Міністерство освіти і науки України

Національний університет водного господарства та природокористування

Кафедра автоматизації, електротехнічних та

комп’ютерно-інтегрованих технологій

04-03-445М

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт

з навчальної дисципліни

«Програмування в робототехніці»

для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» спеціальності 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» денної та заочної форм навчання

Частина 1

Рекомендовано науково-методичною радою з якості ННІЕАВГ

Протокол № 6 від 28 січня 2025 р.

Рівне – 2025

Методичні вказівки до лабораторних робіт з навчальної дисципліни «Програмування в робототехніці» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» спеціальності 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» денної та заочної форм навчання. Частина 1 [Електронне видання] / Реут Д. Т. – Рівне : НУВГП, 2025. – 42 с.

Укладач: Реут Д. Т., к.т.н., доцент кафедри автоматизації, електротехнічних та комп’ютерно-інтегрованих технологій.

Відповідальний за випуск: Древецький В. В., д.т.н., професор, завідувач кафедри автоматизації, електротехнічних та комп’ютерно-інтегрованих технологій.

Керівник групи забезпечення спеціальності 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка»: Христюк А. О., к.т.н., доцент, доцент кафедри автоматизації, електротехнічних та комп’ютерно-інтегрованих технологій.

Попередня версія методичних вказівок 04-03-363М

Зміст

Вступ

Лабораторні роботи з навчальної дисципліни «Програмування в робототехніці» дають змогу на практиці вивчити підходи й прийоми програмування в галузі робототехніки, зокрема програмованих квадрокоптерів DJI Robomaster Tello Talent, наземних роботів на базі мікропроцесорних плат Raspberry Pi 4B, Arduino Uno в якості контролерів, навчитись створювати програми керування робототехнікою на основі Robot Operating System. Передумовою вивчення дисципліни є наявність грунтовних знань з дисциплін «Мікропроцесорна техніка та програмування мікроконтролерів», «Мехатроніка та роботизовані комплекси».

У лабораторних роботах розглянуто:

– встановлення Robot Operating System і взаємодію з платою Arduino Uno;

– візуалізацію даних, отриманих Robot Operating System;

– керування сервоприводами за допомогою бібліотеки WiringPi;

– відстеження об’єктів на відео засобами бібліотеки комп’ютерного зору OpenCV;

– керування квадрокоптером DJI Robomaster TT та наземними платформами на базі Raspberry Pi 4B та Arduino Uno.

Лабораторна робота №1. Встановлення Robot Operating System. Обмін даними з Arduino.

Мета роботи: навчитись встановлювати Robot Operating System, налаштовувати підтримку передачі даних в ROS в Arduino IDE, записувати й отримувати повідомлення з топіків ROS.

Теоретичні відомості

Robot Operating System - це набір бібліотек та інструментів (фреймворк) із відкритим кодом, які допомагають створювати програми для роботів, від драйверів до найсучасніших алгоритмів і з потужними інструментами розробника.

ROS дозволяє використовувати вже готові реалізації різноманітних алгоритмів, наприклад, для створення карти і визначення місцезнаходження робота, а також надає узагальнений інтерфейс для взаємодії різних програм і компонентів між собою.

Особливістю системи ROS є те, що вона працює більше як операційна система, ніж як додаток. У всіх інтерфейсах моделювання та управління використовується архітектура клієнт-сервер;

У ROS використовуються вузли (nodes), які приєднуються до майстра (roscore), та є відповідно клієнтами та сервером.

ROS є надбудовою над операційною системою, яка дозволяє легко та просто розробляти системи управління роботам. Власне, ROS – це набір із різних відомих бібліотек:

OpenCV – бібліотека, що містить алгоритми комп'ютерного зору та обробки зображень з відкритим вихідним кодом. Підтримує C, C++, Python, Java та інші мови;

- PCL (Point Cloud Library) – бібліотека з відкритим вихідним кодом для обробки 2D/3D зображень та хмар точок;
- OGRE (Object-Oriented Graphics Rendering Engine) – це об'єктно-орієнтований графічний рушій, написаний мовою C, з відкритим вихідним кодом, призначений для спрощення та візуалізації моделювання поведінки робота;
- OROCOS (Open Robot Control Software) – бібліотека для управління роботами, наприклад, розрахунку кінематики;
- CARMEN (Carnegie Mellon Robot Navigation Toolkit) – бібліотека для управління мобільними роботами. Вона призначена для виконання базових операцій таких як: сенсорне управління, обхід перешкод, побудова шляху та створення карт.

На рівні файлової системи основним блоком для організації

програмного забезпечення в ROS є пакет. Пакет ROS може містити вихідні коди та виконувані файли вузлів, бібліотеки, опис повідомлень та сервісів, бази даних, файли конфігурації та інші ресурси, які логічно організувати разом. Кожен пакет має містити файл маніфесту, який надає метадані про пакет, включаючи відомості про ліцензії та залежності, а також прапори компілятора і так далі. Мета такого структурування - підвищення можливості повторного використання.

Вузли (Nodes) це процеси, які виконують обчислення. ROS представляє з себе модульну систему, система управління роботом зазвичай включає в себе безліч вузлів, вкладених у виконання певних функцій. Кожен вузол керує яким-небудь процесом, наприклад, один вузол керує лазерним далекоміром, один вузол керує колісними двигунами, один вузол виконує локалізацію, один вузол виконує планування шляху, один вузол забезпечує графічне представлення системи, тощо.

Вузол ROS написаний з використання клієнтських бібліотек ROS, таких як roscpp або rospy.

ROS Майстер (Master) забезпечує зв'язок між вузлами. Без майстра вузли не зможуть знайти одне одного, обмінюватися повідомленнями чи викликати сервіси.

Повідомлення (Messages). Вузли взаємодіють один з одним шляхом передачі повідомлень. Повідомлення – це структура даних, що складається з полів із зазначенням типу поля. Підтримуються стандартні типи (цілі числа, з плаваючою точкою, логічні тощо), а також масиви. Повідомлення можуть включати довільно вкладені структури та масиви (подібно до синтаксису мови C).

Теми (Topics). Повідомлення надсилаються через транспортну систему з семантикою видавець/підписник. Вузол посилає повідомлення, опублікувавши його в тому чи іншому топіку. Фактично це ім'я, яке використовується для ідентифікації вмісту повідомлення. Вузол, який зацікавлений у певного роду даних може підключитись до потрібної теми/топіка. Може бути більше одного видавця і підписника, один вузол може писати та/або підписатися на кілька тем. Можна представити тему як строго типізовану шину повідомлень.

Порядок виконання роботи

1. При роботі в ОС, відмінній від Ubuntu 20.04, встановити Docker (пакет docker.io).

2. Завантажити образ зі встановленою ROS: *sudo docker pull*

ros:noetic . Якщо у вас Ubuntu 20.04, ви можете також додати репозиторій відповідно до інструкцій http://wiki.ros.org/noetic/Installation/Ubuntu та встановити безпосередньо в систему командою *sudo apt install ros-noetic-desktop* .

3. Запустити контейнер на основі завантаженого образу командою *sudo docker run -it ros:noetic*

4. Для зручності роботи довстановіть пакети bash-completion, mc, nano, оновивши список пакетів перед цим: *apt update && apt install bash-completion mc nano*

5. Виконайте *source /opt/ros/noetic/setup.bash* . Ця команда змінює налаштування середовища виконання та має бути виконана перед викликом компонентів ROS (на зразок roscd, roslaunch тощо).

6. Для передачі даних між ROS і Arduino будемо використовувати rosserial - програму, що отримуватиме дані від Arduino через послідовний інтерфейс і пересилатиме їх в топіки ROS. Спочатку встановимо нижній рівень для Arduino: *apt install ros-noetic-rosserial-arduino* , далі - верхній рівень із самою програмою *apt install ros-noetic-rosserial .*

7. Встановити Arduino з офіційного сайту: встановити wget або curl за допомогою *apt install wget*, завантажити файл з сайту командою *wget https://downloads.arduino.cc/arduino-1.8.19-linux64.tar.xz* , розпакувати архів *tar xvf arduino-1.8.19-linux64.tar.xz* . Підключити плату Arduino Uno.

8. Зберегти зміни в контейнері у новий образ командою *sudo docker commit id_контейнера ros:ваше_прізвище* . id_контейнера можна побачити в запрошенні терміналу, напр. root@**0214702545b3**:/#

9. Вийти з контейнера командою *exit*. Перезайти з підтримкою графічного інтерфейсу командою *sudo docker run -e DISPLAY=unix$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix --device=/dev/ttyUSB0 -it ros:ваше_прізвище* . Запустити Arduino IDE. У випадку помилки авторизації на хост-системі виконати команду *xhost +local:* В налаштуваннях запущеного Arduino IDE дізнатись шлях до каталогу зі скетчами. В терміналі виконати *cd каталог_зі_скетчами/libraries* . Зібрати бібліотеки для Arduino командою *rosrun rosserial_arduino make_libraries.py* . (крапка – частина команди). Запустити знову Arduino IDE та в прикладах знайти групу прикладів для ROS, в ній приклад pubsub.

10. Скомпілювати та завантажити програму в Arduino Uno.

11. Запустити roscore.

12. Запустити rosserial командою *rosrun rosserial_python serial_node.py /dev/ttyUSB0* . Перевірити відповідність назв топіків.

13. Паралельно запустити rostopic і пересвідчитись в отриманні даних командою *rostopic echo /назва_топіка* . Для одночасної роботи з кількома терміналами можна використати програму screen.

14. За допомогою rostopic надіслати дані в Arduino, використовуючи команду *rostopic pub /назва_топіка std_msgs/String повідомлення*.

15. Зберегти змінений контейнер в образ *sudo docker commit id_контейнера ros:ваше_прізвище*

16. Оформити звіт, що повинен містити титульну сторінку, тему, мету роботи, порядок виконання, текст програми pubsub, скріншоти виконання rosrun, rostopic echo, rostopic pub, висновок.

Контрольні запитання

1. Що собою являє ROS?
2. Яка модель обміну даними в ROS?
3. Яка програма дозволяє пересилати повідомлення з послідовного інтерфейсу від мікроконтролерних плат в топіки ROS?
4. Яка бібліотека використовується в Arduino для надсилання повідомлень, сумісних з rosserial?
5. Як запустити docker з підтримкою графічної підсистеми X11?
6. Як надати доступ docker-контейнеру до послідовного порту?
7. Яка програма запускається як master-процес в ROS?
8. Чим зумовлене використання docker при роботі з ROS1?

Лабораторна робота №2. Обмін даними через Bluetooth Low Energy з Robot Operating System.

Мета роботи: навчитись використовувати модуль Bluetooth Low Energy для зв’язку контролера робота з вузлом, що виконує Robot Operating System.

Теоретичні відомості

Bluetooth та WiFi - поширені технології для бездротового зв'язку в приміщенні на невеликі відстані, що можуть використовуватись для керування мобільним роботом.

Класична версія Bluetooth надає можливість організації послідовних (COM) портів для обміну даними SPP (Serial Port Profile) (наприклад, використовуючи програму rfcomm), проте Bluetooth Low Energy не містить даного профіля. UART-подібний інтерфейс може бути реалізований періодичним обміном повідомленнями, наприклад,

програмою ble-serial. Відповідно на хості (ПК) дані будуть виглядати як такі, що надходять з послідовного порта, але не апаратного, а створеного програмно. Представлення даних з Bluetooth-модуля у послідовному порту надає можливість реалізувати прозорий канал передачі через Bluetooth і програми, що очікують даних з послідовного порта /dev/ttyUSB0, можуть бути переналаштовані на роботу з портом /dev/ttyNET0 без необхідності переписувати їх під використання саме Bluetooth Low Energy. Однак /dev/ttyNET0 буде створюватись програмно на рівні застосунків, а не драйвером відповідного обладнання, що працює в просторі ядра Linux.

Порядок виконання роботи

1. Запустити Arduino IDE.

2. Завантажити в Arduino Uno прошивку, яка надсилатиме періодично дані (напр., AnalogInOutSerial) через послідовний інтерфейс. Вони потрібні будуть для перевірки роботи Bluetooth-з'єднання.

3. Підключити модуль HM-10 до контактів 5V і GND та контактів RX, TX. Врахувати, що передані платою Arduino дані з контакту TX повинні надходити на приймаючий контакт RX модуля і навпаки. Подати живлення на отриману схему.

4. Встановити ble-serial, який надасть можливість обмінюватись даними з bluetooth-модулем HM-10 як через створений програмно послідовний порт /dev/ttyBLE, так і через TCP, виступаючи TCP-сервером: *pip3 install ble-serial* .

5. Ознайомитись з документацією, доступною на сторінці https://github.com/Jakeler/ble-serial. Оскільки порт /dev/ttyBLE створюється користувацькою програмою, а не ядром Linux, то тунелювання його всередину контейнера docker не дасть можливості передавти дані. Використаємо другу можливість - передачу даних від bluetooth-пристрою через TCP-підключення, а саме організуємо TCP-тунель між послідовними інтерфейсами.

6. Запустити сканування доступних Bluetooth-пристроїв командою */home/student/.local/bin/ble-scan* . Знайти MAC-адресу модуля HM-10 (запис на зразок 30:E2:83:8A:DA:CF (RSSI=-69): HMSoft).

7. Дізнатись IP-адресу комп'ютера, до якого підключатиметься TCP-клієнт, командою *ip a* .

8. Запустити сервер командою виду */home/student/.local/bin/ble-serial -d 30:E2:83:8A:DA:CF --expose-tcp-port 4002 -v --expose-tcp-host 10.1.130.223 .* Переглянути вивід команди і пересвідчитись у отриманні

даних від Bluetooth-модуля.

9. Запустити контейнер на основі образу, збереженого на попередній лабораторній роботі, командою *sudo docker run -e DISPLAY=unix$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix --device=/dev/ttyUSB0 -it ros:ваше_прізвище* . У випадку помилки авторизації на хост-системі виконати команду *xhost +local:*

10. Запустити головний процес ROS командою *roscore&* .

11. Використати socat для передачі даних з TCP в послідовний порт, з яким “вміє” працювати rosserial. Встановити socat командою *apt install socat*. Запустити у фоні тунель між TCP й послідовним портом /dev/ttyNET0 командою *socat pty,link=/dev/ttyNET0 tcp:10.1.130.239:4002&* .

12. В файлі *шлях_до_теки_зі_скетчами_Arduino/libraries/ros_lib/ArduinoHardware.h* встановити значення швидкості передачі даних 9600 бод (вища швидкість 57600 бод призводить до збільшення частки пошкоджених пакетів), знайшовши відповідний рядок і замінивши на

baud_ = 9600

13. Використовуючи приклад pubsub з бібліотеки ros_lib та приклади для бібліотеки Servo, написати програму, що підписуватиметься на топік servo з повідомленнями типу std_msgs/Int16, і при отриманні кожного повідомлення отримане число використовуватиме як завдання кута повороту для сервопривода SG-90, підключеного до цифрового вводу-виводу 9. В топік chatter замість "hello, world!" програма повинна писати поточне положення сервопривода, отримане методом Servo.read().

14. Після завантаження програми в Arduino Uno запустити rosserial командою

rosrun rosserial_python serial_node.py _port:=/dev/ttyNET0 _baud:=9600

15. Перевірити роботу програми, використавши команду rostopic pub для надсилання нового кута повороту й rostopic echo для отримання поточного кута повороту з топіка chatter.

16. Зберегти змінений контейнер в образ *sudo docker commit id_контейнера ros:ваше_прізвище*

17. Оформити звіт, що повинен містити титульну сторінку, тему, мету роботи, порядок виконання, скріншоти виконання ble-serial в ролі TCP-сервера, rostopic echo, rostopic pub, текст програми для керування

сервоприводом через Bluetooth, висновок.

Контрольні запитання

1. Який пакет Python дозволяє використовувати послідовний інтерфейс в модулях Bluetooth LE?

2. Який профіль Bluetooth дозволяє передавати дані з послідовного інтерфейсу плати Arduino?

3. Яка програма дозволяє отримати перелік BLE-пристроїв навколо?

Лабораторна робота №3. Передача даних про положення в Robot Operating System і їх візуалізація. Керування гусеничною латформою з Arduino Uno

Мета роботи: навчитись отримувати в середовище Robot Operating System дані IMU від акселерометра-гіроскопа MPU6050 і візуалізувати їх.

Теоретичні відомості

Інерційний вимірювальний пристрій (Inertial Measurement Unit, IMU) — це електронний пристрій, який вимірює кутову швидкість, а іноді й орієнтацію тіла, використовуючи комбінацію акселерометрів, гіроскопів і іноді магнітометрів.

На основі даних IMU формуються повідомлення типу sensor_msgs/Imu, які використовуються для отримання geometry_msgs/PoseStamped, що містять дані про час, координати й положення (орієнтацію). Останні можуть використовуватись програмою візуалізації Rviz для відображення об'єкта на екрані. Дані IMU в системі ROS можуть бути використані при визначенні пройденого шляху/траєкторії робота й в парі з сенсорами (лазерним сканером або камерами) надавати можливість застосувати SLAM. Також дані IMU можуть використовуватися для визначення фактичної орієнтації захвата маніпулятора й маніпульованого об'єкта.

Порядок виконання роботи

1. Запустити контейнер на основі образу, збереженого на попередній лабораторній роботі, командою *sudo docker run -e DISPLAY=unix$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix --device=/dev/ttyUSB0 -it ros:ваше_прізвище* . У випадку помилки авторизації на хост-системі виконати команду *xhost +local:*

2. Налаштувати змінні оточення командою source

/opt/ros/noetic/setup.bash

3. Встановити пакет imu-tools, необхідний для візуалізації даних IMU, командою *apt install ros-noetic-imu-tools*

4. Створити каталог /root/catkin_ws/src

5. Перейти в створений каталог командою *cd ~/catkin_ws/src/* .

6. Клонувати репозиторій mpu6050_imu_ros в поточний каталог командою *git clone https://github.com/soarbear/mpu6050_imu_ros.git* . Він містить вузол, що зчитує дані з Arduino за допомогою rosserial, та вузол конвертування отриманих даних у формат повідомлень, що може візуалізуватись Rviz'ом. За відсутності в системі Git-клієнта встановити його командою *apt install git* .

7. Перейти на рівень вгору *cd ~/catkin_ws*

8. Скомпілювати пакет командою *catkin_make* .

9. Виконати *source devel/setup.bash*, щоб мати можливість запустити програми з пакету без його встановлення.

10. Запустити Arduino IDE.

11. Відкрити скетч catkin_ws/src/mpu6050_imu_ros/ mpu6050_imu_driver/firmware/MPU6050_DMP6/MPU6050_DMP6.ino, змінити швидкість послідовного інтерфейсу на 57600 і завантажити прошивку в Arduino Uno.

12. Підклчити модуль з акселерометром-гіроскопом MPU6050 до Arduino Uno згідно рис. 1.

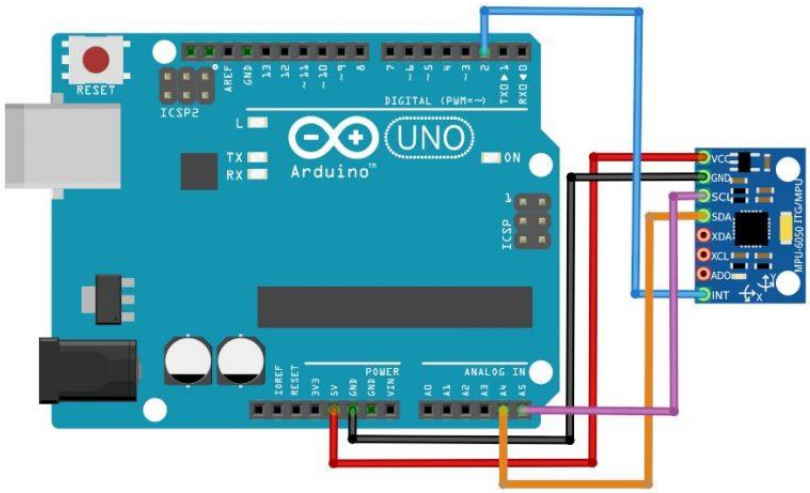

Рис. 1. Схема підключення MPU6050 до Arduino Uno

13. Запустити головний процес ROS командою roscore& .

14. Запустити *roslaunch mpu6050_imu_driver mpu6050_imu.launch* ,

за потреби виправити послідовний порт в файлі catkin_ws/src/mpu6050_imu_ros/mpu6050_imu_driver/launch/mpu6050_imu.launch

15. У запущеному Rviz перевірити реакцію на переміщення модуля з MPU6050 (рис. 2).

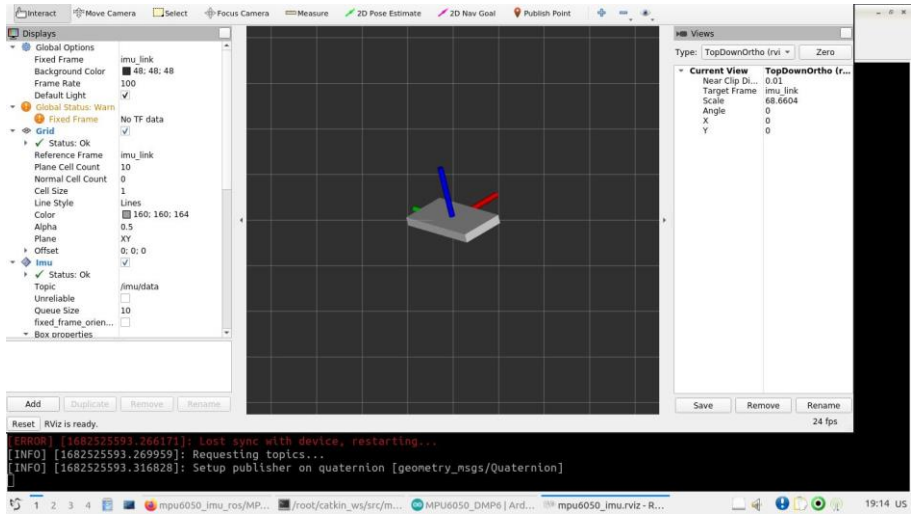

Рис. 2. Візуалізація поточного положення, визначеного акселерометром-гіроскопом MPU6050, в Rviz

16. В Arduino Uno у складі Keystudio Tank Robot завантажити прошивку (живлення від акумуляторів 18650 і модуль HM-10 повинні бути відключені), яка надаватиме можливість віддаленого керування через Bluetooth-модуль, що комунікує з Arduino Uno через послідовний інтерфейс:

```
/*
 keyestudio Mini Tank Robot v2.0
 http://www.keyestudio.com
*/
//Array, used to store the data of the pattern, can be calculated by yourself or obtained from the modulus tool
unsigned char start01[] = {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01};
unsigned char front[] = {0x00,0x00,0x00,0x00,0x00,0x24,0x12,0x09,0x12,0x24,0x00,0x00,0x00,0x00,0x00,0x00};
unsigned char back[] = {0x00,0x00,0x00,0x00,0x00,0x24,0x48,0x90,0x48,0x24,0x00,0x00,0x00,0x
```

```
00,0x00,0x00};
    unsigned char left[] =
{0x00,0x00,0x00,0x00,0x00,0x00,0x44,0x28,0x10,0x44,0x28,0x10,0x44,0x
28,0x10,0x00};
    unsigned char right[] =
{0x00,0x10,0x28,0x44,0x10,0x28,0x44,0x10,0x28,0x44,0x00,0x00,0x00,0x
00,0x00,0x00};
    unsigned char STOP01[] =
{0x2E,0x2A,0x3A,0x00,0x02,0x3E,0x02,0x00,0x3E,0x22,0x3E,0x00,0x3E,0
x0A,0x0E,0x00};
    unsigned char clear[] =
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
00,0x00,0x00};
    #define SCL_Pin  A5  //Set clock pin to A5
    #define SDA_Pin  A4  //Set data pin to A4

    #define ML_Ctrl 13  //define direction control pin of left motor
    #define ML_PWM 11   //define PWM control pin of left motor
    #define MR_Ctrl 12  //define direction control pin of right motor
    #define MR_PWM 3   //define PWM control pin of right motor

    #define Trig 5  //ultrasonic trig Pin
    #define Echo 4  //ultrasonic echo Pin
    const int max_speed=200;
    int distance; //save the distance value detected by ultrasonic, follow
function
    int random2; //save the variable of random numberssave the variable
of random numbers
    //save the distance value detected by ultrasonic, obstacle avoidance
function
    int a;
    int a1;
    int a2;

    #define servoPin 9  //servo Pin
    int pulsewidth;

    #define light_L_Pin A1   //define the pin of left photo resistor sensor
```

```
#define light_R_Pin A2   //define the pin of right photo resistor sensor
int left_light;
int right_light;

char bluetooth_val; //save the value of Bluetooth reception
int flag;  //flag variable, it is used to entry and exist function
void setup(){
  Serial.begin(9600);
  pinMode(Trig, OUTPUT);
  pinMode(Echo, INPUT);
  pinMode(ML_Ctrl, OUTPUT);
  pinMode(ML_PWM, OUTPUT);
  pinMode(MR_Ctrl, OUTPUT);
  pinMode(MR_PWM, OUTPUT);

  pinMode(servoPin, OUTPUT);
  procedure(90); //set servo to 90°
 pinMode(SCL_Pin,OUTPUT);
  pinMode(SDA_Pin,OUTPUT);
  matrix_display(clear);    //Clear the display
  matrix_display(start01);  //display start pattern
  pinMode(light_L_Pin, INPUT);
  pinMode(light_R_Pin, INPUT);
}

void loop(){
  if (Serial.available())
  {
    bluetooth_val = Serial.read();
    Serial.println(bluetooth_val);
  }
  switch (bluetooth_val)
  {
    case 'F':  //Forward instruction
      Car_front();
      matrix_display(front);  //display forward pattern
      break;
    case 'B':  //Back instruction
```

```
          Car_back();
          matrix_display(back);  // display back pattern
          break;
        case 'L':  //left-turning instruction
          Car_left();
          matrix_display(left);  //show left-turning pattern
          break;
        case 'R':  //right-turning instruction
          Car_right();
          matrix_display(right);  //show right-turning pattern
          break;
        case 'S':  //stop instruction
          Car_Stop();
          matrix_display(STOP01);  //display stop pattern
          break;
        case 'Y':
          matrix_display(start01);  //show start pattern
          follow();
          break;
        case 'U':
          matrix_display(start01);  //show start pattern
          avoid();
          break;
        case 'X':
          matrix_display(start01);  //show start pattern
          light_track();
          break;
      }}
    /*****************Obstacle Avoidance Function**************/
    void avoid()
    {
      flag = 0;  //the design that enter obstacle avoidance function
      while (flag == 0)
      {
        random2 = random(1, 100);
        a = checkdistance();  //assign the front distance detected by
ultrasonic sensor to variable a
```

```
if (a < 20) //when the front distance detected is less than 20cm
{
  Car_Stop();  //robot stops
  delay(200); //delay in 200ms

  procedure(160);  //Ultrasonic platform turns left
  for (int j = 1; j <= 10; j = j + (1)) { ///for statement, the data will be more accurate if ultrasonic sensor detect a few times.
    a1 = checkdistance();  //assign the left distance detected by ultrasonic sensor to variable a1
  }
  delay(200);
  procedure(20); //Ultrasonic platform turns right
  for (int k = 1; k <= 10; k = k + (1)) {
    a2 = checkdistance(); //assign the right distance detected by ultrasonic sensor to variable a2
  }
  if (a1 < 50 || a2 < 50)  //robot will turn to the longer distance side when left or right distance is less than 50cm.if the left or right distance is less than 50cm, the robot will turn to the greater distance
  {
    if (a1 > a2) //left distance is greater than right
    {
      procedure(90);  //Ultrasonic platform turns back to right ahead ultrasonic platform turns front
      Car_left();  //robot turns left
      delay(500);  //turn left 500ms
      Car_front(); //go forward
    }
    else
    {
      procedure(90);
      Car_right(); //robot turns right
      delay(500);
      Car_front();  //go forward
    }
  }
  else  //both distance on two side is greater than or equal to 50cm,
```

```
turn randomly
            {
              if ((long) (random2) % (long) (2) == 0)  //when the random
number is even
              {
                procedure(90);
                Car_left(); //robot turns left
                delay(500);
                Car_front(); //go forward
              }
              else
              {
                procedure(90);
                Car_right(); //robot turns right
                delay(500);
                Car_front(); ///go forward
             } } }
          else  //If the front distance is greater than or equal to 20cm, robot car
will go front
          {
             Car_front(); //go forward
          }
          // receive the Bluetooth value to end the obstacle avoidance function
          if (Serial.available())
          {
            bluetooth_val = Serial.read();
            if (bluetooth_val == 'S')  //receive S
            {
              flag = 1;  //when assign 1 to flag, end loop
            }}}}
        /*******************Follow***************/
        void follow() {
          flag = 0;
          while (flag == 0) {
            distance = checkdistance();  //assign the distance detected by
ultrasonic sensor to distance
            if (distance >= 20 && distance <= 60) //the range to go front
            {
```

```
      Car_front();
    }
    else if (distance > 10 && distance < 20)  //the range to stop
    {
      Car_Stop();
    }
    else if (distance <= 10)  // the range to go back
    {
      Car_back();
    }
    else  //other situations, stop
    {
      Car_Stop();
    }
    if (Serial.available())
    {
      bluetooth_val = Serial.read();
      if (bluetooth_val == 'S')
      {
        flag = 1;  //end loop
      }}}}
  //The function to control ultrasonic sensor the function controlling
ultrasonic sensor
  float checkdistance() {
    digitalWrite(Trig, LOW);
    delayMicroseconds(2);
    digitalWrite(Trig, HIGH);
    delayMicroseconds(10);
    digitalWrite(Trig, LOW);
    float distance = pulseIn(Echo, HIGH) / 58.00;  //58.20 means
2*29.1=58.2
    delay(10);
    return distance;
  }
  //The function to control servo the function controlling servo
  void procedure(int myangle) {
    for (int i = 0; i <= 50; i = i + (1)) {
      pulsewidth = myangle * 11 + 500;
```

```
    digitalWrite(servoPin,HIGH);
    delayMicroseconds(pulsewidth);
    digitalWrite(servoPin,LOW);
    delay((20 - pulsewidth / 1000));
  }}

/****************Light Follow*****************/
void light_track() {
  flag = 0;
  while (flag == 0) {
    left_light = analogRead(light_L_Pin);
    right_light = analogRead(light_R_Pin);
    if (left_light > 650 && right_light > 650) //the value detected by
photo resistor, go forward
    {
      Car_front();
    }
    else if (left_light > 650 && right_light <= 650)  //the value detected
by photo resistor, turn left
    {
      Car_left();
    }
    else if (left_light <= 650 && right_light > 650) //the value detected by
photo resistor, turn right
    {
      Car_right();
    }
    else  //other situations, stop
    {
      Car_Stop();
    }
    if (Serial.available())
    {
      bluetooth_val = Serial.read();
      if (bluetooth_val == 'S') {
        flag = 1;
     }}}}
/***************Dot Matrix *****************/
```

```
// this function is used for dot matrix display
void matrix_display(unsigned char matrix_value[])
{
  IIC_start();
  IIC_send(0xc0);  //Choose address

  for(int i = 0;i < 16;i++) //pattern data has 16 bits
  {
    IIC_send(matrix_value[i]); //convey the pattern data
  }
  IIC_end();   //end the transmission of pattern data
  IIC_start();
  IIC_send(0x8A);  //display control, set pulse width to 4/16
  IIC_end();
}
//The condition starting to transmit data
void IIC_start()
{
  digitalWrite(SCL_Pin,HIGH);
  delayMicroseconds(3);
  digitalWrite(SDA_Pin,HIGH);
  delayMicroseconds(3);
  digitalWrite(SDA_Pin,LOW);
  delayMicroseconds(3);
}
//convey data
void IIC_send(unsigned char send_data)
{
  for(char i = 0;i < 8;i++)  //each byte has 8 bits
  {
      digitalWrite(SCL_Pin,LOW);  //pull down clock pin SCL Pin to
change the signals of SDA
      delayMicroseconds(3);
      if(send_data & 0x01)  //set high and low level of SDA_Pin
according to 1 or 0 of every bit
      {
        digitalWrite(SDA_Pin,HIGH);
      }
```

```
        else
        {
          digitalWrite(SDA_Pin,LOW);
        }
        delayMicroseconds(3);
        digitalWrite(SCL_Pin,HIGH); //pull up clock pin SCL_Pin to stop
transmitting data
        delayMicroseconds(3);
        send_data = send_data >> 1;  // detect bit by bit, so move the data
right by one
      }}
    //The sign that data transmission ends
    void IIC_end()
    {
      digitalWrite(SCL_Pin,LOW);
      delayMicroseconds(3);
      digitalWrite(SDA_Pin,LOW);
      delayMicroseconds(3);
      digitalWrite(SCL_Pin,HIGH);
      delayMicroseconds(3);
      digitalWrite(SDA_Pin,HIGH);
      delayMicroseconds(3);
    }
    /**********the function to run motor**********/
    void Car_front()
    {
      digitalWrite(MR_Ctrl,LOW);
      analogWrite(MR_PWM,max_speed);
      digitalWrite(ML_Ctrl,LOW);
      analogWrite(ML_PWM,max_speed);
    }
    void Car_back()
    {
      digitalWrite(MR_Ctrl,HIGH);
      analogWrite(MR_PWM,max_speed);
      digitalWrite(ML_Ctrl,HIGH);
      analogWrite(ML_PWM,max_speed);
    }
```

```
void Car_left()
{
  digitalWrite(MR_Ctrl,LOW);
  analogWrite(MR_PWM,max_speed);
  digitalWrite(ML_Ctrl,HIGH);
  analogWrite(ML_PWM,max_speed);
}
void Car_right()
{
  digitalWrite(MR_Ctrl,HIGH);
  analogWrite(MR_PWM,max_speed);
  digitalWrite(ML_Ctrl,LOW);
  analogWrite(ML_PWM,max_speed);
}
void Car_Stop()
{
  digitalWrite(MR_Ctrl,LOW);
  analogWrite(MR_PWM,0);
  digitalWrite(ML_Ctrl,LOW);
  analogWrite(ML_PWM,0);
}
void Car_T_left()
{
  digitalWrite(MR_Ctrl,LOW);
  analogWrite(MR_PWM,max_speed);
  digitalWrite(ML_Ctrl,LOW);
  analogWrite(ML_PWM,max_speed*3/4);
}
void Car_T_right()
{
  digitalWrite(MR_Ctrl,LOW);
  analogWrite(MR_PWM,max_speed*3/4);
  digitalWrite(ML_Ctrl,LOW);
  analogWrite(ML_PWM,max_speed);
}
```

17. Підключити модуль HM-10 і ввімкнути живлення від акумуляторів.

18. Використовуючи ble-serial або мобільний застосунок BLE

Scanner, підключитись до bluetooth-модуля робота й перевірити функціонування робота відповідно до рис. 3.

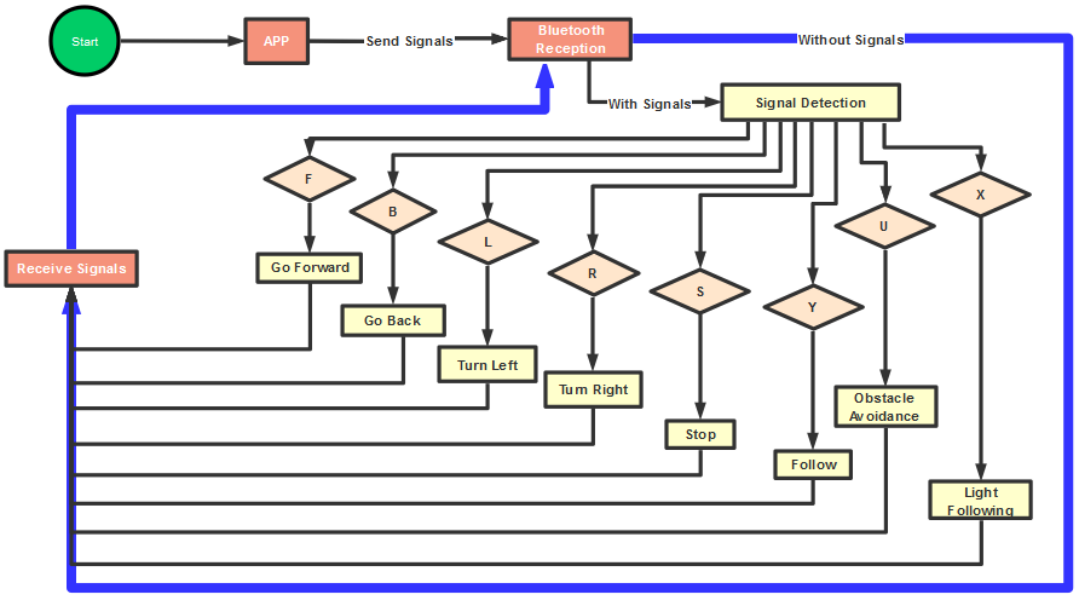

Рис. 3. Блок-схема алгоритму програми робота

19. Підключити до робота модуль акселерометра-гіроскопа замість світлодіодної матриці через інтерфейс I2C, модифікувати базову програму так, щоб дані про положення передавались в ROS і візуалізовувались за допомогою Rviz.

20. Оформити звіт, що повинен містити титульну сторінку, тему, мету роботи, порядок виконання, кілька скріншотів роботи Rviz (п.15) при різних положеннях MPU6050, текст програми для роботом через Bluetooth, висновок.

Контрольні запитання

1. Яка програма в ROS дозволяє візуалізувати дані про положення?
2. Повідомлення якого типу формується платою Arduino Uno з підключеним MPU6050 у цій роботі?
3. Який протокол використовується для зв'язку ROS з платою Arduino?
4. Як у програмі задається швидкість і напрямок обертання лівої та правої гусениць робота?

Лабораторна робота №4. Керування гусеничною платформою з Raspberry Pi.

Мета робота: навчитись керувати гусеничним роботом з контролером Raspberry Pi.

Теоретичні відомості

Рис. 4. Гусеничний робот XiaorGeek

Керування гусеничним роботом з Raspberry Pi (рис. 4) неможливе без силового перетворювача, який забезпечуватиме подачу напруги живлення на приводи відповідно до сигналів керування з мікропроцесорної плати.

Для підключення приводів використовується плата розширення з драйвером двигуна L298P (рис. 5). Входи даної силової мікросхеми підключені до 6 контактів Raspberry Pi: два входи Enable ввімкнення-вимкнення півмостів драйвера, 4 входи IN1…IN4 для задання рівня напруги на силових виходах, підключених до двигунів M1 та M2. Двигун обертатиметься, якщо на відповідну пару входів IN1-IN2 або IN3-IN4 подати протилежні логічні рівні (на один - логічну 1, на другий - логічний 0). Встановлення логічного 0 на обох входах призведе до зупинки двигуна. Змінюючи скважність сигналів, можна керувати швидкістю руху гусениць робота.

До плати розширення підключені сервоприводи маніпулятора (SER1…SER4) та сервоприводи нахилу і повороту камери (SER7, SER8). Генеруючи імпульси керування на відповідні контакти, можна керувати положенням маніпулятора та напрямком зору камери.

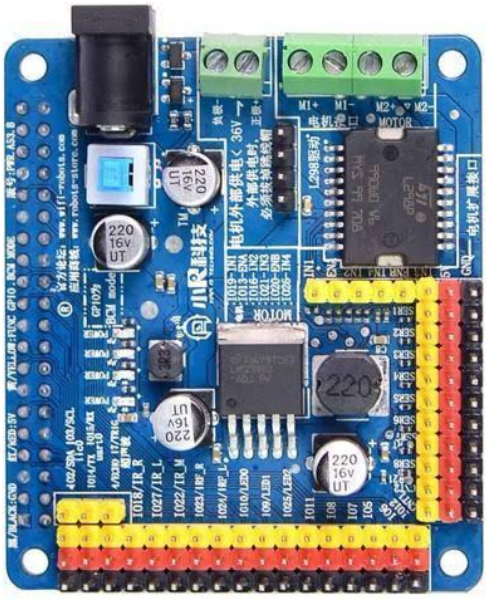

Рис. 5. Плата для керування приводами робота

Камера має USB-інтерфейс і розпізнається в Raspberry Pi OS як UVC-камера, що дозволяє захопити зображення з неї засобами OpenCV, надати доступ до відеопотоку іншому вузлу мережі за допомогою FFmpeg, VLC або іншого сервера трансляції потокового відео.

Порядок виконання роботи

1. Перевірити підключення приводів робота до відповідних контактів плати розширення.

2. Завантажити програму дистанційного керування гусеничним роботом з камерою і маніпулятором з сторінки https://www.xiaorgeek.com/Software/index.html

3. Ввімкнути живлення перемикачем на платі розширення. Почекати завантаження Raspberry Pi. Підключитись до нової точки доступу WiFi XiaorGeek_* й запустити програму WifiRobot.

4. Перевірити можливість дистанційного керування кожним приводом у системі (приводи лівої та правої гусениць, сервоприводи маніпулятора, сервоприводи камери).

5. Вимкнути живлення перемикачем на платі розширення.

6. Написати програму для визначення граничної скважності ШІМ-сигналу, потрібної для приведення в рух робота (для руху в одному напрямку сигнали на двох контактах INx драйвера двигуна повинні бути лог. 0, а на інших двох - ШІМ).

7. Налагодити програму на окремій Raspberry Pi з світлодіодами, підключаючись по SSH до неї та запускаючи програму з різними значеннями скважності.

8. Переставити microSD-картку з написаною програмою в контролер робота. Ввімкнути живлення на підключитись по SSH.

9. Дослідити залежність швидкості від коефіцієнта заповнення ШІМ сигналу, задаючи все більший коефіцієнт заповнення ШІМ-сигналу. Знайти порогове значення, менше якого формувати імпульси немає сенсу, оскільки гусеничний робот не починає рух.

10. Побудувати графік залежності швидкості від коефіцієнта заповнення ШІМ-сигналу.

11. Додати трансляцію відео з камери робота під час руху, використовуючи VLC або FFmpeg (на вибір).

12. Оформити звіт, що повинен містити титульну сторінку, тему, мету роботи, порядок виконання, текст програми для керування швидкістю робота, графік залежності швидкості від коефіцієнта заповнення, висновок.

Контрольні запитання

1. Яка роль драйвера двигуна L298P в роботі?
2. Чим пояснюється відсутність руху робота при невеликих значеннях тривалості імпульсів?
3. Як можна згенерувати ШІМ-сигнал в Raspberry Pi?
4. Як можна отримати зображення з камери робота?

Список рекомендованої літератури

1. Damith Herath, David St-Onge. Foundations of Robotics: A Multidisciplinary Approach with Python and ROS. – Springer/eBook, 2022. – 564 p.

2. YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim. ROS Robot Programming. – ROBOTIS Co., Ltd, 2017. – 460 p.

3. Margolis Michael. Arduino Cookbook. O'Reilly Media, 2011. 662 p.

4. Evans B. Arduino programming notebook. First edition. 2007. 38 p. URL: https://playground.arduino.cc/uploads/Main/arduino_notebook_v1-1.pdf.

Лабораторна робота №5. Керування сервоприводом за допомогою програмного ШІМ плати Raspberry Pi.

Мета роботи: навчитись керувати сервоприводами робота з плати Raspberry Pi, використовуючи програмний ШІМ.

Теоретичні відомості

Сервопривод - це привод з управлінням через негативний зворотний зв'язок, що дозволяє точно керувати параметрами руху (рис. 6). Сервоприводом є будь-який тип механічного приводу, що має в складі давач (положення, швидкості, зусилля тощо) і блок керування приводом, який автоматично підтримує необхідні параметри згідно заданому завданню (рис. 7). Сервопривод отримує на вхід значення завдання, наприклад, кут повороту. Блок управління порівнює це значення зі значенням на своєму датчику. На основі результату порівняння привод виконує певну дію, наприклад: поворот, прискорення або сповільнення так, щоб значення з внутрішнього давача стало якомога ближче до значення зовнішнього керуючого параметра. Найбільш поширені сервоприводи, які утримують заданий кут і сервоприводи, що підтримують задану швидкість обертання.

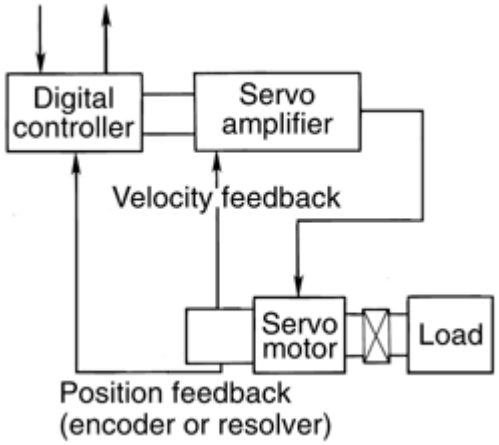

Рис. 6. Структурна схема сервопривода з давачем швидкості обертів

Рис. 7. Конструкція сервопривода

Отже, сервопривод – це регульований редукторний електродвигун. Він зазвичай складається з приводного механізму з двигуном постійного струму, плати управління і потенціометра, котрий забезпечує зворотний зв'язок.

Керуючий сигнал для сервопривода – імпульси постійної частоти і змінної ширини. Найчастіше в малих сервоприводах імпульси виробляються з частотою 50 Гц. Це означає, що імпульс випускається і приймається раз в 20 мс. Зазвичай при цьому тривалість імпульсу 1520 мкс означає, що сервопривод повинен зайняти середнє положення. Збільшення або зменшення довжини імпульсу змусить сервопривод повернутися за годинниковою або проти годинникової стрілки відповідно. При цьому існують верхня і нижня межі тривалості імпульсу (рис. 8).

У бібліотеці Servo для Arduino за замовчуванням виставлені наступні значення довжин імпульсу: 544 мкс – для 0° і 2400 мкс – для 180°. Фактичні параметри конкретного екземпляра можуть відрізнятися від вказаних. Для точної роботи кожен конкретний сервопривод повинен бути відкалібрований: шляхом експериментів необхідно підібрати коректний діапазон, характерний саме для нього.

Сигнали керування сервоприводом можуть генеруватись апаратно з використанням вбудованих таймерів або програмно, коли для зміни стану цифрового виходу процесором періодично мають виконуватись відповідні інструкції.

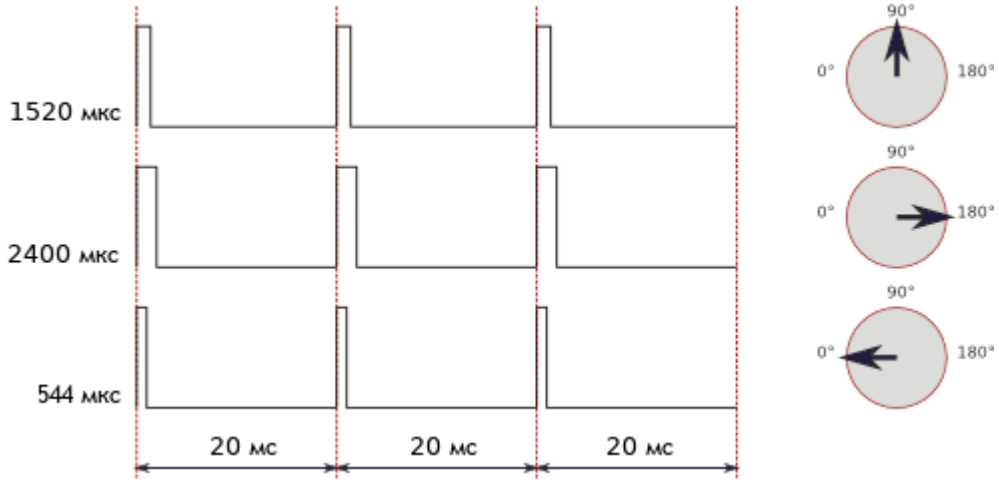

Рис. 8. Залежність кута повороту сервопривода від трвалості керуючих імпульсів

Завершуючи роботу з сервоприводом, варто перевести його в безпечний стан.

Наступна програма демонструє зупинку генерування ШІМ-сигналу, встановлення низького логічного рівня на контакті й переналаштування його на вхід при завершенні роботи програми. Відповідно сервопривод, підключений до відповідного контакту GPIO, перестане отримувати імпульси керування й залишиться у попередньому положенні.

```
#include <wiringPi.h> //бібліотека для роботи з GPIO Raspberry Pi
#include <stdio.h>
#include <softPwm.h>  //реалізація програмного ШІМ
#include <stdlib.h>
#include <signal.h>

const int PWM_pin=6;
void gpioExit(void);
void sig_action(int sig);

int main(){
    printf("Налаштовую ШІМ\n");
    int intensity;              //тривалість імпульсу 0…200
    wiringPiSetup();            //налаштування для роботи GPIO
    if(atexit(gpioExit)!=0) {
```

```
            printf("Не вдалось зареєструвати функцію, виконувану при виході\n");
            exit(EXIT_FAILURE);
          }
        signal(SIGINT, sig_action);
        signal(SIGHUP, sig_action);
        signal(SIGTERM, sig_action);
        pinMode(PWM_pin, OUTPUT);       /* set GPIO as output */
        softPwmCreate(PWM_pin, 15, 200);//контакт-вихід, тривалість імпульсу, період
        while (1) //закоментувати для перевірки завершення програми з return
         {
            for(intensity = 10; intensity <= 20; intensity++)
            {
              softPwmWrite(PWM_pin, intensity); //
              delay(300) ;
            }
            delay(1000);

            for(intensity = 20; intensity >= 10; intensity--)
            {
              softPwmWrite(PWM_pin, intensity);
              delay(300);
            }
            delay(1000);
        }
        printf("Завершую програму з return\n");
        return 0; //виконає функцію, задану atexit(), і безпечно завершить роботу
    }

    void gpioExit(void) {
        printf("Виконую функцію, зареєстровану atexit\n");
        softPwmStop(PWM_pin);
        delay(5);
        digitalWrite(PWM_pin,LOW);
        pinMode(PWM_pin,INPUT);
```

```
}

void sig_action(int sig) {
    printf("Один з сигналів SIGINT, SIGHUP, SIGTERM отримано процесом. Завершую роботу\n");
    exit(0); //виконає функцію, задану atexit(), і безпечно завершить роботу
}
```

Окрім виконання перемикань стану виводів власною програмою, що виконується в просторі користувача, можна використати servod, який задіює модуль ServoBlaster, що працює в просторі ядра, а отже менше залежить від завантаженості процесора іншими процесами, або запускається як демон від імені root.

ServoBlaster - це програмне забезпечення для RaspberryPi, яке забезпечує інтерфейс для керування кількома сервоприводами через контакти GPIO. Користувач керує положеннями сервоприводу, надсилаючи команди драйверу, вказуючи, якої ширини повинен мати імпульс на конкретному виході на сервопривод. Драйвер підтримує цю ширину імпульсу, доки не надійде нова команда із запитом іншої ширини. За замовчуванням він налаштований на керування 8 сервоприводами, хоча можна налаштувати і до 21. Окрім керування сервоприводами, ServoBlaster можна налаштувати для генерування імпульсів шириною від 0 до 100% часу циклу, що робить його придатним для керування яскравістю до 21 світлодіода, наприклад, або швидкістю колекторного двигуна постійного струму. Драйвер створює файл пристрою, /dev/servoblaster, до якого можна надсилати команди. Формат команди:

<servo-number>=<servo-position>

або

P<header>-<pin>=<servo-position>

Для першого формату <servo-number> є номером сервопривода, який за замовчуванням є числом від 0 до 7 включно. Для другого формату <header> є числом від 1 або 5, залежно від того, до якого роз’єму під’єднано сервопривод, і <pin> є номером піна цього роз’єму, до якого він підключений. За замовчуванням <servo-position> — ширина імпульсу, де одиниця відповідає 10 мкс, хоча це можна змінити за допомогою аргументів командного рядка, також можна вказати в

мікросекундах або у відсотках від максимально дозволеної ширини імпульсу.

За замовчуванням сервоприводи керуються з наступних контактів (табл. 3):

Таблиця 3

Відповідність номерів сервоприводів і контактів у ServoBlaster

Servo number	GPIO number	Pin in P1 header
0	4	P1-7
1	17	P1-11
2	18	P1-12
3	21/27	P1-13
4	22	P1-15
5	23	P1-16
6	24	P1-18
7	25	P1-22

Відповідно команди

echo 0=150 > /dev/servoblaster #кількість кроків

echo 0=50% > /dev/servoblaster #коефіцієнт заповнення ШІМ у відсотках

echo 0=1500us > /dev/servoblaster #тривалість імпульсу в мікросекундах

еквівалентні наступним

echo P1-7=150 > /dev/servoblaster

echo P1-7=50% > /dev/servoblaster

echo P1-7=1500us > /dev/servoblaster

Переміщення відносно поточної позиції можна задати наступними командами:

echo 0=+10 > /dev/servoblaster

echo 0=-20 > /dev/servoblaster

Зчитати поточні положення сервоприводів можна, звернувшись до файлу пристрою для читання, наприклад, командою *cat /dev/servoblaster* .

Порядок виконання роботи

1. Виконати SSH-підключення до Raspberry Pi.

2. Створити нову програму, що використовуватиме бібліотеку WiringPi з попередньої роботи й виводитиме програмно ШІМ-сигнал періодом 50 Гц та тривалістю від 1 до 2 мс на 22-ий контакт у 40-контактному роз'ємі GPIO. Програма повинна впродовж 10 секунд переводити сервопривод з одного крайнього положення в інше. Використати функцію *softPwmWrite().*

3. Підключити сервопривод до роз'єму Servo2, в групі DIP-перемикачів U$35 перевести вгору 8-ий перемикач. Запустити програму, поспостергіати за стабільністю положення сервопривода при незмінному завданні положення. Спробувати додатково навантажити процесор іншим процесом, запустивши його через окреме SSH-підключення. Перевірити, чи змінилась стабільність положення сервоприводу.

4. В файлі https://github.com/WiringPi/WiringPi/blob/master/wiringPi/softPwm.c здійснити зменшення бази часу й у програмі надіслати те саме задане кутове положення сервоприводу. Оцінити вплив зменшення на стабільність положення.

5. Клонувати репозиторій https://github.com/richardghirst/PiBits в каталог з власним прізвищем на Raspberry Pi. Перейти в каталог *cd PiBits/ServoBlaster/user*. Виконати *make* з ціллю за замовчуванням. Встановити отриманий *servod* у систему командою *sudo make install*. Перевірити наявність у /dev пристрою /dev/servoblaster.

6. З термінала змінити положення сервопривода, записавши в пристрій /dev/servoblaster нову тривалість імпульсів керування сервоприводом командою *echo P1-7=1500us > /dev/servoblaster* .

6. Розробити програму, що керує сервоприводом з використанням демона servod. Для передачі на пристрій /dev/servoblaster використати функцію system(). Програма повинна впродовж 10 секунд переводити сервопривод з одного крайнього положення в інше.

7. Запустити програму й порівняти стабільність положення сервоприводу з раніше розробленою програмою.

8. Розробити програму керування сервоприводами захвату й нахилу маніпулятора на роботі Xiaorgeek.

9. Видалити ServoBlaster командою *sudo make uninstall*.

10. Оформити звіт, що повинен містити титульну сторінку, тему, мету роботи, порядок виконання, тексти програм, висновок.

Контрольні запитання

1. З чого складається сервопривод?

2. Який вигляд мають сигнали керування сервоприводом?

3. Яку функцію WiringPi можна використати для генерування сигналів керування сервоприводом?

4. В чому різниця між програмним та апаратним генеруванням ШІМ-сигналу?

5. Який файл пристрою створює ServoBlaster для своєї роботи?

6. Який демон відповідає за генерування сигналів керування сервоприводами відповідно до даних, що надходять на відповідний файл пристрою?

7. Як за допомогою servod надіслати імпульси тривалістю 1200 мкс на сервопривод, підключений до контакту 22 у 40-контактному роз'ємі?

8. Як за допомогою servod надіслати ШІМ-сигнал скважністю 25%, підключений до контакту 22 у 40-контактному роз'ємі?

9. Як за допомогою servod повернути на 15 позицій вперед сервопривод, підключений до контакту 22 у 40-контактному роз'ємі?

10. Як за допомогою servod повернути на 60 позицій назад сервопривод, підключений до контакту 22 у 40-контактному роз'ємі?

Лабораторна робота №6. Пошук об’єктів у полі зору робота засобами бібліотеки комп’ютерного зору OpenCV.

Мета роботи: навчитись знаходити об’єкти в відеопотоці за допомогою порогової фільтрації засобами бібліотеки комп’ютерного зору OpenCV.

Теоретичні відомості

OpenCV – одна з найвідоміших бібліотек комп’ютерного зору, що використовується зокрема й в робототехніці.

Бібліотека OpenCV складається з таких компонентів:

• opencv_core - основна функціональність. Включає в себе базові структури, обчислення (математичні функції, генератори випадкових чисел) і лінійну алгебру, DFT, DCT, введення / виведення для XML і YAML і т. д.

• opencv_imgproc - обробка зображень (фільтрація, геометричні перетворення, перетворення колірних просторів і т. д.).

• opencv_highgui - простий інтерфейс користувача, введення / виведення зображень і відео.

• opencv_ml - моделі машинного навчання (SVM, дерева рішень, навчання зі стимулюванням і т. д.).

• opencv_features2d - розпізнавання і опис плоских примітивів (SURF, FAST й інші, включаючи спеціалізований фреймворк).

• opencv_video - аналіз руху і відстеження об'єктів (оптичний потік, шаблони руху, усунення фону).

• opencv_objdetect - виявлення об'єктів на зображенні (перебування осіб за допомогою алгоритму Віоли-Джонса, розпізнавання людей HOG і т. д.).

• opencv_calib3d - калібрування камери, пошук стерео-відповідності та елементи обробки тривимірних даних.

• opencv_flann - бібліотека швидкого пошуку найближчих сусідів (FLANN 1.5) і обгортки OpenCV.

• opencv_contrib - супутній код, ще не готовий для застосування.

• opencv_legacy - застарілий код, збережений заради зворотної сумісності.

• opencv_gpu - прискорення деяких функцій OpenCV за рахунок використання відеокарт, що підтримують CUDA.

За допомогою функцій бібліотеки OpenCV можна конвертувати зображення з відео у інший колірний простір, застосувати порогову фільтрацію для виділення об’єкта, якщо він характеризується істотними

відмінностями від фону в одному з трьох каналів HSV. Мовою C++ це можна реалізувати наступним чином:

```
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/videoio.hpp"
#include <iostream>
using namespace cv;
const int max_value_H = 360/2;
const int max_value = 255;
const String window_capture_name = "Video Capture";
const String window_detection_name = "Object Detection";
int low_H = 0, low_S = 0, low_V = 0;
int high_H = max_value_H, high_S = max_value, high_V = max_value;
static void on_low_H_thresh_trackbar(int, void *)
{
    low_H = min(high_H-1, low_H);
    setTrackbarPos("Low H", window_detection_name, low_H);
}
static void on_high_H_thresh_trackbar(int, void *)
{
    high_H = max(high_H, low_H+1);
    setTrackbarPos("High H", window_detection_name, high_H);
}
static void on_low_S_thresh_trackbar(int, void *)
{
    low_S = min(high_S-1, low_S);
    setTrackbarPos("Low S", window_detection_name, low_S);
}
static void on_high_S_thresh_trackbar(int, void *)
{
    high_S = max(high_S, low_S+1);
    setTrackbarPos("High S", window_detection_name, high_S);
}
static void on_low_V_thresh_trackbar(int, void *)
{
    low_V = min(high_V-1, low_V);
    setTrackbarPos("Low V", window_detection_name, low_V);
}
```

```
static void on_high_V_thresh_trackbar(int, void *)
{
    high_V = max(high_V, low_V+1);
    setTrackbarPos("High V", window_detection_name, high_V);
}
int main(int argc, char* argv[])
{
    VideoCapture cap(argc > 1 ? atoi(argv[1]) : 0);
    namedWindow(window_capture_name);
    namedWindow(window_detection_name);
    // повзунки для задання рівня для порогового фільтра
    createTrackbar("Low H", window_detection_name, &low_H,
max_value_H, on_low_H_thresh_trackbar);
    createTrackbar("High H", window_detection_name, &high_H,
max_value_H, on_high_H_thresh_trackbar);
    createTrackbar("Low S", window_detection_name, &low_S,
max_value, on_low_S_thresh_trackbar);
    createTrackbar("High S", window_detection_name, &high_S,
max_value, on_high_S_thresh_trackbar);
    createTrackbar("Low V", window_detection_name, &low_V,
max_value, on_low_V_thresh_trackbar);
    createTrackbar("High V", window_detection_name, &high_V,
max_value, on_high_V_thresh_trackbar);
    Mat frame, frame_HSV, frame_threshold;
    while (true) {
        cap >> frame;
        if(frame.empty())
        {
            break;
        }
        // перетворення у колірний простір HSV
        cvtColor(frame, frame_HSV, COLOR_BGR2HSV);
        // виділення об'єктів, що відповідають межам порогового
фільтра
        inRange(frame_HSV, Scalar(low_H, low_S, low_V),
Scalar(high_H, high_S, high_V), frame_threshold);
        // виведення кадрів
        imshow(window_capture_name, frame);
```

```
            imshow(window_detection_name, frame_threshold);
            char key = (char) waitKey(30);
            if (key == 'q' || key == 27)
            {
                break;
            }
        }
        return 0;
    }
```

При використанні компілятора g++ та pkg-config скомпілювати виконуваний ELF-файл можна наступним чином: *g++ -o main main.cpp `pkg-config opencv4 --cflags --libs`*

Мовою Python теж можна виконати вказані операції:

```
import cv2 as cv
import argparse
max_value = 255
max_value_H = 360//2
low_H = 0
low_S = 0
low_V = 0
high_H = max_value_H
high_S = max_value
high_V = max_value
window_capture_name = 'Video Capture'
window_detection_name = 'Object Detection'
low_H_name = 'Low H'
low_S_name = 'Low S'
low_V_name = 'Low V'
high_H_name = 'High H'
high_S_name = 'High S'
high_V_name = 'High V'
def on_low_H_thresh_trackbar(val):
    global low_H
    global high_H
    low_H = val
    low_H = min(high_H-1, low_H)
    cv.setTrackbarPos(low_H_name, window_detection_name, low_H)
def on_high_H_thresh_trackbar(val):
```

```
        global low_H
        global high_H
        high_H = val
        high_H = max(high_H, low_H+1)
        cv.setTrackbarPos(high_H_name, window_detection_name,
high_H)
    def on_low_S_thresh_trackbar(val):
        global low_S
        global high_S
        low_S = val
        low_S = min(high_S-1, low_S)
        cv.setTrackbarPos(low_S_name, window_detection_name, low_S)
    def on_high_S_thresh_trackbar(val):
        global low_S
        global high_S
        high_S = val
        high_S = max(high_S, low_S+1)
        cv.setTrackbarPos(high_S_name, window_detection_name,
high_S)
    def on_low_V_thresh_trackbar(val):
        global low_V
        global high_V
        low_V = val
        low_V = min(high_V-1, low_V)
        cv.setTrackbarPos(low_V_name, window_detection_name, low_V)
    def on_high_V_thresh_trackbar(val):
        global low_V
        global high_V
        high_V = val
        high_V = max(high_V, low_V+1)
        cv.setTrackbarPos(high_V_name, window_detection_name,
high_V)
    parser = argparse.ArgumentParser(description='Code for Thresholding
Operations using inRange tutorial.')
    parser.add_argument('--camera', help='Camera divide number.',
default=0, type=int)
    args = parser.parse_args()
    cap = cv.VideoCapture(args.camera)
```

cv.namedWindow(window_capture_name)

cv.namedWindow(window_detection_name)

cv.createTrackbar(low_H_name, window_detection_name , low_H, max_value_H, on_low_H_thresh_trackbar)

cv.createTrackbar(high_H_name, window_detection_name , high_H, max_value_H, on_high_H_thresh_trackbar)

cv.createTrackbar(low_S_name, window_detection_name , low_S, max_value, on_low_S_thresh_trackbar)

cv.createTrackbar(high_S_name, window_detection_name , high_S, max_value, on_high_S_thresh_trackbar)

cv.createTrackbar(low_V_name, window_detection_name , low_V, max_value, on_low_V_thresh_trackbar)

cv.createTrackbar(high_V_name, window_detection_name , high_V, max_value, on_high_V_thresh_trackbar)

```
while True:
    ret, frame = cap.read()
    if frame is None:
        break
    frame_HSV = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
    frame_threshold = cv.inRange(frame_HSV, (low_H, low_S, low_V), (high_H, high_S, high_V))
    cv.imshow(window_capture_name, frame)
    cv.imshow(window_detection_name, frame_threshold)

    key = cv.waitKey(30)
    if key == ord('q') or key == 27:
        break
```

Порядок виконання роботи

1. Становити OpenCV. Для цього виконайте команду *sudo apt install libopencv-dev opencv-data* .

2. Створити нову програму мовою C++ на базі прикладу з теоретичних відомостей, яка виділятиме в кадрі вебкамери об’єкт, що відповідає заданим межам для каналів H, S, V.

3. Скомпілювати програму командою *g++ -o main main.cpp `pkg-config opencv4 --cflags --libs`* , запустити та налагодити її.

4. Встановити підтримку OpenCV для Python 3 командою *sudo apt install python3-opencv .*

5. Написати аналогічну програму мовою Python і налагодити її.

6. Одну з програм модифікувати так, щоб виділені функцією inRange об'єкти шукались функцією findContours і номери об’єктів виводились біля них. Використати функції drawContours та putText.

7. Вивести координати першої точки першого об’єкта в термінал.

8. Модифікувати програму так, щоб в окрему змінну записувалась горизонтальна координата центра найбільшого знайденого в кадрі об'єкта.

9. Пов'язати горизонтальну координату об'єкта з числом 0...180, яке надалі відповідатиме куту повороту сервоприводу, й вивести його поверх кадру.

10. На основі горизонтальної координати об'єкта розрахувати пропорційну тривалість імпульсу керування сервоприводом маніпулятора на гусеничному роботі й вивести відповідний сигнал з плати Raspberry Pi. Запустити отриману програму й переконатись, що при переміщенні об'єкта в полі зору камери або на відео сервопривод змінює своє положення.

11. Оформити звіт, що повинен містити титульну сторінку, тему, мету роботи, порядок виконання, тексти програм, скріншоти програми, що виводить номери об'єктів, висновок.

Контрольні запитання

1. Що являє собою бібліотека OpenCV?

2. Які основні складові можна виділити в OpenCV?

3. Чи містить бібліотека засоби виводу графічного інтерфейсу користувача? Де саме?

4. Яка функція OpenCV використовується для перетворення зображення в інший колірний простір?

5. Яка функція OpenCV використовується для порогової фільтрації зображення із заданням нижнього та верхнього порогу за кожним каналом H, S, V?

6. Яка функція використовується для одноканальної порогової фільтрації зображення?

7. Яка функція використовується для пошуку контурів на зображенні?

8. Яка функція використовується для нанесення контурів на зображення?

9. Яка функція дозволяє вивести зображення у вікні?