Міністерство освіти і науки України

|  |
| --- |
| Підпис і дата |

|  |
| --- |
| Інв. № дубл. |

|  |
| --- |
| Зам. інв. № |

|  |
| --- |
| Підпис і дата |

|  |
| --- |
| Інв. № ориг. |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

Вінницький коледж

Національного університету харчових технологій

Відділення автоматизації та комп’ютерної техніки

Комісія інформатики та комп’ютерної техніки

РОЗРОБКА УЛЬТРАЗВУКОВОГО РАДАРУ

На базі плати проектування Arduino

Пояснювальна записка

до курсового проекту з дисципліни архітектура комп’ютерів

за спеціальністю

5.05010201 Обслуговування комп’ютерних систем і мереж

5.05010201.КП.018.00.000 ПЗ

Розробив: студент гр. 3 - ОК - 1

\_\_\_\_\_\_\_\_\_\_\_\_ В.П. Шевченко

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2019 р.

Керівник: викладач спец. дисциплін

\_\_\_\_\_\_\_\_\_\_\_\_ С.С. Заліська

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2019 р.

Вінниця 2019

Зміст

[Вступ 3](#_Toc6786084)

[1. Застосування мікроконтролерів. 4](#_Toc6786085)

[2. Дослідження структурних та функціональних можливостей обраних інтегрованих середовищ розробки. 6](#_Toc6786086)

[2.1 Visual Studio 2017 6](#_Toc6786087)

[2.2 Arduino IDE. 7](#_Toc6786088)

[3. Розробка проектованого об’єкта. 9](#_Toc6786089)

[3.1 Обґрунтування вибору елементної бази. 9](#_Toc6786090)

[3.1.1 Принцип роботи та вимоги до проекту. 9](#_Toc6786091)

[3.1.2 Розробка загального алгоритму проектування. 10](#_Toc6786092)

[3.1.3 Склад та структура пристрою, опис та характеристики складових. 11](#_Toc6786093)

[3.2 Розробка схеми радару. 19](#_Toc6786094)

[4. Розробка програмного забезпечення. 21](#_Toc6786095)

[4.1 Програмне забезпечення для мікроконтролера. 21](#_Toc6786096)

[4.2 Програмне забезпечення для персонального комп’ютера. 24](#_Toc6786097)

[Список літератури 38](#_Toc6786098)

# Вступ

Електронна техніка стрімко ввійшла в повсякденне життя та діяльність людини в вигляді офісних, промислових комп’ютерів, потужних електронно-обчислювальних машин та в вигляді контролерів і мікроконтролерів, які вбудовані на сьогодні практично у всі побутові прилади та промислові установки і виконують функції управління, контролю, захисту та діагностики.

Мікропроцесорні системи докорінно змінили технології створення електронної техніки, підняли суттєво технологічну культуру та рівень знань розробників нової техніки, користувачів і ремонтного персоналу.

Розвиток мікропроцесорної техніки дав поштовх до створення нового класу мікроЕОМ, широко використовуваних в самих різних сферах – від автоматизації робочих місць до створення систем управління технологічними процесами і пристроями.

З розвитком мікропроцесорної техніки функціональні можливості мікроЕОМ значно розширюються, Їх продуктивність наближається до продуктивності мініЕОМ, що зумовлює затребуваність мікроЕОМ в області міні-машин.

1. Застосування мікроконтролерів.

Прогрес технології інтегральних схем і поява великих інтегральних схем (ВІС) привели до багаторазового розширення застосування мікропроцесорів в комп'ютерах, системах управління, вимірювальних приладах і системах реєстрації даних.

Перед тим як розпочати роботу з апаратною обчислювальною платформою Arduino, важливо отримати загальні відомості про мікроконтролери. Мікроконтролери, перш за все, використовуються для автоматизації і метрології, техніці управління і автоматичного регулювання. Переваги мікроконтролерів полягають в тому, що можна ефективно і з малими витратами вимірювати і інтерпретувати фізичні величини, щоб потім приймати необхідні рішення і виконувати необхідні дії.

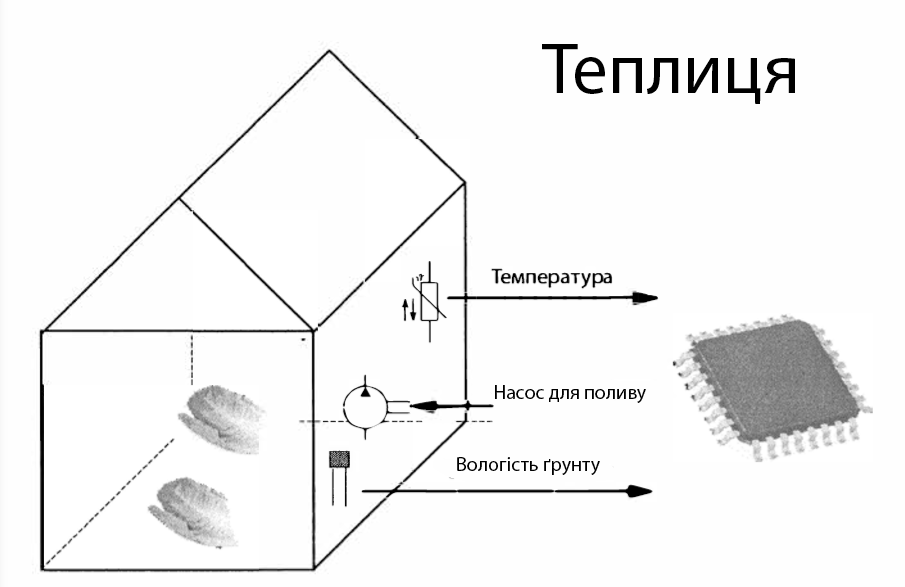


Рисунок 1.1. Система управління кліматом теплиці.

Область можливих додатків мікроконтролерів надзвичайно обширна: від приватного домогосподарства (наприклад, для управління теплицею чи освітленням) до промислового виробництва, де можуть обслуговуватися і експлуатуватися комплексні пристрої, керовані системами контролерів. На Рисунок 1.1. наведено типовий приклад обробки даних для управління зрошувальною установкою теплиці. Контролер Фіксує дані про температуру навколишнього середовища і вологість ґрунту, отримані від датчиків. Результати вимірювання далі обробляються в мікроконтролері. Потім формуються сигнали для управління насосом для поливу.

1. Дослідження структурних та функціональних можливостей обраних інтегрованих середовищ розробки.

## 2.1 Visual Studio 2017

Розробка якісного програмного забезпечення не була б можлива без зручної та багатофункціональної ІСР (англ. Integrated Development Environment). Тому я зупинився на продукті компанії Microsoft Visual Studio 2017 Community що являє собою інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. ІСР дозволяє розробляти як консольні програми так і програми з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-застосунки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight.

Visual Studio включає в себе редактор вихідного коду з підтримкою технології IntelliSense (технологія автодоповнення Microsoft) і можливістю найпростішого рефакторінга коду. Вбудований відладчик може працювати як відладчик рівня вихідного коду, так і відладчик машинного рівня. Решта вбудованих інструментів включають в себе редактор форм для спрощення створення графічного інтерфейсу додатку, веб-редактор, дизайнер класів і дизайнер схеми бази даних. Visual Studio дозволяє створювати і підключати сторонні додатки (плагіни) для розширення функціональності практично на кожному рівні, включаючи додавання підтримки систем контролю версій вихідного коду (як, наприклад, Subversion і Visual SourceSafe), додавання нових наборів інструментів (наприклад, для редагування і візуального проектування коду на предметно-орієнтованих мовах програмування) або інструментів для інших аспектів процесу розробки програмного забезпечення (наприклад, клієнт Team Explorer для роботи з Team Foundation Server).

Visual Studio включає один або декілька з наступних компонентів:

* Visual Basic .NET, а до його появи — Visual Basic
* Visual C++
* Visual C#
* Visual F# (входить до складу Visual Studio 2010);
* [Visual Studio Debugger](https://uk.wikipedia.org/wiki/Visual_Studio_Debugger)

Багато варіантів постачання також включають:

* [Microsoft SQL Server](https://uk.wikipedia.org/wiki/Microsoft_SQL_Server) або
* MSDE Visual Source Safe — файл-серверна система управління версіями

У минулому, до складу Visual Studio також входили продукти:

* Visual InterDev
* Visual J++
* [Visual J#](https://uk.wikipedia.org/wiki/Visual_J_Sharp)
* [Visual FoxPro](https://uk.wikipedia.org/w/index.php?title=Visual_FoxPro&action=edit&redlink=1)
* Visual Source Safe – файл-серверна система управління версіями.

Visual Studio побудована на архітектурі, яка підтримує можливість використання вбудованих додатків (англ. Add-Ins) - плагінів від сторонніх розробників, що дозволяє розширювати можливості середовища розробки.

Деякі з найбільш популярних доповнень:

* ReSharper
* Review Assistant - плагін перегляду і редагування коду
* Visual Assist X
* AnkhSVN - вільна реалізація клієнта Subversion в Visual Studio (в даний час підтримуються версії з 2005 по 2013).
  1. Arduino IDE.

Інтегроване середовище розробки Arduino (IDE) - це мультиплатформний додаток (для Windows, MacOS, Linux), написаний на мові програмування Java. Він використовується для запису та завантаження програм на плати Arduino, а також плат розробки від інших постачальників.

Вихідний код для IDE випускається GNU General Public License. IDE Arduino підтримує мови C і C ++, використовуючи спеціальні правила структурування коду. Arduino IDE постачає бібліотеку програмного забезпечення з проекту Wiring, яка надає багато загальних процедур введення та виведення даних. Користувальницький код вимагає лише двох основних функцій, для запуску ескізу і основного циклу програми, які компілюються і зв'язуються з програмною заглушкою main () в виконувану циклічну виконавчу програму з інструментом GNU, також включеним до розподілу IDE. Arduino IDE використовує програму avrdude для перетворення виконуваного коду в текстовий файл у шістнадцятковому кодуванні, який завантажується в плату Arduino програмою завантажувача в прошивці плати.

1. Розробка проектованого об’єкта.

## 3.1 Обґрунтування вибору елементної бази.

### 3.1.1 Принцип роботи та вимоги до проекту.

Ультразвуковий радар – це радар який за допомогою ультразвуку може визначити положення об’єкту в просторі.

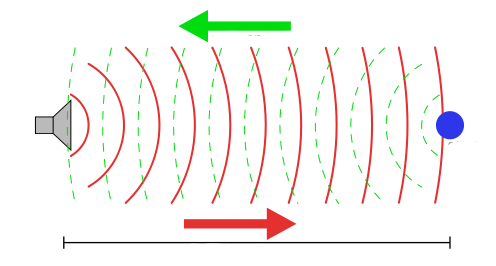


Рисунок 3.1.1 – Принцип роботи ультразвукового радара

На Рисунок 3.1.1 сірий об’єкт являє собою випромінювач та приймач ультразвукових хвиль, синя куля – об’єкт до якого вимірюється відстань, червона дуга – випущена ультразвукова хвиля яка поширюється в повітрі, зелена пунктирна дуга – відбитий ультразвук від поверхні об’єкту який вловлюється приймачем. Так як ми знаємо швидкість поширення звуку і можемо замірити час за який ультразвук прямує до поверхні об’єкта маємо можливість визначити відстань.

де,

S – відстань,

V – швидкість звуку в повітрі за нормальних умов що рівна 343,1 м/с,

T – час.

Пристрій повинен реєструвати та виводити на екран комп’ютера об’єкти що попадають в область дії радару яка рівна одному метру та куту 180.

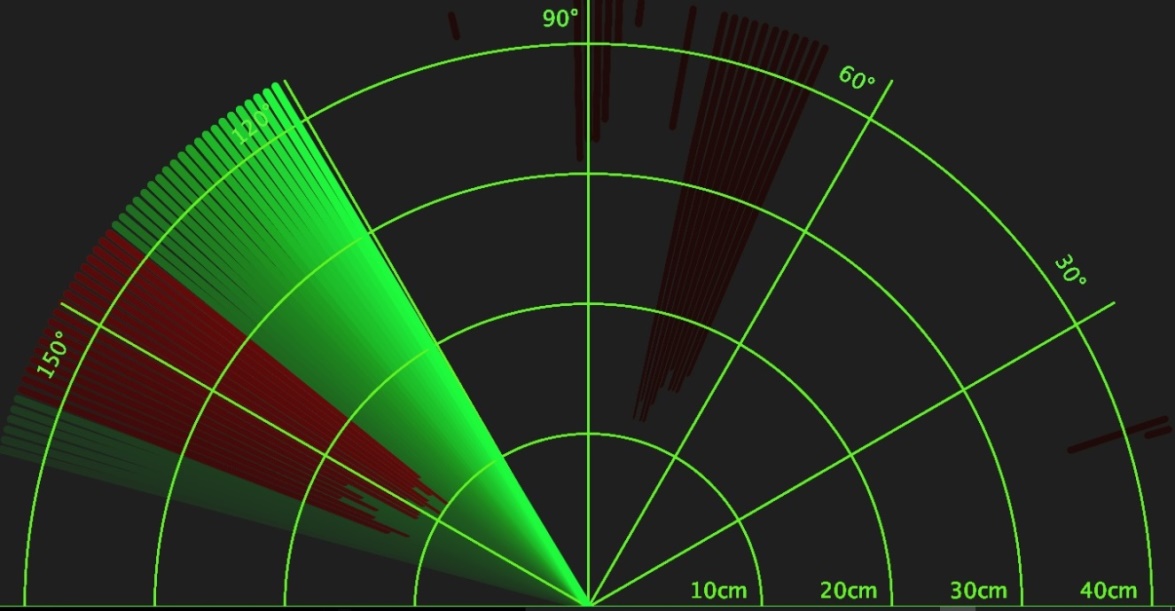


Рисунок 3.1.1.1 – Приклад виводу на екран комп’ютера

* + 1. Розробка загального алгоритму проектування.

Проектування – процес вичерпного обґрунтування та опису об’єктів призначеного для виконання певних цільових завдань.

Проектування можна розбити на наступну послідовність стадій:

1. постановка проблеми та формування загальної мети проектування (творчий процес).
2. знаходження загальної концепції засобів досягнення мети; виявлення основних функцій та принципів дії засобів для об’єктів проектування, що забезпечує досягнення запланованих результатів.
3. ієрархічна декомпозиція загальної мети, функцій та принципів побудови до тої степені конкретизації , коли вимальовуються технологічні конструктивні рішення елементів декомпозиції з кількісними оцінками критеріїв їх якості.
4. розробка технічних завдань на проектування об’єктів та їх складових частин.
5. розробка технічних пропозицій на розробку об’єктів під сформульовані технічні завдання.
6. ескізне проектування об’єктів.
7. розробка технічного проекту.
8. розробка діючих зразків.
   * 1. Склад та структура пристрою, опис та характеристики складових.

Для того щоб скласти список складових з яких складається пристрій потрібно розбити його на функціональні частини.



Рисунок 3.2.1 – Структура проектованого об’єкта

На Рисунок 3.2.1 представлено найпростішу структурну схему проектованого приладу. Він складається з таких частин:

* Мікроконтролер
* Датчик відстані
* Інтерфейс виведення
* Поворотний механізм

Мікроконтро́лер (англ. microcontroller), або однокристальна мікроЕОМ — виконана у вигляді мікросхеми спеціалізована мікропроцесорна система, що включає мікропроцесор, блоки пам'яті для збереження коду програм і даних, порти вводу-виводу і блоки зі спеціальними функціями (лічильники, компаратори, АЦП та інші).

Використовується для керування електронними пристроями. По суті, це — однокристальний комп'ютер, здатний виконувати прості завдання. Використання однієї мікросхеми значно знижує розміри, енергоспоживання і вартість пристроїв, побудованих на базі мікроконтролерів.

Мікроконтролери можна зустріти в багатьох сучасних приладах, таких як телефони, пральні машини, вони відповідають за роботу двигунів і систем гальмування сучасних автомобілів, з їх допомогою створюються системи контролю і системи збору інформації. Більшість процесорів, що випускаються у світі — мікроконтролери.

Датчик відстані - пристрій, який необхідний для того, щоб проводити «безконтактне» вимірювання довжини, висоти або ширини об'єкта.

Інтерфейс виведення – це вивід, на який виводять результат роботи мікроконтролера для подальшого зчитування і обробки на комп’ютері.

Поворотний механізм – двигун, який буде фізично обертати датчик навколо своєї осі.

Ось тепер можна зайнятися вибором складових.

У якості мікроконтролера буде використовуватися Arduino.

Arduino (Ардуіно) — апаратна обчислювальна платформа для аматорського конструювання, основними компонентами якої є плата мікроконтролера з елементами вводу/виводу та середовище розробки Processing/Wiring на мові програмування, що є спрощеною підмножиною C/C++. Arduino може використовуватися як для створення автономних інтерактивних об'єктів, так і підключатися до програмного забезпечення, яке виконується на комп'ютері (наприклад: Processing, Adobe Flash, Max/MSP, Pure Data, SuperCollider). Інформація про плату (рисунок друкованої плати, специфікації елементів, програмне забезпечення) знаходяться у відкритому доступі і можуть бути використані тими, хто воліє створювати плати власноруч.

Плата Arduino складається з мікроконтролера Atmel AVR, а також елементів обв’язки для програмування та інтеграції з іншими пристроями. На багатьох платах наявний лінійний стабілізатор напруги +5В або +3,3В. Тактування здійснюється на частоті 16 або 8 МГц кварцовим резонатором. У мікроконтролер записаний завантажувач (bootloader), тому зовнішній програматор не потрібен.

На концептуальному рівні усі плати програмуються через RS-232 (послідовне з’єднання), але реалізація даного способу різниться від версії до версії. Новіші плати програмуються через USB, що можливо завдяки мікросхемі конвертера USB-to-Serial FTDI FT232R. У версії платформи Arduino Uno як конвертер використовується контролер Atmega8 у SMD-корпусі. Дане рішення дозволяє програмувати конвертер таким чином, щоб платформа відразу розпізнавалася як миша, джойстик чи інший пристрій за вибором розробника зі всіма необхідними додатковими сигналами керування. У деяких варіантах, таких як Arduino Mini або неофіційній Boarduino, для програмування потрібно підключити до контролера окрему плату USB-to-Serial або кабель.

Плати Arduino дозволяють використовувати значну кількість виводів мікроконтролера як вхідні/вихідні контакти у зовнішніх схемах. Наприклад, у платі Decimila доступно 14 цифрових входів/виходів, 6 із яких можуть генерувати ШІМ сигнал, і 6 аналогових входів. Ці сигнали доступні на платі через контактні площадки або штирові роз'єми. Також існує багато різних зовнішніх плат розширення, які називаються «shields» («щити»), які приєднуються до плати Arduino через штирові роз'єми.

Для потреб проекту було вибрано версію Arduino – Nano. Вибір зумовлений дешевизною, малими розмірами і достатньою продуктивністю даної плати.

Arduino Nano надає ряд можливостей для здійснення зв'язку з комп'ютером, ще одним Ардуіно або іншими микроконтроллерами.

Arduino Nano може живитися через кабель Mini-B USB, від зовнішнього джерела живлення з нестабілізованою напругою 6-20В (через вивід 30) або зі стабілізованою напругою 5В (через вивід 27). Пристрій автоматично вибирає джерело живлення з більшою напругою.

Плата має такі характеристики:

|  |  |
| --- | --- |
| **Мікроконтролер** | Atmel ATmega168 або ATmega328 |
| **Робоча напруга** (логічний рівень) | 5В |
| **Напруга живлення** (рекомендована) | 7-12В |
| **Напруга живлення** (гранична) | 6-20В |
| **Цифрові входи / виходи** | 14 (з яких 6 можуть використовуватися як ШІМ-виходи) |
| **Аналогові входи** | 8 |
| **Максимальний струм одного виводу** | 40 мА |
| **Flash-пам'ять** | 16 КБ (ATmega168) або 32 КБ (ATmega328) з яких 2 КБ використовуються завантажувачем |
| **SRAM** | 1 КБ (ATmega168) або 2 КБ (ATmega328) |
| **EEPROM** | 512 байт (ATmega168) або 1 КБ (ATmega328) |
| **Тактова частота** | 16 МГц |
| **Розміри плати** | 1.85 см х 4.3 см |

Був вибраний Ультразвуковий датчик відстані HC-SR04 - призначений для вимірювання відстаней від 2 до 400 см, причому межа точності може досягати до 3 мм. Модуль включає ультразвуковий передавач, приймач і вузол контролю. На покази датчика практично не впливають сонячне випромінювання і електромагнітні шуми. На передній частині HC-SR04 розташовано два ультрозвукових датчика, перший з написом T (Transmiter) - це передавач ультрозвукових хвиль (TCT40-16T), а другий з написом R (Receive) - це приймач відбитих ультрозвукових хвиль (TCT40-16R), по центру розташований вивідний кварцовий генератор на 27 МГц.

Характеристики:

- діапазон вимірювання відстані 0,03м-4м

- частота ультразвуку 40kHz

- кут зору 30град.

- інтерфейс 2 логічні TTL лінії

- вихідна інформація імпульс 0,15..25mS

- напруга живлення "Vcc" 5V

- струм споживання в активному режимі 15мA

- розмір модуля 45х20х15mm

Принцип роботи:

1. На вихід trig (тригер) посилаємо високий рівень протягом як мінімум 10мкс.

2. Модуль починає посилати ультразвукові імпульси з частотою 40 кГц і приймає їх назад, якщо в зоні видимості є будь-які перешкоди.

3. Якщо сигнал повертається, модуль встановлює низький рівень на виході echo на 150мс. За часом, який минув з п.1 до низького рівня на виході echo можна розрахувати відстань до перешкоди за формулою:

відстань = (time \* sound velocity) / 2

де time - виміряне час імпульсу, sound velocity - швидкість звуку (340 м / с)

Точність датчика залежить від декількох факторів:

- температури і вологості повітря;

- відстані до об'єкта;

- розташування щодо датчика (згідно діаграми випромінювання);

- якості виконання елементів модуля датчика.

В основу принципу дії будь-якого ультразвукового датчика закладено явище відображення акустичних хвиль, що поширюються в повітрі. Як відомо з курсу фізики, швидкість поширення звуку в повітрі залежить від властивостей цього самого повітря (в першу чергу від температури). Датчик ж, випускаючи хвилі і заміряючи час до їх повернення, не здогадується, в якому саме середовищі вони будуть поширюватися і бере для розрахунків деяку середню величину. В реальних умовах через фактор температури повітря HC-SR04 може помилятися від 1 до 3-5 см. Фактор відстані до об'єкта важливий, тому що росте ймовірність відбиття від сусідніх предметів, до того ж і сам сигнал загасає з відстанню.

Також для підвищення точності треба правильно направити датчик: зробити так, щоб предмет був в рамках конуса діаграми спрямованості. Простіше кажучи, «очі» HC-SR04 повинні дивитися прямо на предмет. Діаграма спрямованості HC-SR04:

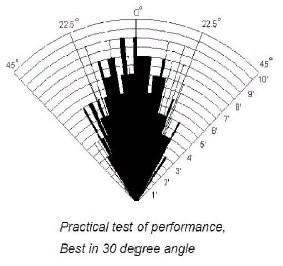
[](https://wiki.tntu.edu.ua/%D0%A4%D0%B0%D0%B9%D0%BB:OLvbeuqZwFY.jpg)

Рисунок 3.2.2 Діаграма спрямованості HC SR04

Для зменшення помилок і похибки вимірювань зазвичай виконуються наступні дії:

- усереднюються значення (кілька разів заміряємо, прибираємо сплески, потім знаходимо середнє);

- за допомогою датчиків (наприклад, DHT11 або DHT22) визначається температура і вносяться поправочні коефіцієнти;

- датчик встановлюється на серводвигун, за допомогою якого ми «повертаємо голову», переміщаючи діаграму спрямованості вліво або вправо.

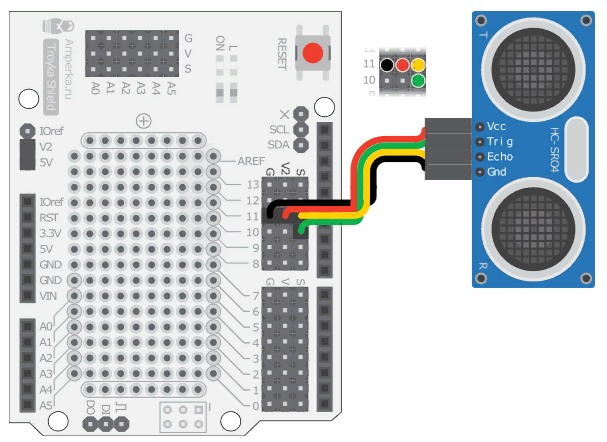
[](https://wiki.tntu.edu.ua/%D0%A4%D0%B0%D0%B9%D0%BB:Erwgrh.png)

Рисунок 3.2.3 Підключення модуля HC SR04 до Arduino

Модуль підключається чотирма проводами. Контакти VCC і GND служать для підключення живлення, а Trig і Echo- для відправки і прийому сигналів далекоміра. Підключаємо їх до пінів 10 і 11 відповідно.

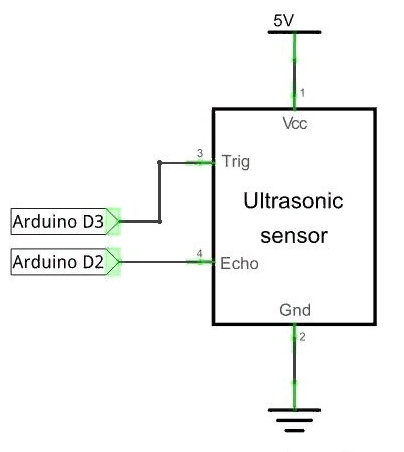


Рисунок 3.2.4 Принципова схема підключення модуля HC SR04 до Arduino

Функцію поворотного механізму буде виконувати серводвигун - Servo TS90A.

Технічні характеристики:

Маса ts90a-9 г

Розмір 22.5 \* 11.9 \* 22 мм

Стопорний крутний момент 1.5 кг / см (4.8 В)

Робоча швидкість 0.1сек / 60 градусів (4.8 В)

Робоча напруга 4.8 В

Діапазон температури 0 ℃ \_ 55 ° з

Мертва ширина 10us

IMG\_0864

Габарити: 23.2 × 12.5 × 22 мм

Вага: 9 г

Робоча швидкість:

110 об. / Хв. (4.8 В)

130 об. / Хв. (6 В)

Пусковий момент:

1.3kg.cm/18.09oz.in (4.8 В)

1.5kg.cm/20.86oz.in (6 В)

Робоча напруга: 4.8 В ~ 6 В

Система управління: аналоговий

Напрямок: проти годинникової стрілки

Робочий кут: 360 градусів

Потрібно імпульсу: 1000us-2000us

Тип підшипника: немає

Тип двигуна: метал

Довжина з'єднувача: 25 см

## 3.2 Розробка схеми радару.

Схема буде розроблена за допомогою пакету програм для автоматизованого проектування електронних схем Proteus Design. Пакет являє собою систему схемотехнічного моделювання, що базується на основі моделей електронних компонентів, прийнятих в PSpice. Відмінною рисою пакету Proteus Design є можливість моделювання роботи програмованих пристроїв: мікроконтролерів, мікропроцесорних систем, DSP і ін. Proteus Design включає в себе більше 6000 електронних компонентів з усіма довідковими даними, а також демонстраційні ознайомчі проекти. Додатково в пакет PROTEUS VSM входить система проектування друкованих плат. Пакет Proteus складається з двох підпрограм: ISIS - програма синтезу та моделювання безпосередньо електронних схем і ARES - програма розробки друкованих плат. Разом з програмою встановлюється набір демонстраційних проектів для ознайомлення. Також програма включає в себе інструменти USBCONN і COMPIM, які дозволяють підключити віртуальний пристрій до USB- і COM-портів комп'ютера.

На основі підібраних складових можна змоделювати пристрій в Proteus Design. Для цього щоб додати в програму потрібні складові знадобилося імпортувати додаткові бібліотеки ArduinoNano.lib та UltrasonicTEP.LIB. Після цього в вікні Pick Device можна знайти потрібну деталь по її назві.

Добавимо наступні деталі:

* 1. ARDUINO NANO
  2. ULTRASONIC SENSOR
  3. MOTOR-PWMSERVO
  4. BATTERY

Крім вже відомих деталей появилася BATTERY. Це звичайне зовнішнє джерело живлення яке потрібне для окремого живлення датчика і сервомашинки. Заживити ці комплектуючі від плати напряму без наслідків не вийде. Це приведе до аварійного вимкнення плати так як сервомашинка споживає занадто великий струм. Це зробить плату нестабільною. В одному випадку плата може перегріватися, а в іншому просто вимикатися. Другий варіант це мій випадок. Тому було прийнято рішення вивести USB кабель який буде підключатися до джерела живлення 5 вольт і живити компоненти радару незалежно від плати.

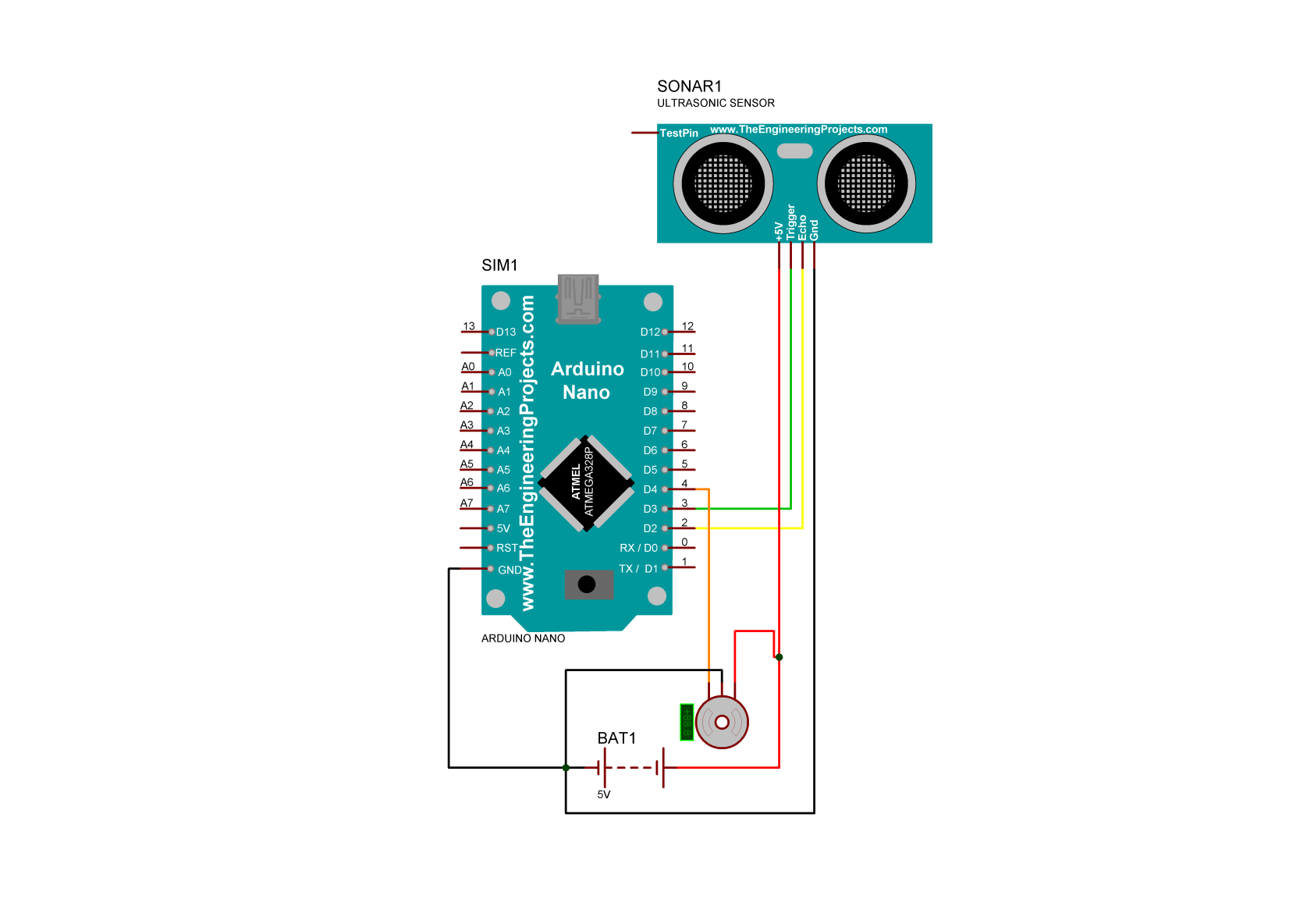


Рисунок 3.2.1 Схема радару

На Рисунок 3.2.1 зображено компоненти радару з’єднані між собою певним чином. Чорний провід – заземлення, червоний – 5 вольт, оранжевий – керуючий серводвигуном вивід, зелений – керування випромінювачем ультразвуку, жовтий – зчитування показників з приймача ультразвукових частот.

Згідно цієї схеми можна спробувати запрограмувати мікроконтролер.

1. Розробка програмного забезпечення.

Доцільно було б розділити розробку програмного забезпечення на дві незалежні частини: Розробку програмного забезпечення для мікроконтролера і для комп’ютера. Вони будуть взаємодіяти між собою за допомогою COM інтерфейсу.

## 4.1 Програмне забезпечення для мікроконтролера.

Мною була розроблена така програма в середовищі програмування плат розробки Arduino IDE.

//Підключення бібліотеки Servo.h для роботи з сервоприводом

#include <Servo.h>

//Оголошення цілих змінних які містять номер піна.

const int trigPin = 3;

const int echoPin = 2;

//Змінні для зберігання куту повороту і відстані відповідно

long duration;

int distance;

int angle;

//Оголошуємо об’єкт типу Servo

Servo myServo;

//Функція яка виконується при увімкненні мікроконтролера

void setup()

{

//Вказуємо, що на пін trig буде подаватися напруга

//а з піна echo буде проводитися зчитування

pinMode(trigPin, OUTPUT);

pinMode(echoPin, INPUT);

//Вказуємо швидкість передачі даних по COM порту

Serial.begin(9600);

//Вказуємо, що керування сервоприводом буде

//здійснюватися по піну номер 4

myServo.attach(4);

//Встановлюємо початковий кут

angle = 10;

//Функція write об'єкту типу Servo задає кут

//на який буде розвернутий сервопривод

myServo.write(angle);

//Встановлюємо затримку для того щоб сервопривод встиг повернутися на //заданий кут до того, як програма продовжить виконуватися

delay(100);

}

//Функція яка циклічно виконується поки плата увімкнена

void loop()

{

for(angle; angle < 180; angle+=1)

{

myServo.write(angle);

distance = calculateDistance();

delay(30);

Serial.print(angle);

Serial.print(",");

Serial.print(distance);

Serial.print(".");

}

for(angle; angle > 0; angle-=1)

{

myServo.write(angle);

distance = calculateDistance();

delay(30);

Serial.print(angle);

Serial.print(",");

Serial.print(distance);

Serial.print(".");

}

}

//Функція для обрахування відстані який вертає датчик

int calculateDistance()

{

//На trig подається 5В протягом 10 мікросекунд для

//створення короткого ультразвукового імпульсу.

digitalWrite(trigPin, LOW);

delayMicroseconds(2);

digitalWrite(trigPin, HIGH);

delayMicroseconds(10);

digitalWrite(trigPin, LOW);

//Функція pulseIn вертає відстань в дюймах яку вертає нам датчик

//Інформація зчитується з echo.

duration = pulseIn(echoPin, HIGH);

//Переведення в сантиметри

distance = duration\*0.034/2;

return distance;

}

Навіть з детальними коментарями в коді програма може потребувати детальніших пояснень.

#include <Servo.h>

Ця стрічка підключає бібліотеку Servo яка призначена для керування сервомашинками. Бібліотека містить багато функцій але оглянемо тільки ті що ми використовували.

uint8\_t attach(int pin);

Вказує пін до якого підключений керуючий провід серво.

void write(int value);

Встановлює кут повороту серво в градусах.

Розглянемо функцію loop. Вона містить два цикла з однаковим тілом циклу з різницею лиш в напрямку відрахування змінної angle. Перший цикл від 0 до 180 та другий – від 180 до 0. Розберемо команди в кожному з циклів.

myServo.write(angle);

Встановлює кут сервомашинки. З кожним циклом машинка повертаєбся рівно на 1 градус.

distance = calculateDistance();

В змінну distance записується результат функції calculateDistance яка вертає відстань до об’єкта в сантиметрах.

Serial.print(angle);

Serial.print(",");

Serial.print(distance);

Serial.print(".");

Кожна з цих команд надсилає в буфер потру довільне значення розміром до одного байта. Буфер має розмір 4 байта. В перший байт записується кут повороту, в другий – кома як роздільник, в третій – дистанція яку вернув там датчик, в четвертий – крапка. Коли буфер заповниться сформується пакет який буде негайно переданий на вихід порту.

Отже, вміст кожного пакета буде мати вигляд {[angle],[distance].}. Саме таку строку ми будемо зчитувати з порту плати на комп’ютері і на основі цих даних будемо будувати картинку. В цьому можна переконатися за допомогою Arduino IDE. Нажавши на кнопку Serial Monitor відкриється вікно в якому ми побачимо потік даних в вигляді описаному вище.

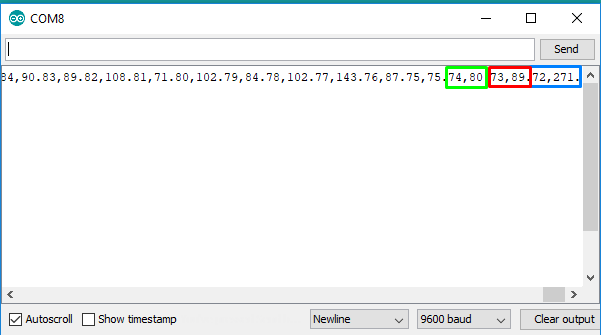


Рисунок 4.1.1 Приклад вікна моніторингу порту

* 1. Програмне забезпечення для персонального комп’ютера.

Згідно з вимог до проекту мною було створено таку програму у середовищі програмування Visual Studio 2017 на мові C# під управлінням .NET платформи. Для створення візуального інтерфейсу програми були використані WPF форми. Вона складається з двох частин: MainWindow та WinRadar.

Спочатку розглянемо детальніше WPF форми і об’єкти які ми будемо використовувати в них.

Windows Presentation Foundation (WPF, кодова назва — Avalon) — графічна (презентаційна) підсистема, яка починаючи з .NET Framework 3.0 в складі цієї платформи. Має пряме відношення до XAML. WPF разом з .NET Framework 3.0 вбудована в Windows Vista, а також доступна для установки в Windows XP Service Pack 2 і Windows Server 2003.

Це перше реальне оновлення технологічного середовища призначеного для користувача інтерфейсу з часу випуску Windows 95. Воно включає нове ядро для заміни GDI і GDI+, використовувані в Windows Forms. WPF є високорівневим об'єктно-орієнтованим функціональним шаром (англ. framework), що дозволяє створювати двовимірні та тривимірні інтерфейси.

Було використано наступні графічні об’єкти WPF:

* Label
* Rectangle
* Path
* Canvas

Label - Елемент призначений для виведення текстової інформації, для створення простих написів.

Rectangle – фігура яка представляє прямокутник.

Path – використовується для створення фігур з складної структурою.

Canvas – найпростіший контейнер в якому можна задати положення кожного дочірнього елемента.

Розглянемо другу частину програми яка являє собою клас вікна з публічними функціями.

using System;

using System.Collections.Generic;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Media;

using System.Windows.Media.Animation;

using System.Windows.Shapes;

using System.Windows.Input;

namespace Радар

{

public partial class WinRadar : Window

{

//Оголошуємо змінні

Label angle, distance;

List<Rectangle> lines = new List<Rectangle>();

double ResolWidth, ResolHeight;

//Конструктор класу

public WinRadar()

{

InitializeComponent();

ResolHeight = SystemParameters.PrimaryScreenHeight;

ResolWidth = SystemParameters.PrimaryScreenWidth;

for (int i = 0; i <= 180; i += 2)

lines.Add(DrowLine(i));

DrowRadar();

}

//Функція яка малює лінію з заданими параметрами

Rectangle DrowLine(int angle, int width, int height,

int right, int bottom, Brush brush)

{

Rectangle rectangle = new Rectangle

{

Width = width,

Height = height,

Fill = brush,

Opacity = 0.15,

};

Canvas.SetBottom(rectangle, bottom);

Canvas.SetRight(rectangle, right);

RotateTransform rotate = new RotateTransform(angle)

{

CenterX = rectangle.Width,

CenterY = rectangle.Height / 2

};

rectangle.RenderTransform = rotate;

OnScren(rectangle);

return rectangle;

}

Rectangle DrowLine(int angle)

{

return DrowLine(angle, (int)AdaptResol(830, Sides.Width),

(int)AdaptResol(10, Sides.Height) ,

(int)AdaptResol(960, Sides.Width), (int)AdaptResol(70, Sides.Height),

Brushes.Green);

}

//Функція яка малює радар

void DrowRadar()

{

for (int i = 0; i <= 180; i += 45)

{

Rectangle rect = DrowLine(i, (int)AdaptResol(900, Sides.Width), 2,

(int)AdaptResol(960, Sides.Width), (int)AdaptResol(70, Sides.Height),

Brushes.Yellow);

rect.Opacity = 1;

}

Path arc20, arc40, arc60, arc80, arc100;

arc20 = CreatePath(new Point(0, (int)AdaptResol(300, Sides.Height)),

new Point((int)AdaptResol(332, Sides.Width), (int)AdaptResol(300, Sides.Height)),

(int)AdaptResol(70, Sides.Height), (int)AdaptResol(794, Sides.Width));

arc40 = CreatePath(new Point(0, (int)AdaptResol(300, Sides.Height)),

new Point((int)AdaptResol(664, Sides.Width), (int)AdaptResol(300, Sides.Height)),

(int)AdaptResol(70, Sides.Height), (int)AdaptResol(628, Sides.Width));

arc60 = CreatePath(new Point(0, (int)AdaptResol(300, Sides.Height)),

new Point((int)AdaptResol(996, Sides.Width), (int)AdaptResol(300, Sides.Height)),

(int)AdaptResol(70, Sides.Height), (int)AdaptResol(462, Sides.Width));

arc80 = CreatePath(new Point(0, (int)AdaptResol(300, Sides.Height)),

new Point((int)AdaptResol(1328, Sides.Width), (int)AdaptResol(300, Sides.Height)),

(int)AdaptResol(70, Sides.Height), (int)AdaptResol(296, Sides.Width));

arc100 = CreatePath(new Point(0, (int)AdaptResol(300, Sides.Height)),

new Point((int)AdaptResol(1660, Sides.Width), (int)AdaptResol(300, Sides.Height)),

(int)AdaptResol(70, Sides.Height), (int)AdaptResol(130, Sides.Width));

Label label1, label2, label3, label4, label5;

label1 = CreateLabel("0°", (int)AdaptResol(45, Sides.Width), (int)AdaptResol(70, Sides.Height));

label2 = CreateLabel("45°", (int)AdaptResol(325, Sides.Width), (int)AdaptResol(700, Sides.Height));

label3 = CreateLabel("90°", (int)AdaptResol(930, Sides.Width), (int)AdaptResol(960, Sides.Height));

label4 = CreateLabel("135°", (int)AdaptResol(1500, Sides.Width), (int)AdaptResol(700, Sides.Height));

label5 = CreateLabel("180°", (int)AdaptResol(1800, Sides.Width), (int)AdaptResol(70, Sides.Height));

angle = CreateLabel("Angle: ", (int)AdaptResol(320, Sides.Width), 2);

distance = CreateLabel("Distance: ", (int)AdaptResol(1200, Sides.Width), 2);

OnScren(arc20, arc40, arc60, arc80, arc100,

label1, label2, label3, label4, label5, angle, distance);

}

//Перерахування яке ширину і висоту

enum Sides

{

Width,

Height

}

//Функція яка обчислює розміри об'єктів відносно розширення системи

double AdaptResol(Double length, Sides sides)

{

if (sides == Sides.Height)

return length / 1080 \* ResolHeight;

else return length / 1920 \* ResolWidth;

}

//Функція яка встановлює значення напису Angle

void SetAngle(int ang)

{

string temp = angle.Content.ToString();

temp = temp.Split(':')[0];

temp += ": " + (ang \* 2);

angle.Content = temp;

}

//Функція яка встановлює значення напису Distance

void SetDistance(int dis)

{

string temp = distance.Content.ToString();

temp = temp.Split(':')[0];

if(dis <= 100)

temp += ": " + dis;

else temp += ": " + ">100";

distance.Content = temp;

}

//Функція яка виводить всі UIElement на екран

void OnScren(params UIElement[] element)

{

foreach(UIElement el in element)

plate.Children.Add(el);

}

//Функція яка створює напис

Label CreateLabel(string content, int left, int bottom)

{

Label label = new Label

{

Content = content,

FontSize = (int)AdaptResol(48, Sides.Height),

Foreground = Brushes.Green,

FontWeight = FontWeights.Bold

};

Canvas.SetBottom(label, bottom);

Canvas.SetLeft(label, left);

return label;

}

//Обробник подій KeyDown для закриття вікна радару

private void MyWinRadar\_KeyDown(object sender, KeyEventArgs e)

{

if (e.Key == Key.Escape) this.Close();

}

// Функція для створення форми дуг

Path CreatePath(Point point1, Point point2, int bottom, int left)

{

PathFigure figure = new PathFigure

{

StartPoint = point1

};

figure.Segments.Add(new ArcSegment(point2,

new Size(1, 1), 180, false, SweepDirection.Clockwise, true));

PathGeometry geometry = new PathGeometry();

geometry.Figures.Add(figure);

Path path = new Path

{

Stroke = Brushes.Yellow,

StrokeThickness = 2

};

path.Data = geometry;

Canvas.SetBottom(path, bottom);

Canvas.SetLeft(path, left);

return path;

}

//Функція яка запускає анімацію вибраної лінії

public void AnimLine(int num, int time, double from, double to, int distance)

{

DoubleAnimation myDoubleAnimation = new DoubleAnimation

{

From = from,

To = to,

Duration = new Duration(TimeSpan.FromMilliseconds(time))

};

double myDist = distance > 100 ? 100 : distance;

LinearGradientBrush myVerticalGradient =

new LinearGradientBrush

{

StartPoint = new Point(0, 0.5),

EndPoint = new Point(1, 0.5)

};

myVerticalGradient.GradientStops.Add(

new GradientStop(Colors.Red, 0.0));

myVerticalGradient.GradientStops.Add(

new GradientStop(Colors.Red, (100 - myDist) / 100));

myVerticalGradient.GradientStops.Add(

new GradientStop(Colors.Green, (100 - myDist) / 100));

myVerticalGradient.GradientStops.Add(

new GradientStop(Colors.Green, 1.0));

lines[num].BeginAnimation(Rectangle.OpacityProperty, myDoubleAnimation);

lines[num].Fill = myVerticalGradient;

SetAngle(num);

SetDistance(distance);

}

}

}

Розглянемо цю частину програми детальніше.

//Оголошуємо змінні

Label angle, distance;

List<Rectangle> lines = new List<Rectangle>();

double ResolWidth, ResolHeight;

В першому рядку оголошуються дві змінні типу Label, що являє собою напис з форматуванням. У типу Label одне з властивостей Content задає строку яка буде виводитися на екран. List<Rectangle> - це список який складається з Rectangle, тобто з прямокутників. Дробові змінні ResolWidth і ResolHeight зберігатимуть розширення екрану по ширині і висоті відповідно.

//Функція яка виводить всі UIElement на екран

void OnScren(params UIElement[] element)

{

foreach(UIElement el in element)

plate.Children.Add(el);

}

Функція додає в дочірні елементи нашого головного полотна plate будь-який елемент інтерфейсу. Кожен елемент інтерфейсу походить від класу UIElement що дозволяє передавати в якості одного об’єкта здавалося б різні елементи.

//Функція яка малює лінію з заданими параметрами

Rectangle DrowLine(int angle, int width, int height,

int right, int bottom, Brush brush)

{

//Створення прямокутника з заданими параметрами

Rectangle rectangle = new Rectangle

{

Width = width,

Height = height,

Fill = brush,

Opacity = 0.15,

};

//Позиціонування відносно контейнера

Canvas.SetBottom(rectangle, bottom);

Canvas.SetRight(rectangle, right);

//Поворот відносно заданої точки

RotateTransform rotate = new RotateTransform(angle)

{

CenterX = rectangle.Width,

CenterY = rectangle.Height / 2

};

rectangle.RenderTransform = rotate;

//Вивід на екран

OnScren(rectangle);

return rectangle;

}

Функція вертає об’єкт Rectangle з заданими параметрами, а саме: кут повороту, ширина, висота, положення відносно контейнера та заливка. Також існує перевантажений варіант цієї функція в якої є лише один параметр – кут повороту.

//Функція яка обчислює розміри об'єктів відносно розширення системи

double AdaptResol(Double length, Sides sides)

{

if (sides == Sides.Height)

return length / 1080 \* ResolHeight;

else return length / 1920 \* ResolWidth;

}

Функція вертає число яке змінене рівно в стільки разів на скільки розширення 1920 на 1080 точок відрізняється від розширення встановленого системою. В параметри передається саме число і сторона, відносно якої буде проводитися масштабування.

//Функція яка створює напис

Label CreateLabel(string content, int left, int bottom)

{

Label label = new Label

{

Content = content,

FontSize = (int)AdaptResol(48, Sides.Height),

Foreground = Brushes.Green,

FontWeight = FontWeights.Bold

};

Canvas.SetBottom(label, bottom);

Canvas.SetLeft(label, left);

return label;

}

Функція вертає об’єкт Label з заданим текстом і положенням в контейнері.

//Функція яка встановлює значення напису Angle

void SetAngle(int ang)

{

string temp = angle.Content.ToString();

temp = temp.Split(':')[0];

temp += ": " + (ang \* 2);

angle.Content = temp;

}

//Функція яка встановлює значення напису Distance

void SetDistance(int dis)

{

string temp = distance.Content.ToString();

temp = temp.Split(':')[0];

if(dis <= 100)

temp += ": " + dis;

else temp += ": " + ">100";

distance.Content = temp;

}

Функції які присвоюють напису заданий текст. Напис містить такий текст: {Angle: [число]}. Потрібно змінювати тільки число, тому потрібні додаткові маніпуляції з стрічкою щоб розділити сам напис і число яке має змінювати функція. Також в функції SetDistance встановлено обмеження на максимальне значення яке може бути присвоїне.

// Функція для створення форми дуг

Path CreatePath(Point point1, Point point2, int bottom, int left)

{

//Створюємо фігуру з заданим початком

PathFigure figure = new PathFigure

{

StartPoint = point1

};

//Додаємо новий сегмент до існуючої фігури що буде її кінцем.

figure.Segments.Add(new ArcSegment(point2,

new Size(1, 1), 180, false, SweepDirection.Clockwise, true));

//Створюємо об’єкт Path за допомогою об’єкта типу PathGeometry

PathGeometry geometry = new PathGeometry();

geometry.Figures.Add(figure);

Path path = new Path

{

Stroke = Brushes.Yellow,

StrokeThickness = 2

};

path.Data = geometry;

Canvas.SetBottom(path, bottom);

Canvas.SetLeft(path, left);

return path;

}

Функція для створення дуги. Дуга будується по двох точках та положенню відносно контейнера.

//Функція яка малює радар

void DrowRadar()

{

for (int i = 0; i <= 180; i += 45)

{

Rectangle rect = DrowLine(i, (int)AdaptResol(900, Sides.Width), 2,

(int)AdaptResol(960, Sides.Width), (int)AdaptResol(70, Sides.Height),

Brushes.Yellow);

rect.Opacity = 1;

}

//Створення дуг різного радіусу для розмітки радару

Path arc20, arc40, arc60, arc80, arc100;

arc20 = CreatePath(new Point(0, (int)AdaptResol(300, Sides.Height)),

new Point((int)AdaptResol(332, Sides.Width), (int)AdaptResol(300, Sides.Height)),

(int)AdaptResol(70, Sides.Height), (int)AdaptResol(794, Sides.Width));

arc40 = CreatePath(new Point(0, (int)AdaptResol(300, Sides.Height)),

new Point((int)AdaptResol(664, Sides.Width), (int)AdaptResol(300, Sides.Height)),

(int)AdaptResol(70, Sides.Height), (int)AdaptResol(628, Sides.Width));

arc60 = CreatePath(new Point(0, (int)AdaptResol(300, Sides.Height)),

new Point((int)AdaptResol(996, Sides.Width), (int)AdaptResol(300, Sides.Height)),

(int)AdaptResol(70, Sides.Height), (int)AdaptResol(462, Sides.Width));

arc80 = CreatePath(new Point(0, (int)AdaptResol(300, Sides.Height)),

new Point((int)AdaptResol(1328, Sides.Width), (int)AdaptResol(300, Sides.Height)),

(int)AdaptResol(70, Sides.Height), (int)AdaptResol(296, Sides.Width));

arc100 = CreatePath(new Point(0, (int)AdaptResol(300, Sides.Height)),

new Point((int)AdaptResol(1660, Sides.Width), (int)AdaptResol(300, Sides.Height)),

(int)AdaptResol(70, Sides.Height), (int)AdaptResol(130, Sides.Width));

//Створення підписів градусної розмітки

Label label1, label2, label3, label4, label5;

label1 = CreateLabel("0°", (int)AdaptResol(45, Sides.Width), (int)AdaptResol(70, Sides.Height));

label2 = CreateLabel("45°", (int)AdaptResol(325, Sides.Width), (int)AdaptResol(700, Sides.Height));

label3 = CreateLabel("90°", (int)AdaptResol(930, Sides.Width), (int)AdaptResol(960, Sides.Height));

label4 = CreateLabel("135°", (int)AdaptResol(1500, Sides.Width), (int)AdaptResol(700, Sides.Height));

label5 = CreateLabel("180°", (int)AdaptResol(1800, Sides.Width), (int)AdaptResol(70, Sides.Height));

//Створення напису за допомогою функції CreateLabel

angle = CreateLabel("Angle: ", (int)AdaptResol(320, Sides.Width), 2);

distance = CreateLabel("Distance: ", (int)AdaptResol(1200, Sides.Width), 2);

//Вивід на екран всіх створених елементів за допомогою функції OnScren

OnScren(arc20, arc40, arc60, arc80, arc100,

label1, label2, label3, label4, label5, angle, distance);

}

Функція яка створює радар з примітивів. Всі розміри елементів радару були відмаштабовані для поточного розширення за допомогою функції AdaptResol.

//Функція яка запускає анімацію вибраної лінії

public void AnimLine(int num, int time, double from, double to, int distance)

{

//Створюємо об’єкт анімації з заданою тривалістю

DoubleAnimation myDoubleAnimation = new DoubleAnimation

{

From = from,

To = to,

Duration = new Duration(TimeSpan.FromMilliseconds(time))

};

//Якщо дистанція більша за 100 приймаємо її за 100

double myDist = distance > 100 ? 100 : distance;

//Створюємо лінійну градієнтну заливку

LinearGradientBrush myVerticalGradient =

new LinearGradientBrush

{

StartPoint = new Point(0, 0.5),

EndPoint = new Point(1, 0.5)

};

myVerticalGradient.GradientStops.Add(

new GradientStop(Colors.Red, 0.0));

myVerticalGradient.GradientStops.Add(

new GradientStop(Colors.Red, (100 - myDist) / 100));

myVerticalGradient.GradientStops.Add(

new GradientStop(Colors.Green, (100 - myDist) / 100));

myVerticalGradient.GradientStops.Add(

new GradientStop(Colors.Green, 1.0));

//Запускаємо анімацію вибраної лінії

lines[num].BeginAnimation(Rectangle.OpacityProperty, myDoubleAnimation);

lines[num].Fill = myVerticalGradient;

//Обновляємо написи

SetAngle(num);

SetDistance(distance);

}

Функція запускає анімацію вибраної лінії з масиву цих самих ліній з заданою амплітудою і тривалістю. Послідовна анімація кожної лінії створює ілюзію переміщення.

//Обробник подій KeyDown для закриття вікна радару

private void MyWinRadar\_KeyDown(object sender, KeyEventArgs e)

{

if (e.Key == Key.Escape) this.Close();

}

Обробник події KeyDown потрібний для закриття вікна WinRadar при натисканні на кнопку Esc.

public WinRadar()

{

InitializeComponent();

//Записується в змінні висота і ширина розширення на теперішній конфігурації системи.

ResolHeight = SystemParameters.PrimaryScreenHeight;

ResolWidth = SystemParameters.PrimaryScreenWidth;

//Створення і додавання ліній в список для подальшої маніпуляції з ними

for (int i = 0; i <= 180; i += 2)

lines.Add(DrowLine(i));

//Малюємо радар за допомогою функції DrowRadar

DrowRadar();

}

Це конструктор вікна WinRadar. Конструктор викликається тоді, коли створюється це вікно. Саме з нього починається створення радару.

Перейдемо до наступної частини програми.

using System;

using System.Windows;

//Підключаємо бібліотеку для зчиткування даних з Serial порту

using System.IO.Ports;

namespace Радар

{

public partial class MainWindow : Window

{

//Оголошуємо змінні порту, вікна радару та інші

SerialPort myPort;

WinRadar winRadar;

string[] ports;

string Data;

public MainWindow()

{

InitializeComponent();

ports = SerialPort.GetPortNames();

AddItems(ports);

selectPort.Text = ports[0];

selectSpeed.Text = "9600";

myPort = new SerialPort();

myPort.DataReceived += MyPort\_DataReceived;

butRadar.IsEnabled = false;

}

//Обробник події DataReceived

private void MyPort\_DataReceived(object sender, SerialDataReceivedEventArgs e)

{

Action action = () =>

{

Data = myPort.ReadExisting();

int angle = Convert.ToInt32(GetParam(Data)[0]);

int distance = Convert.ToInt32(GetParam(Data)[1]);

if(angle % 2 == 0)

winRadar.AnimLine(angle / 2, 1000, 1, 0.15, distance);

};

Dispatcher.Invoke(action);

}

//Функція яка розбиває єдину строку в масив строк

string[] GetParam(string str)

{

return str.Split('.', ',');

}

//Обробник події Click на кнопці старт

private void ButStart\_Click(object sender, RoutedEventArgs e)

{

if (butStart.Content.ToString() == "Пуск")

{

string portName = selectPort.SelectedItem.ToString();

string portRateStr = selectSpeed.SelectedItem.ToString();

butStart.Content = "Стоп";

selectPort.IsEnabled = false;

selectSpeed.IsEnabled = false;

butRadar.IsEnabled = true;

winRadar = new WinRadar();

winRadar.InitializeComponent();

winRadar.Closing += WinRadar\_Closing;

Connect(portName, portRateStr);

}

else

{

butStart.Content = "Пуск";

selectPort.IsEnabled = true;

selectSpeed.IsEnabled = true;

myPort.Close();

butRadar.IsEnabled = false;

}

}

//Обробник події Clossing для форми winRadar

private void WinRadar\_Closing(object sender, EventArgs e)

{

butStart.Content = "Пуск";

selectPort.IsEnabled = true;

selectSpeed.IsEnabled = true;

myPort.Close();

butRadar.IsEnabled = false;

}

//Функція яка відкриває порт для зчитування

void Connect(string name, string speed)

{

myPort.PortName = name;

myPort.BaudRate = Convert.ToInt32(speed);

myPort.Open();

}

//Функція яка додає всі наявні порти в випадаючий список

void AddItems(string[] strs)

{

foreach (string str in strs)

{

selectPort.Items.Add(str);

}

}

//Обробник події Click на кнопці радар

private void ButRadar\_Click(object sender, RoutedEventArgs e)

{

winRadar.Show();

}

}

}

Розглянемо даний код детальніше.

//Оголошуємо змінні порту, вікна радару та інші

SerialPort myPort;

WinRadar winRadar;

string[] ports;

string Data;

Оголошується змінна типу SerialPort яка являє собою об’єкт з якого ми будемо зчитувати інформацію, яку нам буде передавати наш пристрій. Змінна winRadar являє собою посилання через яке ми будемо взаємодіяти з першою частиною коду. Масив стрічок ports ніщо інше як список всіх доступних на даний момент підключень по порту. Змінна Data виконує функцію буфера. В неї буде записуватися останній пакет даних який надійшов з порту.

//Функція яка розбиває єдину строку в масив строк

string[] GetParam(string str)

{

return str.Split('.', ',');

}

Функція розбиває строку вигляду {[angle],[distance].}на масив стрічок {{angle},{distance}}. Потрібна для зручності роботи з даними.

//Функція яка відкриває порт для зчитування

void Connect(string name, string speed)

{

myPort.PortName = name;

myPort.BaudRate = Convert.ToInt32(speed);

myPort.Open();

}

Функція яка відкриває порт для зчитування.

//Функція яка додає всі наявні порти в випадаючий список

void AddItems(string[] strs)

{

foreach (string str in strs)

{

selectPort.Items.Add(str);

}

}

Функція додає в список selectPort всі наявні на даний момент порти для того, щоб ми могли вибрати нам потрібний.

//Обробник події Click на кнопці радар

private void ButRadar\_Click(object sender, RoutedEventArgs e)

{

winRadar.Show();

}

Обробник події Click на кнопці Радар. При натисканні лівої кнопки миші буде виконуватися команда Show() об’єкту winRadar що в свою чергу виведе радар на екран.

//Обробник події DataReceived

private void MyPort\_DataReceived(object sender, SerialDataReceivedEventArgs e)

{

Action action = () =>

{

Data = myPort.ReadExisting();

int angle = Convert.ToInt32(GetParam(Data)[0]);

int distance = Convert.ToInt32(GetParam(Data)[1]);

if(angle % 2 == 0)

winRadar.AnimLine(angle / 2, 1000, 1, 0.15, distance);

};

Dispatcher.Invoke(action);

}

Обробник події DataReceived. Ця подія виникає тоді, коли прибув новий пакет на відкритий порт. В ньому проводиться зчитування даних за допомогою команди ReadExisting. За допомогою функції GetParam пакет розбивається на дві цілі змінні angle і distance. Далі запускається анімація для лінії якій призначений цей пакет. Варто звернути увагу, що процес зчитування проводиться не в головному потоці. Тому для того, щоб дістати інформацію з іншого потоку було використано команду Dispatcher.Invoke. В потік можна одночасно передати лише один об’єкт тому обертаємо нашу частину коду в делегат Action.

//Обробник події Click на кнопці старт/стоп

private void ButStart\_Click(object sender, RoutedEventArgs e)

{

if (butStart.Content.ToString() == "Пуск")

{

string portName = selectPort.SelectedItem.ToString();

string portRateStr = selectSpeed.SelectedItem.ToString();

butStart.Content = "Стоп";

selectPort.IsEnabled = false;

selectSpeed.IsEnabled = false;

butRadar.IsEnabled = true;

winRadar = new WinRadar();

winRadar.InitializeComponent();

winRadar.Closing += WinRadar\_Closing;

Connect(portName, portRateStr);

}

else

{

butStart.Content = "Пуск";

selectPort.IsEnabled = true;

selectSpeed.IsEnabled = true;

myPort.Close();

butRadar.IsEnabled = false;

}

}

Даний обробник подій реалізує логіку взаємодії користувача з кнопками на формі. Якщо кнопка Пуск нажата, то вона перетворюється в кнопку Стоп, кнопка Радар стає активною, список доступних портів заморожується і відкривається порт для зчитування інформації. Це підготовка до запуску радара. Якщо нажата кнопка Стоп, то все відбувається дзеркально першому варіанту.

//Обробник події Clossing для форми winRadar

private void WinRadar\_Closing(object sender, EventArgs e)

{

butStart.Content = "Пуск";

selectPort.IsEnabled = true;

selectSpeed.IsEnabled = true;

myPort.Close();

butRadar.IsEnabled = false;

}

Якщо вікно радару закрилося, то інтерфейс користувача повертається в свій початковий стан та закривається порт.

public MainWindow()

{

InitializeComponent();

//Дістаємо список наявних портів

ports = SerialPort.GetPortNames();

//Додаємо наявні порти в випадаючий список за допомогою функції AddItems

AddItems(ports);

//За замовчуванням залишаємо вибраним порт номер 1

selectPort.Text = ports[0];

//За замовчуванням виставляємо швидкість передачі по порту

selectSpeed.Text = "9600";

//Ініціалізуємо порт

myPort = new SerialPort();

//Підписуємо порт на обробник подій

myPort.DataReceived += MyPort\_DataReceived;

//Деактивовуємо кнопку радар для запобігання передчасного її натискання.

butRadar.IsEnabled = false;

}

Конструктор MainWindow є вхідною точкою для програми.

# Список літератури

1. Мікропроцесорна техніка / Якименко Ю.І., Терещенко Т.О., Сокол Є.І. та ін.]. – К.: ІВЦ "Видавництво "Політехніка"; "Кондор", 2008. – 440 с.
2. Программирование микроконтроллерных плат Arduino/Freeduino / Соммер У. - СПб.: БХВПетербург, 2012. - 256 с.
3. <https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio>
4. <https://uk.wikipedia.org/wiki/%D0%9C%D1%96%D0%BA%D1%80%D0%BE%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%B5%D1%80>
5. <https://studopedia.su/10_12088_zagalniy-algoritm-avtomatizovanogo-proektuvannya.html>
6. <https://uk.wikipedia.org/wiki/Arduino_Nano>
7. <https://wiki.tntu.edu.ua/%D0%A3%D0%BB%D1%8C%D1%82%D1%80%D0%B0%D0%B7%D0%B2%D1%83%D0%BA%D0%BE%D0%B2%D0%B8%D0%B9_%D0%B4%D0%B0%D1%82%D1%87%D0%B8%D0%BA_%D0%B2%D1%96%D0%B4%D1%81%D1%82%D0%B0%D0%BD%D1%96_Arduino_HC_SR04>