ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки
Кафедра ЕОМ

Моделі використання технологій доповненої, віртуальної та змішаної реальності у медичній сфері

Кваліфікаційна робота
Другий (магістерський) рівень

Автор:
Лутай В.О.
студ. гр. КСМм–21–1

Керівник:
Мартовицький В.О.
доц. кафедри ЕОМ

Мета і задачі роботи

Мета: реабілітаційна терапія на основі LMC яка виконує аналіз на основі візуального судження терапевта, використовуючи лише наявний вміст та може бути застосована до пацієнтів з нервовою системою в клініці, щоб терапевт міг розпізнати будь-яку аномальну картину на основі об’єктивних даних.

Задачі:

- Вивчення медичного матеріалу та знайомство з відомими експериментами
- Вивчення технології віртуальної реальності
- Розробка архітектури програми, її розробка та тестування

Додаткова, Віртуальна та Змішана реальність. Загальні Поняття

Доповнена реальність - це середовище, в реальному часі доповнює фізичний світ, яким ми його бачимо, цифровими даними за допомогою будь-яких пристроїв - планшетів, смартфонів або інших, і програмної частини. Наприклад, Google Glass або шолом Залізної Людини.

Віртуальна реальність (VR) — це симуляція досвіду, яка використовує відстеження поз і 3D-дисплеї поблизу очей, щоб дати користувачеві відчуття занурення у віртуальний світ.

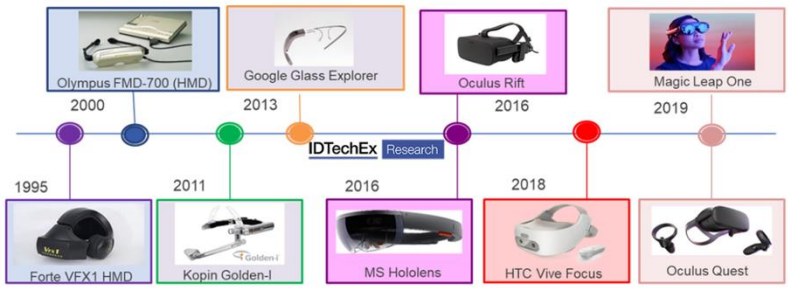
Змішаною реальністю - це злиття реальних і віртуальних світів для створення нових середовищ і візуалізації, де фізичні та цифрові об'єкти спів існують і взаємодіють в режимі реального часу.

Актуальність Роботи

Зараз, у постпандемічний період, доповнена реальність і віртуальні технології створили новий шлях, який може вплинути на майбутнє навчання та реабілітації з охорони праці.

Програми віртуальної реальності використовуються в процесах реабілітації людей похилого віку, у яких діагностовано хворобу Альцгеймера та інш.

Еволюція VR-пристоїв



До 2016 року було щонайменше 230 компаній, які розробляли продукти, пов'язані з VR. Amazon, Apple, Facebook, Google, Microsoft, Sony і Samsung мали спеціальні групи AR і VR. Однак тактильні інтерфейси не були добре розроблені, і більшість апаратних пакетів включали телефони з кнопками для сенсорної інтерактивності. Візуально дисплеї все ще мали досить низьку роздільну здатність і частоту кадрів, щоб зображення все ще можна було ідентифікувати як віртуальні.

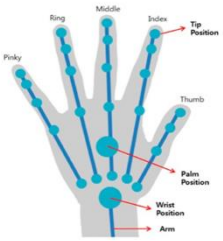
Призначення віртуальної реальності

- Мобільні ігри
- Настільні ігри
- Реклама
- Оборона
- Медицина
- Інформаційна підтримка

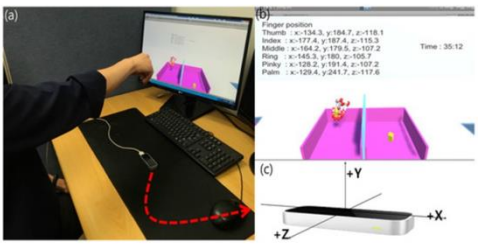
AR та VR у медичній сфері

Терапія віртуальної реальності також використовувалася, щоб допомогти пацієнтам з інсультом відновити контроль над м'язами, для лікування інших розладів, таких як дисморфія тіла, і для покращення соціальних навичок у тих, у кого діагностовано аутизм.

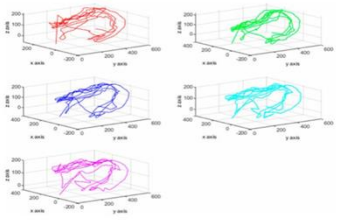
Розробка програми



Діапазон розпізнавання кистьового суглоба LMC. LMC розпізнає верхню кінцівку в режимі реального часу та дає змогу дізнатися модель руху користувача.

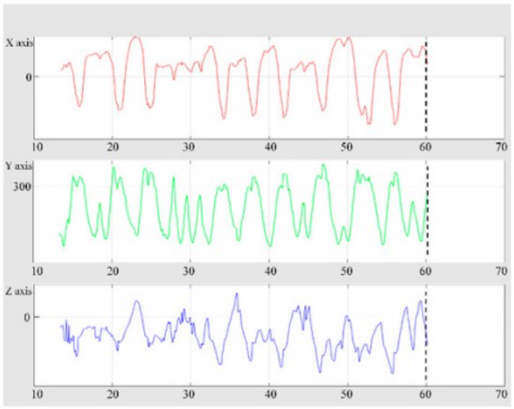


Відстеження координат долоні та пальців через тривимірний графік



Система ВВТ реалізована на C# через Unity на основі LMC. Коли система запускається, користувач може ефективно виконувати ВВТ системи за допомогою значень часу та координат

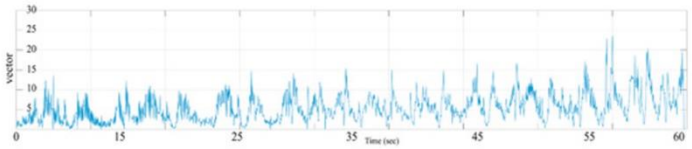
Результати



Можливість розтягувати руку користувача і аналізувати рух кожної осі з часом. Вісь X відповідно до часу - це переміщення в ширину, а вісь Y - це поздовжнє переміщення. Крім того, вісь Z означає глибину руху.

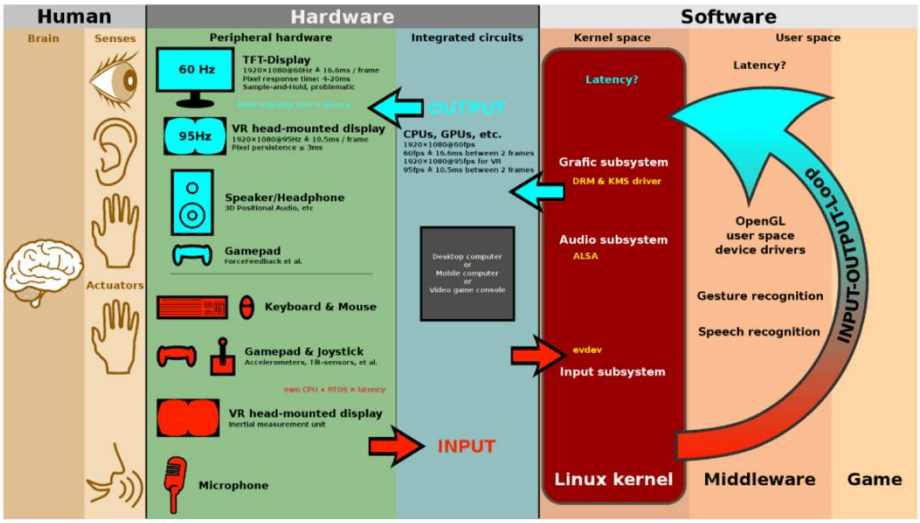
Алгоритм, розроблений у цьому дослідженні, дозволяє вимірювати розтягування (тремтіння), що не можна було б зробити у звичайній ВВТ. Також можна проаналізувати об'єктивний тремор рук, церебральний параліч, атаксію, хворобу Паркінсона.

Приклад роботи програми



Аналіз відстеження зміни вектора руху користувача як повторювану дію. У міру зміни часу можна аналізувати ділянку, де змінюється вектор. Також можна покращити функціональну здатність переміщати коробку до протилежної коробки без помилок, стимулюючи пропріорецептори через повторювані дії користувача.

Схема взаємодії людини та VR



ДОДАТОК Б

Лістинг класів

Б.1 Основні класи

```
using UnityEngine;
using UnityEngine.Rendering;
using UnityEngine.UI;
using TMPro;
using System.Collections.Generic;

//attach this script to your camera object
public class CreateStereoCubemaps : MonoBehaviour {
    public GameObject FrameRateSlider;
    public GameObject LengthSlider;
    public GameObject CamObject;
    public RenderTexture cubemapLeft;
    public RenderTexture cubemapRight;
    public RenderTexture equirect;
    public RenderTexture vidCubeLeft;
    public RenderTexture vidCubeRight;
    public RenderTexture vidEquirect;
    public bool renderStereo = true; //8k 4k
    public bool preview = true;
    public float stereoSeparation = 0.064f;
    public GameObject Menu;
    private bool recording = false;
    private int length = 0;
    private int frameRate = 0;
    private int counter = 0;
    private List<byte[]> vidBuffer = new List<byte[]>();
    private float _timer;
    private float _hudRefreshRate = 1f;

    void Start() {

System.IO.Directory.CreateDirectory(Application.persistentDataPa
th + "/vidImages/");
    }

    void LateUpdate() {

        if (!recording) {
            return;
        }

        if (counter == length * frameRate) {
```

```
            saveImages();
        }

        if (Time.unscaledTime > _timer) {
            counter++;
            _timer = Time.unscaledTime + (_hudRefreshRate /
frameRate);
            if (recording) {
                saveFrame();
            } else {
                UpdateTexture();
            }
        }
    }
    public void ToggleRecord() {
        recording = !recording;
        UpdateValues();
    }

    public void UpdateValues() {
        length =
int.Parse(LengthSlider.GetComponent<TextMeshPro>().text);
        frameRate =
int.Parse(FrameRateSlider.GetComponent<TextMeshPro>().text);
        Debug.Log($"{length} {frameRate}");
    }

    public void TakePhoto() {
        Camera cam = CamObject.GetComponent<Camera>();

        if (cam == null) {
            cam = GetComponentInParent<Camera>();
        }

        if (cam == null) {
            Debug.Log("stereo 360 capture node has no camera or
parent camera");
        }

        if (renderStereo) {
            cam.stereoSeparation = stereoSeparation;
            cam.RenderToCubemap(cubemapLeft, 63,
Camera.MonoOrStereoscopicEye.Left);
            cam.RenderToCubemap(cubemapRight, 63,
Camera.MonoOrStereoscopicEye.Right);
        } else {
            cam.RenderToCubemap(cubemapLeft, 63,
Camera.MonoOrStereoscopicEye.Mono);
        }

        //optional: convert cubemaps to equirect

        if (equirect == null) {
```

```
            return;
        }

        if (renderStereo) {
            cubemapLeft.ConvertToEquirect(equirect,
Camera.MonoOrStereoscopicEye.Left);
            cubemapRight.ConvertToEquirect(equirect,
Camera.MonoOrStereoscopicEye.Right);
        } else {
            cubemapLeft.ConvertToEquirect(equirect,
Camera.MonoOrStereoscopicEye.Mono);
        }

        // Creates buffer
        Texture2D tempTexture = new Texture2D(equirect.width,
equirect.height);

        // Copies EquirectTexture into the tempTexture
        RenderTexture currentActiveRT = RenderTexture.active;
        RenderTexture.active = equirect;
        tempTexture.ReadPixels(new Rect(0, 0, equirect.width,
equirect.height), 0, 0);

        // Exports to a PNG
        var bytes = tempTexture.EncodeToPNG();
        var bytes2 = tempTexture.EncodeToJPG();


System.IO.File.WriteAllBytes(Application.persistentDataPath +
"/test.jpg", bytes2);

        // Restores the active render texture
        RenderTexture.active = currentActiveRT;
    }

    public void UpdateTexture() {
        Camera cam = CamObject.GetComponent<Camera>();

        if (cam == null) {
            cam = GetComponentInParent<Camera>();
        }

        if (cam == null) {
            Debug.Log("stereo 360 capture node has no camera or
parent camera");
        }

        if (renderStereo) {
            cam.stereoSeparation = stereoSeparation;
            cam.RenderToCubemap(vidCubeLeft, 63,
Camera.MonoOrStereoscopicEye.Left);
            cam.RenderToCubemap(vidCubeRight, 63,
Camera.MonoOrStereoscopicEye.Right);
```

```
        } else {
            cam.RenderToCubemap(cubemapLeft, 63,
Camera.MonoOrStereoscopicEye.Mono);
        }

        //optional: convert cubemaps to equirect

        if (vidEquirect == null) {
            return;
        }

        if (renderStereo) {
            vidCubeLeft.ConvertToEquirect(vidEquirect,
Camera.MonoOrStereoscopicEye.Left);
            vidCubeRight.ConvertToEquirect(vidEquirect,
Camera.MonoOrStereoscopicEye.Right);
        } else {
            cubemapLeft.ConvertToEquirect(vidEquirect,
Camera.MonoOrStereoscopicEye.Mono);
        }
    }

    public void StopRecording(){
        recording = false;
    }

    private void saveImages() {
        recording = false;
        counter = 0;
        for (int i = 0; i < vidBuffer.Count; i++)
        {

System.IO.File.WriteAllBytes(Application.persistentDataPath +
"/vidImages/frame" + i + ".jpg", vidBuffer[i]);
        }
        vidBuffer.Clear();

    }

    private void saveFrame() {
        UpdateTexture();
        // Creates buffer
        Texture2D tempTexture = new Texture2D(vidEquirect.width,
vidEquirect.height);

        // Copies EquirectTexture into the tempTexture
        RenderTexture currentActiveRT = RenderTexture.active;
        RenderTexture.active = vidEquirect;
        tempTexture.ReadPixels(new Rect(0, 0, vidEquirect.width,
vidEquirect.height), 0, 0);

        // Exports to a PNG
        var bytes = tempTexture.EncodeToPNG();
```

```
        var bytes2 = tempTexture.EncodeToJPG();
        vidBuffer.Add(bytes2);

//System.IO.File.WriteAllBytes(Application.persistentDataPath +
"/test.jpg", bytes2);

        // Restores the active render texture
        RenderTexture.active = currentActiveRT;
    }

}
```