МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт искусственного интеллекта (ИИИ)
Кафедра проблем управления

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №1
по дисциплине
«Программное обеспечение мехатронных и робототехнических систем»

Тема: Отладка программного обеспечения роботехнических систем с использованием виртуального моделирования

Выполнил студент группы КВБО-06-22 — Гончаров А.С., Древаль В.А., Пилия Т.С.

Принял преподаватель кафедры ПУ — Морозов А.А.

Лабораторная работа выполнена «__»_____2025 г.

«Зачтено» «__»_____2025 г.

Москва
2025

Цель работы: получение навыков моделирования объекта управления в промышленных системах автоматического управления и создание функциональных блоков.

Задание: создать виртуальную систему управления (рис. 1.1), включающую: модель объекта управления (рис. 1.2), ПИ-регулятор (рис. 1.3), сумматор и обратную связь. Передаточная функция объекта:

$$W = \frac{1}{k_e(T_M s+1)}$$

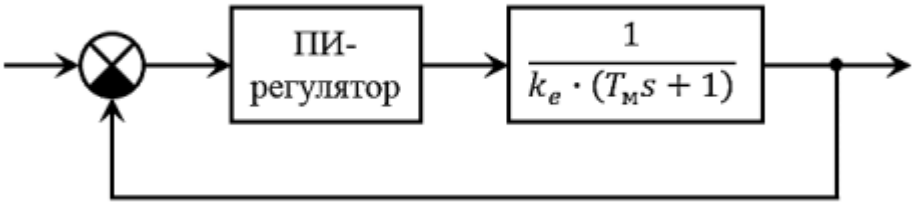


Рисунок 1. Структура системы управления.

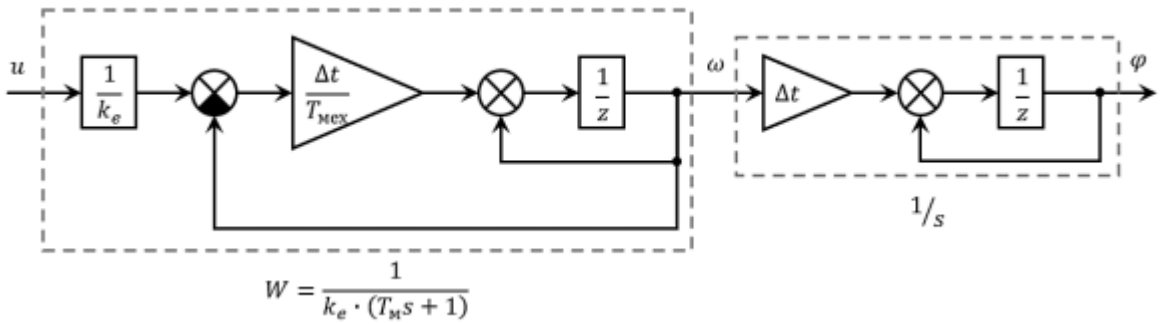


Рисунок 2. Структура объекта управления.

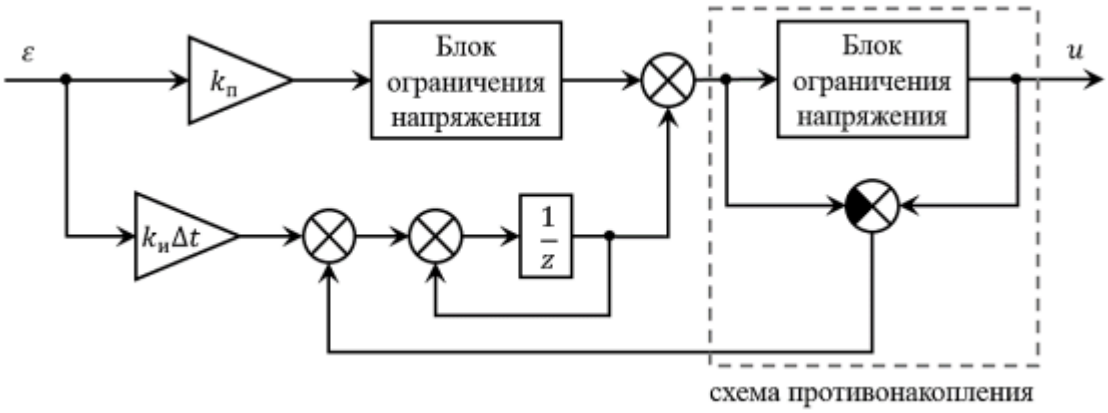


Рисунок 3. Структура ПИ-регулятора.

Практическая часть

Для выполнения лабораторной работы с использованием Automation Studio был создан новый проект. После создания проекта был начат процесс создания конфигурации оборудования. На данном этапе был произведен выбор промышленного компьютера, номер которого указан на обратной стороне панели над штрих-кодом.

Поскольку лабораторная работа выполняется на реальном оборудовании было установлено онлайн-соединение с контроллером.

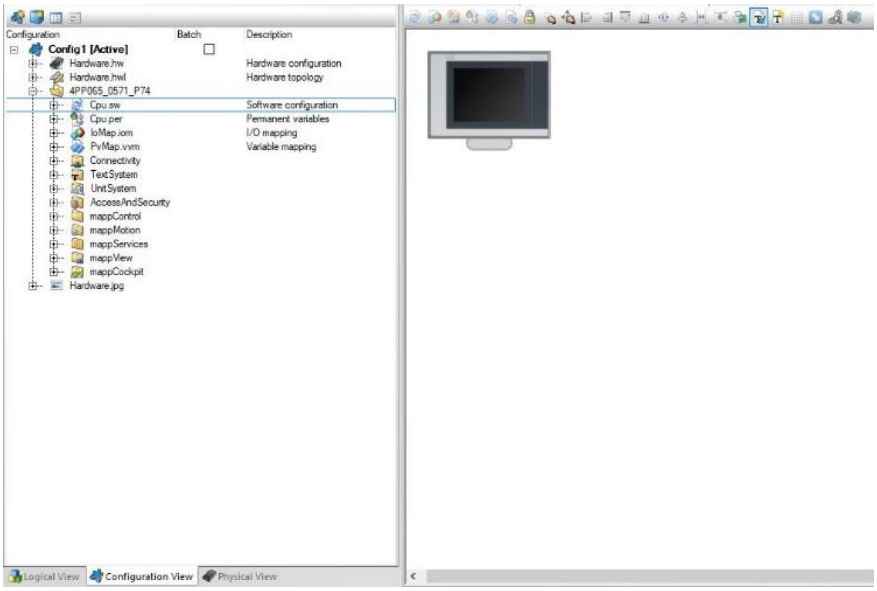


Рисунок 4. Конфигурация оборудования проекта.

Создание программного обеспечения проекта было произведено во вкладке Logical View путем добавления к проекту объекта Program. Для выполнения лабораторной работы в проект был добавлен объекты ANSI C Program All In One и ANSI C Library «MotorControl». Структурно программа представляет собой три функциональных модуля:

• void Init() – функция, код которой выполняется единожды при запуске проекта

• void Cyclic() – функция, код которой выполняется в процессе работы стенда циклически с указанной пользователем частотой

• void Exit() – функция, код которой выполняется единожды при завершении программы

Для выполнения лабораторной работы был назначен класс задачи 10 мс.

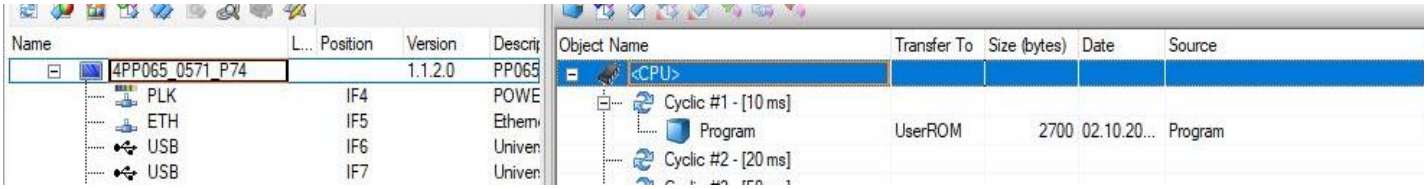


Рисунок 5. Выбор класса задач для выполнения программного кода

Далее в добавленной ранее библиотеке были созданы 3 функциональных блока и даны им следующие имена:

Рисунок 6. Созданные функциональные блоки.

Данные блоки реализуют следующий функционал:

Таблица 1. Назначение функциональных блоков.

Название	Назначение
FB_Motor	Модель двигателя постоянного тока
FB_Regulator	Модель ПИ-регулятора
FB_Integrator	Модель интегрирующего звена

Поскольку для создания функциональных блоков мотора и регулятора необходим интегратор, изначально был разработан он.

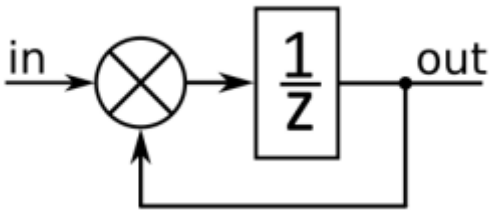


Рисунок 7. Структурная схема интегратора.

Логика работы интегратора заключается в накоплении сумм разностей входного и выходного значений в соответствии с шагом расчета. Это было реализовано в программном коде данного функционального блока.

Таблица 2. Параметры функционального блока FB_Integrator.

Конфигурация	Имя	Тип данных	Описание
Вход	in	REAL	вход интегрирующего звена
Выход	out	REAL	выход интегрирующего звена
Внутреннее состояние	dt	REAL	шаг расчета [с]

В структуру функционального блока «FB_Motor» были введены следующие поля, описанные в таблице 3.

Таблица 3. Параметры функционального блока FB_Motor.

Конфигурация	Имя	Тип данных	Описание
Вход	u	REAL	входное напряжение [В]
Выход	w	REAL	частота вращения [об/мин]
Выход	phi	REAL	положение [рад]
Внутреннее состояние	integrator	FB_Integrator	интегратор
Внутреннее состояние	Tm	REAL	электромеханическая постоянная времени [с]
Внутреннее состояние	ke	REAL	постоянная ЭДС двигателя [В•мин/об]
Внутреннее состояние	dt	REAL	шаг расчета [с]

Был разработан программный код функционального блока «FB_Motor», использующий приведенные выше поля в соответствии с их описанием. Расчет значения на выходе блока происходит в соответствии со схемой ДПТ (рис. 2) или путем вывода из передаточной функции объекта разностного уравнения методом Z-преобразования.

В функциональном блоке «FB_Regulator» были внесены следующие поля, которые описаны в таблице 4.

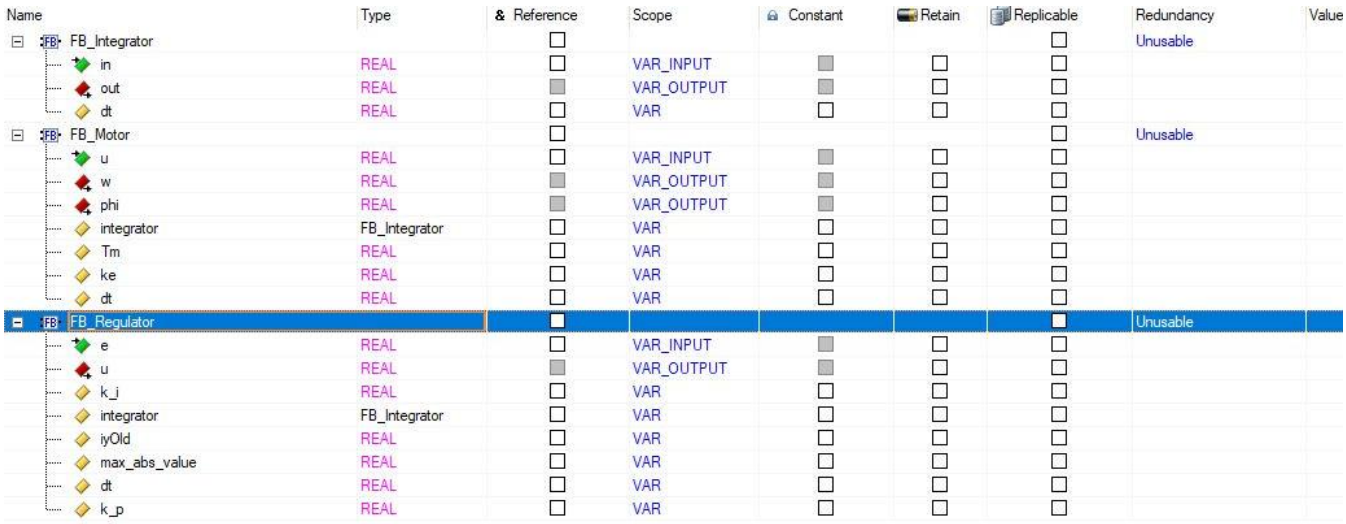
Таблица 4. Параметры функционального блока FB_Regulator.

Конфигурация	Имя	Тип данных	Описание
Вход	e	REAL	рассогласование между задающим воздействием и реальной скоростью вращения вала ДПТ [об/мин]
Выход	u	REAL	напряжение, подаваемое на вход ДПТ [В]
Внутреннее состояние	k_p	REAL	пропорциональный коэффициент регулятора
Внутреннее состояние	k_i	REAL	интегральный коэффициент регулятора
Внутреннее состояние	integrator	FB_Integrator	интегратор
Внутреннее состояние	iyOld	REAL	хранение предыдущего значения схемы противонакопления
Внутреннее состояние	max_abs_value	REAL	граница блока ограничения [В]
Внутреннее состояние	dt	REAL	шаг расчета [с]

Был написан программный код, использующий приведенные выше поля в соответствии с их описанием. Расчет значения на выходе блока происходит в соответствии со схемой ПИ-регулятора (рис.3). Схема противонакопления используется для ограничения напряжения, поступающего на двигатель, в целях обеспечения стабильности его работы.

Также в схеме присутствуют два ограничителя напряжения: после П-звена и выходного напряжения. Величина ограничения задается с помощью внутреннего состояния Umax.

Во все вышеописанные функциональные блоки в программе были внесены соответствующие переменные:



Name	Type	Reference	Scope	Constant	Retain	Replicable	Redundancy	Value
FB_Integrator		☐				☐	Unusable	
in	REAL	☐	VAR_INPUT	☐	☐	☐		
out	REAL	☐	VAR_OUTPUT	☐	☐	☐		
dt	REAL	☐	VAR	☐	☐	☐		
FB_Motor		☐				☐	Unusable	
u	REAL	☐	VAR_INPUT	☐	☐	☐		
w	REAL	☐	VAR_OUTPUT	☐	☐	☐		
phi	REAL	☐	VAR_OUTPUT	☐	☐	☐		
integrator	FB_Integrator	☐	VAR	☐	☐	☐		
Tm	REAL	☐	VAR	☐	☐	☐		
ke	REAL	☐	VAR	☐	☐	☐		
dt	REAL	☐	VAR	☐	☐	☐		
FB_Regulator		☐				☐	Unusable	
e	REAL	☐	VAR_INPUT	☐	☐	☐		
u	REAL	☐	VAR_OUTPUT	☐	☐	☐		
k_i	REAL	☐	VAR	☐	☐	☐		
integrator	FB_Integrator	☐	VAR	☐	☐	☐		
iyOld	REAL	☐	VAR	☐	☐	☐		
max_abs_value	REAL	☐	VAR	☐	☐	☐		
dt	REAL	☐	VAR	☐	☐	☐		
k_p	REAL	☐	VAR	☐	☐	☐		

Рисунок 8. Переменные функциональных блоков.

В переменных(Variables) основной программы Main были созданы следующие поля:

Таблица 5. Переменные основной программы.

Имя	Тип данных	Описание
fb_controller	FB_Controller	рассогласование между задающим воздействием и реальной скоростью вращения вала ДПТ [об/мин]
fb_motor	FB_Motor	напряжение, подаваемое на вход ДПТ [В]
Speed	REAL	уставка по скорости
Enable	BOOL	интегральный коэффициент регулятора

Рисунок 9. Переменные основной программы.

В основной программе, в части инициализации «Init», были заполнены все постоянные (коэффициенты регуляторов, постоянные времени, граничные значения и шаги расчета) созданных объектов (fb_controller и fb_motor).

В основном цикле программы была реализована система управления (рис.1), имеющая на вход значение переменной speed и активирующаяся при наличии логической «1» в переменной enable.

Второй мотор был добавлен в переменные основной программы, в полях инициализации данные были указаны данные, аналогичные уже созданному ранее мотору. Также было добавлено исполнение функционального блока второго мотора в основной цикл программы, подавая на его вход уставку speed.

Было реализовано, используя вспомогательную переменную-счетчик counter ступенчатое воздействие на вход системы управления и второго двигателя.

С помощью средства Trace были сняты графики входного воздействия (скорость вращения), двигателя без регулятора, двигателя с регулятором.

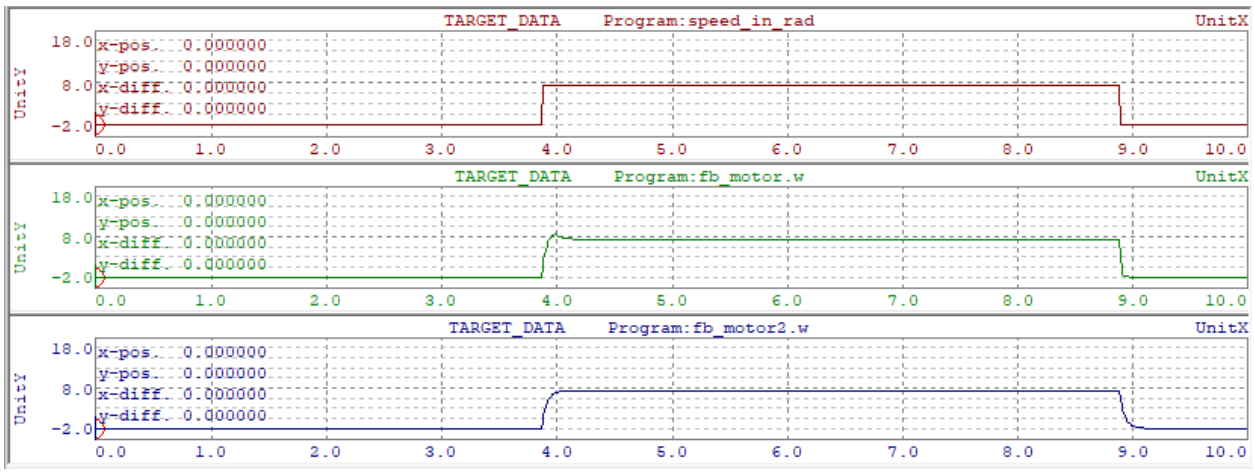


Рисунок 10. График уставки, скорости вращения двигателя без регулятора и скорости вращения двигателя с настроенным регулятором

Вывод

В результате выполнения лабораторной работы были получены навыки моделирования объекта управления в промышленных системах автоматического управления и создание функциональных блоков.

Приложение

```
#include <bur/plctypes.h>
#ifdef __cplusplus
 extern "C"
 {
#endif
 #include "Library.h"
#ifdef __cplusplus
 };
#endif
/* REGULATOR */
void FB_Regulator(struct FB_Regulator* inst)
{
    REAL p_part = inst->e * inst->k_p;

    REAL i_part = inst->e * inst->k_i;

    inst->integrator.in = i_part;

    FB_Integrator(&inst->integrator);

    i_part = inst->integrator.out;

    REAL total_output = p_part + i_part;

    if (total_output > inst->max_abs_value)
    {
        inst->u = inst->max_abs_value;
        inst->integrator.out = inst->max_abs_value - p_part;

    } else if (total_output < -inst->max_abs_value)
    {
        inst->u = -inst->max_abs_value;
        inst->integrator.out = -inst->max_abs_value - p_part;
    } else
    {
        inst->u = total_output;

    }
    inst->iyOld = i_part;
}
```

Листинг 1. Регулятор

```
#include <bur/plctypes.h>
#ifdef __cplusplus
 extern "C"
 {
#endif
 #include "Library.h"
#ifdef __cplusplus
 };
#endif
/* INTEGRATOR */
void FB_Integrator(struct FB_Integrator* inst)
{
    inst->dt= 0.01;

    inst->out = inst->dt * inst->in + inst->state;
    inst->state = inst->out;
}
```

Листинг 2. Интегратор

```
#include <bur/plctypes.h>
#ifdef __cplusplus
 extern "C"
 {
#endif
 #include "Library.h"
#ifdef __cplusplus
 };
#endif
/* MOTOR */
void FB_Motor(struct FB_Motor* inst)
{
    REAL b1 = inst->u * (1/inst->ke) - inst->w;
    REAL b2 = b1/ inst->Tm;

    inst->motor_integrator.in = b2;
    FB_Integrator(&inst->motor_integrator);

    inst->w = inst->motor_integrator.out;

    inst->sensor_integrator.in = inst->w;
    FB_Integrator(&inst->sensor_integrator);

    inst->phi = inst->sensor_integrator.out ;


}
```

Листинг 3. Мотор

```
#include <bur/plctypes.h>

#ifdef _DEFAULT_INCLUDES
     #include <AsDefault.h>
#endif

void _INIT ProgramInit(void)
{
    enable = 1;
    counter = 0;
```

```
    speed = 0;
     speed_in_rad = 0;

    fb_motor.dt=0.01;
    fb_motor.ke=2.65;
    fb_motor.Tm=0.05;

    fb_motor2.dt = 0.01;
    fb_motor2.ke = 2.65;
    fb_motor2.Tm = 0.05;

    fb_controller.dt=0.01;
    fb_controller.k_p = (fb_motor.ke* fb_motor.Tm) / 0.02;
    fb_controller.k_i = fb_motor.ke / 0.02;
    fb_controller.max_abs_value=24;
}

void _CYCLIC ProgramCyclic(void)
{
    REAL PI = 3.141592653589793;
    REAL desired_w = speed * 2.0 * PI / 60.0;
     speed_in_rad = desired_w;

    if (enable == 1){
        counter += 1;
        if ((counter >= 500 ) && (counter <= 1000)) speed = 70;
        else speed = 0;
    }
    else speed = 0;

    fb_controller.e = desired_w - fb_motor.w;
    FB_Regulator(&fb_controller);
     fb_motor.u = fb_controller.u;

    fb_motor2.u = desired_w * fb_motor2.ke;

    FB_Motor(&fb_motor);
    FB_Motor(&fb_motor2);

}

void _EXIT ProgramExit(void)
{

}
```

Листинг 4. Main