МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний університет будівництва і архітектури

Міщук Д. О., Рашківський В. П.

ПРОГРАМУВАННЯ РОБОТОТЕХНІЧНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

Конспект лекцій
для здобувачів першого (бакалаврського) рівня освіти
спеціальностей 126 «Інформаційні системи та технології»
і 122 «Комп`ютерні науки»

Київ 2024

УДК 621.8
М21

Рецензент **Олександр Терентьєв**, д-р техн. наук, професор, Київський національний університет будівництва і архітектури

Затверджено на засіданні навчально-методичної ради Київського національного університету будівництва і архітектури, протокол № 8 від 28 березня 2024 року.

Міщук Д.О.

М21 Програмування робототехнічних інформаційних систем: конспект лекцій / Д.О. Міщук, В.П. Рашківський. – Київ : КНУБА, 2024. – 220 с.

Містить загальні відомості про робототехнічні інформаційні системи та способи їхнього програмування. Викладено теоретичні основи робототехніки і апаратної та програмної частин роботів, а також основи програмування робототехнічних систем.

Розглянуто принципи і технології створення автоматизованих систем програмного керування на основі інформаційних систем.

Призначено для здобувачі першого (бакалаврського) рівня освіти спеціальностей 126 «Інформаційні системи та технології» і 122 «Комп`ютерні науки».

УДК 621.8

ЗМІСТ

ВСТУП

Сучасна робототехніка основана на комплексних знаннях з різних галузей науки, а саме інформаційних технологій, механіки, мехатроніки, електротехніки, електроніки та автоматичного управління. Інформаційні системи є складовою частиною будь-якої роботизованої системи, тому їхнє вивчення важливо для якісної підготовки спеціалістів за цією тематикою.

В конспекті лекцій висвітлено основні теми, які треба опанувати для засвоєння курсу «Програмування робототехнічних інформаційних систем».

Метою курсу є вивчення основ робототехнічних систем, засобів комунікації між роботами і людиною та способів програмування логіки функціонування інформаційної системи робота.

Програмування роботів може здійснюватися як онлайн, так і офлайн. В першому випадку програмування роботів здійснюється безпосередньо на місці їх встановлення. При цьому можне застосовуватися метод *Teach-In*, коли оператор пробота за допомогою консолі направляє робота в задану ділянку простору і виконує необхідні завдання, а робот запам'ятовує координати розташування, швидкість руху на кожній ділянці. За методом *Playback* оператор вручну здійснює обхід робочими елементами робота заданої траєкторії руху. Інший розвинений спосіб управління роботом – офлайн-програмування. У такому разі програми пишуться на звичайному комп'ютері в редакторі, що постачається з комплексом програмного забезпечення робота: написання програм для сучасних роботів доступне будь-якому співробітнику після недовгого навчання.

В курсі викладено методики онлайн та офлайн програмування інформаційних робототехнічних систем.

За результатами навчання здобувачі освіти мають

знати:

- види і класифікацію роботів, їхні основні технічні параметри;
- основні відомості про кострукції колісних та літальних робототехнічних систем;
- компоненти виконавчої та інформаціної систем робота;

вміти:

- добирати роботів для виконання поставленої задачі;
- програмувати інформаційні системи роботів;
- створювати алгоритми управління та здійснювати їхню реалізацію.

Модуль 1. ТЕОРЕТИЧНІ ОСНОВИ АПАРАТНОЇ ЧАСТИНИ РОБОТІВ

Лекція № 1. Загальні відомості з робототехнічних систем

Лекція № 2. Теорія БПЛА

Лекція № 3. Теорія транспортних роботів

Лекція № 4. Інформаційні системи роботів

Лекція № 5. Системи навігації та управління роботів

Лекція № 1. Загальні відомості з робототехнічних систем

Робот – це програмований пристрій з двома або більше ступенями рухливості з певним рівнем автономності, здатний переміщуватися у зовнішньому середовищі з метою виконання програмованих завдань. Умовно всіх роботів можна класифікувати на *промислові* та *сервісні*.

Промисловий робот – це автоматично керований, програмований пристрій, який може бути встановлений стаціонарно або на мобільній платформі для застосування в цілях промислової автоматизації (рис. 1.1).

Сервісний робот є програмованим пристроєм, який виконує деяку роботу, проте не застосовується для промислової автоматизації (рис. 1.2).

Промислові роботи поділяються так:

- *роботи І покоління*, або промислові маніпулятори, здатні переміщувати будь-які предмети (заготівки, готові вироби, інструменти) за заданою траєкторією, швидкістю та точністю, проте мають переважно спрощену інформаційну систему (обмежена кількість датчиків), яку зазвичай проєктують під певну технологічну функцію робота, а отже, такі роботи призначені для вузькопрофільного використання;
- *роботи ІІ покоління* можуть бути мобільними, інформаційними та адаптивними і на відміну від промислових маніпуляторів мають більш розвинену інформаційну систему з датчиками зворотного зв’язку, системами комп’ютерного зору тощо, що дає можливість коригувати їхні програми під різні види не пов’язаних між собою завдань;
- *роботи ІІІ покоління* є найбільш складними та сучасними, а програмне забезпечення побудовано із застосуванням штучного інтелекту.

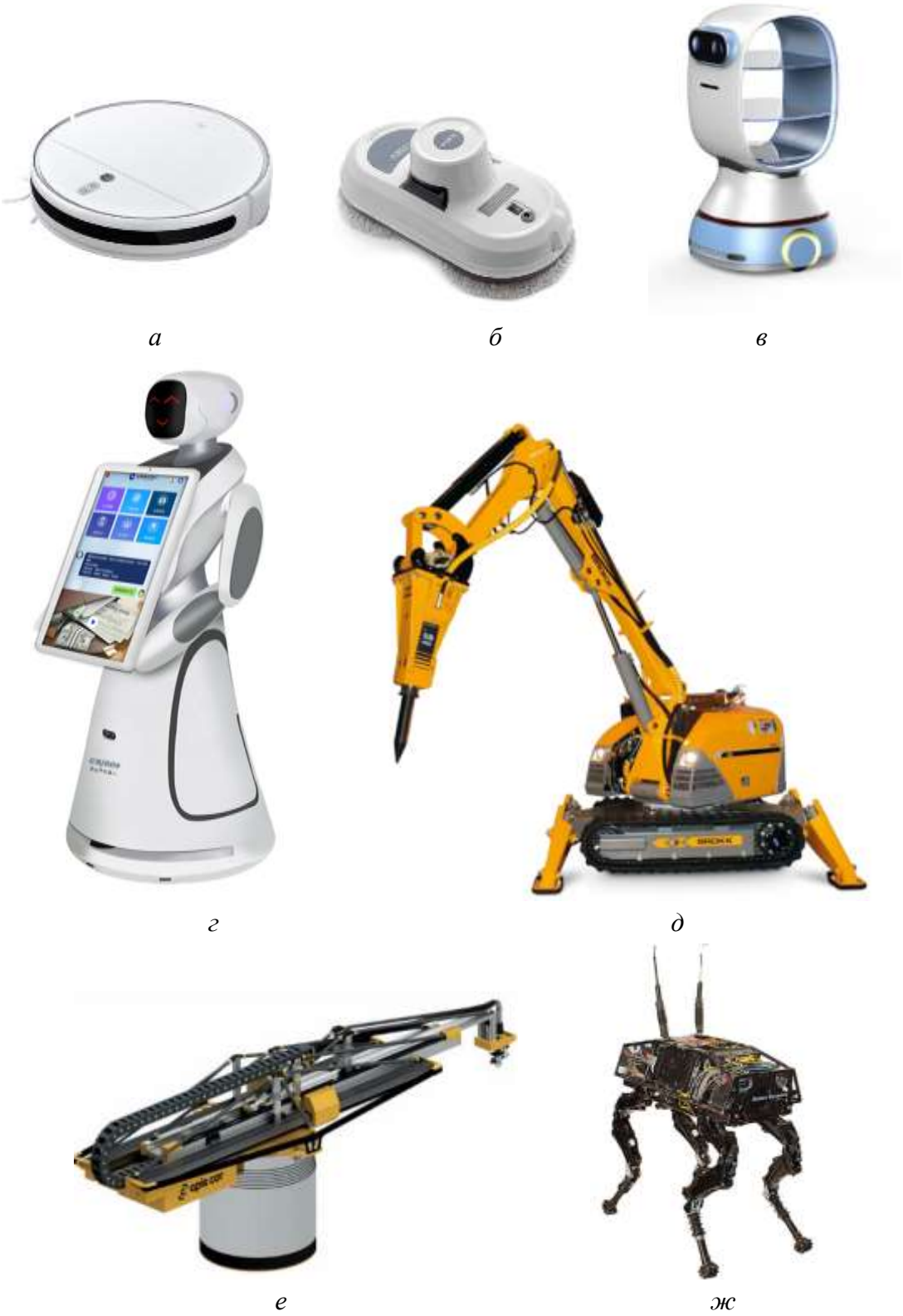
а *б* *в*

г *д* *е*

Рис. 1.1. Промислові роботи: *а* – універсальний маніпулятор KR Quantec 210 R2700 (KUKA); *б* – пакетувальний маніпулятор IRB8700 (ABB); *в* – високоточний складальний SCARA маніпулятор MYS450F (YASKAWA); *г* – зварювальний робот BA006N (KAWASAKI0; *д* – фарбувальний маніпулятор MPX2600 (YASKAWA); *е* – вантажний маніпулятор M-2000iA/2300 (FANUC)

Промислові маніпулятори, які виконують основні технологічні функції на підприємстві, можуть бути:

- універсальними, які виконують різноманітні технологічні операції залежно від встановленого робочого органа;
- складальними, що виконують пакетування і складання;
- зварювальними, що виконують зварювальні операції;
- фарбувальними, призначені для фарбувальних операцій;
- перевантажувальними, що виконують завантажувально-розвантажувальні операції;
- вимірювальними, що виконують вимірювальні операції;
- оброблювальними, призначені для операцій механообробки (шліфування, видалення, різання тощо).



а *б* *в*

г *д*

е *ж*

Рис. 1.2. Сервісні роботи: *а* – робот-пилосос Vacuum-Mop 2 (Xiaomi Mi Robot); *б* – робот Hobot 188 для чищення вікон; *в* – робот-офіціант BenBen; *г* – інформаційни робот для реклами Csjbot Amy Plus Artificial; *д* – демонтажний робот Brokk-180; *е* – 3D-принтер будівельний; ж – робот BigDog (Boston Dynamics)

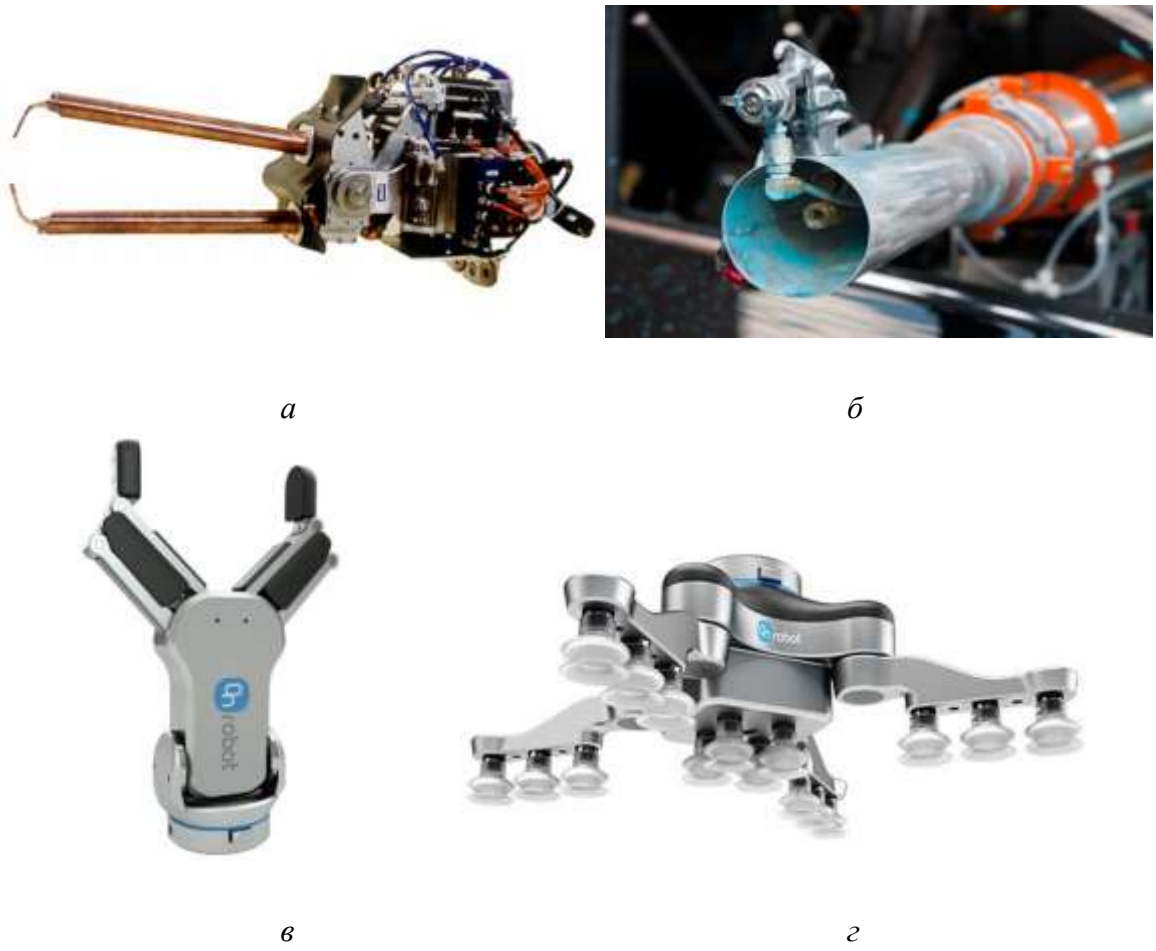
Сервісні роботи поділяють так:

- роботи-прибиральники, які призначені для чищення підлоги, стін, вікон, очищення цистерн, труб, великогабаритних корпусів, тощо;
- роботи для роботи в громадських місцях, які обслуговують готелі, ресторани, надають інформацію, вказують маршрути, рекламують товари тощо;
- роботи для обслуговування виробничих приміщень, обладнання, комунікацій, резервуарів;
- роботи для будівництва;
- роботи для обслуговування логістичних центрів;
- медичні роботи;
- сільськогосподарські роботи;
- аварійно-рятувальні роботи;
- роботи спеціального військового призначення.

Промислові роботи використовують для переміщення, подавання або зняття деталей та інструментів між різними верстатами або технічним обладнанням. Промислові роботи можуть працювати 24 години на добу без вихідних, тим сами зменшуючи витрати на виробництво на 20%, а в разі їхнього обладнання технічним зором та спеціальними робочими органами можна виконувати складання високотехнологічного обладнання, наприклад, друкованих плат.

Робочий орган промислового робота (автооператора) – складова частина виконавчого пристрою промислового робота (автооператора) для безпосереднього виконання технологічних функцій та (або) допоміжних переходів (рис. 1.3). Прикладами робочих органів можуть бути зварювальні кліщі, краскопульт (розпилювач фарби), захоплювальний пристрій.

Виконавчий механізм промислового робота (автооператора) – пристрій робота (автооператора), який виконує всі його функції руху відповідно до заданих командних сигналів. Виконавчий пристрій закінчується робочим органом у вигляді захоплювача для утримання інструмента, деталей або контрольно-вимірювальних приладів.



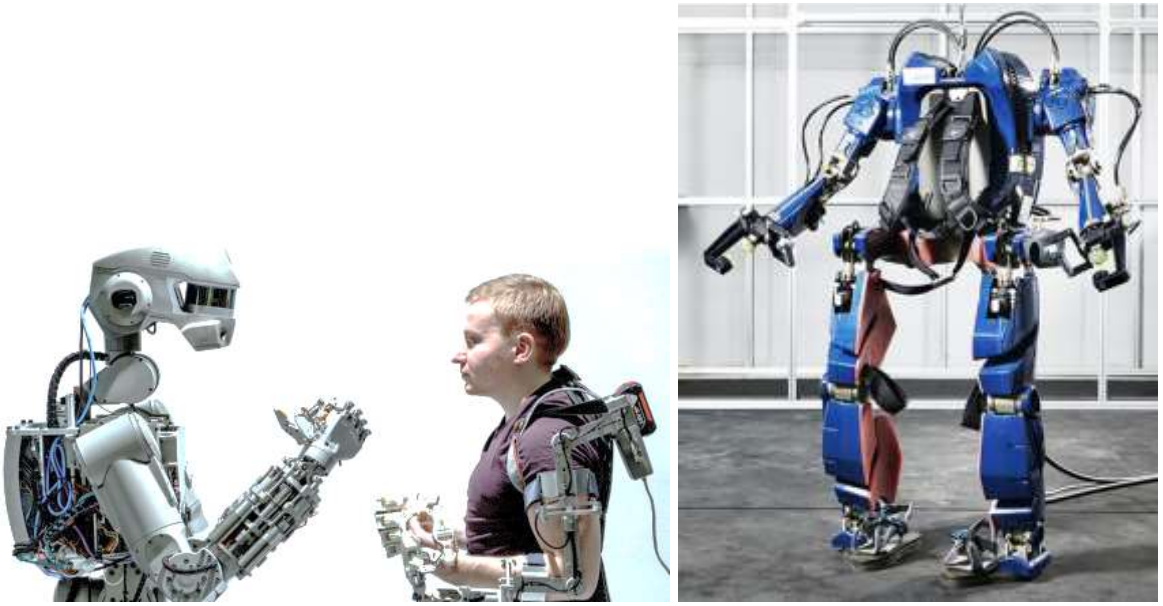
а *б*

в *г*

Рис. 1.3. Види робочих органів промислових роботів: *а* – зварювальні кліщі; *б* – краскопульт; *в* – механічний захоплювач; *г* – вакумний захоплювач

Промислові роботи класифікують за такими ознаками: *спеціалізація*, *вантажопідйомність*, *кількість ступенів вільності*, *можливість переміщення*, *спосіб встановлення*, *тип системи координат*, *тип приводу*, *вид керування*, *спосіб програмування*. Основна термінологія в галузі промислових роботів визначається ГОСТ 25686-85 «Манипуляторы, автооператоры и промышленные роботы. Термины и определения», а технічні показники – ГОСТ 4.480-87 «Система показателей качества продукции. Роботы промышленные. Номенклатура основных показателей».

Копіювальні роботи мають органи управління, наприклад маніпулятор, які повністю ідентичні виконавчому органу робота (із зменшеним, збільшеним або однаковим масштабом за геометричними розмірами та зусиллями) та систему керування, яка забезпечує бажаний рух, заданий людиною-оператором завдяки копіюванню рухів людини з точністю до масштабного коефіцієнта (рис. 1.4, *а*).



а *б*

Рис. 1.4. Робот-копір (*а*) та екзоскелетон (*б*)

Екзоскелетони є антропоморфними конструкціями, які зазвичай «одягаються» на руки, ноги або тіло людини і слугують для відтворення (копіювання) їхніх рухів з масштабним коефіцієнтом за зусиллями (рис. 1.4, *б*). Інший різновид роботів з біотехнічним управлінням – це роботи, керовані людиною з пульта управління.

Антропоморфний робот – це робот, який має схожу з живою істотою (людиною або твариною) будову та аналогічні особливості (рис. 1.5). Антропоморфні роботи максимально універсальні, і вони здатні виконувати не передбачені заздалегідь функції, наприклад, під час аварійних або інших позаштатних ситуацій. Проте антропоморфний вигляд робота може створювати не завжди сприятливий психологічний клімат для людини.

а *б*

Рис. 1.5. Антроморфні роботи: *а* – робот-гуманоїд Atlas (Boston Dynamic); *б* – робот Spot (Boston Dynamic)

Колаборативний робот (кобот) – це автоматичний пристрій, який може працювати спільно з людиною для створення або виробництва різних продуктів. Коботи в основному складаються з маніпулятора та перепрограмованого пристрою управління, який формує керівні впливи, що задають потрібні рухи виконавчих органів маніпулятора (рис. 1.6).

а *б*

Рис. 1.6. Колоборативні роботи-маніпулятори UR16e (Universal Robots) (*а*) та CR-7iA/L (Fanuc) (*б*)

Колаборативні роботи застосовують на виробництві для вирішення завдань, які не можна повністю автоматизувати. Колаборативні роботи розроблено для безпечної роботи з операторами завдяки таким технологіям, як зворотне зусилля, малоінерційні серводвигуни, еластичні виконавчі механізми та технологія виявлення зіткнень, що обмежують потужність та силу роботів до рівня, прийнятного для контакту. Стандарти безпеки ISO 10218-1, ISO 10218-2 та технічні характеристики ISO TS-15066 визначають функції безпеки та продуктивність колаборативного робота.

Система координат, в якій працює робот, визначає кінематику основних його рухів, форму робочої зони та тип структурної схеми.

Прямокутна (декартова) система координат буває двох видів (рис. 1.7, *а*):

- прямокутна плоска – характеризується переміщенням робочого органа робота за двома взаємно перпендикулярними осями;
- прямокутна просторова – характеризується переміщенням робочого органа робота за трьома взаємно перпендикулярними осями.

В прямокутній системі координат робоча зона робота має форму прямокутника або паралелепіпеда.

Полярна система координат буває трьох виконань (рис. 1.7, *б, в*):

- полярна плоска – характеризується радіус-вектором, що змінюється залежно від лінійного переміщення робочого органа вздовж однієї осі та кутового переміщення навколо іншої взаємно перпендикулярної осі;
- полярна циліндрична – характеризується радіус-вектором, що змінюються залежно від лінійних переміщень робочого органа вздовж двох взаємно перпендикулярних осей та кутового переміщення навколо однієї із зазначених осей;
- полярна сферична – характеризується радіус-вектором, що змінюється залежно від лінійного переміщення робочого органа вздовж однієї осі та кутових переміщень навколо двох інших взаємно перпендикулярних осей.

У полярній системі координат робоча зона робота має форму кола, циліндра або сфери.

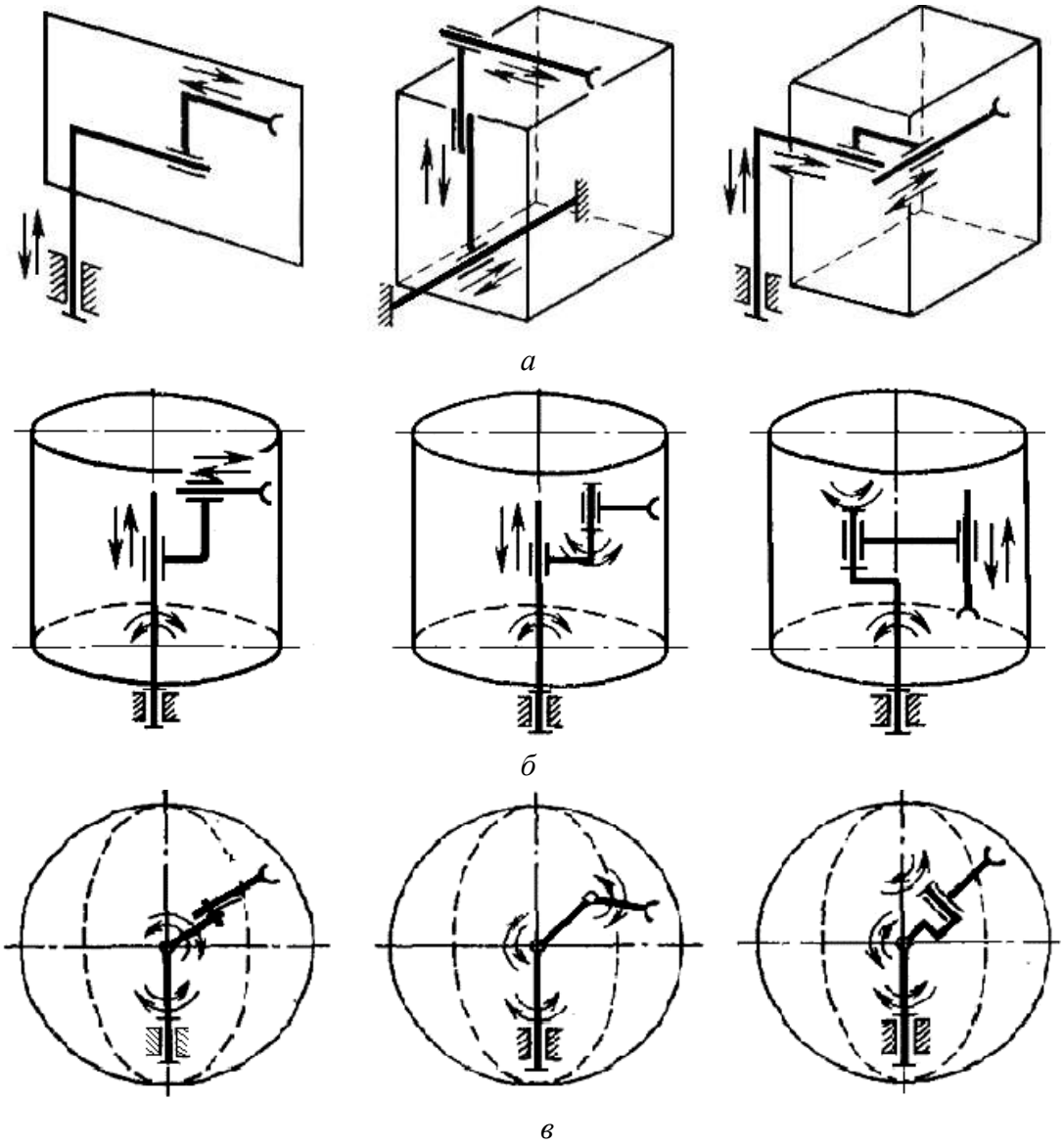


Рис. 1.7. Структурні схеми роботів: *а* – прямокутна система координат; *б* – полярна циліндрична система координат; *в* – полярна сферична (ангулярна) система координат

Окремим різновидом полярної системи координат є *кутова (ангулярна)* система, яка також може бути трьох видів (рис. 1.7, *в*):

- ангулярна плоска – характеризується радіус-вектором, що змінюються залежно від відносних кутових переміщень робота в одній координатній площині;
- ангулярна циліндрична – характеризується радіус-вектором, що змінюється залежно від відносних кутових переміщень робота в одній координатній площині та лінійного переміщення вздовж осі, перпендикулярної до координатної площини;

- ангулярна сферична – характеризується радіус-вектором, що змінюється залежно від відносних кутових переміщень ланок маніпулятора у двох взаємно перпендикулярних площинах.

Промислові і сервісні роботи та програмно керовані технічні системи, які складаються з комплексу роботів та відповідного обладнання, називають *робототехнічними системами*. Вони поділяються на *маніпуляційні*, *транспортні* (*мобільні*) та *інформаційні*.

Маніпуляційні робототехнічні системи призначені для імітації руху руки людини під час переміщення об'єктів у просторі робочої зони за довільною або наперд заданою траєкторією. *Маніпулятори* є складовою частиною більшості промислових роботів (див. рис. 1.1). Тверді тіла, що належать до механічної системи маніпулятора, називають ланками. Рухоме з'єднання двох приєднаних ланок називається кінематичною парою. Кінематичні пари можуть класифікуватися за кількістю ступенів вільності ланок в їхньому відносному русі, а також за кількістю зв'язків, які накладаються парою на відносний рух ланок.

Кількість ступенів рухливості маніпулятора – це сума можливих рухів виконавчого пристрою, без урахування рухів захоплювання. Кількість ступенів рухомості механічної системи, яка рухається у тривимірному просторі, може бути визначена за формулою Сомова–Малишева:

$$W = 6n - 5p_5 - 4p_4 - 3p_3 - 2p_2 - p_1, \qquad (1.1)$$

де n – кількість рухомих ланок кінематичних ланцюгів маніпулятора; $p_1 \ldots p_5$ – кількість кінематичних пар $I \ldots V$ класів.

Якщо розглядувана система рухається лише в одній площині, ступінь її рухомості визначатиметься за формулою Чебишова:

$$W = 3n - 2p_5 - p_4. \qquad (1.2)$$

Зв'язки в кінематичних парах, що обмежують відносний рух ланок, визначають рівняннями та нерівностями, які встановлюють залежності між координатами та їхніми похідними за часом. Такі зв'язки називають кінематичними, або диференціальними, якщо вони встановлюють зв'язок між координатами та похідними за часом. Голономний зв'язок – це зв'язок, рівняння якого залежить лише від координат матеріальних точок і часу. Незалежну параметричну змінну, яка визначає взаємне відносне положення ланок виконавчого механізму маніпулятора, називають *узагальненою координатою* (див. рис. 1.8).

Маневреністю маніпулятора називають кількість його ступенів рухомості за нерухомого захоплювача. Кількісною оцінкою маневреності є кут та коефіцієнт сервісу. *Кутом сервісу* називають тілесний кут, відповідний кожній точці робочого об'єму, всередині якого можна підвести захоплювач до заданої точки простору.

Зоною обслуговування (робочою зоною) робота/маніпулятора називають частину робочого об'єму (або весь об'єм), в якому може розміщуватися виконавча система робота. Зона обслуговування характеризується формою, системою координат маніпулятора та розмірами максимальних переміщень за ступеням рухомості.

За вантажопідйомністю маніпулятори класифікують так: середні (100…2000 Н), важкі (2000…10000 Н) та надважкі (понад 10000 Н). За способом встановлення виокремлюють: встановлені на підлозі, підвісні та вбудовані.

У маніпуляторах застосовують електромеханічний, гідравлічний, пневматичний та комбінований типи привода. Будову промислового маніпулятора відтворено на рис. 1.8.

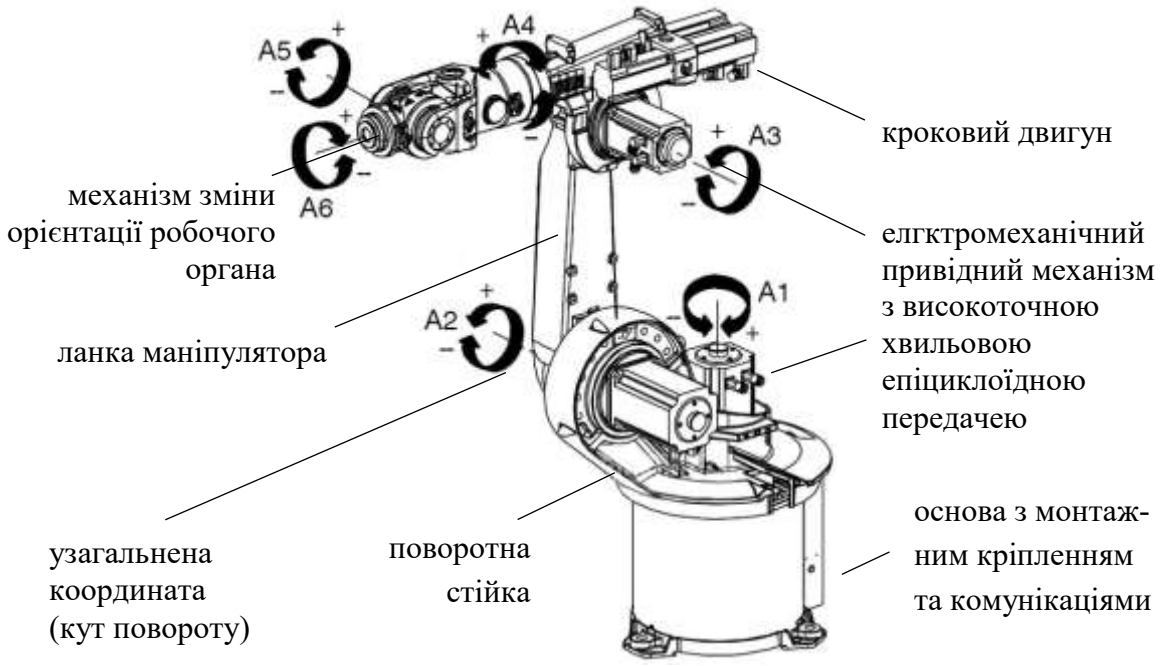


Рис. 1.8. Схема будови маніпулятора

У загальному випадку за функціональною структурою робот або маніпулятор є системою рухомих ланок з незалежними ступенями рухомості, які реалізовано у вигляді механізмів мехатронних модулів

руху, встановлених на його несучій конструкції. Така конструктивна особливість зумовлює те, що більшість систем автоматизованого керування роботів і маніпуляторів складається з двох рівнів: верхнього (інтелектуального та стратегічного) та нижнього (тактичного та виконавчого). Інформаційна система забезпечує обробку інформації як на верхньому, так і на нижньому рівні автоматизації робота.

Інформаційні робототехнічні системи – це комплекси вимірювально-інформаційних систем та засобів керування, які автоматично виконують збір, передачу та обробку інформації (рис. 1.9).

а *б*

Рис. 1.9. Застосування оптичних пристроїв в робототехніці для вимірювання:
а – з оптичною камерою; *б* – з лазерним далекоміром

Одним з різновидів сучасних інформаційних робототехнічних систем є безпілотні системи (літальні, наземні, надводні, підводні апарати).

Безпілотний літальний апарат (БПЛА), або *дрон*, – літальний апарат, який виконує політ, зліт та посадку без фізичної присутності пілота на його борту та може управлятися дистанційно оператором через інфор-маційну систему або автономно виконувати політ за заданою програмою з можливістю автоматичного або ручного коригування польотного завдання. Останніми роками БПЛА часто використовують в географічних дослідженнях, зокрема, у кадастрових та геодезичних сферах відпрацьовано технології лазерного та фотосканування, а також застосовують в потенційно небезпечних роботах з можливостями оперативного об’єднання даних повітряного та наземного лазерного сканування для коридорного картографування і моніторингу ліній електропередачі, трубопроводів, контролю стану об’єктів виробничої інфраструктури, об’єктів життєзабезпечення населення, зйомках

лінійних об’єктів, з метою інвентаризації об’єктів нерухомості, а також для отримання оперативної інформації в надзвичайних ситуаціях. Сучасні БПЛА є різновидом автономних літальних робототехнічних систем з великим різноманіттям конфігурацій, аеродинамічних схем та комплектації. В нинішній час немає будь-яких стандартів щодо класифікації дронів, а виробники не обмежені жорсткими умовами створення та оснащення безпілотних літальних апаратів з боку авіаційних регуляторів, тому логічною є класифікація, в якій БПЛА ранжують за сферами використання або за призначенням.

У цивільному секторі залежно від застосування БПЛА поділяють так:

- у сільському господарстві – для обробки рослин від бур’янів і комах, обробки тварин від гнусу та відстеження їхньої міграції;
- у будівництві – для топографічних зйомок, геодезичних досліджень, землеустрою, контролю за висотним будівництвом;
- у нафтогазовому секторі і секторі безпеки – для контролю цілісності нафтогазопроводів, пошуку витоків та обриву електромереж тощо;
- у рекламних кампаніях – для створення світлових шоу, знімання рекламних роликів, передачі інформації (draw in sky);
- у засобах масової інформації – аерофотозйомка репортажів.

За розмірами БПЛА умовно поділяють на чотири групи:

- мікро (маса до 10 кг, час польоту – до 40 хв, висота польоту – до 1 км);
- міні (маса – до 50 кг, час польоту – до 5 год, висота – до 5-8 км);
- середні (маса – до 1000 кг, час польоту – до 15 год, висота – до 10 км);
- великорозмірні (маса – до 5000 кг, час польоту – до 24 год, висота – до 10 км);
- важкі (маса – понад 5000 кг, час польоту – понад 24 год., висота – до 20 км).

За типом керування БПЛА можуть бути такі: з автоматичним керуванням; з керуванням оператором (дистанційно керовані) у межах видимості через наземну станцію або із застосуванням відеокамери на борту робота (FPV); гібридні.

За принципом польоту БПЛА поділяють так (рис. 1.10): планери (з жорткми або гнучким крилом); з обертальним крилом (гелікоптери, квадрокоптери, мультикоптери); автожири; аеростати; гібридні.

Наземні мобільні робототехнічні системи – це системи, які складаються з пересувних роботів і виконують автоматичні переміщення

робочих об’єктів у просторі. В основному це підіймально-транспортні роботи у вигляді візків, штабелерів та мобільних роботів із запрограмованими маршрутами переміщень. Мобільності наземні роботи досягають завдяки колісним, гусеничним або кроковим системам.

а *б*

в *г*

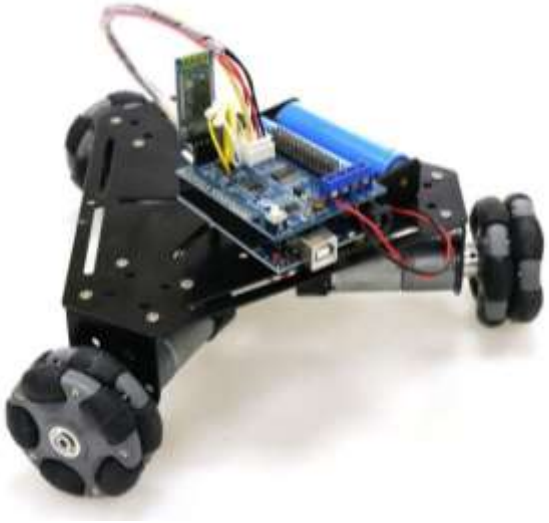
Рис. 1.10. Різновиди безпілотних літальних апаратів за видом аеродинамічної схеми: *а* – одномоторний планер високоплан RQ-7 Shadow (США); *б* – гелікоптер Kawasaki K-RACER IV (Японія); *в* – мультироторний МС-16 Optimal (Україна); *г* – аєростат Teregöz (Турція)

Колісною базою робота називається основна несуча частина конструкції, до якої кріплять колеса та елементи, що надають роботу руху, а також елементи допоміжних конструкцій, електроніки, навісного обладнання та інші периферійні пристрої робота. Основні вимоги до колісною бази – жорсткість та міцність.

Колісні бази роботів можуть бути прямокутної конструкції з двома привідними колесами, чотирма або більше (рис. 1.11, *а*); трикутної схеми з трьома незалежними привідними колесами (рис. 1.11, *б*).

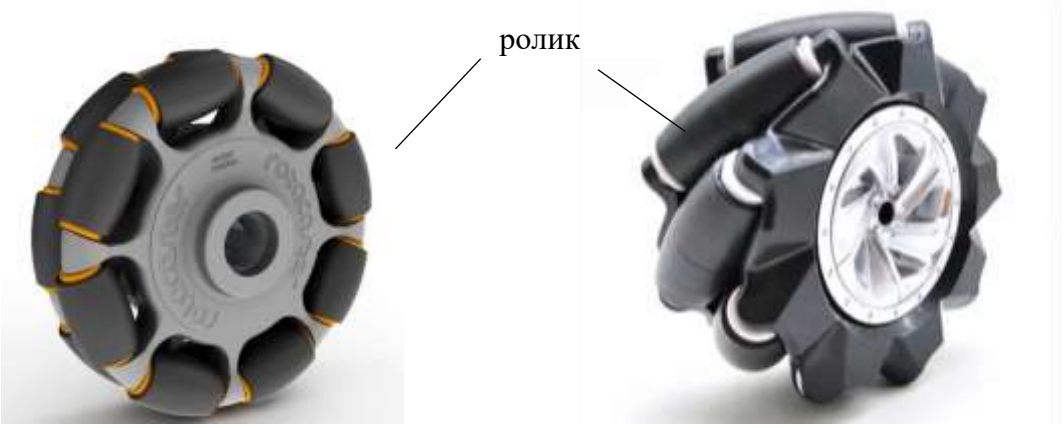
У ролі опорного колеса можуть бути застосовані традиційні пневмоколеса, прогумовані, металеві, омні-колеса (рис. 1.12, *а, б*) та профілювальні пружинні колеса (рис. 1.13, *а, б*).

а



б

Рис. 1.11. Шестиколісний робот-марсохід Curiosity (*а*) з пружними профілювальними колесами та триколісна навчальна роботизоваа платформа (*б*) з омні-колесами



а *б*

Рис. 1.12. Вигляд омні-колеса (*а*) та колеса Ілона (меканум-колесо) (*б*)

Традиційні конструкції пневматичних та пружних коліс з профілювальними поверхнями мають просту конструкцію, добру

здатність до демпфування ударів опорної поверхні, на якій є нерівності, проте вони менш маневрові. Омні-колеса більш маневрові, проте не стійкі до нерівностей і підходять більше для рівних плоских поверхонь.

а *б*

Рис. 1.13. Вигляд пружинних профілювальних коліс: *а* – ланцюгове; *б* – листове

Розглянуті контрукції роботів засвідчують, що будь-який робот архітектурно складається з трьох основних систем: механічної, інформаційної та системи керування. Функціональна схема будь-якого робота представлена у вигляді взаємопов'язаних базових структурних блоків (рис. 1.14). При цьому на інформаційно-вимірювальну систему та систему управління робота може мати вплив зовнішнє середовище, яке формує випадкові сигнали, тому досить часто зовнішнє середовище також вводять в структуру робота як окремий функціональний елемент.

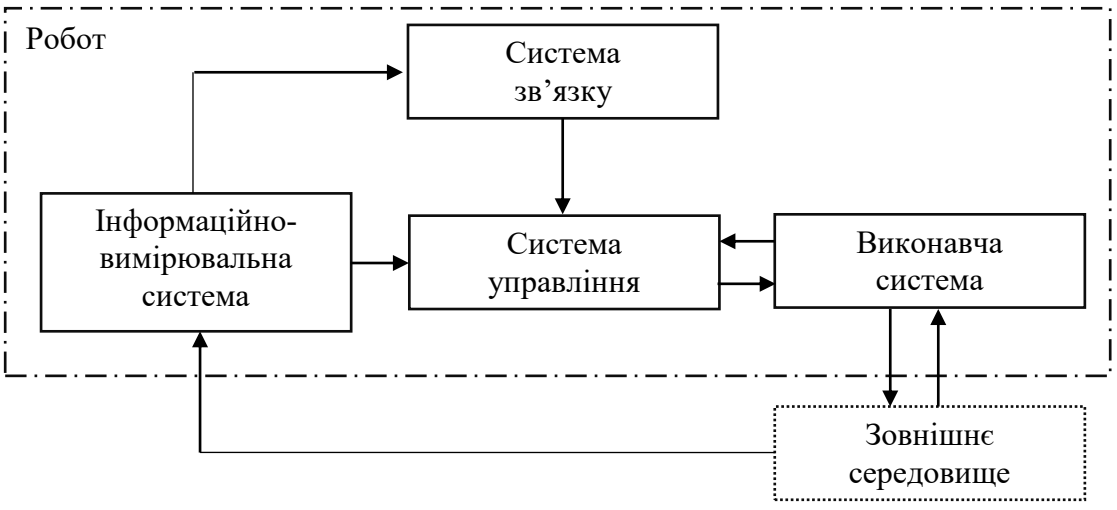


Рис. 1.14. Функціональна схема робота

Механічна система робота, яка забезпечує фактичне опрацювання програми керування за всіма ступенями рухомості, складається з виконавчої системи, приводу та робочого органа. Виконавча система з приводом (*effectors*) виконує всі рушійні функції робота, за допомогою яких відбуваються переміщення за ступенями рухомості. Відносні рухи механічної системи робота поділяються на *орієнтувальні* (локальні), *транспортні* (регіональні) та *координатні* (глобальні). Орієнтувальні рухи надають робочому органу робота орієнтацію в заданій точці робочої зони. Транспортні рухи призначені для переміщення робочого органа в межах робочого простору. Координатні рухи потрібні для переміщення робота між окремими виробничими позиціями. До основних рухів належать всі рухи механічної системи, крім рухів захоплювання, орієнтувальних і додаткових переміщень.

Інформаційно-вимірювальна система робота («сенсорика») – це система взаємопоєднаних штучних органів відчуття (сенсори або датчики), які призначені для сприйняття та перетворення інформації про стан зовнішнього середовища і самого робота (рис. 1.15).

Рис. 1.15. Класифікація інформаційних систем

Система керування (*controllers*) робота призначена для створення законів управління приводами (двигунами) механізмів виконавчої системи за допомогою сигналів зворотного зв'язку від інформаційно-вимірювальної системи та комунікації з оператором (рис. 1.16).

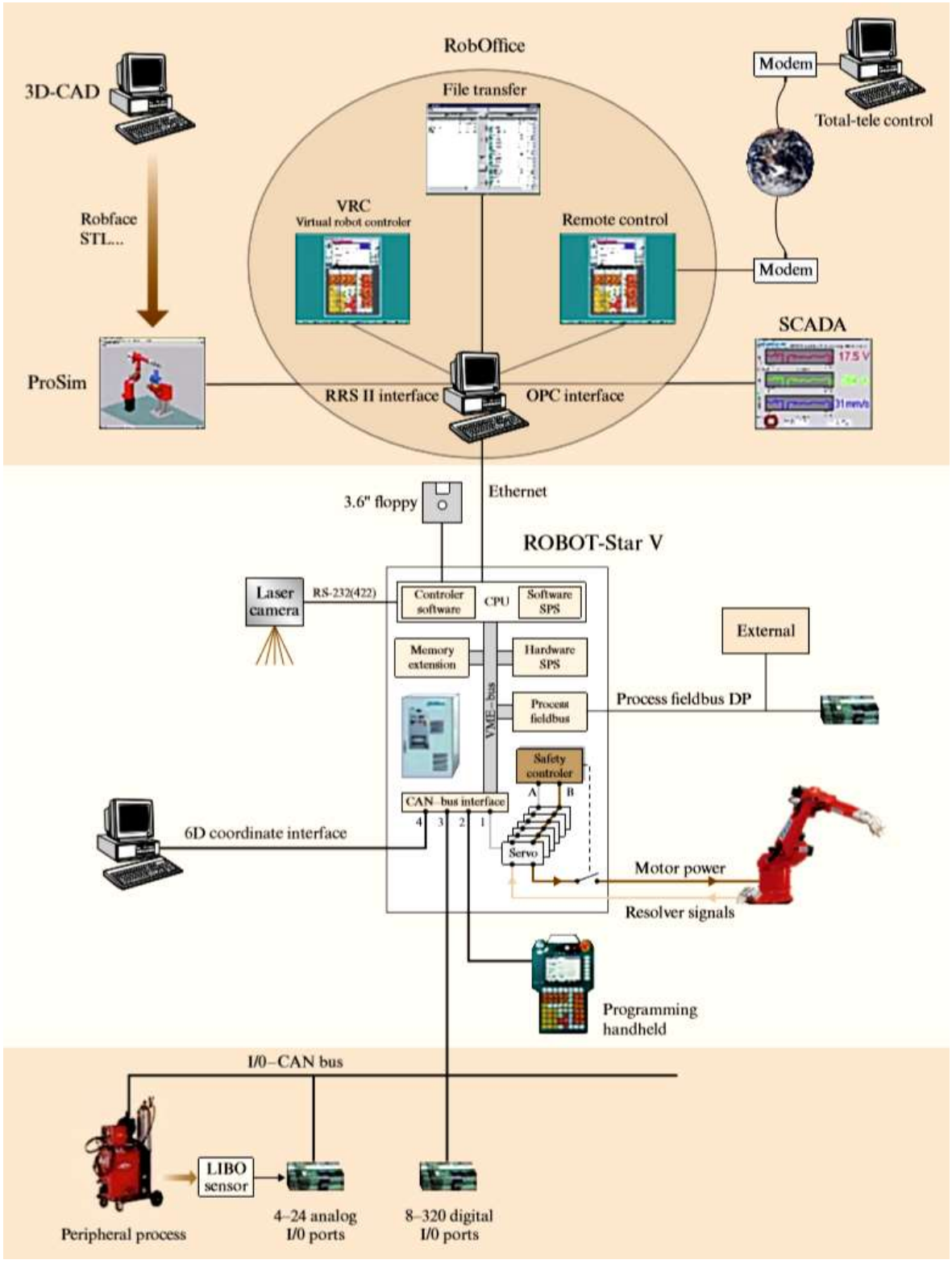


Рис. 1.16. Багаторівнева схема системи керування промисловим роботом

Програмування робототехнічних систем часто виконують, використовуючи математичні моделі роботів, – абстрактне математичне представлення робота або його підсистем дає змогу аналізувати та прогнозувати поведінку робота, описувати фізичні характеристики, динаміку і кінематику робота у вигляді математичних рівнянь, графів, моделей стану тощо. Математична модель може бути побудована на основі різних підходів, зорема таких, як механічні, електричні, кінематичні, динамічні та інші аспекти робота. Вона може описувати різні рівні абстракції: від простих лінійних систем до складних нестабільних динамічних систем.

У робототехніці розрізняють кінематичні і динамічні моделі.

Кінематична модель робота описує його рух поза силами, що діють на нього, та визначає геометричні співвідношення між різними елементами робота, це дає змогу передбачати його рух та позиції в просторі. Кінематичні моделі часто використовують для планування рухів, навігації, обчислення обертових кутів та переміщень.

Динамічна модель робота описує рух робота з огляду на сили, які діють на нього, враховуючи масу та інерцію його частин, що дає змогу аналізувати поведінку робота під впливом зовнішніх сил, таких як гравітація, та визначає реакцію робота на внутрішні сили, що виникають внаслідок руху. Динамічні моделі використовують для проєктування контролерів, аналізу стійкості, оптимізації енергоефективності та відображення взаємодії з навколишнім середовищем.

Контрольні запитання

1. Що таке робот та які види промислових роботів вам відомі?
2. Які функції виконують сервісні роботи?
3. Які види керування застосовують у роботах?
4. У чому полягає призначення механічної системи робота?
5. Що таке кількість ступенів рухливості маніпулятора?
6. Наведіть класифікацію інформаційних систем.
7. Назвіть основні компоненти (підсистеми) робота.

Лекція № 2. Теорія безпілотних літальних апаратів (БПЛА)

Теоретичні основи польотів літальних апаратів з крилом, важчих за повітря, були розроблені Є. Жуковським на початку XX ст., який визначив основні принципи польоту на відомих законах Ньютона та Бернулі, а також доповнив власними уявленнями про цей процес. Для горизонтального потоку повітря, що обтікає профіль крила, справедливим є рівняння Бернуллі

$$\frac{\rho\upsilon^2}{2} + p = const, \qquad (2.1)$$

де ρ – щільність повітря, p – тиск, v – швидкість повітря, що обтікає профіль крила.

З формули (2.1) випливає: що більша швидкість повітря, то менший його тиск, і навпаки, що менша швидкість повітря, то більший тиск. Профіль крила зазвичай має не симетричний вигляд (показано на рис. 2.1): верхня частина крила більш опукла, ніж нижня. Через це повітря, яке обтікає, НАД верньою поверхнею крила за однаковий проміжок часу рухатиметься швидше, ніж ПІД нижньою повернею. Оскільки шлях, пройдений повітрям по верньому профілю крила є більшим, ніж шлях, пройдений по нижньому профілю, тиск повітря на крило зверху, відповідно до рівняння Бернуллі, буде меншим, ніж тиск знизу. Різниця тиску над крилом і під крилом дає результаційний тиск, який і створює підіймальну силу, яка врівноважує в польоті силу тяжіння.

Рис. 2.1. Профіль крила

Інший «підіймальний ефект» виникає завдяки тому, що крило розмішують під кутом до зустрічного потоку повітря, тож згідно з законами Ньютона повітря, взаємодіючи з поверхнею крила, починає його відхиляти, тобто утворюється сила протидії (рис. 2.2). Загальний результат полягає в тому, що сила протидії та підіймальна сила утворюються у відповідь на зміну напрямку руху повітря, тобто крило породжує силу, спрямовану вниз, а повітря – силу, спрямовану вгору. Кут, під яким розміщується профіль крила (лінія хорди крила) в потоці повітря, називається *кутом атаки*.

Унаслідок обтікання крила повітрям напрямок руху повітря відхиляється від початкового. Є. Жуковський показав, що криловий профіль можна замінити еквівалентним вихором або циліндром, що обертається. Напрямок обертання вихору (циліндра) такий, що нижня половина рухається назустріч потоку, а верхня – вздовж потоку. Цей ефект називають «ефектом Магнуса».

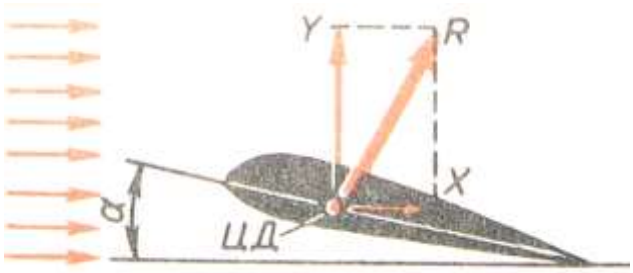


Рис. 2.2. Вигляд крила збоку: *ЦД* – центр тиску, α – кут атаки крила; R – сила тиску на крило з боку зустрічного потоку повітря; X – сила тертя повітря або лобового опору

Підіймальну силу на крилі можна визначити за допомогою швидкості циркуляції потоку навколо крила $\Delta\upsilon$. Швидкість повітря НАД крилом буде більшою за швидкість набігання повітря на величину швидкості циркуляції потоку, а швидкість ПІД крилом буде меншою за швидкість набігання потоку на таку саму величину. Таким чином, для польоту БПЛА з жорстким крилом потрібно створити набігання потоку повітря на нерухоме крило для утворення підіймальної сили на крилі, яку визначають за формулою

$$P = C_{Ra}\frac{\rho\upsilon^2}{2}S, \qquad (2.2)$$

де C_{Ra} – безрозмірний коефіцієнт; $\frac{\rho\upsilon^2}{2}$ – швидкісний напір; S – площа крила.

У горизонтальному прямолінійному польоті збільшення швидкості і кута атаки призводить до збільшення підіймальної сили, створюваної крилом. Збільшення кута атаки супроводжується зростанням індуктивного опору, тому спроба далі виконувати набір висоти завдяки збільшенню кута атаки без збільшення тяги двигунів може призвести до зриву потоку повітря з поверхні крила, внаслідок чого різко зменшується підіймальна сила крила і відбуватиметься явище звалювання.

Для більшості відомих профілів критичний кут атаки крила становить15°–18°. Для військових реактивних БПЛА цей кут може бути збільшений до 20°–22°.

Мінімально допустима швидкість польоту БПЛА з жорстким крилом, на якій може відбуватися політ без звалювання:

$$\upsilon_{\min} = \sqrt{\frac{2mg}{\rho S C_{L\max}}}, \qquad (2.3)$$

де m – маса літака, кг; S – площа крила; ρ – щільність повітря, кг/м^3; $C_{L\max}$ – допустимий коефіцієнт підіймальної сили, відповідний куту атаки профілю крила (рис. 2.3).

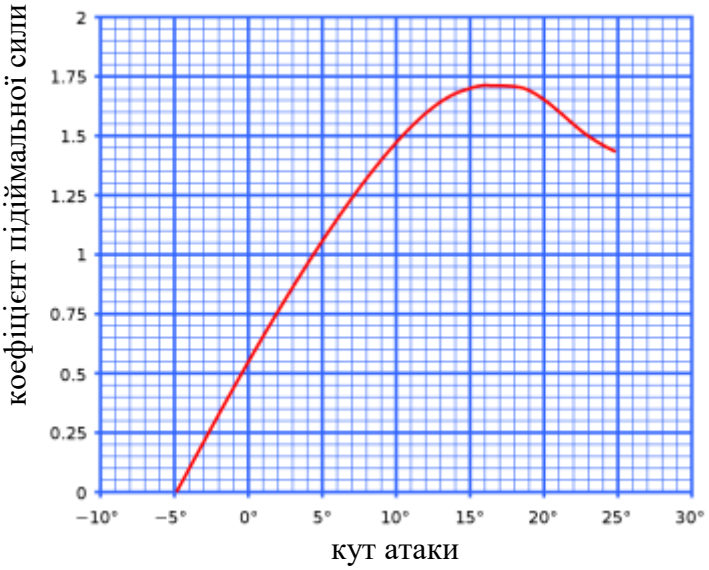


Рис. 2.3. Значення коефіцієнта $C_{L\max}$ за різних значень кута атаки для стандартного профілю крила компанії Boeing

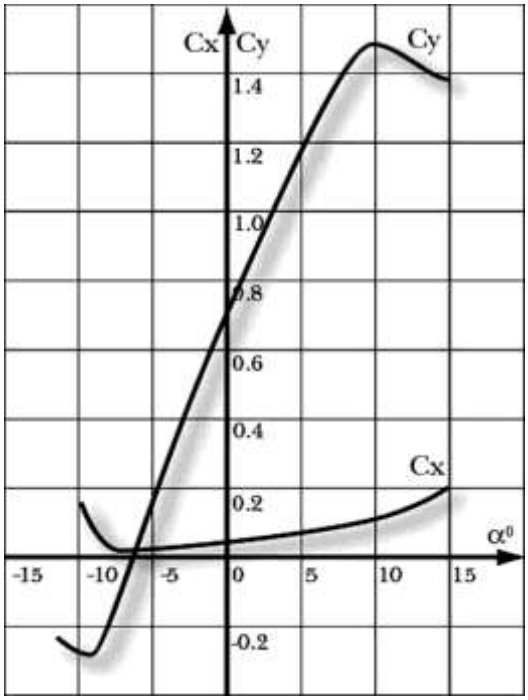


Рис. 2.4. Значення коефіцієнтів $C_y(\alpha)$ та $C_x(\alpha)$ для літального апарату компанії Airbus

Аеродинамічна якість літального апарата залежить від відношення підіймальної сили до сили лобового опору (або відношення їхніх коефіцієнтів) у потоковій системі координат за заданого кута атаки. Аеродинамічна якість визначається співвідношенням

$$K(\alpha) = \frac{C_y(\alpha)}{C_x(\alpha)}, \qquad (2.4)$$

де α – кут атаки; $C_y(\alpha)$ – коефіцієнт підіймальної сили; $C_x(\alpha)$ – коефіцієнт лобового опору (рис. 2.4).

У простішому поданні аеродинамічну якість можна

розцінювати як відстань, яку здатен пролетіти літальний апарат з жортским крилом з деякої висоти в штиль і вимкненим двигуном (якщо він взагалі є). Наприклад, якщо для планера якість $K(\alpha) = 30$, а для дельтаплана $K(\alpha) = 10$, тоді з висоти 1 км спортивний планер зможе пролетіти в ідеальних умовах приблизно 30 км, а дельтаплан – 10 км.

Є велика кількість БПЛА планерного типу з жорстким крилом, конструкція і компонування яких часто подібні до конструкції літаків. Виготовляють їх з композиційних матеріалів та алюмінієвих сплавів. У носовій частині може розміщуватися радіолокаційне й оптичне обладнання та апаратура зв'язку. В ролі двигунів можуть бути застосовані тягові, штовхальні гвинти, імпелери та реактивні двигуни. Такі БПЛА за масштабом завдань поділяються на тактичні, тобто дальність їхнього польоту не перевищує 80 км, оперативно-тактичні – до 300 км, оперативно-стратегічні – до 1700 км. Паливна система БПЛА бензинових та керосинових двигунів поділяє їх на такі види: монозаправні – одноразова заправка паливної системи, яку виконує в промислових умовах виробник на заводі; полізаправні – багаторазова заправка, яка може бути наземною – виконується на землі, платформна – морська (на борту морського судна) та бортова (на борту пілотованого літального апарата). Додаткові паливні баки дають змогу збільшувати дальність до 2 000 км та тривалість польоту до 24 год. Якщо в БПЛА застосовують електродвигуни, тоді їхнє живлення виконують від акумуляторів з багаторазовим заряджанням.

Планерна схема з жорстким крилом зумовлює більшу тривалість та дальність польоту БПЛА (понад 12 год), проте за типом аеродинамічної схеми виокремлюють БПЛА гелікоптерного типу, які мають більшу маневреність.

Основним елементом, який приводить в рух БПЛА гелікоптерного типу, є *пропелер* – лопатевий агрегат, що приводиться в обертання двигуном і призначений для перетворення потужності (крутного моменту) двигуна в тягу. Обертаючись, лопатевий апарт переміщує масу повітря. Якщо на одній з лопатей виділити дві невеликі ділянки: одну ближче до осі обертання, другу – далі, на кінці лопаті, тоді в процесі обертання лопаті обидві ділянки описуватимуть концентричні кола, але та частина лопаті, що на більшому радіусі кола, матиме більшу швидкість відносно повітря, ніж ділянка лопаті, що на колі меншого радіуса. На осі обертання швидкість дорівнюватиме нулю, а на кінці

лопаті вона буде найбільшою. Поперечний переріз лопаті має вигляд обтічного профілю, внаслідок обтікання якого потоком повітря під кутом атаки виникають підіймальна сила та сила опору. Розділяючи лопать на безліч дрібних ділянок, можна визначити їхні підіймальні сили та сили опору і, склавши разом відповідні сили на всіх ділянках, визначити підіймальну силу та силу лобового опору однієї лопаті. Підіймальна сила (або сила тяги) всього пропелера (гвинта) дорівнюватиме добутку підіймальної сили однієї лопаті на кількість лопатей.

Є різні види повітряних гвинтів, які залежать від аеродинамічних профілів, кількості лопатей та діаметра. У виборі гвинта беруть до уваги такі параметри:

- діаметр пропелера. Пропелери більшого діаметра потребують більшої потужності двигуна на свою розкрутку. Потрібно переконатися, що двигун може розвивати достатню потужність. Крім того, великі і важкі пропелери мають більшу інерцію, тому вони не здатні миттєво прискорюватися, що позначиться на маневреності коптера;
- крок пропелера (prop pitch). Вказують другою цифрою, після позначки «x», у маркуванні пропелера. Також може вказуватися просто, як третя і четверта цифри марки, наприклад, 1260 – це пропелери з кроком 6 дюймів. Фізично це величина стовпа повітря, який пропелер пересуває вниз за один свій оберт. Чим більший крок, тим вища підіймальна сила. Наприклад, пропелери 14×7 мають більшу підіймальну силу, ніж 14×5. Для ідеального випадку крок пропелера, помножений на кількість обертів за секунду, дорівнює швидкості повітряного потоку від гвинта;
- кількість лопатей. Класично – це дві лопаті, однак пропелери з трьома лопатями мають велику підіймальну силу – приблизно еквівалентну на один дюйм більшого діаметра та на один дюйм більшого кроку дволопатевого пропелера;
- пропелерна константа (prop const) – впливає на підіймальну силу і на потрібну для розкручування пропелера потужність мотора, оскільки фізично ця константа означає величину втрат на опір повітря під час обертання пропелера: що тонший матеріал, з якого виготовлено пропелер, то менша ця константа і менше потрібно потужності, що розвивається на моторі для розкручування такого пропелера.

Аеродинамічна сила тяги гвинта є рівнодійною сил тяги всіх елементів лопаті гвинта:

$$F = \sum f_i . \tag{2.5}$$

Сила тяги, що прикладається до центра обертання гвинта, спрямована вздовж його осі обертання і може наближено бути визначена через параметри гвинта та частоту його обертання:

$$F = \alpha \rho n^2 D^4 , \tag{2.6}$$

де α – коефіцієнт тяги, який залежить від форми профілю лопатей, кута атаки; ρ – щільність повітря; n – частота обертання гвинта; D – діаметр гвинта.

Коефіцієнт тяги гвинта залежить від параметрів профілю гвинта і визначається екпериментально або через співвідношення ККД гвинта:

$$\eta = \frac{\alpha}{\beta} \frac{V}{nD}, \tag{2.7}$$

де $\beta = 7{,}5 \pm 1$ – коефіцієнт потужності гвинта, який визначають емпіричним шляхом під час продування гвинта в аеродинамічній трубі; V – повздовжня швидкість переміщення центра обертання гвинта (швидкість польоту).

Для типового гвинта зміна його ККД може бути оцінена графічно за рис. 2.5.

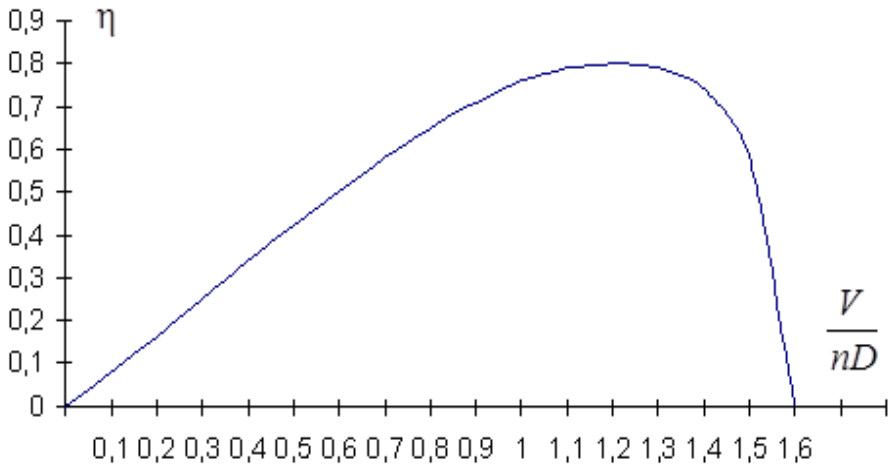


Рис. 2.5. Залежність ККД гвинта від його відносного переміщення

Потужність на роторі гвинта приблизно дорівнюватиме:

$$N = \frac{1}{D}\sqrt{\frac{F^3}{\beta^3}}. \qquad (2.8)$$

Сила тяги та потужність на гвинті розвиваються до деякої граничної швидкості обертання гвинта, відтак тяга та потужність починають зменшуватися, що пояснюється виникненням хвильових процесів на гвинті.

Під час обертання лопатевого механізму в повітрі на привідному механізмі виникає ефект зворотного обертання (*реактивний момент*), який створюється силами протидії з боку зовнішнього середовища (повітря). Гвинт, який швидко обертається, також має *гіроскопічний момент* (ефект дзиги), який полягає в намаганні зберегти положення своєї осі обертання в просторі. Якщо примусово нахилити вісь обертання гвинта в будь-якій бік, тоді така система протидіятиме цьому відхиленню та намагатиметься створити зворотне відхилення в напрямку, перпендикулярному до зовнішнього збурення. Для зменшення явища реактивного моменту та частково гіроскопічного моменту застосовують попарне встановлення семетрично обертальних гвинтів.

Мультикоптер – це літальний апарат з довільною кількістю несучих гвинтів, розміщених в одній площині, що обертаються діагонально в протилежних напрямках (рис. 2.6).

Квадрокоптер є різновидом мультикоптера з чотирма несучими гвинтами (рис. 2.7).

Мультикоптер маневрує шляхом нерівномірної зміни швидкості обертання гвинтів. Наприклад, якщо збільшити рвномірно й одночасно оберти всіх гвинтів, відбудеться вертикальне підіймання коптера, та якщо збільшити оберти гвинтів з одного боку коптера і не змінювати їх з іншого – відбудеться рух у бік, де оберти гвинтів будуть меншими, а якщо збільшити оберти гвинтів, які обертаються спільно в один бік та зменшити оберти гвинтів, які спільно обертаються в іншій бік – відбудеться поворот коптера (рис. 2.8).

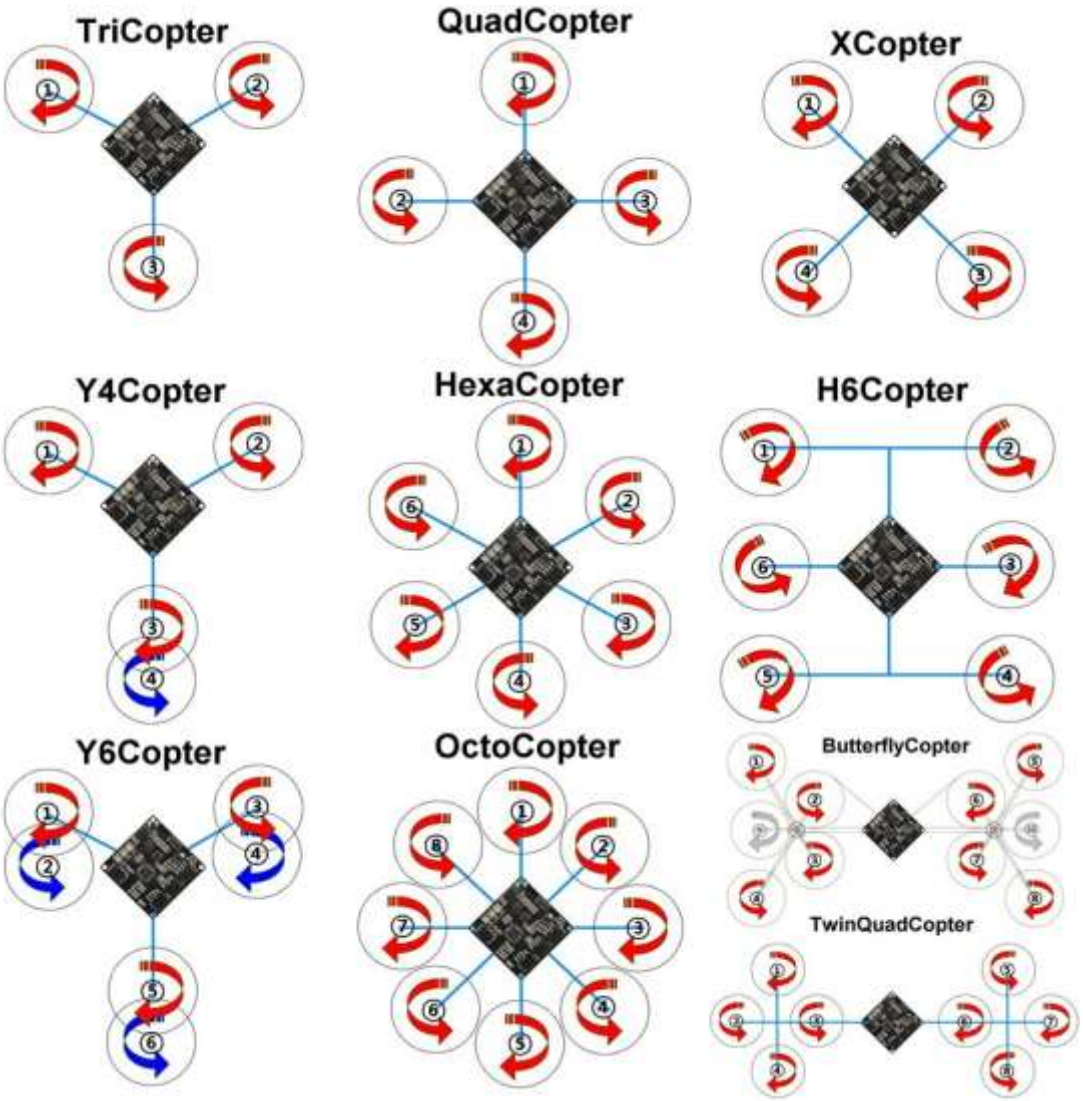


Рис. 2.6. Конфігурації мультикоптерів та напрям обертання їхніх гвинтів

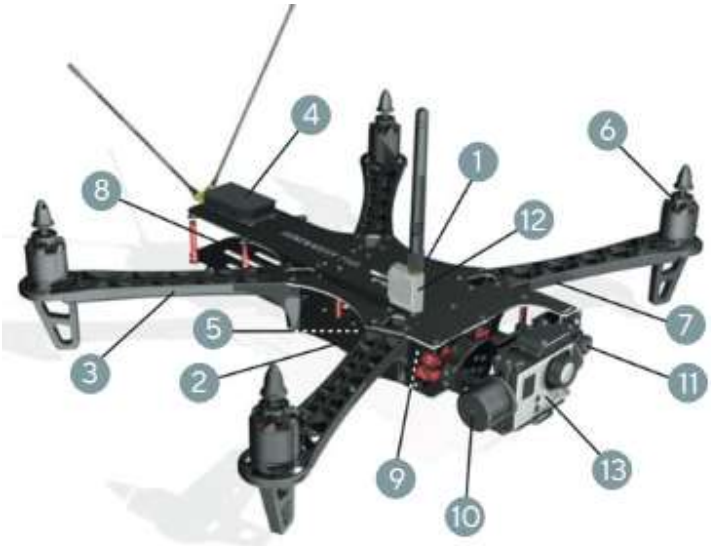


Рис. 2.7. Будова квадрокоптера з електроприводом:
1 – верхня пластина;
2 – нижня пластина;
3 – ланка рами;
4 – приймач радіочастот;
5 – політний контролер;
6 – двигун;
7 – регулятор швидкості;
8 – акумулятор;
9 – контролер підвісу камери;
10 – безколекторний двигун управління підвісу;
11 – FPV камера;
12 – відеопередавач;
13 – відеокамера

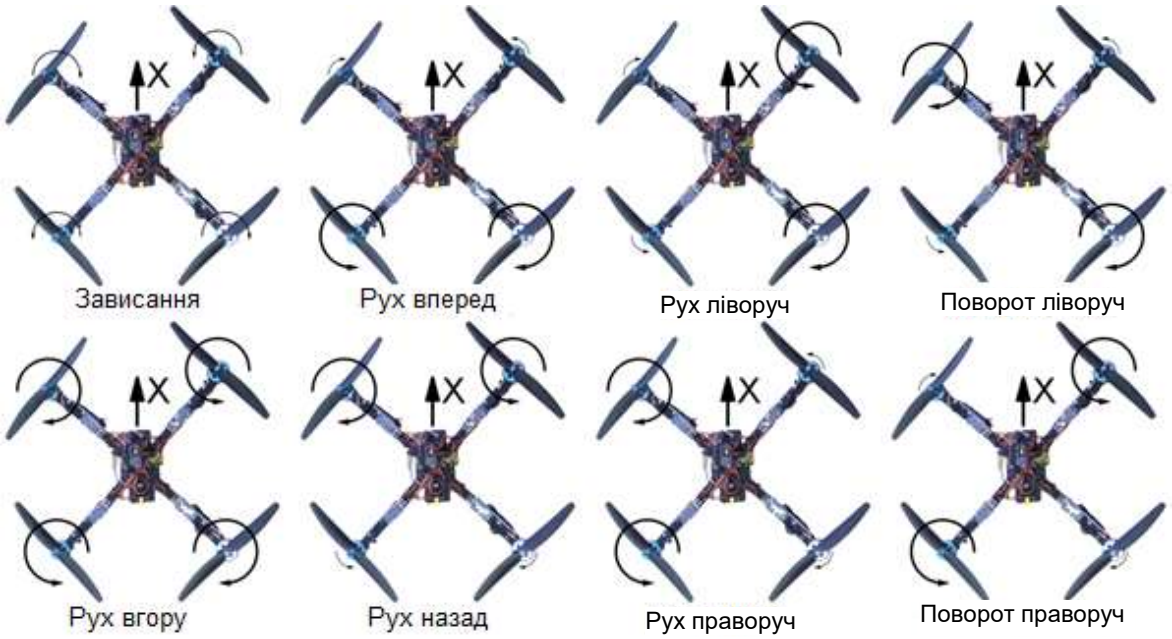


Рис. 2.8. Схеми руху квадрокоптера залежно від правила обертання гвинтів

Квадрокоптери розрізняються розміщенням роторів відносно повздовжньої осі (див. рис. 2.6), а за кількістю використань поділяються на одноразові, в яких не передбачена система посадки, та багаторазові, які використовують багато разів.

У разі нерівномірної зміни обертів двигунів коптера змінюватиметься вектор сили тяги та відбуватиметься відхилення коптера від горизонтального положення у бік рузультаційного вектора (рис. 2.9), а за нерівномірної зміни обертів всіх моторів коптер почне хаотичну зміну напрямку польоту. Очевидно, що за нахилу рами коптера внаслідок появи горизонтальної складової вектора тяги F_x зменшиться вертикальна складова F_y і квадрокоптер почне втрачати висоту. Для підтримання висоти під час нахиляння коптера потрібно збільшити оберти всіх двигунів коптера на деяку компенсаційну величину. Також для повороту коптера за збільшення обертів двигунів, що спільно обертають гвинти в один бік, слід пропорційно зменшити оберти тих

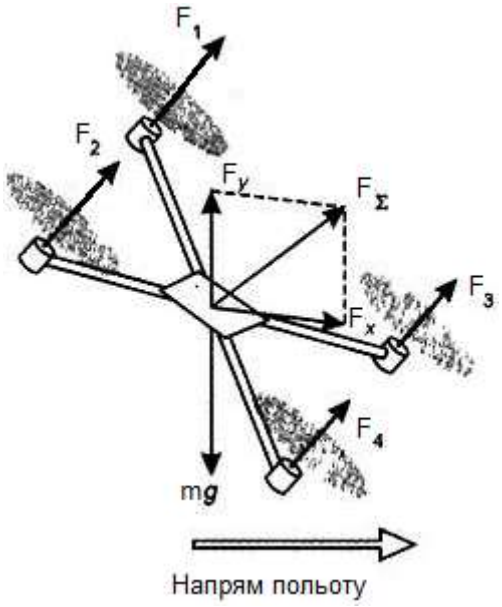


Рис. 2.9. Векторна схема розподілу векторів тяги коптера

двигунів, які разом обертають гвинти в інший бік, тим сами забезпечивши горизонтальний режим повороту.

Для БПЛА є три критичних параметри, якими визначають стабільність польоту та керованість літального апарата. Ці параметри задають в трьох вимірах у вигляді кутів повороту навколо центра мас, називають їх кутами крену, тангажу і рискання (рис. 2.10).

Рис. 2.10. Кути напряму польоту БПЛА

Кут тангажа – кут між поздовжньою віссю літального апарата і горизонтальною площиною. В авіації розрізняють тангаж із збільшенням кута – кабрування, та зі зменшенням кута – пікування.

Кут крену – це кут між поперечною віссю літального апарата та горизонтом. *Кут рискання* – це кут повороту корпуса літального апарата в горизонтальний площині.

Будову БПЛА з жорстким крилом наведено на рис. 2.11.

Рис. 2.11. Будова та основні характеристики БПЛА PD 1900: крейсерська швидкість – 50 км/год; розмах крила – 190 см; тривалість польоту – 100 хв; опір вітру – до 27 км/год; дальність трансляції сигналу – до 40 км

Аеродинамічним фокусом крила БПЛА називають точку прикладання нарощення аеродинамічної підіймальної сили. *Центр мас* – це точка, відносно якої всі сили ваги БПЛА утворюють зрівноважений

момент, тобто сумарний момент сил ваги повинен дорівнювати нулю. Якщо центр мас знаходиться перед аеродинамічним фокусом на осі польоту, тоді БПЛА в польоті буде самовільно нахилятися вперед, для компенсації такого відхилення потрібна додаткова стабілізація за допомогою хвостової частини, що створює додаткове відхилення керма висоти вгору. У випадку, коли центр мас знаходиться за аеродинамічним фокусом на осі польоту, виникатиме явище довільного підіймання носової частини БПЛА, що також компенсується хвостовою частиною внаслідок відхилення керма висоти вниз. Окрім того, бажано, щоб центр мас був на повздовжній осі БПЛА, оскільки його бокове зміщення призводитиме до зміни кута крену літального аппарата. *Центр тиску крила* (аеродинамічний центр) – це точка перетину лінії дії аеродинамічної сили з площиною хорд крила. В загальному випадку положення центру тиску зміщується до передньої кромки крила внаслідок збільшення кута атаки. Розроблено форму профілю з постійним центром тиску, для якого положення центра тиску у разі зміни кута атаки залишається незмінним. Для прямокутного крила центр тиску знаходиться на відстані приблизно 1/4 довжини хорди від передньої твірної крила. Під час руху з надзвуковою швидкістю центр тиску зміщується в бік задньої твірної крила до 1/2 довжини хорди.

Для ефективного керування польотом БПЛА важливо знати центр його тяжіння, який для різних типів БПЛА визначається по-різному. Для БПЛА з аеродинамічною схемою планера схема центрування показана на рис. 2.12.

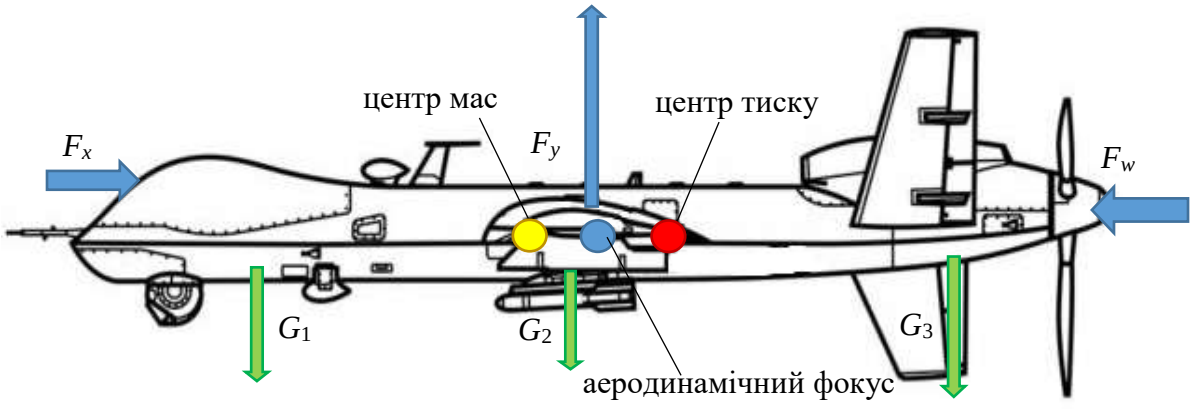


Рис. 2.12. Схема центрування БПЛА з жорстким крилом: F_y – аеродинамічна підіймальна сила; F_x – сила аеродинамічного опору; F_w – сила тяги; G_1 – сила ваги обладнання; G_2 – сила ваги корпуса; G_3 – сила ваги силової установки

Типовий БПЛА виготовляють з легких композитних матеріалів, що забезпечує підвищену маневреність, міцність, малу вагу, зменшення шуму під час польоту. Композиційні матеріали практично на змінюють своїх характеристик внаслідок зміни параметрів навколишнього середовища, це дає змогу застосовувати їх на великих висотах і з великими навантаженнями в будь-яку погоду.

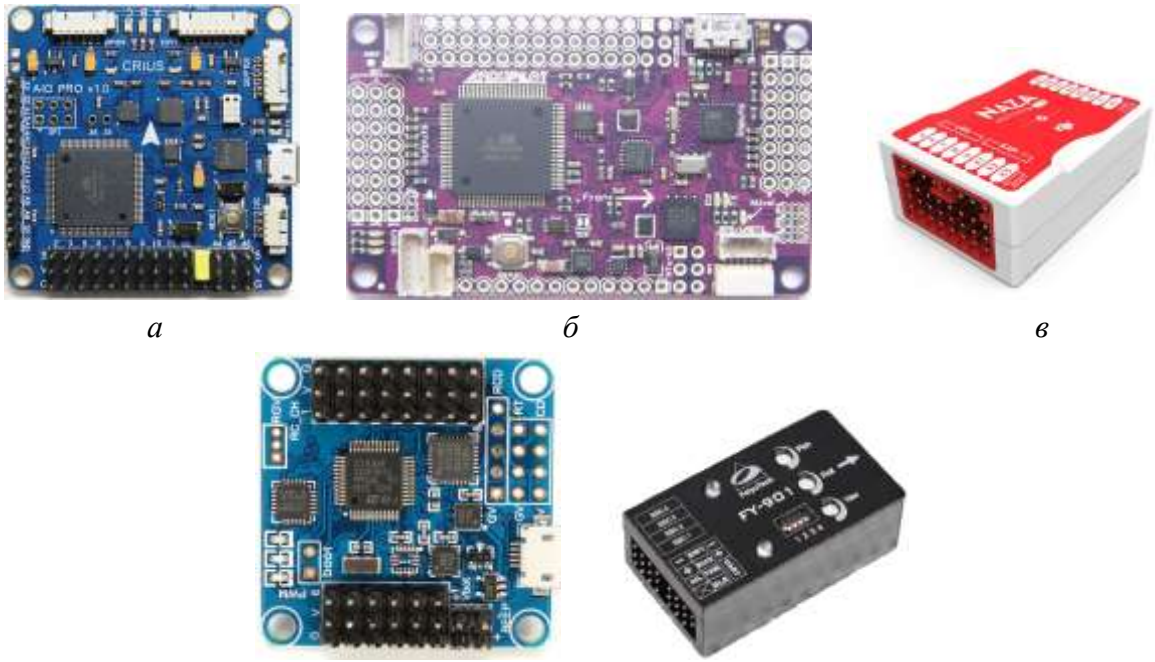
Електронна система управління будь-якого БПЛА складається з обчислювальної системи і сенсорів, що містять

- процесор та/або мікроконтролер з модулями оперативної та енергонезалежної пам'яті;
- модуль визначення просторового положення, що складається з багатоосьових MEMS-сенсорів, таких як гіроскопи, акселерометри, магнетометри;
- модуль аналогових або цифрових барометричних датчиків для визначення висоти та повітряної швидкості;
- модуль управління двигунами та енергопостачанням;
- модуль управління сервоприводами для управління польотом та режимами двигунів;
- модуль прийому супутникової навігації GPS для точного геопозиціонування;
- модуль радіозв'язку для ручного управління і передавання даних телеметрії.

Базова система управління БПЛА може бути доповнена радіолокаційними системами, лідарами, ультразвуковими датчиками відстаней, системами стабілізації фото- і відеообладнання.

Бортовий контролер БПЛА (рис. 2.13), призначений для стабілізації польоту за поривів вітру та неоднорідності повітря, дає змогу керувати польотом шляхом регулювання швидкості обертання двигунів, завдяки цьому досягають зміни кутів нахилу літального апартата (тангажу, крену та рискання).

Політний бортовий контролер БПЛА оснащено набором мініатюрних інтегральних сенсорів, які безперервно відстежують положення рами БПЛА у просторі, кутові прискорення, атмосферний тиск і напрямок силових ліній магнітного поля.



а *б* *в*

г *д*

Рис. 2.13. Види цивільних політних контролерів БПЛА: *а* – Crius AIOP; *б* – ArduPilot Mega2.5 ArduFlyer; *в* – NAZA; *г* – Multiwii; *д* – FY-901

Завдання політного контролера, окрім автоматичного регулювання сигналів управління, полягає в передаванні команд від пульта керування до електронної силової системи управління, яка перетворюватиме сигнали управління (зазвичай цифрові) в аналогові величини. Якщо на БПЛА застосовують електричні двигуни, тоді для їхнього ввімнення і регулювання обертів застосовують регулятори обертів (ESC) (рис. 2.14) або драйвери двигунів.

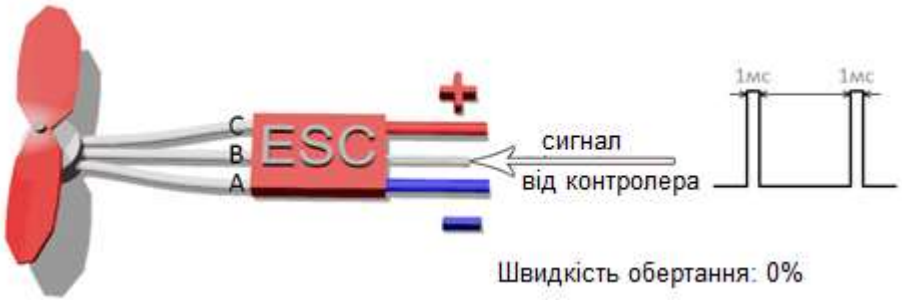


Рис. 2.14. Схема під’єднання ESC до електродвигуна

Політні контролери на базі мікроконтролера або мікрокомп’ютера здатні формувати ШІМ-імпульси (PWM) для ESC залежно від отриманої команди або програми. Наприклад, для того щоби подати команду регулятору обертів на ввімкнення максимальної швидкості обертів двигуна, контролер формує імпульси тривалістю 2 мс із затримкою

10-20 мс. В стандартних цивільних ESC тривалість імпульсу в 1 мс відповідає режиму зупинки двигуна, 1,1 мс – 10% від максимальної швидкості, 1,2 мс – 20% і т.д.

ESC – це регулятор обертів електродвигуна, який здатен комутувати двофазний або трифазний струм з високою частотою перемикання, яку може змінювати залежно від частоти поданого сигналу управління. На рис. 2.15, *а* показано типову будову регулятора обертів квадрокоптера. Аналогом регулятора обертів двигунів квадрокоптера у промислових електродвигунів великої потужності є частотні регулятори (рис. 2.15, *б*), які також застосовують в ситемах управління промислових роботів.

Кожен окремий двигун БПЛА може мати індивідуальний регулятор обертів (для коптерів це є обовязковим правилом для їхнього польоту), а всі регулятори обертів під'єднують до політного контролера. Живляться регулятори безпосередньо від акумуляторів, а послідовність під'єднання ліній зв'язку регулятора до електродвигуна визначатиме напрям обертів двигуна.

а *б*

Рис. 2.15. Регулятори обертів: *а* – малопотужний регулятор обертів електродвигуна квадрокоптера; *б* – промисловий частотний ругулятор (перетворювач частоти)

Керування БПЛА відбувається за допомогою пульта керування, що передає команди радіоприймачу, від якого команди надходять на політний контролер. Політний котролер управляє обертами двигунів коптера в режимі реального часу за допомогою спеціальної обчислювальної системи швидкодійного мікроконтролера. Політний контролер постійно зчитує дані з вбудованих приладів – гіроскопа,

акселерометрів, барометра, приймача радіоуправління та на їхній основі розраховує керівні сигнали для кожного двигуна окремо.

Контрольні запитання

1. Що означає термін БПЛА та що таке мультикоптер?
2. Завдяки чому літають БПЛА?
3. Як визначити підіймальну силу крила?
4. Якими кутами управляють під час польоту БПЛА?
5. Що таке центрування?
6. Навіщо БПЛА потрібен політний контролер?
7. Від чого залежить напрям руху квадрокоптера?
8. Наведіть класифікацію дронів за сферами застосування.
9. Які різновиди безпілотних літальних апаратів бувають за видом аероднамічної схеми?

Лекція №3. Теорія транспортних роботів

Транспортний робот – це програмнокерований механізм, який автоматично переміщається за програмою управління в межах дільниці, цеху, заводу (рис. 3.1).

Роль рухомої системи робота зазвичай відіграє колісне або гусеничне шасі разом з вбудованими тяговими і кермовими приводами. Мобільні транспортні роботи, окрім основних завдань з переміщення вантажів, можуть бути налаштовані на додаткові функціх, наприклад, обстеження обєктів і побудови їхньої мапи або роль мобільної платформии для інформаційних систем.

Умова руху мобільного робота визначається за відсутності буксування ходового механізму, тобто:

$$F_{зч} > F_к^{max}, \tag{3.1}$$

де $F_к^{max}$ – максимальна тягова сила; $F_{зч}$ – сила зчеплення рушія.

Сила зчеплення коліс або гусениць з опорною поверхнею:

$$F_{зч} = \mu_0 N_{пр}, \tag{3.2}$$

де μ_0 – коефіцієнт зчеплення (μ_0 = 0.6...0,75 – для колісного ходу по цементобетонному покриттю; μ_0 = 0,15...0,35 – для колісного ходу по

сніговій ущільненій дорозі; μ_0 = 0,38...0,45 – для гусеничного ходу по асфальтобетонному покриттю ; μ_0 = 0,22...0,25 – для гусеничного ходу по сухій піщаній дорозі); $N_{пр}$ – навантаження на привідні колеса або гусениці.

а

б

в

г

д

Рис. 3.1 – Транспортні роботи: *а* – система iRobot Packbot 510, яка застосовувалася на атомній станції Фукусима; *б* – траулер для обстеження трубопроводів; *в* – AGV (Automated Guided Vehicle), який застосовують в порту Роттердам для перевезення контейнерів; *г* – платформа QUICKTRON, застосовувана для автоматизації складу Amazon; *д* – дрон XAG R150 для сільського господарства

Статичний опір пересування ходового механізму по шляху з кутом нахилу γ:

$$W_{\text{с}} = W_{\text{п}} \pm W_{\text{у}} \pm P_{\text{в}} + W_{\text{кр}}, \qquad (3.3)$$

де $W_{\text{п}} = \varpi m_{\text{м+в}} g \cos\gamma$ – сила опору кочення колеса або гусениці, Н; ϖ – коефіцієнт опору пересування колеса або гусениці (ϖ = 0,015...0,035 – для асфальтобетонного покриття дороги; ϖ = 0,1...0,3 – для сухої піщаної дороги); $W_{\text{у}} = m_{\text{м+в}} g \sin\gamma$ – опір від ухилу опорної поверхні, Н; $W_{\text{кр}}$ – сила опору пересування під час повороту пневмоколісного рушія, яка залежить від колісної та осьової формул, схеми трансмісії, бази та колії, радіуса повороту (наближено можна вважати рівною $W_{\text{п}}$, а для гусеничних рушіїв під час повороту навколо загальмованої гусениці $W_{\text{кр}} = 0{,}5\mu_{\text{кр}} G_{\text{м+в}} \frac{L}{B}$, де $\mu_{\text{кр}}$ – коефіцієнт тертя ковзання гусениці по грунту ($\mu_{\text{кр}}$ = 0,35...0,45 – для асфальтобетонного покриття доріг; $\mu_{\text{кр}}$ = 0,2...0,25 – для сухої піщаної дороги); L – довжина гусениць по осях зірочок; B – ширина між гусеницями по їхніх повздовжніх осях); $m_{\text{м+в}}$ – маса машини та вантажу, кг; g – прискорення вільного падіння, м/с^2.

Вітрове навантаження $P_{\text{в}}$ розраховують за швидкості руху, більшеої від 50 км/год. Опір від вітрового навантаження:

$$P_{\text{в}} = g_{\text{в}} F_{\text{фр}}, \qquad (3.4)$$

де $g_{\text{в}}$ – тиск вітру; $F_{\text{фр}}$ – фронтальна площа машини.

Тягова сила $F_{\text{к}}$ на колесах або гусеницях дорівнює:

$$F_{\text{к}} = W_{\text{с}} + F_{\text{ін}}, \qquad (3.5)$$

де $F_{\text{ін}}$ – сила інерції маси машини.

Опір від сил інерції пропорційний масі машини $G_{\text{м}}/g$ та її прискоренню за час руху з місця чи в разі зміни швидкості машини. Цей опір можна визначити приблизно за формулою

$$F_{\text{ін}} = \frac{G_{\text{м}}(V_2 - V_1)}{g t_{\text{р}}}, \qquad (3.6)$$

де V_1, V_2 – швидкості на нижній і вищій передачах, з якої і на яку перемикає система (для руху з місця V_1 = 0); $t_{\text{р}}$ – час розгону машини (для колісних – $t_{\text{р}}$ = 5...10 с; для гусеничних – $t_{\text{р}}$ = 3...6 с).

Рух з місця відбувається на першій передачі, а тому для гусеничних машин можна брати V_2 = 0,25...0,3 м/с, для колісних – V_2 = 1...1,5 м/с.

Необхідний момент двигуна за заданої швидкості пересування $\upsilon_{\text{пер}}$:

$$M_{\text{дв}} = \frac{F_{\text{к}} r_{\text{к}}}{i_{\text{т}} \eta_{\text{т}}}, \tag{3.7}$$

де $r_{\text{к}}$ – динамічний радіус пневматичного колеса або радіус приводного колеса гусениці; $\eta_{\text{т}}$ = 0,75...0,9 – ККД трансмісії; $i_{\text{т}} = i_{\text{к}} \cdot i_{\text{д}} \cdot i_{\text{г}}$ – передатне число трансмісії; $i_{\text{к}}$, $i_{\text{д}}$, $i_{\text{г}}$ – передатні числа відповідно коробки передач, додаткової коробки передач і головної передачі (в разі гідромеханічної трансмісії враховують кінематичне передатне число гідротрансформатора).

Потужність двигуна на переміщення транспортного робота:

$$N = \frac{F_{\text{к}} \upsilon_{\text{пер}}}{\eta_{\text{т}}}. \tag{3.8}$$

Ефективне управління мобільним роботом є основним предметом дослідження подібних транспортних систем, з цією метою вирішують дві основні задачі: побудови траєкторії руху робота та визначення керівних сигналів для переміщення робота за наперед заданою траєкторією. При цьому фокусуються на факті, що мобільний робот може бути не наділений досконалими засобами для плавного маневрування та стабілізації, що породжує проблему стабільності руху і керованості роботом.

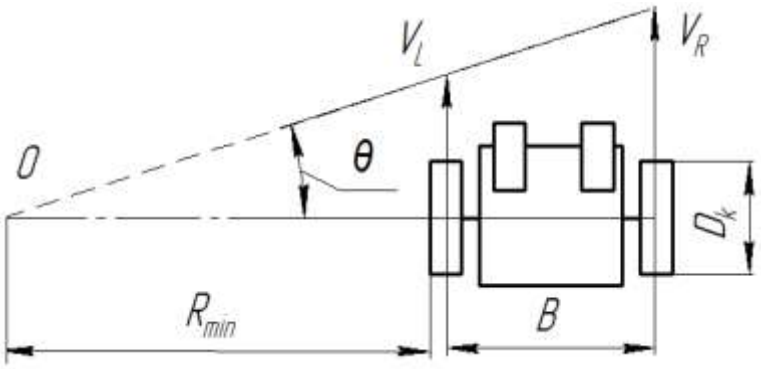
Рушійні системи роботів конструктивно можна поділити на колісні або гусеничні; з бортовим поворотом або з маневрувальними колесами (рис. 3.2).

У двоколісного робота з диференційним приводом так само, як і для робота з гусеничними рушієм, виконання повороту на заданий кут є можливим завдяки обертанню коліс (гусениць) з різною швидкістю. При цьому швидкості можуть мати й однаковий напрям, і різний. Якщо напрям швидкостей коліс двоколісного робота однаковій, його переміщення відбуватиметься за криволінійною траєкторією. Кут повороту робота буде визначатися за лінійними швидкостями центрів коліс:

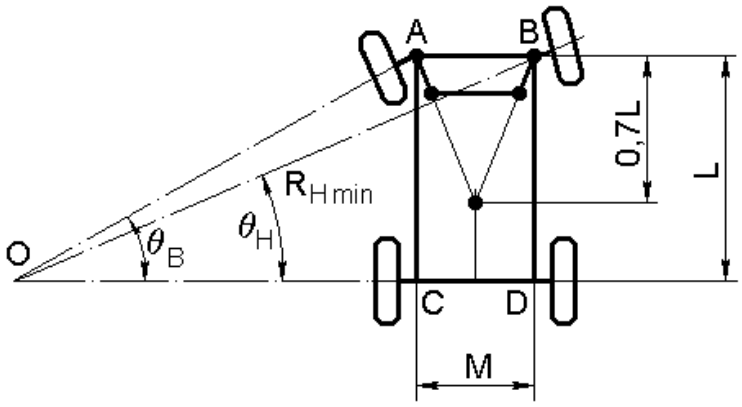
$$\theta = \arctan\left(\frac{V_R - V_L}{B}\right), \tag{3.9}$$

де V_L та V_R – швидкості центру лівого та правого коліс робота; B – колія або відстань між центрами лівого та правого коліс (рис. 3.2, *а*).

У випадку, коли одна зі швидкостей колеса дорівнюватиме нулю, робот буде обертатися навколо центра контакту цього колеса з опорною поверхнею. Якщо за виразом (3.9) кут буде зі знаком «-», тоді обертання відбуватиметься в протилежний бік до визначеного за рис. 3.2, *а*.



а



б

Рис. 3.2. Схеми транспортних роботів: *а* – з бортовим поворотом; *б* – з маневреними колесами: B, M – колія; L – база; R_{min} – радіус повороту; V_L, V_R – швидкість лівого та прового коліс відповідно

Відстань від центра повороту (миттєвого центра швидкостей) до центра, наприклад, лівого колеса згідно з рис. 3.2, *а*:

$$R_L = \frac{B}{(\frac{V_R}{V_L} - 1)}. \tag{3.10}$$

Кут повороту платформи двоколісного робота може бути визначений через довжину дуги, пройденої одним його колесом під час повороту, град:

$$\theta = \frac{180^\circ l_L}{\pi R_L} \tag{3.11}$$

або

$$\theta = \frac{180^\circ V_L \Delta t(\frac{V_R}{V_L} - 1)}{\pi B} = \frac{180^\circ \Delta t(V_R - V_L)}{\pi B}, \tag{3.12}$$

де l_L – довжина дуги, пройденої колесом.

Якщо на колесі робота встановлено датчик, який розраховує кількість його обертів, тоді довжина шляху, пройденого колесом, буде

$$l_{L(R)} = \pi D_k \frac{\sum N}{N_{ен1}}, \tag{3.13}$$

де $N_{ен1}$ – кількість тактів перемикання датчика (енкодера) за один оберт колеса; $\sum N$ – загальна кількість перемикань датчика за заданий проміжок часу або на заданій ділянці переміщення; D_k – діаметр колеса робота.

Кількість обертів колеса на заданій ділянці переміщення:

$$n = \frac{l_{L(R)}}{\pi D_k}. \tag{3.14}$$

Частота обертання колеса за період часу Δt :

$$n_k = \frac{n}{\Delta t} = \frac{V_{L(R)}}{\pi D_k}. \tag{3.15}$$

Кутова швидкість обертання колеса:

$$\omega_k = \frac{2V_{L(R)}}{D_k}. \tag{3.16}$$

Щодо двоколісного робота з диференціальним приводом в деякій плоскій нерухомій системі координат, то його положення відносно центра такої системи координат буде визначатися через координати *x* та *y*. Лінійна швидкість *V* центра робота буде залежати від швидкостей за координатами і визначатиметься зі співвідношення:

$$\begin{cases} \dfrac{dx}{dt} = \dot{x} = V\cos\theta; \\ \dfrac{dy}{dt} = \dot{y} = V\sin\theta, \end{cases} \tag{3.17}$$

де V – швидкість руху центра робота; θ – кут повороту платформи робота ($\dot{\theta} = \omega$ – кутова швидкість повороту робота).

Лінійна швидкість робота *V* може бути знайдена за усередненням лінійних швидкостей його лівого і правого колеса:

$$V = \frac{V_R + V_L}{2}. \tag{3.18}$$

де V_L – лінійна швидкість лівого колеса робота; V_R – лінійна швидкість правого колеса робота.

Кутова швидкість ω може бути знайдена залежно від швидкостей на кожному колесі робота та розміру колії:

$$\omega = \frac{V_R + V_L}{B} \ . \tag{3.19}$$

Беручи до уваги формули (3.16), (3.18) та (3.19), можемо записати таку залежність:

$$\omega_R + \omega_L = \frac{4V}{D_k} = \frac{2V}{R_k} \ , \tag{3.20}$$

де ω_R та ω_L – кутові швидкості обертання відповідно правого та лівого колеса; D_k , R_k – діаметр та радіус колеса відповідно; V – швидкість робота.

Кутова швидкість повороту платформи робота:

$$\omega = \frac{V_R - V_L}{B} \tag{3.21}$$

або

$$\omega = \frac{(\omega_R - \omega_L)D}{2B} = \frac{(\omega_R - \omega_L)R}{B} \ , \tag{3.22}$$

де B – колія між центрами лівого та правого колеса.

Кутові швидкості правого та лівого колеса платформи робота можна визначити через швидкість руху робота та його кутову швидкість повороту:

$$\omega_R = \frac{2V + \omega B}{2R_k} \ ; \tag{3.23}$$

$$\omega_L = \frac{2V - \omega B}{2R_k} \ . \tag{3.24}$$

Контролювати кутову швидкість обертання коліс транспортної платформи можна через частоту їхнього обертання:

$$\omega_k = 2\pi n_k \, , \tag{3.25}$$

де n_k – частота обертання колеса в оберт/с.

Для робота з маневреними колесами (див. рис. 3.2, *б*) під час руху на повороті траєкторії всіх коліс повинні бути дугами концентричних кіл зі спільним центом в точці *O*. Для цього керовані колеса мають бути повернені на різні кути. Зв’язок між кутами повороту зовнішнього та внутрішнього колеса визначається з геометричних співвідношень (згідно зі схемою на рис. 3.2, *б*):

$$ctg\theta_H - ctg\theta_B = \frac{OD - OC}{L} = \frac{CD}{L} = \frac{M}{L} \, , \tag{3.26}$$

де θ_H і θ_B – кути повороту відповідно зовнішнього та внутрішнього колеса; L – довжина бази; M – відстань між осями шворнів ($AB = CD$).

Для наведеної схеми колісного робота з передніми керованими колесами на рис. 3.2, *б* центр повороту взято на продовженні осі задніх коліс, що відображає *принцип Анкермана* (зв'язок між керованими колесами здійснюється за допомогою кермової трапеції). Таким чином кути повороту керованих коліс на внутрішньому та зовнішньому радіусах повороту будуть різними та визначатимуться за такими співвідношеннями:

$$tg\theta_B = \frac{L}{OC};\ tg\theta_H = \frac{L}{OD}. \tag{3.27}$$

Мінімальний радіус повороту двовісної, тривісної платформи робота з передніми керованими колесами можна визначити так:

$$R_{\min} = \frac{L}{\sin\theta_{H\max}}, \tag{3.28}$$

де θ_{max} – максимальний кут повороту зовнішнього керованого колеса.

Мінімальний радіус повороту робота з всіма керованими колесами (і передніми і задніми):

$$R_{\min} = \frac{L}{2\sin\theta_{H\max}}. \tag{3.29}$$

Положення центра колісної платформи робота:

$$x_1 = x + l_1\cos(\varphi_1 + \varphi); \tag{3.30}$$

$$y_1 = y + l_1\sin(\varphi_1 + \varphi), \tag{3.31}$$

де x, y – координати попереднього положення центра транспортної платформи робота; l_1 – відстань, пройдена центральною точкою робота; φ_1 та φ – відповідно кут, на який повернуто робота під час переходу з попереднього положення в поточне та кут попереднього повороту.

Переміщення центра робота під час руху по радіусу може бути знайдене через довжину пройденого шляху правою та лівою стороною колісної платформи робота:

$$l_c = \frac{l_L + l_D}{2}. \tag{3.32}$$

Мобільний робот з омні-колесами та меканум-колесами може рухатися на площині (або на незначно нерівній поверхні) у довільному напрямку без повороту корпуса або на довільних поворотах. Зміна напряму руху подібних роботизованих платформ відбувається завдяки

зміні комбінації обертання коліс. Наприклад, для колісної платформи з чотирма меканум-колесами схеми руху залежно від обертання коліс наведено на рис. 3.3.

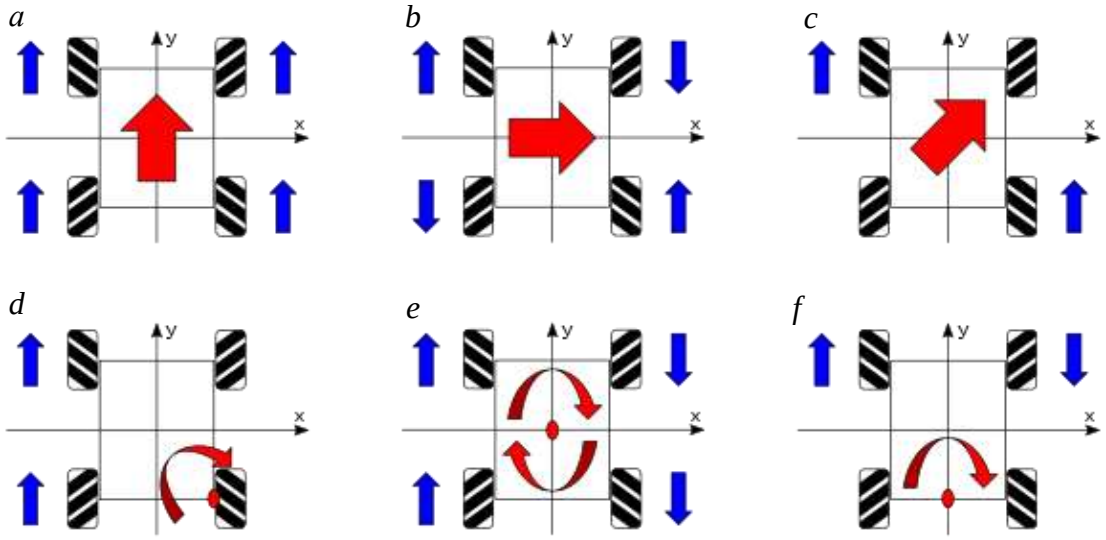
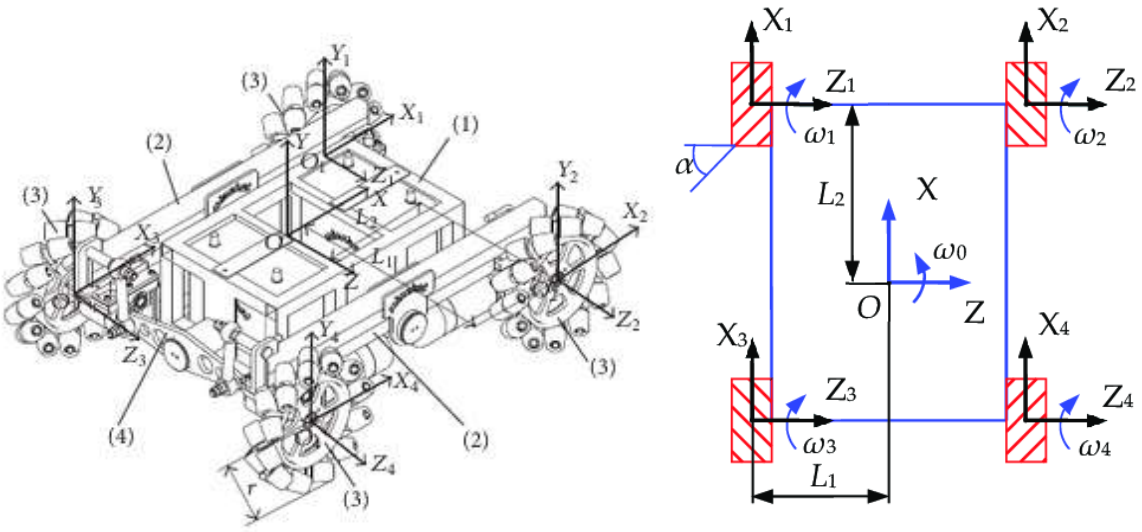


Рис. 3.3. Схема комбінації рухів колісної платформи з меканум-колесами

Робот, яий має декілька меканум-коліс, для власного маневрування повинен реалізовувати різну швидкість їхнього обертання. Кінематика робота з меканум-колесами залежатиме від параметрів контрукції колеса та їхнього розміщення. Для моделі чотириколісного меканум-робота, у якого колеса розміщенно в два ряди вздовж сторін прямокутного корпуса (рис. 3.4) з колесами, які мають однакову орієнтацію, розміщеними по діагоналі платформи робота (на рис. 3.4, *б* орієнтацію показано похилими лініями), швидкість центру платформи та кутова частота повороту буде визначатися з рівняння

$$\begin{bmatrix} \upsilon_X \\ \upsilon_Z \\ \omega_O \end{bmatrix} = \frac{r}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ tg\alpha & -tg\alpha & -tg\alpha & tg\alpha \\ \dfrac{-tg\alpha}{L_1 tg\alpha + L_2} & \dfrac{tg\alpha}{L_1 tg\alpha + L_2} & \dfrac{-tg\alpha}{L_1 tg\alpha + L_2} & \dfrac{tg\alpha}{L_1 tg\alpha + L_2} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix}, \qquad (3.33)$$

де υ_X, υ_Z – швидкості чотириколісної платформи робота за відповідними осями координат; ω_O – кутова швидкість обертання платформи; $\omega_1 \dots \omega_4$ – кутові швидкості кожного колеса; r – радіус колеса.



а *б*

Рис. 3.4. Конструкція меканум-робота (*а*) та його розрахункова схема (*б*)

Розв’язок зворотної задачі для такої конструкції робота:

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} 1 & \frac{1}{tg\alpha} & \frac{-L_1 tg\alpha + L_2}{tg\alpha} \\ 1 & \frac{-1}{tg\alpha} & \frac{L_1 tg\alpha + L_2}{tg\alpha} \\ 1 & \frac{-1}{tg\alpha} & \frac{-L_1 tg\alpha + L_2}{tg\alpha} \\ 1 & \frac{1}{tg\alpha} & \frac{L_1 tg\alpha + L_2}{tg\alpha} \end{bmatrix} \begin{bmatrix} \upsilon_X \\ \upsilon_Z \\ \omega_O \end{bmatrix}. \qquad (3.34)$$

Розглянемо основне рівняння руху мобільного робота з синхронним приводом. Для цього припустимо, що поступальною та обертальною швидкістю робота можна керувати незалежно (з обмеженими крутними моментами). Щоб рівняння стали практичними, вводимо наближення, яке моделює швидкість як кусково-постійну функцію в часі. Згідно з цим припущенням траєкторії роботів складаються з послідовностей кінцевої кількості сегментів кіл. Такі уявлення дуже зручні для перевірки зіткнень, оскільки перетини перешкод з колами легко перевірити. Нехай $x(t)$ і $y(t)$ описують координати робота в момент часу t у деякій глобальній системі координат, а орієнтація (напрямок руху) описується функцією кута $Q(t)$. Таким чином, вектор $<x, y, Q>$ описує кінематичну конфігурацію робота. Рух робота з синхронним приводом обмежений таким чином, що поступальна швидкість v завжди

спрямована до Q-керування, що є неголономним обмеженням. Якщо $x(t_0)$ і $x(t_n)$ позначити координати робота в моменти часу t_0 і t_n відповідно, а $v(t)$ та $\omega(t)$ – поступальну швидкість та швидкість обертання в ті самі моменти часу, тоді $x(t_n)$ та $y(t_n)$ можна виразити як функцію:

$$x(t_n) = x(t_0) + \int_{t_0}^{t_n} \upsilon(t)\cos Q(t)dt; \tag{3.35}$$

$$y(t_n) = y(t_0) + \int_{t_0}^{t_n} \upsilon(t)\sin Q(t)dt, \tag{3.36}$$

де Q – кут повороту робота.

Швидкість $v(t)$ залежить від початкової поступальної швидкості $v(t_0)$ та поступального прискорення $\dot{\upsilon}(t)$ в інтервалі часу від t_0 до t. Подібним чином орієнтація $Q(t)$ є функцією початкової орієнтації $Q(t_0)$, початкової швидкості обертання $\omega(t_0)$ у момент часу t_0 та прискорення обертання $\dot{\omega}(t)$ в інтервалі $t \in [t_0, t]$. Заміна швидкості $v(t)$ та кута повороту $Q(t)$ призведе до зміни координат положення робота таким чином:

$$x(t_n) = x(t_0) + \int_{t_0}^{t_n}\left[\left(\upsilon(t_0) + \int_{t_0}^{t_n}\dot{\upsilon}(t)dt\right)\cdot\cos\left(Q(t_0) + \int_{t_0}^{t_n}\left(\omega(t_0) + \int_{t_0}^{t_n}\dot{\omega}(t)dt\right)dt\right)\right]dt; \tag{3.37}$$

$$y(t_n) = y(t_0) + \int_{t_0}^{t_n}\left[\left(\upsilon(t_0) + \int_{t_0}^{t_n}\dot{\upsilon}(t)dt\right)\cdot\sin\left(Q(t_0) + \int_{t_0}^{t_n}\left(\omega(t_0) + \int_{t_0}^{t_n}\dot{\omega}(t)dt\right)dt\right)\right]dt. \tag{3.38}$$

Цифрове обладнання накладає обмеження на інтервали часу, коли можна вмикати струм двигуна, тоді припускають, що між двома довільними моментами часу t_0 і t_n, роботом можна керувати лише кінцевою кількістю команд прискорення. Нехай n позначає цю кількість часових тактів, тоді прискорення $\dot{\upsilon}_i$ та $\dot{\omega}_i$ для інтервалу часу $[t_i, t_{i+1}]$ вважають постійними і визначеними. Використовуючи дискретну форму, динамічну поведінку робота з синхронним приводом можна виразити рівнянням в припущенні кусково-постійних прискорень:

$$x(t_n) = x(t_0) + \sum_{i=0}^{n-1}\left(\int_{t_i}^{t_{i+1}}\left[\left(\upsilon(t_0) + \dot{\upsilon}_i\Delta t\right)\cdot\cos\left(Q(t_0) + \omega(t_i)\Delta t + \frac{1}{2}\dot{\omega}_i(\Delta t)^2\right)\right]dt\right); \tag{3.39}$$

$$y(t_n) = y(t_0) + \sum_{i=0}^{n-1}\left(\int_{t_i}^{t_{i+1}}\left[\left(\upsilon(t_0) + \dot{\upsilon}_i\Delta t\right)\cdot\sin\left(Q(t_0) + \omega(t_i)\Delta t + \frac{1}{2}\dot{\omega}_i(\Delta t)^2\right)\right]dt\right), \tag{3.40}$$

де $\Delta t = t - t_i$ – проміжок часу, с.

Отримані рівняння (3.39) та (3.40) описують загальний випадок мобільного кореневого контролю. Створені за цими рівняннями траєкторії є складними геометричні кривими.

У практиці керування мобільними роботами застосовують апроксимовані траєктрорії для даних рівнянь. Наприклад, метод апроксимації за швидкостю полягає в тому, що беруть її стале значення в інтервалі часу $[t_i, t_{i+1}]$. Таке припущення дає змогу описувати траєкторію руху робота кусковими рівняннями дуги кола, що добре підходить для керування рухом у реальному часі.

Якщо проміжки часу $[t_i, t_{i+1}]$ достатньо малі, складову $\upsilon(t_i)+\dot{\upsilon}_i\Delta t$ можна апроксимувати довільною поступальною швидкістю $\upsilon_i \in [\upsilon(t_i), \upsilon(t_{i+1})]$ завдяки плавності руху робота. Аналогічно $Q(t_0)+\omega(t_i)\Delta t+\frac{1}{2}\dot{\omega}_i(\Delta t)^2$ можна апроксимувати виразом $Q(t_0)+\omega(t_i)\Delta t$, $\omega_i \in [\omega(t_i), \omega(t_{i+1})]$, що призведе до перетворення залежностей (3.39) та (3.40):

$$x(t_n)=x(t_0)+\sum_{i=0}^{n-1}\left(\int_{t_i}^{t_{i+1}}\left[\upsilon_i\cdot\cos\left(Q(t_0)+\omega_i\Delta t\right)\right]dt\right). \qquad (3.41)$$

Розв'язок інегралу являє собою таке рішення:

$$x(t_n)=x(t_0)+\sum_{i=0}^{n-1}\left(F_x^i(t_{i+1})\right); \qquad (3.42)$$

$$y(t_n)=y(t_0)+\sum_{i=0}^{n-1}\left(F_y^i(t_{i+1})\right), \qquad (3.43)$$

де

$$F_x^i(t)=\begin{cases}\frac{\upsilon_i}{\omega_i}(\sin Q(t_i)-\sin(Q(t_i)+\omega_i\Delta t)), \omega_i\neq 0;\\ \upsilon_i\cos(Q(t_i))\cdot t,\ \omega_i=0;\end{cases}$$

$$F_y^i(t)=\begin{cases}-\frac{\upsilon_i}{\omega_i}(\cos Q(t_i)-\cos(Q(t_i)+\omega_i\Delta t)), \omega_i\neq 0;\\ \upsilon_i\sin(Q(t_i))\cdot t,\ \omega_i=0.\end{cases}$$

Якщо $\omega_i = 0$, тоді робот рухатиметься прямою лінією, а якщо $\omega_i \neq 0$, траєкторія робота описуватиме коло, для якого буде справедливим таке співвідношення:

$$(F_x^i-M_x^i)^2+(F_y^i-M_y^i)^2=(\frac{\upsilon_i}{\omega_i})^2, \qquad (3.44)$$

де $M_x^i = -\frac{\upsilon_i}{\omega_i}\sin Q(t_i)$; $M_y^i = -\frac{\upsilon_i}{\omega_i}\cos Q(t_i)$.

Це означає, що *i*-та траєкторія робота є колом з координатами миттєвого центра приблизно (M_x^i, M_y^i) та радіусом $M_к^i = \frac{\upsilon_i}{\omega_i}$.

Контрольні запитання

1. Якими є умови руху транспортного робота? Від чого залежить опір руху колеса робота?
2. Як визначити потужність на двигуні транспортної системи робота?
3. Якими параметрами робота визначають його положення в просторі?
4. Як керують роботами з омні-колесами та яка різниця між омні-колесом та колесом Ілона?
5. Як визначити шлях, який проходить колісний робот?
6. Від чого залежить кут повороту платформи двоколісного робота?
7. Як визначити кутову швидкість колеса через параметри колісної платформи робота?
8. Як визначити положення колісної платформа робота в просторі за кінематичними характеристиками?
9. Завдяки чому, на вашу думку, в пружинних профілювальних колесах досягається гнучкість?

Лекція № 4. Інформаційні системи роботів

Інформаційна система робота призначена для збору інфомації про зовнішнє середовище і ухвалення рішень, пов'язаних з аналізом отриманих даних та вибору програми керування, яка формуватиме сигнали для системи управління. Оскільки інформація часто буває неповною, а робототехнічна система функціонує в умовах невизначеності, і при цьому можуть змінюватися параметри робототехнічної системи, структура і алгоритм її функціонування, на інформаційну систему накладаються значні вимоги до якості. Інформаційно-вимірювальна система робота отримує інформацію про зовнішнє середовище через систему сенсорів, за допомогою яких визначає стан середовища, та систему управління, яка сприймає зв'язок із зовнішнім середовищем через виконавчу систему, тому сенсори та їхні схеми комутації є основою інформаційних систем роботів.

Сьогодні терміни «сенсор» і «датчик» використовують як рівнозначні для позначення вимірювального перетворювача, що виконує функції сприйняття вхідної величини та формування вимірювального сигналу, проте термін «сенсор» більшою мірою стосується систем, які сприймають вхідні величини, а термін «датчик» – до систем, які формують вимірювальні сигнали. По суті, *датчиком* можна вважати конструктивно окремий пристрій, що містить один або кілька первинних вимірювальних перетворювачів (сенсорів). Датчики під впливом вимірюваної величини видають еквівалентний сигнал (струм, напруга), що є однозначною функцією вимірюваної величини. Датчики (рис. 4.1) можуть вимірювати характеристики робота, такі як положення узагальнених координат, швидкість робочого органа, зусилля і моменти в кінематичних парах ланок робота (кінестетичні датчики) або надавати інформацію про зовнішнє середовище (візуальні, локаційні, тактильні датчики).

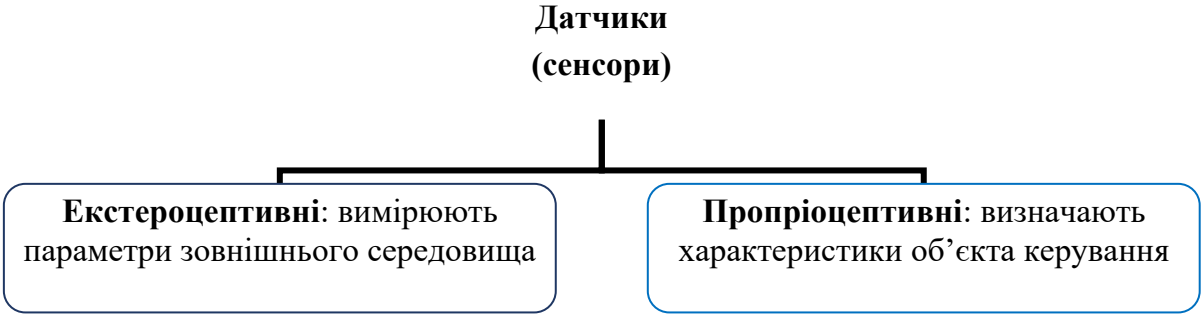


Рис. 4.1. Основні різновиди датчиків

Властивості датчиків визначаються їхніми статичними, динамічними, частотними та іншими характеристиками.

1. *Статична характеристика* – це залежність вихідної величини від вхідної: $Y = f(X)$.
2. *Динамічна характеристика* – це залежність вихідної величини від вхідної в перехідний період.
3. *Частотна характеристика* визначається розмахом між найнижчими та найвищими вхідними даними, часто записується в дБ:

$$DR = 10 \cdot \log\left(\frac{X_{\max}}{X_{\min}}\right).$$

4. *Чутливість* (коефіцієнт перетворення) – це відношення зміни вихідного ΔY-сигналу до величини вхідного ΔX-сигналу: $k = \Delta Y/\Delta X$. У лінійних датчиків чутливість у всьому робочому діапазоні вимірювань однакова і рівняння статики має вигляд: $Y = kX + X_0$.
5. *Межа чутливості* – мінімальна величина на вході датчика, що зумовлює зміну його вихідної величини (залежить від внутрішніх та зовнішніх факторів – тертя, люфту, гістерезисну, перешкод).
6. *Похибка*: абсолютна – різниця між дійсним значенням Y вихідної величини та виміряним значенням Y_i ($\Delta Y = Y_i - Y$), а відносна – це відношення абсолютної похибки до дійсного значення у відсотках ($\delta = \Delta Y/Y \cdot 100\%$).
7. *Роздільна здатність* – це мінімальна відстань між двома виміряними значеннями. Для аналогових датчиків ця величина визначається роздільною здатністю аналого-цифрового перетворювача (АЦП), тобто якщо вхідний діапазон АЦП від 0 до +10 вольт з роздільною здатністю 12 біт, то діапазон 10 вольт ділиться на 12 біт, або 4096 поділок. Таким чином, зміна на вході АЦП на 1 біт є відповідною різниці 0,00244 вольт (10 вольт/4096 поділок).
8. Точність – це різниця між виміряним і дійсними значенням датчика ($w = (1- \Delta Y/Y) \cdot 100\%$).

До датчиків роботів ставлять такі вимоги:

- уніфікованість (має нормативний діапазон вихідного сигналу);
- висока надійність і стійкість до електромагнітних перешкод, коливань напруги і частоти;

- малогабаритність і простота конструкції;
- опторозв'язки вихідних і вхідних ланцюгів, простота тарування та обслуговування;
- можливість абсолютного відліку параметрів.

Первинним перетворювачем, або *чутливим елементом*, називається найпростіший елемент інформаційної системи, що змінює свій стан під дією зовнішнього збурення (наприклад, фототранзистор, фотодіод або тензорезистор). Найбільш розповсюдженими електричними первинними перетворювачами є перетворювачі опору, ємнісні, електромагнітні, п'єзоелектричні, термоелектричні тощо.

Перетворювачі опору змінюють активний опір вимірювального ланцюга під дією вимірювальної величини (механічної, термічної, оптичної). В перетворювачах опору застосовано закон Ома про пропорційність сили струму в провіднику прикладеній напрузі. Якщо для елемента електричного кола є справедливим закон Ома, то цей елемент має лінійну вольт-амперну характеристику.

В електротехніці закон Ома записують в інтегральному вигляді:

$$U = I*R, \tag{4.1}$$

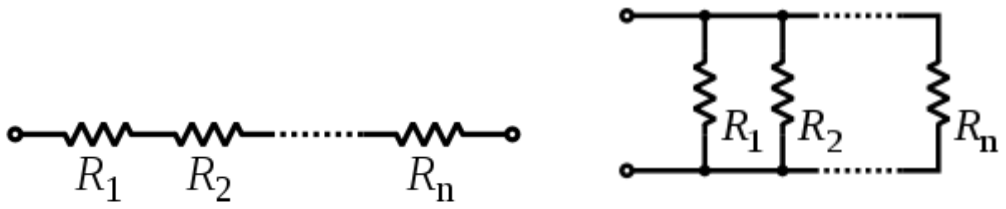
де U – прикладена напруга, В; I – сила струму, A; R – електричний опір провідника, Ом.

Якщо провідник має площу перерізу S, тоді опір R визначають через питому провідність

$$R = \rho \frac{l}{S}, \tag{4.2}$$

де $\rho = 1/\sigma$ – питомий опір; σ – питома провідність.

Послідовне і паралельне з'єднання в електротехніці (рис. 4.2) – це два основних способи з'єднання елементів електричного кола. За послідовного з'єднання всі елементи пов'язані один з одним так, що ділянка кола не має жодного вузла, тоді як за паралельного з'єднання всі вхідні в коло елементи об'єднані двома вузлами і не мають зв'язків з іншими вузлами.



а *б*

Рис. 4.2. Схеми послідовного (*а*) та паралельного з'єднання провідників (*б*)

У разі послідовного з’єднання провідників сила струму в усіх провідниках однакова:

$$I_1 = I_2 = \ldots = I_N. \quad (4.3)$$

Повна напруга в колі за послідовного з’єднання дорівнює сумі напруг на окремих ділянках кола:

$$U = U_1 + U_2 + \ldots + U_N. \quad (4.4)$$

Загальний опір усієї ділянки кола дорівнює сумі опорів:

$$R = R_1 + R_2 + \ldots + R_N. \quad (4.5)$$

У разі паралельного з’єднання спад напруги між двома вузлами, що поєднують елементи кола, однаковий для всіх елементів. При цьому величина, обернена загальному опору кола, дорівнює сумі величин, обернених опору паралельно з’єднаних провідників.

Сила струму в нерозгалуженій частині кола дорівнює сумі сил струмів в окремих паралельно з’єднаних провідниках:

$$I = I_1 + I_2 + \ldots + I_N. \quad (4.6)$$

Напруга на ділянках кола і на кінцях всіх паралельно з’єднаних провідників однакова:

$$U_1 = U_2 = \ldots = U_N. \quad (4.7)$$

Опір ділянки визначається з рівняння – провідність ділянки є сумою провідностей елементів:

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} + \ldots + \frac{1}{R_N}. \quad (4.8)$$

Загальна ємність батареї паралельно з’єднаних конденсаторів дорівнює сумі ємностей всіх конденсаторів, з яких складено батарею:

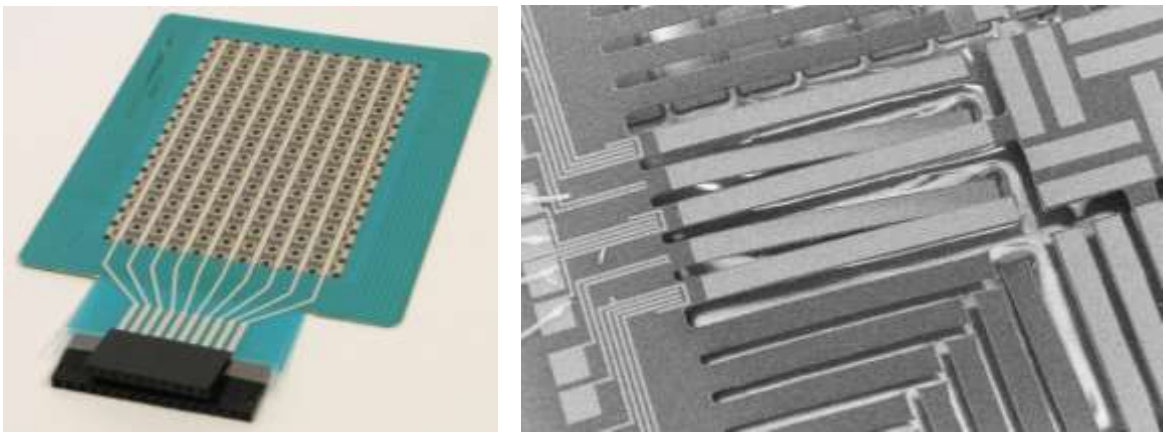
$$C = \sum_{i=1}^{n} C_i. \quad (4.9)$$

Для отримання великих ємностей конденсатори з’єднують паралельно.

За послідовного з’єднання заряди всіх конденсаторів однакові. Загальна ємність батареї послідовно з’єднаних конденсаторів дорівнює:

$$C = \frac{1}{\sum_{i=1}^{n} \frac{1}{C_i}}. \quad (4.10)$$

Тактильні датчики є датчиками контактного типу, які визначають характер контакту з об’єктами зовнішнього середовища. Такі датчики мають вид тактильних матриць (рис. 4.3) або силомоментних датчиків (рис. 4.4).



а *б*

Рис. 4.3. Тактильна матриця: *а* – вид датчика; *б* – структура матриці датчика

а *б*

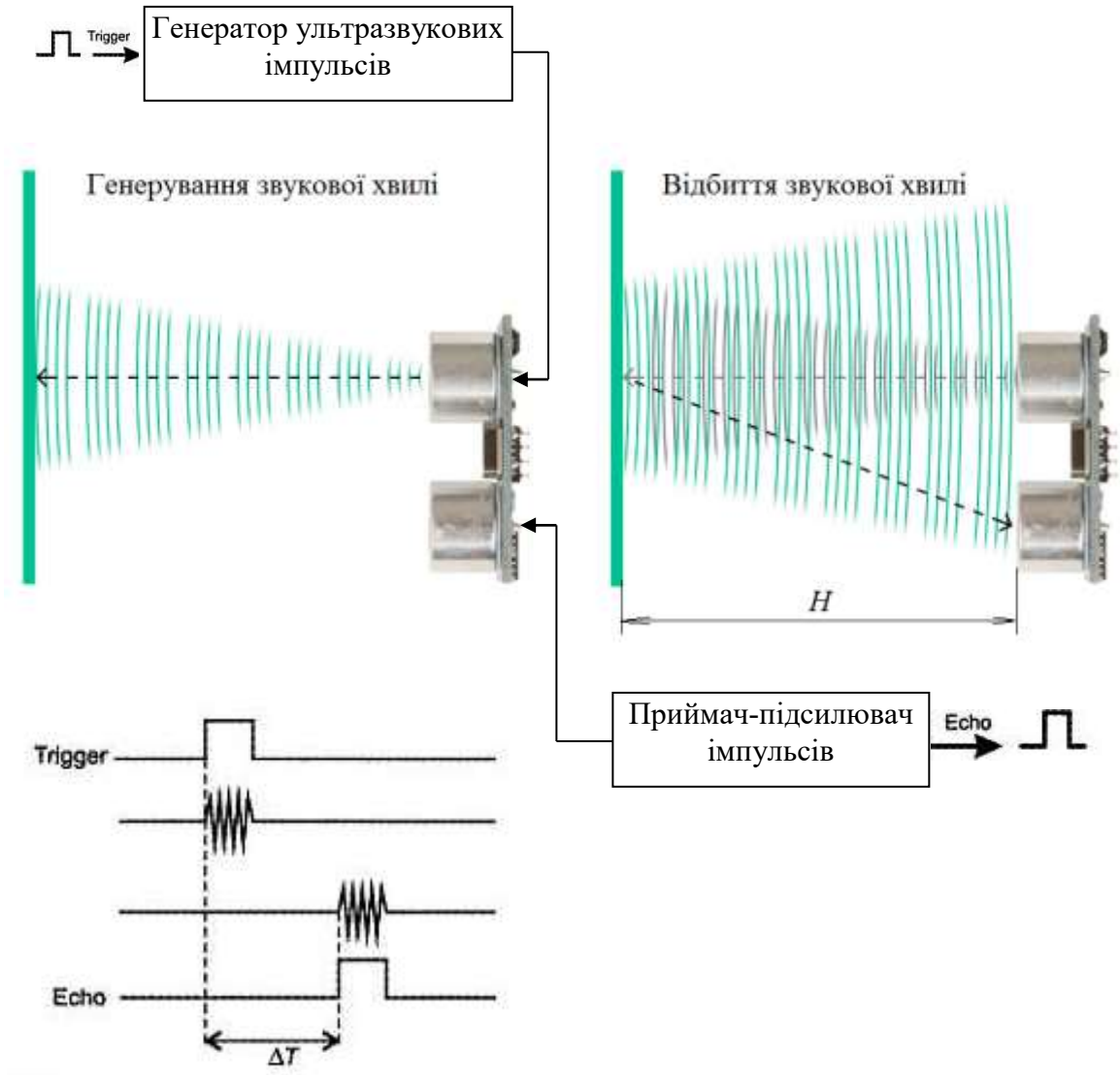
Рис. 4.4. Принцип дії та схема (*а*) вмикання електроконтрактного датчика (*б*)

Локаційні датчики призначені для визначення і вимірювання фізичних параметрів середовища шляхом випромінювання та приймання відбитих від об’єктів сигналів (оптичних, акустичних, електромагнітних) на основі яких формується локальний образ середовища.

Ультразвуковий сонар – це датчик відстані, який складається з генератора ультразвуку, випромінювача, приймача та підсилювача-формувача вихідного сигналу (рис. 4.5).

Промислові ультразвукові сонари працюють на частотах 40-50 кГц, при цьому швидкість поширення звукових хвиль у повітрі становить 343 м/с та 1496 м/с – у чистій воді.

а *б*



в

Рис. 4.5. Ультразвуковий сонар: *а* – модуль HC-SR04 ультразвукового сонара; *б* – промисловий ультразвуковий датчик BUS003F; *в* – принцип роботи сонара

Швидкість поширення υ звукових хвиль залежить від частоти f генератора хвиль та довжини хвилі λ:

$$\upsilon = \lambda f \ . \qquad (4.11)$$

Найпростіші сонари не мають власного мікроконтролера і працюють завдяки зовнішньому пристрою. Коли на вхід *Trigger* (рис. 4.5) приходить пусковий імпульс, електронна схема сонара генерує короткий пакет коливань ультразвукової частоти, що надходить на випромінювач. Відбившись від твердої поверхні, коливання надходять на приймач і далі на підсилювач-формувач, на виході якого формується імпульс *Echo* (відлуння) з логічним рівнем. Зовнішній пристрій обчислює відстань до поверхні через інтервал часу між фронтами пускового імпульсу та відлуння за формулою

$$H = \frac{\Delta T \upsilon}{2} \cos\alpha, \tag{4.12}$$

де υ – швидкість звуку, ΔT – різницю часу між фронтами пуску та відлуння; α – кут відбиття хвилі (зазвичай від 0 до 30°).

Кут розсіюваня хвилі від генератора хвиль.

Кут розсіювання ультразвукової хвилі (рис. 4.6):

$$\alpha = \arcsin\left(\frac{0{,}51\lambda}{D}\right), \tag{4.13}$$

де D – діаметр ультразвукового генератора; λ – довжина хвилі.

Під час руху об’єкта частота відбитих хвиль не збігається з частотою випромінюваних хвиль (ефект Доплера). У класичній фізиці, де швидкості джерела та приймача відносно середовища нижчі за швидкість хвиль у середовищі, співвідношення між спостережуваною частотою f і частота випромінювання f_0 задається формулою

$$f = \left(\frac{c \pm \upsilon_r}{c \pm \upsilon_s}\right) f_0, \tag{4.14}$$

де c – швидкість поширення хвилі в середовищі; v_r – швидкість приймача відносно середовища (додається, якщо приймач рухається до джерела, віднімається, якщо приймач віддаляється від джерела); v_s – швидкість джерела відносно середовища (додається, якщо джерело віддаляється від приймача, віднімається, якщо джерело рухається до приймача).

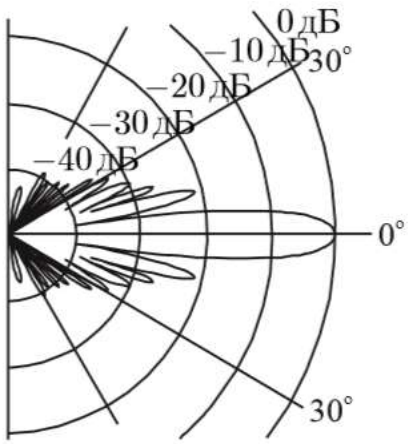
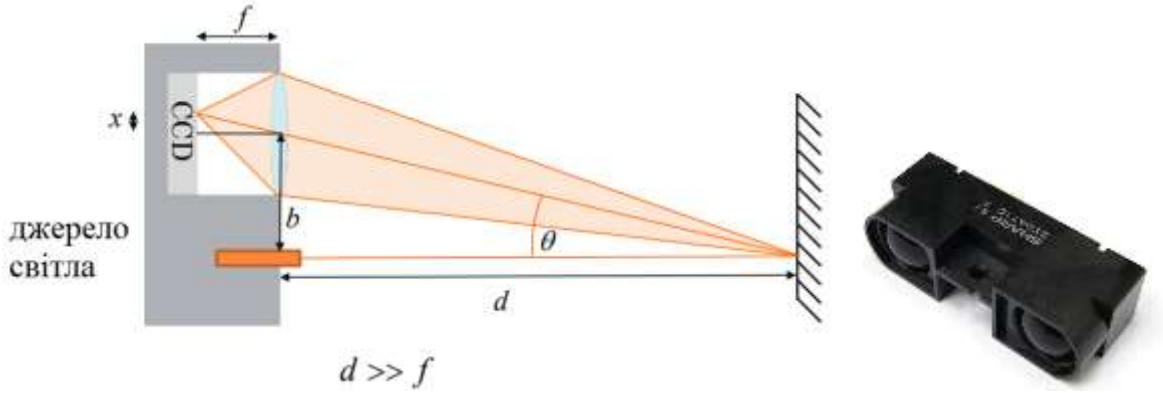


Рис. 4.6. Схема поширення ультразвукової хвилі

Більш складні сонари містять вбудований контролер, можуть самостійно виконувати процес вимірювання, фільтрацію та усереднення результатів вимірювання і видають готове значення відстані за протоколом I2C або SPI.

Ультразвукові сонари мають специфічні недоліки, а саме: вимірюють дистанцію з точністю до одного-двох сантиметрів, працюють на дуже обмеженій відстані до поверхні, яка відбиває сигнал, як правило, не більше ніж 3-4 м для цивільних систем, але є й спеціальні сонари з відстанню детектування до 12 м (Senscomp Mini-AE). Сонари не чутливі до зміни потоків повітря, проте важливою є якість поверхні, яка відбиває хвилі і може їх розсіювати. Також потрібно брати до уваги кут відбиття ультазвукової хвилі, який може значно спотворювати вимірювання реалного значення.

Інфрачервоний (IR) оптичний далекомір – це оптичний датчик, який використовує інфрачервоне випромінювання для вимірювання відстаней до об’єктів. Зазвичай застосовуть принцип триангуляції (рис. 4.7, *а*) для вимірювання кута та визначення розміру трикутника. Принцип роботи IR-далекоміра полягає в тому, що інфрачервоне світло випромінюється в напрямку об’єкта і частина цього світла відбивається від об’єкта та повертається до датчика. Знаючи кут між напрямком випромінювання і напрямком приймання світла, а також застосовуючи геометричні принципи триангуляції, можна визначити відстань до об’єкта.



а *б*

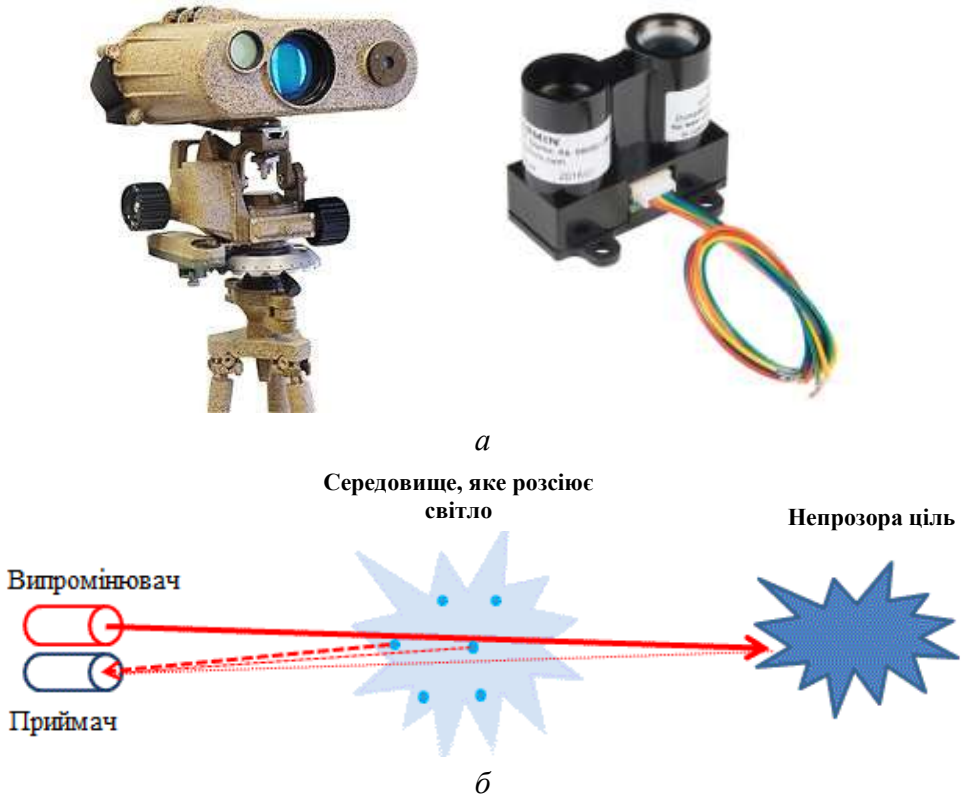
Рис. 4.7. Принцип оптичної триангуляції (*а*) та вигляд IR-датчика (*б*)

Згідно з рис. 4.7, *а* матимемо такі співвідношення:

$$\tan\theta = \frac{b}{d} = \frac{x}{f}. \qquad (4.15)$$

Знаючи параметри датчика *b*, *f*, *x*, визначають відстань *d*.

Лазерний далекомір – прилад, що вимірює відстані за допомогою лазерного променя (рис. 4.8). Подібні датчики застосовуються для вимірювання відстані, переміщення, зсуву, довжини, ширини, висоти, діаметра, товщини, прямолінійності, площинності, овальності, ексцентриситету, шорсткості, профілю деталей, відхилення від вертикальності тощо. Висока роздільна здатність і швидкодія дають змогу використовувати лазерні датчики для вимірювання биття та вібрації (наприклад, валів), оскільки вони фактично також є переміщеннями. При цьому одержана точність може досягати однієї тисячної частки загальної відстані.



а

б

Рис. 4.8. Вигляд лазерних датчиків (*а*) та принцип їхньої роботи (*б*)

Лазерний промінь може бути круглим або лінійним (у вигляді штриха). Розмір лазерної плями або переріз плоского променя становить приблизно 0,03...0,3 мкм.

Імпульсний лазерний далекомір складається з імпульсного лазера та детектора випромінювання і працює аналогічно до ультразвукового сонара. Вимірявши час, затрачений імпульсом на шлях до перешкоди і назад, помноживши його на швидкість світла, дізнаємось відстань від лазера до перешкоди:

$$H = \frac{c\Delta T}{2n}, \qquad (4.16)$$

де H – відстань до перешкоди; c – швидкість світла; n – показник заломлення середовища; ΔT – час проходження імпульсу до перешкоди і назад.

Точність вимірювання відстані визначається точністю вимірювання часу (за відстані до цілі 1 м час відгуку становитиме 6,7 нс, для відстані 10 м час відгуку буде 67 нс, для 100 м – 0,67 мкс, для 1 км – 6,7 мкс). Отже, що коротший фронт імпульсу, то краще для точності.

Фазовий лазерний далекомір вимірює відстані на основі порівняння фаз посланого і відбитого променів. У нього більша точність порівняно з імпульсним далекоміром. Крім того, він дешевший у виробництві, а тому більш поширений. За цим методом лазер виромінює постійний промінь заданої амплітуди та частоти (зазвичай це частоти, менші за 500 МГц). Довжина хвилі лазера залишається незмінною і дорівнює 500 – 1100 нм. Відбите від об’єкта випромінювання приймає фотоприймач, а його фаза порівнюється з фазою опорного сигналу – від лазера (рис. 4.9).

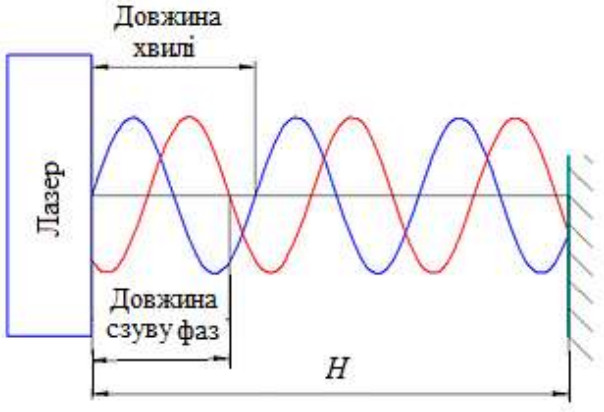


Рис. 4.9. Принцип роботи фазового лазерного далекоміра

Наявність затримки під час поширення хвилі створює зсув фаз, який вимірюється далекоміром. Відстань визначають за формулою

$$H = \frac{c}{2f}\frac{\varphi}{2\pi}, \tag{4.17}$$

де c – швидкість світла; f – частота модуляції лазера; φ – зсув фаз.

Наведена залежність справедлива, тільки якщо відстань до об’єкта менша за половину довжини хвилі модулювального сигналу, яка дорівнює $1/2f$. Якщо частота модуляції дорівнює 10 МГц, то вимірювана відстань може доходити до 15 метрів, а за зміни відстані від 0 до 15 м різниця фаз буде змінюватися від 0° до 360°. Зміна зсуву фаз на 1° у

такому разі означає переміщення об’єкта приблизно на 4 см. Унаслідок перевищення цієї відстані виникає неоднозначність – неможливо визначити, скільки періодів хвилі вкладається у вимірюваній відстані. Для вирішення неоднозначності частоту модуляції лазера перемикають, після чого розв’язують складену систему рівнянь. Найпростіший випадок – використання двох частот: на низькій приблизно визначають відстань до об’єкта (але максимальна відстань все одно обмежена), на високій – відстань з потрібною точністю. За однакової точності вимірювання фазового зсуву і високої частоти точність вимірювання відстані буде помітно вищою.

Нині використовують три основних способи лазерних вимірювань: оптичної тріангуляції, лазерного сканування («відтінки») та радарний. Під час вимірювання відстані за допомогою лазерної оптичної тріангуляції сфокусований лазерний промінь проходить крізь нерухому напівпрозору площину відліку, частково відбиваючись від неї на приймальну лінзу та лінійний світлочутливий елемент (детектор) (рис. 4.10).

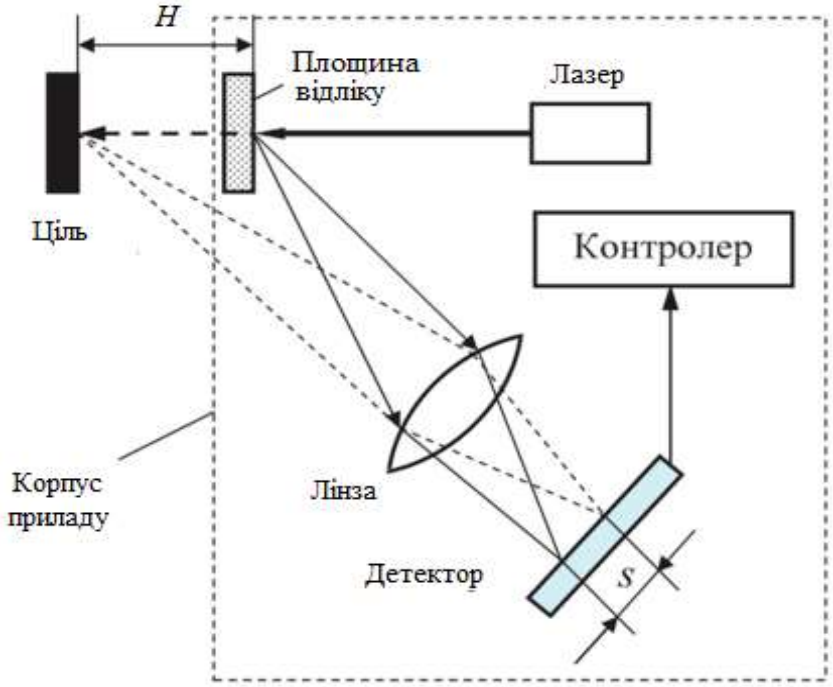
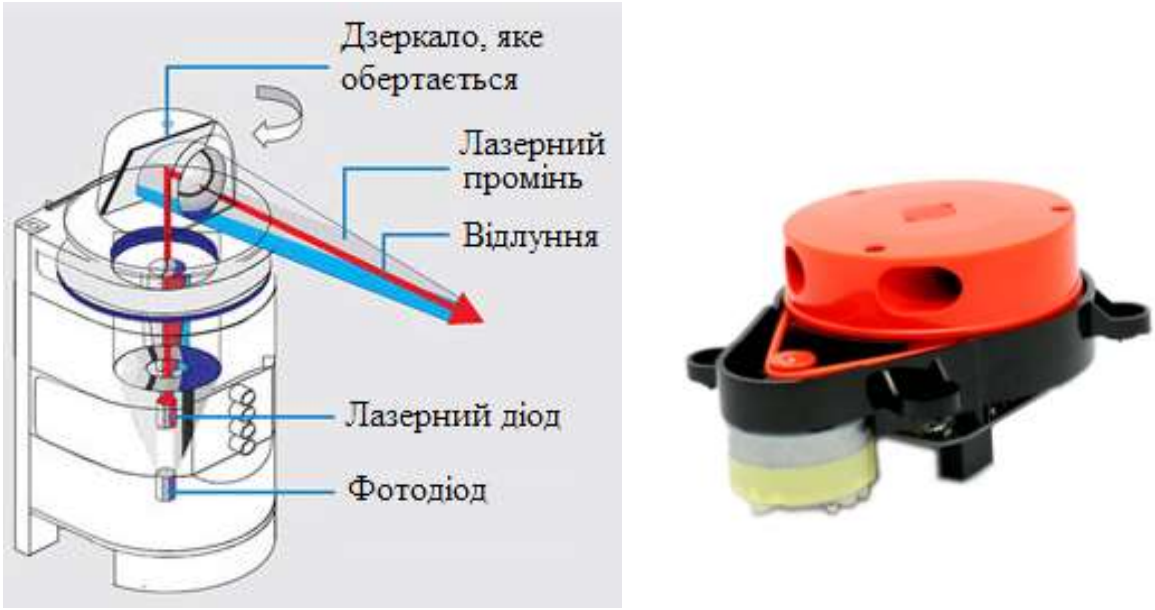


Рис. 4.10. Принцип роботи оптичної лазерної тріангуляції

У цьому методі використано здатність лазерного променя поширюватися з малим розсіянням на великі відстані. Зазвичай промінь лише освітлює точку, відстань до якої треба виміряти. Таким чином, лазер застосовується фактично лише як покажчик. Відбиття від цієї точки (розсіяне або дзеркальне) контролюються детектором,

встановленим на деякій відстані від лазерного променя, на якому джерело лазерного випромінювання, об’єкт та детектор утворюють трикутник. На детекторі лінза фокусує відбите світло, а положення світлової плями на ньому вказує напрямок вхідного світла, тобто кут між лазерним променем і поверненим світлом, звідки відстань і може бути розрахована. В ролі детекторів зазвичай застосовують напівпровідникові сегментні детектори та матриці. Удосконалена оптична система, що дає малі розміри плями проєкції, разом з високою роздільною здатністю детектора дають змогу в умовах машинобудівного виробництва вимірювати лінійні переміщення та відстані з точністю близько 0,02 мкм.

Лідар – це пристрій для сканування й обробки інформації про віддалені об’єкти за допомогою активних оптичних систем, у яких використано явища відбиття світла і його розсіювання в прозорих і напівпрозорих середовищах. За своєю суттю лідаром є лазерний датчик відстані, який може працювати в різних площинах. Для цього лазерний датчик переміщують в просторі або обертають навколо деякої осі (рис. 4.11), внаслідок чого отримують набір точок. Що більша кількість таких точок, то якісніше зображення вдається отримати (рис. 4.12).



а *б*

Рис. 4.11. Принцип роботи (*а*) та зовнішній вигляд роторного лідара (*б*)

Сканувальні лідари в системах машинного зору формують двовимірну або тривимірну картину навколишнього простору.

«Атмосферні» лідари здатні не тільки визначати відстань до непрозорих цілей, що відбивають світло, а й аналізувати властивості прозорого середовища, що розсіює світло. Різновидом атмосферних лідарів є доплерівські лідари, що визначають напрямок і швидкість переміщення повітряних потоків в різних шарах атмосфери.

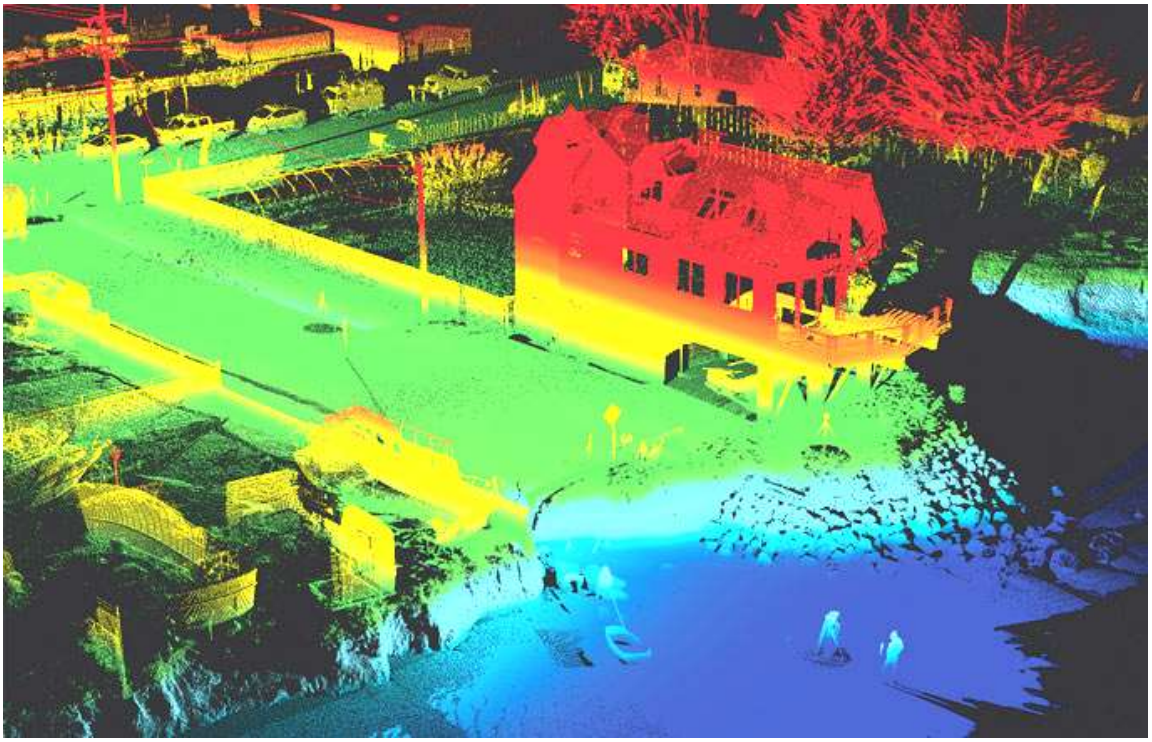
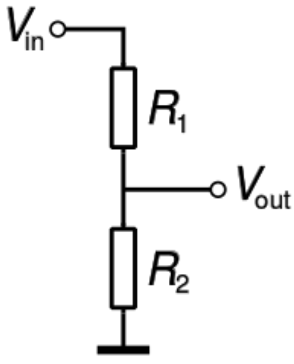


Рис. 4.12. Зображення, отримане з лазерного лідара

Подільник напруги – лінійна електронна схема, напруга на виході якої (V_{out}) становить частину напруги на вході (V_{in}). Найпростіший ділильник напруги складається з двох послідовно увімкнених резисторів. Згідно із законом Ома зв'язок між вхідною і вихідною напругою визначають за формулою

$$V_{out} = \frac{R_2}{R_1 + R_2} V_{in}, \quad (4.18)$$

де R_1, R_2 – опір провідників за схемою на рис. 4.13.



а

б

Рис. 4.13. Схема ділильника напруги (*а*) та вигляд змінного резистора (потенціометра) (*б*)

Подільники напруги застосовують у схемах під'єднання аналогових

датчиків, наприклад, фоторезисторів та потенціометрів.

Енкодер (перетворювач кутових переміщень) – це електронний пристрій, який дає змогу з достатньою точністю вимірювати різні параметри обертання, переважно вала електродвигуна або редуктора. Вимірюваними параметрами можуть бути: швидкість обертання, кутове положення відносно нульової мітки, напрямок обертання. Застосовують два види енкодерів – інкрементальний та абсолютний (рис. 4.14). Інкрементальний енкодер конструктивно простіший за абсолютний, використовують його для вимірювання кількості кроків переміщення. Цей пристрій містить диск з прорізами, що просвічуються оптичним датчиком (рис. 4.14, *а*), або диск із зубцями, які контактують з вимикачем (рис. 4.14, *б*), або ж магнітний вал/диск, який обертається поряд з датчиком Гола (рис. 4.14, *в*). Під час обертання, наприклад, диска оптичного енкодера відбувається перекриття прорізами на диску світлових променів всередині енкодера залежно від положення прорізу. В результаті на виході енкодера формується послідовність дискретних імпульсів (рис. 4.15), частота яких залежить від роздільної здатності пристрою та частоти його обертання.

а *б* *в*

Рис. 4.14. Різновиди інкрементальних енкодерів: *а* – оптичний; *б* – контактний; *в* – магнітний

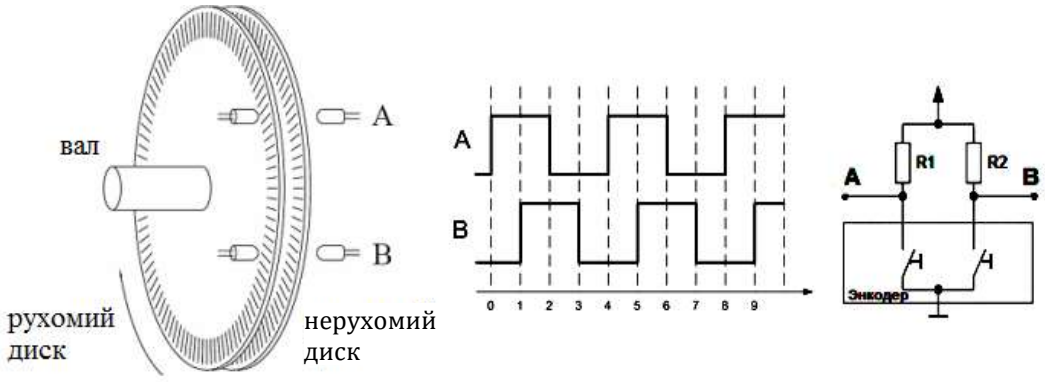


Рис. 4.15. Принцип роботи інкрементного енкодера

Абсолютний енкодер дає змогу не лише фіксувати кількість обертів вала, але й фіксувати його положення. Для абсолютного енкодера не потрібне калібрування після кожного циклу, а значення положення можна отримувати у реальному часі. Абсолютні енкодери генерують цифровий сигнал для кожної окремої позиції у заданому діапазоні (рис. 4.16). Абсолютні енкодери бувають оптичними, магнітними і ємнісними.

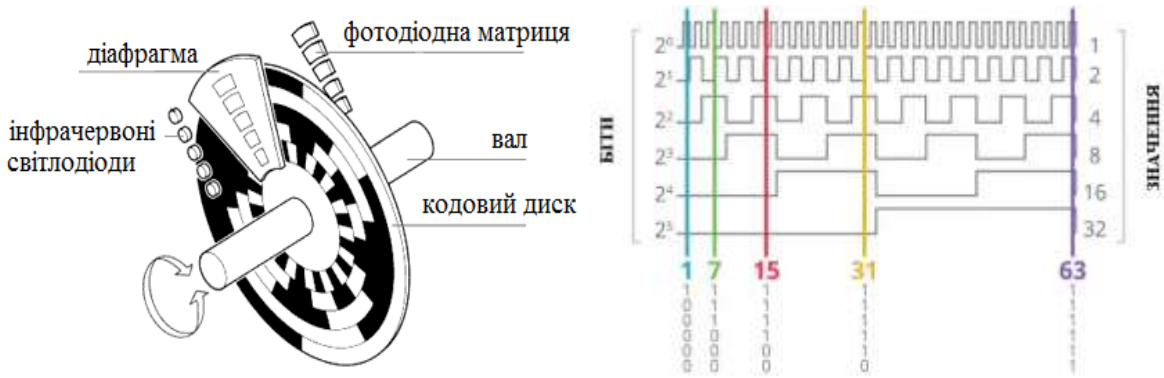


Рис. 4.16. Принцип роботи абсолютного енкодера

Вимірювальна система абсолютного оптичного енкодера складається з таких основних компонентів: поворотний вал на підшипниках, кодовий оптичний диск на валу, інфрачервоний світлодіод (джерело світла), оптоелектронна (фототранзисторна) матриця, система обробки сигналу. В багатоповоротний енкодер додатково вбудовують редуктор з кількох кодових оптичних дисків зі світлодіодами та матрицями.

Інфрачервоні промені світлодіода просвічують кодовий диск і попадають на фототранзисторну матрицю, розміщену зі зворотного боку кодового диска. На кожному кроці кутового положення кодового диска його темні ділянки перешкоджають попаданню світла на різні фототранзистори. Електричні сигнали в подальшому перетворюються електронікою енкодера в двійковий (бінарний) код. Бінарні коди складаються тільки з двох символьних станів, наприклад, чорний або білий, світлий або темний тощо. Бінарний код у цифровій техніці – це спосіб представлення даних (чисел, слів та ін.) як комбінації двох знаків, які можна позначити як 0 і 1. Знаки, або одиниці бінарного коду, називають бітами. Бінарні коди є простими та надійними для накопичення інформації у вигляді комбінації всього двох фізичних станів, наприклад, у вигляді зміни або сталості світлового потоку під час зчитування з оптичного кодового диска.

Тензометричний датчик – датчик, що перетворює величину деформації в зручний для вимірювання сигнал (зазвичай електричний), основним компонентом якого є тензорезистор. Тензорезистор – резистор, опір якого змінюється залежно від його деформації. Унаслідок розтягування провідних елементів тензорезистора збільшується їхня довжина та зменшується поперечний переріз, що збільшує опір тензорезистора, у разі стискання – зменшує.

Чутливість тензорезистора характеризується безрозмірним параметром – коефіцієнтом тензочутливості, який визначають як

$$K_f = \frac{\Delta R / R_0}{\varepsilon}, \tag{4.19}$$

де ΔR – абсолютна зміна опору, пропорційного деформації, Ом; R_0 – початковий опір недеформованого тензорезистора, Ом; $\varepsilon = \Delta l / l$ – відносна деформація, Δl, l – відповідно приріст довжини та початкова довжина провідника.

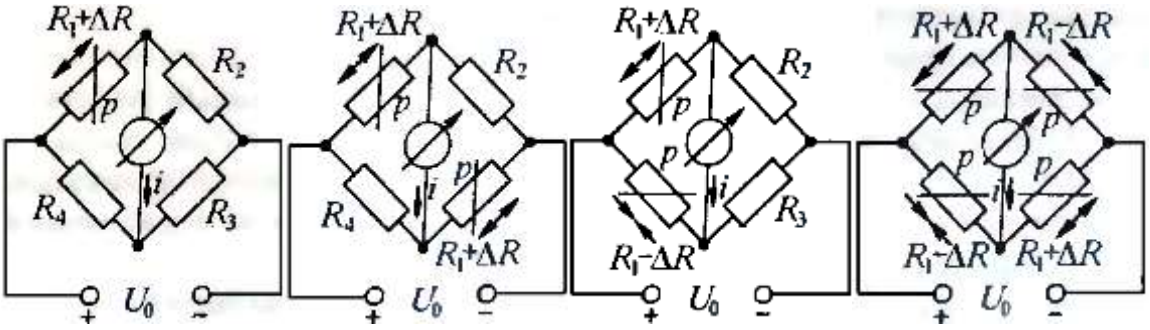
Серійні тензорезистори мають опір від 30 Ом до 3 кОм за типових значень 120 Ом, 200 Ом, 350 Ом та 1 кОм. На рис. 4.17 наведено будову дротового та фольгових тензорезисторів. Фольгові датчики мають товщину провідного покриття 3…15 мкм, а їхній опір становить від 30 до 300 Ом.

а *б* *в* *г*

Рис. 4.17. Будова тензорезисторів: *а* – дротовий тензодатчик; *б* – фольговий тензорезистор для вимірювання лінійних переміщень; *в* – фольговий тензорезистор для вимірювання деформацій у двох взаємно перпендикулярних напрямках; *г* – фольговий тензорезистор для вимірювання тиску

Для того щоб одержати найбільшу зміну опору датчика, його треба розмістити в напрямку дії деформувального зусилля (стиск чи розтяг), тобто напрямок вимірювальної бази повинен збігатися з віссю, по якій спрямоване зусилля. Якщо напрямок бази і зусилля взаємно перпендикулярні, тоді деформація і зміна опору дуже малі, отже, зміна опору тензорезистора буде незначною. Наприклад, відносна зміна опору,

спричинена відносним розтягуванням на 0,0005 за тензорезистивного коефіцієнта 2, становитиме 0,1%, що для тензодатчика опором 120 Ом еквівалентне зміні опору всього лише на 0,12 Ом. Щоб вимірювати настільки малу зміну опору, тензорезистори вмикають за зрівноважувальною мостовою схемою (міст Вітстона, рис. 4.18). Якщо розмістити декілька датчиків під кутом один до одного, то можна визначити не тільки величину деформації, а й напрямок прикладених до деталі зусиль.



а *б* *в* *г*

Рис. 4.18. Схеми вмикання тензорезисторів: *а* – мостова схема з одним тензорезистором; *б* та *в* – мостові схеми з компенсацією температурної похибки; *г* – мостова схема ввімкнення тензодатчиків підвищеної чутливості (міст Вітстона) (тензорезистори на схемі зображно у вигляді змінних резисторів)

Вихідна напруга вимірювального моста Вітстона дорівнюватиме:

$$U_0 = \left[\frac{R_3}{R_3 + R_4} - \frac{R_2}{R_1 + R_2} \right] U . \qquad (4.20)$$

Для живлення моста зазвичай застосовують блоки живленя зі стабілізованою напругою 3 В та 10 В. Струм через тензодатчики не повинен перевищувати значень від 2 мА до 30 мА для датчиків з опором від 1 кОм до 120 Ом. Напруга живлення моста має бути такою, щоб збільшити відношення сигналу до шуму та мінімізувати похибку, спричинену нагріванням датчика.

Візуальні датчики потрібні для отримання інформації про геометричні та фізичні характеристики зовнішнього середовища на підставі аналізу освітленості в оптичному діапазоні (інфрачервоному, рентгенівському тощо).

Сукупність датчиків, підсилювачів, комутаторів та пристроїв памяті утворюють *робототехнічну інформаційну систему* (рис. 4.19),

призначену для збору сигналів, їхнього попереднього підсилення та перетворення в цифрову форму для передавання на комп'ютер керування, де виконується інтегральна оцінка вимірювального процесу.

На рис. 4.20 показано вигляд структурної схеми робота, який може адаптуватися до зміни зовнішніх умов, а інформаційна система такого робота складається з підсистем сприйняття середовища, зв'язку, планування та моделі робочого середовища.

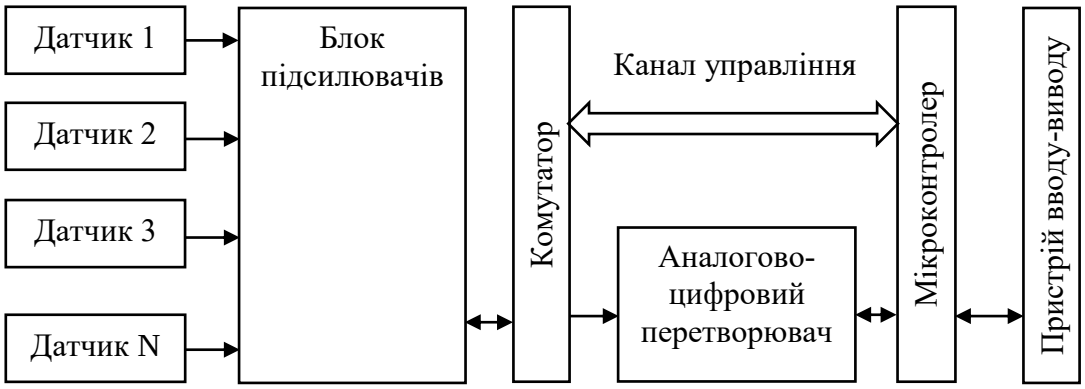


Рис. 4.19. Схема типової інформаційної системи робота

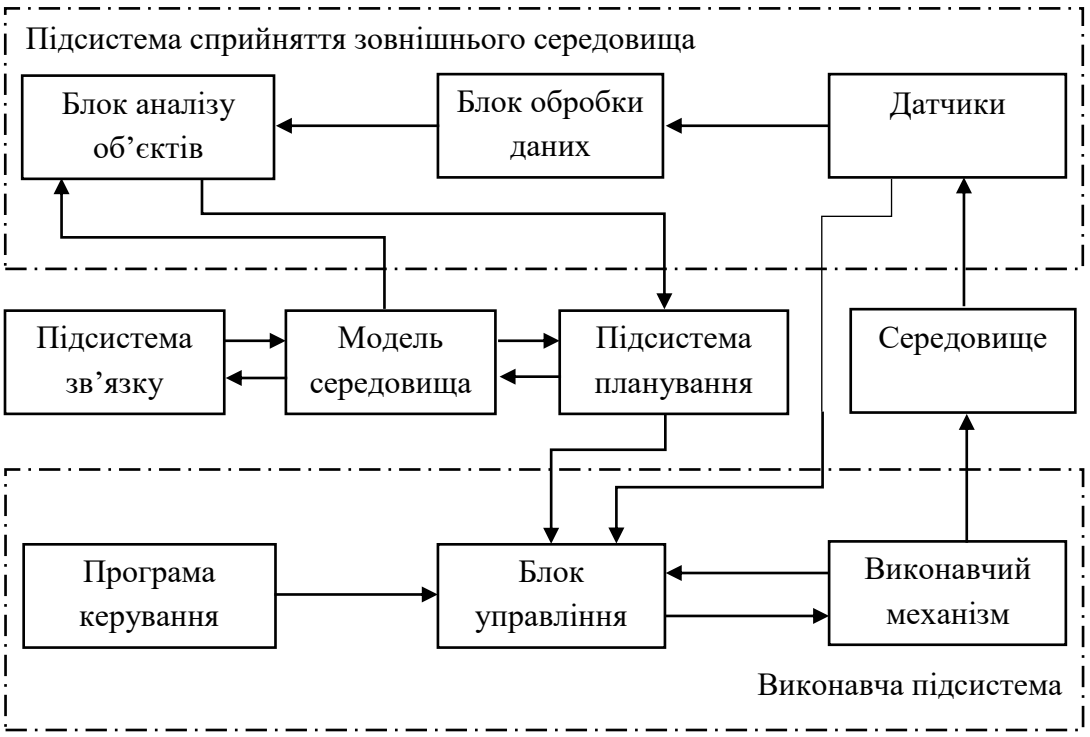


Рис. 4.20. Структурна схема адаптивного робота

Сигнал є засобом передавання інформації в просторі з плином часу. Сигнал описує стан матеріального об'єкта, а для утворення сигналів можуть бути використані лише такі об'єкти, стани яких мають достатню стійкість відносно зміни часу або положень в просторі. З точки зору стійкості всі сигнали можна розділити на два класи.

До *першого класу* належать сигнали, в ролі яких використовують стійкі, стабільні стани фізичних систем. Прикладами сигналів такого типу можуть бути: текст, фотографія, стан плівки магнітофона, стан феритової матриці пам'яті електронної обчислювальної машини, стан регістру (системи тригерів) обчислювальної машини тощо. Такі сигнали називають статичними.

Другий клас об'єднує сигнали, які використовують динамічні стани силових полів. Відмінність від інших матеріальних систем поля полягає в тому, що зміна їхнього стану не може бути локалізована в частині поля і призводить до поширення збурення. У разі поширення збурення в полі, параметри конфігурації, будова збурення мають певну стійкість, що дає змогу використовувати такі стани полів як сигнали. Прикладами таких сигналів можуть бути звукові сигнали (зміна стану поля сил пружності в газі, рідині або твердому тілі), світлові і радіосигнали (зміни стану електромагнітного поля). Такі сигнали називають динамічними.

Динамічні сигнали використовують переважно для передавання, а статичні – для зберігання інформації, проте динамічні сигнали також можуть використовуватися для збереження інформації, наприклад, в пристроях памяті на мультизвукових лініях затримки електронних цифрових обчислювальних машин, а статичні сигнали, такі як газети та листи, більшою мірою призначені для передавання, ніж для зберігання інформації.

Сигнал завжди є функцією часу. Залежно від того, яких значень може набувати аргумент (час t) і рівні сигналів, їх поділяють на чотири типи (рис. 4.21).

Безперервні, або аналогові, сигнали (рис. 4.21, *а*) визначені для всіх моментів часу, вони можуть набувати всіх значень із заданого діапазону. Найчастіше фізичні процеси, які породжують сигнали, є безперервними. Цим пояснюється друга назва сигналів цього типу – аналоговий, тобто аналогічний тим процесам, які його створили (випадкові сигнали цього типу називаються безперервними випадковими процесами).

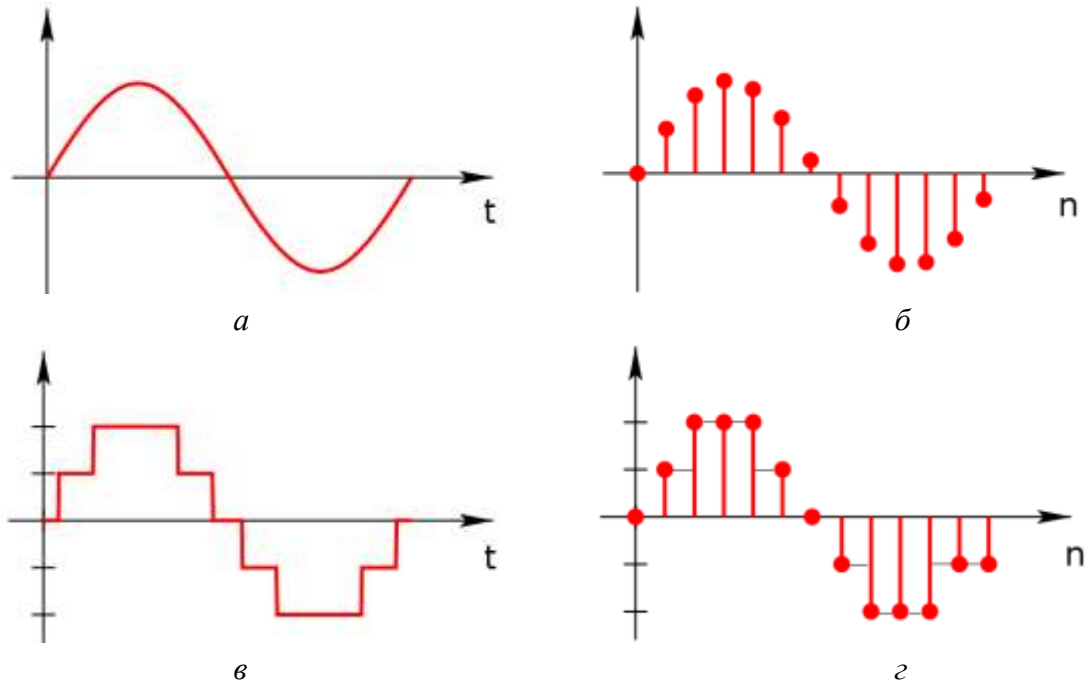


Рис. 4.21. Види сигналів: *а* – аналоговий безперервний; *б* – дискретнизований не квантований; *в* – квантований безперервний; *г* – квантований дискретний

Дискретизовані, або дискретно-безперервні, сигнали (рис. 4.21, *б*) визначають лише в деякі моменти часу, вони можуть набувати будь-яких значень рівня. Інтервал часу Δt між сусідніми відліками називається кроком дискретизації. Часто такі сигнали називають дискретними за часом (випадкові сигнали цього типу називають процесами з дискретним часом або неперервними випадковими послідовностями).

Дискретні за рівнем, або квантовані, сигнали (рис. 4.21, *в*) визначають для кожного моменту часу, проте сигнал набуває лише дозволених значень рівнів, вони відокремлені один від одного на величину квантування $\Delta x = x_{k+1} + x_k$ (випадкові сигнали цього типу називають дискретними випадковими процесами). Процедура перетворення сигналу, коли відбувається дискретизація його за рівнем або за часом або одночасно за рівнем і часоом, називається квантуванням.

Квантування часу – перетворення сигналу в послідовність імпульсів, амплітуда, тривалість або частота яких залежать від амплітуди вхідного сигналу. Квантування за рівнем полягає в перетворенні миттєвого значення сигналу до деякої найближчої, наперед заданої і фіксованої величини, яку називають рівнем.

Сигнал, до якого застосована і дискретизація і квантування, називається цифровим.

Дискретні за рівнем і часом сигнали (рис. 4.21, *г*), визначені в деяки дозволені моменти часу, можуть набувати лише дозволених значень рівнів (випадкові сигнали цього типу називають дискретними випадковими послідовностями). Перетворення дискретної інформації одним з способів: шифрування, стиснення, захист від шуму, – називають кодуванням. Коди розділяють на двійкові та недвійкові. Унаслідок збільшення кількості розрядів сигналу кодування збільшується кількість значень, які можна закодувати. Наприклад, кількість значень для двійкових кодів обчислюється за формулою

$$N = 2^m, \tag{4.21}$$

де N – кількість незалежних значень, які можна закодувати; m – кількість розрядів двійкового кодування (наприклад, число 101 має три розряди).

Для представлення чисел у пам'яті комп'ютера використовують двійкову систему числення, в якій є лише дві цифри: «0» та «1», тобто два стійкі стани фізичних елементів (двійковий запис «111» є відповідним числу $1\cdot2^2+1\cdot2^1+1\cdot2^0 = 4+2+1 = 7$ у десятковій системі). Вісімковий код, відомий як октальний код, має основу 8, оскільки використовує вісім різних цифр від «0» до «7» включно (вісімковий запис «34» є відповідним числу $3\cdot8^1+4\cdot8^0 = 24+4 = 28$ у десятковій системі числення). Шістнадцятковий формат (HEX) має основу з числом 16 і використовує цифри від «0» до «9» та літери від «A» до «F». Кожна цифра в HEX відповідна 4-бітовому рядку в двійковій системі. Таким чином, один байт (8 біт) може бути представлений двома символами у HEX форматі, що дуже зручно. Наприклад, представлення числа 255 з десяткової системи буде FF у системі HEX через те, що 15 у десят-ковій системі є відповідним **F** у HEX, а дві цифри **FF** утворюють 255.

Часто в цифровій техніці потрібно, щоб в числовому коді під час переходу від одного числа до наступного, змінювалася б тільки одна цифра. Дотримання цієї умови потрібне, коли внаслідок технічних неточностей момент перемикання не може бути точно дотриманий, оскільки під час перемикання двох чисел можуть виникнути неправильні комутаційні операції. Як приклад подібної помилки розглянемо перемикання від 0001 до 0010 в двійковому коді (рис. 4.22). За цього перемикання змінюються 0 і 1 біти за одночасного перемикання

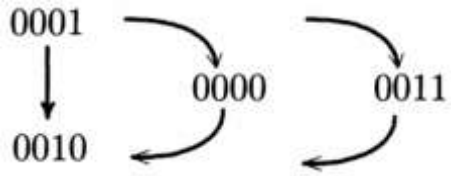


Рис. 4.22. Приклад перемикання бітів

00 → (бінарне 00)
01→ (бінарне 01)
11 → (бінарне 10)
10 → (бінарне 11)

Рис. 4.23. Приклад коду Грея для 3-бітного числа

безпосередньо та досягається нове число. Якщо спочатку змінюється біт 0, то з'являється число 0000 і тільки коли змінюється біт 1, отримуємо правильне число 0010. Та якщо спочатку змінюється біт 1, а потім змінюється біт 0, то в проміжку виникає число 0011, що *може бути неприпустимим для* механічних систем систем відліку. Коди Грея дають змогу уникнути цієї значної помилки завдяки тому, що внаслідок переходу від одного кодового слова до наступного змінюється тільки один розряд (рис. 4.23).

Якщо будь-який сигнал s(k) за допомогою математичних алгоритмів перетворюється в деякий інший сигнал $s_1(k)$, який має потрібні властивості, процес перетворення називається фільтрацією, а пристрій, що виконує фільтрацію, фільтром. Оскільки значення сигналів надходять з постійною швидкістю F_d, фільтр повинен встигати обробляти поточний сигнал до надходження наступного (частіше – до надходження наступних n відліків, де n – затримка фільтра), тобто обробляти сигнал в реальному часі. Для обробки сигналів (фільтрації) в реальному часі застосовують спеціальні обчислювальні пристрої – цифрові сигнальні процесори. Це стосується не лише безперервних сигналів, але і перервчастих, а також сигналів, записаних на пристрої зберігання інформації. Розрізняють методи обробки сигналів у часовій і в частотній області. Еквівалентність частотно-часових перетворень однозначно визначається через перетворення Фур'є.

Фільтри бувають:

- безперервними, дискретними, лінійними і нелінійними;
- електричні, механічні, акустичні та ін.

Основне завдання фільтрації:

- *лінійна фільтрація* – селекція сигналів частотної області; синтез фільтрів, узгоджених сигналами; частотне розділення каналів; цифрові перетворювачі Гільберта і диференціатор; коректорних характеристик каналів;

- *спектральний аналіз* – обробка мовних, звукових, сейсмічних, гідроакустичних сигналів; розпізнавання образів;
- *частотно-часовий аналіз* – компресія зображень, гідро- і радіолокація, різноманітні завдання з виявлення;
- *адаптивна фільтрація* – обробка мови, зображень, розпізнавання образів, придушення шумів, адаптивні антенні решітки;
- *нелінійна обробка* – обчислення кореляцій, медіанна фільтрація; синтеза амплітудних, фазових, частотних детекторів, обробка мови, векторне кодування;
- *багатошвидкісна обробка* – інтерполяція (збільшення) і десимація (зменшення) частоти дискретизації в багатьох швидкісних системах телекомунікації, аудіосистемах.

Фільтр Калмана – рекурсивний фільтр, що оцінює вектор стану динамічної системи, використовуючи ряд неповних та зашумлених вимірювань. Алгоритм фільтра Калмана працює у два етапи. На етапі прогнозування фільтр екстраполює значення змінних станів, а також їхньої невизначеності. На другому етапі за даними вимірювання (отриманих з деякою похибкою) результат екстраполяції уточнюється. Завдяки покроковій природі алгоритму цей фільтр може у реальному часі відстежувати стан об’єкта, використовуючи лише поточні виміри та інформацію про попередній стан і його невизначеність. Істинний стан системи у момент k визначається за істинним станом в момент $k-1$ відповідно до рівняння

$$x_k = F_k x_{k-1} + B_k u_k + w_k, \qquad (4.22)$$

де F_k – матриця еволюції системи (процесу), яка впливає на вектор x_{k-1} (вектор стану в момент k-1); B_k – матриця управління, прикладена до вектора керівних впливів u_k; w_k – нормальний випадковий процес з нульовим математичним очікуванням і коваріаційною матрицею Q_k, який описує випадковий характер еволюції системи (процесу):
$w_k \sim N(0, Q_k)$.

В момент k відбувається спостереження (вимірювання) z_k істинного вектора стану x_k, які пов’язані між собою рівнянням

$$z_k = H_k x_k + v_k, \qquad (4.23)$$

де H_k – матриця вимірювань, що пов’язує істинний вектор стану і вектор виконаних вимірювань, v_k – гаусівський шум вимірів з нульовим математичним очікуванням і матриці коварії R_k: $v_k \sim N(0, R_k)$.

Початковий стан та вектори випадкових процесів на кожному такті $\{x_0, w_1, \ldots, w_k, v_1, \ldots, v_k\}$ вважають незалежними.

Контрольні запитання

1. У чому полягає призначення інформаційної системи робота?
2. Що таке датчик?
3. Чи є принципова різниця між датчиком та сенсором?
4. Як працює ультразвуковий датчик відстані?
5. Поясніть суть закону Ома.
6. Що таке ділильник напруги?
7. Що таке енкодер?
8. Поясніть принцип роботи інкрементного енкодера.
9. Поясніть принцип роботи цифрового енкодера.
10. Що являє собою матричний фотодатчик?
11. З якою метою застосовують лазерні датчики?
12. Що таке лідар?
13. Чи можна за допомогою лазерного датчика отримати зображення обєкта?
14. Поясніть принцип роботи тензодатчика.
15. Які види сигналів вам відомі?
16. Чим відрізняються цифрові сигнали від аналогових?

Лекція № 5. Системи приводу, навігації та управління роботів

У конструкціях роботів застосовують двигуни внутрішнього згоряння, колекторні або безколекторні (вентильні) електродвигуни.

Безколекторні електродвигуни (Brush Les Direct Current Motor) – електричні двигуни постійного струму, які мають кращі характеристики порівнянно з колекторними та більшу довговічність через те, що вони не мають вузлів (щіток) тертя, за допомогою яких передається струм. Так само, як й у звичайного електродвигуна, у якого є центральна рухома частина – ротор і зовнішня нерухома – статор, у безколекторних двигунів можуть бути контрукції з рухомою частиною у вигляді зовнішнього корпуса з постійними магнітами та нерухомою – центральним ротором з обмотками трьох фаз. Напруга живлення

обмоток безконтактного двигуна формується електронною системою залежно від положення ротора (рис. 5.1). В двигунах постійного струму для цієї мети використовується механічний колектор, а у вентильному двигуні – напівпровідниковий комутатор (*ESC*). Для визначення положення ротора можна застосовувати безконтактний датчик, проте є контрукції, у яких роль датчика виконує одна з обмоток, наприклад, у FPV-квадрокоптерах з BLDC електродвигунами не застосовують датчики. Для того щоб змусити обертатися таку систему, треба в певному порядку змінювати напрямок магнітного поля в обмотках ротора, тоді постійні магніти статора будуть взаємодіяти з магнітними полями ротора і рухомий статор почне рухатися. Цей рух побудовано на властивості магнітів з однойменними полюсами відштовхуватися, а з протилежними – притягуватися. На практиці безколекторні двигуни нагріваються менше, ніж колекторні, та створюють більший крутний момент. Завдяки використанню в конструкції безколекторного двигуна потужних неодимових магнітів такі двигуни компактніші. Конструкція безколекторного двигуна дає змогу експлуатувати його у воді та агресивному середовищі. Безколекторні двигуни створюють менше радіоперешкод, ніж колекторні.

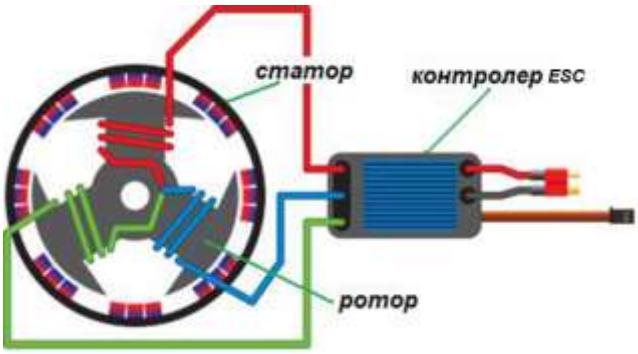


Рис. 5.1 – Схема комутації вентильного (BLDC) електродвигуна

Безколекторні двигуни бувають одно-, дво-, три- і більше фазними. Що більше фаз, то більш плавне обертання магнітного поля й складніша система управління двигуном. Трифазна система найбільш оптимальна за співвідношенням ефективність/складність, тому найбільш поширена. Фактично фази – це обмотки двигуна. Три обмотки з'єднують за схемою «зірка» або «трикутник». Трифазний безколекторний двигун має три виходи, а двигун з датчиками – п'ять (два дроти – живлення датчиків положення та три дроти – сигнали від датчиків). В трифазній системі у кожний момент часу напруга подається на дві з трьох обмоток, таким чином, за один повний оберт можна зробити шість фаз подачі постійної напруги на обмотки двигуна (рис. 5.2).

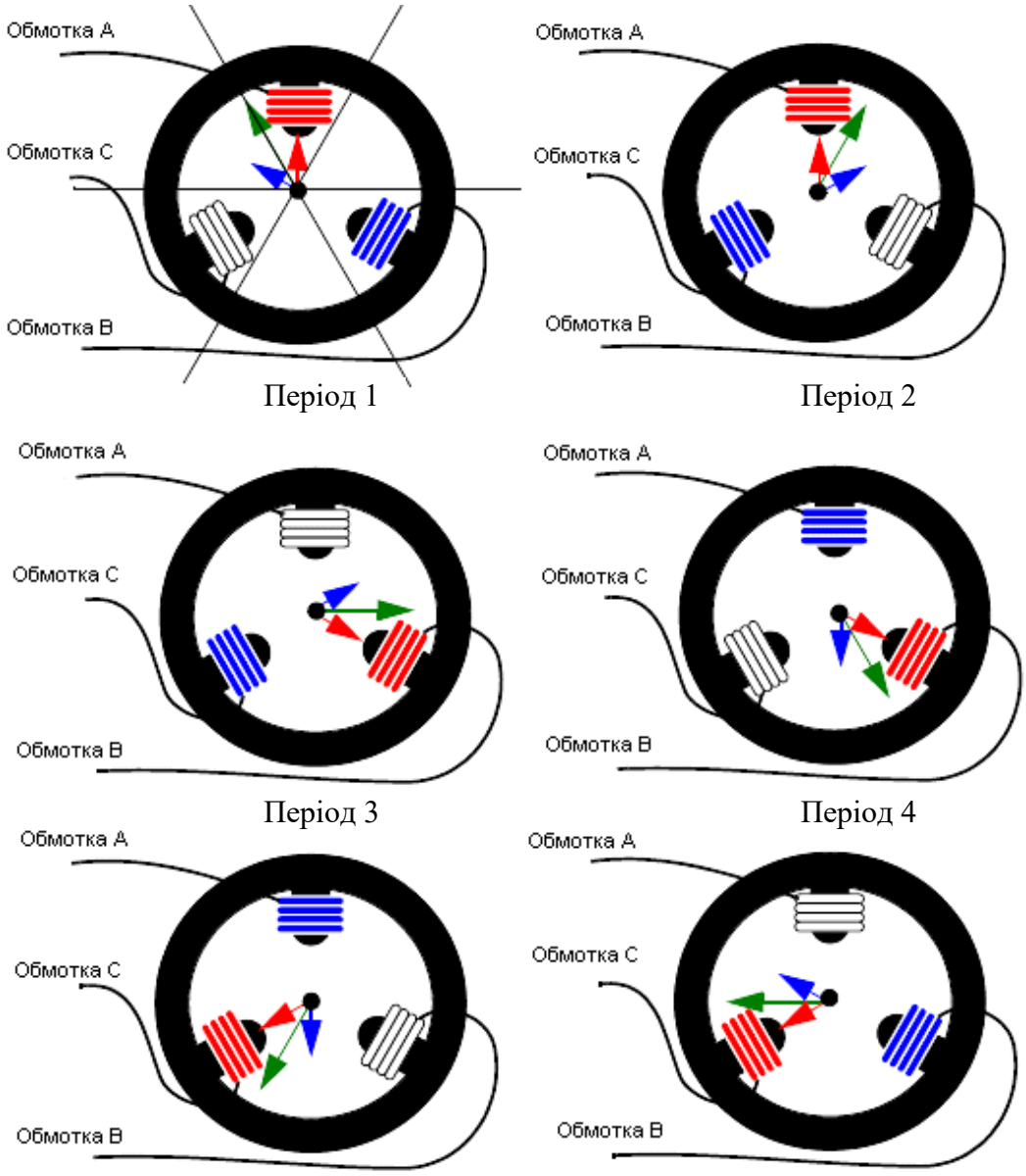


Рис. 5.2. Схеми роботи безколекторного (вентильного) трифазного двигуна

Безколекторні двигуни можуть бути з вмонтованими датчиками положення ротора. Алгоритм керування такими двигунами значно простіший, однак потрібно забезпечити живлення датчиків та розміщення провідників від датчиків в двигуні до електроніки. В разі виходу з ладу одного з датчиків, двигун припиняє роботу, а заміна датчиків зазвичай потребує розбирання двигуна. У випадку, коли конструктивно неможливо розмістити датчики в корпусі двигуна, використовують двигуни без датчиків. Конструктивно такі двигуни практично не відрізняються від двигунів з датчиками, проте електронний блок, який управляє двигуном без датчиків при цьому повинен чітко відповідати характеристикам конкретної моделі двигуна.

Якщо двигун повинен стартувати із суттєвим навантаженням на валу двигуна (електротранспорт, підйомні механізми тощо) – застосовують двигуни з датчиками. Двигун без датчиків положення повинен стартувати без навантаження на валу. В момент старту двигуна без датчиків можливе коливання осі двигуна у різні боки.

Сервопривід (*серводвигун, сервомеханізм*) – це електромеханічний керований привід систем автоматичного або дистанційного керування, який відповідно до заданого сигналу управління здійснює кероване переміщення органа управління. Сервопривід (рис. 5.3) працює від імпульсів змінної тривалості (PWM/ШІМ), які отримує через сигнальний канал і складається з електродвигуна, плати управління, редуктора та датчика повороту. Якщо тривалість імпульсів становить близько $1{,}5 \cdot 10^{-3}$ с, сервопривід перебуває в нейтральному положенні (тобто у нього однаковий потенціал обертання в обидвох напрямках).

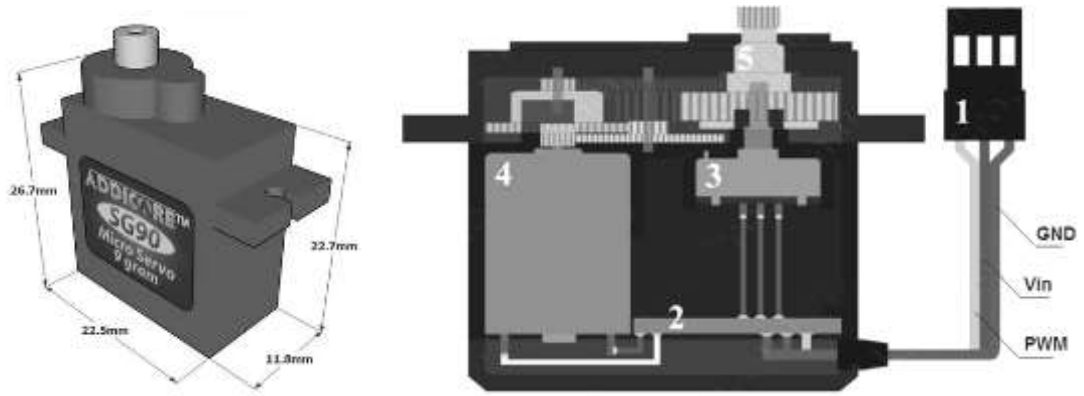


Рис. 5.3. Конструкція лабораторного сервоприводу: *1* – комутаційний рознімач; *2* – внутрішня плата керування; *3* – потенціометр; *4* – колекторний електродвигун; *5* – вихідний вал редуктора; Vin – вхід живлення; GND – «земля»; PWM – сигнал керування

Основними параметрами сервоприводу є максимальний кут повороту вихідного вала, швидкість повороту на заданий кут (характеризується часом, який потрібен сервоприводу для повороту його вихідного валу на заданий кут), крутний момент (характеризується масою вантажу, який здатний утримати сервопривід за заданої довжини важеля, встановленого на його вихідному валу), діапазон робочої напруги живлення, ширина діапазону сигналу управління, за якого вихідний вал сервоприводу не змінює свого положення у відповідь на зміну зовнішнього навантаження.

Кут повороту вихідного вала сервоприводу залежить від тривалості імпульсу (рис. 5.4). Збільшення або зменшення імпульсу призводить до повороту вихідного вала сервоприводу на заданий кут. Частота сигналу керування зазвичай становить 50 Гц, що відповідне довжині періоду 20 мс.

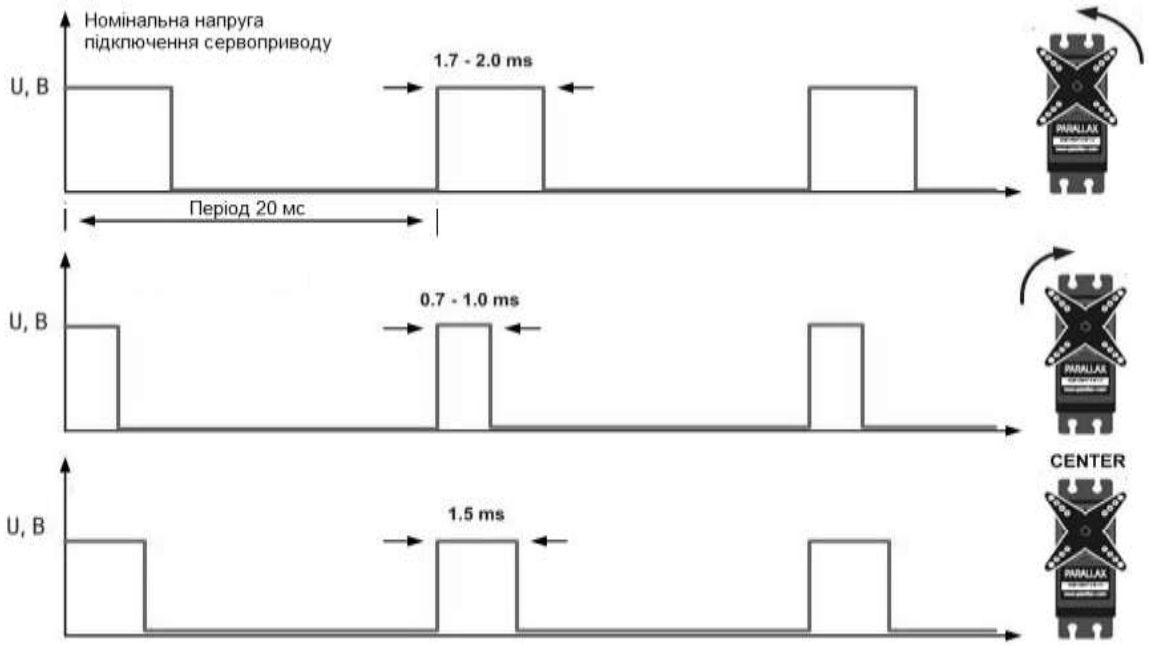


Рис. 5.4. Принцип керування серводвигуном

Для під’єднання сервоприводу потрібно, відповідно до його технічної характеристики, на входи Vin та GND подати живлення, а вхід PWM під’єднати до цифрового контакту мікроконтролера з функцією широтно-імпульсної модуляції. У разі під’єднання сервоприводу від окремого джерела живлення, контакт GND на контролері повинен обов’язково бути під’єднаний до аналогічного контакту на сервоприводі (рис. 5.5).

Кроковий двигун – це електричний безколекторний електродвигун, в якому імпульсне живлення електричним струмом призводить до того, що його ротор виконує поворот на заданий кут.

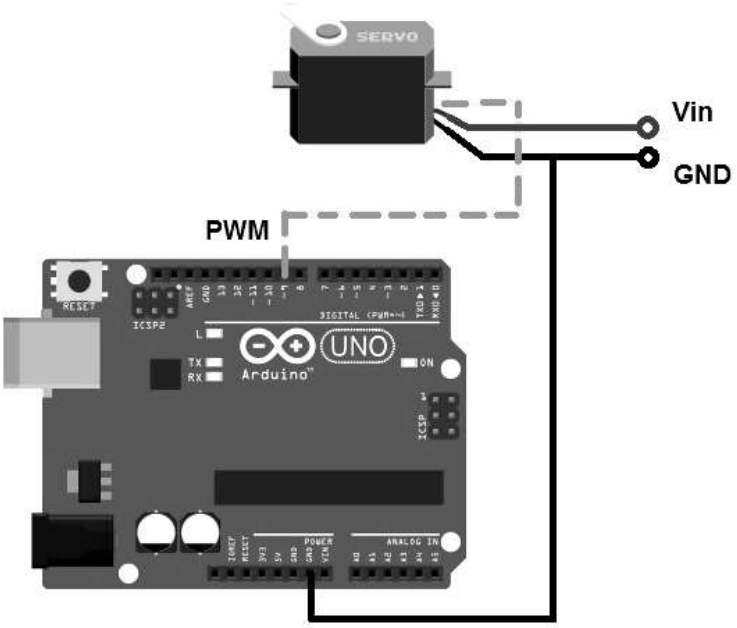


Рис. 5.5. Приклад під’єднання малопотужного сервоприводу до плати укправління Arduino

Кут повороту ротора крокового двигуна залежить від кількості поданих імпульсів струму, а кутова швидкість ротора дорівнює добутку частоти імпульсів та кута повороту ротора за один цикл. Конструкція крокового двигуна розроблена таким чином, щоб зрушення його ротора відбувалося стрибкоподібно (покроково) у відповідь на імпульс струму в обмотках статора (рис. 5.6).

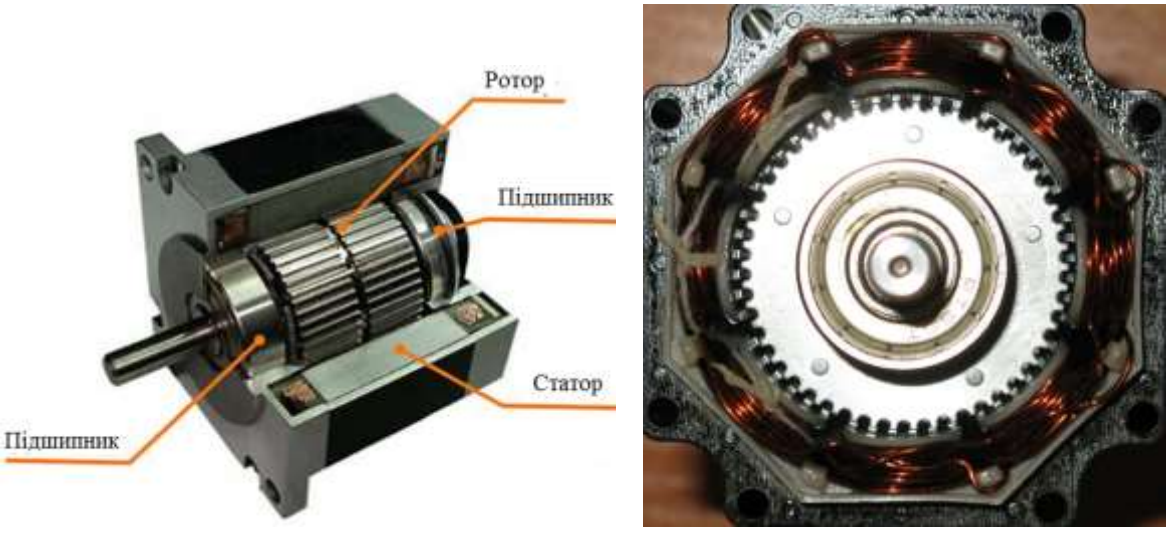
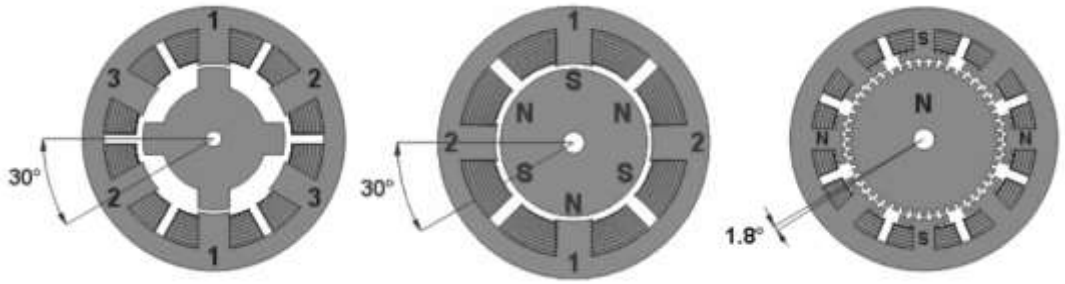


Рис. 5.6. Конструкція крокового двигуна

Ротор крокового двигуна встановлено на підшипниках і має або явно виражені магнітні полюси, або тонкі металеві зубці. Початковий крок задається конструктивно і може бути зменшений програмним способом, але не збільшений. Крокові двигуни, на відміну від колекторних, практично не збільшують рівень паразитних електромагнітних радіоперешкод, що пов'язане з браком рухомого контакту струмознімача. Відсутність колектора потребує застосування зовнішнього комутатора – драйвера двигуна. Момент утримання у крокових двигунів перевищує момент обертання, тому утримувати ротор можна струмом меншого номіналу. В крокові двигуни не встановлюють датчика зворотного зв'язку.

Є три основних типи крокових двигунів (рис. 5.7): двигуни зі змінними магнітами (застосовуються рідко); двигуни з постійними магнітами; гібридні двигуни (складніші у виготовленні, коштують дорожче, але є найпоширенішим видом крокових двигунів).



а *б* *в*

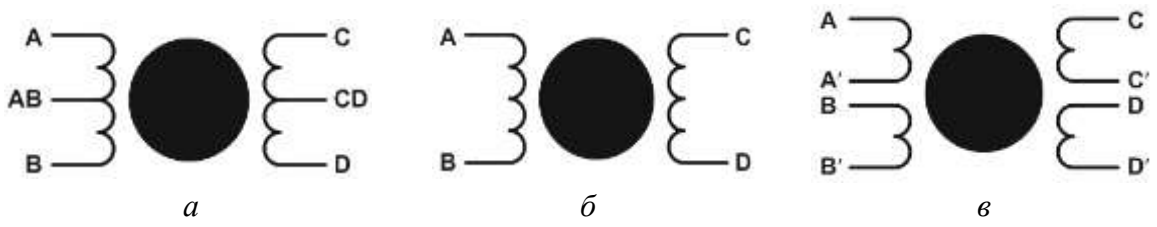
Рис. 5.7. Конструкції крокових двигунів: *а* – зі змінними магнітами; *б* – з постійними магнітами; *в* – гібридний

У *двигунах зі змінними магнітами* (рис. 5.7, *а*) обертальний момент створюється магнітними потоками статора і ротора. Статор такого двигуна, виготовлений з матеріалу з високою магнітною проникністю, має декілька полюсів. Полюси мають і статор, і ротор. Цей тип двигуна не чутливий до напрямку струму в обмотках, а обертальний момент є пропорційним величині магнітного поля, яке пропорційне струму в обмотці і кількості витків. Такі двигуни використовують для роботи на високих швидкостях.

Двигун з постійними магнітами (рис. 5.7, *б*) складається зі статора, який має обмотки, і ротора, що містить постійні магніти. Завдяки намагніченості ротора в таких двигунах досягається більший магнітний потік і, як наслідок, більший момент, ніж у двигунів зі змінним

магнітним опором. Цей тип двигунів схильний до впливу зворотної електричної рушійної сили з боку ротора, що обмежує їхню максимальну швидкість. Двигуни цього типу мають великий крок, зазвичай 18° або 7,5°, тому в ряді застосувань використовується редуктор.

Гібридні двигуни (рис. 5.7, *в*) забезпечують значно меншу початкову величину кроку, більший момент і більшу швидкість, ніж двигуни з постійними магнітами. Ротор такого двигуна складається з постійного магніта та стальних зубчастих наконечників, які розміщені в осьовому напрямку. Типова кількість зубців гібридного двигуна – від 100 до 400 (кут кроку – 3,6°…0,9°). Статор гібридного двигуна має зубці, що збільшують кількість еквівалентних полюсів, на яких розміщені обмотки. Зазвичай на статорі використовують чотири основних полюси для кроку 3,6° і вісім основних полюсів для кроку 1,8° або 0,9°. Більшість двигунів цього типу має ротор зі 100 полюсами (50 пар), а в разі двофазного виконання кількість полюсів ротора можна збільшити до 200, що дорівнює куту повороту 1,8° (360/200). Крім відмінностей в загальній конструкції, крокові двигуни вирізняються також схемою ввімкнення обмоток. Є три варіанти їхньої конфігурації (рис. 5.8), залежно від якої двигуни поділяють на уніполярні (*англ. «Unipolar»*) та біполярні (*англ. «Bipolar»*).



а *б* *в*

Рис. 5.8. Схеми ввімкнення обмоток статора крокових двигунів: *а* – уніполярна; *б* – біполярна; *в* – з роздільним виводом обмоток

Уніполярний двигун (рис. 5.8, *а*) має одну ввімкнену обмотку в кожній фазі з виводом від середини кожної обмотки. Це дає змогу змінювати напрямок магнітного поля, створюваного обмоткою, шляхом перемикання її половинок. Такий двигун має шість виводів, але середні виводи обмоток можуть бути об’єднані всередині самого двигуна в один вихід, тому такий двигун може мати й п’ять виводів. Уніполярний двигун можна використовувати в біполярному ввімкненні, не застосовуючи середні виходи.

Біполярний двигун (рис. 5.8, *б*), має дві обмотки, які вмикаються по черзі в кожній фазі, що діє змогу розвивати на 40% більший крутний момент.

Двигуни з роздільними виводами обмоток (рис. 5.8, *в*) або двигуни з вісьмома контактами дають змогу досягти максимальної гнучкості, тобто вони можуть бути під'єднані як біполярний, так і уніполярний з шістьма або п'ятьма виводами. Пара обмоток може бути під'єднана послідовно для високовольтного біполярного керування малими струмами або паралельно для низьковольтного керування великим струмом.

В управлінні кроковим двигуном важливим параметром є струм, а не прикладена до обмоток напруга, яка має прямокутну форму. Є декілька варіантів управління кроковим двигуном, а саме: хвильовий, повнокроковий, напівкроковий, мікрокроковий.

На рис. 5.9 показано форму струму в обмотках двигуна відносно нуля для чотирьох основних варіантів керування. Найбільш простий варіант – це почергова комутація фаз, за якої вони не перекриваються і в кожний момент часу ввімкненою є тільки одна фаза (рис. 5.9, *а*). Цей режим називають хвильовим (*«Wave drive mode»*) або повнокроковим режимом керування без перекриття фаз (*«One phase on full step mode»*). Недоліком такого способу керування є те, що для біполярного двигуна в один і той самий момент часу використовується тільки 50% обмоток, а для уніполярного – 25%. Це означає, що в такому режимі не можна досягти максимально можливого моменту обертання.

Повнокроковий режим керування з перекриттям фаз (*«Full step mode» або «Two-phase-on»*) – це такий режим керування, коли в один і той самий момент часу вмикають обидві обмотки двигуна (рис. 5.9, *б*). За цього способу керування ротор фіксується в проміжних позиціях між полюсами статора, даючи приблизно на 40% більший момент, ніж у попередньому варіанті, при цьому забезпечуючи такий самий кут кроку, але зі зміщенням позиції точок рівноваги ротора на півкроку, що часто не є критичним. Іноді це треба враховувати щодо двигунів з великим кроком, оскільки знеструмлений двигун, наприклад, з кроком 18° після зупинки зміститься на 9°. Для того щоби ротор такого двигуна не зміщувався під час вимкнення, на його обмотки в режимі зупинки подають деякий струм утримання, який збереже задане положення ротора.

Напівкроковий режим управління (*«Half step mode» або «One and two-phase-on»*) є комбінацією двох описаних раніше. У цьому режимі двигун за один імпульс керування робить крок, який дорівнює половині основного (рис. 5.9, *в*).

Мікрокрокові системи управління (*«Micro stepping mode»*) мають ще меншу градацію (дроблення кроку), вони основані на поступовій зміні струму в обмотках двигуна (рис. 5.9, *г*). Такі системи керування є складними та потребують застосування спеціальних інтегральних мікросхем, драйверів з цифро-аналоговими перетворювачами та мікропроцесорного управління.

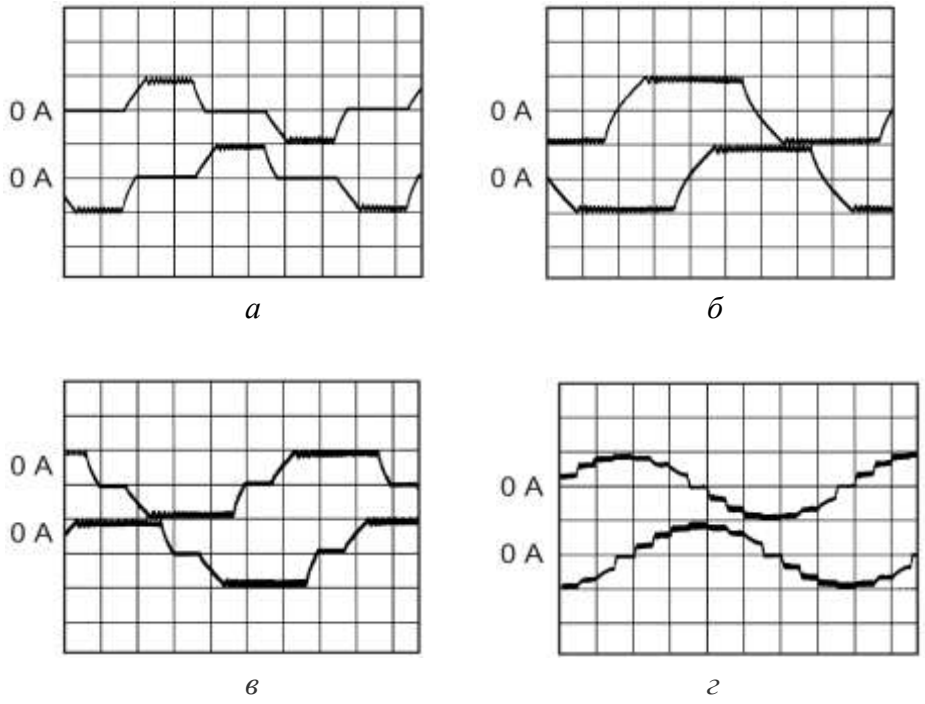
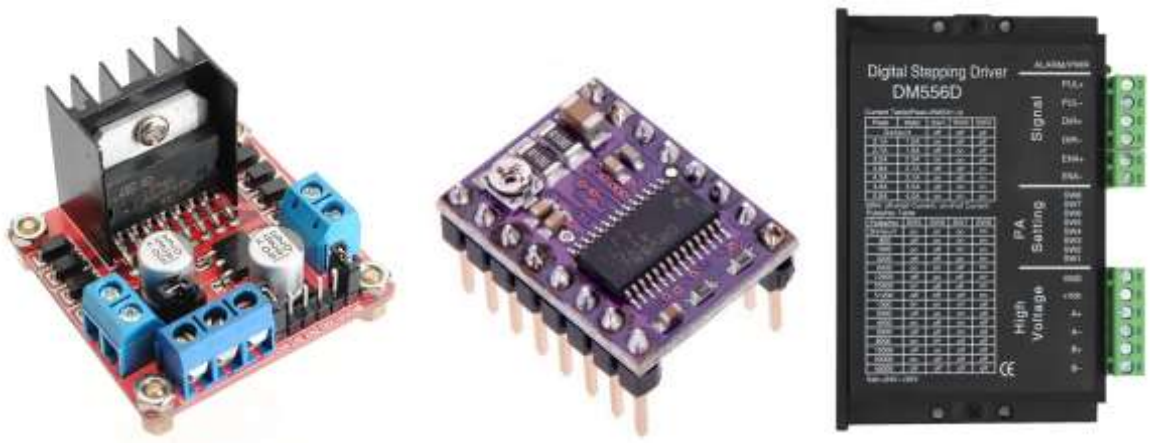


Рис. 5.9. Діаграми зміни струму в обмотках біполярного крокового двигуна для різних режимів керування: *а* – хвильовий; *б* – повнокроковий; *в* – напівкроковий; *г* – мікрокроковий

Для під’єднання крокового двигуна до контролера керування застосовують драйвер двигуна, наприклад, для керування малопотужними біполярним кроковим двигуном найчастіше використовують драйвер L298N (рис. 5.10, *а*). Цей мостовий драйвер управляє двигунами зі струмом до 2 А і напругою живлення до 46 В.



а *б* *в*

Рис. 5.10. Драйвери крокових двигунів: *а* – L298N; *б* – DRV8825; *в* – DM556D

Модуль на основі драйвера L298N складається з мікросхеми L298N, системи охолодження, клемних колодок, рознімачів для під'єднання сигналів, стабілізатора напруги і захисних діодів. Застосовують також інший вид драйверів – STEP/DIR-драйвери. Це апаратні модулі, які працюють за протоколом STEP/DIR для зв'язку з мікроконтролером. В STEP/DIR-драйверах використовується три сигнали:

– STEP – імпульс, який ініціює поворот на крок (частину кроку) залежно від режиму. Від частоти проходження імпульсів залежатиме швидкість обертання двигуна;
– DIR – сигнал, який задає напрямок обертання. Зазвичай після подачі сигналу високого рівня відбувається обертання за годинниковою стрілкою. Цей тип сигналу формується перед імпульсом STEP;
– ENABLE – дозвіл/заборона роботи драйвера. За допомогою цього сигналу можна зупинити роботу двигуна в режимі без струму утримання.

Схема драйвера L298N складається з двох H-мостів, що дає змогу під'єднувати один біполярний кроковий двигун або одночасно два щіткових двигуни постійного струму. При цьому є можливість змінювати швидкість та напрям обертання моторів. Управління відбувається за допомогою відповідних сигналів на командні входи. На рис. 5.11 наведено електричну схему L298N.

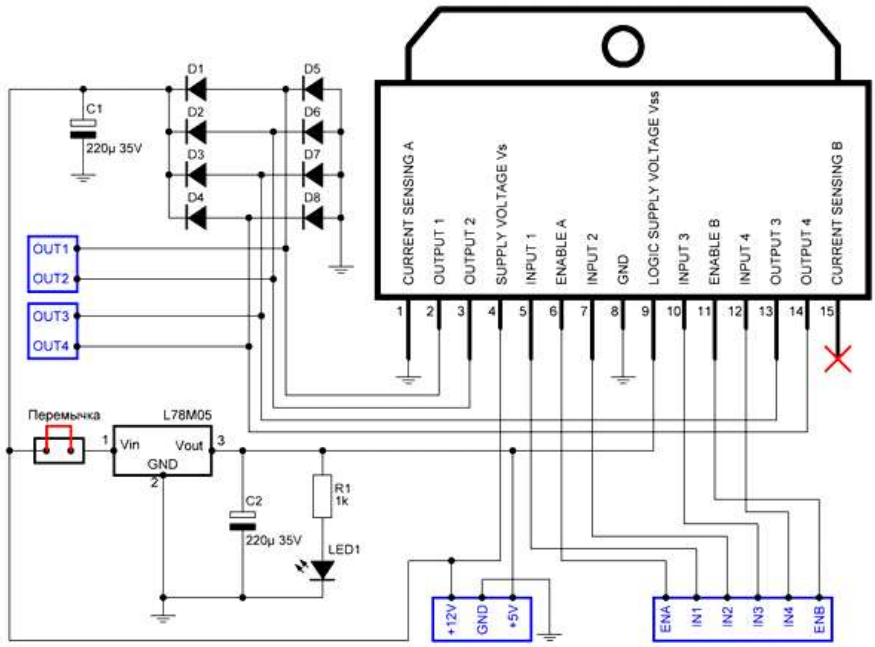


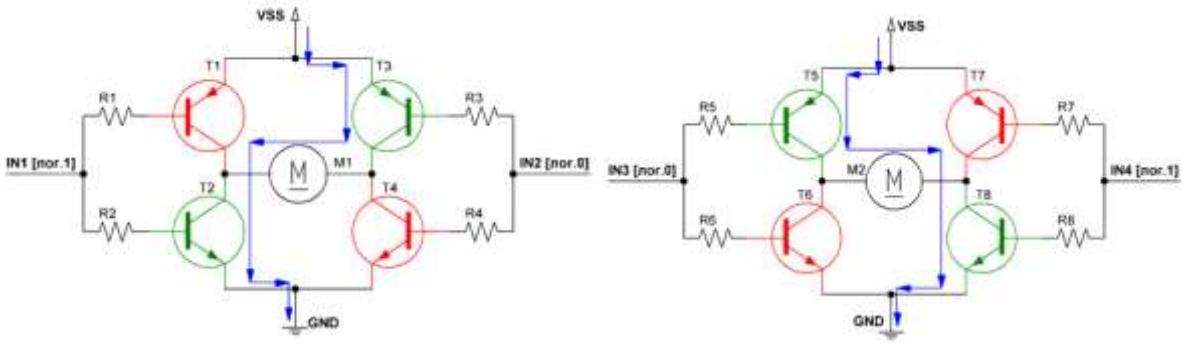
Рис. 5.11. Електрична схема драйвера L298N

На рис. 5.11 вжито такі позначення:

– OUT1, OUT2 – рознімачі для під'єднання одного зі щіткових двигунів або однієї з обмоток крокового двигуна;
– OUT3, OUT4 – рознімачі для під'єднання другого щіткового двигуна або другої з обмоток крокового двигуна;
– VSS – вхід для живлення двигунів (максимальний рівень +35V);
– GND – загальний контакт (з'єднується з аналогічним входом плати керування);
– Vs – вхід для живлення логіки +5V, через який безпосередньо живиться мікросхема L298N (інший спосіб живлення – від вбудованого модуля стабілізатора напруги 5V для L298N). У такому разі на рознімач подається тільки живлення для двигунів (Vss), контакт Vs залишається непід'єднаним, а на платі встановлюється перемичка живлення від стабілізатора, який обмежить напругу живлення до прийнятних 5V);
– IN1, IN2 – контакти керування першим щітковим двигуном або першою обмоткою крокового двигуна;
– IN3, IN4 – контакти керування другим щітковим двигуном або другою обмоткою крокового двигуна;
– ENA, ENB – контакти для активації/деактивації першого та другого двигунів або відповідних обмоток крокового двигуна. Подача логічної

одиниці контакти ENA та ENB запускає обертання двигунів, а логічний нуль – зупиняє. Для зміни швидкості обертання щіткових двигунів на контакти подається ШІМ-сигнал, а для роботи з кроковим двигуном, як правило, на контакти ENA та ENB встановлюється перемикач, для під'єднання до +5V.

Кожен Н-міст драйвера L298N складається з чотирьох транзисторних ключів із включеним у центрі навантаженням у вигляді обмотки двигуна. Такий підхід дає змогу змінювати полярність в обмотці і, як наслідок, напрямок обертання ротора двигуна шляхом чергування пар відкритих та закритих ключів (рис. 5.12).



а *б*

Рис. 5.12. Схема роботи транзстоного моста Н-типу

Якщо на вході IN1 (рис. 5.12, *а*) транзисторного моста Н-типу драйвера L298N є сигнал, а на вході IN2 – його немає, тоді внаслідок того, що транзистори в схемі мосту мають різний тип провідності, транзистори T1 і T4 залишаться в закритому стані, а через транзистори T2 і T3 проходитиме струм. Зважаючи на те, що єдиний шлях протікання струму – це обмотки двигуна, останній виявиться під'єднаним правою клемою до плюса живлення, а лівою – до мінуса. Це призведе до обертання двигуна в певному напрямку. Протилежну картину показано на рис. 5.12, *б*, де на IN3 немає сигналу, а на IN4 він є. Тепер струм тече у зворотному напрямку (ліва клема – плюс, права – мінус), змушуючи ротор двигуна обертатися у протилежний бік.

Одним з найдешевших недорогих STEP/DIR драйверів для промислових крокових двигунів є модуль TB6560-V2, проте його робота подібна до вже розглянутого.

Колекторний тип регульованих двигунів є найбільш поширеним. Частоту обертання ротора такого електродвигуна можна регулювати

рівнем напруги з мережі змінного струму за допомгою симистора (двонаправленого тиристора) (рис. 5.13).

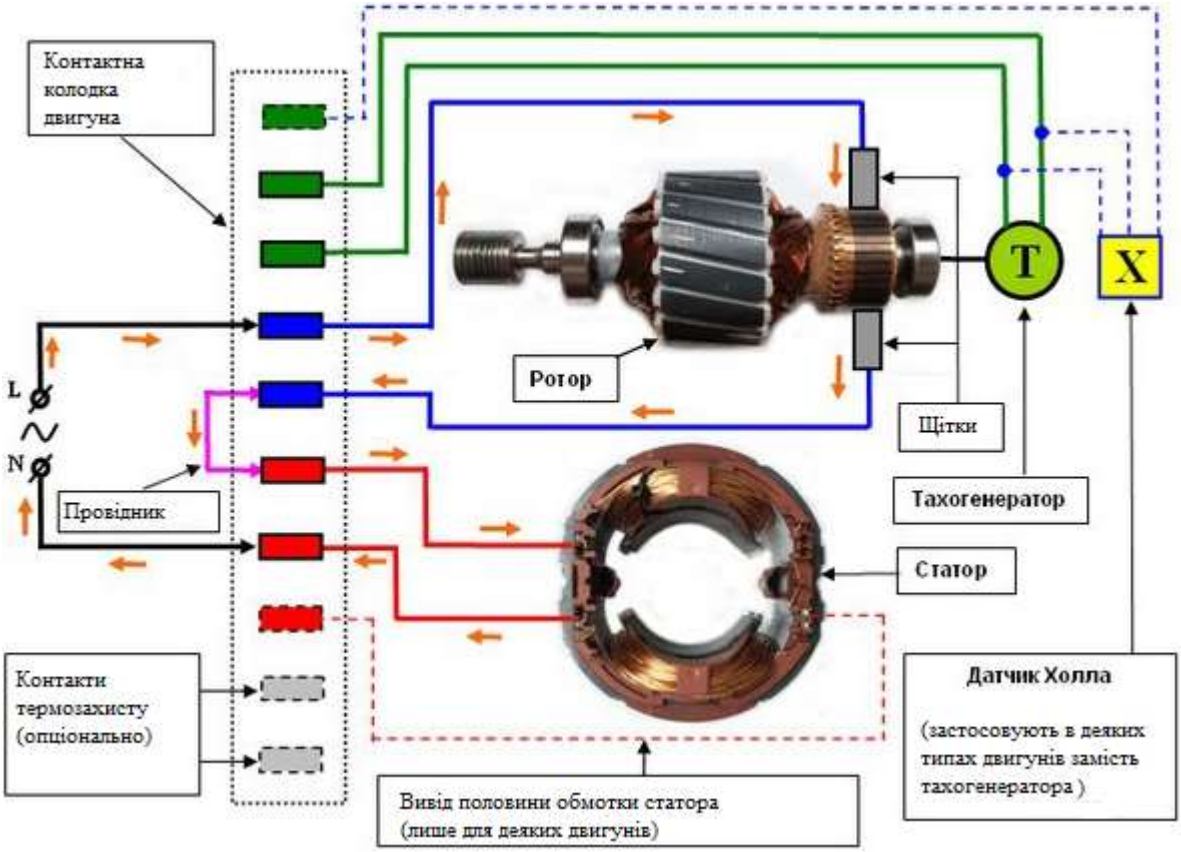


Рис. 5.13. Будова та схема керування колекторного електродвигуна

Для управління роботами та БПЛА на відстані застосовують різні види радіосистем (радіотранслятори дальної дії або мережеві Wi-Fi та Bluetooth).

Апаратура радіоуправління БПЛА складається з передавача, який знаходиться у пілота, та розміщених на моделі приймача і пілотного контролера, який керує літальним апаратом через регулятори потужності. Передавачі можуть бути спеціальними компактними переносними, великогабаритними автономними (рис. 5.14), а також у вигляді персонального компьютера або телефона з комунікаційними передавачами сигналів Wi-Fi або Bluetooth.

а

б

в

Рис. 5.14. Види радіопередавачів: *а* – комбінований передавач DJI Mavic з під'єднанням до смартфона; *б* – 12-и канальний Radiolink AT10 II радіопередавач малого радіуса дії; *в* – спеціалізована станція-передавач з FPV UAV

Програма керування роботом може бути записана на переносний носій інформації, на компютері або на сервері в мережі, а робот та БПЛА будуть під'єднуватися до такого сервера, наприклад, за допомогою відкритої мережі Wi-Fi, та зчитувати програму управління.

Кількість каналів радіопередавача – це кількість функцій системи управління, якими можна керувати. Кожна функція, наприклад акселератор, зміна напрямку руху, зміна кута тангажа та крену БПЛА, потребує окремого каналу взаємодії. Для комфортного керування, наприклад, квадрокоптером потрібно мінімум чотири канали (рис. 5.15).

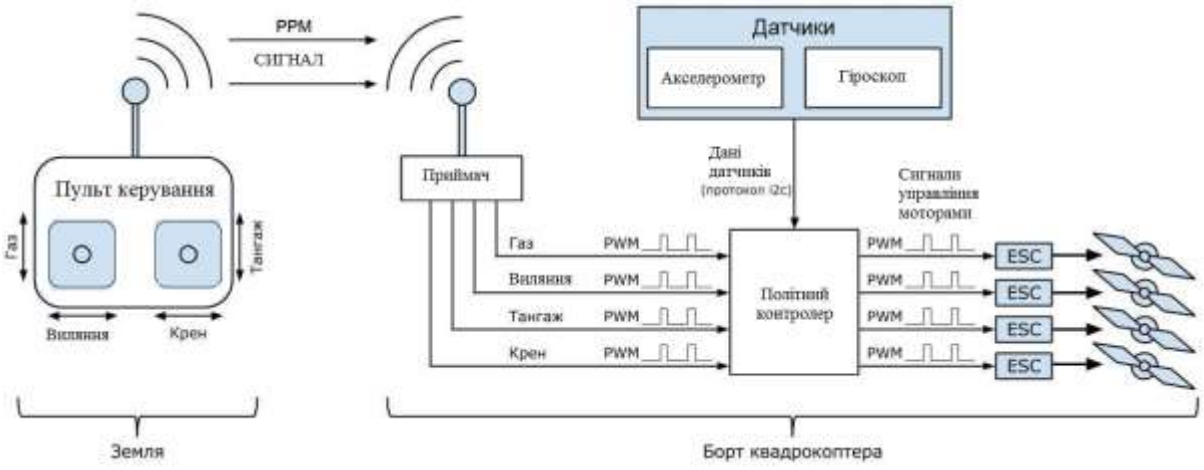


Рис. 5.15. Типова схема системи управління квадрокоптером

Команди кожного каналу з пульта керування кодуються за допомогою ШІМ-імпульсів. Канали управління бувають двох типів: пропорційні та дискретні, наприклад, пропорційний канал – керування акселерометром, а дискретний – увімкнення/вимкнення підсвічування.

Дискретні канали радіопередавачів використовують лише для допоміжних функцій, а всі основні функції передаються за допомогою пропорційних каналів. Що більше каналів управління, то більше функцій можна передати на виконавчу систему (управління підвісом камери, передавання відео, телеметрія). На виході з пульта модулюється сигнал, який передає дані на квадрокоптер. Модуляція сигналу дає змогу накласти корисний сигнал на випромінювані хвилі. Для цього всі канали ущільнюють в один за допомогою кодування. Найчастіше для цього використовується фазово-імпульсна модуляція PPM (Pulse Position Modulation). PPM-сигнал має фіксовану довжину періоду T=20 мс, що дає змогу подавати сигнал 50 разів за секунду. Пульт та приймач різних виробників можуть працювати разом, а їхня комунікація між собою відбувається за допомогою кварцових резонаторів.

Приймач – це пристрій, який встановлюють на роботизованій системі або БПЛА і який виконує виділення сигналів з радіовипромінювання передавача.

Важливою властивістю мобільної робототехніки є можливість самостійно здійснювати навігацію в просторі. Часто точної інформації про місцевість може не бути, тому робот повинен вміти самостійно будувати карти невідомої місцевості та зберігати інформацію про об’єкти сцени. Також робот повинен вміти індентифікувати своє власне положення в обмеженому просторі на відомій карті.

Системи навігації роботів – це набір технологій, які дають роботам змогу виконувати навігацію в просторі: визначати своє місцезнаходження, шлях до заданої точки або об'єкта та уникати перешкод. Системи навігації складаються з різноманітних датчиків, таких як GPS, лазерні сканери, відеокамери, радари тощо.

Супутникові системи позиціювання (GPS) призначені для того щоби:
• виконувати автоматичне переміщення за запланованим маршрутом;
• визначати поточні координати та швидкість робота.

Абревіатура *GPS* (Global Positioning System) означає і приймачів сигналів, і американську навігаційну систему, яка офіційно називається NAVSTAR, або об'єднану європейську систему (GALILEO), китайську – KOMPASS та ін. GPS (узагальнювальна назва Global Navigation Satellite System) належить до систем глобальної супутникової навігації. Система супутникової навігації (рис. 5.16) складається з орбітального угруповання (супутники), наземного (центри управління та стеження) та абонентського (приймачі споживачів) сегментів. Визначення координат об'єкта за допомогою GPS здійснюється за допомогою вимірювання відстані до супутників. Наприклад, для трьох навігаційних передавачів *A, B* та *C* на площині, вимірявши відстань *RA, RB* та *RC* до передавачів *A, B* та *C* отримаємо три кола, які своїми радіусами визначать однозначно положення об'єкта (рис. 5.16). Аналогічно відбувається визначення місця в тривимірному просторі, тільки замість кіл визначають розміщення сфер, які перетинаються в деякій спільній точці.

Супутникова система GPS складається із супутників, розподілених по шести орбітах. Супутники рухаються на висоті близько 20 000 км зі швидкістю 3000 м/с і здійснюють два оберти навколо Землі за добу. Деякі GPS (такі як NAVSTAR) використовують кодове шифрування каналів на декількох частотах, інші (такі як KOMPASS) –

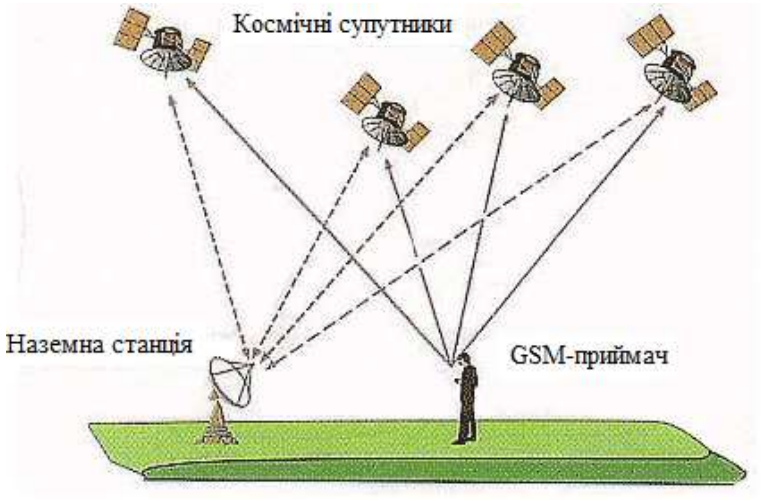
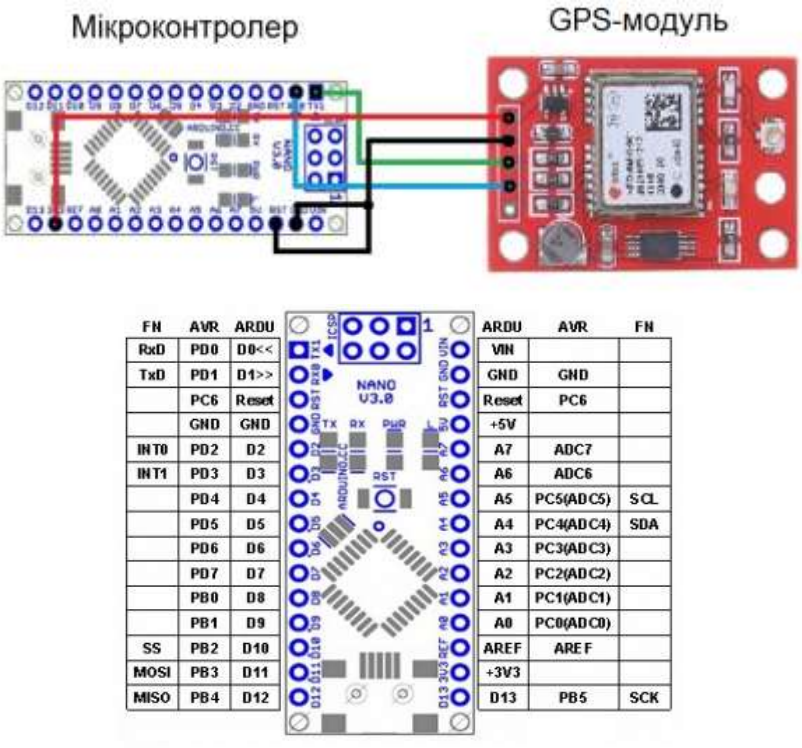


Рис. 5.16. Принцип роботи системи GPS

частотне. Приймачі GPS за один радіочастотний тракт можуть приймати сигнали декількох навігаційних систем. Використання мультисистемних приймачів значно підвищує якість навігації, особливо у складній обстановці (міська забудова, ліс, гори). Схему комутації GPS-модуля з найпростішим мікроконтролером подано на рис. 5.17.



FN	AVR	ARDU
RxD	PD0	D0<<
TxD	PD1	D1>>
	PC6	Reset
	GND	GND
INT0	PD2	D2
INT1	PD3	D3
	PD4	D4
	PD5	D5
	PD6	D6
	PD7	D7
	PB0	D8
	PB1	D9
SS	PB2	D10
MOSI	PB3	D11
MISO	PB4	D12

ARDU	AVR	FN
VIN		
GND	GND	
Reset	PC6	
+5V		
A7	ADC7	
A6	ADC6	
A5	PC5(ADC5)	SCL
A4	PC4(ADC4)	SDA
A3	PC3(ADC3)	
A2	PC2(ADC2)	
A1	PC1(ADC1)	
A0	PC0(ADC0)	
AREF	AREF	
+3V3		
D13	PB5	SCK

Рис. 5.17. Схема комутації GPS-модуля з мікроконтролером ArduinoNano

Акселерометр – це пристрій, який вимірює проєкцію прискорення (різницю між істинним прискоренням об’єкта і гравітаційним прискоренням). Як правило, акселерометр є чутливою масою, закріпленою в пружному підвісі, а її відхилення від початкового положення за наявності прискорення вказує на величину цього прискорення (рис. 5.18).

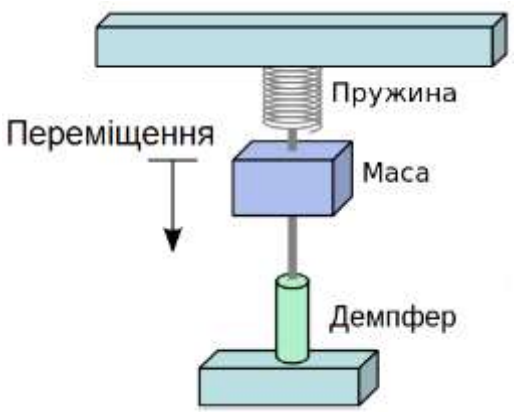


Рис. 5.18. Будова акселерометра

За конструктивним виконанням акселерометри поділяються на одно-, дво-, трикомпонентні, які можуть вимірювати проєкції прискорення відповідно в одній, двох і трьох площинах. Деякі акселерометри мають вбудовані

системи збору та обробки даних, що дає змогу створювати завершені системи для вимірювання прискорення та вібрації з усіма належними елементами.

Акселерометр можна застосовувати і для вимірювання проєкцій абсолютного лінійного прискорення (якщо відомі величина і напрямок гравітаційного прискорення в точці простору), і непрямих вимірювань проєкції гравітаційного прискорення (за нерухомого акселерометра в гравітаційному полі). Першу властивість використовують для створення інерційних навігаційних систем, коли отримані за допомогою акселерометрів вимірювання інтегрують, отримуючи інерційну швидкість та координати носія. Акселерометри є невід'ємними компонентами систем навігації БПЛА. Друга властивість дає змогу використовувати акселерометри для вимірювання кутів нахилу. Схему комутації акселерометра з найпростішим мікроконтролером подано на рис. 5.19.

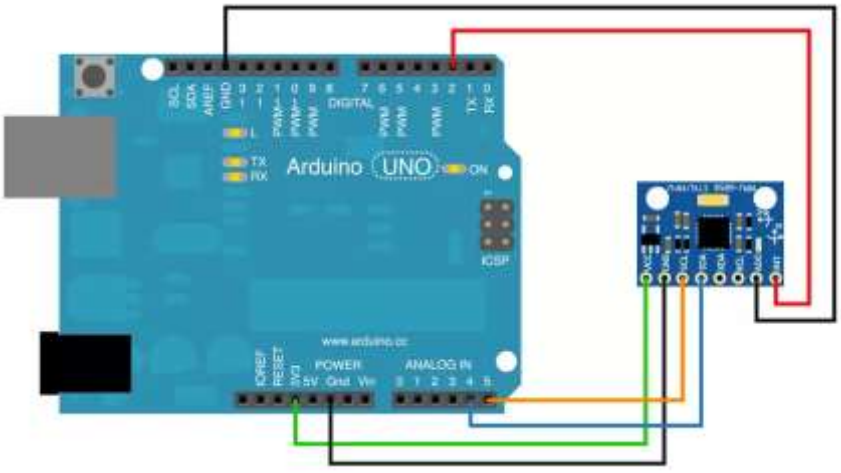


Рис. 5.19. Схема комутації акселерометра з мікроконтролером ArduinoUNO

Для управління роботом треба вміти запрограмувати його контролер. *Мікроконтролер* – це програмована мікросхема, призначена для керування різними електронними пристроями та взаємодії між ними відповідно до заданої програми. Мікроконтролер (рис. 5.20) здатен виконувати швидкі низькорівневі обчислення за деякою послідовністю інструкцій програмної логіки і може комунікувати з пристоями управління. Мікроконтролери містять регістри пам'яті, в яких виконуються обчислення, порти вводу/виводу сигналів, цифро-аналогові та аналого-цифрові перетворювачі відповідно до моделі контролера.

PDIP - Plastic Dual In-Line Package

SOIC - Small Outline Integrated Circuit

PLCC - Plastic Leader Chip Carrier

TQFP - Thin Profile Quad Flat Package

MLF - Micro Lead Frame Package

Рис. 5.20. Різновиди корпусів мікроконтролерів

Сигнал, який подається на мікроконтролер може бути ввімкненим на максимум (цифрове значення «1») або мінімум (цифрове значення «0» або «вимк.»). Окрім того, мікроконтролер може зчитувати аналогові сигнали з проміжними значеннями. За допомогою програмних інструкції мікрокотролер може перетворювати вимірювані аналогові сигнали на цифрові, оброблювати цифрові сигнали, а також можна запрограмувати виводи мікроконтроллера на ввімкнення і вимкнення.

Як і будь-яка обчислювальна машина, мікроконтролер складається з модуля пам’яті, арифметично-логічного пристрою (процесора) та інтерфейсів (порти вводу/виводу I/O).

Пристрої пам’яті контролерів складаються з оперативної пам’яті (RAM – енергозалежна) та постійної (ROM) перепрограмованої (EPROM або EEPROM – електроннoперепрограмованої). Оперативна пам’ять використовується процесором контролера для запису та тимчасового зберігання інформації. Якщо вимкнути живлення контролера, RAM-пам’ять і дані зникнуть. EEPROM-пам’ять часто називають FLASH-пам’яттю. Після вимкнення пристрою інформація зберігається у FLASH-пам’яті. Перезапис інформації в такій пам’яті відбувається програмним шляхом завдяки електронній дії на мікросхему. Недоліком такого виду пам’яті є її низька швидкодія, тому вона не підходить для застосування в RAM-пам’яті.

Для будь-якого оброблення даних їх спочатку потрібно завантажити в регістр, виконати необхідні процедури, а тоді знову зберегти у пам’яті. Процесор виконує обробку даних в регістрах контролера, а також функції додавання, віднімання, логічного порівняння, інкрементацію даних. Коректна робота процесора мікроконтролера має відбуватися за деяким часовим тактом, тому процесори обов’язково містять таймери і генератори тактів або ці системи можуть бути під’єднані ззовні. Таймери складаються з годинника реального часу і таймера переривань. Вбудовані пристрої мають підвищену надійність, оскільки вони не потребують жодних зовнішніх електричних кіл.

Основною класифікаційною ознакою мікроконтролерів є розрядність даних, оброблюваних арифметико-логічним пристроєм. За цією ознакою мікроконтролери поділяють на 4-, 8-, 16-, 32- та 64-розрядні. Тактова частота мікроконтролера – це швидкість його шини, яка визначає кількість обчислень за одиницю часу. Продуктивність мікроконтролера та споживана потужність збільшуються в міру підвищення тактової частоти. Продуктивність мікроконтролера вимірюють у MIPS (Million Instructions Per Second – мільйон інструкцій за секунду).

Залежно від кількості виконуваних команд мікроконтролери поділяють на пристрої CISC (Complex Instruction Set Computer), RISC (Reduced Instruction Set Computer), ARM (Advanced RISC Machine вдосконалена RISC) архітектури команд. Мікроконтролери з RISC-набором команд мають зменшений набір інструкцій через те, що система команд не містить арифметико-логічних операцій з операндами у пам’яті. В мікроконтролерах з CISC-набором команд більшість інструкцій є комплексними, тобто реалізують певний набір простіших інструкцій процесора або шляхом зіставлення з кожною CISC-командою деякої мікропрограми, або ж такі команди можуть бути зведені до набору простих інструкцій. Крім того, ознаками CISC-архітектури можна вважати також наявність великої кількості методів адресації пам’яті з можливістю безпосередньої роботи з операндами в основній пам’яті комп’ютера. Таким чином, одній CISC-команді повинні відповідати кілька RISC-команд, проте виграш у швидкодії RISC перекриває недолік в кількості команд. В теперішній час відмінності між RISC і CISC не суттєві, а основною характеристикою фактично вважають виконання команд за один такт.

Основними архітектурами компонувннями мікроконтролерів є такі:

- Фон-Нейманівська (прінстонська);
- гарвардська;
- модифікована (оптимізована) гарвардська CISC-архітектура.

Архітектура мікроконтролера Фон-Неймана є простою в апаратній реалізації (рис. 5.21), ій властива універсальність виконання команд.

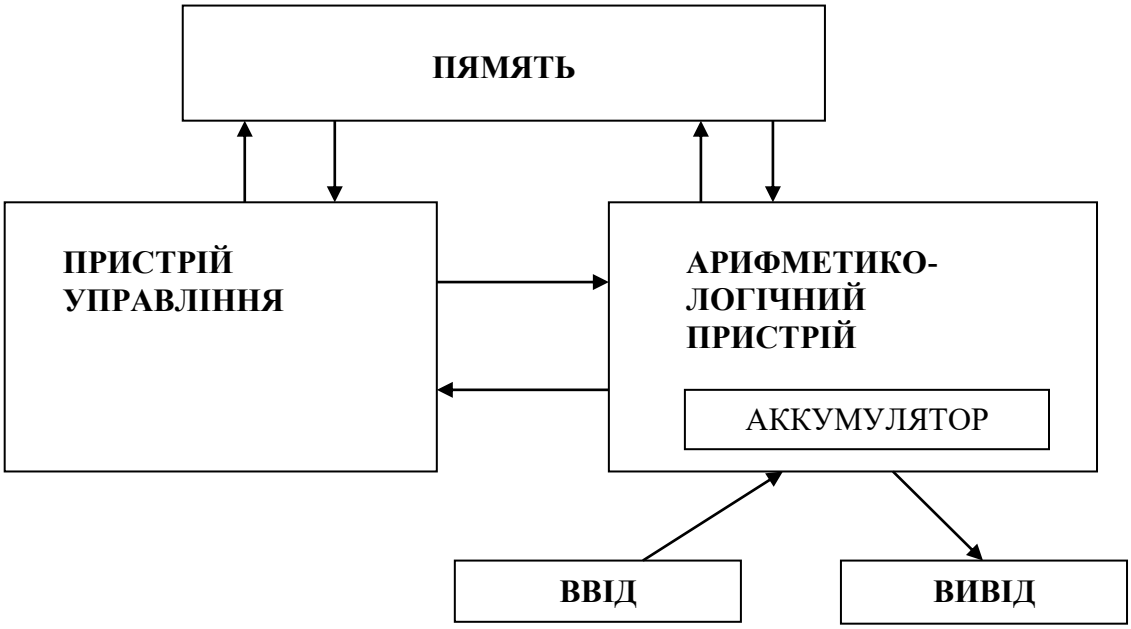


Рис. 5.21. Архітектура Фон-Неймана

Гарвардська архітектура мікроконтролера (мікропроцесора) (рис. 5.22) реалізовує пам’ять у вигляді різних пристроїв фізичної пам’яті (пам’яті для програм ПП і пам’яті для даних ПД) та за допомогою двох незалежних шин, що працюють паралельно для читання даних і команд.

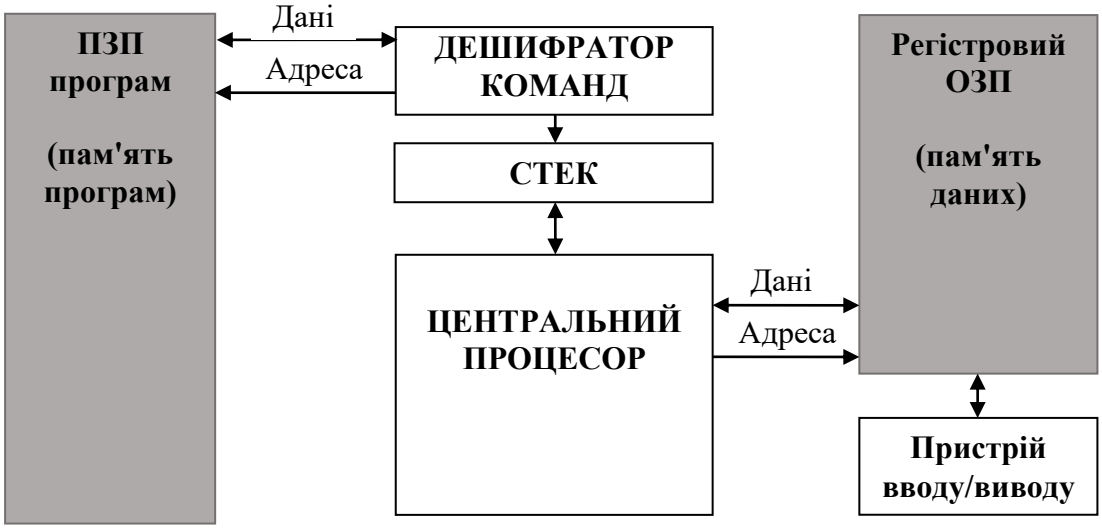


Рис. 5.22. Гарвардська архітектура: ОЗП – оперативний запам’ятовувальний пристрій; ПЗП – постійний запам’ятовувальний пристрій

Недоліком гарвардскої архітектури є неможливість динамічного розподілення даних і коду програм в памяті, що збільшує час завантаження та призводить до неефективного використання пам'яті. В оптимізованій гарвардській архітектурі (рис. 5.23) використовуються роздільні шини адресів і даних, що сприяє швидкості та економічності використання ресурсів пам'яті.

Нині використовують обидві архітектури пам'яті: гарвардську у 8-розрядних контролерах і Фон-Нейманівську в універсальних 16-розрядних і вищої розрядності.

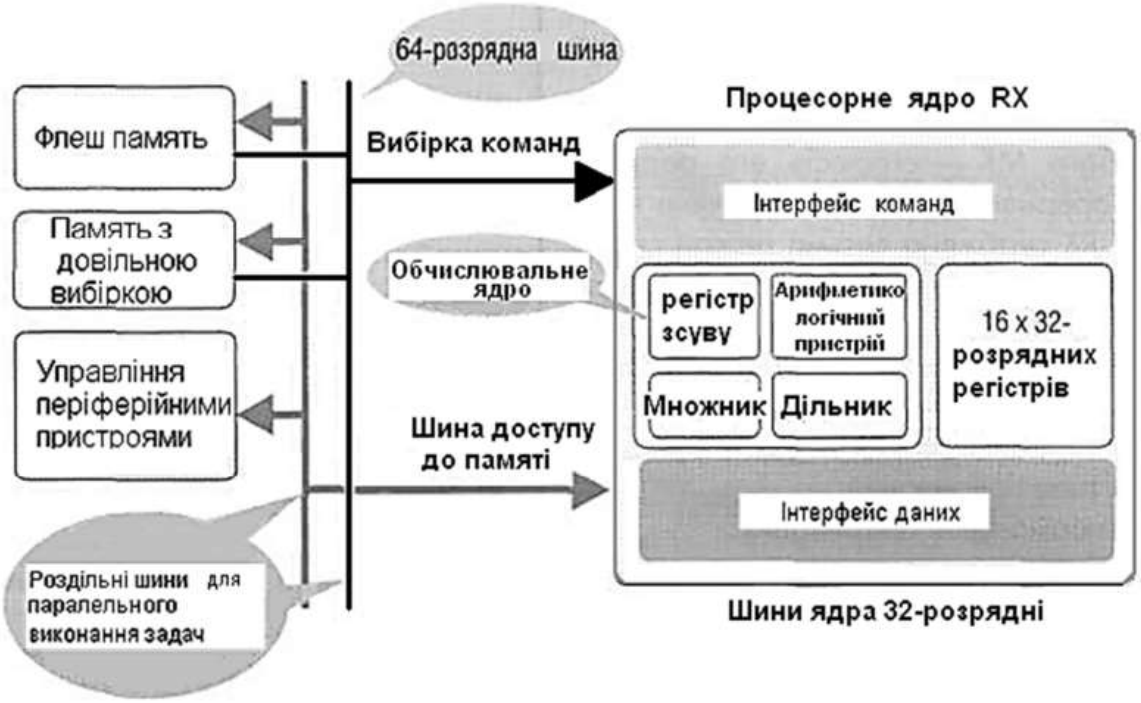


Рис. 5.23. Оптимізована гарвардська архітектура

AVR (Alf-Egil Bogen, Vegard Wollen, RISC) – найбільша виробнича лінія серед мікроконтролерів корпорації Atmel. Особливістю AVR-мікроконтролерів, яка сприяла їхній популяризації, є використання RISC-архітектури, що характеризується потужним набором інструкцій (118-133), кожна з яких має довжину в одне слово (16 біт) і більшість яких виконуються за один машинний цикл. Це означає, що за однакової частоти тактового генератора вони досягають продуктивності в 12 (6) разів більшої за продуктивність попередніх мікроконтролерів на основі CISC-архитектури (наприклад, MCS51). AVR-мікроконтролери досягають продуктивності до 16 млн процедур за секунду і підтримують FLASH-пам'ять програм різної ємності (від 1 до 256 кБайт).

Архитектура AVR-контролерів (рис. 5.24) оптимізована під мови високого рівня (Сі), а більшість представників сімейства megaAVR містять 8-канальний 10-розрядний аналого-цифровий перетворювач (АЦП), а також сумісний з IEEE 1149.1 інтерфейс JTAG або 110 debugWIRE для вбудованого налагодження. Всі мікроконтролери megaAVR з флэш-памятью ємністю 16 кБайт і більше можуть програмуватися через інтерфейс JTAG.

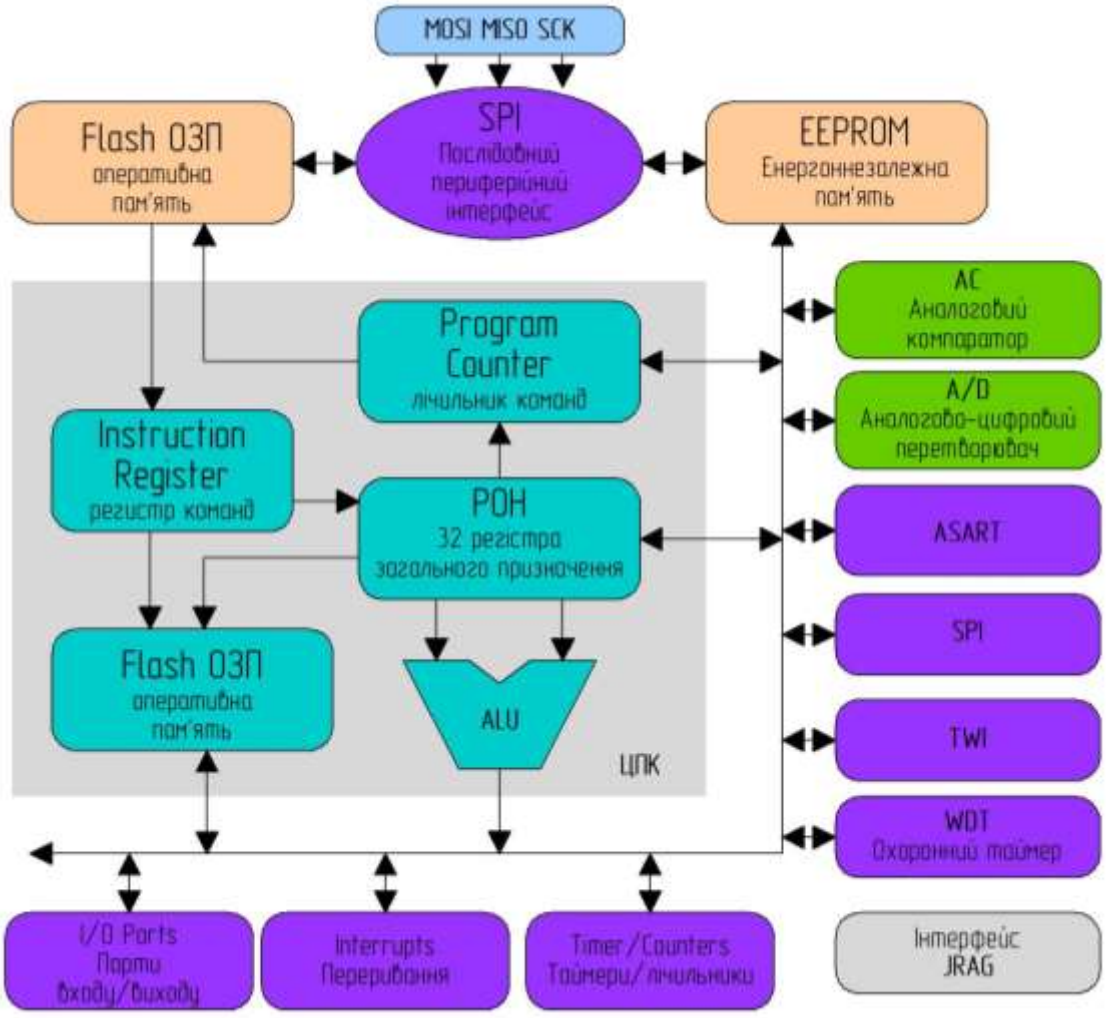


Рис. 5.24. Архітектура мікроконтролера Atmega8 фірми Atmel (див. додаток)

PIC (Peripheral Interface Controller) – це сукупність 8-, 16- та 32-розрядних мікроконтролерів від фірми Microchip, що мають гарвардську архітектуру. Основним призначенням мікроконтролерів є виконання інтерфейсних функцій для управління системами автоматики електроприводів, побутовими пристроями, для вимірювання і контролю. PIC-мікроконтролери мають RISC-систему команд з характерним малим набором одноадресних інструкцій (33, 35 або 55), кожна з яких має довжину в одне слово (12, 14 або 16 біт) і більшість яких виконується за

один машинний цикл. У цій системі немає складних арифметичних команд (множення, ділення), гранично зменшено набір умовних переходів. Час машинного циклу за тактової частоти 20 МГц становить 200 нс, швидкодія дорівнює 5 MIPS (млн процедур/с). На лініях портів вводу/виводу наявні потужні драйвери (до 24 мА) з низькою споживаною потужністю. Пам'ять даних PIC-контролерів організовано у вигляді реєстрового файла об'ємом 32-128 байт, у якому від сімох до 16 регістрів відведено для управління системою та обміну даними з зовнішніми пристроями. Однією з основних переваг цих пристроїв є дуже широкий діапазон напруг живлення (2-6 В). Струм споживання на частоті 32768 Гц не перевищує 15 мкА, за частоти 4 МГц – 1-2 мА, на частоті 20 МГц 5-7 мА і в режимі мікроспоживання (режим SLEEP) – 1-2 мкА. Випускають модифікації для роботи в трьох температурних діапазонах: від 0 до +70 °С, від -40 до +85 °С та від -40 до +125 °С.

Одним з популярних видів контролерів є STM-контролери. В 2004 році було розроблене 32-розрядне процесорне ядро ARM Cortex-M3, а компанія STMicroelectronics однією з перших реалізувала розробку на цьому ядрі, мікроконтролери дістали назву STM32.

Нині є безліч готових контролерних плат з розміщеними на них мікроконтролерами та засобами завантаження програм. Такі плати призначені для налагодження роботи мікроконтролера, а також можуть працювати як готові закінчені пристрої.

NodeMCU (ESP8266) (рис. 5.25) – це плата розміром 6×3 см, на якій встановлено мікроконтролерний модуль ESP8266 з процесором Tensilica Xtensa L106, що працює на частоті 80 МГц (можна розігнати до 160 МГц).

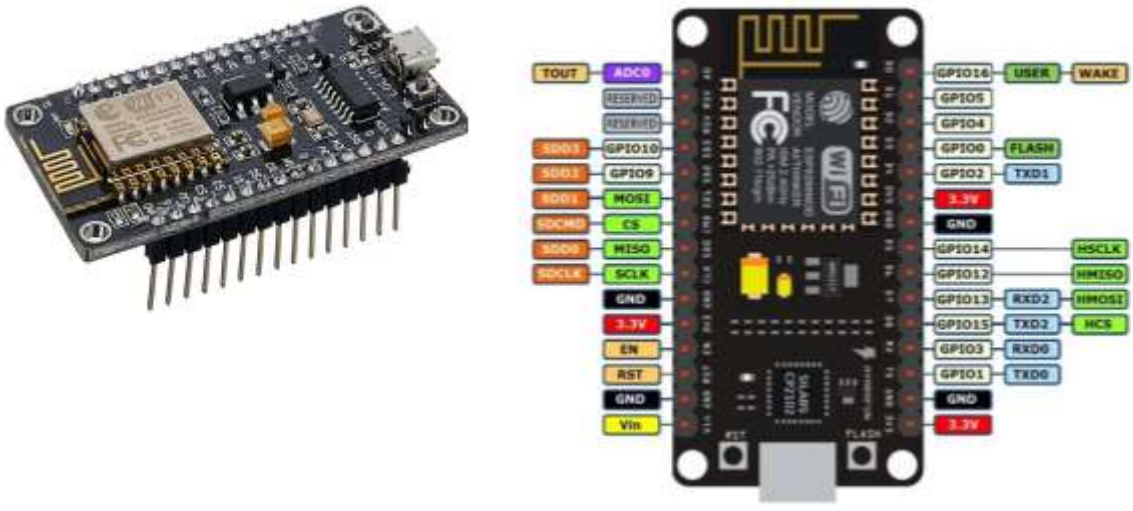


Рис. 5.25. Плата NodeMCU (ESP8266)

Плата містить чотири мегабайти Flash-пам’яті. Плата NodeMCU вміє працювати з локальною мережею або з інтернетом через Wi-Fi. Програми завантажуються через рознімач Micro USB на платі. Наявність інтерфейсу UART-USB дає змогу легко під’єднати плату до комп’ютера. NodeMCU можна програмувати мовою С, проте є багато «прошивок», що дають можливість писати програми для ESP8266 мовами високого рівня: Lua, MicroPython, JavaScript, Basic, Lisp. Використовують NodeMCU для створення систем розумного будинку, інтернету речей або роботів, керованих на відстані.

STM8S-Discovery (рис. 5.26) – це налагоджувальна плата від STMicroelectronics з 8-розрядним мікроконтролером STM8S105C6T6, з 32 КБ Flash-пам’яті для програм, 2 КБ оперативної пам’яті, 1 КБ енергонезалежної пам’яті EEPROM, користуцький світлодіод і сенсорні кнопки, а також макетний майданчик для монтажу додаткових елементів схеми. В плату інтегровано програматор/налагоджувач ST-Link, який дає змогу завантажувати програми в пам’ять мікроконтролера за допомогою USB-порту.

а *б*

Рис. 5.26. STM мікроконтролери: *а* – STM8S-Discovery; *б* – STM32VL-Discovery

Arduino – це лінійка плат на базі мікроконтролерів ATmega. Найбільшого застосування набули плати Arduino Uno (рис. 5.27) на базі мікроконтролера ATmega328P та Arduino Mega на базі котролера ATmega2560.

Програмування мікроконтролерів плат Arduino виконується спеціальною мовою на основі мови Сі та скомпільованою бібліотекою AVR Libc. Для програмування засосовують спеціальне середовище Arduino IDE. Для роботи з платою Arduino можна застосовувати і персональний комп’ютер з операційною системою Mac, Windows, Linux, і Raspberry Pi.

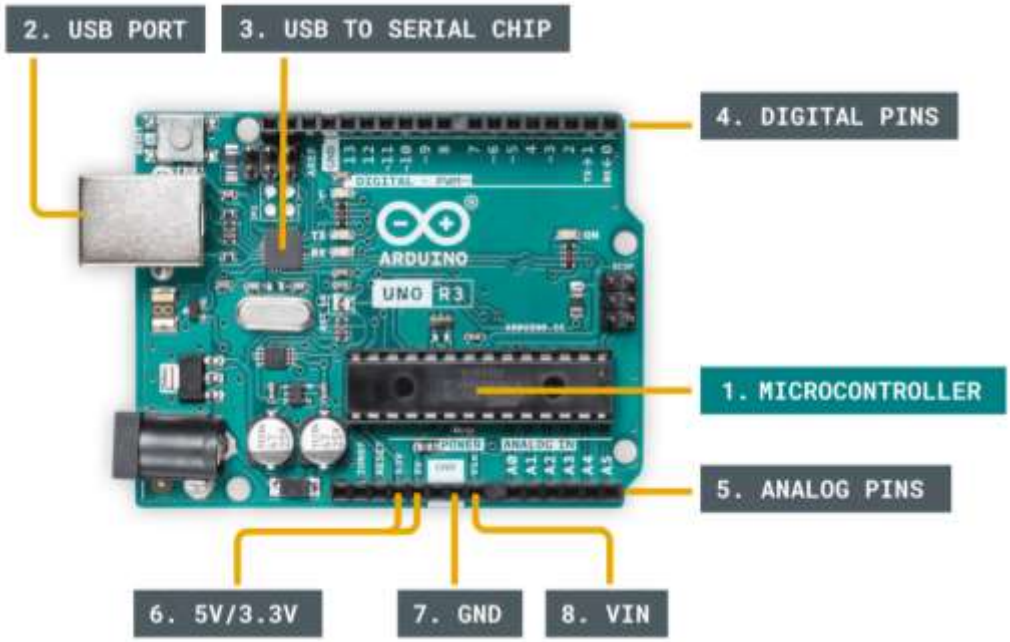


Рис. 5.27. Плата Arduino Uno

Програми для Arduino називаються скетчами. Для завантаження скетчу використовують завантажувач Arduino – невелику програму, заздалегідь завантажену в мікроконтролер на платі. Запрограмувати Arduino Uno можна, під'єднавши плату до комп'ютера звичайним кабелем USB. Для китайських клонів Arduino треба встановлювати драйвер CH340.

Контрольні запитання

1. Опишіть будову «вентильного» електродвигуна.
2. У чому полягає принцип роботи сервоприводу?
3. Як працює кроковий електродвигун?
4. Яким чином виконують керування сервоприводом?
5. Яким чином виконують керування крековим електродвигуном?
6. Які види навігації роботів вам відомі?
7. Охарактеризуйте принцип роботи акселерометра.
8. Яким чином можна застосувати акселерометр для навігації?
9. Як працює GPS-навігація?
10. Що таке мікроконтролер?
11. Які види архітектури для мікроконтролерів вам відомі?

Рекомендована література

1. *Пелевін Л.Є.* Синтез робототехнічних систем в машинобудуванні: підручник / Л.Є. Пелевін, К.І. Почка, О.М. Гаркавенко, Д.О. Міщук, І. В. Русан. – Київ : Інтерсервіс, 2016. – 258 с.
2. *Поліщук М.М.* Робототехнічні системи: проєктування і моделювання [Електронний ресурс]: навч. посіб. / М.М. Поліщук, М.М. Ткач. – Київ : КПІ ім. Ігоря Сікорського, 2021. – 112 с.
3. *Цвіркун Л.І.* Робототехніка та мехатроніка: навч. посіб. для студ. вищ. навч. закл. / Л.І. Цвіркун, Г. Глулер. – Дніпро : НГУ, 2017. – 224 с.
4. *Siciliano B.* Handbook of robotics / B. Siciliano, O. Khabib, eds., Springer-Verlag Berlin Heidelberg, 2008. – 1611 p.
5. *Everett H.R.* Sensors for Mobile Robots. Theory and Applications. / H.R. Everett. – New York: Natick M.A, A.K. Peters Ltd, 1995. – 390 p.
6. *Siegwart R.* Introduction to Autonomous Mobile Robots / R. Siegwart, R. Illah. – Massachusetts Institute of Technology, 2004. – 321 p.
7. *Момот М.В*. Мобільні роботи на базі Arduino / М.В. Момот. – БХВ, 2017. – 336 с.
8. *Ловейкін В.С.* Мехатроніка: підручник / В.С. Ловейкін, Ю.О. Ромасевич, В.В. Крушельницький. – Київ : КОМПРІНТ, 2020. – 404 с.
9. *Артюх О.М.* Основи мехатроніки: навч. посіб. / О.М. Артюх, О.В. Дударенко, В.В. Кузьмін та ін. – Запоріжжя : НУ «Запорізька політехніка», 2021. – 372 с.
10. *Дослідження* динамічної моделі гідравлічного циліндра об’ємного гідроприводу [Електронний ресурс] / Д. Міщук // Гірничі, будівельні, дорожні та меліоративні машини. – 2016. – Вип. 87. С. 74–81. – Режим доступу: http://nbuv.gov.ua/ UJRN/gbdmm_2016_87_13 / (дата звернеення: 01.09.2023). – Назва з екрана.

Модуль 2. ТЕОРЕТИЧНІ ОСНОВИ ПРОГРАМНОЇ ЧАСТИНИ РОБОТІВ

Лекція № 6. Засоби програмування робототехнічних систем

Лекція № 7. Візуальне ручне та програмне керування роботом

Лекція № 8. Ознайомлення із прикладними інтерфейсами керування робототехнічних систем (API)

Лекція № 6. Засоби програмування робототехнічних систем

Системи програмного керування призначені для програмування роботів, запам'ятовування програми керування, її збереження, відтворення, видачі керівних команд, а також для контролю за їхнім виконанням. *Програма* – це послідовність інструкцій, які описують алгоритм. *Перепрограмованість* – це властивість робота замінювати керівну програму автоматично або за допомогою людини-оператора. До перепрограмування належить зміна послідовності або величин переміщення за ступенями рухомості, а також керівних функцій за допомогою засобів керування.

Для програмування роботів застосовують електронні обчислювальні машини, контролери, комп'ютери. Комп'ютери та контролери працюють на виконання інструкцій машинної мови – це послідовність двійкових чисел, які являють собою команди для процесора. Процесор декодує інструкції, які надходять з пам'яті, для визначення того, яка процедура повинна виконуватись.

Програмування робота може бути низькорівневим, коли програмуються безпосередньо мехатронні модулі робота та його окремі привідні механізми, та високорівневим, коли виконують концептуальне програмування робота як єдиного механізму, не вдаючись при цьому до програмування окремих його частин. Використовуючи персональні комп'ютери для підготовки програм, застосовують табличні або текстові формалізовані мови, в термінах яких задають вихідні дані та алгоритми поставленого завдання, тому їх часто називають алгоритмічними мовами. Застосовуючи конкретні компютери, потрібно перекладати інформацію з формальної мови на машинну за допомогою трансляторів (компіляторів та інтерпретаторів).

Процесор комп’ютера розуміє тільки машинну мову, на якій програма може мати тисячі або навіть мільйони бінарних інструкцій, написання яких може стати досить обтяжливим процесом. Як альтернатива машинній мові, була створена мова програмування асемблера, у якій замість двійкових чисел для написання інструкцій використовують короткі слова – мнемокоди. У зв’язку з тим, що мова асемблера близька за своєю природою до машинної мови, вона є мовою низького рівня. Альтернатива мови низького рівня – мови програмування високого рівня, особливістю яких є те, що людина пише програму на абстрактному рівні із застосуванням інструкцій, зрозумілих для програміста, а спеціальні бібліотеки в подальшому транслюють написані інструкії в код мови низкого рівня та машинний код.

Мова програмування контролера – це штучна мова, що являє собою систему позначень і правил для запису алгоритмів у формі, придатній для їхнього виконання комп’ютером. За рівнем абстракції мови програмування поділяють так:

– низького рівня – базуються на машинних командах процесора;

– високого рівня – оперують сутностями, зрозумілішими людині – об’єктами, функціями тощо, які в подальшому перекладають (транслюють) в мову низького рівня.

Транслятор – це програма-перекладач, яка перетворює програму, написану мовою високого рівня, в програму, що складається з машинних команд. Транслятори поділяються на інтерпретатори і компілятори.

Інтерпретатор перекладає і виконує програму рядок за рядком. Інтерпретатор перетворює невеликий фрагмент вихідної програми на машинний код, який одразу виконується процесором. Далі інтерпретатор опрацьовує наступний фрагмент програмного коду. Машинний код для повторного виконання не зберігається.

Компілятор перетворює відразу всю програму на машинні коди, або байт-коди, не виконуючи їх. Скомпільовану програму можна зберегти для подальшого використання. Збережений результат компіляції називається виконуваним файлом (наприклад, із розширенням *.exe для виконання операційною системою Windows або *.hex для виконання мікроконтролером або *.class для виконання віртуальною Java-машиною).

Залежно від системи ідей та понять, які визначають стиль написання комп’ютерних програм, а також уявлення про роботу

програми, розрізняють об’єктноорієнтовані, процедурні, функціональні *парадигми* створення програм.

За *процедурної* та *функціонгальної парадигм* програма складається з окремих блоків команд – процедур або функцій, які дають змогу використовувати фрагменти коду, записавши їх один раз та надавши їм назву, а потім запускати за потреби за назвою.

За *об’єктноорієнтованої парадигми* програма розглядається як сукупність об’єктів, що взаємодіють між собою, а сам об’єкт, що є складною структурою, має набір властивостей (полів, змінних) і здатен виконувати певний набір дій над даними (набір методів) та може реагувати на події, які з ним відбуваються.

Прикладом високорівневої інтерпретованої об’єктноорієнтованої мови є Python, а компільованої в машинний код процедурної мови – C (Ci). Крім того, Python, C/C++, Java, Delphi/Pascal, Go, Rust є універсальними мовами програмування, на яких можна програмувати процесори верхнього рівня, тоді як PHP, Perl, VBScript, JavaScript – спеціалізованими (мови, призначені для веб-програмування). Тепер є багато бібліотек для різних мов програмування, які дають змогу розширювати їхній стандартний функціонал, тому межа між універсальними та спеціалізованими мовами програмування на сьогодні вже не є чіткою. Розглянемо особливості мов програмування Python та C (Ci).

Базовими поняттями в будь-якій мові програмування є типи данних, операнди, вирази та оператори.

Тип даних визначає безліч значень, набір операцій, які можна застосовувати до таких значень та спосіб реалізації зберігання значень та виконання операцій. Процес перевірки та накладання обмежень на типи використовуваних даних називається контролем типів, або типізацією програмних даних. Розрізняють такі види типізації:

– *статична типізація* – контроль типів відбувається до компіляції;

– *динамічна типізація* – контроль типів відбувається під час виконання програми.

Мова Сі підтримує статичну типізацію, і типи всіх даних, що використовуються в програмі, повинні бути вказані перед її компіляцією. Мова Python є інтерпретованою об’єктноорієнтованою мовою програмування загального високого рівня. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду, підтримується кілька парадигм програмування,

зокрема: об'єктноорієнтоване, процедурне, функціональне та аспектно-орієнтоване програмування. Python підтримує динамічну типізацію, і при цьому типи данних можуть змінюватися в процесі виконання програми. З базових типів слід зазначити підтримку цілих чисел довільної довжини і комплексних чисел. Python має багату бібліотеку для роботи з рядками, зокрема, кодованими в юнікоді.

Інтерактивний інтерпретатор, який постачається разом з Python, дає можливість експериментувати з невеликими програмами, вводячи команди рядок за рядком і миттєво отримуючи результат кожної з них. Проте, як правило, програми містять дуже велику кількість рядків коду, тож їх зберігають у вигляді текстових файлів з розширенням .py, а потім запускають. Працювати в інтерактивному режимі можна в консолі, для чого потрібно запустити команду python в терміналі операційної системи (рис. 6.1).

```
>>> 2 ** 100 # піднесення 2 до 100-го степеня
1267650600228229401496703205376L
>>> from math import * # імпорт математичних функцій
>>> sin (pi * 0.5) # обчислення синуса від половини пі
1.0
```

Рис. 6.1. Фрейм коду з інтепретатора в Python

Структура даних являє собою іменовану сукупність простих елементів, між якими є зв'язки і співвідношення. Загалом структуру даних можна поділити на прості та складні структури.

Проста структура даних – це структура, що містить дані, які для виконавця не можуть бути розділені на складові частини і не залежать від інших даних. Такі дані називаються скалярними, або примітивними.

Складна структура даних містить кілька простих структур даних, при цьому дані простих структур можуть бути однотипними або різнотипними. Є певні структури даних, усталені в програмуванні й підтримувані всіма мовами програмування. Такими структурами є, наприклад, масив та рядок символів.

У мові програмування Сі розрізняють примітивні та складні структури данних, тоді як в мові Python всі структури даних є складними. Кожна структура даних має відповідне ім'я та значення. Значення даних можуть бути різними: цілі числа, дійсні числа, символи, логічні значення тощо. Різні дані обробляються по-різному. Для того

щоб розрізняти дані, вводиться поняття про тип даних, яке пов'язане з поняттям виконавця і відіграє важливу роль у програмуванні.

Тип даних визначає:

- структуру даних (сукупність частин та їхні взаємозв'язки);
- множину значень даних (визначається розміром пам'яті);
- семантику даних (відображення та інтерпретація в пам'яті виконавця);
- сукупність процедур над значеннями.

Для кожної обчислювальної системи є прості та складні типи. В табл. 6.1 наведено базові (примітивні) типи данних, які підримуються мовою Сі, та аналогічний відповідник, що є в мові Python, всі типи данних мають складну структуру і побудовані на основі стандарних класів.

Таблиця 6.1

Базові типи данних для мов C та Python

Тип даних	Розмір	Діапазон
мова Сі		
bool: логічний тип true (істина) і false (брехня)	точно не визначено	0 та 1
char: один символ в кодуванні ASCII	8 біт	-127 … 127 або 0…255 в залежності від компілятора
signed char: один символ або число зі знаком	8 біт	-127 … 127
unsigned char: один символ або позитивне число (без знаку)	8 біт	0…255
short int, int: ціле число зі знаком	16 біт	-32767 … 32767
unsigned short, unsigned int: позитивне ціле число	16 біт	0 … 65535
long int: ціле число зі знаком	32 біт	-2147483647 … 2147483647
unsigned long int: позитивне ціле число	32 біт	0 … 4294967295
long long int: ціле число зі знаком	64 біт	-9223372036854775807 … 9223372036854775807
unsigned long long int: позитивне ціле число	64 біт	0 … 18446744073709551615
float: дійсне число ординарної точності з рухомою комою	32 біт	$3{,}4 \cdot 10^{-38}..3{,}4 \cdot 10^{38}$
double: дійсне число ординарної точності з рухомою комою	64 біт	$1{,}7 \cdot 10^{-308}..1{,}7 \cdot 10^{308}$
long double: дійсне число ординарної точності з рухомою комою	80 біт	$3{,}4 \cdot 10^{-4932}..3{,}4 \cdot 10^{4932}$
void: тип без значення		

Закінчення табл. 6.1

Тип даних	Опис
мова Python	
Numbers: **int** **long** **float** **complex** **bool**	24 байти мінімально для 64-бітної ОС. В мові Python всі типи є обєктами і вони можуть динамічно розширюватися, тобто розмір памяті збільшується інтерпретатором автоматично. Визначення розміру в байтах: sys.getsizeof(<data>)
None	16 байт мінімально для 64-бітної ОС. 1 байт = 8 біт
Strings: **str** **unicode**	50 байт мінімально для 64-бітної ОС

Для нестандартних типів, зазвичай складних, передбачені конструктори типів, які дозволяють користувачеві створювати свої власні типи даних.

Окрім базових типів даних в Сі, є ще й складні дані:

- масив – індексований набір елементів одного типу, наприклад:

```
//створення одновимірного масиву з 5 елементів символьного
  типу
char symbols[5];
//ініціалізація першого елемента символом 'a'
char symbols[0] = 'a';
```

- структури – тип даних, що об'єднує кілька змінних (однакових або різних типів) в єдине ціле для організації зручного доступу до них. Елементи структури називають полями. Опис структури виконується за допомогою ключового слова `struct`, наприклад

```
//опис структури, яка визначатиме користувача
struct user {
          int age;
          char name[];
          int phone_number;
       };
//ініціалізація змінної типу структури user
struct user john = {10, "John", 1234567};
```

- об'єднання – це засіб, який дає змогу звертатись до одних і тих самих даних (у різні моменти часу) по-різному, як до кількох різних змінних

різних типів. Синтаксис об'єднань аналогічний синтаксису структур за винятком того, що замість ключового слова `struct` використовується ключове слово `union`;

- перерахування (enumeration) – це синонім слова «список». В програмах мовою С перераховний тип – це іменований список цілих констант, кожна з яких має власне ім'я. Імена констант повинні бути унікальними. Для створення такого списку використовують ключове слово `enum`:

```
enum boolean {no, yes};
```

– вказівник – це змінна, яка містить адресу іншої змінної (вказує на цю змінну). Вказівники є ефективним засобом доступу до значень змінних. Задається вказівник за допомогою символу «*», а за допомогою символу «&» надається значення адреси:

```
int* ptr;
int i = 10;
ptr=&i;
```

Уже згадані в табл. 6.1 дані в Python доповнено ще такими стандартними типами даних: lists (списки); dict (словники); tuple (кортежі); set (множини); frozenset (незмінні множини). При цьому в Python виокремлено такі дані:

- змінювані (списки, словники та множини);
- незмінні (числа, рядки та кортежі);
- упорядковані (списки, кортежі, рядки та словники);
- невпорядковані (множини).

В Python створення розглянутих типів даних матиме такий вигляд:

```
#створення списку
list = ["apple", "banana", "cherry"]
#створення множини
set = {"apple", "banana", "cherry"}
#створення кортежу
tuple = ("apple", "banana", "cherry")
#створення словника
dict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

Однорядкові коментарі в Python позначають символом «#», а в Сі – «//».

В мові Сі змінні оголошують оператором опису, який складається з специфікації типу та списку імен змінних, розділених комою. В кінці оголошення змінної або методу обов'язково має стояти крапка з комою: **[модифікатори] специфікатор_типу ідентифікатор [ідентифікатор] ...** , де **модифікатори** – це ключові слова **signed**, **unsigned**, **short**, **long** або їхня відсутність; **специфікатор_типу** – це ключові слова типу **char, int**, **float**, **double**, що визначає тип оголошуваної змінної; **ідентифікатор** – це ім'я змінної, наприклад:

```
char x;              //оголошення змінної
int a, b, c;
unsigned long long y;
```

Таким чином, будуть оголошені змінні **x**, **a**, **b**, **c**, **y**. У змінну **x** можна записувати значення від -128 до 127, у змінні **a**, **b**, **c** – від -32768 до 32767, а в змінну **y** – від 0 до 18446744073709551615.

Оголошуючи змінну, її можна ініціалізувати, тобто надати їй початкового значення або не робити цього, тоді в змінній буде записано дефолне значення. Зробити в Сі це можна так

```
int x = 100; //оголошення та одночасна ініціалізація змінної
```

Таким чином, в змінну **x** під час оголошення відразу ж буде записане число 100. Ініціалізовані змінні краще оголошувати в окремих рядках.

У мові Python оголошення змінних не є обвязковим, а для їхньої інійіалізації можна не застосовувати жодних індетифікаторів, крім назви змінної, наприклад:

```
#ініціалізовано змінну числовим типом
x = 100
#ініціалізовано змінну в яку передано рядок «100»
y = str(100)
```

У мові Сі змінна будь-якого типу може бути оголошена як немодифікована завдяки слову **const** до специфікатора типу. Змінні з типом **const** є даними, які можна лище зчитувати, тобто цим змінним не може бути надане нове значення після їхньої ініціалізації. Якщо після слова **const** немає специфікатора типу, тоді константи розглядаються як величини зі знаком і їм надають тип **int** або **long int** відповідно до

значення константи: якщо константа менша за число 32768, їй надають тип **int**, в інших випадках – **long int**.

Приклад:

```
const long int k = 25;
const m = -50;  // мається на увазі const int m=-50
const n=100000; // мається на увазі const long int n=100000
```

В мові Python константи створюють так само, як і звичайні змінні, проте назву константи вказують у верхньому регістрі, таким чином буде зрозуміло, що це константа і її не потрібно змінювати. Заборони на зміну змінних типу констант в Python на рівні синтаксису немає.

Рядок символів (String) – це структура, що об'єднує рядок символів в єдине ціле. Символи у рядку розміщуються підряд, кожен рядок має своє ім'я. String визначає довжину рядка (максимальну кількість символів), проте в пам'яті компютера рядок може бути порожнім.

В мові Сі так само, як і в мові Python, для передачі значення в змінну застосовують знак «=» (присвоєння). Це означає, що вираз, який стоїть праворуч від знака «=», обчислюється, а отримане значення передається в змінну, яка стоїть ліворуч від знака присвоєння. При цьому попереднє значення, що зберігається в змінній, стирається та замінюється на нове.

Приклади:

```
int x = 5 + 3; //скласти число 5 та 3,
               //результат записати в змінну x
int b = a + 4; //додати 4 і зберігти в змінній a,
               //отриманий результат присвоїти змінній b
b = b + 2;     //додати 2 і зберігт в змінній b,
               //отриманий результат записвим в змінну b
```

В правій частині значення змінної може бути використане кілька разів:

```
c = b * b + 3 * b;
```

Математичні процедури як в Сі так і в Python записуються у виразах таким чином: + (плюс), - (мінус), * (помножити), / (розділити)

Відмінності між Python та Сі полягають в тому, як будуть виконуватися математичні процеури. Наприклад, в Сі буде відображений тип змінних, з якими застосовують арифметичні процедури, а в Python тип змінної буде генеруватися автоматично, тому і результат роботи буде відрізнятися.

Приклади для мови Сі:

```
int x = 3;       //змінній x буде присвоєно значення 3
int y = x+5.1;   //до значення в змінній x, буде
                 //додано число 5 отриманий результат 8 буде
                 //записано до змінної y
int z = x * y;   //значення змінних x та y будуть перемножені,
                  //результат 24 буде записане у змінну z
z = z - 0.99;    //від значення в змінній z, буде віднято 0
                 //результат 24 записано у змінну z
y = 5 / 3;       //в змінну y буде записане значення
                 // цілочисельного ділення 5/3 = 1
```

Приклади для мови Python:

```
x = 3         #змінній x буде надане значення 3
y = x + 5.1   #до значення, що зберігається в змінній x, буде
              #додано число 5.1, отриманий результат 8.1 буде
              #записано в змінну y
z = x * y     #значення змінних x та y будуть перемножені,
               #результат 24.2999 буде записано в змінну z
z = z - 0.99 #від значення, що зберігається в змінній z, буде
              #віднято 0.99 результат 23.31 буде записано в z
y = 5 / 3     #в змінну y буде записано значення ділення 5/3 =
1.66
y = 5 // 3   #в змінну y буде записано значення цілочисельного
              #ділення 5//3 = 1
```

Звертаємо увагу на запис та результат виконаних дій, зокрема, в Python результат виконання математичних процедур не відразнятиметься від математики звичайного калькулятора, тоді як в Сі результати будуть здаватися округленими. В Сі це пояснюється тим, що цифри з крапкою не можуть бути записані в цілочисельний тип данних.

Крім простого оператора надання «=», в Сі та Python існує ще кілька комбінованих операторів надання: «+=», «-=», «*=», «/=», «%=».

Приклади:

```
x += y;  //те саме, що і x=x+y; - скласти x та y і записати
         //результат в змінну x
x -= y;  //те саме, що і x=x-y; - відняти від x значення y
         //і записати результат в змінну x
x *= y;  //те саме, що і x=x*y; - помножити x на y і записати
         //результат y змінну x
x /= y;  //те саме, що x=x/y; - розділити x на y і записати
         //результат y змінну x
x %= y;  //те саме, що x=x%y; - обчислити залишок від ділення
```

```
//x на y і записати результат у змінну x
```

В мові Сі, якщо треба змінити значення змінної на 1, використовують **інкремент** або **декремент**. **Інкремент** – процедура збільшення значення, що зберігається у змінній, на одиницю.

Приклад:

```
x++; // значення змінної x буде збільшене на 1
```

Декремент – процедура зменшення значення, що зберігається у змінній, на одиницю.

Приклад:

```
x--; // значення змінної x буде зменшене на 1
```

Інкремент та декремент належать до процедур присвоєння, в Python їх не застосовують. При використанні декремента та інкремента спільно з оператором надання "=" застосовують постфіксний (x++) або префіксний (++x) запис. Першим виконується префіксний запис.

Приклади:

```
y = x++;
```

Припустимо, що в змінній **x** зберігалося значення 5, тоді в **y** буде записано значення 5, після чого значення змінної **x** буде збільшене на 1. Таким чином, в **y** буде 5, а в **x** - 6.

```
y = --x;
```

Якщо в **x** зберігалося значення 5, то спочатку буде виконане зменшення **x** до 4, а потім це значення буде надане змінній **y**. Таким чином, **x** та **y** буде надане значення 4.

Логічними виразами називають вирази, результатом яких є істина (True) або хибність (False). У найпростішому випадку будь-яке твердження може бути істинним або хибним, наприклад, «2 + 2 дорівнює 4» – істинний вираз, а «2 + 2 дорівнює 5» – хибний. У мовах програмування Сі та Python для порівняння виразів використовуються спеціальні знаки, подібні до тих, які використовують в математичних виразах: > (більше), < (менше), >= (більше або дорівнює), <= (менше або дорівнює), == (порівняння), ! = (не дорівнює); in (включення, що фактично означає належність). Ці оператори повертають булеві значення True або False.

Умовні оператори використовують тоді, коли, залежно від якоїсь умови, треба виконати один з двох операторів. Умова – це вираз, який

складається з допустимих процедур. Синтаксис умовного оператора в Сі та Pyrhon не однаковий:

в Сі	в Python

`if (умова) {`
`  оператор1;`
`}`
`else {`
`  оператор2;`
`}`
	`if умова:`
	`  оператор1`
	`  оператор2`
	`else`
	`  оператор3`
	`  оператор4`

Як бачимо, в мові Сі логічну умову записано в круглих дужках, а оператор – у фігурних, при цьому, якщо після логічної умов стоїть лише одни оператор, то фігурні дужки є не обовязковими. Відступи в мові Сі ігноруються компілятором, застосовують їх лише для покращення читання коду програмістом. В мові Python, навпаки, кожен оператор повине бути виділений відступом, відповідним за належність оператора до логічної умови. Відступ в Python виконується за допомогою чотирьох пробілів. Логічна умова в Python записується без круглих дужок.

Якщо умова виконується (є істинною), тобто результат виразу не дорівнює нулю, то виконується оператор з розділу «if». Якщо ж умова не виконується (результат виразу дорівнює 0), то виконується оператор з розділу «else».

Цикли є однією з ключевих конструкцій мов програмування, які дають змогу запрограмувати контролери на виконання розрахунків. В мові Сі оператор покрокового циклу має такий синтаксис:

```
for ( <початковий вираз>; <умовний вираз>; <вираз приросту>)
{
    оператор1;
    оператор2;
}
```

Оператор покрокового циклу в Сі виконується таким чином.

1) Розраховується початковий вираз.

2) Перевіряється умовний вираз. Якщо результат умовного виразу дорівнює нулю (умова не дотримується), тоді закінчуємо виконання циклу, та якщо результат умовного виразу не дорівнює нулю (умова дотримується), тоді виконуємо оператор.

3) Виконується вираз приросту і здійснюється перехід до п. 2.

Якщо умовний вираз від самого початку дорівнює нулю, то цикл не виконується жодного разу. Як правило, початковий вираз – це початкове

значення змінної, що використовується в циклі, умовний вираз – кінцеве значення змінної, а вираз приросту – це вираз, за яким обчислюється наступне значення змінної.

Цикл for в Python використовується для перебору послідовностей (списків, кортежів, рядків) та інших ітерованих об'єктів, наприклад:

```
for <ітератор> in <послідовність>:
    оператор1
    оператор2
```

У Python в циклі for не застосовується явно вираз приросту та умовний вираз, а роль ітератора виконує вираз змінної, яка «перебирає» значення послідовності.

Цикл while, або цикл з умовою, повторюється, поки істинною є умова циклу:

в Сі	в Python

`while (умова){`
`  // тіло_циклу`
`}`
	`while (умова):`
	`  // тіло_циклу`

Цикли з умовою найчастіше використовуються, коли заздалегідь не відома кількість повторень і цикл повинен працювати, поки виконується умова. Для того щоби цикл завершився, у тілі циклу повинна змінюватись якась змінна, від якої залежить істинність умови повторень.

Приклад для Сі:

```
int x;
x = 50;
while (x > 0){
     x = x - 1;
}
```

У цьому прикладі тіло циклу повторюватиметься 50 разів, доки змінна **x** буде більшою від нуля. На кожному кроці циклу **x** зменшуватиметься на **1**. Коли **x** дорівнюватиме нулю, цикл припиниться.

В мові Сі є оператор циклу з післяумовою, який має такий синтаксис:

```
do{
    оператор;
}
while (умова);
```

Алгоритм виконання оператора циклу з післяумовою такий:

1) виконується оператор;

2) перевіряється умова. Якщо умова не виконується, цикл завершується. Якщо ж умова виконується, тоді відбувається перехід до п. 1.

На відміну від покрокового циклу та циклу з передумовою цикл завжди виконується хоча б один раз.

Як в мові Сі, так само і в мові Python в циклах можна застосовувати оператори продовження та примусового закінчення циклу. Синтаксис оператора продовження циклу має вигляд `continue`, а примусового закінчення – `break`.

В мові Python цикл `for` зручно застосовувати з включеннями для створення масивів. Включення дають змогу об'єднувати цикли разом з умовними перевірками, не використовуючи при цьому громіздкого синтаксису. Найпростіша форма включення в Python має такий вигляд:

```
Num = [n for n in range (1, 10)]
print(Num)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Включення списку також може містити умовний вираз:

```
Num = [n for n in range (6, 12) if n % 2 == 1]
print(Num)
```

Програмування мовою Сі ґрунтується на понятті функції. *Функція* – це самостійна одиниця програми, спроєктована програмістом для реалізації конкретної задачі. Функція містить визначений та іменований фрагмент коду, відокремлений від інших структур. Функції можуть приймати будь-яку кількість вхідних параметрів і віддавати (повертати) будь-яку кількість результатів. Використання функцій дає змогу більш ефективно використовувати результати попередніх робіт під час розроблення нової програми й створювати бібліотеки готових рішень для задач, які часто трипляються в програмуванні. У складі кожної програми мовою Сі є щонайменше одна функція з назвою `main`, з якої розпочинається виконання програми. Визначення функції має такий вигляд:

```
<тип_результату> <ім'я_функції> (<декларація аргументів>){
   тіло функції, декларації та інструкції;
}
```

Деяких частин визначення може не бути. Наприклад, функція може не мати аргументів, а також може бути випущений тип результату, тоді, за замовчуванням, тип результату **int**.

Назва (ім’я) функцій повинно починатися з літери або символу «_» і містити тільки літери, цифри або символи «_»). Функція може не містити параметрів, проте круглі дужки при визначенні функції завджди треба вказувати.

Визначення функції може міститись у будь-якому файлі проєкту. Якщо функцію потрібно виконати в іншому файлі, в ньому повинен бути опис функції:

```
<тип_результату> <ім’я_функції> (<декларація аргументів>);
```

Порядок і тип аргументів та тип результату у визначенні й описі функції в мові Сі повинні збігатися. Згідно з термінологією – імена змінних, які використовують для визначення функції, називають параметрами, а змінні, які передаються під час виклику функції, називають аргументами.

Для того щоби запустити функцію на виконання, слід виконати оператор виклику функції, який має такий вигляд:

```
<ім’я_функції> (<параметри виклику функції>);
```

Типи параметрів виклику та їхній порядок повинні бути узгоджені з типами й порядком аргументів визначеної функції.

В Python функції визначаються за допомогою ключового слова `def`. Для того щоби створити функцію, потрібно розмістити ключове слово `def` перед ідентифікатором функції (її ім’ям), потім вказати пару дужок в середині яких можуть міститися імена змінних (аргументи функції), в кінці рядка дві крапки. Після двох крапок (в новому фізичному ряду) з відступами в чотири знаки символу пустого рядка розміщують блок інструкцій, який є тілом функції, наприклад:

```
def sayHello():
    #цей код належить функції.
    #він виконується кожен раз, коли функція викликається.
    #його ще називають тілом функції
    print('Hello world!')
    #кінець функції

sayHello() # запускаємо функцію
```

Тут визначено функцію, яка називається `sayHello`. Ця функція не приймає жодних параметрів, тому в дужках не оголошено жодної змінної. Параметри для функцій – це вхідні дані, які функція якимось чином опрацьовує. Це параметри, змінні значення яких визначаються, коли запускають функцію на виконання. Щоби виконати (запустити)

функцію, вказують її ім’я та вживають дужки з відповідними параметрами (аргументами) або без них.

В Python, на відміну від Сі, визначення функції має міститися безпосередньо перед викликом функції.

Змінні, оголошені всередині визначення функції, жодним чином не належать до змінних з такими самими іменами, які знаходяться зовні функції. Іншими словами: імена змінних є локальними відносно функції (вони існують тільки під час виконання функції, всередині якої вони містяться). Це називається областю видимості змінних. Всі змінні мають область видимості, відповідну області видимості блока, всередині якого вони оголошені, починаючи від того місця, де визначено ім’я змінної. Змінні, оголошені поза межами функції чи класу, є глобальними для модуля (програми) і доступні з будь-якого місця всередині цього модуля.

Приклад для Python:

```
x = 50
def func(x):
    print('x is', x)
    x = 2
    print('Changed local x to', x)
func(x)
print('x is still', x)
```

Першого разу, коли було використане значення змінної **x**, Python взяв значення параметра, оголошеного у функції. Далі число «2» надано змінній **x**. Ім’я **x** є локальним для функції, тому, коли ми змінюємо значення **x** всередині функції, **x**, визначений в головному блоці, залишається незмінним. Останній виклик функції **print** вивів значення **x** з головного блока, таким чином ми переконалися, що значення змінної не змінилося.

Якщо потрібне значення змінної, яка визначена на найвищому рівні програми (тобто не всередині функції чи класу), тоді в Python потрібно вазати, що застосовується ім’я не локальне, а глобальне. Цього досягають за допомогою оператора **global**. Неможливо надати значення змінній, яка визначена ззовні функції із самої функції, не використавши оператор **global**.

Коли у функції з великою кількістю параметрів потрібно визначити тільки деякі з них, тоді можна надати значення таким параметрам, іменуючи їх, це називається іменованими аргументами, коли використовують ім’я (ключове слово) замість позиції для того, щоби

вказати аргументи для функції. Такий підхід має дві переваги: перша – використання функцій стає легшим, оскільки не потрібно перейматися послідовністю аргументів. Друга причина – надавати значення можна тільки тим параметрам, яким потрібно, за умови, що інші параметри мають типові значення.

Приклад для Python:

```
def func(a, b=5, c=10):
    print('a is', a, 'and b is', b, 'and c is', c)
func(3, 7)
func(25, c=24)
func(c=50, a=100)
```

Результат виконання функції:

```
a is 3 and b is 7 and c is 10
a is 25 and b is 5 and c is 24
a is 100 and b is 5 and c is 50
```

В наведеному прикладі функція з іменем `func` має один параметр без типового значення аргументу, за яким ідуть два параметри з типовими значеннями аргументів. В першому виклику (`func(3, 7)`) параметр **a** отримує значення 3, параметр **b** – значення 7, а параметр **c** – типове значення 10. В другому виклику цієї ж функції (`func(25, c=24)`) змінна **a** отримує значення 25 через позицію аргументу, і параметр **c** набуває значення 24 (були використані іменовані аргументи). Змінна **b** теж набуває свого типового значення, визначеного під час оголошення функції. В третьому випадку (`func(c=50, a=100)`), коли використано лише іменовані аргументи. Значення цих змінних будуть визначені.

Інколи виникає потреба визначити функцію, яка може приймати довільну кількість аргументів (список аргументів довільної довжини). В такому разі застосовують зірочку.

Приклад для Python:

```
def total(initial=5, *numbers, **keywords):
    count = initial
    for number in numbers:
        count += number
    for key in keywords:
        count += keywords[key]
    return count

print(total(10, 1, 2, 3, vegetables=50, fruits=100))
```

Тут оголошено параметр, якому передує символ зірочки, такий, наприклад, як ***numbers**, тоді всі позиційні аргументи починаючи з цієї позиції і до кінця будуть зібрані як список з іменем «**numbers**». Схожою є ситуація і тоді, коли оголошено параметр з двома символами «**», наприклад, ****keywords,** тоді всі іменовані аргументи, починаючи з цієї позиції і до кінця будуть зібрані в словник з іменем «**keywords**».

Якщо потрібно вказати, що певні іменовані параметри мають бути доступні тільки як іменовані параметри, а не як позиційні аргументи, тоді їх можна оголосити після параметра з зірочкою.

Приклад для Python:

```
def total (initial=5, *numbers, vegetables):
    count = initial
    for number in numbers:
        count += number
        count += vegetables
    return count
print(total(10, 1, 2, 3, vegetables=50))
print(total(10, 1, 2, 3))
```

Під час запуску такої фукнції буде:

```
$ python keyword_only.py
66
Traceback (most recent call last):
File "test.py", line 12, in
print(total(10, 1, 2, 3))
TypeError: total() needs keyword-only argument vegetables
```

Оголошення параметрів після параметра із символом «*» призводить до того, що ці параметри будуть доступні тільки як іменовані аргументи. Якщо для цих аргументів не вказано типове значення, то виклик такої функції призведе до помилки, якщо не вказано значення для іменованого аргументу (як це показано в прикладі, що зумовлене відсутністю параметра `vegetables`).

Якщо потрібні тільки ключові аргументи, але немає потреби в параметрі з зірочкою, тоді можна просто вказати лише символ «»* без імені параметра: `def total(initial=5, *, vegetables).`

Оператор `return` використовується для того, щоб повернути значення з функції у програму, яка викликала цю функцію.

Приклад для Python:

```
def maximum(x, y):
    if x > y:
```

```
        return x
    else:
        return y
```

Оператор `return` в Python без значення, еквівалентний конструкції `return None`, де `None` – спеціальний тип в Python, який використовується, наприклад, для позначення того, що змінна не має значення. Кожна функція в Python неявно містить в собі інструкцію `return None`.

Оператор `pass` використовується в Python для позначення порожнього блоку операторів. Python дає змогу передавати результати виконання функцій іншим функціям (як аргументи) без використання додаткових змінних.

Отже, функції є зручним способом організувати об'ємний фрагмент коду в більш зручні окремі блоки, тобто для виконання декомпозиції коду. Результат запуску функції можна надати змінній та використовувати його як операнд математичних виразів, тобто утворювати складніші вирази.

Існує багато стандартних функцій, таких, наприклад, як `abs(<number>)`, яка приймає на вхід один аргумент і повертає модуль переданого їй числа. Існують і специфічні функції для конкретної мови програмування, як, наприклад, функція `zip()` для мови Python, яка повертає кортежі, що складаються з відповідних елементів, тобто аргументів-послідовностей. Це дає змогу утворювати великі діапазони, не використовуючи при цьому всю пам'ять комп'ютера і не обриваючи виконання програми. Якщо опустити значення start, тоді діапазон почнеться з нуля. Потрібне значення end, яке визначає останнє значення, буде створене прямо перед зупинкою, а значення step, за замовчуванням, дорівнює одиниці, але змінити його можна, наприклад, на –1.

Класи надають засоби для об'єднання даних і функціональних можливостей. Створення нового класу створює новий тип об'єкта, що дає змогу створювати нові примірники цього типу. Кожен екземпляр класу може мати атрибути, приєднані до нього для підтримки його стану. Екземпляри класу також можуть мати методи (визначені його класом) для зміни свого стану. Класи Python забезпечують усі стандартні функції об'єктноорієнтованого програмування: механізм успадкування класів дає змогу створювати кілька базових класів, похідний клас здатен перевизначати будь-які методи свого базового

класу або класів, а метод може викликати метод базового класу з тим самим іменем. Об'єкти можуть містити довільні обсяги та типи даних. Як і для модулів, класи мають динамічну природу Python: вони створюються під час виконання та можуть бути змінені далі після створення.

Найпростіша форма визначення класу має такий виглядає:

```
class ClassName:
    <statement-1>
    .
    <statement-N>
```

Визначення класів, як і визначення функцій (оператори `def`), мають бути виконані, перш ніж вони матимуть будь-який ефект. Оператори всередині визначення класу зазвичай є визначеннями функцій. Коли вводиться визначення класу, створюється новий простір імен, який використовується як локальна область, – таким чином усі призначення локальним змінним переходять до цього нового простору імен.

Для отримання доступу на посилання до атрибутів класу використовують стандартний синтаксис, який застосовується для всіх посилань на атрибути в Python: `obj.name`. Дійсні імена атрибутів – це всі імена, які були в просторі імен класу під час створення об'єкта класу. Отже, якщо визначення класу в Python мало такий вигляд:

```
class MyClass:
    """A simple example class"""
    i = 12345

    def f(self):
        return 'hello world',
```

то `MyClass.i` і `MyClass.f` є дійсними посиланнями на атрибути, які повертають ціле число та об'єкт функції відповідно, проте вивід `MyClass.i` надасть очікуваний вивід значення, тоді як `MyClass.f` надасть посилання на функцію, а не результат роботи самої функції. Атрибути класу також можна призначити, змінивши значення за посиланням `MyClass.i`. Так само `__doc__` є дійсним атрибутом, який повертає рядок документації.

Клас в Python є загальним і абстрактним поняттям, який являє собою набір інструкцій того, як потрібно працювати з даними всередині класу або із зовнішніми даними. Для кожного класу може бути створений його екземпляр, який називають об'єктом. Для створення об'єктів класу

(інстанціювання) треба запустити функцію конструктора об’єктів цього класу, яка поверне новий об’єкт – екземпляр класу, наприклад, для вжезгаданого класу, процедура матиме такий вигляд:

```
x = MyClass()
```

Запуск конструктора має такий вигляд, як запуск класу з круглими дужками, а результатом виконання інструкції буде створення нового екземлпляра класу, який буде записаний в змінну `x`. В наведеному прикладі інстанціювання об’єкта класу `MyClass` буде створений об’єкт з логікою та параметрами, які визначено в описі цього класу. Об’єкти класів можна створювати з налаштуваннями на деякий початковий стан. Для ініціалізації обєкта класу потрібно визначити функцію-конструктор класу, яка називається методом `__init__(self)`, наприклад:

```
class Complex:
    def __init__(self): #функція-конструктор об’єкта
        self.data = []
```

Метод `__init__` може мати аргументи для більшої гнучкості. У цьому випадку аргументи, надані оператору ініціалізації об’єкта класу, передаються до `__init__()`, наприклад:

```
class Complex:
    def __init__(self, realpart, imagpart):
        #self - це посилання на інстанс даного класа
        self.r = realpart   #створення змінної (поля) r
        self.i = imagpart   #створення змінної (поля) i
x = Complex(3.0, -4.5)
print(x.r, x.i)
```

Вивід буде таким:

```
3.0, -4.5
```

Ключеве слово `self` в методі `__init__` є обов’язковим, воно вказує на посилання на екземпляр даного класа. Для створення об’єкта класу в Python є й інші методи, які мають спеціальне призначення.

Кожен об’єкт має три групи атрибутів:

– ім’я, яке унікально ідентифікує його в домашньому просторі імен (хоча також можуть бути деякі анонімні об’єкти);

– набір індивідуальних властивостей (даних), які роблять його оригінальним та унікальним (хоча можливо, що деякі об’єкти можуть взагалі не мати властивостей);

– набір методів для виконання певних дій, здатних змінювати сам об’єкт або деякі інші об’єкти.

Атрибути даних в Python можна не оголошувати, як і локальні змінні, вони створюються, коли їх вперше визначають. Наприклад, якщо «x» є екземпляром `MyClass`, який створено раніше, наступний фрагмент коду є коректним:

```
x.counter = 1   #створення поля об'єкта
while x.counter < 10:
    x.counter = x.counter * 2
print(x.counter)
del x.counter
```

Проте така поведінка з атрибутами класу не завжди є прийнятною, оскільки, по-перше, створене поле об'єкта таким чином належатиме конкретному об'єкту, в іншого об'єкта з цього ж класу конструктора такого поля не буде, по-друге, зазвичай для кожного об'єкта заданого класу необхідно мати ряд власних атрибутів, які неможливо було б змінити перевизначенням цього атрибуту в об'єкт. В такому разі в класі треба створити приватні атрибути, до яких не буде доступу ззовні. Наприклад, розглянемо код:

```
class Stack:
    def __init__(self):
        #створення публічного поле інстансу даного класу
        self.__stack_list=[]
```

В цьому прикладі створено клас з назвою `Stack`, в якому через конструктор описано поле `__stack_list`, яке є порожнім списком. Подвійне нижнє підкреслення перед назвою атрибута вказує на те, що він є приватним, тобто зміна цього атрибута не доступна поза класом в разі прямого звернення до атрибуту через об'єкт такого класу.

Якщо потрібно створити поле, яке належатиме тільки класу, тобто буде спільним для всіх об'єктів цього класу, то опис такого поля в класі матиме вигляд:

```
class Stack:
    id = 0               #публічне поле класу
    __serial_id = 1F45Q  #приватне поле класу
```

Створюючи класи в Python, можна застосувати механізм спадкування, тобто зробити клас наступником іншого класу. За такого підходу клас-натупник отримає всі атрибути суперкласу – тобто класу, від якого відбувається спадкування. Механізм спадкування класу в Python реалізується таким чином:

```
class AddingStack(Stack): #в дужках вказується суперклас
    pass
```

Цей клас не визначає жодного нового компонента, але це не означає, що він порожній. Він отримує всі компоненти, визначені його суперкласом, – ім'я суперкласу пишеться перед двокрапкою безпосередньо після імені нового класу. Додавання нової властивості до класу виконується конструктором.

```
class AddingStack(Stack):
    def __init__(self):
        Stack.__init__(self)
        self.__sum = 0
```

На відміну від багатьох інших мов, Python змушує явно викликати конструктор суперкласу. Пропуск цього пункту матиме шкідливі наслідки – об'єкт буде позбавлений списку __stack_list. Такий стек не працюватиме належним чином. Це єдиний раз, коли можна явно викликати будь-який з доступних конструкторів – це можна зробити всередині конструктора підкласу так:

– вказують назву суперкласу (клас, конструктор якого потрібно запустити);

– ставлять крапку (.), після цього вказують назву конструктора з посиланням на об'єкт (екземпляр класу), який має бути ініціалізований конструктором, – ось чому потрібно вказати аргумент і використовувати змінну `self` (для виклику будь-якого методу (зокрема з конструкторами) поза класом не потрібно вказувати аргумент `self` в списку аргументів – виклик методу з класу потребує явного використання аргументу `self`, тож його потрібно поставити першим у списку).

Рекомендується викликати конструктор суперкласу перед будь-якими іншими ініціалізаціями, які потрібно виконати всередині підкласу.

Приклад коду на Python

– суперклас

```
class Stack:
    def __init__(self):
        self.__stack_list = []

    def push(self, val):
        self.__stack_list.append(val)

    def pop(self):
        val = self.__stack_list[-1]
        del self.__stack_list[-1]
        return val
```

– клас-наслідник

```
class AddingStack(Stack):
    def __init__(self):
        Stack.__init__(self)
        self.__sum = 0

    def push(self, val):
        self.__sum += val
        Stack.push(self, val)
    def pop(self):
        val = Stack.pop(self)
        self.__sum -= val
        return val

    def get_sum(self):
        return self.__sum
```

Python на відміну від інших мов програмування підтримує парадигму множинного спадкування, для цього всі класи-наступники зазначають в класі-наступнику.

Іншим типом посилання на атрибут екземпляра є *метод*. Метод – це функція, яка «належить» об'єкту (наприклад, об'єкти списку мають методи, що називаються додаванням, вставленням, видаленням, сортуванням тощо). Є одна фундаментальна вимога – метод повинен мати принаймні один параметр (методів без параметрів немає – метод можна викликати без аргументу, але не оголошувати без параметрів). Перший (або єдиний) параметр зазвичай називається `self`. Якщо потрібно, щоб метод приймав параметри, їх потрібно розміщувати після ключового слова `self`:

```
class Classy:
    #декларація методу без параметрів
    def method(self):
        print("method")
    #декларація методу з параметром par
    def method(self, par):
        print("method:", par)
```

Як відомо, метод `__init__`, відповідний за створення об'єкта в Python, призначений для повернення створеного об'єкта. Якщо під час деклалації класу вказати декілька методів `__init__`, то основним методом контруктора буде той, який містить найбільшу кількість параметрів, а інші будуть призводити до помилки запуску програми. Також можна вказати метод-коструктор з параметрами за

замовчуванням, тоді в процесі створення об’єкта буде можливість не вказувати деякі параметри, які в такому разі набуватимуть стандартних значень, наприклад:

```
class Classy:
    def __init__(self, value = None):
        self.var = value

obj_1 = Classy("object") #запуск коструктора з параметрами
obj_2 = Classy()         #запуск коструктора без параметрів
```

Якщо все-таки потрібно сконструювати клас з невизначеною кількістю параметрів конструктора, то можна застосувати конструкцію змінної кількості параметрів, яка працюватиме не лише в методі-кострукторі, а й у будь-якому методі класу:

```
class One:
    #*args - конструкція змінної кількості параметрів
    def __init__(self, *args):
        if len(args) == 0:
            self.name = ""
            self.age = 0
        elif len(args) == 1:
            self.name = args[0]
        elif len(args) == 2:
            self.name = args[0]
            self.age = args[1]

obj_1 = One()
obj_2 = One("Olena")
obj_3 = One("Maxim", 10)
print(obj_1.name)
print(obj_2.name)
print(obj_3.name)
```

Якщо в класі треба визначити приватний метод, то ім’я методу повинно починатися з подвійного підкреслення (__) так само, як і для полів класу.

Для перегляду атрибутів класу та об’єктів в Python є вбудований метод `__dict__`, який повертає словник, в якому ключами будуть назви полів, методів та інших атрибутів, а значеннями – їхній поточний стан (це можуть бути дані або посилання) .

Поле `__name__`, яке є в класу, містить рядок назви класу і не може бути визначене в об’єкта класу, а властивість під назвою `__bases__` поверне кортеж, що містить суперкласи класу.

Зазвичай метод викликають одразу після його зв'язування:

```
x.f()
```

Проте не обов'язково викликати метод одразу: `x.f` є об'єктом методу, його можна зберегти та викликати пізніше. Наприклад:

```
xf = x.f
while True:
    print(xf())
```

Загалом виклик методу зі списком *n* аргументів еквівалентний виклику відповідної функції зі списком аргументів, який створюється шляхом вставляння об'єкта екземпляра методу перед першим аргументом.

Розглянемо код в Python:

```
class ExampleClass:
    def __init__(self, val = 1):
        self.first = val

    def set_second(self, val):
        self.second = val

example_object_1 = ExampleClass()
example_object_2 = ExampleClass(2)
example_object_2.set_second(3)
example_object_3 = ExampleClass(4)
example_object_3.third = 5
print(example_object_1.__dict__)
print(example_object_2.__dict__)
print(example_object_3.__dict__)
```

Створені об'єкти Python отримують невеликий набір попередньо визначених властивостей і методів, наприклад:

- клас під назвою `ExampleClass` має конструктор, який безумовно створює змінну екземпляра з іменем `first` і встановлює їй значення, передане через перший аргумент (з погляду користувача класу) або другий аргумент;
- клас також має метод, який створює іншу змінну екземпляра з назвою `second`;

Створені в прикладі вище об'єкти класу `ExampleClass` відрізняються: `example_object_1` має лише властивість, названу першою; `example_object_2` має дві властивості: першу та другу; `example_object_3` збагачено властивістю з іменем `third` прямо на льоту, поза кодом класу – це можливо і цілком допустимо.

Результат програми чітко показує:

```
{'first': 1}
{'second': 3, 'first': 2}
{'third': 5, 'first': 4}
```

Таким чином, зміна атрибута екземпляра будь-якого об'єкта не впливає на всі інші об'єкти, оскільки змінні екземпляра ізольовані одна від одної. Така властивість називається інкапсуляцією.

Стандартний метод Python `__str__` застосовують для того, щоби можна було відображати обєкти класу у вигляді довільного рядка, тому цей метод рекомендується імплементувати для кожного створеного класу індивідуально, оскільки стантартна його реалізація призводитиме до відображення адресу посилання на обєкт. Метод `__str__` у обєктів запускається автоматично після прямого звернення до назви обєкта.

Більший код означає більшу складність обслуговування, оскільки пошук помилок завжди легший там, де код менший, тому в багатьох мовах програмування є механізми розділення коду на складові частини (модулі). Механізм модульності дає змогу розподілити написання коду між розробниками, а в проєкті з'єднати всі створені частини в одне робоче ціле.

В Python модуль визначається як файл, що містить визначення та оператори, які пізніше можна імпортувати та використовувати за потреби. Обробка модулів складається з двох потреб:

– перша (найпоширеніша), коли потрібно використовувати вже наявний модуль;
– друга виникає, коли потрібно створити новий модуль.

В Python є модулі з вбудованими функціями, які утворюють стандартну бібліотеку. Кожен модуль складається з сутностей функцій, змінних, константант, класів та об'єктів. Для того щоби модуль був придатним для використання, його потрібно імпортувати. Найпростіший спосіб імпортувати окремий модуль – це використати інструкцію імпорту, наприклад:

```
import math
```

Інструкція імпорту може бути розміщена в будь-якому місці коду, але її потрібно розмістити перед першим використанням будь-якої сутності модуля.

Для імпорту більш ніж одного модуля бажано виконати імпорт, повторивши пункт імпорту:

```
import math
```

```
import sys
```

або перерахувавши модулі після ключового слова import, як тут:

```
import math, sys
```

Якщо модуль із вказаною назвою існує та доступний (модуль насправді є вихідним файлом Python), тоді Python імпортує його вміст, тобто всі імена (змінні та функції), визначені в модулі, стають відомими, проте вони не входять у простір імен коду, куди здійснено імпорт. Це означає, що можна мати власні сутності з іменами, наприклад `sin` або `pi`, а імпорт модуля `math` жодним чином не вплине на ці змінні, хоча містить такі ж назви змінних всередині модуля. Для того щоби скористатися змінними з модуля `math`, треба виконати звернення до таких змінних через точкову нотацію і назву модуля, тобто:

```
math.pi
math.sin
```

Інший синтаксис імпорту точно вказує, яку сутність модуля (або сутності) слід передати в код:

```
from math import pi
```

Інструкція складається з таких елементів: ключове слово `from`, ім'я модуля, який імпортується, ключове слово `import` та ім'я або список імен (сутності/сутностей), які імпортуються в простір імен заданого коду. Такий імпорт призведеде до того, що тільки згадані сутності будуть імпортовані з зазначеного модуля, а доступ до імпортованих об'єктів буде відбуватися безпосередньо через ім'я сутності без звернення до назви модуля.

Третій спосіб імпорту є більш агресивною формою представленого раніше імпорту, наприклад:

```
from module import *
```

Така інструкція імпортує всі сутності із зазначеного модуля. Цей спосіб імпорту є зручним оскільки не зобов'язує називати всі потрібні імена сутностей, але інколи він не прийнятний, тому що невідомо про всі назви, надані модулем, які будуть імпортовані, що в подальшому може призвести до конфліктів імен коду, куди був здійснений такий імпорт і де траплятимуться сутності з такими самими іменами.

Якщо в Python використовується варіант імпорту модуля без зазначення сутностей (перший варіант), можна скористатися додатковою можливістю переіменувати модуль, надавши йому так звний аліас (псевдонім). Наприклад, імя модуля збігається з однією з

визначених сутностей, тому модулю в коді можна надати нове імя, при цьому зміна працюватиме лише в коді, де визначено таку поведінку, а глобально для інших модулів ім'я модуля не зміниться:

```
import module as alias
```

Якщо використовується варіант імпорту сутності з модуля і потрібно змінити назву сутності, також можна застосувати псевдонім, але для сутності. Це призведе до того, що ім'я сутності з модуля буде замінене на обраний псевдонім:

```
from module import name as alias
```

Крім модулів в Python, які призначені для об'єднання деяких пов'язаних сутностей, таких як функції, змінні або константи, можна застосовувати пакети – контейнери, які дають змогу об'єднувати декілька пов'язаних модулів під одним спільним ім'ям. Пакети можуть поширюватися у вигляді каталогів або архівів в zip-файлі.

Для створенння власного модуля в Python потрібно створити файл з розширенням .py та помістити його в будь-яке місце (бажано для власних модулів визначати окремі каталоги). Для того щоб імпортувати власний модуль до основного коду, потрібно скористатися однією з інструкцій імпорту, при цьому слід вказати повний шлях до потрібного модуля. Якщо файл з модулем знаходиться поряд з файлом, куди здійснюється імпорт, то імпорт матиме, наприклад, такий вигляд:

```
import module
```

Та якщо файл модуля розміщено в каталозі, який також поряд із виконуваним файлом, то імпорт буде, наприклад, такий:

```
import directory.module
```

Файл модуля в Python ніяким чином не відразняється від звичайного файла з кодом, тому часто у файлах модулів вказують коментарі документації, а також конструкції з перевіркою системної змінної `__name__`:

```
if __name__ == "__main__":
   print("I prefer to be a module.")
else:
   print("I like to be a module.")
```

Замість інструкцій `print` можуть бути інструкції з тестами коду цього модуля або дані поля взагалі не вказують.

Системна змінна Python `__name__` залежно від контексту запуску файла з цією змінною набуватиме значення назви модуля або рядка «`__main__`». Наприклад, якщо беспосередньо запустити файл зі

змінною __name__, яка записана в цьому файлі, то така змінна автоматично набуде значення __main__, і навпаки, якщо запустити файл через імпорт його змісту в інший файл (тобто застосувати файл як модуль), тоді така змінна набуде значення назви модуля, в якому цю змінну застосовано. Якщо створити файл module.py в якому буде лише одна інтрукція

```
print(__name__),
```

то після запуску такого файла, буде отриманий наступний вивід в консоль

__main__.

Якщо файл проімпортувати в іший файл, наприклад main.py, з таким кодом:

```
import module,
```

то після запуску файла main.py вивід в консоль буде таким:

module.

Розглянутий підхід дає змогу застосовувати код модуля лише в разі його імпорту та ігнорувати цей модуль, якщо відбувається безпосередній запуск коду з модуля.

Після першого імпорту модуля разом з файлом модуля з'явиться новий вкладений каталог з назвою __pycache__, всередині якого буде розміщений автоматично скомпільований Python-код для цього модуля, у файл із назвою module.cpython-xy.pyc, де x і y є цифрами, отриманими з версії Python. У разі зміни коду модуля Python автоматично під час наступного запуску модуля перекомпілює файл модуля, але якщо змін в коді модуля не було, то в наступних імпортах модуля буде виконуватися саме скомпільований файл.

Імпортуючи модуль, Python завжди спочатку перевіряє, чи був цей модуль завантажений раніше, і якщо цього не зроблено, завантажує всі інструкції модуля (саме тому виконано інтрукцію модуля з друком в попередньому прикладі). Для повторного імпорту модуля його інструкції повторно не заватнажуються.

Якщо файл модуля буде розміщуватися в окремому каталозі з деяким шляхом, який відрізняється від шляху до виконуваного файла, то системній змінній sys.path Python потрібно виконати примусовий імпорт шляху, де розміщено імпортований модуль. Розглянемо приклад на рис. 6.1, де показано, що в каталозі module зі шляхом C:\Users\user\ py\modules розміщено файл з модулем

module.py, а головну програму main.py – в каталозі progs зі шляхом C:\Users\user\py\ progs. Імпортуючи модуль, Python виконує пошук ресурсів в папках, які містяться в списку змінної path, і якщо серед шляхів для пошуку в списку немає потрібного, модуль не буде імпортований.

Для включення в список path потрібного шляху до модуля, потрібно в коді програми перед інструкцією імпорту вказати інтрукцію з додавання такого шляху, наприклад:

```
from sys import path
path.append('..\\modules')
```

або

```
path.append('C:\Users\user\ py\modules')
import module
print(module.funk())
```

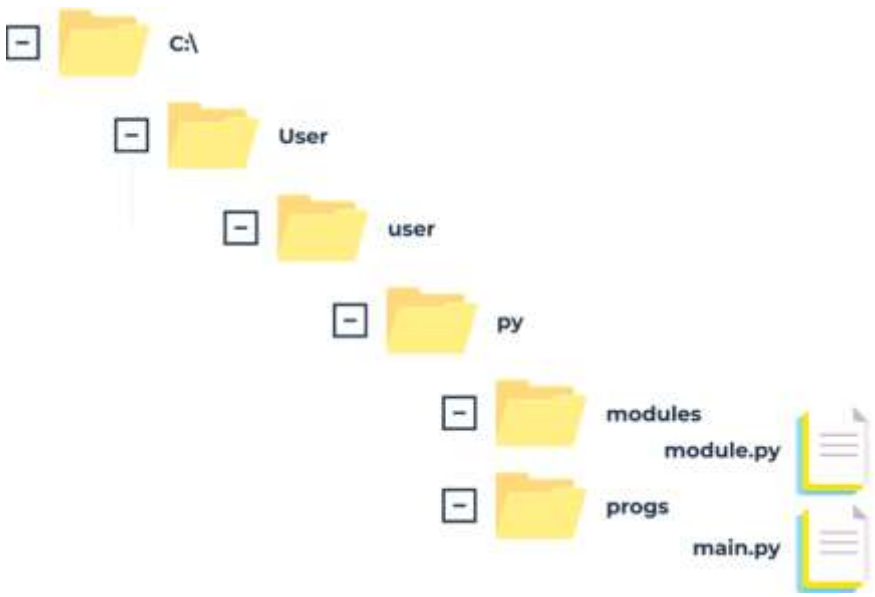


Рис. 6.1. Приклад шляху розміщення каталогу з модулями Python

Як видно з представленого фрагменту коду на Python, шлях можна додати або відносний, або абсолютний. При цьому «. .» означає перехід на каталог вище в ієрархії шляху, а подвійний символ «\» застосовується для екранування наступного символу «\».

Якщо структура каталогу з модулями складна, наприклад, як на рис. 6.2, тоді потрібно використати інший підхід – Python очікує, що в каталозі пакета буде розміщено файл з унікальною назвою: __init__.py. Вміст цього файла буде виконаний під час імпорту будь-якого з модулів пакета, і якщо не потрібно спеціальних

ініціалізацій, можна залишити цей файл порожнім, проте відсутність такого файла спричинить помилку імпорту. Таким чином, наявність файла `__init__.py` в файлі-каталозі модулів визначає такий каталог як пакет.

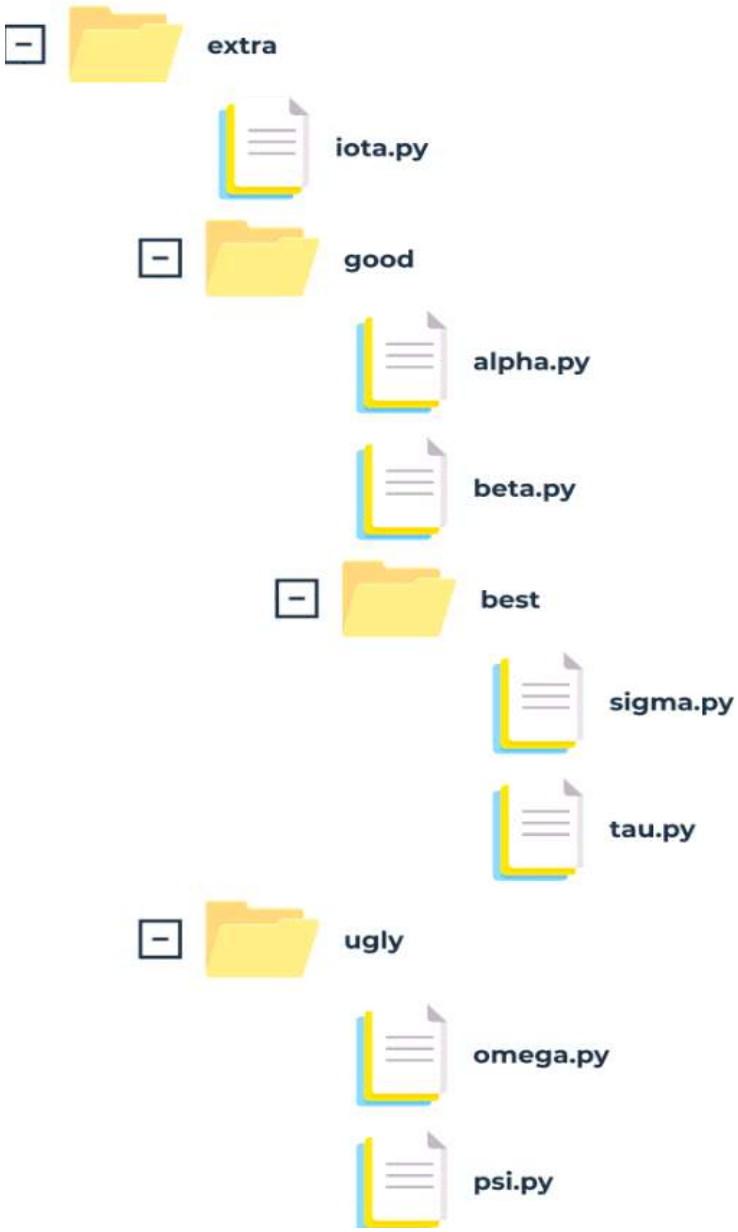


Рис. 6.2. Приклад складної структури каталогу з модулями Python

Python створено як програмне забезпечення з відкритим кодом, що надає підтримку його екосистемі багатьма програмістами. Для розвитку екосистеми Python розроблено централізоване сховище всіх доступних програмних пакетів та інструмент, який дає користувачам змогу отримувати доступ до репозиторію готових пакетів. Основний репозиторій Python називається PyPI (https://pypi.org/ Python Package Index), який підтримує робоча група під назвою Packaging Working

Group. Інструмент, який застосовується для завантаження потрібних пакетів з репозиторію PyPI, називається pip – це спеціальна програма, яка встановлюється разом з встановленням Python, хоча може бути встановлена й окремо.

Для перевірки, чи встановлено pip в операційну систему, треба відкрити командний рядок (термінал) та виконати команду `pip --version`. Якщо pip встановлено, в консоль буде виведена версія pip. Для встановлення стороннього пакета з репозиторію PyPI потрібно в командному рядку виконати команду `pip install <name package>`, де `<name package>` – назва потрібного пакета з репозторію PyPI. Переглянути втановлені пакети у віртуальному середовищі можна за допомогою команди `pip list`.

Видалити встановлену бібліотеку можна, застосувавши команду pip: `pip uninstall < name package > -y`.

Під час встановлення заданого пакета з репозиторію за допомогою команди pip, можуть бути додатково встановлені інші бібліотеки, які не були визначені для встановлення, але від яких може залежати робота вказаного для встановлення пакета. У разі видалення пакета інші пакеті, від яких залежить цей пакет, видалені не будуть.

Для створення текстового файла з відомостями про встановлені в проєкт пакети, можна застосувати команду pip: `pip freeze > requirements.txt`, де `requirements.txt` – це назва текстового файла, куди pip запише всі залежності для проєкту.

Якщо в проєкті вже є файл `requirements.txt` із залежностями, він буде перезаписаний. Файл `requirements.txt` може бути застосований також для того, щоб встановити до проєкту потрібні залежності, для чого слід виконати команду `pip install -r requirements.txt`. При цьому `pip` прочитає вказаний файл та виконає автоматичне встановлення всіх залежностей, зазначеих у файлі.

Поки програма виконується, її дані зберігаються в пам’яті (Random AccessMemory (RAM)), проте коли програма завершується або комп’ютер вимикають, дані з пам’яті стираються. Для постійного зберігання даних треба помістити їх у файл. Файли зазвичай зберігають на жорсткому носії. Для великої кількості файлів часто організовують директорії, тобто каталоги. Кожен файл розпізнається за своїм унікальним ім’ям або за комбінацією імені файла та директорії.

Під час зчитування та запису файлів програми можуть обмінюватися інформацією одна з одною та створювати формати для друку, подібні до PDF. Робота з файлами подібна до роботи з книжками. Для того щоби прочитати книжку, треба спочатку її розгорнути (відкрити), а коли прочитано – закрити. Доки книжка розгорнута (відкрита), її можна або читати, або в неї щось записувати. В обох випадках завжди відомо, про яке місце в книжці відбувається запис. Це стосується й файлів. Щоб відкрити файл, треба вказати його ім'я та визначити, який тип процедури буде виконаний – читання чи запис. Процес відкриття файла в Python ініціює створення файлового об'єкта `file_obj = open(file_name, var_mode)`, де: `file_obj` – об'єкт файла, що повертається функцією `open()`; `file_name` – рядок, що являє собою ім'я файла; `var_mode` – це рядок, який вказує на тип файла і дії, які слід над ним виконати. Перша літера рядка `mode` вказує на процедуру: ''`r`'' – читання; ''`w`'' – запис (якщо вказаного файла не існує, він буде створений, та якщо файл існує, відбудеться його перезапис); ''`x`'' –запис нового файла; ''`a`'' – додавання даних в кінець файла. Друга літера рядка `mode` вказує на тип файла, з яким ми маємо діло: ''`t`'' або нічого означає, що ми маємо діло з текстовим файлом; ''`b`'' – означає, що йдеться про бінарний файл.

Після відкриття файла викликають функції для читання або для запису даних. По завершенні потрібно обов'язково закрити файл. Текстові файли – це файли, які містять друковані символи і пробіли, згруповані в рядки, відокремлені один від одного символами нового рядка. Оскільки мова Python спеціально створена для обробки текстових файлів, вона пропонує методи, які суттєво спрощують роботу з рядками:

```
var_str = """Понеділок, Вівторок, Середа, Четвер, П'ятниця."""
print(len(var_str))
```

Функція write() записує зазначений рядок у файл. Якщо параметром вказано Unicode-рядок, тоді виконується спроба перетворити його в звичайний рядок. Оскільки за замовчуванням використовується кодування ASCII, тоді спроба перетворити Unicode-рядок, що містить українські літери, у звичайний рядок призведе до генерації винятку Unicode Encode Error.

```
var_file = open("some file.txt", "wt")
var_file.write(var_str)
var_file.close()
```

Функція `write()` також повертає кількіть записаних байтів. Вона не додає жодних пробілів або символів нового рядка (\n), як це робить функція `print()`. Функція `print()` також дає змогу записувати дані в текстовий файл:

```
var_file = open("some file.txt", "wt")
print(var_str, file = var_file)
var_file.close()
```

За замовчуванням `print()` додає пробіл після кожного аргументу і символ нового рядка в кінці. Для того щоб `print()` працювала як `write()`, треба передати їй два таких аргументи: `sep` – роздільник, за замовчуванням це пробіл (`' '`); `end` – символ кінця файла, за замовчуванням це символ нового рядка (`\n`).

Функція `print()` завжди використовує значення за замовчуванням, якщо тільки не передати їй щось інше:

```
var_file = open("some file.txt", "wt")
print(var_str, file = var_file, sep = "", end = "")
var_file.close()
```

Щоб повністю зчитати весь файл, використовують функцію `read()` без аргументів, проте слід мати на увазі, що файл розміром 1 Гбайт споживає 1 Гбайт пам'яті відповідно:

```
var_file = open("some file.txt", "rt")
var_days = var_file.read()print(var_days)
var_file.close()
```

Можна вказати максимальну кількість символів, яку `read()` поверне за один виклик:

```
var_file = open("some file.txt", "rt")
var_days = var_file.read(23)
print(var_days)
var_file.close()
```

Після того як прочитано весь файл, подальші виклики функції `read()` будуть повертати порожній рядок. Також досить зручно зчитувати файл по одному рядку одночасно за допомогою функції `readline()`:

```
var_file = open("some file.txt", "rt")
print(var_file.readline())
var_file.close()
```

Для текстового файла пустий рядок має довжину, яка дорівнює одиниці (тобто символ нового рядка), такий рядок буде вважатися `True`.

Коли весь файл буде прочитаний, функція `readline()`, як і `read()`, поверне порожній рядок, який вже буде розцінено як `False`. Найпростіший спосіб зчитати текстовий файл – це використати цикл, що буде повертати по одному рядку:

```
var_file = open("some file.txt", "rt")
for var_line in var_file:
    print(var_line)
var_file.close()
```

Якщо забути закрити файл після завершення роботи над ним, його закриє Python самостійно після того, як буде вилучене останнє посилання на нього. Це означає, що, якщо відкрити файл і не закрити його, тоді він буде закритий автоматично по завершенні функції. Проте бувають випадки, коли файл можна відкрити всередині довгої функції або навіть основного розділу програми. Файл повинен бути обов'язково закритий, щоб всі операції, що залишилися, були завершені.

У Python є менеджер контексту для очищення об'єктів на зразок відкритих файлів.

Винятки – це сповіщення інтерпретатора, які створюються в разі помилки в коді або виникнення якоїсь події. Якщо в коді не передбачено обробки винятків, тоді програма переривається та виводить повідомлення про помилку. Відомо про три типи помилок в програмуванні. Синтаксичні – це помилки в імені оператора або функції, невідповідність початкових та кінцевих лапок тощо, тобто помилки в синтаксисі мови. Як правило, інтерпретатор попереджає про наявність помилки, а програма не виконуватиметься при цьому зовсім. Приклад синтаксичної помилки:

```
print("Невідповідність вікритих та закритих лапок!)
```

Результатом запуску цього коду буде таке:

```
SyntaxError: EOL while scanning string literal
```

Помилки виконання – це помилки, які виникають під час роботи коду програми. Причиною такої помилки є події, які не були передбачені програмістом. Класичним прикладом є ділення на нуль:

```
def test(var_x, var_y):
    returnvar_x/var_y
print(test(4,0))
```

Результат запуску такого коду:

```
Traceback (most recent call last):
   File "<pyshell#4>", line 1, in <module>
```

```
    print(test(4,0))
  File "<pyshell#2>", line 2, in test    return var_x/var_y
ZeroDivisionError: division by zero
```

Семантичні – це помилки в логіці роботи програми, яку можна виявити тільки за результатами тестування програми. Інтерпретатор не попереджає про наявність таких помилок, а програма буде виконуватися, тому що не містить синтаксичних помилок. Такі помилки досить важко виявити та виправити.

В мові Python винятки створюються не тільки внаслідок помилки, але і як повідомлення про настання певної події. Наприклад, метод `index()` генерує виняток ValueError, якщо шуканого фрагмента немає в рядку:

```
"String".index("S")
0
"String".index("s")
Traceback (most recent call last):
  File "<pyshell#65>", line 1, in <module>
    "String".index("s")
ValueError: substring not found
```

Для обробки винятків використовується оператор `try`. Формат цього оператора такий: інструкції, в яких перехоплюються винятки, повинні обов'язково бути розміщені всередині блока `try`. У блоці коду except в параметрі <Виняток_1> вказується клас оброблюваного винятку. Наприклад, для того щоб обробити виняток, що виникає в разі ділення на нуль, потрібно:

```
var_x = 1
try:
    var_x = 1/0
except ZeroDivisionError:
    print("Обхід ділення на нуль.")
    var_x = 0
    print(var_x)
```

Якщо в блоці `try` генерується виняток, то управління передається блоку `except`. У разі, якщо виняток не відповідний зазначеному класу, відбувається передача управління наступному блоку `except`.

Якщо жоден з блоків `except` не є відповідним винятку, то виняток «спливає» до обробника вищого рівня. Якщо виняток ніде не обробляється, управління передається обробнику за замовчуванням, який своєю чергою зупиняє виконання програми та виводить стандартну

інформацію стосовно помилки. Отже, в обробнику може бути декілька блоків except, при цьому з різними класами винятків. Окрім того, один обробник можна вставити в інший:

```
var_x = 1
try:
    try:
        var_x = 1/0
  except NameError:
    print("NameError - невизначений ідентифікатор.")
except IndexError:
    print("IndexError - неіснуючий індекс.")
    print("Вираз після внутрішнього try")
except ZeroDivisionError:
    print("Обхід ділення на нуль.")
    var_x = 0
    print(var_x)
```

Результат виконання:

```
Обхід ділення на нуль.
0
```

Якщо у вкладеному обробнику не вказано виняток ZeroDivisionError, виняток «виринатиме» до обробника вищого рівня. Після оброблення винятку відбувається передача управління інструкції, яка розміщена відразу після обробника. В інструкції except можна також вказати відразу декілька винятків, перерахувавши їх за допомогою коми всередині круглих дужок:

```
var_x = 1
try:
    var_x = 1/0
# Обробка трьох винятків: NameError, IndexError,
ZeroDivisionError
except (NameError, IndexError, ZeroDivisionError):
    var_x = 0
    print(var_x)
```

Якщо в інструкції except не вказано клас винятку, то відповідно такий блок перехоплює всі винятки. На практиці слід уникати порожніх інструкцій except, тому що можна перехопити виняток, який є лише сигналом, а не власне помилкою. Коли в обробнику є блок e1se, тоді інструкції всередині цього блока будуть виконані, тільки якщо немає будь-яких помилок. За потреби виконати завершальні дії незалежно від того, згенеровано виняток або ні, потрібно використувати блок final1y:

```
try:
    var_x = 14/2#var_x = 14/0
except ZeroDivisionError:
    print("/0")
    var_x = 0
else:
    print("else!")
finally:
    print("finally!")
print("x =", var_x)
```

Результат виконання цього винятку:

```
else!
finally!
x = 7.0
```

За наявності винятку та відсутності блока except інструкції всередині блока finally будуть виконані, але при цьому виняток не буде оброблений. Він і далі «виринатиме» до обробника вищого рівня. Якщо обробника немає, управління передається обробнику, який стоїть за замовчуванням, перериває виконання програми і виводить повідомлення про відповідну помилку.

Контрольні запитання

1. Якими мовами програмування програмують роботів?
2. Назвіть основну парадигму програмування мовою Сі.
3. Назвіть основну парадигму програмування мовою Python.
4. Які типи даних є в мові Сі та Python?
5. Як створити змінну мовою Сі чи Python?
6. Що таке «тип» змінної?
7. Скількии памяті потрібно для зберігання змінних в мові Сі?
8. Як створити функцію в мові Сі?
9. Які типи циклів є в Python?
10. Як створити тип даних рядок в мові Сі?
11. Які види колекцій є в Python?
12. Які незмінювані види колекцій є в Python?
13. Що таке клас?
14. Які основні парадигми є в об’єктноорієнтованому програмуванні?
15. Що означає термін «поліморфізм»?
16. Навіщо потрібні об’єкти?

17. Поясніть термін «інкапсуляція».
18. Як створити спадкування в Python?
19. Чи можна створити примітивні змінні в Python?
20. Поясніть термін «метод».
21. Що означає термін «поле класу»?
22. Що означає термін «сигнатура методу»?
23. Які параметри може приймати функція?

Лекція № 7. Візуальне ручне та програмне керування роботом

Програмне забезпечення роботів призначене для:
- управління роботою системи керування роботизованої машини;
- підтримання діалогу з користувачем;
- автоматизації процесу розроблення та відлагодження програм;
- перекладу мов високого рівня програмування на коди машинної мови;
- архівації файлів;
- забезпечення роботи периферійних пристроїв.

Залежно від виду робота, розріняють різні системи програмного керування:
- *циклове програмне керування* – керування виконавчим пристроєм робота, за якому здійснюється програмування послідовності виконання його руху;
- *позиційне програмне керування* – керування виконавчим пристроєм робота, за якого рух його робочого органа відбувається по заданих точках позиціювання без контролю траєкторії руху між ними;
- *контурне програмне керування* – керування виконавчим пристроєм промислового робота, за якого рух його робочого органа відбувається заданою траєкторією зі встановленим розподілом у часі параметрів швидкості;
- *адаптивне програмне керування* – керування виконавчим пристроєм робота з автоматичною зміною керівної програми залежно від контрольованих параметрів стану зовнішнього середовища.

Основний принцип циклового позиційного керування полягає в позиціюванні робота відповідно до упорів або дискретних датчиків. У позиційних системах керування в програмі управління задають початкове та кінцеве положення робота або його робочого органа, якщо

його розміщення програмується, а проміжні положення повинні бути відомі в кожен момент часу. Інформація про час та послідовність виконання кроків програми може задаватися переналагоджуваними схемами релейної автоматики. У циклових системах керування команди геометричного обмеження задаються упорами під'єднання кінцевих вимикачів, а перебудова циклу за використання внутрішніх комутаторів зводиться до встановлення перемикачів у відповідні положення.

Відповдно до стандартів програмування роботів може бути виконане одним з методів: навчанням, аналітичним програмуванням або комбінованим способом (рис. 7.1).

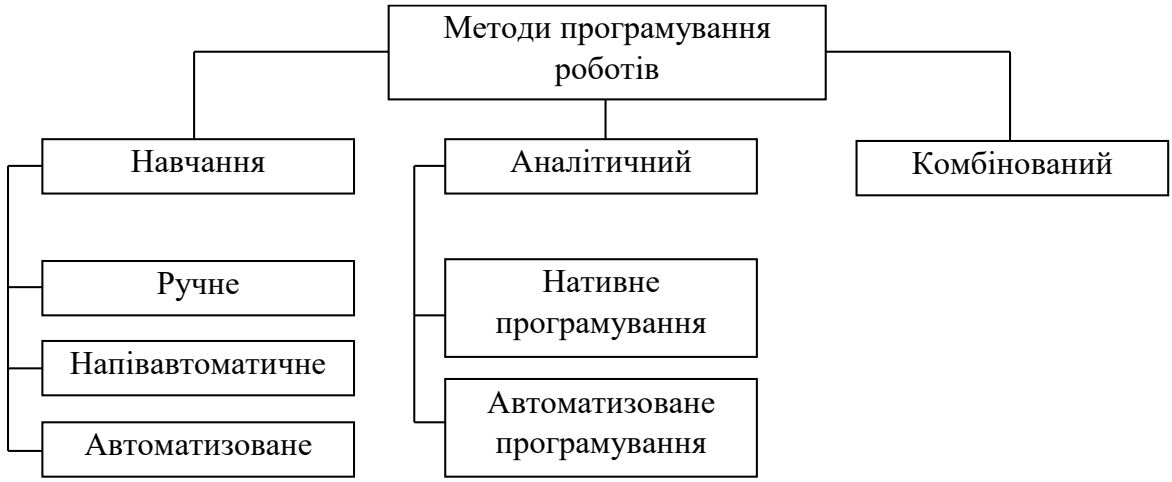


Рис. 7.1. Класифікація методів програмування роботів

Аналітичне програмування робота – це програмування промислового робота, за якого керівну програму складають на підставі розрахунків, а потім встановлюють на пристрій керування.

Навчання робота – це програмування робота, коли складання та введення керівної програми виконує людина-оператор після попереднього руху робочого органа із занесенням в пристрій керування значень параметрів цього руху у вигляді керівної програми, після чого робот може автоматичного повторювати записану програму. Цей метод набув найширшого застосування в промисловій робототехніці. Такий спосіб керування досить простий і не потребує високої кваліфікації оператора.

Метод навчання має суттєвий недолік, який пов'язаний з виникненням суб'єктивної похибки внаслідок неточного позиціювання інструмента в процесі навчання. Така похибка в навчанні залежатиме від

людини, яка виконувала навчання робота, та динамічних властивостей виконавчої системи робота.

Застосовують три види навчання – *ручне*, *напівавтоматичне* та *автоматизоване*.

Ручне навчання можна застосовувати до роботів, у яких привідний механізм дає змогу передавати рух від двигуна до виконавчого пристрою робота в прямому напрямку і в зворотному, не прикладаючи при цьому значних зусиль. В такому разі під час переміщення робота або його елементів у просторі, окрім обертання приводних ланок кінематичної схеми, рухатися повинні також датчики зворотного зв'язку, встановлені для фіксації кожної з узагальнених координат робота. Якщо цієї вимоги не дотримуватися, здійснити ручне навчання неможливо. Одним з недоліків такого способу програмування роботів є низька точність позиціонування або неможливість виконати задану траєкторію та керувати декількома технологічними параметрами. Найбільшого поширення набуло ручне навчання для автоматизації фарбувальних робіт, тому що для цього не потрібна висока точність, а технологія зводиться до одного параметра – ввімкнення та вимкнення розпилювача.

Системи програмного керування роботів повинні мати такі особливості:

– виконувати програмування методом навчання;
– містити значну кількість входів/виходів для зв'язку з основним і допоміжним устаткуванням;
– мати додаткові модулі вимірювання показників стану механізмів робота і параметрів зовнішнього середовища;
– мати модулі діагностики для реалізації функцій диспетчеризації і контролю роботи устаткування і пристроїв, приєднаних до робота;
– мати спеціальне математичне забезпечення;
– забезпечувати керування величинами дискретизації;
– містити наявні спеціалізовані цикли для роботи конкретного робота (завантаження – розвантаження устаткування);
– мати системи для розпізнавання і вимірювання зовнішніх об'єктів;
– забезпечувати можливість адаптивного керування.

Програмування робота завжди виконують на його абстрактному представленні. Тоді на основі абстрактного представлення системи керування роботів можна подати у вигляді блоків, кожен з яких окремо

характеризує виконавчу, інформаційну та програмну частини, відповідальні за один процес (принци єдиної відповідальності), а саме:
– виконання руху (здійснюється виконавчим механізмом);
– сприйняття зовнішнього середовища (виконується датчиками);
– планування (виконується програмованими контролерами).

Залежно від порядку з'єднання цих процесів (оскільки вони є будівельними блоками, їх називають також примітивами), отримаємо різні архітектури (парадигми) управління роботом з різними властивостями.

Реактивна парадигма управління є найпростішою архітектурою, яку можна створити з описаних примітивних блоків. В такій парадигмі (рис. 7.2) не застосований процес планування, а прямий зв'язок між сприйняттям зовнішнього середовища і виконанням роботи означає, що інформація, яка надходить від датчиків, не впливає на ухвалення рішення щодо планування переміщень виконавчої системи. Стан зовнішнього середовища вважають наперед заданим.

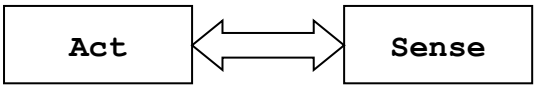


Рис. 7.2. Схема системи з реактивним управлінням: Act – виконавча система; Sense – система сенсорів

Парадигма управління з ієрархічним плануванням полягає в тому, що система керування робота побудована за принципом «заміряв-спланував-виконав». Такій системі керування властива чітка послідовність дій, прив'язана до відомих примітивних структур (рис. 7.3).

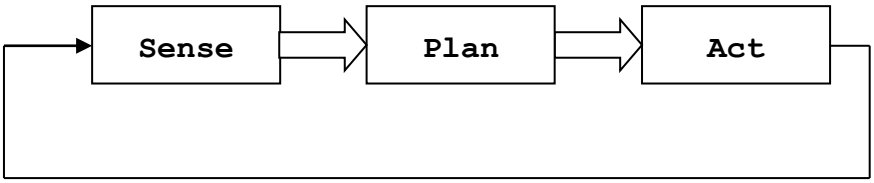


Рис. 7.3. Схема системи з ієрархічним плануванням: Plan – система планування

За таких умов робот не обов'язково може бути дуже швидким, однак завдяки зменшенню швидкості виконання програми така система здатна прогнозувати подальше переміщення за оптимальними

траєкторіями. На рис. 7.3 видно, що цикл планування, виконання та сприйняття є замкнутим. Таким чином, система може циклічно повторювати задану послідовність, активно рухаючись до поставленої мети.

Прадигма гібридного управління є середнім між реактивним управлінням, яке є швидким і гнучким, проте не «розумним», та управлінням ієрархічного планування, «розумним», але досить повільним (рис. 7.4).

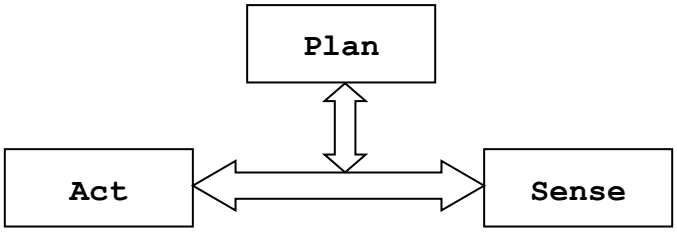


Рис. 7.4. Схема системи гібридного управління

Припустимо, що треба створити робота-офіціанта, який буде подавати напої в кав'ярні. Такому офіціанту обов'язково потрібно мати свій внутрішній простір кав'ярні (де стоять столи, стільці тощо). Щойно такому роботу дали завдання доставити чашку кави клієнту, йому потрібно буде спланувати свій шлях, а потім рухатися цим шляхом. Оскільки всередині простору кав'ярні можуть бути інші гості або перешкоди, потрібно, щоби робот не навштовхнувся на якийсь стілець або стіл, не кажучи вже про випадкові зіткнення з гостем, поки він намагається доставити каву. Для цього потрібний добре налаштований реактивний контролер. Таким чином, роботу спочатку потрібно спланувати своє завдання, розділяючи його на низку дій, які можуть бути виконані реактивною парадигмою. При цьому дані з інформаціної системи будуть доступні для системи планування (оскільки потрібно виконати деяке планування) та для виконавчої системи (оскільки потрібно виконувати реактивний контроль).

Алгоритмізація процесу є одним з фундаментальних понять в програмуванні роботів. Алгоритмом називають послідовність точно визначених дій, що однозначно приводять до виконання поставленого завдання. Алгоритмічна структура – це частина загальної математичної структури і культури мислення, яка зумовлює формування умінь, пов'язаних з розумінням суті поняття алгоритму і його властивостей,

тобто сукупність знань, умінь і навичок, які дають змогу успішно розв'язувати задачі.

Алгоритм складають відповідно до таких вимог:

– бути простим для розуміння, переведення в програмний код та налагодження;
– ефективно використовувати комп'ютерні ресурси і виконуватись якнайшвидше.

З практичного досвіду побудови алгоритмів і їхньої реалізації у вигляді програм випливають такі рекомендації:

- планування етапів розроблення програми – спочатку чорновий варіант алгоритму у неформальному стилі, потім псевдопрограма, далі – послідовна формалізація псевдопрограми, тобто перехід до рівня виконуваного коду;
- використання інкапсуляції, коли всі процедури, що реалізують абстрактні типи даних, подаються в одному місці програмного тексту. Надалі, якщо виникне потреба змінити реалізацію абстрактних типів даних, можна буде коректно і без особливих витрат шляхом внесення змін, виконати налаштування параметрів, оскільки всі потрібні процедури локалізовані в одному місці програми;
- використання та модифікація вже наявних програм. Після створення завершеної програми слід передбачити сфери, де її можна використати.

Кожний алгоритм розробляють під конкретну задачу та для конкретного типу обладнання, на якому цей алгоритм буде дотриманий для заданої мови програмування. Команди алгоритму виконуються послідовно, і на кожному кроці наперед відомо, яка команда буде виконуватися. Для правильної побудови алгоритму треба бути впевненим, що виконання алгоритму завершиться за кінцеву кількість кроків та знати систему команд, яка здатна виконати алгоритм.

У розробленні алгоритму можливе використання різних методів. Перша група методів – це методи типу «зверху вниз», коли відбувається поділ складної задачі на простіші. Такий підхід називається *декомпозицією*. Друга група методів – це методи типу «знизу вверх», які характеризуються виокремленням простих елементів та їхнім поступовим об'єднанням у складні, що називається *розширенням ядра*.

Є три основні способи зображення алгоритмів: словесний спосіб, графічний у вигляді блок-схеми, алгоритмічною мовою (рис. 7.5) або мовою програмування.

Рис. 7.5. Вид алгоритмічної мови прогамування

Блок-схема алгоритму являє собою графічне зображення алгоритму у вигляді спеціальних блоків, за потреби зі словесними поясненнями. Кожен етап алгоритму зображується у формі геометричної фігури (блока), яка має певний вигляд залежно від характеру процедури, що розглядається. Блоки на схемі з'єднуються лініями зі стрілками (лініями зв'язку), які визначають послідовність виконання процедур і, по суті, утворюють логічну схему алгоритму.

Важливою особливістю алгоритмічних структур є те, що вони мають один вхід і один вихід, що дає змогу за потреби створювати алгоритмічні блоки окремо один від одного, а потім з'єднувати їх між собою (вихід однієї базової структури сполучається з входом іншої).

Базові алгоритмічні структури – це структури, за допомогою яких створюється алгоритм для розв'язання певної задачі. Існують три основні (базові) алгоритмічні структури, або три основні типи алгоритмів: лінійний, розгалужений та циклічний.

Лінійний алгоритм (послідовне виконання, структура слідування) – це алгоритм, який дає змогу отримати результат шляхом одноразового виконання послідовності дій незалежно від вхідних даних і проміжних результатів. Дії в таких алгоритмах виконуються послідовно одна за одною, тобто лінійно. В лінійній структурі спочатку виконується дія *P*, після завершення якої виконується дія *Q* (рис. 7.6).

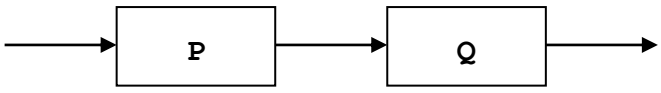
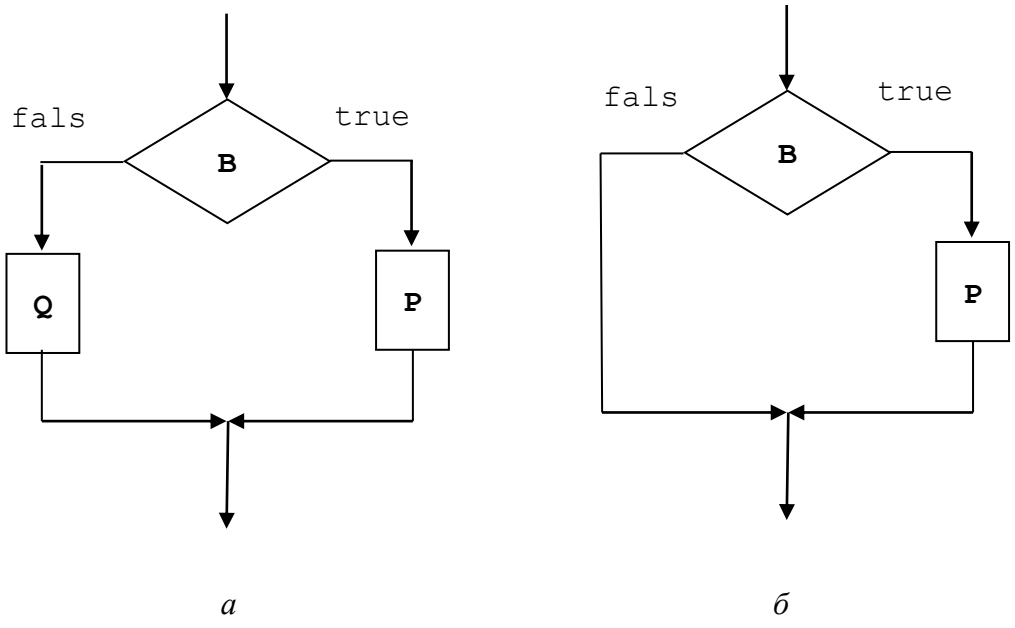


Рис. 7.6. Лінійний алгоритм

Лінійний алгоритм означає одноразове виконання однієї і тієї самої послідовності дій за будь-яких допустимих вхідних даних задачі.

Розгалужений алгоритм (умова або структура вибору) – це структура, що розглядає вибір дій у разі виконання або невиконання заданої умови. Розгалуження може бути повними або неповними (рис. 7.7).

Повне розгалуження – це розгалуження, в якому всі дії визначені для випадку істиності та хибності заданої дії. Неповне розгалуження – це розгалуження, в якому дії визначені тільки для одного випадку (в разі виконання або невиконання умови).



а *б*

Рис. 7.7. Алгоритм з повним (*а*) та неповним (*б*) розгалуженням

Структура «*вибір*» – це така структура, в якій перевіряється умова B, що здатна набувати *n* значень: B1, B2, ..., Bn. Якщо умова набуває значення B_i, тоді виконується дія P_i і робота структури завершується, інакше (умова B не набуває жодного з указаних значень) виконується дія Q і робота структури завершується (рис. 7.8).

У структурі «вибір» дії Q може не бути, тоді за недотримання умови B жодна дія не виконується, робота структури завершується. Можна вважати, що структура «вибір» є розширенням структури розгалуження або, навпаки, структура розгалуження – окремий випадок структури «вибір».

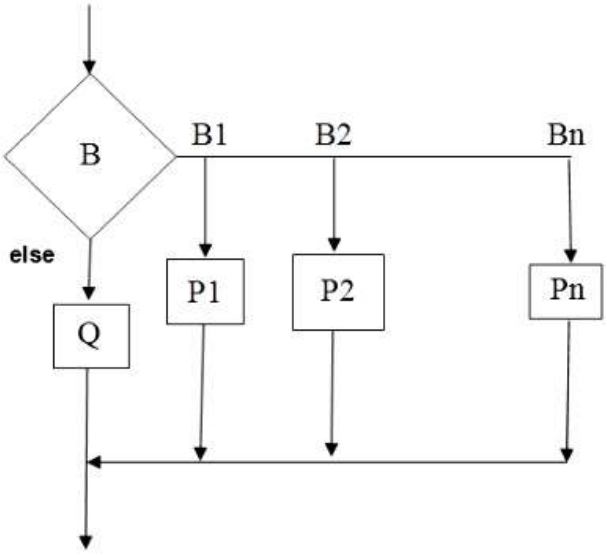


Рис. 7.8. Структура алгоритму «вибір»

Циклічний алгоритм (цикл або структура повторення) – це алгоритм, у якому передбачено повторення деякої серії команд. За допомогою цієї структури описуються однотипні дії, що повторюються декілька разів. Саме використання циклів дає змогу повною мірою реалізувати швидкодію комп'ютерів (рис. 7.9). Одноразове виконання тіла циклу називається ітерацією.

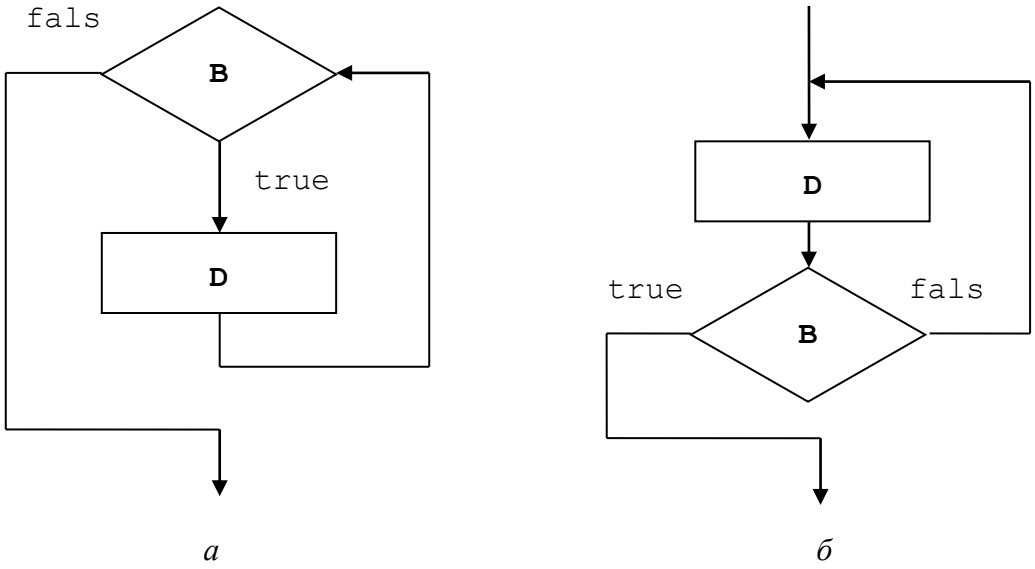


Рис. 7.9. Цикл з передумовою (*а*) та післяумовою (*б*)

Вираз, що визначає, чи буде вчергове виконуватися ітерація, чи цикл завершиться, називається умовою виходу, або умовою завершення

циклу (або умовою продовження залежно від того, як інтерпретується його істинність – як ознака необхідності завершення чи продовження циклу. Змінна, в якій зберігається номер поточної ітерації, називається лічильником ітерацій циклу, або просто лічильником циклу. Цикл не обов'язково містить лічильник, також лічильник не обов'язково має бути одним – умова виходу з циклу може залежати від декількох змінюваних в циклі змінних, а може визначатися зовнішніми умовами (наприклад, настанням певного часу, тоді лічильник взагалі не знадобиться). Цикли бувають з передумовою, з післяумовою та безумовні.

Перші системи навігації роботів створювалися на основі сканувальних датчиків, зокрема телекамер, локаційних лазерних та стереодальномірів. Стандартними ознаками перешкод, які сприймаються роботом, є стіна, навіс, виїмка, нахили, інші спрощені або укрупнені деталі сцени. Зазвичай завдання технічного зору робота під час навігації розділяють на три рівні, відповідно дальньої, середньої і ближньої навігації. Система дальньої навігації призначена для планування основного маршруту руху робота. Головною функцією машинного зору при цьому є розпізнавання орієнтирів. Оптико-електронна схема, що забезпечує вирішення задачі, складається з об'єктива зі змінною фокусною відстанню (трансфокатора), електронного блока, що керує камерою, механізму, що реалізує нахил і поворот камери, а також системи розпізнавання орієнтира. Вхідні сигнали визначаються картою видимості, візуальними моделями орієнтирів, картою місцевості та описом завдання. Подання зовнішнього середовища ґрунтуються на карті ділянок видимості (прохідності робота), місцезнаходження робота, послідовності розміщення ділянок на маршруті руху.

Для коректної навігації в просторі роботам потрібні алгоритми та програмне забезпечення, які дають змогу обробляти інформацію, отриману з датчиків. Ці алгоритми можуть містити різні методи навігації, такі як локалізація, картографування та планування маршруту.

Одним з використовуваних методів навігації є SLAM (Simultaneous Localization and Mapping), за допомогою якого роботи можуть одночасно визначати своє місцезнаходження та створювати карту навколишнього середовища. Інші методи полягають у використанні інерціальних датчиків, машинного зору та інших технологій.

Система проміжної навігації (Intermediate Navigation System) – це система, яку застосовують для навігації роботів в межах великих просторів, таких як склади, заводські цехи, лабораторії тощо. Ці системи забезпечують точну інформацію про місцезнаходження робота, що дає змогу планувати оптимальний маршрут та уникати перешкод на шляху руху. У системі проміжної навігації можуть бути використані різні технології, такі як GPS, візуальна навігація, сенсори для визначення руху та орієнтації робота, відстанційні датчики тощо.

Завдання системи проміжної (середньої) навігації – створення точної карти місцевості або карти перешкод, яка є підмножиною карти системи дальньої навігації робота. Система проміжної навігації означає послідовне коригування шляху робота зі збільшенням його ширини і поділ маршруту на дрібніші ділянки. Вхідні сигнали цієї системи основані на картах дальньої навігації, моделях відомих перешкод і явних орієнтирів місцевості, на маршруті, спланованому на основі системи дальньої навігації. Система проміжної навігації слугує загальному аналізу зображень для подальшої сегментації та розпізнавання, якісного визначення відстаней, накопичення орієнтування і планування маршруту. Вхідними даними слугує інформація, що надходить від модулів обчислення пройденого шляху, відомості про вільний простір. Система повинна вимірювати відстані, оцінювати структуру місцевості, визначати безпечний обхід перешкод і планувати проходження по певних трасах. Окремим завданням системи ближньої навігації є відслідковування маршрутів. В її склад входять система планування послідовної зміни траєкторії шляху, криволінійних ділянок, крутих спусків і підйомів, а також забезпечення навігації за наявності іншого робота.

Системи навігації можуть бути *пасивними* або *активними*. Пасивна система навігації приймає інформацію про власні координати та інші характеристики власного переміщення від зовнішніх джерел, активна система розрахована на визначення місця розташування лише власними складовими системи. Як правило, глобальні схеми навігації пасивні, локальні – завжди активні.

Перші моделі роботів з автономною навігацією пересувалися маршрутом, жорстко заданим за допомогою електричних кабелів. На роботах встановлювали пристрої приймання електромагнітних хвиль, що давало змогу визначати напрямок переміщення. Такі апарати могли

рухатися різними маршрутами завдяки тому, що по декількох кабелях передавався сигнал з різною частотою, проте така схема була дорогою і негнучкою. З появою перших систем машинного зору вдалося відмовитися від кабелів і перейти до навігації за картинками або флюоресцентними лініями. Робот за допомогою камери стежив за такою лінією і самостійно рухався. Проте лінії часто стиралися, загороджувалися іншими об'єктами, а в місцях, де сходилося декілька маршрутних ліній, роботи зазвичай зупинялися і не могли зрозуміти, куди рухатися далі. Випробовувано й інші схожі концепції. Наприклад, на маршруті руху на певній висоті розміщували предмети-маркери заданої форми, які робот за допомогою датчиків фіксував та визначав своє місцезнаходження. Схема навігації, основана на активному контакті машини з навколишнім світом має недолік, пов'язаний з безпекою роботи, крім того, не завжди можна правильно ідентифікувати маркери. Поступово моделі маркерної навігації були оснащені більш досконалими аналоговими датчиками, які навчилися вимірювати силу реакції контакту і визначати форму маркера і поступово перейшли до застосовуння цифрових матричних датчиків, здатних отримувати від маркерів докладні дані про навколишнє середовище. Наступний рівень навігації – це використання лазерних далекомірів і ультразвукових сонарів. При цьому недолік лазерного проміння полягає в тому, що ним можна отримати дані тільки про середовище прямої видимості, а ультразвукові датчики характеризуються великим часом відгуку (якщо робот перебуває у великому відкритому просторі), що заважає роботу переміщатися швидко. Швидкість звуку в різних умовах також може змінюватися, впливаючи на точність оцінки відстані, через що система управління робота спотворює загальну картину навколишнього середовища.

Більшість роботів, які орієнтуються на місцевості, використовують одометрію (odometry – вимір пройденого шляху) як основу навігаційної системи. Звичайний одометричний вимірювач має сенсори, якими вимірюють переміщення і швидкість:

- кодери з точковими контактами;
- потенціометри;
- оптичні кодери;
- магнітні кодери;
- індуктивні кодери;

▪ ємнісні кодери.

Одним зі способів організації руху робота в заздалегідь не визначеному середовищі може бути використання алгоритмів системи управління за оптронною лінійкою – датчиком стеження за лінією на поверхні полігона. Застосовують також метод організації руху робота на оснащеному системою маяків полігоні, оснований на побудові віртуальної смуги, яка формується в бортовому комп'ютері робота, так щоби вона огинала включені маяки, що забезпечує проходження заданої траси. Автономне визначення узагальнених координат дає змогу сформувати «віртуальну оптронну лінійку», сигнал якої пропорційний відхиленню робота від віртуальної смуги.

Одометрія дає хорошу короткочасну точність, недорога і має дуже велику частоту дискретизації. Інерційна навігація є альтернативним методом для одометрії, принцип роботи якої полягає у безперервному зчитуванні найменшого прискорення по кожній з трьох осей напрямків переміщення в часі, щоб обчислити положення.

У системі позиціонування за природними орієнтирами виділяють такі базові компоненти: а) сенсор (зазвичай зоровий), який виділяє орієнтири на сцені; б) метод порівняння отриманих під час спостереження особливостей відомих орієнтирів; в) метод обчислення розміщення і локалізації помилок порівнянь.

У сучасних робототехнічних системах найкраще зарекомендувала себе багаторівнева ієрархічна структура системи управління (рис. 7.10). Застосовують три рівні управління роботом: стратегічний, тактичний і виконавчий.

На *стратегічному рівні* вирішують загальні задачі планування траєкторій руху в недетермінованому просторі, а також пошук і розпізнавання об'єктів зовнішнього середовища та їхнє взаємне розміщення за допомогою далекомірів, систем технічного зору і силомоментної дії.

На *тактичному рівні* розглядають задачі з визначення достатніх зусиль на робочих органах, самонаведення та зупинки поблизу перешкод. Для цього застосовують тактильні датчики, оптичні, індуктивні та ультразвукові датчики, відеокамери.

Виконавчий рівень застосовують для монотонності руху елементів робота, визначення їхнього взаємного розташування та дотримання динамічних характеристик приводу.

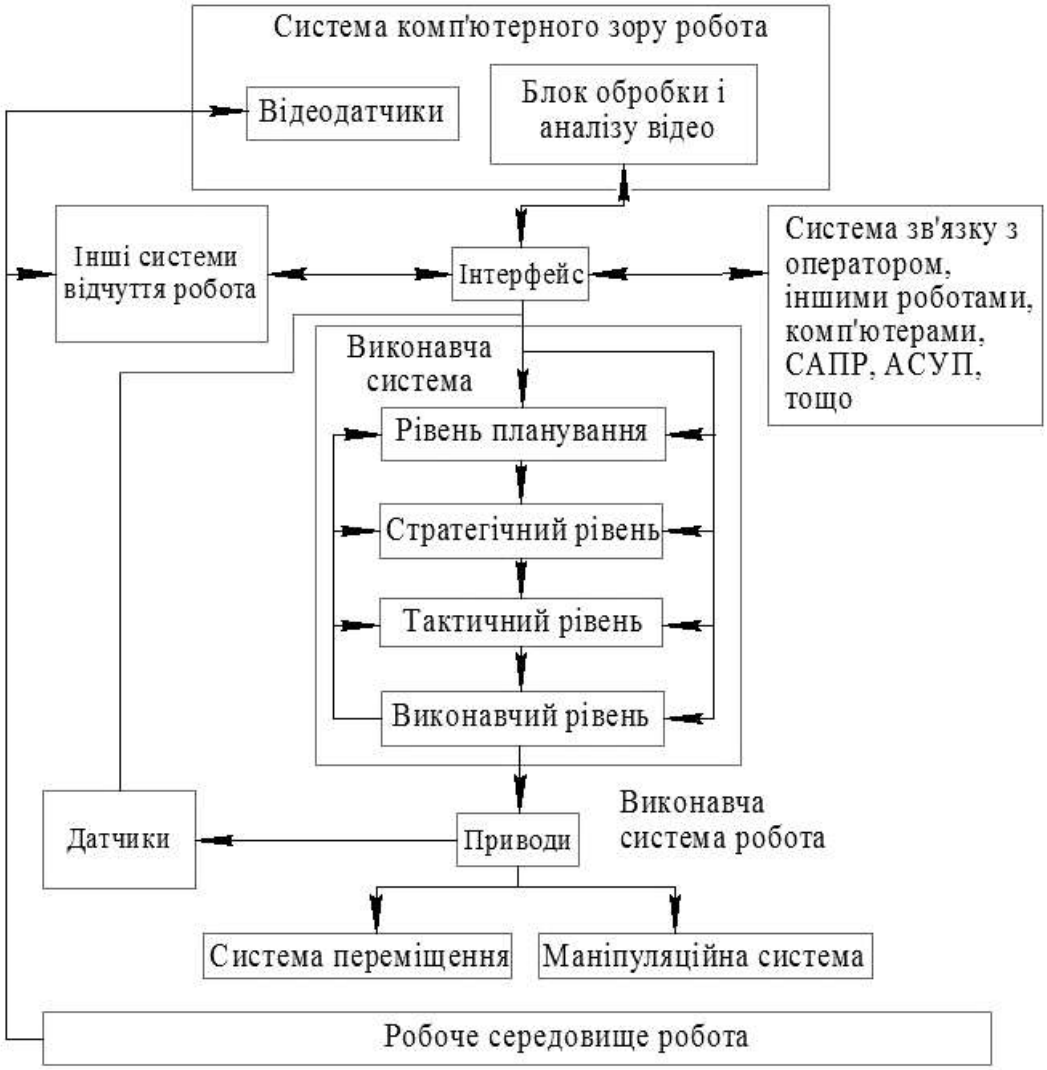


Рис. 7.10. Функціональна схема ієрархічної системи управління інтелектуального робота

Розглянемо деякі стандартні алгоритми керування роботами.

Значна кількість алгоритмів відома для мобільних роботів, які призначені для переміщення вантажів на складах, для роботів-пилососів, роботів для інтерактивного спілкування тощо. Аглоритми навігації мобільних роботів, залежно від сфер їхнього застосування, значно відрізняються. Це залежить від датчиків, які застосовують для управління та простору, в якому працює робот. Наприклад, для навігації на відкритій місцевості може застосовуватися GPS-навігація, яка не придатна для навігації всередині приміщення, тому що супутниковий сигнал, як правило, не досягає пристроїв крізь бетонні і металеві конструкції. При цьому для оминання перешкод можна застосовувти схожі алгоритми як зовні, так і всередині приміщень.

Для навігації в приміщеннях завдяки тому, що простір всередині будинків часто обмежений досить невеликими площами, можна скористатися такими засобами навігації, як тріангуляція, навігація за мітками (QR коди із зазначенням наступних команд для робота, сигнальні лінії вздовж руху, мітки на стінах для корекції розміщення), SLAM-навігація, а також комбінації названих методів.

Тріангуляція – це метод визначення точного положення об'єкта (точки) в просторі або на площині за допомогою вимірювань відносних відстаней або кутів до цього об'єкта від двох або більше відомих точок.

Модель керування на основі добору траєкторії – це підхід до планування траєкторії руху, в якому використано моделі й алгоритми для визначення оптимального шляху в просторі для досягнення певної мети чи виконання завдання. Основна ідея полягає в тому, щоб розробити систему, яка може автоматично визначати, як робот повинен рухатися в середовищі, щоб виконати конкретне завдання, зокрема уникнення перешкод, оптимізацію шляху, врахування динаміки робота. Алгоритм моделі керування на основі добору траєкторії наведено на рис. 7.11.

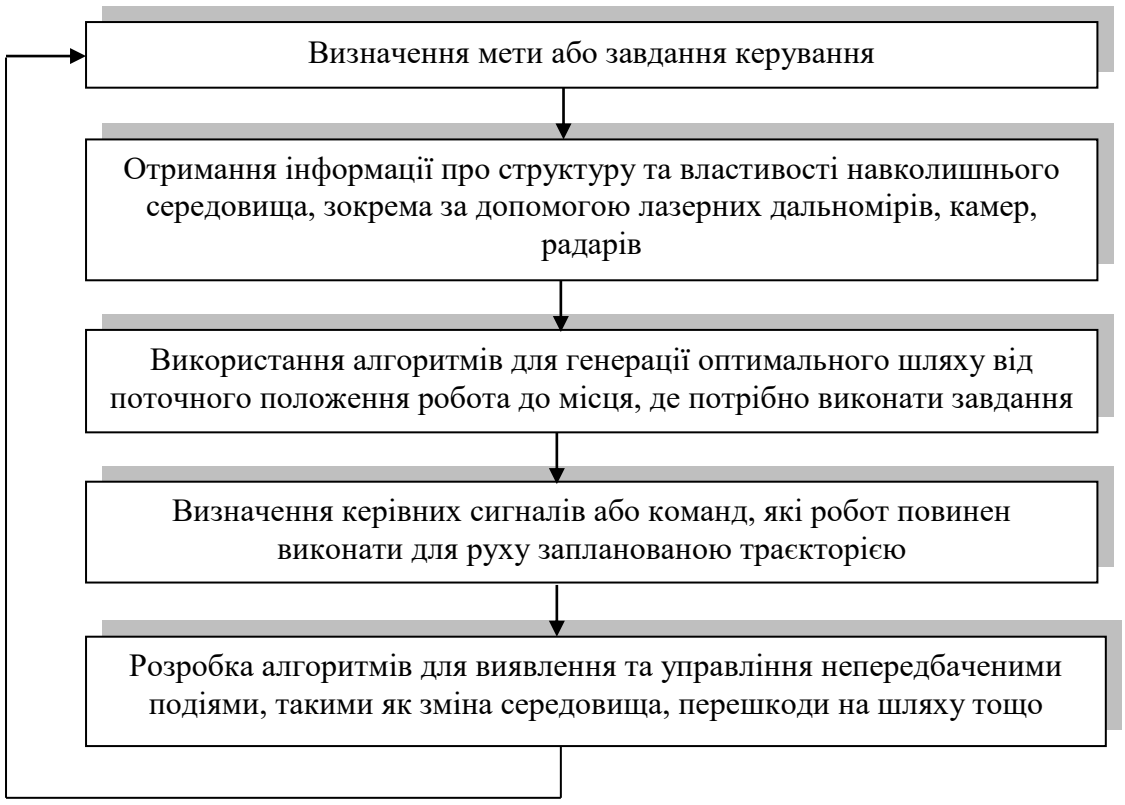


Рис. 7.11. Алгоритм керування на основі добору траєкторії

Циклічність алгоритму на рис. 7.11 пояснюється тим, що в разі зміни положення робота змінюється його внутрішній стан і представлення зовнішнього середовища, а це означає, що потрібно наново виконати планування маршруту відповідно до умови поточного стану.

Модель керування на основі добору траєкторії скаладається з двох підходів:

– DWA (Dynamic Window Approach) – алгоритм, який визначає можливі комбінації швидкостей та напрямків руху робота для створення найкращої траєкторії руху в реальному часі, зважаючи на обмеження та характеристики динаміки руху;

– застосування референсної траєкторії для генерації можливих треєкторій.

Алгоритм DWA виконує пошук простих траєкторій з постійною кривизною, на яких не відбувається зіткнень з перешкодами та які можуть бути реалізовані. Таким алгоритмом не передбачене застосувння зворотного звязку, а пошук команд керування роботом виконує безпосередньо в просторі швидкостей. Динаміка робота обмежується простором пошуку, на якому досягається гранична швидкість. В подальшому відбувається оптимізація швидкості шляхом пошуку максимуму деякої цільової функції за параметром швидкості. Простір пошуку швидкості зображено на рис. 7.12. На вертикальній осі відкладають лінійну швидкість, а на горизонтальній – кутову швидкість робота. Таким чином кожна точка такого простору описує рух робота по деякій кривій.

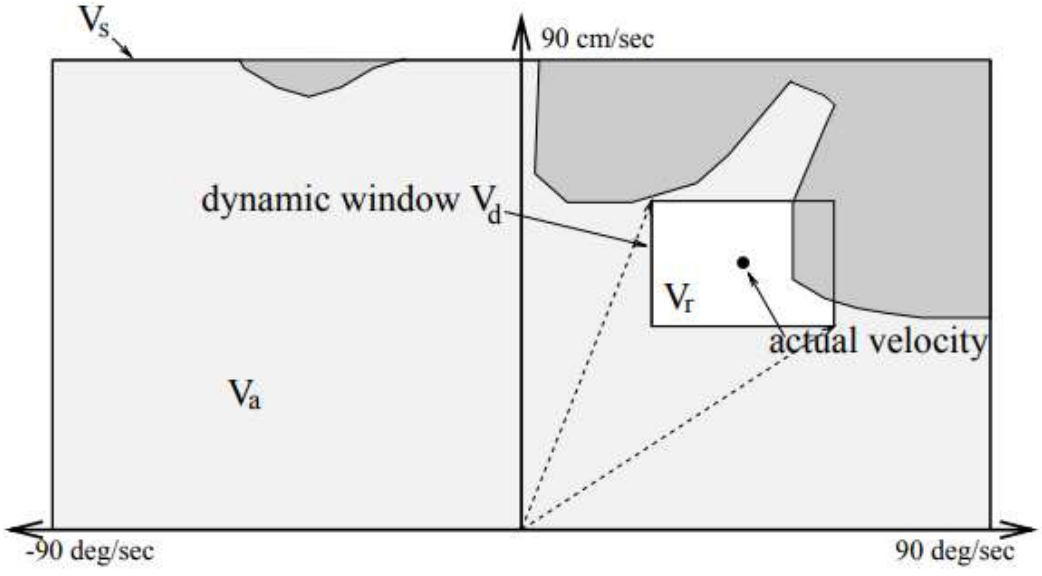


Рис. 7.12. Представлення простору зовнішнього середовища за алгоритмом Dynamic Window Approach

Алгоритм DWA складається з двох основних кроків:

1. Створення простору пошуку можливих швидкостей:
   1.1. Визначення локального простору для пошуку шидкості: розглядають лише кругові траєкторії, які однозначно визначені парами поступальних і обертальних швидкостей (υ, ω). Це призводить до формування двовимірного простору пошуку швидкості.
   1.2. Визначають допустимі швидкості: обмеження допустимих швидкостей гарантує, що розглядаються лише безпечні траєкторії. Пару (υ, ω) вважають допустимою, якщо робот здатен зупинитися до того, як досягне найближчої перешкоди на відповідній кривизні.
   1.3. Формується «динамічне вікно» – обмеження допустимих швидкостей до тих значень, яких можна досягти за короткий проміжок часу, враховуючи обмежені прискорення робота.
2. Оптимізація цільової функції:

$$G(\upsilon,\omega) = \sigma(\alpha \cdot \text{heading}(\upsilon,\omega) + \beta \cdot \text{dist}(\upsilon,\omega) + \gamma \cdot \text{vel}(\upsilon,\omega)) \to \max, \quad (7.1)$$

де $\text{heading}(\upsilon,\omega)$ – функція, яка оцінює напрям руху (місцезнаходження, максимум, якщо робот рухається прямо до вказаної мети); $\text{dist}(\upsilon,\omega)$ – функція, яка оцінює відстань до найближчої перешкоди на траєкторії (що менша відстань до перешкоди, то сильніше бажання робота обійти її); $\text{vel}(\upsilon,\omega)$ – функція швидкості руху вперед, яка підтримує швидке переміщення; σ – функція, яка згладжує зважену суму трьох компонентів і забезпечує бічний зазор від перешкод; α, β, γ – деякі коефіцієнти.

Таким чином, для оцінки отриманих треєкторій за алгоритмом DWA застосовують пошук похибки за напрямком робота в кінцевій точці, похибку, зумовлену відстанню до перешкоди та швидкістю.

Алгоритм референсної траєкторії:

– перемістити референсну траєкторію;
– обрати референсну позицію на новій траєкторії (можливе семплювання за референсною швидкістю);
– змоделювати положення робота, зважаючи на обмеження;
– додати отриману ділянку траєкторії до дерева рішень.

SLAM розшифровується як одночасна локалізація та картографування (Simultaneous Localization and Mapping). Метод SLAM-

навігації є найперспективнішим із згаданих для роботи в недослідженому просторі. Цей метод цікавий тим, що може використовуватися для визначення розміщення та орієнтації автономних роботів на заздалегідь невідомій їм місцевості для створення карти та її подальшого використання. Варто мати на увазі, що для роботи SLAM потрібні різні джерела даних, зокрема одометрія.

Загалом алгоритм роботи SLAM складається так. Роботу потрібно в кожний момент часу знати своє розміщення, а також поступово сканувати навколишній простір за допомогою сенсорів, складаючи, таким чином, мапу місцевості. Мапа будується поступово з досліджених роботом нових просторів. У міру вибудовування мапи робот починає звірятися з нею. Наприклад, якщо робот проїжджає по тій ділянці приміщення, яку він вже відсканував, відбувається звіряння за певними патернами. Якщо апарат розуміє, що поточні показання одометрії не збігаються з даними мапи, відбувається коригування одометрії.

З математичного погляду SLAM намагається оцінити мапу та весь шлях, пройдений роботом за дискретний проміжок часу із застосуванням теорії ймовірності. Позиція робота розраховується тільки в кінці траєкторії, виконаної роботом. Ймовірне визначення підходу повного SLAM може бути дане таким чином:

$$p(x_{\{1:t\}}, m \mid z_{\{1:t\}}, u_{\{1:t\}}), \tag{7.2}$$

де $u_{\{1:t\}} = \{u_1, u_2, u_3, ..., u_t\}$ – функція керування роботом в момент часу *t*; $z_{\{1:t\}} = \{z_1, z_2, z_3, ..., z_t\}$ – інформація про навколишнє середовище, що оглядається роботом в момент часу *t*; *m* – це побудована карта; $x_{\{1:t\}} = \{x_1, x_2, x_3, ..., x_t\}$ – отримані дані про розміщення робота в момент часу *t*.

Алгоритм SLAM можна застосовувати і в реальному часі (онлайн), і за наперед визначеним положенням робота (офлайн). Послідовність процесів алгоритму SLAM відображено на рис. 7.13.

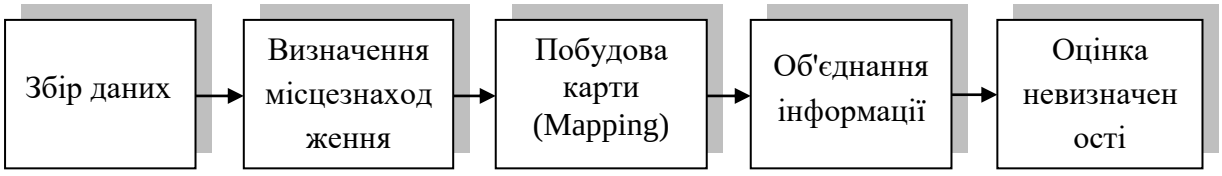


Рис. 7.13. Послідовність процесів алгоритму SLAM

Збір даних – це процес отримання інформації з різних джерел для подальшого використання або аналізу. До визначення місцезнаходження робота (Localization) існує кілька підходів, вибір конкретного методу може залежати від характеристик робочого середовища, наявних датчиків та вимог до точності. Одним з найбільш поширених підходів до *Localization* є використання фільтра Калмана. Далі наведено загальний алгоритм локалізації на основі фільтра Калмана:

- *Initialization (ініціалізація)*: визначення початкового наближеного положення робота та коваріаційної матриці помилок. Це може бути визначено на основі інформації від GPS, відомих точок на мапі або інших джерел;
- *Prediction Step (передбачення стану)*: визначення нового стану робота на підставі його попереднього стану та даних про керування (швидкість, кутова швидкість та ін.). Це може бути виконане за допомогою кінематичних рівнянь руху;
- *Update Step (оновлення стану)*: отримання нових спостережень (з лазерних дальномірів, камер, інших сенсорів) та визначення матриці і вектора спостережень на підставі фізичних характеристик об’єктів, які може виявити робот;
- *Innovation (обчислення інновації)*: визначення інновації, яка відображає різницю між спостереженнями та прогнозованими значеннями;
- *Covariance Estimation (оцінка коваріації інновації)*: оцінювання коваріації інновації, що відображає впевненість в правильності оцінки стану;
- *State and Covariance Updates (оновлення стану та коваріації)*: оновлення стану та коваріації з використанням інновації та матриці коваріації інновації;
- *Repeat (повторення)*: повторення кроків *Prediction Step* та *Update Step* для кожного нового проміжку часу або нового спостереження.

Коваріація – це міра ступеня того, як дві випадкові величини змінюються разом. Вона вказує на те, як зміна однієї змінної пов’язана зі зміною іншої. Коваріація може бути використана для визначення, чи дві змінні змінюються разом і як сильно. Формульно коваріацію між двома випадковими величинами X і Y обчислюють так:

$$\mathrm{cov}(X,Y)=\frac{\sum_{i=1}^{n}(X_i-\bar{X})(Y_i-\bar{Y})}{n-1}, \qquad (7.3)$$

де X_i і Y_i – значення відповідних випадкових величин на i-му спостереженні; $\bar{X}$, $\bar{Y}$ – середні значення випадкових величин X і Y; n – кількість спостережень.

Коваріація не нормована і може набувати значень від $-\infty$ до ∞. З цієї причини використовують стандартизовану міру – коефіцієнт кореляції Пірсона, який обмежений від -1 до 1 і нормалізує коваріацію відносно дисперсій відповідних змінних.

Локалазація може виконуватися також на основі фільтра частинок.

Застосування правила Баєса дає основу для послідовного оновлення апостеріорного розміщення за наявної мапи і функції переходу:

$$p(x_{\{1:t\}} \mid z_{\{1:t\}}, m) = \sum_{m_{t-1}} p(z_t \mid x_t, m_t) \sum_{x_{t-1}} p(x_t \mid x_{t-1}) p(x_{t-1} \mid m_t, z_{1:t-1}) / Z\,. \qquad (7.4)$$

Процес розрахунку траєкторії робота представлено на рис. 7.14.

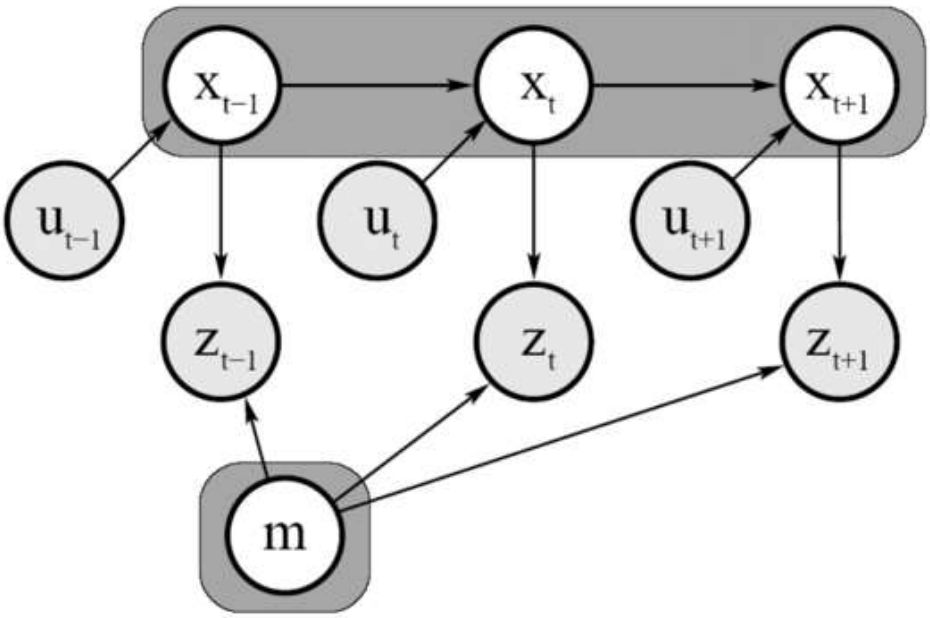


Рис. 7.14. Графічна модель підходу SLAM

У разі використання SLAM відображення мапи може бути як у 2D-, так і в 3D-форматі. Відображення геометрії місцевості в 3D є складнішим і зазвичай потребує більше пам’яті. Однак це не завжди дає якісне розуміння навколишнього середовища. Крім того, методи SLAM можна розділити за підходами SLAM, що використовуються для

представлення мап. Наприклад, SLAM основані на сітці точок (*grid-based*), на розпізнаванні унікальних об'єктів (*feature-based*) і на графіках (*graph-based*) для більш розріджених зображень, топологічні (*topological*) для розпізнавання типів і семантичні (*semantic*) для вищого рівня розуміння довкілля.

Feature-based SLAM використовують елементи, що легко ідентифікуються в середовищі і створюють внутрішнє уявлення про простір з огляду на розміщення цих орієнтирів. Історично найраніший і найвпливовіший алгоритм SLAM основано на розширеному фільтрі Калмана (*EKF – Extended Kalman Filter*).

SLAM на основі графів або мереж також намагається створити мапу за допомогою графа, вузли якого відповідні позиціям робота в різні моменти часу, а ребра є просторовими обмеженнями, що пов'язують позиції робота. Обмеження полягають у розподілі ймовірностей щодо перетворення між позиціями. Використовуючи граф, побудований відповідно до вимірювань датчиків, система визначає ймовірну конфігурацію позицій з урахуванням ребер графа. Одним із найбільш популярних підходів до SLAM на основі графів є метод відображення в реальному часі (***VSLAM Rtabmap***).

Grid-based SLAM – найпростіший метод на основі сітки. За такого підходу середовище поділяється на сітку точок певного розміру. Кожна точка може бути перешкодою, незайнятою точкою або не дослідженою. Наприклад, осередок зі значенням «1» буде зайнятим, а інший зі значенням «0» буде завсім порожнім. Значення точки може змінюватись від 0 до 1 за допомогою проміжних значень.

Програмування інформаційних систем роботів виконують за допомогою спеціальних пультів (рис. 7.15) або програмованих комп'ютерних мереж, в яких зберігаються пакети програмного забезпечення, наприклад VinPISa, WinCC, ZenOn, CoDeSys. Також можуть застосовувати спеціальне програмне забезпечення для візуального програмування, зокрема RobotStudio, Roboguide, Gazebo, Microsoft Robotic Studio.

Процес дискретного позиційного програмного управління має такий вигляд. У пристрої управління зберігається керівна програма, яка складається з окремих підпрограм управління окремими приводами. Ці підпрограми є послідовністю чисельних значень кроків позиціювання приводу певного ступеня рухомості робота.

Рис. 7.15. Пульт управління контролера промислового робота

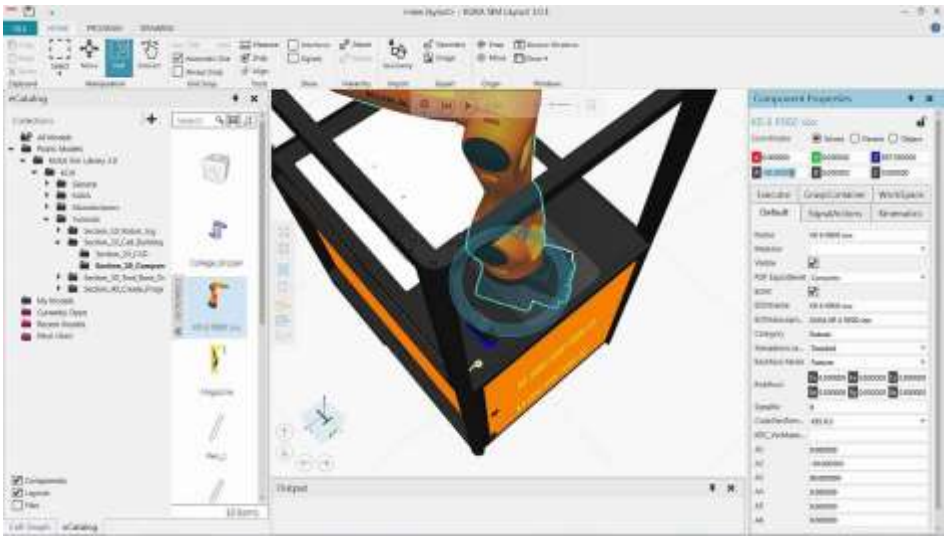
Відпрацювання керівної програми полягає в одночасній подачі на всі приводи значень чергового кроку та відпрацювання приводами цього завдання. Після того як всі приводи зупиняться, робочий орган маніпулятора займе відповідну чергову позицію у просторі та орієнтацію. Після цього керівна програма видає команду на виконання приводами наступного кроку і.т.д. Далі робочий орган робота переміщатиметься кроками по запланованій дискретній траєкторії, зупиняючись після кожного кроку. Програмування керівної програми виконують методом навчання на самому роботі або аналітично на комп'ютері. Розвиток цього методу програмування шляхом навчання став доступним завдяки використанню системи технічного зору. На час програмування на робочому органі робота кріплять телевізійну камеру, яка передає зображення об'єктів зовнішнього середовища, з якими взаємодіє робот та формується відповідна дія на систему приводу, що записується в програму.

Другий варіант програмування методом навчання полягає у копіюванні робочим органом робота рухів оператора (людини) та запису при цьому показань датчиків положення приводів.

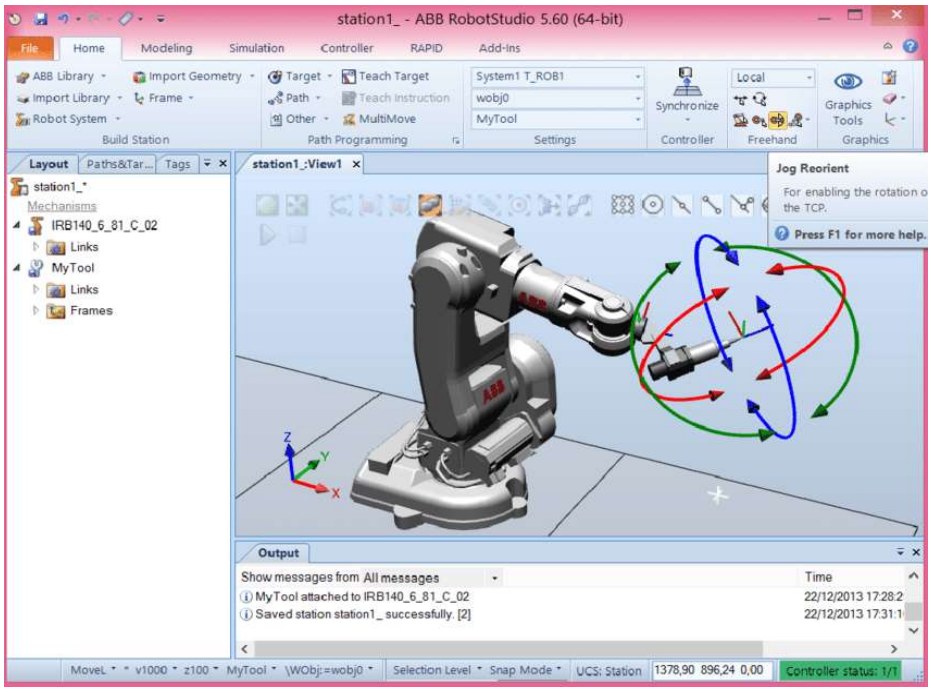
Напівавтоматичне навчання є найпоширеним та зручним видом навчання. Суть його у тому, що оператор, керуючи роботом від спеціального пульта навчання, послідовно виводить робочий орган у потрібне положення (точку), а потім, натискаючи спеціальну кнопку, дає сигнал на запис. У цей момент записуються всі координати маніпулятора, що однозначно визначають положення робочого органа в

просторі. Після цього маніпулятор переводять у нову позицію для подальшого навчання. У потрібній точці позиціонування оператор може багаторазово переміщувати маніпулятор для досягнення потрібної точності. Всі ці рухи не будуть записані в програму автоматичного циклу, поки оператор не переконається в потрібності такого запису.

Системи автоматизованого програмування (рис. 7.16) є комплексом обчислювальних програм, які знаходяться на носіях інформації і завантажуються в оперативну пам’ять комп’ютера у разі потреби у введенні вихідних програм мовою програмування, розшифрування змісту програм, виконання потрібних обчислень, кодування результатів розрахунку та їхній запис на відповідний носій.



а



б

Рис. 7.16. Системи автоматизації програмування роботів Kuka (*а*) та ABB (*б*)

Автоматизована підготовка програм полягає в застосуванні автоматизованих комплексів, наприклад, RobotStudio, KukaSim, Gazebo, ROS, Robotbenchmark та ін. До складу таких комплесів входять досконалі засоби обчислювальної техніки та відповідне математичне забезпечення. Записати програму управління в такому разі можна на компютері в спеціальному програмному симуляторі роботів, а потім перенести збережену копію на контролер реального робота. Такий підхід дає змогу автоматизувати кілька етапів:

- визначити кількість проходів робочого органа робота і всіх режимів його роботи;
- визначити еквідистанти;
- апроксимувати та інтерполювати потрібні траєкторії за заданими точками;
- автоматично визначити команди стандартних циклів обробки.

Складність і трудомісткість розрахунку керівних програм, а отже, і створення відповідних систем автоматизованого програмування різко зростають у міру ускладнення траєкторій, якими повинен переміщуватися виконавчий пристрій робота, та збільшення кількості координат, керованих програмно.

Контрольні запитання

1. Що таке позиційне керування?
2. Що таке адапртивне керуввання?
3. Що таке умовний перехід?
4. Які алгоритми керування вам відомі?
5. Як оцінити коваріацію?
6. До якої системи робота, на вашу думку, належить підсистема програмного керування?

Лекція № 8. Ознайомлення з інтерфейсами керування робототехнічних систем

У робототехніці та електроніці термін «інтерфейс» вказує на спосіб, за допомогою якого різні компоненти чи системи обмінюються інформацією, командами чи сигналами. Це може бути фізичне з'єднання, електричні та протокольні аспекти.

Фізичний інтерфейс – пристрій, що перетворює сигнали і передає їх від одного компонента устаткування до іншого. Фізичний інтерфейс визначається набором електричних зв'язків і характеристиками сигналів. Наприклад, це може бути здійснене за допомогою різних видів рознімачів, кабелів, бездротових технологій тощо (рис. 8.1).

а *б* *в*

Рис. 8.1. Різновиди рознімачів: *а* – автомобільний OBD2; *б* – mini XLR M12 5; *в* – RJ-11 EV3

Електричний аспект інтерфейсу визначає електричні параметри і зв'язки між різними компонентами, він містить рівень напруги, струму, типи сигналів (аналогові чи цифрові), інші електричні характеристики. Прикладами електричник інтерфейсів можуть бути USB, HDMI, Ethernet, SPI, I2C, PCI Express.

USB (Universal Serial Bus) є одним з найпоширеніших електричних інтерфейсів для під'єднання різноманітних пристроїв, таких як комп'ютери, принтери, камери, мобільні телефони тощо і підтримує передавання даних та живлення пристроїв. Специфікація USB охоплює різні швидкості передавання даних, зокрема USB 1.x (Low-Speed з максимальною швидкістю 1.5 Мбіт/с та Full-Speed з максимальною швидкістю 12 Мбіт/с), USB 2.0 (High-Speed з максимальною швидкістю 480 Мбіт/с) та USB 3.x (SuperSpeed з максимальною швидкістю 5 Гбіт/с, 10 Гбіт/с або 20 Гбіт/с). USB використовує різні типи рознімачів та

конекторів для фізичного з'єднання пристроїв. Найпоширеніші з них – тип A, тип B, Micro-USB та USB-C (рис. 8.2).

Рис. 8.2. Різновиди типів USB

USB використовує різні протоколи для обміну даними, такі як USB Mass Storage для флеш-накопичувачів, USB Human Interface Device (HID) для інтерфейсів комп'ютера, USB Audio для аудіо-пристроїв. Порядок розміщення контактів USB-кабелю стандартизовано і визначено специфікацією USB (рис. 8.3).

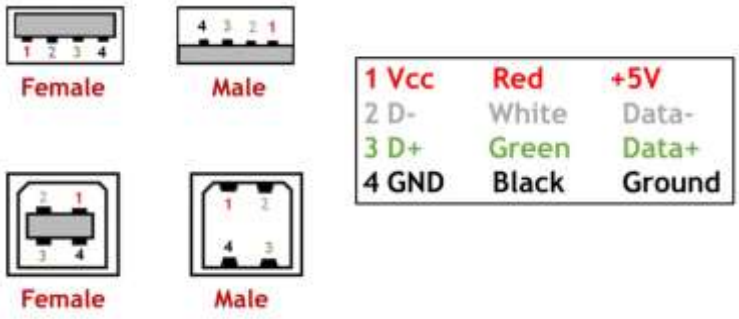
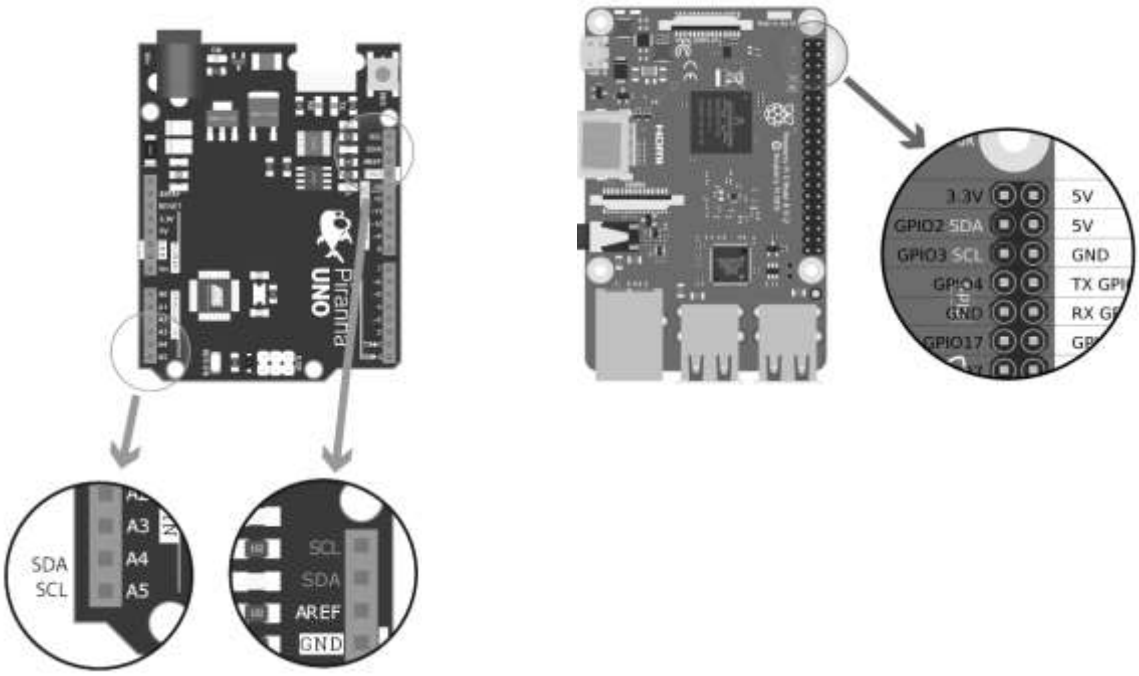


Рис. 8.3. Специфікація розміщення контактів USB 2.0

HDMI (High-Definition Multimedia Interface) є стандартом для передавання відео- та аудіосигналів між пристроями, а також може передавати додаткові дані, такі як керування пристроями.

Електричний інтерфейс Ethernet використовується для надійного мережного з'єднання між пристроями у локальних мережах (LAN), який використовує кабелі з рознімачами RJ45 для передавання даних.

I2C (Inter-Integrated Circuit) – це інтерфейс й одночасно протокол з'єднання мікроконтролерів, сенсорів та інших пристроїв, який використовує два провідники – лінію даних та лінію годинника – для здійснення комунікації. На кожному пристрої, який підтримує I2C, зазначають два контакти: SDA і SCL (рис. 8.4). *SDA* (serial data) – означає послідовні дані, *SCL* (serial clock) – послідовне тактування. Контакти I2C є виходами відкритого колектора або відкритого стоку, а це означає, що кожну з ліній передачі головного і веденого пристроїв потрібно буде під'єднати через резистори до стабілізованого електричного потенціалу живлення (зазвичай 5В). Опір резисторів розраховують залежно від паразитної ємності лінії: для швидкостей приблизно 400 Кбіт/с встановлюють резистори від 2 кОм, а для швидкостей 100 Кбіт/с – до 10 кОм.

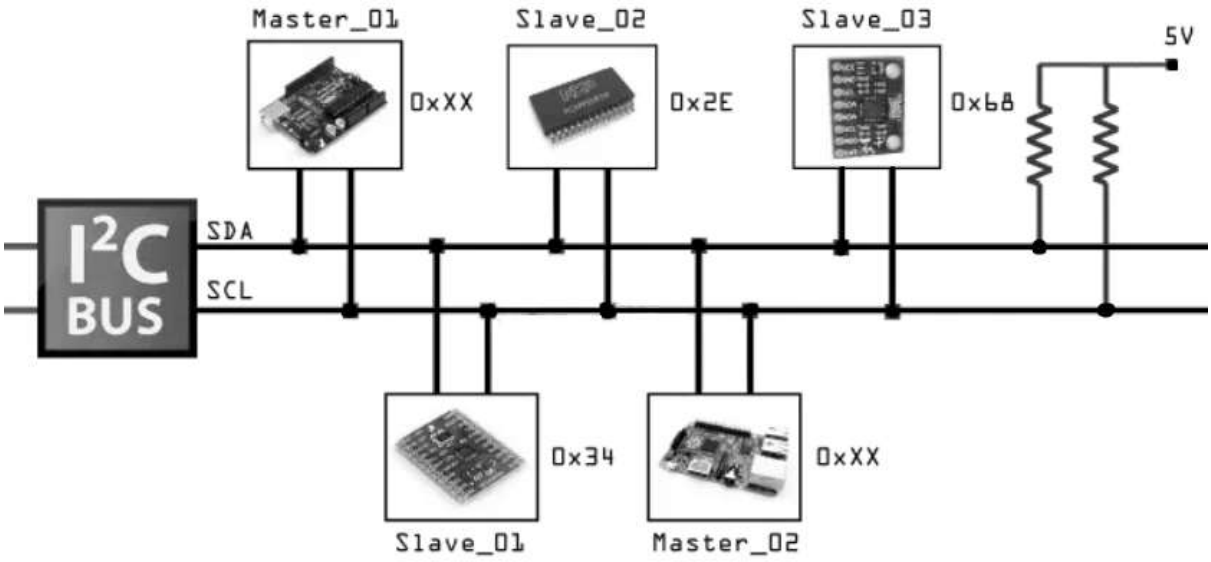


а *б*

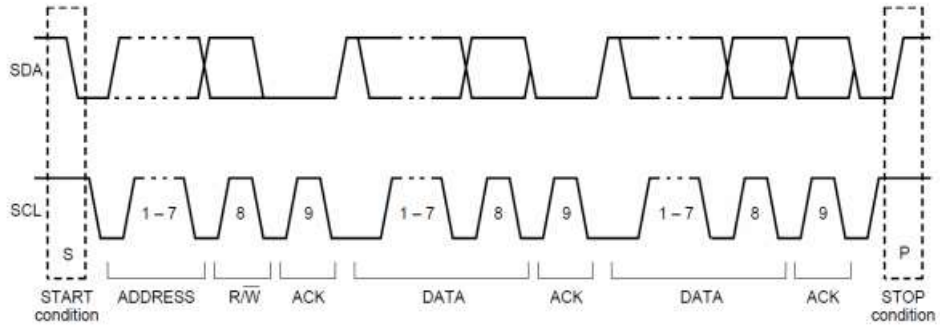
Рис. 8.4. Приклад розміщення I2C інтерфейсу на Arduino UNO (*а*) та Raspberry (*б*)

Інтерфейс I2C дає змогу під'єднувати до 127 пристроїв до одного головного (або master) пристрою. Кожен пристрій в мережі має свою унікальну 7-бітну (іноді 10-бітну) адресу, яку визначає виробник пристрою та ідентифікує його в мережі (рис. 8.5, *а*). Дані шиною I2C

передаються почергово, біт за бітом. Черговість передачі бітів контролюється сигналом годинника (рис. 8.5, *б*). Після передачі кожного байту може відбуватися підтвердження прийому (ACK), щоб master знав, що дані отримані.



а



б

Рис. 8.5. Комунікація пристроїв з інтерфейсом I2C (*а*); часова діаграма зв’язку I2C (*б*)

Транзакції даних за протоколом I2C складаються з трьох частин, які інтерпретуються таким чином: адреса пристрою → реєстр пристрою → дані (рис. 8.6).

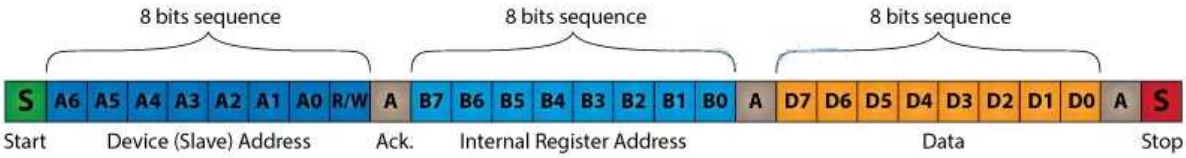


Рис. 8.6. Вигляд протоколу транзакції даних I2C: Device (Slave) Address – адреса пристрою, який приймає дані; Ack – підтвердження від пристрою, що приймає; Internal Register Address – адреса внутрішнього реєстру; Data – дані

У протоколі I2C передавання даних завжди ініціює головний пристрій, а пристрій, який зчитує дані (ведений), повинен завжди фіксувати байт (Ask, див. рис. 8.6). Наприклад, коли головний надсилає 8-бітну адресу після умови START, адресований ведений повинен підтвердити 9-й біт (встановити рівень «0»). Аналогічно, коли головний надсилає перший байт після адреси, ведений має підтвердити цей байт, якщо він бажає продовжити передавання. Після першого біта підтвердження в більшості випадків приходить інша послідовність адресації, але цього разу для внутрішніх регістрів підлеглого пристрою. Відразу після послідовностей адресації йдуть послідовності даних доти, доки дані не будуть повністю надіслані, завершується процес спеціальною умовою зупинки.

SPI (Serial Peripheral Interface) є протоколом для з’єднання мікроконтролерів та інших пристроїв, який використовує чотири провідники для передавання даних, тактового сигналу, сигналу вибору пристрою та сигналу передавання даних. Отже, SPI є синхронним інтерфейсом, в якому кожна передача синхронізована з тактовим сигналом, що генерується головниим пристроєм (мікроконтролером).

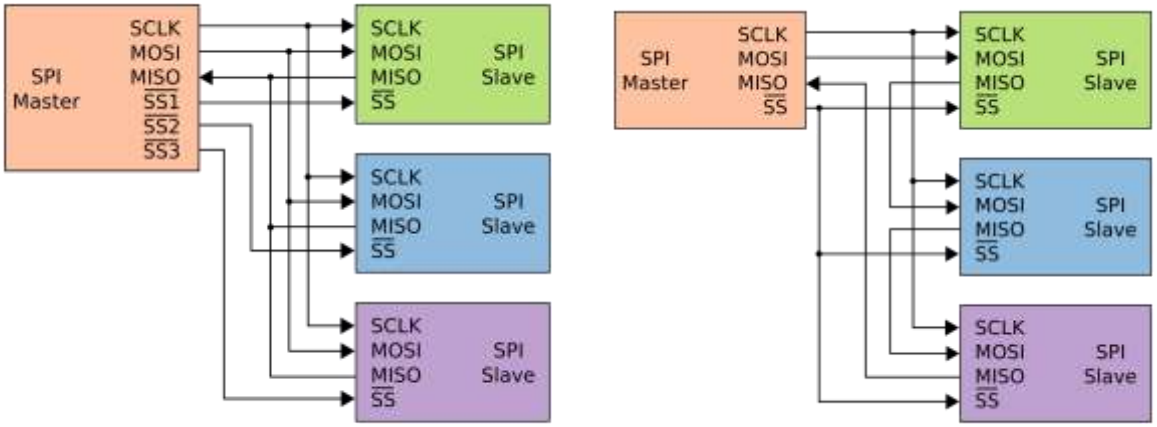
Для передавання даних інтерфейс SPI використовує чотири сигнальні лінії (провідники):

- MOSI або SI (Master Out Slave In) – вихід головного, вхід веденого призначено для передавання даних від головного до веденого пристрою;
- MISO або SO (Master In Slave Out) – вхід головного і вихід веденого призначено для передавання даних від веденого пристрою до головного;
- SCLK або SCK (Serial Clock) – послідовний тактовий сигнал, який призначено для передавання тактового сигналу для ведених пристроїв;
- CS або SS (Chip Select, Slave Select) – сигнал початку/завершення сеансу зв’язку, який призначено для вибору веденого пристрою. По

завершенню обміну даних має бути знятий, що дасть приймачу даних змогу вийти з режиму читання/запису та перейти до режиму обробки даних.

Рекомендується на кожному з провідників SPI встановити слабкий підтягувальний резистор, інакше деякі пристрої, наприклад, SD-карта, можуть не працювати (наприклад, резистор 50 кОм, однак менший номінал резистора дає змогу досягти вищої швидкості SPI).

Завдяки стандарту SPI можна під'єднувати до одного головного декілька ведених пристроїв за структурою з паралельним (Regular SPI Mode) та послідовним (Daisy Chain) зв'язком (рис. 8.7). В SPI-протоколі дані передаються паралельно по лініях MOSI та MISO, пакетами, біт за бітом. Довжина пакета, як правило, становить 1 байт (8 біт), при цьому відомі реалізації SPI з іншою довжиною пакета, наприклад, 4 біти.



а *б*

Рис. 8.7. SPI-шина: *а* – один головний, три незалежні (паралельні) ведені; *б* – один головний, три залежні (послідовні) ведені

Сигнал SCK визначає час передавання кожного біта. Головний пристрій ініціює цикл зв'язку.

Кроки передавання даних у SPI:

- головний пристрій ініціює комунікацію встановленням низького рівня на контакті CS вибору пристрою, тобто того пристрою, з яким треба встановити з'єднання;
- за низького рівня сигналу на контакті CS у веденого пристрою його вихід MISO перетворюється на режим «вихід»;
- головний пристрій визначає частоту годинника (SCK), від якої залежить швидкість передавання даних;

- тактовий сигнал SCLK від головного пристрою сприймається веденим і викликає зчитування на вході MOSI значень, переданих від головного бітів та зсуву регістру веденого пристрою (рис. 8.8);
- ведений передає дані через лінію MOSI. Кожен біт даних надсилається на високому рівні годинника (SCK);
- ведений приймає дані через лінію MISO;
- підтвердження прийому даних: після передавання кожного байту ведений пристрій може очікувати на підтвердження від веденого пристрою, яке називається підтвердженням прийому (ACK). Це підтвердження вказує, що дані були успішно отримані;
- після завершення передавання всіх даних головний пристрій деактивує лінію CS, що вказує на завершення комунікації.

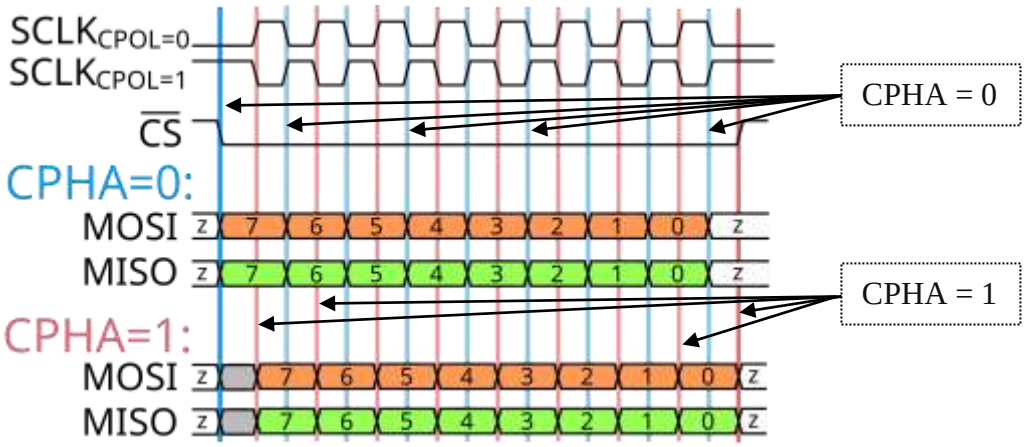


Рис. 8.8. Тимчасові діаграми роботи інтерфейсу SPI

Для позначення режимів роботи інтерфейсу SPI прийнято такі домовленості:

- **режим 0** (CPOL = 0 – вихідний стан сигналу синхронізації – низький рівень, CPHA = 0 – вибірка даних виконується по передньому фронту (перемикання) сигналу синхронізації);
- **режим 1** (CPOL = 0, CPHA = 1 – вибірка даних відбувається по задньому фронту (перемикання) сигналу синхронізації);
- **режим 2** (CPOL = 1 – вихідний стан сигналу синхронізації – високий рівень, CPHA = 0);
- **режим 3** (CPOL = 1, CPHA = 1).

UART (Universal Asynchronous Receiver/Transmitter) – це універсальний асинхронний тип приймача-передавача та протокол для

асинхронного послідовного зв’язку, у якому можна налаштувати формат даних і швидкість передавання. Комунікація в найпростому випадку виконується двома сигнальними каналами (RX, TX), а також треба мати спільний низький рівень (рис. 8.9).

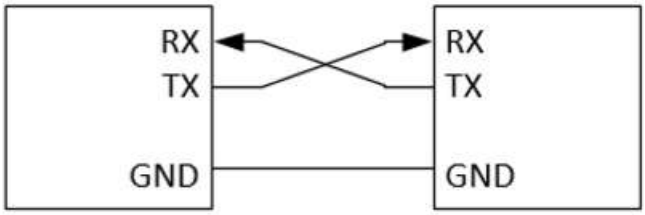


Рис. 8.9. Схема комунікації з UART-інтерфейсом

UART передає пакети даних асинхронно без зовнішнього тактового сигналу, а замість цього використовуються біти початку і кінця для кожного байта даних. Передавання даних відбувається по одному біту за однакові проміжки часу, які визначаються заданою швидкістю UART і для конкретного з’єднання вказується в *бодах* (біт за секунду). Існує загальновизнана низка стандартних швидкостей: 300; 600; 1200; 2400; 4800; 9600; 19200; 38400; 57600; 115200; 230400; 460800; 921600 бод. UART може передавати дані, як від головного до веденого (TX - transmit), так і від веденого до головного (RX - receive) одночасно. Кожний пакет в UART містить стартовий біт (логічне значення «0»), байт даних (зазвичай 8 бітів), необов’язковий біт перевірки парності (parity) та стоповий біт (один або два, логічне «1»). Під час приймання ці зайві біти видаляються з потоку. Оскільки синхронізаційні біти займають частину бітового потоку, то результаційна пропускна здатність UART менша за швидкість з’єднання. Наприклад, для 8-бітних посилок формату 8-N-1 синхронізаційні біти становлять 20% потоку, що за фізичної швидкості лінії 115200 бод означає корисну швидкість передавання даних 92160 біт/с, або 11520 байт/с.

Якщо використовується біт парності, він буде розміщений після всіх бітів даних. Біт парності – це спосіб для приймального UART визначити, чи змінилися якісь дані під час передавання.

Лінія TX (*transmit*) використовується для передавання даних, а лінія RX (*receive*) – для приймання. Для формування часових інтервалів передавальний та приймальний UART мають джерело точного часу (тактування). Точність цього джерела повинна бути такою, щоб сума

похибок (приймача та передавача) встановлення часового інтервалу від початку стартового імпульсу до середини стопового імпульсу не перевищувала половини (а краще – чверті) бітового інтервалу.

Стандартною функцією UART є збереження останнього символу під час отримання наступного – це «подвійна буферизація», яка дає одержувачу весь час передавання символів для отримання символу. Багато UART мають невелику буферну пам'ять типу «перший увійшов – перший вийшов» між регістром зсуву приймача та інтерфейсом головної системи. Це дає хост-процесору ще більше часу для обробки переривання від UART і запобігає втраті отриманих даних на високих швидкостях.

UART звичайно використовується спільно з іншими комунікаційними стандартами, такими як EIA RS-232.

PLC (Power Line Communication) – телекомунікаційна технологія категорії «остання миля», що базується на використанні внутрішньо-будинкових і внутрішньоквартирних електромереж для високошвид-кісного інформаційного обміну. У цій технології, основаній на частотному поділі сигналу, високошвидкісний потік даних розділяється на декілька низькошвидкісних, кожен з яких передається на окремій частоті з подальшим їхнім об'єднанням в один сигнал.

Програмний інтерфейс (API – Application Programming Interface) – система уніфікованих зв'язків, призначених для обміну інформацією між компонентами обчислювальної системи. Програмний інтерфейс задає набір процедур, їхніх параметрів і способів звернення. API приховує внутрішню реалізацію функціональності компонентів, надаючи зовнішній інтерфейс для їхнього використання. Це дає змогу досягти високого рівня абстракції та спрощує використання API для розробників.

Локальні API – це API, які доступні для використання на локальному пристрої або в межах локальної мережі. Наприклад, Win32 API – це набір функцій, які надають доступ до різних операційних системних ресурсів в середовищі Windows; POSIX (Portable Operating System Interface) – це набір стандартів API, які визначають функції та структури даних для програм, що працюють в UNIX-подібних операційних системах і дають змогу взаємодіяти з файловою системою, процесами, потоками, мережними з'єднаннями; Android API – це набір API, які надають доступ до функцій операційної системи Android для розроблення мобільних застосунків.

Мережні API – це набір визначень та протоколів, які дають змогу взаємодіяти з веб-серверами та веб-сервісами. Наприклад, HTTP-протокол – це веб-API, який використовується для передавання текстових даних між клієнтом та сервером (зазвичай використовуються такі методи, як GET, POST, PUT, DELETE). Веб-API можуть повертати дані у різних форматах, таких як JSON (JavaScript Object Notation), XML (eXtensible Markup Language) або інші JSON.

Серед веб-API поширеними архітектурними стилями створення програмних інтерфейсів є REST, SOAP, WSDL, WebSocket, XML-RPC. Всі ці технології досить різні і архітектурно, і за форматом обміну даними.

Для створення програмних інтерфейсів для роботизованих систем часто застосовують різні схеми проєктування. Розглянемо основні.

Абстракція – це процес виділення загальних характеристик програмованих об’єктів, їхніх властивостей і методів за ігноруванням деталей реалізації. Абстракція надає користувачеві лише потрібну інформацію для роботи з об’єктами і складними системами (рис. 8.10). Це важливий принцип об’єктноорієнтованого програмування та загальний концепт, який допомагає створювати більш зрозумілий, модульний та масштабований код.

Рис. 8.10. Приклад реалізації абстракції

Інкапсуляція – це концепція об’єктноорієнтованого програмування, що полягає в обмеженні доступу до даних та методів об’єкта та скритті їх реалізації в межах самого об’єкта (рис. 8.11). Це дає змогу об’єднати дані та методи, які працюють з цими даними, у єдиний об’єкт, який контролює доступ до цих даних.

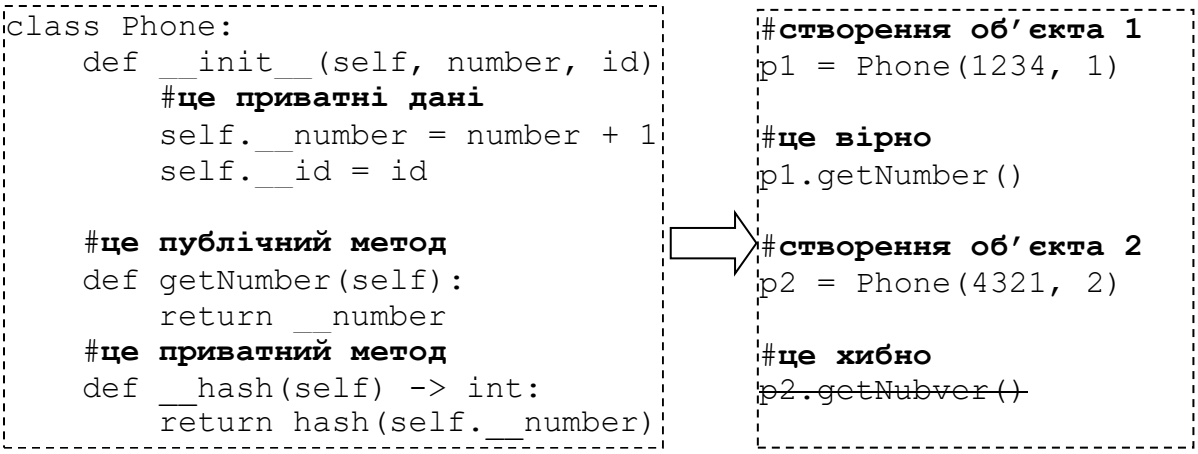


Рис. 8.11. Приклад реалізації інкапсуляції в Python

Поліморфізм – це принцип об'єктноорієнтованого програмування, який дає змогу об'єктам класів з однаковим інтерфейсом поводитися по-різному залежно від їхньої конкретної реалізації. Основна ідея полягає в тому, що об'єкти одного й того самого інтерфейсу можуть виконувати одну й ту саму дію по-різному. Поліморфізм в обєктноорієнтованих мовах програмування реалізують за принципом перевизначення методів (функцій) або через наслідування.

Поліморфізм за допомогою успадкування (успадкований поліморфізм) – це поліморфізм, який виникає внаслідок використання успадкування класів. Коли підклас успадковує методи свого батьківського класу, він може перевизначити ці методи, тобто надати їм власну реалізацію. При цьому об'єкти підкласу можуть використовуватися там, де очікується об'єкт батьківського класу (рис. 8.12).

Поліморфізм за допомогою перевантаження функцій (параметризований поліморфізм) – це поліморфізм, який виникає внаслідок використання функцій з однаковим ім'ям, але з різними параметрами. Залежно від типів та кількості переданих параметрів, можна запускати різні версії функції.

```
class Shape:
    def area(self):
        pass

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height
    def area(self):
        return self.width * self.height

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return 3.14 * (self.radius ** 2)

# Функція, яка приймає об'єкт будь-якого класу, що
наслідується від класу Shape
def print_area(shape):
    print("Area:", shape.area())

# Створення об'єктів різних класів
rectangle = Rectangle(5, 4)
circle = Circle(3)

# Запуск функції з різними типами об'єктів
print_area(rectangle)  # Виведе площу прямокутника
print_area(circle)     # Виведе площу кола
```

Рис. 8.12. Приклад реалізації поліморфізму в Python

Схема проєктування «*Спостерігач*» (*Observer*) – це поведінковий шаблон проєктування, який використовують для створення залежностей між об’єктами таким чином, щоб зміни в одному об’єкті програми управління автоматично призводили до надсилання інформації всім іншим об’єктам, які підписані на ці зміни (рис. 8.13).

Шаблон проєктування «*Підписник-Видавець*» (*Publisher-Subscriber*) є схожим до шаблону «Спотерігач», який використовується також для створення зв’язку між об’єктами і зумовлює складну систему зв’язків між об’єктами, тобто багато до багатьох, тоді як «Спостерігач» має зв’язок один до багатьох. На відміну від «Спотерігача» зв’язок між об’єктами здійснюється через канал зв’язку Event Channel (шини подій) (рис. 8.14).

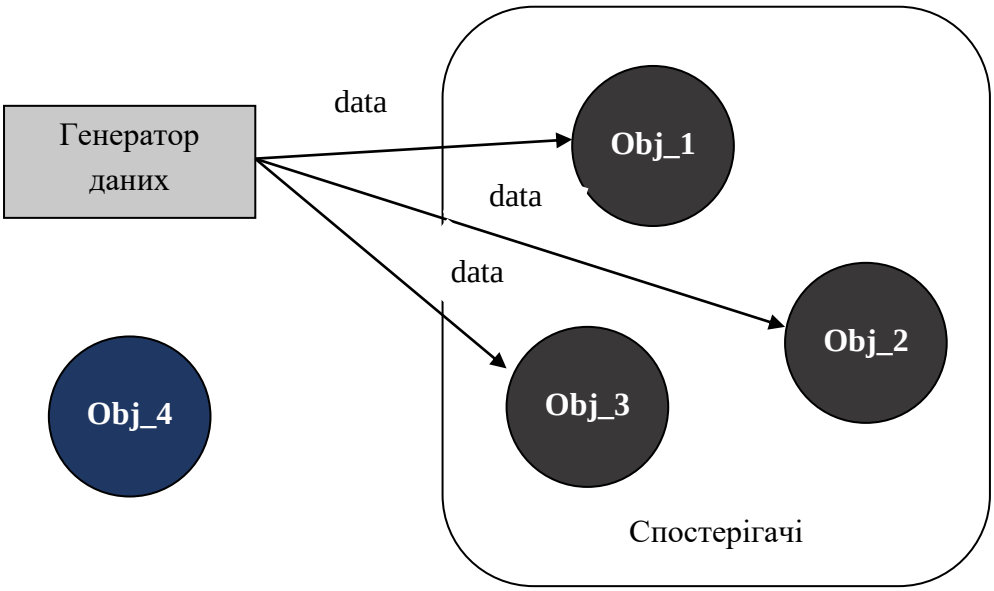


Рис. 8.13. Схема принципу Observer: об’єкти Obj_1, Obj_2, Obj_3 підписані на розсилання даних від «генератора даних», а Obj_4 – ні, тому дані будуть надсилатися тільки підписаним об’єктам

Рис. 8.14. Схема принципу Publisher-Subscriber

Видавець (Publisher) – це об’єкт, який має можливість створювати події або зміни і дає можливість підписатися на ці події іншим об’єктам. Підписник (Subscriber) – це об’єкт, який підписується на події або зміни, що відбуваються у видавця, і коли відбуваються зміни, автоматично отримує дані про зміни.

Таким чином, можна виділити основні відмінні особливості між Pub-sub та Observer:

- відсутність прямого зв’язку між об’єктами;
- об’єкти сигналізують один одному подіями, а не станами об’єкта;
- можливість підписуватись на різні події на одному об’єкті з різними обробниками.

Реактивна парадигма проєктування – це спосіб побудови програмного забезпечення, який ставить акцент на реакцію на події та зміни стану системи. Основними принципами реактивної парадигми є асинхронність, гнучкість, розподіленість та здатність до масштабування.

Асинхронність – це концепція в програмуванні, яка описує режим виконання, коли дії відбуваються паралельно без чіткого очікування завершення попередніх дій. В асинхронному програмуванні програмні інструкції виконуються безпосередньо, коли вони готові, а не послідовно.

У традиційних *синхронних програмах*, які виконуються послідовно, коли одна процедура завершується, інша починається, а блокування потоку виконується доти, доки не завершиться попередній стан виконання.

В асинхронному програмуванні, якщо одна процедура зупиняється або загальмовує (наприклад, запуск зовнішнього сервісу або зчитування даних з диска), основна програма може не зупиняти власного виконання і виконувати інші дії, які не залежать від цієї процедури. Після відновлення процедури виконується відповідний обробник або колбек.

В промислових робототехнічних системах, залежно від способів обміну даними, виділяють чотири варіанти комунікації комп'ютера з контуром системи управління роботів.

Варіант 1. Комп'ютер застосовується як програмний пристрій, який генерує сигнали керування на приводи та виконавчі механізми без зворотного зв'язку від системи управління. Перевагою такого підходу є простота математичного забезпечення процесів.

Варіант 2. Інформація про стан робототехнічної системи безперервно надсилається в систему розрахунку спеціальним каналом, що дає змогу виконувати постійний контроль руху виконавчої системи робота.

Варіант 3. Комп'ютер є складовою частиною контуру системи стеження виконавчої системи, а зворотний зв'язок також надсилає інформацію до комп'ютера. В такому разі комп'ютер виконує невеликий об'єм розрахункової роботи й одночасно контролює точність виконання роботи приводом.

Варіант 4. Застосування системи керування комп'ютером та мікроконтролером, що паралельно функціонують. У такому разі робота комп'ютера розвантажена від задач виконавчого рівня за допомогою додаткового мікроконтролера або іншого комп'ютера.

Застосовують стандарти не лише для протоколу зв'язку, а й для визначень команд, які надсилаються через цей протокол. Наприклад, вузлова архітектура для безпілотних систем (JAUS) визначає набір

повторно використовуваних команд та інтерфейсів, які можна застосовувати для управління автономними системами.

JAUS архітектура (Joint Architecture for Unmanned Systems) (рис. 8.15) – це відкрита програмна архітектура, яка містить такі компоненти:

- *транспортний рівень* – відповідний за передавання даних між компонентами JAUS, використовує протокол UDP/IP для зв'язку;
- *мережевий рівень* – відповідний за маршрутизацію даних між компонентами JAUS, використовує протокол OSPF для маршрутизації;
- *службовий рівень* – відповідний за надання загальних служб для компонентів JAUS, таких як синхронізація часу та управління доступом;
- *доменний рівень* – відповідний за конкретні функції БПС, такі як управління рухом, маніпулювання та сенсорика.

OSPF (Open Shortest Path First) – це протокол динамічної маршрутизації, який використовується для обчислення найкращих маршрутів до мережних адрес в IP-мережах. Він належить до класу протоколів IGP (Interior Gateway Protocol) і використовується для маршрутизації трафіку всередині автономної системи (AS).

Розрізняють два типи топологій JAUS:

– централізовану, у якій всі компоненти JAUS з'єднуються з центральним сервером;

– децентралізовану, де компоненти JAUS з'єднуються один з одним без центрального сервера.

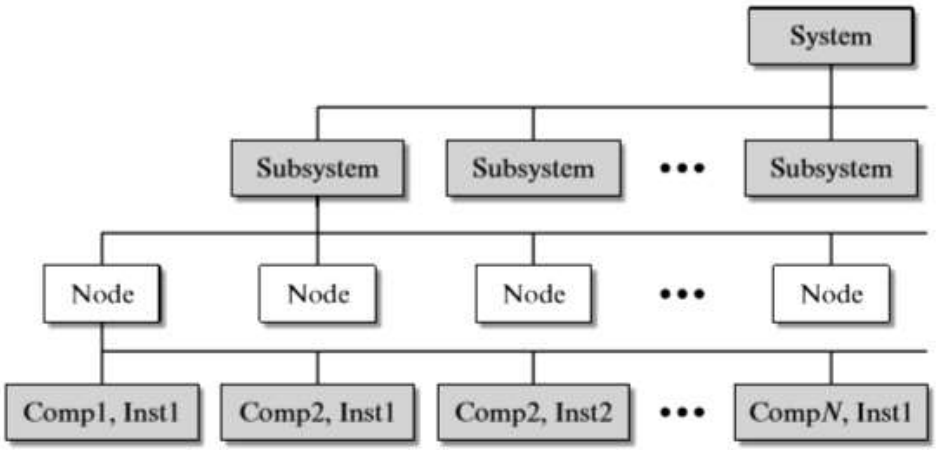


Рис. 8.15. Топологія еталонної архітектури JAUS

Node – це базовий програмний елемент топології JAUS, відповідальний за зв’язок з іншими Node. Це може бути Physical node, що являє собою фізичний пристрій, такий як робот або датчик; Virtual node – являє собою віртуальний пристрій, такий як симулятор або програмне забез-печення для моніторингу. Node має такі характеристики: унікальний ідентифікатор; тип, який описує його функціональність; IP-адресу, яка використовується для зв’язку з іншими Node; служби, які можуть бути використані іншими Node. Node з’єднуються один з одним за допомогою каналів зв’язку: проводними – Ethernet, RS-232 та безпроводними – Wi-Fi, Bluetooth. Типи зв’язків між Node можуть бути такі:

– одноранговий, коли два Node з’єднуються один з одним безпосередньо;

– клієнт-сервер, де один Node (клієнт) з’єднується з другим Node (сервером);

– мережна трансляція, коли один Node використовує мережний транслятор для зв’язку з іншим Node.

Згідно з архітектурою JAUS система має складатися з окремих одиниць (Subsystem), які є блоками управління. Вузли (Node) визначають здатність обробки задач в межах архітектури та перенаправляють задачі з обробки до визначених компонент. Компоненти (Components) надають різні можливості для виконання задачі. Комонентами можуть бути датчики, приводи робота. Компонування конкретної системи, підсистем, вузлів і компонентів визначається розробниками системи на основі вимог до завдання.

Система програмного керування роботів повинна забезпечувати формування, редагування та налаштування програм їхнього руху, а також підтримувати режим виконання в покроковому й автоматичному режимах. Потреба в реалізації цих функцій визначає схему інтерфейсів програмного забезпечення для роботів з комп’ютерним управлінням (рис. 8.16).

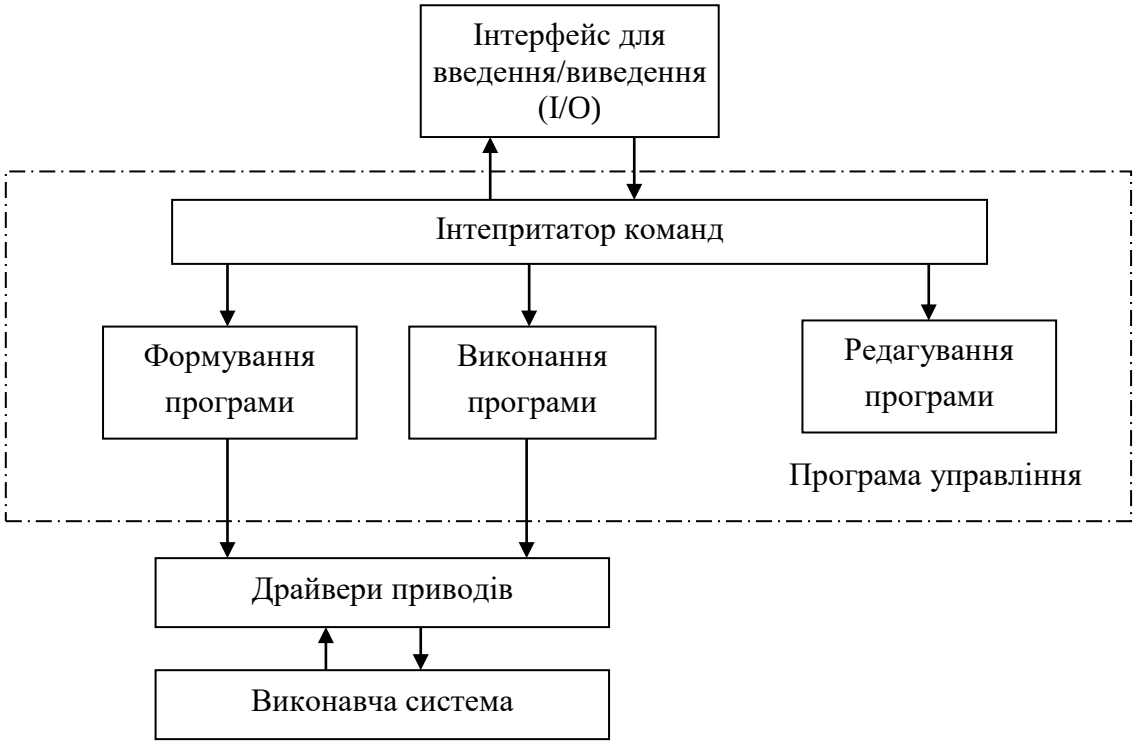


Рис. 8.16. Схема програмного забезпечення системи управління робота з комп'ютерним керуванням

У розробленні програмного забезпечення систем керування промисловими роботами особливе місце належить питанням організації структури даних, що пов'язане і з вимогами до об'єму займаної пам'яті, і з потребою у швидкому доступі до даних з метою запису, читання, редагування. Наприклад, в системі керування промислового робота УКМ-772 структура даних організована згідно зі структурою, як на рис. 8.17. Набір даних управління являє собою послідовність кадрів, які обробляються один за одним і можуть містити як основну (технологічну), так і допоміжну (керівну) інформацію. До технологічної інформації належить інформація про величини переміщення (абсолютні та відносні), швидкості переміщення, що пов'язані з одним кадром програми, інформацію про режим роботи та вид траєкторії переміщення робота для поточного та наступного кадру програми. Допускається також організація підпрограм керування («корутинів»), тобто визначених послідовностей кадрів, на які може передаватися керування з різних частин основної програми.

Організація структури даних, яка розділяє дані на такі, що описують точки позиціювання виконавчого механізму, і такі, що визначають послідовність проходження цих точок, дає змогу забезпечити швидкий доступ, простоту навчання робота та значно

заощаджує пам’ять комп’ютера (контролера), оскільки не потрібно запам’ятовувати всі послідовності точок траєкторії в порядку їхньої обробки механізмом. Достатньо буде лише запам’ятати різним точкам позиціювання і заповнити відповідним чином масив послідовності, зважаючи на збіг точок.

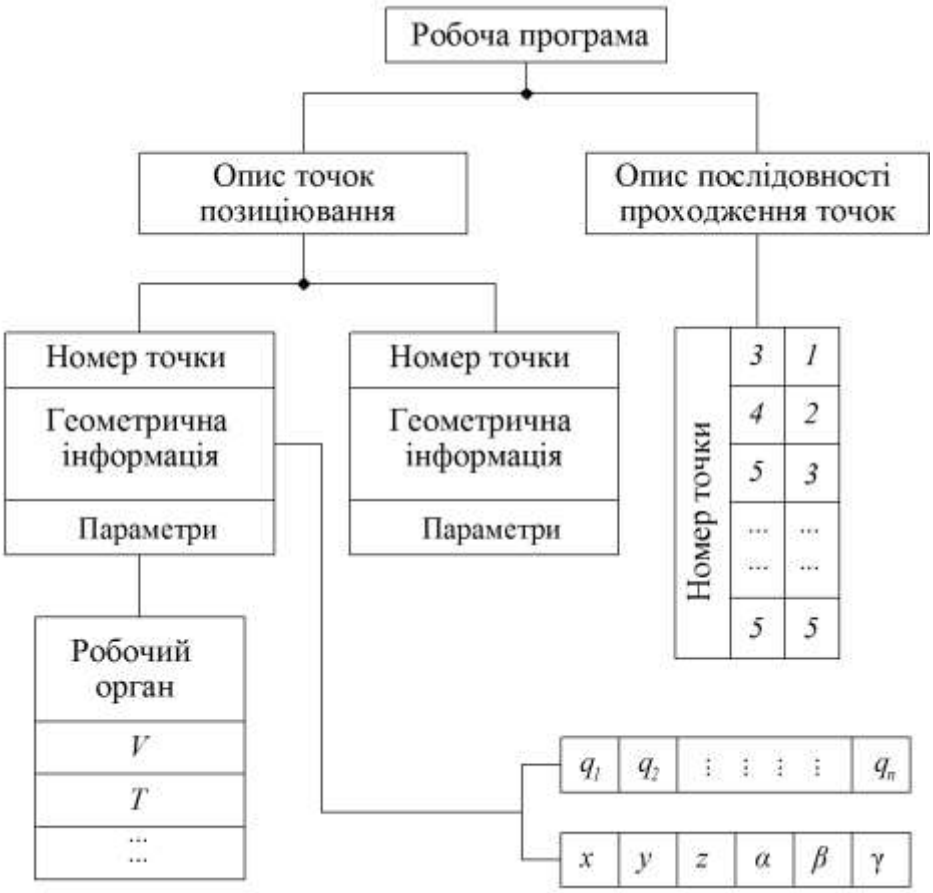


Рис. 8.17. Приклад структури даних в системі програмного забезпечення промислового робота

Особливістю функціонування програми робота є багатозадачність виконання виконавчих програм, оскільки функціонування системи керування являє собою взаємодію різних розрахункових процесів, які виконуються паралельно. Наприклад, це може бути розрахунок координат точок позиціювання, інтерполяція, розрахунок і формування сигналів управління. Такі процеси є закінченими розрахунковими процедурами і взаємодіють один з одними тільки на рівні даних, а оскільки не завжди вдається забезпечити синхронізацію процесів, дані накопичуються і передаються від однієї задачі до іншої за принципом «черги» (FIFO) (рис. 8.18).

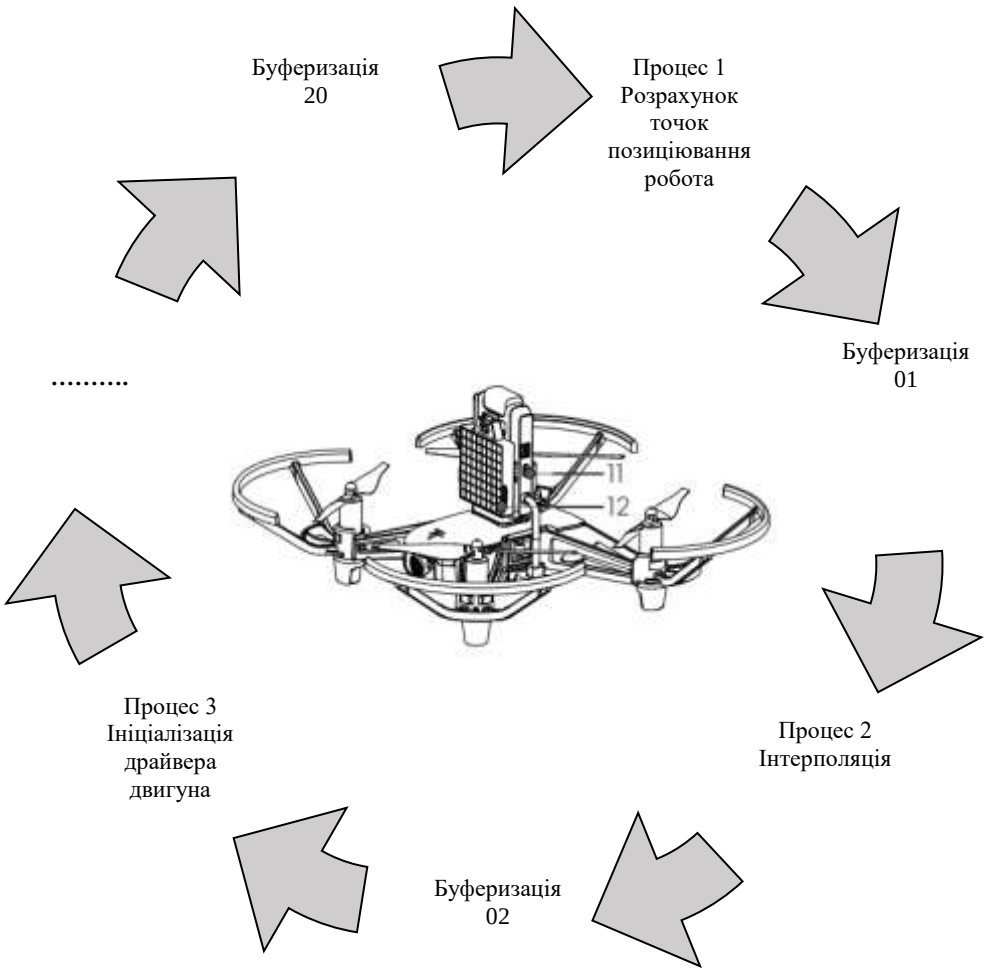
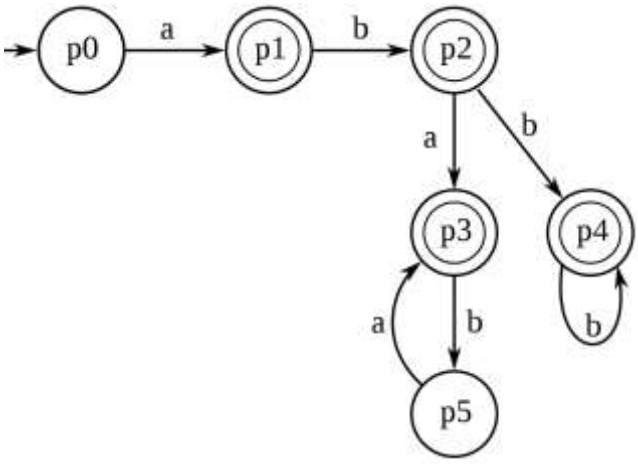


Рис. 8.18. Взаємодія процесів в системі управління програмного забезпечення

Механізм організації математичного і програмного забезпечення системи керування з обмеженим об’ємом пам’яті інформаційної системи робота може бути побудований за принципом скінченної множини станів, тобто моделі, використовуваної для опису зміни стану об’єкта залежно від поточного стану та інформації, отриманої ззовні. Скінченна множина станів визначає набір усіх можливих станів, які може мати система, наприклад, у простому випадку можуть бути такі стани, як «початковий стан», «стан 1», «стан 2» і т.д. У кожному стані системи можуть бути визначені правила, які вказують, в який стан система переходить після певних подій або дій. Ці правила визначають перехідну функцію. Початковий стан – це стан, в якому система перебуває після початку роботи або після скидання до початкового стану. Термінальні стани – це стани, в яких система може зупинитися або завершити свою роботу. Вхідні та вихідні події – це події, які впливають на стан системи (вхідні події) або є результатом перебування системи в певному стані (вихідні події). Представлення переходів між станами системи задається

у вигляді графа або таблиці, де кожен рядок є відповідним певному стану, а кожен стовпчик – певній вхідній події (рис. 8.19). Елементи таблиці вказують, в який стан система перейде після певної події.



Вихідний стан	Наступний стан		
	Вхідний символ *a*	Вхідний символ *b*	Будь-який інший символ
p0	p1	p0	p0
p1	p1	p2	p1
p2	p3	p4	p2
p3	p3	p5	p3
p4	p4	p4	p4
p5	p3	p5	p5

а *б*

Рис. 8.19. Способи представлення скінченних множин станів: *а* – приклад детермінованої системи; *б* – у вигляді таблиці

Детермінованою множиною називають таку множину станів, в якій немає позицій, що не містять жодного символу, і з будь-якого стану за будь-яким символом можливим є перехід не більш ніж в один стан.

Недетермінована множина є узагальненням детермінованої. Недетермінованості досягають двома способами: або можуть бути переходи зі стану в стан, які запускаються порожнім ланцюжком символів, тобто переходи без зовнішніх впливів, або з одного стану можна переходити в різні стани під впливом однієї події.

Виокремлюють дві групи скінченних множин станів: акцептори (розпізнавачі) і трансдуктори (перетворювачі). Акцептори і розпізнавачі (розпізнавачі послідовностей) дають двійковий результат, відповідаючи «так» або «ні» на запитання, чи приймаються системою вхідні дані. Всі стани системи автомата можуть бути такими, що приймають або не приймають. Якщо поточний стан є таким, що приймає, то послідовність, подана на вхід, приймається. Звичайно на вхід подаються символи (літери), дії не виконуються. На рис. 8.20 зображено механізм, який приймає слово «nice». В цьому механізмі єдиний допустимий стан – це «7».

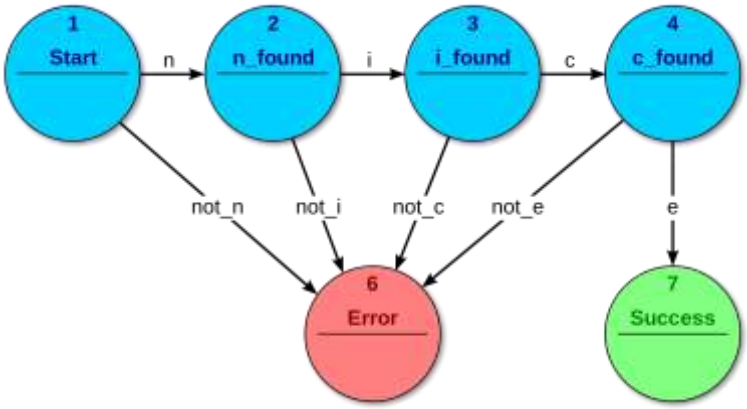


Рис. 8.20. Акцептор FSM: розбір рядка «nice»

Скінченна множина станів також може бути описана як така, що визначає мову, яка містить всі слова, розпізнавані цією множиною, але не ті, які нею відхиляються (тоді кажуть, що ця мова розпізнається скінченною множиною станів). За визначенням, мови розпізнавані – це регулярні вирази, тобто є деяка множина станів, яка розпізнає заданий вираз.

Програмні API роботів можуть застосовуватися з використанням скінченних множин станів. Наприклад, в табл. 8.1 наведено специфікацію команд Tello версії 1.0 для квадрокоптера DJI Tello RoboMaster TT.

Таблиця 8.1

Команди API Tello 1.0

Команда	Опис	Відклик
`command`	Увійти в командний режим	OK|False
`takeoff`	Автоматичний зліт	OK|False
`land`	Автопосадка	OK|False
`up` **xx**	Політ вгору [20, 500] см	OK|False
`down` **xx**	Політ вниз [20, 500] см	OK|False
`left xx`	Політ ліворуч [20, 500] см	OK|False
`right` **xx**	Летіть праворуч [20, 500] см	OK|False
`forward` **xx**	Політ вперед [20, 500] см	OK|False
`cw` **xx**	Поворот за годинниковою стрілкою [1, 360]°	OK|False
`ccw` **xx**	Поворот проти годинникової стрілки [1, 360]°	OK|False
`flip` **x**	Переворот [l, r, f, b]	OK|False
`speed` **xx**	Встановити швидкість [1, 100] см/с	OK|False
`Speed?`	Отримати поточну швидкість	xx
`Battery?`	Отримати поточний заряд батареї	xx
`Time?`	Отримати поточний час польоту	xx

Контролер дрона DJI Tello побудовано на мікроконтролері ESP32 і може бути запрограмований мовами Arduino, Scratch, Python та ін. Для програмування дрона Tello треба встановити UDP-з'єднання через Wi-Fi з портом 8889 Tello та почати надсилати команди. На дроні вже є встанолене програмне забезпечення, взаємодія з яким відбувається через програмний інтерфейс (API). Для зв'язку з Tello є версії API 1.0 SDK і 2.0 SDK відповідно.

Бібліотека `djitellopy` є однією з багатьох бібліотек для Tello, яка підтримує повністю всі API досліджуваного дрона. Альтернативою до вивчення API Tello можуть бути бібліотеки easytello, code4fun tello під Python.

Команди `connect()`, `takeoff()` та `land()` виконуються без параметрів і автоматично передають на систему управління стандартні сигнали, а команди `move_forward(90)` та `rotate_counter_clock-wise(90)`, за допомогою яких відбувається переміщення вперед на 90 см та поворот но 90° ліворуч потребує передавання параметра в метод об'єкта.

На квадрокоптері Tello встановлено датчики, за якими можна визначати телеметрію робота. *Телеметрія* – це сукупність технічних засобів і методів вимірювання, за допомогою яких відбувається фіксація на відстані різних фізичних величин та їхнє перетворення в інформаційний сигнал і передавання на відстань з будь-якої точки на віддалений термінал. *Терміналом* в такому разі вважають кінцеву або початкову частину деякої системи, від якої залежить її зв'язок із зовнішнім середовищем або доступ до системи із зовнішнього середовища.

Контрольні запитання

1. Що таке фізичний і електричний інтерфейси? В чому різниця?
2. Що таке програмний інтерфейс?
3. Які види електричних інтерфейсів вам відомі?
4. Назвіть види програмних інтерфейсів.
5. Що називають абстракцією? Які види абстакцій вам відомі?
6. Як працює клієнт-серверна архітектура?
7. Назвіть види інтерфейсів в клієнт-серверній ахітектури.
8. Поясніть принцип скінченної множини станів.

Рекомендована література

1. *Діктерук М. Г.* Основи автоматизації будівельних машин: навчальний посібник / М. Г. Діктерук, О. В. Човнюк. – Київ: КНУБА, 2006. – 232 с.
2. *Розробка* концепції системи керування роботом для штукатурних робіт на основі нейронної мережі *[Електроннрий ресурс]* / Д. Міщук, А. Бойченко // Гірничі, будівельні, дорожні та меліоративні машини. 2019. – №93. – С. 46-60. – Режим доступу: https://doi.org/10.32347/gbdmm2019.93.0501 / (дата звернення: 25.03.2024). – Назва з екрана.
3. *Системи* промислової автоматизації на основі IoT *[Електроннрий ресурс]* / Є. Міщук, Д. Міщук // Гірничі, будівельні, дорожні та меліоративні машини. – 2022. – №96. – С. 42–50. – Режим доступу: https:// doi.org/10.32347/gbdmm2020.96.0501 / (дата звернення: 20.02.2024). – Назва з екрана.
4. *Еванс Е.* Предметно-орієнтоване проектування. Структуризація складних програмних систем / Е. Еванс. – Вільямс, 2015. – 448 с.
5. *Васильєв О.* Програмування мовою Python / О. Васильєв. – Львів: Навчальна книга – Богдан, 2019. – 504 с.
6. *Орловський Б.В.* Інформаційні системи та пристрої в робототехніці та мехатроніці: нав. посібник / Б.В. Орловський. – Київ: КНУТД, 2018. – 416 с.
7. *Adaptive* Monte Carlo localization *[Електронний ресурс]* – Режим доступу: http://wiki.ros.org/amcl / (дата звернення: 03.08.2023). – Назва з екрана.
8. *Архангельский В.И.* Нейронные сети в системах автоматизации / В.И. Архангельский, И.Н. Богаенко и др. – Київ: Техніка, 1999. – 364 с.
9. *Gmapping [Електронний ресурс]* Режим доступу: http://wiki.ros.org/gmapping / (дата звернення: 14.02.2024). – Назва з екрана.
10. *Siciliano Bruno*. Handbook of robotics / B. Siciliano, O. Khabib, eds. – Berlin: Springer-Verlag Heidelberg, 2008. – 1611 p.

Модуль 3. ПРОГРАМУВАННЯ РОБОТОТЕХНІЧНИХ СИТЕМ

Лекція № 9. Основи теорії автоматичного керування. Автономні безпілотні системи

Лекція № 10. Основи комп'ютерного зору

Лекція № 9. Основи теорії автоматичного керування. Автономні безпілотні системи

Теорія автоматичного керування вивчає методи та алгоритми керування динамічними системами для досягнення заданих цілей або вимог, а також розроблення та аналіз різних методів керування, таких як пропорційно-інтегральні диференційні (ПІД) контролери, лінійні та нелінійна системи керування, оптимальне керування, теорію стійкості. Основною метою теорії автоматичного керування є розроблення алгоритмів керування, які дають системі змогу досягати програмованих станів або підтримувати заданий стан, зокрема, це може бути стабілізація системи, підтримка певного значення вихідної змінної (наприклад, швидкості або положення), або оптимізація певного критерію, такого як час реакції або енергоефективність.

Система автоматизованого керування є інструментом, який застосовують для досягнення цілей теорії автоматизованого керування. Основними елементами систем автоматизованого керування є виконавчі пристрої, пристрої для збору інформації та пристрої керування (контролери).

Керуванням в системах автоматизованого управління називають процес формування керівних впливів, що забезпечують необхідний стан або режим роботи об'єкта керування. Процес керування можна подати у вигляді послідовності різних впливів (передавання енергії) в системі (рис. 9.1).

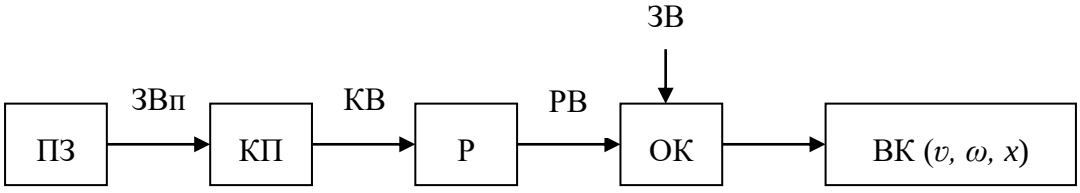


Рис. 9.1. Процеси системи керування: ПЗ – програмний задатчик; КП – керівний пристрій; Р – регулювальний орган; ОК – об'єкт керування; ВК – вихідні координати; ЗВп – заданий вплив; КВ – керівний вплив; РВ – регулювальний вплив; ЗВ – збурювальний вплив

Керівний вплив – це заданий вплив керівного пристрою на об’єкт керування.

Збурювальний вплив – це вплив зовнішнього середовища на об’єкт керування.

Задавальний вплив – це вплив на вході керівного пристрою, сформований програмним задатчиком.

Зворотний вплив об’єкта керування на керівний пристрій називають *контрольованим впливом*.

Вихідні координати – це величини, які характеризують поточний стан об’єкта.

Системи автоматизованого керування класифікують за різними ознаками.

За типом керованої системи:

– одноканальні – це системи з одним входом і одним виходом;

– багатоканальні (мультивимірні) – це системи з більш ніж одним входом або виходом;

– лінійні – це системи, які можуть бути описані лінійними рівняннями;

– нелінійні – системи, які мають нелінійні характеристики або не можуть бути адекватно описані лінійними рівняннями.

За методом керування:

– з розімкненим зв’язком – метод, за якого вхідні сигнали задаються наперед і не залежать від реакції системи (рис. 9.2, *а*);

– зі зворотним зв’язком – метод, за якого вихідні дані системи порівнюють з бажаним значенням та відповідно до цього коригують вхідні сигнали (рис. 9.2, *б*);

– комбіновані системи.

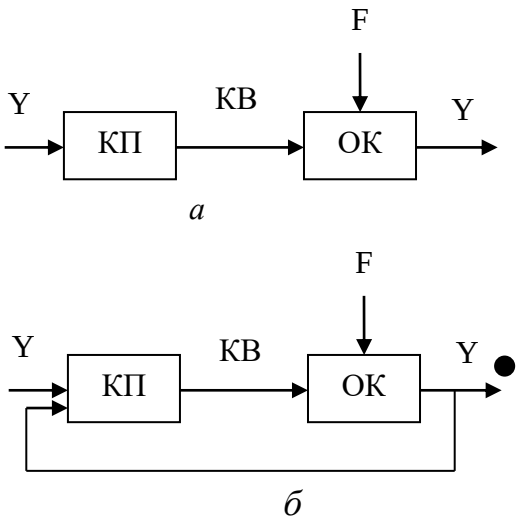


Рис. 9.2. Різновиди систем керування: *а* – розімкнена; *б* – зі зворотним зв’язком; КП – керівний пристрій; КВ – керівний вплив; F – перешкоди; X – вплив, що задається; Y – вихідні координати

За призначенням:

- регуляторні системи, призначені для збереження певного стану або параметрів на певному рівні;
- адаптивні системи, які можуть автоматично адаптуватися до змін в динаміці керованої системи або зовнішніх умов;
- оптимальні системи, призначені для досягнення оптимального критерію, такого як мінімальність чи максимальність певного функціоналу.

За характером входів та виходів:

- цифрові системи керування (імпульсні, релейні);
- аналогові системи керування (безперервні).

За структурою:

- прості системи керування з одним контуром управління;
- складні системи керування багатьма взаємопов'язаними контурами управління та підсистемами.

Якщо характер зміни керованої величини можна передбачити на підставі раніше виконаних вимірів контрольованих зовнішніх впливів, а характеристика об'єкта впливу відома, то застосовують принцип ***керування за збуренням*** (рис. 9.3). Цей принцип означає, що система вносить корективи у свою роботу перед появою збурень або змін в середовищі. Застосовується в розімкнених системах керування.

Рис. 9.3. Функціональна схем керування за збуренням: РО – регулювальний орган; САК – система автоматичного контролю вихідного параметра; ВМ – виконавчий механізм; μ(t) – керівний вплив; *Z*(t) – збурювальний вплив; *X*(t) – вхідний параметр; Y(t) – значення вихідного параметра

Якщо під час роботи машини зявляється неконтольований фактор, що збурює режим роботи, то застосовують принцип ***керування за відхиленням*** (рис. 9.4). Цей принцип означає реагування системи на відхилення вихідних даних від бажаного значення. Контрольний сигнал,

отриманий з вимірювань вихідних даних системи, порівнюється з бажаним значенням, і відповідно до різниці між ними вносять корективи в керівні сигнали.

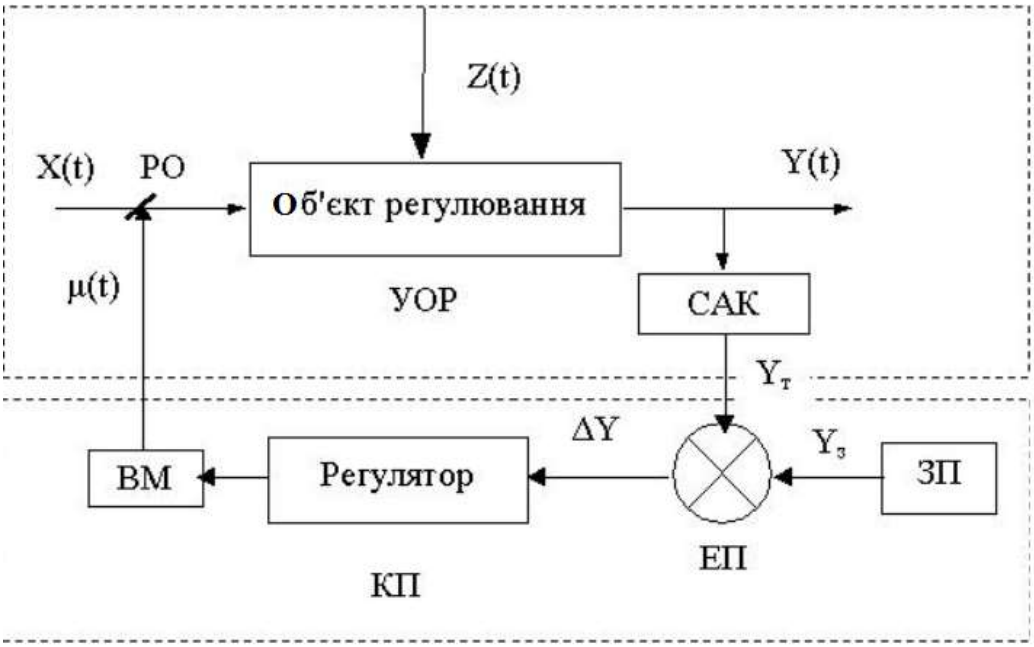


Рис. 9.4. Функціональна схема керування за відхиленням: КП – керівний пристрій; РО – регулювальний орган; САК – система автоматичного контролю вихідного параметра; УОР – узагальнений об’єкт регулювання; ЗП – задавач; ЕП – елемент порівняння; ВМ – виконавчий механізм; μ(t) – керівний вплив; *Z*(t) – збурювальний вплив; *X*(t) – вхідний параметр; Y(t) – значення вихідного параметра

Комбінації принципів керування за збуренням та за відхиленням використовують для досягнення кращої ефективності та стійкості системи.

ПІД-регулятор (пропорційно-інтегрально-диференціальний регулятор) – це один з найпоширеніших типів контролерів, який використовують в системах автоматичного керування для регулювання виходу системи так, щоб досягнути бажаного значення.

Рівняння ПІД-закону мають такий вигляд:

$$u = P \cdot \varepsilon(t) + I \cdot \int \varepsilon(t)dt + D \cdot \frac{d\varepsilon(t)}{dt}, \qquad (9.1)$$

де $\varepsilon(t)$ – похибка в системі керування; *P*, *I*, *D* – коефіцієнти, відповідно пропорційної, інтегральної та диференціальної складових.

Пропорційна складова в регуляторі пропорційно відповідна величині відхилення між поточним значенням вихідної величини та бажаним значенням. Це означає, що величина керівного сигналу

залежить від того, наскільки великим є відхилення. Що більше відхилення, то більший вихідний сигнал генерується регулятором.

Інтегральна складова накопичує величину відхилення від бажаного значення в часі та пропорційно використовує це накопичене відхилення для генерації керівного сигналу. Це дає змогу вирівнювати систему та усувати систематичні помилки.

Диференціальна складова пропорційно відповідна швидкості зміни відхилення. Вона використовується для зменшення часу реакції системи та запобігання зайвим коливанням, спричиненим різкими змінами вхідних даних.

В роботах застосовують три способи керування – *програмне*, *адаптивне* та *інтелектуальне*. Практично тільки програмне керування знайшло застосування в чистому вигляді, хоча і до нього можуть додавати елементи адаптації. Загалом всі три способи керування застосовують комплексно. *Адаптивне* управління зазвичай ґрунтується на основі програмного як наступний рівень керування. *Інтелектуальне* керування в свою чергу реалізується як надбудова над першими двома рівнями. Назви систем керування конкретних роботів зазвичай визначаються використаним в них способом керування.

За ступенем участі людини-оператора в процесі керування розрізняють системи *автоматичного*, *автоматизованого* та *ручного* управління.

За типом контролю керованих змінних вирізняють такі системи:

- керування з контролем положення;
- швидкості;
- зусилля (моменту).

Всі названі способи керування застосовують в комбінації, або різні способи за різними координатами, або з послідовним переходом від одного до іншого, або у вигляді функціональної залежності керованої змінної (наприклад, керування за силою, величина якої задається як функція від положення).

Структура модулів систем керування роботів припускає багато-рівневу організацію, вона охоплює стратегічний, тактичний і виконавчий рівні керування, що мають доступ до сенсорної інформації для розв’язування завдань керування певного рівня.

На рис. 9.5 показано структурну блок-схему жорсткопрограмованого робота, яка має виконавчі механізми приводу, пульт керування,

перетворювач, датчики положення і «блок пам’яті». Для виконання цілеспрямованих дій робот повинен мати програму, записану на програмоносії у певному коді, або бути «навченим». Введення програми виконує оператор, для цього він перемикає пульт на ручне керування і послідовно виконує весь цикл дій. Відтак датчики положення передають сигнали у «пам’ять» робота, де відбувається їхній запис у вигляді програми. Після введення програми в блок керування пульт керування переводиться на робочий (автоматичний) режим роботи.

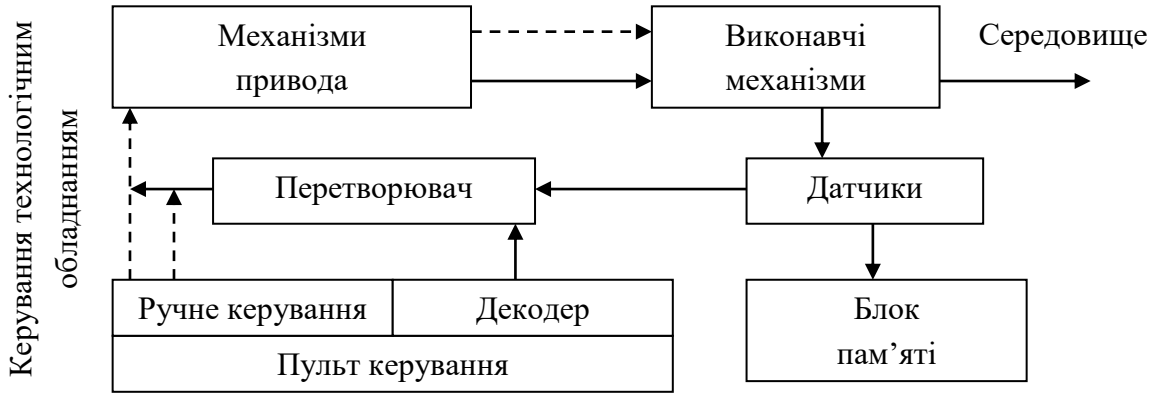


Рис. 9.5. Блок-схема керування жорсткопрограмованим роботом

Для дистанційного керування кожним «ступенем вільності» виконавчих органів роботів застосовують силові системи стеження, основною відмітністю яких від звичайних стежних систем є властивість відображення на задавальному органі зусилля, що розвивається. При цьому розрізняють два принципово різних види силових стежних систем: системи з пасивним відображенням зусилля і системи з активним відображенням зусилля – обернені стежні системи.

Системи з пасивним відображенням зусилля дають оператору змогу відчувати зусилля, що розвиваються виконавчим органом лише в процесі зміни положення органа керування. При цьому зворотний зв’язок не виявляє жодної дії на положення органа керування.

Обернені стежні системи забезпечують найповнішу імітацію впливу на орган управління і мають одночасно властивості активного відображення зусилля і оберненості з погляду стеження за положенням органа керування, забезпечуючи двосторонню дію, оскільки рух передається у двох напрямах. На рис. 9.6 показано блок-схему оберненої стежної системи. Оператор прикладає до органів керування момент M_{oi},

який перетворюється на керівну дію α_{oi}. Будова відображення зусилля вимірює момент $M_{ні}$ на виході приводу виконавчого механізму і впливає на органи керування визначеним моментом, під дією якого величина переміщення окремих керівних ланок узгоджується з переміщенням керівного органа загалом.

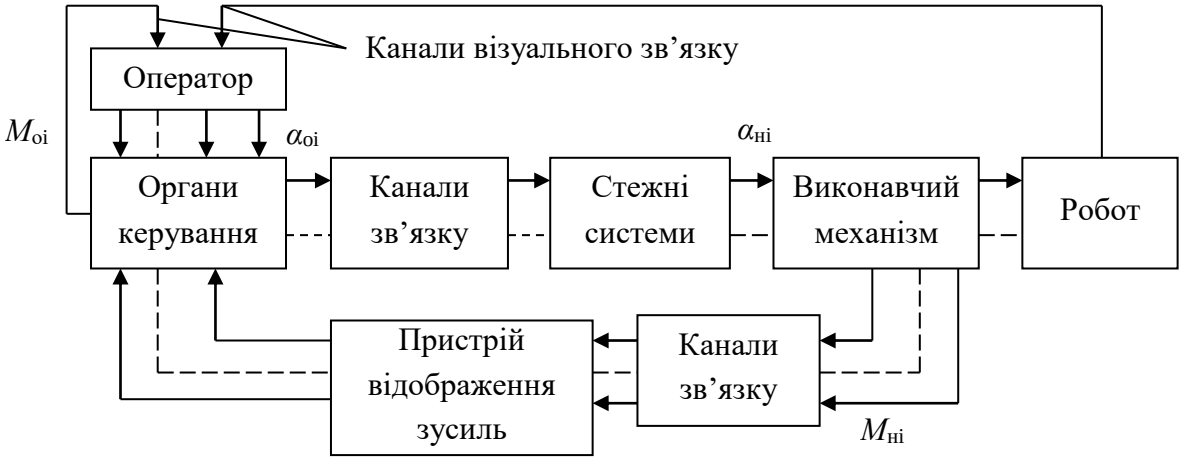


Рис. 9.6. Блок-схема оберненої стежної системи

На рис. 9.7 показано блок-схему системи програмного керуванням робота, яка побудована за принципом синхронного мікропрограмного контролера з кінцевим числом стану та жорстким циклом керування. Система складається з накопичувача інформації, блока керування, блока вводу/виводу, оперативно-логічного блока, блока синхронізації, мікропрограмного контролера, вимірювальної схеми.

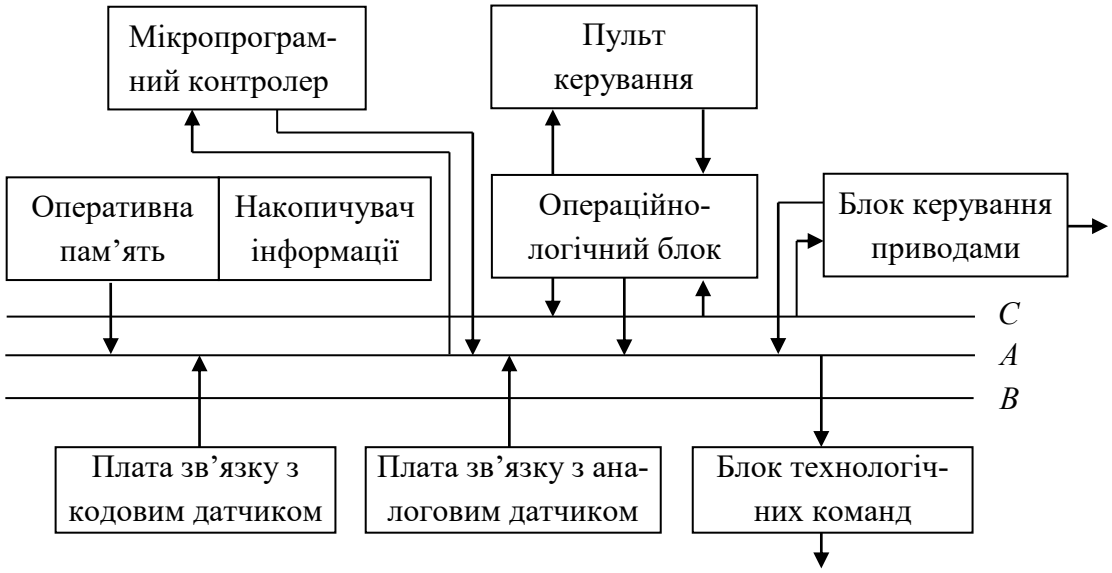


Рис. 9.7. Блок-схема системи програмного керуванням

Автономні безпілотні системи – це технологічні рішення, які дають змогу таким технічним системам, як роботи, автомобілі, літаки, човни, функціонувати та ухвалювати рішення без прямого втручання людини. Ці системи використовують різні типи сенсорів, алгоритми обробки інформації, зокрема елементи штучного інтелекту, машинного навчання, інтегрованої навігації, та системи керування для виконання завдань у визначеному середовищі. Залежно від типу керування безпілотники можна поділити на дистанційнокеровані оператором і автоматичні, що виконують рух за заданою програмою. Залежно від типу середовища, в якому рухається апарат (вода, земля, повітря), безпілотники можна класифікувати на підводні, надводні, наземні і повітряні (літальні). Основні складові автономних безпілотних систем наведено на рис. 9.8.

Рис. 9.8. Компоненти та технології автономних безпілотних систем

Керування автономною безпілотною системою на високому рівні автоматизації вирішує ряд завдань, серед яких основними є виявлення та реагування (рис. 9.9), і має такі особливості:

- виявлення перешкод (*detection*): система повинна виявляти перешкоди на шляху, зокрема інші автономні безпілотні системи, транспортні засоби, живих істот або будь-які інші об’єкти, що можуть вплинути на безпеку руху, а також виявляти різні дорожні ситуації, такі як зміна

траєкторій руху, об'їзд інших засобів, перекриття шляху та інші події, які можуть впливати на рух;

- розпізнавання розмітки, дорожніх знаків і сигналізацій (*traffic sign and signal recognition*): система повинна вміти розпізнавати різноманітну допоміжну інформацію та інтерпретувати її для правильної реакції;
- моніторинг руху живих істот (*pedestrian and cyclist monitoring*): важливо виявляти рух живих істот передбачаючи їхні можливі дії та прогнозуючи траєкторії для безпечного спільного руху;
- реагування на перешкоди і допоміжну інформацію (*reaction*);
- планування траєкторії (*trajectory planning*): зважаючи на інформацію про навколишнє середовище виконується розроблення безпечних та ефективних траєкторії для руху;
- адаптація до змін у середовищі (*adaptation to environmental changes*);
- екстремальне управління (*extreme control*): система повинна мати здатність екстреного управління для уникнення аварій.

Рис. 9.9. Завдання автономних безпілотних систем

Для опису рівнів системи автоматизації автономних транспортних систем застосовують термін «таксономія». Наприклад, згідно з Міжнародною асоціацією автомобільних інженерів (SAE) та стандартом J3016 визначено п'ять рівнів автоматизації від повного контролю до повної автоматизації без людського втручання:

- **Рівень 0**: жодної автоматизації, людина повністю відповідає за управління транспортним засобом.
- **Рівень 1**: транспортним засособом людині допомагають керувати такі системи, як круїз-контроль або система автоматичного управління утримання, при цьому людина повинна бути готовою перебрати контроль на себе в будь-який момент.
- **Рівень 2**: часткова автоматизація, коли транспортний засіб може виконувати певні функції управління, такі як управління швидкістю та втримання на смузі руху, але людина повинена бути готовою перебрати керування на себе в будь-який час.

- **Рівень 3**: умовно повна автоматизація, коли людина може відвернути увагу від керування за певних умов, наприклад, на автостраді. Автомобіль може виконувати багато функцій самостійно, але можливим є втручання людини за потреби.
- **Рівень 4**: цілковита автоматизація, коли транспортний засіб здатен керувати всіма аспектами переміщення в обмежених умовах, таких як певні географічні області або конкретні сценарії.
- **Рівень 5**: транспортний засіб повністю автономний і немає потреби у втручанні людини за будь-яких умов.

В процесі проєктування безпілотних систем керування завжди треба брати до уваги також умови їхньої комфортної роботи. Наприклад, для автомобіля критичним прискоренням є значення 5 м/с^2, а нормальним – 2 м/с^2. Ці знання вплинуть на алгоритм системи керування, яка буде визначати дистанцію до об'єктів та здійснюватиме завчасне і безпечне гальмування. Дистанція, яку проходить автомобіль під час постійного розгону/гальмування визначається за формулою, с:

$$d = \frac{\upsilon^2}{2a}, \tag{9.2}$$

де υ – максимальна швидкість розгону/гальмувння, м/с; a – прискорення м/с^2.

Одометрія – це спосіб оцінювання переміщення за допомогою даних, отриманих із сенсорів руху. Припустимо, що робот має два колеса, здатних рухатись вперед і назад, які розміщені одне навпроти одного паралельно на однаковій відстані від центра робота. Далі припустимо, що кожен двигун має поворотний датчик, таким чином можна визначити, що колесо зробило рух вперед або назад на «одиницю» відстані. Ця одиниця величини є відношенням довжини обода колеса до роздільної здатності датчика. Якщо ліве колесо рухалось вперед на одну позицію, тоді як праве колесо залишалося нерухомим, тоді праве колесо поводить себе як центр кола, яким переміщується ліве колесо в напрямку за годинниковою стрілкою. Оскільки одиниця відстані, як правило, мала, можемо припустити, що ця дуга є прямою. Таким чином, вихідна позиція, остаточне положення лівого колеса і позиція правого колеса утворюють трикутник, подібний до трикутника, який утворюють вихідне положення центра, остаточне положення центра і позиція правого колеса, оскільки від центра робота однакова відстань до кожного з коліс. Тоді величина зміни позиції центра робота під час повороту дорівнюватиме половині одиниці

відстані, пройденої рухомим колесом, а кут повороту може бути визначений за допомогою теореми синусів. Це і є прикладом обометрії.

Розглянемо архітектуру інформаційної системи безпілотної машини на прикладі автомобіля (рис. 9.10).

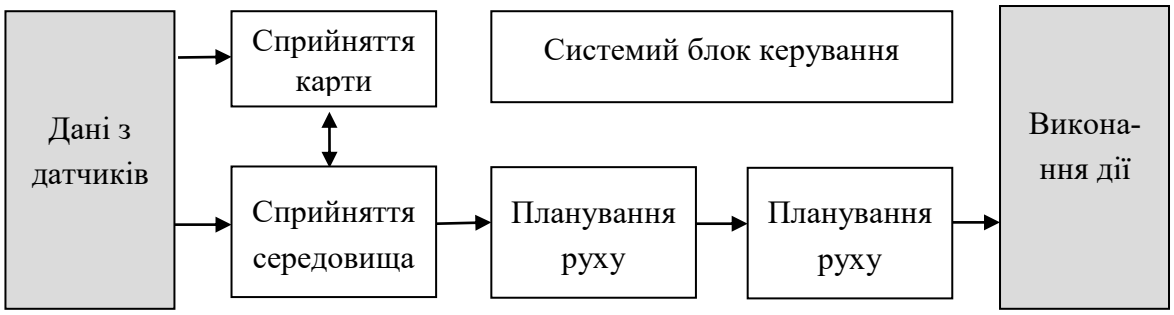


Рис. 9.10. Високорівнева структура програмної архітектури безпілотної системи керування автомобілем

У безпілотних системах керування сприйняття навколишнього середовища складається з двох важливих частин: локалізації транспортного засобу у просторі, а також класифікації та визначення місцезнаходження важливих елементів довкілля. Модуль локалізації приймає безліч потоків інформації, наприклад, поточне місцезнаходження (GPS), вимірювання прискорення (IMU) й одометрію коліс. Вся ця інформація оброблюється і об’єднується для отримання точного розміщення транспортного засобу. Для більшої точності деякі модулі локалізації потребують також даних LIDAR та камер. Завдання локалізації поділяють на проблеми класифікації та локалізації елементів довкілля, зокрема, виявлення динамічних та статичних об’єктів у навколишньому середовищі.

Модуль динамічного виявлення об’єктів потребує набору камер та хмари точок LIDAR для створення тривимірних обмежувальних рамок навколо динамічних об’єктів на сцені. Тривимірні обмежувальні рамки кодують клас або тип об’єкта, а також точне положення, орієнтацію та розмір об’єкта. Після виявлення динамічні об’єкти повинні відстежуватися за часом за допомогою модуля стеження. Модуль трекера надає інформацію як поточне становище динамічних об’єктів, так й історію їхнього переміщення у навколишньому середовищі. Історія шляху разом з дорожніми картами використовується для прогнозування майбутнього шляху всіх динамічних об’єктів. Зазвичай цим займається

модуль прогнозування, який поєднує всю інформацію про динамічний об’єкт та поточне середовище для прогнозування траєкторії всіх динамічних об’єктів.

Модуль виявлення статичних об’єктів також використовує комбінацію вхідних даних камери та даних LIDAR для ідентифікації важливих статичних об’єктів на сцені. До таких важливих даних належить смуга руху безпілотного транспортного засобу і розміщення регулювальних елементів, таких як знаки і світлофори.

Мапи середовища створюють кілька різних типів уявлення про поточне середовище довкола автономного автомобіля. Є три типи мап: сітка зайнятості, мапа локалізації та докладна дорожня мапа.

Сітка зайнятості – це мапа всіх статичних об’єктів у навколишньому середовищі автомобіля. Для побудови сітки зайнятості переважно використовується LIDAR. Спочатку до даних LIDAR застосовується набір фільтрів, щоб їх можна було використовувати у сітці зайнятості. Наприклад, видаляються керовані точки поверхні та точки динамічних об’єктів. Мапа сітки зайнятості представляє навколишнє середовище у вигляді набору осередків сітки і визначає ймовірність того, що кожен осередок зайнятий. Це дає змогу вирішити проблему з невизначеністю даних вимірювань та з часом покращувати мапу.

Мапа локалізації створюється також з огляду на дані LIDAR та відеокамер і використовується модулем локалізації для покращення оцінювання стану. Дані датчиків порівнюються з цією мапою під час руху, щоби визначити рух автомобіля відносно мапи локалізації.

Докладна дорожня мапа є мапою дорожніх ділянок, що відображає умови руху, за яких в даний час рухається автономний транспортний засіб. Вона фіксує знаки та розмітку смуг руху таким чином, щоб їх можна було використовувати для планування руху. Ця мапа традиційно являє собою комбінацію попередньо записаних даних та інформації, що надходить з поточного статичного середовища, зібраної стеком сприйняття.

Модулі мапування та сприйняття зовнішнього середовища значною мірою взаємодіють один з одним, підвищуючи продуктивність обох модулів. Наприклад, модуль сприйняття надає статичну інформацію про навколишнє середовище, потрібне оновлення докладної дорожньої мапи, яка потім використовується модулем прогнозування для створення більш точних прогнозів щодо динамічних об’єктів.

Планування руху безпілотних автомобілів – завдання, яке складно вирішити у рамках єдиного інтегрованого процесу. Натомість у більшості безпілотних автомобілів нині використовується декомпозиція, яка поділяє проблему на кілька рівнів абстракції в такий спосіб. На найвищому рівні планування місій займаються довгостроковим плануванням і визначають місію на всьому горизонті виконання завдання з керування автомобілем – від поточного місця розміщення до дорожньої мережі в кінцевому пункті призначення. Для того щоби знайти повне завдання, планувальник місії визначає оптимальну послідовність ділянок дороги, що з'єднують пункт відправлення та пункт призначення, а потім передає цю послідовність на наступний рівень. Планувальник поведінки – це новий рівень абстракції, що дає змогу вирішувати завдання з короткострокового планування. Планувальник поведінки відповідає за розроблення комплексу безпечних дій або маневрів, які потрібно виконувати під час руху маршрутом. Прикладом рішень, ухвалених планувальником поведінки, може бути питання, чи слід транспортному засобу виїхати на сусідню смугу, зважаючи на бажану швидкість та прогнозовану поведінку прилеглих транспортних засобів. Крім рішень про маневр, планувальник поведінки також передбачає низку обмежень, яких слід дотриматися під час кожній дії, наприклад, скільки часу потрібно залишатися на поточній смузі до переміщення на іншу смугу руху. Зрештою, місцевий планувальник виконує негайне або реактивне планування та відповідає за визначення конкретного маршруту та профілю швидкості руху. Місцевий планувальник має бути чітким, безпечним та ефективним, зважати на усі поточні обмеження, що накладаються навколишнім середовищем та маневром. Щоби скласти такий план, місцевий планувальник об'єднує інформацію, надану планувальником поведінки, дані про заповнюваність, ліміти експлуатації транспортних засобів та інші динамічні об'єкти навколишнього середовища. Результатом роботи місцевого планувальника є спланована траєкторія, що являє собою комбінацію бажаної траєкторії та профілю швидкості на короткий період часу в майбутньому.

Контролер транспортного засобу приймає задану траєкторію та перетворює її на набір точних команд керування, які може використовувати автомобіль. Типовий контролер задає завдання управління на поздовжнє та поперечне керування. Поперечний

контролер визначає кут повороту, потрібний для підтримки запланованої траєкторії, а поздовжній контролер регулює дросельну заслінку і гальмівну систему для досягнення правильної швидкості. Обидва контролери розраховують поточні помилки та відстежують виконання локального плану, а також коригують поточні команди управління, щоб надалі звести помилки до мінімуму.

Системий блок керування – це модуль, який безперервно контролює всі аспекти автономного автомобіля та виконує відповідні дії у разі збою підсистем. Модуль складається з двох частин: диспетчера апаратного забезпечення та керівника програмного забезпечення. Диспетчер апаратного забезпечення постійно контролює всі апаратні компоненти з погляду наявності несправностей, таких як відмова датчика, брак результатів вимірювань або погіршення якості інформації. Ще один обов'язок диспетчера апаратного забезпечення полягає в постійному аналізі виходів апаратного забезпечення щодо наявності будь-яких виходів, не відповідних тій галузі, в якій автономний автомобіль був запрограмований на роботу, наприклад, якщо один з датчиків камери заблокований паперовим пакетом або коли падає сніг, що ушкоджує дані хмари точок LIDAR. Керівник програмного забезпечення відповідає за перевірку цього програмного стека, щоб переконатися, що всі елементи працюють належним чином на потрібних частотах та забезпечують повноцінні вихідні дані. Керівник програмного забезпечення відповідає за аналіз невідповідностей між вихідними даними всіх модулів.

Контрольні запитання

1. Що означає термін «одометрія»?
2. Які системи автоматизованого управління вам відомі?
3. Що таке управління за збудженням?
4. Що таке управління за відхиленням?
5. Що таке ПІД-регулятор?

Лекція № 10. Основи комп’ютерного зору

Комп’ютерний зір – це галузь науки, що займається розробленням та вивченням методів і технологій, які дають комп’ютерам змогу отримувати, аналізувати та розуміти візуальні дані зображень та відео. Комп’ютерний зір виконує ряд задач, таких як розпізнавання і визначення обєктів на зображеннях, відстеження руху, аналіз зображень тощо.

Комп’ютерний зір належить до теорії та технологій створення штучних систем, які отримують інформацію у вигляді зображень. Однією із задач комп’ютерного зору є застосування теорій комп’ютерної графіки для створення інформаційних моделей. Прикладами систем комп’ютерного зору можуть бути:

- системи керування процесами (автономні транспортні засоби);
- системи відеоспостереження;
- системи організації інформації (наприклад, для індексації баз даних зображень);
- системи моделювання об’єктів або навколишнього середовища (аналіз зображень, топографічне моделювання).

Галузь обробки статичних (фото) та динамічних (відео) зображень розрізняють залежно від різновидів виконуваних підзадач поділяється так: обробка й аналіз, власне комп’ютерний зір, машинний зір, візуалізація.

Обробку і аналіз зображень зосереджено переважно на роботі з двовимірними зображеннями, зокрема для вирішення таких задач, як перетворення одного зображення в інше, наприклад, попіксельне збільшення контрастності, виділення країв, усунення шумів чи геометричні перетворення, такі як обертання зображення.

Комп’ютерний зір застосовують для обробки тривимірних сцен, спроєктованих на одне чи декілька зображень, наприклад, відновлення структури чи іншої інформації про тривимірну сцену за одним чи декількома зображеннями. Комп’ютерний зір часто також вирішує задачу з формування припущень відносно того, що представлено на зображеннях.

Машинний зір зосереджується на промисловому застосуванні, наприклад, локалізації автономних роботів і системах візуальної перевірки та вимірювання. Це означає, що технології датчиків зображення і теорії управління пов’язані з обробкою відеоданих для

управління роботом і обробки даних в реальному часі, що виконується апаратно чи програмно.

Візуалізація пов'язана з процесом створення зображень, але іноді застоовується для обробки й аналізу.

Кожна зі сфер застосування комп'ютерного зору пов'язана з низкою задач.

Розпізнавання – це класична задача комп'ютерного зору, мета якої полягає в обробці зображень і визначенні того, чи містять зображення деякий характерний об'єкт, особливість чи активність. Ця задача може бути достовірно і легко розв'язана людиною, але досі не вирішена задовільно в комп'ютерному зорі в загальному випадку. Відомі методи вирішення цієї задачі ефективні тільки для окремих об'єктів, таких як прості геометричні об'єкти (наприклад, багатокутники), людські обличчя, друковані чи рукописні символи, автомобілі, і лише у визначених умовах, зазвичай це певне освітлення, фон і положення об'єкта відносно камери. Існує кілька задач, що ґрунтуються на розпізнаванні, наприклад: виявлення, пошук зображень за вмістом, ідентифікація, оцінка положення, оптичне розпізнавання символів.

Виявлення полягає у перевірці на наявність визначених умов і може ґрунтуватися на відносно простих і швидких обчисленнях. Розпізнавання застосовують для класифікації одного або декількох попередньо заданих чи вивчених об'єктів або класів об'єктів на зображенні. Ідентифікація – це розпізнавання індивідуального екземпляра об'єкта, наприклад, людського обличчя чи автомобіля. Пошук зображень за вмістом полягає у знаходженні всіх зображень серед великого набору зображень, які мають певний вміст, що може бути визначений різними шляхами, наприклад, в термінах схожості з конкретним зображенням (знайти всі зображення, схожі на дане зображення). Оцінка положення – це визначення положення чи орієнтації певного об'єкта відносно камери, наприклад, розвантаження об'єкта з конвеєра на лінії складання. Оптичне розпізнавання символів полягає у розпізнаванні символів на зображеннях друкованого чи рукописного тексту, зазвичай для перекладу в текстовий формат (наприклад, ASCII).

Оціннювання руху – це задачі, в яких послідовність зображень (відеодані) обробляється для знаходження швидкості кожної точки

зображення чи 3D-сцени або навіть швидкості камери, яка виконує зйомку. Приклади таких задач:

- одометрія – визначення руху камери (переміщення і обертання) в тривимірному просторі на основі серії знімків;
- стеження, тобто прямування за переміщенням об'єкта (наприклад, машин чи людей);
- оптичний потік – це технологія визначення руху кожної точки зображення відносно площини зображення, тобто видимий рух, що є результатом руху як самої точки, так і камери.

Відновлення сцени має за мету відтворення тривимірної моделі сцени із заданого зображення сцени або відеопотоку. В найпростішому випадку моделлю може бути набір точок тривимірного простору. Складніші методи відтворюють повну тривимірну модель.

Задача з *відновлення зображень* – це видалення шумів (шум датчика, розмитість об'єкта, що рухається тощо). Найпростішим підходом до вирішення цієї задачі є різноманітні типи фільтрів, таких як фільтри низьких чи середніх частот. Складніші методи використовують уявлення про те, який вигляд повинні мати ті або інші ділянки зображення, і на основі цього їхні перетворення. Більш високого рівня видалення шумів досягають протягом первинного аналізу відеоданих на наявність різноманітних структур, таких як лінії чи межі, а потім управління процесом фільтрації на основі цих даних.

Комп'ютерний зір має справу з комп'ютерною графікою – галуззю інформатики, яка займається створенням та обробкою графічних зображень за допомогою комп'ютерних технологій, він охоплює аспекти створення 2D- та 3D-графіки, обробки зображень, анімації, віртуальної реальності. Розрізняють три види комп'ютерної графіки – це растрова графіка, векторна і фрактальна графіка, які відрізняються принципами формування зображення для відображення на екрані монітора.

Фізика зображення пов'язана з розподілом енергії відбитого від об'єктів випромінювання за просторовими координатами x, y і довжинами хвиль λ, що математично описується функцією $E(x,y,\lambda)$. Відомо, що енергія випромінювання пропорційна квадрату амплітуди електричного поля, тому є дійсною позитивною величиною. У системах, що створюють зображення, максимальна енергія зображення завжди обмежена, як і розміри зображення обмежені формувальною системою і середовищем. З метою спрощення вважатимемо, що всі значеня енергії

точок зображення відмінні від нуля тільки у визначеній прямокутній області (*x,y*). Тоді відчуття *світлоти*, що виникає в зоровій системі людини, визначається *миттєвою яскравістю* колірного поля, тобто величиною

$$I(x,y)=\int E(x,y,\lambda)S(\lambda)d\lambda \ , \qquad (10.1)$$

де $S(\lambda)$ – спектральна чутливість людського зору; $E(x,y,\lambda)$ – обмежена функцією трьох змінних, яка безперервна в області її визначення.

На рис.10.1, *а* наведено приблизний графік функції $S(\lambda)$. Слід зауважити, що відчуття світлоти нелінійно пов'язане з миттєвою яскравістю. Функціональну залежність зміни відчуття світлоти людиною від зміни яскравості наведено на рис.10.1, *б*.

Для *панхроматичної* системи спостереження функція *I(x,y)* являє собою розподіл яскравості чи будь якої іншої фізичної величини, пов'язаної з яскравістю (оптичною щільностью, відбивною здатністю та ін.). Вона може бути описана виразом (10.1), де $S(\lambda)$– спектральна чутливість датчика, а в *багатозональних* системах спостереження зображення для *n*-ї спектральної зони $S_n(\lambda)$ – спектральна чутливість датчика *n*-го діапазону.

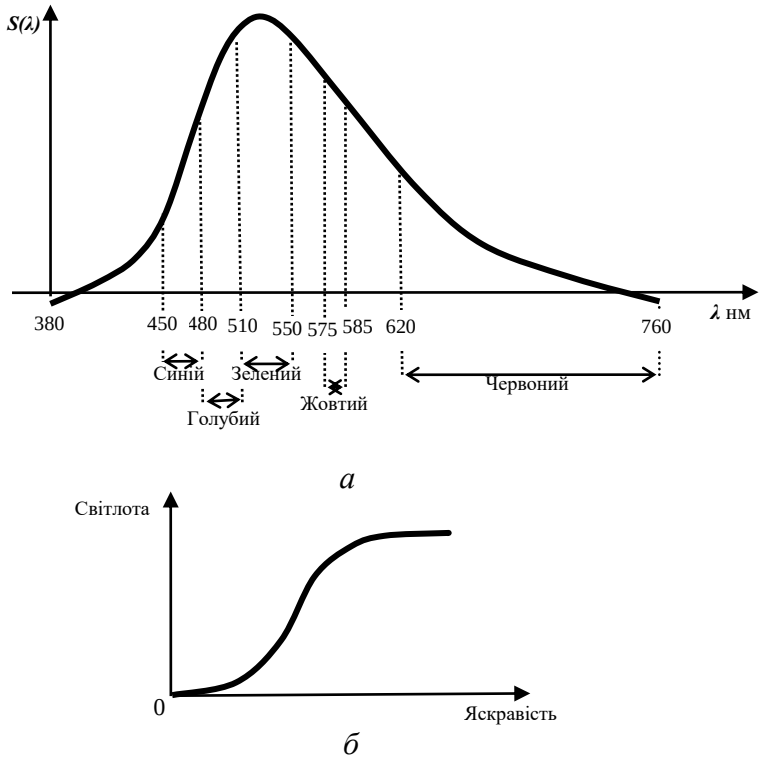


Рис.10.1. Спектральна чутливість людського зору (*а*) та залежність зміни відчуття світлоти людиною від зміни яскравості (*б*)

Колір можна описати набором так званих координат кольору $\{I_R(x,y), I_G(x,y), I_B(x,y)\}$, які є пропорційними інтенсивності червоного, зеленого і синього кольорів, суміш яких дає заданий колір. Можна використовувати інші системи колірних координат, наприклад HSB.

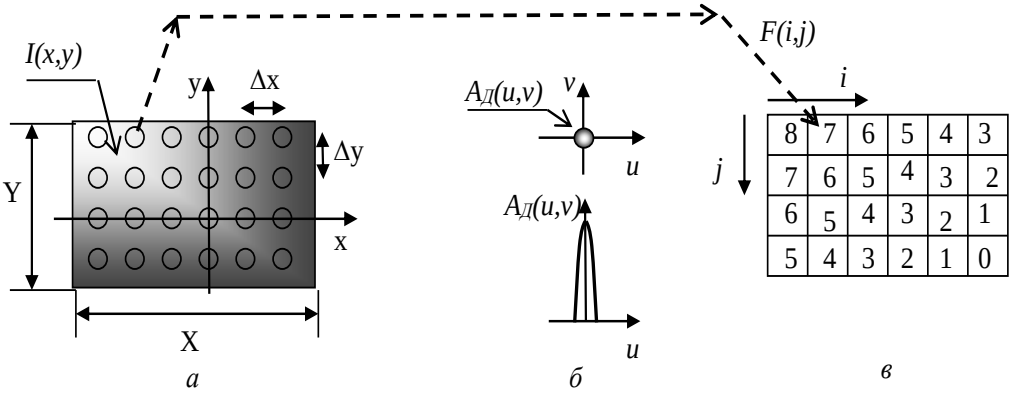
Для обробки безперервних зображень у цифрових процесорах треба насамперед перетворити їх у цифрову форму. Це перетворення являє собою перший етап обробки, що виконується за допомогою спеціальних пристроїв – перетворювачів аналог-кодів. Принциповим питанням є обсяг цифрового подання безперервних сигналів, від чого безпосередньо залежатиме складність цифрової системи обробки сигналів. Перетворення безперервних сигналів у цифрові найчастіше виконують у вигляді послідовності процедур – *дискретизації за простором і квантування за значеннями*. Дискретизація належить до класу лінійних перетворень сигналу, поелементне квантування – до класу поелементних нелінійних перетворень.

Дискретизація сигналу зображення – це заміна безперервного в просторі (x,y) сигналу яскравості $I(x,y)$ на послідовність дискретних значень $F(i,j)$. Найбільш зручним з погляду організації обробки і природним способом дискретизації є подання сигналів у вигляді вибірок їхніх значень (*відліків*) в окремих, регулярно розміщених точках ($x=i\Delta x$, $y=j\Delta y$). Цей спосіб дискретизації називається *растуванням*, а послідовність точок ($x=i\Delta x$, $y=j\Delta y$), у яких беруться відлікі, називається *растром*. Растрування виконується шляхом вимірювання значень сигналу за допомогою датчика, дію якого можна описати як *згортку* сигналу зображення з деякою двовимірною функцією:

$$F(i,j) \approx \iint I(i\Delta x - u, j\Delta y - v) A_Д(u,v) du dv. \qquad (10.2)$$

Функція $A_Д(u,v)$ є характеристикою розподілу чутливості елементарного приймача за полем зору. Її перетин площиною $A_Д(u,v)=0$ задає *миттєве поле зору* елементарного дискретного фотоприймача. Набір одержуваних значень $F(i,j)$ становить дискретне подання сигналу. Сукупність цих значень подається у вигляді матриці $F(i,j)$ відліків значень яскравості зображення в точках вузлів растра (рис.10.2).

Функція $A_Д(u,v)$ називається *апертурою дискретизації*. У пристроях дискретизації зображень апертура дискретизації $A_Д(u,v)$ описує чутливість датчика відеосигналу як функцію координат у рухливій системі координат з початком у точці $(i\Delta x, j\Delta y)$.



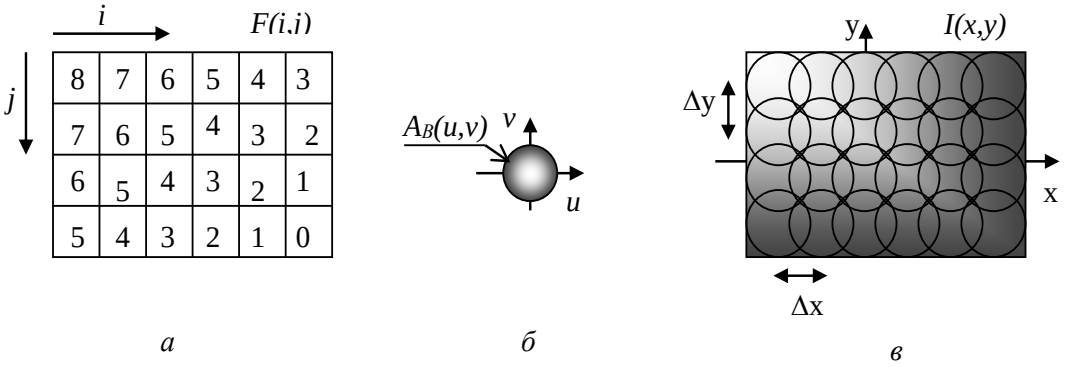
а *б* *в*

Рис. 10.2. Вихідне зображення *I(x,y)* з растром (*а*), апертура дискретизації зображення $A_Д(u,v)$ – (*б*) і результат дискретизації *F(i,j)* (*в*)

Відновлення безперервного сигналу з отриманої послідовності його наближених значень *F(i,j)* виконується шляхом його *інтерполяції* за цими значеннями за допомогою *інтерполювальної функції* $A_B(u,v)$:

$$I(x,y) \approx \sum_i \sum_j F(i,j) A_B(u + i\Delta x, v + j\Delta y). \qquad (10.3)$$

Функція $A_B(u,v)$ називається *апертурою відновлення*. Наприклад, в електронно-променевих пристроях запису і телевізійних дисплеях апертура відновлення – це розподіл яскравості світлової плями люмінофора в рухливій системі координат з початком у точці *(iΔx,jΔy)*. На рис. 10.3 колами показано положення апертури відновлення, де *Δx,Δy* – це крок растра відновлення в горизонтальному і вертикальному напрямках.



а *б* *в*

Рис.10.3. Дискретизоване зображення *F(i,j)* (*а*), апертура відновлення $A_B(u,v)$ (*б*) і відновлене зображення *I(x,y)* (*в*)

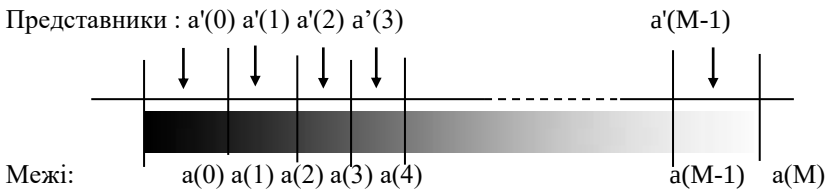
Ідеально дискретизоване зображення може бути відновлене без втрат шляхом інтерполяції на растрі відновлення з кроком *(Δx,Δy)* за допомогою відновлювального променя з розподілом виду *sin(x)/x* і шириною, рівною *Δx/2*, та *Δy/2*. Практично крок дискретизації потрібно вибирати з умови, щоб у найменшу деталь зображення потрапляло не

менш як дві точки растра по кожній координаті, тобто крок відновлення (крок переміщення відновлювального променя) треба обирати як половину ширини променя відновлення в площині відновлюваного зображення. Більшість реальних пристроїв цифрування зображень мають апертуру дискретизації прямокутної форми із згладженими гаусоїдою краями.

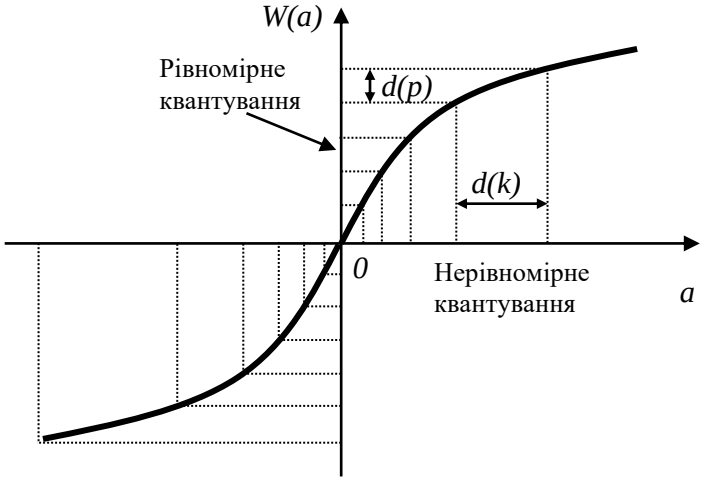
На практиці дискретизацію і відновлення за теоремою відліків здійснити неможливо. По-перше, реальний фотоприймач бере не значення сигналу в точці растра, а середнє значення сигналу за деякою околицею цієї точки. Реальна апертура дискретизації має цілком реальний розмір, що не дає змоги нескінченно зменшувати крок дискретизації без перекриття апертур. Це, у свою чергу, реально збільшує величину мінімально можливого кроку дискретизації, отже, і реальну роздільну здатність апаратури оцифровування до величини порядку ширини апертури дискретизації (розміру миттєвого поля зору одиничного фотоприймача). Звичайно конструкцію фотоприймача і раструвального пристрою обирають так, що найменший крок дискретизації дорівнює приблизно напівширині миттєвого поля зору фотоприймача в площині зображення. Тому у першому наближенні можна вважати, що реальна роздільна здатність пристрою дискретизації дорівнює подвоєному мінімальному кроку дискретизації цього пристрою. По-друге, жоден пристрій, що відновлює (електронно-променева трубка монітора, друкувальний пристрій, фотореєстратор), не в змозі створити відновлювальну апертуру (відновлювальний промінь) з розподілом енергії у просторі виду *sin(x)/x*. Реально апертура відновлення (розподіл енергії в промені, що відновлює) має розподіл у вигляді унімодальної функції (наприклад, гаусоїди). Тобто реальні відновлювальні пристрої забезпечують трохи гірше відновлення за такої растрової сітки відновлення і ширини відновлювального променя. Звичайно це виявляється у вигляді деякої втрати різкості відновленого зображення порівняно з вихідним (тим, що було до цифрування).

Поелементне квантування полягає в тому, що в області значень сигналу обирають відрізок кінцевої довжини, котрий розділяють на інтервали квантування (рис. 10.4, *а*).

Значення, що попадають у кожен інтервал, позначаються одним числом – номером інтервалу. У процесі відновлення сигналу номер заміняється значенням, що є представником цього інтервалу.



а



б

Рис. 10.4. Нерівномірне квантування: *а* – схема розміщення меж і представників інтервалів квантування; *б* – нерівномірне квантування за допомогою нелінійного викривлення

Спосіб поділу на інтервали і значення-представники інтервалів обирають так, щоб задовольнялися вимоги до точності представлення безперервного сигналу цифровим. Слід зазначити, що задача поелементного квантування виникає в цифровій обробці сигналів не тільки під час перетворення безперервних сигналів у цифрові, а й на різних стадіях обчислень у цифрових процесорах під час переходу від однієї форми представлення чисел до іншої. (наприклад, від формату чисел з рухомою комою, до байтового формату для збереження їх в архівних запам’ятовувальних пристроях).

Під час відновлення квантованих зображень можуть спостерігатися деяки викривлення. Якщо рівнів квантування недостатньо, на відтвореному зображенні замість плавних переходів напівтонів будуть спостерігатися стрибкоподібні переходи. Межі ланок сусідніх напівтонів будуть помітними (рис. 10.5). Ці межи називаються хибними контурами.

Одночасно об’єкти на зображенні, які мали слабкий контраст (невелику різницю напівтонів фону і об’єкта) взагалі не будуть спостерігатися на відновленому після квантування зображенні (будуть втрачені).

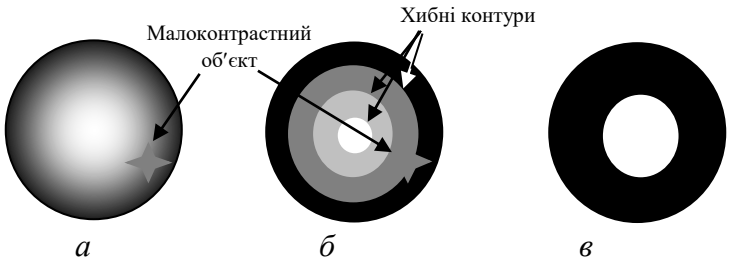


Рис.10.5. Спотворення відновлених після квантування зображень за недостатньої кількості рівнів квантування: *а* – безперервне зображення; *б* – відновлене зображення після квантування на чотири рівні; *в* – відновлене зображення після квантування на два рівні

Рис 10.6 ілюструє передачу напівтонів за різної кількості рівнів квантування. Цифри на рисунку показують застосовану кількість рівнів квантування. Невеликий контраст цифр призводить до втрати їхнього зображення за малої кількості рівнів.

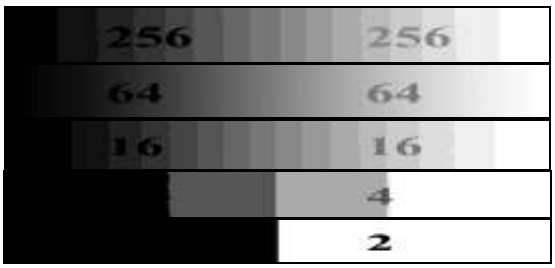


Рис. 10.6. Зображення «Шкала сірого» за різної кількості рівнів квантування

Растрова графіка (англ. raster graphics) є частиною комп’ютерної графіки, яка має діло зі створенням, обробкою та зберіганням растрових зображень. Растрове зображення є масивом кольорових точок (пікселів) (рис. 10.7). Растрові зображення можуть бути збережені в різних форматах, таких як JPEG, PNG, BMP, GIF і TIFF.

Основною одиницею растрової графіки є піксель, який являє собою найменшу точку на зображенні. Кожен піксель має своє значення кольору.

Кольори на растровому зображенні визначаються за допомогою комбінацій червоного, зеленого і синього кольорів (RGB). Кожен піксель

може мати свою власну RGB-комбінацію, що дає змогу створювати барвисті та деталізовані зображення.

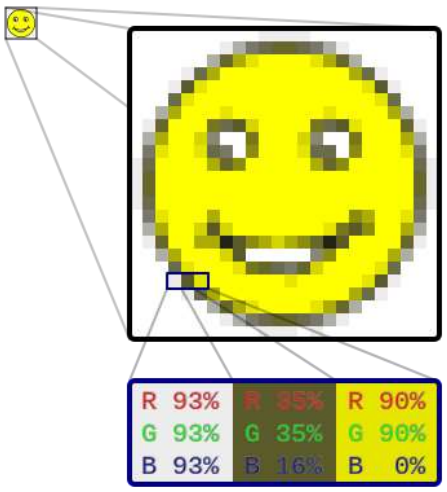


Рис. 10.7. Растрове зображення з пікселів

RGB (червоний, зелений, синій) – адитивна колірна модель, що описує спосіб синтезу кольору чотирма каналами (три канали кольору та один канал яскравості), за якою червоне, зелене та синє світло накладаються разом, змішуючись у різноманітні кольори. Кількість градацій кожного каналу залежить від розрядності бітового значення RGB. Зазвичай використовують 24-бітну модель, у котрій припадає по 8 біт на кожен канал, і тому кількість градацій відтінків кольорів дорівнює 256, що дає змогу закодувати $256^3 = 16\ 777\ 216$ кольорів (рис. 10.8).

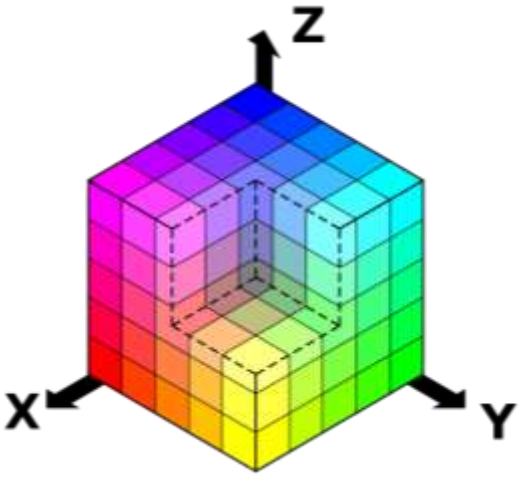


Рис. 10.8. Тривимірне представлення моделі RGB

Найчастіше використовують триколірні моделі: RGB (червоний, зелений, синій) і CMYK (блакитний, пурпуровий, жовтий, чорний), а також HSB (тон, насиченість, яскравість).

Модель RGB будується за принципом сполучення червоного (Red), зеленого (Green) і синього (Blue) кольорів різної яскравості та використовується під час виведення зображення на екран монітора. Ці компоненти називаються *первинними адитивними* і дають змогу одержати більшість кольорів видимого спектра, а в результаті додавання дають білий колір.

Під час обробки кольорових RGB-об’єктів графічна програма надає кожному елементу зображення (пікселю) значення інтенсивності, що може змінюватися в межах від 0 (чорний) до 255 (білий). Причому що більшим є значення байта колірної складової, то яскравіший цей колір.

Після накладення однієї складової на іншу яскравість сумарного кольору також збільшується.

Адитивні кольори використовуються в системах висвітлення, слайд-рекордерах (пристроях запису на фотоплівку), відеосистемах і моніторах. У режимі RGB відбувається сканування зображень, таку саму модель покладено в основу конструкції екрана монітора.

Модель CMYK використовують для підготовки зображень до чотирибарвного друку (із застосуванням блакитного, пурпурового, жовтого і чорного кольорів). Ці барви називають тріадними (чи *субтрактивними*) і в сумі дають чорний колір. Графічний редактор кожному пікселю CMYK-зображення надає значення, що зумовлює процентний уміст тріадних компонентів. Причому найясніші тони характеризуються їхнім низьким вмістом, а найтемніші (тіні) — відповідно більш високими значеннями. Чистому білому кольору відповідними є нульові значення всіх чотирьох складників.

Відтворення кольорів за допомогою моделі RGB залежить від джерела світла, а в основі моделі CMYK лежить здатність друкувальних фарб до *світлопоглинання* (абсорбції). Під час проходження білого світла крізь світлопроникну фарбу частина спектра поглинається. Непоглинений колір відбивається і попадає назад в око людини. Перехід до формату CMYK відбувається на завершальній стадії обробки зображення.

В основі *моделі HSB* лежить сприйняття кольорів людським оком (рис. 10.9). У ній усі кольори визначаються трьома базовими параметрами.

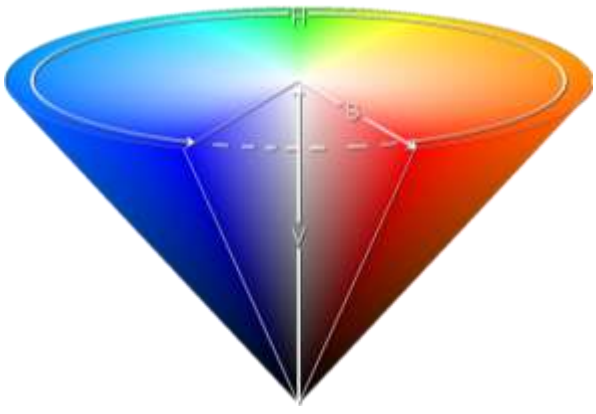


Рис. 10.9. Модель HSB

Колірний тон (hue) – це довжина світлової хвилі, відбитої чи минулої через об'єкт. Він займає визначене положення на стандартному колірному колі, для його опису використовують назву кольору, характеризується величиною кута в діапазоні 0...360 градусів.

Насиченість (saturation), чи хроматичність, – це ступінь чистоти кольору. Насиченість визначає співвідношення сірого кольору і поточного колірного тону; вона виражається у відсотках – від 0% (сірий) до 100% (цілком насичений). На стандартному колірному колі насиченість збільшується від центра до його меж.

Яскравість (brightness) характеризує відносну світлоту кольору. Звичайно вона вимірюється у відсотках у діапазоні від 0% (чорний) до 100% (білий).

Модель HSB зручно застосовувати для створення зображення, а по закінченні роботи зображення можна перетворити в модель RGB чи CMYK.

Зображення у машинному поданні – двовимірний масив NxM, де N – його ширина, M – висота. Під час сканування звичайно використовують розділення від 72 до 2400 dpi (dots per inch – точок на дюйм). Найчастіше – 300 dpi. Якщо взяти аркуш паперу 21/29 см із зображенням і відсканувати його в RGB Truecolor, то нестисненне зображення буде займати ~27300000 байтів, чи 26 Мбайт.

Аерокосмічні зображення можуть бути подані в набагато більших обсягах. Так, смуга огляду датчика МСУ-8 подається 6 000 елементами розділення в трьох частинах діапазону світлових хвиль (два у видимій частини спектра та один у ближній інфрачервоній). Кодування в кожнім піддіапазоні – 11 біт. Вважаючи кадр квадратним, отримаємо, що його зображення буде займати 6000×6000×3×11=1188 млн біт (приблизно 150 Мбайт). Зображення, які можуть бути більш детальними, мають розмір до 1,5 Гбайт.

Растрові файли, відрізняючись один від одного деталями, мають загальну структуру, заголовок, растрові дані та іншу інформацію, зокрема і про колірну палітру. Програмну інформацію, що не розміщується в заголовку, розміщають у кінцівці файла. Якщо застосовується палітра (набір кольорів, наявних у зображенні), то її можна зберегти в заголовку файла, але зручніше розмістити її всередині файла, після заголовка. Крім того, палітру можна зберігати після даних зображення наприкінці файла чи безпосередньо в її складі. Таблиці

рядків розгорнення і таблиці колірної корекції можуть розміщуватися після заголовка і перед даними зображення, і після них: заголовок → палітра → таблиця рядків розгорнення → таблиця колірної корекції → растрові дані → таблиця колірної корекції.

Організація піксельних даних у виді колірних площин ілюструє рис. 10.10.

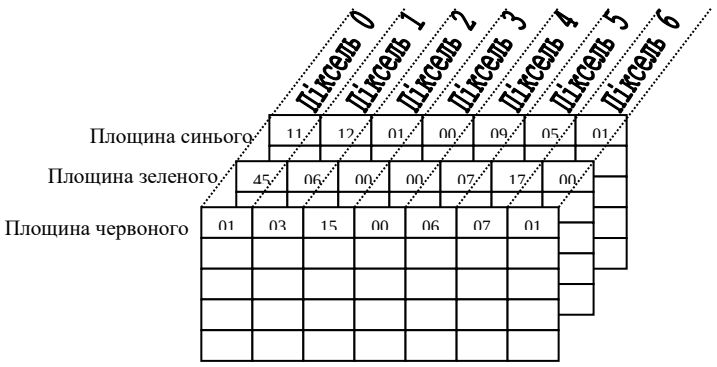


Рис.10.10. Організація піксельних даних у вигляді колірних площин

Векторні зображення представляють списки об’єктів, що мають координати в загальній системі координат. Об’єкти зображують графічними примітивами (крапкою, лінією, прямокутником, колом), які характеризуються стилем зображення, до якого належать різні види заливання, штрихування, товщина ліній, колір, типи символів тощо. У загальній системі координат об’єкти відображаються у вигляді графічних примітивів на площині зображення з використанням стилів.

Крім геометричних і стильових характеристик, об’єкти приймають атрибутивні характеристики, які властиві типу об’єкта й унікальні для цього об’єкта. Для запису атрибутивних характеристик об’єкта використовують бази даних. Це дає змогу виконати аналіз, селекцію об’єкта за різними критеріями – фактично векторне зображення являє собою бази даних, де об’єкти мають ще й свій вид.

Контрольні запитання

1. За якими принципами обирають роздільну здатність сканера для сканування фотознімків?
2. За якими принципами обирають глибину передавання кольору під час сканування з метою збереження та з метою обробки зображень?

3. Яким чином подається колір в системах кодування RGB, HSB, CMYK?

4. Яким чином растрові зображення зберігаються у файлах?

5. Яким чином векторні зображення зберігаються у файлах?

Рекомендована література

1. Форсайт Дэвид А., Понс Жан. Компьютерное зрение. Современный подход.: пер. с англ. – М.: Издательский дом «Вильямс», 2004. – 928 с.

2. Шапиро Л. Компьютерное зрение. Л. Шапиро, Дж. Стокман / Пер. с англ. – М.: БИНОМ. Лаборатория знаний, 2006. – 752 с.

3. Ямпольський Л.С. Гнучкі комп'ютеризовані системи: проєктування, моделювання і управління: підручник / Л.С. Ямпольський, П.П. Мельничук, Б.Б.Самотокін, М.М.Поліщук, М.М. Ткач, К.Б. Остапченко, О.І. Лісовиченко. – Житомир: ЖДТУ. – 2005. – 680 с.

4. Siciliano B. Handbook of robotics / B. Siciliano, O. Khabib, eds. – 2008. – Springer-Verlag Berlin Heidelberg. – 1611 p.

5. Орловський Б.В. Інформаційні системи та пристрої в робототехніці та мехатроніці: навч. посібн. / Б.В. Орловський. – Київ: КНУТД, 2018.–416 с.

6. Діктерук М.Г. Основи автоматизаціяї будівельних машин: навч. посіб. / М. Г. Діктерук, О. В. Човнюк. – Київ : КНУБА, 2006. – 232 с.

7. Пелевін Л.Є. Гідравліка та приводи механотронних систем: підручник / Л.Є. Пелевін, Д.О. Міщук. – Київ: КНУБА, 2016. – Ч. 2. – 136 с.

8. Siciliano B. Robotics / B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo. – 2009. – Springer-Verlag London Limited. – 632 p.

Список літератури

1. *Артюх О.М.* Основи мехатроніки : навч. посіб. / О.М. Артюх, О.В. Дударенко, В.В. Кузьмін та ін. – Запоріжжя: НУ «Запорізька політехніка», 2021. – 372 с.

2. *Архангельский В.И.* Нейронные сети в системах автоматизации / В.И. Архангельский, И.Н. Богаенко и др. – Київ: Техніка, 1999. – 364 с.

3. *Введення* в мехатроніку: навч. посіб. / А.І. Грабченко, В.Б. Клепіков, В.Л. Доброскок та ін. – Харків: НТУ «ХПІ», 2014. – 264 с.

4. *Діктерук М. Г.* Основи автоматизації будівельних машин: навч. посіб. / М. Г. Діктерук, О. В. Човнюк. – Київ: КНУБА, 2006. – 232 с.

5. *Клетте Р.* Комп'ютерний зір. Теорія і алгоритми / Р. Клетте, пер. с англ. – АМК: Springer, – 2019. – 506 с.

6. *Ловейкін В.С.* Мехатроніка: підручник / В.С. Ловейкін, Ю.О. Ромасевич, В.В. Крушельницький. – Київ: КОМПРІНТ, 2020. – 404 с.

7. *Міщук Д.* Розробка концепції системи керування роботом для штукатурних робіт на основі нейронної мережі / Д. Міщук, А. Бойченко // Гірничі, будівельні, дорожні та меліоративні машини. – 2019. – № 93. – С. 46-60. – https://doi.org/10.32347/ gbdmm2019.93.0501.

8. *Міщук Є.* Системи промислової автоматизації на основі IoT. / Є. Міщук, Д. Міщук // Гірничі, будівельні, дорожні та меліоративні машини. – № 96. – 2022. – С. 42–50. – https://doi.org/10.32347/ gbdmm2020.96.0501.

9. *Момот М.В.* Мобільні роботи на базі Arduino / М.В. Момот. – БХВ. – 2017. – 336 с.

10. *Поліщук М.М.* Робототехнічні системи: проєктування і моделювання [Електронний ресурс]: навч. посіб. / М.М. Поліщук, М.М. Ткач. – Київ: КПІ ім. Ігоря Сікорського, 2021. – 112 с.

11. *Пелевін Л.Є.* Гідравліка та приводи механотронних систем: підручник / Л.Є. Пелевін, Д.О. Міщук. – Київ: КНУБА, 2016. – Ч. 1. – 192 с.

12. *Пелевін Л.Є.* Гідравліка та приводи механотронних систем: підручник / Л. Є. Пелевін, Д. О. Міщук. – Київ: КНУБА, 2016. – Ч. 2. – 136 с.

13. *Михайлов Є.П.* Галузеве машинобудування: навч. посіб. / Є.П. Михайлов, В.М. Лінгур. – Одеса: ОНПУ, 2019. – 233 с.

14. *Васильєв О.* Програмування мовою Python / О. Васильєв. – Навчальна книга – Богдан, 2019. – 504 с.

15. *Орловський Б.В.* Інформаційні системи та пристрої в робототехніці та мехатроніці: навч. посіб. / Б.В. Орловський. – Київ: КНУТД, 2018. – 416 с.

16. *Цвіркун Л.І.* Робототехніка та мехатроніка: навч. посіб. для студ. вищ. навч. закл. / Л.І. Цвіркун. – Д.: вид-во ДНГУ, 2010.

17. *Эванс Эрік*. Предметно-орієнтоване проектування. Структуризація складних програмних систем / Э. Эванс. – М.: Вільямс, 2015. – 448 с.

18. *Форсайт Дэвид А*. Компьютерное зрение. Современный подход / Форсайт Дэвид А., Понс Жан / пер. с англ. – М.: Вильямс, 2004. – 928 с.

19. *Шапиро Л.* Компьютерное зрение / Л. Шапиро, Дж. Стокман; пер. с англ. – М.: БИНОМ. Лаборатория знаний, 2006. – 752 с.

20. *Ямпольський Л.С.* Гнучкі комп'ютеризовані системи: проєктування, моделювання і управління: підручник / Л.С. Ямпольський, П.П. Мельничук, Б.Б.Самотокін. – Житомир: ЖДТУ, – 2005. – 680 с.

21. *Bruno Siciliano*. Handbook of robotics / B. Siciliano, O. Khabib, eds., – 2008. - Springer-Verlag Berlin Heidelberg, – 1611 p.

22. *Everett H.R.* Sensors for Mobile Robots, Theory and Applications. – New York, Natick M.A, A.K. Peters Ltd. – 1995. – 390 p.

23. *Roland Siegwart*. Introduction to Autonomous Mobile Robots / R. Siegwart, R. Illah, – Massachusetts Institute of Technology. – 2004. – 321 p.

24. *Pedrosa E., L. Reis, C.M.D Silva, H.S Ferreira*. Autonomous Navigation with Simultaneous Localization and Mapping in/outdoor. – 2020. – 75 p.

25. *Gmapping* [Електронний ресурс] URL: http://wiki.ros.org/gmapping, – Мова англ. Дата звернення: 14.02.2024 р.

26. *Google Cartographer ROS* [Електронний ресурс] URL: https://google-cartographer-ros.readthedocs.io/en/latest/#, – Мова англ. Дата звернення: 04.11.2023 р.

27. RTAB-Map, Real-Time Appearance-Based Mapping [Електронний ресурс] URL: http://introlab.github.io/rtabmap/, – Мова англ. Дата звернення: 22.06.2023 р.

28. Adaptive Monte Carlo localization [Електронний ресурс] URL: http://wiki.ros.org/amcl, – Мова англ. Дата звернення: 03.08.2023 р.

29. *Siciliano B.* Robotics / B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo. – 2009. – Springer-Verlag London Limited. – 632 p.

Додаток

Компоненти макроконтролера AVR

Арифметико-логічний пристрій (АЛП), що виконує всі обчислення, безпосередньо під'єднаний до 32 робочих регістрів, об'єднаних у реєстровий файл. Завдяки цьому АЛП виконує одну операцію (читання вмісту регістрів, виконання операції і запису результату назад у реєстровий файл) за один машинний цикл.

I/O – порти вводу/виводу AVR мають від трьох до 53 незалежних ліній «вхід/вихід». Кожна лінія порту може бути запрограмована на вхід або вихід. Номінальне навантаження на одну лінію порту становить 20 мА, а максимальне і короткочасне значення – 40 мА, що дає змогу, наприклад, безпосередньо під'єднувати до мікроконтролера світлодіоди і біполярні транзистори. Загальне струмове навантаження на всі лінії одного порту не повинне перевищувати 80 мА (всі значення наведено для напруги живлення 5 В).

Архітектурна особливість побудови портів вводу/виводу у AVR полягає в тому, що для кожного фізичного виведення (піну) є 3 біти контролю/управління, а не два, як у поширених 8-розрядних мікроконтролерів (Intel, Microchip, Motorola тощо). Це дає змогу уникнути потреби мати копію вмісту порту в пам'яті для безпеки та підвищує швидкість роботи мікроконтролера під час взаємодії із зовнішніми пристроями, особливо в умовах зовнішніх електричних перешкод.

Interrupts – система переривань, одна з найважливіших частин мікроконтролера. Усі мікроконтролери AVR мають багаторівневу систему переривань. Переривання припиняє нормальний перебіг програми для виконання пріоритетного завдання, яке визначається внутрішньою або зовнішньою подією. Для кожної такої події розробляють окрему програму, яку називають підпрограмою обробки запиту на переривання і яка розміщується в пам'яті програм. У разі події, що викликає переривання, мікроконтролер зберігає вміст лічильника команд, перериває виконання центральним процесором поточної програми і переходить до виконання підпрограми обробки переривання. Після виконання підпрограми переривання здійснюється відновлення попередньо збереженого лічильника команд і процесор повертається до виконання перерваної програми. Для кожної події можна встановити пріоритет. Поняття «пріоритет» означає, що виконувана підпрограма переривання може бути перервана іншою подією тільки за умови, що вона має більш високий пріоритет, ніж поточна.

Timer/Counters – таймери/лічильники. Мікроконтролери AVR мають у своєму складі від одного до чотирьох таймерів/лічильників з розрядністю 8 або 16 біт, які можуть працювати як таймери від внутрішнього джерела тактової частоти, і як лічильники зовнішніх подій. Їх можна використовувати для точного формування часових інтервалів, підрахунку імпульсів на виводах мікроконтролера, формування послідовності імпульсів, тактування приймача послідовного каналу зв'язку. В режимі ШІМ (PWM) таймер/лічильник може бути широтно-імпульсним модулятором і використовується для генерування сигналу з програмованими частотою і добротністю.

WDT – сторожовий таймер (WatchDog Timer) призначений для запобігання катастрофічним наслідкам від випадкових збоїв програми. Він має власний RC-генератор, який працює на частоті 1 МГц. Як і для основного внутрішнього RC-генератора, значення 1 МГц є наближеним і залежить насамперед від величини напруги живлення мікроконтролера і від температури. Ідея використання сторожового таймера полягає в регулярному його скиданні під керуванням програми або зовнішнього впливу до того, як закінчиться час його витримки і не станеться скидання процесу. Якщо програма працює нормально, то команда скиду сторожового таймера повинна регулярно виконуватися, захищаючи поцесор від скидання. Якщо ж мікропроцесор випадково вийшов за межі програми (наприклад, від сильної перешкоди по ланцюгу живлення) або зациклився на будь-якій ділянці програми, команда скидання сторожового таймера швидше за все не буде виконана, а процес закінчить процесор.

AC – аналоговий компаратор, який порівнює напруги двох виводів микроконтролера. Результатом порівняння буде логічне значення, яке можна прочитати з програми.

A/D Converter – аналого-цифровий перетворювач (АЦП) застосовують для отримання числового значення напруги заданої розрядності (точності) з аналогової величини поданої на його вхід.

UART або *USART* – універсальний асинхронний або універсальний синхронно/асинхронний приймач (Universal Synchronous/Asynchronous Receiver and Transmitter). Послідовний інтерфейс для організації інформаційного каналу обміну мікроконтролера із зовнішнім світом. Здатен працювати в дуплексному режимі (одночасне передавання та приймання даних). Він підтримує протокол стандарту RS-232, що забезпечує можливість зв'язку з персональним комп'ютером. Для стикування мікроконтролера та комп'ютера обов'язково знадобиться

схема поєднання рівнів сигналів (для цього створено спеціальні мікросхеми, наприклад MAX232).

SPI – послідовний периферійний трипровідний інтерфейс SPI (Serial Peripheral Interface), призначений для організації обміну даними між двома пристроями. За його допомогою може відбуватися обмін даними між мікроконтролером та різними пристроями, такими як цифрові потенціометри, ЦАП/АЦП, FLASH-память та ін. За допомогою цього інтерфейсу зручно виконувати обмін даними між декількома мікроконтролерами AVR. Крім того, через інтерфейс SPI може відбуватись програмування мікроконтролера.

TWI – двопровідний послідовний інтерфейс (Two-wire Serial Interface) є повним аналогом базової версії інтерфейсу I2C (двопровідна двонаправлена шина) фірми Philips. Цей інтерфейс дає змогу об'єднати до 128 різних пристроїв за допомогою двонаправленої шини, що складається з лінії тактового сигналу (SCL) і лінії даних (SDA).

JTAG – чотирипровідний інтерфейс (Joint Test Action Group) стандарту IEEE Std 1149.1-1990, який використовується для тестування друкованих плат, внутрішньосхемного налагодження, програмування мікроконтролерів.

Тактовий генератор виробляє імпульси для синхронізації всіх вузлів мікроконтролера. Внутрішній тактовий генератор AVR може запускатися від кількох джерел опорної частоти (зовнішній генератор, зовнішній кварцовий резонатор, внутрішній або зовнішній RC-ланцюжок). Мінімальна допустима частота нічим не обмежена (аж до покрокового режиму). Максимальна робоча частота визначається конкретним типом мікроконтролера і вказується у його характеристиках, хоча практично будь-який AVR-мікроконтролер із заявленою робочою частотою, наприклад, 10 МГц за кімнатної температури легко може бути «розігнаний» до 12 МГц і вище.

RTC – система реального часу, яка реалізована в усіх мікроконтролерах Mega. Таймер/лічильник RTC має окремий подільник, який може бути програмним способом під'єднаний або до джерела основної тактової частоти, або до додаткового джерела асинхронної опорної частоти (кварцовий резонатор або зовнішній синхросигнал). З цією метою зарезервовано два виводи мікросхеми. Внутрішній осцилятор оптимізовано для роботи із зовнішнім «вартовим» кварцовим резонатором на частоті 32,768 кГц.

Навчальне видання

МІЩУК Дмитро Олександрович,
РАШКІВЬКИЙ Володимир Павлович

ПРОГРАМУВАННЯ РОБОТОТЕХНІЧНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

Конспект лекцій

Редагування та коректура *Г.В. Кобриної*
Комп'ютерне верстання *Т.В. Кукарєвої*

Підписано до другу 12.02.2025. Формат 60 × 84 1/ 16
Ум. друк. арк. 13,02. Обл.-вид. арк. 14,0.
Електронний документ. Вид № 24/І-24.

Видавець і виготовлювач
Київський національний університет будівництва і архітектури

Проспект Повітряних Сил, 31, Київ, Україна, 03037

Свідоцтво про внесення до державного реєстру суб'єктів
видавничої справи ДК № 808 від 13.02.2002 р.