«ВІРТУАЛЬНА РУКАВИЦЯ»

«РОЗРОБЛЕННЯ РУКАВИЦІ ВІРТУАЛЬНОЇ РЕАЛЬНОСТІ»

ЗМІСТ

Вступ…… 3
Розділ 1. АНАЛІЗ СУЧАСНИХ ПРИСТРОЇВ КЕРУВАННЯ ЕЛЕКТРОННИМИ ЗАСОБАМИ ...……... 4
Розділ 2. ВИБІР КОМПОНЕНТІВ ДЛЯ ПРИСТРОЮ КЕРУВАННЯ ВІРТУАЛЬНОЮ РЕАЛЬНІСТЮ.……..……... 8
2.1. Розроблення структурної та функціональної схеми …… 8
2.2. Особливості функціональних вузлів …….……... 11
Розділ 3. РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ РУКАВИЦІ ВІРТУАЛЬНОЇ РЕАЛЬНОСТІ ……...……. 15
3.1 Розроблення вбудованого програмного забезпечення…… 15
3.1.1. Середовище розробки……..15
3.1.2. Програмний код для контролера ATMega328P ..……..16
3.2. Розроблення програмного коду для драйвера …… 23
3.3. Практична реалізація рукавиці віртуальної реальності ……. 27
Висновки ……...……...……..... 29
Список використаної літератури ……. 30
Додаток А. Алгоритм роботи пристрою керування рукавицею віртуальної реальності …….. 33
Додаток Б. Структурна схема алгоритму коду для драйвера ……... 34
Анотація …… 35

ВСТУП

На сьогоднішній день широкого застосування набули різноманітні електронні пристрої керування електронними засобами та комп’ютерною технікою, такі як оптичні мишки, джойстики, графічні планшети тощо. Дані пристрої дозволяють відстежувати рухи з подальшою можливістю керування приладом, що суттєво спрощують роботу оператора. Більшість сучасних комерційних продуктів віртуальної реальності (VR) покладаються на геймпади або контролери в питанні взаємодії з предметами віртуальної реальності, що в певній мірі є практичними та ефективними у застосуванні. Проте, вони не зовсім ідеальні та не забезпечують отримання відчуття, що і в реальному житті при взаємодії з предметом за допомогою своїх рук та пальців. Тому виникає потреба у розробленні нових систем для відстеження рухів, з допомогою яких можна б було чітко відстежувати положення пальців та руки. Дана наукова робота спрямована на вирішення цього завдання, а саме розроблення рукавиці віртуальної реальності, яка забезпечить зв'язок сигналів руху та зміни їхнього розташування у просторі з можливістю захоплення об'єктів у віртуальній реальності.

РОЗДІЛ 1. АНАЛІЗ СУЧАСНИХ ПРИСТРОЇВ КЕРУВАННЯ ЕЛЕКТРОННИМИ ЗАСОБАМИ

Існує велика кількість різноманітних пристрої керування електронними засобами та комп’ютерною технікою, які виготовлені за різними технологіями, володіють різними характеристики, можливостями та різняться ціною. Розробленням цих пристроїв займаються, як відомі на весь світ, так і маловідомі компанії. На даний час широкого використання набули різноманітні засоби керування – комп’ютерні мишки, джойстики, геймпади, також широкої популярності набувають нові пристрої, такі як контролери VR та пристрої розпізнавання жестів.

Оптична комп'ютерна миша використовує джерело світла - світлодіод та масив фотодіодів, для виявлення руху відносно поверхні [1], що є альтернативою механічної миші, яка використовує рухомі частини під час руху. Сучасні поверхнево-незалежні оптичні миші (рис. 1.1) використовують оптоелектронний сенсор (мініатюрні відеокамери) які послідовно приймають зображення поверхні, на якій працює миша.

Рис. 1.1. Оптична миша GRESSO GM-893 [2].

До переваг оптичної миші можна віднести різноманіття функціональних можливостей та дизайну, великий час роботи, точність визначення зміни позиціонування на поверхні. Недоліками є: виникнення синдрому зап’ястного каналу [3], необхідність рівної поверхні достатніх розмірів, нестійкість до вібрацій. З цієї причині миша практично не використовується у військових пристроях.

Джо́йстик (англ. *joystick*) – пристрій-руків'я керування у відеоіграх (рис.1.2): важіль на підставці, який можна відхиляти у двох площинах. На важелі можуть бути різного роду спускові гачки та перемикачі [4].

Рис.1.2. Джойстик Trust GTX 555.

За кількістю ступенів свободи і, відповідно, площини, у яких можлива зміна стану контрольованого об'єкта, джойстики поділяються на: одномірні (управління переміщенням об'єкта або вгору-вниз, або вліво-вправо); двомірні (управління об'єктом в двох площинах) та тривимірні (управління об'єктом у всіх трьох площинах).

Перевагами джойстиків є: точність переміщення курсора, надійність роботи, невисока ціна та простота виготовлення, невибагливий до поверхні розташування. До недоліків відносять габарити та вагу.

Геймпад – це пристрій введення для двох рук, що застосовується для керування ігровим процесом відеогри або управління пристроєм. Геймпади можуть бути як дротовими та бездротовими (рис. 1.3), універсальними, чи тільки для одного типу пристроїв [5].

До переваг геймпадів відносять можливість управління в трьохмірному просторі, значну кількість додаткових функцій таких як: спускові гачки (в сучасних геймпадах навіть можуть відслідковувати силу натиску), використання жестів за наявності сенсорної панелі [5] та можливість реалізації зворотного зв’язку (Force feedback) [6]. Недоліками згаданих геймпадів є висока ціна [7], обмежені можливості їх застосування (виключно для використання в іграх) та нереалістичність відчуття взаємодії.

Рис 1.3. Геймпад PS4 [7].

VR контролери – пристрій для керування у відеоіграх, може точно відстежувати положення рук в реальному масштабі (рис. 1.4). В грі контролер може набути форми ліхтарика, меча або іншого пристосування, задуманого розробником.

Рис.1.4. VR контролери PS Move.

Перевагами VR контролера є можливість відстеження положення рук в реальному масштабі та зручність використання як меч або ліхтарик у грі. До недоліків відносять обмеженість у використанні (тільки як меч, ліхтарик тощо) та призначення для використання виключно в іграх.

Розпізнавання жестів. Розпізнавання жестів являє собою перетворення людських жестів з допомогою математичних алгоритмів для управління пристроєм (рис. 1.5). Жести створені рухами тіла або обличчя [8] можна використати для управління та взаємодії з пристроєм без фізичного дотику до них.

Перевагами пристроїв з можливістю керування ними за допомогою жестів є відсутність контакту з пристроєм та необхідності додаткових пристроїв взаємодії, у випадку керування автомобілем не відволікає водія на пошук відповідних клавіш. Недоліки пристроїв на основі розпізнання жестів є їхня ціна, складність алгоритмів роботи та недосконаліть технології.

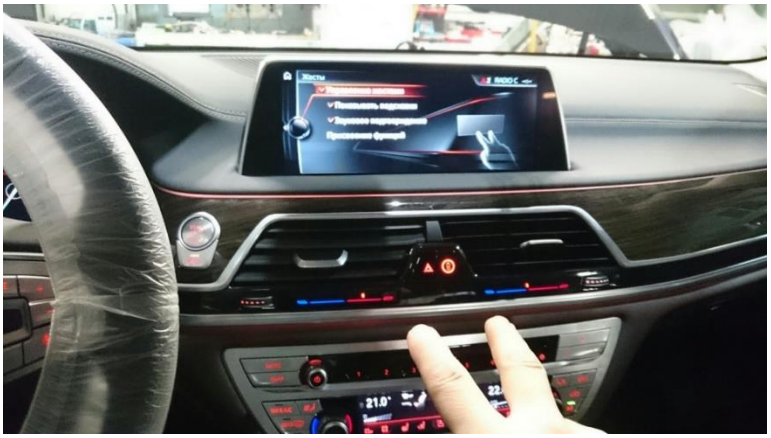


Рис.1.5. Управління медіа файлами за допомогою жестів в автомобілі.

Отже, існуючі пристрої керування електронними засобами та комп'ютерною технікою мають обмежені функціональні можливості та високу ціну, що потребує їхнього удосконалення. Постає завдання розроблення пристрою керування віртуальною реальністю, який реагуватиме на рухи пальців та руки у просторі для забезпечення реалістичних відчутті під час його використання.

РОЗДІЛ 2. ВИБІР ЗАСОБІВ ДЛЯ ПРИСТРОЮ КЕРУВАННЯ ВІРТУАЛЬНОЇЮ РЕАЛЬНІСТЮ

2.1 Розроблення структурної та функціональної схеми

В результаті проведеного аналізу встановлено основні вимоги до структурних елементів пристрою керування віртуальною реальністю. Реалізацію пристрою доцільно здійснити у вигляді рукавиці з можливістю керування електронними засобами за звичними рухами рук та пальців, що дозволить забезпечити відчуття реалістичності. Розроблювана рукавиця повинна реагувати на рухи пальців та рук у просторі для забезпечення реалістичних відчутті під час її використання у віртуальній реальності. Для відстеження рухів пальців та рук використано сенсори згину, прискорення та переміщення. Для зв'язку із зовнішніми електронними засобами чи комп'ютерною технікою розроблено прикладне програмне забезпечення та драйвер.

Для реалізації рукавиці віртуальної реальності розроблено структурну та функціональну схеми. Структурна схема (рис. 2.1) складається з блока сенсорів (сенсорів згину, прискорення та розташування), керування, зв'язку та пристрою керування (ПК).

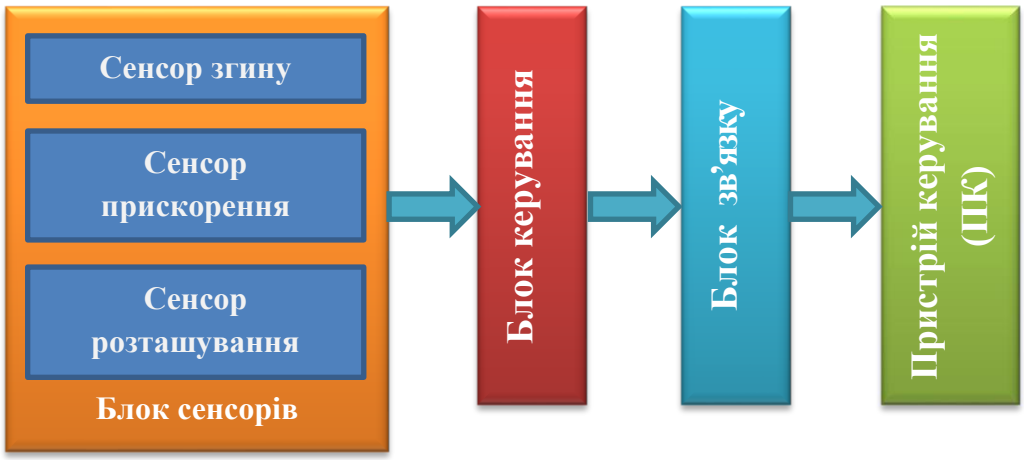


Рис.2.1. Структурна схема

Сигнали із блоку сенсорів, які відповідають певним командам передаються на блок керування. Після відповідних перетворень оцифровуються та через блок

зв’язку передаються до пристрою керування, в якому реалізується процес управління середовищем програми віртуальної реальності.

На основі структурної схеми розроблено функціональну схему рукавиці віртуальної реальності (рис.2.2, 2.3) в якій використано три сенсора згину для забезпечення відстеження рухів великого, вказівного та середнього пальців. Згинаючи палець сенсор змінює опір в межах 10-22 кОм, контролер фіксує спад напруги на першому, другому та третьому аналогових портах відповідно для кожного сенсора згину. Для зменшення похибок при вимірюванні значень, використовується підтягуючи резистор на 10 кОм.

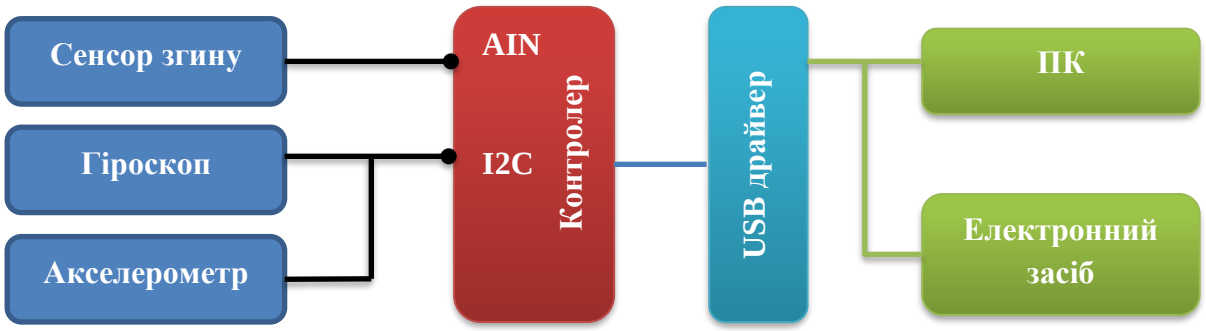


Рис.2.2. Функціональна схема рукавиці віртуальної реальності.

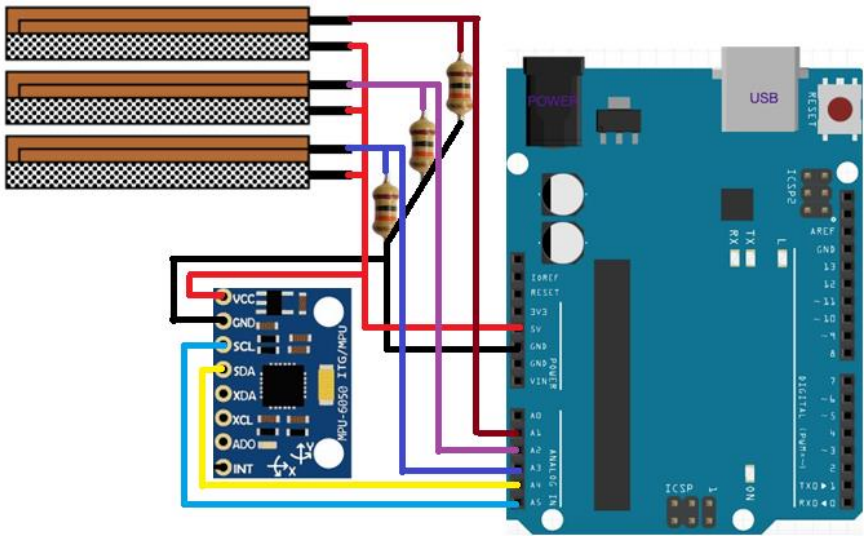


Рис. 2.3. Схема з’єднань функціональних вузлів.

Для реалізації функцій контролю розташування та прискорення використано модуль MPU6050, який під’єднується до мікропроцесора через

поширений цифровий інтерфейс I2C. Для контролю згину пальців використовується аналоговий сенсор опору резистивного типу FLEX SENSOR 2.2. Для зв’язку модуля Arduino з комп’ютером чи іншим електронним засобом використано послідовний інтерфейс USB. Драйвер цього інтерфейсу розташовано безпосередньо на платі Arduino UNO.

Для відстежування рухів руки та пальців, застосовані сенсори згину та гіроскоп, сигнал від яких подається на контролер, який в свою чергу конвертує його з аналогового в цифровий та відправляє на USB драйвер електронного засобу чи комп’ютера. Програма на комп’ютері перехоплює цей сигнал й відповідно до значень виконує функції (рух курсором, скролінг, управління в іграх тощо).

Для відстеження положення руки в просторі використано гіроскоп, який реєструє кут нахилу руки відносно інерційного простору. Для стабільної роботи гіроскопа використано акселерометр для виключення накопичувальної помилки, яка може виникнути. На рис. 2.4 наведено графічне відображення помилок які виникають в процесі роботи гіроскопу та акселерометра. Жовтий блок відповідає положенню гіроскопа який за деякий час піднявся вгору, зелений блок відповідає за комбінацію гіроскопа й акселерометра та працює без відхилення, а голубий блок показує узгодження роботи жовтого та зеленого блоків з врахуванням накопичувальної помилки.

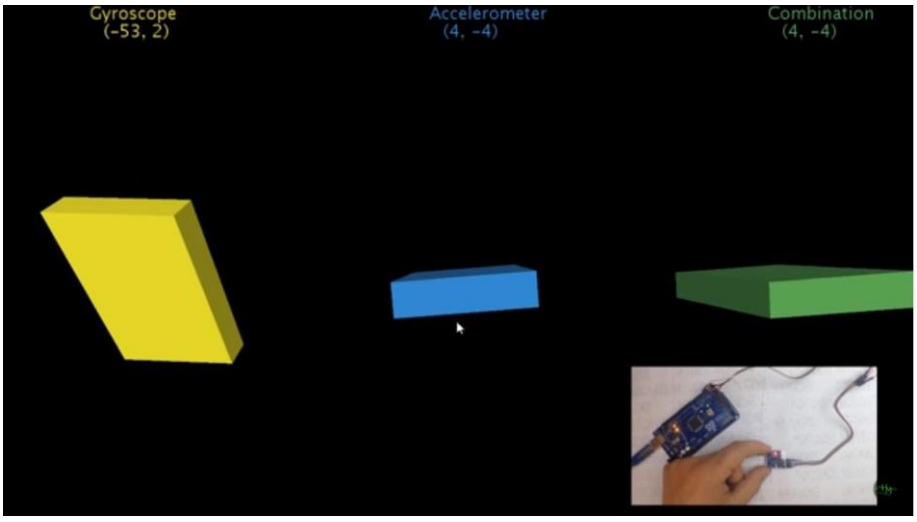


Рис. 2.4. Накопичувальна помилка гіроскопу, акселерометра.

Застосуванням лише двох ліній зв’язку забезпечує передавання даних від сенсора до контролера шиною передачі I2C [9] та обмін даними з безліччю пристроїв із застосуванням платформи Arduino.

Оптимальним варіантом для побудови схеми керування пристроєм віртуальної реальності є використання мікропроцесорної платформи Arduino UNO на основі мікроконтролера сімейства AVR ATMega328P [10]. Дана платформа володіє широкими функціональними можливостями, дозволяє здійснювати аналіз та оброблення як аналогових так і цифрових сигналів.

2.2. Особливості функціональних вузлів

У якості мікропроцесора використовується мікроконтролер ATmega328P (рис.2.5) [11], який складається з 14 цифрових входів/виходів (з них 6 можуть використовуватися в якості ШІМ-виходів), 6 аналогових входів, вузла синхронізації з кварцовим резонатором на 16 МГц, роз’ємів USB, для живлення та програмування всередині схеми (ICSP) та вузла Reset з кнопкою [12]. Для налагодження використовується плата Arduino Uno. Для початку роботи з пристроєм досить просто подати живлення від AC/DC-адаптера або батарейки, або підключити його до комп’ютера за допомогою USB-кабелю.

Рис. 2.5. Плата Arduino Uno з мікроконтролером ATmega328P.

Основні технічні характеристики мікроконтролера ATmega328 [12]:

- робоча напруга 5В,
- напруга живлення (рекомендована) 7-12В,

- напруга живлення (гранична) 6-20В,
- цифрові входи/виходи: 14 (з них 6 можуть використовуватися в якості ШІМ-виходів),
- 6 аналогових входів,
- максимальний струм одного виводу 40 мА,
- максимальний вихідний струм виводу 3.3 В, 50 мА,
- flash-пам’ять 32 КБ (ATmega328) з яких 0.5 КБ використовується завантажувачем,
- SRAM 2 КБ (ATmega328),
- EEPROM 1 КБ (ATmega328),
- тактова частота 16 МГц.

Вибір гіроскопа та акселерометра. Гіроскоп реагує на зміну орієнтації основи, на якій його встановлено відносно інерційного простору. Акселерометр вимірює силу реакції індукованої прискоренням або гравітацією. Одно- та багато-вісні моделі можуть визначати величину та напрям прискорення у вигляді векторної величини, тому можуть бути використані для визначення орієнтації, вібрації та ударів.

На ринку представлено різноманітні сенсори від різних виробників. Для вибору сенсора проаналізуємо популярні сенсори серії MPU. MPU6000 і MPU6050 мають однакові 3-вісний гіроскоп, 3-вісний акселерометр та дозволяють отримати максимальну швидкість передачі 8 кГц. З точки зору контролерів, єдиною різницею між ними є шина, яка з’єднує їх з процесором. MPU6000 дозволяє використовувати як I2C, так і SPI, тоді як MPU6050 має лише I2C [13]. Тому MPU6000 кращий пристрій за умови використання шина SPI. I2C занадто повільний, щоб обробляти гіроскопи з швидкістю 8kHz.

Сенсор MPU6500 підтримує як I2C, так і SPI, швидкість оновлення гіроскопа 32 кГц, має значно більшу пропускну здатність гіросигналу, споживає менше енергії та більш чутливий до вібрації та кращий за параметрами від сенсора MPU6000.

На основі проведеного аналізу обрано MPU-6500 (рис.2.6) для розроблення рукавиці віртуальної реальності [14].

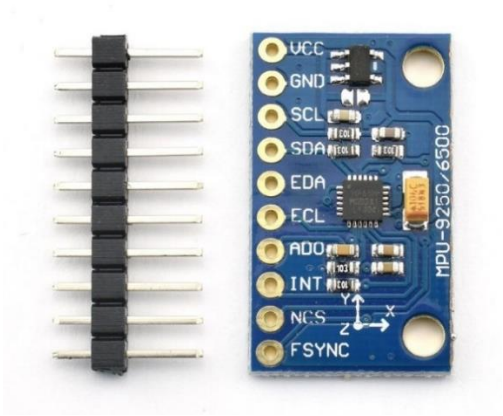


Рис.2.6. Модуль MPU-6500 гіроскоп, акселерометр.

Основні технічні характеристики модуля MPU-6500 [14]:

- мікросхема MPU6500;
- напруга живлення 3-5В;
- інтерфейс I2C(400кГц)/SPI(1мГц);
- діапазон вимірювання акселерометра +/-2G, +/-4G, +/-8G, +/16G;
- діапазон вимірювання гіроскопу +/-250, +/-500, +/-1000, +/2000 dps;
- буфер FIFO 512В;
- габарити 15х25 мм.

Вибір сенсорів згину. Для керування в розробленій рукавиці потрібно застосувати сенсор тиску, який реєструє збільшення опору при згині пальця. Сенсор тиску призначений для визначення руху пальця в системі керування для рукавиці віртуальної реальності [15].

У розробленій системі використано “FLEX SENSOR 2.2” який зображений на рис.2.7 [16].

Рис.2.7. Сенсор згину “FLEX SENSOR 2.2”.

Основні технічні характеристики сенсора тиску “FLEX SENSOR 2.2”:

- опір: 10 кОм (+/-30%);
- опір при згині на 90°: 14 кОм(+/-30%);
- опір при згині на 180°: 22 кОм(+/-30%);
- номінальна потужність: 0,5 Вт;
- пікова потужність: 1 Вт;
- максимальна кількість згинів > 100000;
- робоча температура: -35°C...80°C.

На основі аналізу промислових зразків комплектуючих обрано потрібні елементи та засоби для розроблення системи керування рукавиці віртуальної реальності. За основу пристрою був вибраний мікропроцесор ATmega328P. В якості сенсорів обрано сенсори для відстеження руху руки та згину пальців. Для розробки програмної частини було обрано мову високого рівня C#. В якості середовища для написання коду обрано Visual Studio 2019.

РОЗДІЛ 3. РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ РУКАВИЦІ ВІРТУАЛЬНОЇ РЕАЛЬНОСТІ

3.1. Розроблення вбудованого програмного забезпечення

Для розробки програмної частини було вибрано мову програмування C#, яка об'єктно-орієнтована з безпечною системою типізації для платформи .NET. На сьогоднішній день C# займає третю сходинку за популярністю серед всіх мов програмування, поступаючись лише мові програмуванні Java і JavaScript (рис.3.1) [17].

Рис.3.1. Рейтинг мов програмування за 2020 рік.

Основні переваги мови програмування C#:

- об'єктно орієнтована, проста і в той же час потужна мова програмування, яка дозволяє розробити створені багатофункціональні додатки.
- відноситься до мов компільованого типу, тому вона володіє всіма перевагами таких мов.
- об'єднує найкращі ідеї сучасних мов програмування таких як Java, C ++, Visual Basic.
- велика різноманітність синтаксичних конструкцій.
- можливість працювати з платформою .Net.
- надійна та проста мова програмування.

3.1.1. Середовище розробки

Як середовище для розробки було вибране розроблено компанією Microsoft середовище Visual Studio 2019 (рис.3.2). Воно націлене на роботу з мовами

програмування сімейства .Net, тому воно є чудовим вибором для мови програмування C#.

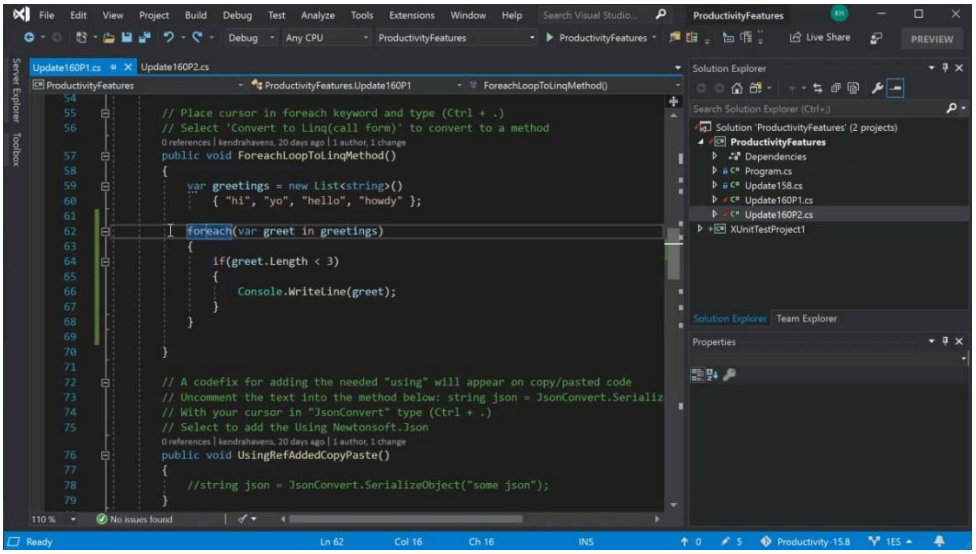


Рис.3.2. Графічний інтерфейс середовища Visual Studio 2019.

Visual Studio 2019 підтримує багато мов програмування при розробці, автоматично генерує шаблонний код, володіє інтуїтивним стилем кодування, функцією IntelliSense, яка виділяти синтаксичні помилки і пропонувати інші варіанти та підтримує пакети NuGet [18, 19].

3.1.2. Програмний код для контролера ATMega328P

Для розробки програмного коду контролера, написаний алгоритму роботи пристрою керування рукавицею віртуальної реальності (див. Додаток А).

Першим кроком в коді визначається бібліотека функцій, які будуть використовуватися для розробки робочого коду. Для завдання обрано бібліотеку **Wire.h** [20]. Дана бібліотека використовується для зв’язку мікроконтролера з пристроями і модулями через інтерфейс I2C.

Наступним кроком є визначення імен для роботи з модулем MPU-6500 (рис. 3.3), за допомогою директиви #define, відповідно до таблиці даних, яка відповідає документу карти реєстрації **InvenSense “MPU-6000 and MPU-6050 Register Map”**.

Для написання коду самої програми, потрібно визначити та описати основні функції, структуру та змінні, що представлені блоками завдань.

```
#define MPU6050_AUX_VDDIO          0x01   // R/W
#define MPU6050_SMPLRT_DIV         0x19   // R/W
#define MPU6050_CONFIG             0x1A   // R/W
#define MPU6050_GYRO_CONFIG        0x1B   // R/W
#define MPU6050_ACCEL_CONFIG       0x1C   // R/W
#define MPU6050_FF_THR             0x1D   // R/W
#define MPU6050_FF_DUR             0x1E   // R/W
#define MPU6050_MOT_THR            0x1F   // R/W
#define MPU6050_MOT_DUR            0x20   // R/W
#define MPU6050_ZRMOT_THR          0x21   // R/W
#define MPU6050_ZRMOT_DUR          0x22   // R/W
#define MPU6050_FIFO_EN            0x23   // R/W
#define MPU6050_I2C_MST_CTRL       0x24   // R/W
#define MPU6050_I2C_SLV0_ADDR      0x25   // R/W
#define MPU6050_I2C_SLV0_REG       0x26   // R/W
#define MPU6050_I2C_SLV0_CTRL      0x27   // R/W
#define MPU6050_I2C_SLV1_ADDR      0x28   // R/W
#define MPU6050_I2C_SLV1_REG       0x29   // R/W
#define MPU6050_I2C_SLV1_CTRL      0x2A   // R/W
#define MPU6050_I2C_SLV2_ADDR      0x2B   // R/W
#define MPU6050_I2C_SLV2_REG       0x2C   // R/W
#define MPU6050_I2C_SLV2_CTRL      0x2D   // R/W
#define MPU6050_I2C_SLV3_ADDR      0x2E   // R/W
#define MPU6050_I2C_SLV3_REG       0x2F   // R/W
#define MPU6050_I2C_SLV3_CTRL      0x30   // R/W
#define MPU6050_I2C_SLV4_ADDR      0x31   // R/W
#define MPU6050_I2C_SLV4_REG       0x32   // R/W
#define MPU6050_I2C_SLV4_DO        0x33   // R/W
#define MPU6050_I2C_SLV4_CTRL      0x34   // R/W
#define MPU6050_I2C_SLV4_DI        0x35   // R
#define MPU6050_I2C_MST_STATUS     0x36   // R
#define MPU6050_INT_PIN_CFG        0x37   // R/W
#define MPU6050_INT_ENABLE         0x38   // R/W
#define MPU6050_INT_STATUS         0x3A   // R
#define MPU6050_ACCEL_XOUT_H       0x3B   // R
#define MPU6050_ACCEL_XOUT_L       0x3C   // R
#define MPU6050_ACCEL_YOUT_H       0x3D   // R
#define MPU6050_ACCEL_YOUT_L       0x3E   // R
#define MPU6050_ACCEL_ZOUT_H       0x3F   // R
#define MPU6050_ACCEL_ZOUT_L       0x40   // R
#define MPU6050_TEMP_OUT_H         0x41   // R
#define MPU6050_TEMP_OUT_L         0x42   // R
```

Рис. 3.3. Визначення імен для роботи з модулем MPU-6500.

Першим блоком завдань є об’єднання структур для визначення змінних регістрів та осей акселерометра (рис. 3.4).

```
633  typedef union accel_t_gyro_union
634  {
635    struct
636    {
637      uint8_t x_accel_h;
638      uint8_t x_accel_l;
639      uint8_t y_accel_h;
640      uint8_t y_accel_l;
641      uint8_t z_accel_h;
642      uint8_t z_accel_l;
643      uint8_t t_h;
644      uint8_t t_l;
645      uint8_t x_gyro_h;
646      uint8_t x_gyro_l;
647      uint8_t y_gyro_h;
648      uint8_t y_gyro_l;
649      uint8_t z_gyro_h;
650      uint8_t z_gyro_l;
651    } reg;
652    struct
653    {
654      int x_accel;
655      int y_accel;
656      int z_accel;
657      int temperature;
658      int x_gyro;
659      int y_gyro;
660      int z_gyro;
661    } value;
662  };
```

Рис. 3.4. Об’єднання структур для визначення змінних регістрів та осей акселерометра.

Наступним блоком – глобальні змінні в яких визначено коди пінів на які запаяні резистивні сенсори згину (рис.3.5).

```
779   //-------------// Піни на які запаяні резиствині датчики згину
780   const int firstAnalogInPin = A1;
781   const int secondAnalogInPin = A2;
782   const int thirdAnalogInPin = A3;
783   //-------------//
```

Рис.3.5. Визначення пінів для резистивних датчиків згину.

Наступний блок – функція яка зберігає в глобальні змінні останній прочитаний кут повороту сенсору акселерометра та гіроскопу (рис.3.6).

```
674 void set_last_read_angle_data(unsigned long time, float x, float y, float z, float x_gyro, float y_gyro, float z_gyro) {
675   last_read_time = time;
676   last_x_angle = x;
677   last_y_angle = y;
678   last_z_angle = z;
679   last_gyro_x_angle = x_gyro;
680   last_gyro_y_angle = y_gyro;
681   last_gyro_z_angle = z_gyro;
682 }
```

Рис. 3.6. Функція зберігання останнього кута повороту акселерометра та гіроскопу.

Наступним блоком є функція яка записує декілька байтів на I2C пристрій (рис.3.7), що приймає наступні параметри: *start* – стартовий адрес який використовується для визначення регістрів; *pData* – вказівник на даних для запису; *size* – кількість байт для запису.

```
1009  int MPU6050_write(int start, const uint8_t *pData, int size)
1010  {
1011    int n, error;
1012
1013    Wire.beginTransmission(MPU6050_I2C_ADDRESS);
1014    n = Wire.write(start);        // записує стартову адресу
1015    if (n != 1)
1016      return (-20);
1017
1018    n = Wire.write(pData, size);  // записує кількість байтів даних
1019    if (n != size)
1020      return (-21);
1021
1022    error = Wire.endTransmission(true); // звільняє I2C шину
1023    if (error != 0)
1024      return (error);
1025
1026    return (0);         // повертає значення 0 в якості успішного виконання функції
1027  }
```

Рис.3.7. Функція запису на I2C пристрій.

Функція повертає код успішності виконання. Також є додаткова функція *MPU6050_write_reg* яка виконує цей самий запис тільки для одного регістру.

Наступним блоком є функція аналогічна попередній функції, але за допомогою неї виконується читання з I2C пристрою (рис.3.8).

```
963  int MPU6050_read(int start, uint8_t *buffer, int size)
964  {
965    int i, n, error;
966
967    Wire.beginTransmission(MPU6050_I2C_ADDRESS);
968    n = Wire.write(start);
969    if (n != 1)
970      return (-10);
971
972    n = Wire.endTransmission(false);
973    if (n != 0)
974      return (n);
975
976    Wire.requestFrom(MPU6050_I2C_ADDRESS, size, true);
977    i = 0;
978    while(Wire.available() && i<size)
979    {
980      buffer[i++]=Wire.read();
981    }
982    if ( i != size)
983      return (-11);
984
985    return (0);
986  }
```

Рис.3.8. Функція читання з I2C пристрою.

Наступним блоком є функція яка читає вхідні дані акселерометра та гіроскопу (рис.3.9), повертає код успішності виконання функції. В ній не визначений фільтр даних, тому вони можуть бути досить не стійкими.

```
703  int read_gyro_accel_vals(uint8_t* accel_t_gyro_ptr) {
704
705    accel_t_gyro_union* accel_t_gyro = (accel_t_gyro_union *) accel_t_gyro_ptr;
706
707    int error = MPU6050_read (MPU6050_ACCEL_XOUT_H, (uint8_t *) accel_t_gyro, sizeof(*accel_t_gyro));
708
709    uint8_t swap;
710    #define SWAP(x,y) swap = x; x = y; y = swap
711
712    SWAP ((*accel_t_gyro).reg.x_accel_h, (*accel_t_gyro).reg.x_accel_l);
713    SWAP ((*accel_t_gyro).reg.y_accel_h, (*accel_t_gyro).reg.y_accel_l);
714    SWAP ((*accel_t_gyro).reg.z_accel_h, (*accel_t_gyro).reg.z_accel_l);
715    SWAP ((*accel_t_gyro).reg.t_h, (*accel_t_gyro).reg.t_l);
716    SWAP ((*accel_t_gyro).reg.x_gyro_h, (*accel_t_gyro).reg.x_gyro_l);
717    SWAP ((*accel_t_gyro).reg.y_gyro_h, (*accel_t_gyro).reg.y_gyro_l);
718    SWAP ((*accel_t_gyro).reg.z_gyro_h, (*accel_t_gyro).reg.z_gyro_l);
719
720    return error;
721  }
```

Рис. 3.9. Функція читання даних акселерометра та гіроскопу.

Наступним блоком є функція яка калібрує сенсори акселерометра та гіроскопу (рис.3.10).

```
735  void calibrate_sensors() {
736    int                   num_readings = 10;
737    float                 x_accel = 0;
738    float                 y_accel = 0;
739    float                 z_accel = 0;
740    float                 x_gyro = 0;
741    float                 y_gyro = 0;
742    float                 z_gyro = 0;
743    accel_t_gyro_union    accel_t_gyro;
744
745    read_gyro_accel_vals((uint8_t *) &accel_t_gyro);
746
747    for (int i = 0; i < num_readings; i++) {
748      read_gyro_accel_vals((uint8_t *) &accel_t_gyro);
749      x_accel += accel_t_gyro.value.x_accel;
750      y_accel += accel_t_gyro.value.y_accel;
751      z_accel += accel_t_gyro.value.z_accel;
752      x_gyro += accel_t_gyro.value.x_gyro;
753      y_gyro += accel_t_gyro.value.y_gyro;
754      z_gyro += accel_t_gyro.value.z_gyro;
755      delay(100);
756    }
757    x_accel /= num_readings;
758    y_accel /= num_readings;
759    z_accel /= num_readings;
760    x_gyro /= num_readings;
761    y_gyro /= num_readings;
762    z_gyro /= num_readings;
763
764    base_x_accel = x_accel;
765    base_y_accel = y_accel;
766    base_z_accel = z_accel;
767    base_x_gyro = x_gyro;
768    base_y_gyro = y_gyro;
769    base_z_gyro = z_gyro;
770  }
```

Рис. 3.10. Функція калібрування акселерометра та гіроскопу.

Після визначень основних структур функцій та змінних користувачам необхідно визначити лише дві функції, для того щоб створити програму, яка буде працювати за принципом циклічного виконання:

setup(): функція виконується лише раз при старті програми і дозволяє визначити початкові параметри;

loop(): функція виконується періодично доки плата не буде вимкнена.

В функції *setup()* відбувається інціалізація класу Wire для I2C шини, калібрування акселерометра та гіроскопу і перевірка зчитування та запис даних на наявність помилки (рис.3.11).

```
797  void setup()
798  {
799    int error;
800    uint8_t c;
801
802    Serial.begin(38400);
803
804    Wire.begin();
805
806    error = MPU6050_read (MPU6050_WHO_AM_I, &c, 1);
807    error = MPU6050_read (MPU6050_PWR_MGMT_2, &c, 1);
808
809    MPU6050_write_reg (MPU6050_PWR_MGMT_1, 0);
810
811    calibrate_sensors();
812    set_last_read_angle_data(millis(), 0, 0, 0, 0, 0, 0);
813  }
```

Рис.3.11. Функція setup() приграми.

Функція *loop()* розділена на декілька блоків, які виконують певний етап програми (рис.3.12). В першому блоці відбувається отримання даних, їх конвертація в градуси за секунду, а також застосування фільтру для отримання більш чіткіших значень.

```
834  //--------------// Конвертує значення гіроскопу в градуси за секунду
835  float FS_SEL = 131;
836
837  float gyro_x = (accel_t_gyro.value.x_gyro - base_x_gyro)/FS_SEL;
838  float gyro_y = (accel_t_gyro.value.y_gyro - base_y_gyro)/FS_SEL;
839  float gyro_z = (accel_t_gyro.value.z_gyro - base_z_gyro)/FS_SEL;
840  //--------------//
841
842  //--------------// Отримає необроблені значення для акселерометру
843  float accel_x = accel_t_gyro.value.x_accel;
844  float accel_y = accel_t_gyro.value.y_accel;
845  float accel_z = accel_t_gyro.value.z_accel;
846  //--------------//
847
848  //--------------// Отримає значення кута з акселерометра
849  float RADIANS_TO_DEGREES = 180/3.14159;
850  float accel_angle_y = atan(-1*accel_x/sqrt(pow(accel_y,2) + pow(accel_z,2)))*RADIANS_TO_DEGREES;
851  float accel_angle_x = atan(accel_y/sqrt(pow(accel_x,2) + pow(accel_z,2)))*RADIANS_TO_DEGREES;
852
853  float accel_angle_z = 0;
854  //--------------//
855
856  //--------------// Обчислює фільтровані дані з гіроскопу
857  float dt =(t_now - get_last_time())/1000.0;
858  float gyro_angle_x = gyro_x*dt + get_last_x_angle();
859  float gyro_angle_y = gyro_y*dt + get_last_y_angle();
860  float gyro_angle_z = gyro_z*dt + get_last_z_angle();
861  //--------------//
862
863  //--------------// Обчислює дрейфуючі кути гіроскопу
864  float unfiltered_gyro_angle_x = gyro_x*dt + get_last_gyro_x_angle();
865  float unfiltered_gyro_angle_y = gyro_y*dt + get_last_gyro_y_angle();
866  float unfiltered_gyro_angle_z = gyro_z*dt + get_last_gyro_z_angle();
867  //--------------//
868
869  //--------------// Застосовує додатковий фільтр, щоб визначити зміну кута
870  float alpha = 0.96;
871  float angle_x = alpha*gyro_angle_x + (1.0 - alpha)*accel_angle_x;
872  float angle_y = alpha*gyro_angle_y + (1.0 - alpha)*accel_angle_y;
873  float angle_z = gyro_angle_z;  //Акселерометр не дає значення кута осі z
874  //--------------//
```

Рис. 3.12. Блок коду функції loop() для читання та оброблення даних акселерометра та гіроскопу.

Другим блоком функції є читання та обробка резистивних сенсорів згину для трьох пальців середнього, вказівного та великого (рис.3.13).

```
879   //-------------// Читання та обробка датчику згину для великого пальця
880   firstSensorValue = analogRead(firstAnalogInPin);
881   firstOutputValue = map(firstSensorValue, 0, 1023, 0, 255);
882   analogWrite(firstAnalogOutPin, firstOutputValue);
883   //-------------//
884
885   //-------------// Читання та обробка датчику згину для середнього пальця
886   secondSensorValue = analogRead(secondAnalogInPin);
887   secondOutputValue = map(secondSensorValue, 0, 1023, 0, 255);
888   analogWrite(secondAnalogOutPin, secondOutputValue);
889   //-------------//
890
891   //-------------// Читання та обробка датчику згину для вказівного пальця
892   thirdSensorValue = analogRead(thirdAnalogInPin);
893   thirdOutputValue = map(thirdSensorValue, 0, 1023, 0, 255);
894   analogWrite(thirdAnalogOutPin, thirdOutputValue);
895   //-------------//
```

Рис. 3.13. Блок коду функції loop() для читання та обробки резистивних сенсорів згину.

Останнім блоком функції є виведення отриманих даних сенсору руху та резистивних сенсорів згину (рис.3.14).

```
897   //-------------// Виведення даних акселерометра
898   Serial.print(angle_x, 2);
899   Serial.print(F(" "));
900   Serial.print(angle_y, 2);
901   Serial.print(F(" "));
902   Serial.print(angle_z, 2);
903   Serial.print(F(" "));
904   //-------------//
905
906   //-------------// Виведення даних датчику згину великого пальця
907   Serial.print(firstOutputValue);
908   Serial.print(F(" "));
909   //-------------//
910
911   //-------------// Виведення даних датчику згину середнього пальця
912   Serial.print(secondOutputValue);
913   Serial.print(F(" "));
914   //-------------//
915
916   //-------------// Виведення даних датчику згину вказівного пальця
917   Serial.print(thirdOutputValue);
918   Serial.println(F(""));
919   //-------------//
```

Рис. 3.14. Блок коду функції loop() для виведення отриманих даних.

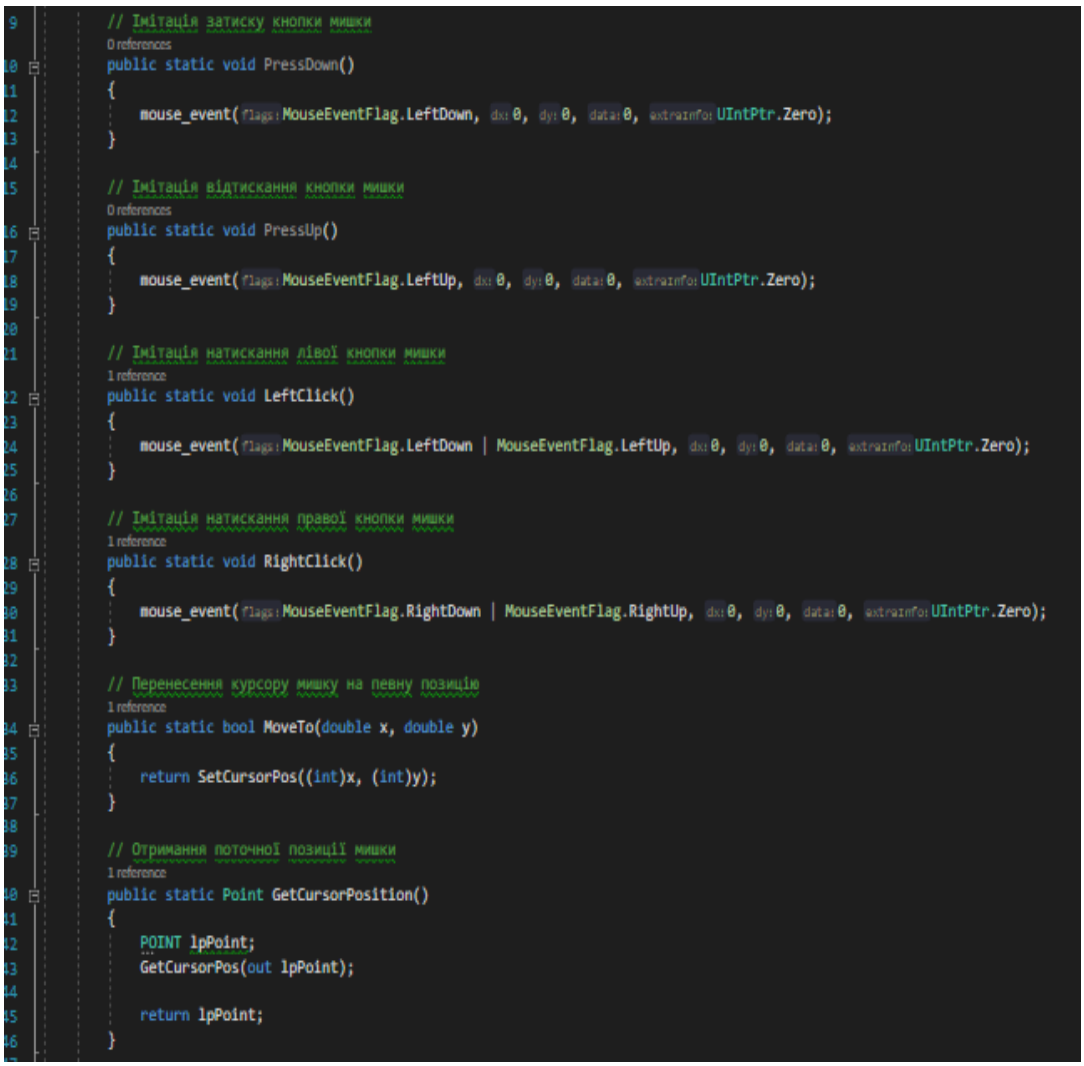
3.2. Розроблення програмного коду для драйвера

Для коду драйвера рукавиці віртуальної реальності (див. Додаток Б) розроблено структурну схему алгоритму його роботи.

Програма складається з класів MouseControl.cs (визначення основних функції для керування комп'ютерною мишкою) та Program.cs (здійснення основної логіки програми). Для керування рукавицею віртуальної реальності використано функції C++ бібліотеки **user32.dll, [DllImport("user32.dll")]**. Клас **MouseControl** розділений на дві частини, з яких одна призначена для визначення функцій та структур C++ (рис.3.15), а друга – для взаємодії з ними (рис.3.16) [21].

```
48      // Бібліотеки C++ для керування мишкою
49      [DllImport(dllName: "user32.dll")]
        1 reference
50      private static extern bool SetCursorPos(int X, int Y);
51      [DllImport(dllName: "user32.dll")]
52
53      // Функція C++ для відтворення подій мишки
        4 references
54      private static extern void mouse_event(MouseEventFlag flags, int dx, int dy, uint data, UIntPtr extraInfo);
55
56      // Перечислення можливих подій для мишки
57      [Flags]
        7 references
58      enum MouseEventFlag : uint
59      {
60          Move = 0x0001,
61          LeftDown = 0x0002,
62          LeftUp = 0x0004,
63          RightDown = 0x0008,
64          RightUp = 0x0010,
65          MiddleDown = 0x0020,
66          MiddleUp = 0x0040,
67          XDown = 0x0080,
68          XUp = 0x0100,
69          Wheel = 0x0800,
70          VirtualDesk = 0x4000,
71          Absolute = 0x8000
72      }
73
74      // Структура C++ для відслідковування позиції мишки
75      [StructLayout(LayoutKind.Sequential)]
        3 references
76      private struct POINT
77      {
78          public int X;
79          public int Y;
80          public static implicit operator Point(POINT point)
81          {
82              return new Point(point.X, point.Y);
83          }
84      }
85
86      // Функція C++ для отримання поточної позиції мишки
87      [DllImport(dllName: "user32.dll")]
        1 reference
88      private static extern bool GetCursorPos(out POINT lpPoint);
```

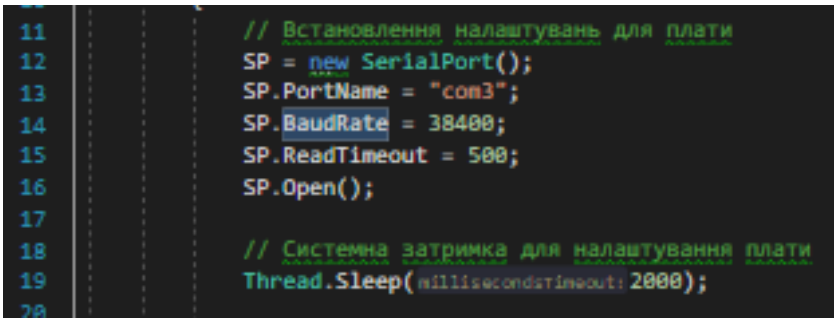
Рис. 3.15. Визначенні функції C++.



```
// Імітація затиску кнопки мишки
0 references
public static void PressDown()
{
    mouse_event(flags: MouseEventFlag.LeftDown, dx: 0, dy: 0, data: 0, extrainfo: UIntPtr.Zero);
}

// Імітація відтискання кнопки мишки
0 references
public static void PressUp()
{
    mouse_event(flags: MouseEventFlag.LeftUp, dx: 0, dy: 0, data: 0, extrainfo: UIntPtr.Zero);
}

// Імітація натискання лівої кнопки мишки
1 reference
public static void LeftClick()
{
    mouse_event(flags: MouseEventFlag.LeftDown | MouseEventFlag.LeftUp, dx: 0, dy: 0, data: 0, extrainfo: UIntPtr.Zero);
}

// Імітація натискання правої кнопки мишки
1 reference
public static void RightClick()
{
    mouse_event(flags: MouseEventFlag.RightDown | MouseEventFlag.RightUp, dx: 0, dy: 0, data: 0, extrainfo: UIntPtr.Zero);
}

// Перенесення курсору мишку на певну позицію
1 reference
public static bool MoveTo(double x, double y)
{
    return SetCursorPos((int)x, (int)y);
}

// Отримання поточної позиції мишки
1 reference
public static Point GetCursorPosition()
{
    POINT lpPoint;
    GetCursorPos(out lpPoint);

    return lpPoint;
}
```

Рис. 3.16. Функції для взаємодії з функціями C++.

Під час запуску програми, перш за все викликається функція **Main(string[] args)** класу Program, яка розділена на певні блоки коду програми. Перший блок коду призначений для ініціалізації класу **SerialPort()** (рис.3.17) та його налаштування такі як ім’я порту, швидкість читання та швидкість передачі даних [22].



```
// Встановлення налаштувань для плати
SP = new SerialPort();
SP.PortName = "com3";
SP.BaudRate = 38400;
SP.ReadTimeout = 500;
SP.Open();

// Системна затримка для налаштування плати
Thread.Sleep(millisecondsTimeout: 2000);
```

Рис. 3.17. Ініціалізація класу SerialPort.

Наступний крок програми це входження у вічний цикл, де вона починає читати дані з плати, а також перетворювати ці дані у цифрові значення (рис.3.18).

```
while (true)
{
    // Читання отриманих даних
    var data = SP.ReadLine();
    var dataArray = data.Split();

    // Встановлення поточної позиції
    var currentPosition = MouseControl.GetCursorPosition();


    // Перетворення строкового значення отриманого з плати в цифрове
    double.TryParse(dataArray[0], out double positionX);
    double.TryParse(dataArray[1], out double positionY);
    double.TryParse(dataArray[2], out double bigFinger);
    double.TryParse(dataArray[3], out double middleFinger);
    double.TryParse(dataArray[4], out double foreFinger);
```

Рис. 3.18. Читання даних з плати.

Наступний блок програми призначений для визначення змінних для поточної позиції мишки, швидкості мишки та максимальне та мінімальне розширення екрану по осі X та Y (рис.3.19).

```
// Поточна швидкість мишки
var movePositionX = 0;
var movePositionY = 0;

// Швидкості мишки
var mouseSpeed1 = 1;
var mouseSpeed2 = 2;
var mouseSpeed3 = 3;

// Максимальна і мінімальна позиція мишки по X
var xMaxPosition = 1920;
var xMinPosition = 1535;

// Максимальна і мінімальна позиція мишки по Y
var yMaxPosition = 863;
var yMinPosition = 0;
```

Рис. 3.19. Ініціалізація основних змінних.

Далі відбувається визначення напряму руху мишки залежно від отриманих даних з плати (рис.3.20).

```
83
84              // ---------- // Перенесення мишки по осі X вправо при повертання руки вправо
85              if (positionX < -20)
86              {
87                  movePositionX = mouseSpeed1;
88              }
89
90              if (positionX < -40)
91              {
92                  movePositionX = mouseSpeed2;
93              }
94
95              if (positionX < -60)
96              {
97                  movePositionX = mouseSpeed3;
98              }
99              // ---------- //
100
101             // ---------- // Перенесення мишки по осі Y вниз при повертання руки вниз
102             if (positionY > 20)
103             {
104                 movePositionY = -1 * mouseSpeed1;
105             }
106
107             if (positionY > 40)
108             {
109                 movePositionY = -1 * mouseSpeed2;
110             }
111
112             if (positionY > 60)
113             {
114                 movePositionY = -1 * mouseSpeed3;
115             }
116             // ---------- //
117
118             // ---------- // Перенесення мишки по осі Y вверх при повертання руки вверх
119             if (positionY < -20)
120             {
121                 movePositionY = mouseSpeed1;
122             }
123
124             if (positionY < -40)
125             {
126                 movePositionY = mouseSpeed2;
127             }
128
129             if (positionY < -60)
130             {
131                 movePositionY = mouseSpeed3;
132             }
133             // ---------- //
```

Рис. 3.20. Визначення напряму мишки.

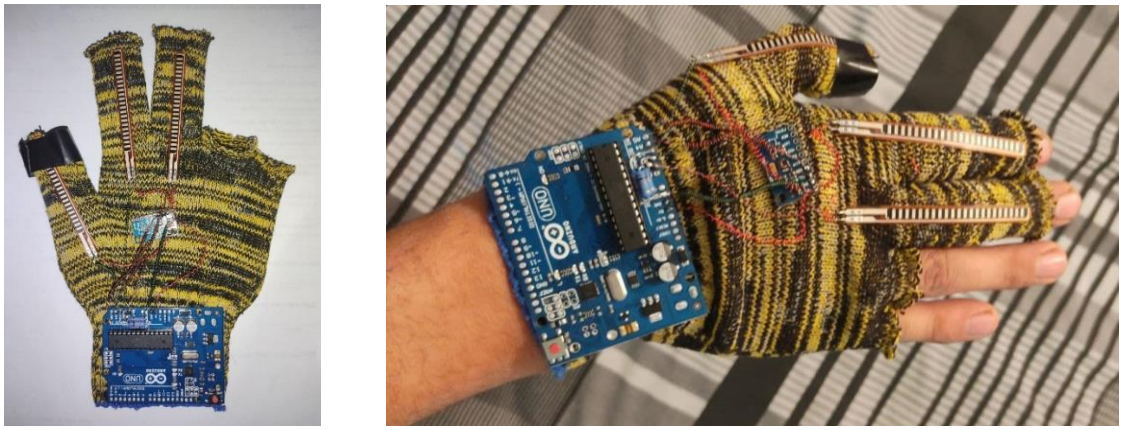
Останнім етапом є виклик функцій взаємодії з мишкою залежно від отриманих даних (рис.3.21).

```
147             // Натискання лівої кнопки мишки при згинанні середнього пальця
148             if (middleFinger < 40)
149             {
150                 MouseControl.LeftClick();
151             }
152
153             // Натискання правої кнопки мишки при згинанні вказівного пальця
154             if (foreFinger < 40)
155             {
156                 MouseControl.RightClick();
157             }
158
159             // Переміщення мишки на нову позицію
160             MouseControl.MoveTo(currentPosition.X + movePositionY, currentPosition.Y + movePositionX);
```

Рис. 3.21. Взаємодія з мишкою.

3.3 Практична реалізація рукавиці віртуальної реальності

В результаті виконання роботи розроблено діючий макет рукавиці віртуальної реальності (рис.3.22).



а *б*

Рис. 3.22. Зовнішній вигляд макету пристрою (а) та його функціонування (б).

Макет був випробуваний для керування комп'ютером під час виконання елементарних рухів зап'ястям та пальцями на дисплеї, приводився в рух курсор та реалізувалася функція відкривання вкладок програм (рис.3.23-3.25). З допомогою рукавиці здійснювалось пересування курсора в різних напрямках, а також імітація кліків правою та лівою кнопками на зразок роботи оптичної мині.

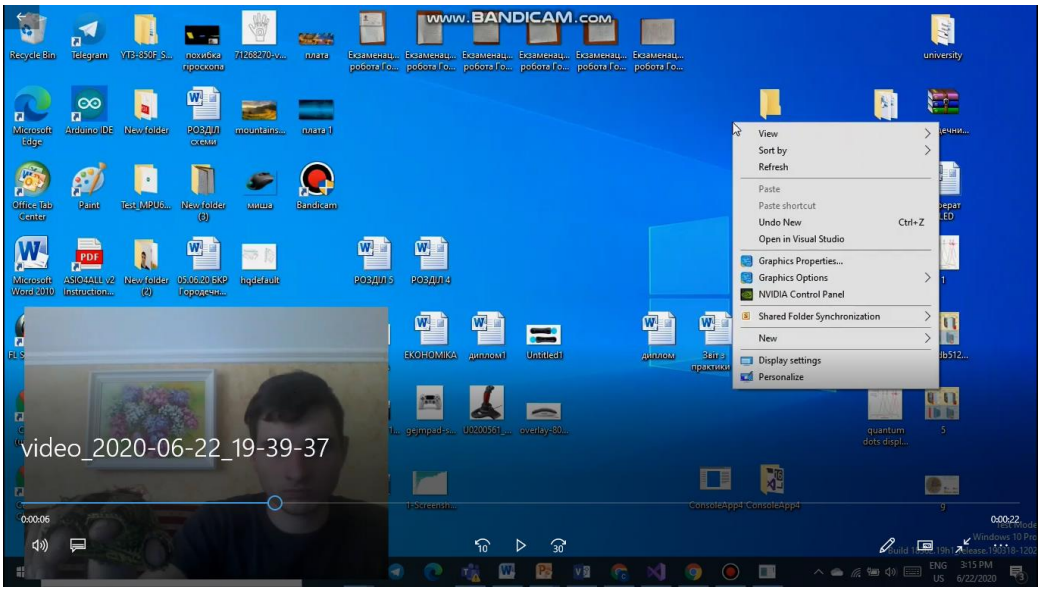


Рис.3.23. В результаті згину пальця відкрилась вкладка прогами.

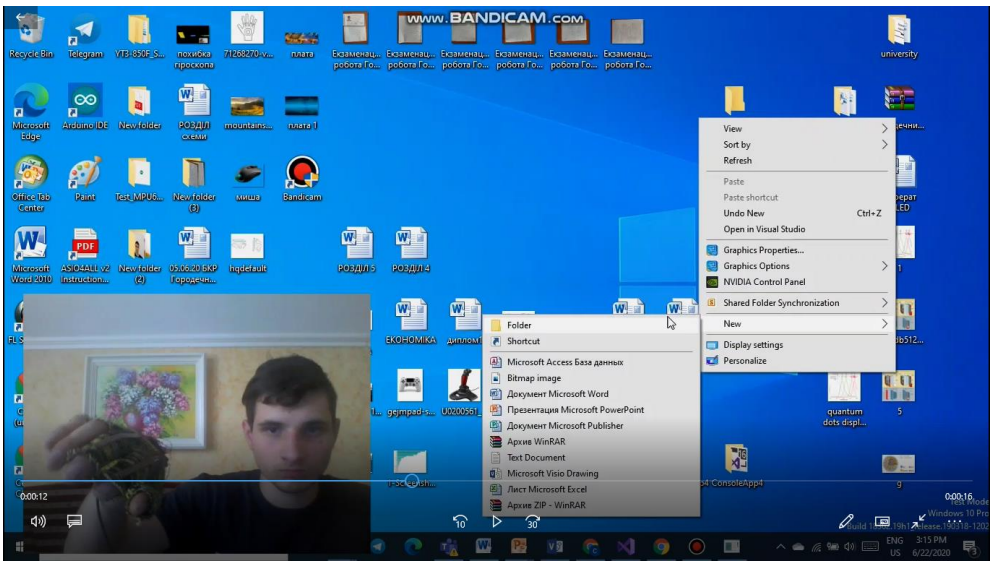


Рис.3.24. В результаті рухів рукою вліво пересунувся курсор та відкрилась розширення меню вкладки прогами.

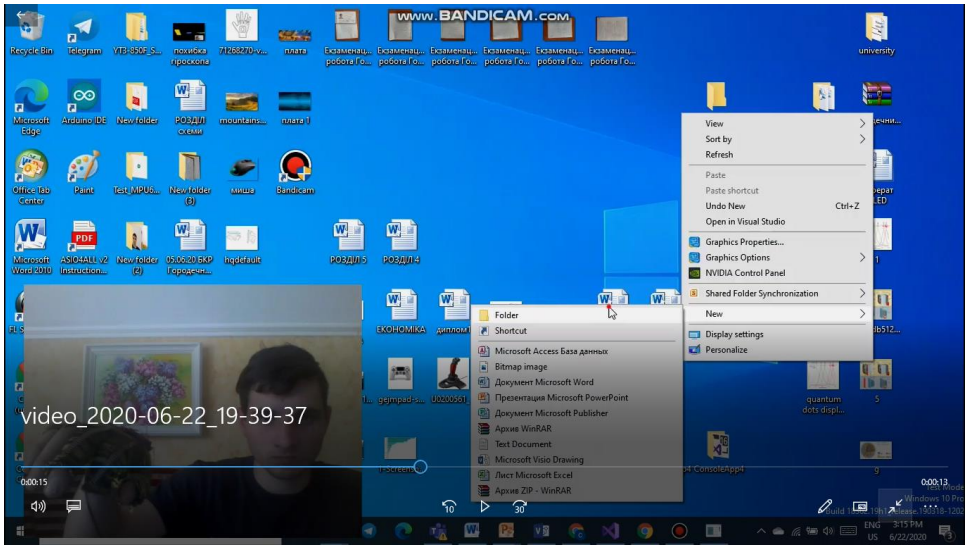


Рис.3.25. В результаті рухів вказівного пальця у відкритій вкладці меню обрано функцію «Folder» та створено папку.

Розроблений пристрій віртуальної реальності володіє наступними характеристиками: напруга живлення 5В, струм споживання 62 мА, діапазон вимірювання прискорення ±2 G, точність вимірювання гіроскопа ± 5 dps, кут повороту: 50°, зв'язок з ПК USB2, маса 0.183 г.

Застосування розробленої рукавиці віртуальної реальності в реалізації цих завдань, підтвердило функціональність даного пристрою та відповідність поставленим вимогам.

ВИСНОВКИ

На основі літературного аналізу сучасних систем керування пристроями віртуальної реальності визначено особливості їхньої будови та функціонування. Показано, що більшість сучасних комерційних VR-продуктів покладаються на геймпади або VR-контролери в питанні взаємодії з предметами віртуальної реальності. Такі пристрої не позбавлені недоліків, особливо, не забезпечують отримання відчуття, що і в реальному житті при взаємодії з предметом за допомогою своїх рук та пальців.

На основі результатів проведеного аналізу розроблено рукавицю віртуальної реальності.

Наведено особливості функціонування структурних елементів рукавиці віртуальної реальності.

Пристрій реалізовано на основі мікропроцесор ATmega328P. В якості сенсорів обрано сенсори для відстеження руху руки і згину пальців. Flex Snsor 2.2 сенсор тиску який міняє опір в залежності від ступеня деформації. MPU 6500 сенсор в якому використовується акселерометр та гіроскоп для відстеження положення в просторі.

Розроблено вбудоване програмне забезпечення для мікроконтролера та драйвера зовнішніх пристроїв.

Список використаної літератури

1. Комп'ютерна миша [Електронний ресурс] // Вікіпедія : вільна енцикл. Електрон. дані. – Режим доступу : https://uk.wikipedia.org/wiki/Комп%-27ютерна_миша. Назва з екрана. – Дата останньої правки : 24.12.2019. – Дата перегляду :02.05.2020.
2. Інтернет магазин [Електронний ресурс] : [Веб-сайт] – Електронні дані – Режим доступу: https://hotline.ua/computer-myshi-klaviatury/gresso-gm-893-usb/ (дата звернення 02.05.2020) – Назва з екрана.
3. Сафронова І. В. Компю'терній мишсці - 40 [Електронний ресурс] / Сафронова І. В // Комсомольська правда : електронне наукове фахове видання. – Електронні дані. – Режим доступу: https://www.kp.ru/online/-news/175727/ – Назва з екрана – (дата звернення 03.05.2020 р.).
4. Джойстик [Електронний ресурс] // Вікіпедія : вільна енцикл. Електрон. дані. – Режим доступу : https://uk.wikipedia.org/wiki/Джойстик. Назва з екрана. – Дата останньої правки : 24.12.2019. – Дата перегляду :03.05.2020.
5. Геймпад [Електронний ресурс] // Вікіпедія : вільна енцикл. Електрон. дані. – Режим доступу : https://uk.wikipedia.org/wiki/. Геймпад. Назва з екрана. – Дата останньої правки : 21.10.2019. – Дата перегляду :04.05.2020.
6. Стаття [Електронний ресурс] : [Веб-сайт] – Електронні дані – Режим доступу:https://docs.yoyogames.com/source/dadiospice/002_reference/mouse, %20keyboard%20and%20other%20controls/gamepad%20input/index.html – Назва з екрана – (дата звернення 04.05.2020).
7. Інтернет магазин [Електронний ресурс] : [Веб-сайт] – Електронні дані – Режим доступу: https://comfy.ua/gejmpad-sony-playstation-dualshock-v2-le-the-last-of-us-part-ii.html – Назва з екрана – (дата звернення 09.05.2020).
8. Bruce Everiss The future, it is all a Gesture [Електронний ресурс] / Bruce Everiss https://www.bruceongames.com/2007/10/02/the-future-it-is-all-a-gesture/ – Назва з екрана – (дата звернення 09.05.2020).

9. How I2C Communication Works and How To Use It with Arduino//[Електронний ресурс] – Режим доступу: https://howtomechatronics.com/tutorials/arduino/how-i2c-communication-works-and-how-to-use-it-with-arduino/. Назва з екрана. – Дата перегляду :14.05.2020.
10. ATmega328P [Електронний ресурс] – Режим доступу до ресурсу: https://www.microchip.com/wwwproducts/en/ATmega328p.
11. Аронець О. А. Arduino для початківців / Олександр Андрійович Аронець. – Івано-Франківськ: Тепле Місто, 2018. – 254 с.
12. Інтернет магазин [Електронний ресурс] : [Веб-сайт] – Електронні дані – Режим доступу: https://arduino.ua/prod2800-arduino-uno-rev3-official-chinese-version. Назва з екрана. – Дата перегляду :10.05.2020.
13. Blum J. Exploring Arduino: Tools and Techniques for Engineering Wizardry / Jeremy Blum. – London: Wiley, 2013. – 384 с.
14. Специфікація сенсора [Електронний ресурс] : [Веб-сайт] – Електронні дані – Режим доступу: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6500-Datasheet2.pdf. Назва з екрана. – Дата перегляду :10.05.2020.
15. Culkin J. Learn Electronics with Arduino: An Illustrated Beginner's Guide to Physical Computing / J. Culkin, E. Hagan. – Ottava: Maker Media, 2017. – 384 с.
16. Специфікація сенсора [Електронний ресурс] : [Веб-сайт] – Електронні дані – Режим доступу: https://components101.com/sensors/flex-sensor-wor king-circuit-datasheet. Назва з екрана. – Дата перегляду :10.05.2020.
17. Кулеш С.А. // DOU.UA опубликовал рейтинг языков программирования среди украинских IT-специалистов. В лидерах Java, JavaScript, C#, PHP и Python [Електронний ресурс] – Режим доступу: /https://itc.ua/news/dou-ua-opublikoval-reyting-yazyikov-programmirova niya-sredi-ukrainskih-it-spetsialistov-v-liderah-java-javascript-c-php-i-python/. Назва з екрана. – Дата перегляду :10.05.2020.

18. Джон П. М. C# для чайников / Пол Мюллер Джон. – Харків: Діалектика, 2019. – 608 с.
19. Visual Studio Technologies Develop C and C++ applications [Електронний ресурс] – Режим доступу: https://visualstudio.microsoft.com/vs/features/cplusplus/. Назва з екрана. – Дата перегляду :10.05.2020.
20. Edwards R. C#.NET ODBC CODE BUILDER IN A BOOK: Using Microsoft Paradox Driver (*.db) / Richard Edwards. – London: Amazon.com Services LLC. – 222 с.
21. Deitel H. C# for Programmers / H. Deitel, P. Deitel. – Indiana: R.R.Domneley in Crawfordsville, 2005. – 1317 с.
22. Arduino Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://www.ladyada.net/learn/arduino/lesson4.html.

Додаток А.

Алгоритм роботи пристрою керування рукавицею віртуальної реальності

Рис. А.1. Алгоритм роботи рукавиці віртуальної реальності.

Додаток Б.

Структурна схема алгоритму коду для драйвера

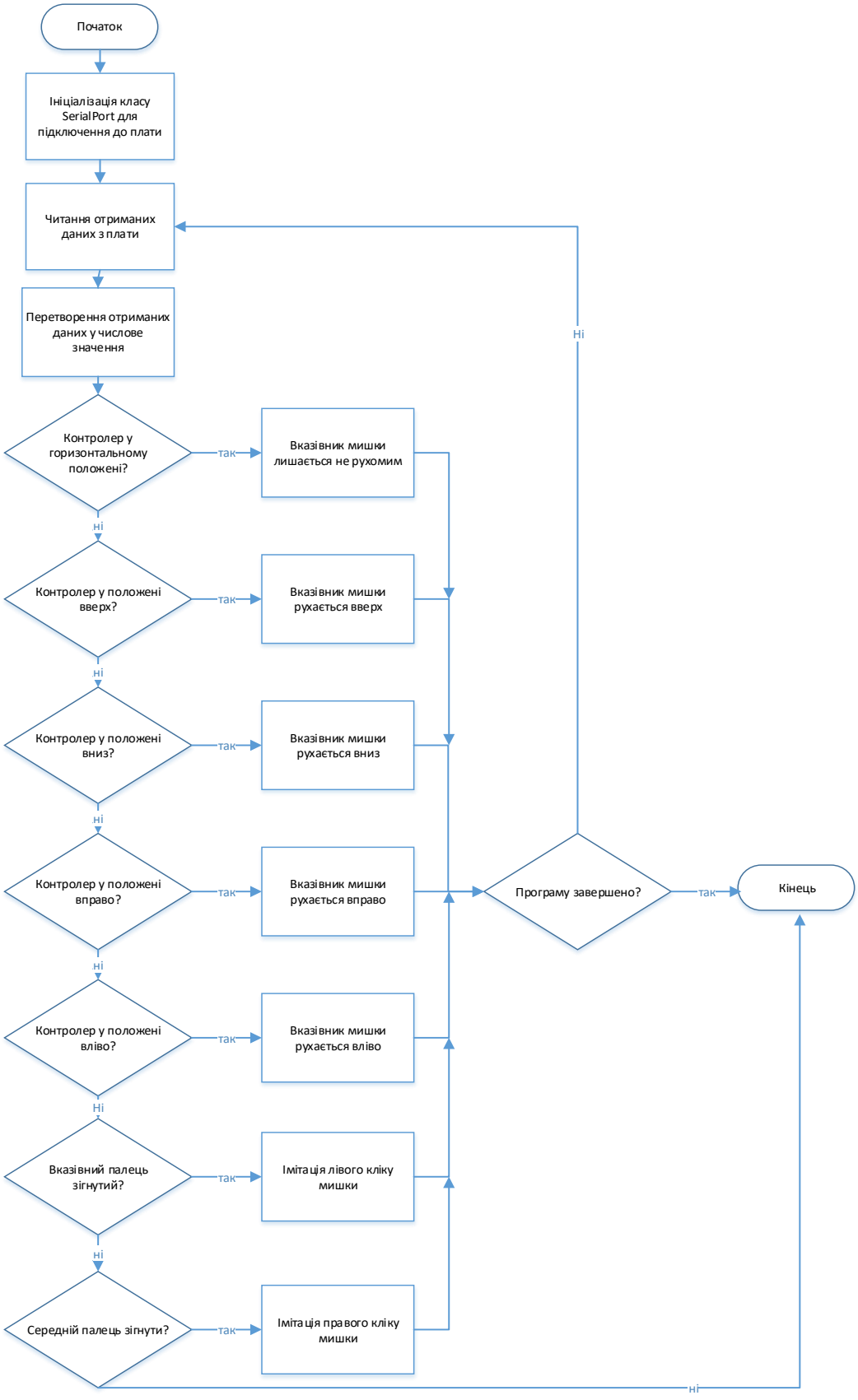


Рис. Б.1. Структурна схема алгоритму коду для драйвера.

Анотація

Значна частина сучасних комерційних VR-продуктів покладаються на геймпади або контролери під час взаємодії з предметами віртуальної реальності. Проте вони не забезпечують реалістичного відчуття як під час взаємодії з предметом за допомогою своїх пальців та рук. Тому виникає потреба у розробленні рукавичок керування, які б дозволили проводити сигнали з ваших рук у віртуальну реальність.

Метою роботи є створення рукавиці віртуальної реальності.

Завдання полягало у проведені літературного аналізу сучасних систем керування пристроями віртуальної реальності та розроблені функціональної та структурної схеми керування рукавиці для віртуальної реальності та написати програмного забезпечення для її роботи.

В роботі використано методики схемотехнічного синтезу, параметричної оптимізації та визначення просторового положення об’єктів.

Робота присвячена розробці рукавиці віртуальної реальності на основі мікроконтролера ATmega 328P. Проведено літературний аналіз сучасних маніпуляторів для керування електронними засобами та комп’ютерною технікою. Виявлено особливості функціонування різноманітних пристроїв та їх можливості в управлінні приладами. На основі проведеного аналізу розроблено рукавичку-маніпулятор, придатну для використання вводу інформації в комп’ютер, приставки, а також в інші пристрої, де необхідно використовувати маніпулятори. В рукавиці віртуальної реальності, керування здійснюється за допомогою сенсорів тиску, які реагують на згин пальців змінюючи опір та акселерометра, який відстежує положення руки відносно простору, та зв’язані програмно.