ВІЙСЬКОВИЙ ІНСТИТУТ ТЕЛЕКОМУНІКАЦІЙ ТА ІНФОРМАТИЗАЦІЇ

*Інформаційних технологій*

(назва факультету)

*Комп’ютерних інформаційних технологій*

(повна назва кафедри)

*не таємно*

гриф

**Пояснювальна записка**

*до кваліфікаційної роботи*

*бакалавр*

(освітньо-кваліфікаційний рівень)

на тему: *“Програмний модуль автоматизації складення графіку чергувань чергових Факультету та систему оповіщення”*

Виконав: курсант 24 курсу, групи №241

спеціальності

*122 Комп’ютерні науки*

(шифр і назва спеціальності)

*Незнаюхто П.О.*

(прізвище та ініціали)

Керівник *Редзюк Є.В.*

(прізвище та ініціали)

Рецензент *Мацаєнко А.М.*

(прізвище та ініціали)

Київ – 2018

АНОТАЦІЯ

Кваліфікаційна робота бакалавра містить:56 сторінок, 1 таблицю, 26 рисунків, 13 джерел, 3 додаток.

В роботі проведено аналіз можливих сфер застосування програмних додатків що автоматизації процесів та сучасні методи і засоби їх розробки Обгрунтовано необхідність та реалізовано програмний модуль з метою подальшого розширення та застосування у спеціальних підрозділах військових формувань.

Ключові слова: алгоритм, програмний модуль, *Java.*

*ANOTATION*

The qualification paper of the bachelor's degree includes: 56 pages, 1 table, 26 figures, 13 sources, 3 applications.

Possible using areas of ground robotic platforms and modern methods and means of their development are implemented in the work. The necessity and implementation of the land mobile platform management radio channel with the help of a software and hardware complex for the purpose of further application in special units of military formations was substantiated.

Keywords: ground robotized platform, radio channel, Arduino.

# ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

*API – Application Programming Interface (Інтерфейс програми програмування)*

*USB –* *Universal Serial Bus (Універсальна послідовна шина)*

МК  *–* *мікроконтролер*

*SPI –* [*Serial Peripheral Interface*](https://ru.wikipedia.org/wiki/Serial_Peripheral_Interface) *(Серійний периферійний інтерфейс)*

ШІМ *–* широтно-імпульсна модуляція

*RX – Receive Data (*Отримувати дані*)*

*TX –* Transmit Data (Відправляти дані)

*COM –*  послідовний порт комп'ютера

ОЗП *–* оперативно запам'ятовуючий пристрій

ЕОМ  *–* електронна обчислювальна машина

*MB – Megabyte (Мегабайт)*

*POE – Power over Ethernet (Потужність через Ethernet)*

*SM – Serial Monitor (Серійний монітор)*

ПЗУ *– постійний запам'ятовуючий пристрій*

*IDE – Integrated Development Environment(*інтегроване середовище розробки*)*

*OS – Operating system (Операційна система)*

*FBD –* *Function Block Diagram (Блок-схема функцій)*

*ІТ – Information Technologies (Інформаційні технології)*

*HTML – HyperText Markup Language (Мова розмітки HyperText)*

*MOSI – MSI cache coherency protocol (Протокол когерентності кешу MSI)*

*GND – Ground*

ВСТУП

Швидке інформування особового складу під час виконання повсякдених задач є однією з головних вимог сучасності. В даний час в багатьох областях людської діяльності є необхідним правильний, упорядкований розподіл часових і людських ресурсів. Оптимальне планування і організація часто є серйьозоною перевагою і ведуть до підвищення продуктивності праці. Не стала винятком і військова справа. Особливої уваги заслуговує оптимальна організація взаємодії старшого офіцерського складу підрозділу з молодшим офіцерським складом та сержантським складом. Від налагодженого процесу взаємодії між командиром і підлеглим, залежить продуктивність праці як підлеглих так і начальників всіх рівнів, а також ефективний розподіл часу.

Одним із можливих шляхів для вирішення цього завдання є використання різноманітного програмного збезпечення що буде мати в собі не тільки засоби інформування алей функціонал який автоматизує буденні процеси які відбуваються на в військовій сфері.

Головною метою даної роботи є створення системи управління особовим складом підрозділу, що дозволяє викладачам взаємодіяти старшому особовому складу з молодшим і навпаки. Для досягнення поставленої мети необхідно вирішити наступні задачі:

- провести дослідження предметної області;

- провести огляд існуючих аналогів;

- сформулювати вимоги до створюваної системи;

- побудувати загальну архітектуру системи і вибрати засоби для її

реалізації;

- провести проектування бази даних інформаційної системи;

- провести проектування інтерфейсу користувача;

- виконати програмну реалізацію системи;

# РОЗДІЛ 1

# АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Оптимальне планування графіків чергувань є дуже важливим завданням для будь якого військового підрозділ.

Складання графіків чергувань є важливим завданням для військового інституту. Задача складання розкладу розв’язується у багатьох галузях. Насамперед, при плануванні дискретного виробництва, організації пасажирських та товарних перевезень, проектуванні та проведенні навчальних занять у середній, професійно-технічній та вищій школі. В її основу покладено необхідність забезпечення оптимального розподілу робіт серед виконавців, враховуючи просторові та часові обмеження

Звісно ж, у кожній військовій частині, військовому навчальному закладі є людина, яка відповідає за організацію навчального процесу, але дана робота є складною для однієї особи. Велика кількість інформації, яку необхідно зібрати, багато різних умов, яких треба досягти та відповідальність, яка лежить на відповідальній людині є занадто тяжкими факторами для однієї особи, яка виконує це завдання паралельно з основними обов’язками. А враховуючи велику кількість поставлених завдань на військово службовців це завдання стає складнішим з розширенням навчального закладу. Це завдання є складним і відповідальним для людини, але може бути виконане комп’ютером набагато простіше і швидше.

Графік сам по собі залежить від багатьох факторів. Їх можна розділити на об’єктивні(жорсткі) та суб’єктивні(непостійні) параметри. Об’єктивні – це заплановані відрядження військовослужбовців. Суб’єктивні – це побажання військовослужбовців.

Якщо ретельно оцінити проблематику, можна помітити, що завдання складання графіків чергувань є складним для людини, бо треба тримати в голові всі заплановані заходи, посади та побажання військовослужбовців, що означає необхідність в інформації і особистій формі збереження її, та необхідність з’єднати всі ці дані в розклад так, щоб всі були задоволені, але, як відомо, це майже ніколи не можливо досягти.

З іншого боку комп’ютер з цим завданням може легко впоратись. Зберігати інформацію про військовослужбовців в базі даних, застосувати певну функцію до бази, яка побудує графік та відредагує даний розклад відносно побажань людей – проста задача для машини. Задача складання розкладів викликає значний інтерес серед військовослужбовців, які викладають в університетах. Оскільки вони особисто зацікавлені в її розв’язанні та їх кількість є порівняно значною, то розроблена потужна множина відповідних моделей і методів.

Оптимальність у даному контексті визначає підвищення продуктивності праці людини що складає графік. І, якщо для промислових задач забезпечення оптимальності розкладу робіт відігравало визначальну роль, то при складанні розкладу чергувань поняття оптимальності втрачало сенс, оскільки відповідна задача не є повною, а відповідальні особи при його складанні виходять лише із запланованих планів та об’єктивних обмежень і працюють в режимі «реального часу», змінюючи розклад на вимогу чи прохання того або іншого військовослужбовця. Очевидно, що одержаний таким чином графік є далеким від оптимального і викликає нарікання. Зрозуміло також, що оптимальність розклад поняттям неформалізованим, залежним від того, який критерій буде покладений в його основу.

Формування графіку чергувань э складною та клопіткою задачею тому, що вимагає від відповідальної людини врахування багатьох факторів: жорстких вимог та нежорстких. В даній роботі ми будемо зосереджувати свою увагу саме на розкладі занять навчального закладу, для якого жорсткими умовами будуть

* військовослужбовці що викладають,
* кількість нарядів чергувань,
* заплановані заходи,
* відпустки.

До нежорстких умов треба віднести:

* вимоги і побажання військовослужбовців

Жорсткі умови повинні виконуватись завжди, бо інакше розклад є хибним і збитковим. Нежорсткі умови можуть і не виконуватись, але їх виконання напряму впливає на ефективність розкладу з психологічної точки зору.

Розглянем елементи навчального процесу у вищому військовому навчальному закладі.

1.1 Що таке розклад

Розклад – це саме по собі поняття тривіальне з точки зору сучасного життя, а от задачу його формування тяжко назвати тривіальною. За класичним означенням, розклад – це документ установи, який регламентує робочий ритм, визначає часові обмеження всіх робочих процесів і формує оптимальне розділення такого важливого ресурсу як час.

1.1.1 Викладачі

Викладачі – це співробітники навчального закладу, які проводять заняття і є рушійною силою навчального процесу. Кожен викладач має свою посаду, наукове звання та ступінь, які прямо відповідають їх важливості та досягненням на кафедрі. Зрозуміло, чим вища посада, тим більш пріоритетні є вимоги даного викладача.

Викладач – жива людина, в даному випадку військовослужбовець тому процес повинен бути строгий при формуванні розкладу, бо на нього окрім завдань з його навчальних дисциплін ще й припадають ряд завдання пов’язані з його службовими обов’язками.

* + 1. Черговий факультету.

Черговий факультету військовослужбовець офіцерського або старшинського складу який відповідає за дотримання порядку на факультеті, і своєчасного вжиття заходів щодо запобігання правопорушень, а також за виконання іншими обов’язків внутрішньої служби.

* + 1. Начальники курсів

Начальники курсів, курсові офіцери – такі самі важливі учасники процесу, як і викладачі, вони є прямими начальниками слухачів (курсантів, студентів) тому важливо щоб вони були присутні біля своїх підрозділів в командирські дні і могли краще контролювати свій особовий склад.

Ці дві групи людей є тими кого призначають до чергування черговими по факультету. І на ці дві групи військовослужбовців покладаються різні обов’язки і різні навантаження. Тому кожній підгрупи буде будуть відповідати один список вимог і один список пріоритетів. А також буть враховуватися особливості кожного військовослужбовця адже но можливо щоб людина яка находиться в відпустці або в відряджені була призначена до несення служби у добовому чергуванні.

Після цього формується матриця порівнянь остаточних вимог та знаходиться власний вектор найбільшого власного числа. Цей вектор і буде відображати пріоритети та бажання всіх військовослужбовців і буде використаний в цільовій функції.

1.2 Розгляд готових рішень

На даний момент часу є сектор ринку програмного забезпечення систем складання розкладу пропонує велику кількість різних програмних продуктів. Проте здебільшого це програмне забезпечення надає послуги в навчальних закладах різних рівнів, вони мають великий функціонал.

Роздивимся декілька таких програм: «ХроноГраф 3.0 Мастер», «Ніка-Люкс».

«Хрогограф 3.0 Мастер»

«ХроноГраф3.0Мастер»-це оригінальна компютерна технологія. оптимального співвідношення багатофункціональний механізм складання розкладу навчальних дисциплін в школі і оперативного керування навчальним процесом з широкими додатковими можливостями створення і ведення баз даних навчальної діяльності, повноціного обміну даними з другими програмними модулями, а також створення редагування і експорту вивідних даних. Інтерфейс програми показано на Рис1.1

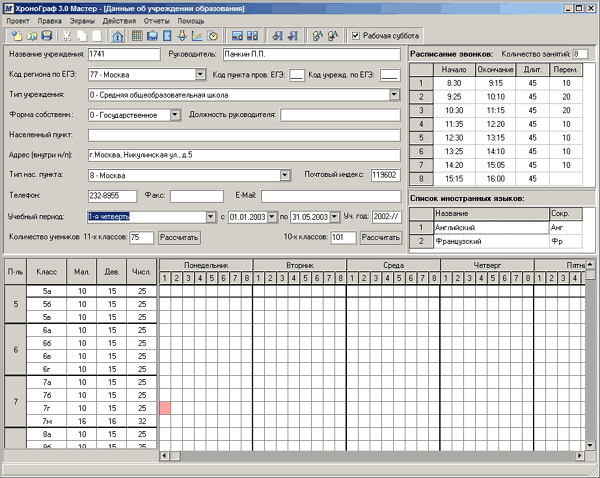


Рис1.1 Інтерфейс програми «ХроноГраф 3.0 Мастер»

З першого погляду програма має простий інтерфейс и налаштовує на те, що робота з нею буде дуже простою и зрозумілою. Но це лише на перший погляд. В процесі використання користувачами були виявлені наступні недоліки:

– програма не являється універсальною і підходить лише для шкіл.

– програма дуже складна в використані. Для того щоб навчитись правілно її використовувати, в пакеті установки є доповнення з 240 сторіноковий збірник по використанню.

Можна зробити висновок що дана програма має великий функціонал, але вона тяжка в використані і є дуже не гнучкою і для вирішення моєї проблеми не зовсім підходить проте саму ідею і деякі її можливості слід запозичити.

1.4 Складання технічного завдання.

Розглянувши продукти, наявні на ринку, визначити їх сильні і слабкі сторони, потрібно написати технічне завдання таким чином, щоб новий продукт мав місце і бути, і не припускалися помилковим схожих продуктів і пропонував більш цікавий і доступний альтернативний варіант для формування чергування. А також був зроблений конкретно під військові цілі. І міг використовуватись в різних військових формуваннях.

Таким чином проектована програма повинна дозволяти:

* складати графік чергувань без пропусків,
* швидка змінювати графік при зміні факторів,
* дозволяти створювати графік без алгоритму,
* Дозволяти адміністратору вносити зміни для самостійної настройки програми,
* створити 3 різні типи користувачів адміністратор, чергові факультету, начальник Факультету. Вони мати свої можливості.
* авторизація для кожного виду користувачі,
* начальник факультету:
* перегляд графіків чергування,
* затверджувати графіки нарядів
* писати, або отримати приватне повідомлення адміністратора.
* отримати файл графіку чергувань формату .docx.
* адміністратор:
* створити, редагувати, видалити користувача.
* створити графіків чергування.
* редагувати графік чергування,
* отримувати, відправляти повідомлення адміністратору, користувачу
* отримувати електроний документ формату .docx.
* користувач:
* отримати файл графіку чергувань формату .docx.
* отримувати, відправляти повідомлення адміністратору.

РОЗДІЛ 2

ВИБІР ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ДЛЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Відколи ПЗ консольного вигляду застаріло, почалася ера програм орієнтованих на користувача. Такий перехід відбувся завдяки появі*graphical user interface*, який і сьогодні використовується в усіх десктопних програмних розробках. Саме *graphical user interface* є найзручнішим засобом взаємодії програми з користувачем, серед усіх відомих на сьогоднішній день, тому при розробці ПЗ програмісти широко використовують елементи *graphical user interface*.

1.2 Загальні алгоритми складання розкладів

1.2.1 Алгоритм розфарбування графа

Завдання складання розкладу можна розглядати як задачу розфарбування графа. Завданням розфарбування графа називають пошук мінімального числа

кольорів (хроматичного числа графа), необхідних для розфарбування вершин

деякого графа так, щоб кожна пара сусідніх вершин була пофарбована в різні кольори. Сама задача пошуку хроматичного числа є NP-повна задача, для вирішення якої в більшості випадків використовуються різні жадібні алгоритми.

Для постановки задачі складання графіку чергувань як завдання розфарбування графа будується граф. Кожна вершина графа – це військовослужбовець який заступити в дане чергування чергування. Якщо між якимись двома вершинами можливі конфлікти, наприклад вибраний військовослужбовець находиться в відряджені то вони з'єднуються ребром, що еквівалентно забороні заступати даному військовослужбовцю до даного наряду. Завдання складання графіку чергувань може бути сформульована як задача мінімізації числа квітів, необхідних для розмальовки графа. Кожен колір відповідає одному періоду графіку чергувань. Зразок розфарбування графів показано на Рис 1.1

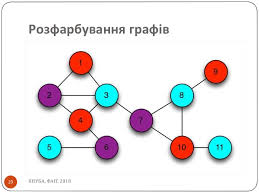


Рисунок 1.1 Зразок схеми розфарбовування графів.

1.2.2 Метод логічного програмування в обмеженнях

Складання розкладу можна розглядати як задачу задоволення обмежень, для вирішення яких завдань розроблено безліч алгоритмів. Виник цілий напрямок в програмуванні - програмування в обмеженнях (constraint programming). Програмування в обмеженнях тісно зв’язане з традиційним логічним програмуванням, в рамках якого воно і було сформовано.

Більшість систем програмування в обмеженнях представляють собою звичайний інтерпретатор Прологу з вбудованим механізмом для вирішення певного класу задач задоволення обмеженням. Програмування в таких системах називають логічним програмуванням в обмеженнях (Constraint Logic Programming або CLP)

Основна ідея вирішення завдань така: програміст визначає деякий безліч змінних x1,. . . , Xn і області їх значень X1,. . . , Xn, описує додаткові обмеження, яким повинні задовольняти змінні, а система знаходить значення змінних, що задовольняють одночасно всім заданим обмеженням.

Основна перевага, що отримується при використанні CLP, - це скорочення простору пошуку, що досягається не шляхом оцінки кожного варіанту розкладу, а за рахунок того, що система сама виключає з розгляду «дороги, що свідомо ведуть в тупик».

1.2.3 Генетичні алгоритми

Наведені вище методи в основному використовують ітераційну техніку поліпшення результатів. При виконанні однієї ітерації вони шукають рішення, краще в околицях поточного рішення, отриманого на попередній ітерації. Якщо таке рішення знайдено, воно стає поточним і починається нова ітерація. Це продовжується до тих пір, поки не виконається правило зупинки: приріст цільової функції не зменшиться практично до нуля або не виконається задану кількість ітерацій. Звичайнож, такі методи здійснюють пошук тільки локальних оптимумів, причому положення знайденого оптимуму залежить від стартової точки, а глобальний оптимум може бути знайдений тільки випадково. Щоб підвищити ймовірність знаходження глобального оптимуму, пошук проводиться багаторазово з різними початковими точками. Час пошуку при цьому істотно збільшується.

Тому зараз є інтерес розробки алгоритмів, що зберігають переваги описаних методів і вільних від зазначеного недоліку. До таким алгоритмам відносяться генетичні алгоритми основані на (нейроній сітці).

Генетичні алгоритми - це стохастичні евристичні оптимізаційні методи, основна ідея яких була взята з теорії еволюційного розвитку видів [26]. Основним механізмом еволюції є природний відбір: більш пристосовані особини мають більше шансів на виживання і розмноження. Вони дають більше потомства, ніж менш пристосовані особини. Завдяки передачі генетичної інформації нащадки успадковують від батьків основні їх якості. Носіями генетичної інформації особини є молекули ДНК. При злитті двох батьківських статевих клітин їх ДНК взаємодіють, утворюючи ДНК нащадка. Основний спосіб взаємодії - кросинговер, при якому ДНК предків діляться на дві частини, а потім обмінюються своїми половинками. Вплив середовища, наприклад, радіоактивність, може викликати мутації (зміни) генів в статевих клітинах одного з батьків. Мутовані гени можуть бути передані нащадку, і він набуде нових властивостей. Нові властивості можуть виявитися корисними для даного виду, підвищуючи його пристосованість, і тоді ці властивості збережуться в даному виді.

Першим кроком при розробці математичної моделі, заснованої на генетичному алгоритмі, є розробка структури хромосоми, в якій буде зберігатися рішення. У нашому випадку такий «хромосомою» є графік чергувань. Обрана структура повинна враховувати всі особливості і обмеження, що пред'являються до шуканого рішення, а також те, що від її вибору безпосередньо залежать реалізації алгоритмів кросинговеру і мутації. В Кінець кінцем, вибір «хромосоми» впливає не тільки на швидкість, а й на збіжність алгоритму взагалі.

Одним з найбільш зручних уявлень вирішення даної завдання є тривимірна матриця, по осях i, j, k якій відображають відповідно групу військовослужбовців, добове чергування, заплановані заходи: відпустки відрядження. Елементом матриці є запит на постановку одного з i - тією групою

військовослужбовців даними викладачем з даної дисципліни в j – добове чергування в k – обмеження що не дають можливості заступити до добового чергування.

Така структура хромосоми зручна тим, що вже на етапі завдання початкових даних можна виключити свідомо невдалі рішення, заблокувавши відповідні осередки.

На наступному кроці алгоритму створюється початкова популяція, розмір якої залежить від розмірності задачі і становить зазвичай кілька сотень рішень.

Для організації більш оптимального процесу необхідно створити спрямовуючу силу розвитку популяції. В якості такої сили виступає вимога мінімізації цільової функції (в термінах генетичних алгоритмів, - фітнес-функції). Як фітнес-функції можна використовувати адитивний показник оптимальності, заснований на штрафах, встановлюваної кожному рішенню за будь-якої незручний момент в розкладі. Важлива особливість такого вибору - можливість налаштувати алгоритм на вирішення конкретного завдання. Це досягається варіюванням коефіцієнтів, що призводить до зміни пріоритетів при пошуку оптимального розкладу. Загальна схема генетичного алгоритму представлена на Рисунку 3.

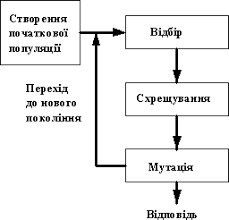


Рисунок 3 Схема стандартного генетичного алгоритму.

# 

РОЗДІЛ 2

ПІДГОТОВЧИЙ ЕТАП, ВИБІР ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ДЛЯ РОЗРОБКИ ПРОГРАМНОГО МОДУЛЮ

Робота над проектом починається з підготовчого етапу, Ціль цього етапу полягає в тому, щоб на основі проблемної теми створити концепцію майбутньої системи і, відштовхуючись від цієї концепції, провести оцінку затребуваності і реалізованості проекту. А також, цей етап фактично є стадією підготовки потенційного виконавця до конкурсу, включаючи формування необхідної документації.

Не потрібно витрачати час і ресурси на проект, чия концепція вважається непотрібною або нездійсненною. Такий проект повинен бути завершений. У ряді випадків потрібна деяка ітеративна робота з постановкою завдання по корекції концепції проекту, поки або не буде досягнутий прийнятний баланс вимог і можливості виконавця, або не буде прийнято рішення про завершення робіт.

Проект, концепція якого виглядає прийнятною для реалізації, виходить на етап розробки вимог. На цьому етапі я повинен сформувати перелік усіх явних і прихованих потреб до програмного модулю. Часто виявляється, що є ідея проекту і немає чітко сформованих вимого до нього або його потреби вступають в протиріччя між собою, або з можливостями виконавця. Цілями етапу є виявлення всіх прихованих потреб, вирішення конфліктів вимог, формування цілісного технічного рішення і аналіз реалізованості підготовленого рішення.

Іноді уточнення вимог призводить до перегляду концепції проекту. Якщо після уточнення всіх вимог не вдається знайти прийнятного технічного рішення, проект доводиться згортати або відкладати на деякий час в очікуванні більш прийнятних.

Коли буде знайдене технічне рішення, виконавець приступає до розробки архітектури майбутньої системи. Мета етапу - визначення верхнерівневого логічної і фізичної архітектури, повністю покриває всі вимоги замовника. При розробці архітектури проводиться та уточнення концепції, вимог і попереднього технічного рішення, що дає можливість попередити найбільш небезпечні ризики

Після завершення проектування архітектури необхідно знову провести ревізію основних параметрів проекту і вирішити, чи в змозі виконавець завершити проект. Корисно на стадії розробки архітектури відмовитися від зайвих і занадто громіздких функцій. Оптимізація архітектурного рішення часто допомагає вписатися в прийнятні параметри проекту. В інших випадках потрібно більш радикальне скорочення функціоналу системи, що розробляється. Однак навіть зупинка проекту на цій стадії, якщо вона відбувається з вагомих причин, повинна сприйматися як перемога: продовження робіт в такому разі може привести тільки до ще більших втрат.

Якщо баланс був знайдений, і вдалося створити прийнятну архітектуру системи, виконавець може переходити до реалізації і постачання системи. Реалізація може проходити в один або кілька етапів. Для невеликих проектів одноетапна поставка всього функціоналу системи може бути цілком прийнятною. Однак, чим більше проект, тим вище залежно підсистем всередині створюваної системи. У цих умовах слід ділити реалізацію на кілька етапів так, щоб в кінці кожного етапу розробник мала готовий до постачання продукт.

Після поставки повністю завершеною системи проект задуманого ПЗ зазвичай переходить до етапу дослідної експлуатації. Мета цього етапу полягає в перевірці якості роботи розробленої системи в реальних умовах експлуатації. Як правило, на цьому етапі виконавець спільно з вимір кількісних метрик, що дозволяють визначити якість створеної системи. В першу чергу перевіряються функціональні характеристики якості, потім - нефункціональні. При наявності невідповідностей виконавець коригує код системи.

Діаграма діяльності на підготовчому етапі розробки ПЗ представлена на Рисунку???

Діаграма діяльності на підготовчому етапі розробки Рис 1.

Формування цілей

проекту

Розробка концепції

проекту

Аналіз потрібності проекту

Потрібна?

Так

Ні

Попередня оцінка ризиків

Попідготовка початкового ТЗ

Підготовка початкового плану роботи

Оцінкака трудозатр і необхідних ресурсів

Аналіз реалізуймості проекту

Ні

Закінечення проекту

Створення пректу

Так

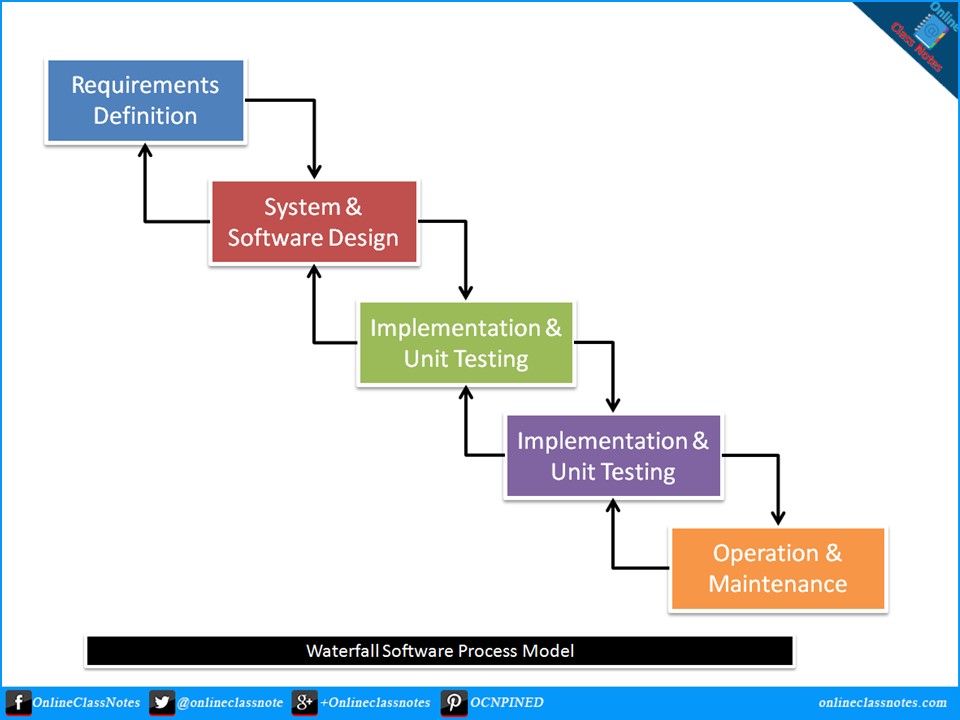
Можлива

для створення

**Методології розробки**

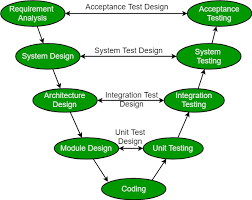
Розробка програмного продукту знає багато гідних методологій - інакше кажучи, усталених *best practices*. Вибір залежить від специфіки проекту, системи бюджетування, суб'єктивних переваг і навіть темпераменту виконавця.

* + 1. **Waterfall Model**



Одна з найстаріших методологій, передбачає послідовне проходження стадій, кожна з яких має завершитися повністю до початку наступної. У моделі Waterfall легко керувати проектом. Завдяки її жорсткості, розробка проходить швидко, вартість і термін заздалегідь визначені. Але це палиця з двома кінцями. Каскадна модель буде давати відмінний результат тільки в проектах з чітко і заздалегідь визначеними вимогами і способами їх реалізації. Нет возможности сделать шаг назад, тестирование начинается только после того, как разработка завершена или почти завершена. Продукты, разработанные по данной модели без обоснованного ее выбора, могут иметь недочеты (список требований нельзя скорректировать в любой момент), о которых становится известно лишь в конце из-за строгой последовательности действий. Стоимость внесения изменений высока, так как для ее инициализации приходится ждать завершения всего проекта. Тем не менее, фиксированная стоимость часто перевешивает минусы подхода. Проти, фіксована вартість часто переважує мінусі підходу. Виправлення усвідомленіх в процесі создания недоліків можливо, І, за моїм досвідом, потребує від одного до троьх Додаткових Угод до вимог з невеликим ТЗ.

* + 1. **V-Model**

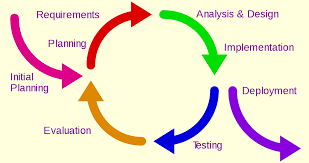
****

Успадкувала структуру «крок за кроком» від каскадної моделі. V-образна модель застосовна до систем, яким особливо важливо безперебійне функціонування. Наприклад, прикладні програми в клініках для спостереження за пацієнтами, інтегроване ПО для механізмів управління аварійними подушками безпеки в транспортних засобах і так далі. Особливістю моделі можна вважати те, що вона спрямована на ретельну перевірку і тестування продукту, що знаходиться вже на початкових стадіях проектування. Стадія тестування проводиться одночасно з відповідною стадією розробки, наприклад, під час кодування пишуться модульні тести.

Проекти на основі V-методології. виконується за чітким ТЗ, але в нього включений значний етап тестування: зручності інтерфейсу, функціонального, навантажувального та в тому числі інтеграційного, яке повинно підтверджувати,

що кілька компонентів від різних виробників разом працюють стабільно.

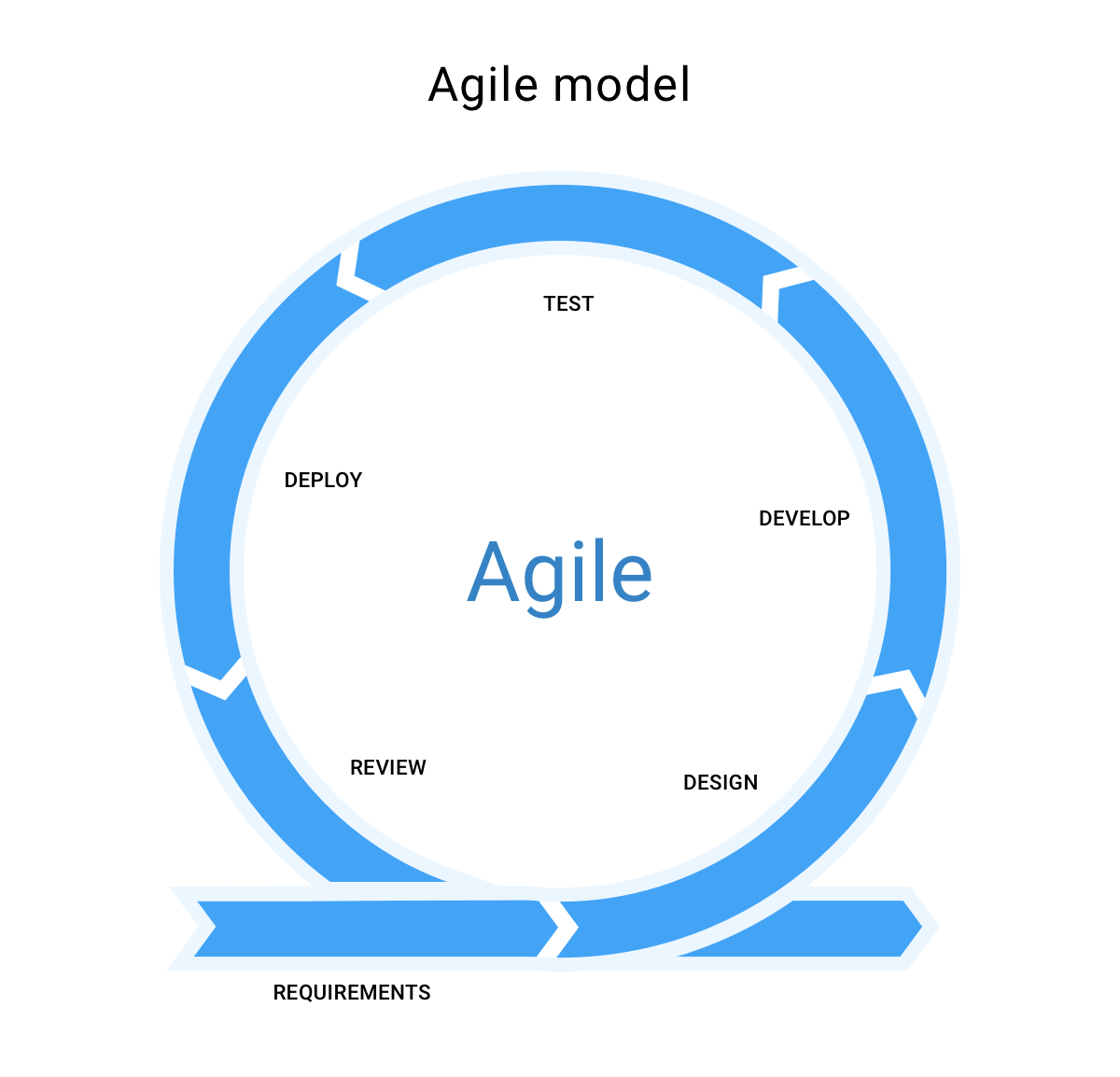
* + 1. «Incremental Model» (інкрементна модель)



У инкрементной моделі повні вимоги до системи діляться на різні збірки. Термінологія часто використовується для опису поетапної зборки ПО. Мають місце кілька циклів розробки, і разом вони складають життєвий цикл «мульти-водоспад». Цикл розділений на більш дрібні легко створювані модулі. Кожен модуль проходить через фази визначення вимог, проектування, кодування, впровадження та тестування. Процедура розробки по инкрементной моделі передбачає випуск на першому великому етапі продукту в базовій функціональності, а потім вже послідовне додавання нових функцій, так званих «інкрементів». Процес триває до тих пір, поки не буде створена повна система.

Інкрементний моделі використовуються там, де окремі запити на зміну ясні, можуть бути легко формалізовані і реалізовані.

* + 1. «Agile Model» (гнучка методологія розробки)



У «гнучкій» методології розробки після кожної ітерації замовник може спостерігати результат і розуміти, задовольняє він його чи ні. Це одна з переваг гнучкої моделі. До її недоліків відносять те, що через відсутність конкретних формулювань результатів складно оцінити трудовитрати і вартість, необхідні на розробку. Екстремальне програмування (XP) є одним з найбільш відомих застосувань гнучкої моделі на практиці.

Методологія підходить для великих або націлених на тривалий життєвий цикл проектів, постійно адаптуються до умов ринку. Відповідно, в процесі реалізації вимоги змінюються. Варто згадати клас творчих людей, яким властиво генерувати, видавати і випробувати нові ідеї щотижня або навіть щодня. Гнучка розробка найкраще підходить для цього психотипу керівників. Внутрішні стартапи компанії ми розробляємо по *Agile*.

* 1. Пректування

Проектування – процес створення проекту, прототипу, прообразу майбутнього об'єкта, стану та способів його виготовлення. У проектуванні застосовують системний підхід, який полягає у встановлені структури системи, типу зв'язків, визначені атрибутів, аналізуванні впливів зовнішнього середовища.

Проектування інформаційних систем охоплює три головні області:

− проектування об’єктів даних, які будуть реалізовані в базі даних;

− проектування програм, екранних форм, звітів, які будуть забезпечувати виконання запитів до даних;

− проектування певного середовища або технології, тобто топології мережі, конфігурації апаратних засобів, використовуваної архітектури (файл серверної або клієнт-серверної), паралельної обробки, розподіленої обробки даних тощо.

Мета проектування полягає у забезпеченні ефективного функціонування інформаційної системи, а також взаємодії користувачів і розробників інформаційної системи. Саме якісне проектування дозволяє створити систему, яка здатна працювати за постійним вдосконаленням своїх технічних, програмних, інформаційних складових і розширювати спектр реалізованих управлінських функцій. У процесі проектування удосконалюється як організація

провідної діяльності, так і організація управлінських процедур. Основу проекту складають такі компоненти:

– методологія проектування (були приведені вище);

– технологія проектування;

– стандарти й методики проектування;

– інструментальні засоби проектування (або CASE-засоби).

Між вказаними компонентами діє щільний зв’язок, реалізація методології здійснюється через конкретні технології, кожна технологія підтримується відповідними стандартами й методиками, а інструментальні засоби забезпечують реалізацію процесів проектування, що описані у методиках та стандартах.

Методологія проектування – це організація процесу розроблення системи й управління цим процесом, яка гарантує дотримання вимог як до самої системи, так і до характеристик щодо процесу її розроблення.

Парадигми проектування інформаційних систем. Існують дві головні парадигми проектування, що реалізують два різних підходи до опису систем:

– структурна (процесно-орієнтована);

– об’єктно-орієнтована.

Перша концепція має в основі на каскадну модель життєвого циклу інформаційної системи. Друга концепція заснована на ітеративній моделі життєвого циклу.

Розглянемо головні відмінності та особливості використання цих підходів. Структурний аналіз Структурний аналіз (від англ. Structured Analysis або SA) і структурне проектування (або SD, від англ. Structured Design) бере свій початок від структурного програмування і класичного системного аналізу . Сутність структурного підходу до розробки ІС полягає в її декомпозиції (розбитті) на функції, що підлягають подальшої автоматизації система розбивається на функціональні підсистеми, ті зі свого боку розподіляються на під функції, завдання тощо. Процес розбиття триває до рівня деталізації процедур. До того ж автоматизована система зберігає цілісне уявлення, коли всі складові компоненти взаємопов’язані та мають загальне підґрунтя

Під час програмного модулю системи у напряму “bottom-up”, тобто від окремих завдань до всієї системи цілісність втрачається, виникають проблеми у разі інформаційного стикування окремих компонентів. Усі найпоширеніші методології структурного підходу базуються на певних загальних принципах, головними з яких є такі:

− принцип “divide et impera” тобто підхід до вирішення складних проблем шляхом їхнього розбиття на низку менших та незалежних завдань, шо є легкими для сприйняття та практичної реалізації;

− принцип ієрархічного упорядкування;

− принцип організації складових частин проблеми та упорядкування їх у вигляді ієрархічної деревоподібної структури з додаванням нових деталей на кожному рівні.

Виділення цих двох базових принципів не означає, що інші принципи є другорядними, оскільки ігнорування кожного з них може призвести до непередбачуваних наслідків, зокрема до провалу проекту загалом. Головними з додаткових принципів є такі:

− принцип абстрагування, що полягає у виділенні ключових аспектів системи і абстрагування від несуттєвих;

− принцип формалізації, що полягає в необхідності чіткого методичного підходу до вирішення проблеми;

− принцип несуперечності, що полягає в обґрунтованості й узгодженості елементів;

− принцип структурування даних, він полягає у тому, що забезпечення: архітектуру ПЗ, структурні схеми програм і діаграми екранних форм. Вказані моделі в сукупності дають повне описання ІС незалежно від того, чи вона є існуючою чи тільки розробляється.

Об’єктно-орієнтований аналіз або OOAD (Object-oriented analysis and disign) – це підхід до інженерії програмного забезпечення, що розглядає систему

як групу взаємодіючих об’єктів. Об’єктно-орієнтований аналіз (Object-oriented analysis, OOA) використовує методи об’єктного моделювання для аналізу функціональних вимог до системи

Об’єктно–орієнтоване проектування, ООП (Object–oriented design, OOD) має за мету розробити аналітичні моделі процесів для подальшого створення специфікацій для їхньої реалізації. Концептуальною основою OOП є об’єктна модель, яка будується з урахуванням принципів абстрагування, інкапсуляції, модульності, ієрархії, типізації, паралелізму, стійкості. Головними поняттями об’єктно-орієнтованого підходу є об’єкт і клас.

Об’єкт – це визначена сутність, що відповідає певному предмету або явищу й характеризується класом, станом і поведінкою. Для цих взаємодіючих між собою об’єктів можна створити різні моделі, що характеризують їхню статичну структуру або динамічну поведінку й розгортання в реальній роботі (run–time deployment).

Клас – це множина об’єктів, що пов’язані спільною структурою та поведінкою.

Наступну групу важливих понять об’єктного підходу складають поліморфізм (здатність класу належати більш ніж одному типу) та успадкування (побудова нових класів на основі існуючих із можливістю додавання або коригування даних та методів). У сучасній практиці використовують велику кількість об’єктно орієнтованих методів проектування.

Тепер розглянемо сучасні мови програмування, як інструмент для реалізації нашого завдання.

Мова програмування – формальна знакова система, призначена для запису комп'ютерних програм. Мова програмування визначає набір лексичних, синтаксичних і семантичних правил, що задають зовнішній вигляд програми і дії, які виконає виконавець (комп'ютер) під її управлінням.

Всі формалізовані мови, призначені для створення, описи програм і алгоритмів для розв'язування задач на комп'ютерах, діляться на дві основні категорії:

− мови програмування низького рівня (список наведено нижче);

− високого рівня.

Поговоримо про кожну з них окремо. Низько рівневі мови призначені для створення машинних команд для процесорів. Головна їхня перевага в тому, що вони використовують мнемонічні позначення, тобто замість послідовності нулів і одиниць (з двійкової системи числення) комп'ютер запам'ятовує осмислене скорочене слово з англійської мови. Найвідоміші мови низького рівня – це Аssembler (рис 2.4.) (існує декілька підвидів цієї мови, кожен з яких має багато спільного, а відрізняється лише набором додаткових директив і макросів), CIL (доступний в платформі .Net) і Байт – код JAVA.

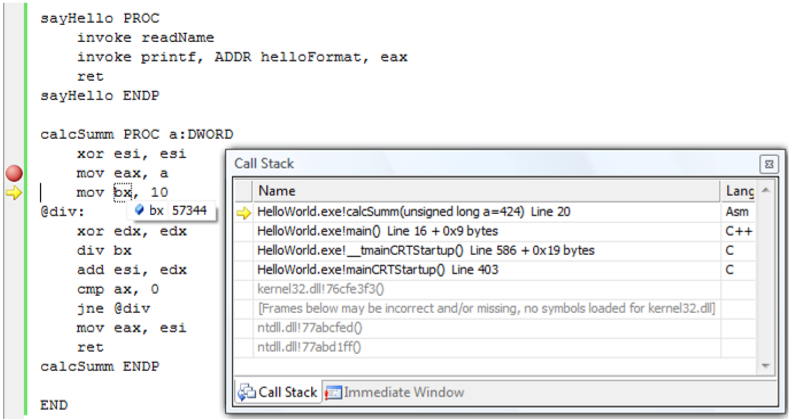


Рисунок. 2.4. Приклад лістингу мови Assembler

Високо рівневі мови створені для зручності та більшої ефективності додатків, вони є повною протилежністю низько-рівневих мов. Їх відмінна риса – наявність смислових конструкцій, які ємко і коротко описують структури і алгоритми роботи програм. У мовах низького рівня їх опис на машинному коді було б занадто довгим і незрозумілим. Мови ж високого рівня володіють незалежністю від платформи. Замість них функцію транслятора здійснюють компілятори: вони переводять текст програми елементарні машинні команди.

Розглянемо сучасні популярні мови програмування, що дозволять вирішити наше завдання з максимальною ефективністю.

Java – об'єктно-орієнтована мова програмування, розроблена компанією Sun Microsystems. Програми Java зазвичай компілюються в спеціальний байткод, тому вони можуть працювати на будь-який віртуальній Java-машині (JVM) незалежно від комп'ютерної архітектури

C# відноситься до сім'ї мов із Cі – подібним синтаксисом, з них її синтаксис найбільш близький до C++ і Java. Мова має статичну типізацію, підтримує поліморфізм, перевантаження операторів (у тому числі операторів явного і неявного приведення типу), делегати, атрибути, події, властивості, узагальнені типи і методи, ітератори, анонімні функції з підтримкою замикань, винятки, коментарі у форматі XML.

Python – високо-рівнева мова програмування загального призначення з акцентом на продуктивність розробника і читаність коду. Синтаксис ядра Python мінімалістичний. У той же час стандартна бібліотека включає великий обсяг корисних функцій.

Ruby – динамічна, рефлективна, інтерпретована високо-рівнева мова програмування для швидкого і зручного об'єктно-орієнтованого програмування. Мова має незалежну від операційної системи реалізацію багато поточності, жорсткою динамічною типізацією, сміттярем і багатьма іншими можливостями. Ruby близька за особливостями синтаксису до мов Perl і Eiffel, за об'єктноорієнтованим підходом - до Smalltalk. Також деякі риси мови взяті з Python, Lisp, Dylan та CLU.

Звернемося до двійки лідерів. Кожен з них, природно, має переваги і недоліки. *Java* дозволяє створювати дуже компактні рішення. Наявність значної кількості бібліотек і документації надає переваги їй в цьому, *C#* у цьому плані програє. У швидкості роботи програм написаних на *Java* за рахунок своєї віртуальної машини є повільнішим, але має значну фору в тому, що для забезпечення сумісності з різними платформами з боку розробника кінцевого продукту потрібні мінімальні зусилля. А це значить що *Java* э крос платформеною мовою програмування. Кількість різних бібліотек написаних для цієї мови є самим більшим адже вона є самою популярною серед розробників, так що в цьому питанні вона явно випереджає C#. C# теж схожа на Java – для роботи програм, написаних цією мовою, також потрібна віртуальна машина. Проте вона є швидшою, але має великий мінус працює тільки на операційній системі *Windows*. Підтримка від сторонніх розробників в неї меша через закритий код, на відміну від *Java.* Одже провівше детальніший розбір було вирішено використовувати мову програмування *Java.*

Відколи ПЗ консольного вигляду застаріло, почалася ера програм орієнтованих на користувача. Такий перехід відбувся завдяки появі GUI, який і сьогодні використовується в усіх десктопних програмних розробках. Саме GUI є найзручнішим засобом взаємодії програми з користувачем, серед усіх відомих на сьогоднішній день, тому при розробці ПЗ широко використовується елементи GUI.

Підсистеми графічного виводу є, напевно, найбільш несумісними елементами різних операційних систем. Три основних інтерфейси, X-Window, Win32 GDI і Cocoa (інтерфейс MacOS) нічого не хочуть знати один про одного. Проблему сумісності вирішують різні крос-платформні бібліотеки, які, крім того, реалізують набори елементів інтерфейсу високого рівня (віджетів). Оскільки розроблювати графічний інтерфейс користувача за допомогою високо-рівневих бібліотек простіше і зручніше, ніж за допомогою графічного інтерфейсу операційної системи, ми використаємо крос-платформне рішення.

В нашому випадку було вибрано *JavaFX. JavaFX -* це програмна платформа для створення та доставки додатків для настільних комп'ютерів, а також багатих інтернет-додатків (*rich Internet applications*), які можуть працювати на різних пристроях. *JavaFX* призначений для заміни *Swing* як стандартної бібліотеки *GUI* для *Java SE*, але обидва будуть включені в доступному для огляду в майбутньому. *JavaFX* підтримує настільні комп'ютери та веб-браузери в *Microsoft Windows, Linux і macOS.*

*Rich Internet Applications* – це ті веб-додатки, які надають подібні функції та досвід роботи з настільними програмами. Вони пропонують кращий візуальний досвід у порівнянні з звичайними веб-додатками для користувачів. Ці програми поставляються у вигляді плагінів для браузерів або як віртуальна машина і використовуються для перетворення традиційних статичних програм у більш розширені, рідкісні, анімовані та привабливі програми.

Для розробки додатків на стороні клієнта з багатими функціями програмісти залежали від різних бібліотек, щоб додати такі функції, як Media, UI, Web, 2D і 3D, тощо. JavaFX включає всі ці функції в єдину бібліотеку. На додаток до цього, розробники можуть також отримати доступ до існуючих функцій бібліотеки Java, таких як Swing.

JavaFX надає багатий набір графічних і медіафайлів, а також використовує сучасний графічний процесор за допомогою апаратно прискореної графіки. JavaFX також надає інтерфейси, за допомогою яких розробники можуть поєднувати графічну анімацію та контроль інтерфейсу.

Особливості JavaFX:

* Написана на Java - Бібліотека JavaFX написана на Java і доступна для мов, які можуть бути виконані на JVM, включаючи Java, Groovy і JRuby. Ці програми JavaFX також є незалежними від платформи.
* FXML - JavaFX має мову, відому як FXML, яка є мовою розмітки HTML, подібною до декларативної. Єдина мета цієї мови - визначити інтерфейс користувача.
* Scene Builder - JavaFX надає програму з назвою Scene Builder. Інтегруючи цю програму в такі середовища, як Eclipse і NetBeans, IntelliJ IDEA, користувачі можуть отримати доступ до інтерфейсу дизайну перетягування, який використовується для розробки додатків FXML
* Сумісність з Swing - У додатку JavaFX можна вбудовувати вміст Swing за допомогою класу Swing Node. Також можна оновити існуючі програми Swing за допомогою функцій JavaFX, таких як вбудований веб-вміст і багаті графічні носії.
* CSS, як Styling - JavaFX надає CSS подібний стиль. Використовуючи це, можливо поліпшити дизайн програми з простого знання CSS.
* Інтегрована графічна бібліотека - JavaFX надає класи для 2d і 3d графіки.
* Конвеєр графіки - JavaFX підтримує графіку, засновану на графічному конвеєрі з апаратним забезпеченням, відомому як Prism. При використанні з підтримуваною графічною картою або графічним процесором він пропонує плавну графіку. У випадку, якщо система не підтримує графічну карту, призма за замовчуванням позначає стек рендеринга програмного забезпечення.

Тепер розглянемо середовища розробки які представлені на сучасному ринку і які дозволять нам створити потрібний програмний додаток.

*IntelliJ IDEA* – це э інтегрованим середовищем розробки Java (IDE) для розробки комп'ютерного програмного забезпечення. Він розроблений компанією JetBrains (раніше відомий як IntelliJ) і доступний як ліцензований випуск спільноти Apache 2, а також у власному комерційному виданні. Обидва можна використовувати для комерційного розвитку.

Eclipse є інтегрованим середовищем розробки (IDE), що використовується в комп'ютерному програмуванні, і є найбільш широко використовуваним Java IDE. Він містить базове робоче місце і розширювану систему плагінів для налаштування середовища. Eclipse написано в основному на Java, і його основне використання - для розробки додатків Java, але також може використовуватися для розробки додатків на інших мовах програмування.

NetBeans є інтегрованим середовищем розробки (IDE) для Java. NetBeans дозволяє розробляти програми з набору модульних компонентів програмного забезпечення, що називаються модулями. NetBeans працює на Windows, MacOS, Linux і Solaris. На додаток до розробки Java, він має розширення для інших мов, таких як *PHP, C, C ++, HTML5,* і *JavaScript*. Програми, засновані на NetBeans, включаючи *IDE NetBeans,* можуть бути розширені сторонніми розробниками.

*JavaFX Scene Builder* надає середовище візуального макета, що дозволяє швидко розробляти користувальницькі інтерфейси (UI) для додатків JavaFX без необхідності писати будь-який код. Це дозволяє просто перетягувати та опускати елементи графічного інтерфейсу (GUI) до сцени JavaFX. Під час створення макета інтерфейсу користувача автоматично створюється код FXML для макета. JavaFX Scene Builder надає простий, але інтуїтивно зрозумілий інтерфейс, який може допомогти навіть непрограмістам швидко прототипувати інтерактивні програми, які з'єднують компоненти графічного інтерфейсу з логікою програми.

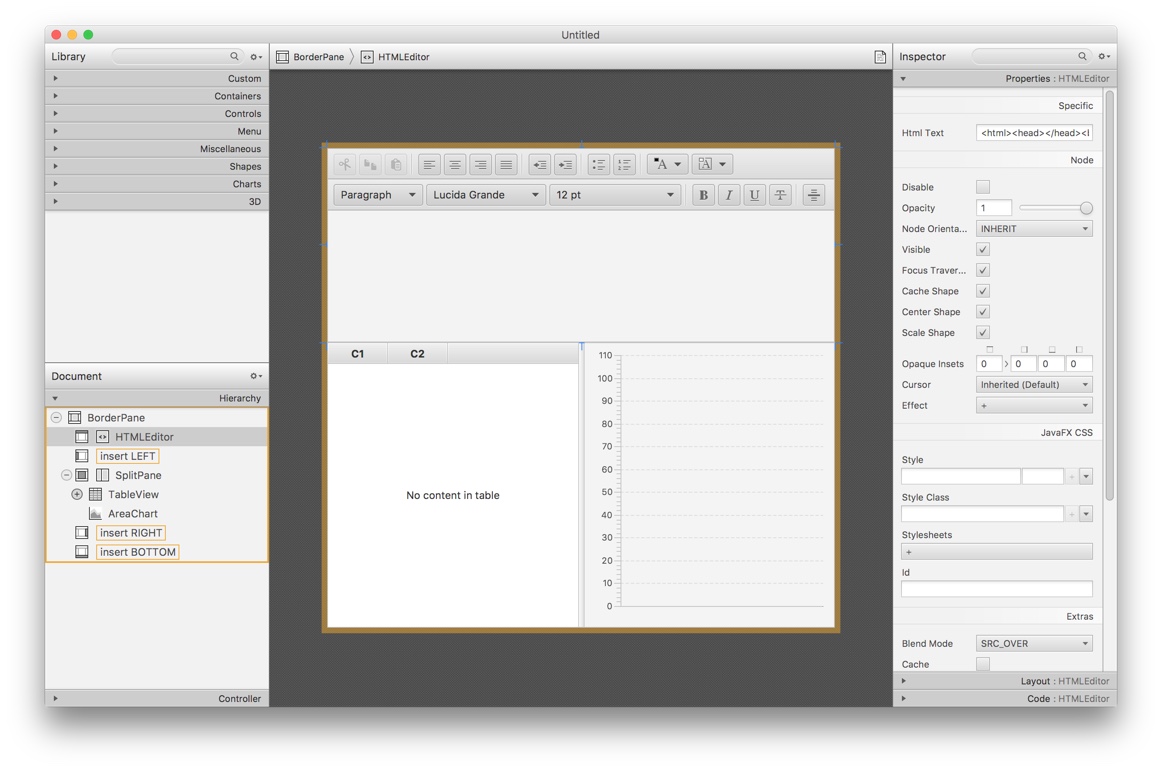
****

Рисунок 2.6 Розробка інтерфейсу користувача засобами *Scene Builder*

Інтерфейс користувача (UI), в області промислового дизайну взаємодії людини з комп'ютером, є простором, де відбуваються взаємодії між людьми і машинами. Метою цієї взаємодії є забезпечення ефективної роботи та управління машиною від людського кінця, у той час як машина одночасно подає інформацію, що допомагає оператору приймати рішення.

Збереження даних

Можна з великим ступенем вірогідності стверджувати, що більшість додатків, які призначені для виконання хоча б який-небудь корисної роботи, тим чи іншим чином використовують структуровану інформацію або, іншими словами, впорядковані дані. Такими даними можуть бути, наприклад, списки замовлень на той чи інший товар, списки пред'явлених і сплачених рахунків або список телефонних номерів ваших знайомих. Звичайне розклад руху автобусів в вашому місті - це теж приклад упорядкованих даних.

База даних – це дуже зручна штука для роботи з стуктурованою інформацією. Багато баз даних - це дуже старі технології. Вони розробляються останні півстоліття, в 70-х роках вже були бази даних, які працювали за схожими принципами, що і зараз.

При комп'ютерній обробці інформації впорядковані якимось чином дані прийнято зберігати в базах даних - особливих файлах, використання яких разом зі спеціальними програмними засобами дозволяє користувачеві як переглядати необхідну інформацію, так і, в міру необхідності, маніпулювати нею, наприклад, додавати, змінювати, копіювати, видаляти, сортувати і т.д, дляуправлыння базами даних використовуються системи керування базами даних (скорочено СКБД).

Таким чином, дати просте визначення базам даних можна дати наступним чином. База даних - це набір інформації, організованої тим, чи іншим способом.

Системи керування базами даних це програмне забезпечення яке використовується для створення та роботи з базами даних. Головна функція СУБД - це управління даними (які можуть бути як у зовнішній, так і в оперативній пам'яті). СУБД обов'язково підтримує мови баз даних, а також відповідає за копіювання та відновлення даних після будь-яких збоїв.

Що стосується класифікації баз даних, то тут можливі різні варіанти.

Наприклад, можна розділити бази по моделі даних: ієрархічні (мають деревоподібну структуру), мережеві (за своєю структурою схожі на ієрархічні), реляційні (використовуються для керування базами даних), об'єктно-орієнтовані (використовуються для об'єктної моделі даних) і об'єктно реляційні (якесь злиття реляційного і об'єктно-орієнтованого виду баз даних).

Інформація, яка зберігається в базах даних, не обмежується тільки текстовими або графічними файлами - сучасні версії СУБД підтримують також формати аудіо і відеофайлів.

Реляційні СКБД є одними з найпоширеніших систем. Вони являють собою таблиці, у яких кожен стовпець (який називається "*field*") впорядкований і має певне унікальну назву. Послідовність рядків (їх називають "*records*") визначається послідовністю введення інформації в таблицю. При цьому оброблення стовпців і рядків може відбуватися в будь-якому порядку. Таблиці з даними пов'язані між собою спеціальними відносинами, завдяки чому з даними з різних таблиць можна працювати - наприклад, об'єднувати їх - за допомогою одного запиту.

Для керування базами даних застосовується особлива мова програмування - SQL. Скорочення розшифровується як "Structured query language", в перекладі на українську «мова структурованих запитів».

Команди, які використовуються в SQL, діляться на ті, які маніпулюють даними, ті, які визначають дані, і ті, які керують даними. Схема роботи з базою даних виглядає наступним чином:

**SQL - ЗАПИТ**

**СКБД**

**ДАНІ**

Для вирішення в не великийх проектах не слід використовувати громізку СКБД томущо вони будуть сповільнювати процес обробки даних. Тому було вибрано СКБД *Microsoft Access*. На даний момент це одна з найбільш компактних СКБД; також вона є вбудованої і реляційної. *Microsoft Access* дозволяє зберігати всі дані в одному файлі і, завдяки своєму невеликому обсягу, відрізняється завидною швидкодією. *Microsoft Access* значно відрізняється від і PostgreSQL своєю структурою: движок і інтерфейс цієї СУБД знаходяться в одній бібліотеці - і саме це дозволяє виконувати всі запити дуже швидко. Інші СУБД (MySQL, PostgreSQL, Oracle і т.д.) використовують парадигму клієнт-сервер, коли взаємодія відбувається через мережевий протокол*.*

З недоліків можна відзначити відсутність системи користувачів і можливості збільшення продуктивності.

*Microsoft Access* можна порадити до використання в проектах, де потрібно мати можливість швидко перенести додаток, і немає необхідності в масштабованості.

Атоматизацыя збірки пректу збірки проекту.

Збірка (англ. assembly) – двыйковий файл, що мыстить виконуючий код програми або другий, підготовлений для виконання інформаційного продукту.

Атоматизація збірки – це етап написання скриптів або автоматизація широкого спектру задач стосовно ПЗ, вживаному розробниками в повсякдневній діяльності, включаючи такі дії, як:

* компіляцыя вихідного коду,
* зборка бінарного коду,
* використання тестів,
* розгортання програми на різних платформах,
* написання документації, або опис змін в новій версії.

Атоматизація збірки є дуже важливим. Щоб отримати .class файли нам потрібно скомпілювати ці класи за допомогою консольної команди javac написавши щось подібне до Рисунку 2.?

D:\h_1488121935_6974891_e5c6fc6b01.png

Рисунок 2.? З зразок строки для компіляції.

Після, вийшов .class файл потрібно перенести в наш каталог, повторити цю процедуру для всіх класів, заархівувати і все, програма готова до деплойменту. А якщо десь допуступитись помилку, доведеться заново скомпілювати клас, перенести файл, і заново заархівувати і задеплоіть.

Щоб якось автоматизувати цю рутину були придумані системи збирання. З'являлися вони в такому порядку: спочатку Ant, потім Maven, потім Gradle (насправді їх більше, але в світі Java ці - найпопулярніші). Якщо говорити зовсім просто, основне завдання системи збирання - це компілювати і упаковувати. Архів, що вийшов в результаті складання називається артефактом. Цей артефакт являє собою або готове додаток, або бібліотеку, яку можна буде використовувати вже в іншому проекті.

Ще дуже важливої функції систем збірки - Управлінні залежність. В даний час жодне більш-менш серйозне додаток на Java не пишеться без використання сторонніх бібліотек. Java-бібліотека - це JAR архів, який містить в собі скомпільовані бібліотечні класи. JAR-файл скачується з інтернету і підключається до проекту. Тому був придумали менеджери залежностей (dependency manager). Суть їх роботи проста - в файлі, що описує конфігурацію збірки (pom.xml для Maven) ми записує які сторонні бібліотеки (залежно) нам потрібні і в процесі побудови система їх викачує, підставляє в classpath і компілює вже разом з ними. Викачуються бібліотеки з публічного сховища (для Java зараз основний репозиторій - <https://mvnrepository.com>). Зразок підключення залежності на Рисунку 2.3

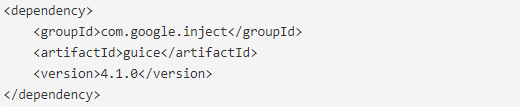


Рисунок 2.? Підключення бібліотеки *google.inject* версії 4.1

В нашому випадку буде використовувать *Apache Maven* - це інструмент управління і розуміння програмного забезпечення. Грунтуючись на концепції об'єктної моделі проекту (POM), Maven може керувати складанням проекту, складанням звітів та документацією з центральної частини інформації. А також середовище розробки *IntelliJ IDEA* містить в собі Графічний плагін що полегще подальшу розробку, який зображено на Рисунку 2.?.

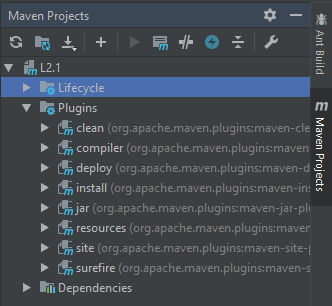


Рисунок 2.? Плагін для роботи з *Mawen* в *IntelliJ IDEA*

Вибір бібліотек поляг

# РОЗДІЛ 1

# АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Аналіз подій в сфері розробки та виробництва наземних роботизованих платформ та дистанційно керованих бойових машин за період 2015-2017 р.р. вказує на низку характерних тенденцій:

* + - головний акцент під час створення роботизованих платформ-носіїв озброєння, окрім оснащення максимально ефективними засобами спостереження та ураження, робиться на отриманні ними здатності змінювати призначення в залежності від завдань, що поставлені, та діяти сумісно з авіаційними безпілотними засобами та іншою військовою технікою. Для отримання дистанційно керованих бойових машин увага зосереджується на створенні стандартизованого універсального (відносно) комплекту обладнання та апаратури, здатного забезпечити швидке переобладнання звичайного військового транспортного засобу у дистанційно керований;
    - триває процес розширення спектру застосування у військах роботизованої техніки - від пошуку і знешкодження радіоактивних, біологічних, хімічних і вибухових речовин, ведення розвідки та матеріально-технічного забезпечення військ до нанесення вогневих ударів та вогневої підтримки власних сил;
    - відмічається поступовий перехід від одиночного застосування роботизованих платформ та виконання конкретного типу завдання до використання у складі групи для вирішення низки різноманітних завдань від ураження та бойового забезпечення.

Основними факторами, які підштовхують зростання ринку безпілотних наземних роботизованих платформ, залишаються сучасний рівень технологічних досягнень та прагнення зменшити вплив на людину середовища з високим рівнем загрози. Проблемними також залишаються й низка інших питань, які, поки що, характерні переважно для армії США, але є такими, що рано чи пізно стануть актуальними і для решти країн, що розвиватимуть подібну техніку, а саме:

* + - існують застереження щодо надання роботам можливості самостійно приймати рішення стосовно знищення об’єктів та живої сили;
    - є потреба в опрацюванні принципів, способів і методів групового застосування роботів для виконання бойових завдань, а також впровадження наземних роботизованих комплексів в організаційно-штатну структуру з’єднань та частин сухопутних військ;
    - є необхідність у проведенні ґрунтовних досліджень стосовно оцінки рівня продуктивності роботизованих комплексів, потреб в них у військах, ролі, місця та можливостей відповідної техніки в системі збройної боротьби майбутнього.
  1. Наземні роботизовані платформи США

Американська компанія General Dynamics спільно з лабораторією Корпусу морської піхоти в Кемп-Пендлтон (штат Каліфорнія, США) влітку 2016 року випробували гусеничну версію бойового робота MUTT (Multi-Utility Tactical Transport - Багатоцільовий тактичний транспорт).

MUTT (рис. 1.1) призначений для виконання завдань з надання вогневої підтримки морської піхоті під час патрулювання та здійснення контролю за місцевістю. Шасі MUTT може мати два виконання - колісне та гусеничне.

MUTT супроводжує бійців, полегшуючи навантаження на них за рахунок зменшення кількості обладнання, яке вони переносять, а також здійснює вогневу підтримку своєму підрозділу за рахунок модуля обладнаного кулеметом M134 Minigun калібру 7,62 мм. Робот здатний забезпечувати не лише вогневе враження та перевезення вантажу, а й бути виключно транспортним засобом, адже бойовий модуль легко знімається і змінюється на інший.



Рисунок 1.1 Multi-Utility Tactical Transport - Багатоцільовий тактичний транспорт

* 1. Бойова роботизована платформа «ЛАСКА»

Запорізькою компанією запропоновано концепцію створення наземної бойової роботизованої платформи (БРП) «ЛАСКА» (рис. 1.2) - як одного з елементів комплексу безпілотних наземних транспортних засобів військового призначення, що рекомендується до застосування в механізованих (мотопіхотних) частинах Сухопутних військ ЗС України.

Платформа розроблена в рамках власної інноваційної програми компанії «Інфоком Лтд» (Запоріжжя) та заснована на більш ніж 20-річному досвіді створення та впровадження складних систем автоматизації технологічних процесів в різних галузях промисловості, розробки і модернізації унікальних п'яти-координатних верстатів ЧПК, робототехнічних комплексів.



Рисунок1.2 Бойова роботизована платформа «ЛАСКА»

Технічні характеристики роботизованої платформи «ЛАСКА»:

* Колісна формула: 4х2
* Силовий агрегат: одноциліндровий 4-тактний двигун
* Потужність агрегату: 19 к.с. (14 кВт)
* Напруга бортової мережі: 12 В
* Матеріал платформи: сталь
* Розмір (ДхШхВ): 1895х1069х1350 мм
* База, мм 1 199
* Кліренс: 136 мм
* Запас ходу: 100 км
* Максимальна швидкість: 80 км/год
* Маса: 310 кг
  1. Застосування мікроконтролерних систем управління при реалізації роботизованих платформ

Мікроконтролер (*MCU*) – мікросхема, призначена для керування електронними пристроями. Типовий мікроконтролер поєднує в собі функції процесора і периферійних пристроїв, може містити ОЗУ і ПЗУ. Це однокристальний комп'ютер, здатний виконувати прості завдання. Використання однієї мікросхеми, замість цілого набору, як у випадку звичайних процесорів, що застосовуються в персональних комп'ютерах, значно знижує розміри, енергоспоживання і вартість пристроїв, побудованих на базі мікроконтролерів.

Мікроконтролери є основою для побудови вбудованих систем, їх можна зустріти в багатьох сучасних приладах, таких, як телефони, системи “Розумний дім” і т. п. Термін ”мікроконтролер” витіснив термін ”однокристальна мікро-ЕОМ”. Перший же патент на однокристальних мікро-ЕОМ був виданий в 1971 році інженерам М. Кочрену і Г. Буну, співробітникам *Texas Instruments*. Саме вони запропонували на одному кристалі розмістити не тільки мікропроцесор, але й пам'ять, пристрої введення-виведення. З появою однокристальних мікро-ЕОМ пов'язують початок ери комп'ютерної автоматизації в галузі управління. Мабуть, ця обставина і визначила термін “мікроконтролер” (*control* - управління). У 1979 році НДІ ТТ розробили однокристальну 16-розрядну ЕОМ К1801ВЕ1, архітектура якої називалася “Електроніка НЦ”. У 1980 році фірма *Intel* випускає мікроконтролер i8048. Трохи пізніше в цьому ж році *Intel* випускає наступний мікроконтролер: i8051. Вдалий набір периферійних пристроїв, можливість гнучкого вибору зовнішньої або внутрішньої програмної пам'яті і прийнятна ціна забезпечили цьому мікроконтролеру успіх на ринку. З погляду технології мікроконтролер i8051 був для свого часу дуже складним виробом - у кристалі було використано 128 тис. транзисторів, що в 4 рази перевищувало кількість транзисторів в 16-розрядному мікропроцесорі i8086.

На сьогоднішній день існує більше 200 модифікацій мікроконтролерів, що сумісних з I8051, випускаються двома десятками компаній, і великої кількості мікроконтролерів інших типів. Популярністю у розробників користуються 8-бітові мікроконтролери *PIC* фірми *Microchip Technology* і *AVR* фірми *Atmel*, шістандцяти бітних MSP430 фірми *TI*, а також *ARM*, архітектуру яких розробляє фірма *ARM* і продає ліцензії іншим фірмам для їх виробництва, процесорів – мікроконтролери.

Мікроконтролер характеризується великим числом параметрів, оскільки він одночасно є складним програмно-керованим пристроєм і електронним приладом (мікросхемою). Приставка "мікро" в назві мікроконтролера означає, що виконується він за мікроелектронної технології.В ході роботи микрконтроллер зчитує команди з пам'яті або порту вводу і виконує їх. Що означає кожна команда, визначається системою команд мікроконтролера. Система команд закладена в архітектурі мікрконтролера і виконання коду команди виражається в проведенні внутрішніми елементами мікросхеми певних мікрооперацій. При проектуванні мікроконтролерів доводиться дотримувати баланс між розмірами і вартістю з одного боку і гнучкістю і продуктивністю з іншої. Для різних застосувань оптимальне співвідношення цих і інших параметрів може розрізнятися дуже сильно. Тому існує величезна кількість типів мікроконтролерів, що відрізняються архітектурою процесорного модуля, розміром і типом вбудованої пам'яті, набором периферійних пристроїв, типом корпусу і т. д.

Мікроконтролери застосовують в побутовій техніці, медичних приладах, системах управління ліфтами, телефонах, раціях та інших засобах зв'язку, електронних музичних інструментах і автомагнітолах, комп'ютерної периферії (клавіатурах, джойстиках, принтерах і т. П.), Світлофорах, автоматичних воротах і шлагбауми, інтерактивних дитячих іграшках, автомобілях, локомотивах і літаках, роботах і промислових верстатах.

* 1. Платформа Arduino

Arduino (Ардуіно) — апаратна обчислювальна платформа основними компонентами якої є плата мікроконтролера з елементами вводу/виводу та середовище розробки Processing/Wiring на мові програмування, що є 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.

Нижче представлені основні версії плат *Arduino*:

* *Due* - нова плата на базі *ARM* процесора *32bit Cortex-M3 ARM SAM3U4E.*
* *Leonardo* - остання версія платформи Arduno на *ATmega32u4* микроконтролері. Відрізняється роз'ємом *microUSB*, за розмірами збігається з *UNO*.
* *Uno* - сама популяраня версія базової платформи Arduino *USB*.
* *Arduino Ethernet* - контролер з вбудованою підтримкою роботи по мережі із можливістю живлення по мережі за допомогою модуля *POE (Power over Ethernet).*
* *Duemilanove* - є передостанньою версією базової платформи *Arduino USB*. Підключення *Duemilanove* здійснюється стандартним кабелем *USB*.
* *Diecimila* - попередня версія базової платформи *Arduino USB*.
* *Nano* - це компактна платформа, яка використовується як макет. *Nano* підключається до комп'ютера за допомогою кабелю *USB Mini-B*.
* *Mega ADK* - версія плати *Mega* 2560 з підтримкою *USB host* інтерфейсу для зв'язку з телефонами на *Android* і іншими пристроями з *USB* інтерфейсом.
* *Mega2560* - нова версія плати серії *Mega*. Побудована на базі *Atmega2560* і з використанням чіпа *ATMega8U2* для послідовного з'єднання по *USB* порту.
* *Mega* - попередня версія серії *Mega* на базі *Atmega1280*.
* *Arduino BT* платформа з модулем *Bluetooth* для бездротового зв'язку і програмування. Сумісна з платами розширення *Arduino*.
* *Mini* - найменша платформа *Arduino*. Прекрасно працює як макетна модель, або, в проектах, де простір є критичним параметром. Платформа підключається до комп'ютера за допомогою адаптера *Mini USB*.
* Адаптер *Mini USB* - плата, яка конвертує підключення *USB* в лінії 5 *В, GND, TX* і *RX* для з'єднання з платформою *Arduino Mini* або іншими микроконтроллерами.
* *Pro* - платформа, розроблена для досвідчених користувачів, може бути частиною більшого проекту. *Pro* дешевша, ніж *Diecimila* і може живитися від акумуляторної батареї, але в той же час вимагає додаткової збірки і компонентів.
* *Pro Mini* - як і платформа *Pro* розроблена для досвідчених користувачів, яким потрібна низька ціна, менші розміри і додаткова функціональність.
* *Serial* - базова платформа з інтерфейсом *RS232* для зв'язку і програмування. Плата легко збирається навіть початківцями.
* *Serial Single Sided* - платформа розроблена для ручної збірки. Має трохи більший розмір, ніж *Diecimila*, але сумісна з платами розширення *Arduino*.
* *USB Serial Light* Адаптер - адаптер, що дозволяє підключати плати *Arduino* до комп'ютера для обміну даними і заливки скетчів. Зручний для програмування таких плат, як *Arduino Mini, Arduino Ethernet* і інших, які не мають свого роз'єму *USB*.

Висновок

Сучасний рівень технологічних досягнень та прагнення зменшити вплив на людину середовища з високим рівнем загрози є основними факторами, які підштовхують зростання ринку безпілотних наземних роботизованих платформ, що є особливо актуальним у військовій сфері, беручи до уваги геополітичну ситуацію в країні. Питання розширення спектру застосування у військах роботизованої техніки стоїть досить гостро, а наявність малої кількості уніфікованих програмно-апаратних рішень зумовлює необхідність вдосконалення існуючих та розробки нових зразків дистанційно керованих засобів озброєння, оснащення їх максимально ефективними засобами спостереження та ураження, а також необхідність в отриманні ними здатності змінювати призначення в залежності від завдань, що поставлені, та діяти сумісно з авіаційними безпілотними засобами та іншою військовою технікою.

РОЗДІЛ 2

ВИБІР ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ДЛЯ РОЗРОБКИ ПРОГРАМНОГО МОДУЛЮ

З кожним роком наземні роботизовані платформи та дистанційно керовані бойові машини успішно поширюють власну присутність на світовому ринку торгівлі озброєнням та безпосередньо в лавах збройних сил багатьох країн світу. Перевагою таких платформ є зниження фізичного навантаження на особовий склад, який пересувається місцевістю, шляхом застосування даних платформ для транспортування зброї, боєприпасів, обладнання, продуктів харчування, води, предметів медичного призначення тощо. Забезпечення збереження життя військовослужбовців шляхом максимального збільшення відстані між загрозою та особовим складом (зокрема, під час виконання завдань з розмінування чи проведення інших небезпечних дій). Достатньо активно в цьому напрямку діють Росія, Китай, США, Великобританія, Тайвань, Естонія та Іран. Великий стрибок за останні кілька років здійснила РФ, яка створила низку бойових роботів. Дані платформи є досить дорогими в створенні, тому пропонуємо реалізувати наземну безпілотну платформу на базі мікроконтролера *Arduino*. Ця платформа має досить низьку вартість в порівнянні з іншими платформами. Програмування мікроконтролера *Arduino* можливо в двох варіантах: програмування за допомогою слів-команд середовища *Arduino IDE* і графічне програмування в середовищі програми *FlProg*. Розглянемо кожну з них детальніше.

2.1 Апаратна частина *Arduino*

*Arduino* – апаратна обчислювальна платформа, основними компонентами якої є плата вводу/виводу та середовище розробки на мові *Processing/Wiring*.*Arduino* може використовуватися як для створення автономних інтерактивних об'єктів, так і підключатися до програмного забезпечення, яке виконується на комп'ютері. На основі *Arduino* розробляють навчальні приклади, роботів, системи спостереження і безпеки, аналоги систем типу ”розумний дім”, дистанційний пульт для керування побутовими пристроями, і багато інших проектів. *Arduino* складається з мікроконтролера *Atmel*, а також елементів для програмування та інтеграції з іншими пристроями. На багатьох платах наявний лінійний стабілізатор напруги +5В або +3,3В. Тактування здійснюється на частоті 84, 16 або 8 МГц кварцовим резонатором. У мікроконтролер записаний завантажувач (*bootloader*), тому зовнішній програматор не потрібен. У платі *Arduino UNO* доступно 14 цифрових входів/виходів, 6 із яких можуть видавати *PWM* сигнал, і 6 аналогових входів. Ці сигнали доступні на платі через контактні площадки або роз’єми. Також існує багато видів зовнішніх плат розширення, які називаються "*shields*", які приєднуються до плати *Arduino* через штирьові роз’єми.Середовище розробки засноване на мові програмування *Processing* та спроектоване для програмування новачками, не знайомими близько з розробкою програмного забезпечення. Програма обробляються за допомогою препроцесора, а потім компілюється за допомогою *AVR-GCC*.

2.2 Види *Arduino*

Докладніше розглянемо плати які є дуже популярними на сьогодні:

* *Arduino NANO;*
* *Arduino UNO;*
* *Arduino MEGA*.
  + 1. Платформа *Arduino NANO*

*Arduino Nano* (рис. 2.1), побудована на мікроконтролері ATmega328 (*Arduino Nano 3.0*) або *ATmega168* (*Arduino Nano 2.x*), має невеликі розміри і може використовуватися в лабораторних роботах.

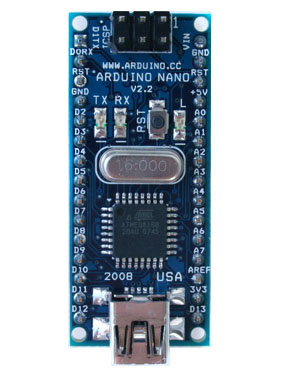
****

Рисунок 2.1 Зовнішній вигляд *Arduino Nano*

Короткі характеристики мікроконтролера *Arduino* Nano:

* Мікроконтролер: *Atmel ATmega168* або *ATmega328*;
* Робоча напруга: 5 В;
* Вхідна напруга (рекомендована): 7-12 В;
* Вхідна напруга (граничне значення): 6-20 В;
* Цифрові Входи / Виходи: 14 (6 з яких можуть використовуватися як виходи ШІМ);
* Аналогові входи: 8;
* Постійний струм через вхід / вихід : 40 мА;
* Флеш пам`ять: 16 Кб (*ATmega168*) або 32 Кб (*ATmega328*) при цьому 2 КБ використовуються для завантажувача;
* ОЗУ: 1 Кб (*ATmega168*) або 2 КБ (*ATmega328*);
* *EEPROM* 512 байт (*ATmega168*) або 1 Кб (*ATmega328*);
* Тактова частота: 16 МГц;
* Розміри: 1.85 см x 4.2 см.

Живлення *Arduino Nano* може отримувати через підключення *Mini-B USB*, або від нерегульованого 6-20 В (вивід 30), або регульованого 5 В (вивід 27), зовнішнього джерела живлення. Автоматично вибирається джерело з найвищою напругою. Мікросхема *FTDI FT232RL* отримує живлення, тільки якщо сама платформа живиться від *USB*. Таким чином при роботі від зовнішнього джерела (НЕ *USB*), буде відсутня напруга 3.3 В, що генерується мікросхемою *FTDI*, при цьому світлодіоди *RX* і *TX* блимають тільки при наявності сигналу високого рівня на виходах 0 і 1.Мікроконтролер *ATmega168* має 16 кБ флеш-пам'яті для зберігання коду програми, а мікроконтролер *ATmega328*, в свою чергу, має 32 кБ (в обох випадках 2 кБ використовується для зберігання завантажувача). *ATmega168* має 1 кБ ОЗУ і 512 байт *EEPROM* (яка читається і записується за допомогою бібліотеки EEPROM), а *ATmega328* - 2 кБ ОЗУ і 1 Кб *EEPROM*.

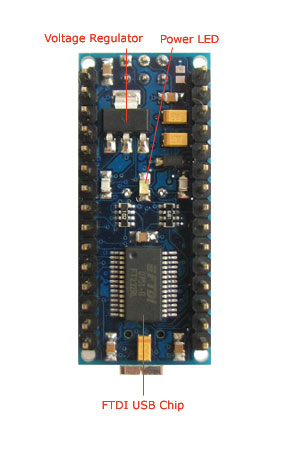
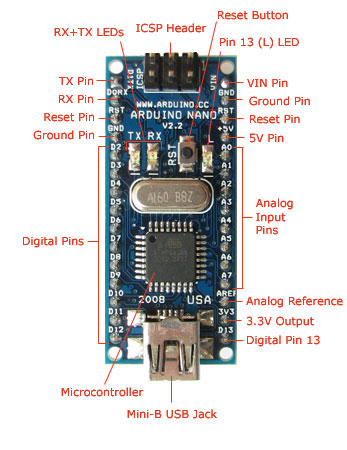


Рисунок 2.2 Входи і виходи *Arduino Nano*

Кожен з 14 цифрових виводів(рис. 2.2) *Nano*, використовуючи функції *pinMode (), digitalWrite (), і digitalRead (),* може налаштовуватися як вхід або вихід. Виходи працюють при напрузі 5 В. Кожен вихід має резистор (стандартно відключений) 20-50 кОм і може пропускати до 40 мА. Деякі виходи мають особливі функції:

* Послідовна шина: 0 (*RX*) і 1 (*TX*). Виходи використовуються для отримання (*RX*) і передачі (*TX*) даних. Дані виходи підключені до відповідних виходів мікросхеми послідовної шини *FTDI USB-to-TTL*;
* Зовнішнє переривання: 2 і 3. Дані виходи можуть бути налаштовані на виклик переривання або на молодшому значенні, або на передньому чи задньому фронті, або при зміні значення;
* ШІМ: 3, 5, 6, 9, 10, і 11. Будь-який з виходів забезпечує ШІМ з роздільною здатністю 8 біт за допомогою функції *analogWrite ()*.
* *LED*: 13. Вбудований світлодіод, підключений до цифрового виходу 13. Якщо значення на виведення має високий потенціал, то світлодіод горить.

На платформі *Nano* встановлені 8 аналогових входів, кожен з дозволом 10 біт (тобто може приймати 1024 різних значення). Стандартно виходи мають діапазон вимірювання до 5В щодо землі, проте є можливість змінити верхню межу за допомогою функції *analogReference ().* *Arduino Nano* має кілька пристроїв для здійснення зв'язку з комп'ютером, іншими пристроями *Arduino* або микроконтроллерами. *ATmega168 і ATmega328* підтримують послідовний інтерфейс *UART TTL* (5 В), здійснюваний виходами 0 (*RX*) і 1 (*TX*). Встановлена ​​на платі мікросхема *FTDI FT232RL* направляє даний інтерфейс через *USB*, а драйвери *FTDI* (включені в програму *Arduino*) надають віртуальний *COM* порт програмі на комп'ютері. Моніторинг послідовної шини (*Serial Monitor*) програми *Arduino* дозволяє посилати і отримувати текстові дані при підключенні до платформи. Світлодіоди *RX* і *TX* на платформі будуть мигати при передачі даних через мікросхему *FTDI* або *USB* підключення (але не при використанні послідовної передачі через виходи 0 і 1).Бібліотекою *SoftwareSerial* можливо створити послідовну передачу даних через будь-який з цифрових виходів *Nano*.

*ATmega168 і ATmega328* підтримують інтерфейси *I2C (TWI) і SPI*. В *Arduino* включена бібліотека *Wire* для зручності використання шини *I2C*. Більш детальна інформація знаходиться в документації. Для використання інтерфейсу *SPI* зверніться до технічних даних мікроконтролерів *ATmega168 і ATmega328*.

2.2.2Платформа *Arduino**Uno*

*Arduino Uno* (рис. 2.3) контролер побудований на *ATmega328*. Платформа має 14 цифрових входів / виходів (6 з яких можуть використовуватися як виходи ШІМ), 6 аналогових входів, кварцовий генератор 16 МГц, роз'єм *USB*, силовий роз'єм, роз'єм *ICSP* і кнопку перезавантаження. Для роботи необхідно підключити платформу до комп'ютера за допомогою кабелю *USB*, або подати живлення за допомогою адаптера *AC / DC* або батареї.

На відміну від всіх попередніх плат, які використовували *FTDI USB* мікроконтролер для зв'язку з *USB*, новий *Arduino Uno* використовує мікроконтролер *ATmega8U2*.



Рисунок 2.3 Зовнішній вигляд *Arduino UNO*

Характеристики мікроконтролера *Arduino UNO(*рис. 2.3*)*:

* Мікроконтролер :*ATmega328*;
* Робоча напруга : 5 У;
* Вхідна напруга (рекомендована): 7-12 В;
* Вхідна напруга (гранична): 6-20 В;
* Цифрові Входи / Виходи: 14 (6 з яких можуть використовуватися як виходи ШІМ);
* Аналогові входи: 6;
* Постійний струм через вхід / вихід: 40 мА;
* Постійний струм для виведення 3.3 В: 50 мА;
* Флеш пам`ять: 32 Кб (ATmega328) з яких 0.5 КБ використовуються для завантажувача;
* ОЗУ: 2 КБ (*ATmega328*);
* *EEPROM*: 1 Кб (*ATmega328*);
* Тактова частота: 16 МГц.

Живлення *Arduino Uno* може отримувати через підключення *USB* або від зовнішнього джерела живлення. Джерело живлення вибирається автоматично. Зовнішнє живлення (НЕ *USB*) може подаватися через перетворювач напруги (блок живлення) або акумуляторною батареєю. Перетворювач напруги підключається за допомогою роз'єму 2.1 мм з центральним позитивним полюсом. Провід від батареї підключаються до виходів *Gnd і Vin* роз'єму живлення. Платформа може працювати при зовнішньому живленні від 6 В до 20 В. При напрузі живлення нижче 7 В, вихід 5V може видавати менше 5 В, при цьому платформа може працювати нестабільно. При використанні напруги вище 12 В регулятор напруги може перегрітися і пошкодити плату. Рекомендований діапазон від 7 В до 12 В.

Виходи живлення Arduino UNO:

* *VIN*. Вхід використовується для подачі живлення від зовнішнього джерела (за відсутності 5 В від роз'єму *USB* або іншого регульованого джерела живлення). Подача напруги живлення відбувається через даний вихід;
* 5V. Регульоване джерело напруги, що використовується для живлення мікроконтролера і компонентів на платі. Живлення може подаватися від виходу *VIN* через регулятор напруги, або від роз'єму *USB*, або іншого регульованого джерела напруги 5 В;
* 3V3. Напруга на виводі 3.3 В генерується вбудованим регулятором на платі. Максимальне споживання струму 50 Ма;
* *GND*. Вихід заземлення.

Мікроконтролер ATmega328 має 32 кБ флеш пам'яті, з яких 0.5 кБ використовується для зберігання завантажувача, а також 2 кБ ОЗУ (SRAM) і 1 Кб EEPROM. (Яка читається і записується за допомогою бібліотеки EEPROM).

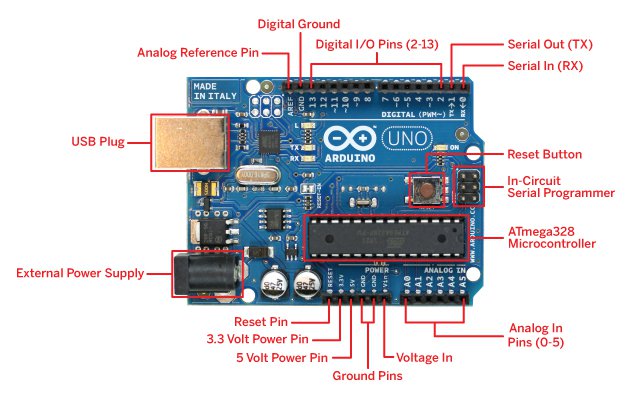


Рисунок 2.4 Входи і виходи *Arduino Uno*

Характеристики входів та виходів Arduino Uno:

* Кожен з 14 цифрових виходів(рис. 2.4) *Uno* може налаштований як вхід або вихід, використовуючи функції *pinMode (), digitalWrite (), і digitalRead ()*. Виходи працюють при напрузі 5 В. Кожен вихід має резистор (за замовчуванням відключений) 20-50 кОм і може пропускати до 40 мА. Деякі виходи мають особливі функції:
* Послідовна шина: 0 (*RX*) і 1 (*TX*). Виходи використовуються для отримання (*RX*) і передачі (*TX*) даних *TTL*. Дані виходи підключені до відповідних виходів мікросхеми послідовної *шини ATmega8U2 USB-to-TTL*.
* Зовнішнє переривання: 2 і 3. Дані виходи можуть бути налаштовані на виклик переривання або на молодшому значенні, або на передньому чи задньому фронті, або при зміні значення.
* ШІМ: 3, 5, 6, 9, 10, і 11. Будь-який з виходів забезпечує ШІМ з роздільною здатністю 8 біт за допомогою функції *analogWrite ().*
* *SPI*: 10 (*SS*), 11 (*MOSI*), 12 (*MISO*), 13 (*SCK*). За допомогою даних виходів здійснюється зв'язок *SPI*, для чого використовується бібліотека *SPI*.
* *LED*: 13. Вбудований світлодіод, підключений до цифрового виходу 13. Якщо значення на виведення має високий потенціал, то світлодіод горить.
* На платформі *Uno* встановлені 6 аналогових входів (позначених як A0 .. A5), кожен дозволом 10 біт (тобто може приймати 1 024 різних значення). Стандартно виходи мають діапазон вимірювання до 5 В щодо землі, проте є можливість змінити верхню межу за допомогою виведення *AREF* і функції *analogReference ().*

На платформі *Arduino Uno* встановлено кілька пристроїв для здійснення зв'язку з комп'ютером, іншими пристроями *Arduino* або мікроконтролерами. *ATmega328* підтримують послідовний інтерфейс *UART TTL* (5 В), здійснюваний виходами 0 (*RX*) і 1 (*TX*). Встановлена ​​на платі мікросхема *ATmega8U2* направляє даний інтерфейс через *USB*, програми на стороні комп'ютера "спілкуються" з платою через віртуальний *COM* порт.

Прошивка *ATmega8U2* використовує стандартні драйвера *USB COM*, ніяких стороніх драйверів не потрібно, але на *Windows* для підключення потрібно файл *ArduinoUNO.inf*. Моніторинг послідовної шини (*Serial Monitor*) програми *Arduino* дозволяє посилати і отримувати текстові дані при підключенні до платформи. Світлодіоди *RX* і *TX* на платформі будуть мигати при передачі даних через мікросхему *FTDI* або *USB* підключення (але не при використанні послідовної передачі через виходи 0 і 1. Бібліотекою *SoftwareSerial* можливо створити послідовну передачу даних через будь-який з цифрових виходів *Uno*.*ATmega328* підтримує інтерфейси *I2C* (*TWI*) і *SPI*. В *Arduino* включена бібліотека *Wire* для зручності використання шини *I2C*.

2.2.3 Платформа *Arduino Mega*

*Arduino Mega* (рис. 2.5) побудована на мікроконтролері *ATmega2560*. Плата має 54 цифрових входів / виходів (14 з яких можуть використовуватися як виходи ШІМ), 16 аналогових входів, 4 послідовних порти *UART*, кварцовий генератор 16 МГц, *USB*, роз'єм живлення, роз'єм *ICSP* і кнопка перезавантаження. Для роботи необхідно підключити платформу до комп'ютера за допомогою кабелю *USB* або подати живлення за допомогою адаптера *AC / DC*, або акумуляторною батареєю. *Arduino Mega* 2560 сумісна з усіма платами розширення, розробленими для платформ *Uno* або *Duemilanove*.

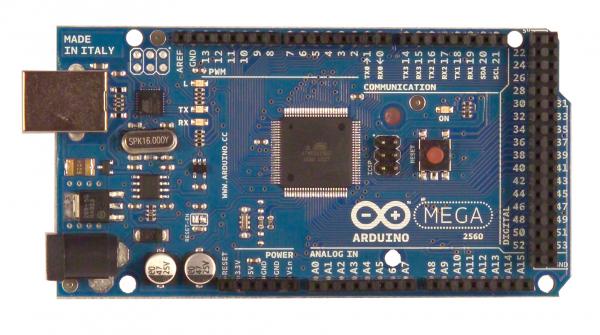
****

Рисунок 2.5 Зовнішній вигляд *Arduino Mega*

Короткі характеристики мікроконтролера *Arduino Mega*:

* Мікроконтролер *ATmega2560*;
* Робоча напруга 5В;
* Вхідна напруга (рекомендована) 7-12В;
* Вхідна напруга (гранична) 6-20В;
* Цифрові Входи / Виходи 54 (14 з яких можуть працювати також як виходи ШІМ);
* Аналогові входи 16;
* Постійний струм через вхід / вихід 40 mA;
* Постійний струм для виведення 3.3 В 50 mA;
* Флеш-пам'ять 256 KB (з яких 8 КB використовуються для завантажувача);
* ОЗУ 8 KB;
* Незалежна пам'ять 4 KB;
* Тактова частота 16 *MHz*.

*Arduino Mega* може отримувати живлення як через підключення по *USB*, так і від зовнішнього джерела живлення. Джерело живлення вибирається автоматично. Зовнішнє живлення (НЕ *USB*) може подаватися через перетворювач напруги *AC / DC* (блок живлення) або акумуляторною батареєю. Перетворювач напруги підключається за допомогою роз'єму 2.1 мм з позитивним полюсом на центральному контакті. Провід від батареї підключаються до виходів Gnd і Vin роз'єму живлення (*POWER*).

Платформа може працювати при зовнішньому живленні від 6 В до 20 В. При напрузі живлення нижче 7 В, вихід 5V може видавати менше 5 В, при цьому платформа може працювати нестабільно. При використанні напруги вище 12 В регулятор напруги може перегрітися і пошкодити плату. Рекомендований діапазон від 7 В до 12 В. Плата *Mega2560*, на відміну від попередніх версій плат, не використовує *FTDI USB* мікроконтролер. Для обміну даними по *USB* використовується мікроконтролер *Atmega8U2*, запрограмований як конвертер *USB-to-serial*.

Виходи живлення Arduino Mega:

* *VIN*. Вхід використовується для подачі живлення від зовнішнього джерела (за відсутності 5 В від роз'єму *USB* або іншого регульованого джерела живлення). Подача напруги живлення відбувається через даний вихід. Якщо живлення подається на роз'єм 2.1*mm*, то від цього входу можна живитися.
* 5V. Регульоване джерело напруги, що використовується для живлення мікроконтролера і компонентів на платі. Живлення може подаватися від виведення *VIN* через регулятор напруги, або від роз'єму *USB*, або іншого регульованого джерела напруги 5 В.
* Напруга на виході 3.3 В генерується мікросхемою *FTDI* на платформі. Максимальне споживання струму 50 мА.
* *GND*. Виходи заземлення.

Мікроконтролер *ATmega2560* має: 256 кБ флеш-пам'яті для зберігання коду програми (4 кБ використовується для зберігання завантажувача), 8 кБ ОЗУ і 4 КБ *EEPROM* (яка читається і записується за допомогою бібліотеки *EEPROM*).Кожен з 54 цифрових виходів *Mega*, використовуючи функції *pinMode ()*, *digitalWrite ()*, і *digitalRead ()*, може налаштовуватися як вхід або вихід. Виходи працюють при напрузі 5 В. Кожен вихід має навантажувальний резистор (стандартно відключений) 20-50 кОм і може пропускати до 40 мА.

Деякі виходи мають особливі функції:

* Послідовна шина: 0 (*RX*) і 1 (*TX*); Послідовна шина 1: 19 (*RX*) і 18 (*TX*); Послідовна шина 2: 17 (*RX*) і 16 (*TX*); Послідовна шина 3: 15 (*RX*) і 14 (*TX*). Виходи використовуються для отримання (*RX*) і передачі (*TX*) даних *TTL*. Виходи 0 і 1 підключені до відповідних виходів мікросхеми послідовної шини *ATmega8U2*.
* Зовнішнє переривання: 2 (переривання 0), 3 (переривання 1), 18 (переривання 5), 19 (переривання 4), 20 (переривання 3), і 21 (переривання 2). Дані виходи можуть бути налаштовані на виклик переривання або на молодшому значенні, або на передньому чи задньому фронті, або при зміні значення.
* *PWM*: 2 до 13 і 44-46. Будь-який з виходів забезпечує ШІМ з роздільною здатністю 8 біт за допомогою функції *analogWrite ().*
* *SPI*: 50 (*MISO*), 51 (*MOSI*), 52 (*SCK*), 53 (*SS*). За допомогою даних виходів здійснюється зв'язок *SPI*, наприклад, використовуючи бібліотеку *SPI*. Також виходи *SPI* можуть бути виведені на блоці *ICSP*,
* *LED*: 13. Вбудований світлодіод, підключений до цифрового висновку 13. Якщо значення на виведення має високий потенціал, то світлодіод горить.

Платформа *Mega2560* є 16 аналогових входів, КОЖЕН дозволить 10 біт (тобто может приймати +1024 різніх значень). Стандартно виходи мають діапазон вимірювання до 5 В овідносно землі, є можлівість змнити верхню межу з опомогою виведення *AREF* і функції *analogReference ()*.

На платформі встановлено кілька пристроїв для здійснення зв'язку з комп'ютером, іншими пристроями *Arduino* або мікроконтролерами. *ATmega2560* підтримує 4 порти послідовної передачі даних *UART* для *TTL*. Встановлена ​​на платі мікросхема *ATmega8U2* направляє один з інтерфейсів через *USB*, надаючи віртуальний *COM* порт програмами на комп'ютері (машинам під управлінням *Windows* для коректної роботи з віртуальним *COM* портом необхідний *.inf* файл, системи на базі *OSX* і *Linux*, автоматично розпізнає *COM* порт). Утиліта моніторингу послідовної шини (Serial Monitor) середовища розробки *Arduino* дозволяє посилати і отримувати текстові дані при підключенні до платформи. Світлодіоди RX і TX на платформі будуть мигати при передачі даних через мікросхему *ATmega8U2* і *USB* підключення (але не при використанні послідовної передачі через виходи 0 і 1).

2.3 Середовище розробки програмного забезпечення *Arduino*

Програмування мікроконтролера *Arduino* можливо в двох варіантах: програмування за допомогою слів-команд середовища *Arduino IDE* і графічне програмування в середовищі програми *FlProg*. Розглянемо кожну з них детальніше.

2.3.1 *Arduino IDE*

Середовище розробки *Arduino* (рис. 2.6) представляє собою текстовий редактор програмного коду, область повідомлень, вікно виведення тексту (консоль), панель інструментів і кілька меню. Для завантаження програм і зв'язку середовище розробки підключається до апаратної частини *Arduino*.

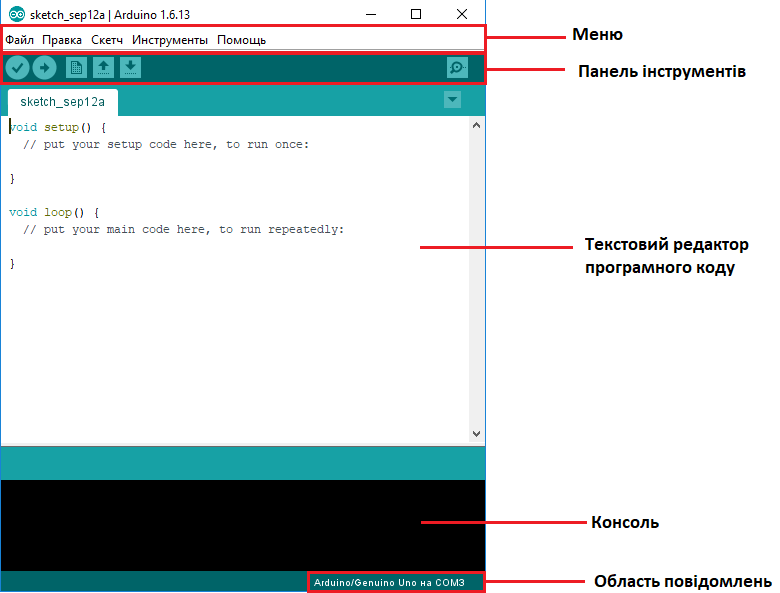


Рисунок 2.6 Загальний вигляд *Arduino IDE*

В меню "Файл" (рис. 2.7) опустивши пункти меню, хотілося б згадати такий пункт, як "Папка зі скетчами". За замовчуванням *Arduino IDE* зберігає кожен скетч в окрему папку. Ім'я папки збігається з ім'ям, зазначеним для скетчу при збереженні. Змінити робочу директорію для папок з скетчами можна в пункті меню "Налаштування".

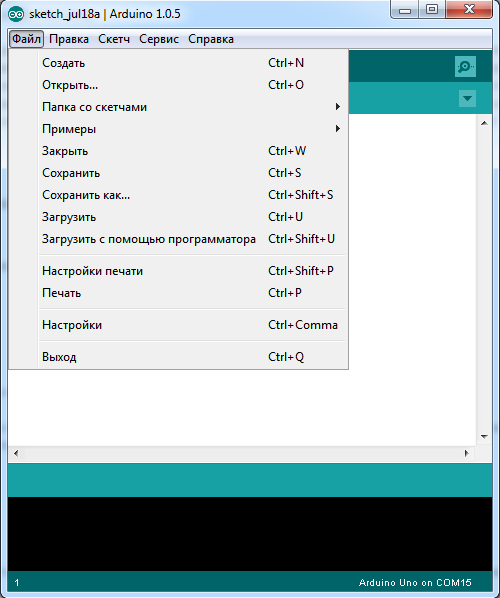


Рисунок 2.7 Пункт меню "Файл"

В меню "Правка"(рис. 2.8) розташовані команди для роботи з кодом вашої програми. Часто використовувані команди зручні наявністю комбінацій для швидкого доступу за допомогою клавіатури. Зручними функціями є можливість копіювання для форумів і в *html* форматі, що дозволяють ділиться вашими скетчами, зберігаючи наочність розмітки у вигляді кодів або *html* розмітки відповідно.

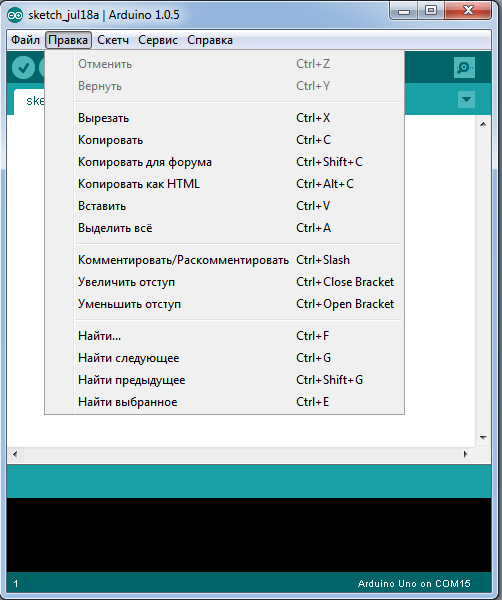


Рисунок 2.8 Пункт меню “Правка”

Меню "Скетч"(рис. 2.9). В даному меню продубльовано команда з панелі управління "Перевірити / Компілювати". Виконання якої призведе до перевірки вашого коду на помилки, і в разі їх відсутності - до компіляції.Пункт меню "Показати папку скетчів" відкриє робочу директорію *Arduino IDE*, зазначену в налаштуваннях. "Додати файл ..." дозволяє відкрити текстовий файл (або скетч) в окремій вкладці.

Бібліотеки додають додаткову функціональність скетчам, наприклад, при роботі з апаратною частиною або при обробці даних. Одна або кілька директив *#include* будуть розміщені на початку коду скетчу з подальшою компіляцією бібліотек і разом зі скетчем. Завантаження бібліотек вимагає додаткового місця в пам'яті *Arduino*.

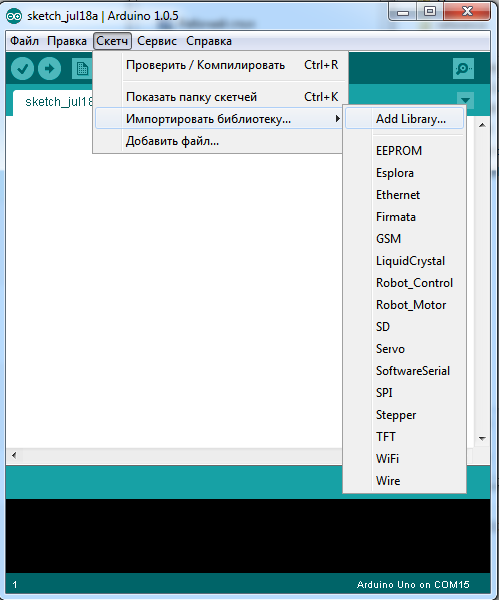


Рисунок 2.9 Пункт меню "Скетч"

Для встановлення сторонніх бібліотек можна скористатися командою "Імпортувати бібліотеку ..." / "*Add Library* ..."(рис. 2.10)

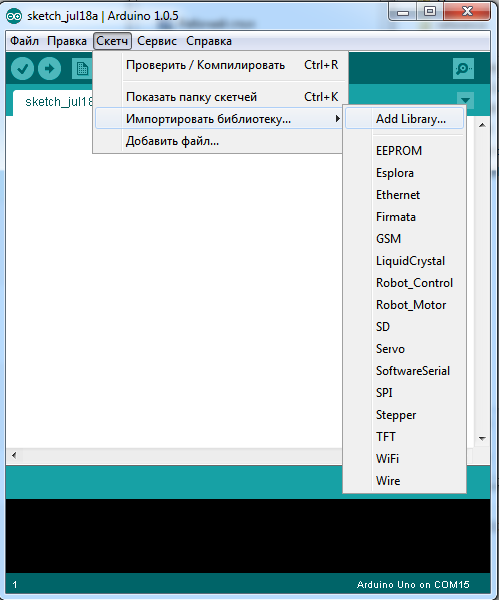


Рисунок 2.11 Підпункт “Імпортувати бібліотеку”

У меню "Сервіс" (рис. 2.12) необхідно вказати модель вашої *Arduino* плати, а так само *COM* порт, до якого вона підключена. Зручною функцією є автоформатування, яка дозволяє виправити помилки в розмітці скетчу і привести його в легкий для читання вигляд. Середовище *Arduino IDE* дозволяє залити *bootloader* на *atmega* мікроконтролери. У пункті меню "Програматор" вказується використовуваний пристрій. Команда "Записати завантажувач" здійснює безпосередньо заливку бутлоадера для зазначеної моделі плати.

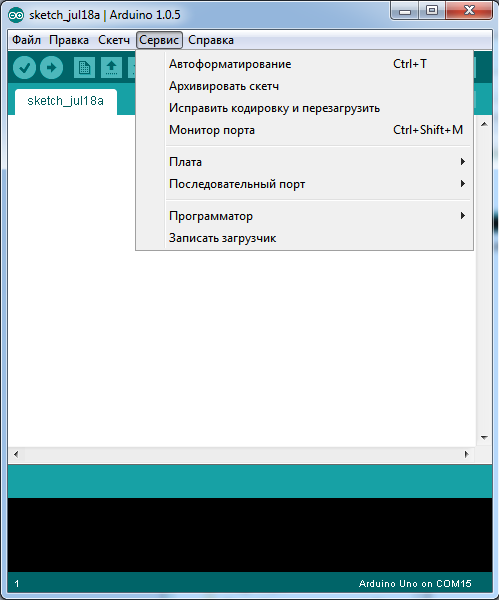


Рисунок 2.12 Пункт меню "Сервіс"

Пункт меню "Монітор порту" (рис. 2.13) викликає вікно для обміну повідомленнями з *arduino* через *COM* порт:

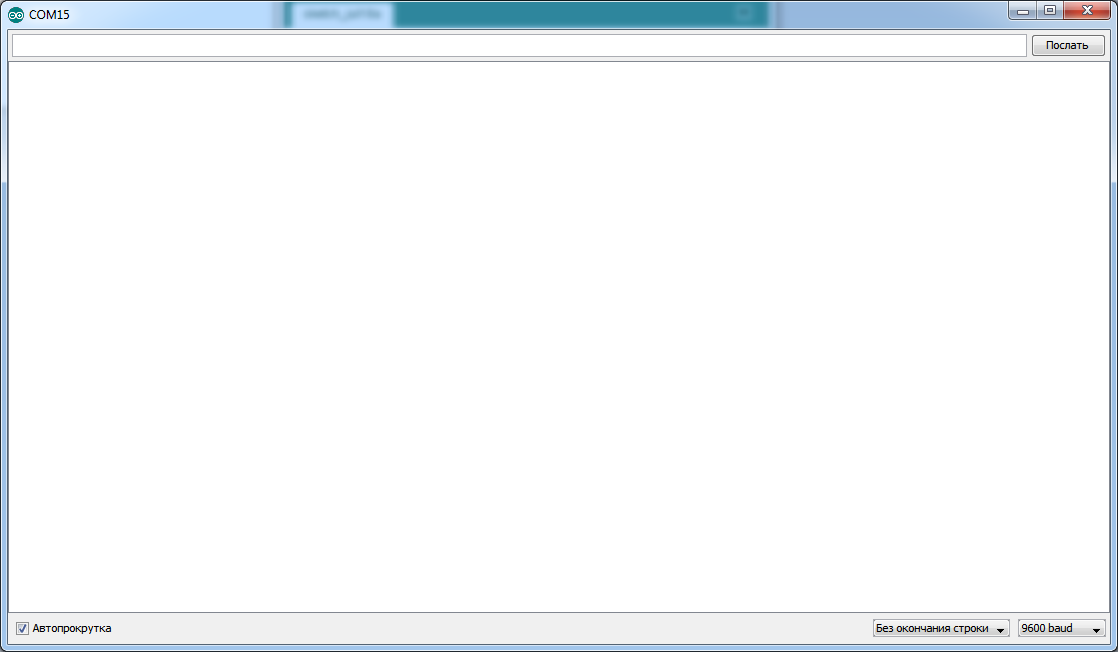


Рисунок 2.13 *COM* порт

Меню "Довідка" (рис 2.14) підпункти меню ведуть на відповідні статті на офіційний сайт (англійська мова).

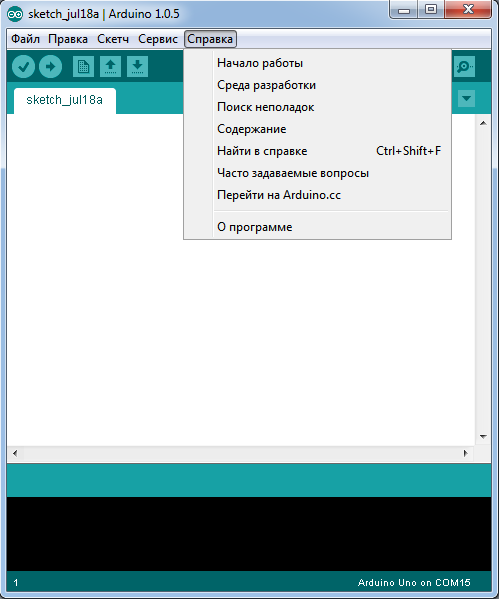


Рисунок 2.14 Пункт меню "Довідка"

Команди панелі управління (рис. 2.15), як ми бачимо, дублюють найбільш актуальні пункти меню.

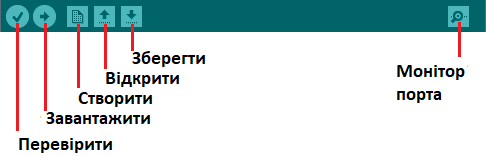


Рисунок 2.15 Команди панелі управління

2.3.2 Графічне середовище програмування *FlProg*

З самого моменту появи контролерів розвиток принципів роботи з ними йде по шляху зростання абстракції. Перший етап являв програмування безпосередньо в машинних кодах. Програмування було складним, довгим і вимагало дуже специфічного складу розуму. Тому програмістів було дуже мало. Але людина істота лінива, а лінь, як відомо двигун прогресу. Тому придумали перший рівень абстракції — асемблер. Писати програми стало простіше і веселіше. Кількість програмістів зросла. Але асемблер не дуже сильно відрізнявся від машинних кодів. Тому з'явився наступний рівень абстракції. Мови високого рівня. Основною метою цих мов була можливість пояснити машині, що від неї хочуть мовою максимально наближеною до людської. Це дозволяло займатися програмуванням людям з менш специфічним складом. Тому з розвитком мов високого рівня кількість програмістів зростала, і відповідно зростала кількість корисних програмних продуктів, які вони створювали.

Проект *FLProg* пропонує новий рівень абстракції з досить сміливою заявою «Щоб програмувати як мікроконтролери, так і комп'ютери не обов'язково знати мови програмування». Заява може здатися надто сміливою, але це можливо і вже доведено в суміжній з комп'ютерами області. Це область програмування промислових систем автоматизованого управління. Практично з самого початку виробники промислових контролерів пішли по цьому шляху. Зараз стандартом для середовищ програмування в основних виробників є мови *FBD і LAD*.

*FBD* (*Function Block Diagram*) – графічна мова програмування. Програма утворюється зі списку ланцюгів, які виконуються послідовно зверху вниз. При програмуванні використовуються набори бібліотечних блоків. Блок (елемент) - це підпрограма, функція або функціональний блок (І, АБО, НІ, тригери, таймери, лічильники, блоки обробки аналогового сигналу, математичні операції і ін.). Кожен окремий ланцюг являє собою вираз, складений графічно з окремих елементів. До виходу блоку підключається наступний блок, утворюючи ланцюг. Всередині кола блоки виконуються строго в порядку їх сполуки. Результат обчислення ланцюга записується у внутрішню змінну або подається на вихід контролера.

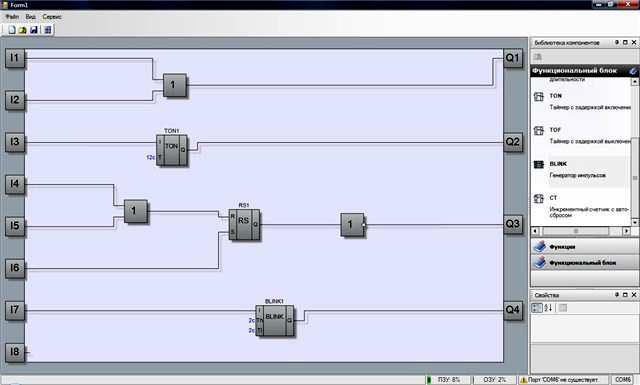


Рисунок 2.16 *Function Block Diagram*

*FLProg* – система візуального програмування плат *Arduino*- мовою релейної логіки. Синтаксис мови зручний для заміни логічних схем, виконаних на релейній техніці. Мова орієнтована на інженерів по автоматизації. Забезпечує наочний інтерфейс логіки роботи контролера, який полегшує не тільки завдання власне програмування і введення в експлуатацію, але і швидкий пошук неполадок в підключенні до контролера обладнання. Основними елементами мови є контакти (рис. 2.16), які можна уявно порівняти з парою контактів реле або кнопки. Пара контактів ототожнюється з логічною змінною, а стан цієї пари - зі значенням змінної.

Такий підхід виявився дуже зручним для легкого входження в розробку систем АСУ інженерів електриків та електронників. Розробляючи проекти установок, вони могли легко прив'язати роботу цих установок до алгоритмів роботи контролера. В обслуговуванні цих установок так само краще коли існуючий обслуговуючий персонал може легко перевірити роботу системи АСУ, знайти проблему. І при цьому немає необхідності викликати по кожній дрібниці програміста з ”Центру”. І це підхід себе виправдав. На сьогоднішній день майже всі системи автоматики створені за допомогою таких засобів розробки.

Проект складається з двох частин:

* Перша частина це десктоп-додаток *FLProg*, що представляє собою графічне середовище програмування плат *Arduino*.
* Друга частина це сайт *FLProg.ru*, за допомогою якого члени спільноти користувачів програми можуть поспілкуватися між собою, дізнатися останні новини проекту, завантажити останню версію програми, ну і знайти необхідну інформацію по роботі з додатком.

Програма працює на комп'ютері під керуванням *OS Windows, Linux-32 і Linux-64*.

При створенні нового проекту (рис. 2.11) вам запропонують вибрати мову програмування, на якій ви будете створювати проект, і контролер, на якому цей проект буде реалізований.

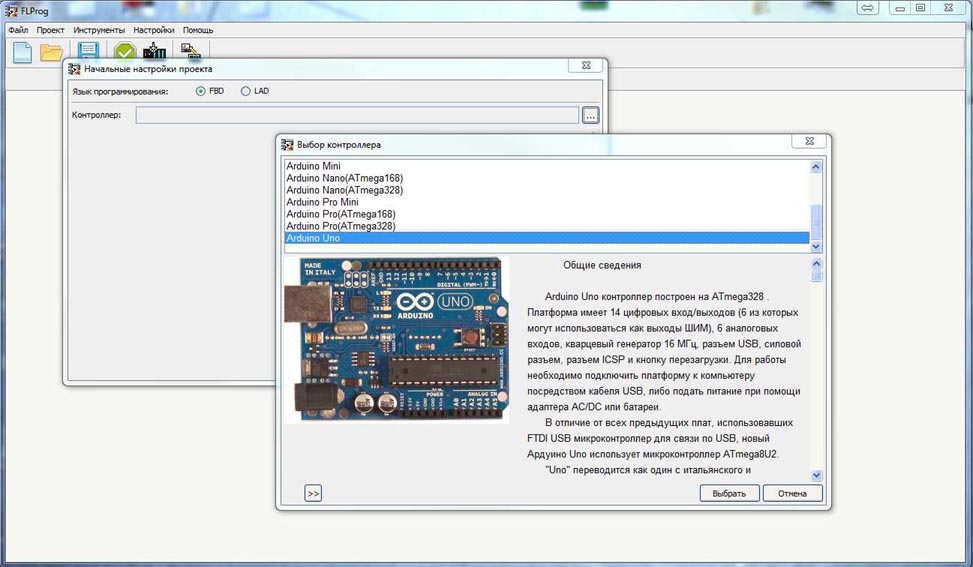


Рисунок 2.17 Створення нового проекту в середовищі *FlProg*

Список плат *Arduino*, які підтримує програма на сьогоднішній день:

* *Arduino Diecimila*
* *Arduino Duemilanove*
* *Arduino Leonardo*
* *Arduino Lilypad*
* *Arduino Mega 2560*
* *Arduino Micro*
* *Arduino Mini*
* *Arduino Nano (ATmega168)*
* *Arduino Nano (ATmega328)*
* *Arduino Pro Mini*
* *Arduino Pro (ATmega168)*
* *Arduino Pro (ATmega328)*
* *Arduino Uno*
* *Intel-Galileo*

У правій частині робочої зони розташована бібліотека елементів. У схему елементи переносяться простим перетягуванням. При подвійному натисканні по елементу буде показана інформація про нього.У верхній частині робочої зони розташований список тегів (змінних і входів виходів) (*FBD*) або встановленого обладнання (*LAD*) (рис. 2.17). Теги або обладнання переносяться на схему простим перетягуванням.

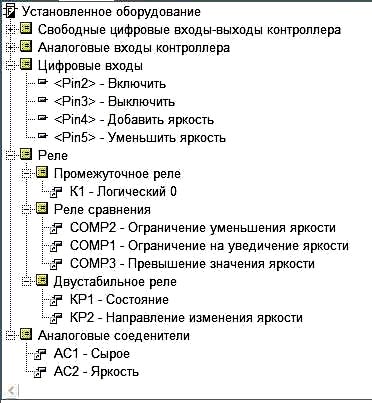
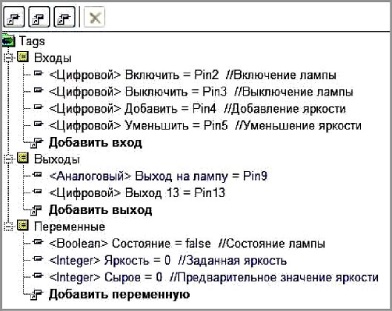


Рисунок 2.18 Список тегів та встановленого обладнання

Після завершення роботи над проектом проводиться його компіляція. Після компіляції автоматично відкриється програма "*Arduino IDE*" з завантаженим скетчем вашого проекту.

2.4 Вибір модуля для реалізації каналу зв’язку

Щоб забезпечити канал зв’язку потрібно підібрати найбільш безпечний і досить потужний варіант . Розглядається кілька варіантів.

ESP8266 – мікроконтролер китайського виробника Espressif з інтерфейсом Wi-Fi. Крім Wi-Fi мікроконтролер відрізняється можливістю виконувати програми з зовнішньої флеш-пам'яті з інтерфейсом SPI. ESP8266 може працювати як в ролі точки доступу так і кінцевої станції. При нормальній роботі в локальній мережі ESP8266 конфігуруєтся в режим кінцевої станції. Для цього пристрою необхідно задати SSID Wi-Fi мережі і, в закритих мережах, пароль доступу. Для початкового конфігурування цих параметрів зручний режим точки доступу. У режимі точки доступу пристрій видно при стандартному пошуку мереж в планшетах і комп'ютерах. Залишається підключитися до пристрою, відкрити HTML сторінку конфігурації і задати параметри мережі. Після чого пристрій штатно підключиться до локальної мережі в режимі кінцевої станції.

Bluetooth-модуль HC-06 – з HC-06 є можливість керувати роботом прямо зі свого смартфона. Поставивши на телефон або планшет одну з численних програм для керування через bluetooth, перетворити його в справжній продвинутий джойстик, і робот зможе повертати слідом за поворотом смартфона. Точно так само є можливість зв'язатися зі своїм пристроєм з ноутбука, або з будь-якого іншого приладу, який може підключатися до bluetooth-пристроїв.

Модуль nRF24L01 – це цифровий приймач і передавач, укладений в одній маленькій мікросхемі. Розмір плати, на якій розміщується мікросхема, необхідна обв'язка і невелика антена становить всього 15 x 29 мм. Коротка специфікація радіомодуля:

– частота: 2.4 ГГц;

– робоча напруга: від 3.3 до 3.6 В;

– дальність: до 100 м на відкритому просторі, і до 30 м в приміщенні;

– швидкість: 2 Мб / c;

– кількість каналів: 125.

Радіомодуль використовують для управління мобільним роботом: рухаємо джойстик , робот їде в потрібному напрямку. У зворотний бік цей же робот може відправляти дані телеметрії: показання одометра, інклінометра, і різних інших датчиків.

Інший варіант застосування – збір даних дистанційних датчиків. nRF24L01 вміє працювати з топологією один-до-багатьох. Це означає, що один радіомодуль може тримати зв'язок одночасно з безліччю інших. Завдяки цьому, датчики температури, вологості і освітленості, встановлені по всій площі великого цеху або теплиці можуть передавати показання на базову станцію для подальшого аналізу. Ще одна можливість використання погодна станція у дворі житлового будинку, або на його даху.

Проект реалізована на модулі nRF24L01, тому що модуль має можливість передачі даних на великих відстанях, не такий вразливий до перешкод і має функцію передачі даних в зворотній бік, може перемикатися в режим низького споживання енергії.

Висновок

На основі проведеного аналізу предметної області та основних функцій цільового програмного модулю складено ряд вимог до програмного забезпечення. Розроблена програмна специфікація відображає деталізовану ієрархію задач та функцій, які повинен виконувати цільовий програний додаток.

Проведено перелік технологій, які дозволять реалізувати практичну складову роботи. Обгрунтовано вибір інструментальних засобів та середовища розробки програмного забезпечення, обрано апаратну базу для реалізації прототипу пересувної платформи з описом всіх необхідних модулів.

РОЗДІЛ 3

РОЗРОБКА ПРОГРАМНОГО МОДУЛЮ

3.1 Розробка схеми програмного модуля та її опис

3.1.1 Опис блок-схеми керуючого пристрою

1. Оголошення змінних;
2. Встановлення налаштувань радіомодуля (виконується один раз при ввімкненні пристрою);
3. Зчитування даних з джойстиків, обробка отриманих значень та відправка через радіомодуль, вивід даних в COM порт.
4. За відсутності живлення на мікроконтролер програма припиняє виконуватися.

3.1.2 Опис блок-схеми наземної патформи

1. Оголошення змінних;
2. Встановлення налаштувань радіомодуля, драйвера двигунів, дальнометра та сервопривода;
3. Встановлення значення роботи двигунів в нуль, перевірка умов та виконання відповідних дій;
4. Зміна положення сервопривода в залежності від значення змінної *i;*
5. За відсутності живлення на мікроконтролер програма припиняє виконуватися.

3.2 Метод розрахунку відстані до перешкоди

*int calculateDistance(){*

*digitalWrite(trigPin, LOW);*

*delayMicroseconds(2);*

*digitalWrite(trigPin, HIGH);*

*delayMicroseconds(10);*

*digitalWrite(trigPin, LOW);*

*duration = pulseIn(echoPin, HIGH);*

*distance= duration\*0.034/2;*

*return distance;*

}

Для того щоб ініціювати відправку сигналу далекоміром, необхідно подати високий сигнал тривалістю 10 мікросекунд на вихід *Trig*.

Після отримання високого сигналу тривалістю 10 мкс на вихід Trig, модуль генерує пучок з восьми сигналів частотою 40 кГц і встановлює високий рівень на виході *Echo*.

Після отримання відбитого сигналу модуль встановлює на виході *Echo* низький рівень.

Знаючи тривалість високого сигналу на виході *Echo* можемо обчислити відстань, помноживши час, який витратив звуковий імпульс, перш ніж повернувся до модуля, на швидкість поширення звуку в повітрі (340 м / с).

Функція *pulseIn()* дозволяє дізнатися тривалість імпульсу в мікросекундах. Запишемо результат роботи цієї функції в змінну *duration*.

Тепер обчислимо відстань перевівши швидкість з м / с в см / мкс:

*distance* = *duration* \* 340 м / с = *duration* \* 0.034 м / мкс (3.1)

3.3 Опис використаних операторів та функцій

* *Int ()* - функція, яка повертає цілі числові значення;
* *If* Умова *Then* [Інструкція] [*Else* Інструкціі\_*else*] - оператор умовного переходу. Якщо умова приймає значення *True*, то виконується інструкція *Then*, якщо *False*, то виконується інструкція\_*else*. Гілка *Else* є необов'язковою;
* *FOR*-цикл. Функції: багаторазове проходження по одному і тому ж коду програми.
* Функція setup (). Функція setup () викликається, коли стартує скетч. Використовується для ініціалізації змінних, визначення режимів роботи виходів, запуску використовуваних бібліотек і т.д. Функція setup запускає тільки один раз, після кожної подачі живлення або скидання плати.
* Функція Loop(). Після виклику функції setup (), яка ініціалізує і встановлює початкові значення, функція loop () робить те ж, що означає її назва, і крутиться в циклі, дозволяючи програмі здійснювати обчислення і реагувати на них.
* Директива #include. Використовується для підключення сторонніх бібліотек в скетч.
* Арифметичні оператори.
* Оператори порівняння.
* *pinMode()* – функція, яка встановлює режим роботи входу.
* *digitalWrite()* – відправляє на вихід *HIGH* або *LOW*.
* *digitalread()* – зчитує рівень сигналу *HIGH* або *LOW*.
* *delay() –* призупиняє виконання програми на вказаний проміжок часу.

3.4 Принцип роботи наземної платформи

Основним завданням автоматизованого наземного мобільного шасі є транспортування засобів зв’язку для розширення зони покриття або для їх розміщення в тяжко доступній місцевості пов’язаній з підвищеним ризиком для життя особового складу. Також автоматизоване наземне мобільне шасі може бути використане, як пересувна платформа автоматизованих систем для збору оперативних даних.

Прототип автоматизованого наземного мобільного шасі (рис. 3.1) працює на базі мікроконтролерів *Arduino UNO*, *Arduino MEGA*, які забезпечують роботу всіх елементів системи та їх живлення.

Радіомодуль *NRF*24L01+ взаємодіє з платою *Arduino* через інтерфейс *SPI*. Живлення 3.3В на радіомодуль йде з плати *Arduino* (в платі є стабілізатор напруги). Скетч в *Arduino* завантажується при відключеному харчуванні.

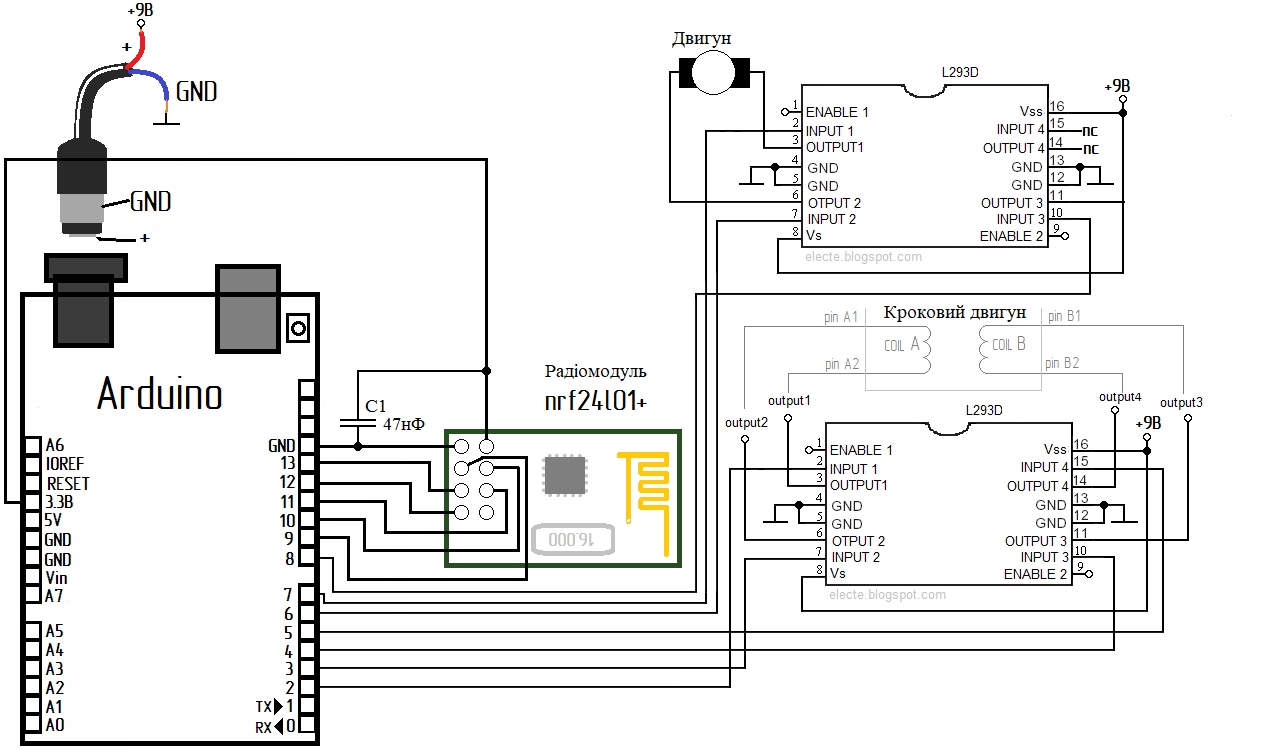


Рисунок 3.1 Схема приймача

У кожного *Arduino* є виходи *SPI* але зазвичай на платі вони не підписані, з'ясувати де ці висновки знаходяться можна з документації на конкретного *Arduino*. На рисунку 3.2 показаний джойстик. Джойстик зазвичай можна представити у вигляді набору кнопок. Якщо відповідна кнопка натиснута то на відповідному виходу *Arduino* з'явитися низька напруга (логічний нуль), якщо кнопку не було натиснуто то на виводі буде висока напруга (логічна одиниця) тому, що цей вихід з'єднаний з напругою +5 В через резистор з опором 10кОм. *Arduino* можна живити однієї батарейкою "Крона" на 9В, це живлення подається на *Arduino* через спеціальний роз'єм.

Технічні характеристики радіомодуля :

* Частота передачі / прийому 2.4ГГц.
* Дальність до 100 М. В приміщенні - до 30 м.
* Швидкість до 2 Мб. Можливі варіанти: 250kbps, 1Mbps і 2Mbps.
* Інтерфейс взаємодії з мікроконтроллером – *SPI*.
* Напруга: 3-3.6В (рекомендується 3,3) В. При спробі підключення 5 В виведення з ладу малоймовірне, але може привести до поломки радіомодуля.
* Кількість каналів: 126. Нульовий канал починається з 2400МГц і далі з кроком 1 мегагерц, наприклад 70 канал знаходиться відповідно на 2470мгц.

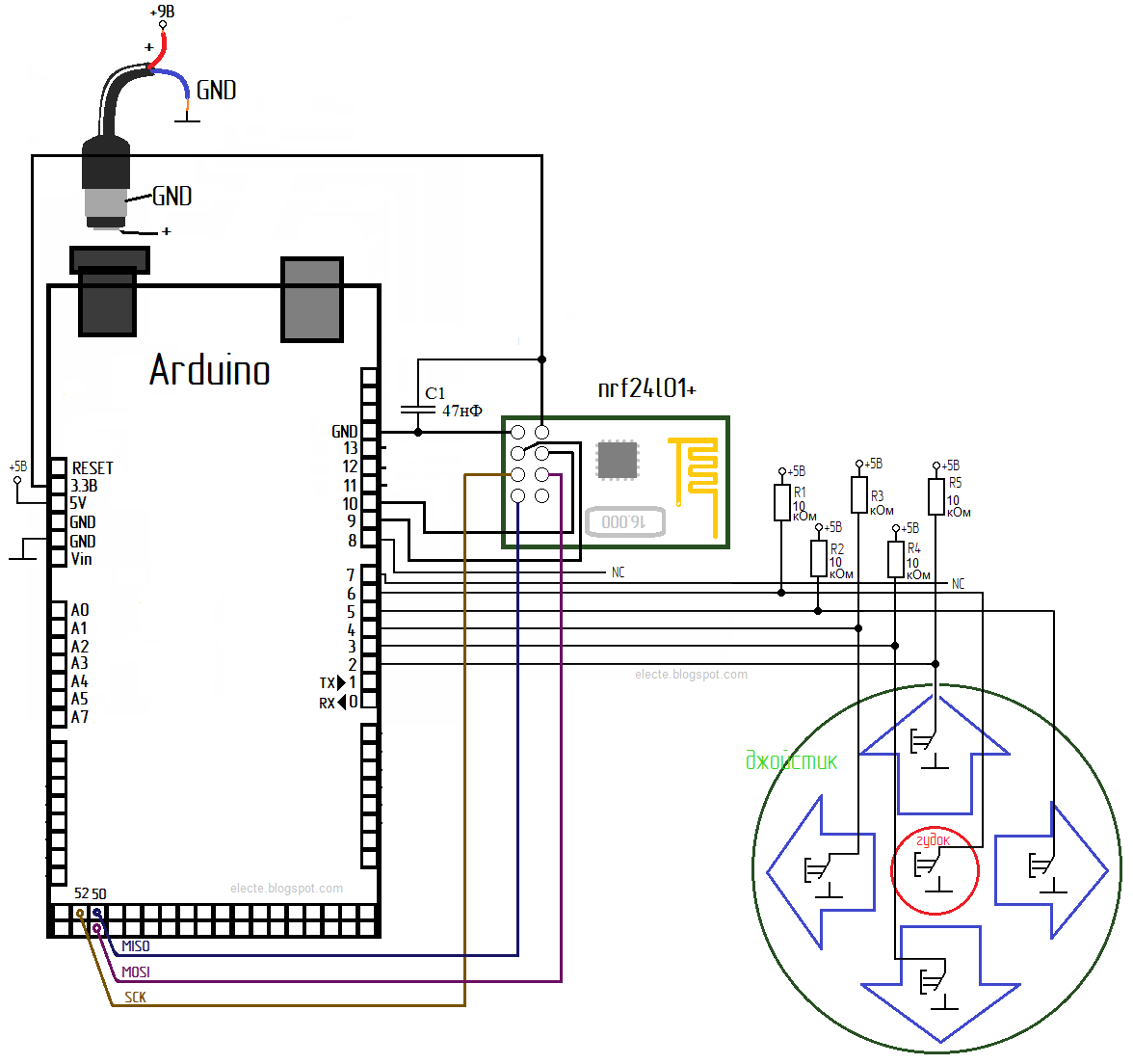


Рисунок 3.2 Схема передавача

3.5 Опис використаних модулів

3.5.1 Опис джойстика

Ручка джойстика (рис. 3.3) закріплена на шарнірі і пов'язана з пружинами, які повертають ручку в центральне положення після відхилення. Всередині два змінних резистора по 10 кОм. У джойстику розміщена кнопка. Її штовхач механічно пов'язаний з ручкою. При натисканні на ручку контакти кнопки замикаються. Цей елемент використовують для управління переміщенням по третій координаті або для інших функцій. Механічна частина розміщена на платі і має отвори. За допомогою отворів джойстик монтується в корпусі пульта управління. Електричні лінії підключаються до контактів за допомогою роз'ємного з'єднання.

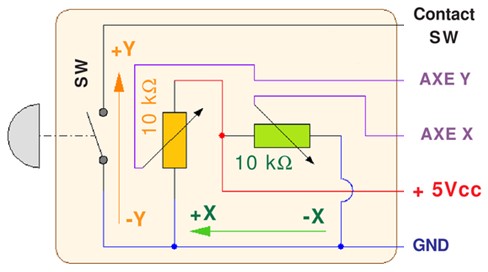


Рисунок 3.3 Схема опису джойстика

3.5.2 Опис драйвера двигунів

Драйвер (англ. *Driver* - керуючий пристрій, водій) - електронний пристрій, призначений для перетворення електричних сигналів, метою якого є управління чимось. Драйвером зазвичай називається окремий пристрій або окремий модуль, мікросхема в пристрої, що забезпечують перетворення електричних сигналів в електричні або інші впливи, придатні для безпосереднього управління виконавчими або сигнальними елементами.

Під визначення драйвера підпадають численні пристрої:

* Шинні формувачі, призначені для передачі сигналів з одного рівня цифрового пристрою на інший з перетворенням рівня, посиленням здатності навантаження та іншими особливостями. Такі пристрої забезпечують передачу даних між різними логічними блоками по загальним лініях зв'язку всередині обчислювальних машин.
* Формувачі сигналів інтерфейсів цифрових електронних пристроїв, призначені для перетворення, прийому і передачі цифрових сигналів і узгодження електричних параметрів з особливостями лінії зв'язку. Найбільш відомими представниками таких драйверів вважаються формувачі інтерфейсів *RS-232 (COM -* порт*), RS-485, RS-422, CAN, LIN, Ethernet, USB, IEEE* 1394 і т. Д.
* Пристрої управління різними типами виконавчих пристроїв, такими як електромагніти, електродвигуни (у тому числі крокові), сигнальні лампи, дозатори (у тому числі друкують голівки принтерів), сервоприводи, звукові сигнали.
* Модулі живлення і управління пристроями, що вимагають дотримання певних робочих параметрів в процесі включення, виключення і роботи. Яскравим прикладом можна вважати драйвери світлодіодів, оскільки до живлення світлодіодних пристроїв пред'являються підвищені вимоги.
* Драйвери силових транзисторів, MOSFET і IGBT-транзисторів. Затвори потужних польових силових транзисторів мають велику електричну ємність (тисячі пікофарад), для зарядки яких на високій частоті потрібен великий струм (ампер). Драйвер забезпечує великий струм для швидкої зарядки затвора транзистора для його відкриття. А також швидко розряджає затвор, коли транзистор потрібно закрити.

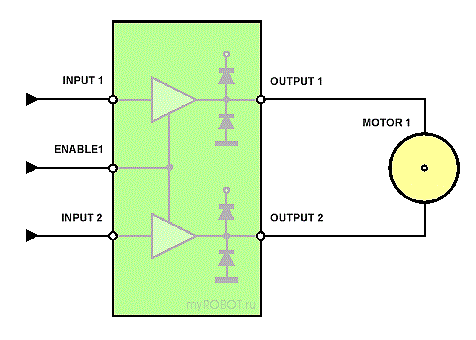


Рисунок 3.4 Схема роботи драйвера двигунів

Якщо на вхід ENABLE1, що включає драйвер, подати сигнал (з'єднаємо з позитивним полюсом джерела живлення + 5V) і при цьому на входи INPUT1 і INPUT2 не подавати сигнали, то мотор обертатися не буде.

Якщо вхід INPUT1 з'єднати з позитивним полюсом джерела живлення, а вхід INPUT2 - з негативним, то мотор почне обертатися.

Тепер з'єднаємо вхід INPUT1 з негативним полюсом джерела живлення, а вхід INPUT2 - з позитивним. Мотор почне обертатися в інший бік. Якщо подати сигнали одного рівня відразу на обидва керуючих входу INPUT1 і INPUT2 (з'єднати обидва входи з позитивним полюсом джерела живлення або з негативним) - мотор обертатися не буде.

3.5.3 Опис серводвигуна

Серводвигун - це мотор-редуктор, зі зворотним зв'язком, завдяки якому можна повернути вихідний вал на строго вказаний кут. Стандартні серводвигуни, крутяться від 0 до 180 градусів.

Функції, які надає бібліотека *Servo library:*

1. *servo.attach(pin, min, max) – pin* номер порта до якого підключений серводвигун, *min* ширина імпульсу в мікросекундах, встановлює положення вала серводвигуна в 0 градусів, max ширина імпульсу в мікросекундах встановлює положення вала сервводвигуна в 180 градусів.
2. *void write(int value) –* встановлює кут повороту серводвигуна в градусах. *Value* значення кута, на яке потрібно повернути вал.
3. *void writeMicroseconds (int value) –* задає значення в мікросекундах для тривалості керуючого імпульсу. На стандартних серводвигуна це призведе до повороту вала на певний кут (1000 - положення повністю проти годинникової стрілки, 2000 - положення повністю за годинниковою стрілкою, і 1500 - в середині).
4. *int read() –* зчитує «поточний» кут повороту серводвигуна (значення, передане в останньому виклику write ()). Повертає значення типу int – кут від 0 до 180 градусів.

3.5.4 Опис Бібліотека *SPI*

Бібліотека *SPI* дозволяє контролеру Arduino взаємодіяти з пристроями що підтримують *SPI* протокол. Arduino в даному випадку виступає в якості керуючого пристрою. Послідовний периферійний інтерфейс (*SPI*) - це послідовний синхронний протокол передачі даних використовується мікроконтролерами для обміну даними з одним або декількома периферійними пристроями на невеликих відстанях.

*SPI.begin ()* – ініціалізує шину *SPI*, встановлюючи піни *SCK, MOSI*, і *SS* як виходи і рівень сигналу на *SCK* і *MOSI* - *LOW* і на *SS* - *HIGH*.

*SPI.end ()* – відключає шину SPI (залишаючи режими вхід / вихід в тому ж стані).

Функція *SPI.setBitOrder()* встановлює порядок виведення даних в/з шини *SPI*, може бути *LSBFIRST* (найменший розряд (біт) перший) або *MSBFIRST* (старший розряд перший).

Синтаксис: *SPI.setBitOrder (order)*;

Параметри: *order*: *LSBFIRST* або *MSBFIRST*;

*SPI.setClockDivider () –* встановлює дільник частоти синхронізації SPI до частоти контролера. Допустимі такі подільники: 2, 4, 8, 16, 32, 64 або 128. Значення за замовчуванням дорівнює *SPI\_CLOCK\_DIV4*, одна чверть від частоти контролера.

3.5.5 Опис бібліотеки *RF24*

*RF24()* – конструктор. Створює новий екземпляр цього драйвера. Перед використанням необхідно створити екземпляр і передати унікальні Піни, до яких цей чіп підключений.

Параметри:

*\_cepin*: контакт модуля *Chip Enable*

\_*cspin*: контакт модуля *Chip Select*

*begin*

Початок роботи радіомодуля. Викликаємо його в *setup ()*, перед викликом будь-яких інших методів.

*setChannel –* набір каналів зв'язку *RF*.

Параметри: *channel* – радіочастотний канал для спілкування, 0-127.

*startListening()* – почати слухати канали, відкриті для читання. Спочатку переконайтеся що відкриті канали для читання. Чи не викликайте «*write*», в цьому режимі, без попереднього виклику «*stopListening*».

*stopListening()* – припинити слухати вхідні повідомлення. Необхідно це зробити перш, ніж писати.

*Write()* – написати повідомлення в відкритий канал. Це блок працює поки повідомлення не матиме успіху прийнято приймачем або тайм-аут / максимум спроб було досягнуто. У поточній конфігурації, максимальна затримка тут 60мс.

Параметри:

*buf*: Покажчик на дані, що підлягають передачі

Повертає:

*True* - якщо дані були успішно доставлено,

*False* - якщо не доставлено.

*Available()*: перевірка, чи є байт, доступний для читання.

Повертає:

*True* - якщо дані є в наявності,

*False* - якщо немає

*Read()* – читає дані. Повернути останні отримані дані.

Параметри:

*buf*: покажчик на буфер, де дані повинні бути записані

повертає:

*True* - якщо дані були успішно доставлені,

*False* - якщо не доставлені.

*openWritingPipe()* – відкрити канал для читання.

Висновок

В даному розділі відповідно до поставлених задач спроектовано та розроблено програмний модуль, наведено основні програмні конструкції додатку з відповідним описом кожного блоку; описано апаратні модулі, які використовуються для демонстрації адекватності роботи програмної логіки визначеному функціоналу; описано сторонні бібліотеки та обгрунтовано необхідність їх застосування в коді програми. Також наведено структурну схему приймача та блок-схему керуючого пристрою.

РОЗДІЛ 4

ОЦІНКА ЯКОСТІ ПРОГРАМНОГО МОДУЛЮ НАЗЕМНОЇ ПЕРЕСУВНОЇ ПЛАТФОРМИ

Програмний модуль було оцінено відповідно до характеристик та завдань, які були поставлені.

1.Функціональність

* + Функціональна придатність: програмний модуль в повній мірі вирішує поставлені задачі.
  + Здатність до взаємодії: модуль легко взаємодіє з іншим програмним забезпеченням.
  + Захищеність: усі функції модулю захищені від використання сторонніми особами засобами авторизації.

2.Надійність

* Зрілість: радіомодуль модуль працює безвідмовно при наявності усіх основних факторів.

3.Зручність використання

* Зрозумілість: модуль має інтуїтивний та зрозумілий інтерфейс взаємодії з користувачем.
* Зручність роботи: в результаті опитування 80% тестувальників модулю відмітили інтерфейс зручним.

4.Ефективність

* Часова ефективність: в ході тестування програмний модуль справився з вчасною передачею даних на відмінно.
* Ефективність використання ресурсів: програмний модуль в ході тестування показав дуже економне використання мікроконтролера (5-25%).

5.Зручність супроводу

* + Зручність внесення змін: програмний модуль написаний по принципу модульності написання коду, що мінімізує витрати часу при внесенні змін.
  + Зручність тестування: в програмному модулі реалізовано тести для перевірки правильності відпрацювання блоків коду, тому модуль зручний для тестування.
* Переносимість: модуль можливо легко переносити на різні робочі станції.

6. Оцінка ефективності програмного та радіомодулів

Таблиця 4.1 Результати оцінки ефективності прототипу

|  |  |
| --- | --- |
| Дальність (м) | Кількість переданих даних (кб) |
| 10 | 100 |
| 40 | 100 |
| 60 | 100 |
| 90 | 99 |
| 100 | 98 |

Програмний та радіомодулі наземної платформи були протестовані на кафедрі № 22. Як свідчить приведена вище оцінка якості згідно стандарту та даних таблиці 4.1 – обидва модулі здатні виконувати поставлені задачі на 100%.

Для тестування було використано навчальна база кафедри №22.

Основним завданням автоматизованого наземного мобільного шасі є транспортування засобів зв’язку для розширення зони покриття або для їх розміщення в тяжко доступній місцевості пов’язаній з підвищеним ризиком для життя особового складу. Також автоматизоване наземне мобільне шасі може бути використане, як пересувна платформа автоматизованих систем для збору оперативних даних.

При подачі живлення на плати *arduino* *uno* *r3* та *arduino* *mega* 2560 активуються радіомодулі, які утворюють між собою радіоканал за допомогою якого створюється управління мобільною платформою, по радіоканалу передаються значення з джойстика .

Значення, які передаються з джойстика приймаються та обробляються в процесорі мікроконтролера, потім подаються на драйвер двигуна, а він в свою чергу подає відповідне живлення на відповідний двигун.

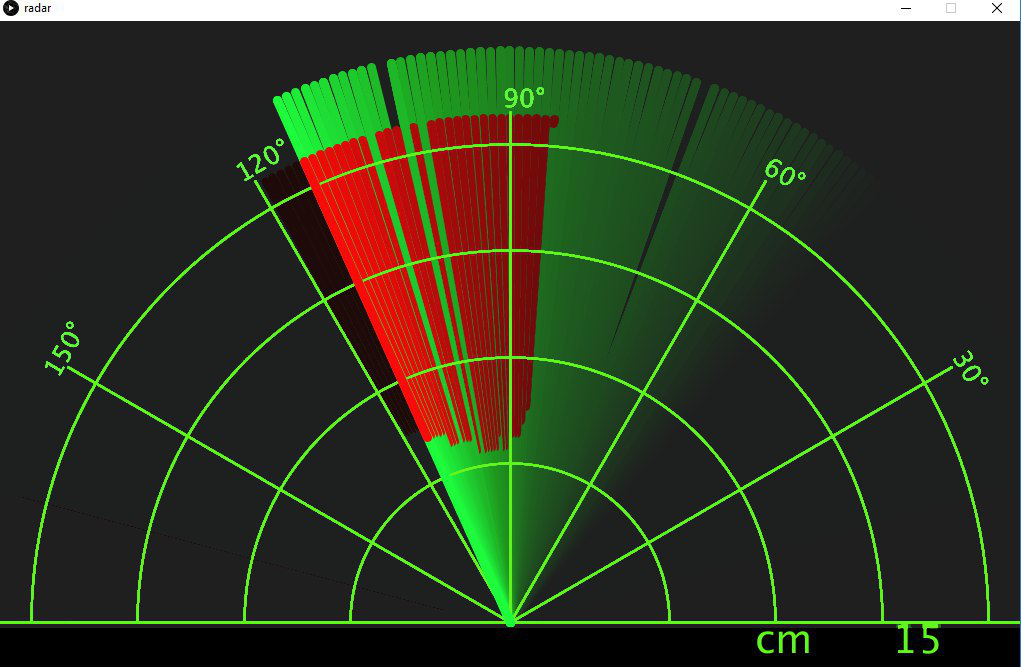


Рисунок 4.1 Функція виявлення перешкод

Функція виявлення перешкод (рис. 4.1) та виміру відстані до них працює за допомогою датчика дальнометр HC-SR04, при подачі на нього живлення він виміряє відстань до перешкоди ультразвуковою хвилею.

Висновок

У даному розділі проведено оцінку якості програмного модулю наземної платформи, а також оцінено ефективність застосування прототипу пересувної платформи, як одного із основних способів транспортування засобів зв’язку для розширення зони покриття або для їх розміщення в тяжко доступній місцевості, пов’язаній з підвищеним ризиком для життя особового складу.

ВИСНОВКИ

Питання розширення спектру застосування у військах роботизованої техніки стоїть досить гостро, а наявність малої кількості уніфікованих програмно-апаратних рішень зумовлює необхідність вдосконалення існуючих та розробки нових зразків дистанційно керованих засобів озброєння, оснащення їх максимально ефективними засобами спостереження та ураження, а також необхідність в отриманні ними здатності змінювати призначення в залежності від завдань, що поставлені, та діяти сумісно з авіаційними безпілотними засобами та іншою військовою технікою.

На основі проведеного аналізу предметної області та основних функцій цільового програмного модулю складено ряд вимог до програмного забезпечення. Розроблена програмна специфікація відображає деталізовану ієрархію задач та функцій, які повинен виконувати цільовий програний додаток. 1

Проведено перелік технологій, які дозволять реалізувати практичну складову роботи. Обгрунтовано вибір інструментальних засобів та середовища розробки програмного забезпечення, обрано апаратну базу для реалізації прототипу пересувної платформи з описом всіх необхідних модулів. Спроектовано та розроблено програмний модуль, наведено основні програмні конструкції додатку з відповідним описом кожного блоку; описано апаратні модулі, які використовуються для демонстрації адекватності роботи програмної логіки визначеному функціоналу; описано сторонні бібліотеки та обгрунтовано необхідність їх застосування в коді програми. Також наведено структурну схему приймача та блок-схему керуючого пристрою.

Розроблений програмно-апаратний прототип наземної пересувної платформи відповідає всім висунутим вимогам та виконує передбачений функціонал у повному обсязі.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

# 1. Улли Сомер Arduino Mikrocontroller-Programmiering mit Arduino/Freeduino 2012.

# 2. Бєлов А.В. мікроконтролери AVR: від азів програмування до створення практичних пристроїв(2017).

# 3. Віктор Петін мікрокомп'ютери Raspberry Pi. практичний посібник.

# 4. Іванов А. О. Теорія автоматичного керування: Підручник. — Дніпропетровськ: Національний гірничий університет. — 2003. — 250 с.

# 5. Енциклопедія кібернетики. тт. 1, 2. — К.: Головна редакція УРЕ, 1973.

# 6. Офіційний сайт Arduino. – Режим доступа: http://www.arduino.cc/. Дата доступа: 23.05.2015.

# 7. Офіційний сайт 8Devices. – Режим доступа: http://www.8devices.com. - Дата доступа: 29.05.2015.

# 8. Хабр-Хабр. – Режим доступу: http://habrahabr.ru. - Дата доступа: 30.05.2015.

# 9. Медведев В. С., Лесков А. Г., Ющенко А. С. Системы управления манипуляционных роботов. — М.: Наука, 1978. — 416 с.

# 10. Фу К., Гонсалес Р., Ли К. Робототехника / Пер. с англ. — М.: Мир, 1989. — 624 с.

# 11. Форум іноваційних технологий – Режим доступу: http://innotech.kiev.ua/. - Дата доступа: 01.06.2015.

# 12. Микушин А. Занимательно о микроконтроллерах. — М.: БХВ- Петербург, 2006

# 13. Шабров О.Ф. Компьютеризация / О. Ф. Шабров. – Социологическая энциклопедия: В 2 т. – Т.1. – М.: Мысль, 2003. – С.462-464.

Додаток А

Лістинг програми керуючого пристрою

*#include <nRF24L01.h>*

*#include <RF24.h>*

*#include <SPI.h>*

*byte X =1;*

*char incomingByte;*

*byte Y = 5;*

*byte mass[2];*

*RF24 radio(9, 10);*

*void setup() {*

*Serial.begin(9600);*

*radio.begin();*

*radio.setChannel(9);*

*radio.setDataRate(RF24\_1MBPS);*

*radio.setPALevel(RF24\_PA\_MAX);*

*radio.openWritingPipe(0x1234567890LL);*

*}*

*void loop() {*

*mass[0] = map(analogRead(X),0 , 1024, 0, 10);*

*mass[1] = map(analogRead(Y),0 , 1024, 20 , 30);*

*radio.write(&mass, sizeof(mass));*

*Serial.println(mass[0]);*

*Serial.print("\t");*

*Serial.print("\t");*

*Serial.println(mass[1]);*

*}*

Додаток Б

Лістинг програми наземної пересувної платформи

*#include <nRF24L01.h>*

*#include <printf.h>*

*#include <RF24.h>*

*#include <RF24\_config.h>*

*#include <Servo.h>.*

*int echoPin = 8;*

*int trigPin = 9;*

*long duration;*

*int distance;*

*Servo myServo;*

*RF24 radio(48,53);*

*byte mass[2];*

*int ENA = 2;*

*int IN1 = 3;*

*int IN2 = 4;*

*int IN3 = 5;*

*int IN4 = 6;*

*int ENB = 7;*

*void setup()*

*{*

*Serial.begin(9600);*

*radio.begin();*

*radio.setChannel(9);*

*radio.setDataRate(RF24\_1MBPS);*

*radio.setPALevel(RF24\_PA\_MAX);*

*radio.openReadingPipe (1, 0x1234567890LL);*

*radio.startListening ();*

*pinMode (ENA, OUTPUT);*

*pinMode (ENB, OUTPUT);*

*pinMode (IN1, OUTPUT);*

*pinMode (IN2, OUTPUT);*

*pinMode (IN3, OUTPUT);*

*pinMode (IN4, OUTPUT);*

*pinMode(trigPin, OUTPUT);*

*pinMode(echoPin, INPUT);*

*myServo.attach(35);*

*}*

*unsigned int time\_us = 0;*

*unsigned int distance\_sm = 0;*

*void loop(){*

*digitalWrite (IN3, LOW);*

*digitalWrite (IN4, LOW);*

*analogWrite(ENB,0);*

*digitalWrite (IN1, LOW);*

*digitalWrite (IN2, LOW);*

*analogWrite(ENA,0);*

*for(int i=15;i<=165;i++){*

*myServo.write(i);*

*delay(20);*

*distance = calculateDistance();*

*if(radio.available())*

*{*

*radio.read(&mass, sizeof(mass));*

*}*

*digitalWrite (IN3, LOW);*

*digitalWrite (IN4, LOW);*

*digitalWrite (IN1, LOW);*

*digitalWrite (IN2, LOW);*

*analogWrite(ENA,0);*

*if ( mass[1] < 25 ){*

*digitalWrite (IN1, LOW);*

*digitalWrite (IN2, HIGH);*

*analogWrite(ENA,100);*

*}*

*if ( mass[1] > 25 ){*

*digitalWrite (IN1, HIGH);*

*digitalWrite (IN2, LOW);*

*analogWrite(ENA,100);*

*}*

*if ( mass[0] < 4 ){*

*digitalWrite (IN3, LOW);*

*digitalWrite (IN4, HIGH);*

*analogWrite(ENB,200);*

*}*

*if ( mass[0] > 4 ){*

*digitalWrite (IN3, HIGH);*

*digitalWrite (IN4, LOW);*

*analogWrite(ENB,200);*

*}*

*Serial.print(i);*

*Serial.print(",");*

*Serial.print(distance);*

*Serial.print(".");*

*}*

*for(int i=165;i>15;i--){*

*myServo.write(i);*

*delay(30);*

*distance = calculateDistance();*

*if(radio.available())*

*{*

*radio.read(&mass, sizeof(mass));*

*}*

*digitalWrite (IN3, LOW);*

*digitalWrite (IN4, LOW);*

*analogWrite(ENB,0);*

*digitalWrite (IN1, LOW);*

*digitalWrite (IN2, LOW);*

*analogWrite(ENA,0);*

*if ( mass[1] < 25 ){*

*digitalWrite (IN1, LOW);*

*digitalWrite (IN2, HIGH);*

*analogWrite(ENA,100);*

*}*

*if ( mass[1] > 25 ){*

*digitalWrite (IN1, HIGH);*

*digitalWrite (IN2, LOW);*

*analogWrite(ENA,100);*

*}*

*if ( mass[0] < 4 ){*

*digitalWrite (IN3, LOW);*

*digitalWrite (IN4, HIGH);*

*analogWrite(ENB,200);*

*}*

*if ( mass[0] > 4 ){*

*digitalWrite (IN3, HIGH);*

*digitalWrite (IN4, LOW);*

*analogWrite(ENB,200);*

*}*

*Serial.print(i);*

*Serial.print(",");*

*Serial.print(distance);*

*Serial.print(".");*

*}*

*Serial.println(mass[0]);*

*Serial.print("\t");*

*Serial.print("\t");*

*Serial.println(mass[1]);*

*}*

*int calculateDistance(){*

*digitalWrite(trigPin, LOW);*

*delayMicroseconds(2);*

*digitalWrite(trigPin, HIGH);*

*delayMicroseconds(10);*

*digitalWrite(trigPin, LOW);*

*duration = pulseIn(echoPin, HIGH);*

*distance= duration\*0.034/2;*

*return distance;*

*}*

Додаток В

Лістинг програми системи виявлення перешкод

import processing.serial.\*;

import java.awt.event.KeyEvent;

import java.io.IOException;

Serial myPort;

// defubes variables

String angle="";

String distance="";

String data="";

String noObject;

float pixsDistance;

int iAngle, iDistance;

int index1=0;

int index2=0;

PFont orcFont;

void setup() {

size (1020, 750);

smooth();

myPort = new Serial(this,"COM4", 9600);

myPort.bufferUntil('.');

}

void draw() {

fill(98,245,31);

noStroke();

fill(0,4);

rect(0, 0, width, height-height\*0.065);

fill(98,245,31); // green color

drawRadar();

drawLine();

drawObject();

drawText();

}

void serialEvent (Serial myPort) {

data = myPort.readStringUntil('.');

data = data.substring(0,data.length()-1);

index1 = data.indexOf(",");

angle= data.substring(0, index1);

distance= data.substring(index1+1, data.length());

iAngle = int(angle);

iDistance = int(distance);

}

void drawRadar() {

pushMatrix();

translate(width/2,height-height\*0.074);

noFill();

strokeWeight(2);

stroke(98,245,31);

// draws the arc lines

arc(0,0,(width-width\*0.0625),(width-width\*0.0625),PI,TWO\_PI);

arc(0,0,(width-width\*0.27),(width-width\*0.27),PI,TWO\_PI);

arc(0,0,(width-width\*0.479),(width-width\*0.479),PI,TWO\_PI);

arc(0,0,(width-width\*0.687),(width-width\*0.687),PI,TWO\_PI);

// draws the angle lines

line(-width/2,0,width/2,0);

line(0,0,(-width/2)\*cos(radians(30)),(-width/2)\*sin(radians(30)));

line(0,0,(-width/2)\*cos(radians(60)),(-width/2)\*sin(radians(60)));

line(0,0,(-width/2)\*cos(radians(90)),(-width/2)\*sin(radians(90)));

line(0,0,(-width/2)\*cos(radians(120)),(-width/2)\*sin(radians(120)));

line(0,0,(-width/2)\*cos(radians(150)),(-width/2)\*sin(radians(150)));

line((-width/2)\*cos(radians(30)),0,width/2,0);

popMatrix();

}

void drawObject() {

pushMatrix();

translate(width/2,height-height\*0.074);

strokeWeight(9);

stroke(255,10,10); // red color

pixsDistance = iDistance\*((height-height\*0.1666)\*0.025);

if(iDistance<40){

line(pixsDistance\*cos(radians(iAngle)),-pixsDistance\*sin(radians(iAngle)),(width-width\*0.505)\*cos(radians(iAngle)),-(width-width\*0.505)\*sin(radians(iAngle)));

}

popMatrix();

}

void drawLine() {

pushMatrix();

strokeWeight(9);

stroke(30,250,60);

translate(width/2,height-height\*0.074);

line(0,0,(height-height\*0.12)\*cos(radians(iAngle)),-(height-height\*0.12)\*sin(radians(iAngle)));

popMatrix();

}

void drawText() { // draws the texts on the screen

pushMatrix();

fill(0,0,0);

noStroke();

rect(0, height-height\*0.0648, width, height);

fill(98,245,31);

textSize(25);

textSize(40);

text("Object: " + noObject, width-width\*0.875, height-height\*0.0277);

text("Distance: ", width-width\*0.26, height-height\*0.0277);

if(iDistance<90) {

text(" " + iDistance +" cm", width-width\*0.225, height-height\*0.0277);

}

textSize(25);

fill(98,245,60);

translate((width-width\*0.4994)+width/2\*cos(radians(30)),(height-height\*0.0907)-width/2\*sin(radians(30)));

rotate(-radians(-60));

text("30°",0,0);

resetMatrix();

translate((width-width\*0.503)+width/2\*cos(radians(60)),(height-height\*0.0888)-width/2\*sin(radians(60)));

rotate(-radians(-30));

text("60°",0,0);

resetMatrix();

translate((width-width\*0.507)+width/2\*cos(radians(90)),(height-height\*0.0833)-width/2\*sin(radians(90)));

rotate(radians(0));

text("90°",0,0);

resetMatrix();

translate(width-width\*0.513+width/2\*cos(radians(120)),(height-height\*0.07129)-width/2\*sin(radians(120)));

rotate(radians(-30));

text("120°",0,0);

resetMatrix();

translate((width-width\*0.5104)+width/2\*cos(radians(150)),(height-height\*0.0574)-width/2\*sin(radians(150)));

rotate(radians(-60));

text("150°",0,0);

popMatrix();

}