Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
 высшего образования   
«Национальный исследовательский университет «МЭИ»

Кафедра «Релейная защита и автоматизация энергосистем»

Лабораторная работа №3

«Дистанционная защита линии»

Выполнил: Максимов Р.С.

Группа: Э-13м-19

Проверил: Холодов А. С.

Москва 2020

**Задание на лабораторную работу №2**

Тип исполнения защиты: **фазная**

Тип характеристики: **многоугольная**

Блокировка от качаний: **по аварийной составляющей обратной последельности**

Количество ступеней: **три.**

Схема сети:

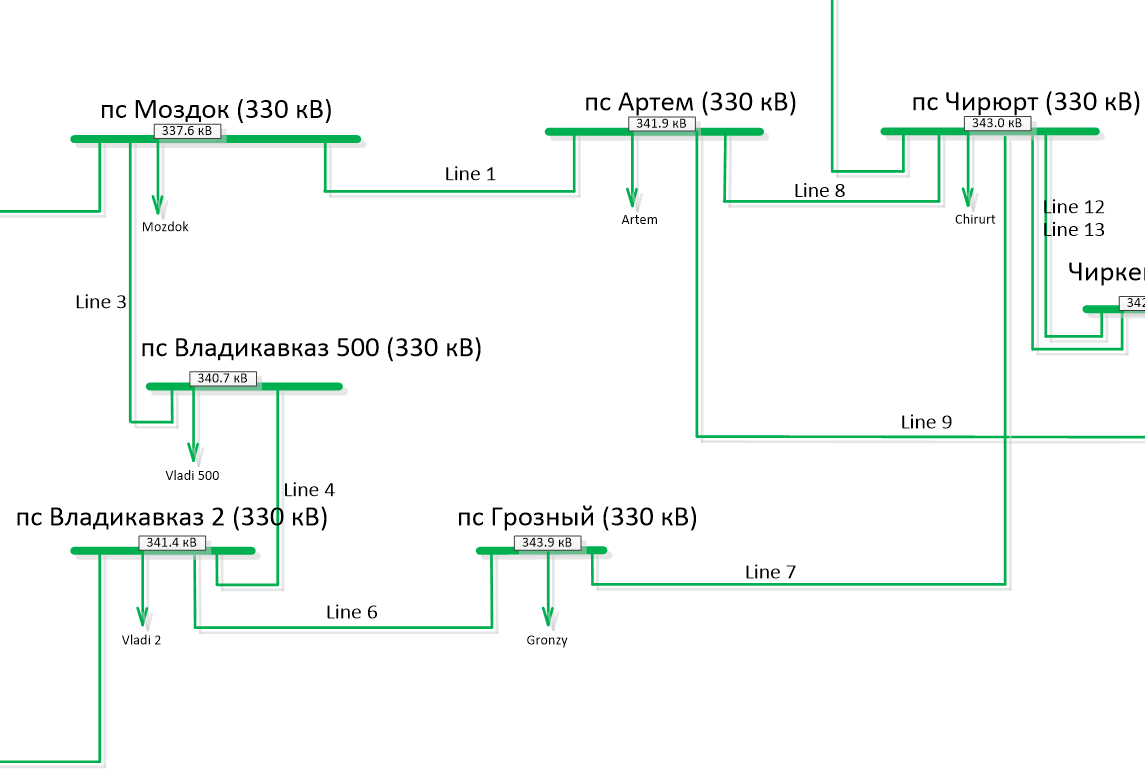


Рис. 1. Схема сети

Модель разработана и верифицирована в ПАК RTDS. Защищаемый объект – ВЛ между ПС 330 кВ Моздок и ПС 330 кВ Артем, нейтраль заземлена, параметры ВЛ: L = 273 км, r1 = 0.048 ом/км, x1 = 0.328 ом/км c1 = 0.293 Мом/км

**Теоретическая часть.**

ДЗ – это сложные направленные или ненаправленные защиты с относительной селективностью, выполненные с использованием минимальных реле сопротивления. Дистанционные защиты реагируют на сопротивление линии до места КЗ, которое пропорционально расстоянию, т. е. дистанции. Отсюда и происходит название ДЗ. Для работы ДЗ необходимо наличие цепей тока от ТТ присоединения и цепей напряжения от ТН. При отсутствии или неисправности цепей напряжения возможна ложная работа ДЗ, например, при КЗ на смежных участках. Селективное отключение КЗ в сложных кольцевых сетях может быть обеспечено с помощью ДЗ. Выдержка времени ДЗ t зависит от расстояния (дистанции), т. е. нарастает с увеличением этого расстояния. Дистанционная защита, расположенная ближе к месту повреждения, имеет меньшую выдержку времени, чем более удаленные ДЗ. Основным элементом ДЗ является дистанционный измерительный орган (ДО), который определяет удаленность точки КЗ от места установки РЗ. В качестве ДО используется реле сопротивления (PC), реагирующие на полное, реактивное или активное сопротивление поврежденного участка ЛЭП.

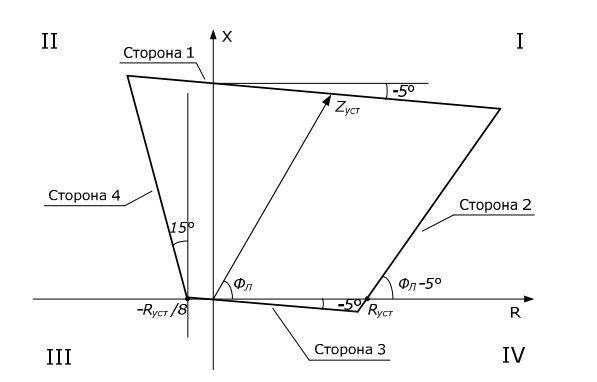


Рис. 2. Характеристика срабатывания для 1 и 2 ступени

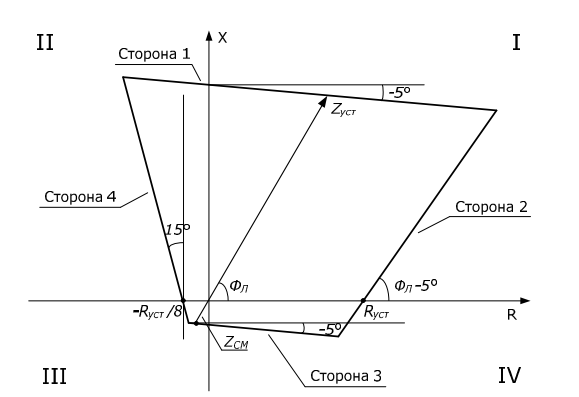


Рис. 3. Характеристика срабатывания для 3 ступени

Уставка первой ступени выбирается с выдержкой по времени 0 с. Сопротивление срабатывания определяется, исходя из условий отстройки от КЗ в начале смежного участка

Для расчета параметров срабатывания второй ступени необходимо отстраиваться от уставок предыдущих участков. Ввиду того, что нам не известны ни уставки предыдущих участков, ни их параметры, примем, что вторая ступень должна обеспечивать защиту всей линии (через коэф. чувствительности). Время срабатывания = 0,5с

Сопротивление срабатывания третьей ступени выбирается из условия обеспечения возврата РС в условиях самозапуска двигателей неотключенных потребителей послеотключения внешнего КЗ. Время срабатывания примем 1,5 с.

**Практическая часть.**

В классе Main запускаем работу программы, создаем объект класса InputData, на вход которого подаем имя файла comtrade (cfg+data) и количество значений аналоговых сигналов.

**public class** Main {  
 **public static void** main(String[] args) **throws** FileNotFoundException {  
  
 InputData inD1 = **new** InputData(**"Kz1"**, 2);  
 inD1.start();  
  
 }  
}

Класс inputData необходим для считывания данных с предоставляемых файлов и работы с ними. В данном случае, осуществляется считывание с файлов формата .cfg и .dat. В методе start() реализован алгоритм работы с файлами, а именно преобразование значений файла в мгновенные значения токов. В данном классе создаются объекты классов SampleValues, MeanValues, Fourie, Breakers в соответствии с заданным количеством аналоговых сигналов, а также объекты классов Logic, Vector, OutPutData, Resistance.

**import** java.io.\*;  
**import** java.lang.reflect.InvocationTargetException;  
**import** java.lang.reflect.Method;  
**import** java.util.ArrayList;  
**import** java.util.Arrays;  
**import** java.util.List;  
  
**public class** InputData {  
  
 **public** File **comtrCfg**, **comtrDat**;  
 **private** BufferedReader **br**;  
 **private** String **line**;  
 **private** String[] **lineData**;  
 **private double**[] **k1**;  
 **private double**[] **k2**;  
 **private** String **comtradeName**;  
 **private** String **nameFile**;  
 **private** SampleValues **sv** = **new** SampleValues();  
 **private** MeanValues **means** = **new** MeanValues();  
 **private** ArrayList<Fourie> **filter** = **new** ArrayList<Fourie>();  
 **private** Resistance **resistance** = **new** Resistance();  
 **private** OutputData **od** = **new** OutputData();  
 **private** Breaker **breaker** = **new** Breaker();  
 **private** Vectors **vectors** = **new** Vectors();  
 **private boolean t** = **false**;  
 **private static int** *i* = 0;  
 **private** Logic **logic** = **new** Logic();  
 **private double del** = 1.;**public** InputData(String nameFile) {  
 **this**.**nameFile** = nameFile;  
 }  
  
 **public void** start() **throws** FileNotFoundException, NoSuchMethodException {  
 **int** period = 80; *// millisec* **double** step = 0.00025;  
 **filter**.add(**new** Fourie(**"Voltage"**));  
 **filter**.add(**new** Fourie(**"Current"**));  
 **for** (Fourie value : **filter**) {  
*// value.set();* value.setSv(**sv**);  
 value.setVector(**vectors**);  
 value.setMeans(**means**);  
 value.setPeriod(period);  
 value.setRes(**resistance**);  
 }  
  
  
 **logic**.setResistance(**resistance**);  
 **logic**.setVector(**vectors**);  
 **logic**.setRes(**resistance**);  
 **od**.setLogic(**logic**);  
 **od**.setBreaker(**breaker**);  
  
 **try** {  
 *//путь к файлам comtrade* **comtradeName** = **nameFile**;  
 String path = **"D:\\education\\Algoritms\\Distance relay protection\\Опыты\\"**;  
 String cfgname = path + **comtradeName** + **".cfg"**;  
 String datName = path + **comtradeName** + **".dat"**;  
 **comtrCfg** = **new** File(cfgname);  
 **comtrDat** = **new** File(datName);  
 *//открываем cfg файл для получения коэф a и b для расчета y = ax+b* **br** = **new** BufferedReader(**new** FileReader(**comtrCfg**));  
 **int** lineNumber = 0, count = 0, numberData = 100;  
 **try** {  
 **while** ((**line** = **br**.readLine()) != **null**) {  
*// System.out.println(line);* lineNumber++;  
 **if** (lineNumber == 2) {  
 *//получаем количество аналоговых сигналов во 2 строке cfg файла "4,3A,1D"* numberData = Integer.*parseInt*(**line**.split(**","**)[1].replaceAll(**"A"**, **""**));  
 *//создаем double " массивы " с размерностью равной количеству* **k1** = **new double**[numberData];  
 **k2** = **new double**[numberData];  
 }  
 *//коэф находятся на 3,4,5 строке это 5 и 6 элемент строки при парсинге* **if** (lineNumber > 2 && lineNumber < numberData + 3) {  
 **k1**[count] = Double.*parseDouble*(**line**.split(**","**)[5]);  
 **k2**[count] = Double.*parseDouble*(**line**.split(**","**)[6]);  
 count++;  
 }  
 }  
 count = 0;  
 **br** = **new** BufferedReader(**new** FileReader(**comtrDat**));  
 *//получение методов из объекта sv* List<Method> methodsSV = Arrays.*asList*(**sv**.getClass().getDeclaredMethods());  
  
 **while** ((**line** = **br**.readLine()) != **null**) {  
 count++;  
 **if** ((count > 0 && count < 4500)) {  
 **lineData** = **line**.split(**","**);  
  
 **if** (Breaker.*isState*()) {  
 **sv**.setCurrentPhA(Double.*parseDouble*(**lineData**[5]) \* **k1**[3] + **k2**[3]);  
 **sv**.setCurrentPhB(Double.*parseDouble*(**lineData**[6]) \* **k1**[4] + **k2**[4]);  
 **sv**.setCurrentPhC(Double.*parseDouble*(**lineData**[7]) \* **k1**[5] + **k2**[5]);  
 } **else** {  
 *//имитация отключния выключателя* **del** = 1;  
 **sv**.setCurrentPhA(0);  
 **sv**.setCurrentPhB(0);  
 **sv**.setCurrentPhC(0);  
 }  
  
 **sv**.setVoltagePhA((Double.*parseDouble*(**lineData**[2]) \* **k1**[0] + **k2**[0])\***del**);  
 **sv**.setVoltagePhB((Double.*parseDouble*(**lineData**[3]) \* **k1**[1] + **k2**[1])\***del**);  
 **sv**.setVoltagePhC((Double.*parseDouble*(**lineData**[4]) \* **k1**[2] + **k2**[2])\***del**);  
  
 **for** (Fourie fourie : **filter**) {  
 **try** {  
 fourie.calculate();  
 } **catch** (ReflectiveOperationException e) {  
 e.printStackTrace();  
 }  
 }  
 **vectors**.startCalculateReverseSeq();  
 **logic**.protect();  
 **od**.controlState();  
  
  
 Charts.*addAnalogData*(0, 0, **sv**.getVoltagePhA());  
 Charts.*addAnalogData*(0, 1, **sv**.getVoltagePhB());  
 Charts.*addAnalogData*(0, 2, **sv**.getVoltagePhC());  
  
 Charts.*addAnalogData*(1, 0, **sv**.getCurrentPhA());  
 Charts.*addAnalogData*(1, 1, **sv**.getCurrentPhB());  
 Charts.*addAnalogData*(1, 2, **sv**.getCurrentPhC());  
  
 Charts.*addAnalogData*(2, 0, **means**.getVoltagePhA());  
 Charts.*addAnalogData*(2, 1, **means**.getVoltagePhB());  
 Charts.*addAnalogData*(2, 2, **means**.getVoltagePhC());  
  
 Charts.*addAnalogData*(3, 0, **means**.getCurrentPhA());  
 Charts.*addAnalogData*(3, 1, **means**.getCurrentPhB());  
 Charts.*addAnalogData*(3, 2, **means**.getCurrentPhC());  
  
 Charts.*addAnalogData*(4, 0, **vectors**.getDI2());  
 Charts.*addAnalogData*(5, 0, **resistance**.getResistanceAbs()[0]);  
 Charts.*addAnalogData*(5, 1, **resistance**.getResistanceAbs()[1]);  
 Charts.*addAnalogData*(5, 2, **resistance**.getResistanceAbs()[2]);  
  
 *//имитация снижения напряжения при КЗ* **if** (**means**.getCurrentPhA() > 0.4 || **means**.getCurrentPhB() > 0.4 || **means**.getCurrentPhC() > 0.4){  
 **double** maxValue1 = Math.*max*(**means**.getCurrentPhA(),**means**.getCurrentPhB());  
 **double** maxValue2 = Math.*max*(**means**.getCurrentPhB(),**means**.getCurrentPhC());  
 **double** maxValue = Math.*max*(maxValue2,maxValue1);  
 **del** = (0.5/maxValue);  
  
 } **else** {  
 **del** = 1;  
 }  
  
 }  
 }  
 } **catch** (ArrayIndexOutOfBoundsException e) {  
 e.printStackTrace();  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
  
 }  
  
}

Класс Fourie преобразует мгновенные значения сигналов в действующее, а также реализует разложение сигналов на ортогональные составляющие по 1 гармонике, последнее нужно для сигнала блокировки. В данной программе было реализовано 2 объекта класса Фурье, один для токов, второй для напряжений. С помощью рефлексии, enum и имени осуществляется перебор методов. Фурье токов отсылает значения токов в объект класса Vector, где рассчитывается аварийная составляющая обратной последовательности (DI2),и отсылается токи и напряжения в объект класса Resistance для рассчета сопротивлений.

**import** java.lang.reflect.InvocationTargetException;  
**import** java.lang.reflect.Method;  
**import** java.util.ArrayList;  
**import** java.util.Arrays;  
**import** java.util.List;  
  
**public class** Fourie {  
 **private double**[] **A0** = **new double**[3]; *//3 - 3 phases* **public double**[] **Ak1** = **new double**[3];  
 **public double**[] **Bk1** = **new double**[3];  
 **private** ArrayList<**double**[]> **buff** = **new** ArrayList<**double**[]>();  
 **private int count** = 0;  
 **private** SampleValues **sv**;  
 **private** MeanValues **means**;  
 **private** Vectors **vector**;  
 **private** String **number**;  
 **private static int** *period* = 80; *//количество точек за период* **private** Resistance **res**;  
  
 Fourie(String number) {  
 **this**.**number** = number;  
 **for** (Phases onePhasa : Phases.*values*()) {  
 **buff**.add(**new double**[80]); *// bufferPh current or voltage* }  
 }  
  
 **public void** calculate() **throws** ReflectiveOperationException {  
  
 **for** (Phases onePhasa : Phases.*values*()) { *//проходимся по enum* Method onePhasaMet = **sv**.getClass().getDeclaredMethod(**"get"** + **number** + onePhasa); *//получение метода из объекта sv* **int** i = onePhasa.ordinal();  
 **double** actualSV = (**double**) onePhasaMet.invoke(**sv**);  
 **double**[] actual\_buf = **buff**.get(i);  
 **double** sumPh = actualSV - actual\_buf[**count**];  
 *//Алгоритм фурье: постоянная составляющая + 1 гармоника  
 //расчет постоянной составляющей, возникает при КЗ. В норм. режиме равна нулю -> интеграл синусоиды = 0* **A0**[i] = **A0**[i] + sumPh / *period*;  
 *//расчет cos и sin составляющей для первой гармоники* **Ak1**[i] = **Ak1**[i] + 2 \* (Math.*cos*(**count** \* 2 \* Math.***PI*** / *period*) \* sumPh) / *period*;  
 **Bk1**[i] = **Bk1**[i] + 2 \* (Math.*sin*(**count** \* 2 \* Math.***PI*** / *period*) \* sumPh) / *period*;  
 *//расчет действующего начения для 1 гармоники по cos и sin составляющей* **double** Ck1 = Math.*sqrt*((Math.*pow*(**Ak1**[i], 2) + Math.*pow*(**Bk1**[i], 2)) / 2);  
 **double** x = Math.*sqrt*(Math.*pow*(Ck1, 2) + Math.*pow*(**A0**[i], 2));  
actual\_buf[**count**] = actualSV;  
 **buff**.set(i, actual\_buf);  
 *//достаем метод из mean* Method methodsMEAN = **means**.getClass().getMethod(**"set"** + **number** + onePhasa, **double**.**class**);  
 methodsMEAN.setAccessible(**true**);  
 methodsMEAN.invoke(**means**, x);  
  
 }  
  
 **if** (**number**.equals(**"Current"**)) {  
 **vector**.getVectors(**Ak1**, **Bk1**, **number**);  
 **res**.getVectorsCurrent(**Ak1**, **Bk1**);  
 } **else** {  
 **res**.getVectorsVoltage(**Ak1**, **Bk1**);  
 }  
  
 **count**++;  
 **if** (**count** == *period*) {  
 **count** = 0;  
 }  
 }  
  
  
 **public void** setSv(SampleValues sv) {  
 **this**.**sv** = sv;  
 }  
  
 **public void** setMeans(MeanValues means) {  
 **this**.**means** = means;  
 }  
  
 **public void** setVector(Vectors vector) {  
 **this**.**vector** = vector;  
 }  
  
 **public void** setPeriod(**int** period) {  
 **this**.*period* = period;  
 }  
  
 **public void** setRes(Resistance res) {  
 **this**.**res** = res;  
 }  
  
  
}

Объект класса Vector принимает значения ортогональных составляющих от объекта класса Fourie для того, чтобы рассчитывать аварийную составляющую обратной последовательности (DI2) и сформировать сигнал блокировки. Блокировка дистанционной защиты осуществляется в нормальном режиме.

**import** java.awt.\*;  
**import** java.util.ArrayList;  
  
**public class** Vectors {  
 **private double**[] **cosFirst** = **new double**[3]; *// 3 - фазы* **private double**[] **sinFirst** = **new double**[3];  
 **private** Complex[] **buffActualValueI2** = **new** Complex[80];  
 **private static** Complex *a* = **new** Complex(-0.5,0.866);  
 **private static** Complex *aa* = **new** Complex(-0.5,-0.866);  
 **private int count** = 0;  
 **private boolean flag** = **false**; *// флаг для разрешния расчета аварийной составляющей* **private double DI2**;  
 **private boolean blocking** = **true**;  
 **private double timeTemporary** = 0;  
 **private double actBlocking** = 2000;  
 **private boolean wait** = **false**;  
  
  
  
 **public void** getVectors(**double**[] x\_harmonic1, **double**[] y\_harmonic1, String numberFourie) {  
 **cosFirst** = x\_harmonic1; *//[A,B,C] - кос. составляющие по 1 гармонике* **sinFirst** = y\_harmonic1;  
 }  
  
 **public void** startCalculateReverseSeq(){  
  
 Complex Ia = **new** Complex(**cosFirst**[0],**sinFirst**[0]);  
 Complex Ib = **new** Complex(**cosFirst**[1],**sinFirst**[1]);  
 Complex Ic = **new** Complex(**cosFirst**[2],**sinFirst**[2]);  
 *//I2 = Ia + Ib \*a^2 +Ic\*a* Complex I2 = Ia.plus(Ib.product(*aa*)).plus(Ic.product(*a*)).divides(3);  
 Charts.*addAnalogData*(9, 0, boolToInt(**blocking**));  
 **if** (**flag**) {  
 Complex oldValue = **buffActualValueI2**[**count**];  
 **DI2** = I2.minus(oldValue).abs() ;  
 **if** ((**DI2** > 1. || (!**blocking**)) & !**wait**){  
 **actBlocking** = **actBlocking** - 0.00025;  
 **if** (**actBlocking** > 0.) {  
 **blocking** = **false**;  
 } **else** {  
 **blocking** = **true**;  
 **actBlocking** = 2000;  
 **wait** = **true**; *//выдержка времени после срабатывания ПО* }  
 }  
 **if** (**wait**) {  
 **timeTemporary** = **timeTemporary** + 0.00025;  
 **if** (**timeTemporary** > 1000) {  
 **timeTemporary** = 0.;  
 **wait** = **false**;  
 }  
 }  
  
 } **else** {  
 **DI2** = 0;  
 }  
  
 **buffActualValueI2**[**count**] = I2;  
 **count**++;  
 **if** (**count** == 80) {  
 **count** = 0;  
 **flag** = **true**;  
 }  
  
 }  
  
 **public double** getDI2() {  
 **return DI2**;  
 }  
  
 **public boolean** isBlocking() {  
 **return blocking**;  
 }  
  
 *// функция для преобразования boolean -> int* **public int** boolToInt(**boolean** b) {  
 **return** Boolean.*compare*(b, **false**);  
 }  
}

Объект класса Resistance принимает значения ортогональных составляющих от всех объектов класса Fourie и рассчитывает пофазно значения сопротивлений, что необходимо для объекта класса Logic,

**public class** Resistance {  
 **private double**[] **cosFirstCurrent** = **new double**[3]; *// 3 - фазы* **private double**[] **sinFirstCurrent** = **new double**[3];  
 **private double**[] **cosFirstVoltage** = **new double**[3]; *// 3 - фазы* **private double**[] **sinFirstVoltage** = **new double**[3];  
 **private** Complex[] **resistanceComplex** = **new** Complex[3]; *//a,b,c* **private double**[] **resistanceAbs** = **new double**[3]; *//a,b,c* **public void** getVectorsCurrent(**double**[] x\_harmonic1, **double**[] y\_harmonic1) {  
 **cosFirstCurrent** = x\_harmonic1; *//[A,B,C] - кос. составляющие по 1 гармонике* **sinFirstCurrent** = y\_harmonic1;  
 }  
  
 **public void** getVectorsVoltage(**double**[] x\_harmonic1, **double**[] y\_harmonic1) {  
 **cosFirstVoltage** = x\_harmonic1; *//[A,B,C] - кос. составляющие по 1 гармонике* **sinFirstVoltage** = y\_harmonic1;  
 }  
  
 **public void** startCalculateResistance(){  
  
 **for** (Phases one: Phases.*values*()){  
 **int** i = one.ordinal();  
 Complex I = **new** Complex(**cosFirstCurrent**[i],**sinFirstCurrent**[i]);  
*// System.out.println("current "+I.re()+' '+I.im());* Complex U = **new** Complex(**cosFirstVoltage**[i],**sinFirstVoltage**[i]);  
*// System.out.println("voltage "+U.re()+' '+U.im());* **resistanceComplex**[i] = U.divides(I);  
*// System.out.println("resistance "+resistanceComplex[i].re()+' '+resistanceComplex[i].im());* **resistanceAbs**[i] = **resistanceComplex**[i].abs();  
 }  
 ChartsXY.*addAnalogData*(0,0, **resistanceComplex**[0].re(),**resistanceComplex**[0].im());  
 ChartsXY.*addAnalogData*(0,1, **resistanceComplex**[1].re(),**resistanceComplex**[1].im());  
 ChartsXY.*addAnalogData*(0,2, **resistanceComplex**[2].re(),**resistanceComplex**[2].im());  
 }  
 **public** Complex[] getResistanceComplex() {  
 **return resistanceComplex**;  
 }  
  
 **public double**[] getResistanceAbs() {  
 **return resistanceAbs**;  
 }  
  
}

Объект класса Logic начинает выполнять свою функцию только если выполнены условия: включен выключатель и отсутствует блокировка защиты. Для каждой ступени проверяется фак попадания точки в их зоны. Зоны представлены на рисунке ниже, они были нарисованы с помощью static класса Graf.

**import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** Logic {  
  
  
 **private** Resistance **resistance**;  
 **private** Vectors **vector**;  
 *// y - kx < b -lower or y - kx > b - higher  
 //The first digit indicates the stage number  
 // 1 stage  
 // bottom line* **private static double** *k1* = -0.087; *//it is the same for all stages* **private static double** *b11* = 0; *//The first digit indicates the stage number  
 // the right line is the same for all stages* **private static double** *k2* = 4.22;  
 **private static double** *b12* = -55.28;  
 *// the left line is the same for all stages* **private static double** *k3* = -3.73;  
 **private static double** *b3* = -6.08;  
 *//top line* **private static double** *k4* = -0.087; *//it is the same for all stages* **private static double** *b14* = 76.9;  
 *// 2 stage* **private static double** *b21* = 0;  
 **private static double** *b22* = -67.01;  
 **private static double** *b24* = 108.59;  
 *// 3 stage* **private static double** *b31* = -2;  
 **private static double** *b32* = -152.1;  
 **private static double** *b34* = 246.46;  
 **private static double**[] *stageFirst* = {*k1*, *b11*, *k2*, *b12*, *k3*, *b3*, *k4*, *b14*};  
 **private static double**[] *stageSecond* = {*k1*, *b21*, *k2*, *b22*, *k3*, *b3*, *k4*, *b24*};  
 **private static double**[] *stageThird* = {*k1*, *b31*, *k2*, *b32*, *k3*, *b3*, *k4*, *b34*};  
 **private static double**[] *counter* = {0, 0, 0};  
 **private static boolean**[] *trip* = {**false**, **false**, **false**};  
 **private static boolean**[] *str* = {**false**, **false**, **false**};  
 **private static double** *timeForFirstStage* = 0.;  
 **private static double** *timeForSecondStage* = 0.5;  
 **private static double** *timeForThirdStage* = 1.5;  
 **private static double**[] *time* = {*timeForFirstStage*, *timeForSecondStage*, *timeForThirdStage*};  
 **private double timeStep** = 0.00025; *//counter for time* **private static** List<**double**[]> *allStages* = **new** ArrayList<>();  
 **private static** Complex[] *Z*;  
 **private boolean checkAllPhases**;  
 **private static int** *numberStage* = 0;  
 **private** Resistance **res**;  
  
 Logic() {  
 *allStages*.add(*stageFirst*);  
 *allStages*.add(*stageSecond*);  
 *allStages*.add(*stageThird*);  
 }  
  
 **public void** protect() {  
 *//если блокировки нет, то происходит определение попадания сопротивления  
 //в нормальном режиме защита заблокирована, разблокировывается по DI2* **if** (!**vector**.isBlocking() && Breaker.*isState*()) { *//алгоритм расчета сопротивления запускается если выполнены эти условия* **res**.startCalculateResistance();  
*// System.out.println("kek");  
 Z* = **resistance**.getResistanceComplex();  
 *//проходимся по всем ступеням  
 numberStage* = 0;  
*// actualTrip = false;  
 allStages*.forEach(actualStage -> {  
 *//проходимся по всем ФАЗАМ - СОПРОТИВЛЕНИЯМ* **checkAllPhases** = **false**;  
 **for** (Phases j : Phases.*values*()) {  
*// условие попадания в четырехугольник  
// System.out.println(" \n");  
// System.out.println("Фаза = " + j);  
// System.out.println("Z[j] = " + Z[j.ordinal()].re() +' '+ Z[j.ordinal()].im() + " ступень - "+numberStage);* **boolean** condOne = conditionHigher(*Z*[j.ordinal()], actualStage, 0);  
 **boolean** condTwo = conditionHigher(*Z*[j.ordinal()], actualStage, 1);  
 **boolean** condThree = conditionHigher(*Z*[j.ordinal()], actualStage, 2);  
 **boolean** condFour = conditionLower(*Z*[j.ordinal()], actualStage, 3);  
 **if** (condOne && condTwo && condThree && condFour) {  
 **checkAllPhases** = **true**;  
 }  
 }  
  
 *//если хотя бы по одной из фазе сработал пусковой орган,то* **if** (**checkAllPhases**) {  
 *//пуск  
 str*[*numberStage*] = **checkAllPhases**;  
 *counter*[*numberStage*] = *counter*[*numberStage*] + **timeStep**;  
 **double** settingTime = *time*[*numberStage*];  
 **if** (*counter*[*numberStage*] > settingTime) {  
 *//срабатывание  
 trip*[*numberStage*] = **checkAllPhases**;  
 }  
 } **else** {  
 *//сбрасывание сигнала и счетчика  
 str*[*numberStage*] = **false**;  
 *counter*[*numberStage*] = 0;  
 }  
 ++*numberStage*;  
  
 });  
 }  
 }  
  
 **private boolean** conditionHigher(Complex z, **double**[] coef, **int** numberLine) {  
 *//index of the array double []* numberLine = numberLine \* 2;  
 **boolean** cond = **false**; *//изначально - false* **double** k = coef[numberLine];  
 **double** b = coef[numberLine + 1];  
 **if** (z.im() - k \* z.re() - b >= 0) {  
 cond = **true**;  
 }  
 **return** cond;  
 }  
  
 **private boolean** conditionLower(Complex z, **double**[] coef, **int** numberLine) {  
 numberLine = numberLine \* 2; *//index of the array double []* **boolean** cond = **false**; *//изначально - false* **double** k = coef[numberLine];  
 **double** b = coef[numberLine + 1];  
 **if** (z.im() - k \* z.re() - b <= 0) {  
 cond = **true**;  
 }  
 **return** cond;  
 }  
  
 **public void** setResistance(Resistance resistance) {  
 **this**.**resistance** = resistance;  
 }  
  
 **public void** setVector(Vectors vector) {  
 **this**.**vector** = vector;  
 }  
  
 **public boolean**[] getTrip() {  
 **return** *trip*;  
 }  
  
 **public boolean**[] getStr() {  
 **return** *str*;  
 }  
  
 **public void** setRes(Resistance res) {  
 **this**.**res** = res;  
 }  
}

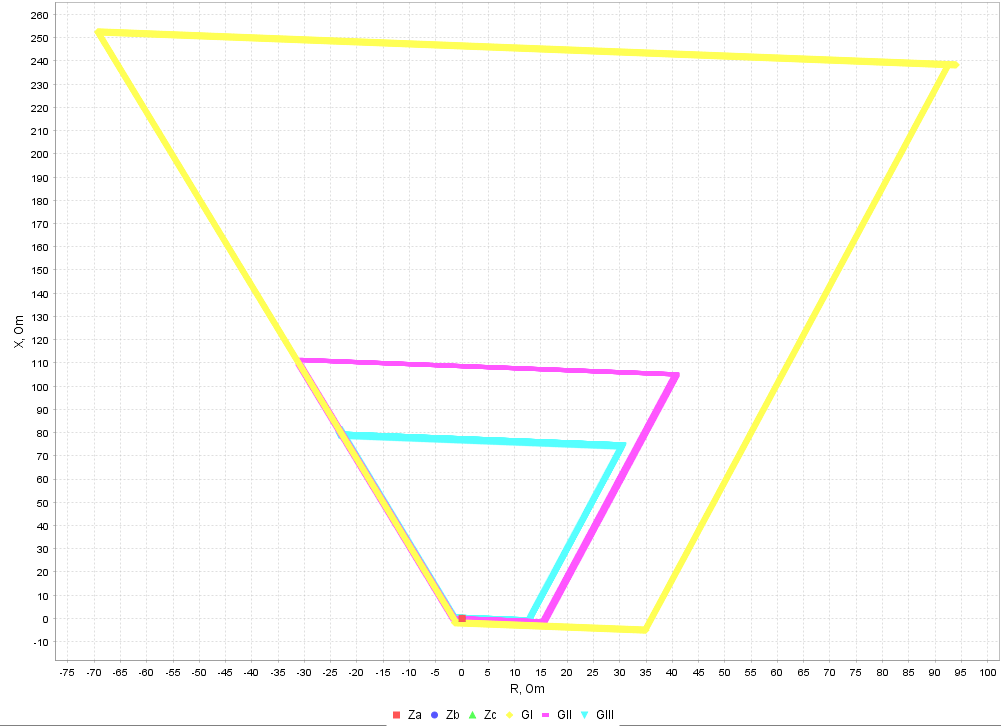


Рис. 4. Характеристика срабатывания для 1,2, 3 ступени

**Результаты.**

1. Файл – KZ1

На рис. 5 представлена работа алгоритма ДЗ. Судя по осциллограммам происходит двухфазное КЗ. Сработала 1 ступень защиты, но запустились все три ступени (str), в результате выключатель отключился, а ток исчез, напряжение восстановилось. На Годографе наглядно видно попадание точки в зону защиты.

1. Файл – KZ2

На рис. 6 представлена работа алгоритма ДЗ. Судя по осциллограммам происходит трехфазное КЗ. Сработала 1 ступень защиты, в результате выключатель отключился, а ток исчез, напряжение восстановилось. На Годографе наглядно видно попадание точки в зону защиты.

1. Файл – KZ3

На рис. 7 представлена работа алгоритма ДЗ. Судя по осциллограммам происходит двухфазное КЗ за спиной, защита не сработала, так как сопротивление не попало в зону срабатывания.

1. Файл – KZ4

На рис. 8 представлена работа алгоритма ДЗ. Судя по осциллограммам происходит однофазное КЗ за спиной, защита не сработала, так как блокирующий орган не сработал (DI2).

1. Файл – KZ5

На рис. 9 представлена работа алгоритма ДЗ. Судя по осциллограммам происходит двухфазное КЗ за спиной, защита не сработала, так как сопротивление не попало в зону срабатывания.

1. Файл – KZ6

На рис. 10 представлена работа алгоритма ДЗ. Судя по осциллограммам происходит двухфазное КЗ за спиной, защита не сработала, так как блокирующий орган не сработал (DI2).

1. Файл – KZ7

На рис. 11 представлена работа алгоритма ДЗ. Судя по осциллограммам происходит однофазное КЗ за спиной, защита не сработала, так как блокирующий орган не сработал (DI2).

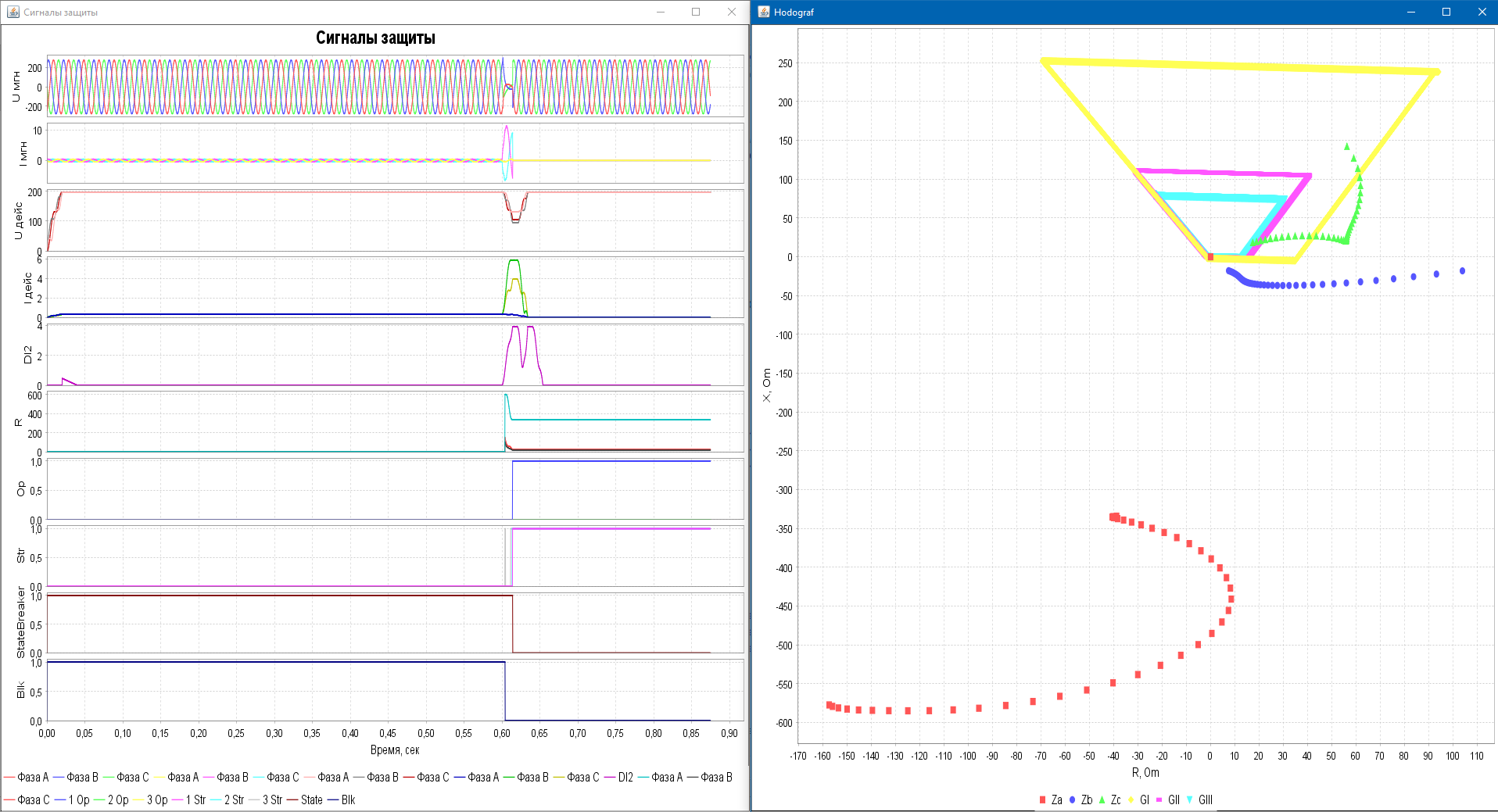


Рис. 5 Осциллограммы аналоговых значений KZ1

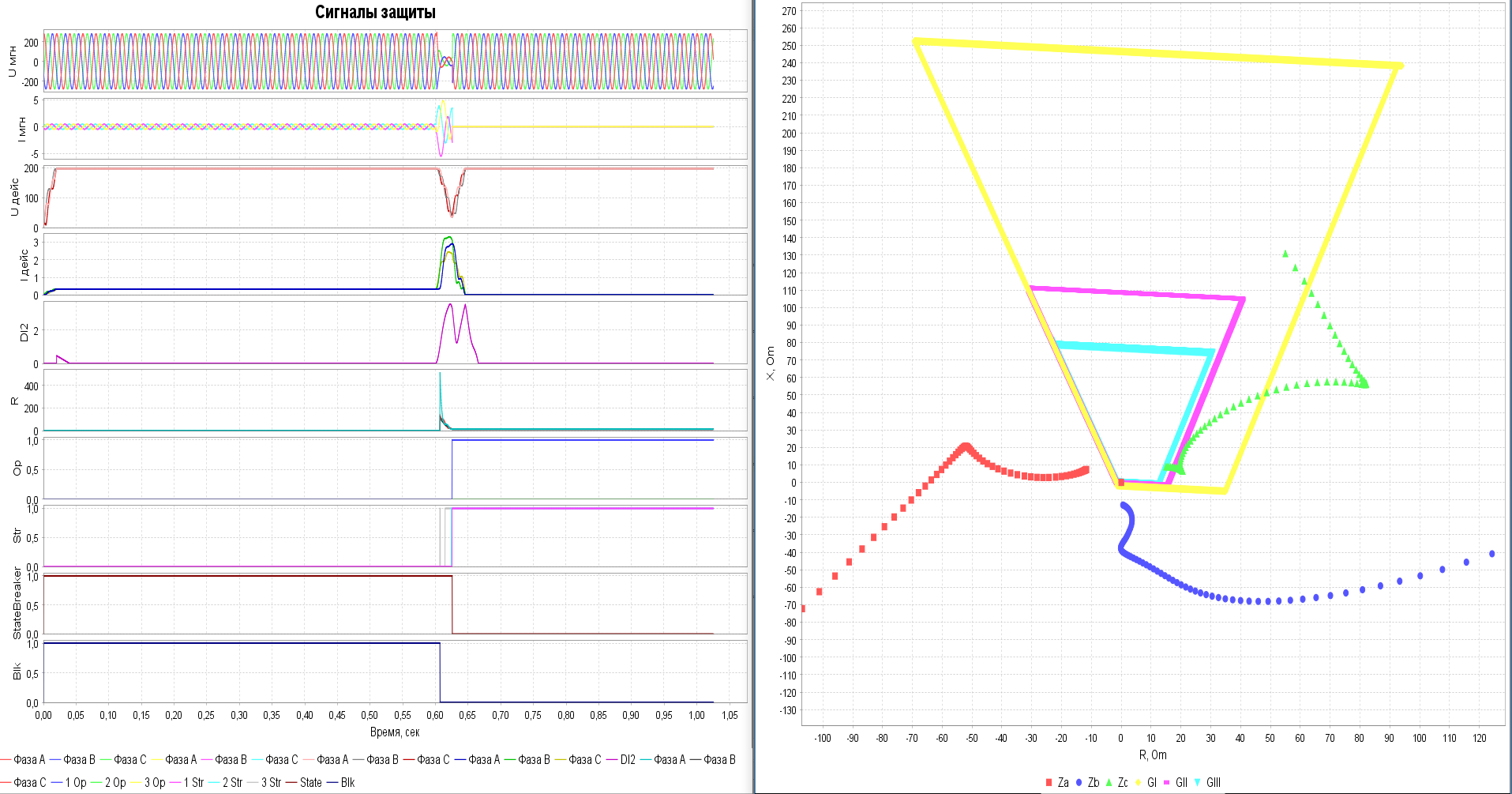


Рис. 6 Осциллограммы аналоговых значений KZ2

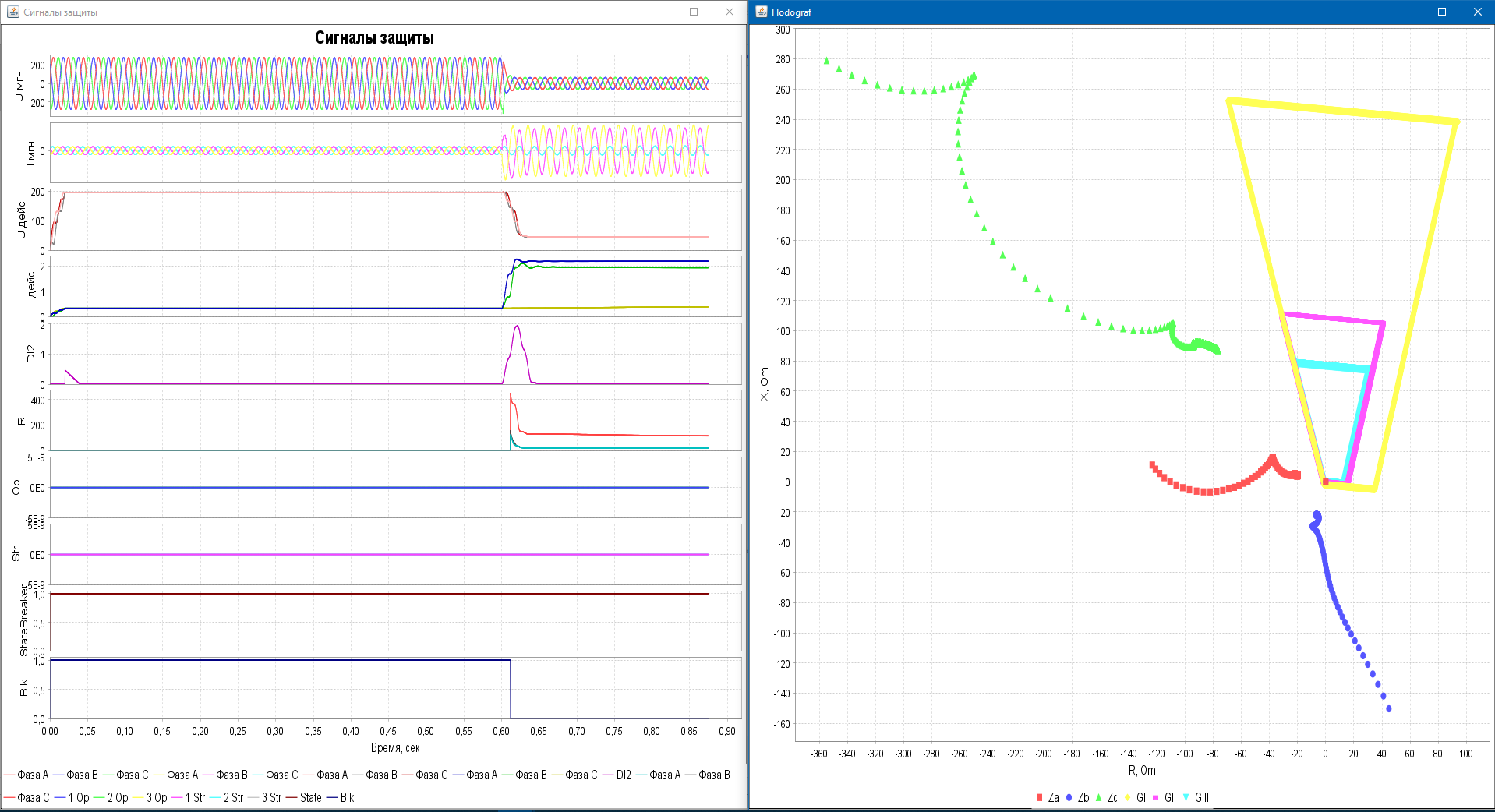


Рис. 7 Осциллограммы аналоговых значений KZ3

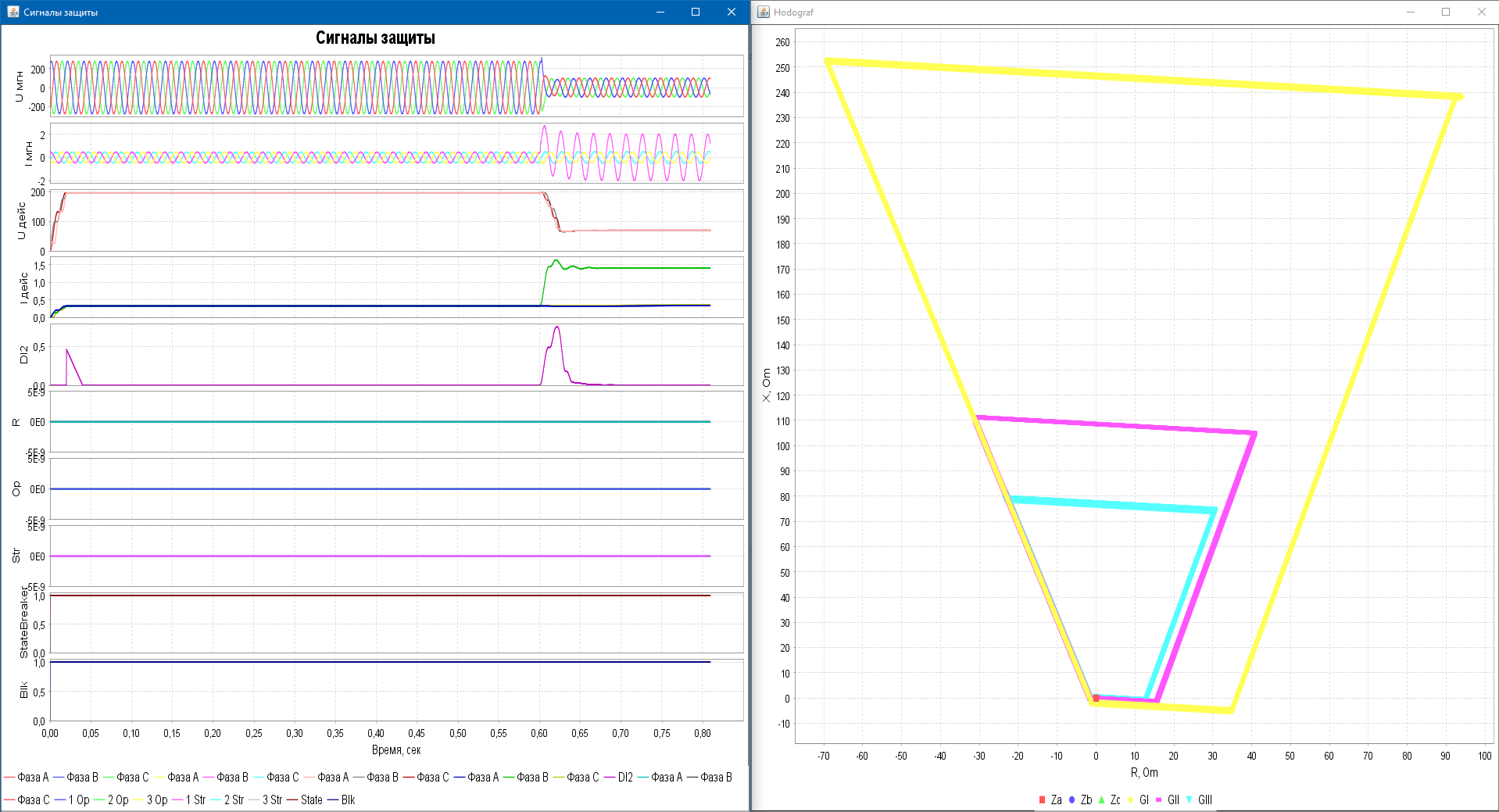


Рис. 8 Осциллограммы аналоговых значений KZ4

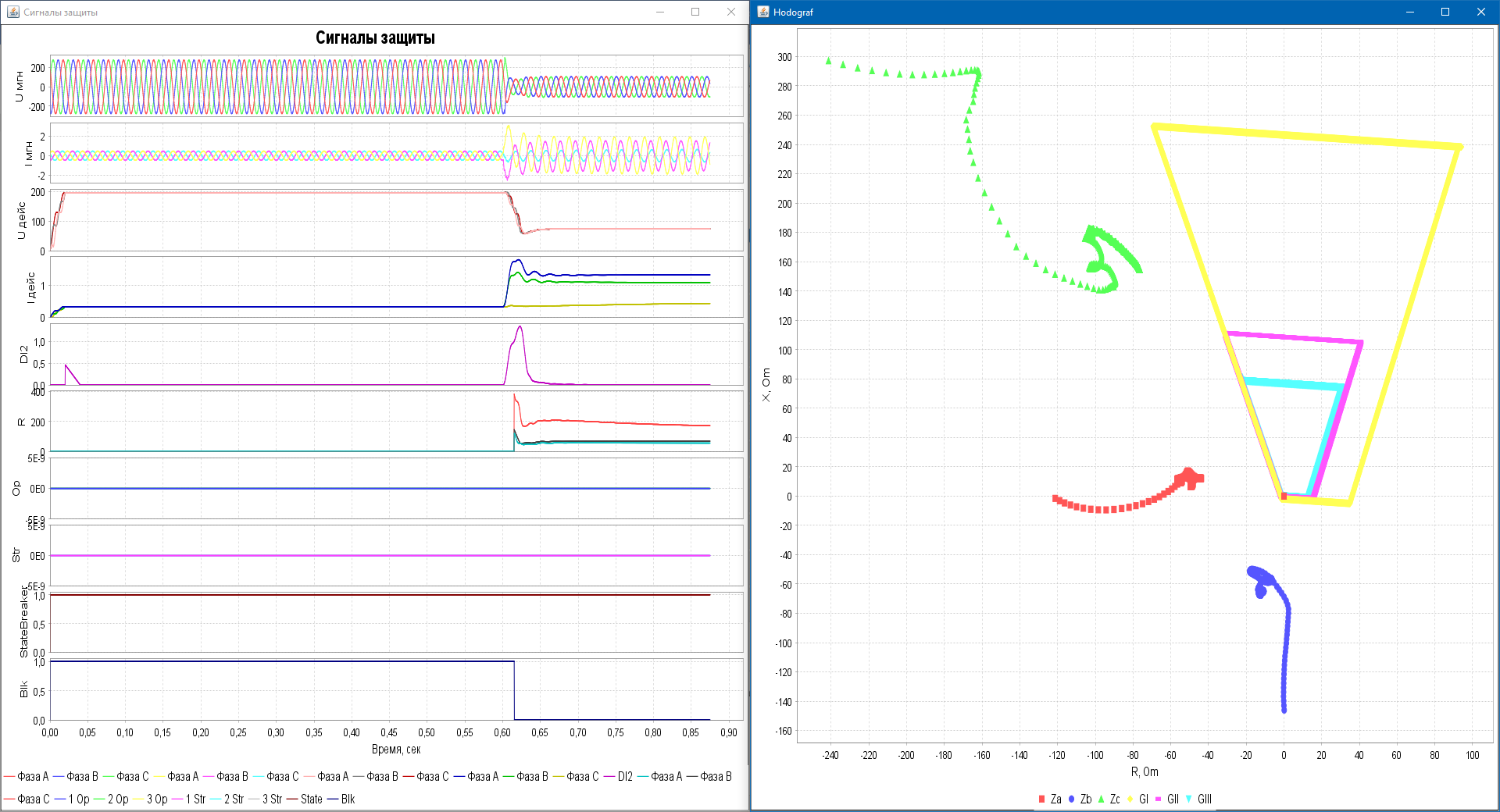


Рис. 9 Осциллограммы аналоговых значений KZ5

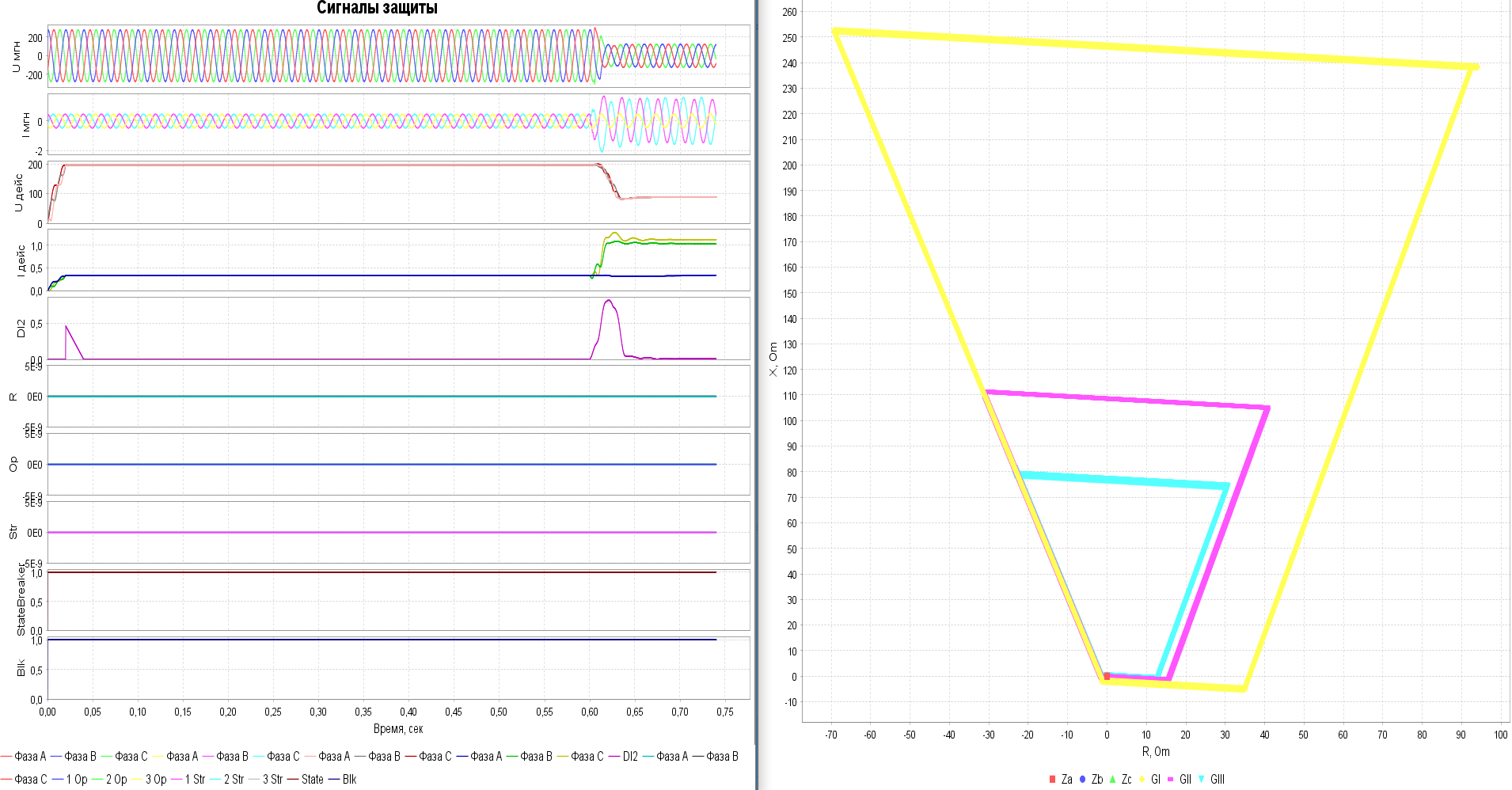


Рис. 10 Осциллограммы аналоговых значений KZ6

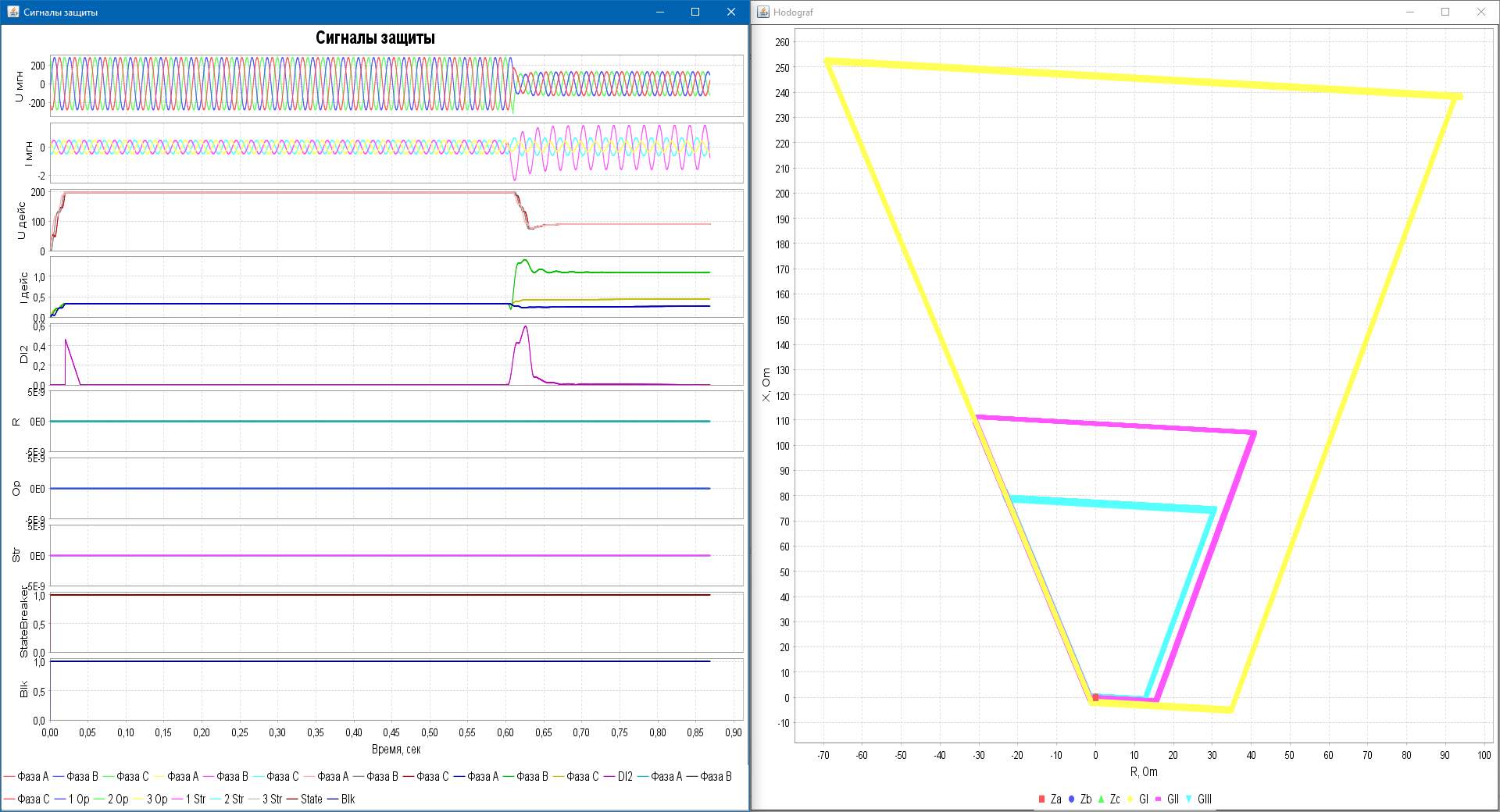


Рис. 11 Осциллограммы аналоговых значений KZ7



Рис. 12 Взаимодействие объектов