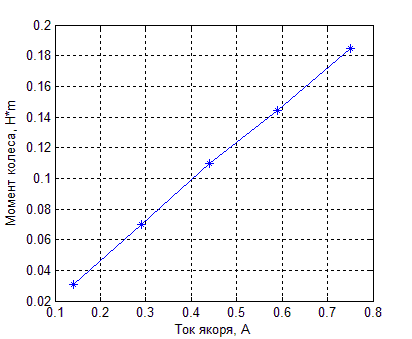


Рис.6.11 Зависимость момента колеса от тока в якорной цепи

Значение можно получить из выражения:

На основании эксперимента было определено следующее значение коэффициента

При известных значениях параметров, а так же активного сопротивления якорной цепи, которое может быть определено с помощью мультиметра, можно вычислить значение момента инерции двигателя. Поскольку в двигатель встроен редуктор, значение момента инерции, полученное согласно процедуре, описанной выше, будет соответствовать выходу редуктора.

## Уточнение модели сервомотора

Для проверки правильности полученных параметров математической модели сервомотора, а так же более детального анализа процессов в нем, была написана тестовая программа. Данная программа реализует систему стабилизации скорости вращения вала сервомотора на основе пропорционально-интегрального регулятора. Программа позволяет получать данные о текущих значениях угловой скорости вращения вала, его угла поворота, а так же напряжения якорной цепи двигателя. Для оценки точности параметров математической модели сервомотора, полученной ранее проведем моделирование процессов в двигателе. Структурная схема модели, соответствующей уравнению (6.1) с учетом ограничения входного напряжения и фильтра оценки угловой скорости вращения вала приведена на рисунке 6.12.

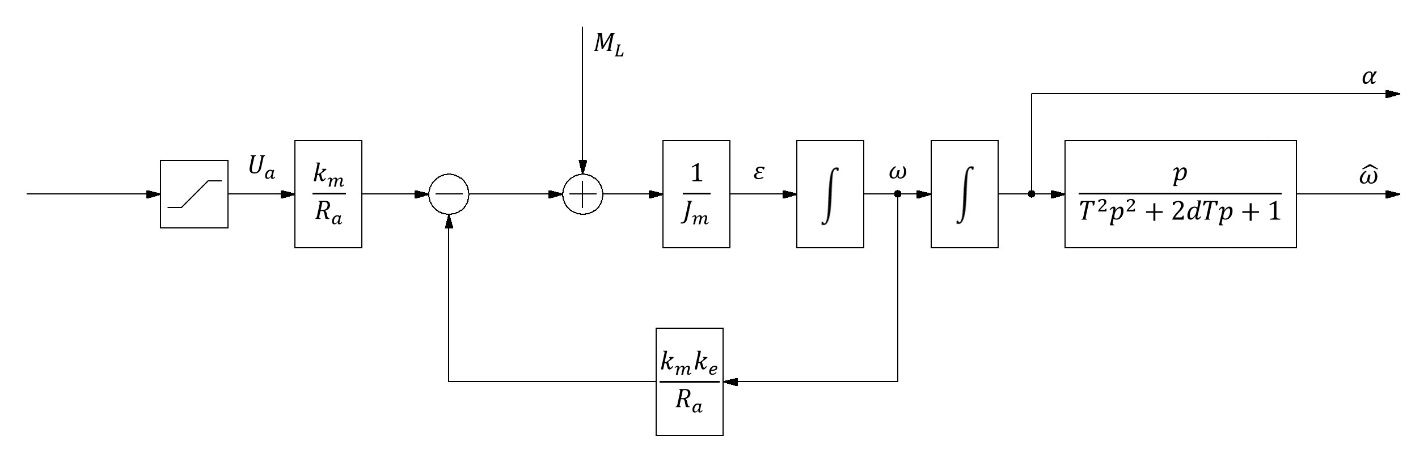


Рис.6.12 Структурная схема модели сервомотора

В данной модели не учитывается нелинейность коэффициента передачи напряжение-скорость. Для учета нелинейности этого параметра аппроксимируем прямой полученную ранее зависимость:

На рисунке 6.13 представлены реальные переходные процессы сервомотора при различных значениях задаваемой скорости и процессы, полученные в результате моделирования. При этом параметры регулятора были следующие:

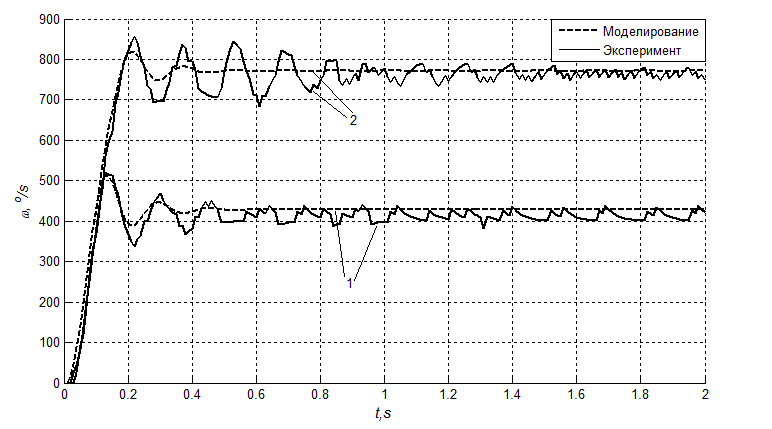


Рис.6.13 Переходные процессы сервомотора и процессы, рассчитанные по модели. 1 – задаваемая скорость 500о/с, 2 – задаваемая скорость 900о/с.

На рисунке 6.14 представлены переходные процессы сервомотора и результаты моделирования при различных значениях пропорционального коэффициента регулятора. Для уменьшения статической ошибки необходимо увеличивать пропорциональный коэффициент, однако при его увеличении начинает сказываться еще одна особенность сервомотора, не учтенная ранее – транспортное запаздывание. Наличие запаздывания объясняется конструкцией сервомотора – наличием зубчатой передачи в редукторе сервомотора, а так же люфтов. По экспериментальным данным можно оценить значение времени запаздывания:

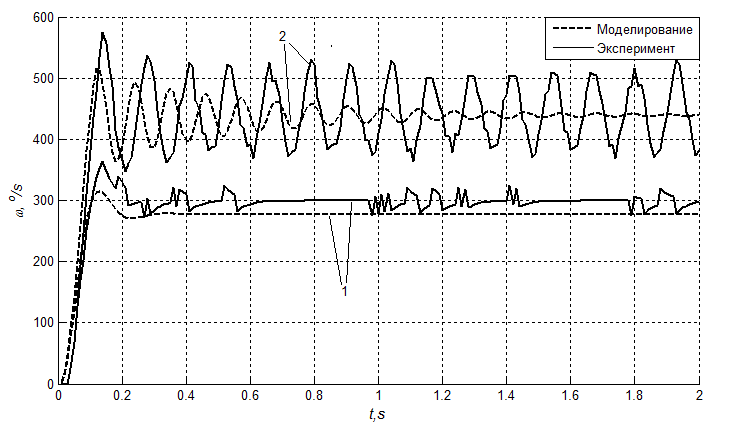


Рис.6.14 Переходные процессы сервомотора и результаты моделирования. Управление с помощью пропорционального регулятора. 1: , 2: .

Для устранения статической ошибки в регулятор введен интегральный канал. На рисунке 6.16 приведены переходные процессы при различных значениях интегрального и пропорционального коэффициентов регулятора. Такая структура регулятора позволяет подавить запаздывание и свести к нулю статическую ошибку, однако снижает быстродействие. На рисунке 6.15 представлена структурная схема модели сервомотора с учетом обнаруженных особенностей. Эта схема может быть использована при моделировании работы системы стабилизации платформы.

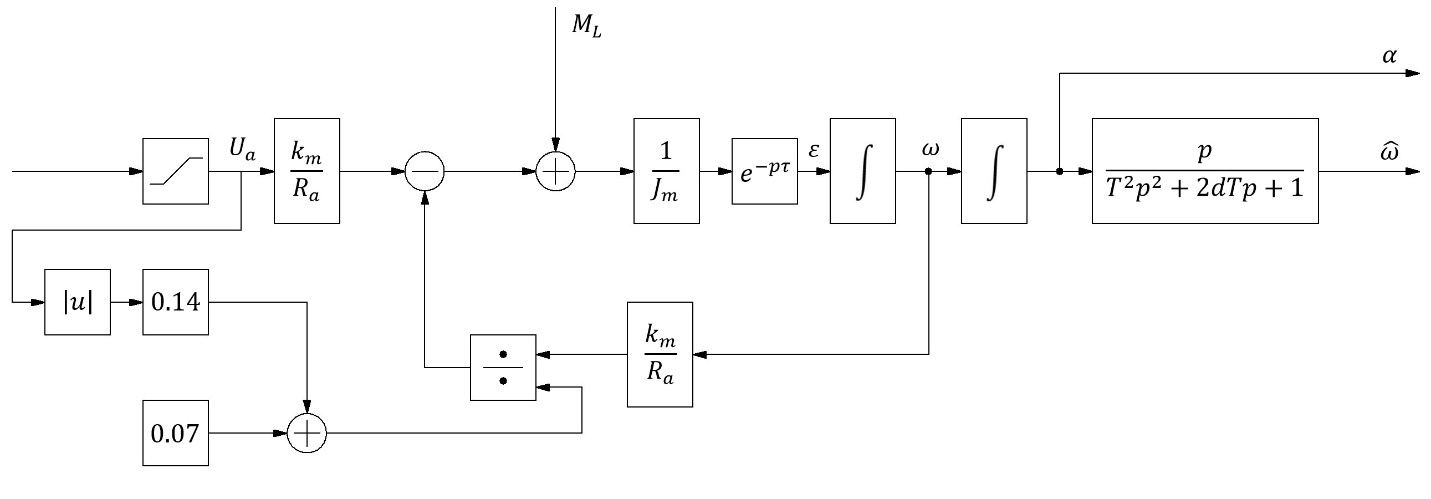


Рис.6.15 Структурная схема уточненной модели сервомотора.

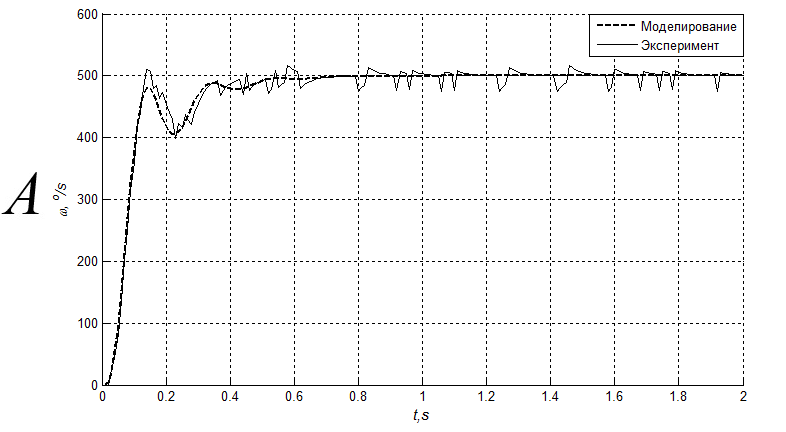
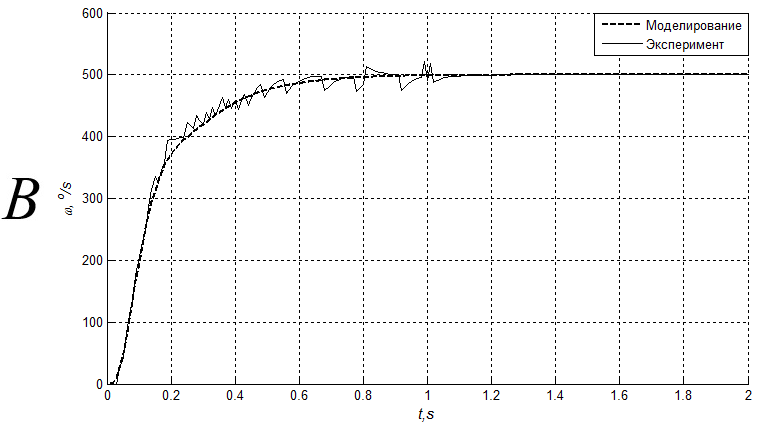
 

Рис.6.16 Переходные процессы сервомотора и результаты моделирования. Величина задаваемой скорости – 500 о/с. Управление с помощью ПИ-регулятора. A: , B: .

# Определение угла наклона

Для реализации алгоритма управления системой, описанного в разделе 4 необходимо иметь оценку угла наклона платформы по отношению к вертикали, а также оценку первой производной данной величины. В данной главе рассмотрены вопросы, связанные с измерением данных величин. Рассмотрены особенности применения МЭМС-датчиков в задаче оценки угла отклонения от вертикали.

## Измерение угла наклона с помощью акселерометра

Для оценивания угла отклонения от вертикали может быть использован акселерометр. Акселерометр – это устройство, измеряющее проекции на его оси суммы всех сил, приложенных к нему. Например, если акселерометр находится в состоянии покоя на некоторой поверхности, его показания равны силе реакции опоры, действующей со стороны этой поверхности, которая, в свою очередь, равна силе тяжести, действующей на акселерометр. В случае, если акселерометр находится в свободном падении, его показания равны нулю. Если на акселерометр действует только сила тяжести, то, определив величину проекций это силы на оси акселерометра, можно вычислить тангенс угла наклона акселерометра относительно вертикального положения:

Однако в реальности данные об угле наклона, полученные таким образом, искажаются, поскольку на акселерометр, как правило, действуют и другие силы. Например, силы, возникающие при резком движении в горизонтальном направлении. На рисунке 7.1 проиллюстрирован такой случай. В конечном счете, эти искажения могут привести к некорректной работе системы стабилизации и сделать неработоспособным устройство.

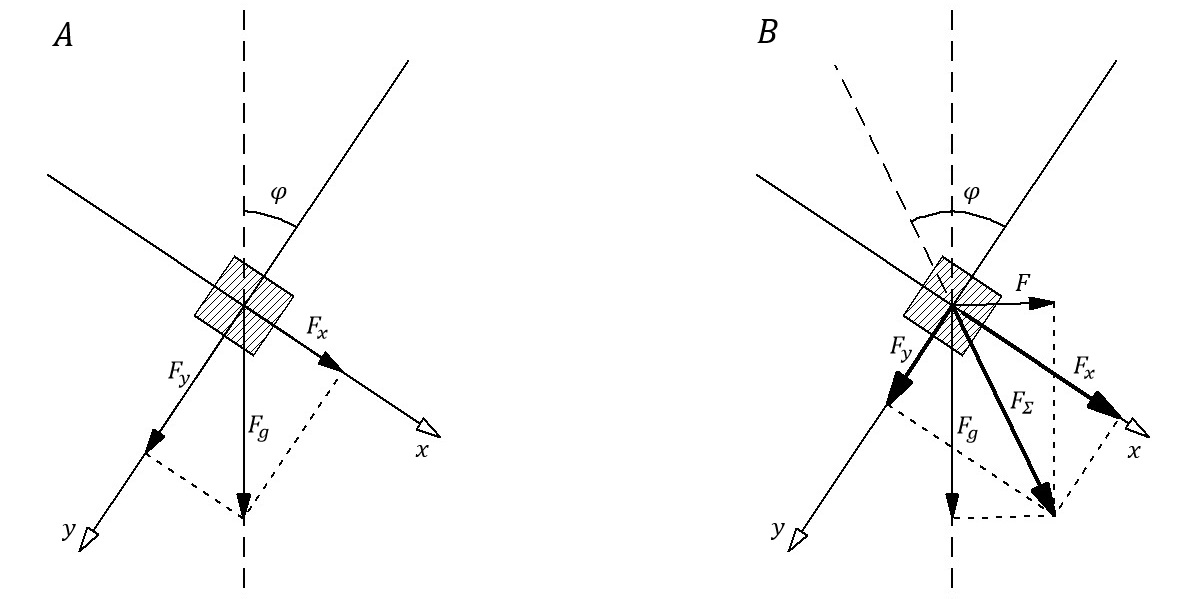


Рис.7.1 Искажение данных при измерении угла наклона с помощью акселерометра. A – на акселерометр действует только сила тяжести, B – на акселерометр действуют сила тяжести и силы в горизонтальном направлении

## Описание тестовой программы акселерометра ADXL335

Подключение датчика ADXL335, а также особенности, связанные с оценкой угла наклона по его данным, рассмотрены в работе [6]. Приведем лишь краткое описание тестовой программы, позволяющей получить данные с этого датчика.

Тестовая программа реализует получение и отправку по интерфейсу UART данных об угле наклона акселерометра ADXL335. Для оценки производной угла наклона при написании данной программы был применен дифференцирующий фильтр второго порядка со структурой, аналогичной структуре фильтра, описанного в разделе 6 данной работы. На рисунках 7.2 и 7.3 приведен график оценки угла наклона и его производной на основе данных, полученных с помощью тестовой программы.

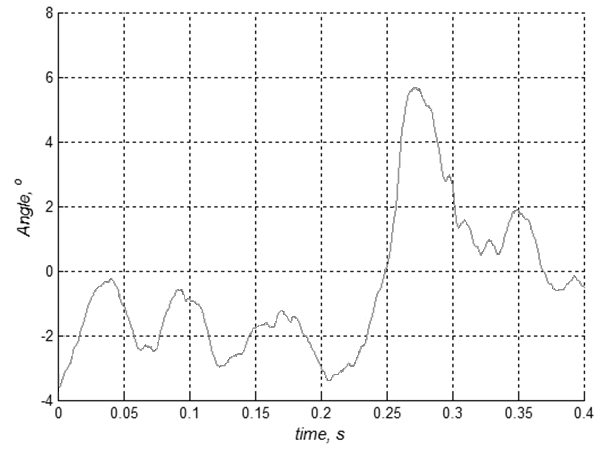


Рис.7.2 Оценка угла наклона, полученная с помощью тестовой программы акселерометра ADXL335.

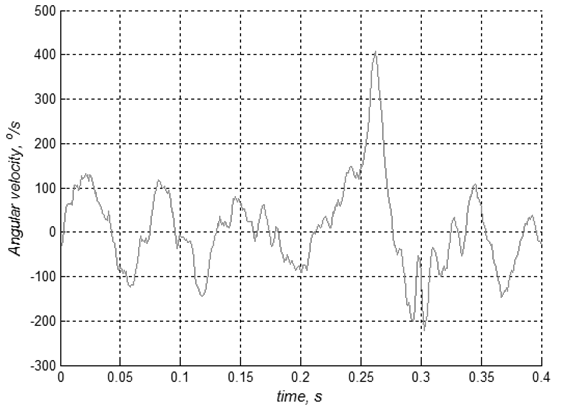


Рис.7.3 Оценка производной угла наклона, полученная с помощью тестовой программы акселерометра ADXL335.

## Анализ данных акселерометра

Для оценки возможности применения акселерометра ADXL335 для определения угла наклона платформы относительно вертикали, воспользуемся тестовой программой, описанной в п.7.2. Для этого проведем измерения зависимости угла наклона платформы от времени при падении с некоторого начального угла. Поведение платформы в этом случае соответствует поведению физического маятника. Движение физического маятника описывает следующее уравнение:

Зависимость угла наклона физического маятника от времени приведена на рисунке 7.4. Как видно из рисунка, угол наклона физического маятника при падении является монотонно возрастающей функцией.

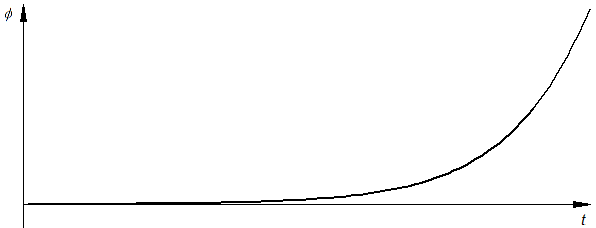


Рис.7.4 Зависимость угла наклона физического маятника от времени.

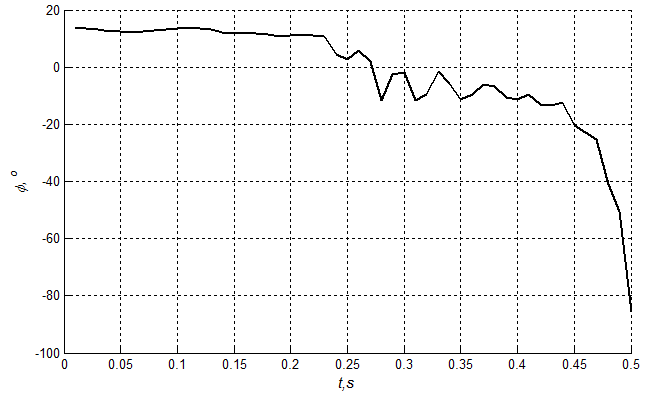


Рис.7.5 Зависимость оценки угла наклона платформы от времени с использованием акселерометра.

На рисунке 7.5 приведены данные об угле наклона платформы при падении из положения с начальным углом отклонения. Как видно из рисунка, оценка угла наклона, получаемая с помощью акселерометра имеет существенные отличия от угла наклона физического мятника. При начальном значении угла наклона около 14 градусов оценка угла наклона уменьшается и уходит в область отрицательных значений, в отличие от истинного значения, которое должно увеличиваться. Такое отличие истинного значения угла и значения, получаемого по данным акселерометра связано с внутренним устройством акселерометра. Следует отметить, что данная особенность проявляется только в динамике. При сравнительно медленном изменении угла наклона значение угла будет достаточно точным. Очевидно, что использовать значение угла, получаемого по данным акселерометра не представляется возможным, а значит, необходимо применять другие методы оценки угла наклона платформы и ее производной.

## Применение гироскопа для оценки угла наклона

В качестве датчика угла наклона по отношению к вертикали может быть использован другой тип МЭМС-датчиков – гироскоп. Гироскоп – это устройство, измеряющее проекцию угловой скорости вращения на его оси. Следовательно, с помощью данного типа датчиков можно получить как угол наклона по отношению к вертикали, так и его производную. Фактически выходной сигнал гироскопа соответствует производной угла наклона. Для получение угла наклона необходимо проинтегрировать выходной сигнал гироскопа. Для измерения угловой скорости вращения был применен измерительный модуль MPU6050, включающий в себя акселерометр, гироскоп, а так же температурный датчик. MPU6050 имеет цифровой выход, обмен данными осуществляется по интерфейсу I2C. В следующем разделе описана процедура подключения MPU6050 к микроконтроллеру STM32.

## Подключение MPU6050

MPU 6050 был подключен к I2C модулю микроконтроллера STM32F100RBT6B для отправки команд и считывания из регистров необходимых данных. После отправки модулю команды о начале измерений происходит постоянная оцифровка показаний со всех осей гироскопа, акселерометра и термодатчика. Требуется только считывать байты из необходимых регистров. Частота записи новых данных в эти регистры аналогово-цифровым преобразователем зависит от выбранной пользователем чувствительности сенсора и, следовательно, диапазона измерений. Полную информацию о регистрах MPU6050 можно получить в [7].

Подключение датчика к микроконтроллеру осуществляется по интерфейсу I2C. Данный интерфейс обладает рядом особенностей: имеет архитектуру ведущий – ведомый, то есть одно устройство (ведущее) производит запрос на чтение или запись ведомых устройств. В нашем случае ведомым является MPU 6050, а ведущим модуль I2C-периферии, встроенный в микроконтроллер. Для связи используются две двунаправленные линии: тактирования и данных. Ведомое устройство имеет свой адрес, который на данной линии должен быть уникальным (в нашем случае это 0хD0). Линии должны быть подтянуты к уровню логической единицы. Как правило, устройства подключаются к линиям через выводы с открытым коллектором (стоком). Фактически, устройства подключены по схеме «монтажное И». Преимуществом является возможность подключения большого количества ведомых устройств, а также сравнительно высокая дальность передачи данных. Но при увеличении дальности фактическая скорость передачи данных снижается. Это связано с тем, что фронты сглаживаются из-за увеличения емкости проводов.

Приведем краткое описание процедуры обмена данными по интерфейсу I2C. Диаграммы сигналов в линии связи приведены на рисунке 7.6. Обмен данными инициирует ведущее устройство. Сигналом начала обмена данными является выставление в линии связи Старт-условия ведущим. Далее ведущее устройство отправляет восьмибитный адрес ведомого. Причем младший бит адреса определяет тип запроса: на чтение данных ил на запись. Если ведомое устройство успешно идентифицирует адрес, оно выставляет в линию связи бит подтверждения.

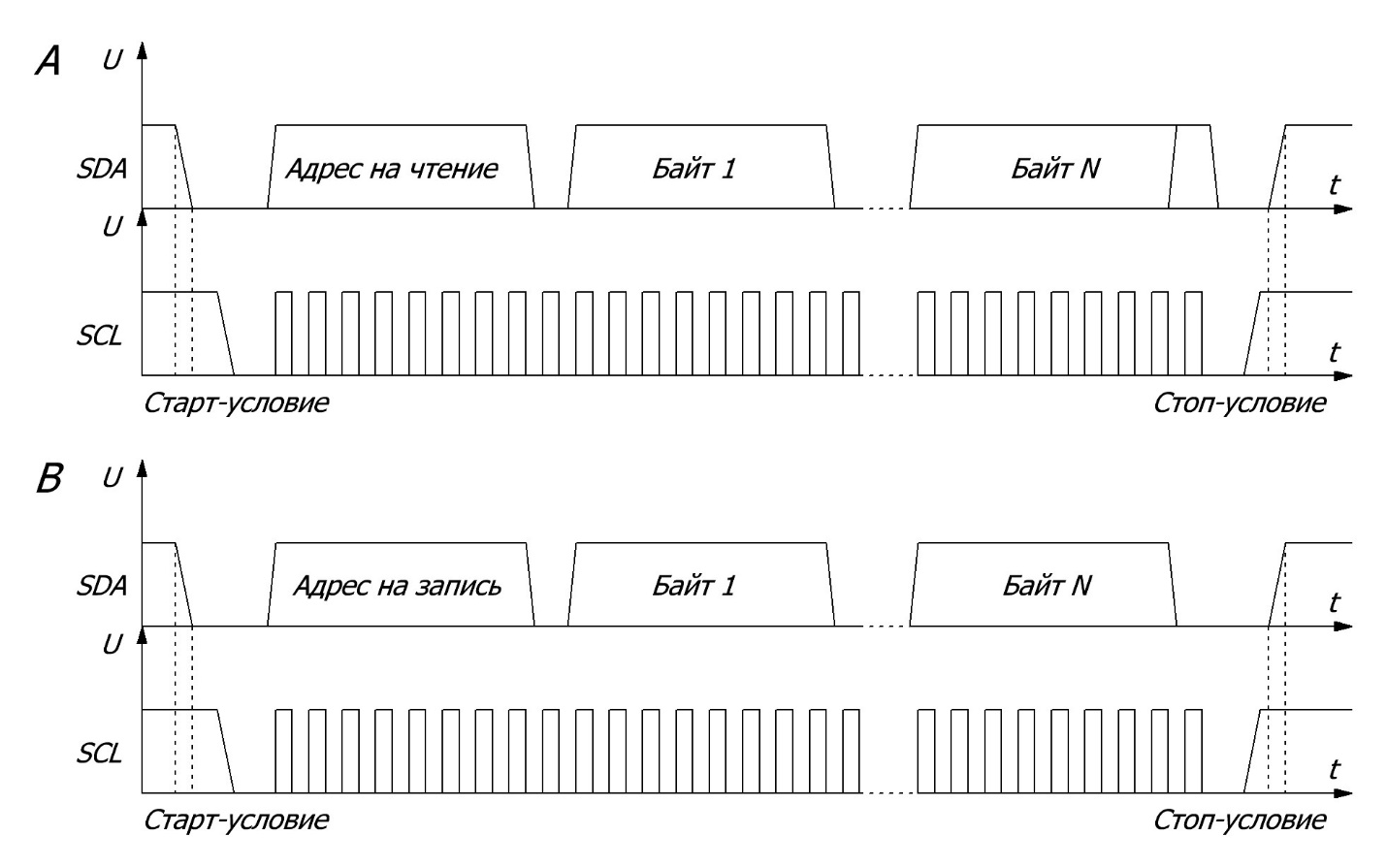


Рис.7.6 Диаграммы сигналов в линии связи при обмене данными по интерфейсу I2C. A – запрос на чтение, B – запрос на запись.

После этого, в зависимости от типа запроса, происходит отправка байтов данных от ведущего к ведомому, или наоборот. При этом каждый байт сопровождается выставлением бита подтверждения. При запросе на чтение сигналом об окончании сеанса связи служит отсутствие бита подтверждения после последнего байта, оправляемого ведомым. После этого ведущий выставляет в линии связи Стоп-условие, и обмен данными завершается. При запросе на запись обмен данными завершается после оправки ведущим Стоп-условия в линию связи.

На рисунке 7.7 представлена блок-схема программы, осуществляющей считывание данных из регистров MPU 6050. Она составлена с учетом особенностей реализации протокола I2C для датчика и контроллера.

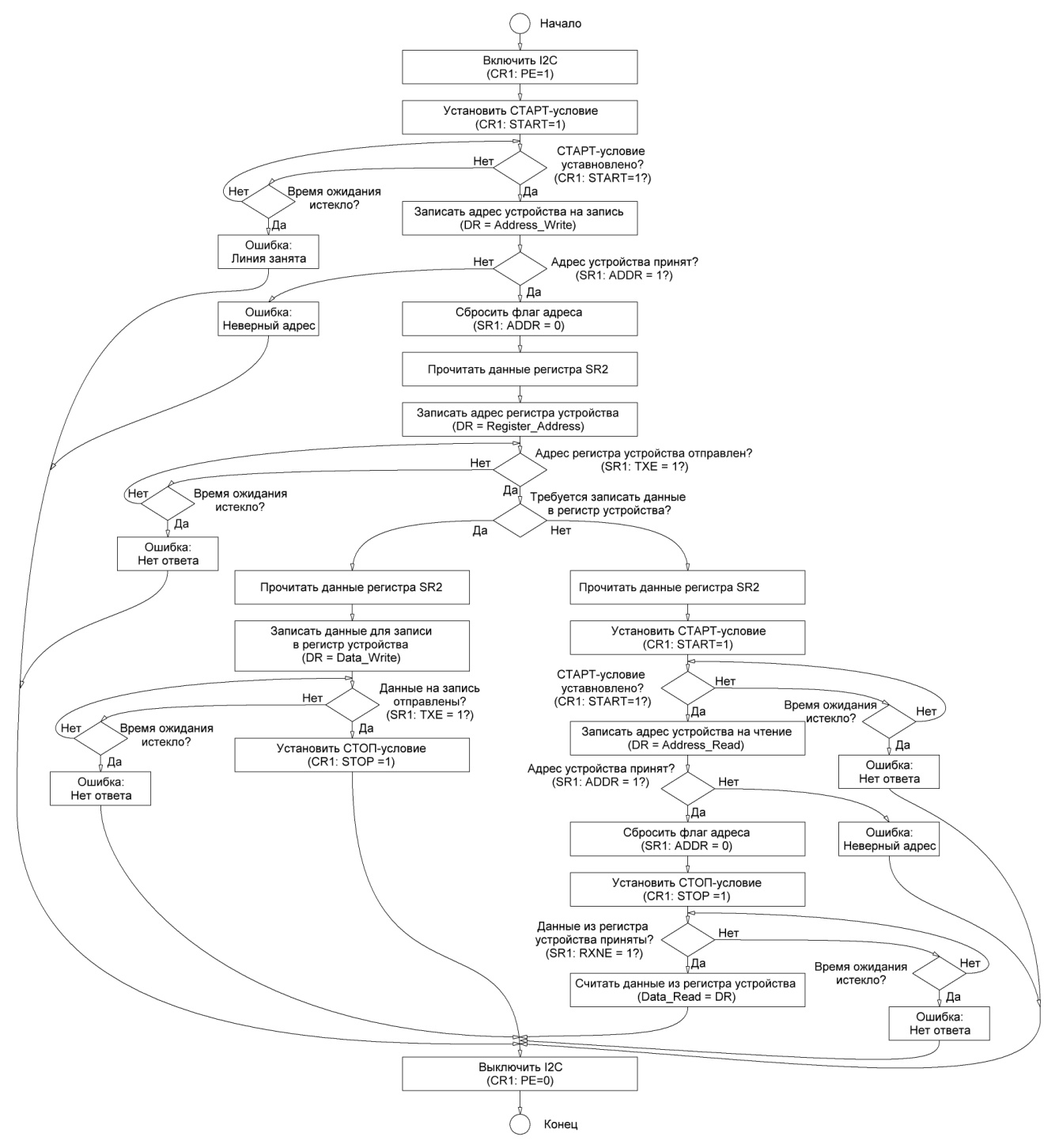


Рис.7.7 Блок-схема алгоритма записи и считывания данных из регистров MPU6050.

Стоит обратить внимание на следующие особенности, связанные с работой этого алгоритма. Во-первых, необходимость чтения регистра SR2 там, где это указано в блок-схеме. Считывание содержимого этого регистра приводит к сбросу некоторых битов-флагов в нем. Если этого не сделать, I2C-модуль MPU6050 не сможет перейти к следующей стадии работы и произойдет зависание программы. Вторая особенность связана с завершением сеанса связи при чтении данных из регистров MPU6050. Сигналом завершения обмена данных является установка стоп-условия на линии данных. В случае с микроконтроллером STM32F100 команду на генерацию стоп-условия для него необходимо отдать *после прочтения предпоследнего байта*. В случае если принимается один байт, то после получения подтверждения принятия адреса от ведомого устройства. Если это сделать после прочтения последнего байта, микроконтроллер будет ожидать прием еще одного байта, которого не будет, и программа зависнет.

На блок-схеме представлен алгоритм для чтения или записи одного байта, однако он легко может быть дополнен для прочтения или записи нескольких байт. После чтения или записи в регистр, MPU6050 автоматически увеличивает адрес регистра на один, что позволяет за один сеанс связи считывать или записывать данные в несколько регистров подряд.

## Анализ данных гироскопа

Для ознакомления с работой измерительной системы MPU6050 была написана тестовая программа, реализующая чтение данных из регистров MPU6050, их обработку и отправку по интерфейсу UART. Данная программа может быть применена для получения данных об угле наклона платформы и его первой производной.

С помощью тестовой программы были получены данные об угловой скорости MPU6050. На рисунке 7.8 приведена зависимость угловых скоростей вращения относительно осей гироскопа от времени. На рисунке 7.9 приведена зависимость углов, получаемых путем интегрирования угловых скоростей. Как видно из рисунка 7.9, абсолютные значения углов с течением времени возрастают. Это следствие так называемого дрейфа нуля – характерной особенности данного типа датчиков. Дрейф появляется в следствии того, что угловая скорость, измеряемая гироскопом имеет ненулевое значение даже тогда, когда гироскоп находится в состоянии покоя.

Из всего вышесказанного можно сделать вывод, что показания гироскопа, так же, как и показания акселерометра, не могут быть использованы напрямую для оценки угла наклона платформы по отношению к вертикали и его производной. Для корректной оценки этих величин может быть применен специальный фильтр, называемый комплементарным фильтром. Разработка алгоритма работы данного фильтра выходит за рамки данной работы. Процесс разработки описан в работе [1]. Здесь же приведем лишь краткое описание принципа его работы.

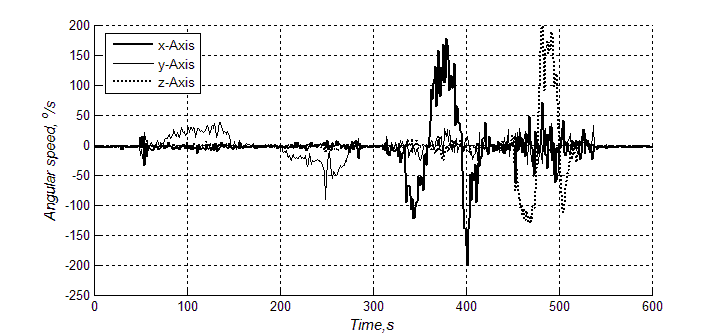


Рис.7.8 Зависимость оценки угловой скорости от времени.

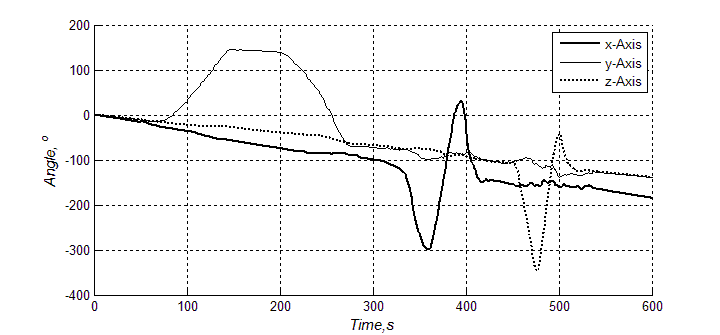


Рис.7.9 Зависимость оценки угла от времени

## Описание принципа работы комплементарного фильтра

Комплементарный фильтр, называемый так же альфа-бета фильтром, использует в своей работе показания как акселерометра, так и гироскопа. Значение угла на текущем шаге определяется выражением:

где – оценка угла, – приращение угла гироскопа за промежуток времени, равный шагу дискретизации фильтра, – значение угла акселерометра, – коэффициент фильтра, имеющий значение в диапазоне от 0 до 1. Приращение угла гироскопа определяется выражением:

Комплементарный фильтр является упрощенным вариантом фильтра Калмана для одномерного случая, где первое слагаемое представляет собой звено предсказания. Однако в данном случае приращение угла определяется исходя из угловой скорости на предыдущем шаге, которая измеряется напрямую гироскопом, в отличие от классического случая, где приращение определяется на основе математической модели системы и известной величины управления. При правильно выбранном значении коэффициента фильтра показания фильтра будут стремиться к показаниям гироскопа, которые будут корректироваться показаниями акселерометра, устраняя дрейф нуля. Таким образом данный фильтр позволяет компенсировать недостатки гироскопа с помощью показаний акселерометра.



Рис.7.10 Зависимость угла наклона от времени по показаниям комплементарного фильтра при неподвижном датчике

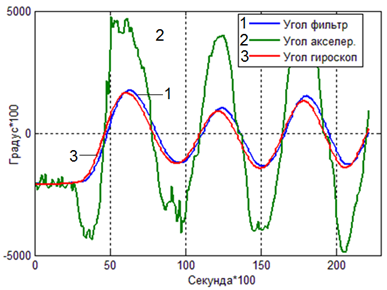


Рис.7.11 Зависимость угла наклона от времени по показаниям комплементарного фильтра при вращении датчика

На рисунках 7.10 и 7.11 приведены данные зависимости угла наклона от времени, полученные с помощью комплементарного фильтра. На первом рисунке приведены показания при неподвижном датчике, на втором – при вращении его вокруг оси y.

# Получение параметров математической модели

Для расчета регулятора необходимо определить неизвестные значения оставшихся параметров математической модели, а именно:

* Расстояние от оси колес до центра масс платформ;
* Масса платформы и колес;
* Радиус инерции платформы относительно оси вращения колес.

Массы платформы и колес были определена с помощью цифровых весов, значения этих параметров:

Из этого следует, что массой колес при расчетах можно пренебречь. Ниже приведено описание экспериментов по определению остальных неизвестных величин.

## Определение координат центра масс платформы

Для определения центра масс платформы воспользуемся методом подвешивания. Суть данного метода заключается в том, что исследуемый объект подвешивается на нити в различных точках, в этом случае нить будет лежать на прямой, проходящей через центр масс объекта. Точка пересечения всех таких линий является точкой, в которой расположен центр масс. Платформа обладает симметрией относительно плоскости *yOz* (см. рисунок 8.1,а). Следовательно, x-координата центра масс равна нулю. Симметрией относительно плоскости *xOy* устройство не обладает, но, поскольку угол отклонения от вертикали сравнительно мал, можно считать, что расстояние от центра масс системы до точки подвеса равно проекции этого расстояния на ось *y* (см. рисунок 8.1,б). Поэтому, для определения y-координаты центра масс достаточно получить две прямые, проходящие через центр масс. Результаты эксперимента приведены на рисунке 8.2. Было определено следующее значение расстояния от центра масс платформы до точки подвеса:

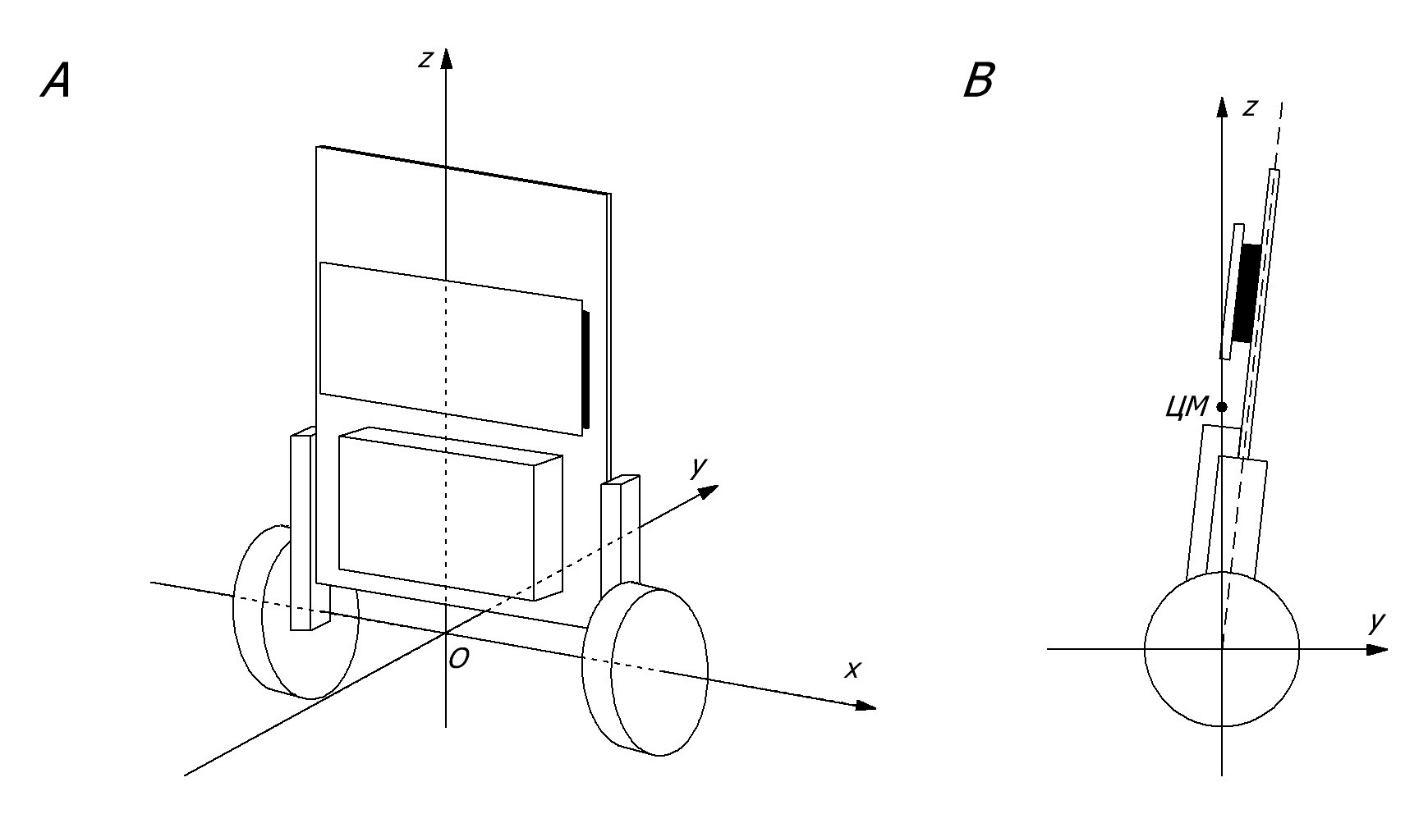


Рис.8.1 Пояснения к эксперименту по определению центра масс платформы

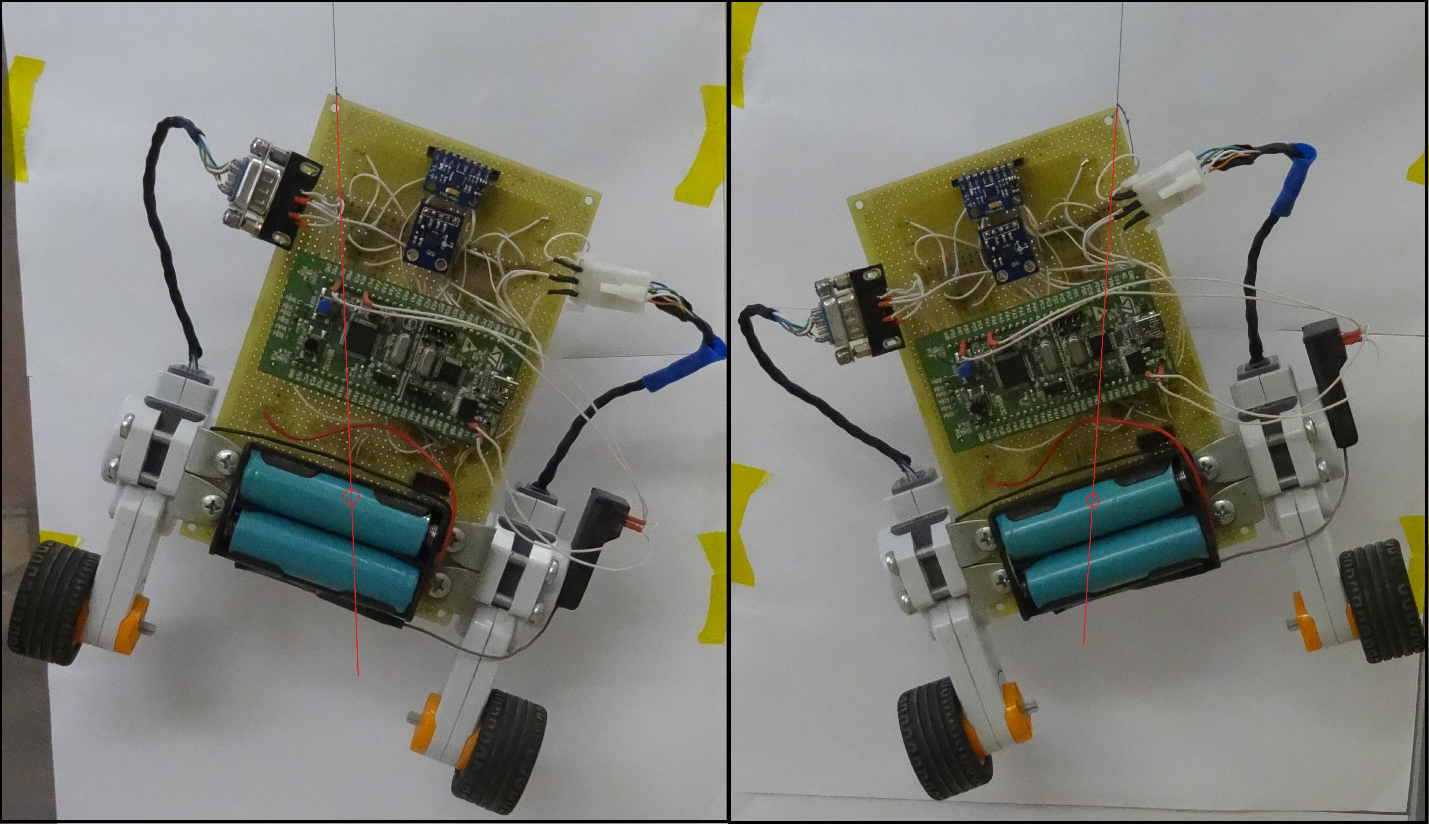


Рис.8.2 Результаты эксперимента по определению центра масс платформы

## Определение радиуса инерции платформы

Для определения радиуса инерции платформы воспользуемся уравнением периода малых колебаний физического маятника:

где – радиус инерции маятника относительно точки подвеса. Из данного соотношения может быть определен радиус инерции платформы относительно оси колеса:

Представление платформы как физического маятника с малым периодом колебаний допустимо в том случае, если:

* Точкой подвеса является ось колеса;
* Колебания совершаются около нижнего устойчивого положения равновесия;
* Амплитуда колебаний не превышает 3-5 градусов.

Результат эксперимента по определению периода малых колебаний приведен на рисунке 8.3.

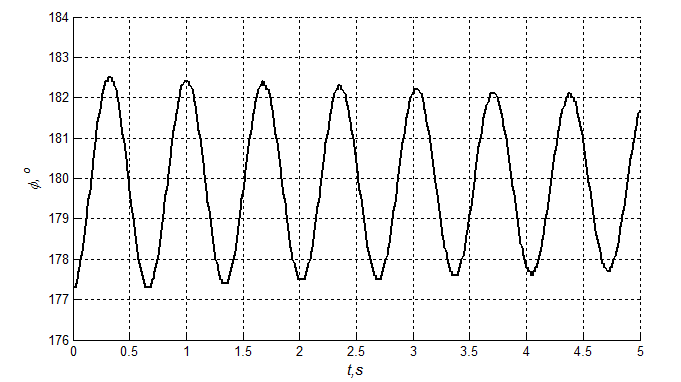


Рис.8.3 Результат эксперимента по измерению периода малых колебаний платформы.

Исходя из данных эксперимента, было определено следующее значение периода малых колебаний:

По данному значению было определено значение радиуса инерции платформы:

На рисунке 8.4 приведены результаты моделирования физического маятника с параметрами, соответствующими параметрам платформы, и результаты эксперимента. Как видно из рисунка, производные угла отклонения от вертикали платформы и модели физического маятника так же хорошо совпадают (за исключением медленного затухания колебаний платформы из-за наличия трения).

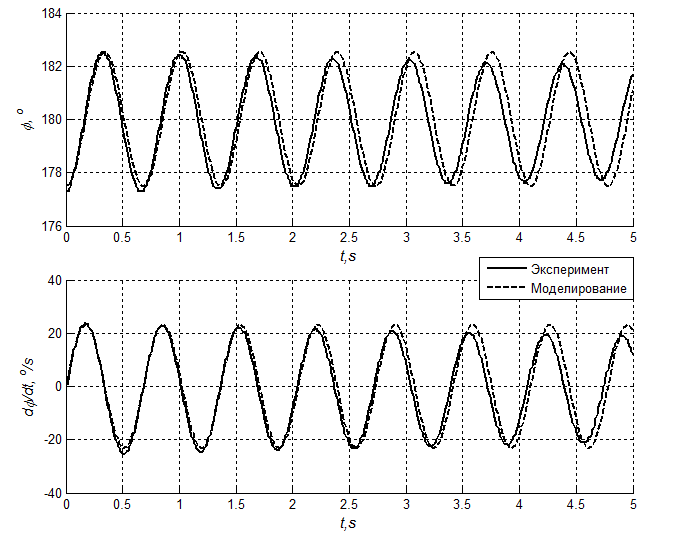


Рис.8.4 Результат моделирования физического маятника и данные эксперимента по измерению малых колебаний платформы

Для проверки правильности полученного значения радиуса инерции был проведен эксперимент, аналогичный тому, который был описан в разделе 7. В ходе эксперимента измерялся угол наклона платформы при падении из некоторого начального положения (вблизи верхнего неустойчивого положения равновесия). Так же был смоделирован аналогичный процесс для представления платформы в виде физического маятника. При этом, значение рассчитанного радиуса инерции платформы было скорректировано, поскольку точкой подвеса в этом случае является не ось колеса, а точка его соприкосновения с поверхностью:

Результаты эксперимента и моделирования представлены на рисунке 8.5. Из рисунка видно, что данные модели и эксперимента совпадают для углов, меньших 5-7 градусов, то есть в пределах значений, в которых допустима линеаризация модели, проведенная в разделе 4.

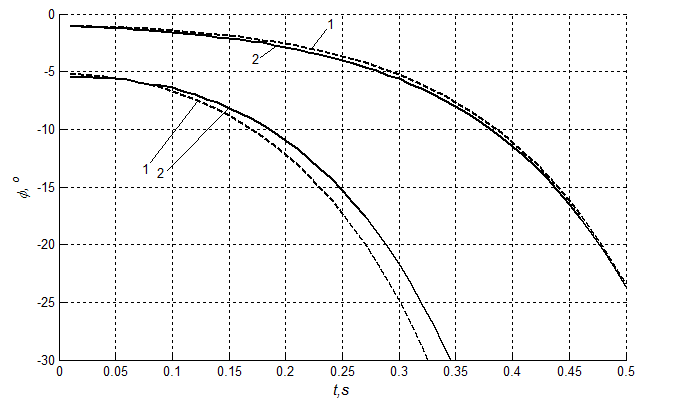


Рис.8.5 Результаты эксперимента по проверке значения радиуса инерции платформы. 1 – результат моделирования, 2 – результат эксперимента.

## Проверка параметров математической модели

Для проверки правильности найденных значений параметров математической модели (радиус инерции, расстояние от центра масс до точки подвеса, параметры сервомотора) был проведен эксперимент. Платформа отклонялась на некоторый начальный угол, затем прикладывался максимальный уровень напряжения к якорной цепи сервомоторов. В результате платформа начинала движение в направлении положения равновесия. Данный эксперимент с одной стороны позволил оценить правильность полученных значений параметров модели, с другой стороны – определить максимальный угол отклонения, при котором платформа еще может быть стабилизирована. На рисунке 8.6 приведены результаты эксперимента и моделирования системы с полученными параметрами при различных начальных условиях.

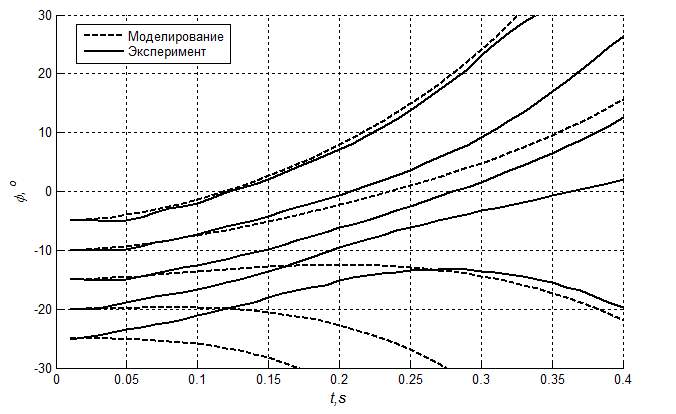


Рис.8.6 Результаты эксперимента по проверке параметров математической модели платформы.

Из рисунка видно, что в пределах допустимых для линеаризации модели значений угла наклона (0-5 градусов) результаты эксперимента соответствуют результатам моделирования. При больших значениях угла начинает проявляться нелинейность реальной системы. Таким образом, можно сделать вывод о соответствии найденных параметров математической модели реальным параметрам системы. Максимальное значение угла наклона, при котором платформа еще может стабилизироваться составляет примерно 20 градусов, что говорит о наличии запаса по ресурсу управления, т.к. рабочий диапазон угла наклона платформы составляет ± 5o. Таким образом, третье условие разрешимости задачи синтеза, рассмотренное в разделе 4, выполняется.

# Расчет и реализация регулятора

## Расчет коэффициентов регулятора

Для расчета коэффициентов регулятора воспользуемся соотношениями, полученными в разделе 4 (4.25). Составим желаемое уравнение динамики. Выбор корней полинома желаемого уравнения определяется требованиями к качеству переходного процесса. Для обеспечения устойчивости системы необходимо, чтобы корни желаемого уравнения лежали в левой полуплоскости. Определим расчетное время переходного процесса системы исходя из данных экспериментов, проведенных ранее. Поскольку для расчета регулятора используется процедура, основанная на линеаризованной модели, нежелательно, чтобы в процессе работы угол наклона выходил за допустимые для линеаризации значения. Из результатов опыта, описанного в разделе 8 (рис.8.5), видно, что к границе допустимых значений при отсутствии управления угол наклона подходит за время 0.3 – 0.5 секунды, в зависимости от начальных условий. Исходя из этого, примем для расчетов значение времени переходного процесса:

Таким образом, на основании соотношения

где – правая граница допустимого распределения корней при заданном быстродействии, выберем следующее распределение корней желаемого уравнения:

По заданным корням с учетом численных значений параметров модели платформы, определенных ранее, были получены следующие значения коэффициентов регулятора.

|  |  |
| --- | --- |
|  | (9.1) |

На рисунке 9.1 приведены результаты моделирования системы стабилизации с рассчитанными значениями коэффициентов для линеаризованной модели. Следует отметить, что из-за большого количества нелинейностей в реальном объекте управления, которые не учитываются в линеаризованной модели, полученные значения коэффициентов могут не обеспечивать заданное быстродействие, а в худшем случае даже не обеспечивать устойчивость системы. Поэтому после реализации регулятора потребуется корректировка коэффициентов с учетом реальных данных.

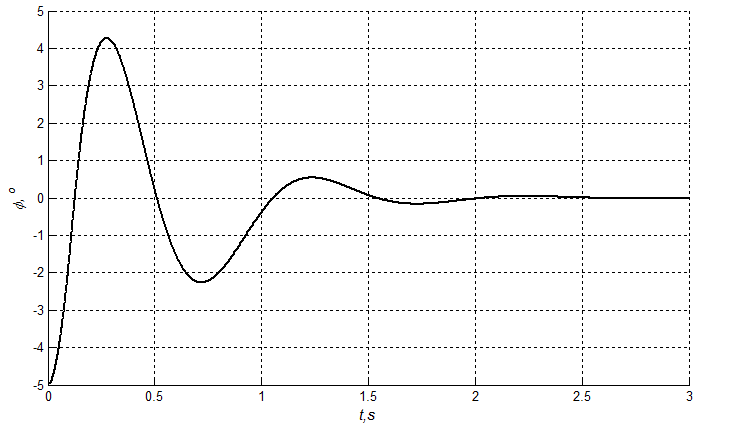


Рис.9.1 Результаты моделирования платформы на основе линеаризованной модели.

Для проверки работоспособности регулятора с рассчитанными коэффициентами было проведено моделирование нелинейной модели системы, структурная схема которой приведена на рисунке 9.2.

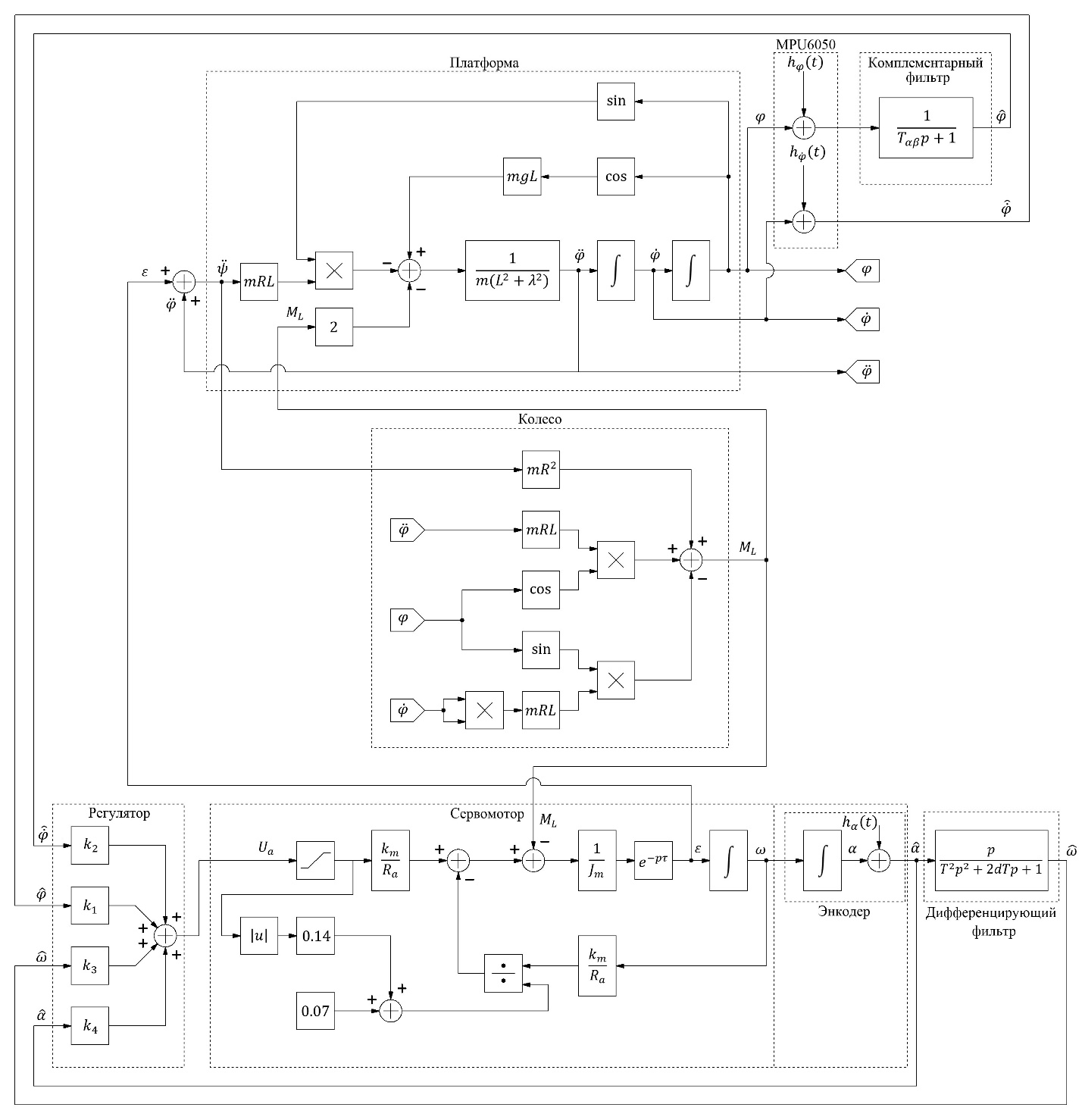


Рис.9.2 Структурная схема нелинейной модели системы

Результаты моделирования системы с рассчитанными коэффициентами регулятора приведены на рисунке 9.3. Как видно из рисунка, нелинейная система неустойчива. По характеру процесса можно сделать предположение, что усиление регулятора слишком велико: переходный процесс расходится с течением времени и в конечном счете система становится неустойчивой. Для проверки предположения уменьшим коэффициенты в 10 раз.

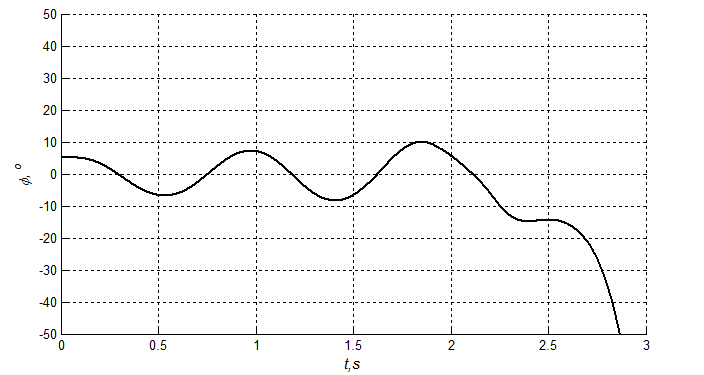


Рис.9.3 Моделирование системы по нелинейной модели с рассчитанными значениями коэффициентов регулятора (9.1).

|  |  |
| --- | --- |
|  | (9.2) |

После этого проведем повторное моделирование. Его результаты представлены на рисунке 9.4. По данным рисунка 9.4 можно сделать вывод, что на этот раз усиления регулятора недостаточно, поскольку значения угла увеличивается, то есть система изначально движется в направлении, противоположном положению равновесия. Исходя из проведенных численных экспериментов можно сделать предположение, что рабочие значения коэффициентов находятся в области, ограниченной расчетными и уменьшенными значениями коэффициентов. Для проверки предположения увеличим коэффициент регулятора, соответствующий углу наклона, в 5 раз.

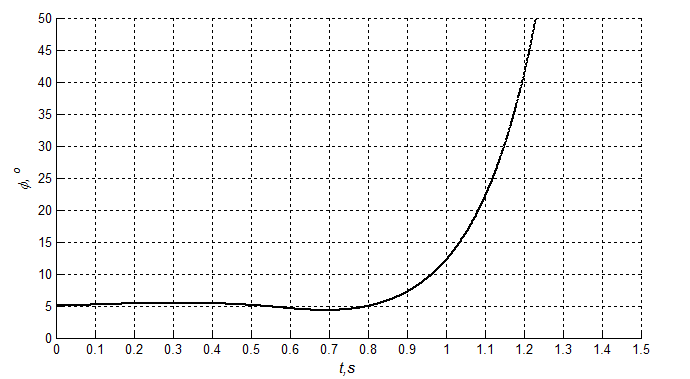


Рис.9.4 Моделирование системы по нелинейной модели со значениями коэффициентов регулятора (9.2).

Увеличение коэффициента при производной угла наклона нежелательно, поскольку данная величина содержит большую помеху, по сравнению с углом наклона.

|  |  |
| --- | --- |
|  | (9.3) |

Результат моделирования системы с измененным значением коэффициента представлен на рисунке 9.5. При заданных значениях коэффициентов система стабилизирована, но находится близко к границе устойчивости. Это означает, что такой регулятор не сможет стабилизировать реальный объект управления из-за неучтенных параметров объекта и действия возмущений. Однако данные коэффициенты регулятора могут использоваться для оценки степени соответствия нелинейной модели реальному объекту управления, а так же анализа его поведения.

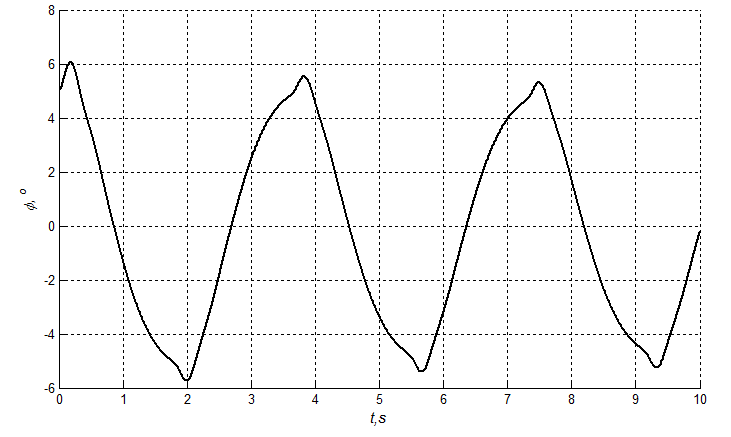


Рис.9.5 Моделирование системы по нелинейной модели со значениями коэффициентов регулятора (9.3).

## Реализация регулятора

При реализации регулятора следует учитывать особенности реального объекта управления. В отличие от модели, в которой платформа представлялась как линейный одноканальный объект, в реальности объект управления имеет два входных канала, поскольку в системе используется два сервомотора. Систему допустимо представлять одноканальной в случае синхронной работы сервомоторов, что реализуется в режиме балансировки (см. раздел 2). Возникает вопрос, каким образом следует организовать обратную связь от двигателей. Возможны варианты:

* Использование данных одного из сервомоторов;
* Комбинирование данных двух сервомоторов;
* Организация отдельных связей для каждого сервомотора.

В реальности двигатели имеют разброс параметров, в частности – коэффициент передачи напряжение-скорость отличается у различных сервомоторов. Это означает, что при приложении напряжения одинаковой величины скорости вращения роторов сервомоторов будут отличаться. В результате при движении платформа будет поворачивать в сторону сервомотора, скорость которого ниже. Во избежание этого выберем третий способ. Для этого управляющие воздействия для двигателей будем формировать следующим образом:

Здесь – общая составляющая управления, связанная с углом наклона и его производной, , - частные составляющие, связанные с углом поворота и скоростью вращения ротора конкретного двигателя. Такая организация позволит подавить различия в параметрах сервомоторов и добиться прямолинейного перемещения.

На рисунке 9.6 представлены результат работы регулятора со значениями коэффициентов (9.3) и результат моделирование нелинейной системы.

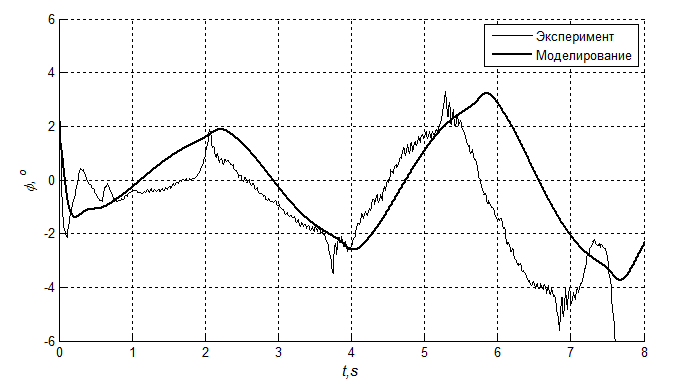


Рис.9.6 Работа системы стабилизации со значениями коэффициентов (9.3).

Как и предполагалось ранее, система не стабилизирована, но близка к границе устойчивости. При этом, хотя процессы в реальном объекте отличаются от рассчитанных по модели, они все же достаточно близко к ним. Для получения устойчивых процессов в объекте потребуется дальнейшая корректировка коэффициентов.

## Настройка коэффициентов регулятора

Для корректировки коэффициентов регулятора была проведена процедура автоматической настройки его параметров. Данная процедура схожа с процедурой численной оптимизации параметров, однако в данном случае задача нахождения именно оптимальных параметров регулятора не ставится, требуется лишь получить устойчивые процессы с приемлемым качеством. Процедура автоматической оптимизации предполагает введение единого критерия качества процессов в системе. При этом, критерий качества должен зависеть от оптимизируемых параметров, соответствовать целям оптимизации и иметь минимум [8]. Для проведения процедуры настройки параметров критерием качества выберем следующий функционал:

На рисунке 9.7 приведена блок-схема примененного алгоритма. Стоит отметить, что данный алгоритм имеет существенные недостатки. В частности, из-за нелинейности системы результат настройки, проведенной по данному алгоритму сильно зависит от начальных значений параметров регулятора, начальных условий переменных объекта и порядка настройки коэффициентов. Кроме того, данный алгоритм чувствителен к наличию локальных минимумов критерия качества, которые могут присутствовать из-за нелинейности объекта. Однако этот алгоритм прост в реализации, а в данной работе, как уже было сказано выше, не требуется получить собственно оптимальные процессы.

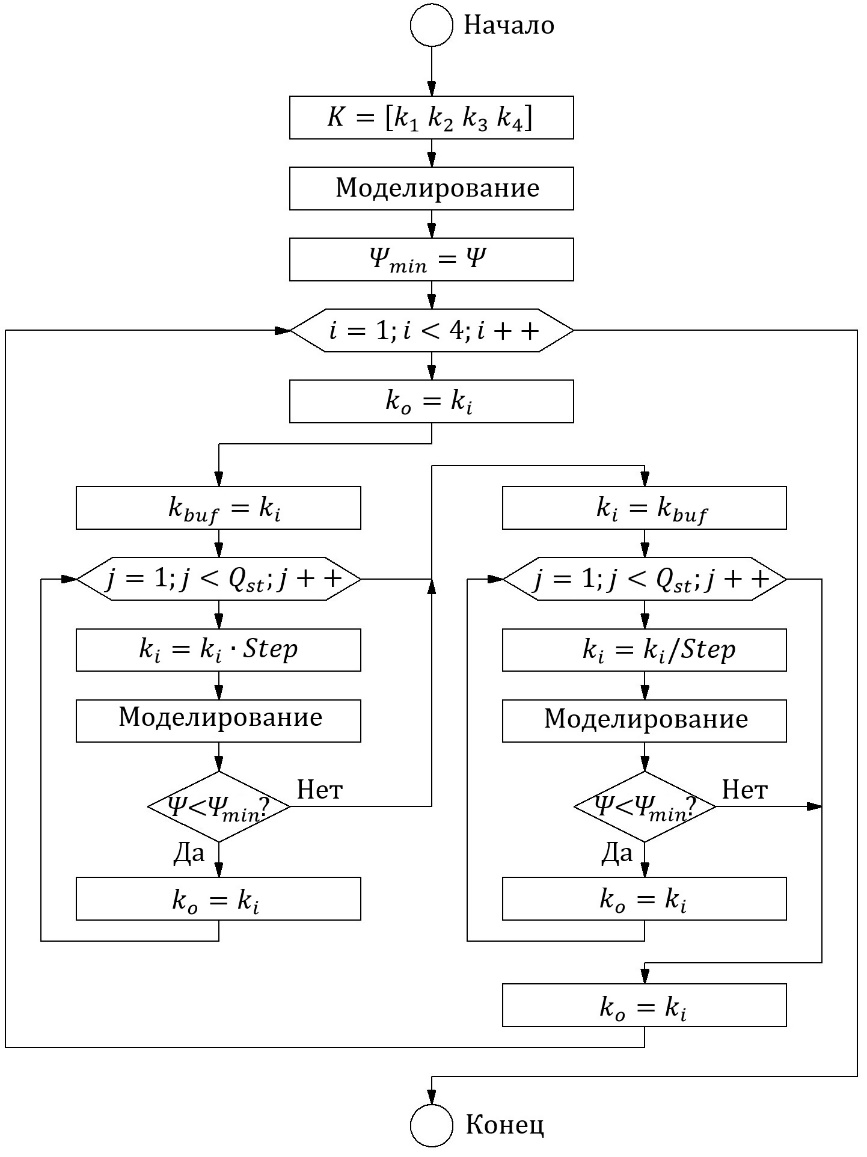


Рис.9.7 Алгоритм настройки параметров регулятора. Здесь – матрица коэффициентов регулятора, – количество шагов алгоритма, – шаг изменения настраиваемого параметра. Параметры и определяют точность поиска параметра и количество итераций поиска. Для улучшения результата данный алгоритм можно повторить несколько раз.

В результате проведенной настройки параметров были получены следующие значения коэффициентов регулятора:

|  |  |
| --- | --- |
|  | (9.4) |

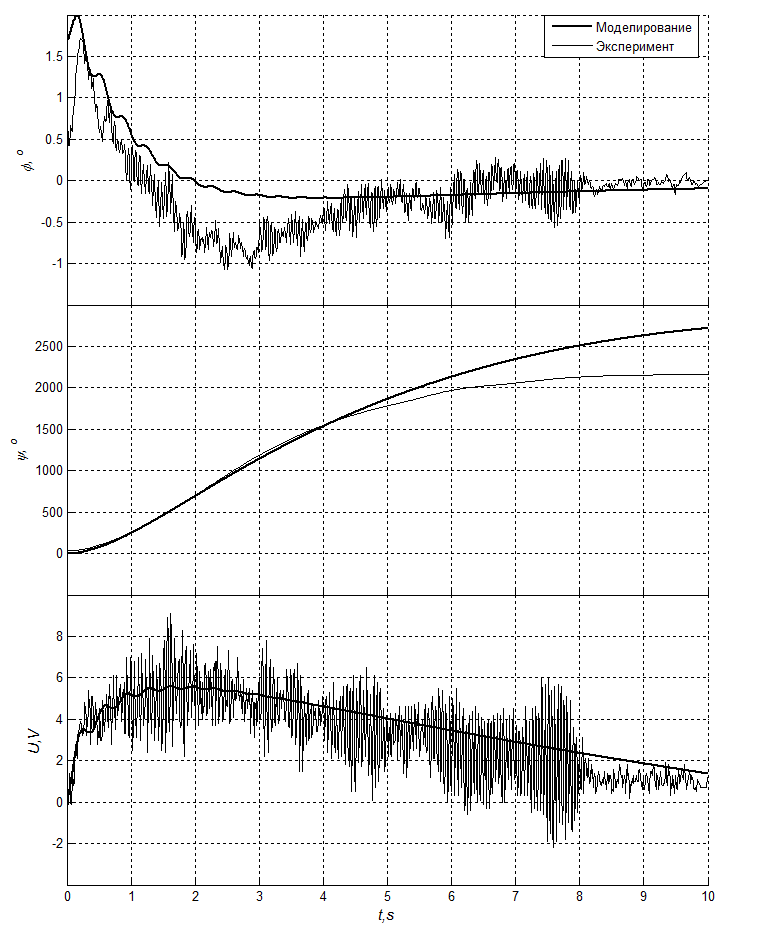


Рис.9.8 Работа системы стабилизации со значениями коэффициентов (9.4).

На рисунке 9.8 представлен результат моделирования процессов в системе и результат работы реальной системы стабилизации с коэффициентами регулятора (9.4). Как видно из рисунка, система стабилизирована, процессы в ней сравнительно хорошо соответствуют модели. Однако в данной системе присутствует значительная статическая ошибка по углу поворота колеса. Это связано с тем, что коэффициент, связанный с углом поворота колеса мал.

Для настройки коэффициентов так же может быть применен следующий критерий качества:

В этом случае ошибка в конце переходного процесса имеет больший вес, чем ошибка на старте. В результате настройки по данному критерию были получены следующие значения коэффициентов:

|  |  |
| --- | --- |
|  | (9.5) |

На рисунке 9.9 представлен результат моделирования процессов в системе и результат работы реальной системы стабилизации с коэффициентами регулятора (9.5). Из рисунка видно, что система имеет большую колебательность по сравнению с системой с коэффициентами (9.4). Однако, статическая ошибка угла поворота колеса существенно ниже. Так же, система более качественно отрабатывает возмущения и в целом быстрее реагирует на изменения. Примем данные значения коэффициентов регулятора в качестве рабочих.

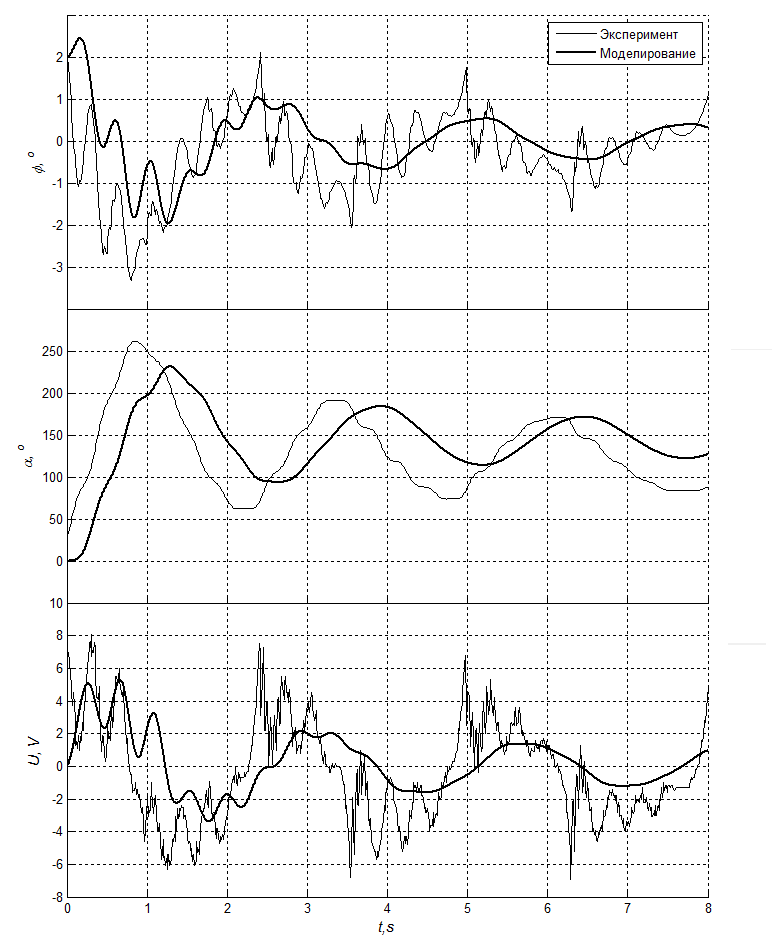


Рис.9.9 Работа системы стабилизации со значениями коэффициентов (9.4).

На основании полученных результатов можно сделать вывод, что модальный метод синтеза не всегда дает хороший результат для нелинейных объектов такого типа. Тем не менее, несмотря на то, что расчетные значения коэффициентов регулятора, найденных для линеаризованной модели, существенно отличаются от найденных экспериментально рабочих значений, сам принцип организации обратной связи по переменным состояния, который соответствует данному методу, дает возможность получить качественную систему стабилизации. Так же, линейная структура регулятора позволяет осуществлять настройку каждого коэффициента отдельно от остальных, анализируя влияние на поведение системы реальной физической величины, которой данный коэффициент соответствует. Пример методики настройки коэффициентов модального регулятора дан в работе [1]. Далее описан результат работы системы с найденными значениями коэффициентов.

## Возможности системы управления в режиме балансировки

В данном разделе приведены результаты работы полученной системы стабилизации при различных условиях. На рисунках 9.10 и 9.11 приведены переходные процессы платформы при различных начальных условиях.

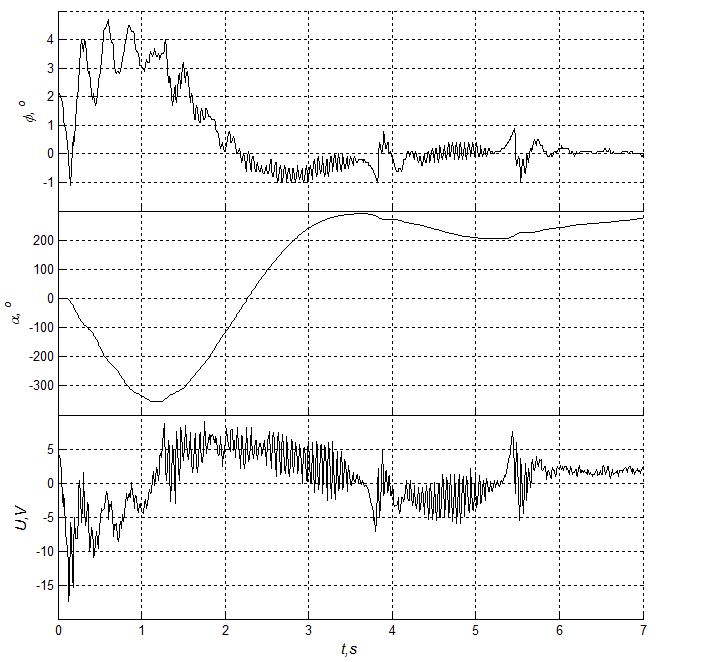


Рис.9.10 Результат работы системы стабилизации: ненулевые начальные условия по углу наклона и его производной.

В первом случае система управления начинала работу по прошествии 0.1 секунды с момента подачи питания. К этому времени платформа отклонялась на угол примерно четыре градуса, а так же имела некоторую скорость падения. Во втором случае платформа с работающей системой стабилизации принудительно откатывалась на некоторое расстояние от точки равновесия, что эквивалентно ненулевым начальным условиям по углу наклона и углу поворота колеса.

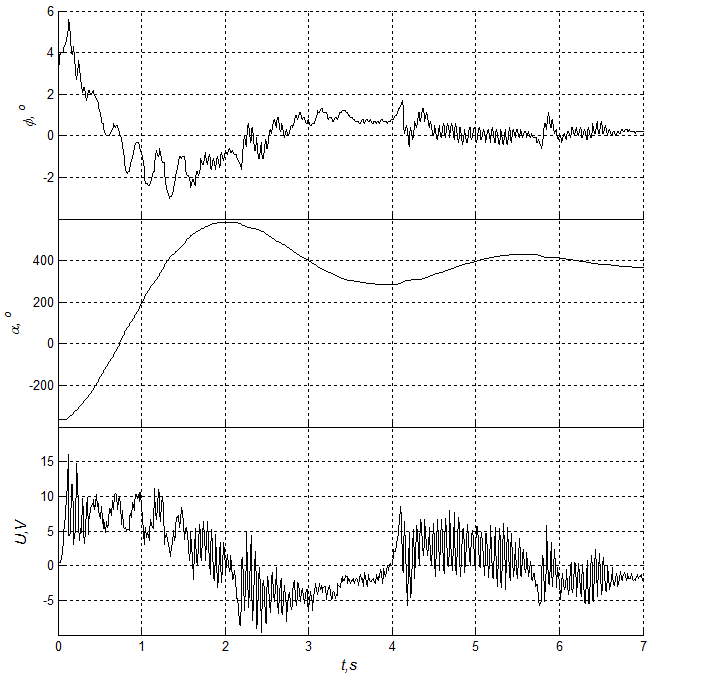


Рис.9.11 Результат работы системы стабилизации: ненулевые начальные условия по углу наклона и углу поворота колеса.

В обоих случаях система стабилизации успешно отрабатывает начальные условия. Статическая ошибка возникает в том числе из-за того, что угол наклона, получаемый с помощью датчика MPU6050 имеет некоторый уровень смещения. Связано это с неточностью калибровки датчика. Следует отметить, что полностью откалибровать смещение не возможно, можно лишь снизить его до некоторого приемлемого уровня, что и было сделано. Компенсировать статическую ошибку возможно, вводя в закон управления величину, пропорциональную интегралу угла поворота колеса, однако на данном этапе работ такая задача не ставилась.

На рисунке 9.12 приведены данные о работе системы при отработке возмущения, вызванного действием силы, приложенной к верхней части платформы (резкий «толчок»). Как видно из рисунка, возмущение успешно отработано системой стабилизации. При этом действие силы вызывало отклонение платформы на угол до семи градусов, с которым система справилась.

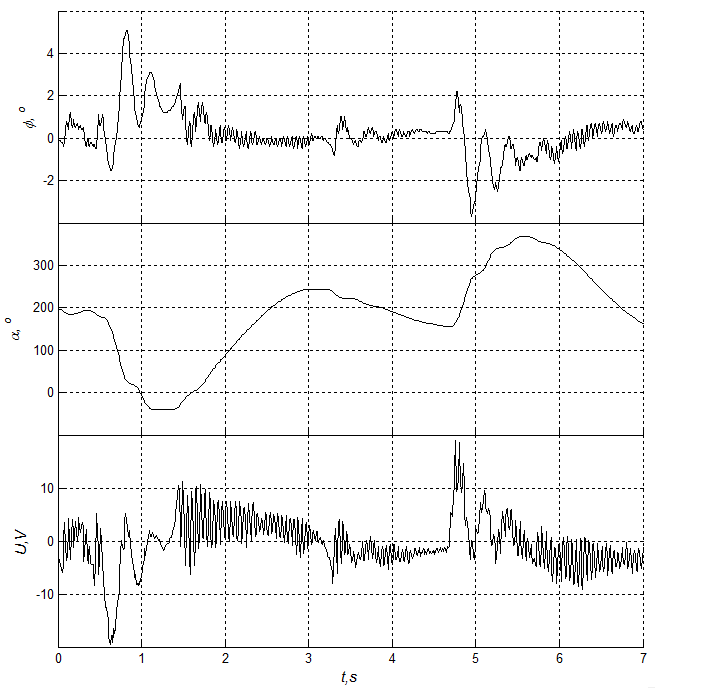


Рис.9.12 Результат работы системы стабилизации: отработка возмущения.

На рисунке 9.13 приведены данные о работе системы в случае, когда на платформе была размещена дополнительная нагрузка (масса нагрузки около 300г, т.е 60% от массы платформы). Система стабилизации успешно справилась с дополнительной нагрузкой, однако существенно возросла статическая ошибка угла поворота колеса. Связано это с тем, что нагрузка была закреплена с одной стороны платформы, при этом центр масс платформы сместился в сторону нагрузки, а это, в свою очередь, привело к увеличению величины смещения датчика угла наклона (до 4о), поскольку датчик был откалиброван по обычному положению центра масс.

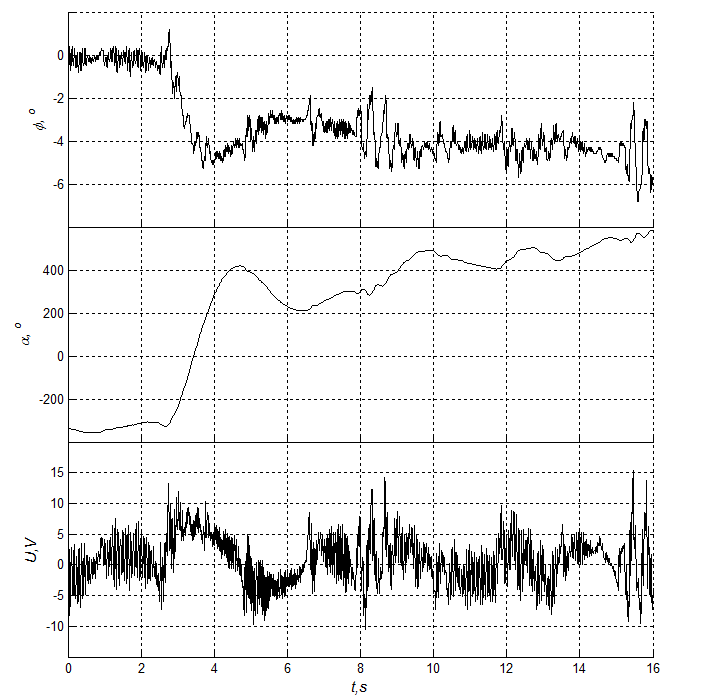


Рис.9.13 Результат работы системы стабилизации: дополнительная нагрузка.

По результатам испытания работы системы можно сделать вывод, что полученная система стабилизация эффективно отрабатывает не только различные начальные условия, но так же способна обеспечить устойчивость при действии возмущений и приложении дополнительной нагрузки.

## Реализация режимов перемещения и вращения

Для реализации перемещения платформы был выбран следующий алгоритм управления:

|  |  |
| --- | --- |
|  | (9.6) |

Однако напрямую данный алгоритм не может быть применен. При текущем алгоритме работы величина уставки определяет угол поворота колеса, который должна поддерживать платформа, а значит – и расстояние, пройденное платформой. Однако при увеличении значения уставки возрастает и максимальный угол, на который отклоняется платформа при движении из начального положения в заданное. Поскольку максимально допустимый угол отклонения ограничен, то ограничен и максимальный угол поворота колеса. Для решения этой проблемы алгоритм (9.6) был скорректирован следующим образом:

|  |  |
| --- | --- |
|  | (9.7) |

То есть, в процессе работы значение уставки непрерывно увеличивается, а скорость движения определяется скоростью роста уставки. При таком подходе ограничена лишь максимальная скорость движения, а не максимальное расстояние, на которое может перемещаться платформа.

На рисунке 9.14 приведены данные о работе системы при движении, а на рисунке 9.15 – при действии возмущения, вызванного неровностью поверхности (высота препятствия около 7мм, ширина 15 мм), по которой перемещалась платформа. Как видно из рисунка, возмущение было успешно подавлено.

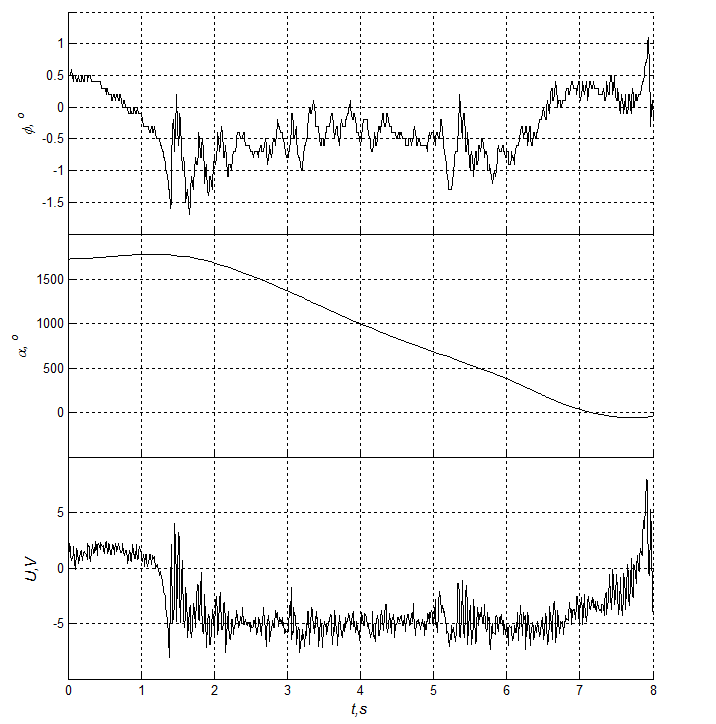


Рис.9.14 Результат работы системы стабилизации в режиме перемещения.

Для реализации вращения может быть применен модифицированный вариант алгоритма (9.7):

|  |  |
| --- | --- |
|  | (9.8) |

В этом случае направления вращения колес будут различными, что приведет к повороту платформы. Комбинирование алгоритмов (9.7) и (9.8) позволяет осуществлять движение с поворотом.

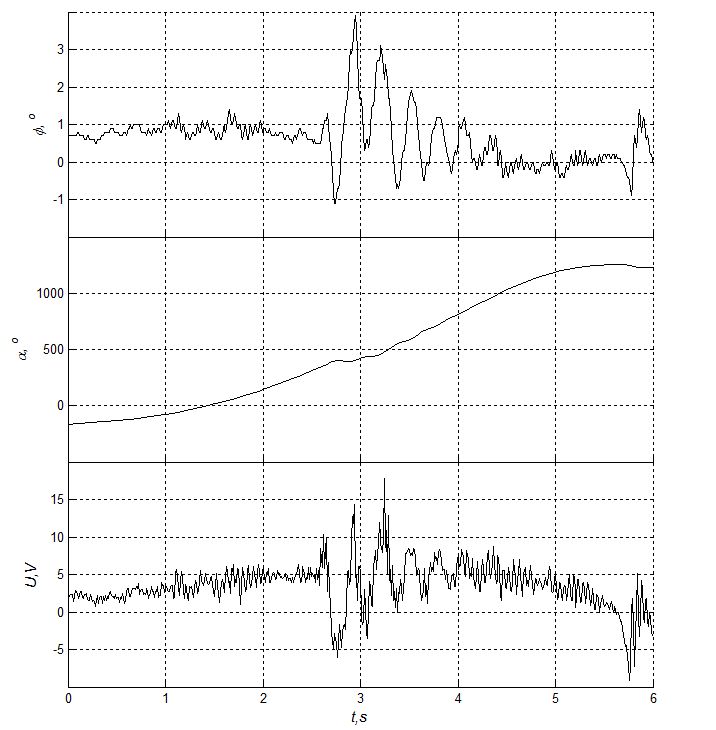


Рис.9.15 Результат работы системы стабилизации в режиме перемещения: отработка возмущения.

В общем виде алгоритм управления имеет вид:

|  |  |
| --- | --- |
|  | (9.9) |

На рисунке 9.16 приведены данные о работе системы при повороте платформы.

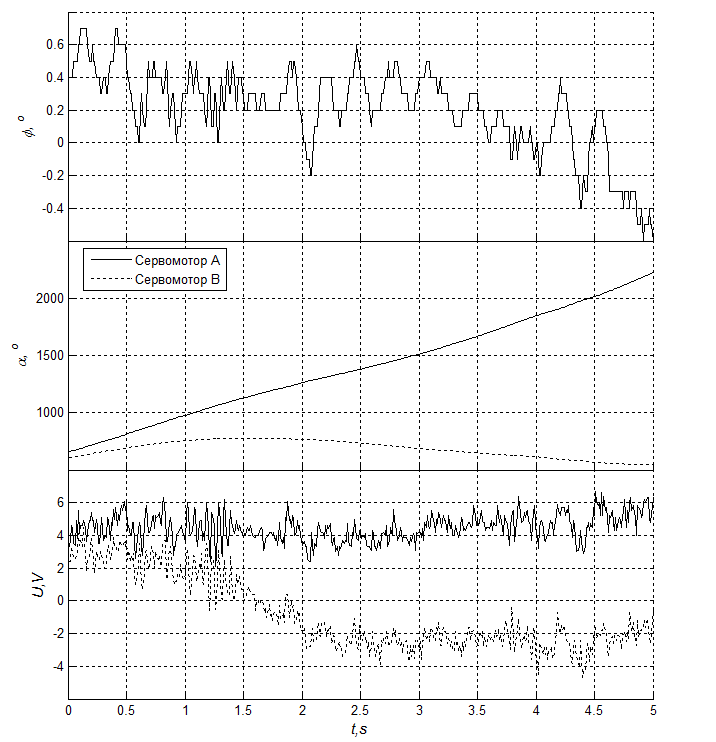


Рис.9.16 Результат работы системы стабилизации в режиме вращения.

# Заключение

В результате проделанной работы был разработан алгоритм работы системы стабилизации двухколесной платформы. При разработке алгоритма была получена математическая модель платформы как объекта класса «перевернутый маятник». Получены численные значения параметров модели, по которым были рассчитаны параметры регулятора на основе модального метода синтеза.

Разработанный регулятор был реализован на микроконтроллере STM32F100RBT6B на языке программирования СИ++. После испытаний на реальной системе, значения коэффициентов регулятора были скорректированы для обеспечения необходимого качества работы системы.

Разработанный алгоритм позволяет осуществлять управление системы в режиме балансировки, а так же в режимах перемещения и вращения. Система стабилизации обеспечивает эффективное подавление возмущений, связанных с неровностями поверхности, по которой перемещается платформа, и действием внешних сил, приложенных к платформе. Диапазон отработки начальных условий и возмущений по углу наклона составляет до 7-8о. Так же, система стабилизации позволяет размещать на платформе дополнительную нагрузку, массой равной массе платформы.

По результатам работы можно сделать вывод, что модальный метод не всегда дает приемлемые результаты для объекта типа «перевернутый маятник», к которому относится платформа. Связано это с тем, что при расчетах используется линеаризованная модель. Реальный же объект обладает целым рядом нелинейностей, которые ухудшают работу системы стабилизации. Однако, принцип организации обратной связи, который предполагает модальный метод синтеза, позволяет добиться хорошего результата даже для нелинейного объекта. Данный метод является удобным в реализации для системы управления двухколесной платформой, поскольку значения переменных состояния могут быть измерены напрямую с помощью датчиков. Структура же модального регулятора позволяет корректировать его коэффициенты, анализируя влияние на работу системы реальных физических величин, которым соответствуют коэффициенты.

Данный объект управления предполагается использовать в будущем в качестве лабораторного стенда для изучения теории автоматического управления магистрантами. Платформа является устройством, на котором могут быть поэтапно отработаны навыки синтеза систем управления и работы с датчиками: есть возможность реализовать управление двигателями, дифференцирующий фильтр или комплементарный фильтр. Помимо модального алгоритма, на платформе могут быть реализованы и другие, например ПИД-регулятор или алгоритм на основе скользящего режима.

Результаты, полученные в данной работе предполагается использовать в проектах, связанных с разработкой шагающих механизмов и экзоскелетов, поскольку данные устройства имеют схожую математическую модель, но более сложную, а именно – объект класса «многозвенный перевернутый маятник». Сама платформа может быть использована в качестве испытательного стенда для отработки алгоритмов управления.

# Список литературы

1. Федоров Д.С. Разработка аппаратной части и программного обеспечения системы стабилизации двухколесной платформы на базе микроконтроллера STM32. – магистерская диссертация. – Новосибирск, 2015 г.
2. Саблина Г.В. О некоторых свойствах системы «двухзвенный перевернутый маятник на тележке». – Автоматика и программная инженерия. 2014, №2(8), стр. 9-12
3. Формальский А.М. Управление движением неустойчивых объектов. – М.: ФИЗМАТЛИТ, 2012. – 232 с. – ISBN 978-5-9221-1460-8
4. Саблина Г.В. Разработка и исследование методики стабилизации объекта управления «каретка-маятник»: Автореф. дис. … канд. техн. наук. – Новосибирск, 2000 г.
5. Основы теории непрерывных и дискретных систем регулирования. – 5-е изд., перераб. и доп.: учеб. пособие / А.С. Востриков, Г.А. Французова, Е.Б. Гаврилов. – Новосибирск : Изд-во НГТУ, 2008. – 476 с. ISBN 978-5-7782-1129-9
6. Федоров Д.С., Ивойлов А.Ю., В.А. Жмудь, В.Г. Трубин Использование акселерометра ADXL335 для определения угла отклонения от вертикали. – Автоматика и программная инженерия. 2014, №2(8), стр. 68-72
7. MPU-6000 and MPU-6050 Register Maps and Descriptions [Электронный ресурс]. – режим доступа : <http://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/>
8. Жмудь В.А. Моделирование и оптимизация систем управления лазерным излучением в среде VisSim: учеб. пособие / В.А. Жмудь.– Новосибирск, Изд-во НГТУ, 2012. – 124 с. ISBN 978-5-7782-2101-1

# Приложение

# Приложение А

Содержит принципиальную схему устройства и перечень используемых элементов.

# Приложение Б

Приложение содержит тексты программ заголовочных и исполняемых файлов.

Файл main.cpp

/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Проект управляющей программы балансирующего двухколесного робота

Устройство: Отладочная плата STM32VLDISCOVERY

Заказчик: НГТУ

Микроконтроллер: STM32F100RBT6B

Язык программирования: C++

Среда разработки: Eclipse Helios SR2

Набор средств компиляции: Sourcery CodeBench Lite for ARM EABI

Авторы: Федоров Дмитрий Сергеевич, Ивойлов Андрей Юрьевич,

Жмудь Вадим Аркадьевич, Трубин Виталий Геннадьевич

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/

#include <stm32f10x.h>

#include "MotorClass.hpp"

#include "BluetoothClass.hpp"

#include "MPU6050.hpp"

#include "Radio.hpp"

#define CyclesToTranferData 10 // Счетчик номера очереди передачи данных

#define CyclesToUpdateTimeout 10 // Делитель частоты инкремента счетчика времени ожидания ответа MPU6050

#define SysTickDivider 24000 // Делитель частоты системного таймера

#define BIAS\_ANGLE 1110 // Значение смещения угла наклона платформы

const int32\_t K\_platform\_angle = 54000; // 54.00

const int32\_t K\_platform\_speed = 1350; // 1.35

const int32\_t K\_wheel\_angle = 270; // 0.27

const int32\_t K\_wheel\_speed = 300; // 0.30

#define Main\_Algorithm

// Функция инициализации микроконтроллера

void System\_Init(void);

// Переменная-Флаг срабатывания прерывания системного таймера

bool flag = false;

// Переменная-счетчик времени ожидания модуля MPU6050

extern uint16\_t MPU6050\_TimeOut;

#ifdef Main\_Algorithm

int main()

{

// Установка частоты переполнения системного таймера равной

// 24000000/SysTickDivider

SysTick\_Config(SysTickDivider);

// Инициализация системы тактирования

System\_Init();

// Инициализация приемного устройства связи по радиоканалу

cRadio.Init();

// Инициализация датчика MPU6050

cMPU6050.Init();

// Инициализация сервомоторов А и В

cMotorA.Init();

cMotorB.Init();

// Инициализация Bluetooth-модуля

cBluetooth.Init();

// Счетчик для определения частоты передачи данных

uint8\_t data\_transfer\_cnt = 0;

// Определение угла отклонения в начальный момент времени

cMPU6050.ReadAxis(Axis\_Acc\_X);

cMPU6050.ReadAxis(Axis\_Acc\_Z);

cMPU6050.Calc\_Angel\_Acc();

cMPU6050.\_angle\_acc += BIAS\_ANGLE;

// Калибровка параметров системы радиоуправления

cRadio.Calib();

// Задание начальных условий

cMPU6050.\_angle\_gyr = cMPU6050.\_angle\_acc\*100;

cMPU6050.\_angle\_abfilter = cMPU6050.\_angle\_acc\*100;

// Переменные составляющих управляющего воздействия

int32\_t ControlAction\_A = 0;

int32\_t ControlAction\_B = 0;

int32\_t Control\_1 = 0;

int32\_t Control\_2 = 0;

int32\_t Control\_3 = 0;

int32\_t Control\_4 = 0;

int32\_t Speed\_A = 0;

int32\_t Speed\_B = 0;

// Основной программный цикл

while(1){

//Передача данных выполняется с частотой 24000000/SysTickDivider

if(flag == true){

flag = false;

// Считываются показания акселерометра по осям X и Z,

// а также гироскопа относительно оси Y

cMPU6050.ReadAxis(Axis\_Acc\_X);

cMPU6050.ReadAxis(Axis\_Acc\_Z);

cMPU6050.ReadAxis(Axis\_Gyr\_Y);

// Определение угла с помощью акселерометра,

// гироскопа и альфа-бета фильтра

cMPU6050.Calc\_Angel\_Acc();

cMPU6050.\_angle\_acc += BIAS\_ANGLE;

cMPU6050.Calc\_Angel\_Gyr();

cMPU6050.A\_B\_Filter();

// ------------ Модальный регулятор ------------------------

// Вычисление составляющей по углу наклона

Control\_1 = -(((cMPU6050.\_angle\_abfilter +10000)/1000)\*K\_platform\_angle)/10/100;

// Вычисление составляющей по производной угла наклона

Control\_2 = -(((cMPU6050.\_speed\_gyr + 100)/100)\*K\_platform\_speed)/100;

// Сервомотор А

// Составляющая по скорости вращения колеса А

Control\_3 = cMotorA.uiMotorSpeedFilter\*K\_wheel\_angle/100;

// Вычисление уставки по углу колеса А

Speed\_A += (-cRadio.RemoteControl\_F\_B()/3 - cRadio.RemoteControl\_L\_R()/3)\*100;

// Составляющая по углу поворота колеса А

Control\_4= Speed\_A/50000 + cMotorA.iMotorAngel\*K\_wheel\_speed/100;

// Вычисление управления А

ControlAction\_A = Control\_1 + Control\_2 + Control\_3 + Control\_4;

//ControlAction\_A = ControlAction\_A\*6/10;

// Сервомотор Б

// Составляющая по скорости вращения колеса Б

Control\_3 = cMotorB.uiMotorSpeedFilter\*K\_wheel\_angle/100;

// Вычисление уставки по углу колеса Б

Speed\_B += (-cRadio.RemoteControl\_F\_B()/3 +

cRadio.RemoteControl\_L\_R()/3)\*100;

// Составляющая по углу поворота колеса Б

Control\_4 = Speed\_B/50000 + cMotorB.iMotorAngel\*K\_wheel\_speed/100;

// Вычисление управления Б

ControlAction\_B = Control\_1 + Control\_2 + Control\_3 + Control\_4;

// Приложение управляющего воздействия

cMotorA.MotorAMoveOn(ControlAction\_B/10);

cMotorB.MotorBMoveOn(ControlAction\_A/10);

// ------------ Отправка данных ------------------------

// Передача угла наклона

if(data\_transfer\_cnt == 0){

cBluetooth.Transmit\_Str(cBluetooth.IntToStr((cMPU6050.\_angle\_abfilter/100)));

cBluetooth.Transmit\_Str("\t");

}

// Передача производной угла наклона

if(data\_transfer\_cnt == 1){

cBluetooth.Transmit\_Str(cBluetooth.IntToStr(cMPU6050.\_angle\_gyr/100));

cBluetooth.Transmit\_Str("\t");

}

// Передача скорости вращения колеса А

if(data\_transfer\_cnt == 2){

cBluetooth.Transmit\_Str(cBluetooth.IntToStr

(cMotorA. uiMotorSpeedFilter));

cBluetooth.Transmit\_Str("\t");

}

// Передача угла поворота колеса А

if(data\_transfer\_cnt == 3){

cBluetooth.Transmit\_Str(cBluetooth.IntToStr(cMotorA.iMotorAngel));

cBluetooth.Transmit\_Str("\t");

}

// Передача скорости вращения колеса Б

if(data\_transfer\_cnt == 4){

cBluetooth.Transmit\_Str(cBluetooth.IntToStr

(cMotorB. uiMotorSpeedFilter));

cBluetooth.Transmit\_Str("\t");

}

// Передача угла поворота колеса Б

if(data\_transfer\_cnt == 5){

cBluetooth.Transmit\_Str(cBluetooth.IntToStr(cMotorB.iMotorAngel));

cBluetooth.Transmit\_Str("\t");

}

// Если прошло 10 циклов работы алгоритма, отправка данных повторяется

// с угла наклона

if(data\_transfer\_cnt < 9){

data\_transfer\_cnt++;

}

else{

data\_transfer\_cnt = 0;

cBluetooth.Transmit\_Str("\r");

}

}

}

}

#endif

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* Функция инициализации микроконтроллера

\* Назначение: выбор HSE как источника системной частот

\* устанановка системной частоты равной 24МГц

\* настройка расположение таблицы

\* векторов прерываний

\* Входные параметры: Нет

\* Выходные параметры: Нет

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

void System\_Init(){

//Включение HSE (High speed external clock)

RCC->CR |= RCC\_CR\_HSEON;

// Ожидание готовности HSE

while(!(RCC->CR & RCC\_CR\_HSERDY));

// Установка частоты линии HCLK равной системной SYSCLK

RCC->CFGR |= RCC\_CFGR\_HPRE\_DIV1;

// Установка частоты линии PCLK2 равной частоте HCLK

RCC->CFGR |= RCC\_CFGR\_PPRE2\_DIV1;

// Установка частоты линии PCLK2 равной частоте HCLK

RCC->CFGR |= RCC\_CFGR\_PPRE1\_DIV1;

// Сброс настроек модуля умножения частоты PLL

RCC->CFGR &= ~(RCC\_CFGR\_PLLSRC | RCC\_CFGR\_PLLXTPRE | RCC\_CFGR\_PLLMULL);

// Установка выходной частоты PLL равной 24МГц ((HSE/2)\*6)

RCC->CFGR |= (RCC\_CFGR\_PLLSRC\_PREDIV1 | RCC\_CFGR\_PLLXTPRE\_PREDIV1\_Div2 | RCC\_CFGR\_PLLMULL6);

// Включение PLL

RCC->CR |= RCC\_CR\_PLLON;

// Ожидание готовности PLL

while(!(RCC->CR & RCC\_CR\_PLLRDY));

// Выбор выхода PLL как источника системной частоты

RCC->CFGR &= ~(RCC\_CFGR\_SW);

RCC->CFGR |= RCC\_CFGR\_SW\_PLL;

// Ожидание готовности выхода PLL в качестве источника системной частоты

while ((RCC->CFGR & RCC\_CFGR\_SWS) != RCC\_CFGR\_SWS\_PLL);

// Настройка таблицы векторов прерывания: размещение во внутренней FLASH

SCB->VTOR = FLASH\_BASE | 0x0;

}

// Данное прерывание вызывается с частотой 24000000/SysTickDivider,

// что определяет интенсивность опроса датчика в единицу времени и число

// передач этих данных по UART (Bluetooth)

extern "C" void SysTick\_Handler()

{

static uint16\_t systick\_cnt=0;

//Поднять флаг срабатывания прерывания

flag = true;

systick\_cnt++;

// Если число вызовов прерывания достигло заданного значения,

// то переменная MPU6050\_TimeOut инкрементируется. Если датчик завис,

// и не происходит передачи данных, то эта переменная не обнуляется и

// в результате достигнет граничного значения, при котором датчик

// будет перезагружен.

if(systick\_cnt>=CyclesToUpdateTimeout){

systick\_cnt=0;

MPU6050\_TimeOut++;

}

}

Файл MPU6050.cpp

/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Исполняемый файл драйвера датчика MPU6050

Устройство: Отладочная плата STM32VLDiscovery

Заказчик: НГТУ

Микроконтроллер: STM32F100RBT6B

Язык программированиЯ: C++

Среда разработки: Eclipse Helios SR2

Набор средств компиляции: Sourcery CodeBench Lite for ARM EABI

Авторы: Федоров Дмитрий Сергеевич, Ивойлов Андрей Юрьевич,

Жмудь Вадим Аркадьевич, Трубин Виталий Геннадьевич

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/

#include "MPU6050.hpp"

// Объявление объекта класса

MPU6050 cMPU6050;

// Пременная для отсчета времени ожидания

uint16\_t MPU6050\_TimeOut;

/\* Функция инициализации модуля

\* Настройка I2C-периферии и программная настройка

\*/

void MPU6050::Init(void)

{

Error\_Status Error;

// Ожидаем 100 мс

MPU6050\_TimeOut = 0;

while(MPU6050\_TimeOut < 12);

// Разрешаем тактирование I2C и порта GPIOB

RCC->APB1ENR |= RCC\_APB1ENR\_I2C2EN;

RCC->APB2ENR |= RCC\_APB2ENR\_IOPBEN;

// Настройка ног PB10 и PB11 для работы с I2C-периферией МК

// PB10

GPIOB->CRH &= ~GPIO\_CRH\_MODE10;

GPIOB->CRH |= GPIO\_CRH\_MODE10\_1; // Выход 2 МГц

GPIOB->CRH |= GPIO\_CRH\_CNF10; // Альтернативная функция, open-drain

// PB11

GPIOB->CRH &= ~GPIO\_CRH\_MODE11; // Аналогично

GPIOB->CRH |= GPIO\_CRH\_MODE11\_1;

GPIOB->CRH |= GPIO\_CRH\_CNF11;

// Настройка параметров работы I2C-периферии МК

I2C2->CR2 = 0b011000; // Частота тактирования модуля 24Мгц

I2C2->CCR = 0x1E; // Частота SCL 400К, старое значение 0x3C,

I2C2->CCR &= ~I2C\_CCR\_FS; // Standard mode

I2C2->TRISE = 0b011001; // Rise time

I2C2->CR1 |= I2C\_CR1\_ACK; // Включить подтверждение

I2C2->OAR1 = 10; // Собственный адрес

I2C2->CR1 |= I2C\_CR1\_PE; // Включить переферию

do{

// Настройка параметров работы датчика

// Датчик тактируется от встроенного 8Мгц осциллятора

Error = cMPU6050.RegisterWrite(Register\_PWR\_M1,0x00);

if(Error == Error\_No){

// Выполнить очистку встроенных регистров датчика

Error = cMPU6050.RegisterWrite(Register\_UsCtrl,0x01);

}

}

while(Error != Error\_No);

// Ожидаем 10 мс

MPU6050\_TimeOut = 0;

while(MPU6050\_TimeOut < 3);

}

/\* Функция чтения регистра

\* Чтение данных из регистра по указанному адресу

\* Входные параметры: адрес регистра

\*/

uint8\_t MPU6050::RegisterRead(Register\_Sensor RegisterAddress)

{

uint8\_t temp;

I2C2->CR1 |= I2C\_CR1\_PE;

// формирование сигнала старт

I2C2->CR1 |= I2C\_CR1\_START;

// ждем окончания формирования сигнала "Старт"

MPU6050\_TimeOut = 0;

while((!(I2C2->SR1 & I2C\_SR1\_SB))&&(MPU6050\_TimeOut<10));

if(MPU6050\_TimeOut >= 10)

{

// Если время ожидания истекло - перезапускем I2C

I2C2->CR1 |= I2C\_CR1\_SWRST;

MPU6050\_TimeOut = 0;

while(MPU6050\_TimeOut < 1);

I2C2->CR1 &= ~I2C\_CR1\_SWRST;

this->Init();

}

else

{

// Передаем адрес на чтение

MPU6050\_TimeOut = 0;

I2C2->DR = (uint8\_t)(Address\_Write);

while( (!(I2C2->SR1 & (I2C\_SR1\_AF | I2C\_SR1\_ADDR))) &&

(MPU6050\_TimeOut<10));

if(MPU6050\_TimeOut >= 10)

{

// Если время ожидания истекло - перезапускем I2C

I2C2->CR1 |= I2C\_CR1\_SWRST;

MPU6050\_TimeOut = 0;

while(MPU6050\_TimeOut < 1);

I2C2->CR1 &= ~I2C\_CR1\_SWRST;

this->Init();

}

else

{

// Проверка, принят ли адрес

if(I2C2->SR1 & I2C\_SR1\_ADDR)

{

I2C2->SR1 &= ~I2C\_SR1\_ADDR;

// Важно! Здесь необходимо чтение регистра SR2

(void)(I2C2->SR2);

// Пишем адрес регситра

I2C2->DR = (uint8\_t)(RegisterAddress);

MPU6050\_TimeOut = 0;

while((!(I2C2->SR1 & I2C\_SR1\_TXE)) && (MPU6050\_TimeOut < 10));

if(MPU6050\_TimeOut >= 10)

{

// Если время ожидания истекло - перезапускем I2C

I2C2->CR1 |= I2C\_CR1\_SWRST;

MPU6050\_TimeOut = 0;

while(MPU6050\_TimeOut < 1);

I2C2->CR1 &= ~I2C\_CR1\_SWRST;

this->Init();

}

else

{

// Ставим заново старт-условие

I2C2->CR1 |= I2C\_CR1\_START;

// ждем окончания формирования сигнала "Старт"

MPU6050\_TimeOut = 0;

while ((!(I2C2->SR1 & I2C\_SR1\_SB))&&(MPU6050\_TimeOut < 10));

if(MPU6050\_TimeOut >= 10)

{

I2C2->CR1 |= I2C\_CR1\_SWRST;

MPU6050\_TimeOut = 0;

while(MPU6050\_TimeOut < 1);

I2C2->CR1 &= ~I2C\_CR1\_SWRST;

this->Init();

}

else

{

// Отправляем адрес на считывание

I2C2->CR1 &= ~I2C\_CR1\_ACK;

I2C2->DR = (uint8\_t)(Address\_Read);

MPU6050\_TimeOut = 0;

while((!(I2C2->SR1 & (I2C\_SR1\_AF | I2C\_SR1\_ADDR))) &&

(MPU6050\_TimeOut < 10));

if(MPU6050\_TimeOut >= 10)

{

I2C2->CR1 |= I2C\_CR1\_SWRST;

MPU6050\_TimeOut = 0;

while(MPU6050\_TimeOut < 1);

I2C2->CR1 &= ~I2C\_CR1\_SWRST;

this->Init();

}

else{

// Проверка, принят ли адрес

if(I2C2->SR1 & I2C\_SR1\_ADDR)

{

I2C2->SR1 &= ~I2C\_SR1\_ADDR;

// Важно! Здесь необходимо чтение регистра SR2

(void)(I2C2->SR2);

// Важно! Ставим стоп-условие до чтения данных

I2C2->CR1 |= I2C\_CR1\_STOP;

MPU6050\_TimeOut = 0;

while((!(I2C2->SR1 & I2C\_SR1\_RXNE)) && (MPU6050\_TimeOut < 10));

if(MPU6050\_TimeOut >= 10)

{

I2C2->CR1 |= I2C\_CR1\_SWRST;

MPU6050\_TimeOut = 0;

while(MPU6050\_TimeOut < 1);

I2C2->CR1 &= ~I2C\_CR1\_SWRST;

this->Init();

}

else

{

// Читаем данные

temp = I2C2->DR;

}

}

else

{

I2C2->SR1 &= ~I2C\_SR1\_AF;

}

}

}

}

}

else

{

I2C2->SR1 &= ~I2C\_SR1\_AF;

}

}

}

return temp;

}

/\* Функция чтения данных по оси датчика

\* Чтение данных указанной оси

\* Входные параметры: ось датчика

\*/

Error\_Status MPU6050::ReadAxis(Axis\_Type Axis){

int16\_t temp\_data;

uint8\_t temp\_reg;

// Выбор адреса регистра данных оси, в зависимости от требуемой

switch(Axis){

case Axis\_Acc\_X:{

temp\_reg = (uint8\_t)(Register\_Acc\_XH);

break;

}

case Axis\_Acc\_Y:{

temp\_reg = (uint8\_t)(Register\_Acc\_YH);

break;

}

case Axis\_Acc\_Z:{

temp\_reg = (uint8\_t)(Register\_Acc\_ZH);

break;

}

case Axis\_Gyr\_X:{

temp\_reg = (uint8\_t)(Register\_Gyr\_XH);

break;

}

case Axis\_Gyr\_Y:{

temp\_reg = (uint8\_t)(Register\_Gyr\_YH);

break;

}

case Axis\_Gyr\_Z:{

temp\_reg = (uint8\_t)(Register\_Gyr\_ZH);

break;

}

}

Error\_Status Error;

// Включаем периферию

I2C2->CR1 |= I2C\_CR1\_PE;

// Старт-условие

Error = this->I2C\_SetStartCondition();

if(Error == Error\_Start){

return Error;

}

else{

// Передаем адрес на чтение

Error = this->I2C\_SendAddress(Address\_Write);

if(Error != Error\_No){

return Error;

}

else{

// Важно! Читаем SR2

(void)(I2C2->SR2);

I2C2->DR = temp\_reg;

MPU6050\_TimeOut = 0;

while((!(I2C2->SR1 & I2C\_SR1\_TXE)) && (MPU6050\_TimeOut < 10));

if(MPU6050\_TimeOut >= 10){

this->I2C\_Reset();

return Error\_I2C;

}

else{

// Старт-условие

Error = this->I2C\_SetStartCondition();

if(Error != Error\_No){

return Error;

}

else{

// Отправляем адрес на считывание

Error = this->I2C\_SendAddress(Address\_Read);

if(Error != Error\_No){

return Error;

}

else{

I2C2->CR1 |= I2C\_CR1\_ACK;

// Важно! Читаем SR2

(void)(I2C2->SR2);

MPU6050\_TimeOut = 0;

while((!(I2C2->SR1 & I2C\_SR1\_RXNE)) && (MPU6050\_TimeOut < 10));

if(MPU6050\_TimeOut >= 10){

this->I2C\_Reset();

return Error\_I2C;

}

else{

I2C2->CR1 &= ~I2C\_CR1\_ACK;

// Важно! Ставим стоп-условие до прочтения последнего байта!

I2C2->CR1 |= I2C\_CR1\_STOP;

// Читаем первый байт

temp\_data = I2C2->DR;

temp\_data <<= 8;

MPU6050\_TimeOut = 0;

while((!(I2C2->SR1 & I2C\_SR1\_RXNE)) && (MPU6050\_TimeOut < 10));

if(MPU6050\_TimeOut >= 10){

this->I2C\_Reset();

return Error\_I2C;

}

else{

// Читаем второй байт

temp\_data |= I2C2->DR;

// Пишем в нужное поле класса

switch(Axis){

case Axis\_Acc\_X:{

this->\_acc\_x = temp\_data;

break;

}

case Axis\_Acc\_Y:{

this->\_acc\_y = temp\_data;

break;

}

case Axis\_Acc\_Z:{

this->\_acc\_z = temp\_data;

break;

}

case Axis\_Gyr\_X:{

this->\_gyr\_x = temp\_data;

break;

}

case Axis\_Gyr\_Y:{

this->\_gyr\_y = temp\_data;

break;

}

case Axis\_Gyr\_Z:{

this->\_gyr\_z = temp\_data;

break;

}

}

return Error\_No;

}

}

}

}

}

}

}

}

/\* Функция установки старт-условия I2C

\*/

Error\_Status MPU6050::I2C\_SetStartCondition(){

// формирование сигнала старт

I2C2->CR1 |= I2C\_CR1\_START;

// ждем окончания формирования сигнала "Старт"

MPU6050\_TimeOut = 0;

while((!(I2C2->SR1 & I2C\_SR1\_SB))&&(MPU6050\_TimeOut<10));

if(MPU6050\_TimeOut >= 10){

this->I2C\_Reset();

return Error\_Start;

}

else{

return Error\_No;

}

}

/\* Функция программного сброса I2C

\*/

void MPU6050::I2C\_Reset(){

// Сброс периферии

I2C2->CR1 |= I2C\_CR1\_SWRST;

// Ожидания паузы

MPU6050\_TimeOut = 0;

while(MPU6050\_TimeOut < 1);

// Инициализация

I2C2->CR1 &= ~I2C\_CR1\_SWRST;

this->Init();

}

/\* Функция отправки адреса ведомого I2C

\* Входные параметры: тип адреса

\*/

Error\_Status MPU6050::I2C\_SendAddress(Address\_Type Address){

// Передаем адрес на чтение

MPU6050\_TimeOut = 0;

// Пишем адрес ведомого в регистр

I2C2->DR = (uint8\_t)(Address);

// Ждем отправки

while((!(I2C2->SR1 & (I2C\_SR1\_AF | I2C\_SR1\_ADDR))) && (MPU6050\_TimeOut<10));

// Проверка, итсекло ли время ожидания

if(MPU6050\_TimeOut >= 10){

// Если время ожидания истекло - реинициализация устройства

this->I2C\_Reset();

return Error\_Address;

}

else{

// Проверка, принят ли адрес ведомого

if(I2C2->SR1 & I2C\_SR1\_ADDR){

I2C2->SR1 &= ~I2C\_SR1\_ADDR;

return Error\_No;

}

else{

I2C2->SR1 &= ~I2C\_SR1\_AF;

return Error\_Ack;

}

}

}

/\* Функция записи в регистр

\* Входные параметры: адрес регистра, данные для записи

\*/

Error\_Status MPU6050::RegisterWrite(Register\_Sensor RegisterAddress,

uint8\_t Data){

Error\_Status Error;

I2C2->CR1 |= I2C\_CR1\_PE;

// Старт-условие

Error = this->I2C\_SetStartCondition();

if(Error == Error\_Start){

return Error;

}

else{

// Передаем адрес на чтение

Error = this->I2C\_SendAddress(Address\_Write);

if(Error != Error\_No){

return Error;

}

else{

// Важно! Читаем регистр SR2

(void)(I2C2->SR2);

// Пишем адрес регистра

I2C2->DR = (uint8\_t)(RegisterAddress);

MPU6050\_TimeOut = 0;

while((!(I2C2->SR1 & I2C\_SR1\_TXE)) && (MPU6050\_TimeOut < 10));

if(MPU6050\_TimeOut >= 10){

this->I2C\_Reset();

return Error\_I2C;

}

else{

// Пишем данные

I2C2->DR = (uint8\_t)(Data);

MPU6050\_TimeOut = 0;

while((!(I2C2->SR1 & I2C\_SR1\_TXE)) && (MPU6050\_TimeOut < 10));

if(MPU6050\_TimeOut >= 10){

this->I2C\_Reset();

return Error\_I2C;

}

else{

// Ставим стоп-условие

I2C2->CR1 |= I2C\_CR1\_STOP;

}

}

}

}

}

/\* Функция измерения акселерометром

\*/

void MPU6050::Measure\_Acc(void)

{

\_acc\_x = cMPU6050.RegisterRead(Register\_Acc\_XH);

\_acc\_x <<= 8;

\_acc\_x |= cMPU6050.RegisterRead(Register\_Acc\_XL);

\_acc\_y = cMPU6050.RegisterRead(Register\_Acc\_YH);

\_acc\_y <<= 8;

\_acc\_y |= cMPU6050.RegisterRead(Register\_Acc\_YL);

\_acc\_z = cMPU6050.RegisterRead(Register\_Acc\_ZH);

\_acc\_z <<= 8;

\_acc\_z |= cMPU6050.RegisterRead(Register\_Acc\_ZL);

}

/\* Функция измерения гироскопом

\*/

void MPU6050::Measure\_Gyr(void)

{

\_gyr\_x = cMPU6050.RegisterRead(Register\_Gyr\_XH);

\_gyr\_x <<= 8;

\_gyr\_x |= cMPU6050.RegisterRead(Register\_Gyr\_XL);

\_gyr\_y = cMPU6050.RegisterRead(Register\_Gyr\_YH);

\_gyr\_y <<= 8;

\_gyr\_y |= cMPU6050.RegisterRead(Register\_Gyr\_YL);

\_gyr\_z = cMPU6050.RegisterRead(Register\_Gyr\_ZH);

\_gyr\_z <<= 8;

\_gyr\_z |= cMPU6050.RegisterRead(Register\_Gyr\_ZL);

}

/\* Функция вычисления угла и угловой скорости по гироскопу

\*/

void MPU6050::Calc\_Angel\_Gyr(void)

{

// Скорость в [градусы\*100/с]

\_speed\_gyr = \_gyr\_y \* 25000 / 32768;

// шаг интегрирования 0.001 с, для сохранения точности умножаем на 100

// В результате [угол\*10000]

\_angle\_gyr = \_angle\_gyr + \_speed\_gyr / 10;

}

/\* Функция вычисления угла акселерометра

\* Вычисляет угол по значению угла и квадранта (0-45 градусов -> 0-360 градусов)

\*/

void MPU6050::Calc\_Angel\_Acc(void)

{

int16\_t x;

int16\_t z;

x = -this->\_acc\_x;

z = -this->\_acc\_z;

// Первый квадрант

if ((x>=0)&&(z>=0))

{

// Граничные условия

if (x==0)

{

this->\_angle\_acc = 9000;

return;

}

if (z==0)

{

this->\_angle\_acc = 0;

return;

}

if (x==z)

{

this->\_angle\_acc = 4500;

return;

}

// Итоговове приведение

if(x>z)

{

this->\_angle\_acc = Calc\_Angle\_Quarter\_PI(x,z);

return;

}

else

{

this->\_angle\_acc = 9000 - Calc\_Angle\_Quarter\_PI(x,z);

return;

}

}

// Второй квадрант

if ((x>=0)&&(z<0))

{

z\*=-1;

// Граничные условия

if(x==0)

{

this->\_angle\_acc = - 9000;

return;

}

if(x==z)

{

this->\_angle\_acc = - 4500;

return;

}

// Итоговове приведение

if(x>z)

{

this->\_angle\_acc = -Calc\_Angle\_Quarter\_PI(x,z);

return;

}

else

{

this->\_angle\_acc = -9000 + Calc\_Angle\_Quarter\_PI(x,z);

return;

}

}

// Третий квадрант

if ((x<0)&&(z<=0))

{

x\*=-1;

z\*=-1;

// Граничные условия

if(z==0)

{

this->\_angle\_acc = 18000;

return;

}

if(x==z)

{

this->\_angle\_acc = -13500;

return;

}

// Итоговове приведение

if(x>z)

{

this->\_angle\_acc = -18000+Calc\_Angle\_Quarter\_PI(x,z);

return;

}

else

{

this->\_angle\_acc = -9000 - Calc\_Angle\_Quarter\_PI(x,z);

return;

}

}

// Четвертый квадрант

if ((x<0)&&(z>0))

{

x\*=-1;

// Граничные условия

if(x==z)

{

this->\_angle\_acc = 13500;

return;

}

// Итоговове приведение

if(x>z)

{

this->\_angle\_acc = 18000 - Calc\_Angle\_Quarter\_PI(x,z);

return;

}

else

{

this->\_angle\_acc = 9000 + Calc\_Angle\_Quarter\_PI(x,z);

return;

}

}

}

/\* Функция вычисления угла акселерометра

\* Вычисляет угол в пределах половины квадранта (0-45 градусов)

\*/

uint16\_t MPU6050::Calc\_Angle\_Quarter\_PI(int16\_t x,int16\_t z)

{

uint16\_t relation;

uint8\_t a=0,b=200,m;

// Вычисление отношения проекций.

// Если y>z, вычисление производится обычным образом,

// если z>y, вычисляется обратное отношение,

// которое соответствует не значению угла

// а значению 90 - [значение угла].

// Для сохранение точности производится сдвиг

// полученного значения отношения на 10 разрядов влево

if(z<x)

relation=(z<<10)/x;

if(z>x)

relation=(x<<10)/z;

// Выполняется поиск значения из массива array1,

// наиболее близкого к значению отношения

for (;b-a>1;)

{

// Вычисляется текущее значение индекса

m = (a+b)/2;

// Если точное равенству значению из массива с таким индексом –

// выход из цикла

if(array3[m]==relation)

return array4[m];

if(array3[m]>relation)

b = m;

else a = m;

}

if ((relation - array3[a])<(array3[b]-relation))

return array4[a];

else

return array4[b];

}

/\* Функция вычисления угла с помощью альфа-бета-фильтра

\*/

void MPU6050::A\_B\_Filter()

{

// Коэффициенты 99,9 и 0,1;

//(-32768..32768)\*999\*250\*10000/32768/1000/10

\_angle\_abfilter = ((999\*\_angle\_abfilter/10 + (2498\*\_gyr\_y)/328) + 10\*\_angle\_acc)/100;

}

Файл MPU6050.hpp

/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Заголовочный файл драйвера двигателей NXT

Устройство: Отладочная плата STM32VLDiscover

Заказчик: НГТУ

Микроконтроллер: STM32F100RBT6B

Язык программированиЯ: C++

Среда разработки: Eclipse Helios SR2

Набор средств компиляции: Sourcery CodeBench Lite for ARM EABI

Авторы: Федоров Дмитрий Сергеевич, Ивойлов Андрей Юрьевич,

Жмудь Вадим Аркадьевич, Трубин Виталий Геннадьевич

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/

#ifndef MPU6050\_HPP\_

#define MPU6050\_HPP\_

#include <stm32f10x.h>

// Перечилсения адресов ведомого

enum Address\_Type{

Address\_Write = 0xD0,

Address\_Read = 0xD1,

};

// Перечисление осей чувствительности MPU6050

enum Axis\_Type{

Axis\_Acc\_X,

Axis\_Acc\_Y,

Axis\_Acc\_Z,

Axis\_Gyr\_X,

Axis\_Gyr\_Y,

Axis\_Gyr\_Z,

};

// Перечисление адресов регистров

enum Register\_Sensor{

Register\_Config = 0x1A,

Register\_G\_Conf = 0x1B,

Register\_A\_Conf = 0x1C,

Register\_Acc\_XH = 0x3B,

Register\_Acc\_XL = 0x3C,

Register\_Acc\_YH = 0x3D,

Register\_Acc\_YL = 0x3E,

Register\_Acc\_ZH = 0x3F,

Register\_Acc\_ZL = 0x40,

Register\_Temp\_H = 0x41,

Register\_Temp\_L = 0x42,

Register\_Gyr\_XH = 0x43,

Register\_Gyr\_XL = 0x44,

Register\_Gyr\_YH = 0x45,

Register\_Gyr\_YL = 0x46,

Register\_Gyr\_ZH = 0x47,

Register\_Gyr\_ZL = 0x48,

Register\_SigRes = 0x68,

Register\_UsCtrl = 0x6A,

Register\_PWR\_M1 = 0x6B,

Register\_PWR\_M2 = 0x6C,

Register\_WhoAmI = 0x75,

};

// Перечисление ошибок

enum Error\_Status{

Error\_No = 0,

Error\_Start = -1,

Error\_Address = -2,

Error\_Ack = -3,

Error\_I2C = -4,

};

// Описание класса

class MPU6050

{

public:

// Текущие показания по осям акселерометра

int16\_t \_acc\_x;

int16\_t \_acc\_y;

int16\_t \_acc\_z;

// Текущие показания по осям гироскопа

int16\_t \_gyr\_x;

int32\_t \_gyr\_y;

int16\_t \_gyr\_z;

// Текущее значение угла по данным апериодического фильтра

int32\_t \_angle\_aper\_filter;

// Текущее значение угла по данным а-б-фильтра, [градусы\*10000]

int32\_t \_angle\_abfilter;

// Текущее значение угла по данным акселерометра, [градусы\*100]

int32\_t \_angle\_acc;

// Текущее значение угловой скорости по данным гироскопа,

// в [градусы/с\*100]

int32\_t \_speed\_gyr;

// Текущее значение угла по данным гироскопа [градусы\*10000]

int32\_t \_angle\_gyr;

// Текущее значение интеграла угла по данным а-б-фильтра

int32\_t \_angle\_summ;

// Функции I2C

// Инициализация датчика

void Init(void);

// Установка старт-условия

Error\_Status I2C\_SetStartCondition();

// Функция отправвки адреса ведомого

Error\_Status I2C\_SendAddress(Address\_Type Address);

// Программный сброс I2C-периферии

void I2C\_Reset();

// Функция чтения регистра

uint8\_t RegisterRead(Register\_Sensor RegisterAddress);

// Функция записи в регистр

Error\_Status RegisterWrite(Register\_Sensor RegisterAddress, uint8\_t Data);

// Функция чтения показаний по оси датчика

Error\_Status ReadAxis(Axis\_Type Axis);

// Функции измерений

// Измерение показаний акселерометра

void Measure\_Acc(void);

// Измерение показаний гироскопа

void Measure\_Gyr(void);

// Вычисление угла по акселерометру

void Calc\_Angel\_Acc(void);

// Вычисление угла и угловой скорости по гироскопу

void Calc\_Angel\_Gyr(void);

// Вычисление угла по тангенсу

static uint16\_t Calc\_Angle\_Quarter\_PI(int16\_t x,int16\_t z);

// Вычисление угла по альфа-бета-фильтру

void A\_B\_Filter();

};

const uint16\_t Delta\_t = 1000;

static const uint16\_t array3[201]{

0, 5, 10, 15, 20, 25, 30, 35, 40, 46,

51, 56, 61, 66, 71, 76, 81, 87, 92, 97,

102, 107, 112, 117, 122, 128, 133, 138, 143, 148,

153, 158, 163, 168, 174, 179, 184, 189, 194, 199,

204, 209, 215, 220, 225, 230, 235, 240, 245, 250,

256, 261, 266, 271, 276, 281, 286, 291, 296, 302,

307, 312, 317, 322, 327, 332, 337, 343, 348, 353,

358, 363, 368, 373, 378, 384, 389, 394, 399, 404,

409, 414, 419, 424, 430, 435, 440, 445, 450, 455,

460, 465, 471, 476, 481, 486, 491, 496, 501, 506,

512, 517, 522, 527, 532, 537, 542, 547, 552, 558,

563, 568, 573, 578, 583, 588, 593, 599, 604, 609,

614, 619, 624, 629, 634, 640, 645, 650, 655, 660,

665, 670, 675, 680, 686, 691, 696, 701, 706, 711,

716, 721, 727, 732, 737, 742, 747, 752, 757, 762,

768, 773, 778, 783, 788, 793, 798, 803, 808, 814,

819, 824, 829, 834, 839, 844, 849, 855, 860, 865,

870, 875, 880, 885, 890, 896, 901, 906, 911, 916,

921, 926, 931, 936, 942, 947, 952, 957, 962, 967,

972, 977, 983, 988, 993, 998, 1003, 1008, 1013, 1018,

1024

};

//Массив значений угла наклона, соответствующих значениям массива array1

static const uint16\_t array4[201]{

0, 28, 57, 85, 114, 143, 171, 200, 229, 257,

286, 314, 343, 371, 400, 428, 457, 485, 514, 542,

571, 599, 627, 656, 684, 712, 740, 768, 796, 825,

853, 881, 909, 936, 964, 992, 1020, 1048, 1075, 1103,

1130, 1158, 1185, 1213, 1240, 1268, 1295, 1322, 1349, 1376,

1403, 1430, 1457, 1484, 1510, 1537, 1564, 1590, 1617, 1643,

1669, 1696, 1722, 1748, 1774, 1800, 1826, 1852, 1877, 1903,

1929, 1954, 1979, 2005, 2030, 2055, 2080, 2105, 2130, 2155,

2180, 2204, 2229, 2253, 2278, 2302, 2326, 2350, 2374, 2398,

2422, 2446, 2470, 2493, 2517, 2540, 2564, 2587, 2610, 2633,

2656, 2679, 2702, 2724, 2747, 2769, 2792, 2814, 2836, 2859,

2881, 2903, 2924, 2946, 2968, 2989, 3011, 3032, 3054, 3075,

3096, 3117, 3138, 3159, 3179, 3200, 3221, 3241, 3261, 3282,

3302, 3322, 3342, 3362, 3382, 3401, 3421, 3441, 3460, 3479,

3499, 3518, 3537, 3556, 3575, 3594, 3612, 3631, 3650, 3668,

3686, 3705, 3723, 3741, 3759, 3777, 3795, 3813, 3830, 3848,

3865, 3883, 3900, 3918, 3935, 3952, 3969, 3986, 4003, 4019,

4036, 4053, 4069, 4085, 4102, 4118, 4134, 4150, 4166, 4182,

4198, 4214, 4230, 4245, 4261, 4276, 4292, 4307, 4322, 4338,

4353, 4368, 4383, 4397, 4412, 4427, 4442, 4456, 4471, 4485,

4500

};

extern MPU6050 cMPU6050;

#endif /\* MPU6050\_HPP\_ \*/

Файл MotorClass.cpp

/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Исполняемый файл драйвера двигателей NXT

Устройство: Отладочная плата STM32VLDiscover

Заказчик: НГТУ

Микроконтроллер: STM32F100RBT6B

Язык программированиЯ: C++

Среда разработки: Eclipse Helios SR2

Набор средств компиляции: Sourcery CodeBench Lite for ARM EABI

Авторы: Федоров Дмитрий Сергеевич, Ивойлов Андрей Юрьевич,

Жмудь Вадим Аркадьевич, Трубин Виталий Геннадьевич

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/

#define HARD\_CONTROL\_A

#define HARD\_CONTROL\_B

#include <stm32f10x.h>

#include "MotorClass.hpp"

#include "System.hpp"

#include "Accelerometer.hpp"

// Создание экземпляра класса двигателя А

MotorAClass cMotorA;

// Создание экземпляра класса двигателя B

MotorBClass cMotorB;

//--------------- РЕАЛИЗАЦИЯ МЕТОДОВ ДВИГАТЕЛЯ А -----------------------

/\* Конструктор драйвера двигателя А

\*/

MotorAClass::MotorAClass()

{

// Двигатель остановлен - напряжение равно нулю

this->uchMotorPower = 0;

// Текущее состояние двигателя

this->eState = eFreeStop;

// Текущий угол поворота

this->iMotorAngel = 0;

// Текущая скорость вращениявычисленная фильтром

this->uiMotorSpeedFilter = 0;

}

/\* Инициализация драйвера двигателя А

\*/

void MotorAClass::Init()

{

// Разрешение тактирования порта А и таймера TIM1

RCC->APB2ENR |= RCC\_APB2ENR\_IOPAEN | RCC\_APB2ENR\_TIM1EN;

// Настройка NVIC под таймер TIM1

// Установка наивысшего (нулевого) приоритета для прерывания по

// обновлению значения регистра счётчика таймера TIM1

NVIC->IP[TIM1\_UP\_TIM16\_IRQn] = 0ul << 4;

// Установка наивысшего (нулевого) приоритета для прерывания по равенству

// значений регистра сравнения и регистра счётчика

NVIC->IP[TIM1\_CC\_IRQn] = 0ul << 4;

// Разрешение прерывания по обновлению регистра счётчика

NVIC->ISER[0] |= 1ul << TIM1\_UP\_TIM16\_IRQn;

// Разрешение прерывания по равенству регистров сравнения и счётчика

NVIC->ISER[0] |= 1ul << TIM1\_CC\_IRQn;

// Настройка портов ввода вывода под выводы TIM1

// Очистка битов настройки

GPIOA->CRH &= ~(GPIO\_CRH\_MODE9 | GPIO\_CRH\_CNF9 |

GPIO\_CRH\_MODE11 | GPIO\_CRH\_CNF11);

// Пушпульный выход для периферии

GPIOA->CRH |= GPIO\_CRH\_MODE9 | GPIO\_CRH\_CNF9\_1 |

GPIO\_CRH\_MODE11 | GPIO\_CRH\_CNF11\_1;

// Настройка выводов 8 и 10 порта А под входы сигналов тахометра движка

// Очистка битов настройки

GPIOA->CRH &= ~(GPIO\_CRH\_MODE8 | GPIO\_CRH\_CNF8 |

GPIO\_CRH\_MODE10 | GPIO\_CRH\_CNF10);

// Плавающий вход

GPIOA->CRH |= GPIO\_CRH\_CNF8\_0 | GPIO\_CRH\_CNF10\_0;

// Настройка таймера TIM1

// Настройка таймера TIM1 для управления напряжением подаваемым на движок

// Не делить частоту тактирования цифрового фильтра

// Включить режим счёта с выравнивание по фронту

// Считать вверх

TIM1->CR1 &= ~TIM\_CR1\_CKD & ~TIM\_CR1\_CMS & ~TIM\_CR1\_DIR;

// Разрешить работу регистра предварительной загрузки

// Прерывание будет генерировать только переполнение счётчика

TIM1->CR1 |= TIM\_CR1\_ARPE | TIM\_CR1\_URS;

// Установим частоту переполнений таймера TIM1 равной 100 Гц

TIM1->ARR = (\_MotorTimerFreq / 100)-1;

// Снизить частоту счёта таймера с системной частоты до \_MotorTimerFreq

TIM1->PSC = (unsigned short)((24000000/\_MotorTimerFreq)-1);

// Событие обновления таймера генерировать без пропуска

TIM1->RCR = 0u;

// Принудительно обновить состояния регистров для задания частоты ШИМ

TIM1->EGR = TIM\_EGR\_UG;

// Настройка каналов таймера TIM1

// Отключить все каналы таймера TIM1 для дальнейшей настройки

TIM1->CCER &= ~(TIM\_CCER\_CC4E | TIM\_CCER\_CC3E |

TIM\_CCER\_CC2E | TIM\_CCER\_CC1E);

// Перевести в выходной режим канал 2 и сбросить режим работы канала 1

TIM1->CCMR1 &= ~(TIM\_CCMR1\_CC2S | TIM\_CCMR1\_CC1S);

// Перевести в выходной режим канал 4 и сбросить режим работы канала 3

TIM1->CCMR2 &= ~(TIM\_CCMR2\_CC4S | TIM\_CCMR2\_CC3S);

// Перевести выходы канала 2 в состоянии с неактивным уровнем

// (Force inactive level)

// Включить регистр предварительной загрузки канала 2

// Установить режим работы канала 1 на захват сигнала с вывода TI1

TIM1->CCMR1 |= TIM\_CCMR1\_OC2M\_2 | TIM\_CCMR1\_OC2PE | TIM\_CCMR1\_CC1S\_0;

// Перевести выход канала 4 в состоянии с неактивным уровнем

// (Force inactive level)

// Включить регистр предварительной загрузки канала 4

TIM1->CCMR2 |= TIM\_CCMR2\_OC4M\_2 | TIM\_CCMR2\_OC4PE;

// Установить частоту фильтрации входного сигнала TI1 в 0 –

// сигнал не фильтруется (IC1F = 0b0000)

// Установить значение коэффициента деления частоты сигнала с вывода TI1

// в 0 (IC1PSC = 0b00)

TIM1->CCMR1 &= ~(TIM\_CCMR1\_IC1F | TIM\_CCMR1\_IC1PSC);

// Установить частоту фильтрации входного сигнала TI3 в 0 –

// сигнал не фильтруется (IC3F = 0b0000)

// Установить значение коэффициента деления частоты сигнала с вывода TI3

// в 0 (IC3PSC = 0b00)

TIM1->CCMR2 &= ~(TIM\_CCMR2\_IC3F | TIM\_CCMR2\_IC3PSC);

// Установить величину заполнения ШИМ равной нулю

TIM1->CCR2 = 0u;

TIM1->CCR4 = 0u;

// Сбросить, соответственно, мощность двигателя

this->uchMotorPower = 0;

// Установить активный уровень каналов 4 и 2 - Low

TIM1->CCER |= TIM\_CCER\_CC4P | TIM\_CCER\_CC2P;

// Установить захват значения по каналу 1 по нарастающему фронту сигнала

TIM1->CCER &= ~ TIM\_CCER\_CC1P;

// Перевести в состояние High выходы каналов 4 и 2 управления двигателем

TIM1->CR2 |= (TIM\_CR2\_OIS4 | TIM\_CR2\_OIS2);

// Устанавливаем состояние мотора в свободная остановка

this->eState = eFreeStop;

// Отключаем комплементарные выводы всех каналов таймера TIM1

TIM1->CCER &= ~TIM\_CCER\_CC3NE & ~TIM\_CCER\_CC2NE & ~TIM\_CCER\_CC1NE;

// Включаем все каналы таймера TIM1

TIM1->CCER |= TIM\_CCER\_CC4E | TIM\_CCER\_CC3E | TIM\_CCER\_CC2E | TIM\_CCER\_CC1E;

// Разрешаем прерывания от канала 1

TIM1->DIER = TIM\_DIER\_UIE | TIM\_DIER\_CC1IE;

//Разрешаем главный выход таймера 1

TIM1->BDTR |= TIM\_BDTR\_MOE;

// Разрешить работу таймера 1

TIM1->CR1 |= TIM\_CR1\_CEN;

}

/\* Функция установки управляющего напряжения

\* Входные параметры: напряжение (0-100% от максимума = 10В)

\*/

void MotorAClass::MotorAMoveOn(int Power)

{

if(Power>0)

{

if(Power>100){

// Ограничиваем 100 процентами

Power = 100;

}

// Если меняется состояние движка, то...

if(this->eState != eRotateForward)

{

// Отключить каналы 4 и 2 таймера TIM1 для дальнейшей настройки

TIM1->CCER &= ~(TIM\_CCER\_CC4E | TIM\_CCER\_CC2E);

// Перевести канал 4 таймера в режим Frozen для выключения

TIM1->CCMR2 &= ~TIM\_CCMR2\_OC4M;

// Перевести канал 2 таймера в режим Frozen для выключения

TIM1->CCMR1 &= ~TIM\_CCMR1\_OC2M;

#ifndef HARD\_CONTROL\_A

// Канал 4 перевести в состоянии с активным уровнем

// при значении счётчика меньше чем регистр сравнения (PWM mode 1)

TIM1->CCMR2 |= TIM\_CCMR2\_OC4M\_2 | TIM\_CCMR2\_OC4M\_1;

// Канал 2 перевести в состояние

// с неактивным уровнем (Force inactive level)

TIM1->CCMR1 |= TIM\_CCMR1\_OC2M\_2;

#endif

#ifdef HARD\_CONTROL\_A

// Канал 4 перевести в состоянии с активным уровнем

// при значении счётчика меньше чем регистр сравнения (PWM mode 2)

TIM1->CCMR2 |= TIM\_CCMR2\_OC4M\_2 | TIM\_CCMR2\_OC4M\_1 | TIM\_CCMR2\_OC4M\_0;

// Канал 2 перевести в состояние

// с активным уровнем (Force active level)

TIM1->CCMR1 |= (TIM\_CCMR1\_OC2M\_2 | TIM\_CCMR1\_OC2M\_0);

#endif

// Установить новую мощность двигателя

TIM1->CCR4 = (TIM1->ARR \* Power) / 100;

// ...установить состояние движка как вращение вперёд

this->eState = eRotateForward;

// Включить каналы 4 и 2 таймера TIM1

TIM1->CCER |= TIM\_CCER\_CC4E | TIM\_CCER\_CC2E;

}

// Устанавливаем величину напряжения только если она изменилась

else

{

if(Power != this->uchMotorPower)

{

// Установить новую мощность двигателя

TIM1->CCR4 = (TIM1->ARR \* Power) / 100;

}

}

// Сохраняем величину мощности в классе

this->uchMotorPower = Power;

}

else

{

Power = -Power;

if(Power == 0) Power = 1;

if(Power >= 100) Power = 100;

// ... и если меняется состояние движка, то...

if(this->eState != eRotateBack)

{

// Отключить каналы 4 и 2 таймера TIM1 для дальнейшей настройки

TIM1->CCER &= ~(TIM\_CCER\_CC4E | TIM\_CCER\_CC2E);

// Перевести канал 4 таймера в режим Frozen для выключения

TIM1->CCMR2 &= ~TIM\_CCMR2\_OC4M;

// Перевести канал 2 таймера в режим Frozen для выключения

TIM1->CCMR1 &= ~TIM\_CCMR1\_OC2M;

#ifndef HARD\_CONTROL\_A

// Канал 4 перевести в состояние

// с неактивным уровнем (Force inactive level)

TIM1->CCMR2 |= TIM\_CCMR2\_OC4M\_2;

// Канал 2 перевести в состоянии с активным уровнем

// при значении счётчика меньше чем регистр сравнения (PWM mode 1)

TIM1->CCMR1 |= TIM\_CCMR1\_OC2M\_2 | TIM\_CCMR1\_OC2M\_1;

#endif

#ifdef HARD\_CONTROL\_A

// Канал 4 перевести в состояние

// с активным уровнем (Force active level)

TIM1->CCMR2 |= (TIM\_CCMR2\_OC4M\_2 | TIM\_CCMR2\_OC4M\_0);

// Канал 2 перевести в состоянии с активным уровнем

// при значении счётчика меньше чем регистр сравнения (PWM mode 2)

TIM1->CCMR1 |= TIM\_CCMR1\_OC2M\_2 | TIM\_CCMR1\_OC2M\_1 | TIM\_CCMR1\_OC2M\_0;

#endif

// Установить новую мощность двигателя

TIM1->CCR2 = (TIM1->ARR \* Power) / 100;

// ...установить состояние движка как вращение назад

this->eState = eRotateBack;

// Включить каналы 4 и 2 таймера TIM1

TIM1->CCER |= TIM\_CCER\_CC4E | TIM\_CCER\_CC2E;

}

// Устанавливаем величину мощности только если она изменилась

else

{

if(Power != this->uchMotorPower)

{

// Установить новую мощность двигателя

TIM1->CCR2 = (TIM1->ARR \* Power) / 100;

}

}

// Сохраняем величину мощности в классе

this->uchMotorPower = Power;

}

}

/\* Обработчик прерывания по переполненияю таймера TIM1

\*/

extern "C" void TIM1\_UP\_TIM16\_IRQHandler()

{

// Переменные оценки фильтра скорости

static int fi\_o;

static int om\_o;

static int ep\_o;

// Сбросить флаг переполнения таймера

TIM1->SR &= ~TIM\_SR\_UIF;

//T = 0.02, d=1

ep\_o = 2500\*(cMotorA.iMotorAngel\*1000) - 100\*om\_o - 2500\*fi\_o;

om\_o = om\_o + ep\_o/100;

fi\_o = fi\_o + om\_o/100;

// Градусы в секунду

cMotorA.uiMotorSpeedFilter = om\_o/1000;

}

/\* Обработчик прерывания по захвату фронтов таймером TIM1

\*/

extern "C" void TIM1\_CC\_IRQHandler()

{

// Если пришёл какой-либо фронт по тахометру A...

if(TIM1->SR & TIM\_SR\_CC1IF)

{

// При опережении сигнала A перед B (движение вперёд), тоесть...

// ...тогда когда при приходе нарастающего фронт по А

// сигнал тахометра B нуль или...

if( ( !(GPIOA->IDR & GPIO\_IDR\_IDR10) && !(TIM1->CCER & TIM\_CCER\_CC1P) )

||

// ...тогда когда при приходе спадающего фронта по А сигнал тахометра В

// единица, то...

( (GPIOA->IDR & GPIO\_IDR\_IDR10) && (TIM1->CCER & TIM\_CCER\_CC1P) ) )

{

// ...увеличиваем угол на градус

cMotorA.iMotorAngel++;

}

// При опережении сигнала B перед A (движение назад), тоесть...

// ...тогда когда при приходе нарастающего фронта по В сигнал

// тахометра А нуль или...

// ...тогда когда при приходе спадающего фронта по В

// сигнал тахометра А единица, то...

else

{

// ...уменьшаем угол на градус

cMotorA.iMotorAngel--;

}

// Меняем полярность захвата для канала 1 на противоположную

TIM1->CCER ^= TIM\_CCER\_CC1P;

// Сбрасываем флаг прерывания по захвату фронта по тахометру А

TIM1->SR &= ~TIM\_SR\_CC1IF;

}

}

//--------------- РЕАЛИЗАЦИЯ МЕТОДОВ ДВИГАТЕЛЯ B -----------------------

/\* Конструктор драйвера двигателя B

\*/

MotorBClass::MotorBClass()

{

// Движок остановлен - напряжение равно нулю

this->uchMotorPower = 0;

// Текущее состояние движка

this->eState = eFreeStop;

// Текущий угол поворота

this->iMotorAngel = 0;

// Текущая скорость вращения, вычисленная фильтром

this->uiMotorSpeedFilter = 0;

}

/\* Инициализация драйвера двигателя B

\*/

void MotorBClass::Init()

{

// Разрешение тактирования порта А

RCC->APB2ENR |= RCC\_APB2ENR\_IOPAEN;

// Разрешение тактирования таймера TIM2

RCC->APB1ENR |= RCC\_APB1ENR\_TIM2EN;

// Настройка NVIC под таймер TIM2

// Установка наивысшего (нулевого) приоритета для глобального прерывания

// по таймеру TIM2

NVIC->IP[TIM2\_IRQn] = 0ul << 4;

// Разрешение прерывания

NVIC->ISER[0] |= 1ul << TIM2\_IRQn;

// Настройка портов ввода вывода под вывод TIM2

// Настройка выводов 1 и 3 порта А под выходы управления движком

// Очистка битов настройки

GPIOA->CRL &= ~(GPIO\_CRL\_MODE1 | GPIO\_CRL\_CNF1 |

GPIO\_CRL\_MODE3 | GPIO\_CRL\_CNF3);

// Пушпульный выход для периферии

GPIOA->CRL |= GPIO\_CRL\_MODE1 | GPIO\_CRL\_CNF1\_1 |

GPIO\_CRL\_MODE3 | GPIO\_CRL\_CNF3\_1;

// Настройка выводов 0 и 2 порта А под входы сигналов тахометра движка

// Очистка битов настройки

GPIOA->CRL &= ~(GPIO\_CRL\_MODE0 | GPIO\_CRL\_CNF0 |

GPIO\_CRL\_MODE2 | GPIO\_CRL\_CNF2);

// Плавающий вход

GPIOA->CRL |= GPIO\_CRL\_CNF0\_0 | GPIO\_CRL\_CNF2\_0;

// Настройка таймера TIM2

// Настройка таймера TIM2 для управления напряжением подаваемым на движок

// Не делить частоту тактирования цифрового фильтра

// Включить режим счёта с выравнивание по фронту

// Считать вверх

TIM2->CR1 &= ~TIM\_CR1\_CKD & ~TIM\_CR1\_CMS & ~TIM\_CR1\_DIR;

// Разрешить работу регистра предварительной загрузки

// Прерывание будет генерировать только переполнение счётчика

TIM2->CR1 |= TIM\_CR1\_ARPE | TIM\_CR1\_URS;

// Установим частоту переполнений таймера TIM2 равной 100 Гц

TIM2->ARR = (\_MotorTimerFreq / 100)-1;//100

// Снизить частоту счёта таймера с системной частоты до \_MotorATimerFreq

TIM2->PSC = (unsigned short)((24000000/\_MotorTimerFreq)-1);

// Принудительно обновить состояния регистров для задания частоты ШИМ

TIM2->EGR = TIM\_EGR\_UG;

// Настройка каналов таймера TIM2

// Отключить все каналы таймера TIM2 для дальнейшей настройки

TIM2->CCER &= ~(TIM\_CCER\_CC4E | TIM\_CCER\_CC3E |

TIM\_CCER\_CC2E | TIM\_CCER\_CC1E);

// Перевести в выходной режим канал 2 и сбросить режим работы канала 1

TIM2->CCMR1 &= ~(TIM\_CCMR1\_CC2S | TIM\_CCMR1\_CC1S);

// Перевести в выходной режим канал 4 и сбросить режим работы канала 3

TIM2->CCMR2 &= ~(TIM\_CCMR2\_CC4S | TIM\_CCMR2\_CC3S);

// Перевести выходы канала 2 в состоянии с неактивным уровнем

// (Force inactive level)

// Включить регистр предварительной загрузки канала 2

// Установить режим работы канала 1 на захват сигнала с вывода TI1

TIM2->CCMR1 |= TIM\_CCMR1\_OC2M\_2 | TIM\_CCMR1\_OC2PE | TIM\_CCMR1\_CC1S\_0;

// Перевести выход канала 4 в состоянии с неактивным уровнем

// (Force inactive level)

// Включить регистр предварительной загрузки канала 4

TIM2->CCMR2 |= TIM\_CCMR2\_OC4M\_2 |TIM\_CCMR2\_OC4PE;

// Установить частоту фильтрации входного сигнала TI1 в 0 –

// сигнал не фильтруется (IC1F = 0b0000)

// Установить значение коэффициента деления частоты сигнала с вывода TI1

// в 0 (IC1PSC = 0b00)

TIM2->CCMR1 &= ~(TIM\_CCMR1\_IC1F | TIM\_CCMR1\_IC1PSC);

// Установить частоту фильтрации входного сигнала TI3 в 0 –

// сигнал не фильтруется (IC3F = 0b0000)

// Установить величину заполнения ШИМ равной нулю

TIM2->CCR2 = 0u;

TIM2->CCR4 = 0u;

// Сбросить, соответственно, мощность двигателя

this->uchMotorPower = 0;

// Установить активный уровень каналов 4 и 2 - Low

TIM2->CCER |= TIM\_CCER\_CC4P | TIM\_CCER\_CC2P;

// Установить захват значения по каналу 1 по нарастающему фронту сигнала

TIM2->CCER &= ~(TIM\_CCER\_CC1P);

// Перевести в состояние High выходы каналов 4 и 2 управления двигателем

TIM2->CR2 |= (TIM\_CR2\_OIS4 | TIM\_CR2\_OIS2);

// Устанавливаем состояние мотора в свободная остановка

this->eState = eFreeStop;

// Отключаем комплементарные выводы всех каналов таймера TIM2

TIM2->CCER &= ~TIM\_CCER\_CC3NE & ~TIM\_CCER\_CC2NE & ~TIM\_CCER\_CC1NE;

// Включаем все каналы таймера TIM2

TIM2->CCER |= TIM\_CCER\_CC4E | TIM\_CCER\_CC2E | TIM\_CCER\_CC1E;

// Разрешаем прерывания от канала 1

TIM2->DIER = TIM\_DIER\_UIE | TIM\_DIER\_CC1IE;

//TIM2->DIER = TIM\_DIER\_UIE;

// Разрешить работу таймера 1

TIM2->CR1 |= TIM\_CR1\_CEN;

}

/\* Функция установки управляющего напряжения

\* Входные параметры: напряжение (0-100% от максимума = 10В)

\*/

void MotorBClass::MotorBMoveOn(int Power)

{

if(Power>0)

{

if(Power>100) Power = 100;

if(this->eState != eRotateForward)

{

// Отключить каналы 4 и 2 таймера TIM2 для дальнейшей настройки

TIM2->CCER &= ~(TIM\_CCER\_CC4E | TIM\_CCER\_CC2E);

// Перевести канал 4 таймера в режим Frozen для выключения

TIM2->CCMR2 &= ~TIM\_CCMR2\_OC4M;

// Перевести канал 2 таймера в режим Frozen для выключения

TIM2->CCMR1 &= ~TIM\_CCMR1\_OC2M;

#ifndef HARD\_CONTROL\_B

// Канал 4 перевести в состоянии с активным уровнем

// при значении счётчика меньше чем регистр сравнения (PWM mode 1)

TIM2->CCMR2 |= TIM\_CCMR2\_OC4M\_2 | TIM\_CCMR2\_OC4M\_1;

// Канал 2 перевести в состояние

// с неактивным уровнем (Force inactive level)

TIM2->CCMR1 |= TIM\_CCMR1\_OC2M\_2;

#endif

#ifdef HARD\_CONTROL\_B

// Канал 4 перевести в состоянии с активным уровнем

// при значении счётчика меньше чем регистр сравнения (PWM mode 2)

TIM2->CCMR2 |= TIM\_CCMR2\_OC4M\_2 | TIM\_CCMR2\_OC4M\_1 | TIM\_CCMR2\_OC4M\_0;

// Канал 2 перевести в состояние

// с активным уровнем (Force active level)

TIM2->CCMR1 |= (TIM\_CCMR1\_OC2M\_2 | TIM\_CCMR1\_OC2M\_0);

#endif

// Установить новую мощность двигателя

TIM2->CCR4 = (TIM2->ARR \* Power) / 100;

// ...установить состояние движка как вращение вперёд

this->eState = eRotateForward;

// Включить каналы 4 и 2 таймера TIM2

TIM2->CCER |= TIM\_CCER\_CC4E | TIM\_CCER\_CC2E;

}

// Устанавливаем величину мощности только если она изменилась

else if(Power != this->uchMotorPower)

{

// Установить новую мощность двигателя

TIM2->CCR4 = (TIM2->ARR \* Power) / 100;

}

// Сохраняем величину мощности в классе

this->uchMotorPower = Power;

}

else

{

Power = -Power;

if(Power == 0) Power = 1;

if(Power >= 100) Power = 100;

// ... и если меняется состояние движка, то...

if(this->eState != eRotateBack)

{

// Отключить каналы 4 и 2 таймера TIM2 для дальнейшей настройки

TIM2->CCER &= ~(TIM\_CCER\_CC4E | TIM\_CCER\_CC2E);

// Перевести канал 4 таймера в режим Frozen для выключения

TIM2->CCMR2 &= ~TIM\_CCMR2\_OC4M;

// Перевести канал 2 таймера в режим Frozen для выключения

TIM2->CCMR1 &= ~TIM\_CCMR1\_OC2M;

#ifndef HARD\_CONTROL\_B

// Канал 4 перевести в состояние

// с неактивным уровнем (Force inactive level)

TIM2->CCMR2 |= TIM\_CCMR2\_OC4M\_2;

// Канал 2 перевести в состоянии с активным уровнем

// при значении счётчика меньше чем регистр сравнения (PWM mode 1)

TIM2->CCMR1 |= TIM\_CCMR1\_OC2M\_2 | TIM\_CCMR1\_OC2M\_1;

#endif

#ifdef HARD\_CONTROL\_B

// Канал 4 перевести в состояние

// с активным уровнем (Force active level)

TIM2->CCMR2 |= (TIM\_CCMR2\_OC4M\_2 | TIM\_CCMR2\_OC4M\_0);

// Канал 2 перевести в состоянии с активным уровнем

// при значении счётчика меньше чем регистр сравнения (PWM mode 2)

TIM2->CCMR1 |= TIM\_CCMR1\_OC2M\_2 | TIM\_CCMR1\_OC2M\_1 | TIM\_CCMR1\_OC2M\_0;

#endif

// Установить новую мощность двигателя

TIM2->CCR2 = (TIM2->ARR \* Power) / 100;

// ...установить состояние движка как вращение назад

this->eState = eRotateBack;

// Включить каналы 4 и 2 таймера TIM2

TIM2->CCER |= TIM\_CCER\_CC4E | TIM\_CCER\_CC2E;

}

// Устанавливаем величину мощности только если она изменилась

else if(Power != this->uchMotorPower)

{

// Установить новую мощность двигателя

TIM2->CCR2 = (TIM2->ARR \* Power) / 100;

}

// Сохраняем величину мощности в классе

this->uchMotorPower = Power;

}

}

/\* Обработчик прерывания по захвату фронтов от тахометров двигателя и переполнению таймера

\*/

extern "C" void TIM2\_IRQHandler()

{

static int fi\_o;

static int om\_o;

static int ep\_o;

// В случае, если произошло прерывание по переполнению счётчика таймера....

if(TIM2->SR & TIM\_SR\_UIF)

{

// Сбросить флаг переполнения таймера

TIM2->SR &= ~TIM\_SR\_UIF;

// Вычислить скорость вращения колес

// T = 0.02, d=1

ep\_o = 2500\*(cMotorB.iMotorAngel\*1000) - 100\*om\_o - 2500\*fi\_o;

om\_o = om\_o + ep\_o/100;

fi\_o = fi\_o + om\_o/100;

// Градусы в секунду

cMotorB.uiMotorSpeedFilter = om\_o/1000;

}

// Если произошёл захват фронтов от тахометров двигателя...

else

{

// Если пришёл фронт по тахометру A...

if(TIM2->SR & TIM\_SR\_CC1IF)

{

// При опережении сигнала A перед B (движение вперёд), то есть...

// ...тогда когда при приходе нарастающего фронт по А

// сигнал тахометра B нуль или...

if( ( !(GPIOA->IDR & GPIO\_IDR\_IDR2) && !(TIM2->CCER & TIM\_CCER\_CC1P) )

||

// ...тогда когда при приходе спадающего фронта по А

// сигнал тахометра В единица, то...

( (GPIOA->IDR & GPIO\_IDR\_IDR2) && (TIM2->CCER & TIM\_CCER\_CC1P) ) )

{

// ...увеличиваем угол на градус

cMotorB.iMotorAngel--;

}

// При опережении сигнала B перед A (движение назад), то есть...

// ...тогда когда при приходе нарастающего фронта по В

// сигнал тахометра А нуль или...

// ...тогда когда при приходе спадающего фронта по В

// сигнал тахометра А единица, то...

else

{

// ...уменьшаем угол на градус

cMotorB.iMotorAngel++;

}

// Меняем полярность захвата для канала 1 на противоположную

TIM2->CCER ^= TIM\_CCER\_CC1P;

// Сбрасываем флаг прерывания по захвату фронта по тахометру А

TIM2->SR &= ~TIM\_SR\_CC1IF;

}

}

}

Файл MotorClass.hpp

/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Заголовочный файл драйвера двигателей NXT

Устройство: Отладочная плата STM32VLDiscover

Заказчик: НГТУ

Микроконтроллер: STM32F100RBT6B

Язык программированиЯ: C++

Среда разработки: Eclipse Helios SR2

Набор средств компиляции: Sourcery CodeBench Lite for ARM EABI

Авторы: Федоров Дмитрий Сергеевич, Ивойлов Андрей Юрьевич,

Жмудь Вадим Аркадьевич, Трубин Виталий Геннадьевич

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/

#ifndef MOTORCLASS\_HPP\_

#define MOTORCLASS\_HPP\_

// Частота тактирования таймеров (всех описанных в данном модуле)

#define \_MotorTimerFreq 100000

// Перечисление режимов работы двигателей

enum MotorStateEnumeration

{

eRotateForward, // Свободное вращение вперёд

eRotateBack, // Свободное вращение назад

eFreeStop, // Остановка двигателя с выбегом

};

//----------------------------- КЛАСС ДВИГАТЕЛЯ А ----------------------------

class MotorAClass

{

// Закрытые поля класса

private:

// Управляющее напряжение

unsigned int uchMotorPower;

// Открытые поля класса

public:

// Текущее состояние двигателя

MotorStateEnumeration eState;

// Текущий угол поворота

int iMotorAngel;

// Текущее значение угловой скорости, получаемое по фильтру

int uiMotorSpeedFilter;

public:

// Коструктор класса

MotorAClass();

// Инициализация

void Init();

// Функция задания управляющего напряжения

void MotorAMoveOn(int Power);

};

//----------------------------- КЛАСС ДВИГАТЕЛЯ B ----------------------------

class MotorBClass

{

// Закрытые поля класса

private:

// Управляющее напряжение

unsigned char uchMotorPower;

// Открытые поля класса

public:

// Текущее состояние двигателя

MotorStateEnumeration eState;

// Текущий угол поворота

int iMotorAngel;

// Текущее значение угловой скорости, получаемое по фильтру

int uiMotorSpeedFilter;

public:

// Коструктор класса

MotorBClass();

// Инициализация

void Init();

// Функция задания управляющего напряжения

void MotorBMoveOn(int Power);

};

// Объявление экземпляров классов драйверов в файле реализации

extern MotorAClass cMotorA;

extern MotorBClass cMotorB;

#endif /\* MOTORCLASS\_HPP\_ \*/

Файл Radio.cpp

/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Исполняемый файл драйвера приемника радиоканала

Устройство: Отладочная плата STM32VLDiscovery

Заказчик: НГТУ

Микроконтроллер: STM32F100RBT6B

Язык программированиЯ: C++

Среда разработки: Eclipse Helios SR2

Набор средств компиляции: Sourcery CodeBench Lite for ARM EABI

Авторы: Федоров Дмитрий Сергеевич, Ивойлов Андрей Юрьевич,

Жмудь Вадим Аркадьевич, Трубин Виталий Геннадьевич

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/

#include "Radio.hpp"

Radio cRadio;

/\* Функция инициализации

\*/

void Radio::Init()

{

// Разрешение тактирования таймера TIM3, порта GPIOA, GPIOB

RCC->APB1ENR |=RCC\_APB1ENR\_TIM3EN;

RCC->APB2ENR |=RCC\_APB2ENR\_IOPAEN;

RCC->APB2ENR |=RCC\_APB2ENR\_IOPBEN;

// Настройка ног микроконтроллера для работы с таймером

// PA7 - вход CH2

GPIOA->CRL &=~GPIO\_CRL\_MODE7;

GPIOA->CRL |=GPIO\_CRL\_CNF7\_0;

GPIOA->CRL &=~GPIO\_CRL\_CNF7\_1;

// PB1 - вход CH4

GPIOB->CRL &=~GPIO\_CRL\_MODE1;

GPIOB->CRL |=GPIO\_CRL\_CNF1\_0;

GPIOB->CRL &=~GPIO\_CRL\_CNF1\_1;

// Настройка таймера

TIM3->PSC = 0x0018; //Предделитель 24

TIM3->ARR = 0xFFFF; //Период > 22000

// Сигнал TI2 канала CH\_2 устанавливается активным входом

// для TIM3\_CCR1 через TI2FP1 (TI2->TI2FP1->IC1)

TIM3->CCMR1 |=TIM\_CCMR1\_CC1S\_1;

TIM3->CCMR1 &=~TIM\_CCMR1\_CC1S\_0;

// Сигнал TI4 канала CH\_4 устанавливается активным входом

// для TIM3\_CCR3 через TI4FP3 (TI4->TI4FP3->IC3)

TIM3->CCMR2 |=TIM\_CCMR2\_CC3S\_1;

TIM3->CCMR2 &=~TIM\_CCMR2\_CC3S\_0;

// Сигнал TI2 канала CH\_2 устанавливается активным входом для

// TIM3\_CCR2 через TI2FP2 (TI2->TI2FP2->IC2)

TIM3->CCMR1 |=TIM\_CCMR1\_CC2S\_0;

TIM3->CCMR1 &=~TIM\_CCMR1\_CC2S\_1;

// Сигнал TI4 канала CH\_4 устанавливается активным входом

// для TIM3\_CCR4 через TI4FP4 (TI4->TI4FP4->IC4)

TIM3->CCMR2 |=TIM\_CCMR2\_CC4S\_0;

TIM3->CCMR2 &=~TIM\_CCMR2\_CC4S\_1;

// Сигнал TI2FP1 генерируется по отрицательному фронту TI2 и используется

// для захвата значения в счетчике CNT в регистр CCR1

TIM3->CCER |=TIM\_CCER\_CC1P;

// Сигнал TI4FP3 генерируется по отрицательному фронту TI4 и используется

// для захвата значения в счетчике CNT в регистр CCR3

TIM3->CCER |=TIM\_CCER\_CC3P;

// Сигнал TI2FP2 генерируется по положительному фронту TI2

TIM3->CCER &=~TIM\_CCER\_CC2P;

// Сигнал TI4FP4 генерируется по положительному фронту TI4

TIM3->CCER &=~TIM\_CCER\_CC4P;

// Сигнал TI2FP2 определяется входным для Trigger controller

TIM3->SMCR |=(TIM\_SMCR\_TS\_1|TIM\_SMCR\_TS\_2);

TIM3->SMCR &=~TIM\_SMCR\_TS\_0;

// Положительный фронт выбранного сигнала (в нашем случае TI2FP2)

// будет использоваться для сброса счетчика CNT

TIM3->SMCR |=TIM\_SMCR\_SMS\_2;

TIM3->SMCR &=~TIM\_SMCR\_SMS\_0;

TIM3->SMCR &=~TIM\_SMCR\_SMS\_1;

// Разрешаются захваты сигналов

TIM3->CCER |=TIM\_CCER\_CC1E;

TIM3->CCER |=TIM\_CCER\_CC2E;

TIM3->CCER |=TIM\_CCER\_CC3E;

TIM3->CCER |=TIM\_CCER\_CC4E;

// Команда запускает счетчик

TIM3->CR1 |=TIM\_CR1\_CEN;

}

/\* Чтение значения канала "Вперед-назад"

\*/

int32\_t Radio::RemoteControl\_F\_B()

{

return (\_middle\_f\_b - (int32\_t)(TIM3->CCR1));

}

/\* Чтение значения канала "Вправо-влево"

\*/

int32\_t Radio::RemoteControl\_L\_R()

{

return (\_middle\_l\_r - ((int32\_t)(TIM3->CCR3)-(int32\_t)(TIM3->CCR4)));

}

/\* Калиброка каналов

\*/

void Radio::Calib()

{

this->\_middle\_f\_b = (int32\_t)(TIM3->CCR1);

this->\_middle\_l\_r = ((int32\_t)(TIM3->CCR3)-(int32\_t)(TIM3->CCR4));

}

Файл Radio.hpp

/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Заголовочный файл драйвера приемника радиоканала

Устройство: Отладочная плата STM32VLDiscovery

Заказчик: НГТУ

Микроконтроллер: STM32F100RBT6B

Язык программированиЯ: C++

Среда разработки: Eclipse Helios SR2

Набор средств компиляции: Sourcery CodeBench Lite for ARM EABI

Авторы: Федоров Дмитрий Сергеевич, Ивойлов Андрей Юрьевич,

Жмудь Вадим Аркадьевич, Трубин Виталий Геннадьевич

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/

#ifndef RADIO\_HPP\_

#define RADIO\_HPP\_

#include <stm32f10x.h>

// Описание класса

class Radio

{

private:

// Значения, принятые по радиоканалу

int32\_t \_middle\_f\_b;

int32\_t \_middle\_l\_r;

public:

// Инициализация

void Init();

// Считывание данных по каналу "Вперед-назад"

int32\_t RemoteControl\_F\_B();

// Считывание данных по каналу "Влево-вправо"

int32\_t RemoteControl\_L\_R();

// Калибровка каналов

void Calib();

};

extern Radio cRadio;

#endif /\* RADIO\_HPP\_ \*/

Файл BluetoothClass.cpp

/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Исполняемый файл Bluetooth модуля

Устройство: Отладочная плата STM32VLDiscovery

Заказчик: НГТУ

Микроконтроллер: STM32F100RBT6B

Язык программированиЯ: C++

Среда разработки: Eclipse Helios SR2

Набор средств компиляции: Sourcery CodeBench Lite for ARM EABI

Авторы: Федоров Дмитрий Сергеевич, Ивойлов Андрей Юрьевич,

Жмудь Вадим Аркадьевич, Трубин Виталий Геннадьевич

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/

#include <stm32f10x.h>

#include "BluetoothClass.hpp"

// Создание экземпляра объекта класса BluetoothClass

BluetoothClass cBluetooth;

/\* Конструктор класса

\*/

BluetoothClass::BluetoothClass()

{

// Инициализация поля modeflag

this->\_recieved\_data=0;

}

/\* Функция инициализации модуля

\* Передает строку символов через USART1 и Bluetooth-модуль на ПК

\* Входные параметры: Нет

\*/

void BluetoothClass::Init()

{

// Включение тактирования порта А, B, блока переназначения и USART1

RCC->APB2ENR |= RCC\_APB2ENR\_IOPAEN | RCC\_APB2ENR\_IOPBEN |

RCC\_APB2ENR\_AFIOEN;

RCC->APB2ENR |= RCC\_APB2ENR\_USART1EN;

// Вывод 6 порта B переводится в режим альтернативной функции и

// используется как выход USART1 - TXD (передатчик)

GPIOB->CRL |= (GPIO\_CRL\_CNF6\_1 | GPIO\_CRL\_MODE6);

GPIOB->CRL &= ~GPIO\_CRL\_CNF6\_0;

// Вывод 7 порта B переводится в режим floating,

// он будет использоваться как вход USART1 - RXD (приемник)

GPIOB->CRL &=~GPIO\_CRL\_MODE7;

GPIOB->CRL |=GPIO\_CRL\_CNF7\_1;

GPIOB->CRL &=~GPIO\_CRL\_CNF7\_0;

// Включить переназначение выводов USART1

AFIO->MAPR |=AFIO\_MAPR\_USART1\_REMAP;

// USART1 настраивается на работу с 1 стартовым и

// 8-ю информационными битами

USART1->CR1 &=~USART\_CR1\_M;

// USART1 настраивается на работу с 1 стоповым битом

USART1->CR2 &=~USART\_CR2\_STOP;

// Скорость обмена данными устанавливается равной 115200 бод

USART1->BRR = 0xd0;

// Этими командами включаются передатчик и приемник USART1

USART1->CR1|=USART\_CR1\_TE;

USART1->CR1|=USART\_CR1\_RE;

// Разрешается генерация прерывания USART1 при приеме данных

USART1->CR1 |= USART\_CR1\_RXNEIE;

// Разрешается обработка прерываний от USART1

NVIC->ISER[1] |= NVIC\_ISER\_SETENA\_5;

// С помощью данной команды устанавливается приоритет(наивысший)

// прерываний от USART1

NVIC->IP[37] = 0;

// Эта команда включает USART1

USART1->CR1 |=USART\_CR1\_UE;

}

/\* Функция передачи строки символов

\* Педелает строку символов через USART1 и Bluetooth-модуль на ПК

\* Входные параметры: указатель на последовательность символов, которые

\* необходимо передать. Последовательность должна заканчиваться символом конца строки.

\*/

void BluetoothClass::Transmit\_Str(char \*str)

{

unsigned char i = 0;

//Перебор до '\0' - символа конца строки

while(str[i])

{

//Ожидание завершения предыдущей передачи

while(!(USART1->SR & USART\_SR\_TC));

USART1->DR = str[i];

i++;

}

}

/\* Функция преобразования числа в строку

\* Преобразует целое число в строку символов, длина которой

\* равна 12 (1-й символ "-" - может отсутствовать, 10 цифр 32-разрядного числа

\* и 12-й - символ конца строки)

\* Входные параметры: целое знаковое число с максимальной разрядностью –

\* 32 бита

\*/

char\* BluetoothClass::IntToStr(int iNumber)

{

static char data\_to\_transmit[12];

// Делитель (на число кратное 10) преобразуемого числа

unsigned int uiDecimalDivisor = 1;

unsigned char i = 0;

unsigned char j = 0;

if (iNumber < 0)

{

data\_to\_transmit[i]= '-';

i++;

iNumber = -iNumber;

}

int iNumberCopy = iNumber;

for (; iNumberCopy != 0; j++)

{

iNumberCopy /= 10;

}

for (j--; j > 0; j--)

{

uiDecimalDivisor \*= 10;

}

// Разбиваем число на ASCII символы...

do

{

// Делим преобразуемое число на делитель - в частном крайняя левая цифра

// Код ASCII числа смещен относительно значения этого числа на 0x30

// (см. таблицу ASCII)

// ASCII код заносим по указателю и смещаем этот указатель

// на следующую позицию

data\_to\_transmit[i] = char(0x0030u + (iNumber / uiDecimalDivisor));

i++;

// Для дальнейшего преобразования получаем остаток от

// деления на делитель

iNumber %= uiDecimalDivisor;

// Уменьшаем в 10 раз величину делителя

uiDecimalDivisor /= 10;

}

// Деление будет продолжаться пока делитель не нулевой

while(uiDecimalDivisor != 0);

// В конце строки символов ставим ограничительный нуль

data\_to\_transmit[i] = '\0';

return data\_to\_transmit;

}

/\* Обработчик прерывания USART1 по приему данных

\* Код принятого символа будет сохранен в поле класса

\*/

extern "C" void USART1\_IRQHandler()

{

cBluetooth.\_recieved\_data = USART1->DR;

}

Файл BluetoothClass.hpp

/\*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Заголовочный файл модуля Bluetooth

Устройство: Отладочная плата STM32VLDiscover

Заказчик: НГТУ

Микроконтроллер: STM32F100RBT6B

Язык программированиЯ: C++

Среда разработки: Eclipse Helios SR2

Набор средств компиляции: Sourcery CodeBench Lite for ARM EABI

Авторы: Федоров Дмитрий Сергеевич, Ивойлов Андрей Юрьевич,

Жмудь Вадим Аркадьевич, Трубин Виталий Геннадьевич

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\*/

//Описание класса

class BluetoothClass

{

public:

// Поле для хранения кода символа, переданного пользователем

unsigned char \_recieved\_data;

// Конструктор класса

BluetoothClass();

//Функция инициализации

void Init();

// Функция передачи строки символов

void Transmit\_Str(char \*str);

// Функция преобразования числа в строку

char\* IntToStr(int iNumber);

};

extern BluetoothClass cBluetooth;