Санкт-Петербургский национальный исследовательский университет ИТМО
Факультет систем управления и робототехники

Отчёт по лабораторной работе №4

по курсу «Дизайн вещей будущего»

Вариант 6.

Выполнил: Марухленко Д.С.
Группа: R3235 (ДВБ 1.2)
Преподаватель: Власов С. М.

Санкт-Петербург 2021г.

1 Цель работы

Моделирование управления двигателем постоянного тока при помощи ПИД регулятора, реализованного на микроконтроллере.

2 Материалы работы

2.1 Схема симуляции Proteus

В схему, данную в качестве образца, были внесены некоторые изменения:

- Физическая подтяжка пинов к питанию, считывающих сигналы с эндекодера, была заменена внутренними подтяжками, так как: 1) Это упрощает схему 2) При использовании внешних подтяжек микроконтроллер отказывался считывать показания.
- Исправлено неправильное положение транзисторов IRF4905, создававшее в схеме короткое замыкание между линиями 12V и GND.
- Между пинами микроконтроллера и затворами (базами) транзисторов добавлены токоограничивающие резисторы.
- В целях отладки схемы добавлен осциллограф, отображающий сигналы микроконтроллера и отображение напряжения в ключевых точках схемы.

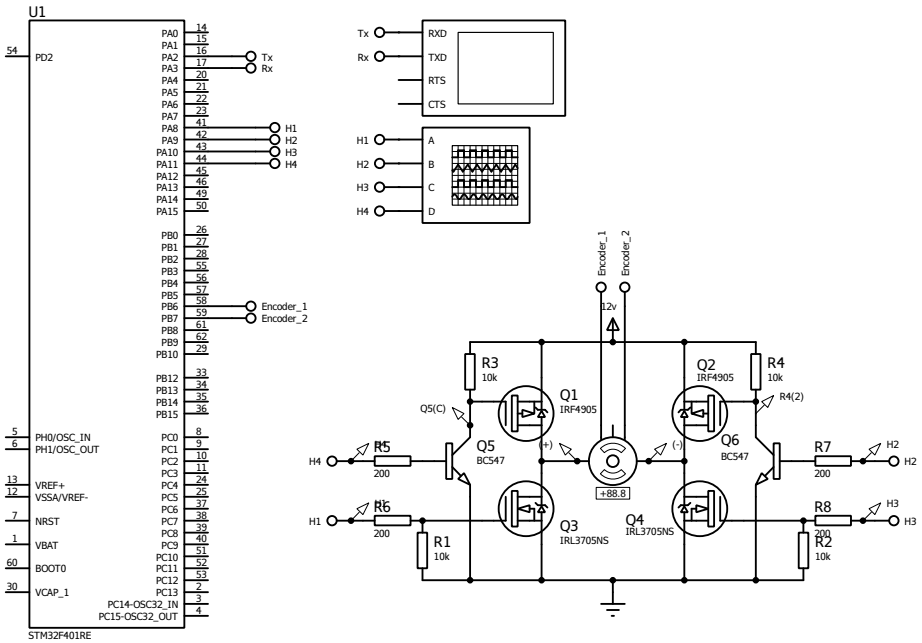

Рис. 1: Схема симуляции Proteus

2.2 Код программы управления двигателем

Для настройки микроконтроллера использовались средства автоматической генерации кода STM32CubeIDE. Далее представлен только код, написанный вручную.

Управление двигателем (проверка показаний и изменение управляющих сигналов) осуществляется с частотой 100 Гц, чего хватает для корректной регулировки, учитывая шаг срабатывания энкодера «1 тик на 2 градуса». В цикле осуществляется вычисление пропорциональной, интегральной и дифференциальной составляющих управляющего сигнала, их суммирование, ограничение по максимальному значению вывод отладочной информации через UART и непосредственно подача сигналов на управляющие контакты. Одна итерация цикла длится менее 10мкс., поэтому в случае необходимости частоту можно увеличить, не изменяя алгоритм. Управление подразумевает включение режима электромагнитного тормоза при нулевом управляющем сигнале.

```
/* USER CODE BEGIN PD */
#define K_P 1.2
#define K_I 0.0
#define K_D 0.2

#define FREQUENCY       100
#define TARGET_POSITION 1024
/* USER CODE END PD */

/* USER CODE BEGIN 0 */
void serial_print(int theta,int u,int error){
        uint8_t DATA[29];
        sprintf(DATA, "%6d %6d %6d %6d\n\r",HAL_GetTick(), theta,u,error);
        HAL_UART_Transmit(&huart2, DATA, 29, 100);
}

void rotare( int pwm){
        if(pwm > 0){
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, 0); //H1
                __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 0);//H2
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, 1); //H3
                __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, pwm); //H4;
        }else if (pwm < 0){
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, 0); //H3
                __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, 0);//H4
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, 1); //H1
                __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, -pwm);//H2
        } else {
                __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, 0);//H4
                __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 0);//H2
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, 1); //H1
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, 1); //H3
        }
}
/* USER CODE END 0 */

/* USER CODE BEGIN 2 */
HAL_TIM_Encoder_Start(&htim4, TIM_CHANNEL_ALL);          // start encoder
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);                // start pwm
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_4);                // start pwm
HAL_TIM_Base_Start(&htim2);                              // start timer 2

float u_p = 0, u_i = 0, u_d = 0;
```

```
int error = TARGET_POSITION, last_error = TARGET_POSITION;
int sum_u = 0, theta = 0;
int period = 1000000/FREQUENCY;
float period_s = 1/FREQUENCY;
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
TIM4->CNT = 10000;
TIM2->CNT= 0;
while(HAL_GetTick() < 660 ){}
while (1){
        /* USER CODE END WHILE */
        /* USER CODE BEGIN 3 */
        theta = 2*(TIM4->CNT-10000);
        error = (TARGET_POSITION - theta);
        u_p = error * K_P;
        u_i = u_i + (error * (K_I * period_s));
        u_d = ((error - last_error) * FREQUENCY) * K_D;
        sum_u = (u_p + u_d + u_i);

        if (sum_u > 100) sum_u = 100;
        else if (sum_u < -100) sum_u = -100;

        rotare(sum_u);
        serial_print(theta, sum_u,error);
        while(TIM2->CNT < period ){}
        last_error = error;
        TIM2->CNT= 0;
}
/* USER CODE END 3 */
}
```

2.3 Настройка PID-регулятора

Настройка регулятора производилась двумя способами:

- Методом Зингера - Никольса ($K_p = 1.32, K_i = 0.4, K_d = 0.1$).
- Методом подбора ($K_p = 1.2, K_i = 0, K_d = 0.2$).

В первом случае присутствует перерегулирование 25% и время переходного процесса составляет 1 c.. Во втором случае перерегулирование менее 2%, время переходного процесса около 0.5 c. Дальнейшие графики представлены для коэффициентов, подобранных вручную.

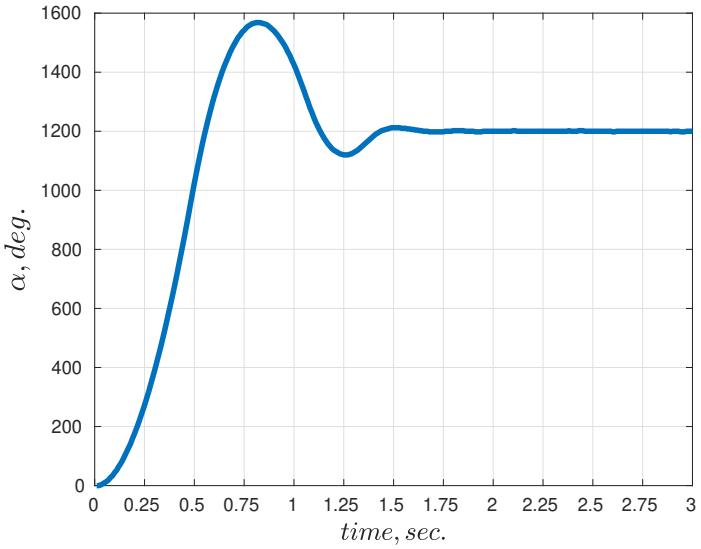

Рис. 2: График переходного процесса. Целевой угол 1200°. Коэффициенты вычислены методом Зиглера - Никольса

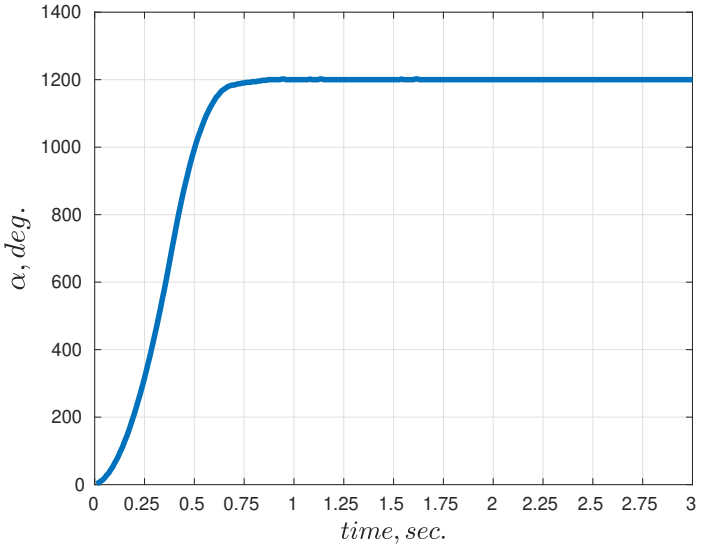

Рис. 3: График переходного процесса. Целевой угол 1200°. Коэффициенты подобраны.

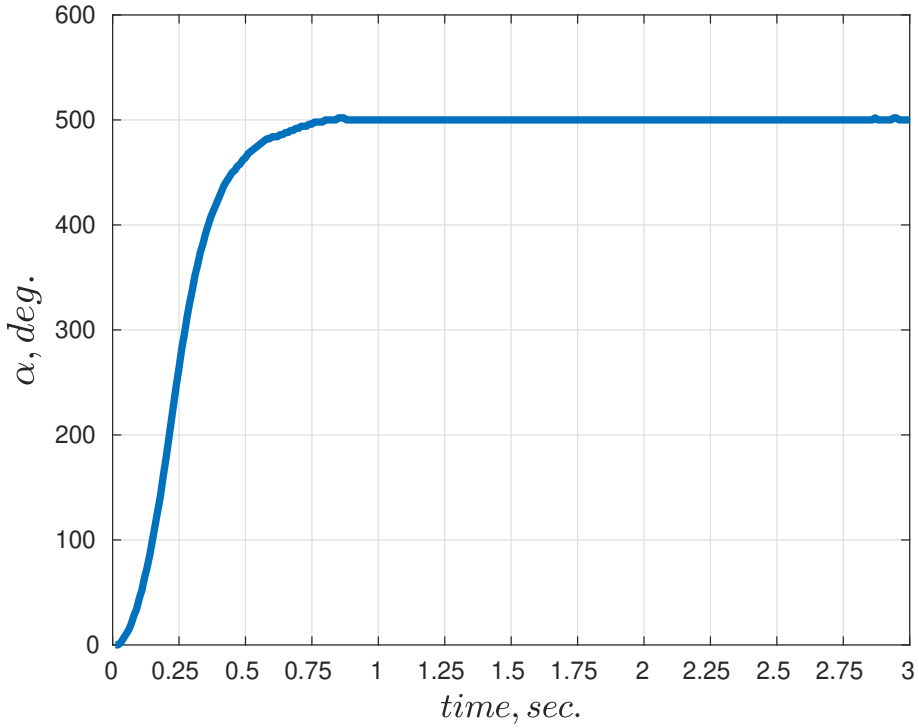

Рис. 4: График переходного процесса. Целевой угол 500°

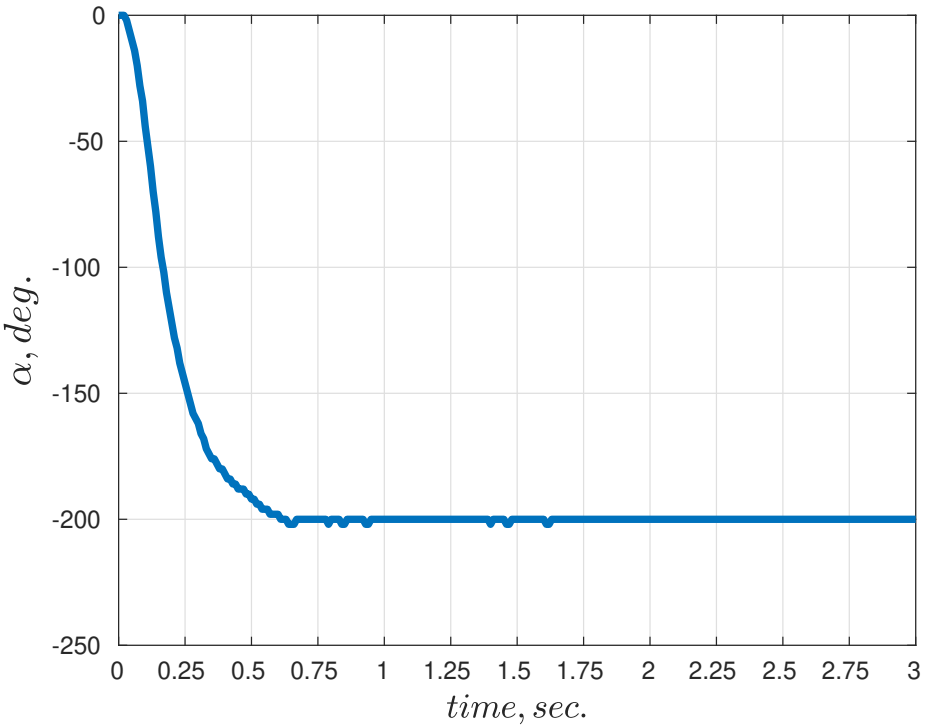

Рис. 5: График переходного процесса. Целевой угол $-200°$

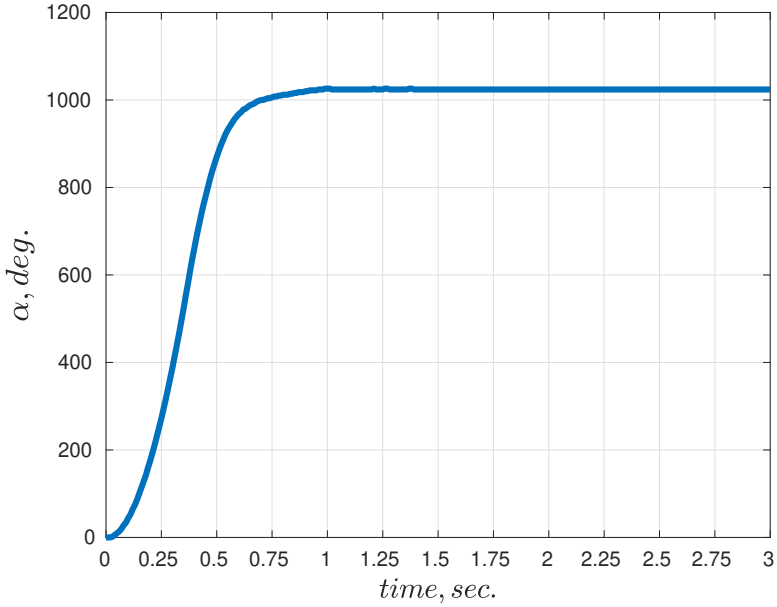

Рис. 6: График переходного процесса. Целевой угол 1024°

2.4 Ссылка на GitHub репозиторий

https://github.com/japersik/Design_of_Future_Things/

3 Вывод

В ходе выполнения работы я усовершенствовал навыки программирования микроконтроллера STM32 и сделал рабочий PID - регулятор для управления углом поворота двигателя.