ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки
Кафедра ЕОМ

Методи управління мобільним роботом в умовах динамічних перешкод

Кваліфікаційна робота

Другий (магістерський рівень)

Автор:
Сорока К. Ю.,
студ. гр. СПм-23-2

Керівник:
Каргін А. О.,
проф. каф. ЕОМ

Мета дослідження

Метою даної роботи є аналіз, розробка та оптимізація методу управління мобільним роботом у складних динамічних умовах. У роботі розглядаються як класичні алгоритми управління, так і сучасні інтелектуальні підходи, включно з методами штучного інтелекту, адаптивного планування траєкторій.

Об’єкт та предмет дослідження

- Об’єктом дослідження є процес управління мобільним роботом у середовищі з динамічними перешкодами.
- Предметом дослідження є автономні методи управління роботом, зокрема такого алгоритму як VFH, що має потенціал забезпечити ефективну та безпечну навігацію робота в динамічних умовах.

Інструменти тестування

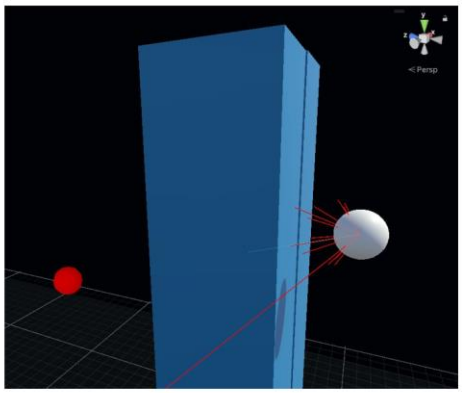
Unity — платформа для розробки застосунків та рушій для виконання програм із дво- та тривимірною графікою.

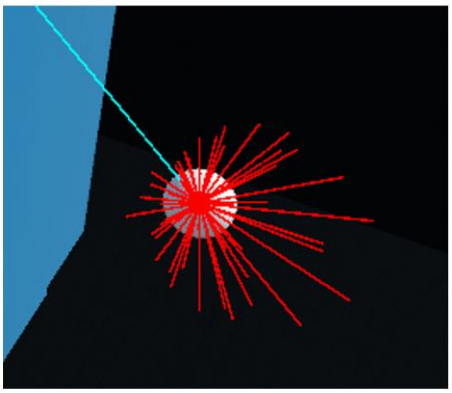
Переваги:

- Основний вектор - симуляції, навчальні програми, візуалізація фізичних процесів, робототехніка;
- Налаштування фізичних параметрів середовища;
- Відтворення поведінки БПЛА в реалістичних умовах.

3

Об’єкти тестування

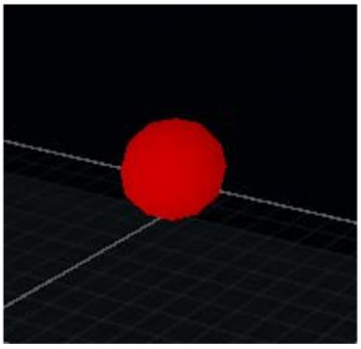




- Тестове оточення – 3D-простір, де відбувається тестування.
- Дрон – об’єкт у вигляді кулі для полегшення дослідження, симулює фізичні властивості дрона. Має синій промінь, що вказує напрямок польоту та червоні промені – 3D демонстрація гістограми VFH алгоритму.

4

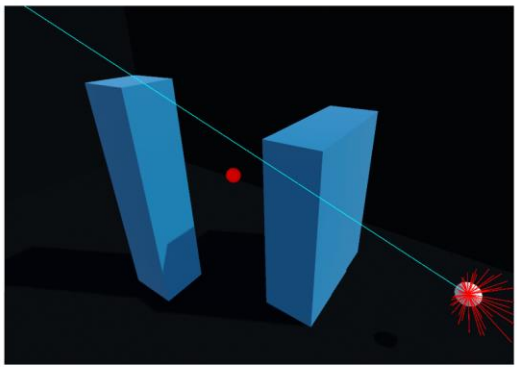
Об’єкти тестування



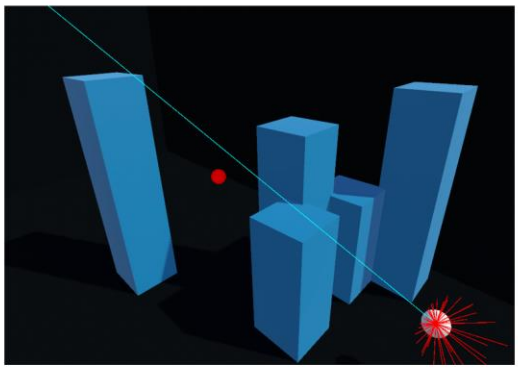
- Точка прибуття – червона куля, що вказує на фінальний пункт призначення.
- Стіна – об’єкт, що має властивість фізичної перешкоди. В подальшому дослідженні будуть продемонстровані, як статичні, так і динамічні перешкоди.

5

Класична реалізація 3D-VFH алгоритму у статичних умовах



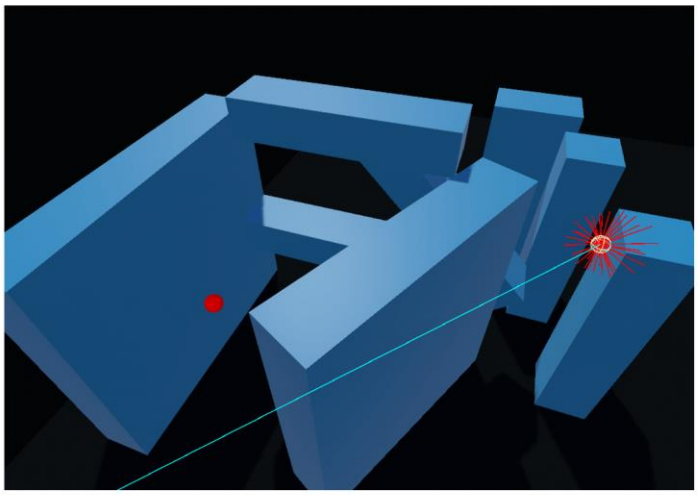
Труднощі реалізації: алгоритм не здатен оминати великі перешкоди через відсутність оптимізації зміни напрямку руху.



Результат успішного проходження невеликих за розміром статичних перешкод.

6

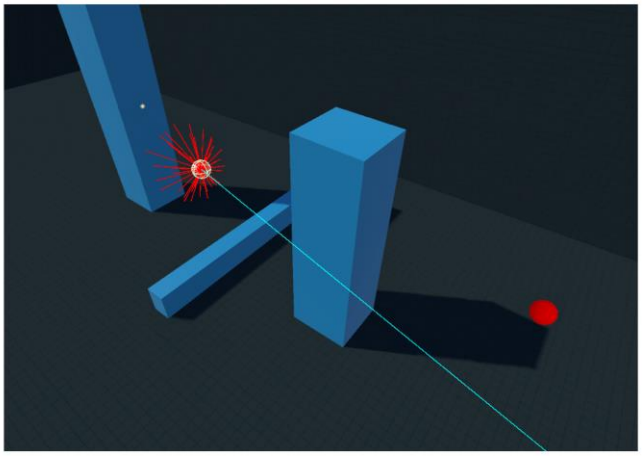
3D-VFH алгоритм у статичних умовах



Успішність алгоритму полягає у розмежованості етапів знаходження оптимального шляху та аналізу середовища у реальному часі.

7

3D-VFH алгоритм в динамічних умовах

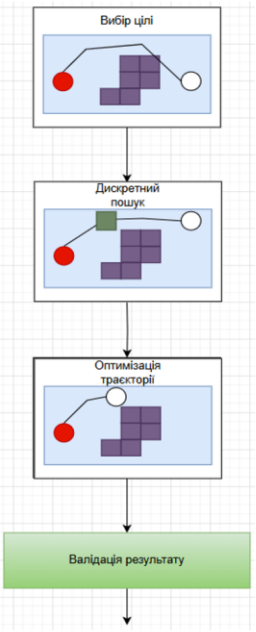


Як бачимо на відео демонстрації, класичний 3D-VFH алгоритм з тріском провалює тест з динамічними умовами.

8

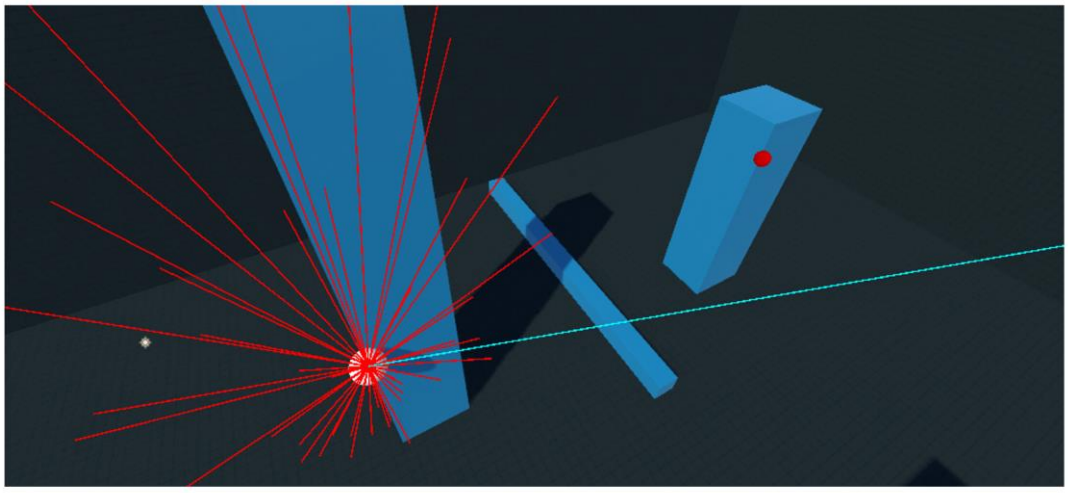
Алгоритмічне рішення для динамічних умов

Оптимізація алгоритму базується на дискретно-секторному пошуку. Балансування алгебраїчних структур вартості в реальних умовах із залученням ШІ забезпечує прозорість і ефективність навчання.



9

3D-VFH алгоритм в динамічних умовах



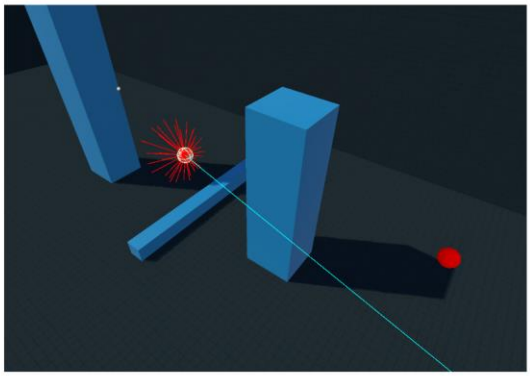
Кінцевий результат

10

Результати дослідження

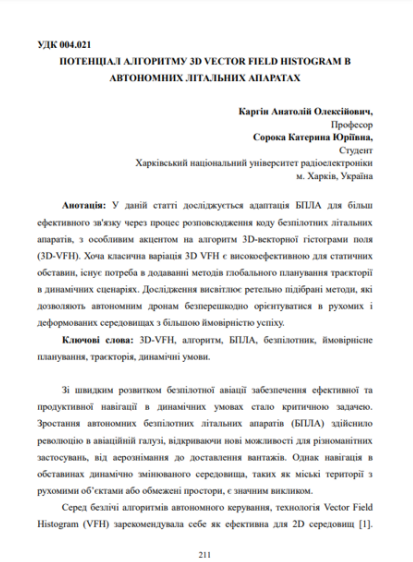
На вибірці із 54 сценаріїв:

- середній час проходження маршруту: 16.14 секунди;
- середній час простою: 2.97 секунди;
- середня оптимальність траєкторії: 112.53%;
- доля успішних спроб подолати маршрут: 88%;
- доля блокування: 6%;
- доля зіткнень із перешкодами: 10%.



11

Апробація результатів кваліфікаційної роботи



УДК 004.021

ПОТЕНЦІАЛ АЛГОРИТМУ 3D VECTOR FIELD HISTOGRAM В АВТОНОМНИХ ЛІТАЛЬНИХ АПАРАТАХ

Каргін Анатолій Олексійович,
Професор
Сорока Катерина Юріївна,
Студент
Харківський національний університет радіоелектроніки
м. Харків, Україна

Анотація: У даній статті досліджується адаптація БПЛА для більш ефективного зв'язку через процес розповсюдження коду безпілотних літальних апаратів, з особливим акцентом на алгоритм 3D-векторної гістограми поля (3D-VFH). Хоча класична варіація 3D VFH є високоефективною для статичних обставин, існує потреба в додаванні методів глобального планування траєкторії в динамічних сценаріях. Дослідження висвітлює ретельно підібрані методи, які дозволяють автономним дронам безперешкодно орієнтуватися в рухомих і деформованих середовищах з більшою ймовірністю успіху.

Ключові слова: 3D-VFH, алгоритм, БПЛА, безпілотник, ймовірнісне планування, траєкторія, динамічні умови.

Зі швидким розвитком безпілотної авіації забезпечення ефективної та продуктивної навігації в динамічних умовах стало критичною задачею. Зростання автономних безпілотних літальних апаратів (БПЛА) здійснило революцію в авіаційній галузі, відкриваючи нові можливості для різноманітних застосувань, від аерознімання до доставлення вантажів. Однак навігація в обставинах динамічно змінюваного середовища, таких як міські території з рухомими об'єктами або обмежені простори, є значним викликом.

Серед безлічі алгоритмів автономного керування, технологія Vector Field Histogram (VFH) зарекомендувала себе як ефективна для 2D середовищ [1].

211

Посилання: https://sci-conf.com.ua/wp-content/uploads/2025/01/CURRENT-TRENDS-IN-SCIENTIFIC-RESEARCH-DEVELOPMENT-16-18.01.25.pdf

12

ДОДАТОК Б
DRONE DISCRETE SEARCH

Б.1 Фрагменти коду створеного програмного застосунку

```
using System.Collections.Generic;
using UnityEngine;

public class Drone3DVFH : MonoBehaviour
{
    public float obstacleDetectionRadius = 5f;
    public int horizontalSectors = 12;
    public int verticalSectors = 6;
    public float safetyMargin = 1f;
    public float forwardSpeed = 3f;
    public Vector3 targetDestination;
    public float destinationThreshold = 1f;
    public int directionMemorySize = 10;
    public float alignmentCostValue = 0.7f;
    public float momentumCostValue = 0.5f;
    public float obstracleCostValue = 1.0f;
    public float dynamicObstacleCostValue = 2.0f;
    public float timeHorizon = 2f;

    private Rigidbody rb;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
        rb.useGravity = false;
    }

    void Update()
    {
        if (IsAtDestination())
        {
            rb.velocity = Vector3.zero;
            return;
        }

        Vector3 bestDirection = CalculateBestDirection();
        MoveInDirection(bestDirection);
    }

    private bool IsAtDestination()
    {
        return Vector3.Distance(transform.position,
```

```
targetDestination) <= destinationThreshold;
    }

    private Vector3 CalculateBestDirection()
    {
        float bestCost = float.MaxValue;
        Vector3 destinationDirection = (targetDestination -
transform.position).normalized;
        Vector3 currentDirection = rb.velocity.normalized;
        Vector3 bestDirection = destinationDirection;

        for (int h = 0; h < horizontalSectors; h++)
        {
            for (int v = 0; v < verticalSectors; v++)
            {
                Vector3 direction = GetDirectionFromSectors(h,
v);

                float obstacleCost =
CalculateCostWithNeighbors(h, v);
                float alignmentCost = Vector3.Angle(direction,
destinationDirection) / 180f;
                float momentumCost = Vector3.Angle(direction,
currentDirection) / 180f;
                float dynamicObstacleCost =
PredictDynamicObstacleCost(direction);

                float totalCost = obstacleCost *
obstracleCostValue
                    + alignmentCost * alignmentCostValue
                    + momentumCost * momentumCostValue
                    + dynamicObstacleCost *
dynamicObstacleCostValue;

                Debug.DrawLine(transform.position,
transform.position + direction * totalCost * 5, Color.red);

                if (totalCost < bestCost)
                {
                    bestCost = totalCost;
                    bestDirection = direction;
                }
            }
        }
        return bestDirection;
    }

    private Vector3 GetDirectionFromSectors(int hSector, int
vSector)
    {
        float horizontalAngle = hSector * (360f /
horizontalSectors) * Mathf.Deg2Rad;
        float verticalAngle = vSector * (180f / (verticalSectors
```

```
- 1)) * Mathf.Deg2Rad - Mathf.PI / 2;
        float x = Mathf.Cos(verticalAngle) *
Mathf.Cos(horizontalAngle);
        float y = Mathf.Sin(verticalAngle);
        float z = Mathf.Cos(verticalAngle) *
Mathf.Sin(horizontalAngle);

        return new Vector3(x, y, z).normalized;
    }

    private float CalculateCostWithNeighbors(int hSector, int
vSector)
    {
        const int neighborRange = 1; // Number of neighboring
sectors to consider
        float totalCost = 0f;
        int samples = 0;

        for (int hOffset = -neighborRange; hOffset <=
neighborRange; hOffset++)
        {
            for (int vOffset = -neighborRange; vOffset <=
neighborRange; vOffset++)
            {
                int h = (hSector + hOffset + horizontalSectors)
% horizontalSectors;
                int v = Mathf.Clamp(vSector + vOffset, 0,
verticalSectors - 1);

                Vector3 neighborDirection =
GetDirectionFromSectors(h, v);
                totalCost += CalculateCost(neighborDirection);
                samples++;
            }
        }

        return totalCost / samples;
    }

    private float CalculateCost(Vector3 direction)
    {
        RaycastHit hit;
        if (Physics.SphereCast(transform.position, safetyMargin,
direction, out hit, obstacleDetectionRadius))
        {
            return 1f / hit.distance;
        }

        return 0f;
    }

    private void MoveInDirection(Vector3 direction)
    {
```

```
        if (direction == Vector3.zero)
        {
            rb.velocity = Vector3.zero;
            return;
        }

        rb.velocity = direction.normalized * forwardSpeed;
    }

    private float PredictDynamicObstacleCost(Vector3 direction)
    {
        float dynamicCost = 0f;
        float stepSize = obstacleDetectionRadius / 5f;

        for (float t = 0; t <= timeHorizon; t += stepSize)
        {
            Vector3 futurePosition = transform.position +
direction * t;
            Collider[] hits =
Physics.OverlapSphere(futurePosition, safetyMargin);

            foreach (Collider hit in hits)
            {
                if (hit.gameObject != this.gameObject)
                {
                    float distance =
Vector3.Distance(transform.position,
hit.ClosestPoint(transform.position));
                    dynamicCost += distance > 0 ? 1f / distance
: float.MaxValue;
                }
            }
        }

        return dynamicCost / (timeHorizon / stepSize + 1);
    }

    void OnDrawGizmos()
    {
        if (!Application.isPlaying) return;
        Vector3 destinationDirection = (targetDestination -
transform.position).normalized * 5;
        //Gizmos.color = Color.cyan;
        //Gizmos.DrawRay(transform.position,
destinationDirection);

        Gizmos.color = Color.cyan;
        Gizmos.DrawRay(transform.position, rb.velocity * 10);

        Gizmos.color = Color.red;
        Gizmos.DrawSphere(targetDestination, 1);
    }
}
```

ДОДАТОК В

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕНЬ В ПУБЛІКАЦІЇ