Міністерство освіти і науки України

Національний університет водного господарства та природокористування

Кафедра автоматизації, електротехнічних та
комп'ютерно-інтегрованих технологій

04-03-363М

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт
з освітнього компоненту
«Програмування робототехнічних засобів»
для здобувачів вищої освіти першого (бакалаврського) рівня
за освітньо-професійною програмою «Робототехніка та штучний інтелект»
спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології»
денної та заочної форм навчання

Рекомендовано науково-методичною
радою з якості ННІ АКОТ
Протокол № 9 від 31 серпня 2023 р.

Рівне – 2023

Методичні вказівки до лабораторних робіт з освітнього компоненту «Програмування робототехнічних засобів» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Робототехніка та штучний інтелект» спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» денної та заочної форм навчання [Електронне видання] / Реут Д. Т. – Рівне: НУВГП, 2023. – 64 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 освітньої програми «Робототехніка та штучний інтелект»: Сафоник Андрій Петрович, д.т.н., професор, професор кафедри автоматизації, електротехнічних та комп'ютерно-інтегрованих технологій.

© Д. Т. Реут, 2023
© НУВГП, 2023

Зміст

Вступ

Лабораторні роботи з освітньої компоненти «Програмування робототехнічних засобів» дають змогу на практиці вивчити підходи й прийоми програмування робототехнічних засобів на базі мікропроцесорних плат Raspberry Pi 4B, Arduino Uno в якості контролерів, навчитись створювати програми для операційних систем на базі GNU/Linux для керування ходовою частиною робота, розпізнавання відео, включення в середовище в Robot Operating System. Передумовою вивчення дисципліни є наявність грунтовних знань з дисциплін «Мікропроцесорні системи і програмування мікропроцесорних засобів», «Мехатроніка та роботизовані комплекси», «Штучний інтелект в робототехніці».

У лабораторних роботах розглянуто:

– обробка сигналів операційної системи для безпечного завершення роботи з приводами робота;

– віддалене підключення до контролера робота за допомогою SSH;

– керування сервоприводами за допомогою бібліотеки WiringPi та ServoBlaster;

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

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

– візуалізацію даних, отриманих Robot Operating System;

– керування гусеничними платформами на базі Raspberry Pi 4B та Arduino Uno.

Лабораторна робота №1. Виконання операцій при завершенні програми. Обробка сигналів процесом.

Мета роботи: навчитись виконувати операції при завершенні програми за зовнішнім сигналом.

Теоретичні відомості

При завершенні програми, яка взаємодіяла з апаратними ресурсами, часто виникає задача коректного закінчення роботи з ними, деініціалізації, зупинки генерування сигналів керування на приводи й інші елементи. Інакше увімкнутий сигналом логічної “1” привод продовжуватиме отримувати живлення й обертатись, хоча потреби в цьому може не бути й програма керування ним завершилась. На сервопривод, що керується імпульсним сигналом, замість імпульсів допустимої тривалості може надійти неперервна логічна “1”, якщо імпульси генерувались програмно й програма завершилась до того, як логічний сигнал змінився з “1” назад в “0”.

У випадку завершення програми “за її ініціативою” можна реалізувати потрібні дії коректного закінчення роботи з ресурсом безпосередньо в кінці програми, перед return або викликом функції exit(). Але завершуватись програма, що виконується в POSIX-сумісній операційній системі, може за командою користувача, що обрав завершення процесу в програмі керування процесами (Process Explorer, диспетчер задач, системний монітор, монітор активності тощо), натиснув Ctrl+C в терміналі, за ініціативою операційної системи при нестачі ресурсів, виконанні процесом програми недопустимих операцій, а також при отриманні сигналу від іншого процесу. Це відбувається в непередбачувані заздалегідь моменти часу, які не можна спрогнозувати на етапі створення програми. Тобто під час виконання будь-якого рядка програми може надійти команда завершення. Це відбувається з використанням сигналів операційної системи.

Сигнал являє собою асинхронне повідомлення, що посилається процесу, щоб проінформувати його про подію, яка відбулась. Коли процес отримує сигнал, операційна система перериває хід його виконання і запускає підпрограму обробки цього сигналу. Якщо в програмі явно не задана реакція на сигнал, запускається його стандартний обробник (дія за замовчуванням). Усі сигнали визначені як константи у файлі <signal.h>. Ім'я константи складається з префіксу «SIG» та ще декількох символів, що несуть в собі визначення функції сигналу. Кожна константа має відповідний їй числовий код; числові коди можуть відрізнятися в залежності від платформи.

Таблиця 1

Перелік сигналів POSIX

Сигнал	Код	Дія за замовчуванням	Опис
SIGABRT	6	Д — Завершити процес з додатковими діями	Сигнал переривання процесу
SIGALRM	14	З — Аварійно завершити процес	Сигнал, який посилається, коли спливає час, визначений alarm функцією
SIGBUS	10	Д	Невірне звернення до фізичної пам'яті
SIGCHLD	18	І — Ігнорувати сигнал	Дочірній процес перервано
SIGCONT	25	П — Продовжити перебіг процесу	Продовжує виконання, якщо перед цим процес було зупинено.
SIGFPE	8	Д	Помилкова арифметична операція.
SIGHUP	1	З	Термінал закрито.
SIGILL	4	Д	Недопустима інструкція процесора.
SIGINT	2	З	Сигнал переривання із терміналу (Ctrl-C).
SIGKILL	9	З	Вбити процес (не може бути оброблено або проігноровано процесом).
SIGPIPE	13	З	Спроба записати в конвеєр за відсутності процесу-приймача.
SIGQUIT	3	Д	Сигнал «Quit» з терміналу (Ctrl-\).
SIGSEGV	11	Д	Невірне звернення до віртуальної пам'яті.
SIGSTOP	23	С — Зупинити процес	Зупинити виконання (не може бути оброблено або проігноровано процесом).

SIGTERM	15	З	Сигнал завершення (встановлений за умовчанням для утиліти kill).
SIGTSTP	20	С	Сигнал зупинки, викликаний із терміналу (Ctrl-Z).
SIGTTIN	26	С	Спроба зчитування з терміналу фоновим процесом.
SIGTTOU	27	С	Спроба запису на термінал фоновим процесом.
SIGUSR1	16	З	Сигнал користувача 1.
SIGUSR2	17	З	Сигнал користувача 2.
SIGPOLL	22	З	Сигнал опитування процесу.
SIGPROF	29	З	Таймер профілювання сплинув.
SIGSYS	12	Д	Неправильний системний виклик.
SIGTRAP	5	Д	Сигнал, викликаний точкою зупину або трасуванням.
SIGURG	21	І	Термінові дані, доступні на сокеті.
SIGVTALRM	28	З	Час на «віртуальному таймері» сплинув.
SIGXCPU	30	Д	Ліміт процесорного часу перевищено.
SIGXFSZ	31	Д	Ліміт розміру файлу перевищено.

Для всіх сигналів, крім SIGKILL та SIGSTOP, можна задати власний обробник. Саме в ньому є можливість виконати всі операції з коректного завершення.

Задати обробник на мові С можна функцією

sighandler_t signal(int signum, sighandler_t action)

Наприклад, виклик signal(SIGTERM, sig_action); задасть обробником сигналу завершення процесу SIGTERM функцію sig_action(). Відповідно при виборі “завершити процес”, наприклад, у вікні системного монітору, буде виконана ця функція, а не одразу завершуватиметься процес. У даній функції можна зупинити генерування імпульсів керування на сервоприводи, відключити живлення двигунів, деініціалізувати й перевести в безпечний стан

інше апаратне забезпечення, що використовувалось процесом, але після його завершення може виявитись некерованим, пошкодити інші об'єкти або вийти з ладу саме.

Для кожного з сигналів можна задати власну функцію-обробник або для кількох сигналів задати той самий обробник, якщо потрібні дії при надходженні даних сигналів однакові.

Наприклад, задати один обробник при завершенні програми натисканням Ctrl+C в терміналі, закритті вікна терміналу/SSH-з'єднання, виборі “завершити процес” у вікні системного монітору можна викликами

signal(SIGINT, sig_action);
signal(SIGHUP, sig_action);
signal(SIGTERM, sig_action);

Після виконання необхідних операцій з безпечного завершення роботи з апаратним забезпеченням закінчити роботу програми можна викликом функції

void exit(int status),

яка виконує звичайне завершення програми з очисткою буферів і звільненням виділеної пам'яті. Як правило, функція, що викликає *exit()*, задає для *status* значення 0 для нормального завершення або інше значення, щоб вказати про помилку.

Оскільки і при завершенні програми самостійно з виконанням return, і при завершенні функцією exit() при надходженні сигналу потрібно виконувати одні й ті ж операції безпечного завершення роботи з апаратною частиною, їх зручно помістити в окрему функцію, що викликатиметься при завершенні роботи. Її можна задати функцією

*int atexit(void (*function)(void));*

Функція atexit() реєструє функцію, що передається, в якості функції, що викликається при нормальному завершенні роботи програми, наприклад, за допомогою exit(), або при завершенні роботи функції main. Функцій можна зареєструвати декілька. Зареєстровані функції викликаються у порядку, зворотному порядку реєстрації; жодних аргументів їм не передається.

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); //налаштовуємо на цифр. вихід
    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
    }
```

Аналогічні можливості доступні в мові Python.

Функція signal.signal() дозволяє визначати спеціальні обробники, які будуть виконуватися під час отримання сигналу:

signal.signal(signalnum, handler)

Обробник викликається з двома аргументами: номером сигналу *signalnum* (описані в класі signal.Signals) та власне функцією-обробником *handler*.

```
import signal, sys

# реалізуємо функцію-обробник сигналу
def handler_term(signum, frame):
        print('Отримано сигнал', signum)
        sys.exit(0)

# спершу вкажемо обробник сигналу
```

```
signal.signal(signal.SIGTERM, handler_term)
...
```

Щоб завершити програму, використовується функція sys.exit() з тим самим кодом завершення 0, що вказує на нормальне завершення роботи без помилок.

Зареєструвати функцію, що виконується при завершенні роботи програми, можна за допомогою atexit.register()

```
import atexit

# реалізуємо функцію, що виконується при завершенні
def gpioExit():
        pwmStop()
        gpioDeInit()

# спершу зареєструємо функцію, що викликатиметься при завершенні роботи
atexit.register(gpioExit)
...
```

Порядок виконання роботи

1. Запустити Geany IDE. Створити новий файл main.cpp та зберегти в файловій системі.

2. Написати програму, що під час своєї роботи періодично записуватиме в файл числа, а в кінці файлу записуватиме рядок-маркер кінця файлу “Кінець файлу”.

```
#include <iostream>
#include <fstream>
#include <unistd.h>

int main()
{
   std::ofstream out;    // створюємо потік для запису
   out.open("file.txt"); // відкриваємо файл для запису
   if (out.is_open())
   {
         std::cout << "Файл відкрито" << std::endl;
```

```
        for(int i=1; i<=30; i++) {
                std::cout << "Записую " << i << std::endl;
                out << i << std::endl;
                sleep(1);    //затримка 1 с
        }
        std::cout << "Закриваю файл" << std::endl;
        out << "Кінець файлу";
        out.close();
    }
    return 0;
}
```

Скомпілювати програму, зібрати виконуваний файл, запустити на виконання й перевірити, що без зовнішніх сигналів програма записує всі числа в файл і ставить маркер кінця.

3. Додати в програму функцію, яка виконуватиметься при завершенні, за допомогою atexit(). Перенести в цю функцію всі операції з додаванням маркера кінця й закриттям файлу. Налагодити програму, щоб повідомлення про закриття файлу виводилось лише раз.

4. Додати обробку сигналів SIGINT, SIGHUP, SIGTERM таким чином, щоб при завершенні процесу ними виконувались всі ті самі дії, що й при закінченні по досягненню максимального числа (повідомлення про закриття файлу, додавання маркера кінця й закриття файлу).

5. Отриману програму запустити в QTerminal чи іншому емуляторі термінала. Завершити її натисканням Ctrl+C до досягнення максимального числа. Перевірити наявність маркера кінця файлу в file.txt.

6. Знову запустити програму, але завершити її закриттям вікна емулятора термінала. Перевірити наявність маркера кінця файлу в file.txt.

7. Відкрити емулятор термінала та qps або інший системний монітор/диспетчер задач. В останньому відобразити дерево процесів (ієрархію батьківський-дочірній процес), знайти процес емулятора термінала. Запустити програму, як в попередньому пункті, проте завершити її, обравши Terminate/Завершити в контекстному меню процесу запущеної програми. Перевірити наявність маркера кінця файлу в file.txt.

8*. Аналогічну програму написати мовою Python і перевірити її поведінку при завершенні, ініційованому сигналами SIGINT, SIGHUP, SIGTERM. У випадку помилки “python not found” в пункті “Команди збірки” вказати замість *python* на початку *python3*.

9. Оформити звіт, що повинен містити титульну сторінку, тему, мету роботи, порядок виконання, тексти програм, скріншоти їх завершення й вміст file.txt для всіх 3 сигналів.

Контрольні запитання

1. Чому при роботі з апаратною частиною потрібно забезпечити виконання додаткових операцій з ним замість стандартних обробників сигналів завершення?
2. Який сигнал генерується при завершенні програми натисканням Ctrl+C в терміналі?
3. Який сигнал генерується при завершенні (terminate) програми у системному моніторі/диспетчері задач?
4. Який сигнал генерується при завершенні програми закриттям вінка терміналу?
5. Який сигнал не може бути оброблений процесом і аварійно його завершує?
6. Який заголовний файл повинен бути підключений для обробки сигналів?
7. Як задати функцію, що виконується при нормальному завершенні програми?

Лабораторна робота №2. Навігація в терміналі Linux. Створення програми через термінал. Аналіз логів.

Мета роботи: навчитись використовувати термінал для навігації файловою системою, створення та редагування файлів, компіляції та виконання програм. Навчитись використовувати конвеєри.

Теоретичні відомості

Конвеєр — це механізм міжпроцесної взаємодії, що використовує передачу повідомлень, що представляє собою набір поєднаних між собою процесів, які забезпечують передачу даних так, що стандартний вихідний потік кожного процесу (stdout) безпосередньо з'єднується зі стандартним вхідним потоком (stdin) наступного.

Використання конвеєрів дозволяє уникнути створення проміжних файлів для збереження результатів і виклику користувачем окремо кожної програми конвеєра.

Часто конвеєри використовуються разом зі стандартними утилітами для фільтрування великих файлів, наприклад, логів системи й програм. Наприклад, якщо потрібно знайти 5 останніх повідомлень в лозі ядра щодо USB, можна

організувати наступний конвеєр:

cat /var/log/kern.log | grep usb | tail -n 5

Тут cat виведе вміст файлу в стандартний вивід, grep знайде всі рядки, що містять послідовність символів usb, а tail залишить лише 5 останніх рядків, що надійшли йому на стандартний вхід.

Порядок виконання роботи

1. Запустити QTerminal. Створити командою mkdir в домашньому каталозі поточного користувача каталог cprog. Довідку щодо використання команди можна отримати, виконавши *mkdir --help*.

2. Перейти в створений каталог командою *cd*.

3. Створити в ній файл main.cpp редактором nano, виконавши команду *nano main.cpp*. Вставити текст програми з попередньої роботи. Зберегти комбінацією клавіш Ctrl+O та вийти Ctrl+X.

4. Скомпілювати отриманий файл командою *g++ -Wall -o main main.cpp*. Вивести список файлів у поточному каталозі командою *ls*.

5. Визначити тип отриманого файлу main, використавши команду file.

6. Виконати файл, ввівши *./main*

7. Забрати права на виконання файлу в усіх користувачів з використанням команди chmod. Спробувати знову виконати файл.

8. Повернути права на виконання. Знову виконати файл.

9. Видалити file.txt, створений програмою main, виконавши *rm file.txt*.

10. Відредагувати файл main.cpp за допомогою nano, змінивши маркер кінця файлу file.txt на інший, й зберегти файл з джерельним кодом зміненої програми з новим ім’ям test.cpp.

11. Скомпілювати змінену програму й запустити в фоновому режимі командою *./test &*

12. Переглянути список процесів у поточному терміналі командою ps. Знайти ідентифікатор процесу *test*.

13. Завершити процес test, використавши команду kill (за ідентифікатором процесу) або killall (за ім'ям процесу).

14. Перемістити виконуваний файл в каталог ~/Документи за допомогою команди *mv*.

15. Перейти в робочий каталог ~/Документи.

16. Запустити файловий менеджер mc, за її допомогою перемістити її назад в каталог ~/cprog.

17. Створити в домашньому каталозі користувача каталог pyprog, в ньому новий файл main.py. Написати програму, яка функцією print виводитиме “Hello, world”. Зберегти зміни.

18. Запустити написаний скрипт *python3 main.py*

19. Отримати 5 останніх повідомлень в лозі ядра щодо USB конвеєром, вказаним в теоретичних відомостях.

20. Організувати конвеєр, який виведе всі повідомлення ядра, що містять Vendor ID та Product ID обладнання.

21. Аналогічним чином знайти в syslog перші 10 рядків, які містять wpa_supplicant.

22. Оформити звіт, що повинен містити титульну сторінку, тему, мету роботи, порядок виконання, тексти програм, скріншоти їх завершення й вміст file.txt для всіх 3 сигналів, результати виконання конвеєрів.

Контрольні запитання

1. Для чого призначена команда ls?
2. Для чого призначена команда cd?
3. Для чого призначена команда rm?
4. Для чого призначена команда mv?
5. Для чого призначена команда mkdir?
6. Для чого призначена команда chmod?
7. Для чого призначена команда grep?
8. Для чого призначена команда tail?
9. Для чого призначена команда head?
10. В якому каталозі зберігаються лог-файли?
11. Як скомпілювати програму, використовуючи термінал?
12. Як виконати файл, використовуючи термінал?

Лабораторна робота №3. Налаштування мережі в Linux. Використання SSH.

Мета: навчитись виводити на екран та змінювати налаштування мережевих підключень в ОС на базі GNU/Linux; навчитись використовувати протокол SSH для віддаленого доступу й передачі файлів.

Теоретичні відомості

Для встановлення віддаленого підключення до контролера робота він має бути підключений до мережі. Для налаштування та керування мережевими інтерфейсами можуть використовуватись утиліти *ifconfig, arp, iwconfig, iw* або більш нова *ip*, яка здатна замінити дві перші. Для встановлення підключення до WiFi-мереж часто використовується *wpa_supplicant*. Верхнім рівнем у керуванні мережевою підсистемою часто є Network Manager, для якого розроблений графічний інтерфейс та інтерфейс командного рядка *nmcli* (останній дозволяє налаштувати мережу в конфігураціях без дисплея).

SSH (Secure Shell) - це криптографічний мережевий протокол для безпечного забезпечення зв'язку між двома пристроями через небезпечну мережу, таку як Інтернет. SSH використовується для забезпечення безпеки доступу до віддалених пристроїв, забезпечення захищеної передачі даних та виконання команд на віддаленому пристрої.

SSH забезпечує шифрування даних, які передаються між пристроями, та автентифікацію користувача з використанням публічних та приватних ключів. Це забезпечує високий рівень безпеки, оскільки навіть якщо зловмисник зможе перехопити комунікацію між пристроями, він не зможе розшифрувати передані дані.

SSH використовується в багатьох сферах, таких як адміністрування віддалених серверів, резервне копіювання даних, віддалений доступ до комп'ютерів та інше.

Порядок виконання роботи

1. Вивести налаштування мережі власного ПК командою *ip a*. Знайти й виділити в виводі команди всі мережеві інтерфейси, їх назви, MAC-адреси, IP-адреси, маски мережі, широкомовні адреси й занести у звіт.

2. Використовуючи *iwconfig*, вивести інформацію про бездротові підключення. Занести у звіт.

3. Дізнатись ім’я та IP-адресу віддаленого комп'ютера, до якого надалі здійснюватиметься віддалений доступ.

4. Використовуючи *ping*, перевірити наявність зв'язку з віддаленим комп'ютером (наприклад, *ping aspirea315.local* або *ping 10.1.134.4*).

5. Під’єднатись до віддаленого комп'ютера за допомогою SSH командою на зразок *ssh student@10.1.134.4*. Для успішного підключення на віддаленому комп'ютері повинен бути встановлений та запущений SSH-сервер.

6. Створити на віддаленому комп'ютері текстовий файл *main.cpp* з попередньої роботи. Скомпілювати та виконати її на віддаленому комп'ютері. Занести скріншот у звіт.

7. Аналогічно створити і виконати *main.py*. Занести скріншот у звіт.

8. Відключитись командою *exit*. Підключитись з опцією *-Y* (підтримка тунелювання графічної підсистеми). Запустити програму *geany*. Перевірити коректність роботи, створивши і запустивши програму з п. 6. Занести скріншот у звіт.

9. Змінити порт SSH на нестандартний. Для цього знайдіть відповідний параметр в /etc/ssh/sshd_config, вкажіть інше число та розкоментуйте. Збережіть зміни й перезапустіть SSH-сервер командою *sudo systemctl restart ssh*.

10. Підключіться до сервера через новий порт, вказавши якого номер за допомогою ключа *-p*. Занести скріншот у звіт.

11. Скопіюйте на локальний ПК в каталог */tmp* файл налаштувань *sshd_config* за допомогою програми scp: *scp student@10.1.134.4:/etc/ssh/sshd_config /tmp.* Вставте вміст файлу з локального ПК у звіт.

12. Скопіюйте на віддалений комп'ютер з локального будь-який графічний файл. Виведіть командою *ls* вміст каталогу з цим файлом на віддаленому комп'ютері й занесіть у звіт.

13. Оформити звіт, що повинен містити титульну сторінку, тему, мету роботи, порядок виконання, текст файлу налаштувань, скріншоти, висновок.

Контрольні запитання

1. Як підключитись до віддаленого вузла по SSH?
2. Що потрібно вказати для тунелювання графічного інтерфейсу X11 через SSH?
3. Як налаштувати нестандартний порт SSH?
4. В чому різниця між SSH-входом з використанням пароля і ключа?
5. Як через SSH скопіювати файл з локального вузла на віддалений?
6. Як через SSH скопіювати файл з віддаленого вузла на локальний?
7. Як дізнатись IP-адресу вузла через термінал в Linux?

Лабораторна робота №4. SSH-підключення до Raspberry Pi. Створення програми для цифрового вводу-виводу

Мета роботи: навчитись виконувати SSH-підключення до одноплатних комп'ютерів; навчитись розробляти програми для цифрового вводу-виводу з використанням GPIO одноплатних комп'ютерів, що працюють під керуванням ОС на базі GNU/Linux.

Теоретичні відомості

Для використання портів вводу-виводу загального призначення GPIO ядро Linux повинне бути скомпільовано з активованою їх підтримкою для потрібної плати/процесора/SoC.

Щоб взаємодіяти з портами вводу-виводу загального призначення GPIO з власної прикладної програми, як правило, використовують бібліотеки, що взаємодіють з відповідним API ядра Linux. Однією з поширених таких бібліотек для одноплатного комп'ютера Raspberry Pi є WiringPi.

Її можна отримати, завантаживши вміст її Github-репозиторію командою *git clone https://github.com/WiringPi/WiringPi.git ~/wiringpi*

Для початку потрібно підключити бібліотеку:

#include <wiringPi.h>

Першим кроком має бути ініціалізація. На цьому етапі визначається, яку схему нумерації пінів ви використовуватимете у програмі. Вкажіть один із цих викликів функцій, щоб ініціалізувати бібліотеку:

wiringPiSetup(); // ініціалізація з використанням спрощеної нумерації GPIO

wiringPiSetupGpio(); // ініціалізація з використанням нумерації Broadcom GPIO

У схемі нумерації Broadcom номери пінів відповідають порядку виводів GPIO на процесорі (SoC). Спрощена нумерація GPIO використовує номера від 0 до 16 для ідентифікації контактів. Побачити нумерацію фізичних контактів у різних схемах можна, виконавши команду *gpio readall* (рис. 1).

```
+-----+-----+---------+------+---+---Pi 4B--+---+------+---------+-----+-----+
| BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
+-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
|     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
|   2 |   8 |   SDA.1 | ALT0 | 1 |  3 || 4  |   |      | 5v      |     |     |
|   3 |   9 |   SCL.1 | ALT0 | 1 |  5 || 6  |   |      | 0v      |     |     |
|   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 1 | ALT5 | TxD     | 15  | 14  |
|     |     |      0v |      |   |  9 || 10 | 1 | ALT5 | RxD     | 16  | 15  |
|  17 |   0 | GPIO. 0 |   IN | 0 | 11 || 12 | 0 | IN   | GPIO. 1 | 1   | 18  |
|  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
|  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 1 | OUT  | GPIO. 4 | 4   | 23  |
|     |     |    3.3v |      |   | 17 || 18 | 1 | OUT  | GPIO. 5 | 5   | 24  |
|  10 |  12 |    MOSI | ALT0 | 0 | 19 || 20 |   |      | 0v      |     |     |
|   9 |  13 |    MISO | ALT0 | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
|  11 |  14 |    SCLK | ALT0 | 0 | 23 || 24 | 1 | OUT  | CE0     | 10  | 8   |
|     |     |      0v |      |   | 25 || 26 | 1 | OUT  | CE1     | 11  | 7   |
|   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
|   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
|   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
|  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
|  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
|  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
|     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29  | 21  |
+-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
| BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
+-----+-----+---------+------+---+---Pi 4B--+---+------+---------+-----+-----+
```

Рис. 1. Вивід команди gpio readall з нумерацією контактів

Далі слід налаштувати режим роботи GPIO на вхід, вихід або ШІМ-вихід. Це можна зробити функцією *pinMode([pin], [mode])*. *mode* може бути *INPUT, OUTPUT* або *PWM_OUTPUT* відповідно. Наприклад, pinMode(18, PWM_OUTPUT).

Цифровий вихід

Функцію *digitalWrite([pin], [HIGH/LOW])* можна використовувати для встановлення високого або низького рівня на цифровому виході. Функція

аналогічна такій в Arduino.

Наприклад, щоб встановити вивід 23 як HIGH, можна викликати:

digitalWrite(23, HIGH);

ШІМ-вихід

Для окремих контактів, які підтримують апаратний ШІМ можна використовувати функцію *pwmWrite([pin], [0-1023])*, яка встановить коефіцієнт заповнення ШІМ-сигналу від 0 до 100%, що відповідає другому аргументу функції від 0 до 1023. Наприклад, *pwmWrite(18, 723)* встановить на контакті 18 коефіцієнт заповнення ШІМ-сигналу близько 70%.

Цифровий вхід

Зчитувати цифровий вхід можна функцією *digitalRead([pin])*. Наприклад,

```
if (digitalRead(17)==HIGH)
   printf("Pin 17 is HIGH\n");
else
   printf("Pin 17 is LOW\n");
```

виведе стан контакту 17.

Підтягуючі резистори

До цифрових входів можуть бути підключені вбудовані підтягуючі резистори (активована підтяжка до землі або до напруги живлення). Це забезпечує функція *pullUpDnControl([pin], [PUD_OFF, PUD_DOWN, PUD_UP])*, другим аргументом якої є вид підтяжки: вимкнута, до землі, до напруги живлення відповідно.

Наприклад, якщо до контакту 22 підключена кнопка і потрібно підтягнути її до напруги живлення:

pullUpDnControl(17, PUD_UP);

Затримки

WiringPi включає дві функції затримки: *delay([milliseconds])* і *delayMicroseconds([microseconds])*, прототипи яких аналогічні Arduino. Звичайна затримка delay() призупиняє виконання програми на певну кількість мілісекунд. Якщо потрібно відкласти, наприклад, на 2 секунди, виконання дії, використовується *delay(2000).*

Порядок виконання роботи

1. Перевірити доступність Raspberry Pi командою *ping ri41rpi.local*
2. Підключитись до Raspberry Pi командою *ssh student@ri41rpi.local*
3. Створити каталог з вашим прізвищем.
4. Склонувати репозиторій https://github.com/WiringPi/WiringPi.git у створений каталог.

5. Перейти в каталог wiringPi та скомпілювати бібліотеку WiringPi командою *./build*

6. Відкрити приклад Blink, розібрати ключові елементи програми.

7. Скомпілювати приклад компілятором gcc, вказавши додатково ключ -lwiringPi, щоб використати отриману бібліотеку. Запустити програму й перевірити стан світлодіода.

8. Відповідно до порядкового номеру в групі розробити програму для періодичного включення-виключення навантаження на вказаному контакті (таблиця 2).

Таблиця 2

Варіанти завдань для лабораторної роботи 4

Варіант	Номер фізичного контакту	Номер GPIO в WiringPi
1	29	21
2	31	22
3	33	23
4	35	24
5	37	25
6	11	0
7	7	17
8	22	6
9	38	28
10	3	8

9. Додати у програму код з лабораторної роботи 1, який гарантовано вимкне навантаження при завершенні програми по Ctrl+C, сигналу SIGTERM або SIGHUP (закриття терміналу/втрата зв'язку).

10. Знайти одну з кнопок на CrowPi та визначити номер контакту/GPIO, на який надходить сигнал з неї. Модифікувати програму так, щоб світлодіод блимав лише при натиснутій кнопці.

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

Контрольні запитання

1. Як підключитись до віддаленого вузла по SSH за його іменем?
2. Яка бібліотека може використовуватись для взаємодії з GPIO плати Raspberry Pi 4?
3. Яка функція WiringPi дозволяє задати режим нумерації контактів плати Raspberry Pi?
4. Як налаштувати контакт як цифровий вхід?
5. Як налаштувати контакт як цифровий вихід?
6. Як налаштувати контакт як ШІМ-вихід?
7. Якою функцією можна зчитати стан цифрового входу?
8. Якою функцією можна змінити стан цифрового виходу?
9. Якою функцією можна вивести ШІМ-сигнал 50%?
10. Як можна увімкнути вбудований підтягуючий резистор до напруги живлення?
11. Як можна увімкнути вбудований підтягуючий резистор до землі?

Лабораторна робота №5. Керування сервоприводом за допомогою програмного ШІМ на мові C++.

Мета роботи: навчитись керувати сервоприводом SG-90 з плати Raspberry Pi, використовуючи програмний ШІМ.

Теоретичні відомості

Під сервоприводом найчастіше розуміють механізм з електродвигуном, якому можна вказати повернутися в заданий кут і утримувати це положення. Якщо сказати повніше, сервопривід - це привід з управлінням через негативний зворотний зв'язок, що дозволяє точно керувати параметрами руху (рис. 2). 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. 3). Тобто сервопривод отримує на вхід значення керуючого параметра, 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.

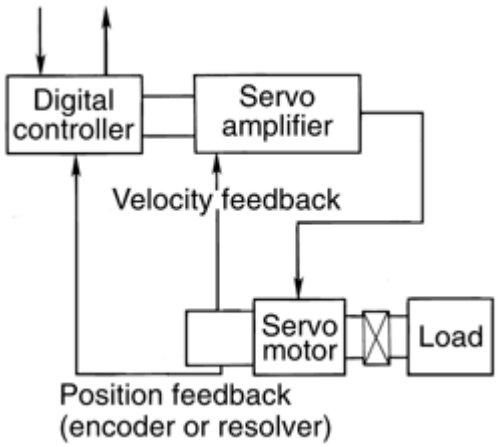

Рис. 2. Структурна схема сервопривода з давачем швидкості обертів

Рис. 3. 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. 4).

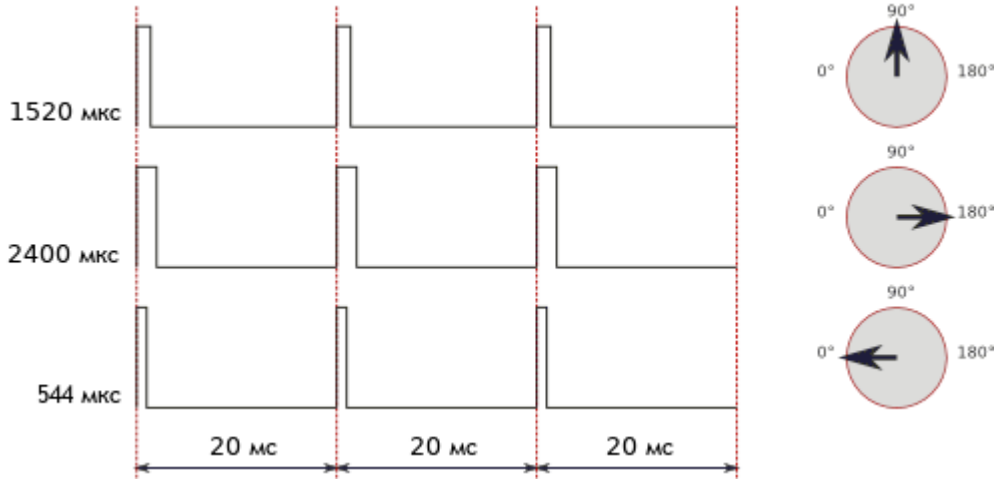

Рис. 4. Залежність кута повороту сервопривода від трвалості керуючих імпульсів

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.

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. Видалити ServoBlaster командою *sudo make uninstall*.

9. 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. Керування сервоприводом за допомогою програмного ШІМ на мові Python.

Мета робота: навчитись керувати сервоприводом SG-90 з плати Raspberry Pi, використовуючи програмний ШІМ на мові Python.

Теоретичні відомості

Аналогічно C++ у попередній лабораторні роботі в Python можна використовувати функції бібліотеки WiringPi й ServoBlaster.

Приклад програми на мові Python, яка використовує ServoBlaster, наведений нижче.

```
from tkinter import *
import os
import time

servo_min = 800  # uS
servo_max = 2100  # uS

servo = 2 # GPIO 18

def map(value, from_low, from_high, to_low, to_high):
  from_range = from_high - from_low
```

```
  to_range = to_high - to_low
  scale_factor = float(from_range) / float(to_range)
  return to_low + (value / scale_factor)

def set_angle(angle):
  pulse = int(map(angle, 0, 180, servo_min, servo_max))
  command = "echo {}={}us > /dev/servoblaster".format(servo, pulse)
  os.system(command)

class App:

    def __init__(self, master):
        frame = Frame(master)
        frame.pack()
        scale = Scale(frame, from_=0, to=180,
              orient=HORIZONTAL, command=self.update)
        scale.grid(row=0)

    def update(self, angle):
        set_angle(float(angle))

root = Tk()
root.wm_title('Servo Control')
app = App(root)
root.geometry("200x50+0+0")
root.mainloop()
```

Порядок виконання роботи

1. Виконати SSH-підключення до Raspberry Pi.

2. Створити нову програму, що використовуватиме бібліотеку WiringPi (https://github.com/WiringPi/WiringPi-Python) 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 функції *softPwmWrite(), atexit.register(), signal.signal().*

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. Аналогічну програму розробити, використовуючи RPi.GPIO (https://pypi.org/project/RPi.GPIO/). Перевірити, чи змінилась стабільність положення сервоприводу.

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 мовою Python, що керує сервоприводом з використанням демона servod. Для передачі на пристрій /dev/servoblaster використати os.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. Видалити ServoBlaster командою *sudo make uninstall*.

9. 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 №7. Пошук об’єктів у відеопотоці пороговою фільтрацією 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. 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?

Лабораторна робота №8. Обмін повідомленнями між програмами відстеження об'єкта і керування сервоприводом.

Мета роботи: навчитись передавати повідомлення між комп'ютером і платою керування сервоприводами засобами міжпроцесної взаємодії в GNU/Linux.

Теоретичні відомості

Часто в функціональність обробки відеопотоку й функціональність обміну даними через інтерфейс роздіялють в окремі програми, оскільки програма, що реалізує аналіз відеопотоку в реальному часі, споживає багато обчислювальних ресурсів. В лабораторній роботі передбачається обробка відеопотоку з камери чи файлу на комп’ютері та надсилання повідомлень іншій програмі, яка забезпечує надсилання команд про зміну положення сервопривода платі Arduino з використанням протоколу Modbus, реалізованому в бібліотеці libmodbus.

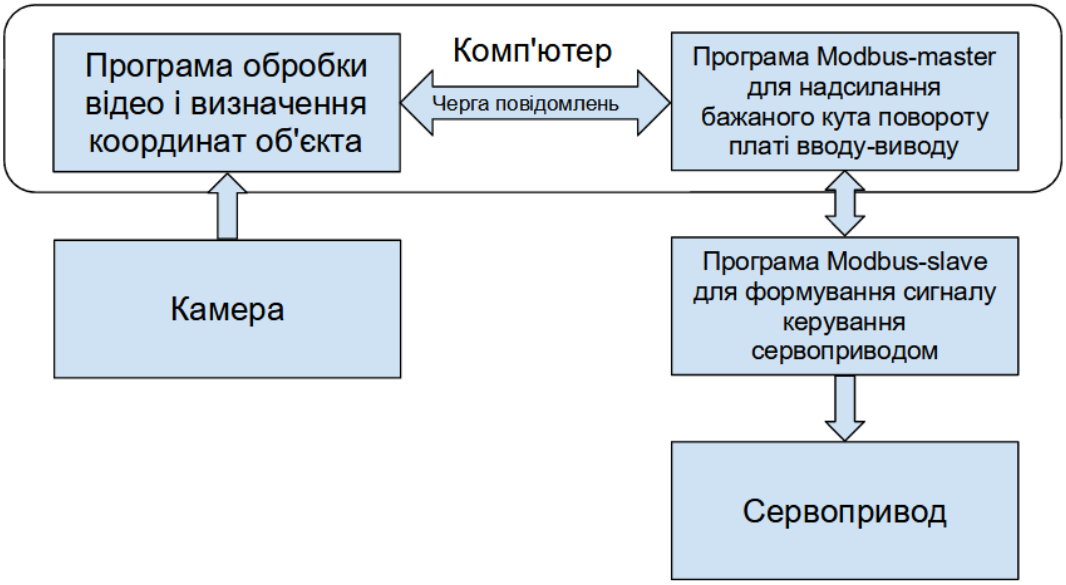

Рис. 5. Схема взаємодії компонентів у системі камера-комп’ютер-Arduino-сервопривод

При розробці програми мовою C++ потрібно підключити заголовні файли відповідних бібліотек

#include "opencv2/imgproc.hpp"

#include "opencv2/highgui.hpp"

#include "opencv2/videoio.hpp"

#include <modbus.h>

Задати адреси пристроїв та регістрів можна макросами або константами

#define HMI_SLAVE_ADDR 8 //Modbus-адреса

#define POSITION_REG 0 //регістр Modbus з положенням сервопривода

Протокол Modbus передбачає адресацію 16-бітними регістрами, тому для зберігання отриманих/відправлених даних за можливості краще використовувати тип uint16_t

*modbus_t *ctxhmi;*

*uint16_t *hmiRegisters;*

При створенні конексту потрібно вказати порт, швидкість та формат передачі.

ctxhmi = modbus_new_rtu("/dev/ttyUSB0", 9600, 'N', 8, 1);

if(ctxhmi == NULL) {

fprintf(stderr, "Unable to create the libmodbus context: %s\n",modbus_strerror(errno));

```
return -1;
}
```

Таймаут очікування відповіді від веденого пристрою задають з використанням struct timeval

```
modbus_set_debug(ctxhmi, TRUE);
modbus_set_slave(ctxhmi, HMI_SLAVE_ADDR);
struct timeval response_timeout;
response_timeout.tv_sec = 0;
response_timeout.tv_usec = 500000;
modbus_set_response_timeout(ctxhmi, &response_timeout);
```

Після чого можна виконувати підключення

```
modbus_connect(ctxhmi);
hmiRegisters = (uint16_t *) malloc(7 * sizeof(uint16_t));
memset(hmiRegisters, 0, 7 * sizeof(uint16_t));
sleep(3);
```

Остання затримка на 3 с потрібна, щоб дати завантажувачу Arduino вийти з режиму очікування прошивки.

Після запуску основного циклу обробки відеопотоку кожен кадр, як правило, містить шуми. Оскільки планується відстеження положення порівняно великого об’єкта, вплив шумів можна зменшити згладжуванням

```
blur(frame,frame,Size(10,10));
```

Часто виділення об’єктів у кадрі зручніше робити в колірній моделі HSV. Виокремити об’єкти з насиченістю кольору вище 198 можна так

```
cvtColor(frame,frame_HSV,COLOR_BGR2HSV);
split( frame_HSV, hsv_planes );
threshold( hsv_planes[1], dst, 198, 255, THRESH_BINARY );
```

Вивести користувачу знайдені об’єкти можна, показавши їх контури

```
findContours( dst, contours, hierarchy, RETR_TREE, CHAIN_APPROX_SIMPLE );
for( size_t i = 0; i < contours.size(); i++ )
 {
    Scalar color = Scalar( 255, 0, 0 );
    drawContours( frame, contours, (int)i, color, 2, LINE_8, hierarchy, 0 );
 }
```

Кожен контур містить масив точок. Наприклад, координату x початкової точки першого контуру можна отримати

```
x=contours[0][0].x;
```

Записати значення в регістр іншого веденого Modbus-пристрою можна

функцією

modbus_write_register(ctxhmi,POSITION_REG,angle);

За умови встановлених бібліотек при використанні компілятора GCC скопілювати програму можна командою

g++ -o main main.cpp -I/usr/include/modbus `pkg-config opencv --cflags --libs` -lmodbus

Порядок виконання роботи

1. Програму з попередньої роботи модифікувати так, щоб в окрему змінну записувалась горизонтальна координата центра найбільшого знайденого в кадрі об'єкта.

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

3. Використовуючи бібліотеку ModbusSlave, розробити програму для Arduino, що вміст першого (нульового) регістра зберігання (адреса 40001 в даній бібліотеці) інтерпретує як кут повороту сервопривода і пересилає відповідний імпульс керування на сервопривод, підключений до 9-го контаткту плати Arduino Uno. Адреса пристрою – 8. Для керування сервоприводом використати бібліотеку Servo.

```
#include <modbus.h>
#include <modbusDevice.h>
#include <modbusRegBank.h>
#include <modbusSlave.h>
#include <Servo.h>
#define HMI_SLAVE_ADDR 8
modbusDevice regBank;
modbusSlave slave;
Servo s1;
void setup()
{
  s1.attach(9);
  pinMode(13,OUTPUT);
  regBank.setId(HMI_SLAVE_ADDR);
  regBank.add(40001);
  slave._device = &regBank;
  slave.setBaud(9600);
  regBank.set(40001,1); //запис 1 в регістр 40001
}
```

```
void loop()
{
 slave.run(); //отримання Modbus-запитів, надсилання у відповідь значення регістра або його модифікація
 int angle=regBank.get(40001); //зчитуємо поточне значення регістра й надсилаємо на сервопривод з врахуванням обмежень
 if(angle>180) angle=180;
 s1.write(angle);
}
```

4. Використовуючи бібліотеку libmodbus (https://libmodbus.org/download/), підключитись за протоколом Modbus RTU та пересвідчитись, що значення, які надсилаються з ПК, Arduino коректно отримує та змінює положення сервоприводу.

5. Додати код програми з libmodbus до програми, написаної в п.2 порядку виконання роботи так, щоб змінна, пов'язана з горизонтальною координатою об'єкта, передавалась з використанням протоколу Modbus RTU на Arduino Uno. Запустити отриману програму й переконатись, що при переміщенні об'єкта в полі зору камери або на відео сервопривод змінює своє положення.

6*. Розділити програму для комп'ютера на ресурсоємну програму пошуку об'єкта і визначення координати та програму, що відповідає за обмін даними через Modbus RTU. Обмін даними між програмами організувати за допомогою черги повідомлень.

7. 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. Яка бібліотека може бути використана для обміну даними за протоколом Modbus RTU в ОС на базі GNU/Linux?

2. Яка бібліотека може бути використана для обміну даними за протоколом Modbus RTU в Arduino IDE?

3. Який формат фрейму передбачає протокол Modbus RTU?

4. Як адресуються регістри при використанні протоколу Modbus RTU?

5. Як задається операція (читання регістра, запис групи регістрів тощо) в Modbus RTU?

6. Яка функція використовується для створення черги повідомлень в GNU/Linux?

7. Яка функція використовується для надсилання повідомлення в чергу?

8. Яка функція використовується для отримання повідомлення з черги?

Лабораторна робота №9. 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* . 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 №10. 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 №11. Передача даних про положення в Robot Operating System і їх візуалізація. Керування гусеничною латформою

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. 6.

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

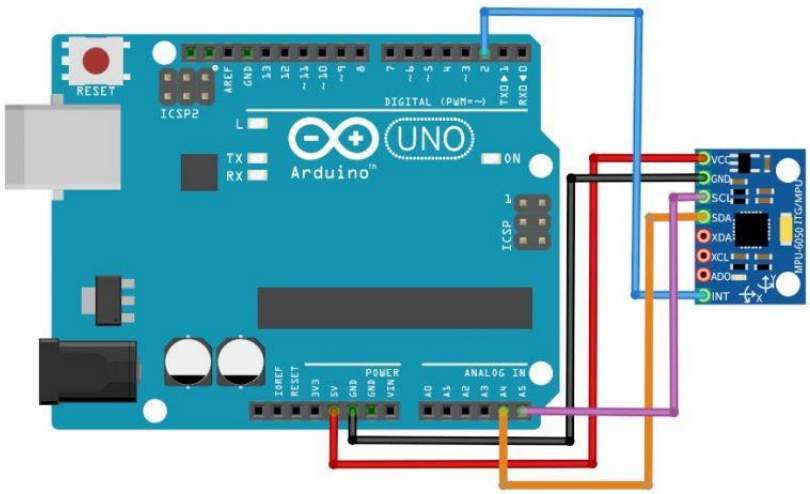

Рис. 6. Схема підключення MPU6050 до Arduino Uno

15. У запущеному Rviz перевірити реакцію на переміщення модуля з MPU6050 (рис. 7).

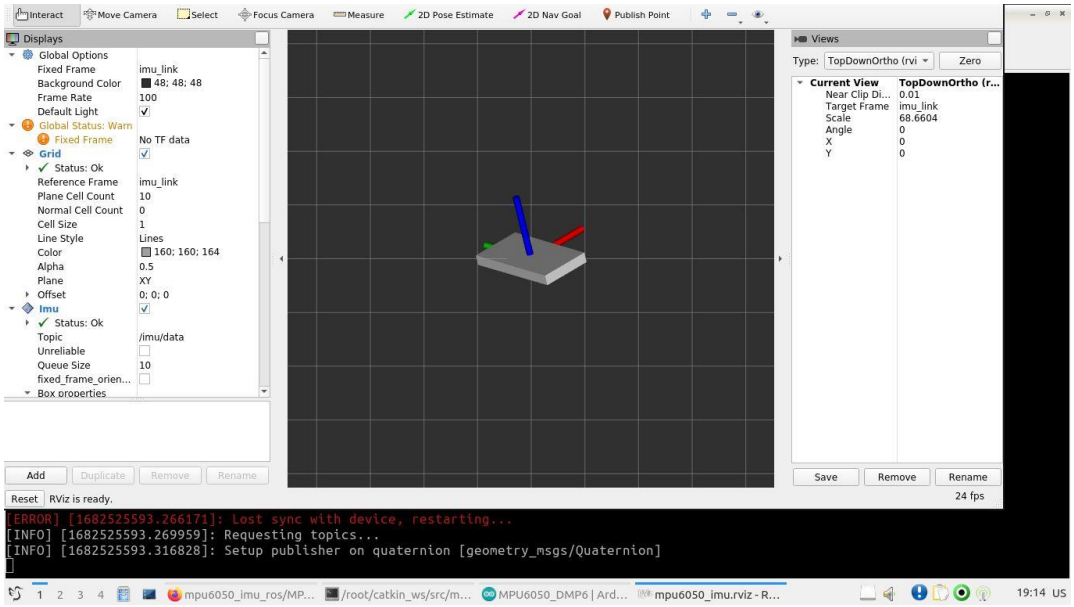

Рис. 7. Візуалізація поточного положення, визначеного акселерометром-гіроскопом MPU6050, в Rviz

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[] = {0x00,0x00,0x00,0x00,0x00,0x24,0x48,0x90,0x48,0x24,0x00,0x00,0x00,0x00,0x00,0x00};
unsigned char left[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x44,0x28,0x10,0x44,0x28,0x10,0x44,0x28,0x10,
```

```
0x00};
    unsigned char right[] =
{0x00,0x10,0x28,0x44,0x10,0x28,0x44,0x10,0x28,0x44,0x00,0x00,0x00,0x00,0x00,
0x00};
    unsigned char STOP01[] =
{0x2E,0x2A,0x3A,0x00,0x02,0x3E,0x02,0x00,0x3E,0x22,0x3E,0x00,0x3E,0x0A,0x0
E,0x00};
    unsigned char clear[] =
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
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. 8.

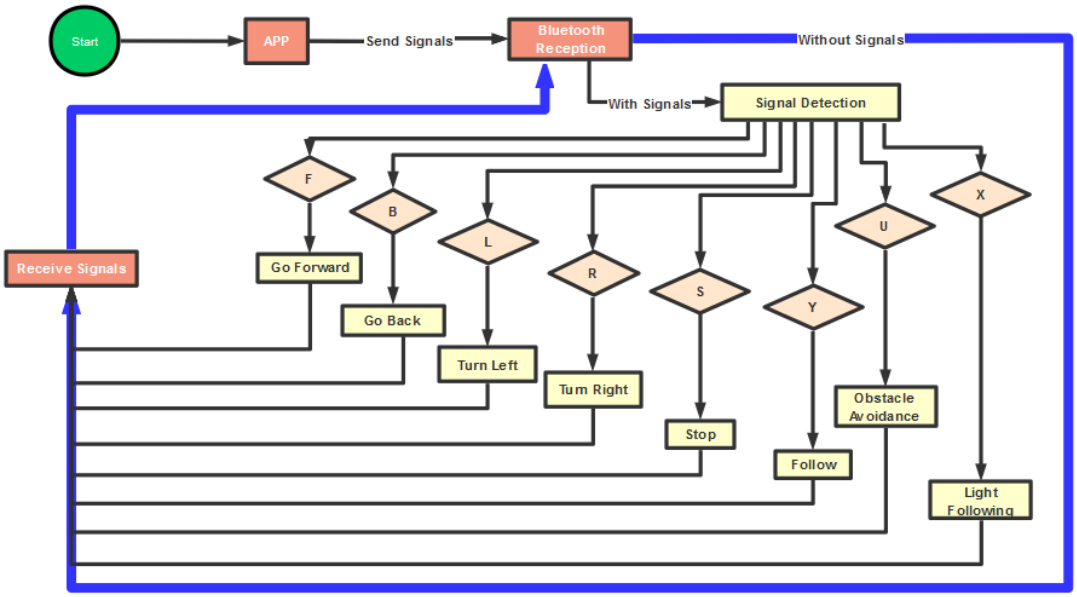

Рис. 8. Блок-схема алгоритму програми робота

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?

Raspberry Pi.

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

Теоретичні відомості

Рис. 9. Гусеничний робот XiaorGeek

Керування гусеничним роботом з Raspberry Pi (рис. 9) 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. 10). 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.

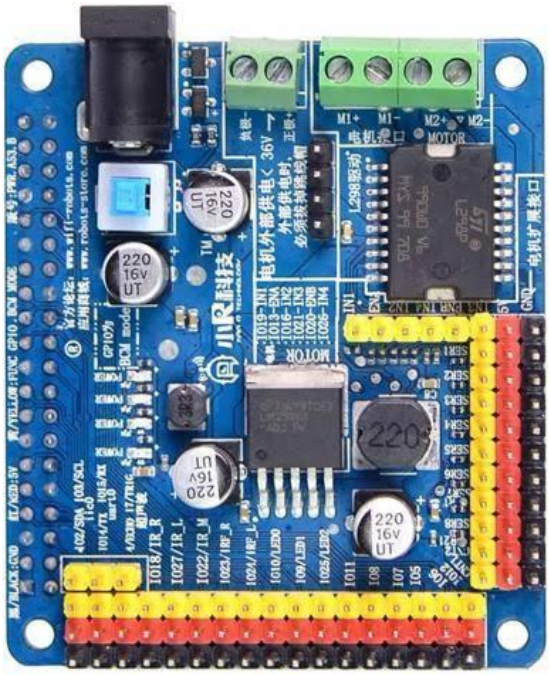

Рис. 10. Плата для керування приводами робота

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.

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.