ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:
«Інтелектуальна система керування кліматом у офісних приміщеннях на основі IoT»

на здобуття освітнього ступеня бакалавр
за спеціальності 126 Інформаційні системи та технології
(код,найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Михайло ГОРБАНЬ
(підпис) *(ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти
група ІСЗ-51 Михайло ГОРБАНЬ
(ім'я, ПРІЗВИЩЕ)

Керівник Ольга ЖИДКА
(ім'я, ПРІЗВИЩЕ)

Рецензент: _____
науковий ступінь, вчене звання *(ім'я, ПРІЗВИЩЕ)*

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних систем та технологій
Ступінь вищої освіти бакалавр
Спеціальність 126 Інформаційні системи та технології
Освітньо-професійна прорама Інформаційні системи та технології

ЗАТВЕРДЖУЮ

Завідувач кафедрою ІСТ
_____ Каміла СТОРЧАК

“____” _____2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Горбаню Михайлу Михайловичу
(прізвище, ім’я, по батькові здобувача)

1.Тема кваліфікаційної роботи: «Інтелектуальна система керування кліматом у офісних приміщеннях на основі IoT»
керівник кваліфікаційної роботи: Ольга ЖИДКА
(ім’я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від “24” лютого 2025 р. № 56

2. Строк подання кваліфікаційної роботи «30» травня 2025 р.

3. Вихідні дані кваліфікаційної роботи:
1. Принципи побудови та функціонування IoT
2. Принципи роботи систем керування кліматом.
3. Науково-технічна література.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
1.Огляд існуючих рішень та обґрунтування актуальності.
2. Проектування інтелектуальної системи клімат- контролю.
3. Схеми підключення приладів. Типові сценарії роботи системи.

5.Перелік ілюстраційного матеріалу: *презентація*
6. Дата видачі завдання «24» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Підбір технічної літератури	24.02-03.03.25	
2.	Аналіз існуючих рішень	04.03-17.03.25	
3.	Дослідження існуючих систем клімат-контролю	18.03-31.03.25	
4.	Проектування власної системи клімат-контролю	01.04-24.04.25	
5.	Висновки по роботі	25.04-16.05.25	
6.	Розробка демонстраційних матеріалів, доповіді.	19.05-20.05.25	
7.	Оформлення бакалаврської роботи.	21.05-30.05.25	

Здобувач вищої освіти _____ Михайло ГОРБАНЬ
(підпис) *(ім'я, ПРІЗВИЩЕ)*
Керівник кваліфікаційної роботи _____ Ольга ЖИДКА
(підпис) *(ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 64 стор., 15 рис., 2 табл., 15 джерел.

Мета роботи – Проєктування інтелектуальної системи керування кліматом для офісних приміщень з використанням мікроконтролера Raspberry Pi, нечкої логіки, та Telegram-інтерфейсу.

Об'єкт дослідження – Автоматизована система керування кліматом

Предмет дослідження – Програмно-апаратна система для адаптивного керування температурою та вологістю на основі сенсорних даних, за допомогою нечіткої логіки

Короткий зміст роботи: У першому розділі здійснено аналіз існуючих міжнародних рішень у сфері клімат-контролю (Tado, Netatmo, Ecobee, Xiaomi Smart Home тощо). Розглянуто їх функціональні можливості, принципи керування, відкритість, вартість, можливості інтеграції. Обґрунтовано доцільність створення власної системи на базі відкритих компонентів. У другому розділі описано архітектуру запропонованої системи: Raspberry Pi як центральний контролер, сенсори температури й вологості (DHT22), датчик присутності (PIR), модуль нечіткої логіки, Telegram-бот для керування й моніторингу, а також система логування. Наведено схему взаємодії між модулями та пояснення логіки кожного. У третьому розділі розглянуто підключення та налаштування сенсорів, приклади коду на Python, структуру програми та приклади обробки даних. Продемонстровано, як система приймає рішення про керування кліматичним обладнанням у різних сценаріях – з урахуванням присутності людини, поточних параметрів середовища та встановлених правил нечіткої логіки

КЛЮЧОВІ СЛОВА: Raspberry Pi, DHT22, PIR-сенсор, клімат-контроль, нечітка логіка, автоматизація, IoT, Python, fuzzy logic.

ЗМІСТ

ВСТУП

У сучасному світі, де інтелектуальні комп’ютерні системи стрімко розвиваються, зростає потреба в ефективному й розумному керуванні мікрокліматом у приміщеннях — офісах, виробничих просторах і навіть у житловому секторі. Проте не всі доступні на ринку рішення задовольняють потреби користувачів: одні є надто дорогими, інші — надто обмежені в налаштуванні під конкретні умови. Це зумовлює актуальність створення гнучкої та доступної системи, яка б забезпечувала комфортний мікроклімат з урахуванням реальних потреб і змін середовища.

Такі платформи як Raspberry Pi дають змогу поєднувати різні компоненти та створювати бюджетні аналоги автоматизованих систем керування кліматом тощо. Поєднання простих датчиків температури, вологості, присутності людини з нечіткою логікою та Telegram інтерфейсом дає можливість розробити просту для модифікування і розширення систему.

Історія клімат-контролю починається ще на початку XX століття — з простих механічних та електромеханічних термостатів. Тодішні системи були локальними, мали обмежений функціонал і не дозволяли налаштовувати параметри під умови середовища чи змінну кількість людей.

З 1960–1970-х років, разом із розвитком електроніки, з’явилися перші автоматизовані HVAC-системи — для опалення, вентиляції й кондиціювання повітря. Вони впроваджувалися переважно у великих будівлях: офісах, торговельних центрах, лікарнях.

Наступним етапом розвитку стала поява мікропроцесорної техніки. Це дало змогу створювати складніші алгоритми керування, реалізовувати зональне регулювання температури, враховувати зовнішні кліматичні зміни та присутність людей. Системи почали інтегруватися з іншими інженерними мережами будівель, відкриваючи шлях до централізованого або віддаленого керування середовищем.

Адаптивність, автоматичне навчання на основі даних, енергоефективність і комфорт стали більш доступними для користувачів. Сучасні рішення

використовують сенсори, штучний інтелект і аналітику для глибшого розуміння потреб приміщення. Особливу увагу приділяють інтеграції таких систем у загальну архітектуру розумних будівель, де всі модулі працюють узгоджено та забезпечують цілісне керування.

Створення нових систем керування кліматом – це логічний результат розвитку комп’ютерних систем, впровадження нових технологій та розвитку штучного інтелекту, що дає змогу покращувати комфорт та безпеку користувачів.

1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ОБГРУНТУВАННЯ АКТУАЛЬНОСТІ

1.1. Огляд існуючих міжнародних рішень

На глобальному ринку представлено велику кількість смарт-систем для клімат-контролю, що забезпечують автоматичне підтримання комфортної температури та інших параметрів мікроклімату в приміщеннях. Серед найпопулярніших рішень вирізняються розумні термостати та інтегровані екосистеми "розумного дому", розроблені такими компаніями, як Tado, Netatmo, Google (Nest), Ecobee, Xiaomi Smart Home та іншими.

Кожна з цих систем має власний набір функцій і підходів до керування. Частина з них орієнтована на максимальну зручність для користувача — наприклад, завдяки мобільним застосункам або голосовим асистентам. Інші роблять ставку на енергоощадність, гнучкість налаштувань або розширені можливості інтеграції з іншими пристроями.

Розглянемо ключові характеристики популярних рішень, зокрема:

- доступні функції управління;
- типи інтерфейсів та способи керування (мобільний застосунок, голосові помічники тощо);
- гнучкість у налаштуваннях під індивідуальні потреби;
- вартість системи;
- можливість модифікації або розширення її функціоналу.

1.1.1. Tado Smart Thermostat

Рис. 1.1 Tado Smart Thermostat

Tado (Німеччина) – інтелектуальна термостатична система для керування опаленням і кондиціонуванням, орієнтована переважно на європейські системи. Вона дозволяє дистанційно керувати кліматом у приміщенні через мобільний застосунок і підтримує геолокаційне керування (Geofencing) – автоматичне зниження або підвищення температури в залежності від місцезнаходження користувача, визначеного за геопозицією смартфона. Ключові функції Tado° включають створення розумних розкладів, виявлення відкритого вікна (фіксується різке падіння температури з подальшим тимчасовим вимкненням опалення), а також інші енергозберігаючі алгоритми.

Система підтримує багатозональне керування: встановивши додаткові смарт-термостати або термоголовки на радіаторах, користувач може окремо налаштовувати температуру в кожній кімнаті. Tado° інтегрується в сучасні екосистеми розумного дому – підтримує голосових помічників Apple HomeKit, Google Assistant, Amazon Alexa, а також сумісний зі стандартом Matter, який забезпечує уніфіковану взаємодію між пристроями.

Типовий спосіб керування – через хмарний сервіс та мобільний застосунок Tado. Доступне також ручне регулювання температури безпосередньо на термостаті за допомогою вбудованого індикатора та керуючого колеса. Гнучкість налаштувань досить висока в межах функціоналу, наданого виробником: користувач може створювати власні розклади, задавати пороги для геолокації, групувати пристрої по кімнатах тощо. Проте внутрішні алгоритми є закритими – вони визначені виробником і не підлягають зміні. Для зовнішньої інтеграції Tado° надає хмарний API та підтримує IFTTT, однак не передбачає можливості змінювати прошивку або логіку роботи пристрою самостійно, оскільки платформа є закритою.

Вартість системи належить до високого сегмента: базовий комплект (термостат + інтернет-бридж) коштує близько €200–250. Крім того, для доступу до деяких розширених функцій (наприклад, автоматичного керування без участі користувача) необхідна підписка на сервіс Auto-Assist, що передбачає регулярні витрати.

Хоч Tado і пропонує широкий набір функцій (геофенсинг, виявлення провітрювання, багатозональність), зручну інтеграцію з розумними екосистемами, але залишається дорогим рішенням із обмеженою можливістю кастомізації.

1.1.2. Netatmo Smart Thermostat

Рис. 1.2 Netatmo Smart Thermostat

Netatmo (Франція) - пропонує розумний термостат для індивідуальних котлів опалення, а також сумісні смарт-термоголовки для радіаторів. Функціонально Netatmo забезпечує дистанційне керування опаленням через мобільний додаток або веб-інтерфейс: користувач може змінювати температуру в реальному часі зі смартфона, планшета чи комп'ютера, а також вручну — безпосередньо на пристрої, оснащеному E-Ink дисплеєм.

Під час першого налаштування система пропонує відповісти на кілька базових запитань щодо розпорядку дня, після чого автоматично формує графік опалення — таким чином, приміщення обігрівається лише тоді, коли це справді потрібно. Передбачено також спеціальні режими, зокрема «Від'їзд» і «Антизамерзання», які зменшують витрати під час тривалої відсутності мешканців.

Гнучкість та інтеграція: Netatmo підтримує відкриту інтеграцію з іншими системами — сумісний із Apple HomeKit, Google Assistant, Amazon Alexa, а також має відкритий API для розробників. Це означає, що користувач може підключити термостат до сторонніх рішень типу Home Assistant, отримувати дані та будувати власні сценарії керування. За потреби система легко масштабується: додавання смарт-термоголовок на радіатори дозволяє реалізувати багатозональне керування — кожна кімната може мати свій температурний режим.

Тип керування — через власний хмарний сервіс: термостат з'єднується з інтернетом за допомогою спеціального реле-модуля (входить до комплекту) і синхронізується з додатком. Якщо інтернет-з'єднання недоступне, передбачено ручне керування безпосередньо на пристрої.

Вартість Netatmo дещо нижча порівняно з Tado: базовий комплект термостата оцінюється приблизно у £150 (180–200 €), а додаткові клапани для радіаторів — близько 70–80 € за одиницю. При цьому система має закриту прошивку, тобто користувач не може змінити вбудовані алгоритми, проте завдяки відкритому API можливе зовнішнє налаштування — написання власних сценаріїв, зв'язок із іншими платформами тощо.

1.1.3. Google Nest Learning Thermostat

Рис. 1.3 Google Nest Learning Thermostat

Nest Learning Thermostat (США)– один із перших масових розумних термостатів, розроблений компанією Nest Labs (США), нині випускається під брендом Google Nest. Його ключова особливість — наявність алгоритмів машинного навчання: термостат автоматично вивчає розклад і вподобання користувача. Протягом перших тижнів використання людина змінює температуру вручну або через додаток, а пристрій запам’ятовує ці дії та самостійно формує оптимальний графік нагріву й охолодження. У підсумку Nest може підтримувати комфортні умови без активної участі користувача.

Окрім цього, вбудовані датчики руху й присутності дозволяють визначити, чи перебуває хтось у приміщенні. Якщо ні — система автоматично переходить в енергозберігаючий режим. Паралельно використовується геолокація смартфонів (функція Home/Away Assist): коли всі мешканці залишають дім, температура знижується, а перед поверненням — знову підіймається до комфортного рівня.

Термостат також надає щомісячні енергозвіти, враховує погодні умови (наприклад, зменшує обігрів у сонячну погоду), та здатен сповіщати про неполадки в роботі системи опалення або кондиціонування.

Тип керування - Nest встановлюється замість звичайного настінного термостата й напряму підключається до системи HVAC. Керування можливе як вручну — за допомогою обертання й натискання керуючого кільця на дисплеї, так і дистанційно — через мобільний застосунок Google Home (раніше Nest App).

Nest інтегрується до екосистеми Google: підтримує керування голосом через Google Assistant, а також сумісний з Amazon Alexa (через відповідні навички). Водночас після інтеграції Nest у Google офіційна підтримка сторонніх платформ була зменшена — програма «Works with Nest» припинила роботу, а її замінив Google API з обмеженими можливостями, що ускладнило підключення до систем не з екосистеми Google.

Гнучкість налаштувань із точки зору користувача — висока: система адаптується до стилю життя, дозволяючи редагувати графіки та температурні межі вручну. Проте можливість змінити внутрішню логіку відсутня — Nest є повністю закритим у сенсі як апаратної, так і програмної архітектури.

Вартість Nest — одна з найвищих на ринку: актуальна третя (а також четверта) генерація Learning Thermostat коштує близько $249. Існує також спрощена версія — Nest Thermostat (близько $130), яка не має функції самонавчання, виготовляється з пластику та позбавлена підтримки деяких функцій, зокрема графіків. Флагманська модель підтримує підключення додаткових температурних сенсорів для різних кімнат, але ці датчики лише вимірюють температуру й не мають інфрачервоних сенсорів присутності — їх основна функція полягає в балансуванні мікроклімату по будинку.

Загалом, Nest можна охарактеризувати як преміальне "plug-and-play" рішення: воно самонавчальне, ефективне та зручне для кінцевого користувача, але водночас — жорстко прив’язане до екосистеми Google і не підтримує кастомізацію чи розширення поза офіційними сценаріями.

1.1.4. Ecobee Smart Thermostat

Рис. 1.4 Ecobee Smart Thermostat

Ecobee (Канада) – розумні термостати для систем опалення, вентиляції та кондиціювання повітря (HVAC), яка є одним із головних конкурентів Nest у Північній Америці. Існують моделі наступних версій - Ecobee3 Lite(базова), Ecobee SmartThermostat Enhanced (середня) та Ecobee SmartThermostat Premium (флагманська версія).

Усі моделі Ecobee підтримують дистанційне керування — через мобільний застосунок або веб-інтерфейс — а також інтегруються з голосовими асистентами (Google Assistant, Amazon Alexa, Apple Siri). Як і Nest, Ecobee дозволяє налаштувати графік роботи та активувати енергозберігаючий режим «Away» у разі відсутності мешканців.

Ecobee робить акцент на мультисенсорному підході до контролю мікроклімату. Усі моделі працюють з бездротовими кімнатними сенсорами (SmartSensors), фіксуючи температуру та наявність людей у приміщеннях. Це дозволяє, наприклад, підтримувати комфортну температуру тільки за умови людської присутності і не витрачати енергію на порожні приміщення.

Ще одна унікальна особливість — наявність вбудованого модуля Alexa (мікрофон і динамік), що перетворює Ecobee Premium на повноцінну смарт-колонку. Підтримка Siri реалізується через Apple HomePod.

Типи керування - Ecobee встановлюється замість звичайного термостата та безпосередньо керує системою HVAC. Керування здійснюється через сенсорний екран (для Enhanced і Premium), а також через мобільний застосунок. Ecobee добре інтегрується з екосистемами розумного дому: Apple HomeKit, Google Home, Amazon Alexa, Samsung SmartThings, IFTTT тощо. Крім того, надається відкритий API, що дозволяє підключати пристрій до власних систем автоматизації.

Гнучкість налаштувань у Ecobee висока. Користувач може:

– налаштовувати детальні розклади;
– використовувати геолокацію для активації режимів (геофенсинг);
– підключати до 32 додаткових датчиків;
– створювати складні сценарії через HomeKit або SmartThings.

Хоча функціонально Ecobee дає багато свободи, як і більшість комерційних продуктів, внутрішня логіка системи залишається закритою — змінювати прошивку або алгоритми роботи неможливо.

Вартість - цінова політика подібна до Nest. Модель Premium коштує приблизно $250 (в комплекті з одним SmartSensor), а Ecobee3 Lite — близько $150. Додаткові сенсори продаються окремо (~$40 за одиницю). Важливо, що основний функціонал не вимагає абонплати, на відміну від Tado°. Виробник пропонує підписку на додатковий сервіс моніторингу безпеки, але вона не є обов’язковою.

Завдяки відкритому API та підтримці стандартів автоматизації, Ecobee легко інтегрується і власні проєкти— наприклад, через Home Assistant. Це дозволяє зчитувати дані з термостата та керувати ним зі сторонніх застосунків. Проте пристрій є закритим і сертифікованим, що не дозволяє залазити у прошивку.

Загалом, Ecobee вирізняється широкими можливостями зонального контролю та високим рівнем інтеграції, що робить його вдалим вибором для користувачів, які прагнуть мати гнучкий контроль над мікрокліматом і самостійно налаштовувати параметри своєї системи.

1.1.5. Екосистема Xiaomi Smart Home

Рис. 1.5 Екосистема Xiaomi Smart Home

Xiaomi Smart Home (Китай) - представляє альтернативний підхід до організації клімат-контролю. Компанія не пропонує єдиного універсального «термостата» для котлів, натомість розвиває екосистему Mi Home, де користувач самостійно комбінує сенсори та виконавчі пристрої для автоматизації кліматичних процесів.

1. Функції та компоненти:

До екосистеми Xiaomi входять:

– бездротові сенсори температури та вологості (наприклад, Mijia або Aqara);
– розумні розетки й реле (для керування обігрівачами, теплою підлогою тощо);
– інфрачервоні контролери (бластери), які можуть замінити пульти до кондиціонерів;
– зволожувачі, очищувачі повітря, вентилятори та кондиціонери з Wi-Fi.

Усі ці пристрої можуть взаємодіяти через додаток Mi Home, де створюються автоматичні сценарії (“Scenes”). Наприклад:

– якщо температура в кімнаті < 18 °C — увімкнути обігрівач;

– якщо вологість < 40% — активувати зволожувач.

Датчик передає показник на шлюз (хаб), який дає команду розетці, що живить обігрівач. При досягненні температури вище встановленого значення — пристрій вимикається. Аналогічно можна автоматизувати кондиціонери через ІЧ-бластер, який "запам’ятовує" команди пульта й виконує їх за умовами сцени (наприклад, >26 °C — увімкнути охолодження).

Центральний інтерфейс — додаток Mi Home, де встановлюються сценарії, моніторяться пристрої та здійснюється ручне керування.

Більшість сценаріїв виконуються через хмару Xiaomi, але новіші пристрої з Bluetooth або ZigBee-хабом можуть працювати і локально (без інтернету).

Система підтримує інтеграцію з Google Assistant і Amazon Alexa — можна, наприклад, голосовою командою увімкнути розетку. Гнучкість Xiaomi — висока на апаратному рівні: користувач може під’єднати десятки сенсорів і пристроїв, не обмежуючись лише температурою (також підтримується автоматизація освітлення, безпеки тощо).

Однак логіка сценаріїв обмежена: додаток дозволяє лише прості правила типу “якщо-то”. Підтримки багаторівневих умов або навчальних алгоритмів немає — інтерфейс орієнтований на масового користувача, всі дії мають бути чітко задані.

2. Вартість:

Xiaomi пропонує недороге рішення:

– сенсори — $10–20;

– розетки — ~$15;

– ІЧ-контролери — ~$20–30;

– шлюз (хаб) — ~$30 (необхідний для деяких пристроїв).

Базовий набір для простої автоматизації може обійтись у $50–60, без жодних додаткових підписок: користування хмарою безкоштовне. Офіційно екосистема закрита, проте ентузіасти знаходять шляхи інтеграції. Існують неофіційні методи підключення Xiaomi-пристроїв до Home Assistant, альтернативні прошивки для шлюзів, утиліти для доступу до даних через хмарний

API Xiaomi. Це відкриває шлях до ширшої автоматизації, але вимагає технічних знань.

У підсумку, Xiaomi Smart Home — гнучке й доступне апаратне рішення, яке дозволяє створити власну систему клімат-контролю з недорогих компонентів. Воно більше підходить для ентузіастів або DIY-проєктів, де користувач самостійно визначає структуру, логіку й взаємодію між пристроями.

Порівняльна характеристика розглянутих рішень. наведена нижче в табл. 1.1, у ній узагальненотакі фактори як: основні функції, способи керування, гнучкість, орієнтовна вартість та можливості модифікації запропонованих рішень.

Таблиця 1.1

Порівняння розглянутих рішень.

Система	Гнучкість	Telegram	Ціна	Модифікація	Підписка
Nest	Низька	Ні	Висока	Немає	Так
Tado	Середня	Ні	Висока	Обмежено	Так
Xiaomi	Середня	Частково	Середня	Низька	Так
OpenHAB	Висока	Так	Середня	Так	Ні
Наша система	Висока	Так	Низька	Так	Ні

1.2 Обґрунтування актуальності розробки власної адаптивної системи клімат-контролю

Огляд сучасних комерційних рішень показує, що смарт-системи клімат-контролю справді здатні підвищувати комфорт і зменшувати енергоспоживання. Однак є кілька причин, чому розробка власної адаптивної системи залишається актуальною та виправданою:

1. Вартість та економічна доцільність. Більшість популярних рішень належать до преміум-сегменту — $150–250 лише за термостат, не враховуючи додаткових сенсорів. У контексті українських реалій така ціна часто є бар'єром. Крім того, деякі виробники вводять щомісячну підписку

за доступ до повного функціоналу (як-от у Tado°). Натомість система, зібрана з бюджетних компонентів (наприклад, Raspberry Pi і кілька сенсорів), може бути значно доступнішою — без плати за ліцензії й сервіс.

2. Закритість рішень та обмеженість модифікації. Комерційні пристрої зазвичай працюють "з коробки", але не дозволяють користувачу змінювати їхню логіку. У Nest неможливо додати власний алгоритм, а у Netatmo — налаштувати нестандартний режим. У випадку специфічних завдань або інтеграції з унікальними системами, готові рішення не дають потрібної гнучкості. Власна система дозволяє повністю керувати як апаратною, так і програмною складовою — змінювати конфігурацію, додавати сенсори, адаптувати логіку під конкретні умови.
3. Адаптивність та інтелектуальні алгоритми. Хоча деякі пристрої (наприклад, Nest) використовують машинне навчання, а Ecobee — датчики присутності, більшість систем працює за фіксованими розкладами або простими правилами. Натомість використання нечіткої логіки (fuzzy logic) дозволяє керувати мікрокліматом так, як це робить людина — з урахуванням не тільки точних значень, а й "розмитих" понять на кшталт "трохи прохолодно" чи "дуже душно". Такі алгоритми дають змогу досягти більш плавного і точного керування, знижуючи коливання температури та витрати енергії. Платформа на основі Raspberry Pi відкриває простір для реалізації й тестування таких підходів.
4. Інтеграція та масштабованість. Комерційні рішення часто замкнуті в межах своєї екосистеми. Щоб підключити додаткові сервіси — наприклад, месенджер або власну базу даних — доводиться використовувати обхідні шляхи. У саморобній системі з відкритими технологіями (API, MQTT, Webhooks) користувач може побудувати потрібну архітектуру, яка легко інтегрується в існуючу інфраструктуру. Це особливо важливо для дослідницьких проєктів та складних IoT-сценаріїв.
5. Освітній і науково-дослідний потенціал. Проєкт має високу навчальну цінність: він дозволяє поєднати знання з програмування, електроніки,

інтелектуальних алгоритмів, інтернету речей. У контексті дипломної роботи така система слугуватиме не лише прикладом практичного застосування теорії, а й експериментальною платформою для подальших досліджень і вдосконалення.

Адаптивна система клімат-контролю є оптимальним рішенням як з практичної так і з наукової точки зору. Вона поєднує зручність автоматизації з повною гнучкістю налаштування, що є важливою перевагою в умовах різноманітності потреб користувачів .

1.3 Обґрунтування актуальності дослідження

Метою цієї роботи є проєктування та теоретична реалізація адаптивної програмно-апаратної системи клімат-контролю для приміщення на основі недорогих компонентів і алгоритмів нечіткої логіки. Система має автоматично підтримувати комфортні умови, враховуючи температуру, вологість і присутність людей, а також надавати зручні інструменти для моніторингу та дистанційного керування.

Для досягнення цієї мети передбачено виконання таких завдань:

1. Розробка апаратної частини:
   Обрати одноплатний комп’ютер Raspberry Pi як контролер; підключити сенсор DHT22 (температура і вологість) та PIR-датчик (виявлення присутності). Забезпечити керування кліматичними пристроями через реле або ІЧ-модуль, враховуючи обраний сценарій використання. Подбати про стабільне живлення й надійну роботу компонентів.
2. Розробка алгоритмів нечіткої логіки:
   Представити параметри мікроклімату у вигляді нечітких множин (наприклад, “низька”, “висока” температура; “є люди” / “немає”). Створити базу правил (типу “якщо прохолодно і є люди – підвищити

нагрів”), обрати метод нечіткого висновку (наприклад, Mamdani) і реалізувати дефаззифікацію для отримання керуючого сигналу.

3. Реалізація програмного забезпечення:
   Написати програму для Raspberry Pi, яка зчитує дані з сенсорів, передає їх у модуль нечіткої логіки та керує обладнанням. Зробити роботу програми циклічною та автономною — як сервіс або демон, що запускається під час старту системи.
4. Керування системою через Телеграм-бота:
   - Розробити бот, що дозволяє:
   - переглядати поточні показники (температура, вологість, статус обладнання);
   - отримувати сповіщення (про порушення норм чи виявлення руху);
   - надсилати команди (зміна температури, режиму роботи).

   Взаємодія має реалізовуватись через Telegram API.
5. Додати можливість логування даних:
   Забезпечити збереження історії: параметри з сенсорів, рішення системи, команди від користувача.

2. ПРОЕКТУВАННЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ КЛІМАТ-КОНТРОЛЮ

2.1 Загальна архітектура системи

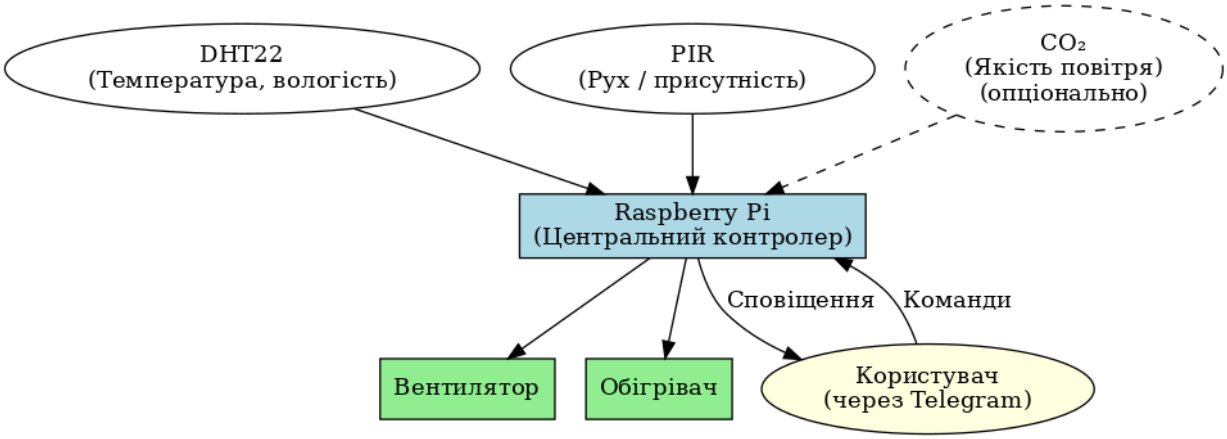


Рис. 2.1 Загальна архітектура системи (апаратна схема)

Інтелектуальна система керування кліматом побудована на основі мікрокомп'ютера Raspberry Pi, до якого підключаються датчики навколишнього середовища та виконавчі пристрої для регулювання мікроклімату. Raspberry Pi зчитує показники температури, вологості, наявності руху, обробляє їх за допомогою вбудованої логіки (зокрема, нечітких множин) та керує кліматичним обладнанням для підтримання комфортних умов середовища.

На рис. 2.1 зображено структурну схему системи. До Raspberry Pi підключені:

– сенсор DHT22 — для вимірювання температури та вологості повітря;
– PIR-датчик — для фіксації присутності людей у приміщенні;
– датчик CO_2(опціонально) — для моніторингу якості повітря.

Контролер не лише здійснює керування виконавчими механізмами (наприклад, вмикає обігрівач чи регулює швидкість вентилятора), але й може надсилати дані користувачу або приймати команди — зокрема, через Telegram-бота.

У рамках цієї системи відсутня необхідність використання платних хмарних серверів, усі обчислення проводяться локально, система працює в реальному часі, автоматично адаптуючи результати своєї роботи до поточних умов.

Центральним елементом логіки управління є нечіткий регулятор, який дозволяє уникнути грубих перемикань «увімкнено/вимкнено» на користь більш плавного та контекстного керування. Наприклад, замість негайного увімкнення вентиляції при досягненні порогової температури, система може зважити ступінь відхилення від норми, вологість, наявність людей — і лише тоді обрати режим роботи пристрою, наприклад, увімкнення вентилятора на середню швидкість.

Коли датчик руху фіксує відсутність людей, система автоматично переходить у енергозберігаючий режим — знижує температуру або зменшує активність виконавчих елементів. Якщо ж у систему інтегровано датчик CO_2, вона може додатково оцінювати якість повітря й, за потреби, активувати вентиляцію для нормалізації складу повітря.

Загалом, архітектура системи поділяється на три основні рівні:

1. Рівень збору даних — сенсори температури, вологості, руху, CO_2.
2. Рівень обробки та логіки — Raspberry Pi з алгоритмами нечіткого регулювання.
3. Рівень дії — виконавчі механізми (вентиляція, обігрів, кондиціювання).

Для зручності взаємодії з користувачем передбачено мережевий інтерфейс — Telegram-бот, який дозволяє отримувати дані та надсилати команди дистанційно.

Система може бути розширена, є можливість масштабування, додавання нових сенсорів, функцій тощо. Що робить її перспективною основою для подальшого впровадження розумної системи керування кліматом для освітнвх або побутових цілей.

2.2 Обгрунтування вибору обладнання

Центральним елементом системи виступає Raspberry Pi — універсальний мікрокомп'ютер на базі ARM-процесора, що поєднує компактність, енергоефективність та достатню обчислювальну потужність для реалізації інтелектуальних алгоритмів. Пристрій підтримує роботу повноцінної Linux-системи, запуск Python-сценаріїв у реальному часі та роботу з інтерфейсами GPIO, I²C, SPI, UART, USB, що дозволяє напряму підключати більшість сенсорів та приводів.

Однією з причин популярності Raspberry Pi в IoT є його вбудовані мережеві модулі (Wi-Fi, Ethernet), що робить його зручним для реалізації віддаленого моніторингу (наприклад, через Telegram-бота). У проєктах клімат-контролю він здатен обробляти навіть складні моделі — як-от нечітке керування — залишаючись при цьому тихим і малопотужним пристроєм, що працює у фоновому режимі.

DHT22 (AM2302) використовується для вимірювання температури та відносної вологості повітря. Його переваги — поєднання доступної вартості, цифрового інтерфейсу та прийнятної точності. Сенсор одразу видає оброблені цифрові значення, тож додатковий АЦП не потрібен. Він живиться від 3.3–5 В, споживає мізерну кількість струму (~2.5 мА) і може бути підключений безпосередньо до GPIO-порту Raspberry Pi. Типові характеристики:

- температура: від –40 до +80 °C, точність ±0.5 °C;
- вологість: 0–100%, точність ±2–5%;
- оновлення даних: кожні ~2 секунди.

Цього цілком достатньо для цілей контролю мікроклімату в реальному часі.

PIR-датчик (Passive Infrared) додає до системи можливість реагувати на присутність людей у приміщенні. Він виявляє зміну інфрачервоного випромінювання, яке випромінює людське тіло. Його використання дозволяє:

- зменшувати енергоспоживання, коли нікого немає (наприклад, вимикати обігрівачі або знижувати швидкість вентиляції);
- повертати комфортні умови, коли рух знову зафіксовано.

Інтеграція з Raspberry Pi проста — PIR видає цифровий сигнал, який зчитується через стандартний GPIO-вхід. Це рішення є дешевим, стабільним і ефективним для автоматичного перемикання режимів. Дослідження показують, що системи HVAC із сенсорами присутності дозволяють зменшити витрати енергії в середньому на 10% і більше.

Датчик CO_2 (опційно) підвищує рівень інтелектуальності системи, додаючи ще один параметр якості повітря. Підвищений рівень CO_2 може вказувати на нестачу вентиляції, що негативно впливає на самопочуття та когнітивну продуктивність. У таких випадках система може:

– автоматично активувати вентиляцію;
– збільшити швидкість повітрообміну до нормалізації показників.

Для цього використовуються датчики типу NDIR, наприклад MH-Z19B, які мають точність та діапазон 400–5000 ppm. Підключення до Raspberry Pi можливе через UART, I^2C або аналоговий сигнал із АЦП. При перевищенні порогу (наприклад, 1000 ppm) система приймає відповідні дії.

Для більш простих задач можуть застосовуватись сенсори типу MQ-135, однак вони забезпечують менш точне вимірювання. Вибір конкретної моделі залежить від балансу між бюджетом і необхідною точністю.

Підсумковий набір обладнання виглядає так:

– Raspberry Pi — головний контролер із вбудованою мережею, достатньо потужний для обробки даних і керування;
– DHT22 — основний сенсор мікроклімату (температура і вологість);
– PIR — сенсор присутності, що впливає на режим роботи;
– CO_2-сенсор (опціонально) — контроль якості повітря;
– Виконавчі пристрої — вентилятор та/або обігрівач, підключені через GPIO (через реле або драйвери).

Контролер Raspberry Pi має достатньо GPIO-портів для одночасного підключення сенсорів та виконавчих модулів. Це дозволяє гнучко змінювати логіку, масштабувати систему та адаптувати її під конкретні задачі.

2.3 Структура програмного забезпечення

Програмне забезпечення системи клімат-контролю побудовано за модульною структурою, що передбачає поділ функцій між окремими Python-файлами. Такий підхід спрощує розробку, налагодження та підтримку коду, оскільки кожен модуль відповідає за конкретну підсистему.

У проєкті виділено такі основні модулі:

1. main.py — центральний скрипт, що ініціалізує систему, налаштовує GPIO, завантажує конфігурацію, періодично опитує сенсори, запускає алгоритм нечіткого керування та надсилає команди на виконавчі пристрої. Також тут запускаються фонові сервіси — наприклад, Telegram-бот і журналювання.
2. sensors.py — модуль взаємодії з датчиками. Містить функції для зчитування даних із DHT22 (температура, вологість), PIR-сенсора (рух) і, за наявності, CO_2-сенсора. Модуль приховує низькорівневу реалізацію зчитування, забезпечуючи зручний інтерфейс для main.py.
3. fuzzy_logic.py — модуль нечіткої логіки. Тут описані нечіткі множини для температури, вологості, присутності, а також вихідні змінні (ступінь обігріву, швидкість вентилятора). Також модуль містить набірправил If–Then та функцій, які обчислюють рішення на основі актуальних даних.
4. control.py — модуль, що керує фізичними виконавчими механізмами. Забезпечує абстрактні команди для керування обладнанням: heater_on(), fan_off(), set_fan_speed(percent) тощо. При цьому реалізується доступ до GPIO-виходів Raspberry Pi або PWM-контролерів.
5. telegram_bot.py — відповідає за взаємодію з користувачем через Telegram. Дає змогу отримувати поточні показники, перемикати режими роботи, змінювати параметри клімату. Також бот надсилає сповіщення про важливі події — наприклад, перевищення температури або зміну стану.
6. logger.py — модуль логування. Записує в лог-файл всі ключові події: зчитування сенсорів, зміни режимів, надходження команд, запуска

зупинку обладнання. Це дає змогу відстежувати коректність роботи системи.

7. config.py — конфігураційний файл. Тут зберігаються налаштування: GPIO-піни, межі нечітких множин, параметри комфорту, токен Telegram-бота, періодичність логування тощо. Винесення налаштувань в окремий модуль дозволяє легко змінювати поведінку системи без втручання в основний код.

2.3.1. Переваги обраної архітектури

Уся логіка керування зосереджена в main.py. У циклі він:

– Зчитує дані із sensors.py.
– Передає їх до fuzzy_logic.py для аналізу.
– Отримує результат і надсилає команду у control.py.
– Паралельно у фоновому режимі працює telegram_bot.py, який очікує команди користувача. У разі зміни налаштувань, main.py реагує на них у наступному циклі.

Усі ключові події передаються до logger.py, який забезпечує фіксацію інформації.

Модульність коду дозволяє:

– легко оновлювати або замінювати окремі частини системи без ризику зламати інші;
– проводити експерименти — наприклад, оптимізувати правила у fuzzy_logic.py, не змінюючи інші модулі;
– швидко додавати нові сенсори, сценарії або канали взаємодії.

Це особливо зручно на етапі досліджень або розширення функціоналу — система зберігає гнучкість і залишається зрозумілою навіть зі зростанням складності.

2.4 Опис функціоналу та фрагменти коду

Далі розглянемо ключові модулі програмного забезпечення більш докладно, доповнюючи опис короткими прикладами коду. Усі модулі написані мовою Python із використанням стандартних та сторонніх бібліотек:

- logging — для журналювання;
- Adafruit_DHT — для роботи з сенсором DHT22;
- RPi.GPIO — для керування портами GPIO Raspberry Pi;
- skfuzzy — для реалізації нечіткої логіки;
- python-telegram-bot — для інтеграції з Telegram API**.**

2.4.1 Модуль main.py – центральна логіка системи

main.py — це головний модуль системи, що відповідає за запуск, ініціалізацію всіх компонентів і виконання основного циклу керування. Його завдання — зв’язати всі частини програми в єдиний процес і забезпечити стабільну роботу системи.

Після запуску (автоматично при завантаженні Raspberry Pi або вручну), main.py:

- імпортує допоміжні модулі (sensors.py, fuzzy_logic.py, control.py, logger.py, config.py);
- зчитує налаштування з конфігураційного файлу config.py;
- налаштовує апаратну частину (GPIO-піни, підключення до Telegram-бота, логування тощо).

Після цього запускається основний цикл, який виконується з заданим інтервалом (наприклад, раз на секунду або рідше — залежно від вимог до швидкості реагування). У межах цього циклу:

1. Зчитуються поточні дані з сенсорів (температура, вологість, рух).

2. Отримані значення передаються в модуль fuzzy_logic.py, де аналізуються за допомогою нечіткої логіки.
3. Результат (наприклад, "увімкнути обігрівач на 50%") надсилається в control.py, який керує виконавчим обладнанням.
4. Усі події можуть записуватись у журнал (logger.py).
5. У фоновому режимі обробляються запити та команди, отримані від Telegram-бота.

```python
# main.py (фрагмент)
import time
from sensors import read_environment, motion_detected
from fuzzy_logic import compute_action
from control import apply_action
from logger import log_status
from config import TARGET_TEMPERATURE

print("Starting Climate Control System...")
# Ініціалізація компонентів, якщо потрібно
# (напр. GPIO, з'єднання з ботом тощо)

while True:
    # 1. Зчитування датчиків
    temp, hum = read_environment()        # температура, вологість
    motion = motion_detected()            # чи є рух (True/False)
    # 2. Обчислення керуючих дій на основі нечіткої логіки
    action = compute_action(temp, hum, motion, TARGET_TEMPERATURE)
    # 3. Застосування дій до пристроїв
    apply_action(action)
    # 4. Логування стану
    log_status(temp, hum, action)
    # 5. Затримка до наступного циклу
    time.sleep(1)
```

Рис. 2.1 Приклад коду main.py

У цьому прикладі показано типовий ланцюг дій у системі: функція read_environment() отримує поточні значення температури та вологості з датчика DHT22, а motion_detected() визначає, чи зафіксовано рух за допомогою PIR-сенсора.

Далі отримані параметри передаються у функцію compute_action(...), реалізовану в модулі нечіткої логіки. Вона аналізує поточний стан середовища (а за потреби — й бажані параметри, наприклад, цільову температуру) та повертає керуюче рішення — як правило, у вигляді словника або набору команд, наприклад:

{'fan_speed': 70, 'heater': False}

Це рішення обробляється через apply_action(action) із модуля control.py, який встановлює відповідні режими роботи виконавчих пристроїв: у цьому випадку — вентилятор запускається на 70% потужності, нагрівач вимикається.

Паралельно функція log_status(...) записує ключові значення поточної ітерації, це дозволяє аналізувати роботу системи у часі.

Цикл виконується постійно, завдяки чому система здійснює безперервний моніторинг та регулювання мікроклімату.

2.4.2 Модуль sensors.py – зчитування показників датчиків

Модуль sensors.py відповідає за взаємодію з усіма сенсорами, підключеними до системи. Тут реалізовані функції та (за потреби) класи, що дозволяють отримувати дані з датчиків у зручному для основного коду форматі.

Основні завдання модуля:

– ініціалізація сенсорів під час запуску системи (якщо цього потребує бібліотека);
– надання простих функцій, які повертають актуальні значення параметрів мікроклімату.

Для DHT22 використовується стороння бібліотека, наприклад Adafruit_DHT, яка реалізує протокол обміну з сенсором. На її основі у sensors.py можна створити функцію read_environment(), що повертає відразу два значення — температуру (в °C) і вологість (у %).

У випадку PIR-датчика (наприклад, HC-SR501) достатньо налаштувати відповідний GPIO-пін як вхід, а потім просто зчитувати логічний рівень (HIGH або LOW). На основі цього визначається, чи зафіксовано рух у приміщенні.

Якщо в систему інтегровано датчик CO_2, у модулі створюється функція на зразок read_co2(), яка повертає поточний рівень в ppm. Спосіб зчитування залежить від типу сенсора — UART, I^2C або аналоговий вхід.

Завдяки sensors.py основний код (наприклад, main.py) не повинен "знати", як саме відбувається взаємодія з апаратурою — досить викликати одну або дві функції, щоб отримати всі потрібні значення.

```python
# sensors.py (фрагмент)
import Adafruit_DHT
import RPi.GPIO as GPIO

# Налаштування типу сенсора і GPIO-піну для DHT22
DHT_SENSOR = Adafruit_DHT.DHT22
DHT_PIN = 4        # номер піна GPIO для DHT22
# Налаштування PIR сенсора
PIR_PIN = 17
GPIO.setup(PIR_PIN, GPIO.IN)

def read_environment():
    #Зчитує температуру (°C) і вологість (%) з DHT22.#
    humidity, temperature = Adafruit_DHT.read_retry(DHT_SENSOR, DHT_PIN)
    return temperature, humidity

def motion_detected():
    #Повертає True, якщо датчик руху виявив присутність#
    state = GPIO.input(PIR_PIN)
    return True if state == GPIO.HIGH else False

```

Рис. 2.2 Приклад коду sensors.py

Функція read_environment() використовує метод Adafruit_DHT.read_retry() з бібліотеки Adafruit, який виконує кілька спроб зчитування даних із сенсора DHT22 для підвищення надійності. На виході повертаються два значення — вологість (humidity) і температура (temperature). У разі невдалої спроби зчитування обидва значення можуть бути None, тому в повній реалізації бажано передбачити обробку таких ситуацій (наприклад, повторну спробу або логування помилки).

Функція motion_detected() зчитує логічний рівень з GPIO-піна, до якого підключено PIR-датчик. Якщо сенсор фіксує рух, на виході з’являється сигнал HIGH, відповідно функція повертає True. Це дозволяє перевірити присутність руху в приміщенні.

Піни для підключення сенсорів (у прикладі — номери 4 і 17) задаються у файлі config.py, що робить систему гнучкішою: при зміні схеми підключення не потрібно вносити правки в код самого модуля.

Аналогічно можна реалізувати функцію read_co2() для зчитування рівня вуглекислого газу з відповідного сенсора. Конкретна реалізація залежатиме від моделі пристрою — деякі працюють через UART, інші — через I²Cабо навіть аналоговий інтерфейс із АЦП.

За потреби модуль sensors.py може виконувати базову обробку даних — наприклад, усереднювати кілька вимірювань для згладжування шумів або перевіряти значення на адекватність (відсікати явні викиди).

Однак у рамках цього проєкту передбачається, що датчики працюють стабільно, а їхні показники достатньо точні для подальшого аналізу без додаткової фільтрації.

2.4.3 Модуль fuzzy_logic.py – нечіткий контролер клімату

fuzzy_logic.py - це ключовий модуль, який надає системі властивість інтелектуальності. Саме тут реалізовано алгоритм нечіткого управління: система приймає рішення не на основі жорстких умов, а за допомогою гнучких лінгвістичних правил If–Then, які ближчі до людського способу мислення.

У модулі визначаються нечіткі змінні (fuzzy variables) як для вхідних, так і для вихідних параметрів. Наприклад:

- для температури — терми "холодно", "комфортно", "жарко";
- для вологості — "сухо", "нормально", "волого";
- для вентиляції — "низька", "середня", "висока" швидкість;
- додатково можуть враховуватись "є люди" / "немає людей".

Кожному терму відповідає функція належності — наприклад, трикутна, трапецієвидна або гаусова — що визначає ступінь відповідності поточного значення обраному опису.

Після цього задається база нечітких правил, наприклад:

- Якщо температура = "жарко" і вологість = "висока", тоді вентиляція = "максимальна"

– Якщо температура = "трохи жарко" і немає людей, тоді вентиляція = "середня", охолодження = "економ"

Такі правила дозволяють системі враховувати кілька факторів одночасно — адже це MIMO-система (Multi-Input, Multi-Output). Замість жорстких перемикань, вона здійснює плавне регулювання.

Основні етапи нечіткого виводу:

– Фазифікація — визначення ступенів належності вхідних значень до термів.

– Обчислення умов правил — оцінка активності кожного правила.

– Агрегація — об’єднання результатів кількох правил.

– Дефазифікація — перетворення результату у чітку дію (наприклад, встановити вентилятор на 72%).

У Python це можна реалізувати як вручну, так і за допомогою бібліотеки scikit-fuzzy (skfuzzy), яка спрощує роботу з нечіткими множинами, термами, правилами та обчисленням результату.

```python
# fuzzy_logic.py (фрагмент)
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

# Нечіткі змінні (Antecedents для входів, Consequents для виходів)
temp_diff = ctrl.Antecedent(np.arange(-5, 6, 1), 'temp_diff')   # відхилення темп. від бажаної
humidity = ctrl.Antecedent(np.arange(0, 101, 1), 'humidity')    # вологість у %
fan_speed = ctrl.Consequent(np.arange(0, 101, 1), 'fan_speed')  # швидкість вентилятора %

# Визначення функцій належності для температури (холодно, нормально, жарко)
temp_diff['cold'] = fuzz.trimf(temp_diff.universe, [-5, -5, 0])
temp_diff['ok']   = fuzz.trimf(temp_diff.universe, [-2, 0, 2])
temp_diff['hot']  = fuzz.trimf(temp_diff.universe, [0, 5, 5])
# Для вологості (сухо, норм, волого)
humidity['dry']   = fuzz.trimf(humidity.universe, [0, 0, 40])
humidity['med']   = fuzz.trimf(humidity.universe, [30, 50, 70])
humidity['humid'] = fuzz.trimf(humidity.universe, [60, 100, 100])
# Для швидкості вентилятора (низька, середня, висока)
fan_speed['low']  = fuzz.trimf(fan_speed.universe, [0, 0, 50])
fan_speed['high'] = fuzz.trimf(fan_speed.universe, [50, 100, 100])

# Приклад правила: Якщо жарко АБО волого, то вентилятор високий
rule1 = ctrl.Rule(temp_diff['hot'] | humidity['humid'], fan_speed['high'])
# Якщо холодно І сухо, то вентилятор низький
rule2 = ctrl.Rule(temp_diff['cold'] & humidity['dry'], fan_speed['low'])

# Створення контролера на основі правил
fan_ctrl = ctrl.ControlSystem([rule1, rule2, ...])
fan_simulation = ctrl.ControlSystemSimulation(fan_ctrl)

def compute_action(temp, hum, motion, target_temp):
    """Обчислює необхідну швидкість вентилятора (0–100%) та інші дії."""
    # Розрахунок відхилення температури
    diff = temp - target_temp
    # Нечіткий висновок для швидкості вентилятора
    fan_simulation.input['temp_diff'] = diff
    fan_simulation.input['humidity'] = hum
    fan_simulation.compute()  # виконує нечітке обчислення
    fan = fan_simulation.output['fan_speed']
    # Врахування PIR: якщо нікого немає, можна знизити швидкість
    if motion is False:
        fan = min(fan, 50.0)  # обмежити макс. 50% при відсутності людей
    return {'fan_speed': fan}
```

Рис. 2.3 Приклад коду fuzzy-logic.py

У цьому прикладі реалізовано базову схему нечіткого керування швидкістю вентилятора. Спочатку оголошуються нечіткі змінні:

– temp_diff — різниця між поточною температурою і цільовим значенням;
– humidity — вологість повітря;
– fan_speed — керована вихідна змінна.

Для кожної з них визначено по три терми з відповідними трикутними функціями належності (наприклад: "низька", "середня", "висока"). Значення для побудови цих функцій задаються вручну — на основі досвіду або експериментального підбору.

Далі формуються нечіткі правила керування. У прикладі створено два:

1. rule1: якщо жарко і волого — увімкнути вентиляцію на максимум;
2. rule2: якщо трохи жарко і нікого немає — вентилювати слабше.

Ці правила описуються за допомогою звичних логічних операторів AND (&) та OR (|). Система керування (ControlSystem) будується на основі набору таких правил, а ControlSystemSimulation відповідає за обчислення вихідних дій.

Основна функція модуля — compute_action(...) — приймає:

– поточну температуру,
– вологість,
– факт наявності людей (за даними PIR),
– та цільову температуру target_temp.

Спочатку вона рахує різницю температур (diff), потім задає значення вхідних змінних для симулятора, виконує розрахунок (fan_simulation.compute()), і отримує результат:

1. Нечітка змінна fan_speed автоматично дефазифікується
(використовується метод центру ваги за замовчуванням).
2. Окремо враховується наявність людей у приміщенні: якщо нікого нема, вентилятор обмежується на рівні 50% — це просте, але ефективне рішення для енергозбереження.
3. Функція повертає словник із діями, наприклад:
{'fan_speed': 60, 'heater': False}

Таким чином, fuzzy_logic.py перетворює вихідні сенсорні дані на розумні команди для керування змінами клімату у приміщенні. Це дає змогу уникнути різкого ввімкнення/вимкнення вентилятора, характерного для простих порогових схем, і забезпечити комфорт разом із енергоефективністю.

У складніших реалізаціях система може враховувати, наприклад, рівень CO_2. У такому випадку додається правило:
"Якщо CO_2 високий — посилити вентиляцію, незалежно від температури". Модуль підтримує такі розширення — достатньо додати відповідну змінну й пов’язане з нею правило.

2.4.4 Модуль control.py – керування виконавчими пристроями

Після того як система визначила, яку саме дію потрібно виконати (наприклад, на яку швидкість запустити вентилятор чи чи варто вмикати обігрівач), ці команди мають бути фізично реалізовані. Саме за це відповідає модуль control.py.

У цьому модулі організовано вивід керуючих сигналів на відповідні GPIO-піни мікрокомп’ютера Raspberry Pi. В залежності від типу виконавчих пристроїв, керування може бути:

– дискретним — через реле, які просто вмикають або вимикають прилад (наприклад, нагрівач);
– аналоговим (ШІМ) — коли потрібно регулювати щось плавно, наприклад, швидкість вентилятора.

У Raspberry Pi є апаратна підтримка ШІМ (PWM) на деяких GPIO, хоча для більшої гнучкості можна застосовувати й програмну реалізацію PWM.

При ініціалізації модуля необхідно:

– задати GPIO-піни як вихідні;
– створити відповідні PWM-об’єкти (наприклад, для вентилятора);
– встановити початковий стан пристроїв (щоб уникнути небажаного запуску одразу після включення системи).

```python
# control.py (фрагмент)
import RPi.GPIO as GPIO
# Припустимо, вентилятор підключено через транзистор до PIN 18 (ШІМ)
FAN_PIN = 18
GPIO.setup(FAN_PIN, GPIO.OUT)
fan_pwm = GPIO.PWM(FAN_PIN, 25)  # PWM з частотою 25 Hz
fan_pwm.start(0)  # запуск з 0% заповнення

def set_fan_speed(speed_percent):
    """Встановлює швидкість вентилятора, % (0–100)."""
    fan_pwm.ChangeDutyCycle(speed_percent)

def heater_on():
    """Увімкнути обігрівач (реле)."""
    GPIO.output(HEATER_PIN, GPIO.HIGH)

def heater_off():
    """Вимкнути обігрівач."""
    GPIO.output(HEATER_PIN, GPIO.LOW)

def apply_action(action):
    """Застосувати набір керуючих дій до пристроїв."""
    if 'fan_speed' in action:
        set_fan_speed(action['fan_speed'])
    if 'heater' in action:
        if action['heater']:
            heater_on()
        else:
            heater_off()

```

Рис. 2.4 Приклад коду control.py

Функція set_fan_speed() відповідає за регулювання швидкості вентилятора. Вона змінює скважність PWM-сигналу (тобто тривалість імпульсів) на відповідному GPIO-піні, що дозволяє плавно регулювати середню напругу, а отже — і оберти вентилятора. Цей метод особливо ефективний для невеликих постійно обертових двигунів, які підключені через транзисторний ключ.

Функції heater_on() і heater_off() керують обігрівачем через реле. Вони просто встановлюють логічний рівень на GPIO-піні HEATER_PIN, який має бути попередньо налаштований як вихід. Коли обігрівач не потрібен — пін вимикається, і живлення на пристрій не подається.

Функція apply_action(action) — універсальна точка входу для команди дій. Вона приймає словник, сформований у fuzzy_logic.compute_action(), і викликає відповідні функції керування. Наприклад:

apply_action({'fan_speed': 60, 'heater': False})

Цей підхід дає можливість відділити логіку управління пристроями від головної програми. Головний цикл передає результат контролера в apply_action, не замислюючись про те, які саме способи управління використовуються.

Модуль control.py легко розширюється у разі додавання нових пристроїв. Наприклад:

Для зволожувача можна додати функції humidifier_on() / humidifier_off().

Для кондиціонера — реалізувати надсилання ІЧ-команд через відповідний передавач.

У нашій системі наразі базовими виконавчими пристроями є:

– вентилятор (що регулюється через PWM),

– обігрівач (вмикається/вимикається через реле).

Raspberry Pi може керувати:

– малопотужними вентиляторами (5 В) напряму, через транзисторний ключ;

– або більш потужними пристроями — опосередковано, через реле або твердотільні реле (SSR).

2.4.5 Модуль Telegram_bot.py – віддалений інтерфейс телеграм

Модуль реалізації Telegram-бота, який дає користувачу змогу зручно взаємодіяти із системою. Через бот можна не тільки отримувати дані, але й керувати системою в реальному часі.

Бот підтримує кілька основних команд:

– /status — надсилає поточні показники: температуру, вологість, стан вентилятора та обігрівача;

– /set_temp 25 — дозволяє встановити бажану температуру (в даному випадку 25 °C);

– /mode auto або /mode manual — перемикає режим роботи між автоматичним та ручним.

Окрім цього, бот може відправляти повідомлення самостійно, наприклад:

– повідомити, якщо перевищено рівень CO_2;

– попередити про різке падіння температури;

– або надіслати плановий звіт про стан системи.

Для реалізації використовується популярна Python-бібліотека python-telegram-bot, яка значно спрощує роботу з Telegram Bot API. Зокрема, вона надає:

– клас Updater — для обробки нових повідомлень;

– клас Dispatcher — для маршрутизації команд.

У telegram_bot.py:

– Під час ініціалізації створюється об’єкт Updater, куди передається токен Telegram-бота (він зберігається в config.py);

– Реєструються функції-обробники (handler'и) для кожної команди;

– Запускається опитування серверів Telegram методом start_polling()

Такий підхід дозволяє користувачу взаємодіяти із системою в будь-який момент, не відкриваючи додаткових застосунків чи веб-інтерфейсів. Весь контроль — у Telegram.

За потреби можливо додати нові команди – наприклад, для зміни вологості або перегляду історії логів.

У цілому, наявність Telegram-бота суттєво підвищує зручність у користуванні системою. Користувач може будь-коли, з будь-якої точки світу, перевірити поточний стан мікроклімату, змінити налаштування або отримати сповіщення про нештатні ситуації.

Це особливо актуально, якщо Raspberry Pi працює автономно — наприклад, встановлений десь у будинку чи на віддаленому об’єкті. У такому випадку Telegram-бот замінює складні інтерфейси, дозволяючи керувати системою без необхідності створення окремого мобільного застосунку чи вебпанелі.

```python
# telegram_bot.py (фрагмент)
from telegram.ext import Updater, CommandHandler
from sensors import read_environment

TOKEN = "XXXXXXXXXXXXXXXXXXXXXXXXX"  # Токен бота (в реальн. коді з config.py)

updater = Updater(token=TOKEN, use_context=True)
dispatcher = updater.dispatcher

def cmd_start(update, context):
    """Обробник команди /start – вітальне повідомлення."""
    update.message.reply_text("Вітаю! Я – бот клімат-контролю. Надішліть /status для перегляду стану.")

def cmd_status(update, context):
    """Обробник команди /status – відповідає поточними показниками."""
    temp, hum = read_environment()
    update.message.reply_text(f"Температура: {temp:.1f}°C\nВологість: {hum:.1f}%")

dispatcher.add_handler(CommandHandler("start", cmd_start))
dispatcher.add_handler(CommandHandler("status", cmd_status))

# Запуск бота у фоновому режимі
updater.start_polling()
print("Telegram bot started...")
```

Рис. 2.5 Приклад коду для Telegram_bot.py

2.4.6 Модуль logger.py – логування даних

Цей модуль відповідає за довготривале збереження інформації про роботу системи. У контексті дипломної роботи цього достатньо для текстового лог-файлу, куди зручно записувати:

– мітки часу;
– показники датчиків;
– дії контролера;
– команди від користувача.

Python має вбудовану бібліотеку logging, яка дозволяє легко організувати логування без сторонніх залежностей.

У файлі logger.py виконується базове налаштування логера:

– задається ім’я лог-файлу;
– вказується формат повідомлень;
– обирається рівень важливості повідомлень (INFO, DEBUG, WARNING тощо).

Після ініціалізації, інші модулі можуть просто імпортувати готовий об’єкт логера і робити виклики на кшталт:

```
import logger
logger.logging.info("Температура: 23.5 °C, вологість: 42%")
```

Рис. 2.6 Приклад Імпортування логера

2.4.7 Модуль конфігурації системи config.py

config.py – містить усі ключові налаштування, які визначають поведінку системи. Основна ідея — відокремити змінні параметри від логіки програми, щоб мати змогу змінювати їх без необхідності лізти в код кожного модуля.

Це зручно як на етапі тестування (можна швидко змінити пін чи цільову температуру), так і в майбутньому при масштабуванні або адаптації системи до нових умов.

У цьому файлі зазвичай містяться такі групи параметрів:

1. Налаштування апаратури
2. Бажані параметри мікроклімату
3. Параметри нечіткої логіки (ці значення можна залишити і в fuzzy_logic.py, але винесення їх сюди дозволяє зручніше експериментувати з налаштуваннями без перекомпіляції чи редагування логіки).
4. Дані для зовнішніх сервісів (Наприклад Токен Телеграм бота) тощо.

```
# config.py (фрагмент)
DHT_PIN = 4
PIR_PIN = 17
CO2_UART_PORT = "/dev/ttyS0"
TARGET_TEMPERATURE = 24.0   # °C
TARGET_HUMIDITY = 50.0      # %
LOG_FILE = "climate.log"
TELEGRAM_TOKEN = "XXXXXXXXXXXXXXXX"
```

Рис. 2.7 Приклад коду config.py

Інші модулі імпортують config.py і використовують ці змінні. Наприклад, sensors.py може зробити:

from config import DHT_PIN, PIR_PIN – замість того, щоб тримати “магічні числа” всередині коду.

Це підвищує читаність і підлаштовуваність програми: перенос контролера на інший Raspberry Pi або зміну піна легко виконати правкою одного рядка в конфігурації. Аналогічно, при розробці нечіткої логіки можна спочатку задати константи порогів у config.py і пробувати різні варіанти без ризику зламати інші частини коду.

2.5 Логіка взаємодії між модулями

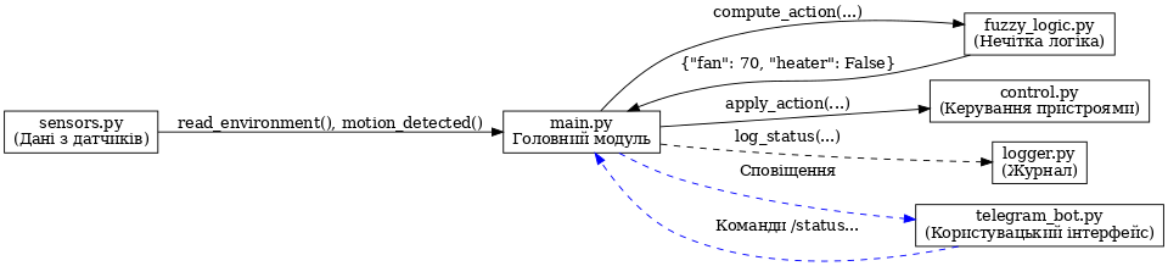


Рис.2.8 Логіка взаємодії між модулями.

На рис. 2.8 схематично показано, як інформація передається між основними програмними модулями системи. У центрі знаходиться main.py, який координує всю роботу системи.

1. Збір даних:

Модуль sensors.py постійно надає актуальні показники — температуру, вологість, наявність руху, рівень CO_2. main.py викликає відповідні функції (read_environment(), motion_detected() тощо) для отримання даних (стрілка від sensors.py до main.py).

2. Прийняття рішень:

Далі дані передаються у fuzzy_logic.py. Там, на основі заданих правил, система вирішує: потрібно вмикати обігрівач, чи зменшити швидкість вентилятора тощо. Повернене рішення — це вже готовий набір команд (наприклад, {'fan_speed': 70, 'heater': False}).

3. Керування пристроями:

Ці команди надсилаються до control.py, який вже напряму керує GPIO: вмикає/вимикає пристрої, регулює ШІМ для вентилятора тощо. Наприклад, функція apply_action() виконує дію за отриманим словником.

4. Логування:

Важливі події — значення з датчиків, спрацьовування правил, зміни станів — записуються в лог через logger.py. Це відображено на діаграмі пунктирною стрілкою від main.py до logger.py.

5. Зв’язок із користувачем:

Паралельно працює telegram_bot.py. Він обробляє команди від користувача (/status, /set_temp 25 тощо), а також надсилає повідомлення — наприклад, “Температура перевищила 30°C!”. Бот може звертатись до sensors.py або навіть безпосередньо до main.py (через імпорт чи передачу функцій при ініціалізації).

Як це працює у циклі

1. Старт:

main.py ініціалізує всі модулі, запускає бота, готує логер.

2. Опитування:

Система регулярно читає дані з датчиків.

3. Обробка:

Передає ці дані в fuzzy_logic.py, який видає рішення.

4. Керування:

Відповідно до рішення, control.py вмикає/вимикає пристрої.

5. Запис:

Все, що відбувається — логуються у logger.py.

6. Взаємодія:

У будь-який момент користувач може звернутись до системи через Telegram.

Гнучкість і розширення

Завдяки такій архітектурі, додавання нових функцій не вимагає повної перебудови системи. Наприклад, для реалізації ручного режиму досить додати змінну manual_mode у config.py і враховувати її в main.py:

Якщо manual_mode=True, рішення не береться з fuzzy_logic, а напряму з команд користувача через Telegram.

Весь інший цикл не змінюється — просто інша “гілка” логіки керування.

Представлена логіка взаємодії модулів демонструє продуману структуру та масштабованість. Кожен компонент виконує свою роль і спілкується з іншими через зрозумілі точки входу, що спрощує розробку, відлагодження і розширення системи.

В результаті отримано адаптивну систему клімат-контролю, яка поєднує - точність сенсорів, гнучкість нечіткої логіки, можливість ручного

втручання,зручність через Telegram, і, головне — реальну користь для повсякденного комфорту.

3. ІНТЕЛЕКТУАЛЬНА СИСТЕМА КЛІМАТ-КОНТРОЛЮ

3.1 Схема підключення Raspberry Pi до сенсорів DHT22, PIR та реле

Система використовує одноплатний комп'ютер Raspberry Pi як центральний контролер. До нього підключено сенсор температури й вологості DHT22, датчик руху PIR (HC-SR501) і релейний модуль, який керує виконавчими пристроями (наприклад, вентилятором чи обігрівачем).

3.1.1 Схема підключення датчика температури/вологості DHT22

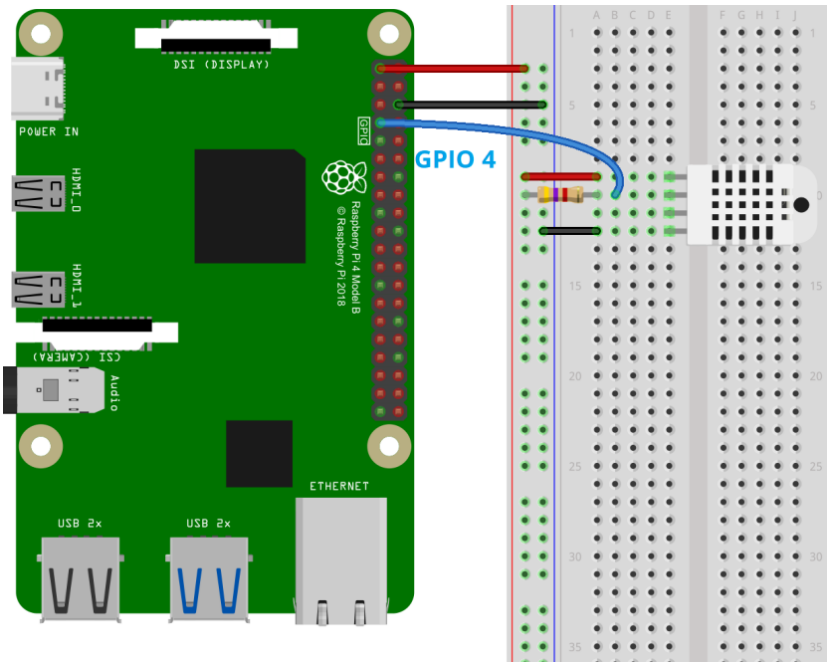


Рис. 3.1 Схема підключення датчика температури/вологості DHT22 до Raspberry Pi

Виводи датчика DHT22 (справа) з'єднані з відповідними пінами Raspberry Pi (зліва) на макетній платі: перший вивід DHT22 підключено до шини 3,3 В живлення (червоний провід), другий – до GPIO 4 Raspberry Pi (синій провід), причому між другим виводом і 3,3 В встановлено резистор 4,7 кОм для підтяжки лінії даних

(помаранчевий резистор), третій вивід датчика не використовується, четвертий з'єднано із землею GND (чорний провід). Таким чином, DHT22 отримує напругу у 3,3 В та передає цифрові виміряні значення температури та відносної вологості на виділений GPIO піна Raspberry Pi.

3.1.2 Схема підключення датчика руху PIR HC-SR501 та релейного модуля до Raspberry Pi.

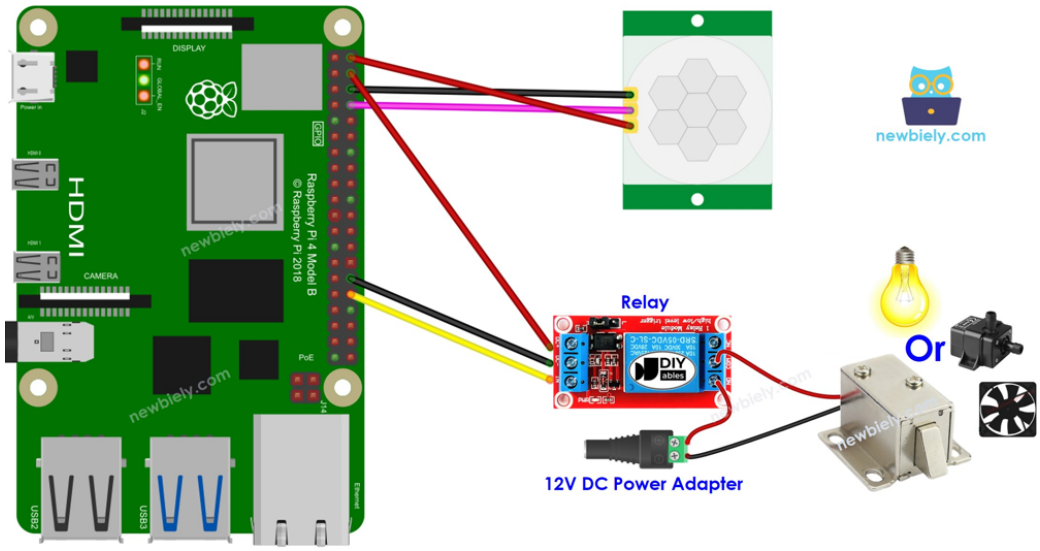


Рис.3.2 Схема підключення датчика руху PIR HC-SR501 та релейного модуля до Raspberry Pi.

Датчик PIR має три виводи: VCC (живлення), OUT (цифровий вихід сигналу) і GND (земля). Як показано на схемі, VCC PIR підключається до піну 5 В Raspberry Pi (червоний провід), GND – до одного з контактів GND Raspberry Pi (чорний провід), а вихід OUT – до вибраного GPIO (фіолетовий провід; у прикладі використано GPIO 14). Релейний модуль (на рисунку – плата реле з опторазв'язкою) також живиться від 5 В Raspberry Pi (жовтий провід на VCC модуля реле) і спільної землі GND (чорний провід). Керувальний вхід реле IN з'єднано з GPIO 12 Raspberry Pi (рожевий провід), що дозволяє вмикати або вимикати реле програмно. На самому модулі реле передбачені клеми для підключення зовнішнього пристрою: у

конфігурації з нормально розімкнутими контактами (NO – Normally Open) вибраний пристрій (наприклад, лампа чи вентилятор, позначені праворуч на схемі) під'єднується між клемою NO та спільною клемою COM реле, а джерело живлення цього пристрою – між COM і NC (або безпосередньо до пристрою в разі живлення від спільної шини) таким чином, що при активації реле коло замикається і пристрій отримує живлення. У підсумку, Raspberry Pi забезпечує живлення сенсорів (3,3 В для DHT22, 5 В для PIR і модуля реле) та приймає/подає сигнали через GPIO: зчитує цифрові дані з DHT22 і PIR та керує станом реле.

3.1.3 Пояснення з'єднань

Наведена схема гарантує правильну роботу всіх компонентів при дотриманні рівнів логічних сигналів Raspberry Pi. Зокрема, датчик DHT22 працює від 3,3 В і передає дані на GPIO 4 по однопровідному інтерфейсу (1-Wire протокол), що потребує резистора підтяжки ~4,7 кОм до живлення. Датчик PIR живиться від 5 В (на модулі HC-SR501 є внутрішній стабілізатор, тому допустимо 5 В живлення), а його вихідний сигнал є 3,3 В логікою, сумісною з GPIO Raspberry Pi. Релейний модуль розрахований на керування від 3,3–5 В логічного сигналу, тому пряме підключення до GPIO припустиме. Необхідно спільно заземлити всі компоненти – об'єднати всі GND Raspberry Pi, датчиків і реле, щоб замкнути електричне коло. Таким чином, апаратна частина системи складається з мінімуму компонентів та є відносно простою для збірки.

3.2 Типові сценарії роботи системи

Розроблена система клімат-контролю автоматично підтримує комфортні умови в приміщенні й дозволяє дистанційне керування через Telegram. Нижче наведено кілька типових сценаріїв, які показують, як вона реагує на різні ситуації.

3.2.1 Сценарій 1: Звичайна робота, є люди

Температура в кімнаті — 25°C, вологість — 60%. Датчик руху фіксує присутність — отже, система працює в активному режимі. Контролер класифікує температуру як комфортну, а вологість — середню, тому вентилятор не вмикається. Telegram-бот на запит /status може відповісти:

«Температура: 25.0°C

Вологість: 60.0%

Рух: так

Вентилятор: OFF.»

3.2.2 Сценарій 2: Підвищилась вологість у приміщенні

Припустимо, люди в кімнаті залишились, але вологість зросла до 80%. Температура стабільна — 25°C. Система спрацьовує за правилом "висока вологість — потрібне провітрювання", і вмикає вентилятор на середню швидкість. Telegram-бот може повідомити:

Вологість 80%. Увімкнено вентиляцію.

Коли вологість знижується нижче порогу (наприклад, до 65–70%), вентилятор автоматично вимикається:

«Вологість у нормі, вентилятор вимкнено.»

3.2.3 Сценарій 3: Відсутність людей - енергозбереження

Температура підвищилась до 28°C, вологість — 50%, але руху немає. Система переходить у “економний” режим: не вмикає вентиляцію, бо немає сенсу витрачати енергію. Якщо показники різко вийдуть за межі (наприклад, температура

перевищить 35°C), система може все ж активуватись для безпеки — охолодити приміщення, навіть без присутності людей. Telegram-бот у цей момент повідомить:

«Люди відсутні, переходжу в економний режим.»

Як тільки хтось зайде — система автоматично повернеться до активного режиму.

3.2.4 Сценарій 4: Користувач керує вручну

Користувач у будь-який момент може надіслати команду, наприклад:

– /on — увімкнути вентилятор вручну. Система відповість: Вентилятор увімкнено користувачем.
– /off — вимкнути автоматичне керування, перейти в ручний режим. Система більше не буде вмикати/вимикати пристрої самостійно.
– /auto — повернення до автоматичного керування.

Ці команди дозволяють змінювати умови у приміщенні вручну та тимчасово зупиняти систему.

3.2.5 Сценарій 5: Виявлено помилку

Якщо один із сенсорів перестає відповідати, система це фіксує. Наприклад, DHT22 кілька разів поспіль повернув None — бот надішле повідомлення:

«Помилка: датчик температури не відповідає.»

Це дозволяє вчасно виявляти та виправляти помилки у керуванні мікрокліматом.

3.2.6 Типові помилки та їх обробка

Зчитування DHT22: Цей датчик може час від часу не відповідати — таке буває при короткочасних збоях зв’язку. Щоб уникнути помилок, у коді передбачено перевірку:

«if humidity is None or temperature is None:»

Якщо дані не зчитались — система не зависає, просто пропускає цей цикл або логує попередження. У Telegram-боті це також враховано: якщо користувач надішле команду /status, а нових показників ще немає, бот відповість: “Дані датчика тимчасово недоступні.” Це краще, ніж показувати застарілі або “нульові” значення.

Хибні спрацювання PIR: PIR-датчик руху іноді реагує на незначні зміни — наприклад, тінь, спалах світла чи завади. Щоб уникнути помилкових дій, сигнал руху вважається дійсним тільки після підтвердження: датчик має кілька разів поспіль показати “рух”. Це реалізовано як таймер або фільтр: наприклад, три цикли підряд із сигналом = рух.

Аналогічно, режим “відсутності людей” активується лише після кількох хвилин повної тиші — тобто, поодиноке “клацання” не змінить стан системи. Такий підхід дозволяє уникнути безпідставних вмикань вентилятора чи обігрівача.

Обробка виключень і відмов модулів: У практиці траплялось, що неправильна робота з GPIO або непередбачений виняток (наприклад, помилка в бібліотеці) зупиняли скрипт. Щоб цього уникнути, головний цикл програми обгорнуто в try/except. Якщо трапляється помилка — вона записується у лог-файл, і система або переходить у безпечний режим, або перезапускає окремий модуль.

Це критично для стабільної тривалої роботи: навіть якщо щось пішло не так, усе продовжує працювати — без потреби перезапускати Raspberry Pi вручну.

3.4 Забезпечення надійності та відмовостійкості

Система проєктувалась так, щоб працювати постійно — автономно, у фоні, без потреби щоденного втручання. Для цього передбачено кілька рівнів захисту: перевірки в коді, системне відновлення через systemd, журналювання подій і сповіщення через Telegram.

3.4.1 Автозапуск і контроль процесу через systemd

Щоб система автоматично запускалась після ввімкнення живлення та у випадку збоїв, створено systemd-сервіс climate.service. Він має параметр Restart=always, який дозволяє перезапустити скрипт, якщо той раптом завершився з помилкою. Наприклад, якщо сталася непередбачувана помилка у коді — процес завершується, але через 5 секунд (RestartSec=5) запускається знову.

Це означає, що навіть якщо система “впала” — вона сама підніметься.

Через команду systemctl status climate.service можна швидко перевірити, чи все працює: коли був останній запуск, яка помилка була (якщо була), і навіть побачити останні кілька повідомлень.

Опція Watchdog: systemd також дозволяє вмикати “сторожовий таймер”, який слідкує, чи програма надсилає сигнали “я живий”. У цьому проєкті його не застосовували (вистачило автоперезапуску), але у критичних системах його варто вмикати.

3.4.2 Перевірки в коді та робота при помилках

Всі ключові ділянки скрипту обгорнуті в try/except. Якщо, наприклад, датчик DHT22 не відповів — система не зупиняється. Замість цього вона просто ігнорує

невдалу спробу зчитування, і використовує останні коректні дані або чекає наступної успішної спроби.

Аналогічно — якщо зникає інтернет і бот не може відправити повідомлення, Telegram-винятки ловляться, записуються у лог, але система працює далі. Бот автоматично пробує відновити з’єднання при наступному циклі.

3.4.3 Журналювання подій

Усі дії логуються у файл climate.log. Це дозволяє бачити історію роботи системи - коли спрацював датчик, коли було увімкнено обігрівач, які дані повертали сенсори.

Для збереження пам’яті передбачено ротацію логу — наприклад, коли розмір перевищує 5 МБ, створюється новий файл. Це простий спосіб уникнути переповнення SD-карти на Raspberry Pi.

3.4.4 Сповіщення в Telegram

Система не мовчить, якщо щось йде не так. Якщо, наприклад, температура перевищує критичне значення або сенсор не відповідає кілька циклів поспіль, користувач отримає повідомлення:

"Помилка: датчик температури не відповідає."

Ця функція користувачу вчасно побачити про наявність проблем та, за потреби їх виправити. Або, якщо проблема в апаратній частині, то виконати заміну несправних компонентів.

3.4.5 Перезапуск після збоїв живлення

Raspberry Pi при раптовому знеструмленні просто перезапускається. Завдяки systemd, скрипт стартує автоматично. Всі попередні дані не зберігаються (окрім тих, що встигли записатися в лог), але це некритично — система продовжує моніторинг “з чистого листа”.

Щоб знизити ризик псування SD-карти, журнал можна зберігати тимчасово в /tmp, а потім копіювати на диск раз на добу. Також можна використовувати файлову систему з журналюванням.

3.4.6 Резервне керування

На випадок, якщо Telegram-бот недоступний (наприклад, через проблеми з мережею або API), залишається можливість зайти на Raspberry Pi через SSH. Там можна подивитись лог, перевірити статус, або навіть вручну ввімкнути чи вимкнути обладнання.

Цей резервний канал — не основний, але він критично важливий для надійності в реальних умовах.

ВИСНОВОК

У межах цієї дипломної роботи було спроєктовано та реалізовано адаптивну інтелектуальну систему керування кліматом для офісних приміщень, засновану на відкритих апаратних і програмних рішеннях. Основна ідея полягала в тому, щоб створити гнучку, доступну й незалежну від сторонніх сервісів систему, яка могла б автономно підтримувати комфортний мікроклімат.

Під час дослідження було проаналізовано популярні комерційні рішення (Nest, Tado, Ecobee, Xiaomi Smart Home тощо). Виявилось, що попри зручність і автоматизацію, вони мають низку обмежень: високу вартість, закриту архітектуру, складність інтеграції та залежність від хмарних платформ. Це й стало підставою для створення власної адаптивної системи з можливістю модифікації під конкретні потреби.

Результатом роботи стала модульна система, до складу якої входять:

– Raspberry Pi як центральний контролер;
– сенсори DHT22, PIR і CO_2 (опційно);
– програмна реалізація алгоритму нечіткої логіки для прийняття рішень;
– Telegram-бот для зручного дистанційного керування;
– система логування для фіксації подій і моніторингу.

Система функціонує автономно. У режимі реального часу реагує на зміну температури, вологості й присутності людей, переходить у енергозберігаючий режим за потреби, а використання нечіткої логіки робить зміну станів непомітною для користувача.

У процесі дослідження було доведено, що побудова інтелектуальної кліматичної системи з відкритих компонентів — цілком реальна задача, яку можна вирішити навіть без доступу до дорогих рішень. При цьому проєкт зберігає простоту, масштабованість і можливість розвитку.

Подальший розвиток цього проекту може включати: інтеграцію мобільного застосунку чи веб-інтерфейсу, додавання сенсорів (світла, шуму, пилу), розширення

логіки керування або аналітику зібраних даних для побудови графіків, виявлення трендів і глибшого аналізу мікроклімату.

Розглянута нами система поєднує гнучкість налаштування, стабільну роботу, простоту використання та відкритість до змін. Вона підходить як для практичного впровадження в умовах обмеженого бюджету, так і у вигляді платформи для навчальних та дослідницьких цілей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Raspberry Pi Foundation. *Getting Started with Raspberry Pi*. https://www.raspberrypi.com (дата звернення: 2025-05-25).
2. Adafruit. *DHT22 Temperature-Humidity Sensor Datasheet*. https://learn.adafruit.com (2023).
3. Ecobee Inc. *Smart Thermostat Documentation*. https://www.ecobee.com (2024).
4. Nest Labs by Google. *Nest Learning Thermostat - Technical Overview*. https://support.google.com/googlenest(2024).
5. Tado GmbH. *Tado° Smart Thermostat Features*. https://www.tado.com (2024).
6. Netatmo. *Smart Thermostat Technical Specifications*. https://www.netatmo.com (2024).
7. Xiaomi Smart Home Ecosystem. *Mi Home Platform Documentation*. https://www.mi.com/global (2024).
8. Python Software Foundation. *Official Python Documentation*. https://docs.python.org (2024).
9. Telegram. *Telegram Bot API Documentation*. https://core.telegram.org/bots/api (2025).
10. scikit-fuzzy. *Fuzzy logic toolkit for Python*. https://github.com/scikit-fuzzy (2023).
11. IEC 61131-3:2023. *Programming Languages for Industrial Control Systems*. International Electrotechnical Commission (2023).
12. Мінцифра України. *IoT-стратегія України 2021–2025*. https://thedigital.gov.ua (2021).
13. Khan, F. et al. *Fuzzy Logic in IoT Systems: A Review*. Journal of Ambient Intelligence and Humanized Computing, Springer, 2021.
14. Zhou, X. et al. *A Smart HVAC Control Framework Based on Fuzzy Logic and Wireless Sensor Networks*. *Sensors*, MDPI, 2022.
15. Мельник І. С. *Системи автоматизації на базі Raspberry Pi*. Наукові праці НТУУ "КПІ", 2021.

ДОДАТКИ

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

ПРЕЗЕНТАЦІЯ ДО КВАЛІФІКАЦІЙНОЇ БАКАЛАВРСЬКОЇ РОБОТИ
на тему:
«Інтелектуальна система керування кліматом у офісних приміщеннях на основі IoT»

Виконав: здобувач вищої освіти гр. ІСЗ-51 Михайло Горбань
Керівник: Ольга Жидка

ДУІКТ

2025

МЕТА ТА ЗАВДАННЯ

Мета: проєктування інтелектуальної системи керування кліматом на основі Raspberry Pi та нечіткої логіки.

Об'єкт: автоматизована система клімат-контролю.

Предмет: програмно-апаратна система керування мікрокліматом з використанням сенсорів і Telegram.

Завдання:
- Аналіз існуючих рішень.
- Розробка архітектури системи.
- Реалізація алгоритмів нечіткої логіки.
- Побудова Telegram-інтерфейсу.

ДУІКТ

АКТУАЛЬНІСТЬ

- Комерційні системи часто дорогі або закриті для модифікації.
- Власне рішення — доступне, адаптивне та гнучке.
- Дає змогу досліджувати інтелектуальні підходи до автоматизації мікроклімату.

ОГЛЯД РІШЕНЬ

- **Nest:** самонавчання, але закритий код.
- **Tado:** геолокація, платна підписка.
- **Xiaomi:** дешево, але слабка кастомізація.
- **Власна система:** гнучка, з відкритим кодом.

Порівняння розглянутих рішень.

Система	Гнучкість	Telegram	Ціна	Модифікація	Підписка
Nest	Низька	Ні	Висока	Немає	Так
Tado	Середня	Ні	Висока	Обмежено	Так
Xiaomi	Середня	Частково	Середня	Низька	Так
OpenHAB	Висока	Так	Середня	Так	Ні
Наша система	Висока	Так	Низька	Так	Ні

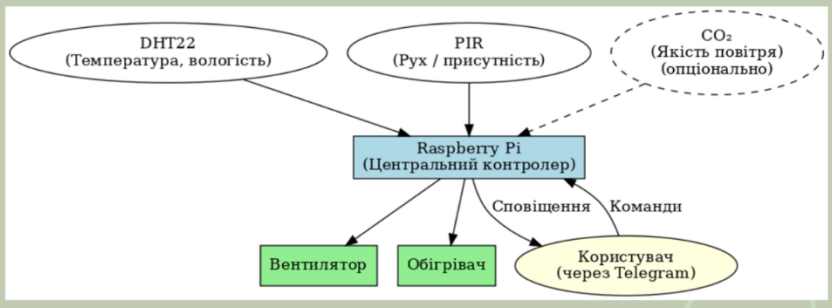
Рис. 1.3 Google Nest Learning Thermostat

Рис. 1.5 Екосистема Xiaomi Smart Home

Рис. 1.1 Tado Smart Thermostat

АРХІТЕКТУРА СИСТЕМИ

- **Raspberry Pi** — центральний вузол.
- **DHT22** — сенсор температури та вологості.
- **PIR** — сенсор присутності.
- **Telegram API** — керування користувачем.
- **Алгоритм нечіткої логіки** — адаптивне керування.



ЗАГАЛЬНА АРХІТЕКТУРА СИСТЕМИ (АПАРАТНА СХЕМА)

МОДУЛІ СИСТЕМИ

- **main.py:** основний цикл логіки.
- **sensors.py:** зчитування з датчиків.
- **fuzzy_logic.py:** обробка нечітких правил.
- **control.py:** керування обладнанням.
- **telegram_bot.py:** взаємодія з користувачем.

НЕЧІТКА ЛОГІКА

- Вхідні змінні: температура, вологість, присутність.
- Вихідні: швидкість вентиляції, обігрів.
- Метод: Mamdani.
- Плавне й контекстне керування без жорстких порогів.

```python
# fuzzy_logic.py (фрагмент)
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

# Нечіткі змінні (Antecedents для входів, Consequents для виходів)
temp_diff = ctrl.Antecedent(np.arange(-5, 6, 1), 'temp_diff')   # відхилення темп. від бажаної
humidity = ctrl.Antecedent(np.arange(0, 101, 1), 'humidity')    # вологість у %
fan_speed = ctrl.Consequent(np.arange(0, 101, 1), 'fan_speed')  # швидкість вентилятора %

# Визначення функцій належності для температури (холодно, нормально, жарко)
temp_diff['cold'] = fuzz.trimf(temp_diff.universe, [-5, -5, 0])
temp_diff['ok']   = fuzz.trimf(temp_diff.universe, [-2, 0, 2])
temp_diff['hot']  = fuzz.trimf(temp_diff.universe, [0, 5, 5])
# Для вологості (сухо, норм, вологo)
humidity['dry']   = fuzz.trimf(humidity.universe, [0, 0, 40])
humidity['med']   = fuzz.trimf(humidity.universe, [30, 50, 70])
humidity['humid'] = fuzz.trimf(humidity.universe, [60, 100, 100])
# Для швидкості вентилятора (низька, середня, висока)
fan_speed['low']  = fuzz.trimf(fan_speed.universe, [0, 0, 50])
fan_speed['high'] = fuzz.trimf(fan_speed.universe, [50, 100, 100])

# Приклад правила: Якщо жарко АБО вологo, то вентилятор високий
rule1 = ctrl.Rule(temp_diff['hot'] | humidity['humid'], fan_speed['high'])
# Якщо холодно І сухо, то вентилятор низький
rule2 = ctrl.Rule(temp_diff['cold'] & humidity['dry'], fan_speed['low'])

# Створення контролера на основі правил
fan_ctrl = ctrl.ControlSystem([rule1, rule2, ...])
fan_simulation = ctrl.ControlSystemSimulation(fan_ctrl)

def compute_action(temp, hum, motion, target_temp):
    """Обчислює необхідну швидкість вентилятора (0–100%) та інші дії."""
    # Розрахунок відхилення температури
    diff = temp - target_temp
    # Нечіткий висновок для швидкості вентилятора
    fan_simulation.input['temp_diff'] = diff
    fan_simulation.input['humidity'] = hum
    fan_simulation.compute()  # виконує нечітке обчислення
    fan = fan_simulation.output['fan_speed']
    # Врахування PIR: якщо нікого немає, можна знизити швидкість
    if motion is False:
        fan = min(fan, 50.0)  # обмежити макс. 50% при відсутності людей
    return {'fan_speed': fan}
```

ПРИКЛАД КОДУ FUZZY-LOGIC.PY

СЦЕНАРІЇ РОБОТИ

1. Присутність людей у кімнаті:
   - Температура: 25°C, вологість: 60%. Система не вмикає вентиляцію, бо умови комфортні.
   - Telegram-бот:
   - «Температура: 25.0°C
   - Вологість: 60.0%
   - Рух: так
   - Вентилятор: OFF.»
2. Вологість підвищена:
   - Вологість: 80%, присутні люди → система вмикає вентилятор.
   - Telegram-бот:
   - «Вологість 80%. Увімкнено вентиляцію.»
3. Немає людей — режим енергозбереження:
   - Система автоматично переходить у пасивний режим, не вмикає вентилятор.
   - Telegram-бот:
   - «Люди відсутні, переходжу в економний режим.»

TELEGRAM-БОТ

- Надсилає повідомлення про стан системи.
- Приймає команди користувача.
- Показує температуру, вологість, режим.
- Повідомляє про рух чи перевищення норм.

```python
# telegram_bot.py (фрагмент)
from telegram.ext import Updater, CommandHandler
from sensors import read_environment

TOKEN = "XXXXXXXXXXXXXXXXXXXXXXXXX"  # Токен бота (в реальн. коді з config.py)

updater = Updater(token=TOKEN, use_context=True)
dispatcher = updater.dispatcher

def cmd_start(update, context):
    """Обробник команди /start – вітальне повідомлення."""
    update.message.reply_text("Вітаю! Я – бот клімат-контролю. Надішліть /status для перегляду стану.")

def cmd_status(update, context):
    """Обробник команди /status – відповідає поточними показниками."""
    temp, hum = read_environment()
    update.message.reply_text(f"Температура: {temp:.1f}°C\nВологість: {hum:.1f}%")

dispatcher.add_handler(CommandHandler("start", cmd_start))
dispatcher.add_handler(CommandHandler("status", cmd_status))

# Запуск бота у фоновому режимі
updater.start_polling()
print("Telegram bot started...")

```

ПРИКЛАД КОДУ ДЛЯ TELEGRAM_BOT.PY

ВИСНОВКИ

- Створено доступну та гнучку систему клімат-контролю.
- Реалізовано інтелектуальне керування за допомогою нечіткої логіки.
- Система легко масштабується і може використовуватись у розумних офісах.


ДЯКУЮ ЗА УВАГУ!
Готовий відповісти на ваші запитання.
ДУІКТ