**Министерство науки и высшего образования Российской Федерации**

федеральное государственное автономное образовательное учреждение   
высшего образования

«Национальный исследовательский Томский политехнический университет»

|  |  |
| --- | --- |
| Школа / филиал | Инженерная школа информационных технологий и робототехники (ИШИТР) |
| Обеспечивающее подразделение | Отделение автоматизации и робототехники (ОАР) |
| Направление подготовки / специальность | 15.03.06 Мехатроника и робототехника |
| Образовательная программа (направленность (профиль)) | Интеллектуальные робототехнические  и мехатронные системы |
| Специализация | Системы управления автономными роботами |

**ДНЕВНИК ОБУЧАЮЩЕГОСЯ ПО ПРАКТИКЕ**

|  |  |
| --- | --- |
| Вид практики | Производственная практика |
| Тип практики | Преддипломная практика |

|  |  |
| --- | --- |
| Обучающийся | Сокуров Руслан Ергалиевич |
| Группа | 8Е02 |

|  |  |  |  |
| --- | --- | --- | --- |
| Учебный год | 2023/2024 | | |
| Сроки практики | с «\_\_\_»\_\_\_\_\_\_\_\_\_\_ 20\_\_ г. по «\_\_\_» \_\_\_\_\_\_\_\_\_ 20\_\_ г. | | |
| Место практики | г. Томск, ТПУ, Отделение автоматизации и робототехники (ОАР) | | |
| Ответственное лицо от принимающего подразделения ТПУ | Скороспешкин Максим Владимирович | | |
| Контактный телефон, е-mail | | smax@tpu.ru |
| Руководитель практики от ТПУ | Ланграф Сергей Владимирович | | |
| Контактный телефон, е-mail | langraf@tpu.ru | |

С программой практики ознакомлен: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись обучающегося) (дата)

Томск 2024

УТВЕРЖДАЮ

Руководитель ООП

\_\_\_\_\_\_\_А.С. Беляев

«\_\_\_»\_\_\_\_\_\_\_\_20\_\_\_ г.

**ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ НА ПРАКТИКУ**

**1. Тема научно-исследовательской работы:**

|  |
| --- |
| Практическая реализация разработанных ранее контуров управления током, скоростью и положением на основе блока управления рулевой рейкой БУРР-30-С. |

**2. Перечень работ (заданий), подлежащих выполнению:**

|  |
| --- |
| Практическая реализация контура управления током электропривода рулевой рейки |
| Практическая реализация контура управления скоростью электропривода рулевой рейки |
| Практическая реализация контура управления положением электропривода рулевой рейки |
| Экспериментальное исследование и анализ полученных результатов |
|  |
|  |
|  |
|  |

**3. Перечень отчетных материалов и требования к их оформлению:**

|  |
| --- |
| Отчёт по практике |
| Дневник по практике |
|  |
|  |
|  |
|  |
|  |
|  |

Руководитель практики

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(должность) (подпись) (Ф. И. О.)

Задание принял к исполнению

(подпись) (Ф. И. О. обучающегося)

«\_\_\_» \_\_\_\_\_\_\_\_\_ 20\_\_\_г.

# Введение

Беспилотные автомобили являются одной из самых актуальных и перспективных тем в автомобильной индустрии. Они представляют собой транспортные средства, которые способны перемещаться без участия водителя. Согласно распоряжению Правительства РФ к 2035 году ожидается увеличении доли беспилотных автомобилей в общей структуре мировых продаж автотранспорта возрастёт до 10–15%. [1]

Актуальность беспилотных автомобилей объясняется несколькими факторами. Во-первых, они могут значительно повысить безопасность на дорогах. Около 90% аварий на дорогах вызваны ошибками водителей [2], и беспилотные транспортные средства, оснащенные передовыми системами безопасности и алгоритмами управления, могут снизить вероятность возникновения аварийных ситуаций.

Основные проблемы внедрения технологий автономности включают в себя отсутствие в настоящее время в Российской Федерации ряда критичных электронных компонентов 2-го и 3-го уровней автономности [3].

Поскольку рулевая рейка является одним из ключевых компонентов систем 2-го и 3-го уровней (например, система удержания в полосе) разработка системы управления рулевой рейкой является актуальной задачей.

Целью данной работы является практическая имплементация разработанной в прошлых работах системы управления рулевой рейкой.

# Практическая имплементация контура тока

Для реализации контура управления тока требуется поддержать снятие показаний с датчика тока, поскольку эта информация используется в системе управления в виде обратной связи.



Рисунок 2.1 — Силовая часть блока управления рулевой рейки

В данном контексте датчик тока (ДТ) представляет собой токоизмерительный шунт, напряжение с которого передается на аналогово-цифровой преобразователь (АЦП) и впоследствии обрабатывается микроконтроллером. Использование двух датчиков тока в ветвях обусловлено необходимостью постоянного мониторинга тока в системе. В противном случае, при работе одной из ветвей, измерение тока было бы невозможно.

Также важно учитывать, что измерения с АЦП необходимо проводить в моменты, когда уровень тока стабилен. Это означает, что измерения не должны проводиться в моменты переключения транзисторов, так как в противном случае сигнал будет сильно искажен.

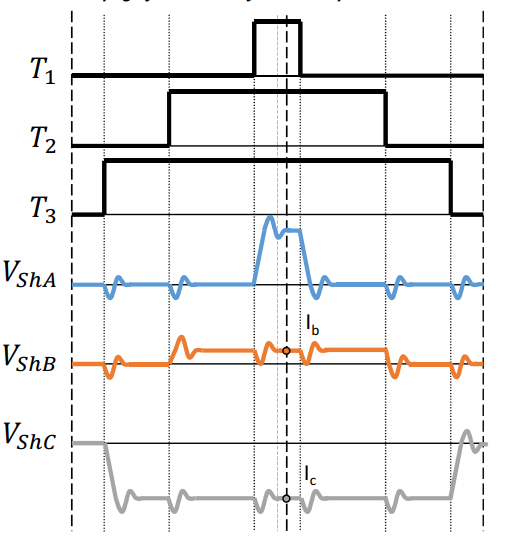


Рисунок №2.2 — Демонстрация искажений тока при коммутациях ШИМ

Каналы АЦП бывают двух типов: регулярные (regular) и инжектированные (injected). Название "инжектированный" означает, что запуск преобразования этого канала может быть "вставлено" между преобразованиями регулярных каналов, т. е. обработка регулярных каналов при этом приостанавливается. Производить аналого-цифровое преобразование будем именно в инжектированном режиме. Для этого будем использовать уже готовый метод из ПО БУРР-30.

Поскольку возможны различные отклонения датчика тока от истинных значений, предусмотрен механизм калибровки показаний датчика. Таким образом, когда известно, что ток отсутствует (ШИМ не активна), производится замер, и, если датчик показывает наличие тока, происходит калибровка нулевого значения. Затем выполняется раскрытие ШИМ на определенный процент. Зная этот процент, можно точно оценить уровень тока в силовых элементах, и таким образом подобрать коэффициент усиления, чтобы этот ток совпадал с подаваемым значением.

Программная реализация контура управления током, разработанного в предыдущих разделах, представлена в листинге кода (Приложение А). Блок-схема алгоритма работы программы представлена на рисунке 2.3.

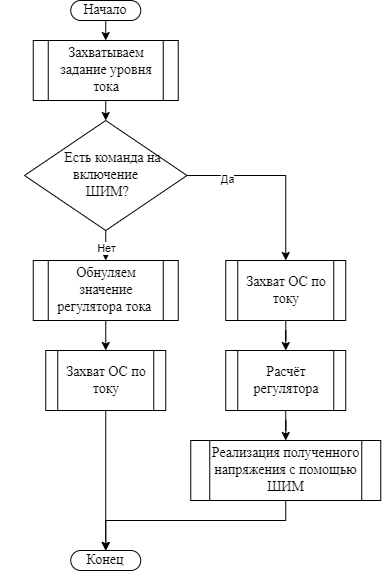


Рисунок 2.3 — Блок-схема контура тока(?)

Для проверки работы контура зафиксируем шток рулевой рейки, предотвратив вращение вала электропривода рулевой рейки (отсутствует противоЭДС), и сравним полученные характеристики с модельными данными.

Переходный процесс реальной рулевой рейки представлен на рисунке 2.4.



Рисунок 2.4 — Переходная характеристика тока

Его показатели качества: перерегулирование 3,37%, время переходного процесса 1,5 мс. Для модели показатели следующие:



Рисунок 2.5 — Переходная характеристика тока в имитационной модели

Перерегулирование: 1.043%, время переходного процесса 1.076 мс.

*Графики 2.4 и 2.5 однозначно имеют общие черты и визуально схожи, поскольку при разработке имитационной модели контура тока стояла задача имитации электромагнитной составляющей именно этого электропривода.*

*Тем не менее, видны и отличия. Например, отличается как время регулирования переходного процесса, так и значение перерегулирования. Также переходный процесс на рисунке 2.4 имеет более «рваный» характер.*

*Несоответствие критериев качества переходного процесса объясняется рядом допущений, выполненных в ходе разработки имитационной модели контура тока, например: предполагалось, что в цепи обратной связи по току используется безынерционный датчик с коэффициентом передачи равным 1, но в действительности небольшая инерция всё же присутствует. Кроме того, влияние оказало наличие конечного времени дискретизации при выполнении расчётов на базе микроконтроллера. Так, поскольку частота дискретизации на МК меньше, чем при расчёте симуляции (20кГц против 1МГц в модели) существует ненулевая вероятность упущения быстрых изменений в сигнале, что может негативно сказаться на точности управления, также данный фактор является причиной «зубчатости» графика переходного процесса. Более того, отличие переходных процессов связано с работой силового преобразователя в режиме широтно-импульсной модуляции, что не учитывалось при разработке имитационной модели.*

*Наиболее сильное влияние оказывает разрешающая способность АЦП, в данном случае это всего 12 бит, что означает, что входной сигнал может быть разделен не более чем на 4096 значений. Также в ходе написания ПО для расчёта значений с плавающей точкой использовалась библиотека IQmath, а именно формат данных IQ24, что ограничивает значение мантиссы числа до 224.*

*В целом результат эксперимента можно считать удачным, поскольку, несмотря на все вышеперечисленные допущения, полученные критерии качества переходного процесса разительно не отличаются от рассчитанных ранее.*

# Контур управления скоростью

Ввиду отсутствия датчика скорости электродвигателя в составе рулевой рейки, для получения сигнала обратной связи предлагается использование датчика положения рулевой рейки, а скорость получить, вычислив первую производную от положения. *Для возможности реализации данного подхода требуется датчик положения с набором определенных качеств, а именно:*

1. *Поскольку все значения шумов, помех и искажений будут значительно усиливаться после взятия производной, требуется кодировка сигнала с датчика положения в помехоустойчивом формате, например импульсном;*
2. *Поскольку для системы управления положением рулевой рейки важно знать позицию рейки сразу после включения питания, должна быть исключена возможность разночтения одной и той же позиции, т.е. датчик положения должен однозначно определять позицию в пределах нескольких оборотах двигателя;*

Оценка положения осуществляется следующим образом:

Алгоритм основан на использовании абсолютного энкодера с импульсным интерфейсом. С данного датчика на микроконтроллер поступают два ШИМ сигнала: А и B (Рисунок 3.1).



Рисунок 3.1 — ШИМ сигнал с датчика положения ротора: здесь сигнал B находится сверху, сигнал A — снизу

Во время движения скважность ШИМ этих сигналов *изменяется в зависимости от угла поворота измерительного вала датчика положения в разных пропорциях, что позволяет организовать абсолютное измерение позиции рулевой рейки в энергонезависимом режиме* (Рисунок 3.2).



Рисунок 3.2 — Осциллограмма ШИМ-сигналов во время движения рулевой рейки, где сигнал A находится снизу, а сигнал B — сверху.

Для характеристики ШИМ-сигнала будем использовать время включенного состояния сигнала за один период. Определять его будем следующим образом: заведём по таймеру для каждого из сигналов. Эти таймеры в ходе работы микроконтроллера будут просто накапливаться, достигать максимального значения и сбрасываться вновь (Рисунок 3.3).

Рисунок 3.3 — График работы TIM2

Начнём фиксировать значения таймера в момент изменения состояния соответствующего сигнала, то есть при переходе из 0 в 1 или из 1 в 0 (Рисунок 3.4).



Рисунок 3.4 — Соотношение сигнала А с независимым таймером

Как видно из рисунка, из соотношения становится известно  и  — время включённого и выключенного состояния сигнала А. Сумма этих двух переменных равняется периоду сигнала.

Поскольку при перемещении рулевой рейки меняется скважность ШИМ, но не меняется период, то введём величину (для каждого сигнала отдельно), которая будет равняться отношению длительности включения (состояния 1, ) к периоду ШИМ:



Здесь T — период ШИМ сигнала А, в среднем  (зависит от рулевой рейки),  принимает разные значения в зависимости от позиции рулевой рейки. Её примерные границы от 0,13 до 0,93 мс.

Выведем *PilaA\_orig* на график и запустим рейку в движение от края до края (Рисунок 3.3).



Рисунок 3.3 — Отображение пилообразного сигнала в канале «А»

Получили пилообразный сигнал. «Пила B» будет иметь такую же форму, но с большим периодом:



Период (что подтверждается рисунками 3.1–3.2),  принимает разные значения в зависимости от позиции рулевой рейки. Её примерные границы: от 0,6 до 4,2 мс.



Рисунок 3.4 — Отображение пилообразного сигнала в канале «B»

Затем эти сигналы были смоделированы в среде динамического моделирования *для разработки и проверки алгоритма получения однозначного определения позиции на их основе*. В качестве блока генерации пилообразного сигнала использовался Repeating Table. Исходя из рисунков 3.3 и 3.4, можно *определить* *количество взаимного соотношения периодов пилообразных сигналов в разных каналах*, при перемещении рейки из одного крайнего положения в другое. Для сигнала А это 29,2 периодов пилообразного сигнала, для сигнала B — 3,94 периодов пилообразного сигнала.

График выглядит данных сигналов отображён на рисунке 3.5.



Рисунок 3.5 — Графики пилообразных сигналов

*Точка взаимного пересечение сигналов на каждом из периодов всегда отличается, что даёт предпосылки для организации определения абсолютного выходного значения для датчика положения.*

*Основная задача заключается при использовании исходных пилообразных сигналов, представленных на рисунке 3.5 получить однозначное представление для выходного абсолютного сигнала датчика положения. Для этой цели* построим основной пилообразный сигнал, *соответствующий абсолютному изменению выходной позиции рулевой рейки. Данный сигнал* будет изменяться от 0 до 1 на всём диапазоне перемещения рулевой рейки от -1000 до +1000. Для этого используем "пилы" A и B. *Для достижения максимальной точности и требуемого абсолютного диапазона измерения положения рулевой рейки выполним следующие преобразования*: *Умножим сигнал A на коэффициент 2, а сигнал B на коэффициент 15, обозначив эти переменные как PilaA\_shift и PilaB\_shift соответственно. Ограничим максимально возможное значение пил маской 0xFFFFFF (2^28).*

*Немного сам не понимаю как это работает на данный момент, но там точно умножение идёт на эти цифры, а в маске точно2^28 степени.*

Таким образом, за перемещение от -1000 до +1000 будет насчитано 58,4 PilaA\_shift и 59,1 PilaB\_shift. *Дополнительные коэффициенты умножения/деления пилообразных сигналов выбираются равными* 2 и 15 для обеспечения точности: в идеале, итоговые значения PilaA\_shift и PilaB\_shift должны отличаться на единицу для повышения точности определения позиции рулевой рейки. В данном случае, разница составила 59,1 – 58,4 = 0,7.

Теперь определим абсолютное выходное значение датчика положения, которое будет считаться по следующему условию:

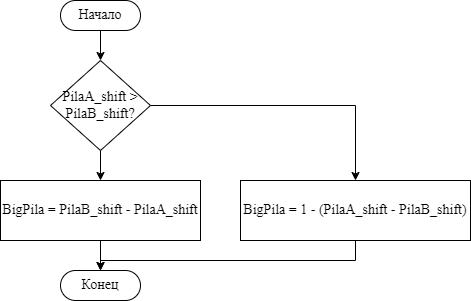


Рисунок 3.6 — Алгоритм расчёта абсолютного выходного сигнала датчика положения в пределах требуемого диапазона

Графически это условие будет выглядеть следующим образом:



Рисунок 3.6 — Расчёт абсолютного выходного сигнала датчика положения в пределах требуемого диапазона



Рисунок 3.7 — График высчитанной «пилы»

Именно используя эту высчитанную «пилу», полученную из двух (А и Б), и считается местоположение рулевой рейки. Величина наклона рассчитанной пилы высчитывается исходя из разницы pilaB\_shift и pilaA\_shift. Чем больше эта разница, тем сильнее наклон прямой, тем точнее мы определяем положение (поскольку разница между двумя соседними значениями больше). Чрезмерно большая разница приведёт к тому, что на один полный ход рейки будет несколько периодов рассчитанного пилообразного сигнала, что недопустимо, поскольку создаёт неоднозначность позиции.

*Далее, поскольку скорость перемещения рулевой рейки не так велика, а также чтобы сэкономить ресурсы микроконтроллера, будем рассчитывать скорость* с частотой 200 раз в секунду, будем оценивать текущее положение рулевой рейки на основе *пилообразного сигнала* А. Этот сигнал 29,2 раза переходит из 0 в 1 на протяжении рабочего диапазона рулевой рейки, что в 6 раз больше, чем *пилообразный сигнал B*. Таким образом, угол наклона А (*это же равнозначно?*) будет в 6 раз больше, что важно для оценивания скорости.

Расчёт производной будет проводиться как разность текущего значения сигнала А и предыдущего значения, *с учётом временного интервала для которого производится расчёт равному* 1/200 секунды (Δt). Дополнительно, введём условия для обработки возможного перехода через «ноль» при определении производной, и на этой основе разработаем функцию определения скорости вращения датчика положения и скорости линейного перемещения штока рулевой рейки. Данная функция описана в приложении Б. Также будем использовать апериодический фильтр первого порядка для уменьшения колебаний и дополнительного сглаживания получаемого результата.

Блок-схема алгоритма управления скоростью представлена на рисунке 3.8. Программная реализация представлена в приложении А.

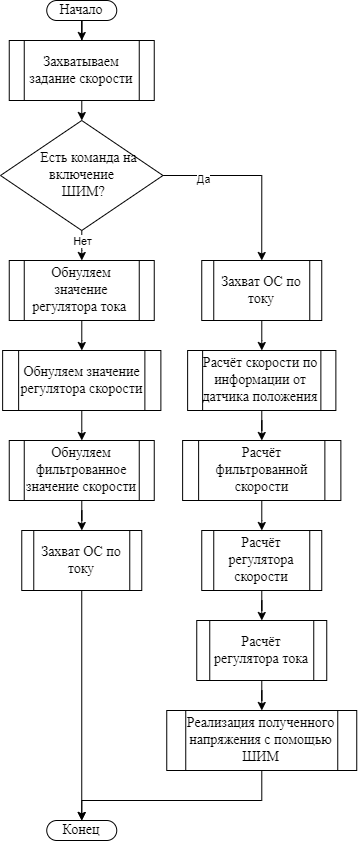


Рисунок 3.8 — Блок схема алгоритма управления скоростью

Используя кинематические преобразования (поскольку все параметры механических передач нам известны из прошлых работ), определим скорость вращения электродвигателя рулевой рейки. Построим график изменения скорости при задании ступенчатого сигнала и сравним её с моделируемыми значениями:

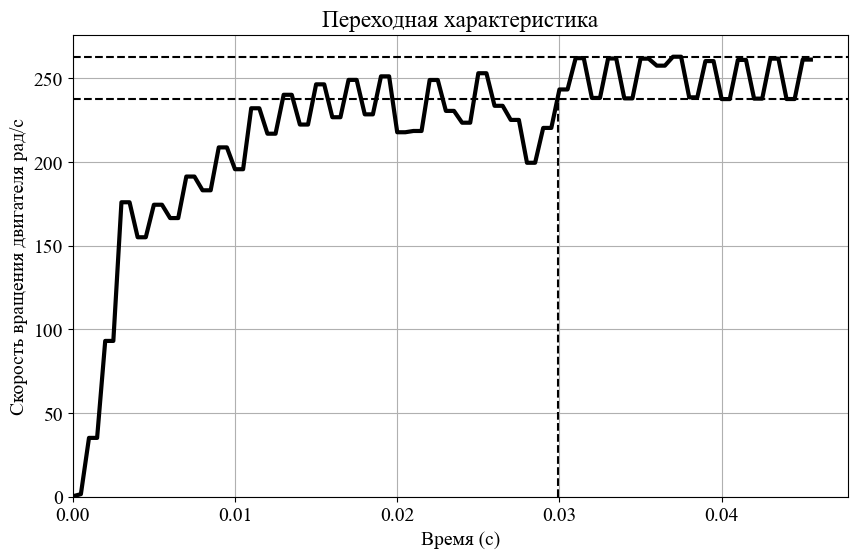


Рисунок 3.9 — Переходный процесс

Показатели качества реального процесса: время переходного процесса 0.03, перерегулирование 0%.



Рисунок 3.10 — Переходный процесс в имитационной модели

В модели же получили время переходного процесса 0,004 секунды, перерегулирование 4%.

*Графики 3.9 и 3.10 однозначно имеют общие черты и визуально схожи, поскольку при разработке имитационной модели контура тока стояла задача имитации именно этого электропривода.*

*Тем не менее, видны и отличия. Например, отличается как время регулирования переходного процесса, так и значение перерегулирования. Также переходный процесс на рисунке 3.9 имеет более «рваный» характер.*

*Несоответствие критериев качества переходного процесса объясняется рядом допущений, выполненных в ходе разработки имитационной модели контура тока, например: предполагалось, что в цепи обратной связи скорости считается идеальным, т е. передаточная функция датчика равна 1, но в действительности существует инерция в канале обратной связи, а также ограничение по частоте расчёта. Так, поскольку частота дискретизации на МК меньше, чем при расчёте симуляции (200 Гц против 1МГц в модели) существует ненулевая вероятность упущения быстрых изменений в сигнале, что может негативно сказаться на точности управления, также данный фактор является причиной «зубчатости» графика переходного процесса. Более того, отличие переходных процессов связано с работой силового преобразователя в режиме широтно-импульсной модуляции, что не учитывалось при разработке имитационной модели. Также наличие противоЭДС при вращении двигателя также оказывает серьёзное влияние. Помимо этого в ходе написания ПО для расчёта значений с плавающей точкой использовалась библиотека IQmath, а именно формат данных IQ24, что ограничивает значение мантиссы числа до 224.*

*В целом результат эксперимента можно считать удачным, поскольку, несмотря на все вышеперечисленные допущения, полученные критерии качества переходного процесса разительно не отличаются от рассчитанных ранее.*

# Контур положения

Положение для формирования сигнала обратной связи считывается по алгоритму, описанному в предыдущем пункте. Блок-схема алгоритма организации управления положением представлена на рисунке 4.1.

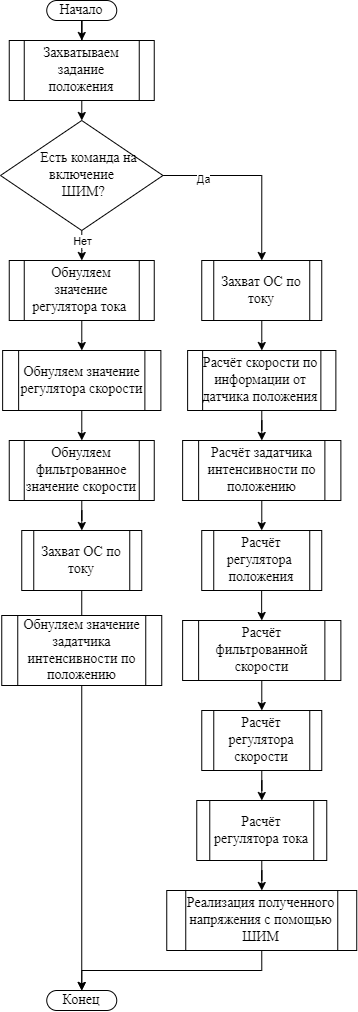


Рисунок 4.1 — Алгоритм управления положением

Программная реализация отображена в приложении А.

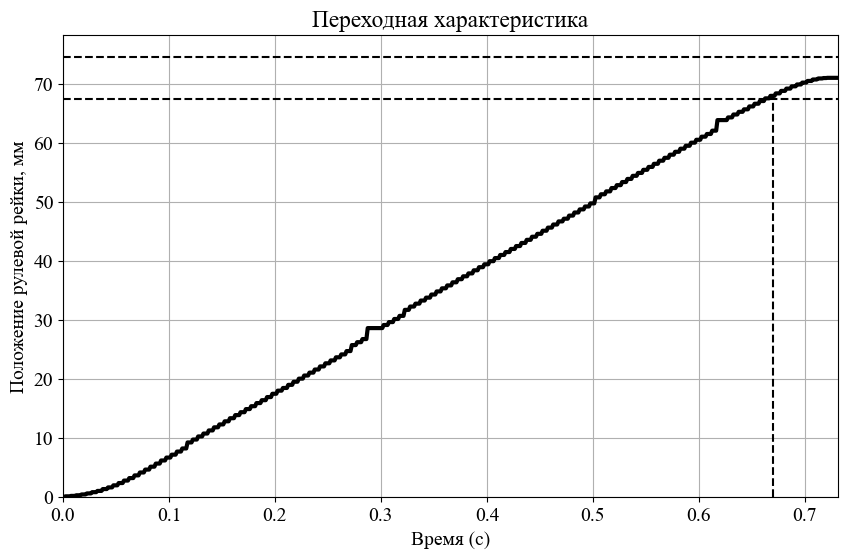


Рисунок 4.1 — Переходный процесс по положению

Показатели качества: перерегулирование 0%, время регулирования 0.73с.



Рисунок 4.2 — Смоделированное перемещение рулевой рейки

Показатели качества: перерегулирование 0%, время регулирования 0.705с.

Небольшое отличие объясняется теми же причинами, что и несоответствие прошлых переходных характеристик. Тем не менее, отличие времени регулирования на 3,45% является небольшим отклонением, что говорит об успешном синтезе системы управления электроприводом рулевой рейкой.

# Заключение

Полученные в ходе экспериментов переходные процессы, несмотря на незначительные отклонения от моделируемых значений, в целом подтвердили точность динамического имитационного моделирования и полученных результатов, поскольку наличие этих отклонений было неизбежным следствием аппроксимаций и допущений, выполненных в ходе разработки имитационной модели. Показатели качества всех процессов сопоставимы с показателями качества моделей по итогам проводимого динамического имитационного моделирования. Таким образом, экспериментальная реализация контуров управления током, скоростью и положением может быть признана успешной.

Полученная система управления используется для управления рулевых реек беспилотных транспортных средств, например, компанией ЭвоКарго?

**Список литературы**

1. Беспилотные автомобили. Состояние рынка, тренды и перспективы развития // iot.ru URL: https://iot.ru/transportnaya-telematika/bespilotnye-avtomobili-sostoyanie-rynka-trendy-i-perspektivy-razvitiya (дата обращения: 28.03.2024).

2. Человеческий фактор как главный виновник дорожных аварий. Как он появился и насколько актуален сегодня // techinsider.ru URL: https://www.techinsider.ru/vehicles/768513-chelovecheskiy-faktor-kak-glavnyy-vinovnik-dorozhnyh-avariy-kak-on-poyavilsya-i-naskolko-aktualen-segodnya/ (дата обращения: 28.03.2024).

3. Распоряжение Правительства РФ от 28.12.2022 N 4261-р <Об утверждении Стратегии развития автомобильной промышленности Российской Федерации до 2035 года>.

Приложение А

Алгоритмы управления током, скоростью и положением

Листинг 1 — Файл motorHiCtrl.c

1. // Тестовая функция для проверки возможностей PWM

2. #include "motorHiCtrl.h"

3. #include "pwm\_test.h"

4. #include "g\_Ram.h"

5. #include "speed\_fr.h"

6. #include "peref.h"

7. #include "calibs.h"

8. #include "servo.h"

9.

10. MTR\_Ctrl Mot;

11.

12. extern int32\_t sin\_table[106];

13. extern int32\_t cos\_table[106];

14. extern int32\_t atg\_table[106];

15. extern float atg\_tableFL[106];

16.

17. extern uint16\_t CAN\_Mode;

18.

19. uint16\_t timerPOS;

20. uint16\_t timerPOS\_1s;

21. uint16\_t timerSPD;

22. uint16\_t timerSPD\_1s;

23.

24. void set\_atg\_tableFL(void);

25. void atanTest(MTR\_Ctrl \*v);

26.

27. void MTR\_Init(MTR\_Ctrl \*v)

28. {

29. uint16\_t RefTF = 4000; // => 0.04 сек.

30. // Начальная инициализация

31. //v->SetFreq = 1000; // 10 Гц

32. v->SetFreq = 20; // 0.2 Гц

33. v->SetVolt = 100; // 100.0 % от всей амплитуды

34. v->Qs = 0;

35.

36. v->atanTest = 0;

37. v->ResetMechTeta = 0;

38.

39. //v->rc1.RampDelta = 83886; // 1/200 = 0.005 в формате Q24 при частоте вызова 2 кГц потребуется 200 шагов для того чтобы 200\*0,005 = 1 или 0,1 сек

40. v->rc1.RampDelta = 8389; // 1/2000 = 0.0005 в формате Q24 при частоте вызова 20 кГц потребуется 2000 шагов для того чтобы 2000\*0,0005 = 1 или 0,1 сек

41. v->rc1.RampDelayMax = 1; // минимальная длительность без формирования дополнительных задержек

42. // v->rc1.RampHighLimit = 16777216; // +1.0 в Q24

43. // v->rc1.RampLowLimit = -16777216; // -1.0 в Q24

44. v->rc1.RampHighLimit = 33554432; // +2.0 в Q24

45. v->rc1.RampLowLimit = -33554432; // -2.0 в Q24

46.

47. //v->m1 = BASE\_FREQ;

48. //v->m2 = PWM\_DELTAT;

49. //v->StepAngleMax = Q24\_MUL(BASE\_FREQ, PWM\_DELTAT);

50. //v->StepAngleMax = Q24\_MUL(v->m2, v->m1);

51.

52. v->StepAngleBase = 41943; // 0,0025 в формате Q24 с целью получить синусоиду с периодом 50Гц на прерывании 20 кГц при задании v->Freq = 1.0 в Q24

53.

54. v->StepAngleMax = v->StepAngleBase;

55.

56. v->Freq = v->SetFreq \* From5000toQ24;

57.

58. v->StartResetTimer = 0;

59. v->StartResetDelay = 1000;

60.

61. set\_atg\_tableFL();

62.

63. for (int i=0; i < 105; ++i)

64. { // формируем синус и косинус

65. sin\_table[i] = FLtoQ16(256.0f \* sin(i \* 0.0628f));

66. cos\_table[i] = FLtoQ16(256.0f \* cos(i \* 0.0628f));

67. atg\_table[i] = FLtoQ16(atg\_tableFL[i] \* 256.0f);

68. }

69. //set\_atg\_table();

70. Peref\_Filter1Init(&v->filter1\_REF, Prd20kHZ, RefTF);

71. }

72.

73. void MTR\_HiUpdt(MTR\_Ctrl \*v) //20 kHz

74. {

75. static uint16\_t ISRScaleTimer;

76.

77. if (ISRScaleTimer < Pwm.PwmFrqScale)

78. {

79. ISRScaleTimer++;

80. return;

81. }

82. else

83. ISRScaleTimer = 1;

84.

85. if (g\_Ram.ramGroupD.MODE\_SET < 3) // режим НЕ КОНТУР СКОРОСТИ и НЕ КОНТУР ПОЛОЖЕНИЯ

86. {

87. if (++v->ScaleTimer >= v->ScaleTimeOut) // переход c 20 кГц на 200 Гц

88. {

89. v->ScaleTimer = 0;

90. // Spd.MechTheta = Mot.MotMechTeta; // Расчёт

91. speed\_fr\_calc(&Spd); // и фильтрация

92. // скорости вращения

93. // if (GrC->SpdDir) // при необходимости меняем знак скорости

94. // g\_Peref.filter1\_Spd.Input = -Spd.Speed;

95. // else

96. g\_Peref.filter1\_Spd.Input = Spd.Speed;

97.

98. Peref\_Filter1Calc(&g\_Peref.filter1\_Spd);

99. }

100. }

101.

102. // if (!Pwm.PwmMode) // если ДПТ

103. // {

104. // if (Pwm.T1\_inp > Pwm.Period/2)

105. // v->IDs = v->I2; // Захват обратной связи с АЦП I1

106. // else v->IDs = -v->I1; // Захват обратной связи с АЦП I2

107. // }

108.

109. if (GrC->PwmDir) // реверс ШИМ

110. {

111. v->IAlpha = -v->I1; // Прямое преобразование Кларка 3->2 IAlpha = IA;

112. v->IBeta = \_IQ24mpy(-v->I1 + (-v->I2 << 1), Q24\_1DIVSQRT3); // IBeta = (IA + IB/2) \* 1/SQRT(3)

113. }

114. else

115. {

116. v->IAlpha = v->I1; // Прямое преобразование Кларка 3->2 IAlpha = IA;

117. v->IBeta = \_IQ24mpy(v->I1 + (v->I2 << 1), Q24\_1DIVSQRT3); // IBeta = (IA + IB/2) \* 1/SQRT(3)

118. }

119.

120. // Spd.MechTheta = Mot.MotMechTeta; // Расчёт

121. // speed\_fr\_calc(&Spd); // и фильтрация // скорости вращения

122. // if (GrC->SpdDir) // при необходимости меняем знак ОС по скорости

123. // g\_Peref.filter1\_Spd.Input = -Spd.Speed;

124. // else

125. // g\_Peref.filter1\_Spd.Input = Spd.Speed;

126. // Peref\_Filter1Calc(&g\_Peref.filter1\_Spd);

127.

128.

129.

130. switch(g\_Ram.ramGroupD.MODE\_SET) // режимы управления мотором

131. {

132. case 0: // скалярное

133. {

134. rampgen\_calc(v);

135. v->Teta = v->RampOut;

136. v->IQs = v->IAlpha; // Захват обратной связи с АЦП I1

137. } break;

138.

139. case 1: // токовое

140.

141. case 2: // токовое с ориентацие ротора по потоку

142. {

143. if (!g\_Ram.ramGroupD.CONTR\_REG) // нет команды на движение

144. {

145. pid\_reg\_reset(&v->CurrId); // В СТОПЕ вызываем

146. pid\_reg\_reset(&v->CurrIq); // сброс регуляторов тока

147.

148. v->Ds = 0;

149. v->Qs = 0;

150. v->StartResetTimer = 0;

151. v->IQs = v->IAlpha; // Захват обратной связи с АЦП I1

152. }

153. else

154. { // Работа в токовом режиме

155. if (++v->StartResetTimer > v->StartResetDelay/Pwm.PwmFrqScale) v->StartResetTimer = v->StartResetDelay/Pwm.PwmFrqScale;

156.

157. if (!Pwm.PwmMode) // если ДПТ

158. v->IQs = v->IAlpha; // Захват обратной связи с АЦП I1

159. else park\_calc(v); // обработка ОС по токам Ia, Ib

160.

161. rampgen\_calc(v); // формирование пилы для вращения вектора тока

162. v->Teta = v->RampOut; // передача пилы для ориентации СК

163.

164. v->CurrId.Fdb = v->IDs; // сигналы составляющих вектора тока во вращающейся

165. v->CurrIq.Fdb = v->IQs; // системе координат

166.

167. pid\_reg\_calc(&v->CurrId); // Вызываем процедуры для

168. pid\_reg\_calc(&v->CurrIq); // расчёта регуляторов составляющих вектора тока

169.

170. v->Ds = v->CurrId.Out; // Выходы регуляторов составляющих вектора тока

171. v->Qs = v->CurrIq.Out; // отправляем на формирование ШИМ

172. }

173. } break;

174.

175. case 3: // контур скорости

176. {

177. if (!g\_Ram.ramGroupD.CONTR\_REG) // нет команды на движение

178. {

179. Peref\_Filter1Reset(&g\_Peref.filter1\_Spd);

180.

181. pid\_reg\_reset(&v->CurrId); // В СТОПЕ вызываем

182. pid\_reg\_reset(&v->CurrIq); // сброс регуляторов тока

183. pid\_reg\_reset(&v->Spd); // сброс регулятора скорости

184.

185. v->Ds = 0;

186. v->Qs = 0;

187. v->StartResetTimer = 0;

188.

189. v->ScaleTimer = v->ScaleTimeOut; // расчёты по КС на первом этапе после запуска

190. v->IQs = v->IAlpha; // Захват обратной связи с АЦП I1

191. }

192. else

193. { // Работа в контуре скорости

194. if (++v->StartResetTimer > v->StartResetDelay/Pwm.PwmFrqScale) v->StartResetTimer = v->StartResetDelay/Pwm.PwmFrqScale;

195.

196. v->Teta = v->MotElecTeta; //v->RampOut; // передача пилы для ориентации СК

197.

198. if (!Pwm.PwmMode) // если ДПТ

199. {

200. v->IQs = v->IAlpha; // Захват обратной связи с АЦП I1

201. }

202. else park\_calc(v); // обработка ОС по токам Ia, Ib

203.

204. if (++v->ScaleTimer >= v->ScaleTimeOut) // переход на 200 Гц

205. {

206. v->ScaleTimer = 0;

207. // Spd.MechTheta = Mot.MotMechTeta; // Расчёт

208. speed\_fr\_calc(&Spd); // и фильтрация

209. // скорости вращения

210. // if (GrC->SpdDir) // при необходимости меняем знак ОС по скорости

211. // g\_Peref.filter1\_Spd.Input = -Spd.Speed;

212. // else

213. g\_Peref.filter1\_Spd.Input = Spd.Speed;

214.

215. // if (GrC->PwmDir) // реверс ШИМ

216. // {

217. // g\_Peref.filter1\_Spd.Input = -Spd.Speed;

218. // }

219. // else

220. // {

221. // g\_Peref.filter1\_Spd.Input = Spd.Speed;

222. // }

223.

224.

225. Peref\_Filter1Calc(&g\_Peref.filter1\_Spd);

226.

227. //GrA->SPEED\_Q24 = g\_Peref.filter1\_Spd.OutputIQ16;

228.

229. v->Spd.Fdb = g\_Peref.filter1\_Spd.OutputIQ16;

230.

231. pid\_reg\_calc(&v->Spd); // Расчёт регулятора скорости

232.

233. //NeutralZoneCtrl(&v->Spd.Out);

234.

235. v->CurrIq.Ref = v->Spd.Out;

236. }

237.

238. v->CurrId.Fdb = v->IDs; // сигналы составляющих вектора тока во вращающейся

239. v->CurrIq.Fdb = v->IQs; // системе координат

240.

241. pid\_reg\_calc(&v->CurrId); // Вызываем процедуры для

242. pid\_reg\_calc(&v->CurrIq); // расчёта регуляторов составляющих вектора тока

243.

244. v->Ds = v->CurrId.Out; // Выходы регуляторов составляющих вектора тока

245. v->Qs = v->CurrIq.Out; // отправляем на формирование ШИМ

246. // if ()

247. // v->Qs +=

248. }

249. } break;

250.

251. case 4: // контур положения

252. {

253. if (!g\_Ram.ramGroupD.CONTR\_REG) // нет команды на движение

254. {

255. Peref\_Filter1Reset(&g\_Peref.filter1\_Spd);

256.

257. v->rc1.SetpointValue = 0; // сброс

258. v->rc1.RampDelayCount = 0; // сброс

259.

260. pid\_reg\_reset(&v->CurrId); // В СТОПЕ вызываем

261. pid\_reg\_reset(&v->CurrIq); // сброс регуляторов тока

262. pid\_reg\_reset(&v->Spd); // сброс регулятора скорости

263. pid\_reg\_reset(&v->Pos); // сброс регулятора положения

264.

265. v->PosError = 0;

266. v->PosregOutTmp = 0;

267.

268. v->StartResetTimer = 0;

269.

270. v->Ds = 0;

271. v->Qs = 0;

272.

273. v->ScaleTimer = v->ScaleTimeOut; // расчёты по КС на первом этапе после запуска

274. v->IQs = v->IAlpha; // Захват обратной связи с АЦП I1

275. }

276. else

277.

278. {

279. if (++v->StartResetTimer > v->StartResetDelay/Pwm.PwmFrqScale) v->StartResetTimer = v->StartResetDelay/Pwm.PwmFrqScale;

280.

281. if (!Pwm.PwmMode) // если ДПТ

282. {

283. v->IQs = v->IAlpha; // Захват обратной связи с АЦП I1

284. }

285. else park\_calc(v); // обработка ОС по токам Ia, Ib

286.

287. rampgen\_calc(v); // формирование пилы для вращения вектора тока

288. v->Teta = v->MotElecTeta; //v->RampOut; // передача пилы для ориентации СК

289.

290. v->rc1.TargetValue = ((LgInt)GrA->POS\_SET) << 8;

291. rmp\_cntl\_calc(&v->rc1);

292.

293. v->filter1\_REF.Input = v->rc1.SetpointValue;

294. Peref\_Filter1Calc(&v->filter1\_REF); // фильтр-сглаживатель ЗИ по положению

295.

296. if (++v->ScalePosTimer >= v->ScalePosTimeOut) // переход на 50 Гц

297. {

298. v->ScalePosTimer = 0;

299. if (timerPOS++ >= GrB->FREQ\_POS\_CALC)

300. {

301. timerPOS\_1s++;

302. timerPOS = 0;

303. }

304.

305. v->PosError = (int32\_t)GrA->POS\_SET - GrA->POS; // Ошибка = Вых.ЗИ - ОС

306. //v->PosError = (int32\_t)(v->filter1\_REF.OutputIQ16 >> 8) - GrA->POS; // Ошибка = Вых.ЗИ - ОС

307. if (labs(v->PosError) < Calib\_ETT.zone\_abs)

308. v->PosError = 0;

309.

310. // if (GrB->CorrGain) // Корректировка упругих связей (УС) включена

311. // {

312.

313. // }

314. // else // нет корректировки УС

315. // {

316.

317. // }

318.

319. if (GrC->PosPIKp) // Если Кп нового регулятора положения не равно 0

320. { // Универсальный регулятор

321. v->Pos.Ref = (int32\_t)GrA->POS\_SET; // задание на положение

322. v->Pos.Fdb = GrA->POS; // обратная связь по положению

323.

324. if (v->PosError) // проверка условия на попадание в зону нечувствительности

325. { // работаем

326. pid\_reg\_calc(&v->Pos); // расчёт регулятора положения

327. v->PosregOutTmp = v->Pos.Out \* 1000; // вывод регулятора положения в аналогичном формате

328. }

329. else

330. { // сброс регулятора, попадание в зону нечувствительности

331. pid\_reg\_reset(&v->Pos); // сброс регулятора положения

332. v->PosregOutTmp = 0; // обнуление выхода РП

333. }

334.

335. }

336. else

337. { // П-Регулятор " по старому "

338.

339. if (v->PosError > v->limPos) v->PosError = v->limPos;

340. else if (v->PosError < -v->limPos) v->PosError = -v->limPos;

341. //v->PosregOutTmp = \_IQ24mpy(v->PosError, v->PosregKp);

342. v->PosregOutTmp = \_IQ24mpy(v->PosError \* 1000, v->PosregKp);

343. }

344. }

345.

346. if (++v->ScaleTimer >= v->ScaleTimeOut) // переход на 200 Гц

347. {

348. v->ScaleTimer = 0;

349. if (timerSPD++ >= GrB->FREQ\_SPD\_CALC)

350. {

351. timerSPD\_1s++;

352. timerSPD = 0;

353. }

354.

355. if (v->StartResetTimer >= v->StartResetDelay/Pwm.PwmFrqScale) v->Spd.Ref = v->PosregOutTmp; // условие спокойного старта при включении ШИМ

356. else

357. {

358. pid\_reg\_reset(&v->Pos); // сброс регулятора положения

359. v->Spd.Ref = 0; // обнуление задания на контур скорости

360. }

361.

362. // if (!GrC->SpdRefDir) // если нет инверсии по заданию скорости

363. // {

364. // if (Calib.InvDirect) // Если есть инверсия положения

365. // {

366. // v->Spd.Ref = (-1)\*v->Spd.Ref; // исправлено для адекватности между КС и КП

367. // }

368. // }

369. // else

370. // { // если есть инверсия по заданию скорости

371. // if (!Calib.InvDirect) // Если нет инверсии положения

372. // {

373. // v->Spd.Ref = (-1)\*v->Spd.Ref; // исправлено для адекватности между КС и КП

374. // }

375. // }

376.

377. if (!Calib.InvDirect) // Если нет инверсии положения

378. {

379. v->Spd.Ref = (-1)\*v->Spd.Ref; // исправлено для адекватности между КС и КП

380. }

381. //

382. // if (GrC->PwmDir) // реверс ШИМ

383. // {

384. // v->Spd.Ref = (-1)\*v->Spd.Ref;

385. // }

386. // else

387. // {

388. // v->Spd.Ref = v->Spd.Ref;

389. // }

390.

391. // Spd.MechTheta = Mot.MotMechTeta; // Расчёт

392. speed\_fr\_calc(&Spd); // и фильтрация

393. // скорости вращения

394. // if (GrC->SpdDir) // при необходимости меняем знак скорости

395. // g\_Peref.filter1\_Spd.Input = -Spd.Speed;

396. // else

397. g\_Peref.filter1\_Spd.Input = Spd.Speed;

398.

399. Peref\_Filter1Calc(&g\_Peref.filter1\_Spd);

400.

401. v->Spd.Fdb = g\_Peref.filter1\_Spd.OutputIQ16;

402. pid\_reg\_calc(&v->Spd); // Расчёт регулятора скорости

403.

404. v->CurrIq.Ref = v->Spd.Out;

405. }

406.

407. v->CurrId.Fdb = v->IDs; // сигналы составляющих вектора тока во вращающейся

408. v->CurrIq.Fdb = v->IQs; // системе координат

409.

410. pid\_reg\_calc(&v->CurrId); // Вызываем процедуры для

411. pid\_reg\_calc(&v->CurrIq); // расчёта регуляторов составляющих вектора тока

412.

413. v->Ds = v->CurrId.Out; // Выходы регуляторов составляющих вектора тока

414. v->Qs = v->CurrIq.Out; // отправляем на формирование ШИМ

415. }

416. } break;

417.

418. case 5: // моментный режим

419. {

420. if (!g\_Ram.ramGroupD.CONTR\_REG) // нет команды на движение

421. {

422. Peref\_Filter1Reset(&g\_Peref.filter1\_Spd);

423.

424. pid\_reg\_reset(&v->CurrId); // В СТОПЕ вызываем

425. pid\_reg\_reset(&v->CurrIq); // сброс регуляторов тока

426. pid\_reg\_reset(&v->Spd); // сброс регулятора скорости

427.

428. v->Ds = 0;

429. v->Qs = 0;

430. v->StartResetTimer = 0;

431.

432. v->ScaleTimer = v->ScaleTimeOut; // расчёты по КС на первом этапе после запуска

433. //Spd.MechThetaPrev = 0;

434. }

435. else

436. { // Работа в контуре скорости

437. if (++v->StartResetTimer > v->StartResetDelay/Pwm.PwmFrqScale) v->StartResetTimer = v->StartResetDelay/Pwm.PwmFrqScale;

438.

439. v->Teta = v->MotElecTeta; //v->RampOut; // передача пилы для ориентации СК

440.

441. if (!Pwm.PwmMode) // если ДПТ

442. {

443. v->IQs = v->IAlpha; // Захват обратной связи с АЦП I1

444. }

445. else park\_calc(v); // обработка ОС по токам Ia, Ib

446.

447. if (++v->ScaleTimer >= v->ScaleTimeOut) // переход на 200 Гц

448. {

449. v->ScaleTimer = 0;

450. // Spd.MechTheta = Mot.MotMechTeta; // Расчёт

451. // speed\_fr\_calc(&Spd); // и фильтрация

452. // // скорости вращения

453. // if (GrC->SpdDir) // при необходимости меняем знак ОС по скорости

454. // g\_Peref.filter1\_Spd.Input = -Spd.Speed;

455. // else

456. // g\_Peref.filter1\_Spd.Input = Spd.Speed;

457. //

458. // Peref\_Filter1Calc(&g\_Peref.filter1\_Spd);

459.

460. v->Spd.Fdb = g\_Peref.filter1\_Spd.OutputIQ16;

461. pid\_reg\_calc(&v->Spd); // Расчёт регулятора скорости

462.

463. // NeutralZoneCtrl(&v->Spd.Out);

464.

465. //моментный режим с ограничением скорости

466. // if (labs(v->Spd.Fdb) > labs(v->Spd.Ref))

467. // v->CurrIq.Ref = v->Spd.Out;

468. // else

469. //v->CurrIq.Ref = (int32\_t)(g\_Ram.ramGroupC.rxSetCurr \* From1000toQ24);

470. v->CurrIq.Ref = Prot.SetTestCurr; //(int32\_t)(g\_Ram.ramGroupC.SetTestCurr \* From1000toQ24);

471. }

472.

473. v->CurrId.Fdb = v->IDs; // сигналы составляющих вектора тока во вращающейся

474. v->CurrIq.Fdb = v->IQs; // системе координат

475.

476. pid\_reg\_calc(&v->CurrId); // Вызываем процедуры для

477. pid\_reg\_calc(&v->CurrIq); // расчёта регуляторов составляющих вектора тока

478.

479. v->Ds = v->CurrId.Out; // Выходы регуляторов составляющих вектора тока

480. v->Qs = v->CurrIq.Out; // отправляем на формирование ШИМ

481. }

482. } break;

483. }

484.

485. if (!Pwm.PwmMode) // если ДПТ

486. {

487. if (g\_Ram.ramGroupD.MODE\_SET)

488. {

489. if (GrC->PwmDir) v->Alpha = -v->CurrIq.Out; // Выходы регулятора тока с инверсией

490. else v->Alpha = v->CurrIq.Out; // Выходы регулятора тока без инверсии

491. }

492. }

493. else ipark\_calc(v);

494.

495. Pwm.Ualpha = v->Alpha; // отправляем Ua для формирования ШИМ

496. Pwm.Ubeta = v->Beta; // отправляем Ub для формирования ШИМ

497.

498. //if (v->atanTest) atanTest(v);

499. }

500.

501.

502. //void PWM\_HiUpdtT3(PWM\_Test \*v, TIM\_HandleTypeDef\* htim3)

503. //{

504. // if (v->TBR > 0)

505. // {

506. // //if (HAL\_GPIO\_ReadPin(GPIOB, BR\_ENABLE\_Pin)) BR\_ENABLE;

507. // if (!v->BR\_PWMInit)

508. // {

509. // PWM\_InitT3(v, htim3);

510. // v->BR\_PWMInit = 1;

511. // }

512. // else if (HAL\_GPIO\_ReadPin(GPIOB, BR\_ENABLE\_Pin)) BR\_ENABLE;

513. //

514. // if (v->TBR > Q24\_MAX\_Tx\_OUT) v->TBR = Q24\_MAX\_Tx\_OUT;

515. // else if (v->TBR < 0) v->TBR = 0;

516. //

517. // v->TBRpu = v->TBR + \_IQ24mpy(v->TBR, -50331648) + Q24\_one;

518.

519. // if (v->TBRpu > Q24\_one) v->TBRpu = Q24\_one;

520. // else if (v->TBRpu < -Q24\_one) v->TBRpu = -Q24\_one;

521.

522. // v->TBR\_inp = \_IQ24mpy((v->TBRpu + Q24\_one) >> 1, v->PeriodBR);

523. //

524. // htim3->Instance->CCR1 = v->TBR\_inp;

525. //

526. // if (v->TBR\_Prev != v->TBR)

527. // {

528. // //htim3->Instance->CNT = htim3->Instance->ARR;

529. // //htim3->Instance->CCR1 = htim3->Instance->CNT - 10;

530. // htim3->Instance->CR1 |= TIM\_CR1\_CEN;

531. // htim3->Instance->CCER |= TIM\_CCER\_CC1E;

532. // }

533. // }

534. // else

535. // {

536. // //if (!HAL\_GPIO\_ReadPin(GPIOB, BR\_ENABLE\_Pin)) BR\_DISABLE;

537. //

538. // if (v->TBR\_Prev != v->TBR)

539. // {

540. // htim3->Instance->CCER &= ~TIM\_CCER\_CC1E;

541. // htim3->Instance->CR1 &= ~TIM\_CR1\_CEN;

542. // htim3->Instance->CNT = 0;

543. // //htim3->Instance->CCR1 = htim3->Instance->ARR;

544. // }

545. // }

546. //

547. // v->TBR\_Prev = v->TBR;

548. //

549. // if (++v->T3.Timer >= v->T3.TimePeriod)

550. // {

551. // v->T3.Timer = 0;

552. // v->T3.BigTimer++;

553. // }

554. //}

555.

556. void MTR\_LoUpdt(MTR\_Ctrl \*v)

557. {

558. switch(g\_Ram.ramGroupD.MODE\_SET)

559. {

560. case 0: // скалярное

561. {

562. if (!Pwm.PwmMode) // если ДПТ

563. {

564. if (GrC->PwmDir) v->Alpha = (int32\_t)(-v->SetVolt \* \_IQ24mpy(From1000toQ24, Q24\_MAX\_Tx\_OUT) ); // задание напряжения

565. else v->Alpha = (int32\_t)(v->SetVolt \* \_IQ24mpy(From1000toQ24, Q24\_MAX\_Tx\_OUT) ); // задание напряжения

566. }

567. else

568. {

569. v->Freq = (int32\_t)(v->SetFreq \* From5000toQ24);

570. v->Ds = (int32\_t)(v->SetVolt \* \_IQ24mpy(From1000toQ24, Q24\_MAX\_Tx\_OUT) );

571. }

572. } break;

573.

574. case 1: // токовое с принудительной ориентацией вектора тока

575. {

576. v->Freq = (int32\_t)(v->SetFreq \* From5000toQ24);

577. if (v->StartResetTimer >= v->StartResetDelay/Pwm.PwmFrqScale)

578. {

579. v->CurrId.Ref = (int32\_t)(g\_Ram.ramGroupD.CurrIdSet \* From1000toQ24); // Задание в токовом режиме, из 100 (1.00 А) делаем 0,1 в о.е.,

580. v->CurrIq.Ref = (int32\_t)(g\_Ram.ramGroupD.CurrIqSet \* From1000toQ24); // что соответствует току в 1А

581. }

582. else

583. {

584. v->CurrId.Ref = 0;

585. v->CurrIq.Ref = 0;

586. }

587. } break;

588.

589. case 2: // токовое с поиском "нулевого" положения ротора

590. {

591. v->Freq = 0; //v->SetFreq \* From5000toQ24;

592.

593. v->CurrId.Ref = (int32\_t)(g\_Ram.ramGroupD.CurrIdSet \* From1000toQ24); // Задание в токовом режиме, из 100 (1.00 А) делаем 0,1 в о.е.,

594. v->CurrIq.Ref = (int32\_t)(g\_Ram.ramGroupD.CurrIqSet \* From1000toQ24); // что соответствует току в 1А

595. } break;

596.

597. case 3: // контур скорости

598. {

599. v->CurrId.Ref = 0; // условие для векторного управления СД

600.

601. if (v->StartResetTimer >= v->StartResetDelay/Pwm.PwmFrqScale)

602. {

603. v->Spd.Ref = GrC->rxSetSpeed << 15;

604. }

605. else v->Spd.Ref = 0;

606.

607. } break;

608.

609. case 4: // контур положения

610. {

611. v->CurrId.Ref = 0; // условие для векторного управления СД

612.

613. } break;

614.

615. case 5: // моментный режим

616. {

617. v->CurrId.Ref = 0; // условие для векторного управления СД

618. if (v->StartResetTimer >= v->StartResetDelay/Pwm.PwmFrqScale)

619. v->Spd.Ref = Q16\_DIV(GrC->rxSetSpeed, INTtoQ16(GrB->FREQ\_MOT\_NOM)) << 8;

620. else

621. v->Spd.Ref = 0;

622.

623. } break;

624. }

625. if ((CAN\_Mode == 2)||(CAN\_Mode == 3)) return;

626. //servo\_calc(&servo);

627. }

628.

629. void rampgen\_calc(MTR\_Ctrl \*v)

630. {

631. // Compute the angle rate

632. v->Ramp += \_IQ24mpy(v->StepAngleMax, v->Freq);

633.

634. // Saturate the angle rate within (0...1)

635. if (v->Ramp > Q24\_one) v->Ramp -= Q24\_one;

636. else if (v->Ramp < 0) v->Ramp += Q24\_one;

637.

638. // Compute the ramp output

639. v->RampOut = v->Ramp;

640. //v->RampOut = Q24\_one - v->Ramp;

641.

642. // // Saturate the ramp output within (0...1)

643. // if (v->RampOut > Q24\_one) v->RampOut -= Q24\_one;

644. // else if (v->RampOut < Q24\_one) v->RampOut += Q24\_one;

645. }

646.

647.

648. void ipark\_calc(MTR\_Ctrl \*v)

649. {

650. int32\_t Cosine, Sine;

651.

652. Sine = sinQ24pu(v->Teta);

653. Cosine = cosQ24pu(v->Teta);

654.

655. v->Alpha = \_IQ24mpy(v->Ds, Cosine) - \_IQ24mpy(v->Qs, Sine);

656. v->Beta = \_IQ24mpy(v->Qs, Cosine) + \_IQ24mpy(v->Ds, Sine);

657. }

658.

659. void park\_calc(MTR\_Ctrl \*v)

660. {

661. int32\_t Cosine, Sine;

662.

663. Sine = sinQ24pu(v->Teta);

664. Cosine = cosQ24pu(v->Teta);

665.

666. v->IDs = \_IQ24mpy(v->IAlpha, Cosine) + \_IQ24mpy(v->IBeta, Sine);

667. v->IQs = \_IQ24mpy(v->IBeta, Cosine) - \_IQ24mpy(v->IAlpha, Sine);

668. }

669.

670. void atanTest(MTR\_Ctrl \*v)

671. {

672. if (v->atanTest == 1) v->TestTeta = atanQ24pu(v->Alpha, v->Beta);

673. else if (v->atanTest == 2) v->TestTeta = atanQ24pu(v->atanAlphaSet \* From1000toQ24, v->atanBetaSet \* From1000toQ24);

674. else if (v->atanTest == 3)

675. {

676. v->atanAlpha = cosQ24pu(v->atanInTetaSet);

677. v->atanBeta = sinQ24pu(v->atanInTetaSet);

678. v->TestTeta = atanQ24pu(v->atanAlpha, v->atanBeta);

679. }

680. }

681.

682. //зона не чувстительности

683. //void NeutralZoneCtrl(int32\_t \*var)

684. //{

685. // int32\_t prIQ = 0;

686. //

687. // if (!GrC->neutralZoneCtrl)

688. // return;

689. //

690. // prIQ = GrC->neutralZoneCtrl \* From1000toQ24;

691. //

692. // if ((\*var < prIQ)&&(\*var > -prIQ))

693. // \*var = 0;

694. //}

695.

Листинг A2 — Файл motorHiCtrl.h

1. //#ifndef \_\_motor\_hi\_ctrl

2. //#define \_\_motor\_hi\_ctrl

3. //#ifdef \_\_cplusplus

4. // extern "C" {

5. //#endif

6.

7. #include "stm32f4xx\_hal.h"

8. #include "config.h"

9. #include "IQmath.h"

10. #include "pid\_reg.h"

11. #include "rmp\_cntl.h"

12. #include "peref\_Filter1.h"

13.

14. #define MOTOR\_ZP 4 // Количество ПАР полюсов

15. #define SPD\_CALC\_FREQ 2000 // Частота расчета скорости

16.

17. typedef struct {

18. int32\_t m1;

19. int32\_t m2;

20. int32\_t StepAngleMax;

21. int32\_t StepAngleBase;

22. int32\_t Freq;

23. int32\_t Ramp;

24. int32\_t RampOut;

25. int32\_t Teta;

26. int32\_t Angle;

27. int32\_t Qs;

28. int32\_t Ds;

29. int32\_t Alpha;

30. int32\_t Beta;

31. int32\_t Reference;

32. int16\_t SetFreq;

33. int16\_t SetVolt;

34. uint8\_t Rez1;

35. uint8\_t Rez2;

36. int32\_t I1; // ток первого датчика (фаза А)

37. int32\_t I2; // ток второго датчика (фаза B)

38. int32\_t IAlpha;

39. int32\_t IBeta;

40. int32\_t IDs;

41. int32\_t IQs; //0.1 о.е. IQ24 -> 1 А

42. uint8\_t PhaseSelect;

43.

44. int32\_t TestTeta;

45.

46. uint8\_t atanTest;

47. int32\_t atanInTetaSet;

48. int32\_t atanAlpha;

49. int32\_t atanBeta;

50. int16\_t atanAlphaSet;

51. int16\_t atanBetaSet;

52.

53. int32\_t MotMechTeta;

54. int32\_t MotMechTeta2;

55. int32\_t MotElecTeta;

56.

57. int16\_t ScaleTimer;

58. int16\_t ScaleTimeOut;

59. uint16\_t StartResetTimer;

60. uint16\_t StartResetDelay;

61.

62. int16\_t ScalePosTimer;

63. int16\_t ScalePosTimeOut;

64.

65. RMPCNTL rc1;

66. TFilter1 filter1\_REF;

67.

68. PIDREG CurrId;

69. PIDREG CurrIq;

70. PIDREG Spd;

71. PIDREG Pos;

72.

73. int32\_t PosError;

74. int32\_t PosregKp;

75. int32\_t limPos;

76. int32\_t PosregOutTmp;

77. int32\_t PosregOutMax;

78. int32\_t PosregOutMin;

79. int32\_t ResetMechTeta;

80.

81. int32\_t tetaCorr;

82. } MTR\_Ctrl;

83.

84. void MTR\_Init(MTR\_Ctrl \*);

85. void MTR\_HiUpdt(MTR\_Ctrl \*);

86. void MTR\_LoUpdt(MTR\_Ctrl \*);

87.

88. void rampgen\_calc(MTR\_Ctrl \*);

89. void ipark\_calc(MTR\_Ctrl \*);

90. void park\_calc(MTR\_Ctrl \*);

91.

92. //void NeutralZoneCtrl(int32\_t \*);

93.

94. extern MTR\_Ctrl Mot;

95.

96.

97. //#ifdef \_\_cplusplus

98. //}

99. //#endif

100. //#endif

101.

Приложение Б

Функция расчёта скорости перемещения рулевой рейки

Листинг Б1 — Код файла speed\_fr.c

1. #include "speed\_fr.h"

2. #include "peref.h"

3. #include "g\_Ram.h"

4. #include "config.h"

5. #include "tim.h"

6.

7. SpeedFR Spd;

8. TmechCalcInc mechCalcInc;

9.

10. void speed\_fr\_init(SpeedFR \*v)

11. {

12. //LgInt Tmp, Ksens;

13. //Uns SyncSpeed = 750; // синхронная скорость

14.

15. // Ksens = (LgInt)SyncSpeed \* (LgInt)PosSens->SensPrec;

16. // v->Mash = \_IQ17div(60L/4 \* SPD\_CALC\_FREQ, Ksens); // = 16

17.

18. // v->DeltaHi = 2097152; // 1500 оборотов

19. v->DeltaHi = 2796203; // максималка в 2000 об/мин

20. // v->Mash = 671088640; // 40.0 в формате Q24 40.0 \* 0.025 = 1.0 (при )

21. // v->Mash = 67108864; //

22.

23. // v->DeltaLo = (LgInt)PosSens->LowQEPLevel;

24. // v->PassCount = PosSens->LowQEPPassCount;

25. v->IgnorFlag = 0;

26.

27. mechCalcInc.delta\_mech = \_IQ24mpy(Q24\_one, DELTA\_MECH\_INC);

28.

29. v->num\_rev = 0;

30. }

31. //---------------------------------------------------------------

32. void speed\_fr\_calc(SpeedFR \*v) // 200 Гц

33. {

34. //LgInt Delta;

35.

36. //v->htim2\_CNT = htim2.Instance->CNT;

37. //v->MechTheta = v->htim2\_CNT \* Spd.gain\_incr;

38. v->MechTheta = GrA->pilaA\_Q24;

39. v->Delta = v->MechTheta - v->MechThetaPrev;

40. v->Delta\_clean = v->Delta;

41.

42. if (v->Delta\_clean > Q24\_half) // скачёк на половину значения пилы более чем +0.5 , предполагаем что это был переход пилы ИЗ 0 В 1, следовательно пила сейчас ОТРИЦАТЕЛЬНАЯ

43. {

44. v->Delta\_clean = v->Delta\_clean - Q24\_one; // вычитаем -1 для получения истинного приращения пилы на участке перехода через 0

45. v->num\_rev--;

46. }

47. else if (v->Delta\_clean < -Q24\_half) // теперь скачёк на половину значения пилы меньше чем -0.5 , предполагаем что был переход пилы ИЗ 1 В 0, следовательно пила сейчас ПОЛОЖИТЕЛЬНАЯ

48. {

49. v->Delta\_clean = v->Delta\_clean + Q24\_one; // прибавляем +1 для получения истинного приращения пилы на участке перехода через 0

50. v->num\_rev++;

51. }

52.

53. //GrA->PosInc = v->htim2\_CNT | (v->num\_rev << 12);

54.

55. if (!v->IgnorFlag) // если на предыдущем шаге не было подстановки Delta из памяти, то проверяем её на шумы

56. {

57. if ((v->Delta\_clean > v->DeltaHi)||(v->Delta\_clean < -v->DeltaHi)) // если образовался скачёк пилы по величине меньше чем +-0,5, но при этом Delta всё равно оказалась больше,

58. { // чем при максимально-возможной скорости вала в 2000 об/мин => 33,333 об/сек

59. v->Delta\_clean = v->DeltaPrev; // то будем игнорировать это значение Delta и прирваниваем "старую" дельту с прошлого шага

60. v->IgnorFlag = 1; // такая подставновка возможна тольок на один шаг, на следующем шаге в любом случае будем приравнивать Delta "как есть".

61. }

62. }

63. else v->IgnorFlag = 0; // Delta без коррекции, на следующем шаге опять возможно приравнивание "старой" Delta

64.

65. //v->Speed = Delta \* 16; // расчёт скорости Q24

66. v->Speed = (\_IQ24mpy(v->Delta\_clean, v->mash)) << 3; // расчёт скорости Q24

67. v->MechThetaPrev = v->MechTheta; // "память" о "пиле" на предыдущем шаге

68. v->DeltaPrev = v->Delta\_clean; // "память" про дельту на предыдущем шаге

69. //v->Speed = \_IQ24mpy(fix16\_div(v->Speed, 100), 90);

70. }

71.

72. //---------------------------------------------------------------

73.

Листинг Б2 — Код файла speed\_fr.h

1. #ifndef SPEED\_FR\_

2. #define SPEED\_FR\_

3.

4. #include "std.h"

5. #include "IQmath.h"

6.

7. #ifdef \_\_cplusplus

8. extern "C" {

9. #endif

10.

11. #define FREQ\_TIM\_2 36000000 //частота таймер 2, режим CAP

12. #define FILTR\_SPEED (Prd200HZ >> 1) //0.5 c

13. #define DELTA\_MECH\_INC 48210 // 1/(14.5 \* 24) в Q24

14. //14.5 - редуктор, 24 - количество меток (12\*2)

15.

16. typedef struct {

17. LgInt MechTheta;

18. LgInt MechThetaPrev;

19. LgInt DeltaLo;

20. LgInt DeltaHi;

21. LgInt Delta;

22. LgInt Delta\_clean;

23. LgInt DeltaPrev;

24. Uns PassIndex;

25. Uns PassCount;

26. // LgInt Mash;

27. LgInt Speed;

28. Uns IgnorFlag;

29. LgInt mash;

30. LgInt num\_rev;

31. uint32\_t htim2\_CNT;

32. //Инкреметный энкодер (квадратурный сигнал) CAP+GPIO

33. int32\_t cap\_value; //захват таймера TIM2 в режиме CAP через CallBack

34. int32\_t cap\_value\_prev;

35. uint16\_t timer;

36. uint8\_t ENC\_CH\_A; //GPIO

37. uint8\_t ENC\_CH\_B; //Косвенное определение второго канала (потому что CAP)

38. uint8\_t sign;

39.

40. //Инкреметный энкодер (квадратурный сигнал) QEP

41. int32\_t qep;

42.

43. uint32\_t gain\_incr;

44.

45. } SpeedFR;

46.

47. typedef struct {

48. int32\_t mechAngle;

49. int32\_t delta\_mech;

50. int32\_t position;

51. } TmechCalcInc;

52.

53. void speed\_fr\_init(SpeedFR \*);

54. void speed\_fr\_calc(SpeedFR \*);

55.

56. extern SpeedFR Spd;

57. extern TmechCalcInc mechCalcInc;

58.

59. #ifdef \_\_cplusplus

60. }

61. #endif // extern "C"

62.

63. #endif

64.