НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”
Теплоенергетичний факультет
Кафедра автоматизації теплоенергетичних процесів

«На правах рукопису»
УДК 62-799

«До захисту допущено»
В.о.завідувача кафедри

_____ / *В.А.Волощук*/
“____” _____ 2019 р.

Магістерська дисертація
на здобуття ступеня магістра

зі спеціальності ***151“Автоматизація та комп’ютерно-інтегровані технології”***

на тему: Система діагностики відмов та автоматизованого обслуговування турбоустановок

Виконав: студент II курсу, групи ТО-81мп
Резник Дмитро Олександрович _____
(прізвище ім’я, по батькові) (підпис)

Науковий керівник ст. викладач, Поліщук Ігор Анатолійович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент_____

Київ – 2019 року

Національний технічний університет України
“Київський політехнічний інститут
імені Ігоря Сікорського”

Факультет ***Теплоенергетичний***
Кафедра ***Автоматизації теплоенергетичних процесів***
Рівень вищої освіти – другий(магістерський) за освітньо-професійною програмою

Спеціальність ***151“Автоматизація та комп’ютерно-інтегровані технології”***

ЗАТВЕРДЖУЮ

В.о.завідувача кафедри

_____ /*В.А.Волощук*/
(підпис) (ініціали, прізвище)
“ “ _____ 2019 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Резнику Дмитру Олександровичу
(прізвище, ім'я, по-батькові)

1. Тема дисертації Система діагностики відмов та автоматизованого обслуговування турбоустановки

науковий керівник дисертації Поліщук Ігор Анатолійович, ст. викладач
(прізвище, ім'я, по-батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «4» листопада 2019 р. № 3812-с

2. Термін подання студентом дисертації «10» грудня 2019 р.

3. Об’єкт дослідження вібродіагностика турбоустановки

4. Предмет дослідження (вихідні дані для магістерської дисертації за освітньо-професійною програмою)
програмно-технічна реалізація методів та алгоритмів віброаналізу

5. Перелік завдань, які потрібно розробити
1) Огляд турбоустановки, як об’єкту діагностики
2) Розробка програмно-технічного комплексу
3) Забезпечення можливості передачі параметрів по універсальному протоколу обміну даними.
4) Розробка стартап-проекту.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу
1) Схема функціональна автоматизації;
2) Схема принципова електрична;
3) Креслення щита автоматизації;
4) Схема зовнішніх з'єднань;
5) Вікно реалізованого програмного додатку;
6) Вікно SCADA-системи.
7. Орієнтовний перелік публікацій
1. Резник Д.О. ,Поліщук І.А. «Проблематика впровадження систем Автоматизованого обслуговування технологічного обладнання»// Матеріали XVII Міжнародної науково-практичної конференції молодих вчених та студентів, м. Київ, 23– 26 квітня 2019 р. У 2 т. – К. КПІ ім. Ігоря Сікорського, 2019.-Т.2-28с.
2. Резник Д.О., Поліщук І.А., Костанецький К.В. «Система вібродіагностики, прогнозування відмов та оцінки стану турбоагрегату»// Матеріали XXII (76)випуску журналу Молодий вчений. грудень 2019.

8. Дата видачі завдання " 04 " вересня 2018 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1	*Видача завдання*	04.09.2018	
2	*Аналітичний огляд проблеми*	15.11.2018	
3	*Огляд математичного розрахунку параметрів вібродіагностики*	25.01.2019	
4	*Аркуш 1. Схема автоматизації функціональна.*	14.03.2019	
5	*Схема принципова електрична*	16.03.2019	
6	*Основні програмні рішення з системи діагностики відмов*	18.06.219	
7	*Реалізація обміну даними*	01.11.2019	
8	*Розробка SCADA-системи*	15.11.2019	
9	*Стартап-проект*	29.11.2019	
10	*Підпис керівника магістерської дисертації*		
11	*Попередній захист магістерської дисертації*	11.12.2019	
12	*Захист*	19.12.2019	

Студент _____ _____
(підпис) (прізвище та ініціали)

Науковий керівник дисертації _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Магістерська дисертація складається зі вступу, чотирьох розділів, висновку, переліку посилань з 9 найменуваннь, **7** додатків, і містить 16 рисунків, 23 таблиць. Повний обсяг магістерської дисертації складає 135 сторінки, з яких перелік посилань займає 1 сторінку, додатки — 66 сторінк.

Актуальність теми

На сьогоднішній день у всьому світі спостерігається тенденція зростання потреб у виробленні електричної енергії внаслідок технологічного прогресу.

Через слабкий розвиток використання відновлювальних джерел енергії близько 98% електроенергії в України виробляється на АЕС (54,4%), ТЕС, ТЕЦ (35,7%) та ГЕС, ГАЕС (8,0%). На всіх цих електростанціях для генерації електроенергії використовується турбогенератор з’єднаний з турбіною, що обертається за рахунок зовнішнього впливу пари, газу або кінетичної енергії води. Отже турбоагрегат (комплекс турбіни з турбогенератором) є однією з головних ланок вироблення електроенергії. Важливо підтримувати працездатність цієї ланки, оскільки при виході її з ладу необхідно зупиняти весь процес вироблення ел. струму на довгий час до моменту відновлення працездатності, що призводить до колосальних збитків на підприємстві.

Система діагностики відмов за допомогою методу віброаналізу дозволила б уникнути неприємних наслідків від несподіваного виходу з ладу обладнання, оскільки, здатна помітити незначні зміни поводження турбіни під час руху, та визначити, що саме призвело до цієї зміни. Такий підхід дозволить заздалегідь виявляти можливі несправності установки, вирішувати методи їх ліквідації та обирати зручний час для ремонту в момент слабкого навантаження електромережі споживачами.

Мета і завдання роботи

Метою даної роботи є створення програмного продукту для аналізу даних з вібродатчиків турбоустановки з можливістю передачі обрахованих параметрів

вібрації у хмарні системи вібродіагностики по уніфікованому протоколу передачі даних.

Об'єкт досліджень – методи виявлення несправностей з показів датчика вібрації.

Предмет досліджень – програмні засоби вібродіагностики обладнання.

Наукова новизна отриманих результатів полягає у виборі ефективних засобів обрахунку вібропараметрів та їх передачі інші допоміжні системи діагностики або SCADA систему.

Практичне значення отриманих результатів

1. Розроблено та реалізовано програмний продукт системи вібродіагностики.
2. Забезпечено передачу параметрів по протоколу обміну даними Modbus TCP.

Апробації результатів дисертації. Результати роботи дисертації було оприлюднено у:

1. XVII Міжнародної науково-практичної конференції молодих вчених та студентів, 23-26 квітня 2019 року. У 2 т.
2. XXII (76)випуск журналу Молодий вчений. грудень 2019

Публікації

Резник Д.О. ,Поліщук І.А. «Проблематика впровадження систем автоматизованого обслуговування технологічного обладнання»// Матеріали XVII Міжнародної науково-практичної конференції молодих вчених та студентів, м. Київ, 23– 26 квітня 2019 р. У 2 т. – К. КПІ ім. Ігоря Сікорського, 2019. – Т. 2. – 28 с.

Резник Д.О., Поліщук І.А., Костанецький К.В. «Система вібродіагностики, прогнозування відмов та оцінки стану турбоагрегату»// Матеріали XXII (76) випуску журналу Молодий вчений. грудень 2019.

Ключові слова. Вібродіагностика, турбоустановка, обмін даними, прогнозування несправностей.

РЕФЕРАТ

Магистерская диссертация состоит из введения, четырех глав, заключения, списка ссылок из 9 наименования, 7 приложений и содержит 16 рисунков, 23 таблиц. Полный объем магистерской диссертации составляет N страницы, с которых перечень ссылок занимает 1 страницу, приложения - 66 страниц.

Актуальность темы

На сегодняшний день во всем мире наблюдается тенденция роста потребностей в выработке электроэнергии в результате технологического прогресса.

Из-за слабого развития использования возобновляемых источников энергии около 98% электроэнергии в Украине производится на АЭС (54,4%), ТЭС, ТЭЦ (35,7%) и ГЭС, ГАЭС (8,0%). На всех этих электростанциях для генерации электроэнергии используется турбогенератор соединен с турбиной, вращается за счет внешнего воздействия пара, газа или кинетической энергии воды. Итак турбоагрегат (комплекс турбины с турбогенератором) является одной из главных звеньев выработки электроэнергии. Важно поддерживать работоспособность этого звена, поскольку при выходе ее из строя необходимо останавливать весь процесс выработки эл. тока на долгое время до момента восстановления работоспособности, что приводит к колоссальным убыткам на предприятии.

Система диагностики отказов с помощью метода виброанализу позволила бы избежать неприятных последствий неожиданного выхода из строя оборудования, поскольку, способна заметить незначительные изменения поведения турбины во время движения, и определить, что именно привело к этому изменению. Такой подход позволит заранее выявлять возможные неисправности установки, решать методы их ликвидации и выбирать удобное время для ремонта в момент слабой нагрузки электросети потребителями.

Цель и задачи работы

Целью данной работы является создание программного продукта для анализа данных с вибродатчиков турбоустановки с возможностью передачи рассчитанных

параметров вибрации в облачные системы вибродиагностики по унифицированному протоколу передачи данных.

Объект исследований - методы выявления неисправностей с показов датчика вибрации.

Предмет исследований - программные средства вибродиагностики оборудования.

Научная новизна полученных результатов заключается в выборе эффективных средств расчета вибропараметров и их передачи другие вспомогательные системы диагностики или SCADA систему.

Практическое значение полученных результатов

1. Разработан и реализован программный продукт системы вибродиагностики.
2. Обеспечено передачу параметров по протоколу обмена данными Modbus TCP.

Апробации результатов диссертации. Результаты работы диссертации были обнародованы в:

1. XVIII Международный научно-практической конференции молодых ученых и студентов, 23-26 апреля 2019 года. В 2 т.
2. XXII (76) выпуск журнала Молодой ученый. декабрь 2019

Публикации

Резник Д.А. Полищук И.А. «Проблематика внедрения систем автоматизированного обслуживания технологического оборудования» // Материалы XVIII Международный научно-практической конференции молодых ученых и студентов., Г. Киев, 23-26 апреля 2019 В 2 т. - М. КПИ им. Игоря Сикорского, 2019. - Т. 2. - 28 с.

Резник Д.А., Полищук И.А., Костанецький К.В. «Система вибродиагностики,

прогнозирования отказов и оценки состояния турбоагрегата »// Материалы XXII (76) выпуска журнала Молодой ученый. Декабрь 2019.

Ключевые слова. Vibrodiagnostics, turbine installations, data exchange, fault prediction.

ABSTRACT

The master's thesis consists of an introduction, four chapters, a conclusion, a list of links from 9 titles, 7 applications and contains 16 figures, 23 tables. The full scope of the master's thesis is 135 pages, from which the list of links occupies 1 pages, applications - 66 page.

Actuality of theme

Today, around the world there is a growing trend in the demand for electricity generation as a result of technological progress.

Due to the weak development of the use of renewable energy sources, about 98% of electricity in Ukraine is produced at nuclear power plants (54.4%), thermal power plants, thermal power plants (35.7%) and hydroelectric power stations, PSPs (8.0%). At all these power plants, a turbogenerator is used to generate electricity; it is connected to a turbine and rotates due to the external influence of steam, gas, or the kinetic energy of water. So, a turbine unit (a complex of a turbine with a turbogenerator) is one of the main links in power generation. It is important to maintain the health of this link, since when it fails, it is necessary to stop the entire process of generating email. current for a long time until the restoration of efficiency, which leads to enormous losses in the enterprise.

A failure diagnosis system using the vibroanalysis method would avoid the unpleasant consequences of an unexpected failure of the equipment, since it is able to notice minor changes in the behavior of the turbine during movement, and determine what exactly led to this change. This approach will allow you to pre-identify possible installation failures, solve methods for their elimination and choose a convenient time for repair at the time of a weak load on the power supply by consumers.

The aims and objectives of the study

The aim of this work is to create a software product for analyzing data from vibration sensors of a turbine unit with the ability to transfer calculated vibration parameters to cloud vibration diagnostics systems using a unified data transfer protocol.

The object of research software vibration diagnostics equipment.

The subject of research - systems, models, methods and algorithms for automated control of creating thermal comfort in the building.

Innovative novelty

Consists in choosing effective means of calculating the vibration parameters and their transmission to other auxiliary diagnostic systems or the SCADA system.

The practical significance of the results

1. The software product of the vibrodiagnostics system has been scattered and implemented.

2. Secure transmission of parameters via the exchange protocol with Modbus TCP data.

Publications

Reznik D.A. Polishchuk I.A. “The Problems of Implementing Automated Maintenance Systems for Technological Equipment” // Materials of the XVIII International Scientific and Practical Conference of Young Scientists and Students., Kiev, April 23-26, 2019 In 2 vols. - M. KPI named after Igor Sikorsky, 2019 .-- T. 2 .-- 28 p.

Reznik D.A., Polishchuk I.A., Kostanetsky K.V. "System of vibration diagnostics, forecasting failures and assessing the condition of the turbine unit ”// Materials of the XXII (76) issue of the journal Young Scientist. December 2019.

Keywords

Vibrodiagnostics, turbine installations, data exchange, fault prediction.

Зміст

Перелік умовних скорочень

АЕС – атомна електрична станція;

АСКВД – автоматична система контролю вібродіагностики;

АЦП – аналогово-цифровий перетворювач;

ВПП – валоповоротний пристрій;

ООП – об’єктно орієнтоване програмування;

ПТЗ СКМВТ – програмно-технічні засоби системи контролю механічних величин турбоустановки;

ЦВТ – циліндр високого тиску;

ЦНТ – циліндр низького тиску;

ЦСТ – циліндр середнього тиску;

ЧВТ – частина високого тиску (частина циліндра високого тиску з парою високого тиску);

ЧСТ – частина середнього тиску(частина циліндра високого тиску з парою низького тиску);

ШПФ – швидке перетворення Фур’є.

Вступ

На сьогоднішній день у всьому світі спостерігається тенденція зростання потреб у виробленні електричної енергії внаслідок технологічного прогресу.

В Україні електричні станції забезпечують вироблення більшої частини електроенергії, оскільки альтернативні джерела енергії ще не зазнали такого широкого застосування, щоб скласти конкуренцію електричним станціям. Отже, зі зростанням енергетичних потреб людства, питання підтримання їх високої надійності, економічності та загалом питання модернізації електричних станцій є надзвичайно важливим та актуальним.

В даній роботі увага приділяється одному з основних елементів виробництва електроенергії на електростанціях, а саме турбоагрегату - сукупності турбіни та електрогенератора. Оскільки, турбіна є дуже важливою частиною виробництва електроенергії, то будь-яка аварійна ситуація, що призводить до зупинки роботи турбоустановки має значний вплив на роботу всієї системи. В більшості випадків з зупинкою турбіни на атомних електростанціях доводиться зупиняти роботу реактору. Це призводить до колосальних збитків. Отже, паротурбінні установки потребують періодичного планового ремонту з метою ретельної перевірки стану турбіни та її компонентів, заміни зношених частин, візуального огляду і т.д. для виключення можливостей аварійних ситуацій під час експлуатації. До планового ремонту обладнання висуваються жорсткі вимоги щодо часу, витраченого на обслуговування компонентів, через високу вартість простою обладнання. Погіршує ситуацію те, що кожен плановий ремонт обладнання займає великий проміжок часу, через повний огляд системи, а не цілеспрямований на існуючі чи спрогнозовані несправності. Для діагностики несправностей та дослідженням коректної роботи турбоустановки користуються методом вібродіагностики. Сучасні установки обладнані системою віброаналізу у режимі реального часу, та здатні зрівнювати виміряне значення вібрації з дозволеним діапазоном відхилень і в разі потреби

повідомити оператора про небезпеку та необхідність зупинки роботи агрегату, але вони не зберігають великі масиви даних для дослідження поводження установки та попередження нестандартних подій. Мета даної роботи наблизити можливість використання предиктивної системи віброаналізу шляхом висвітлення основного набору вібропараметрів, що необхідні для автоматизованої системи контролю вібродіагностики у різних режимах роботи турбоустановки та запропонувати програмний продукт для передачі даних до систем прогнозування аварійних ситуацій турбоустановок різного типу. Переваги автоматизованої системи прогнозування аварійних ситуацій методом вібродіагностики полягає у:

- підвищенні надійності, терміну експлуатації обладнання;
- зниженні витрат на ремонтні роботи (проведення періодичних ремонтних робіт за вимогою, а не планово);
- підвищенні ефективності ремонтних робіт (зниження часу на ремонт обладнання в наслідок розуміння проблеми виходячи з оброблених даних);
- завчасна закупівля деталей, що підлягають заміні.

1. Аналітичний огляд проблеми

1.1 Сучасний стан галузі

Для підтримки працездатності та інформування оператора про нештатні ситуації в роботі турбоагрегату використовується система вібродіагностики, яка порівнює покази вібропараметрів виміряних за допомогою датчиків вібрації.

Підвищена вібрація, що виходить за граничні значення, викликає руйнування в усьому турбоагрегаті. При вібрації вал обертається в прогнутому стані, виникають дотикання між рухомими та нерухомими деталями. В результаті цього відбувається знос ущільнень, збільшення зазорів.

Якщо дотикання значні, то необхідно здійснити аварійну зупинку турбоагрегату, в іншому випадку наслідки можуть бути непередбачуваними.

Велику небезпеку становить вібрація для електричного генератора, тому що вона може привести до зсувів електричних обмоток і коротким замиканням.

Під дією вібрації відбувається ослаблення зв’язку окремих деталей: половин вкладишів і їх обойм, кришок підшипників і їх корпусів, фундаментних рам. Тому при експлуатації турбоагрегату вібрація повинна постійно контролюватися і не виходити за допустимі значення.

ПТЗ СКМВТ на підприємствах найчастіше забезпечує безперервну роботу турбоагрегату з періодом в один рік при будь-яких режимах роботи. Якраз такий період прийнято вважати достатнім для підтримки працездатності турбоагрегату. Тобто, кожного року виконуються планово попереджувальні ремонти з повним технічним оглядом, під час якого на доволі тривалий час повністю зупиняється вироблення електроенергії на даному обладнані.

ПТЗ СКМВТ на турбоагрегаті виконує вимірювання параметрів:

- осьового зсуву ротора турбіни;

- абсолютної вібрації по трьом складовим (вертикальна, поперечна, осьова) на з пор;

- відносного розширення роторів;
- теплового розширення корпусу;
- прогину ротора;
- частоти обертання ротора;
- частоти обертання ротора на ВПУ;

Головним завданням дослідженої системи є попередження оператора про перевищення параметру вібрації в режимі реального часу. При виявленні перевищення допустимого або безпечного рівня вібрації ця інформація передається у турбінний відділ, де спеціалісти визначають можливі несправності, їх наслідки та приймали рішення про подальші дії щодо роботи установки.

Мета роботи полягає у тому, щоб створити систему віброаналізу, тобто опрацювати вібросигнал знятий на працюючій машині, оскільки він містить великий обсяг інформації з якої можна зробити висновки про стан та подальше поводження системи, методи вирішення проблем та надання інструкцій по обслуговуванню без участі людини – спеціаліста в даній області.

Сутність діагностики і обслуговування турбоагрегату лежить в ранньому виявленні зароджуваних несправностей, але для ефективного використання віброконтролю в програмі технічного обслуговування необхідно, щоб ця інформація була належним чином вилучена з отриманих вібросигналів та оброблена.

1.2 Опис технологічної схеми об’єкту управління

Турбіна (від латинського слова turbo-вихор, обертання) - це лопатна машина, яка не має поршня і кривошипношатунного механізму, в якій потенційна і кінетична енергія потоку робочого тіла перетворюється в механічну енергію обертання валу. Залежно від типу робочого тіла турбіни розділяють на парові, газові і гідравлічні. На теплових і атомних електростанціях в якості приводу електричного генератора застосовують парові турбіни.

Турбоагрегат насамперед – це складова паротурбінної установки, яка у спрощеному вигляді складається з парогенератора, парової турбіни, конденсатора, живильного насосу див. Рис 1.1. Спрощений принцип дії паротурбінної установки описаний нижче.

Паротурбінна установка (див. Рис 1.1) працює по замкненому циклу та включає у себе:

- Парогенератор;
- Парову турбіну;
- Конденсатор;
- Живильний насос.

Парогенератор перетворює енергію палива в теплову енергію робочого тіла. Парова турбіна перетворює теплову енергію робочого тіла в механічну енергію ротора. Електрогенератор перетворює механічну енергію ротора в електричну енергію[1].

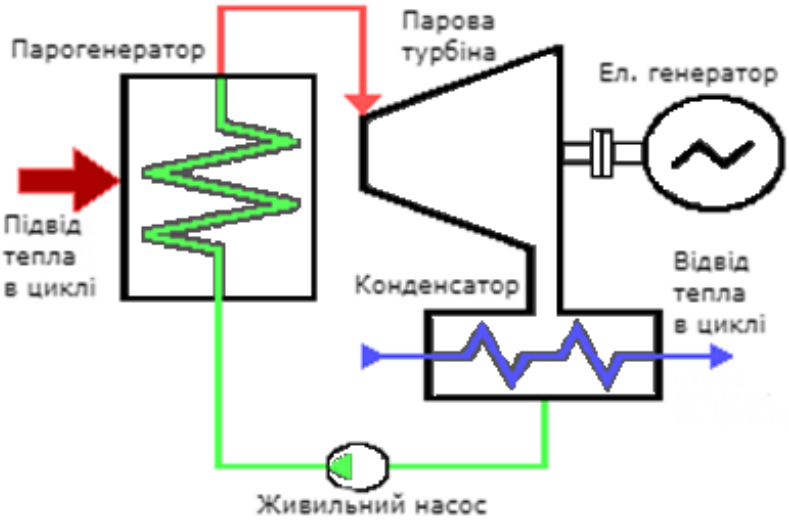


Рис. 1.1 – Замкнений цикл роботи ПТУ

Для турбіни виділяють 4 експлуатаційні режими роботи:

- Режим навантаження;
- Режим валоповороту;
- Режим набору обертів;
- Режим зупинки.

Режим навантаження – це номінальний режим роботи турбоустановки.

Режим валоповороту – режим пуску турбоустановки коли вона обертається на мінімальних обертах за рахунок валоповоротного пристрою.

Режим набору обертів – поступове збільшення частоти обертів при запуску турбоустановки після відключення валоповоротного пристрою до виходу на номінальний режим роботи.

Режим зупинки – поступове зменшення частоти обертів до зупинки турбоустановки.

Безпосереднім джерелом коливань є валопровід турбоагрегату - система з'єднаних між собою роторів. Валопровід, обертаючись на масляній плівці підшипників, передає через неї коливання до підшипників і їх корпусів (опор). Вібрація турбоагрегату може відбуватися в трьох напрямках, тому її вимірюють на всіх підшипникових опорах в трьох взаємно-перпендикулярних напрямках по відношенню до осі валу турбоагрегату: вертикальному, осьовому, поперечному.

Вібрацію турбоагрегату вимірюють на всіх підшипникових опорах. Осьову і поперечну вібрацію вимірюють на рівні осі валу турбоагрегату. Вимірювальні датчики прикріплюються до основи підшипника. Вертикальну вібрацію вимірюють на верхній частині кришки підшипника. На Рис.1.2 зображено принцип розташування вібродатчиків на опорах підшипників.

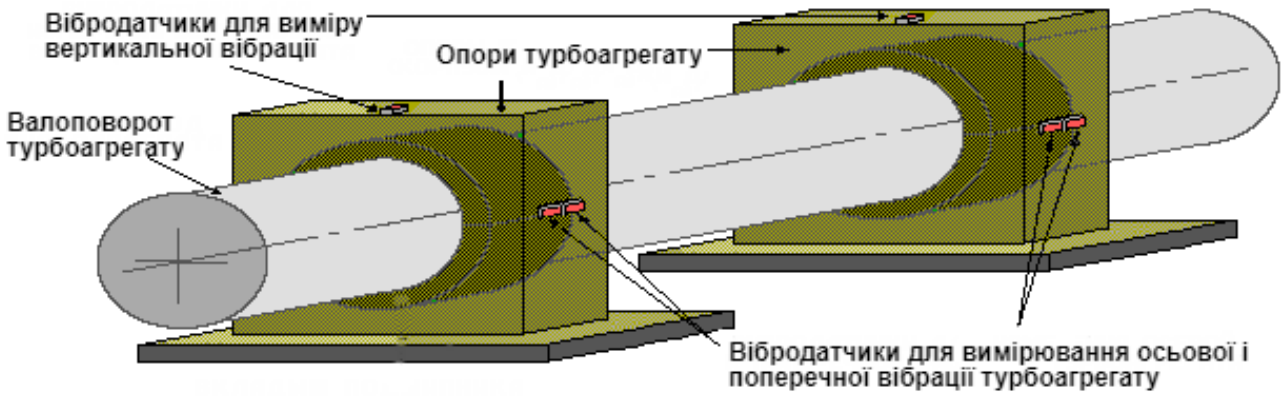


Рис.1.2 Принцип розташування вібродатчиків на опорах підшипників

Далі у існуючих системах сигнали з вібродатчиків прямують до щитів управління де обробляються, зрівнюються з допустимими параметрами, після чого направляються до автоматизованих робочих місць по всьому об’єкту.

1.3 Загальна постановка задачі

Запропонувати систему вібродіагностики турбогенератора певного типу з описом параметрів, що знімаються з датчиків, обраховуються та аналізуються, функціональною схемою, схемою щита, схемою зовнішніх проводок, переліком обладнання, забезпечити можливість передачі даних по універсальному протоколу обміну даними для можливості передачі великого обсягу даних в хмарне сховище, доступу систем диспетчеризації або доступу допоміжних систем для прогнозування можливих несправностей та попередження аварійних ситуацій системи. Це сприяє подальшому запровадженню предиктивного аналізу, що усуває необхідність проведення планового періодичного ремонту, що в свою чергу зменшує час на обслуговування обладнання та звужує фронт робіт при позаплановому обслуговуванні. А також виключає вплив людини при аналізі стану та подальшого поводження системи, знаходженні методів вирішення проблеми та надання інструкцій по обслуговуванню.

2. Опис об’єкту управління

2.1 Об’єкт управління

Як об’єкт управління в даній роботі розглядається турбіна парова турбіна Т-160-130 для використання на теплових електростанціях. Це Двоциліндрова конденсаційна парова турбіна з проміжним перегрівом і розвиненою системою регенеративного підігріву живильної води.

Пара від котла по двом паропроводам підводиться до стопорного клапану і потім направляється до чотирьох регулюючих клапанів, кожен з яких з'єднаний зі своєю сопловою коробкою. Дві соплові коробки встановлені в нижній половині внутрішнього корпусу ЦВТ, а інші дві - у верхній див. у додатку А.

Турбіна має сопловий паророзподіл. Перші два регулюючих клапана діаметром 120 мм відкриваються одночасно і підводять пар до сопловим коробок, розташованим в нижній половині корпусу. Це дозволяє забезпечити рівномірний прогрів корпусу по колу і виключити його викривлення. При повному відкритті двох перших клапанів турбіна розвиває 75% номінальної потужності. Номінальна потужність забезпечується при додатковому відкритті третього клапана діаметром 135 мм (лівого верхнього, якщо дивитися на генератор). Четвертий клапан є перевантажувальний і працює при зниженні початкових параметрів пари аж до 12 МПа і 555ºС або при погіршенні вакууму. Відкриття чотирьох клапанів при номінальних параметрах пари дозволяє отримати потужність 165 МВт. Основні параметри турбіни можна знайти в таблиці 2.1

Табл. 2.1 основні параметри турбіни

Назва параметру	**Величина**
Номінальна потужність, МВт	160
Тиск свіжої пари перед стопорним клапаном, МПа	12,7
Температура свіжої пари перед стопорним клапаном, С	565

продовження табл. 2.1

Тиск пари перед блоками клапанів промперегріву при номінальній потужності, МПа	2,8
Температура пара перед блоками клапанів промперегріву при номінальній потужності, С	565
Температура охолоджуючої води при вході в конденсатор, С	12
Тиск пари при вході в конденсатор, кПа	3,43
Температура підігріву живильної води, С	229
Частота обертання ротора, 1 / с	50
Число циліндрів	2
Число виходів пару	2
Число ступенів в ЧВТ	7
Число ступенів в ЧСТ	8
Число ступенів в ЧНТ	12
Число регенеративних відборів пара	7
Загальна маса турбіни з комплектуючих обладнанням, т	420
Довжина турбіни, м	14,44
Висота, м	5,79
Ширина, м	6,48

В даному виді турбін поєднано ЧВТ і ЧСТ в одному ЦВТ, що дозволило зменшити вдвічі кількість кінцевих ущільнень, зменшити і організувати оригінальну систему ущільнень а також зменшити кількість опорних підшипників. Розміщення підшипників дивись на Рис. 2.1.

Кожен з роторів встановлений на двох опорних підшипниках зі сферичними вкладишами. Передній підшипник є комбінованим опорно-упорним, зі сферичним

вкладишем. Корпус переднього підшипника - виносний, двох інших вбудовані в вихідні патрубки ЦНТ. Кришки підшипників містять аварійні масляні ємності.

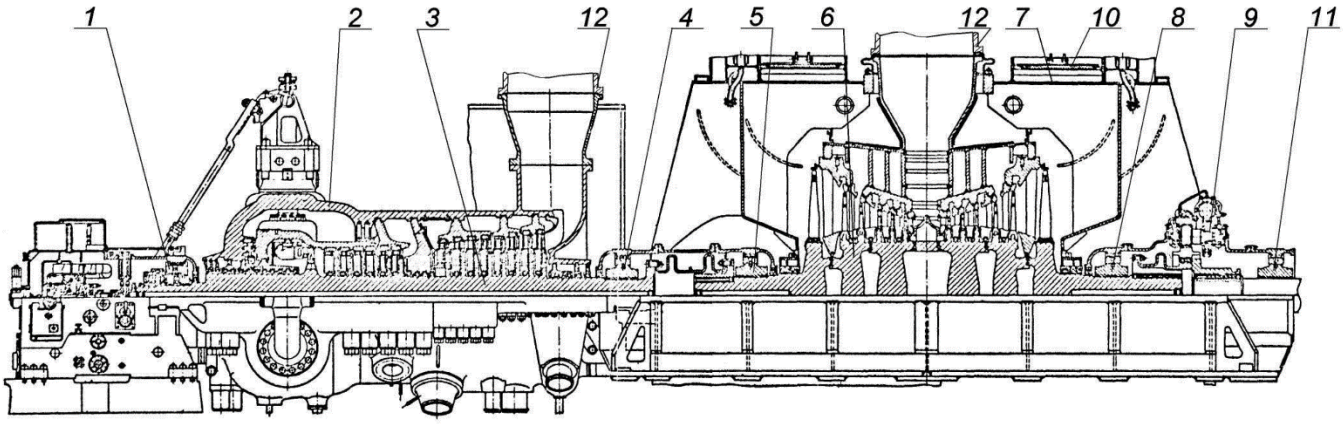


Рис. 2.1. Турбіна К-160-130 повздовжній розріз.

1 – Підшипник передній (№1), 2 – ЦВТ, 3 – РВТ, 4 – підшипник опорний (№2), 5 - підшипник опорний (№3), 6 – РНТ, 7 – ЦНТ, 8 - підшипник опорний (№4), 9 – ВПП, 10 – атмосферний клапан, 11 – підшипник опорний (№5) 12 – ресивер.

2.2 Призначення і функції створюваної системи

Основні функції системи повинні передбачити:

- Оперативний контроль і аналіз вібраційного і механічного стану агрегатів на основі вимірюваних вібраційних параметрів, параметрів механічних величин експлуатаційних параметрів.
- Попереджувальна сигналізація при реєстрації аномальних вібраційних станів, формування сигналів тривоги для зовнішніх пристроїв.
- Формування архіву значень вібраційних, механічних і експлуатаційних параметрів, перегляд та аналіз архівних даних.
- Оперативна діагностика вібраційного стану турбоагрегатів.
- Відображення і документування інформації.
- Розподіл інформації по уніфікованому відкритому протоколу обміну даними, тим самим вирішити проблему інтеграції з іншими системами (програмними

продуктами) для хмарного обчислення даних, прогнозування можливих несправностей та попередження аварійних ситуацій.

2.3 Вимоги до реалізацій функцій системи

До системи вібродіагностики виносяться такі вимоги:

- Безперервний автоматичний контроль абсолютної і відносної вібрації.
- Збір та обробка вібросигналів, виявлення дефектів безперервно працюючого промислового обладнання.
- Передавати масиви даних за допомого відкритого протоколу передачі даних Modbus TCP для можливості здійснити інтеграцію системи в будь-яку АСУТП без додаткових доробок ПО.

2.4 Висновки

Для побудови системи, необхідно дослідити основні необхідні та достатні параметри для побудови системи вібродіагностики турбоустановки, підібрати датчики, пристрої збору та обробки інформації, забезпечити збір та передачу даних по протоколу Modbus TCP, реалізувати Scada програму, щоб перевірити працездатність.

3. Розробка системи управління об’єктом

3.1 Функціональна структура управління об’єктом

В даній роботі розглядається слідкуюча (інформаційна) система с функцією захисту при виникненні позаштатної аварійної ситуації.

3.1.1 Інформаційна функція

Під інформаційною функцією розуміється забезпечення збору та обробки достовірної інформації про дійсний стан об’єкта.

Одним з найбільш відповідальним елементом системи збору вібраційних даних є датчик, що слугує для перетворення механічних коливань в електричний

сигнал. На кожній опорі турбіни К-160-130 розташовано 3 датчики віброшвидкості для діагностування вібраційних параметрів опор та 2 датчики вібропереміщення для діагностування вібраційних параметрів валу.

Кожен датчик віброшвидкості вимірює одну з трьох складових вібрацій опори: вертикальну, поперечну або осьову. Датчики вібропереміщення валу вимірюють складові вібропереміщення у вертикальному та горизонтальному положеннях.

Для виміру параметрів віброшвидкості були обрані датчики ВК-312, а для параметрів вібропереміщення датчики ВК-310НС фірми «ВіКонт». Також вимірюється частота обороту валу за допомогою оптичного датчику DTK-1. Сигнали з датчиків надходять на плату промислового комп’ютера L-780M, де з початкових параметрів вібрації обраховуються похідні параметри, що використовуються для діагностики стану турбіни.

3.1.2 Превентивна функція

Превентивна функція допомагає виявити небажані відхилення та запобігти негативним наслідкам. Саме на превентивну функцію потрібно зробити наголос, оскільки обробка значень вібропараметрів та отримання висновків стану турбіни по ним є основною ідеєю розширення системи вібродіагностики. В ідеальному варіанті при зміні вібропараметрів знятих з турбіни системою повинно видаватись повідомлення оператору у вигляді поради, що містить інформацію про можливі порушення в системі, рекомендації щодо періоду проведення ремонту та типу деталей для замовлення.

Оцінювання появи і розвитку вібронебезпечних несправностей та позаштатних ситуацій, крім реалізованої на промисловому комп’ютері, повинна виконуватись на допоміжних системах за допомогою хмарного опрацювання даних відповідно до бази знань несправностей і бази даних відповідних їм зміні параметрів. База знань включає формалізовані правила розпізнавання дефектів та їх

ознаки, що сформовані на основі даних досліджень і досвіду експлуатації турбоагрегату про вплив механічних дефектів (несправностей) і режимних факторів на вібропараметри роторів і опор підшипників. База знань може поповнюватись, якщо отримані нові додаткові дані про вплив дефекту.

3.1.3 Захисна функція

При діагностуванні різкого перевищення вібрації по двом каналам, тобто аварійної ситуації турбоустановка має бути переведена в режим аварійної зупинки. За вимогами до системи час спрацювання захисної функції не повинен перевищувати 1 с.

3.2 Розрахункова частина

3.2.1 Основи виміру вібрації

Вібрація – такий вид механічного руху, при якому кожна з точок тіла виконує періодичне переміщення, що повторюється поблизу деякого відносного нерухомого положення. Терміни вібрація і механічні коливання вважаються синонімами. Вібрація відбувається під дією сил збудження, які мають різні причини. В роторних машинах сили збудження зв’язані передусім з процесом обертання валу.

Розглядаючи на поверхні машини деяку точку, можна говорити про її вібрацію у заданому напрямі, наприклад вертикальному, при чому в іншому напрямі вібрація цієї точки може суттєво відрізнятися. Роздивляючись вібрацію точки в трьох ортогональних напрямах можна спостерігати, що точка в процесі коливання рухається по деякій траєкторії, що являє собою досить складну, майже завжди замкнену просторову криву, форма якої в загальному випадку може змінюватись в часі.

Вібрацію валу в деякому поперечному перерізі зазвичай розглядають як періодичний рух центру цього перетину по замкнутій траєкторії, при цьому

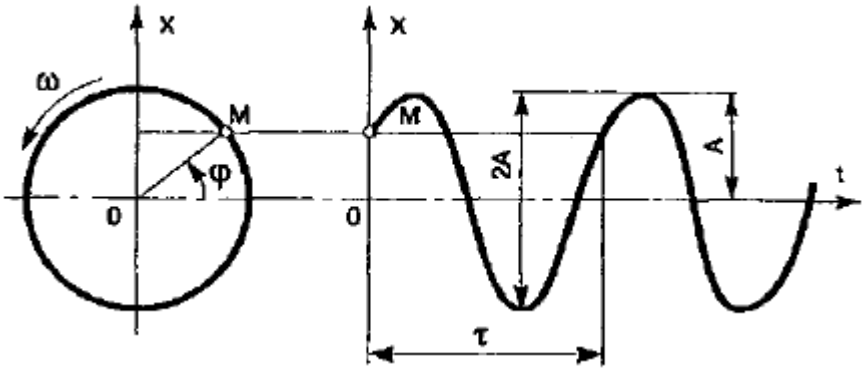
найчастіше говорять про відносну вібрацію, тобто про рух валу щодо прилеглих статорних елементів, наприклад щодо підшипника.

У загальному випадку різні точки машини мають різну вібрацію: вони можуть коливатися одночасно в різних напрямках і з різними амплітудами, деякі, так звані вузлові, точки можуть бути нерухомі, в процесі коливань елементи машини можуть зазнавати пружні деформації, при цьому можна говорити про форми коливань машини. Таким чином, реальний вібраційний процес дуже складний для того, щоб всі його деталі і подробиці могли бути предметом контролю і вивчення в процесі експлуатації машини. Доцільні обмеження інформації по вібрації зазвичай зводяться до того, що регламентують об'єкти і точки контролю вібрації. Так, для більшості випадків вібрацію контролюють на опорах машини в трьох ортогональних напрямках.

У найпростішому випадку вібруюча поверхня здійснює гармонійні (синусоїдальні) коливання, при цьому координата точки, що коливається визначається рівнянням (3.1)

$$X = Asin(\omega t + \varphi) \tag{3.1}$$

де А -амплітуда коливань, яка вимірюється лінійними одиницями (мм, мкм); ω -



кругова частота коливань (c^{-1}); φ, - початкова фаза коливань в кутових одиницях (градус, радіан). Графік гармонійних коливань представлений на Рис. 3.1.

Рис. 3.1 Графік руху точки при гармонійних коливаннях.

Кругова частота коливань визначається частотою коливань f:

$$\omega = 2\pi f \tag{3.2}$$

Частота коливань f визначає число коливань в одиницю часу і вимірюється в герцах (Гц). Період коливань τ - час повного коливання в секундах - пов'язаний з частотою коливань f співвідношенням (3.3)

$$\tau = \frac{1}{f} \tag{3.3}$$

Гармонійні коливання в різних точках деякого об'єкту, можуть збігатися за частотою. Ці коливання є синхронними. Синхронні коливання відрізняються один від одного амплітудою і фазою, їх величини зручно розглядати як комплексні, при цьому амплітуда є модулем, а фаза - аргументом комплексної величини. Для уявлення синхронних коливань може бути використаний один із записів, наведених в (3.4):

$$\left.\begin{aligned} \overline{A} = Acos\varphi + iAsin\varphi = A_x + iA_y; \\ \overline{A} = Ae^{i\varphi};\ \overline{A} = A\angle\varphi. \end{aligned}\right\} \tag{3.4}$$

Графічно коливання (1.4) зображуються у вигляді векторів в прямокутній системі координат, φ = 0 ° співпадає з віссю абсцис x, а φ = 90 ° - з віссю ординат y.

Додавання і віднімання векторів коливань проводяться за правилами дії з комплексними величинами і можуть виконуватися графічно. На Рис. 3.2 представлений приклад графічного додавання і віднімання векторів коливань, при цьому.

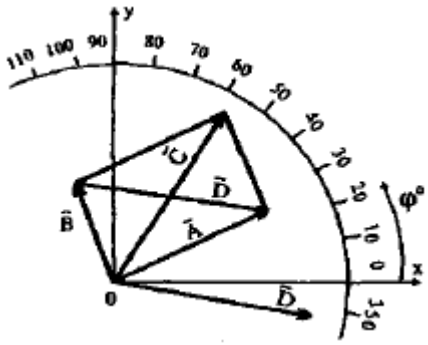


Рис. 3.2 Додавання та віднімання коливань.

$$\left.\begin{aligned}\overline{C} &= \overline{A} + \overline{B};\\ \overline{D} &= \overline{A} - \overline{B}.\end{aligned}\right\} \tag{3.5}$$

При усуненні вібрації обертових машин найбільший інтерес представляють коливання оборотної частоти. Фази таких коливань пов'язують з деяким кутовим положенням ротора. Фазою коливання прийнято вважати кутове положення мітки в момент максимального позитивного відхилення вібруючої поверхні від положення рівноваги.

З виміром фази вібрації пов'язано поняття про ударну точку ротора. Ударна точка ротора - це точка поверхні ротора, що лежить на радіусі, який збігається з напрямком вимірювання вібрації в момент найбільшого відхилення вібруючої поверхні (точка В на Рис. 3.3).

Як видно з Рис. 3.3, фаза вібрації φ визначає кутове положення ударної точки ротора щодо мітки на роторі.

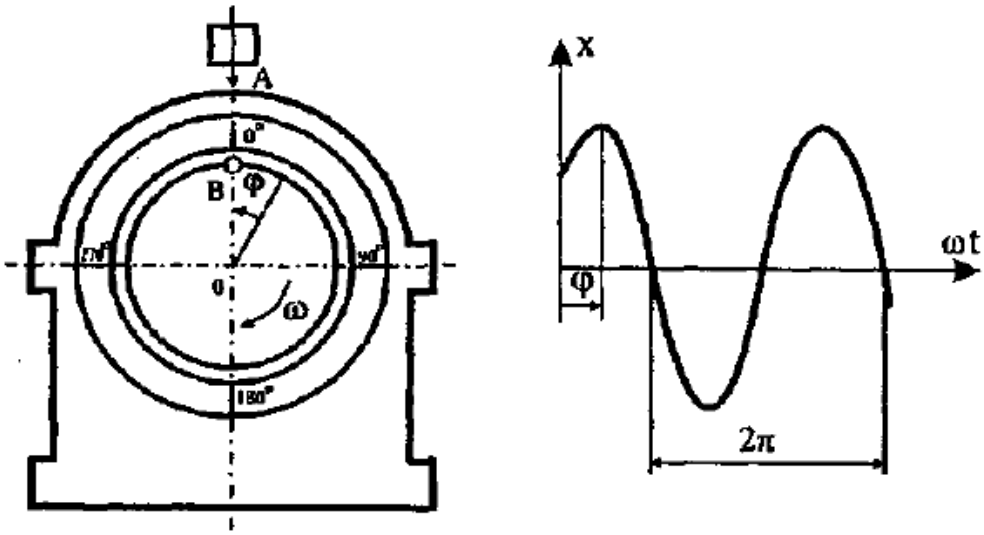


Рис. 3.3 Визначення фази оборотної вібрації.

ОА – направлення виміру вібрації; В- ударна точка ротора; ωt – кут повороту роторної мітки; φ – фаза вібрації.

Потрібно відзначити, що для даного уявлення про фазу вібрації координата точки, що коливається визначається рівнянням, яке відрізняється від (3.1):

$$X = Acos(\omega t - \varphi) \tag{3.6}$$

Це зауваження принципово не змінює описаних вище уявлень про кінематику гармонійних коливань. Важливо також відзначити, що вирази (3.1) і (3.6) припускають протилежні по відношенню до напрямку обертання ротора напрямки відліку фаз, обидва ці напрямки рівноправні і в рівній мірі застосовні в практиці, але їх не можна плутати. Напрямок відліку фаз приймається як деяка умовність. Нами прийнята послідовність фаз, яка визначається описаним представленням про ударну точку ротора і виразом (3.6).

При вібраційних дослідженнях цікавлять насамперед різниці фаз вібрації в контрольованих точках. Для їх визначення достатньо виміряти фази з точністю до постійного доданку. Більшість методів вимірювання фази вібрації якраз і передбачають таке визначення фазових кутів. Якщо все ж таки необхідно внести визначеність у розглянуту сукупність вимірів фаз φ_i, то слід визначити поправку $\Delta\varphi$. Ця поправка для кожної схеми вимірювання при деякій швидкості обертання

для всіх контрольованих точок постійна. Фази вібрації при цьому визначаються зі співвідношення

$$\varphi_i = \varphi_i' + \Delta\varphi, i = 1, 2, 3 \ldots \tag{3.7}$$

де φ_i - фази вібрації по прийнятим визначенням (див. Рис. 3.3); φ_i' - фази вібрації в довільній системі відліку; $\Delta\varphi$ - фазова поправка.

Коливання в двох точках, що збігаються по фазі, називаються синфазними, а тих, що відрізняються за фазами на 180 ° - протифазні.

3.2.2 Віброперемiщення, віброшвидкість, віброприскорення

Формула (3.1) і відповідний графік (Рис. 3.1) являють параметри віброперемiщення при гармонійній вібрації. Диференціювавши по часу формули (3.1) отримаємо вирази для віброшвидкості та віброприскорення при гармонійних коливаннях.

$$\left.\begin{array}{c} X'(t) = A\omega\cos(\omega t + \varphi) = A\omega\sin\left(\omega t + \varphi - \frac{\pi}{2}\right) = \mathrm{V}\sin(\omega t + \varphi_1)\,; \\ V = A\omega; \\ X''(t) = -A\omega^2\sin(\omega t + \varphi) = W sin(\omega t + \varphi_2); \\ W = A\omega^2; \end{array}\right\} \tag{3.8}$$

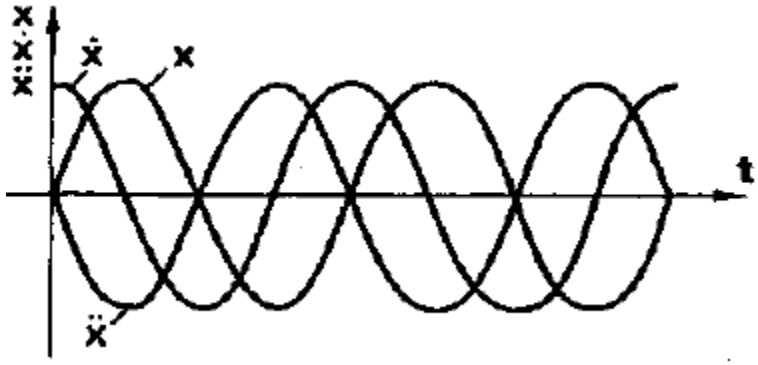


Рис. 3.4 Співвідношення між віброперемiщенням, віброшвидкістю та віброприскоренням (залежність від часу)

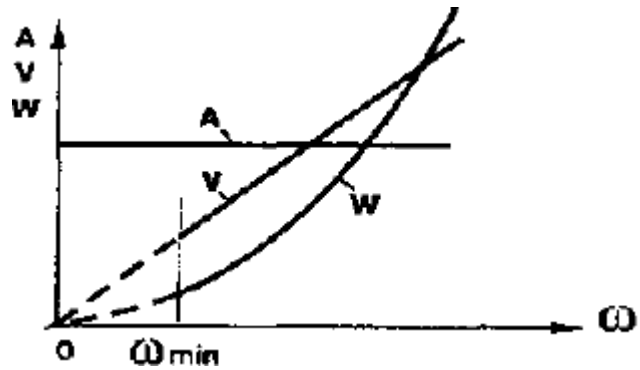


Рис. 3.5 Співвідношення між вібропереміщенням, віброшвидкістю та віброприскоренням (залежність амплітуд від частоти)

Де $X'(t)$ – віброшвидкість; $X''(t)$ – віброприскорення; V – амплітуда віброшвидкості; $\varphi_1 = \varphi - \frac{\pi}{2}$ – фаза віброшвидкості; W – амплітуда віброприскорення; $\varphi_2 = \varphi - \pi$ – фаза віброприскорення.

Як видно з формул (3.8), що розглядає параметри вібрації пов'язані між собою коефіцієнтами, залежними від частоти: при одній і тій самій амплітуді вібропереміщення амплітуда віброшвидкості зростає пропорційно частоті, а амплітуда віброприскорення зростає пропорційно квадрату частоти вібрації. Фаза віброшвидкості для деякої гармоніки вібрації зсунена відносно фази вібропереміщення на 90°, а фаза віброприскорення відповідно на 180°.

Рис. 3.4, 3.5 ілюструють співвідношення між параметрами гармонійної вібрації.

При частотах нижче певної мінімальної частоти, позначеної на рис. 1.5 як ω_{min} зазвичай користуються тільки параметром віброперемешенія внаслідок малості величин V та W.

3.2.3 Полігармонічна вібрація

У загальному випадку вібруюча поверхня здійснює коливання, що представляють собою суму декількох гармонійних коливань різних частот, при цьому про вібрації говорять, як про полігармонічні.

У роторних машинах зазвичай переважає гармонійна складова зі швидкістю обертання ротора, або оборотна вібрація. Вібрацію з частотою нижче оборотної називають низькочастотної; високочастотної вібрацією відповідно називають вібрацію з частотою, що перевищує оборотну.

Тимчасова реалізація віброперемiщення при деякій полігармонійній вібрації представлена на рис. 3.6.

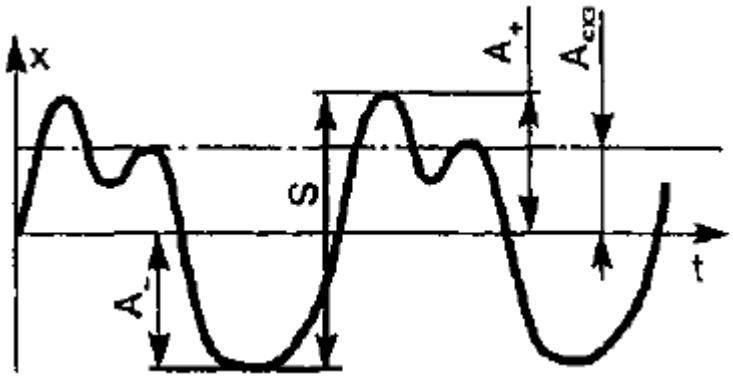


Рис. 3.6 Розгортка полігармонічної вібрації (віброперемiщення)

На часовому графіку відзначені розмах S (від піку до піку), середньоквадратичне значення (СКЗ) $A_{скз}$, позитивна амплітуда сигналу (від нуля до верхнього піку) A_+, негативна амплітуда сигналу (від нуля до нижнього піку) A_-. При цьому має місце співвідношення (1.9)

$$S = A_+ + A_- \tag{3.9}$$

Якщо в спектрі вібрації є тільки кратні частоти, то форма її в часі постійна і вона являє деяку періодичну криву (наприклад, відповідну рис. 3.4). Зміна форми тимчасового сигналу в часі свідчить про наявність в спектрі вібрації некратних гармонік.

Тут доречно зазначити, що сталість форми вібраційного сигналу є досить сильною ознакою кратності частот гармонійних складових вібрації, ніж свідчення використовуваних частотомірів.[3]

3.2.4 Швидке перетворення Фур’є

Для повного гармонічного аналізу необхідно дослідити окремі частотні складові сигналу з вібродатчику. Одним із методів обробки сигналів, який дозволяє охарактеризувати частотний склад вимірюваного сигналу – це спектральний аналіз. Спектри вібрацій є дуже корисними при аналізі аномалій у роботі машинного обладнання. Якщо такі аномалії мають місце, то він несе інформацію, яка може допомогти у визначенні джерела і причини аномалії.

Аналіз спектрів ґрунтується на змінах в поточних спектрах у порівнянні з еталонними (отриманими раніше при свідомо справному стані машини). Відмінності в амплітудах і частотах на графіку поточного спектра вказує на певні аномалії в роботі машини. Регулярний аналіз спектрів є ефективним способом раннього розпізнавання погіршення механічного стану і допомагає правильно спланувати ремонтні заходи до настання критичного моменту.

Математичною основою, яка пов'язує тимчасової або просторовий сигнал (або ж деяку модель цього сигналу) з його поданням у частотній області є перетворення Фур’є. Для перетворення відцифрованого аналогового сигналу у ряд Фур’є використовують формулу дискретного перетворення Фур’є (3.10).

$$X[k] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]W_N^{nk} \tag{3.10}$$

де величина $W_n = \exp\left(-j2\frac{\pi}{n}\right)$ носить назву обертаючого множника.

Але під час практичної реалізації дискретно – часового перетворення Фур’є (ДЧПФ) виникає проблема, що полягає у великій кількості обчислювальних операцій, пропорціональній $N2$. Цю проблему вирішує використання швидкого перетворення Фур’є (ШПФ) – ряду ефективних алгоритмів, призначених для швидкого обчислення дискретно-часового ряду Фур’є в якому число операцій пропорційне $Nlog_2N$. Основна ідея ШПФ – ділення N – точкового ДЧПФ

(3.11)

на два та більше ДЧПФ меншої довжини. Для отримання ШПФ виділимо з послідовності $x[n]$ елементи з парними та непарними номерами.

$$X[k] = \frac{1}{N}\sum_{m=0}^{\frac{N}{2}-1} x[2m]W_N^{2mk} + \frac{1}{N}W_N^k \sum_{m=0}^{\frac{N}{2}-1} x[2m+1]W_N^{2mk}$$

Але так як $(Wn)^z = \exp\left(-\frac{j4\pi}{n}\right) = \exp\left(-\frac{j2\pi}{\frac{N}{2}}\right) = W_{\frac{N}{2}}$, то $W_N^{2mk} = W_{\frac{N}{2}}^{mk}$. Отже (3.11) можна подати у вигляді:

$$X[k] = X_{\frac{N}{2}}[k,0] + W_N^k X_{\frac{N}{2}}[k,1], \tag{3.12}$$

де кожен доданок є перетворенням довжини $\frac{N}{2}$.

$$X_{\frac{N}{2}}[k,p] = \frac{1}{N}\sum_{n=0}^{\frac{N}{2}-1} x[2n+p]W_{\frac{N}{2}}^{nk} \ ,p = 0,1 \tag{3.13}$$

Зауважимо, що послідовність $(WN\ /\ 2)^{nk}$ періодична по k з періодом N/2. Тому, хоча номер k у виразі (3.12) приймає значення від 0 до N-1, кожна із сум обчислюється для значень k від 0 до N/2-1. Можна оцінити число комплексних операцій множення і додавання, необхідних для обчислення перетворення Фур'є відповідно до алгоритму (3.12) - (3.13). Два N/2 - точкових перетворення Фур'є за формулами (3.13) передбачають виконання 2(N/2)2 множень і приблизно стільки ж додавань. Об'єднання двох N/2 - точкових перетворень за формулою (3.12) вимагає ще N множень і N додавань. Отже, для обчислення перетворення Фур'є для всіх N значень k необхідно провести по N+N2/2 множень і додавань. У той же час пряме обчислення за формулою (3.10) вимагає по N^2 множень і додавань. Уже при N> 2 виконується нерівність N+N2/2 < N2, і, таким чином, обчислення за алгоритмом (3.12) - (3.13) вимагають меншого числа математичних операцій в порівнянні з прямим обчисленням перетворення Фур'є за формулою (3-10). Так як обчислення N-точкового перетворення Фур'є через два N/2 – точки призводить до економії

обчислювальних операцій, то кожне з N/2-точкових ДПФ слід обчислювати шляхом зведення їх до N/4-точковим перетворенням:

$$X_{\frac{N}{2}}[k,p] = X_{\frac{N}{4}}[k,p] + W_{N/2}^{k} X_{N/4}[k,p+2], p = 0,1$$

$$X_{\frac{N}{4}}[k,q] = \frac{1}{N}\sum_{n=0}^{\frac{N}{4}-1} x[4n+q] W_{\frac{N}{4}}^{nk}, p = 0,1,2,3$$

При цьому, внаслідок періодичності послідовності $W_{\frac{N}{4}}^{nk}$ по k з періодом N/4, суми (3.15) необхідно обчислювати тільки для значень k від 0 до N/4-1. Тому розрахунок послідовності X[k] за формулами (3.12), (3.14) і (3.15) вимагає вже по 2N+N2/4 операцій множення і додавання.

(3.14)

Слідуючи таким шляхом, обсяг обчислень X[k] можна все більш і більш зменшувати. Після m = log_2N розкладів приходимо до двухточковим перетворенням Фур'є виду:

(3.15)

$$X_2[k,p] = X_1[k,p] + W_2^E X_1\left[k, p + \frac{N}{2}\right], \quad p = 0,1 \dots, \frac{N}{2-1} \tag{3.16}$$

де $X_1[k,p]$ являє собоюпросто відліки сигналу x[n]:

$$X_2[k,p] = \frac{x[q]}{N}, q = 0,1, \dots N-1 \tag{3.17}$$

У підсумку можна записати алгоритм ШПФ у вигляді:

$$X_2[k,p] = \frac{x[p] + W_2^E x\left[p + \frac{N}{2}\right]}{N}$$

де $k = 0,1,\ p = 0,1, \dots, \frac{N}{2-1}$;

$$X_{\frac{2N}{M}}[k,p] = X_{\frac{N}{M}}[k,p] + W_{\frac{2N}{M}}^{k} X_{\frac{N}{M}}\left[k, p + \frac{M}{2}\right], \tag{3.18}$$

де $k = 0,1, \dots, \frac{2\boldsymbol{N}}{\boldsymbol{M}} - 1,\ p = 0,1, \dots, \frac{\boldsymbol{M}}{2-1}$;

$$\dots X[k] = X_N[k] = X_{\frac{N}{2}}[k,0] + W_N^k \boldsymbol{X}_{\frac{N}{2}}[k,1]$$

де $k = 0,1, \dots, N - 1$

На кожному етапі обчислень проводиться по N комплексних множень і додавань. А так, як число розкладів вихідної послідовності на підпослідовності половинної довжини рівне log2N, то повне число операцій множення-складання в алгоритмі БПФ одно Nlog_2N. При великих N має місце суттєва економія обчислювальних операцій в порівнянні з прямим обчисленням ДЧПФ. Наприклад, при N = 2^{10} = 1024 число операцій зменшується в 117 разів.

3.3 Розробка технічного забезпечення

3.3.1 Вимірювальна апаратура

Можливо найбільш відповідальним елементом системи збору вібраційних даних є датчик, що слугує для перетворення механічних коливань в електричний сигнал. В даній системі використовується три типи вібродатчиків для вимірювання трьох різних параметрів, а саме для виміру:

- Частоти повороту валу;
- Віброшвидкості опори підшипнику в трьох напрямках;
- Вібропереміщення валу у двох ортогональних напрямках.

В якості вимірювальної апаратури для виміру частоти обертів валу було обрано датчик **ДТК-1-С5818-90-5** фірми «Турбоконтроль». Він у парі з нерозбірним підсилювачем сигналу забезпечує безконтактне перетворення частоти оберту валу в послідовність прямокутних імпульсів напруги ± 5В. Основні характеристики датчика:

- Діапазон вимірювань – 0,05 … 10 000 Гц;
- Напруга живлення – 12…28 В;
- Робочий діапазон температур – +5…+100 ℃;
- Категорія захисту – IP67;
- Вихідний сигнал – ± 5В;

Датчик виміру частоти оберту валу видає прямокутний сигнал під час оберту валу в момент часу коли мітка нанесена на вал проходить в безпосередній близькості до нього. Міткою на валу може бути заглиблення, виступи або пази у будь-якому феромагнітному матеріалі, тобто позначка на самому валі.

Для виміру віброшвидкості на кожній з п’яти опор валу розташовуються три датчики ВК-312С фірми «ВіКонт». Перший датчик вимірює віброшвидкість вертикальної вібрації опори, другий – осьової, а третій – горизонтальної. Віброперетворювачі складаються з датчика та узгоджувального посилювача, змонтованого в окремому корпусі. Вони з'єднані вібро і термостійким кабелем. З'єднання датчика і зовнішнього підсилювача герметичне і не розбірне, поставляється комплектом.

Віброперетворювачі встановлюються на контрольованому обладнанні, так щоб напрямок осі основної чутливості було паралельно напрямку контрольованих коливань.

П’єзоелектричний перетворювач перетворює механічні коливання в електричний заряд, який поступає на посилювач. На виході з посилювача формується напруга, пропорційна миттєвому значенню віброшвидкості.

Основні характеристики датчика:

- Діапазон вимірювань – 0,1 … 100 мм/с;
- Напруга живлення – 24±1,5 В;
- Робочий діапазон температур – -40…+120 °С;
- Категорія захисту – IP65;
- Вихідний сигнал – ± 5В;

Для виміру вібропереміщення на валу біля кожної опори розташовуються два безконтактних індуктивних датчики ВК-310НС фірми «ВіКонт». Перший датчик вимірює вібропереміщення вертикальної вібрації валу, другий –горизонтальної. Віброперетворювачі складаються з датчика та узгоджувального посилювача,

змонтованого в окремому корпусі. Вони з'єднані вібро і термостійким кабелем. З'єднання датчика і зовнішнього підсилювача герметичне і не розбірне, поставляється комплектом.

Віброперетворювачі встановлюються на контрольованому обладнанні, так щоб напрямок осі основної чутливості було паралельно напрямку контрольованих коливань.

Датчик перетворює механічні коливання в електричний заряд, який поступає на посилювач. На виході з посилювача формується напруга, пропорційна миттєвому значенню віброшвидкості.

Основні характеристики датчика:

- Діапазон вимірювань – 10 … 400 мкм/с;
- Напруга живлення – 24±1,5 В;
- Робочий діапазон температур – -40…+120 °С;
- Категорія захисту – IP65;
- Вихідний сигнал – ± 5В;

3.3.2 Обчислювальна апаратура

В якості контролера використовується промисловий комп’ютер Avalue ESP – AT270 із універсальною платою вводу / виводу аналогових та цифрових сигналів L-780M.

Головною перевагою даної плати над промисловими контролерами є її швидкодіючий АЦП. Її аналогово-цифровий перетворювач здатний працювати з частотою 400 кГц (сумарно на всі канали) в той час коли промислові контролери, такі як, наприклад, Siemens S7-1500, що може обробляти дані з частотою 100 Гц . Оскільки, за один поворот валу турбоагрегату потрібно зняти велику кількість параметрів з кожного датчику вібрації за короткий проміжок часу, то швидкодія АЦП є одним з ключових параметрів при виборі обчислювальної апаратури. Промислові контролери у цьому питанні в значній мірі поступаються платі L-780M.

Також важливим питанням при підборі обладнання була простота реалізації програмного забезпечення з використанням математичних формул швидкого перетворення Фур'є. На промислових контролерах реалізувати швидке перетворення Фур'є було б набагато складніше ніж реалізувати цей самий алгоритм з використанням ООП на мові C# вільно користуючись стандартними бібліотеками та функціями.

Підключення плати до промислового комп'ютера відбувається через роз'єм PCI на материнській платі, забезпечує високу швидкість обміну даними з програмою користувача та виключає конфлікти з іншими платами в ПК.

Більшість важливих характеристик плати наведені в табл. 3.1.

Для легкого підключення сторонніх програмних засобів до спроектованої системи та розширення її можливостей передбачено реалізацію передачі обрахованих вібропараметрів по універсальному протоколу обміну даними Modbus TCP.

Табл 3.1 Характеристики плати L-780M

АЦП	
Кі-ть каналів	16 диференційних чи 32 з "загальною землею"
Розрядність АЦП	14 біт
Ефективна розрядність (вх. сигнал – синус 10 кГц / 4,9В)	13,3 біт (частота перетворення - 300 кГц)
Діапазон вхідного сигналу	± 5 В, ±1,25 В, ±0,3 В, ±0,08 В
Максимальна частота перетворення	400 кГц
Синхронізація	• Внутрішня • По зовнішньому синхросигналу • По рівню аналогового сигналу
Захист входів	± 25В (живл. вкл.) ± 10В (живлення викл.)
Цифровий сигнальний процесор	
Тип	ADSP 2185M
Тактова частота	29,5 МГц
Внутрішнє ОЗУ даних	16 Кслов
Внутрішнє ОЗУ програм	16 Кслов
FIFO-буфер	512 - 14336 слов
Обмін даними по шині PCI	
Швидкість при зверненнях до регістрів плати через порти вводу/виводу PC	До 1,2 Мбайт/с
Швидкість при зверненнях до регістрів плати через пам’ять PC	До 10 Мбайт/с
ЦАП (опція)	
Кількість входів напруги з загальною землею	2

Продовження табл. 3.1

Розрядність	12 біт
Час установки	8 мкс
Вихідний діапазон	±5 В
Вихідний струм, не Більше	2 мА
Цифрові входи и виходи	
Кількість входів	16 (паралельних, асинхронних)
Кількість виходів	16 (паралельних, асинхронних, с загальним дозволом виходів)

На Рис. 3.7 зображено загальний вигляд плати L-780M головними компонентами.

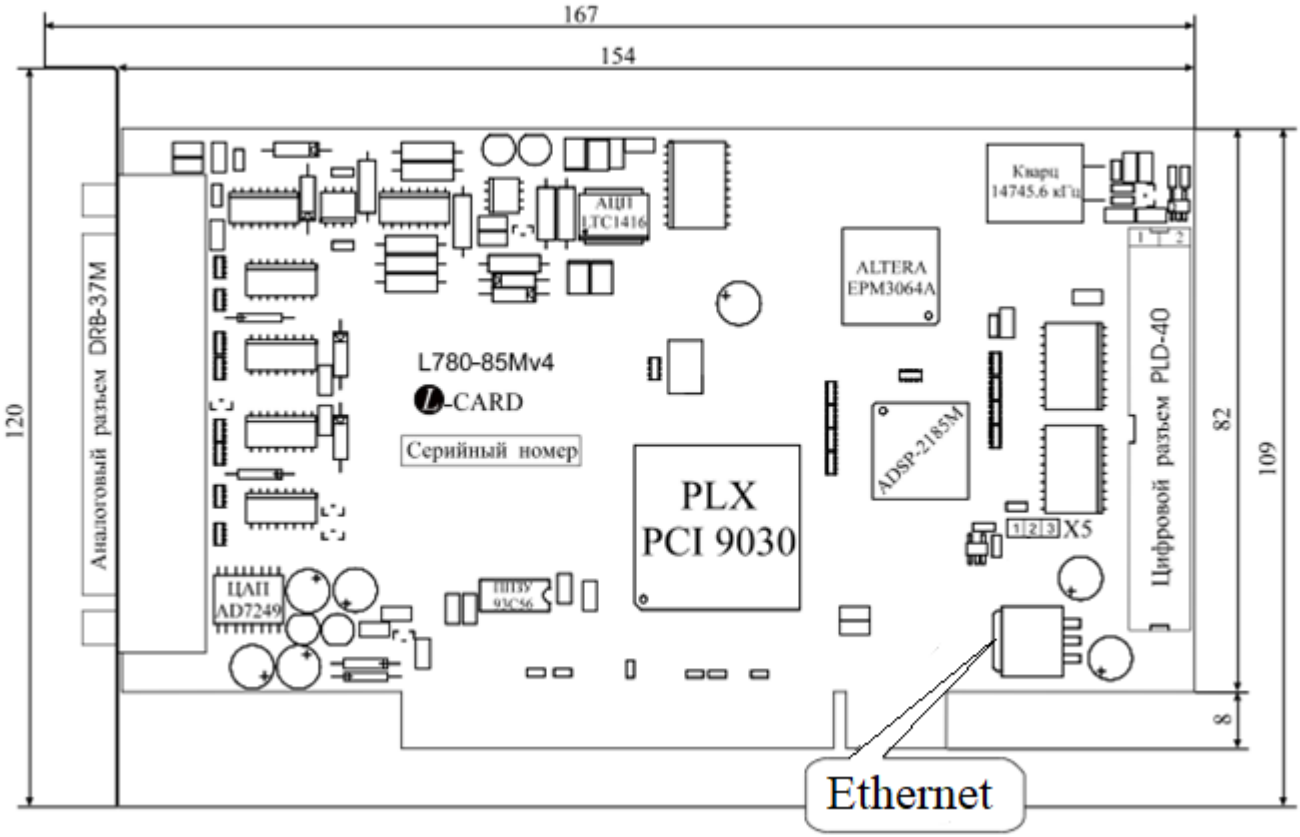


Рис. 3.7 Плата L-780M

Для забезпечення надійнішого каналу зв’язку з датчиком, сигнальні лінії необхідно припаювати на роз’єм DRB – 37M аналогового вводу плати . На Рис. 3.8 наведено схему підключення датчиків до роз’єму.

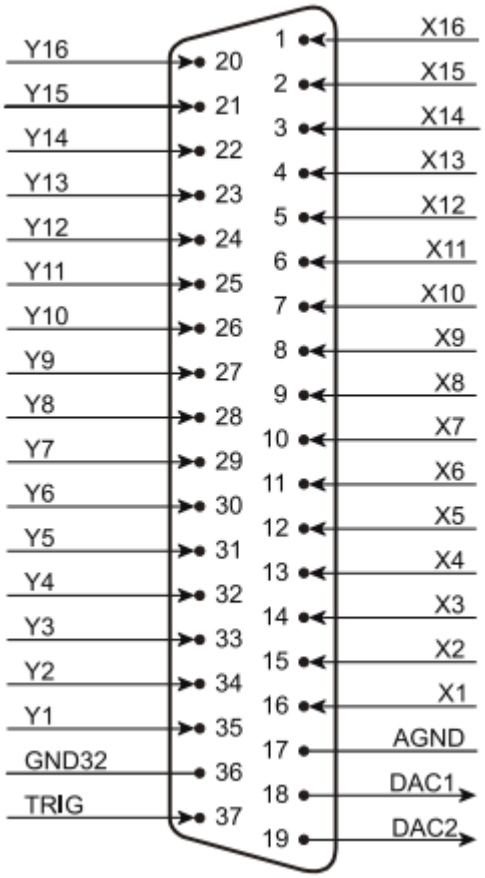


Рис. 3.8 роз’єм аналогового вводу

X1…X16 – перша група датчиків;

Y1…Y16 – Друга група датчиків;

GND32 – загальний інвертуючий вхід каналів 1…32. Призначений для вирівнювання потенціалів на нульових проводах ліній.

Для забезпечення безперервної роботи системи необхідно попіклуватись про захист комп’ютера від стрибків напруги та короткочасного зникнення живлення. Ця проблема вирішується підключенням живлення комп’ютера не напряму від

мережі, а через блок безперебійного живлення. В якості блоку безперебійного живлення був обраний APC Back-UPS 650VA, який захистить комп'ютер від стрибків напруги та короткочасного зникнення живлення до 10 хвилин.

3.4 Розробка програмного забезпечення

В даній роботі розроблена програма автоматизованого контролю вібрації та діагностики, яка обчислює параметри вібрації з сигналів, перетворених за допомогою плати L-780M з швидкодіючим АЦП. Також був реалізований модуль передачі даних по протоколу Modbus TCP, що передає обчислені параметри вібрації в сторонні системи. Також створена Scada програма у середовищі WinCC, що приймає параметри по протоколу Modbus TCP.

3.4.1 Автоматизована система контролю вібрації та діагностики

Для можливості роботи з платою L-780M необхідно завантажити бібліотеку функцій «wlcomp.dll». Вона створена щоб спростити зв'язок додатку з драйверами.

Програмне забезпечення у спрощеному вигляді можна розділити по послідовності виконання функцій на такі пункти:

- Завантаження з файлу стандартних параметрів, уставок, коефіцієнтів, обмежень при відкритті програми;
- Залежно від обраного режиму роботи (симуляція або без симуляції) виконання програми відрізняється, тим що в режимі симуляції є можливість налаштувати отримання даних вібрації без безпосереднього підключення до датчиків. Розглянемо режим роботи без симуляції;
- Ініціалізація плат L-780M (підключення бібліотек, налаштування зв'язку);
- Синхронізація вимірів та запуск циклічного опитування;
- Обрахунок вібропараметрів математичними методами;
- Передача виміряних параметрів за допомогою протоколу Modbus TCP;

- Виявлення дефектів по показам вібродатчиків

3.4.2 Завантаження з файлу параметрів з файлової системи

У кожній програмі щоб використовувати деякі зміні, функції для початку їх потрібно ініціалізувати, тобто об’явити класи, атрибути, змінні, прописати конструктори, деструктори в процесі опису будуть відокремлені елементи на які потрібно звернути увагу, але програмний продукт в повному обсязі описати в даній роботі не вдасться через занадто великий обсяг інформації. Нижче будуть представлені деякі з функцій з коротким описом.

При завантаженні головної форми виконуються функція, що виконує завантаження конфігурації останнього агрегату з файлової структури звертаючись до функції:

```
public bool LoadConfiguration(string DirName, out string Errors)
{... }
```

Яка отримує в якості аргументу шлях до файлової структури та встановлює параметри системи, що збереглись після останнього закриття програми. Під час завантаження встановлюються небезпечні границі вібропараметрів (уставки) при різних режимах роботи системи, конфігурація плати, налаштування сигналів.

3.4.3 Ініціалізація плати

Після завантаження конфігурації системи при запуску вібродіагностики в режимі без симуляції в першу чергу необхідно ініціалізувати плату та синхронізувати вимірювання вібропараметрів.

Але перед цим обов’язково потрібно об’явити бібліотеку та всі необхідні функції для роботи з нею. Далі наведений приклад об'явлення декількох функцій бібліотеки, а саме: створення екземпляру об’єкту для визначеного слоту , тест плати та функція установки події в драйвері.

- ```
  [DllImport("wlcomp.dll")]
  private static extern uint CallCreateInstance (ref uint HDll, uint Slot, ref uint Err);
  ```
  ```

(функція створює об’єкт для конкретного слоту, тип об’єкту визначається автоматично)

- *[DllImport("wlcomp.dll")]*
  private static extern uint PlataTest(ref uint PlataHandle);
  (тест на наявність плати та успішне завантаження)
- *[DllImport("wlcomp.dll")]*
  private static extern uint SetLDeviceEvent(ref uint PlataHandle, IntPtr hEvent, uint EventId);
  (функція для встановлення події в драйвері. В програмі використовується для сповіщення готовності даних при однократному заповненні буферу пам’яті.)

Це лише частина використаних функцій. Загалом використано 19 функцій з бібліотеки для роботи з платою.

Перед початком роботи необхідно її ініціалізувати. Основні завдання цієї функції:

- створення інтерфейсу плати:
- завантаження початкових параметрів налаштувань кількості каналів виміру та кількості вимірів на канал для виділення оперативної пам’яті в комп’ютері (перемноження кількості вимірів в каналі на кількість каналів):
- Заповнення каналу синхронізації (задання типу синхронізації, режиму, каналу та порогу синхронізації):

  Приклад функції ініціалізації плати наведено в додатку Б1

3.4.4 Синхронізація плати

Для коректного зчитування параметрів необхідно синхронізувати зчитування параметрів вібрації по сигналу покажчика обертів коли вал знаходиться у крайньому верхньому положенні. Канал покажчика обертів під номером «1» і вибраний по замовченню, тобто запис даних після кожного проходження валу одного повного оберту. Режим синхронізації Зверху-вниз чи Знизу-вверх визначає в який момент часу відбувається вимір параметрів (при початку фіксації одного повного оберту валу чи зразу після фіксації). Поріг синхронізації визначає граничне

значення у Вольтах після якого вважається, що надійшов сигнал про повний оберт валу (за замовчуванням – 2,5 В).

Після того, як початкова синхронізація встановлена обчислюється необхідна сумарна частота опитування датчиків виходячи зі значення кількості каналів та частоти опитування каналу.

До плати прив'язується подія, що сигналізує про заповнення буферу пам'яті. Вона викликається кожен раз коли заповнюється буфер даних, так ми дізнаємось що дані зібрані і вже передані у оперативну пам'ять комп'ютера звідки зчитуються та оброблюються всі параметри.

Далі виконується функція вводу сигналів з плати L-780 в якій ініціалізуються обраховані раніше параметри синхронізації, реалізується зчитування даних, після чого викликається функція ініціалізації старту, якщо встановились параметри буферу пам'яті:

3.4.5 Обрахунок вібропараметрів

Для обрахунку вібропараметрів опор методом швидкого перетворення Фур'є використовується функція:

```
public void CalculateOporVibroParam(OporSensor Sensor, bool AlarmFlag);
```

в яку передається масив вимірів віброшвидкості за один оберт валу. Далі описується тіло функції.

Змінні, які необхідно обрахувати в даній функції:

Ve – ефективна віброшвидкість;

FirstSpeedGarmonic – перша гармонійна швидкість;

PhaseFirstSpeedGarmonic – фаза першої гармонійної швидкості;

SecondSpeedGarmonic – друга гармонійна швидкість;

PhaseSecondSpeedGarmonic – фаза другої гармонійної швидкості;

Це частина змінних, за якими слідкує система для виявлення несправностей в роботі турбоагрегату.

Далі виконується підготовка масиву з N відліків вихідних даних амплітуди в який будуть вміщенні обраховані спектральні комплексні амплітуди.

Функція прямого перетворення Фур’є має вигляд приймає число відліків та масив комплексних експонент.

Далі виконується обчислення для особливих режимів інтегральних характеристик Ve (ефективної швидкості вібрації) - для опор та S() - для валу:

Використовується Фільтр спектру, обраховується Функція оберненого перетворення Фур’є.

Далі виконується обчислення амплітуди і фази спектру, обчислення розмаху через інтегрування даних по V (в масиві Signaldouble) після Фур'є перетворення.

Викликається функція обчислення розмаху, визначення фази першої гармоніки, визначення фази другої гармоніки.

Після цього відбувається визначення амплітуди першої, другої та інших гармонік, розрахунок високочастотної вібрації, розрахунок низькочастотної вібрації, розрахунок розмаху НЧВ. Схожим чином обраховуються змінні для датчиків валу.

В додатку Б2 наведено реалізацію всіх функцій обрахунку вібропараметрів по показам датчиків вібрації опор.

3.4.6 Реалізація протоколу Modbus TCP

Для реалізації протоколу обміну даними Modbus TCP було створено окремий клас Modbus. У класі ініціалізується стандартний порт для обміну даними по Modbus TCP(номер порта 502). Наступним кроком є створення Modbus сервера. В обробнику подій кнопки запуску системи викликається функція Modbus.Start().

Під час виклику функції Start() сервер приймає значення порта, адреси пристрою, виділений масив вхідних регістрів та функцію Listen(), що очікує запити на передачу даних з вказаними параметрами.

В обробнику подій кнопки виключення програми викликається функція Modbus.Stop(), шо припиняє передачу даних з закриттям порта.

У головному циклі програми викликається функція:

Modbus.ValuesToModbusRegisters(FAgt);

Ця функція отримує в якості аргументу агрегат, який містить в собі параметри призначені для передачі мережею. У функції Modbus.ValuesToModbusRegisters() спочатку перевіряється чи містить агрегат значення, потім присвоюється початкова адреса змінної, після чого дані для датчиків опори групуються по номеру підшипника, та присвоюються значенню input регістра.

Така сама операція виконується для передачі даних по датчику валу. Детально програмний код описаний у додатку Б3.

3.4.7 Виявлення дефектів

Для виявлення дефектів існує окремий клас CrackDiagnostic. В ньому описується класи для діагностичної ознаки та клас опису дефекту:

Також в ньому описані класи відкриття та закриття діагнозу несправності по наявності дефекту, що продовжується деякий період часу, а бо вже не виникає деякий період часу.

Функція, що контролює вібропараметри для виявлення несправностей називається CheckDefects(). Далі буде описано частину її реалізації.

Оскільки, турбоагрегат в різних режимах роботи підтримує різні частоти оберту валу, то для різних режимів необхідно задавати різні уставки вібропараметрів, крім того при одному режимі роботи може виникати такий дефект, який непомітний або незначний при іншому. Саме через це необхідно слідкувати в якому саме режимі зараз знаходиться турбоагрегат за показами датчика частоти оберту валу.

За допомогою оператора switch() який приймає аргумент режиму роботи у функції CheckDefects() визначається в якому саме режимі роботи турбоустановки відбувся дефект:

- Nominal – режим навантаження;
- ValRotation – режим валоповороту;
- Up – режим набору обертів;
- Stop – режим зупинки.

При завантаженні системи з файлу конфігурації записуються уставки для всіх параметрів вібродіагностики для кожного дефекту кожного режиму навантаження. Ці уставки можна змінити з інтерфейсу користувача.

Після визначення режиму роботи для кожного дефекту викликається функція для його ініціалізації:

```
private bool CalculateDefectDiagnosis();
```

Вона визначає де саме виміряні параметри вищі за межу уставки. На місті крапок в операторі в кінцевому результаті в ньому викликається функція:

```
private bool CheckDiagnosisToken (DiagnosisToken Token, int ChannelNumber);
```

Ця функція остаточно визначає параметр, шо вийшов за допустимі межі. Повне представлення програмного продукту для визначення несправностей і повний вміст функції дивись в Додатку Б4.

3.4.7 Реалізація Scada системи

На базі програмного забезпечення TIA Portal та Simatic WinCC була розроблена Scada система, що приймає обчислені параметри ефективної швидкості вібрації (для опор), або розмах віброперемiщення (для валу) з промислового комп’ютера по протоколу Modbus TCP. Ці параметри передають поточний стан об’єкту в режимі реального часу.

При перевищенні уставок параметрів вібрації на небезпечну величину виникає сигнал аварії, що сповіщає оператора.

Головні вікна з параметрами вібрації з датчиків опор та валів системи диспетчеризації зображені на Рис. 3.8, 3.9 відповідно.

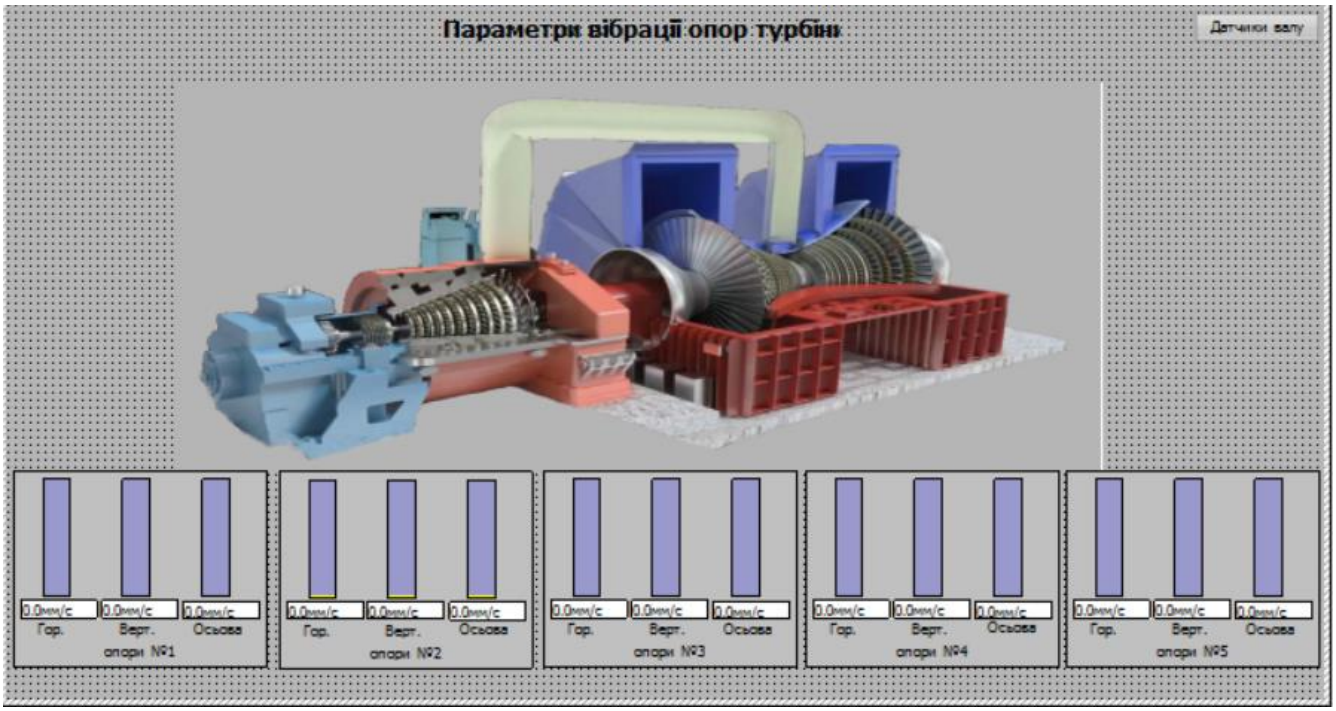


Рис. 3.8 Вікно відображення параметрів вібрації опор турбіни

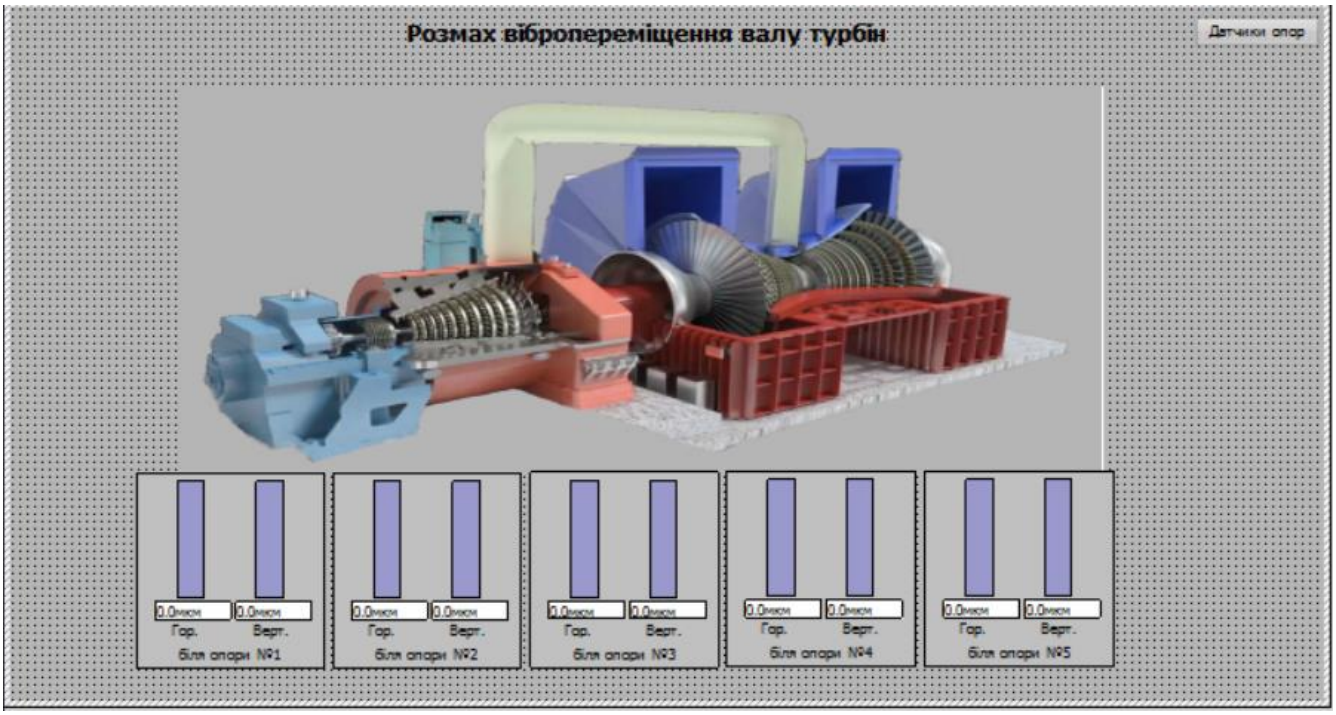


Рис. 3.9 Вікно відображення параметрів вібрації валу турбіни

3.5 Імітаційне моделювання і аналіз функціонування автоматичного комплексу

Для аналізу роботи програмного – технічного комплексу АСКВД можна скористатися функцією симуляції сигналів у самому програному забезпеченні (див. Рис 3.10)

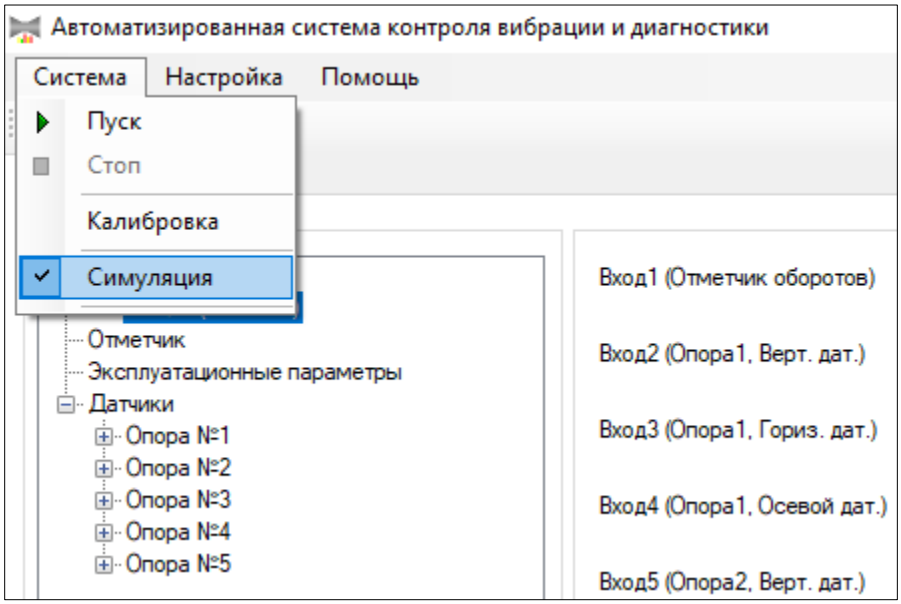


Рис. 3.10 Увімкнення режиму симуляції

В цьому режимі при конфігурації системи представляється можливість задавати закон зміни сигналів по кожному з каналів виміру. На Рис. 3.11 відображено вікно налаштувань параметрів симуляції сигналів.

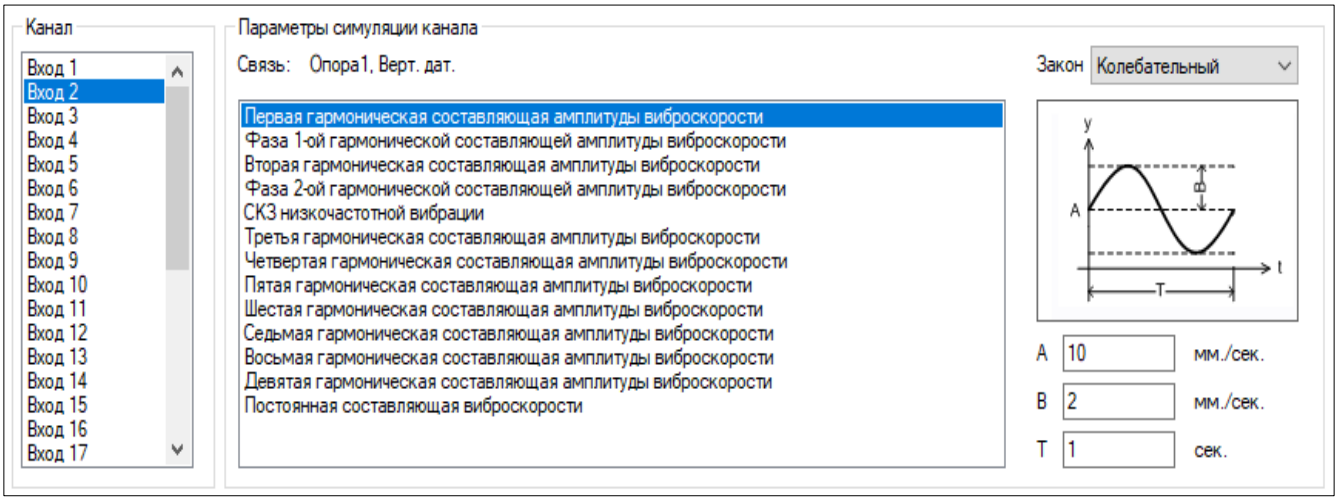


Рис. 3.11 Вікно налаштувань параметрів симуляції

Користувач має можливість обрати наступні закони зміни сигналів:

- Постійне значення;
- Переключення;
- Коливальний;
- Лінійний.

Для закону «постійне значення» змінна приймає задану користувачем постійну величину.

Для закону «переключення» користувач задає величину та два параметри часу, через який буде відбуватись переключення з нульового положення на задану величину та навпаки.

Для закону «коливальний» задається величина параметру, амплітуда та період зміни параметру.

Для закону «лінійний» задається початкова величина параметру, кінцева величина параметру та час за який вона має змінюватись.

Не дивлячись на те, що в програмному комплексі реалізовано зручне налаштування симуляції, що дозволяє в повній мірі перевірити працездатність комплексу будо прийнято рішення провести перевірку працездатності за допомогою підключення генератора синусоїдальних імпульсів GAG-810 напряму до плати аналогового вводу L-780M. Він здатен генерувати синусоїдальний сигнал в діапазоні частот від 10 Гц до 1 МГц з можливістю зміни параметрів частоти та амплітуди сигналу.

Для синхронізації плати необхідно, щоб на перший канал виміру був підключений лічильник обертів, згідно його сигналу починати вимірюватись параметри з інших датчиків вібрації. Для моделювання сигналу лічильника обертів використовується контрольний прилад АСКВД дільник напруги. З сигналу генератора імпульсів за допомогою дільника напруги «обрізається» додатнє

значення амплітуди, та подається, як сигнал оберту валу на перший вхід плати L-780M.

Для імітаційного моделювання необхідно підключити всі прилади до мережі живлення та з’єднати їх між собою інформаційні канали. Після цього необхідно запустити програмний додаток на комп’ютері (Див. Рис. 3.13).

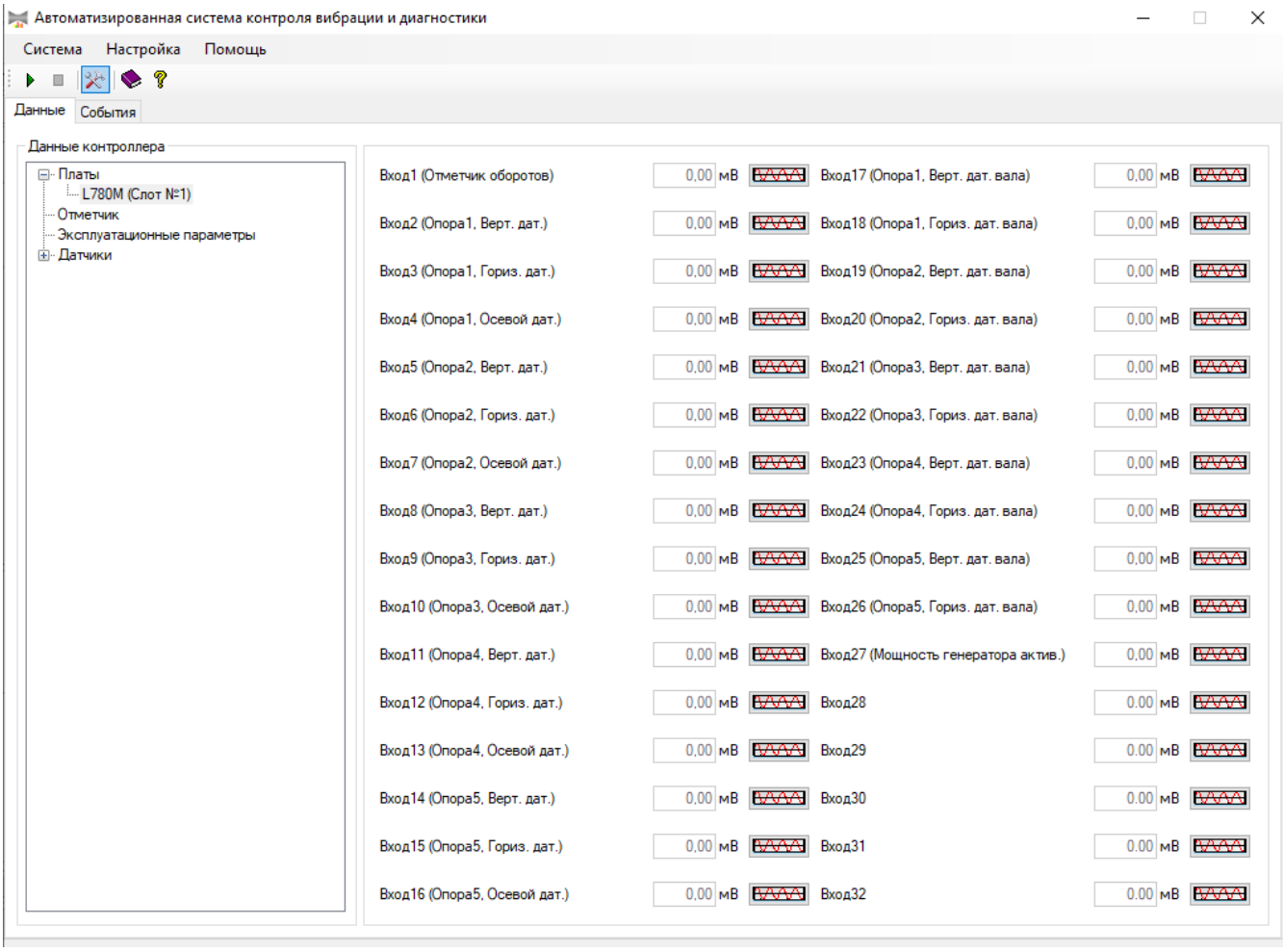


Рис. 3.12 Головне вікно програми

Після запуску програми система починає отримувати та обробляти інформацію з лічильника обертів та імітатора синусоїдальних сигналів. Для перегляду величин вібропараметрів зліва в дереві проекту відкриваємо вложення «датчики» та обираємо параметр, що нас цікавить (Див. Рис. 3.13). У вікні, що відкрилося можна побачити значення вібропараметрів та графік вхідного сигналу.

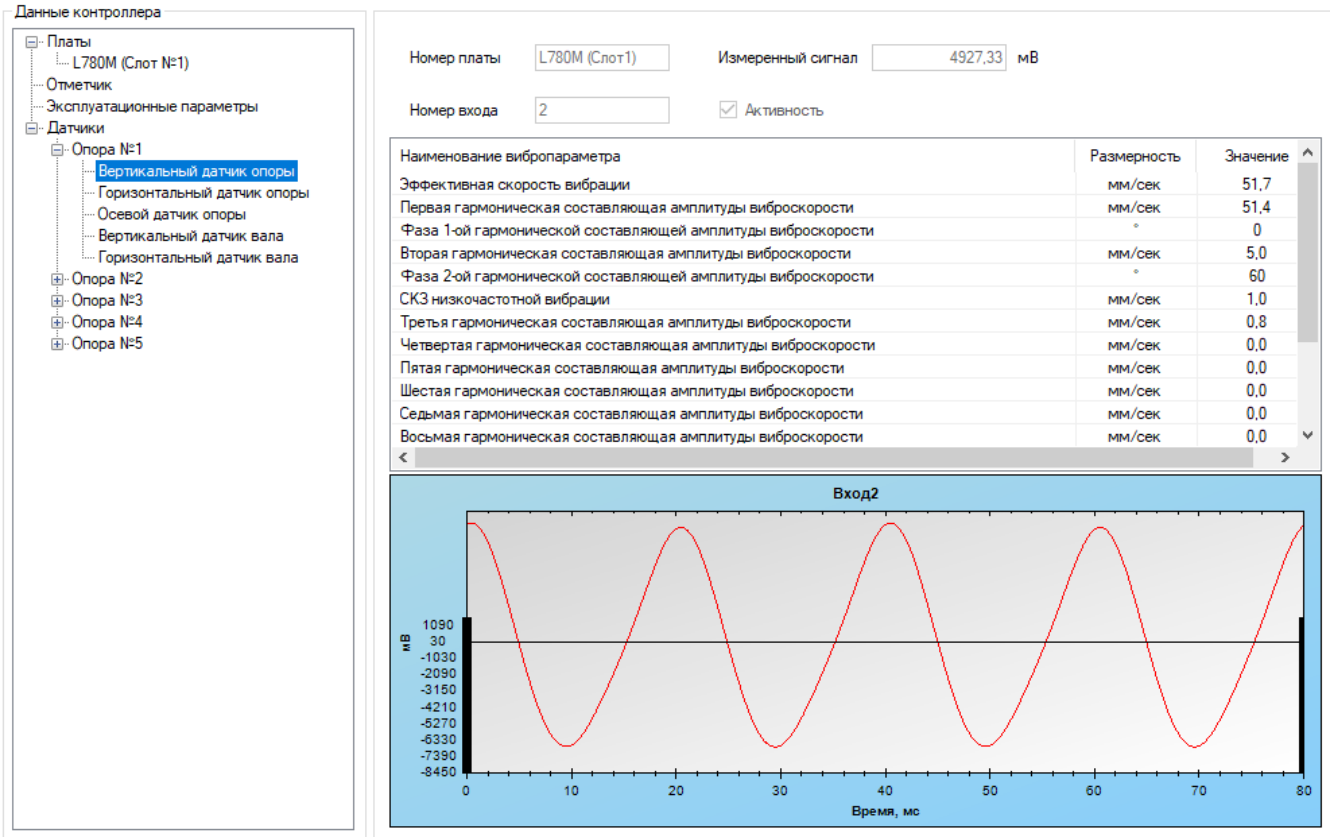


Рис. 3.13 Відображення вібропараметрів, обчислених з сигналу вертикального датчику опори.

Для визначення несправностей необхідно перейти в додаток «Дії» програми на головному екрані. На Рис. 3.14 зображено ряд несправностей, що виявились під час зміни параметру сигналу з генератора імпульсів.

11.12.2019 2:23:19 Режим работы - Номинальный
11.12.2019 2:23:19 Начало опроса плат ввода/вывода
11.12.2019 2:23:25 Внимание!!! Обнаружен дефект - Скачок вибрации.
11.12.2019 2:23:25 Внимание!!! Обнаружен дефект - Аварийная вибрация.
11.12.2019 2:23:25 Внимание!!! Обнаружен дефект - Рост вибрации с частотой 100 Гц.
11.12.2019 2:23:25 Внимание!!! Обнаружен дефект - Тепловой прогиб ротора генератора.
11.12.2019 2:23:25 Внимание!!! Обнаружен дефект - Механические неплотности в муфте.
11.12.2019 2:23:25 Внимание!!! Обнаружен дефект - Прогиб вала.
11.12.2019 2:23:25 Внимание!!! Обнаружен дефект - Дисбаланс ротора.
11.12.2019 2:23:25 Внимание!!! Обнаружен дефект - Превышение предупредительной уставки по валу.
11.12.2019 2:23:25 Внимание!!! Обнаружен дефект - Расцентровка опор.
11.12.2019 4:23:19 Обновление системного времени. Это может занять несколько секунд. Пожалуйста, подождите.
11.12.2019 4:23:19 Адрес несовместим с выбранным протоколом ::1:123

Рис. 3.14 Вікно сповіщення оператора.

В Scada- системі параметри вібрації зображені у вигляді стовпців, заповнених кольором (Рис. 3.16). Якщо параметр вібрації перевищує задане крайнє значення, то колір змінюється на жовтий або червони, та формується сигнал аварії (Рис. 3.17).

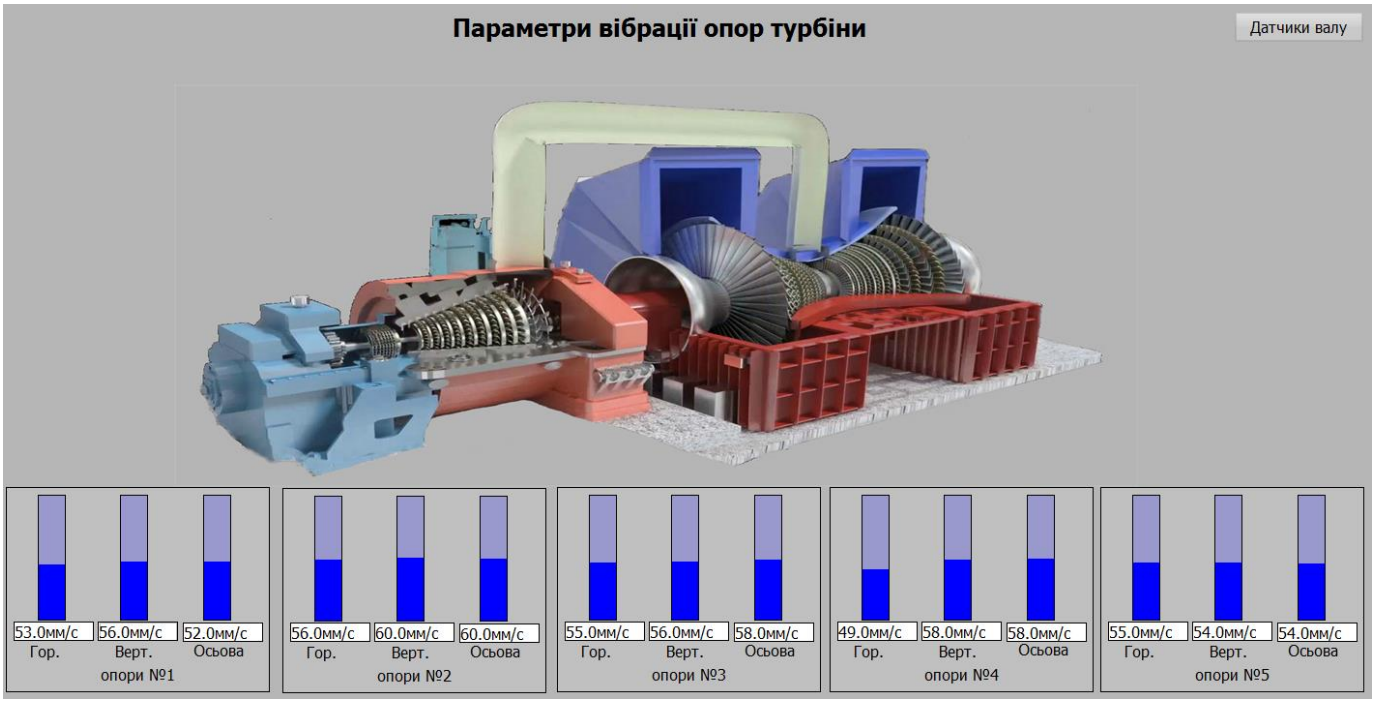


Рис. 3.15 Нормальне значення параметрів вібрації.

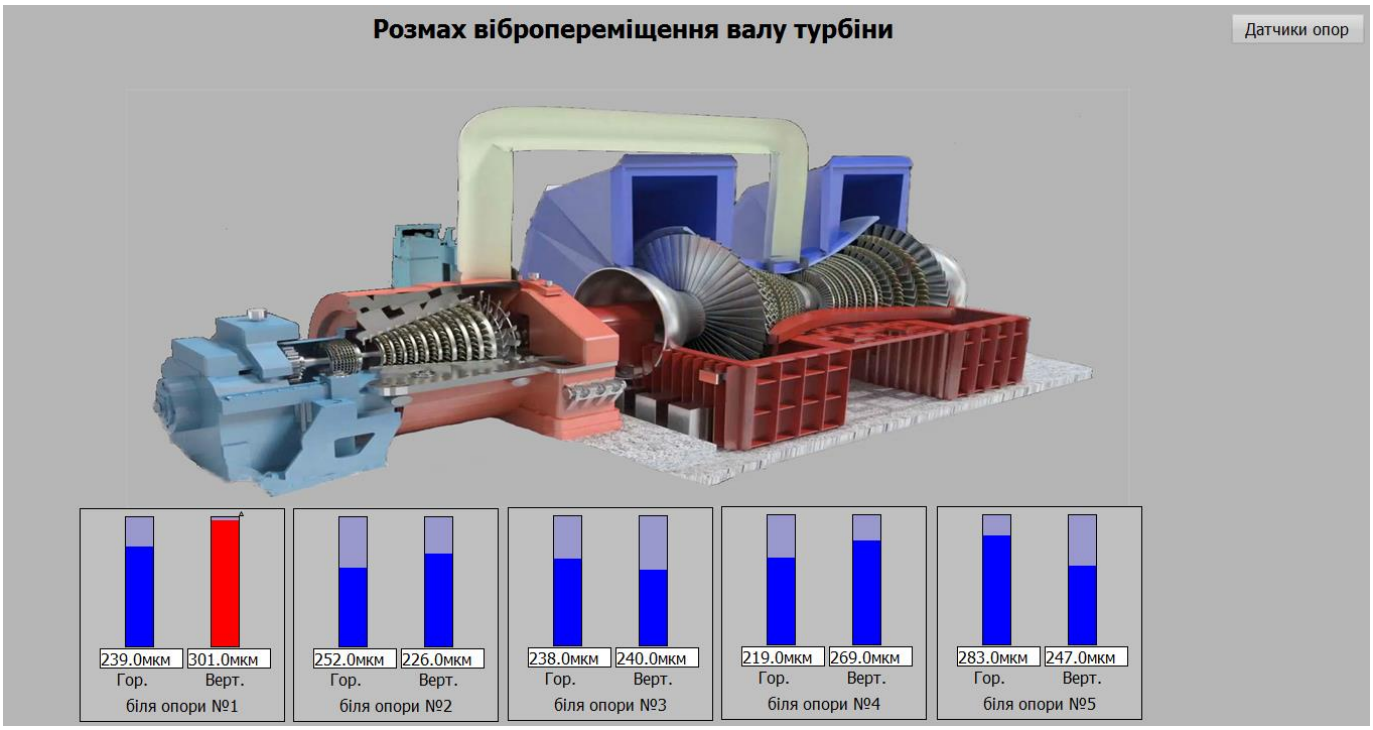


Рис. 3.16 Вихід параметра вібрації за граничну межу.

4. Розроблення стартап-проекту

5.1 Зміст ідеї

Таблиця 4.1. Опис стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Розробка системи вібродіагностики турбоагрегату.	1. ТЕС	- Підвищення надійності, терміну експлуатації обладнання; - зниженні витрат на ремонтні роботи через забезпечення неперервної роботи (якщо не виявлено несправностей); - підвищенні ефективності ремонтних робіт (зниження часу на ремонт обладнання в наслідок розуміння проблеми виходячи з оброблених даних);
	2. ГЕС	
	3. АЕС	

3.1. Визначення характеристик ідеї проекту

Таблиця 5.2. Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№	*Техніко-економічні характерис-тики ідеї*	*Продукція конкурентів*		*W (слабка сторона)*	*N (нейтральна сторона)*	*S (сильна сторона)*
		Мій проект	*Компакс (*СКМВТ*)*			
1.	Інформування оператора про можливі	Визначає можливі несправності вібропарамет рах системи	Не передбачено	-	Відсутність досвіду впровадження	Економічність, ефективність витрати ре, малі затрати на впровадження

3.2. Технологічний аудит ідеї проекту

Таблиця 4.3. Технологічна здійсненність ідеї проекту

№	*Ідея проекту*	*Технології її реалізації*	*Наявність технологій*	*Доступність технологій*
1.	Діагностика вібрації	Контролерна автоматизація	Широко представлена на ринку	Доступна
Названі технології є доступні, наявні на ринку у різних цінових категоріях.				

3.3. Аналіз ринкових можливостей запуску стартап-проекту

Таблиця 4.4. Попередня характеристика потенційного ринку стартап-проекту

№ п/п	*Показники стану ринку (найменування)*	*Характеристика*
1	Кількість головних гравців, од	30
2	Загальний обсяг продаж, грн/ум.од	730 млн.
3	Динаміка ринку (якісна оцінка)	зростаюча
4	Наявність обмежень для входу (вказати характер обмежень)	Специфіка технологічного обладнання та норм
5	Специфічні вимоги до стандартизації та сертифікації	Є загальні вимоги до стандартизації та сертифікації
6	Середня норма рентабельності в галузі (або по ринку), %	50

За попередніми оцінками на основі таблиці 5.4, можна зробити висновок, що ринок є привабливим для входження. Доказами цього є порівняння середньої рентабельності з банківськими відсотками на вкладення за результатами якого

рентабельність проекту перевищує банківські вкладення, отже є сенс вкласти гроші в цей проект. Ще одним з переваг малі витрати на впровадження, проблеми стандартизації та сертифікації є не суттєвими, як і наявності обмежень для входу.

Таблиця 4.5. Характеристика потенційних клієнтів стартап-проекту

№ п/п	*Потреба, що формує ринок*	*Цільова аудиторія (цільові сегменти ринку)*	*Відмінності у поведінці різних потенційних цільових груп клієнтів*	*Вимоги споживачів до товару*
1	Завчасне виявлення загроз роботі обладнання	споживачі	Впровадження європейських стандартів на параметри ГВС. Зростаючі ціни на енергоносії.	до продукції

Таблиця 4.6. Фактори загроз

№ п/п	*Фактор*	*Зміст загрози*	*Можлива реакція компанії*
1	Відсутність досвіду впровадження	В Україні відсутні системи з прогнозуванням відмов. Існують слідкуючі.	Запозичення закордонного досвіду або впровадження власних досліджень з залученням кращих фахівців

Таблиця 4.7. Фактори можливостей

№ п/п	*Фактор*	*Зміст можливості*	*Можлива реакція компанії*
1	Удосконалення системи діагностики несправностей	Передача обрахованих параметрів діагностики до допоміжних систем.	Більш повне вивчення параметрів об’єкту та поведінки

Таблиця 4.8. Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	*В чому проявляється дана характеристика*	*Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)*
1. Тип конкуренції – нецінова	Підвищення якості обслуговування обладнання.	Підвищення продуктивності.
2. За рівнем конкурентної боротьби - інтернаціональний	Така проблема наявна закордоном	Впровадження запропонованого методу
3. За галузевою ознакою - внутрішньогалузева	Різна якість у реалізації товару	Контрольоване управління
4. Конкуренція за видами товарів: - товарно-родова - товарно-видова	Є методи діагностики без аналізу даних. (тільки сигналізація при перевищені уставки)	Використання більш ефективного методу вібродіагностики
5. За характером конкурентних переваг - нецінова	Звернення до споживачів, які платили б за якість	Даний метод для цього і впроваджується, він вже має на меті максимального збільшення якості регулювання

Таблиця 4.9. Аналіз конкуренції в галузі за М. Портером

Складов і аналізу	*Прямі конкуренти в галузі*	*Потенційні конкуренти*	*Постачальники*	*Клієнти*	*Товари-замінники*
	«КОМПАКС»	*Розмір капіталовкладень, доступ до ресурсів*	*Визначити фактори сили постачальників*	*Визначити фактори сили споживачів*	*Фактори загроз з боку замінників*
Висновки:	Невелика конкурентної боротьби з боку прямих конкурентів	- є можливості входу в ринок - потенційних конкурентів немає	Диктують умови роботи на ринку	Чутливість до змін цін, контроль якості	Немає загроз з боку замінників

З огляду на конкурентну ситуацію є принципово можливість роботи на ринку але з деякими труднощами. Так як немає потенційних конкурентів. Для збільшення конкурентної спроможності і диктування умов на ринку, проект повинен мати контроль якості, інформаційне забезпечення.

Таблиця 4.10. Обґрунтування факторів конкурентоспроможності

№ п/п	*Фактор конкурентоспроможності*	*Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)*
1	Складність діагностики несправностей по значенням вібропараметрів.	Відсутність єдиного рішення для ефективної діагностики обладнання.
2	Доступність параметрів	Можливість використання допоміжних, хмарових систем діагностики чи збору даних.

Таблиця 4.11. Порівняльний аналіз сильних та слабких сторін «Маг. проект»

№ п/п	*Фактор конкурентоспроможності*	*Бали 1-20*	*Рейтинг товарів-конкурентів у порівнянні з ГВТОСВТО*						
			–3	*–2*	*–1*	*0*	*+1*	*+2*	*+3*
1	Складність діагностики	20	+						
2	Доступність параметрів	15	+						

Таблиця 4.12. SWOT – аналіз стартап-проекту

Сильні сторони: – Енергозбереження – Якість обслуговування – Малі затрати на впровадження	Слабкі сторони: – Відсутність досвіду впровадження
Можливості: – Удосконалення системи діагностики	Загрози: - Не велика зацікавленість в інноваційному підході

Таблиця 4.13. Альтернативи ринкового впровадження стартап-проекту

№ п/п	*Альтернатива (орієнтовний комплекс заходів) ринкової поведінки*	*Ймовірність отримання ресурсів*	*Строки реалізації*
1	Власник підприємства	15 %	1 рік
2	Комплексна державна програма по екології	20%	2-3 роки
3	Стандартизація вихідних параметрів	50%	2 роки

Обираємо стандартизацію вихідних параметрів, як найбільш альтернативну. Так як вона має середні показники ймовірності отримання ресурсів та строки реалізації.

Таблиця 4.14. Вибір цільових груп потенційних споживачів

№ п/п	*Опис профілю цільової групи потенційних клієнтів*	*Готовність споживачів сприйняти продукт*	*Орієнтовний попит в межах цільової групи (сегменту)*	*Інтенсивність конкуренції в сегменті*	*Простота входу у сегмент*
1	ТЕЦ, ТЕС, АЕС	Готові але є деякі труднощі	Практично всі енергогенеруючі компанії турбінами	Не велика	Не легка
2	Промислові підприємства	Готові але є деякі труднощі	Практично всі енергогенеруючі компанії з турбінами	Не велика	Не легка

Таблиця 4.15. Визначення базової стратегії розвитку

№ п/п	*Обрана альтернатива розвитку проекту*	*Стратегія охоплення ринку*	*Ключові конкурентоспроможні позиції відповідно до обраної альтернативи*	*Базова стратегія розвитку**
1	Стандартизація вібропараметрів	Новизна методики	Складність діагностики, Доступність параметрів.	Стратегія лідерства по якості

Таблиця 4.16. Визначення базової стратегії конкурентної поведінки

№ п/п	*Чи є проект «першопрохідцем» на ринку?*	*Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?*	*Чи буде компанія копіювати основні характеристики товару конкурента, і які?*	*Стратегія конкурентної поведінки**
1	Проект є «першопрохідцем» на ринку	Компанія буде забирати існуючих споживачів у конкурентів	Компанія не буде копіювати основні характеристики товару	Стратегія виклику лідера По якості

Таблиця 4.17. Визначення стратегії позиціонування

№ п/п	*Вимоги до товару цільової аудиторії*	*Базова стратегія розвитку*	*Ключові конкурентоспроможні позиції власного стартап-проекту*	*Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (дві ключових)*
1	Якість	Стратегія лідерства по якості	Складність діагностики, Доступність параметрів.	1. Економічність; 2. Мінімальні витрати на впровадження.

Таблиця 4.18. Визначення ключових переваг концепції потенційного товару

№ п/п	*Потреба*	*Вигода, яку пропонує товар*	*Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)*
1	Якість діагностування	Споживач отримує товар в бажаному вигляді	Поради щодо обслуговування турбоагрегату.
2	Економічність	Економія часу на плановий ремонт	Відпадає необхідність виконувати планові ремонти турбогенератора, можна виконувати ремонт з поради програмного забезпечення з меншою періодичністю.

Таблиця 4.19. Визначення меж встановлення ціни

№ п/п	*Рівень цін на товари-замінники*	*Рівень цін на товари-аналоги*	*Рівень доходів цільової групи споживачів*	*Верхня та нижня межі встановлення ціни на товар/послугу*
1	Вище середнього	Вище середнього	Середній	-Прожитковий мінімум

Таблиця 5.21. Формування системи збуту

№ п/п	*Специфіка закупівельної поведінки цільових клієнтів*	*Функції збуту, які має виконувати постачальник товару*	*Глибина каналу збуту*	*Оптимальна система збуту*
1	Якість продукції	Своєчасна реалізація	Висока	Вертикальна

Таблиця 5.22. Концепція маркетингових комунікацій

№ п/п	*Специфіка поведінки цільових клієнтів*	*Канали комунікацій, якими користуються цільові клієнти*	*Ключові позиції, обрані для позиціонування*	*Завдання рекламного повідомлення*	*Концепція рекламного звернення*
1	*Зменшення витрат на ремонт*	*Реклама*	Якість діагностування Економічність	*Донесення ідеї до споживача*	*Забезпечення результату без дод. витрат*

Висновки

Під час виконання магістерської дисертації на тему «Система діагностики відмов та автоматизованого обслуговування турбоустановки» довелось зіткнутись з великим об’ємом нової для мене інформації, щодо особливостей віброконтролю. Вивчаючи технічну та наукову літературу з даної теми було досліджено особливості та проблематика обслуговування турбоустановок, основні методи вібродіагностики, перелік несправностей турбоустановок, що можна отримати шляхом обробки даних з датчиків вібрації. Досліджено основні проблеми обміну, обробки та збереження інформації в існуючих системах.

Як об’єкт діагностики в даній роботі розглядається турбіна парова турбіна Т-160-130 для використання на теплових електростанціях. На її основі було розроблено та реалізовано програмно-технічний комплекс вібродіагностики, що дозволяє виявити несправність у роботі турбіни за допомогою обчислення параметрів з датчиків вібрації, встановлених безпосередньо на агрегаті.

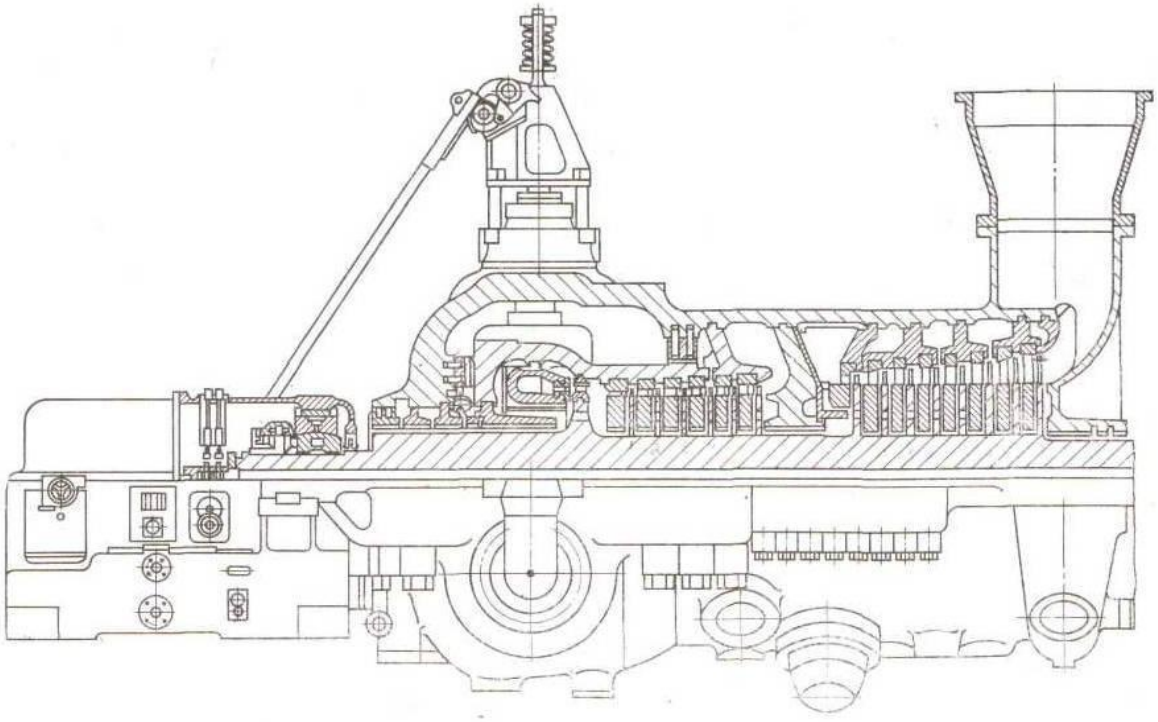
В даній роботі більшість уваги приділялося саме програмним рішенням обробки даних та технічному оснащенню системи діагностики. В якості контролера була обрана плата вводу аналогових сигналів з швидкодіючим АЦП на базі промислового комп’ютера. Вона має в декілька разів вищу розрядність ніж аналогічні модулі вводу аналогових сигналів промислових контролерів.

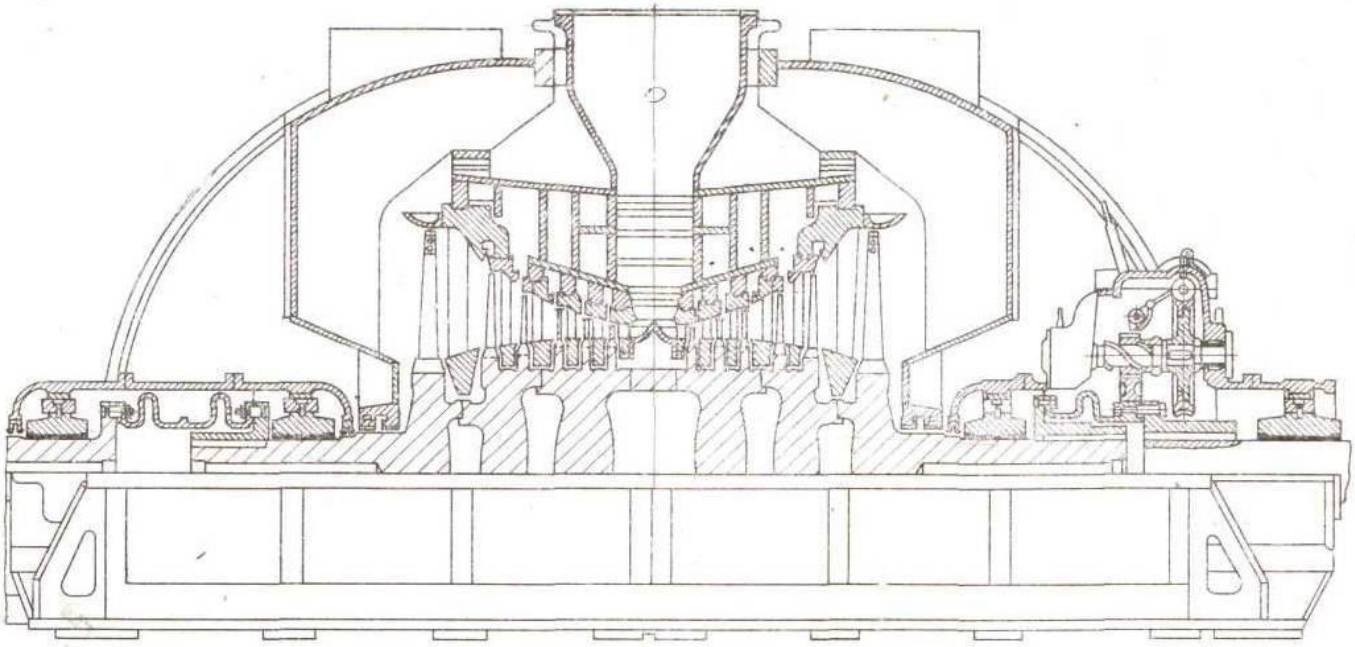
Розроблена система діагностики несправностей турбоагрегату під час імітаційного моделювання показала гарний результат. Вона реагує на зміну параметру вхідного сигналу, обчислює додаткові параметри, передає ці параметри в мережу обміну даними, виявляє дефекти роботи турбоагрегату та інформує про це оператора.

Перелік літератури

1. Резник Д.О., Поліщук І.А., Костанецький К.В. СИСТЕМА ВІБРОДІАГНОСТИКИ, ПРОГНОЗУВАННЯ ВІДМОВ ТА ОЦІНКИ СТАНУ ТУРБОАГРЕГАТУ. Молодий вчений
2. 10.4 Вібрація турбоагрегату //Сайт присвячений атомним станціям URL: http://www.aes.pp.ua/46.php (дата звернення 22.11.2019).
3. Гольдин А.С. Вибрация роторных машин. Москва «Машиностроение» 1999
4. Петрухін В.В., Петрухін С.В. Основи вібродіагностики та засоби вимірювання вібрації. Москва, 2010. 30 с.
5. Медведєв С.Ю. Перетворення Фур’є і класичний цифровий спектральний аналіз. URL: http://www.vibration.ru/preobraz_fur.shtml (дата звернення 24.11.2019).
6. Браун, Датнер. Аналіз вібраций роликовых і шариковых подшипников: Пер. с англ.- Конструирование и технология машиностроения.- М.: Мир, 1979.-т. 101, №1.-с.65-82.
7. Ф.М.Дименейберга и К.С.Колесникова. Вибрации в технике: Справочник.- т. 31/ Под ред. - М.: Машиностроение, 1980.-544с.
8. Русов В.А. "Диагностика дефектов вращающегося оборудования по вибрационным сигналам" 2012 г.
9. XVIII Международный научно-практической конференции молодых ученых и студентов, 23-26 апреля 2019 года. В 2 т. ст. 28

Додаток А. Креслення турбіни К-160-130 у розрізі





Додаток Б1. Ініціалізація плати L-780M

```
public bool Init(ref uint DLLHandle, int DigitalCounter, int PeriodCount, out string
Errors)                        // Инициализация платы
        {
            uint Err = 0;
            //double[] ChannelDelay = new double[4];
            Errors = "";
            FKadr = 0;
            FInitialized = false;
            FDigitalCounter = DigitalCounter;  //количество измерений
            FPeriodCount = PeriodCount;
            FPlatHandle = CallCreateInstance(ref DLLHandle, (uint)(FSlot - 1), ref
Err);

            if (FPlatHandle == 0)
            {
                Errors = "Ошибка создания интерфейса платы";
                FInitialized = false;
                return false;
            }

            if (OpenLDevice(ref FPlatHandle) == 0)
            {
                Errors = "Ошибка подключения к плате";
                return false;
            }

            string Bios;

            switch (FType)
            {
                case "L780M": Bios = "l780"; break;
                case "L780": Bios = "l780"; break;
                default: Bios = "l780"; break;
            };

            if(LoadBios(ref FPlatHandle, Bios) == L_ERROR)
            {
                Errors = "Ошибка загрузки Bios";
                CloseLDevice(ref FPlatHandle);
                return false;
            }

            if (PlataTest(ref FPlatHandle) == L_ERROR)
            {
                Errors = "Ошибка теста платы!!!";
                CloseLDevice(ref FPlatHandle);
                return false;
            }

            PLATA_DESCR PDescr = new PLATA_DESCR();
            int Size = Marshal.SizeOf(typeof(PLATA_DESCR));
            IntPtr ptr = Marshal.AllocHGlobal(Size);
            Marshal.StructureToPtr(PDescr, ptr, false);
            if (ReadPlataDescr(ref FPlatHandle, ptr) == L_ERROR)
            {
```

```
                Marshal.FreeHGlobal(ptr);
                Errors = "Ошибка чтения флэш";
                CloseLDevice(ref FPlatHandle);
                return false;
            }

            int RunPtr = (int)ptr;
            FSerialNumber = Marshal.PtrToStringAnsi(ptr, 8);
            sbyte Rev = (sbyte)Marshal.ReadByte((IntPtr)(RunPtr + 14));
            FQuartz = Marshal.ReadInt32((IntPtr)(RunPtr + 20));
            Marshal.FreeHGlobal(ptr);

            WASYNC_PAR par = new WASYNC_PAR();
            Size = Marshal.SizeOf(typeof(WASYNC_PAR));
            par.Chn = new uint[128];
            par.Data = new uint[128];
            par.s_Type = L_ASYNC_TTL_OUT;
            par.Data[0] = 0xffff;
            ptr = Marshal.AllocHGlobal(Size);
            Marshal.StructureToPtr(par, ptr, false);
            // сбросить сигнализацию
            if (FWorkSignalization || FSignalization)
            {
                IoAsync(ref FPlatHandle, ptr); // сбросим реле сигнализации
                // условная компиляция только для плат L780M - с возможностью
управления
                // доступом к цифровому выводу
                if (((char)Rev != 'A') && ((char)Rev != 'B'))
                {
                    par.s_Type = L_ASYNC_TTL_CFG;
                    par.Mode = 1;
                    Marshal.StructureToPtr(par, ptr, false);
                    uint Res = IoAsync(ref FPlatHandle, ptr);
                    Res += 1;
                    PutWord_DM(ref FPlatHandle, L_ENABLE_TTL_OUT_PLX, 0x1); //
разрешим выходные линии TTL
                    SendCommand(ref FPlatHandle, cmENABLE_TTL_OUT_PLX);
                }
                VKL_SIG();
            }
            Marshal.FreeHGlobal(ptr);

            FKadrDelay = (double)1000 / (double)50 - (double)FChannelDelay *
(double)(FChannelsCount - 1) / (double)1000;
            FRate = (double)1000 / FChannelDelay;
            // обратный пересчет межканальной задержки для более точного определения
нижней границы
            // по оборотам исходя из максимально возможной межкадровой задержки
            FChannelDelay = (double)1000 / FRate;
            FBigBufSize = (uint)(FDigitalCounter * FChannelsCount); // выделение
буфера оперативной памяти в компьютере
            if (RequestBufferStream(ref FPlatHandle, ref FBigBufSize, L_STREAM_ADC)
== L_ERROR)
            {
                Errors = "Ошибка установки размера большого кольцевого буфера";
                CloseLDevice(ref FPlatHandle);
                return false;
            }
```

```
            WDAQ_PAR DaqIntern = new WDAQ_PAR();
            int Sz = Marshal.SizeOf(typeof(WDAQ_PAR));
            IntPtr Pointer = Marshal.AllocHGlobal(Sz);
            DaqIntern.s_Type = 1;
            DaqIntern.AutoInit = 0;
            DaqIntern.dRate = FRate;
            DaqIntern.dKadr = ((double)1000 / (double)50 - (double)FChannelDelay *
(double)(FChannelsCount - 1) / (double)1000) / 1000;
            DaqIntern.dScale = 0;
            DaqIntern.SynchroType = (uint)FSynchroType;
            DaqIntern.SynchroSensitivity = 1;
            DaqIntern.SynchroMode = (uint)FSynchroMode;
            DaqIntern.AdChannel = (uint)((FSynchroChannel - 1) | 0x20);
            DaqIntern.AdPorog = (uint)(2 * FMaxOut / FMaxV );
            DaqIntern.NCh = (uint)FChannelsCount;
            DaqIntern.FPDelay = 0;
            DaqIntern.Chn = new uint[128];

            for (int k = 0; k < FChannelsCount; k++)
                DaqIntern.Chn[k] = (uint)(k | 0x20);

            DaqIntern.FIFO    =    (uint)(FChannelsCount    *    FDigitalCounter    /
FPeriodCount);
            DaqIntern.IrqStep    =    (uint)(FChannelsCount    *    FDigitalCounter    /
FPeriodCount);
            DaqIntern.Pages = (uint)FPeriodCount;
            DaqIntern.AdcEna = 1;
            DaqIntern.IrqEna = 1;
            FKadrDelay = 1000 / 50 / ((double)FDigitalCounter / (double)FPeriodCount)
- 1 / FRate * (double)(FChannelsCount - 1);  // межкадровая задержка
            Marshal.StructureToPtr(DaqIntern, Pointer, false);

            if (FillDAQparameters(ref FPlatHandle, Pointer, 2) == L_ERROR)
            {
                Errors = "Ошибка установки параметров сбора данных";
                Marshal.FreeHGlobal(Pointer);
                CloseLDevice(ref FPlatHandle);
                return false;
            }
            else
            {
                int RunPointer = (int)Pointer;
                FFIFO      =      (uint)Marshal.ReadInt32((IntPtr)(RunPointer      +
sizeof(uint)));
                FIrqStep    =    (uint)Marshal.ReadInt32((IntPtr)(RunPointer    +    2    *
sizeof(uint)));
                FPages    =    (uint)Marshal.ReadInt32((IntPtr)(RunPointer    +    3    *
sizeof(uint)));
            }
            FPlataRate = (uint)(2 * FQuartz / 1000 / (2.0 * FRate) - 0.5);
            FKadr = (uint)((double)FChannelsCount / ((FChannelsCount - 1) / FRate +
FKadrDelay));
            FData = IntPtr.Zero;
            FSync = IntPtr.Zero;
            FUsedSize = (uint)FBigBufSize;
```

```
            if (SetParametersStream(ref FPlatHandle, Pointer, 2, ref FUsedSize, ref
FData, ref FSync, L_STREAM_ADC) == L_ERROR)
                return false;

            Marshal.FreeHGlobal(Pointer);

            if (FEvent != IntPtr.Zero)
                CloseHandle(FEvent);
            FEvent = CreateEvent(IntPtr.Zero, false, false, "");
            if (SetLDeviceEvent(ref FPlatHandle, FEvent, 1) == L_ERROR)
                return false;
            FInitialized = true;
            return true;
        }
```

Додаток Б2. Обрахунок вібропараметрів по показам датчиків вібрації опор

```
public void CalculateOporVibroParam(OporSensor Sensor, bool AlarmFlag)
        {
            short ph, i;
            ushort Nperiod;
            short razm;
            short    Ve    =    0,    FirstSpeedGarmonic,    PhaseFirstSpeedGarmonic,
SecondSpeedGarmonic, PhaseSecondSpeedGarmonic, S1, FS1, S2, FS2;
            double A, fit, Gain;
            double FF1, F, D = 0;
            double[] SignalDouble = new double[FDigitalCounter];
            float Frequency = (float)FAgt.CurrentFrequency / 60;
            Complex[] AmplitudesArray = new Complex[FDigitalCounter];
            Complex[] TempArray = new Complex[FDigitalCounter];
            int PlatIndex = GetPlatByID(Sensor.Config.PlatID);
            int ChannelNumber = Sensor.Config.ChannelNumber - 1;

            if ((PlatIndex == -1) || (ChannelNumber < 0))
                return;

            Nperiod = GetNperiod();

            for (i = 0; i < FDigitalCounter; i++) // подготовка массива
            {
                AmplitudesArray[i] = new Complex();
                TempArray[i] = new Complex();
                AmplitudesArray[i].Re                                                =
FPlats[PlatIndex].Channels[ChannelNumber].Data[i];
                AmplitudesArray[i].Im = 0;
            }

            // прямое БФП
            SpeedFurye(FDigitalCounter, // число отсчетов
                preFFT(FDigitalCounter),      // массив комплексных экспонент
                ref AmplitudesArray,      // массив из N отсчетов исходных данных
                // в него же помещаются вычисленные спектральные
                // комплексные амплитуды
                ref TempArray,       // вспомогат. массив по величине равен массиву
X
                true       // true - прямое БФП
                );

            // вычисление для особых режимов интегральных характеристик Ve - для
опор, S - для вала
            if (FAgt.OporSensorsCommonCfg.EqualFiltrationFlag)
            {
                if (FAgt.CurrentFrequency <= 0)
                {
                        // вычисление Ve без фильтра
                        D     =     myrmsv2(AmplitudesArray);     //     вычисляет
среднеквадратичное значение
                }
            }
            else
            {
                if   ((Frequency   <   FAgt.OporSensorsCommonCfg.LowerFrequency)   ||
(FAgt.CurrentFrequency < MinFreq[0]))
```

```
                    // вычисление Ve без фильтра в случае выхода из полосы частот
оборотной составляющей
                    // при выходе из полосы частот далее весь спектр обнуляется
                    D = myrmsv2(AmplitudesArray); // вычисляет среднеквадратичное
значение
            }

            // фильтр спектра
            filtr(FDigitalCounter, // число отсчетов
                ref AmplitudesArray,     // массив спектральных составляющих после
БФП
                FAgt.OporSensorsCommonCfg.FilterType,         // тип фильтра
                GetLowerIndex(FAgt.OporSensorsCommonCfg.EqualFiltrationFlag,
FAgt.OporSensorsCommonCfg.LowerFrequency),
GetUpperIndex(FAgt.OporSensorsCommonCfg.UpperFrequency));

            // усреднение спектра в OldSpectr - усредненный спектр
            int Index = GetPlatByID(Sensor.Config.PlatID);
            USREDN_SPECTR_KAN(AlarmFlag,                            AmplitudesArray,
Sensor.Config.ChannelNumber - 1 + Index * Plat.MaxChannelsCount);

            // обратное БФП
            // подготовим массив без среднего значения
            for (i = 0; i < FDigitalCounter; i++)
                if (i != 0)
                {
                    AmplitudesArray[i].Re = OldSpectr[(Sensor.Config.ChannelNumber
- 1 + Index * Plat.MaxChannelsCount) * FDigitalCounter + i].Re;
                    AmplitudesArray[i].Im = OldSpectr[(Sensor.Config.ChannelNumber
- 1 + Index * Plat.MaxChannelsCount) * FDigitalCounter + i].Im;
                }
                else
                    AmplitudesArray[i].Re = AmplitudesArray[i].Im = 0;

            SpeedFurye(FDigitalCounter, // число отсчетов
                preFFT(FDigitalCounter),      // массив комплексных экспонент
                ref AmplitudesArray,      // массив из N отсчетов исходных данных
                // в него же помещаются вычисленные спектральные
                // комплексные амплитуды
                ref TempArray,      // вспомогат. массив по величине равен массиву
X
                false       // обратное БФП
                );

            // запомним фильтрованный сигнал без среднего значения в UBob
            // для последующего вычисления размаха из скорости в Signaldouble - для
опор
            // из перемещения в Signaldouble - для вала
            double x;

            for (i = 0; i < FDigitalCounter; i++)
            {
                SignalDouble[i]   =   x   =   2   *   AmplitudesArray[i].Re;   //   +
agr.Channel[kan].Clearance;

                if (x < 0)
                    OldAvgSignal[(Sensor.Config.ChannelNumber   -   1   +   Index   *
Plat.MaxChannelsCount) * FDigitalCounter + i] = (short)(x - 0.5);
```

```
                else
                    OldAvgSignal[(Sensor.Config.ChannelNumber - 1 + Index * Plat.MaxChannelsCount) * FDigitalCounter + i] = (short)(x + 0.5);
            }

            //  Последующее вычисление амплитуды и фазы спектра
            for (i = 0; i < FDigitalCounter; i++) // находим макс.и мин.
            {
                AmplitudesArray[i].Re = OldSpectr[(Sensor.Config.ChannelNumber - 1 + Index * Plat.MaxChannelsCount) * FDigitalCounter + i].Re;
                AmplitudesArray[i].Im = OldSpectr[(Sensor.Config.ChannelNumber - 1 + Index * Plat.MaxChannelsCount) * FDigitalCounter + i].Im;
            }

            calc_ampl(FDigitalCounter, // число отсчетов
                ref AmplitudesArray  // массив комплексных значений спектра
                // в него же помещаются вычисленные
                // значения амплитуды и фазы ( фаза только
                // первых трех составляющих )
                );

                F = FAgt.OporSensorsCommonCfg.Gain * Sensor.Config.Gain;

                if (FAgt.OporSensorsCommonCfg.EqualFiltrationFlag)
                {
                    if (Frequency > 0)
                    {
                        // вычисление Ve
                        D = myrmsv(AmplitudesArray); // вычисляет среднеквадратичное значение
                    }
                }
                else
                {
                    if ((Frequency >= FAgt.OporSensorsCommonCfg.LowerFrequency) && (FAgt.CurrentFrequency >= MinFreq[0]))
                    {
                        // вычисление Ve
                        D = myrmsv(AmplitudesArray); // вычисляет среднеквадратичное значение
                    }
                }

                razm = RAZMAH_S(0.0, GetPlatByID(Sensor.Config.PlatID), Sensor.Config.Gain, SignalDouble); // вычисление размаха из массива Signaldouble, массив вычислен через обратное
                // ПФ без среднего значения
                Ve = (short)(D * F + 0.5);
                FirstSpeedGarmonic = (short)(AmplitudesArray[Nperiod].Re * F + 0.5); // оборотная составляющая

                /*if (FOtm.InputType == OtmInputType.Analog)
                {*/
                    ph = (short)(AmplitudesArray[Nperiod].Im + 0.5);

                    if (FAgt.OporSensorsCommonCfg.PhaseCorrectionFlag) // учет фазовой поправки для 1 гармоники
                    {
```

```
                    ph += (short)Sensor.Config.FrequencyCorrection;
                    ph += (short)FAZA_OT_FR(0);
                }

                if (ph < 0)
                {
                    ph += 360;

                    if (ph < 0)
                        ph += 360;
                }

                if (ph >= 360)
                {
                    ph -= 360;

                    if (ph >= 360)
                        ph -= 360;
                }

                PhaseFirstSpeedGarmonic = ph;
            /*}
            else
                PhaseFirstSpeedGarmonic = 0;
        */
            SecondSpeedGarmonic = (short)(AmplitudesArray[Nperiod * 2].Re * F +
0.5);

        //      if (FOtm.InputType == OtmInputType.Analog)
        //      {
                ph = (short)(AmplitudesArray[Nperiod * 2].Im + 0.5);
                PhaseSecondSpeedGarmonic = ph;
        //      }
        //      else
        //          PhaseSecondSpeedGarmonic = 0;

            Sensor.MeasureVibroParams.EffectiveVibroSpeed = Ve;
            Sensor.MeasureVibroParams.FirstSpeedGarmonic = FirstSpeedGarmonic;
            Sensor.MeasureVibroParams.PhaseFirstSpeedGarmonic                =
PhaseFirstSpeedGarmonic;
            Sensor.MeasureVibroParams.SecondSpeedGarmonic                    =
SecondSpeedGarmonic;
            Sensor.MeasureVibroParams.PhaseSecondSpeedGarmonic               =
PhaseSecondSpeedGarmonic;
            Sensor.MeasureVibroParams.Clearance                              =
(short)(OldSpectr[(Sensor.Config.ChannelNumber - 1 + Index * Plat.MaxChannelsCount)
* FDigitalCounter].Re * F);  // для опор - пост.составляющая

            FF1 = (double)Frequency;     // frq
            A       =       AmplitudesArray[Nperiod].Re       *       F       /
(double)FAgt.OporSensorsCommonCfg.ArchiveSaveGain;             // 1 гармоника
            Gain = 450.15815807855303477759959550337; //

            if (FF1 > 0)
                A = Gain / FF1 * A + 0.5;
            else
                A = 0;
```

```
                S1 = (short)A;
                FS1 = (short)(PhaseFirstSpeedGarmonic + 90);

                if (FS1 > 360)
                    FS1 -= 360;

                A     =     AmplitudesArray[2     *     Nperiod].Re     *     F     /
(double)FAgt.OporSensorsCommonCfg.ArchiveSaveGain;                // 2 гармоника

                if (FF1 > 0)
                    A = Gain / FF1 * A / 2 + 0.5;
                else
                    A = 0;     // ??

                S2 = (short)A;
                FS2 = (short)(PhaseSecondSpeedGarmonic + 90);

                if (FS2 >= 360)
                    FS2 -= 360;

                Sensor.MeasureVibroParams.MaxDisplacement  =  razm;  //  суммарное
виброперемещение

                Sensor.SpecialParams.S = Sensor.MeasureVibroParams.MaxDisplacement;
                Sensor.SpecialParams.S1 = S1;
                Sensor.SpecialParams.FS1 = FS1;
                Sensor.SpecialParams.S2 = S2;
                Sensor.SpecialParams.FS2 = FS2;
                Sensor.MeasureVibroParams.Garm3 = (short)(AmplitudesArray[Nperiod *
3].Re * F + 0.5);
                Sensor.MeasureVibroParams.Garm4 = (short)(AmplitudesArray[Nperiod *
4].Re * F + 0.5);
                Sensor.MeasureVibroParams.Garm5 = (short)(AmplitudesArray[Nperiod *
5].Re * F + 0.5);
                Sensor.MeasureVibroParams.Garm6 = (short)(AmplitudesArray[Nperiod *
6].Re * F + 0.5);
                Sensor.MeasureVibroParams.Garm7 = (short)(AmplitudesArray[Nperiod *
7].Re * F + 0.5);
                Sensor.MeasureVibroParams.Garm8 = (short)(AmplitudesArray[Nperiod *
8].Re * F + 0.5);
                Sensor.MeasureVibroParams.Garm9 = (short)(AmplitudesArray[Nperiod *
9].Re * F + 0.5);

                // РАСЧЕТ ВЧВ
                fit = 0;

                for (i = (short)(2 * Nperiod + 1); i < FDigitalCounter / 2 - 1; i++)
                {
                    A = AmplitudesArray[i].Re; // учет 0.75 Fr в общей нчв в мм/сек
                    fit += A * A;
                }

                if (fit > 0.01)
                    Sensor.MeasureVibroParams.HighFrequencyComponent              =
(short)(Math.Sqrt(fit) * F + 0.5);
                else
                    Sensor.MeasureVibroParams.HighFrequencyComponent = 0;
```

```
                // РАСЧЕТ НЧВ
                Sensor.MeasureVibroParams.SpeedAmplitude = 0;  // mm/sec
                Sensor.SpecialParams.SpeedAmplitude = 0; // mkm

                if (Nperiod >= 2) // расчет нчв = 0.5 Fr
                {
                    // усредненная нчв для скорости
                    fit = 0;

                    for (i = 1; i < Nperiod; i++)
                    {
                        A = AmplitudesArray[i].Re;
                        fit += A * A;
                    }

                    if (fit > 0.01)
                        Sensor.MeasureVibroParams.SpeedAmplitude                    =
(short)(Math.Sqrt(fit) * F + 0.5);
                    else
                        Sensor.MeasureVibroParams.SpeedAmplitude = 0;

                    // размах НЧВ
                    // подготовим массив оставив только низкочастотные составляющие
                    for (i = 0; i < FDigitalCounter; i++)
                        if (i < Nperiod && i != 0)
                        {
                            AmplitudesArray[i].Re                                   =
OldSpectr[(Sensor.Config.ChannelNumber  -  1  +  Index  *  Plat.MaxChannelsCount)  *
FDigitalCounter + i].Re;
                            AmplitudesArray[i].Im                                   =
OldSpectr[(Sensor.Config.ChannelNumber  -  1  +  Index  *  Plat.MaxChannelsCount)  *
FDigitalCounter + i].Im;
                        }
                        else
                            AmplitudesArray[i].Re = AmplitudesArray[i].Im = 0;

                    // обратное БФП
                    SpeedFurye(FDigitalCounter, // число отсчетов
                        preFFT(FDigitalCounter),     // массив комплексных экспонент
                        ref AmplitudesArray,       // массив из N отсчетов исходных
данных
                        // в него же помещаются вычисленные спектральные
                        // комплексные амплитуды
                        ref TempArray,       // вспомогат. массив по величине равен
массиву X
                        false       // обратное БФП
                        );

                    for (i = 0; i < FDigitalCounter; i++)
                        SignalDouble[i] = 2 * AmplitudesArray[i].Re;

                    Sensor.SpecialParams.SpeedAmplitude       =       RAZMAH_S(0.0,
GetPlatByID(Sensor.Config.PlatID), Sensor.Config.Gain, SignalDouble); // вычисление
размаха НЧВ из массива Signaldouble
                    // без среднего значения
                }
        }
```

Додаток Б3 клас Modbus

```
namespace LCardWindows
{
    public static class Modbus
    {
        const int port = 502;

        public static ModbusServer Server = new ModbusServer();

        public static void Start()
        {
            Server.Port = port;
            Server.UnitIdentifier = 10;
            Server.inputRegisters.localArray = new short[5000];

            Server.HoldingRegistersChanged += Server_HoldingRegistersChanged;
            Server.Listen();
        }

        private static void Server_HoldingRegistersChanged(int register, int
numberOfRegisters)
        {
            Console.WriteLine(register);
            Console.WriteLine(numberOfRegisters);
        }

        public static void Stop()
        {
            Server.StopListening();
        }

        public static void ValuesToModbusRegisters(Agregate agregate)
        {
            if (agregate != null && agregate.OporSensors != null)
            {
                int addr = 1;

                var listGroupOporSensors = agregate.OporSensors.GroupBy(s =>
s.Config.PodshipnikNumber).ToList(); //групуємо по номеру підшипника
                foreach(var g in listGroupOporSensors)
                {
                    foreach(var sensor in g) //додаємо датчики опори в таблицю
змінних
                    {
                        Modbus.Server.inputRegisters[addr + 0] =
sensor.MeasureVibroParams.EffectiveVibroSpeed;
                        Modbus.Server.inputRegisters[addr + 1] =
sensor.MeasureVibroParams.FirstSpeedGarmonic;
                        Modbus.Server.inputRegisters[addr + 2] =
sensor.MeasureVibroParams.PhaseFirstSpeedGarmonic;
                        Modbus.Server.inputRegisters[addr + 3] =
sensor.MeasureVibroParams.SecondSpeedGarmonic;
                        Modbus.Server.inputRegisters[addr + 4] =
sensor.MeasureVibroParams.PhaseSecondSpeedGarmonic;
                        Modbus.Server.inputRegisters[addr + 5] =
sensor.MeasureVibroParams.SpeedAmplitude;
```

```
                        Modbus.Server.inputRegisters[addr + 6] =
sensor.MeasureVibroParams.Garm3;
                        Modbus.Server.inputRegisters[addr + 7] =
sensor.MeasureVibroParams.Garm4;
                        Modbus.Server.inputRegisters[addr + 8] =
sensor.MeasureVibroParams.Garm5;
                        Modbus.Server.inputRegisters[addr + 9] =
sensor.MeasureVibroParams.Garm6;
                        Modbus.Server.inputRegisters[addr + 10] =
sensor.MeasureVibroParams.Garm7;
                        Modbus.Server.inputRegisters[addr + 11] =
sensor.MeasureVibroParams.Garm8;
                        Modbus.Server.inputRegisters[addr + 12] =
sensor.MeasureVibroParams.Garm9;
                        Modbus.Server.inputRegisters[addr + 13] =
sensor.MeasureVibroParams.Clearance;
                        Modbus.Server.inputRegisters[addr + 14] =
sensor.MeasureVibroParams.HighFrequencyComponent;
                        Modbus.Server.inputRegisters[addr + 15] =
sensor.MeasureVibroParams.MaxDisplacement;

                        addr = addr + 20;
                    }
                    if (agregate.ValSensors != null)
                    {
                        var valSensors = agregate.ValSensors.Where(el =>
el.Config.OporNumber == g.Key).ToList(); //Виберемо датчики вала для поточної опори

                        if (valSensors != null)
                        {
                            foreach(var sensor in valSensors)
                            {
                                Modbus.Server.inputRegisters[addr + 0] =
sensor.MeasureVibroParams.EffectiveVibroSpeed;
                                Modbus.Server.inputRegisters[addr + 1] =
sensor.MeasureVibroParams.FirstSpeedGarmonic;
                                Modbus.Server.inputRegisters[addr + 2] =
sensor.MeasureVibroParams.PhaseFirstSpeedGarmonic;
                                Modbus.Server.inputRegisters[addr + 3] =
sensor.MeasureVibroParams.SecondSpeedGarmonic;
                                Modbus.Server.inputRegisters[addr + 4] =
sensor.MeasureVibroParams.PhaseSecondSpeedGarmonic;
                                Modbus.Server.inputRegisters[addr + 5] =
sensor.MeasureVibroParams.SpeedAmplitude;
                                Modbus.Server.inputRegisters[addr + 6] =
sensor.MeasureVibroParams.Garm3;
                                Modbus.Server.inputRegisters[addr + 7] =
sensor.MeasureVibroParams.Garm4;
                                Modbus.Server.inputRegisters[addr + 8] =
sensor.MeasureVibroParams.Garm5;
                                Modbus.Server.inputRegisters[addr + 9] =
sensor.MeasureVibroParams.Garm6;
                                Modbus.Server.inputRegisters[addr + 10] =
sensor.MeasureVibroParams.Garm7;
                                Modbus.Server.inputRegisters[addr + 11] =
sensor.MeasureVibroParams.Garm8;
                                Modbus.Server.inputRegisters[addr + 12] =
sensor.MeasureVibroParams.Garm9;
```

```
                                Modbus.Server.inputRegisters[addr + 13] =
sensor.MeasureVibroParams.Clearance;
                                Modbus.Server.inputRegisters[addr + 14] =
sensor.MeasureVibroParams.HighFrequencyComponent;
                                Modbus.Server.inputRegisters[addr + 15] =
sensor.MeasureVibroParams.MaxDisplacement;


                                addr = addr + 20;
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Додаток Б4 Виявлення дефектів.

```
private void CheckDefects()
    {
        string Errors = "";
        string[] Params = {
                    "для опор Ve[мм/сек], для вала S[мкм]",
                    "для опор V1[мм/сек], для вала S1[мкм]",
                    "для опор V2[мм/сек], для вала S2[мкм]",
                    "для опор НЧВ[мм/сек], для вала НЧВ[мкм]",
                    "для опор V3[мм/сек], для вала S3[мкм]",
                    "для опор V4[мм/сек], для вала S4[мкм]",
                    "для опор V5[мм/сек], для вала S5[мкм]",
                    "для опор V6[мм/сек], для вала S6[мкм]",
                    "для опор V7[мм/сек], для вала S7[мкм]",
                    "для опор V8[мм/сек], для вала Zx[мкм]",
                    "для опор V9[мм/сек], для вала Zy[мкм]",
                    "для опор ПС[мм/сек], для вала Zaz[мкм]",
                    "для опор ВЧВ[мм/сек], для вала Dmin[мкм]",
                    "для опор S[мм/сек], для вала Smax[мкм]"
                };
        if (FHistory == null)
            return;
        for (int i = 0; i <= FPeriodCount; i++)
            if (FHistory[i] == null)
                return;
        Plat SignalizationPlat = null;
        Defect[] CurrentDefects = null;
        foreach (Plat pt in FPlats)
        {
            if (pt.Signalization)
            {
                SignalizationPlat = pt;
                break;
            }
        }
        switch (FAgt.Mode)
        {
            case FrequencyModeType.Nominal: CurrentDefects = FPowerDefects; break;
            case FrequencyModeType.ValRotation: CurrentDefects = FValRotationDefects; break;
            case FrequencyModeType.Up: CurrentDefects = FUpDefects; break;
            case FrequencyModeType.Stop: CurrentDefects = FStopDefects; break;
        };

        if (CurrentDefects == null)
            return;

        int[] FailChannels;
        int[] FailNodes;
        foreach (Defect df in CurrentDefects)
        {
            DateTime CurrentDateTime = DateTime.Now;
            if (df.Enable == false)
                continue;
```

```
if (CalculateDefectDiagnosis(df, out FailChannels, out FailNodes))
{
   if (!df.Set) // Если дефект возник в первый раз, то запоминаем время
   {
      df.SetTime = CurrentDateTime;
      WorkErrorMessageLog("Внимание!!! Обнаружен дефект - " + df.Name + ".");
      if ((df.Signalization > 0) && (SignalizationPlat != null))
         SignalizationPlat.OPEN_SIG(df.Signalization);
   }
   df.Set = true;
   df.Reset = false;

   switch (df.DiagOpen.A)
   {
      case 1:
         {
            if ((df.DiagOpen.B == 2) && df.Set && (df.SetTime == CurrentDateTime))
            {
               // запись в файл событий
               string DateName = (CurrentDateTime.Year % 100).ToString("D2") +
CurrentDateTime.Month.ToString("D2") + CurrentDateTime.Day.ToString("D2");
               EventRecord Record = new EventRecord();
               Encoding Unicode = Encoding.Unicode;
               Encoding Dos = Encoding.GetEncoding(866);

               byte[] UnicodeBytes = Unicode.GetBytes(CurrentDateTime.Day.ToString("D2") + "-" +
CurrentDateTime.Month.ToString("D2") + "-" + (CurrentDateTime.Year % 100).ToString("D2") + " " +
CurrentDateTime.Hour.ToString("D2") + ":" + CurrentDateTime.Minute.ToString("D2") + ":" +
CurrentDateTime.Second.ToString("D2"));
               byte[] DosBytes = Encoding.Convert(Unicode, Dos, UnicodeBytes);

               for (int i = 0; i < DosBytes.Length; i++)
                  Record.CurrentDateTime[i] = DosBytes[i];
               Record.CurrentDateTime[17] = 0;

               UnicodeBytes = Unicode.GetBytes(df.Name + ", 1-В 1-П 1-О 2-В");
               DosBytes = Encoding.Convert(Unicode, Dos, UnicodeBytes);

               int Min = DosBytes.Length;

               if (Min > Record.Text.Length)
                  Min = Record.Text.Length - 1;

               for (int i = 0; i < Min; i++)
                  Record.Text[i] = DosBytes[i];
               Record.Text[Min] = 0;

               if (df.CopyFileID == 0)
                  UnicodeBytes = Unicode.GetBytes("Нет");
               else
               {
                  int Counter = 0;
                  string Ext = FFilesStruct.ServerStructs[df.CopyFileID - 1].Extention2;
                  if (Directory.Exists(FServerArchivePath))
```

```
                    {
                        string[] FileNames = Directory.GetFiles(FServerArchivePath, "*" + Ext);
                        foreach (string Name in FileNames)
                        {
                            FileInfo Info = new FileInfo(Name);
                            if (Info.Name.IndexOf(DateName) == 0)
                                Counter++;
                        }
                        if (File.Exists(FFilesStruct.ControllerStructs[df.CopyFileID - 1].FileFullName))
                        {
                            UnicodeBytes = Unicode.GetBytes(DateName + Counter.ToString("D2") + Ext);
                            File.Copy(FFilesStruct.ControllerStructs[df.CopyFileID - 1].FileFullName, FServerArchivePath +
DateName + Counter.ToString("D2") + Ext);
                        }
                        else
                            UnicodeBytes = Unicode.GetBytes("Нет");
                    }
                    else
                        UnicodeBytes = Unicode.GetBytes("Нет");
                }
                DosBytes = Encoding.Convert(Unicode, Dos, UnicodeBytes);
                for (int i = 0; i < DosBytes.Length; i++)
                    Record.FileName[i] = DosBytes[i];
                Record.FileName[DosBytes.Length] = 0;

                if (df.ViewParam < 0)
                {
                    UnicodeBytes = Unicode.GetBytes("");
                    Record.Phase = -1;
                }
                else
                    if (df.ViewParam < Params.Length)
                    {
                        UnicodeBytes = Unicode.GetBytes(Params[df.ViewParam]);
                        int SensorsCount = FAgt.OporSensors.Length + FAgt.ValSensors.Length;
                        if ((df.ViewParam == 1) || (df.ViewParam == 2))
                        {
                            Record.Phase = 1;
                            for (int i = 0; i < SensorsCount; i++)
                                if (df.ViewParam == 1)
                                {
                                    Record.VibroDo[i] = FHistory[FPeriodCount].VibroParamValues[i].FirstSpeedGarmonic;
                                    Record.PhaseDo[i] = FHistory[FPeriodCount].VibroParamValues[i].PhaseFirstSpeedGarmonic;
                                }
                                else
                                {
                                    Record.VibroDo[i] = FHistory[FPeriodCount].VibroParamValues[i].SecondSpeedGarmonic;
                                    Record.PhaseDo[i] =
FHistory[FPeriodCount].VibroParamValues[i].PhaseSecondSpeedGarmonic;
                                }
                            for (int i = 0; i < SensorsCount; i++)
                                if (df.ViewParam == 1)
                                {
                                    Record.VibroPosle[i] = FHistory[0].VibroParamValues[i].FirstSpeedGarmonic;
```

```
            Record.PhasePosle[i] = FHistory[0].VibroParamValues[i].PhaseFirstSpeedGarmonic;
        }
        else
        {
            Record.VibroPosle[i] = FHistory[0].VibroParamValues[i].SecondSpeedGarmonic;
            Record.PhasePosle[i] = FHistory[0].VibroParamValues[i].PhaseSecondSpeedGarmonic;
        }

    }
    else
    {
        Record.Phase = 0;
        switch (df.ViewParam)
        {
            case 0:
                {
                    for (int i = 0; i < SensorsCount; i++)
                    {
                        Record.VibroDo[i] = FHistory[FPeriodCount].VibroParamValues[i].EffectiveVibroSpeed;
                        Record.VibroPosle[i] = FHistory[0].VibroParamValues[i].EffectiveVibroSpeed;
                    }
                    break;
                }
            case 3:
                {
                    for (int i = 0; i < SensorsCount; i++)
                    {
                        Record.VibroDo[i] = FHistory[FPeriodCount].VibroParamValues[i].SpeedAmplitude;
                        Record.VibroPosle[i] = FHistory[0].VibroParamValues[i].SpeedAmplitude;
                    }
                    break;
                }
            case 4:
                {
                    for (int i = 0; i < SensorsCount; i++)
                    {
                        Record.VibroDo[i] = FHistory[FPeriodCount].VibroParamValues[i].Garm3;
                        Record.VibroPosle[i] = FHistory[0].VibroParamValues[i].Garm3;
                    }
                    break;
                }
            case 5:
                {
                    for (int i = 0; i < SensorsCount; i++)
                    {
                        Record.VibroDo[i] = FHistory[FPeriodCount].VibroParamValues[i].Garm4;
                        Record.VibroPosle[i] = FHistory[0].VibroParamValues[i].Garm4;
                    }
                    break;
                }
            case 6:
                {
                    for (int i = 0; i < SensorsCount; i++)
                    {
```

```
            Record.VibroDo[i] = FHistory[FPeriodCount].VibroParamValues[i].Garm5;
            Record.VibroPosle[i] = FHistory[0].VibroParamValues[i].Garm5;
        }
        break;
    }
case 7:
    {
        for (int i = 0; i < SensorsCount; i++)
        {
            Record.VibroDo[i] = FHistory[FPeriodCount].VibroParamValues[i].Garm6;
            Record.VibroPosle[i] = FHistory[0].VibroParamValues[i].Garm6;
        }
        break;
    }
case 8:
    {
        for (int i = 0; i < SensorsCount; i++)
        {
            Record.VibroDo[i] = FHistory[FPeriodCount].VibroParamValues[i].Garm7;
            Record.VibroPosle[i] = FHistory[0].VibroParamValues[i].Garm7;
        }
        break;
    }
case 9:
    {
        for (int i = 0; i < SensorsCount; i++)
        {
            Record.VibroDo[i] = FHistory[FPeriodCount].VibroParamValues[i].Garm8;
            Record.VibroPosle[i] = FHistory[0].VibroParamValues[i].Garm8;
        }
        break;
    }
case 10:
    {
        for (int i = 0; i < SensorsCount; i++)
        {
            Record.VibroDo[i] = FHistory[FPeriodCount].VibroParamValues[i].Garm9;
            Record.VibroPosle[i] = FHistory[0].VibroParamValues[i].Garm9;
        }
        break;
    }
case 11:
    {
        for (int i = 0; i < SensorsCount; i++)
        {
            Record.VibroDo[i] = FHistory[FPeriodCount].VibroParamValues[i].Clearance;
            Record.VibroPosle[i] = FHistory[0].VibroParamValues[i].Clearance;
        }
        break;
    }
case 12:
    {
        for (int i = 0; i < SensorsCount; i++)
        {
```

```
                                Record.VibroDo[i] =
FHistory[FPeriodCount].VibroParamValues[i].HighFrequencyComponent;
                                Record.VibroPosle[i] = FHistory[0].VibroParamValues[i].HighFrequencyComponent;
                            }
                            break;
                        }
                    case 13:
                        {
                            for (int i = 0; i < SensorsCount; i++)
                            {
                                Record.VibroDo[i] = FHistory[FPeriodCount].VibroParamValues[i].MaxDisplacement;
                                Record.VibroPosle[i] = FHistory[0].VibroParamValues[i].MaxDisplacement;
                            }
                            break;
                        }
                    default:
                        {
                            for (int i = 0; i < SensorsCount; i++)
                            {
                                Record.VibroDo[i] = FHistory[FPeriodCount].VibroParamValues[i].EffectiveVibroSpeed;
                                Record.VibroPosle[i] = FHistory[0].VibroParamValues[i].EffectiveVibroSpeed;
                            }
                            break;
                        }
                }
            }
        }
        else
            if (df.ViewParam - Params.Length < FAgt.ExplParamsCount)
            {
                Record.Phase = 2;
                int MaxLength = 28;
                if (FAgt.ExplParams[df.ViewParam - Params.Length].Name.Length < 28)
                    MaxLength = FAgt.ExplParams[df.ViewParam - Params.Length].Name.Length;
                if (MaxLength == 28)
                    UnicodeBytes = Unicode.GetBytes(FAgt.ExplParams[df.ViewParam -
Params.Length].Name.Substring(0, MaxLength) + "..., " + FAgt.ExplParams[df.ViewParam - Params.Length].Razmernost);
                else
                    UnicodeBytes = Unicode.GetBytes(FAgt.ExplParams[df.ViewParam -
Params.Length].Name.Substring(0, MaxLength) + ", " + FAgt.ExplParams[df.ViewParam - Params.Length].Razmernost);

                int EnabledParamsCount = 0;
                foreach (ExplParameter Param in FAgt.ExplParams)
                {
                    if (Param.Enable)
                        EnabledParamsCount++;
                }
                if (EnabledParamsCount < Agregate.MaxSensorsCount - 1)
                    Record.FailChannels[0] = (short)EnabledParamsCount;
                else
                    Record.FailChannels[0] = Agregate.MaxSensorsCount - 1;

                for (int i = 0; i < (int)(Record.FailChannels[0]); i++)
                    Record.FailChannels[i + 1] = (short)i;
```

```
                    int Count = FAgt.ExplParamsCount;
                    if (Count > Agregate.MaxSensorsCount - 1)
                        Count = Agregate.MaxSensorsCount - 1;

                    byte[] array;
                    int Counter = 0;
                    for (int j = 0; j < FAgt.ExplParamsCount; j++)
                    {
                        if (FAgt.ExplParams[j].Enable)
                        {
                            array = BitConverter.GetBytes(FHistory[FPeriodCount].ExplParamValues[j]);
                            if (Counter < Agregate.MaxSensorsCount / 2)
                            {
                                Record.VibroDo[Counter * 2] = BitConverter.ToInt16(array, 0);
                                Record.VibroDo[Counter * 2 + 1] = BitConverter.ToInt16(array, 2);
                            }
                            else
                            {
                                Record.VibroPosle[Counter * 2 - Agregate.MaxSensorsCount] = BitConverter.ToInt16(array,
0);
                                Record.VibroPosle[Counter * 2 + 1 - Agregate.MaxSensorsCount] =
BitConverter.ToInt16(array, 2);
                            }

                            array = BitConverter.GetBytes(FAgt.ExplParams[j].Value);
                            if (Counter < Agregate.MaxSensorsCount / 2)
                            {
                                Record.PhaseDo[Counter * 2] = BitConverter.ToInt16(array, 0);
                                Record.PhaseDo[Counter * 2 + 1] = BitConverter.ToInt16(array, 2);
                            }
                            else
                            {
                                Record.PhasePosle[Counter * 2 - Agregate.MaxSensorsCount] =
BitConverter.ToInt16(array, 0);
                                Record.PhasePosle[Counter * 2 + 1 - Agregate.MaxSensorsCount] =
BitConverter.ToInt16(array, 2);
                            }
                            Counter++;
                        }
                        if (Counter > Count)
                            break;
                    }

                }
                else
                {
                    UnicodeBytes = Unicode.GetBytes("");
                    Record.Phase = -1;
                }

            DosBytes = Encoding.Convert(Unicode, Dos, UnicodeBytes);
            for (int i = 0; i < DosBytes.Length; i++)
                Record.VibroParam[i] = DosBytes[i];
```

```
                Record.VibroParam[DosBytes.Length] = 0;

                UnicodeBytes = Unicode.GetBytes("SOVET");
                DosBytes = Encoding.Convert(Unicode, Dos, UnicodeBytes);
                for (int i = 0; i < DosBytes.Length; i++)
                  Record.Temp[i] = DosBytes[i];
                Record.Temp[5] = 0;
                Record.DefectNumber = (short)df.ID;
                Record.ModeNumber = (short)FAgt.Mode;
                Record.Mode = (short)FAgt.Mode;
                Record.SubMode = 1;
                Record.TimeBefore = ProgramUtils.GetSecondsCountFrom1970(FHistory[FPeriodCount].DataTime);
                Record.TimeAfter = ProgramUtils.GetSecondsCountFrom1970(CurrentDateTime);

                if (FHistory != null)
                {
                  if (Directory.Exists(FServerArchivePath))
                  {
                    // Сохранение вспомогательного файла
                    ArchiveFileHeader Header = new ArchiveFileHeader();
                    Header.RecordLength = (ushort)(sizeof(int) + sizeof(short) * 2 + sizeof(short) * 16 *
(FAgt.OporSensors.Length + FAgt.ValSensors.Length) + sizeof(float) * Agregate.ExplParamMaxCount);
                    Header.CurrentRecord = 0;
                    Header.RecordCount = 0;
                    Header.MaxRecordCount = FFilesStruct.ControllerStructs[(int)EventType.Additional].MaxRecords;
                    Header.Type = new sbyte[10] { 90, 73, 76, 69, 45, 49, 0, 0, 0, 0 };  // Тип файла ZILE-1
                    Header.HeaderLength = FileArchiver.SeekStart;
                    Header.StationName = new byte[40];
                    Header.AgregateName = new byte[40];
                    char[] Temp = FAgt.Name.ToCharArray();
                    byte[] BytesName = Encoding.Convert(Encoding.Unicode, Encoding.GetEncoding(866),
Encoding.Unicode.GetBytes(Temp));
                    for (int k = 0; k < BytesName.Length; k++)
                      Header.AgregateName[k] = BytesName[k];
                    Header.AgregateName[BytesName.Length] = 0;
                    Temp = FAgt.StationName.ToCharArray();
                    BytesName = Encoding.Convert(Encoding.Unicode, Encoding.GetEncoding(866),
Encoding.Unicode.GetBytes(Temp));
                    for (int k = 0; k < BytesName.Length; k++)
                      Header.StationName[k] = BytesName[k];
                    Header.StationName[BytesName.Length] = 0;
                    Header.AgregateNumber = FAgt.StationNumber;
                    Header.OporsCount = FAgt.OporsCount;
                    Header.OporNumbers = new ushort[FAgt.OporsCount];
                    for (int j = 0; j < FAgt.OporsCount; j++)
                      Header.OporNumbers[j] = (ushort)(j + 1);
                    Header.OporsMeasuresCount = 3;
                    Header.ValsMeasuresCount = 2;
                    Header.MeasureNames = new sbyte[5, 4];
                    BytesName = Encoding.Convert(Encoding.Unicode, Encoding.GetEncoding(866),
Encoding.Unicode.GetBytes("В"));
                    Header.MeasureNames[0, 0] = (sbyte)BytesName[0];
                    BytesName = Encoding.Convert(Encoding.Unicode, Encoding.GetEncoding(866),
Encoding.Unicode.GetBytes("П"));
```

```
Header.MeasureNames[1, 0] = (sbyte)BytesName[0];
BytesName = Encoding.Convert(Encoding.Unicode, Encoding.GetEncoding(866),
Encoding.Unicode.GetBytes("O"));
Header.MeasureNames[2, 0] = (sbyte)BytesName[0];
Header.MeasureNames[3, 0] = 49;
Header.MeasureNames[4, 0] = 50;
Header.SensorIndexes = new ushort[FAgt.OporSensors.Length + FAgt.ValSensors.Length];
ushort Index = 1;
foreach (OporSensor Sensor in FAgt.OporSensors)
{
   Header.SensorIndexes[Index - 1] = Index;
   Index++;
}
foreach (ValSensor Sensor in FAgt.ValSensors)
{
   Header.SensorIndexes[Index - 1] = Index;
   Index++;
}
Header.VibroParamCount = 16;
Header.OporsSensorsCount = (ushort)FAgt.OporSensors.Length;
Header.ValsSensorsCount = (ushort)FAgt.ValSensors.Length;
Header.MaxSensors = 128;
int Count = 0;
for (int j = 0; j < FAgt.ExplParamsCount; j++)
   if (FAgt.ExplParams[j].Enable)
      Count++;
Header.ExplParamsCount = (ushort)Count;
Header.ExplParamNumbers = new ushort[Count];
Index = 1;
foreach (ExplParameter Param in FAgt.ExplParams)
{
   if (Param.Enable)
   {
      Header.ExplParamNumbers[Index - 1] = Index;
      Index++;
   }
}

int Counter = 0;
// Определение порядкового номера вспомогательного файла
string[] FileNames = Directory.GetFiles(FServerArchivePath, "*.dgn");
foreach (string Name in FileNames)
{
   FileInfo Info = new FileInfo(Name);
   if (Info.Name.IndexOf(DateName) == 0)
      Counter++;
}

Record.FailChannels[Agregate.MaxSensorsCount - 1] = (short)(Counter + 1); // номер сохраненного
файла
FileArchiver Dgn = new FileArchiver(EventType.Additional, DateName + Counter.ToString("D2"),
FServerArchivePath, FFilesStruct.ServerStructs[(int)EventType.Additional].Extention1,
FFilesStruct.ServerStructs[(int)EventType.Additional].Extention2,
```

```
FFilesStruct.ServerStructs[(int)EventType.Additional].TimeInterval,
FFilesStruct.ServerStructs[(int)EventType.Additional].Save, Header);
                    if (Dgn.Open(out Errors))
                    {

                        int Max = FFilesStruct.ServerStructs[(int)EventType.Additional].MaxRecords;
                        for (int i = Max - 1; i >= 0; i--)
                        {
                            if (FHistory[i] == null)
                                continue;

                            Record Rec = new Record();
                            Rec.FileTime = ProgramUtils.GetSecondsCountFrom1970(FHistory[i].DataTime);
                            Rec.Frequency = FHistory[i].Frequency;
                            Rec.ExplParams = new float[FAgt.ExplParamsCount];
                            int Count1 = 0;
                            for (int j = 0; j < FAgt.ExplParamsCount; j++)
                                if (FAgt.ExplParams[j].Enable)
                                    Count1++;
                            Rec.ExplParams = new float[Count1];
                            Counter = 0;
                            for (int j = 0; j < FAgt.ExplParamsCount; j++)
                                if (FAgt.ExplParams[j].Enable)
                                {
                                    Rec.ExplParams[Counter] = FHistory[i].ExplParamValues[j];
                                    Counter++;
                                }

                            Rec.VirboParams = new short[(FAgt.ValSensors.Length + FAgt.OporSensors.Length) * 16];
                            Counter = 0;
                            for (int j = 0; j < FAgt.OporSensors.Length; j++)
                            {
                                Rec.VirboParams[Counter] = FHistory[i].VibroParamValues[j].EffectiveVibroSpeed;
                                Rec.VirboParams[Counter + 1] = FHistory[i].VibroParamValues[j].FirstSpeedGarmonic;
                                Rec.VirboParams[Counter + 2] = FHistory[i].VibroParamValues[j].PhaseFirstSpeedGarmonic;
                                Rec.VirboParams[Counter + 3] = FHistory[i].VibroParamValues[j].SecondSpeedGarmonic;
                                Rec.VirboParams[Counter + 4] =
FHistory[i].VibroParamValues[j].PhaseSecondSpeedGarmonic;
                                Rec.VirboParams[Counter + 5] = FHistory[i].VibroParamValues[j].SpeedAmplitude;
                                Rec.VirboParams[Counter + 6] = FHistory[i].VibroParamValues[j].Garm3;
                                Rec.VirboParams[Counter + 7] = FHistory[i].VibroParamValues[j].Garm4;
                                Rec.VirboParams[Counter + 8] = FHistory[i].VibroParamValues[j].Garm5;
                                Rec.VirboParams[Counter + 9] = FHistory[i].VibroParamValues[j].Garm6;
                                Rec.VirboParams[Counter + 10] = FHistory[i].VibroParamValues[j].Garm7;
                                Rec.VirboParams[Counter + 11] = FHistory[i].VibroParamValues[j].Garm8;
                                Rec.VirboParams[Counter + 12] = FHistory[i].VibroParamValues[j].Garm9;
                                Rec.VirboParams[Counter + 13] = FHistory[i].VibroParamValues[j].Clearance;
                                Rec.VirboParams[Counter + 14] =
FHistory[i].VibroParamValues[j].HighFrequencyComponent;
                                Rec.VirboParams[Counter + 15] = FHistory[i].VibroParamValues[j].MaxDisplacement;
                                Counter += 16;
                            }
                            for (int j = 0; j < FAgt.ValSensors.Length; j++)
                            {
```

```
                                Rec.VirboParams[Counter] = FHistory[i].VibroParamValues[j].EffectiveVibroSpeed;
                                Rec.VirboParams[Counter + 1] = FHistory[i].VibroParamValues[j].FirstSpeedGarmonic;
                                Rec.VirboParams[Counter + 2] = FHistory[i].VibroParamValues[j].PhaseFirstSpeedGarmonic;
                                Rec.VirboParams[Counter + 3] = FHistory[i].VibroParamValues[j].SecondSpeedGarmonic;
                                Rec.VirboParams[Counter + 4] =
FHistory[i].VibroParamValues[j].PhaseSecondSpeedGarmonic;
                                Rec.VirboParams[Counter + 5] = FHistory[i].VibroParamValues[j].SpeedAmplitude;
                                Rec.VirboParams[Counter + 6] = FHistory[i].VibroParamValues[j].Garm3;
                                Rec.VirboParams[Counter + 7] = FHistory[i].VibroParamValues[j].Garm4;
                                Rec.VirboParams[Counter + 8] = FHistory[i].VibroParamValues[j].Garm5;
                                Rec.VirboParams[Counter + 9] = FHistory[i].VibroParamValues[j].Garm6;
                                Rec.VirboParams[Counter + 10] = FHistory[i].VibroParamValues[j].Garm7;
                                Rec.VirboParams[Counter + 11] = FHistory[i].VibroParamValues[j].Garm8;
                                Rec.VirboParams[Counter + 12] = FHistory[i].VibroParamValues[j].Garm9;
                                Rec.VirboParams[Counter + 13] = FHistory[i].VibroParamValues[j].Clearance;
                                Rec.VirboParams[Counter + 14] =
FHistory[i].VibroParamValues[j].HighFrequencyComponent;
                                Rec.VirboParams[Counter + 15] = FHistory[i].VibroParamValues[j].MaxDisplacement;
                                Counter += 16;
                            }
                            /*if (Dgn.WriteRecord(Rec, out Errors) == false)
                                WorkErrorMessageLog(Errors);*/
                        }
                    }
                    else
                        WorkErrorMessageLog(Errors);
                    Dgn.Close();
                }
            }
            /*if (FFilesStruct.ServerStructs[(int)EventType.Events].WriteRecord(Record, out Errors) == false)
                WorkErrorMessageLog(Errors);*/
        }
        break;
    }
    case 2:
    {
        if (ProgramUtils.GetSecondsCountFrom1970(CurrentDateTime) -
ProgramUtils.GetSecondsCountFrom1970(df.SetTime) > df.DiagOpen.Time)
            if (df.DiagOpen.B == 2)
            {
                // запись в файл событий
            }
        break;
    }
}

int Interval = ProgramUtils.GetSecondsCountFrom1970(DateTime.Now) -
ProgramUtils.GetSecondsCountFrom1970(df.SetTime);
if ((df.SignalizationInterval != 32000) && (Interval > df.SignalizationInterval) && (SignalizationPlat != null))
    SignalizationPlat.CLOSE_SIG(df.Signalization);
}
else
{
    if (df.Reset == false)
```

```
                df.CloseTime = DateTime.Now;

            if ((df.Reset == false) && df.Set)
                df.Reset = true;

            if (SignalizationPlat != null)
                SignalizationPlat.CLOSE_SIG(df.Signalization);
        }

        switch (df.DiagClose.A)
        {
            case 2:
                {
                    if(df.Set)
                        if                    (ProgramUtils.GetSecondsCountFrom1970(DateTime.Now)                    -
ProgramUtils.GetSecondsCountFrom1970(df.SetTime) > df.DiagClose.Time)
                            df.Set = false;
                    break;
                }
            case 3:
                {
                    if(df.Reset)
                        if                    (ProgramUtils.GetSecondsCountFrom1970(DateTime.Now)                    -
ProgramUtils.GetSecondsCountFrom1970(df.CloseTime) > df.DiagClose.Time)
                            df.Set = false;
                    break;
                }
        }
    }
}

private bool CalculateDefectDiagnosis(Defect Def, out int[] FailChannels, out int[] FailNodes)
{
    ushort Diagnosis = 0;
    bool Result = false;
    var FailChannelsList = new List<int>();
    var FailNodesList = new List<int>();

    switch (Def.Location)
    {

        case DefectLocation.Channel:
            {
                int Count = FAgt.OporSensors.Length;
                for (int i = 0; i < Count; i++)
                {
                    Diagnosis = 0;
                    if (FAgt.OporSensors[i].Config.Enable)
                    {
                        foreach(DiagnosisToken Token in Def.DiagnosisTokens)
                            if(Token.Enable)
                                if(Token.ParamType == DiagTokenParamType.VibroOporParam)
                                    if (CheckDiagnosisToken(Token, i))
                                        Diagnosis += Token.Gain;
```

```
                }
                if (Diagnosis >= Def.Diagnosis)
                {
                    Result = true;
                    FailChannelsList.Add(FAgt.OporSensors[i].Config.ChannelNumber);
                }
            }
            Count = FAgt.ValSensors.Length;
            for (int i = 0; i < Count; i++)
            {
                Diagnosis = 0;
                if (FAgt.ValSensors[i].Config.Enable)
                {
                    foreach(DiagnosisToken Token in Def.DiagnosisTokens)
                        if(Token.Enable)
                            if(Token.ParamType == DiagTokenParamType.VibroValParam)
                                if (CheckDiagnosisToken(Token, i + FAgt.OporSensors.Length))
                                    Diagnosis += Token.Gain;
                }
                if (Diagnosis >= Def.Diagnosis)
                {
                    Result = true;
                    FailChannelsList.Add(FAgt.ValSensors[i].Config.ChannelNumber);
                }
            }
            for (int i = 0; i < FAgt.ExplParamsCount; i++)
            {
                Diagnosis = 0;
                foreach (DiagnosisToken Token in Def.DiagnosisTokens)
                {
                    if (FAgt.ExplParams[i].Enable && FAgt.ExplParams[i].InputType == ExplParamInputType.Plat)
                        if (Token.Enable && (Token.ParamType == DiagTokenParamType.ExplParam) && (Token.Param ==
FAgt.ExplParams[i].ID))
                        {
                            if (CheckDiagnosisToken(Token, i))
                                Diagnosis += Token.Gain;
                        }
                }
                if (Diagnosis >= Def.Diagnosis)
                {
                    Result = true;
                    FailChannelsList.Add(FAgt.ExplParams[i].Source);
                }
            }

            break;
        }
    case DefectLocation.Opor:
        {
            for (int i = 0; i < FAgt.OporsCount; i++)
            {
                Diagnosis = 0;
                int Count = FAgt.OporSensors.Length;
                for (int j = 0; j < Count; j++)
```

```
                {
                  if ((FAgt.OporSensors[j].Config.PodshipnikNumber == i + 1) && (FAgt.OporSensors[j].Config.Enable ==
true))
                  {
                    foreach (DiagnosisToken Token in Def.DiagnosisTokens)
                      if(Token.Enable)
                        if(Token.ParamType == DiagTokenParamType.VibroOporParam)
                          if (CheckDiagnosisToken(Token, i))
                          {
                            Diagnosis += Token.Gain;
                            FailChannelsList.Add(FAgt.OporSensors[j].Config.ChannelNumber);
                          }
                  }
                }

                Count = FAgt.ValSensors.Length;
                for (int j = 0; j < Count; j++)
                {
                  if ((FAgt.ValSensors[j].Config.OporNumber == i + 1) && (FAgt.ValSensors[j].Config.Enable == true))
                  {
                    foreach (DiagnosisToken Token in Def.DiagnosisTokens)
                      if (Token.Enable)
                        if (Token.ParamType == DiagTokenParamType.VibroValParam)
                          if (CheckDiagnosisToken(Token, i + FAgt.OporSensors.Length))
                          {
                            Diagnosis += Token.Gain;
                            FailChannelsList.Add(FAgt.ValSensors[j].Config.ChannelNumber);
                          }
                  }
                }

                foreach (DiagnosisToken Token in Def.DiagnosisTokens)
                  if (Token.Enable)
                  {
                    if (Token.ParamType == DiagTokenParamType.Special)
                      if (CheckDiagnosisToken(Token, FailChannelsList.Count))
                        Diagnosis += Token.Gain;

                    if (Token.ParamType == DiagTokenParamType.ExplParam)
                    {
                      for (int k = 0; k < FAgt.ExplParamsCount; k++)
                      {
                        if (FAgt.ExplParams[k].Enable && FAgt.ExplParams[k].ID == Token.Param)
                        {
                          if (CheckDiagnosisToken(Token, k))
                            Diagnosis += Token.Gain;
                          break;
                        }
                      }
                    }
                  }


                if (Diagnosis >= Def.Diagnosis)
```

```
                {
                    Result = true;
                    FailNodesList.Add(i + 1);
                }
            }
            break;
        }
    case DefectLocation.Rotor:
        {
            for (int i = 0; i < FAgt.RotorsCount; i++)
            {
                ushort Number = GetLeftOpor(FAgt.Rotors[i]);

                Diagnosis = 0;
                int Count = FAgt.OporSensors.Length;
                for (int j = 0; j < Count; j++)
                {
                    if ((FAgt.OporSensors[j].Config.PodshipnikNumber == Number + 1) ||
(FAgt.OporSensors[j].Config.PodshipnikNumber == Number + 2))
                    {
                        foreach (DiagnosisToken Token in Def.DiagnosisTokens)
                            if (Token.Enable)
                                if (Token.ParamType == DiagTokenParamType.VibroOporParam)
                                    if (CheckDiagnosisToken(Token, i))
                                    {
                                        Diagnosis += Token.Gain;
                                        FailChannelsList.Add(FAgt.OporSensors[j].Config.ChannelNumber);
                                    }
                    }
                }
                Count = FAgt.ValSensors.Length;
                for (int j = 0; j < Count; j++)
                {
                    if ((FAgt.ValSensors[j].Config.OporNumber == Number + 1) || (FAgt.ValSensors[j].Config.OporNumber ==
Number + 2))
                    {
                        foreach (DiagnosisToken Token in Def.DiagnosisTokens)
                            if (Token.Enable)
                                if (Token.ParamType == DiagTokenParamType.VibroValParam)
                                    if (CheckDiagnosisToken(Token, i + FAgt.OporSensors.Length))
                                    {
                                        Diagnosis += Token.Gain;
                                        FailChannelsList.Add(FAgt.ValSensors[j].Config.ChannelNumber);
                                    }
                    }
                }
                foreach (DiagnosisToken Token in Def.DiagnosisTokens)
                    if (Token.Enable)
                    {
                        if (Token.ParamType == DiagTokenParamType.Special)
                            if (CheckDiagnosisToken(Token, FailChannelsList.Count))
                                Diagnosis += Token.Gain;

                        if (Token.ParamType == DiagTokenParamType.ExplParam)
```

```
                    {
                        for (int k = 0; k < FAgt.ExplParamsCount; k++)
                        {
                            if (FAgt.ExplParams[k].Enable && FAgt.ExplParams[k].ID == Token.Param)
                            {
                                if (CheckDiagnosisToken(Token, k))
                                    Diagnosis += Token.Gain;
                                break;
                            }
                        }
                    }
                }

                if (Diagnosis >= Def.Diagnosis)
                {
                    Result = true;
                    FailNodesList.Add(i + 1);
                }
            }
            break;
        }
    case DefectLocation.Muft:
        {
            for (int i = 0; i < FAgt.MuftsCount; i++)
            {
                int LeftOpor = GetLeftOpor(FAgt.Rotors[FAgt.Mufts[i].LeftRotor]) + 1;
                int RightOpor = GetLeftOpor(FAgt.Rotors[FAgt.Mufts[i].RightRotor]);

                Diagnosis = 0;
                int Count = FAgt.OporSensors.Length;
                for (int j = 0; j < Count; j++)
                {
                    if ((FAgt.OporSensors[j].Config.PodshipnikNumber == LeftOpor + 1) || (FAgt.OporSensors[j].Config.PodshipnikNumber == RightOpor + 1))
                    {
                        foreach (DiagnosisToken Token in Def.DiagnosisTokens)
                            if (Token.Enable)
                                if (Token.ParamType == DiagTokenParamType.VibroOporParam)
                                    if (CheckDiagnosisToken(Token, i))
                                    {
                                        Diagnosis += Token.Gain;
                                        FailChannelsList.Add(FAgt.OporSensors[j].Config.ChannelNumber);
                                    }
                    }
                }
                Count = FAgt.ValSensors.Length;
                for (int j = 0; j < Count; j++)
                {
                    if ((FAgt.ValSensors[j].Config.OporNumber == LeftOpor + 1) || (FAgt.ValSensors[j].Config.OporNumber == RightOpor + 1))
                    {
                        foreach (DiagnosisToken Token in Def.DiagnosisTokens)
                            if (Token.Enable)
                                if (Token.ParamType == DiagTokenParamType.VibroValParam)
```

```
                    if (CheckDiagnosisToken(Token, i + FAgt.OporSensors.Length))
                    {
                        Diagnosis += Token.Gain;
                        FailChannelsList.Add(FAgt.ValSensors[j].Config.ChannelNumber);
                    }
                }
            }
            foreach (DiagnosisToken Token in Def.DiagnosisTokens)
                if (Token.Enable)
                {
                    if (Token.ParamType == DiagTokenParamType.Special)
                        if (CheckDiagnosisToken(Token, FailChannelsList.Count))
                            Diagnosis += Token.Gain;

                    if (Token.ParamType == DiagTokenParamType.ExplParam)
                    {
                        for (int k = 0; k < FAgt.ExplParamsCount; k++)
                        {
                            if (FAgt.ExplParams[k].Enable && FAgt.ExplParams[k].ID == Token.Param)
                            {
                                if (CheckDiagnosisToken(Token, k))
                                    Diagnosis += Token.Gain;
                                break;
                            }
                        }
                    }
                }

            if (Diagnosis >= Def.Diagnosis)
            {
                Result = true;
                FailNodesList.Add(i + 1);
            }
        }
        break;
    }
    case DefectLocation.RotorMuft:
    {
        for (int i = 0; i < FAgt.RotorsCount; i++)
        {
            ushort Number = GetLeftOpor(FAgt.Rotors[i]);

            Diagnosis = 0;
            int Count = FAgt.OporSensors.Length;
            for (int j = 0; j < Count; j++)
            {
                if ((FAgt.OporSensors[j].Config.PodshipnikNumber == Number + 1) ||
(FAgt.OporSensors[j].Config.PodshipnikNumber == Number + 2))
                {
                    foreach (DiagnosisToken Token in Def.DiagnosisTokens)
                        if (Token.Enable)
                            if (Token.ParamType == DiagTokenParamType.VibroOporParam)
                                if (CheckDiagnosisToken(Token, i))
                                {
```

```
                    Diagnosis += Token.Gain;
                    FailChannelsList.Add(FAgt.OporSensors[j].Config.ChannelNumber);
                }
            }
        }
        Count = FAgt.ValSensors.Length;
        for (int j = 0; j < Count; j++)
        {
            if ((FAgt.ValSensors[j].Config.OporNumber == Number + 1) || (FAgt.ValSensors[j].Config.OporNumber ==
Number + 2))
            {
                foreach (DiagnosisToken Token in Def.DiagnosisTokens)
                    if (Token.Enable)
                        if (Token.ParamType == DiagTokenParamType.VibroValParam)
                            if (CheckDiagnosisToken(Token, i + FAgt.OporSensors.Length))
                            {
                                Diagnosis += Token.Gain;
                                FailChannelsList.Add(FAgt.ValSensors[j].Config.ChannelNumber);
                            }
            }
        }
        foreach (DiagnosisToken Token in Def.DiagnosisTokens)
            if (Token.Enable)
            {
                if (Token.ParamType == DiagTokenParamType.Special)
                    if (CheckDiagnosisToken(Token, FailChannelsList.Count))
                        Diagnosis += Token.Gain;

                if (Token.ParamType == DiagTokenParamType.ExplParam)
                {
                    for (int k = 0; k < FAgt.ExplParamsCount; k++)
                    {
                        if (FAgt.ExplParams[k].Enable && FAgt.ExplParams[k].ID == Token.Param)
                        {
                            if (CheckDiagnosisToken(Token, k))
                                Diagnosis += Token.Gain;
                            break;
                        }
                    }
                }
            }

        if (Diagnosis >= Def.Diagnosis)
        {
            Result = true;
            FailNodesList.Add(i + 1);
        }
    }
    if (FailNodesList.Count == 0)
    {
        for (int i = 0; i < FAgt.MuftsCount; i++)
        {
            int LeftOpor = GetLeftOpor(FAgt.Rotors[FAgt.Mufts[i].LeftRotor]) + 1;
            int RightOpor = GetLeftOpor(FAgt.Rotors[FAgt.Mufts[i].RightRotor]);
```

```
                    Diagnosis = 0;
                    int Count = FAgt.OporSensors.Length;
                    for (int j = 0; j < Count; j++)
                    {
                      if ((FAgt.OporSensors[j].Config.PodshipnikNumber == LeftOpor + 1) ||
(FAgt.OporSensors[j].Config.PodshipnikNumber == RightOpor + 1))
                      {
                        foreach (DiagnosisToken Token in Def.DiagnosisTokens)
                          if (Token.Enable)
                            if (Token.ParamType == DiagTokenParamType.VibroOporParam)
                              if (CheckDiagnosisToken(Token, i))
                              {
                                Diagnosis += Token.Gain;
                                FailChannelsList.Add(FAgt.OporSensors[j].Config.ChannelNumber);
                              }
                      }
                    }
                    Count = FAgt.ValSensors.Length;
                    for (int j = 0; j < Count; j++)
                    {
                      if ((FAgt.ValSensors[j].Config.OporNumber == LeftOpor + 1) || (FAgt.ValSensors[j].Config.OporNumber
== RightOpor + 1))
                      {
                        foreach (DiagnosisToken Token in Def.DiagnosisTokens)
                          if (Token.Enable)
                            if (Token.ParamType == DiagTokenParamType.VibroValParam)
                              if (CheckDiagnosisToken(Token, i + FAgt.OporSensors.Length))
                              {
                                Diagnosis += Token.Gain;
                                FailChannelsList.Add(FAgt.ValSensors[j].Config.ChannelNumber);
                              }
                      }
                    }
                    foreach (DiagnosisToken Token in Def.DiagnosisTokens)
                      if (Token.Enable)
                      {
                        if (Token.ParamType == DiagTokenParamType.Special)
                          if (CheckDiagnosisToken(Token, FailChannelsList.Count))
                            Diagnosis += Token.Gain;

                        if (Token.ParamType == DiagTokenParamType.ExplParam)
                        {
                          for (int k = 0; k < FAgt.ExplParamsCount; k++)
                          {
                            if (FAgt.ExplParams[k].Enable && FAgt.ExplParams[k].ID == Token.Param)
                            {
                              if (CheckDiagnosisToken(Token, k))
                                Diagnosis += Token.Gain;
                              break;
                            }
                          }
                        }
                      }
```

```
            if (Diagnosis >= Def.Diagnosis)
            {
                Result = true;
                FailNodesList.Add((i + 1) * 10);
            }
        }
    }
    break;
  }
case DefectLocation.Agregate:
  {
    Diagnosis = 0;
    int Count = FAgt.OporSensors.Length;
    for (int i = 0; i < Count; i++)
    {
        if (FAgt.OporSensors[i].Config.Enable)
        {
            foreach (DiagnosisToken Token in Def.DiagnosisTokens)
                if (Token.Enable)
                    if (Token.ParamType == DiagTokenParamType.VibroOporParam)
                        if (CheckDiagnosisToken(Token, i))
                        {
                            Diagnosis += Token.Gain;
                            FailChannelsList.Add(FAgt.OporSensors[i].Config.ChannelNumber);
                        }
        }

    }
    Count = FAgt.ValSensors.Length;
    for (int i = 0; i < Count; i++)
    {
        if (FAgt.ValSensors[i].Config.Enable)
        {
            foreach (DiagnosisToken Token in Def.DiagnosisTokens)
                if (Token.Enable)
                    if (Token.ParamType == DiagTokenParamType.VibroValParam)
                        if (CheckDiagnosisToken(Token, i + FAgt.OporSensors.Length))
                        {
                            Diagnosis += Token.Gain;
                            FailChannelsList.Add(FAgt.ValSensors[i].Config.ChannelNumber);
                        }
        }
    }

    foreach (DiagnosisToken Token in Def.DiagnosisTokens)
        if (Token.Enable)
        {
            if (Token.ParamType == DiagTokenParamType.Special)
                if (CheckDiagnosisToken(Token, FailChannelsList.Count))
                    Diagnosis += Token.Gain;

            if (Token.ParamType == DiagTokenParamType.ExplParam)
            {
```

```
                    for (int k = 0; k < FAgt.ExplParamsCount; k++)
                    {
                        if (FAgt.ExplParams[k].Enable && FAgt.ExplParams[k].ID == Token.Param)
                        {
                            if (CheckDiagnosisToken(Token, k))
                                Diagnosis += Token.Gain;
                            break;
                        }
                    }
                }
            }

            if (Diagnosis >= Def.Diagnosis)
            {
                Result = true;
                FailNodesList.Add(FAgt.StationNumber);
            }
            break;
        }
    }

    FailChannels = FailChannelsList.ToArray();
    FailNodes = FailNodesList.ToArray();
    return Result;
}

private bool CheckDiagnosisToken(DiagnosisToken Token, int ChannelNumber)
{
    // ChannelNumber = номер датчика вала, номер дачика опоры, номер эксплуатационного параметра или число
пораженных каналов
    if (Token.Enable == false)
        return false;

    double Operand1 = 0, Operand2 = 0;
    bool Result;
    double[] Array = new double[FPeriodCount];
    switch (Token.ParamType)
    {
        case DiagTokenParamType.VibroOporParam:
            {
                switch (Token.Param)
                {
                    case 0:
                        {
                            for (int i = 0; i < FPeriodCount; i++)
                                Array[i]  =  FHistory[i].VibroParamValues[ChannelNumber].EffectiveVibroSpeed  /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain;
                            break;
                        }
                    case 1:
                        {
                            for (int i = 0; i < FPeriodCount; i++)
                                Array[i]  =  FHistory[i].VibroParamValues[ChannelNumber].FirstSpeedGarmonic  /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain;
```

```
                    break;
                }
            case 2:
                {
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i] = FHistory[i].VibroParamValues[ChannelNumber].PhaseFirstSpeedGarmonic;
                    break;
                }
            case 3:
                {
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i] = FHistory[i].VibroParamValues[ChannelNumber].SecondSpeedGarmonic /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain;
                    break;
                }
            case 4:
                {
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i] = FHistory[i].VibroParamValues[ChannelNumber].PhaseSecondSpeedGarmonic;
                    break;
                }
            case 5:
                {
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i] = FHistory[i].VibroParamValues[ChannelNumber].SpeedAmplitude /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain;
                    break;
                }
            case 6:
                {
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i] = FHistory[i].VibroParamValues[ChannelNumber].Garm3 /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain;
                    break;
                }
            case 7:
                {
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i] = FHistory[i].VibroParamValues[ChannelNumber].Garm4 /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain;
                    break;
                }
            case 8:
                {
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i] = FHistory[i].VibroParamValues[ChannelNumber].Garm5 /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain;
                    break;
                }
            case 9:
                {
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i] = FHistory[i].VibroParamValues[ChannelNumber].Garm6 /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain;
```

```
                    break;
                }
            case 10:
                {
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i]                =                FHistory[i].VibroParamValues[ChannelNumber].Garm7                /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain;
                    break;
                }
            case 11:
                {
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i]                =                FHistory[i].VibroParamValues[ChannelNumber].Garm8                /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain;
                    break;
                }
            case 12:
                {
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i]                =                FHistory[i].VibroParamValues[ChannelNumber].Garm9                /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain;
                    break;
                }
            case 13:
                {
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i]                =                FHistory[i].VibroParamValues[ChannelNumber].Clearance                /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain;
                    break;
                }
            case 14:
                {
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i]        =        FHistory[i].VibroParamValues[ChannelNumber].HighFrequencyComponent        /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain;
                    break;
                }
            case 15:
                {
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i] = FHistory[i].VibroParamValues[ChannelNumber].MaxDisplacement;
                    break;
                }
        }

        switch (Token.Function)
        {
            case DiagTokenFunction.Value:
                {
                    switch(Token.Param)
                    {
                        case 0: Operand1 = FAgt.OporSensors[ChannelNumber].MeasureVibroParams.EffectiveVibroSpeed /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain; break;
```

```
                    case 1: Operand1 = FAgt.OporSensors[ChannelNumber].MeasureVibroParams.FirstSpeedGarmonic /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain; break;
                    case 2: Operand1 =
FAgt.OporSensors[ChannelNumber].MeasureVibroParams.PhaseFirstSpeedGarmonic; break;
                    case 3: Operand1 = FAgt.OporSensors[ChannelNumber].MeasureVibroParams.SecondSpeedGarmonic
/ FAgt.OporSensorsCommonCfg.ArchiveSaveGain; break;
                    case 4: Operand1 =
FAgt.OporSensors[ChannelNumber].MeasureVibroParams.PhaseSecondSpeedGarmonic; break;
                    case 5: Operand1 = FAgt.OporSensors[ChannelNumber].MeasureVibroParams.SpeedAmplitude /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain; break;
                    case 6: Operand1 = FAgt.OporSensors[ChannelNumber].MeasureVibroParams.Garm3 /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain; break;
                    case 7: Operand1 = FAgt.OporSensors[ChannelNumber].MeasureVibroParams.Garm4 /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain; break;
                    case 8: Operand1 = FAgt.OporSensors[ChannelNumber].MeasureVibroParams.Garm5 /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain; break;
                    case 9: Operand1 = FAgt.OporSensors[ChannelNumber].MeasureVibroParams.Garm6 /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain; break;
                    case 10: Operand1 = FAgt.OporSensors[ChannelNumber].MeasureVibroParams.Garm7 /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain; break;
                    case 11: Operand1 = FAgt.OporSensors[ChannelNumber].MeasureVibroParams.Garm8 /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain; break;
                    case 12: Operand1 = FAgt.OporSensors[ChannelNumber].MeasureVibroParams.Garm9 /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain; break;
                    case 13: Operand1 = FAgt.OporSensors[ChannelNumber].MeasureVibroParams.Clearance /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain; break;
                    case 14: Operand1 =
FAgt.OporSensors[ChannelNumber].MeasureVibroParams.HighFrequencyComponent /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain; break;
                    case 15: Operand1 = FAgt.OporSensors[ChannelNumber].MeasureVibroParams.MaxDisplacement /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain; break;
                  } break;
                }
              case DiagTokenFunction.SKO:
                {
                  Operand1 = SKO(Array);
                  break;

                }
              case DiagTokenFunction.Rost:
                {
                  if (Rost(Array, (double)FRostData[ChannelNumber *
FAgt.OporSensorsCommonCfg.VibroParameters.Length + Token.Param].StartValue /
FAgt.OporSensorsCommonCfg.ArchiveSaveGain, (double)FASKVDConfig.OporsUp / 10))
                  {
                    FRostData[ChannelNumber * FAgt.OporSensorsCommonCfg.VibroParameters.Length +
Token.Param].StartValue = (short)(Array[0] * FAgt.OporSensorsCommonCfg.ArchiveSaveGain);
                    if (DateTime.Now.Subtract(FRostData[ChannelNumber *
FAgt.OporSensorsCommonCfg.VibroParameters.Length + Token.Param].Start).TotalSeconds < Token.Value)
                      Operand1 = Token.Value;
                    else
                      Operand1 = double.MinValue;
                    FRostData[ChannelNumber * FAgt.OporSensorsCommonCfg.VibroParameters.Length +
Token.Param].Start = DateTime.Now;
```

```
                    }
                    else
                        Operand1 = double.MinValue;
                    break;
                }
            }
            break;
        }
    case DiagTokenParamType.VibroValParam:
        {
            switch (Token.Param)
            {
                case 0:
                    {
                        for (int i = 0; i < FPeriodCount; i++)
                            Array[i]          =          FHistory[i].VibroParamValues[ChannelNumber].EffectiveVibroSpeed          /
FAgt.ValSensorsCommonCfg.ArchiveSaveGain;
                        break;
                    }
                case 1:
                    {
                        for (int i = 0; i < FPeriodCount; i++)
                            Array[i]          =          FHistory[i].VibroParamValues[ChannelNumber].FirstSpeedGarmonic          /
FAgt.ValSensorsCommonCfg.ArchiveSaveGain;
                        break;
                    }
                case 2:
                    {
                        for (int i = 0; i < FPeriodCount; i++)
                            Array[i] = FHistory[i].VibroParamValues[ChannelNumber].PhaseFirstSpeedGarmonic;
                        break;
                    }
                case 3:
                    {
                        for (int i = 0; i < FPeriodCount; i++)
                            Array[i]          =          FHistory[i].VibroParamValues[ChannelNumber].SecondSpeedGarmonic          /
FAgt.ValSensorsCommonCfg.ArchiveSaveGain;
                        break;
                    }
                case 4:
                    {
                        for (int i = 0; i < FPeriodCount; i++)
                            Array[i] = FHistory[i].VibroParamValues[ChannelNumber].PhaseSecondSpeedGarmonic;
                        break;
                    }
                case 5:
                    {
                        for (int i = 0; i < FPeriodCount; i++)
                            Array[i]          =          FHistory[i].VibroParamValues[ChannelNumber].SpeedAmplitude          /
FAgt.ValSensorsCommonCfg.ArchiveSaveGain;
                        break;
                    }
                case 6:
                    {
```

```
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i]  =  FHistory[i].VibroParamValues[ChannelNumber].Garm3  /
FAgt.ValSensorsCommonCfg.ArchiveSaveGain;
                    break;
                }
            case 7:
                {
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i]  =  FHistory[i].VibroParamValues[ChannelNumber].Garm4  /
FAgt.ValSensorsCommonCfg.ArchiveSaveGain;
                    break;
                }
            case 8:
                {
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i]  =  FHistory[i].VibroParamValues[ChannelNumber].Garm5  /
FAgt.ValSensorsCommonCfg.ArchiveSaveGain;
                    break;
                }
            case 9:
                {
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i]  =  FHistory[i].VibroParamValues[ChannelNumber].Garm6  /
FAgt.ValSensorsCommonCfg.ArchiveSaveGain;
                    break;
                }
            case 10:
                {
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i]  =  FHistory[i].VibroParamValues[ChannelNumber].Garm7  /
FAgt.ValSensorsCommonCfg.ArchiveSaveGain;
                    break;
                }
            case 11:
                {
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i]  =  FHistory[i].VibroParamValues[ChannelNumber].Garm8  /
FAgt.ValSensorsCommonCfg.ArchiveSaveGain;
                    break;
                }
            case 12:
                {
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i]  =  FHistory[i].VibroParamValues[ChannelNumber].Garm9  /
FAgt.ValSensorsCommonCfg.ArchiveSaveGain;
                    break;
                }
            case 13:
                {
                    for (int i = 0; i < FPeriodCount; i++)
                        Array[i]  =  FHistory[i].VibroParamValues[ChannelNumber].Clearance  /
FAgt.ValSensorsCommonCfg.ArchiveSaveGain;
                    break;
                }
```

```
                case 14:
                  {
                    for (int i = 0; i < FPeriodCount; i++)
                      Array[i] = FHistory[i].VibroParamValues[ChannelNumber].HighFrequencyComponent /
FAgt.ValSensorsCommonCfg.ArchiveSaveGain;
                    break;
                  }
                case 15:
                  {
                    for (int i = 0; i < FPeriodCount; i++)
                      Array[i] = FHistory[i].VibroParamValues[ChannelNumber].MaxDisplacement /
FAgt.ValSensorsCommonCfg.ArchiveSaveGain;
                    break;
                  }
              }

              switch (Token.Function)
              {
                case DiagTokenFunction.Value:
                  {
                    switch (Token.Param)
                    {
                      case 0: Operand1 = FAgt.ValSensors[ChannelNumber -
FAgt.OporSensors.Length].MeasureVibroParams.EffectiveVibroSpeed / FAgt.ValSensorsCommonCfg.ArchiveSaveGain; break;
                      case 1: Operand1 = FAgt.ValSensors[ChannelNumber -
FAgt.OporSensors.Length].MeasureVibroParams.FirstSpeedGarmonic / FAgt.ValSensorsCommonCfg.ArchiveSaveGain; break;
                      case 2: Operand1 = FAgt.ValSensors[ChannelNumber -
FAgt.OporSensors.Length].MeasureVibroParams.PhaseFirstSpeedGarmonic; break;
                      case 3: Operand1 = FAgt.ValSensors[ChannelNumber -
FAgt.OporSensors.Length].MeasureVibroParams.SecondSpeedGarmonic / FAgt.ValSensorsCommonCfg.ArchiveSaveGain;
break;
                      case 4: Operand1 = FAgt.ValSensors[ChannelNumber -
FAgt.OporSensors.Length].MeasureVibroParams.PhaseSecondSpeedGarmonic; break;
                      case 5: Operand1 = FAgt.ValSensors[ChannelNumber -
FAgt.OporSensors.Length].MeasureVibroParams.SpeedAmplitude / FAgt.ValSensorsCommonCfg.ArchiveSaveGain; break;
                      case 6: Operand1 = FAgt.ValSensors[ChannelNumber -
FAgt.OporSensors.Length].MeasureVibroParams.Garm3 / FAgt.ValSensorsCommonCfg.ArchiveSaveGain; break;
                      case 7: Operand1 = FAgt.ValSensors[ChannelNumber -
FAgt.OporSensors.Length].MeasureVibroParams.Garm4 / FAgt.ValSensorsCommonCfg.ArchiveSaveGain; break;
                      case 8: Operand1 = FAgt.ValSensors[ChannelNumber -
FAgt.OporSensors.Length].MeasureVibroParams.Garm5 / FAgt.ValSensorsCommonCfg.ArchiveSaveGain; break;
                      case 9: Operand1 = FAgt.ValSensors[ChannelNumber -
FAgt.OporSensors.Length].MeasureVibroParams.Garm6 / FAgt.ValSensorsCommonCfg.ArchiveSaveGain; break;
                      case 10: Operand1 = FAgt.ValSensors[ChannelNumber -
FAgt.OporSensors.Length].MeasureVibroParams.Garm7 / FAgt.ValSensorsCommonCfg.ArchiveSaveGain; break;
                      case 11: Operand1 = FAgt.ValSensors[ChannelNumber -
FAgt.OporSensors.Length].MeasureVibroParams.Garm8 / FAgt.ValSensorsCommonCfg.ArchiveSaveGain; break;
                      case 12: Operand1 = FAgt.ValSensors[ChannelNumber -
FAgt.OporSensors.Length].MeasureVibroParams.Garm9 / FAgt.ValSensorsCommonCfg.ArchiveSaveGain; break;
                      case 13: Operand1 = FAgt.ValSensors[ChannelNumber -
FAgt.OporSensors.Length].MeasureVibroParams.Clearance / FAgt.ValSensorsCommonCfg.ArchiveSaveGain; break;
                      case 14: Operand1 = FAgt.ValSensors[ChannelNumber -
FAgt.OporSensors.Length].MeasureVibroParams.HighFrequencyComponent / FAgt.ValSensorsCommonCfg.ArchiveSaveGain;
break;
```

```
                    case 15: Operand1 = FAgt.ValSensors[ChannelNumber - FAgt.OporSensors.Length].MeasureVibroParams.MaxDisplacement / FAgt.ValSensorsCommonCfg.ArchiveSaveGain; break;
                  } break;
                }
              case DiagTokenFunction.SKO:
                {
                  Operand1 = SKO(Array);
                  break;

                }
              case DiagTokenFunction.Rost:
                {
                  int Index = FAgt.OporSensorsCommonCfg.VibroParameters.Length * FAgt.OporSensors.Length + (ChannelNumber - FAgt.OporSensors.Length) * FAgt.ValSensorsCommonCfg.VibroParameters.Length + Token.Param;
                  if (Rost(Array, (double)FRostData[Index].StartValue / FAgt.ValSensorsCommonCfg.ArchiveSaveGain, (double)FASKVDConfig.ValsUp))
                  {
                    FRostData[Index].StartValue = (short)(Array[0] * FAgt.ValSensorsCommonCfg.ArchiveSaveGain);
                    if (DateTime.Now.Subtract(FRostData[Index].Start).TotalSeconds < Token.Value)
                      Operand1 = Token.Value;
                    else
                      Operand1 = double.MinValue;
                    FRostData[Index].Start = DateTime.Now;
                  }
                  else
                    Operand1 = double.MinValue;
                  break;
                }
            }
            break;
          }
        case DiagTokenParamType.ExplParam:
          {
            for (int i = 0; i < FPeriodCount; i++)
              Array[i] = FHistory[i].ExplParamValues[ChannelNumber];

            switch (Token.Function)
            {
              case DiagTokenFunction.Value: Operand1 = FAgt.ExplParams[ChannelNumber].Value; break;
              case DiagTokenFunction.SKO: Operand1 = SKO(Array); break;
            }
            break;
          }
        case DiagTokenParamType.Special: Operand1 = ChannelNumber; break;
        default: return false;
      }

      switch (Token.ValType)
      {
        case DiagTokenValueType.SetPoint:
          {
            switch (Token.ParamType)
            {
              case DiagTokenParamType.VibroOporParam:
```

```
    {
      ModeVibroSetPoints Current = null;
      switch (FAgt.Mode)
      {
        case FrequencyModeType.Nominal: Current = FNominalSetPoints; break;
        case FrequencyModeType.Up: Current = FUpSetPoints; break;
        case FrequencyModeType.Down: Current = FUpSetPoints; break;
        case FrequencyModeType.ValRotation: Current = FValRotationSetPoints; break;
        case FrequencyModeType.Stop: Current = FStopSetPoints; break;
        default: Operand2 = Token.Value; break;
      }
      if (Current != null)
      {
        switch ((int)Token.Value)
        {
          case 1:
            {
              Operand2 = Current.OporSetPoints[Token.Param].Value1 / 10; break;
            }
          case 2:
            {
              Operand2 = Current.OporSetPoints[Token.Param].Value2 / 10; break;
            }
          case 3:
            {
              Operand2 = Current.OporSetPoints[Token.Param].Value3 / 10; break;
            }
          case 4:
            {
              Operand2 = Current.OporSetPoints[Token.Param].SqrtDelta / 10; break;
            }
          case 5:
            {
              Operand2 = Current.OporSetPoints[Token.Param].Jump / 10; break;
            }
          default: Operand2 = Token.Value; break;
        }
      }
      break;
    }
  case DiagTokenParamType.VibroValParam:
    {
      ModeVibroSetPoints Current = null;
      switch (FAgt.Mode)
      {
        case FrequencyModeType.Nominal: Current = FNominalSetPoints; break;
        case FrequencyModeType.Up: Current = FUpSetPoints; break;
        case FrequencyModeType.Down: Current = FUpSetPoints; break;
        case FrequencyModeType.ValRotation: Current = FValRotationSetPoints; break;
        case FrequencyModeType.Stop: Current = FStopSetPoints; break;
        default: Operand2 = Token.Value; break;
      }
      if (Current != null)
      {
```

```
                    switch ((int)Token.Value)
                    {
                      case 1:
                        {
                          Operand2 = Current.ValSetPoints[Token.Param].Value1; break;
                        }
                      case 2:
                        {
                          Operand2 = Current.ValSetPoints[Token.Param].Value2; break;
                        }
                      case 3:
                        {
                          Operand2 = Current.ValSetPoints[Token.Param].Value3; break;
                        }
                      case 4:
                        {
                          Operand2 = Current.ValSetPoints[Token.Param].SqrtDelta; break;
                        }
                      case 5:
                        {
                          Operand2 = Current.ValSetPoints[Token.Param].Jump; break;
                        }
                      default: Operand2 = Token.Value; break;
                    }
                  }
                  break;
                }
              case DiagTokenParamType.ExplParam:
                {
                  foreach (ExplParameter Param in FAgt.ExplParams)
                  {
                    if (Param.ID == Token.Param)
                    {
                      switch ((int)Token.Value - 1)
                      {
                        case 0: Operand2 = Param.MinWarning; break;
                        case 1: Operand2 = Param.MaxWarning; break;
                        case 2: Operand2 = Param.MinAlarm; break;
                        case 3: Operand2 = Param.MaxAlarm; break;
                        default: Operand2 = Token.Value; break;
                      }
                      break;
                    }
                  }
                  break;
                }
              case DiagTokenParamType.Special: Operand2 = Token.Value; break;
            }
            break;
          }
        case DiagTokenValueType.Value: Operand2 = Token.Value; break;
        default: return false;
      }
```

```
if (Token.Function == DiagTokenFunction.Jump)
   Result = Jump(Array, Operand2);
else
{
   switch (Token.Compare)
   {
      case CompareType.Equal:
         {
            if (Operand1 == Operand2)
               Result = true;
            else
               Result = false;
            break;
         }
      case CompareType.LessOrEqual:
         {
            if (Operand1 <= Operand2)
               Result = true;
            else
               Result = false;
            break;
         }
      case CompareType.LessThan:
         {
            if (Operand1 < Operand2)
               Result = true;
            else
               Result = false;
            break;
         }
      case CompareType.MoreOrEqual:
         {
            if (Operand1 >= Operand2)
               Result = true;
            else
               Result = false;
            break;
         }
      case CompareType.MoreThan:
         {
            if (Operand1 > Operand2)
               Result = true;
            else
               Result = false;
            break;
         }
      case CompareType.NotEqual:
         {
            if (Operand1 != Operand2)
               Result = true;
            else
               Result = false;
            break;
         }
```

```
            default: Result = false; break;
        }
    }
    return Result;
}

private double SKO(double[] Array)
{
    int Count = Array.Length;
    if (Count <= 0)
        return 0;
    double Avg = 0;
    for (int i = 0; i < Count; i++)
        Avg += Array[i];
    Avg = Avg / Count;
    double Result = 0;
    for (int i = 0; i < Count; i++)
        Result += Math.Pow(Avg - Array[i], 2);
    return Math.Sqrt(Result);
}

private bool Rost(double[] Array, double Value, double SetPoint)
{
    bool Exist = true;

    if (Array == null)
        return false;
    int Count = Array.Length;
    if (Count <= 0)
        return false;
    for (int k = 0; k < Count; k++)
    {
        if (Array[k] - Value <= SetPoint)
        {
            Exist = false;
            break;
            // Нет привышение уставки монотонного роста
        }
    }
    return Exist;
}

private bool Jump(double[] OldArray, double SetPoint)
{
    int Count = OldArray.Length;
    if (Count <= 0)
        return false;

    bool Result = true;
    for(int i = 0; i < Count - 1; i++)
        if (Math.Abs(OldArray[0] - OldArray[i + 1]) < SetPoint)
        {
            Result = false;
            break;
```

```
        }
    return Result;
}

private double Abs(double A1, double F1, double A2, double F2)
{
    double Phase1 = F1 * Math.PI / 180;
    double Phase2 = F2 * Math.PI / 180;
    double X1, X2, Y1, Y2;
    X1 = A1 * Math.Cos(Phase1);
    Y1 = A1 * Math.Sin(Phase1);
    X2 = A2 * Math.Cos(Phase2);
    Y2 = A2 * Math.Sin(Phase2);
    return Math.Sqrt(Math.Pow(X1 - X2, 2) + Math.Pow(Y1 - Y2, 2));
}

private void CrackDiagnostic()
{

}

private ushort GetLeftOpor(Rotor Rotor1)
{
    ushort Number = 0;
    foreach (Rotor rt in FAgt.Rotors)
    {
        if (rt.LeftOpor == OporType.Common)
            Number++;
        else
            Number += 2;
        if (rt == Rotor1)
            break;
    }
    return (ushort)(Number - 1);
}

private void MakeRostData()
{
    if (Math.Abs(FPowerAtStartRost - FAgt.CurrentPower) / FPowerAtStartRost * 100 > FASKVDConfig.MaxPowerDelta)
    {
        // Превышено отклонение нагрузки
        FPowerAtStartRost = FAgt.CurrentPower;
        int Count = FRostData.Length;
        for (int i = 0; i < Count; i++)
            FRostData[i].Start = DateTime.Now;

        // Запоминаем значения при новом режиме по нагрузке
        Count = FAgt.OporSensorsCommonCfg.VibroParameters.Length;
        for (int i = 0; i < FAgt.OporSensors.Length; i++)
        {
            for (int j = 0; j < Count; j++)
            {
                switch (FAgt.OporSensorsCommonCfg.VibroParameters[j])
                {
```

```
                case VibroOporParam.Ve: FRostData[i * Count + j].StartValue = FAgt.OporSensors[i].MeasureVibroParams.EffectiveVibroSpeed; break;
                case VibroOporParam.V1: FRostData[i * Count + j].StartValue = FAgt.OporSensors[i].MeasureVibroParams.FirstSpeedGarmonic; break;
                case VibroOporParam.FV1: FRostData[i * Count + j].StartValue = FAgt.OporSensors[i].MeasureVibroParams.PhaseFirstSpeedGarmonic; break;
                case VibroOporParam.V2: FRostData[i * Count + j].StartValue = FAgt.OporSensors[i].MeasureVibroParams.SecondSpeedGarmonic; break;
                case VibroOporParam.FV2: FRostData[i * Count + j].StartValue = FAgt.OporSensors[i].MeasureVibroParams.PhaseSecondSpeedGarmonic; break;
                case VibroOporParam.ncv: FRostData[i * Count + j].StartValue = FAgt.OporSensors[i].MeasureVibroParams.SpeedAmplitude; break;
                case VibroOporParam.garm3: FRostData[i * Count + j].StartValue = FAgt.OporSensors[i].MeasureVibroParams.Garm3; break;
                case VibroOporParam.garm4: FRostData[i * Count + j].StartValue = FAgt.OporSensors[i].MeasureVibroParams.Garm4; break;
                case VibroOporParam.garm5: FRostData[i * Count + j].StartValue = FAgt.OporSensors[i].MeasureVibroParams.Garm5; break;
                case VibroOporParam.garm6: FRostData[i * Count + j].StartValue = FAgt.OporSensors[i].MeasureVibroParams.Garm6; break;
                case VibroOporParam.garm7: FRostData[i * Count + j].StartValue = FAgt.OporSensors[i].MeasureVibroParams.Garm7; break;
                case VibroOporParam.garm8: FRostData[i * Count + j].StartValue = FAgt.OporSensors[i].MeasureVibroParams.Garm8; break;
                case VibroOporParam.garm9: FRostData[i * Count + j].StartValue = FAgt.OporSensors[i].MeasureVibroParams.Garm9; break;
                case VibroOporParam.Zmin: FRostData[i * Count + j].StartValue = FAgt.OporSensors[i].MeasureVibroParams.Clearance; break;
                case VibroOporParam.vcv: FRostData[i * Count + j].StartValue = FAgt.OporSensors[i].MeasureVibroParams.HighFrequencyComponent; break;
                case VibroOporParam.Smax: FRostData[i * Count + j].StartValue = FAgt.OporSensors[i].MeasureVibroParams.MaxDisplacement; break;
            }
        }
    }

    Count = FAgt.ValSensorsCommonCfg.VibroParameters.Length;
    int Displacement = FAgt.OporSensors.Length * FAgt.OporSensorsCommonCfg.VibroParameters.Length;
    for (int i = 0; i < FAgt.ValSensors.Length; i++)
    {
        for (int j = 0; j < Count; j++)
        {
            switch (FAgt.ValSensorsCommonCfg.VibroParameters[j])
            {
                case VibroValParam.S: FRostData[i * Count + Displacement + j].StartValue = FAgt.ValSensors[i].MeasureVibroParams.EffectiveVibroSpeed; break;
                case VibroValParam.S1: FRostData[i * Count + Displacement + j].StartValue = FAgt.ValSensors[i].MeasureVibroParams.FirstSpeedGarmonic; break;
                case VibroValParam.FS1: FRostData[i * Count + Displacement + j].StartValue = FAgt.ValSensors[i].MeasureVibroParams.PhaseFirstSpeedGarmonic; break;
                case VibroValParam.S2: FRostData[i * Count + Displacement + j].StartValue = FAgt.ValSensors[i].MeasureVibroParams.SecondSpeedGarmonic; break;
                case VibroValParam.FS2: FRostData[i * Count + Displacement + j].StartValue = FAgt.ValSensors[i].MeasureVibroParams.PhaseSecondSpeedGarmonic; break;
```

```
                case VibroValParam.ncv: FRostData[i * Count + Displacement + j].StartValue = FAgt.ValSensors[i].MeasureVibroParams.SpeedAmplitude; break;
                case VibroValParam.garm3: FRostData[i * Count + Displacement + j].StartValue = FAgt.ValSensors[i].MeasureVibroParams.Garm3; break;
                case VibroValParam.garm4: FRostData[i * Count + Displacement + j].StartValue = FAgt.ValSensors[i].MeasureVibroParams.Garm4; break;
                case VibroValParam.garm5: FRostData[i * Count + Displacement + j].StartValue = FAgt.ValSensors[i].MeasureVibroParams.Garm5; break;
                case VibroValParam.garm6: FRostData[i * Count + Displacement + j].StartValue = FAgt.ValSensors[i].MeasureVibroParams.Garm6; break;
                case VibroValParam.garm7: FRostData[i * Count + Displacement + j].StartValue = FAgt.ValSensors[i].MeasureVibroParams.Garm7; break;
                case VibroValParam.garm8: FRostData[i * Count + Displacement + j].StartValue = FAgt.ValSensors[i].MeasureVibroParams.Garm8; break;
                case VibroValParam.garm9: FRostData[i * Count + Displacement + j].StartValue = FAgt.ValSensors[i].MeasureVibroParams.Garm9; break;
                case VibroValParam.Zmin: FRostData[i * Count + Displacement + j].StartValue = FAgt.ValSensors[i].MeasureVibroParams.Clearance; break;
                case VibroValParam.zaz: FRostData[i * Count + Displacement + j].StartValue = FAgt.ValSensors[i].MeasureVibroParams.HighFrequencyComponent; break;
                case VibroValParam.Smax: FRostData[i * Count + Displacement + j].StartValue = FAgt.ValSensors[i].MeasureVibroParams.MaxDisplacement; break;
            }
        }
    }
}
else
{
    if (FHistory == null)
        return;
    for(int i = 0; i < FPeriodCount; i++)
    {
        if((FHistory[i] == null) || (FHistory[i].VibroParamValues == null))
            return; // Нельзя оценить рост без истории
    }

    int ParamCount = FAgt.OporSensorsCommonCfg.VibroParameters.Length;
    for (int i = 0; i < FAgt.OporSensors.Length; i++)
    {
        for (int j = 0; j < ParamCount; j++)
        {
            switch (FAgt.OporSensorsCommonCfg.VibroParameters[j])
            {
                case VibroOporParam.Ve:
                    {
                        if (FHistory[0].VibroParamValues[i].EffectiveVibroSpeed <= FRostData[i * ParamCount + j].StartValue)
                        {
                            // На этом участке вибрация убывает
                            FRostData[i * ParamCount + j].StartValue = FHistory[0].VibroParamValues[i].EffectiveVibroSpeed;
                            FRostData[i * ParamCount + j].Start = DateTime.Now;
                        }
                        break;
                    }
                case VibroOporParam.V1:
```

```
                    {
                        if (FHistory[0].VibroParamValues[i].FirstSpeedGarmonic <= FRostData[i * ParamCount + j].StartValue)
                        {
                            // На этом участке вибрация убывает
                            FRostData[i * ParamCount + j].StartValue = FHistory[0].VibroParamValues[i].FirstSpeedGarmonic;
                            FRostData[i * ParamCount + j].Start = DateTime.Now;
                        }
                        break;
                    }
                case VibroOporParam.FV1:
                    {
                        if (FHistory[0].VibroParamValues[i].PhaseFirstSpeedGarmonic <= FRostData[i * ParamCount +
j].StartValue)
                        {
                            // На этом участке вибрация убывает
                            FRostData[i * ParamCount + j].StartValue =
FHistory[0].VibroParamValues[i].PhaseFirstSpeedGarmonic;
                            FRostData[i * ParamCount + j].Start = DateTime.Now;
                        }
                        break;
                    }
                case VibroOporParam.V2:
                    {
                        if (FHistory[0].VibroParamValues[i].SecondSpeedGarmonic <= FRostData[i * ParamCount + j].StartValue)
                        {
                            // На этом участке вибрация убывает
                            FRostData[i * ParamCount + j].StartValue = FHistory[0].VibroParamValues[i].SecondSpeedGarmonic;
                            FRostData[i * ParamCount + j].Start = DateTime.Now;
                        }
                        break;
                    }
                case VibroOporParam.FV2:
                    {
                        if (FHistory[0].VibroParamValues[i].PhaseSecondSpeedGarmonic <= FRostData[i * ParamCount +
j].StartValue)
                        {
                            // На этом участке вибрация убывает
                            FRostData[i * ParamCount + j].StartValue =
FHistory[0].VibroParamValues[i].PhaseSecondSpeedGarmonic;
                            FRostData[i * ParamCount + j].Start = DateTime.Now;
                        }
                        break;
                    }
                case VibroOporParam.ncv:
                    {
                        if (FHistory[0].VibroParamValues[i].SpeedAmplitude <= FRostData[i * ParamCount + j].StartValue)
                        {
                            // На этом участке вибрация убывает
                            FRostData[i * ParamCount + j].StartValue = FHistory[0].VibroParamValues[i].SpeedAmplitude;
                            FRostData[i * ParamCount + j].Start = DateTime.Now;
                        }
                        break;
                    }
                case VibroOporParam.garm3:
```

```
    {
        if (FHistory[0].VibroParamValues[i].Garm3 <= FRostData[i * ParamCount + j].StartValue)
        {
            // На этом участке вибрация убывает
            FRostData[i * ParamCount + j].StartValue = FHistory[0].VibroParamValues[i].Garm3;
            FRostData[i * ParamCount + j].Start = DateTime.Now;
        }
        break;
    }
case VibroOporParam.garm4:
    {
        if (FHistory[0].VibroParamValues[i].Garm4 <= FRostData[i * ParamCount + j].StartValue)
        {
            // На этом участке вибрация убывает
            FRostData[i * ParamCount + j].StartValue = FHistory[0].VibroParamValues[i].Garm4;
            FRostData[i * ParamCount + j].Start = DateTime.Now;
        }
        break;
    }
case VibroOporParam.garm5:
    {
        if (FHistory[0].VibroParamValues[i].Garm5 <= FRostData[i * ParamCount + j].StartValue)
        {
            // На этом участке вибрация убывает
            FRostData[i * ParamCount + j].StartValue = FHistory[0].VibroParamValues[i].Garm5;
            FRostData[i * ParamCount + j].Start = DateTime.Now;
        }
        break;
    }
case VibroOporParam.garm6:
    {
        if (FHistory[0].VibroParamValues[i].Garm6 <= FRostData[i * ParamCount + j].StartValue)
        {
            // На этом участке вибрация убывает
            FRostData[i * ParamCount + j].StartValue = FHistory[0].VibroParamValues[i].Garm6;
            FRostData[i * ParamCount + j].Start = DateTime.Now;
        }
        break;
    }
case VibroOporParam.garm7:
    {
        if (FHistory[0].VibroParamValues[i].Garm7 <= FRostData[i * ParamCount + j].StartValue)
        {
            // На этом участке вибрация убывает
            FRostData[i * ParamCount + j].StartValue = FHistory[0].VibroParamValues[i].Garm7;
            FRostData[i * ParamCount + j].Start = DateTime.Now;
        }
        break;
    }
case VibroOporParam.garm8:
    {
        if (FHistory[0].VibroParamValues[i].Garm8 <= FRostData[i * ParamCount + j].StartValue)
        {
            // На этом участке вибрация убывает
```

```
                    FRostData[i * ParamCount + j].StartValue = FHistory[0].VibroParamValues[i].Garm8;
                    FRostData[i * ParamCount + j].Start = DateTime.Now;
                  }
                  break;
                }
              case VibroOporParam.garm9:
                {
                  if (FHistory[0].VibroParamValues[i].Garm9 <= FRostData[i * ParamCount + j].StartValue)
                  {
                    // На этом участке вибрация убывает
                    FRostData[i * ParamCount + j].StartValue = FHistory[0].VibroParamValues[i].Garm9;
                    FRostData[i * ParamCount + j].Start = DateTime.Now;
                  }
                  break;
                }
              case VibroOporParam.Zmin:
                {
                  if (FHistory[0].VibroParamValues[i].Clearance <= FRostData[i * ParamCount + j].StartValue)
                  {
                    // На этом участке вибрация убывает
                    FRostData[i * ParamCount + j].StartValue = FHistory[0].VibroParamValues[i].Clearance;
                    FRostData[i * ParamCount + j].Start = DateTime.Now;
                  }
                  break;
                }
              case VibroOporParam.vcv:
                {
                  if (FHistory[0].VibroParamValues[i].HighFrequencyComponent <= FRostData[i * ParamCount +
j].StartValue)
                  {
                    // На этом участке вибрация убывает
                    FRostData[i * ParamCount + j].StartValue =
FHistory[0].VibroParamValues[i].HighFrequencyComponent;
                    FRostData[i * ParamCount + j].Start = DateTime.Now;
                  }
                  break;
                }
              case VibroOporParam.Smax:
                {
                  if (FHistory[0].VibroParamValues[i].MaxDisplacement <= FRostData[i * ParamCount + j].StartValue)
                  {
                    // На этом участке вибрация убывает
                    FRostData[i * ParamCount + j].StartValue = FHistory[0].VibroParamValues[i].MaxDisplacement;
                    FRostData[i * ParamCount + j].Start = DateTime.Now;
                  }
                  break;
                }
            }
          }
        }

        ParamCount = FAgt.ValSensorsCommonCfg.VibroParameters.Length;
        int Seek = FAgt.OporSensors.Length + FAgt.OporSensorsCommonCfg.VibroParameters.Length;
```

```
            for (int i = 0; i < FAgt.ValSensors.Length; i++)
            {
              for (int j = 0; j < ParamCount; j++)
              {
                switch (FAgt.ValSensorsCommonCfg.VibroParameters[j])
                {
                  case VibroValParam.S:
                    {
                      if (FHistory[0].VibroParamValues[i + FAgt.OporSensors.Length].EffectiveVibroSpeed <= FRostData[i *
ParamCount + Seek + j].StartValue)
                      {
                        // На этом участке вибрация убывает
                        FRostData[i * ParamCount + Seek + j].StartValue = FHistory[0].VibroParamValues[i +
FAgt.OporSensors.Length].EffectiveVibroSpeed;
                        FRostData[i * ParamCount + Seek + j].Start = DateTime.Now;
                      }
                      break;
                    }
                  case VibroValParam.S1:
                    {
                      if (FHistory[0].VibroParamValues[i + FAgt.OporSensors.Length].FirstSpeedGarmonic <= FRostData[i *
ParamCount + Seek + j].StartValue)
                      {
                        // На этом участке вибрация убывает
                        FRostData[i * ParamCount + Seek + j].StartValue = FHistory[0].VibroParamValues[i +
FAgt.OporSensors.Length].FirstSpeedGarmonic;
                        FRostData[i * ParamCount + Seek + j].Start = DateTime.Now;
                      }
                      break;
                    }
                  case VibroValParam.FS1:
                    {
                      if (FHistory[0].VibroParamValues[i + FAgt.OporSensors.Length].PhaseFirstSpeedGarmonic <= FRostData[i
* ParamCount + Seek + j].StartValue)
                      {
                        // На этом участке вибрация убывает
                        FRostData[i * ParamCount + Seek + j].StartValue = FHistory[0].VibroParamValues[i +
FAgt.OporSensors.Length].PhaseFirstSpeedGarmonic;
                        FRostData[i * ParamCount + Seek + j].Start = DateTime.Now;
                      }
                      break;
                    }
                  case VibroValParam.S2:
                    {
                      if (FHistory[0].VibroParamValues[i + FAgt.OporSensors.Length].SecondSpeedGarmonic <= FRostData[i *
ParamCount + Seek + j].StartValue)
                      {
                        // На этом участке вибрация убывает
                        FRostData[i * ParamCount + Seek + j].StartValue = FHistory[0].VibroParamValues[i +
FAgt.OporSensors.Length].SecondSpeedGarmonic;
                        FRostData[i * ParamCount + Seek + j].Start = DateTime.Now;
                      }
                      break;
                    }
```

```
                case VibroValParam.FS2:
                  {
                    if (FHistory[0].VibroParamValues[i + FAgt.OporSensors.Length].PhaseSecondSpeedGarmonic <=
FRostData[i * ParamCount + Seek + j].StartValue)
                    {
                      // На этом участке вибрация убывает
                      FRostData[i * ParamCount + Seek + j].StartValue = FHistory[0].VibroParamValues[i +
FAgt.OporSensors.Length].PhaseSecondSpeedGarmonic;
                      FRostData[i * ParamCount + Seek + j].Start = DateTime.Now;
                    }
                    break;
                  }
                case VibroValParam.ncv:
                  {
                    if (FHistory[0].VibroParamValues[i + FAgt.OporSensors.Length].SpeedAmplitude <= FRostData[i *
ParamCount + Seek + j].StartValue)
                    {
                      // На этом участке вибрация убывает
                      FRostData[i * ParamCount + Seek + j].StartValue = FHistory[0].VibroParamValues[i +
FAgt.OporSensors.Length].SpeedAmplitude;
                      FRostData[i * ParamCount + Seek + j].Start = DateTime.Now;
                    }
                    break;
                  }
                case VibroValParam.garm3:
                  {
                    if (FHistory[0].VibroParamValues[i + FAgt.OporSensors.Length].Garm3 <= FRostData[i * ParamCount +
Seek + j].StartValue)
                    {
                      // На этом участке вибрация убывает
                      FRostData[i * ParamCount + Seek + j].StartValue = FHistory[0].VibroParamValues[i +
FAgt.OporSensors.Length].Garm3;
                      FRostData[i * ParamCount + Seek + j].Start = DateTime.Now;
                    }
                    break;
                  }
                case VibroValParam.garm4:
                  {
                    if (FHistory[0].VibroParamValues[i + FAgt.OporSensors.Length].Garm4 <= FRostData[i * ParamCount +
Seek + j].StartValue)
                    {
                      // На этом участке вибрация убывает
                      FRostData[i * ParamCount + Seek + j].StartValue = FHistory[0].VibroParamValues[i +
FAgt.OporSensors.Length].Garm4;
                      FRostData[i * ParamCount + Seek + j].Start = DateTime.Now;
                    }
                    break;
                  }
                case VibroValParam.garm5:
                  {
                    if (FHistory[0].VibroParamValues[i + FAgt.OporSensors.Length].Garm5 <= FRostData[i * ParamCount +
Seek + j].StartValue)
                    {
                      // На этом участке вибрация убывает
```

```
                    FRostData[i * ParamCount + Seek + j].StartValue = FHistory[0].VibroParamValues[i +
FAgt.OporSensors.Length].Garm5;
                    FRostData[i * ParamCount + Seek + j].Start = DateTime.Now;
                  }
                  break;
                }
              case VibroValParam.garm6:
                {
                  if (FHistory[0].VibroParamValues[i + FAgt.OporSensors.Length].Garm6 <= FRostData[i * ParamCount +
Seek + j].StartValue)
                  {
                    // На этом участке вибрация убывает
                    FRostData[i * ParamCount + Seek + j].StartValue = FHistory[0].VibroParamValues[i +
FAgt.OporSensors.Length].Garm6;
                    FRostData[i * ParamCount + Seek + j].Start = DateTime.Now;
                  }
                  break;
                }
              case VibroValParam.garm7:
                {
                  if (FHistory[0].VibroParamValues[i + FAgt.OporSensors.Length].Garm7 <= FRostData[i * ParamCount +
Seek + j].StartValue)
                  {
                    // На этом участке вибрация убывает
                    FRostData[i * ParamCount + Seek + j].StartValue = FHistory[0].VibroParamValues[i +
FAgt.OporSensors.Length].Garm7;
                    FRostData[i * ParamCount + Seek + j].Start = DateTime.Now;
                  }
                  break;
                }
              case VibroValParam.garm8:
                {
                  if (FHistory[0].VibroParamValues[i + FAgt.OporSensors.Length].Garm8 <= FRostData[i * ParamCount +
Seek + j].StartValue)
                  {
                    // На этом участке вибрация убывает
                    FRostData[i * ParamCount + Seek + j].StartValue = FHistory[0].VibroParamValues[i +
FAgt.OporSensors.Length].Garm8;
                    FRostData[i * ParamCount + Seek + j].Start = DateTime.Now;
                  }
                  break;
                }
              case VibroValParam.garm9:
                {
                  if (FHistory[0].VibroParamValues[i + FAgt.OporSensors.Length].Garm9 <= FRostData[i * ParamCount +
Seek + j].StartValue)
                  {
                    // На этом участке вибрация убывает
                    FRostData[i * ParamCount + Seek + j].StartValue = FHistory[0].VibroParamValues[i +
FAgt.OporSensors.Length].Garm9;
                    FRostData[i * ParamCount + Seek + j].Start = DateTime.Now;
                  }
                  break;
                }
```

```
                case VibroValParam.zaz:
                  {
                    if (FHistory[0].VibroParamValues[i + FAgt.OporSensors.Length].Clearance <= FRostData[i * ParamCount
+ Seek + j].StartValue)
                    {
                      // На этом участке вибрация убывает
                      FRostData[i * ParamCount + Seek + j].StartValue = FHistory[0].VibroParamValues[i +
FAgt.OporSensors.Length].Clearance;
                      FRostData[i * ParamCount + Seek + j].Start = DateTime.Now;
                    }
                    break;
                  }
                case VibroValParam.Zmin:
                  {
                    if (FHistory[0].VibroParamValues[i + FAgt.OporSensors.Length].HighFrequencyComponent <=
FRostData[i * ParamCount + Seek + j].StartValue)
                    {
                      // На этом участке вибрация убывает
                      FRostData[i * ParamCount + Seek + j].StartValue = FHistory[0].VibroParamValues[i +
FAgt.OporSensors.Length].HighFrequencyComponent;
                      FRostData[i * ParamCount + Seek + j].Start = DateTime.Now;
                    }
                    break;
                  }
                case VibroValParam.Smax:
                  {
                    if (FHistory[0].VibroParamValues[i + FAgt.OporSensors.Length].MaxDisplacement <= FRostData[i *
ParamCount + Seek + j].StartValue)
                    {
                      // На этом участке вибрация убывает
                      FRostData[i * ParamCount + Seek + j].StartValue = FHistory[0].VibroParamValues[i +
FAgt.OporSensors.Length].MaxDisplacement;
                      FRostData[i * ParamCount + Seek + j].Start = DateTime.Now;
                    }
                    break;
}
              }
            }
          }
```

Додаток В1. Апробація (Тези)

УДК 62-799

Магістрант 5 курсу, гр. ТО-81мп Резник Д.О.
Ст.викл. Поліщук І.А.

ПРОБЛЕМАТИКА ВПРОВАДЖЕННЯ СИСТЕМ АВТОМАТИЗОВАНОГО ОБСЛУГОВУВАННЯ ТЕХНОЛОГІЧНОГО ОБЛАДНАННЯ

ІТ-система для надійної роботи технологічного обладнання – це сервіс для автоматизованого обслуговування технологічного обладнання, що дозволяє підвищити його надійність та знизити витрати на його обслуговування. Використання такої системи орієнтоване на:

- підвищення виробничих параметрів обладнання без збільшення витрат;
- зменшення витрат на технічне обслуговування обладнання;
- своєчасне обслуговування, ремонт і матеріально-технічне забезпечення обладнання без зниження рівня надійності;
- забезпечення постійної технічної готовності обладнання до роботи.

Реалізація цих задач здійснюється за допомогою інформаційної системи, що забезпечує підтримку процесу моніторингу та оцінки технічного стану обладнання, що засноване на RCM (Reliability-centered Maintenance – Технічне обслуговування, орієнтоване на надійність) методології, що дозволяє визначити необхідні заходи для того, щоб кожна виробнича система і її елементи виконували покладену на них функцію в рамках виробничого процесу.

Складність RCM-аналізу полягає в тому, що він виконується групою експертів, при чому, для аналізу кожної окремої системи необхідно зібрати до десяти спеціалістів, які керуючись власним професійним досвідом роботи в даній області та чинною нормативною документацією повинні визначити перелік можливих функціональних відмов, ознак, що дозволяють визначити факт виникнення відмов, і причин виникнення відмов, а також визначити і описати наслідки відмов. Тобто, для аналізу кожного об'єкту необхідно знов збирати експертів тому, що при різних умовах функціонування навіть для об'єктів, ідентичних з технічної точки зору, можуть істотно різнитися функції і вимоги до продуктивності, види відмов і результати наслідків відмов, а також оперативні заходи в разі відмови. Також проблемою RCM-аналізу є те, що його неможливо реалізувати без достовірних і повних даних по обладнанню та виникаючим дефектам.

Перелік посилань:

1. ООО «СОВРЕМЕННЫЕ СИСТЕМЫ УПРАВЛЕНИЯ» [Електронний ресурс]: «Методология RCM-анализа» - Режим доступу: https://sov-system.ru/files/RCM-analysis.pdf
2. SmartEAM [Електронний ресурс]: «Планово-попереджувальна система обслуговування» - Режим доступу: https://smart-eam.com/ua/news/planovo-predupreditelnaja-sistema-obsluzhivanija/
3. Діловий портал «Управление производством». [Електроний ресурс]: «ФОРМИРОВАНИЕ ОПТИМАЛЬНОЙ ПРОГРАММЫ ОБСЛУЖИВАНИЯ ОБОРУДОВАНИЯ С ИСПОЛЬЗОВАНИЕМ RCM-АНАЛИЗА» - Режим доступу: http://www.up-pro.ru/library/information_systems/toir/rcm-analiz.htm

Додаток В2. Апробація (Стаття)

УДК 62-799

СИСТЕМА ДІАГНОСТИКИ, ПРОГНОЗУВАННЯ ВІДМОВ ТА ОЦІНКИ СТАНУ ТУРБОАГРЕГАТУ МЕТОДОМ ВІБРОДІАГНОСТИКИ

Резник Д.О., Поліщук І.А.

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Анотації

Досліджено особливості та проблематика обслуговування турбоустановок. Досліджено основні методи вібродіагностики. Досліджено перелік несправностей турбоустановок згідно отриманих даних з датчиків вібрації. Досліджено основні проблеми обміну, обробки та збереження інформації в існуючих системах. Розроблено сервер обміну інформацією системи вібродіагностики по універсальному протоколу обміну даними.

Ключові слова: вібродіагностика, турбоустановка, обмін даними, прогнозування несправностей.

Постановка проблеми

В даний час у виробництві електроенергії широко використовуються турбоагрегати (сукупність турбіни та електрогенератора), що потребують періодичного планового ремонту з метою перевірки стану турбіни та її компонентів, заміни зношених частин, візуального огляду і т.д. для виключення аварійних ситуацій під час експлуатації. До планового ремонту обладнання висуваються жорсткі вимоги щодо часу витраченого на обслуговування компонентів через високу вартість простою обладнання. Погіршує ситуацію те, що кожен плановий ремонт обладнання займає великий проміжок часу через повний огляд системи, а

не цілеспрямований на існуючі чи спрогнозовані несправності. Сучасні установки, що обладнані системою віброаналізу здатні зрівнювати виміряне значення вібрації в даний момент часу з дозволеним діапазоном відхилень та в разі потреби повідомити оператора про небезпеку та необхідність зупинки роботи агрегату, але вони не зберігають великі масиви даних для дослідження поводження установки та попередження нестандартних подій. Мета даної статті висвітлити основний набір вібропараметрів, що необхідні для автоматизованої системи контролю вібродіагностики у різних режимах роботи турбоустановки та запропонувати метод передачі даних для прогнозування аварійних ситуацій. Переваги автоматизованої системи контролю вібродіагностики полягає у:

- підвищенні надійності, терміну експлуатації обладнання;
- зниженні витрат на ремонтні роботи (проведення періодичних ремонтних робіт за вимогою, а не планово);
- підвищенні ефективності ремонтних робіт (зниження часу на ремонт обладнання в наслідок розуміння проблеми виходячи з оброблених даних);
- завчасна закупівля деталей, що підлягають заміні.

Аналіз останніх досліджень і публікацій

Безпосереднім джерелом коливань є валопровід турбоагрегату - система з'єднаних між собою роторів. Валопровід, обертаючись на масляній плівці підшипників, передає через неї коливання на до підшипників і їх корпусів (опор). Вібрація турбоагрегату може відбуватися в трьох напрямках, тому її вимірюють на всіх підшипникових опорах в трьох взаємно-перпендикулярних напрямках по відношенню до осі валу турбоагрегату: вертикальному, осьовому, поперечному.

Вібрацію турбоагрегату вимірюють на всіх підшипникових опорах. Осьову і поперечну вібрацію вимірюють на рівні осі валу турбоагрегату. Вимірювальні датчики прикріплюються до основи підшипника. Вертикальну вібрацію вимірюють

на верхній частині кришки підшипника. На Рис.1 зображено принцип розташування вібродатчиків на опорах підшипників [1].

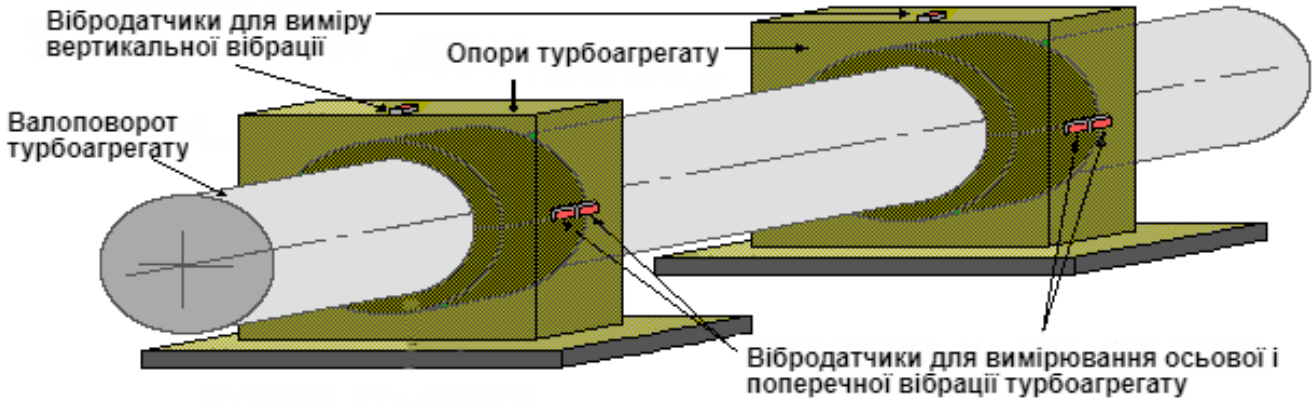


Рис.1 Принцип розташування вібродатчиків на опорах підшипників

Вібросигнал знятий на працюючій машині містить великий обсяг інформації по якому можна зробити висновки про її стан. Сутність діагностики і обслуговування турбоагрегату лежить в ранньому виявленні зароджуваних несправностей, але для ефективного використання віброконтролю в програмі технічного обслуговування необхідно, щоб ця інформація була належним чином вилучена з отриманих вібросигналів. Можливо найбільш відповідальним елементом системи збору вібраційних даних є датчик, що слугує для перетворення механічних коливань в електричний сигнал. Сигнал з датчика має вигляд аналогового часового сигналу, але внаслідок складності форми часового сигналу його інтерпретація ускладнена, тому прийнято аналізувати спектр сигналу, який є представленням часового сигналу у частотній області. В математиці перетворення, що переводить елемент з часової області в частотну називається перетворення Фур’є. Таке перетворення стискає всю інформацію, що міститься у синусоїдальному коливанні безкінечної довжини до єдиної точки Рис.2.

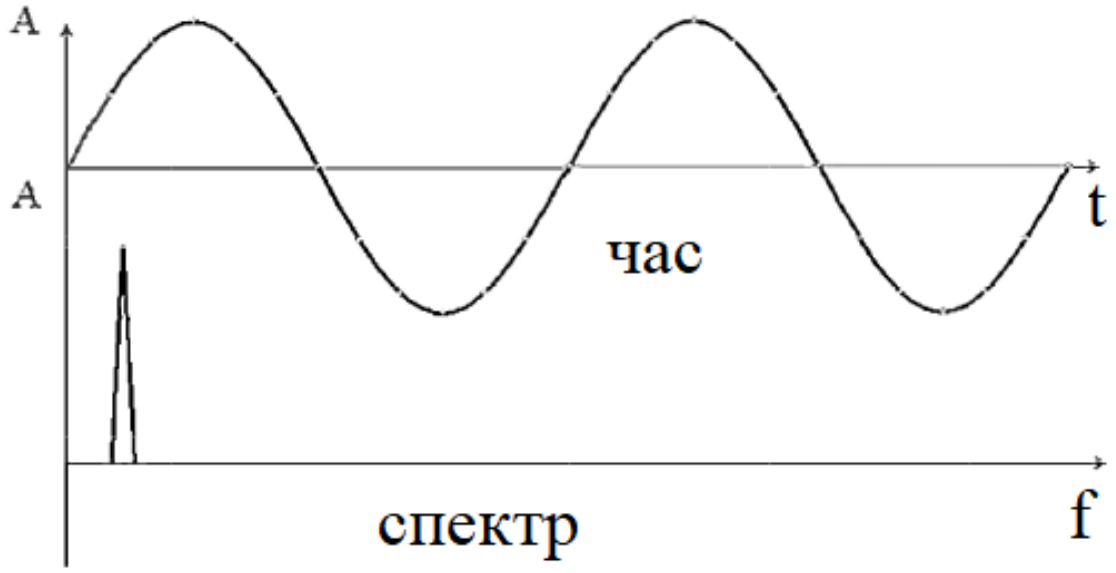


Рис.2 Часова реалізація і спектр гармонічного сигналу

При аналізі вібраційних характеристик часто зустрічаються часові реалізації, зрізані у верхній частині, як на Рис.3. Зазвичай це означає, що виникли деякі послаблення елементів або деталей і щось обмежує рух ослабленого елементу в одному з напрямків [2].

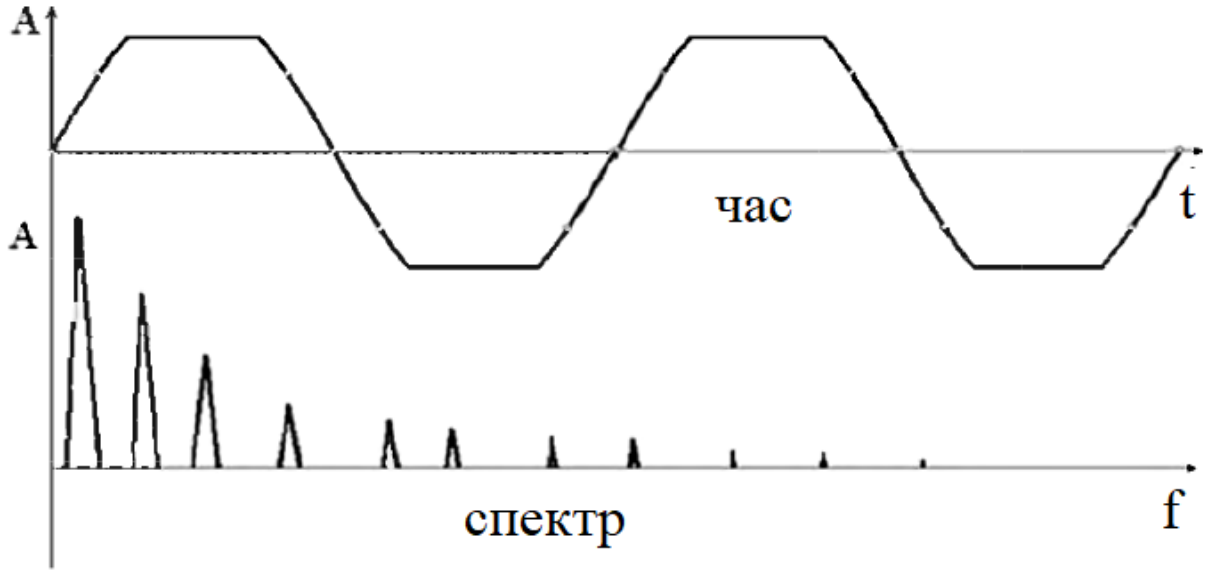


Рис.3 Часова реалізація і спектр сигналу синусоїди зі зрізаними вершинами

Для використанні в програмному продукті доцільніше використовувати ряд ефективних алгоритмів для визначення дискретно - часового ряду Фур’є, що має

назву швидке перетворення Фур'є (ШПФ). Цей ряд алгоритмів вирішує головну проблему , що виникає при практичній реалізації дискретно-часового перетворення Фур'є, що заключається у великій кількості логічних операцій [3].

Виділення невирішених раніше частин загальної проблеми

Аналіз попередніх публікацій показав, що наразі загального рішення по системі діагностики, прогнозування несправностей і автоматизованого обслуговування турбоагрегатів не висувається. У більшості робіт висвітлено конкретні обрахунки по одному з каналів або методи діагностики окремих елементів системи без опису систем збору та обробки інформації для можливості попередження аварійних ситуацій.

Мета статті

Головною метою цієї статті є опис системи вібродіагностики турбоагрегату з виділенням параметрів, що необхідно контролювати при різних режимах роботи. Стаття спрямована на створення основи для проектного рішення системи вібродіагностики турбоустановок різного призначення з можливістю зберігання та обробки отриманих даних для прогнозування виходу з ладу обладнання. Ідея полягає в тому, щоб описати основні необхідні та достатні параметри для контролю та діагностики більшості установок, та розглянути один із можливих методів обміну інформації для збереження та обробки отриманих даних. Дисертаційна робота, що виконується на основі цієї статті спрямована на представлення системи вібродіагностики для турбоустановки певного типу, на основі досліджень, що викладені в даній роботі.

Виклад основного матеріалу

Для збору та обробки інформації необхідно обрати прилад, який зможе з достатньо високою точністю та швидкістю опитувати датчики вібрації. Крім того такий прилад повинен мати можливість передавати інформацію по універсальному протоколу обміну даними для взаємодії з системами візуалізації та диспетчеризації,

чи напряму з хмарним сховищем в якому зберігатимуться всі виміряні величини за великий проміжок часу. Вирішення цієї проблеми відкриває можливість реалізації предиктивного інтелектуального аналізу за рахунок відслідковування зміни параметрів на значному проміжку часу, можливість в майбутньому обробити отримані дані програмними продуктами, що здатні з величезної кількості інформації виділити більш точні нові закономірності між зміною вібропараметрів та можливим дефектом в роботі турбоустановки. Предиктивний інтелектуальний аналіз полягає у прогнозуванні майбутнього поводження об’єкту з метою прийняття оптимальних рішень по усуненню недоліків в роботі до моменту їх настання.

Для поставлених задач гарним рішенням буде використання промислового комп’ютера на базі багатофункціональної плати АЦП/ЦАП з сигнальним процесором L-780M. Він має 14-ти розрядний АЦП з максимальною частотою перетворення 400 кГц. Завдяки наявності мережевої карти мною програмно реалізовано сервер Modbus для передачі даних по протоколу Modbus TCP у хмарне сховище Рис.4.

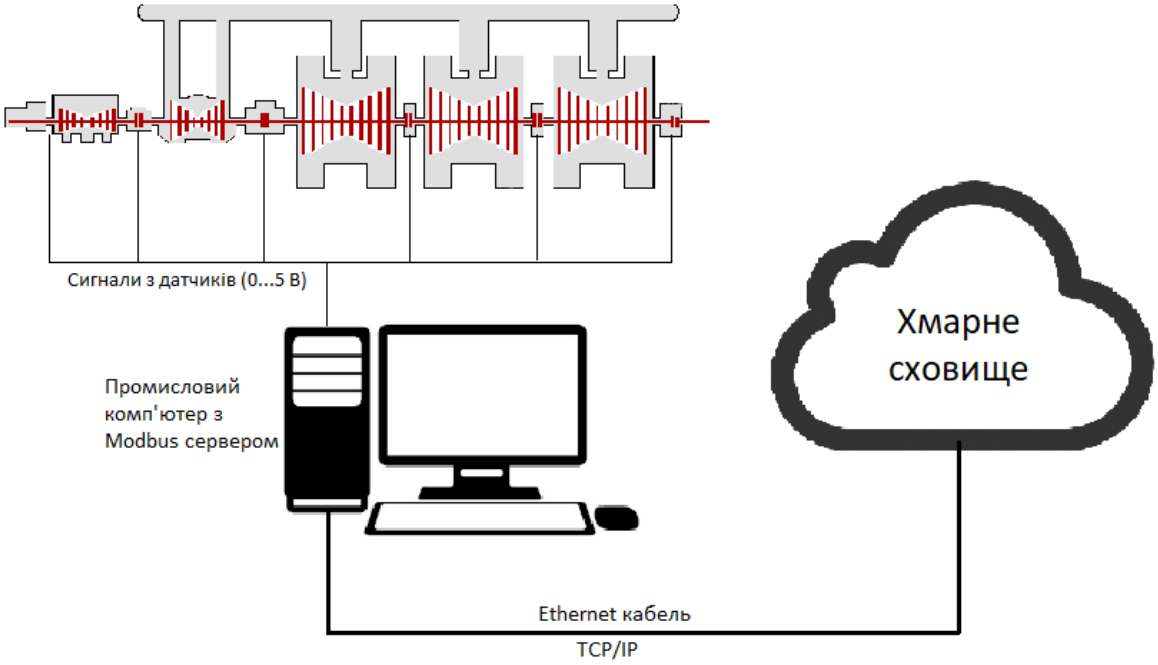


Рис.4 Структурна схема передачі даних у хмарне сховище

При дослідженні турбоагрегатів різного виду було виділено 4 експлуатаційні режими роботи при яких необхідно контролювати вібропараметри, а саме:

- Режим навантаження;
- Режим валоповороту;
- Режим набору обертів;
- Режим зупинки.

Режим навантаження – це номінальний режим роботи турбоустановки.

Режим валоповороту – режим пуску турбоустановки коли вона обертається на мінімальних обертах за рахунок валоповоротного пристрою.

Режим набору обертів – поступове збільшення частоти обертів при запуску турбоустановки після відключення валоповоротного пристрою до виходу на номінальний режим роботи.

Режим зупинки – поступове зменшення частоти обертів до зупинки турбоустановки.

Для контролю стану турбоагрегату у цих режимах спочатку необхідно розглянути які параметри потрібно обрахувати методом швидкого перетворення Фур'є з показів трьох вібродатчиків кожної опори (вертикального, горизонтального та осьового), а також двох датчиків валу (вертикальний та горизонтальний). Основні параметри, що необхідно розрахувати з датчиків опори:

- Ефективна швидкість вібрації;
- Гармонійні складові амплітуди віброшвидкості;
- Фази гармонійних складових амплітуди віброшвидкості;
- Середнє квадратичне значення низькочастотної вібрації;
- Постійна складова віброшвидкості;
- Середнє квадратичне значення високочастотної вібрації;
- Розмах сумарного вібропереміщення.

Ці всі параметри необхідно обраховувати по кожному з трьох вібродатчків встановлених на опорі підшипника.

З двох датчиків, що розташовані на валу необхідно обрахувати такі параметри:

- Розмах віброперемiщення;
- Гармонійні складові амплітуди віброперемiщення;
- Фази гармонійних складових амплітуди віброперемiщення;
- Розмах низькочастотної вібрації;
- Координата спливання Zx;
- Координата спливання Zy;
- Статистичний проміжок між датчиком і валом;
- Мінімальне значення динамічного проміжку;
- Максимальне значення модуля віброперемiщення.

Для різних режимів роботи турбоустановки можна виділити основні дефекти, що які можна діагностувати або попередити зробивши висновки з обрахованих параметрів показів датчиків вібродіагностики. Для більшості дефектів задаються граничні значення при перевищенні яких необхідно видавати попередження про небезпеку, але необхідно мати повну картину зміни виміряних вібрацій для подальшого аналізу. Також можна виділити небезпеки, які повинні відслідковуватись з плином часу. Так, наприклад, для режиму навантаження це:

- Зростання вібрації з певною частотою;
- Прискорення зростання вібрації з певною частотою.

Перша з них виникає по умові монотонного зростання першої гармонійної складової амплітуди віброшвидкості на задану величину за певний проміжок часу. Інша по умові монотонного зростання другої гармонійної складової амплітуди віброшвидкості на задану величину за певний проміжок часу. Крім цього на всіх режимах роботи є важливим слідкувати за показами низькочастотної вібрації, котра

зв’язана з якістю центрування машини, балансуванням ротора та технічним станом з’єднувальних муфт.

Для режиму набору обертів е важливим слідкувати за зміною величини першої гармонійної амплітуди віброшвидкості та вібропереміщення, зміна яких характеризує перегин валу.

Для режиму валоповороту необхідно звернути увагу на вимір ефективної швидкості вібрації та розмаху вібропереміщення для виявлення несправності каналу виміру вібрації валу.

Для режиму зупинки потрібно слідкувати за сьомою складовою амплітуди вібропереміщення щоб мати можливість діагностувати просадку валу в підшипнику.

Висновки і пропозиції

Застосування запропонованої системи діагностики з можливістю передачі даних по відкритому уніфікованому протоколу Modbus TCP вирішує проблеми доступу допоміжних систем для прогнозування можливих несправностей та попередження аварійних ситуацій системи. Запровадження предиктивного аналізу усуває необхідність проведення планового періодичного ремонту, що в свою чергу зменшує час на обслуговування обладнання та звужує фронт робіт при позаплановому обслуговуванні.

Список літератури

1. 10.4 Вібрація турбоагрегату //Сайт присвячений атомним станціям URL: http://www.aes.pp.ua/46.php (дата звернення 22.11.2019).
2. Петрухін В.В., Петрухін С.В. Основи вібродіагностики та засоби вимірювання вібрації. Москва, 2010. 30 с.
3. Медведєв С.Ю. Перетворення фур’є і класичний цифровий спектральний аналіз. URL: http://www.vibration.ru/preobraz_fur.shtml (дата звернення 24.11.2019).