UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Room detector

propusă de

Broască Ioan Iulian

Sesiunea: *iulie, 2020*

Coordonator științific

Drd. Colab. Florin Olariu

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

Room detector

Broască Ioan Iulian

Sesiunea**:** *iulie, 2020*

Coordonator științific

Drd. Colab. Florin Olariu

Cuprins

Introducere

Descrierea aplicației

Aplicația *Room Detector* este o aplicație concepută pentru detectarea spațiului. Cu ajutorul acestei aplicații, utilizatorii vor putea să stocheze și să detecteze diferite spații (camere). Conceptul de spațiu în această aplicație, se referă la o cameră care poate fi identificată după un nume introdus de utilizator. Utilizatorul poate folosi această aplicație accesând unul din cele două module: primul modul este cel de încărcare a spațiilor ce se doresc a fi înregistrate pentru a fi recunoscute mai târziu, iar cel de-al doilea modul se va ocupa de detectarea spațiilor.

Pentru a încărca informații despre o încăpere utilizatorul va selecta din meniu opțiunea *Upload Images*. Această acțiune va deschide un nou ecran care va conține lista camerelor existente și opțiunea de a adăuga o cameră nouă (care va fi identificată după nume). După adăugarea unui nou nume de cameră, utilizatorul trebuie să selecteze o cameră și apoi să apese pe butonul care simbolizează o cameră de fotografiat. Această acțiune va deschide camera de fotografiat a utilizatorului și va da posibilitatea de a trimite cadre din diferite unghiuri pentru camera selectată. Aceste cadre sunt utilizate mai apoi pentru recunoașterea spațiului respectiv.

Cel de-al doilea modul, *Detects space*, oferă utilizatorului posibilitatea de a scana spații pentru a afla numele spațiului, așa cum a fost configurat în pasul anterior. Selectarea acestei opțiuni va deschide camera telefonului și va începe detectarea spațiului. Atunci când un spațiu este detectat numele camerei va apărea în partea de jos a ecranului. De asemenea, dacă în timpul detectării utilizatorul merge în altă cameră, aplicația va detecta schimbarea spațiului și îi va afișa numele camerei curente.

Motivație

În ultimii ani, evoluția tehnologiei a fost amplă și rapidă și oamenii încearcă să atribuie majoritatea sarcinilor unor mașinării. În acest sens detectarea spațiilor este o unealtă foarte puternică și folositoare. Această aplicație își propune să pună bazele unui mecanism de detectare flexibil, care poate fi adaptat mai multor soluții. Întâlnim detectarea spațiului atât în depozitele marilor producători, cât și în diferite electrocasnice care ne fac viața mult mai ușoară. În acest sens acest algoritm de detectarea spațiilor poate fi folosit de micii roboței de la Amazon care se ocupă cu organizarea coletelor în depozite dar și micul și minunatul aspirator care este capabil să facă curat în toată casa pentru noi.

Contribuții

Aplicația *Room detector* este o aplicație care oferă utilizatorului posibilitatea de a configura un spațiu ca mai apoi să poată fi recunoscut de aplicație, într-un timp foarte scurt și cu minim efort. Această aplicație a început ca un proiect de cercetare, trecând astfel prin descoperirea mai multor metode de detectare a spațiului. Majoritatea metodelor de detectare constau în pregătirea datelor înainte, iar utilizarea aplicației nu putea fi realizată imediat după configurare. Utilizatorul își pregătește datele, le procesează și după o vreme acestea vor putea fi folosite de aplicație. Practic, pentru pregătirea datelor utilizatorul trebuie sa meargă într-o cameră și să realizeze un număr mare de fotografii. După acest pas, fotografiile treceau printr-un proces de pregătire care, de cele mai multe ori, dura foarte mult. Acest lucru se datorează faptului că fotografiile treceau printr-un proces de antrenare si apoi la final, modelul antrenat putea fi utilizat. Nici în acel moment utilizatorul nu putea fi sigur că modelul poate da randament 100% iar din această cauză modelul ar fi trebuit să fie testat în prealabil pentru a vedea dacă modelul trebuie supus unui nou antrenament.

După o perioadă îndelungată de cercetare și testare, am descoperit librăria OpenCV, pe care am folosit-o în dezvoltarea acestei aplicații. Cu ajutorul librăriei OpenCV am reușit să găsesc cea mai bună și eficientă abordare, utilizând algoritmul *Oriented Fast and Rotated Brief (ORB).* Acest algoritm m-a ajutat să extrag din poze doar ce este esențial și să trec peste procesul de antrenare care este folosit în majoritatea metodelor de detectare. Practic, în momentul în care utilizatorul dorește să recunoască un spațiu, poate să folosească foarte ușor partea de admin care se numește *Upload images*, trimițând astfel imagini cu spațiul care îl dorește. După ce a terminat de configurat, poate să deschidă partea de utilizator care se numește *Detects space* și să înceapă scanarea și detectarea spațiului. În momentul în care utilizatorul deschide această scenă, pe server sunt pregătite datele instant, fără a fi nevoie să treacă printr-un proces de antrenare, ci doar printr-un proces de extragere a punctelor de pe imagini. Ulterior aceste puncte sunt comparate cu cadrele care vin în timp real de la aplicație și returnează numele camerei cu potrivirea punctelor cea mai bună. Spre deosebire de alte metode de detectare, aplicația *Room detector* oferă utilizatorului posibilitatea de configurare a spațiului într-un timp foarte scurt și de asemenea posibilitatea de a folosi imediat modulul de detectare care are o acuratețe foarte mare de detectare.

Aplicația *Room detector* este o aplicație de bază care este capabilă să se adapteze oricând la noi module cât și la alte tipuri de detectări, astfel încât în urma cercetărilor și implementării actuale, aplicația se poate adapta și poate fi folosită pentru rezolvarea mai multor

probleme. Una dintre ele fiind chiar ajutorarea persoanelor cu dezabilități de vedere, înlocuind în aplicație afișarea textului cu un mesaj auditiv.

Capitole

Aspecte tehnice

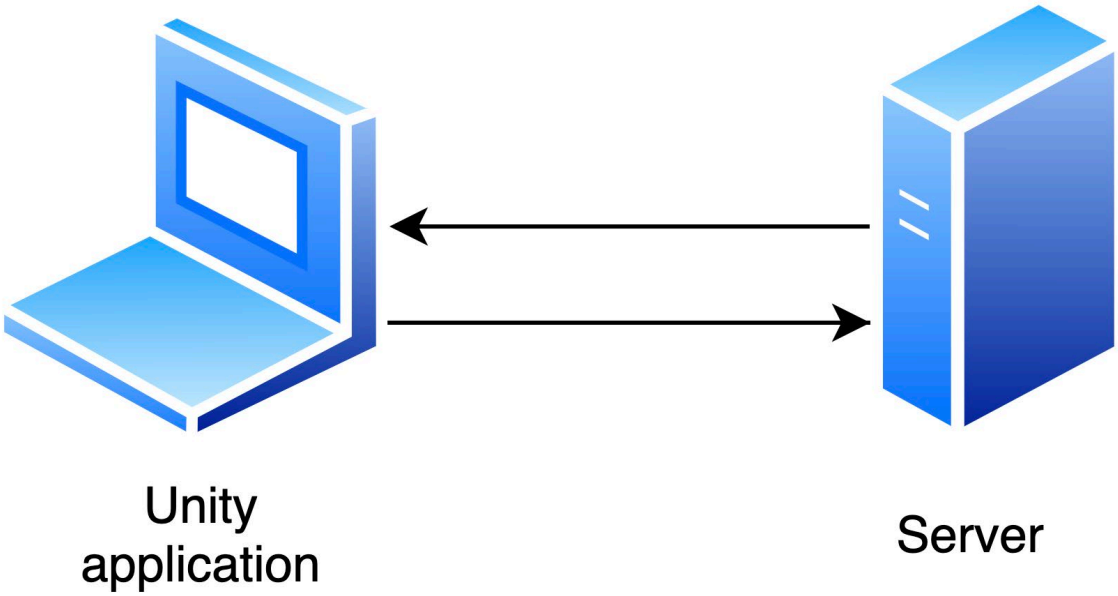

Figura 1 – Structura aplicației

Aplicația *Room Detector* este o aplicație creată în engine-ul Unity. Această aplicație are o comunicare continuă cu un server în care trimite și primește date: aplicația Unity trimite ca date imagini care sunt procesate sau salvate pe server.

Platforme

Unity Engine

Unity Engine este un software care oferă creatorilor de joc setul de funcții necesare pentru a construi jocuri rapid și eficient. Un motor de joc este un cadru pentru dezvoltarea jocurilor care susțin și reunesc mai multe domenii de bază. Poate importa elemente 2D cât și elemente 3D din alte software-uri ca Maya, Photoshop, etc.. Toate aceste elemente adăugate într-o scenă, având și efecte luminoase și audio, animații și fizica pot rezulta un joc.

Tehnologii

AR Foundation

AR Foundation este un framework care ne permite să lucrăm cu platforme de realitate augmentată cu ajutorul engine-ului Unity. Acest framework reprezintă o interfață ce este construită peste framework-urile ARKit XR sau ARCore XR, astfel încât poate fi folosit în momentul construirii unei aplicații, pentru a realiza o aplicație comună atât pentru dispozitive cu iOS cât și pentru dispozitive cu Android.

AR Foundation APIs

AR Foundation folosește un set de metode din clasa de bază MonoBehaviour și are un set de API-uri pentru anumite dispozitive care acceptă următoarele concepte:

- World tracking
    - Urmărirea poziției și orientării dispozitivului în spațiul fizic.
- Plane detection
    - Detectarea suprafețelor orizontale și verticale.
- Point clouds
    - Seturi de date ce reprezintă puncte caracteristice.
- Anchor
    - Poziția și orientarea pe care dispozitivul o recunoaște și o urmărește.
- Light estimation
    - Estimări pentru nuanță de culoare medie pentru detectarea luminozității în spațiul fizic.
- Environment probe
    - Tool pentru generarea unei hărți care reprezintă o anumită zonă în mediul fizic.
- Face tracking
    - Detectarea și urmărirea fețelor umane.
- Image tracking
    - Detectarea și urmărirea imaginilor 2D.

AR Foundation Subsystems

AR Foundation este construit pe subsystems. Un subsystem este o interfață pentru afișarea diferitelor tipuri de informații. Acestea pot fi gasite în namespace-ul UnityEngine.XR.ARSubsystems. Fiecare subsystem gestionează funcționalități specifice pentru fiecare furnizor în parte. Furnizorii sunt ARCore XR și ARKit XR. Fiecare furnizor individual stabilește modul de implementare pentru fiecare subsystem a lui. În general ele înfășoară SDK-uri nativ al platformei respective (ARKit pentru iOS și ARCore pentru Android).

Instalare AR Foundation

Pentru instalarea acestui pachet am folosit Package Manager din Unity.

Flask

Flask este un WSGI (Web Server Gateway Interface) web application framework, fiind unul dintre cele mai rapide și ușoare framework-uri, cu posibilitatea de adaptare la aplicații complexe. Flask oferă instrumente, biblioteci și tehnologii care permit construirea unei aplicații web. Flask face parte din categoriile micro-framework.

Despre WSGI (Web Server Gateway Interface)

WSGI sau Web Server Gateway Interface este un protocol care descrie modul în care un server web comunică cu aplicațiile web și modul în care aplicațiile web pot fi înlănțuite împreună pentru a procesa o cerere. WSGI este un standard Python.

Despre Micro-framework

Micro-framework-urile este un termen utilizat pentru a face referire la Web Application Frameworks minimaliste. Un micro-framework facilitează primirea unei solicitări HTTP și dirijarea cererii HTTP către controlul adecvat, expedierea controlerului și returnarea unui răspuns HTTP. Micro-framework-urile sunt deseori concepute special pentru construirea API-urilor.

Despre APIs

API sau Application Programming Interface a fost proiectat pentru utilizarea sau manipularea unui program prin internet, spre deosebire de o interfață concepută pentru a fi utilizată de om. Caracteristici:

- Setul de date poate fi mare, făcând descărcarea prin FTP
- Utilizatorii accesează datele în timp real
- Datele pot fi actualizate frecvent
- Utilizatorii au nevoie de acces la date simultan
- Utilizatorii au capabilitatea să efectueze și alte acțiuni, cum ar fi contribuirea, actualizarea sau ștergerea datelor.

Open-CV

Este o bibliotecă software de computer vision și software de învățare automată. Open-CV a fost construit pentru a furniza o infrastructură comună pentru aplicațiile de viziune computerizată. Biblioteca are peste 2500 de algoritmi optimizați, care include un set de algoritmi pentru computer vision dar și de învățare automată. Acești algoritmi pot fi folosiți pentru detectarea și recunoașterea fețelor, identificarea obiectelor, clasificarea acțiunilor umane în videoclipuri, urmărirea mișcărilor camerei, urmărirea obiectelor în mișcare, extragerea de modele 3D, producerea unor point clounds 3D din camerele stereo, lipirea imaginilor pentru a produce o imagine cu rezoluție înaltă etc..

Open-CV are interfețe pentru C++, Python, Java și MATLAB și acceptă Windows, Linux, Android și Mac OS. Open-CV se bazează pe aplicațiile de viziune în timp real și profită de instrucțiunile MMX și SSE, atunci când sunt disponibile. În prezent sunt dezvoltate activ o interfață cu caracteristici complete CUDA și OpenCL. Există peste 500 de algoritmi și de aproximativ 10 ori mai multe funcții care compun sau susțin acei algoritmi. Open-CV este scris nativ în C++ și are o interfață șablonată care funcționează perfect cu containerele STL.

Modulele aplicației

Scena principală

Ecranul splash

Splash screen este un ecran introductiv conținând imaginea de fundal și logo-ul aplicației și pe parcursul a 3 secunde se realizează o trecere de la splash screen către primul screen printr-o tranziție alpha.

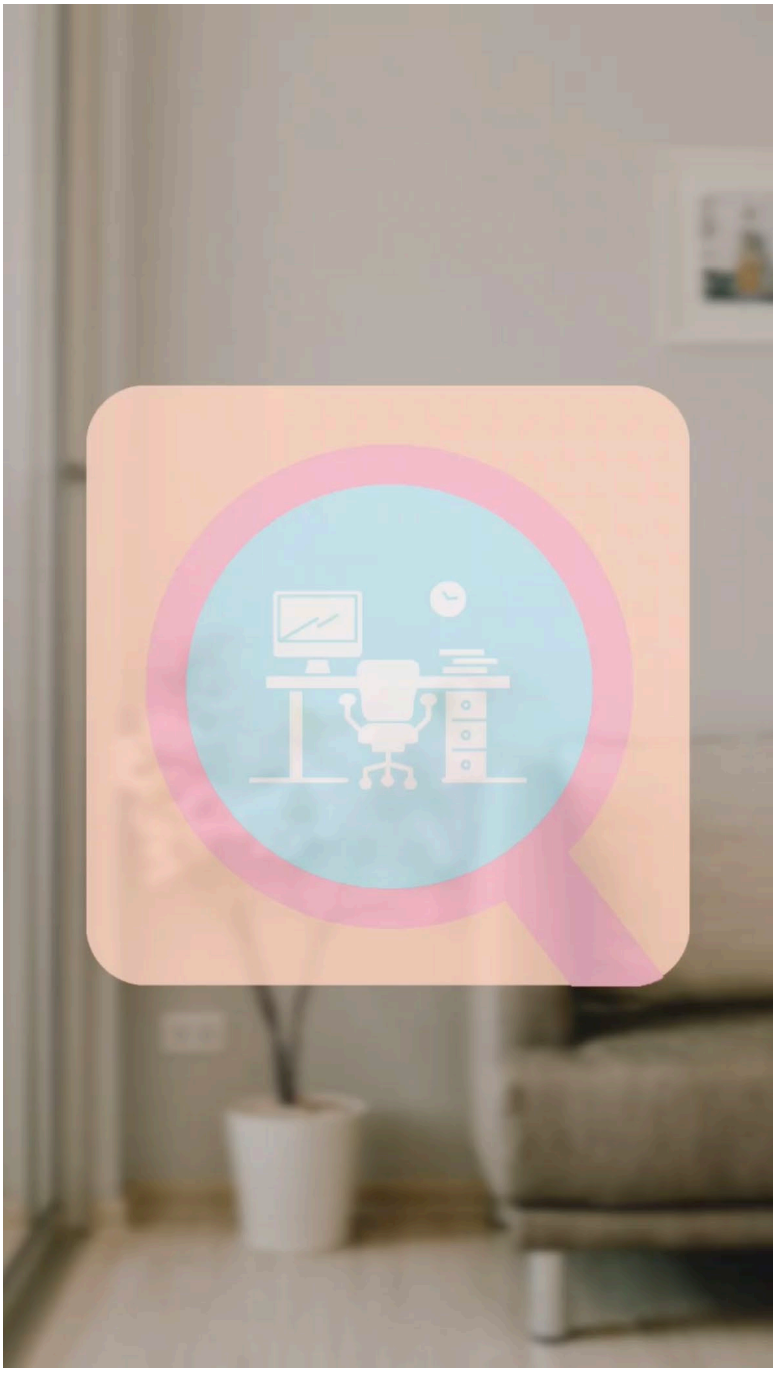

Figura 2 – Ecran introductiv cu logo-ul aplicatiei

Ecranul principal

În primul ecran utilizatorul găsește titlul aplicației împreună cu cele două butoane unde, în funcție de fiecare buton, utilizatorul va ajunge pe un alt ecran. Cele 2 butoane sunt următoarele:

1. Upload Images
2. Detects space

Figura 3 – Primul ecran al aplicației

https://unsplash.com/photos/s0ftlzyQ8No

Scena Upload Images

Configurare cameră

Imediat după apăsarea butonului *Upload Images*, utilizatorul va fi redirecționat către acest ecran, unde poate să aleagă, să adauge o cameră sau să vadă camera curentă care a fost selectată. Imediat după selectarea unei camere poate să apese pe butonul cu iconița tip cameră și va fi redirecționat către cel de-al doilea ecran.

Selectarea unei camere se realizează prin apăsarea unui element din lista de pe pagină. Pentru adăugarea unei camere trebuie să selectăm câmpul din josul paginii și să introducem numele pe care îl dorim să îl asignăm camerei. Ulterior putem apăsa pe butonul *Add room* și camera se va adăuga în listă.

Figura 4 – Configurarea camerelor

Încărcarea imaginilor pentru camere

După ce am apăsat pe butonul tip cameră utilizatorul va fi redirecționat la acest ecran, unde camera dispozitivului se va activa și va reda utilizatorului fiecare cadru. În acest ecran putem găsi numele camerei curente, un buton *Send image* pentru trimiterea imaginei către server, dar și un buton de back dacă am dori să ne întoarcem la ecranul anterior.

Imediat după ce am selectat cadrul pe care-l dorim să îl trimitem la server putem apăsa pe *Send image*.

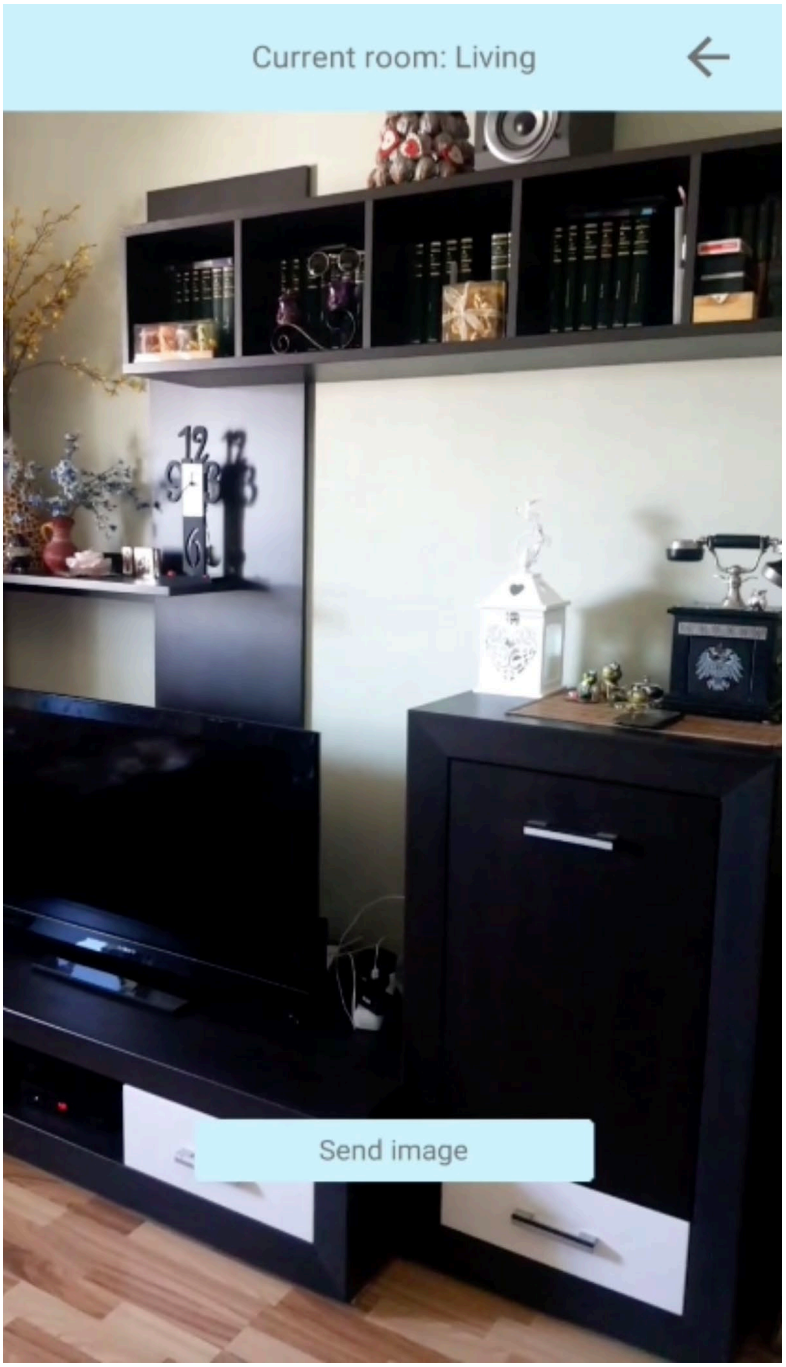

Figura 5 – Încărcarea imaginilor pe server

Pentru confirmarea trimiterii imaginei către server avem un pop-up cu un mesaj: *Room image uploaded!*

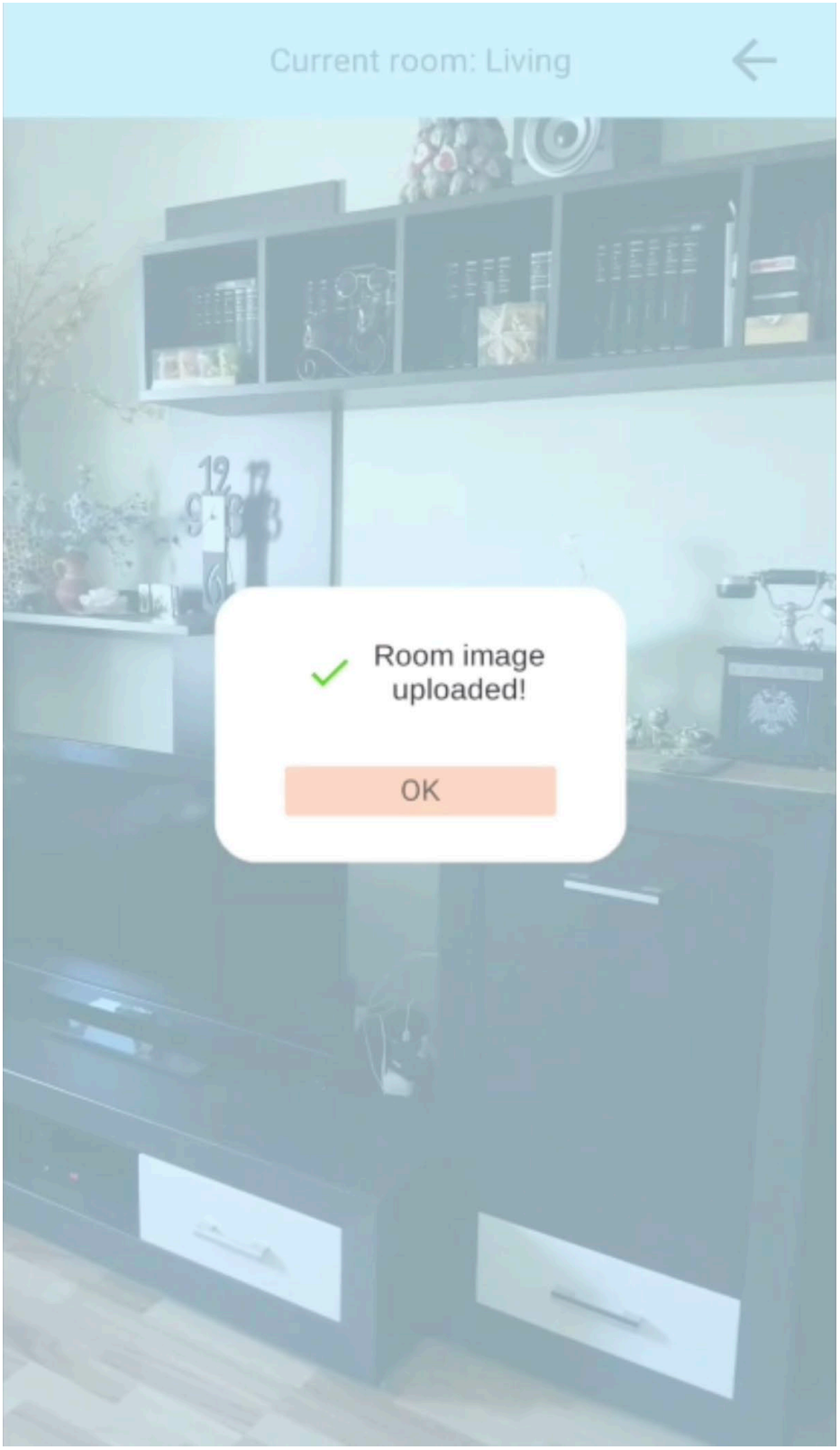

Figura 6 – Notificare încărcare cu succes

Scena *Detects space*

În acest moment, serverul pregătește datele, aplicația trimite cadrele de la cameră către server, iar serverul trimite înapoi un răspuns dacă a fost sau nu recunoscut spațiul respectiv.

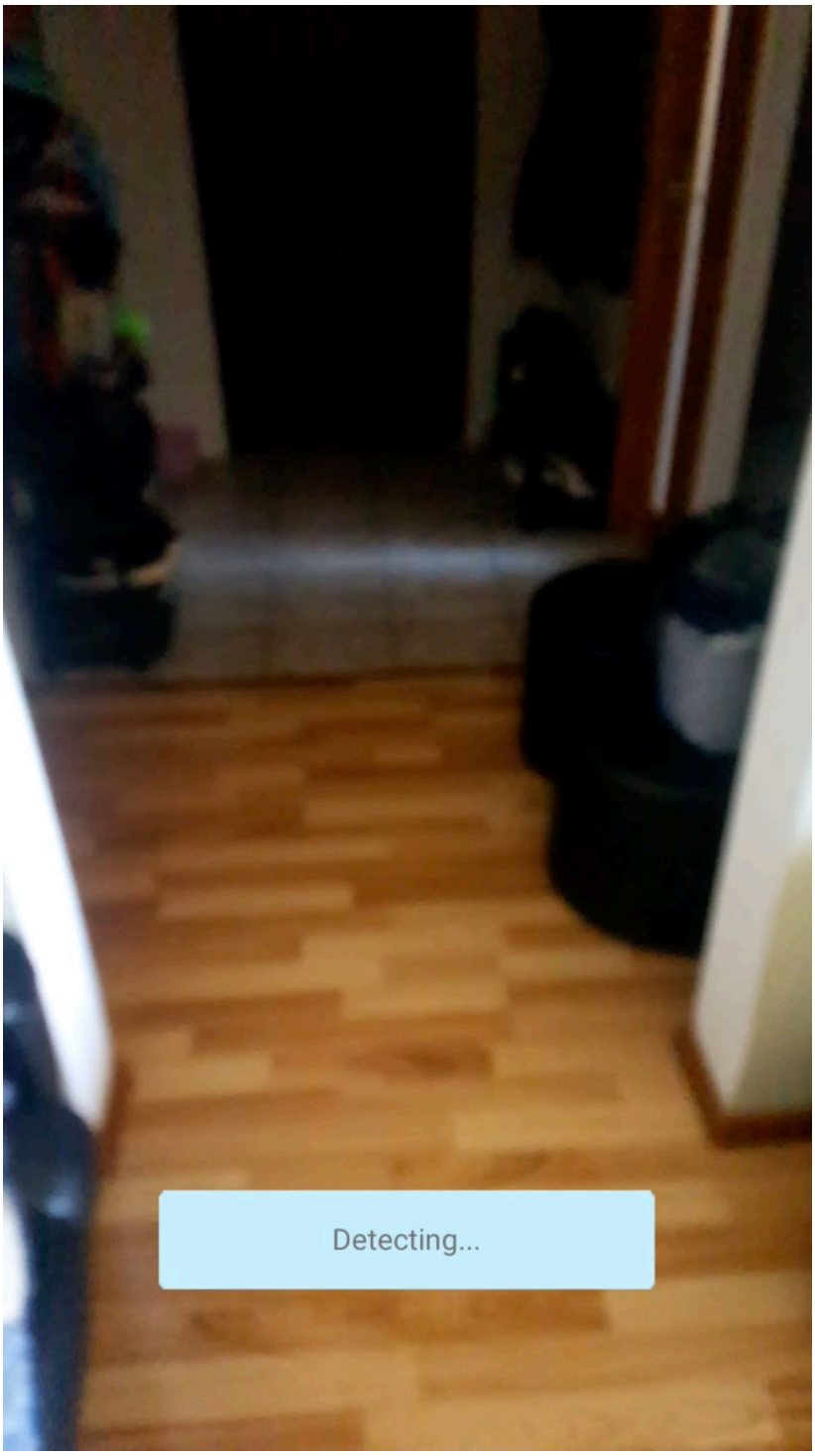

Figura 7 – Primul ecran din scena utilizatorului

În momentul în care spațiul este recunoscut, textul din josul paginii se va modifica de la *Detecting...* sau din numele camerei curente, la ultima cameră care a fost recunoscută.

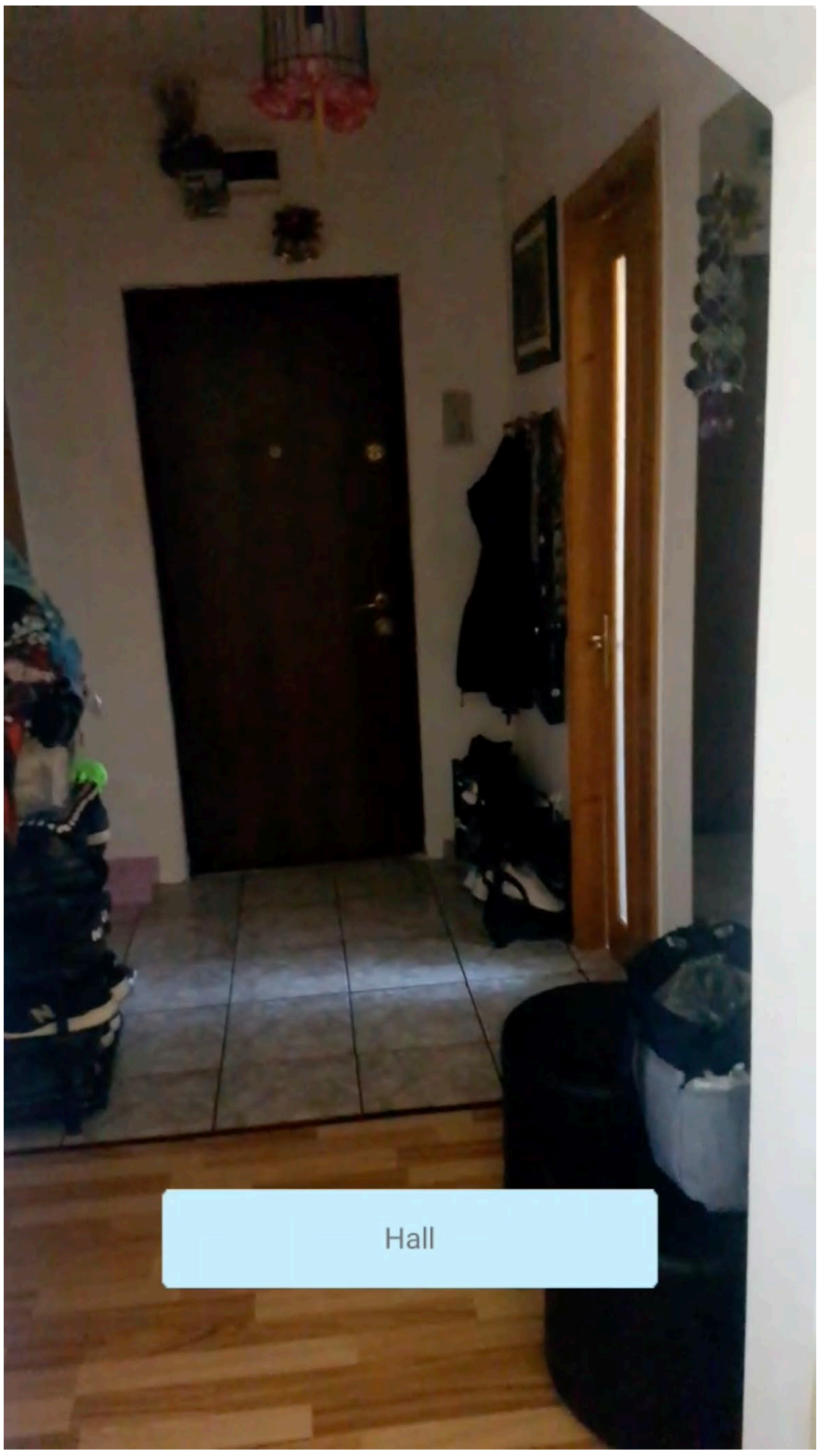

Figura 8 – Afișarea spațiului detectat

Descrierea soluției

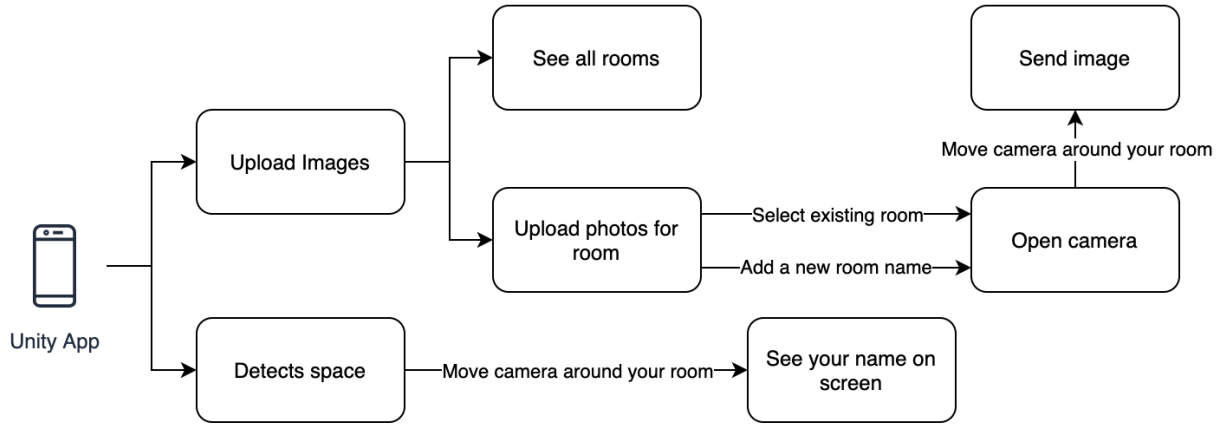

Figura 9 – Diagrama de stări

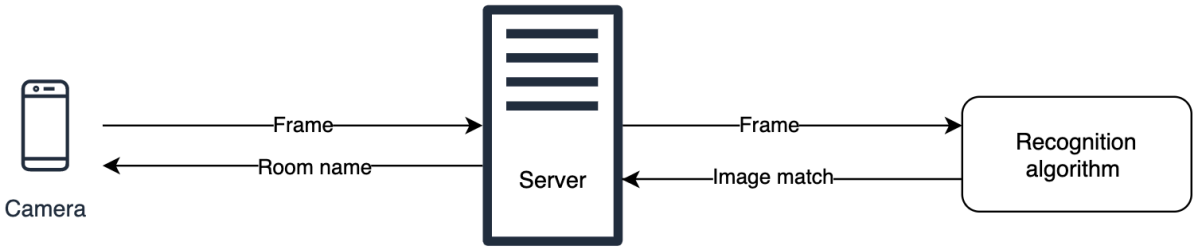

Figura 10 – Diagrama aplicației

Nucleul aplicațiilor Unity se bazează pe Entity Component System (ECS), iar fiecare termen are o anumită semnificație:

- Entity – Entitățile sau lucrurile care populează aplicația Unity (GameObject)
- Component – Datele asociate pe entitățile noastre
- System – Logica care transformă datele din starea curentă în starea următoare

În Unity, o entitate se poate numi și *GameObject*. Ele sunt obiecte fundamentale din Unity ce sunt adăugate în scenă. Ca exemplu un GameObject poate să fie un model 3D sau un obiect gol. Ele nu realizează prea multe, în schimb pot deține componente care implementează funcționalitatea reală, astfel încât ulterior să acționeze asupra GameObject-urilor. O scenă deține GameObject-urile noastre, reprezentând un nivel al aplicației.

Scripturile Unity stau la baza MonoBehaviour-ului. MonoBehaviour este o clasă de bază de la care derivă fiecare script. Această clasă oferă un număr de funcții care ne ajută să dezvoltăm o aplicație Unity. Script-urile au un rol foarte important deoarece ele le spun entităților/GameObject-urilor cum să se comporte. Unity rulează într-o buclă mare, iar script-urile atașate de GameObject-uri creează o interacțiune între ele, citind toate datele existente dintr-o scenă și realizează comportamentul pe care îl dorim.

Majoritatea script-urilor pot să dețină una sau mai multe metode din MonoBehaviour. Cele mai folosite metode sunt următoarele:

- OnEnable()
    - Metoda OnEnable() se apelează atunci când GameObject-ul se activează.
- Awake()
    - Această metodă este apelată o singură dată când este inițiat un GameObject. Dacă GameObject-ul este inactiv, atunci această metodă nu va fi apelată chiar dacă componenta este activă sau nu. În caz că GameObject-ul este activ, dar componenta este sau nu activă, metoda Awake() se va apela.
- Start()
    - Metoda Start() se apelează doar dacă GameObject-ul și Componenta sunt active și mereu se va apela înaintea metodei Update().
- FixedUpdate()
    - Această metodă se folosește atunci când în scenă avem GameObject-uri care au anumită fizică, adică cele care dețin componente Rigidbody.
- Update()
    - Metoda Update() se apelează de fiecare dată când aplicația trece la un nou frame. În această metodă se introduc secvențe de cod pe care dorim sa le executăm continuu, adică la fiecare cadru al aplicației. De exemplu: Animații, AI, etc..
- LateUpdate()
    - Această Metodă se apelează la sfârșitul cadrului. În momentul când metoda Update() termină de executat se va trece la LateUpdate(). Un exemplu de funcționalitate este atunci când o cameră ar trebui să urmarească un personaj. Poziția personajului este actualizată pe metoda Update(), iar pe LateUpdate() este executată camera care urmărește personajul, astfel încât nu se crează diferența vizuală dintre cadre.
- OnDisable()
    - Metoda OnDisable() se apelează atunci când GameObject-ul se dezactivează.

Majoritatea claselor din Unity care au anumite responsabilități importante sunt făcute singleton. Singleton este unul dintre cele mai cunoscute pattern-uri din ingineria software. Singleton este o clasă care permite crearea unei singure instanțe de sine și oferă acces simplu la acea instanță. În aplicația *Room Detector* am folosit pentru câteva clase acest pattern. Am implementat o metodă „Singleton" care verifică dacă instanța este nulă, atunci aceasta poate să fie asignată cu ea însăși. În caz că instanța nu este nulă și este diferită de ea însăși, putem să distrugem componenta respectivă pentru a ne asigura că nu există mai multe instanțe de același tip. Metoda *Singleton* este apelată din metoda *Awake*, deoarece această metodă se apelează prima dată în momentul în care deschidem aplicația. În caz că o altă clasă apelează instanța noastră înainte să fie inițializată, va intra pe proprietate și va căuta în scena Unity, GameObjectul care are numele identic cu numele clasei și astfel forțează inițializarea prin preluarea componentei. Prin această metodă simulez pattern-ul Singleton Lazy Initialization.

```
#region Singleton

private static ApiManager instance;

public static ApiManager Instance
{
    get
    {
        if (instance != null)
            return instance;
        instance = GameObject.FindGameObjectWithTag("ApiManager").GetComponent<ApiManager>();
        return instance;
    }
}

public void Singleton()
{
    if (Instance == null)
        instance = this;
    else if (instance != this)
        Destroy(this);
}

#endregion

private void Awake()
{
    Singleton();
}
```

Figura 11 – Exemplu Singleton pentru Unity

În aplicație s-au mai folosit metode Coroutine. O metodă Coroutine este o funcție care are capacitatea de a întrerupe propria execuție și de a returna controlul din același punct după ce o condiție este îndeplinită. Spre deosebire de funcția simplă care poate returna orice tip, funcțiile coroutine pot returna doar IEnumerator și trebuie utilizat cuvântul cheie *yield* înainte de cuvântul cheie *return*. Aceasta este diferența principală între funcțiile standard C# și funcțiile Coroutine. Aceste metode se folosesc pentru a împărți fluxul de lucru în mai multe cadre, fiind

o soluție mult mai optimă decât folosirea metodei *Update* care nu ne oferă nici un control asupra ei, în timp ce codul Coroutine poate fi executat atunci când îndeplinește o anumită condiție. De exemplu:

- yield return null
    - Această bucată de cod are capacitatea de a relua execuția după ce au fost apelate toate funcțiile de actualizare pe cadrul următor.
- yield return new WaitForEndOfFrame
    - Această bucată de cod așteaptă până la sfârșitul cadrului dupa ce Unity a redat fiecare cameră.
- yield return new WaitForFixedUpdate
    - Această bucată de cod așeaptă până la următorul apel de funcție.
- yield return new WaitForSeconds
    - Suspendă execuția metodei pentru durata dată de secunde folosind timpul scalat. Timpul real suspendat este egal cu timpul dat împărțit la Time.timeScale.
- yield return new WaitForSecondsRealtime
    - Această bucată de cod suspendă execuția metodei pentru durata dată de secunde folosind timpul neschimbat. În comparație cu metoda „WaitForSeconds” nu ține cont de timpul scalat din aplicație.
- yield return new WaitUntil
    - Această bucată de cod suspendă execuția metodei până când condiția delegate-ului furnizat devine true.
- yield return new WaitWhile
    - Această bucată de cod suspendă execuția metodei până când condiția delegate-ului furnizat devine false.
- yield return new WWW(URL)
    - Această bucată de cod are capacitatea de a relua execuția după ce resursa web de la URL a fost descărcată sau eșuată.

Pentru pornirea unei Coroutine se poate apela metoda StartCoroutine(), având parametrul un IEnumerator sau numele funcției. De exemplu:

- StartCoroutine(SampleCoroutine());
- StartCoroutine(„SampleCoroutine”);

Pentru oprirea unei Coroutine se poate apela metoda StopCoroutine(), având parametrul chiar instanța apelului StartCoroutine().

Unity are 3 tipuri de metode Coroutine:

- Asynchronous Coroutines
- Synchronous Coroutines
- Parallel Coroutines

Asynchronous Coroutines sunt coroutine în cadrul unei coroutine existente. Nu interacționează direct și cel mai important nu se așteaptă unul pe celălalt.

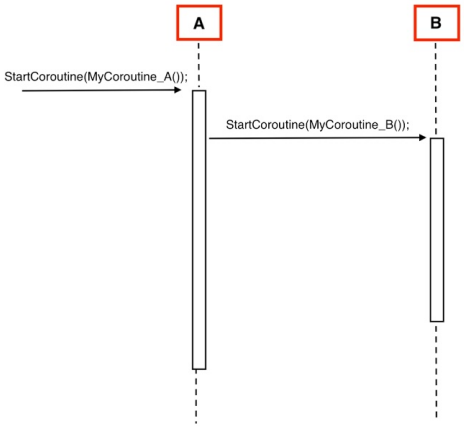

Figura 12 – Asynchronous Coroutines

Synchronous Coroutines sunt coroutine care așteaptă executarea altei coroutine, prin *yield return StartCoroutine(SampleCoroutine());*. În momentul când o coroutine pornește alta coroutina blochează executarea până când acea coroutine nu este gata.

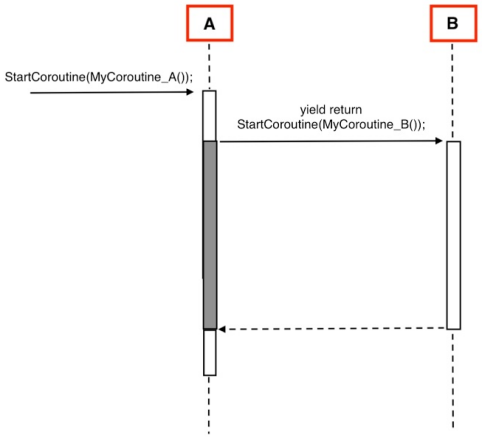

Figura 13 – Synchronous Coroutines

https://vasundharavision.com/blog/Unity/how-to-use-coroutines-in-unity

Parallel Coroutines sunt coroutine care au la finalul metodei interogarea instanței unei alte coroutine. În comparație cu Synchronous Coroutines unde interoghează direct apelul StartCoroutine().

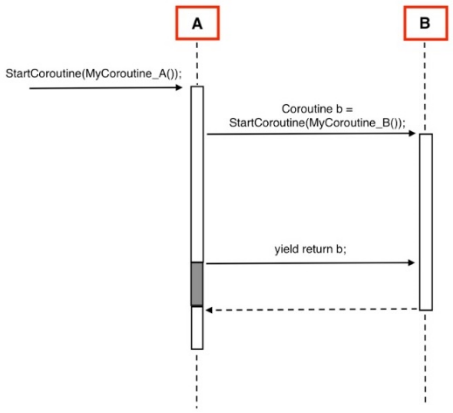

Figura 14 – Parallel Coroutines

Exemple cu manageri.

Pentru comunicarea cu server-ul am creat o clasă numită *ApiManager* având responsabilitatea de a face anumite apel-uri către end-point-urile de pe server. Această clasă este singleton, deoarece avem nevoie să o accesăm din mai multe puncte ale aplicației. În clasă găsim mai multe metode Coroutine:

- **Get**

```
public IEnumerator Get(string uri, System.Action<string> callback)
{
    var www = UnityWebRequest.Get(uri);
    yield return www.SendWebRequest();

    if (www.isNetworkError || www.isHttpError)
    {
        callback(www.error);
    }
    else
    {
        callback(www.downloadHandler.text);
    }
}
```

Figura 15 – Metoda Get

https://vasundharavision.com/blog/Unity/how-to-use-coroutines-in-unity

```
public IEnumerator Post(string uri, string data, System.Action<string> callback)
{
    var www = UnityWebRequest.Post(uri, data);
    yield return www.SendWebRequest();

    if (www.isNetworkError || www.isHttpError)
    {
        callback(www.error);
    }
    else
    {
        callback(www.downloadHandler.text);
    }
}
```

Figura 16 – Metoda Post

- **Put**

```
public IEnumerator Put(string uri, byte[] data, System.Action<string> callback)
{
    var www = UnityWebRequest.Put(uri, data);
    yield return www.SendWebRequest();

    if (www.isNetworkError || www.isHttpError)
    {
        callback(www.error);
    }
    else
    {
        callback(www.downloadHandler.text);
    }
}
```

Figura 17 – Metoda PUT

```
public IEnumerator Put(string uri, string data, System.Action<string> callback)
{
    var www = UnityWebRequest.Put(uri, data);
    yield return www.SendWebRequest();

    if (www.isNetworkError || www.isHttpError)
    {
        callback(www.error);
    }
    else
    {
        callback(www.downloadHandler.text);
    }
}
```

Figura 18 – Metoda PUT

Datele folosite în aplicație pot fi găsite in clasa statică *Constants*, conținând mai multe constante despre rute, host, port, etc..

```
public static class Constants
{
    /// <summary>
    /// AdminScene
    /// </summary>
    public const string UriGetRooms = Host + ":" + Port + RouteGetRooms;
    public const string UriSendImage = Host + ":" + Port + RouteSendImage;

    /// <summary>
    /// UserScene
    /// </summary>
    public const string UriPrepareContentServer = Host + ":" + Port + RoutePrepareContentServer;
    public const string UriDetection = Host + ":" + Port + RouteDetection;

    /// <summary>
    /// UI
    /// </summary>
    public const string MandatorySelectRoom = "Please select a room";
    public const string CurrentRoomText = "Current room: ";
    public const string MessageSuccessServer = "Success";

    private const string Host = "http://192.168.100.5";
    private const string Port = "5000";
    private const string RouteGetRooms = "/admin";
    private const string RoutePrepareContentServer = "/user";
    private const string RouteSendImage = "/send";
    private const string RouteDetection = "/detection";
}
```

Figura 19 – Clasa Constants

Scena principală

În Main Scene vom găsi prima pagină a aplicației, unde putem alege unul dintre cele 2 butoane *Upload Images* sau *Detects space*. În funcție de butonul ales vom fi trimiși la o anumită scenă.

```
public class UIManagerMainScene : MonoBehaviour
{
    public void LoadScene(int number)
    {
        SceneManager.LoadScene(number);
    }
}
```

Figura 20 – Încărcarea scenei bazat pe ID

Fiecare scenă are un id astfel încât să fie ușor identificată și încărcată. Metoda LoadScene este asignată de fiecare buton din MainScene împreună cu id-ul scenei ca parametru.

Scena admin sau Upload Images

În momentul când deschidem această scenă, clasa AdminSceneManager realizează un apel către end-point-ul Constants.UriGetRooms = „/admin” din metoda Awake() pentru a aduce lista cu numele camerelor.

```
private void Awake()
{
    Singleton();
    GetRooms();
}

public void GetRooms()
{
    StartCoroutine
    (
        ApiManager.Instance.Get
        (
            Constants.UriGetRooms,
            (callback) =>
            {
                rooms = JsonConvert.DeserializeObject<List<string>>(callback);
                UIManagerAdminScene.Instance.InstantiateRoomList(rooms);
            }
        )
    );
}
```

Figura 21 – Apel către server pentru aducerea camerelor

Când server-ul primește apelul, se pregătesc datele despre camere pe server și returnează către aplicație o listă având numele fiecăreia.

```
@app.route('/admin', methods = ['GET'])
def GetRooms():
    if request.method == 'GET':
        subfolders = [folder.name for folder in os.scandir(resourcesDirectoryPath) if folder.is_dir()]
        localJson = json.dumps(subfolders)
        return localJson
```

Figura 22 – End-point „/admin”

După primirea listei, aplicația Unity deserializează JSON-ul primit și populează lista cu camerele care sunt pe server (Figura 4). Utilizatorul poate să selecteze una dintre camere și să treacă la următoarea pagină unde este capabil să încarce poze cu camera respectivă. Pe server ajung imaginile unde sunt salvate într-un folder cu numele camerei curente. Dacă camera selectată de abia a fost creată, se va genera un folder nou. Pentru trimiterea imaginilor se apelează end-point-ul „/send", având ca date un JSON reprezentând modelul ImageData. ImageData conține numele camerei, dar si un array de byte care este imaginea noastră.

```
public void SendImage()
{
    if (currentRoom == null)
    {
        UIManagerAdminScene.Instance.SetTextOnTopBar(Constants.MandatorySelectRoom);
        return;
    }

    UIManagerAdminScene.Instance.EnableLoader();
    UIManagerAdminScene.Instance.SetActiveSendImageButton(false);

    var image = new ImageData(currentRoom, CameraManager.instance.GetJpgTexture());
    StartCoroutine
    (
            ApiManager.Instance.Put
            (
                    Constants.UriSendImage,
                    JsonConvert.SerializeObject(image),
                    (callback) =>
                    {
                        UIManagerAdminScene.Instance.EnablePopup(callback == Constants.MessageSuccessServer);
                    }
            )
    );
}
```

Figura 22 – Apel către server pentru trimiterea cadrului

În momentul în care utilizatorul apăsa pe butonul *Send Image*, se va apela metoda din figura de mai sus. În această metodă verificăm dacă a fost selectată o cameră astfel, dacă nu a fost selectată nicio cameră, atenționăm utilizatorul să se întoarcă la ecranul precedent și să selecteze una. În cazul în care o cameră a fost selectată, pregătim trimiterea cadrului și așteptăm răspuns de la server dacă a fost sau nu trimisă poza cu succes.

```
@app.route('/send', methods = ['PUT'])
def Receive():
    if request.method == 'PUT':
        data = request.get_data()
        imageData = json.loads(data.decode("utf-8"))
        WriteImage(imageData.get('roomName'), base64.b64decode(imageData.get('image')))
        return "Success"
```

Figura 23 – End-Point „/send”

```
namespace Models
{
    public class ImageData
    {
        public string roomName;
        public byte[] image;

        public ImageData(string roomName, byte[] image)
        {
            this.roomName = roomName;
            this.image = image;
        }
    }
}
```

Figura 24 – Modelul ImageData

După ce am terminat partea de configurare a camerelor, putem să ne întoarcem la primul ecran pentru a testa dacă datele/imaginile de pe server sunt de ajuns, pentru o recunoaștere ușoară și rapidă.

Scena utilizatorului sau Detects Space

În comparație cu scena anterioară, în această scenă se va folosi camera telefonului care va trimite fără oprire cadre către server. Pentru această scenă, clasa UserSceneManager realizează un apel către end-point-ul UriPrepareContentServer = „/user” în metoda Awake() pregătind datele pe server.

```
private void Awake()
{
    StartCoroutine(ApiManager.Instance.Get(Constants.UriPrepareContentServer, SetIsContentReady));
}
```

Figura 25 – Trimiterea unui apel către server pentru pregătirea datelor

Când server-ul recepționează apelul de la aplicație pregătește local într-o listă descriptorii pentru fiecare imagine și trimite un mesaj de validare dacă totul a funcționat cum trebuie.

```
@app.route('/user', methods = ['GET'])
def PrepareRooms():
    global images
    if request.method == 'GET':
        images = LoadImages()
        return "Success"
```

Figura 26 – End-point „/user”

```
private IEnumerator Start()
{
    // Wait until the content is loaded on server
    yield return new WaitUntil(()=>isContentReady == "Success");

    // Wait a second before sending the first frame
    yield return new WaitForSeconds(1);

    while (true)
    {
        var www = UnityWebRequest.Put(Constants.UriDetection, CameraManager.instance.GetJpgTexture());
        yield return www.SendWebRequest();

        if (www.isNetworkError || www.isHttpError)
        {
            Debug.Log(www.error);
        }
        else
        {
            UIManagerUserScene.Instance.SetTopText(www.downloadHandler.text);
        }
    }
}
```

Figura 27 – Pornirea unei coroutine și procesarea cadrelor

Această clasă are implementată o coroutină la Start, astfel încât la inițializarea componentei nu se va apela metoda Start ca de obicei, ci se va crea o Coroutine care se va bloca până când condiția este îndeplinită. Condiția este să primim de la server mesajul de confirmare. Dacă mesajul de confirmare a ajuns atunci putem începe trimiterea cadrelor către server pentru a începe detectarea spațiului.

Trimiterea cadrelor se realizează printr-o buclă cu ajutorul instrucțiunii *while*. În momentul trimiterii imaginii către server prin end-point-ul UriDetection = „/detection”, se așteaptă răspunsul de succes sau eroare a trimiterii cadrului, astfel încât la primirea răspunsului aplicația va pregăti și va trimite un nou cadru. Cadrele sunt preluate din clasa CameraManager prin metoda GetJpgTexture. Această clasă este preluată de la ARFoundation, fiind folosită pentru a accesa datele camerei.

```
@app.route('/detection', methods = ['PUT'])
def Detection():
    if request.method == 'PUT':
        data = request.get_data()

        array = np.fromstring(data, np.uint8)
        img = cv.imdecode(array, cv.COLOR_BGR2GRAY)

        value = FindRoom(GetDescriptors(img))
        return value
```

Figura 28 – End-point „/detection”

În momentul în care se realizează un apel către end-point-ul „/detection”, se vor prelua datele și se vor transforma într-un format de imagine. Imediat după acest proces, se va apela următoarea linie *value = FindRoom(GetDescriptors(img));*, unde sunt identificate 2 metode. Una dintre ele fiind *GetDescriptors()*, unde primește ca parametru imaginea primită din aplicație și returnează descriptorii acesteia.

```
def GetKeypointsAndDescriptors(img):
    return orb.detectAndCompute(img, None)

def GetDescriptors(img):
    keypointsCamera, descriptorsCamera = GetKeypointsAndDescriptors(img)
    return descriptorsCamera
```

Figura 29 – Preluarea descriptorilor pentru imaginea primită de la aplicație

În metoda aceasta se mai apelează o altă funcție *GetKeypointsAndDescriptors()*, primind imaginea care vine ca parametru. *GetKeypointsAndDescriptors()* returnează keypoint-urile și descriptorii pentru cadrul curent primit de la aplicație.

Cea de-a doua metodă este *FindRoom()* care primește ca parametru descriptorii imaginii curente din aplicație și parcurge astfel prin toți descriptorii fiecărei imagini de pe server returnând cea mai bună potrivire între descriptori.

```
def FilterMatches(matches, matchesThreshold):
    distances = [match.distance for match in matches]
    if(len(distances) != 0):
        sumDistances = sum(distances)
        lenDistances = len(distances)
        threshold_distance = (sumDistances / lenDistances) * matchesThreshold
    matches = [match for match in matches if match.distance < threshold_distance]
    return matches

def GetMatches(des1, des):
    try:
        matches = bf.match(des1 ,des)
    except:
        matches = []
        print("Something was wrong.")
    return FilterMatches(matches, threshould)

def FindRoom(descriptorCamera):
    match = [0,0,0,0,0]
    roomName = ""
    for img in images:
        localMatch = GetMatches(img[0], descriptorCamera)
        if(len(localMatch) > len(match)):
            match = localMatch
            roomName = img[1]
    return roomName
```

Figura 30 – Procesarea imaginilor și returnarea numele camerei

În unele metode de mai sus s-au folosit 2 variabile *orb* și *bf*. Aceste 2 variabile sunt variabilele cheie pentru recunoașterea spațiului. Prima variabilă reprezintă inițierea detectorului *ORB*, iar cea de-a doua reprezintă un obiect *BFMatcher*. Pentru mai multe detalii despre *ORB*, se pot găsi în punctul 4.

```
# Initiate ORB detector
orb = cv.ORB_create()
# create BFMatcher object
bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)
```

Figura 31 – Instanțele algoritmilor ORB și BFMatcher

Algoritmul Oriented FAST and rotated BRIEF (ORB)

ORB a fost dezvoltat în laboratoarele OpenCV de Ethan Rublee, Vincent Rabaud, Kurt Konolige și Gary R. Bradski în 2011, ca o alternativă eficientă și viabilă la algoritmi SIFT și SURF. ORB a fost conceput în principal deoarece algoritmii SIFT și SURF sunt algoritmi patentați, însă algoritmul ORB este liber de utilizat.

ORB efectuează la fel de bine ca algoritmul SIRF sarcina de detectare a caracteristicilor, fiind mai bun și rapid decât SURF de 2 ori mai mult. ORB se bazează pe bine-cunoscutul detector de puncte cheie FAST și descriptorul BRIEF. Ambele tehnici sunt atractive datorită performanței lor bune și a costurilor reduse.

Principalele contribuții ale ORB sunt următoarele:

- Adăugarea unei componențe de orientare rapidă și precisă la FAST
- Calcul eficient al caracteristicilor orientate BRIEF
- Analiza și corelarea caracteristicilor orientate BRIEF
- O metodă de învățare prin funcțiile BRIEF, ceea ce duce la o performanță mai bună

Fast (Features from Accelerated and Segments Test)

Având în vedere un pixel p dintr-o poză, se compară rapid luminozitatea pixelului p cu alți 16 pixeli, aflându-se într-un cerc mic în jurul lui p. Pixelii din cerc sunt apoi sortați în trei clase (mai deschise decât p, mai închise decât p sau similare cu p). Dacă mai mult de 8 pixeli sunt mai întunecați sau mai luminați decât p, este selectat ca punct cheie. Astfel, punctele cheie găsite de algoritmul FAST ne oferă informații despre locație.

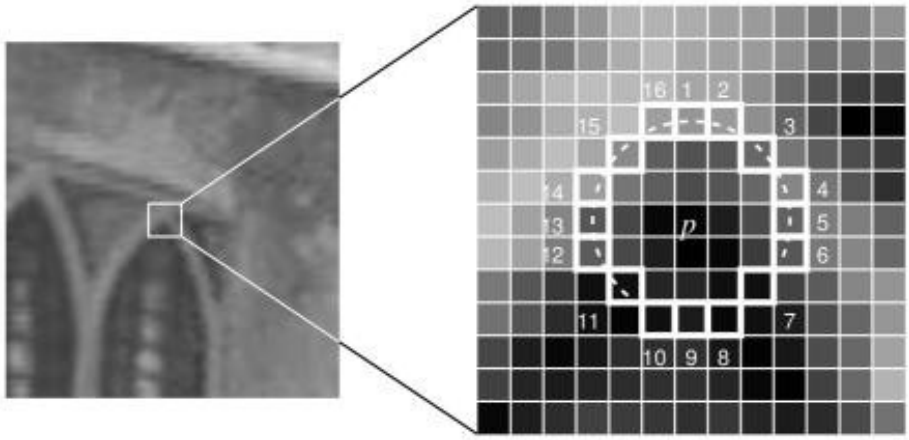

Figura 32 – Exemplu detectare punct cheie

https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf

Cu toate acestea, funcțiile FAST nu au o componentă de orientare și funcții pe mai multe niveluri. Deci, algoritmul ORB folosește o piramidă de imagine pe mai multe niveluri. O piramidă a imaginii este o reprezentare pe mai multe niveluri a unei singure imagini, care constă din secvențe de imagini, toate fiind versiuni ale imaginii la diferite rezoluții. Fiecare nivel din piramidă conține versiunea eșantionată a imaginii față de nivelul anterior. Odată ce algoritmul ORB a creat o piramidă, acesta folosește algoritmul FAST pentru a detecta punctele cheie din imagine. Prin detectarea punctelor cheie la fiecare nivel, ORB localizează în mod eficient punctele cheie la o scară diferită. În acest fel, ORB este invariabil la scară parțială.

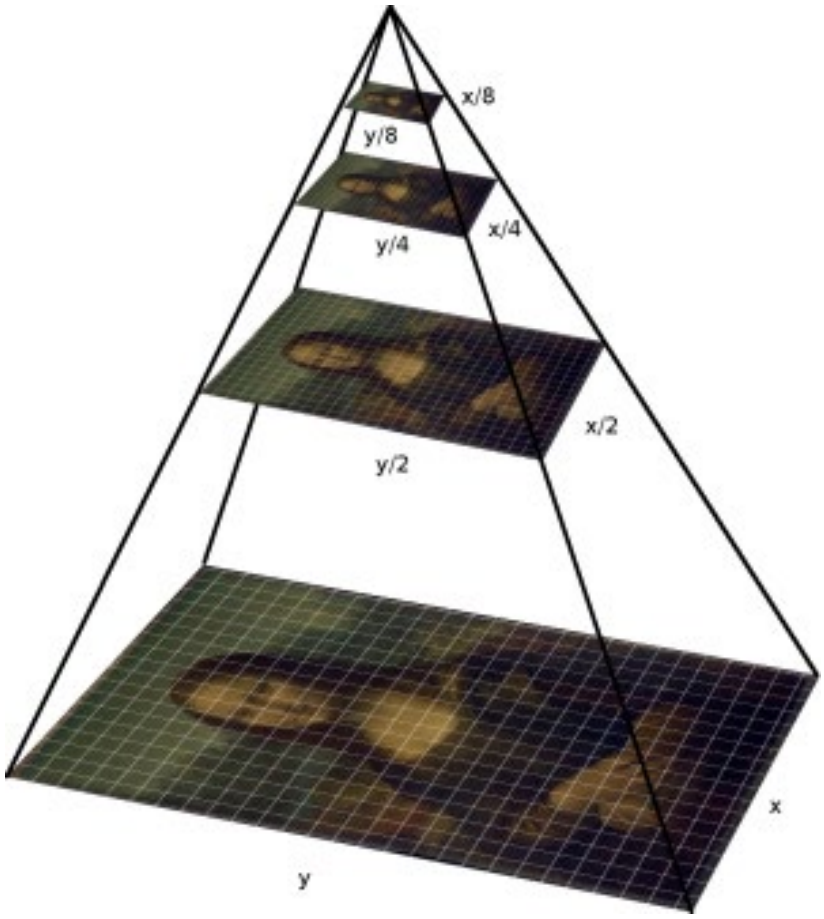

Figura 33 – Piramida imaginei

După localizarea punctelor cheie, ORB atribuie o orientare fiecărui punct, cum este orientat spre stânga sau spre dreapta, în funcție de cum se schimbă nivelurile de intensitate în jurul acelui punct. Pentru detectarea schimbării de intensitate orb folosește *centroid de intensitate*. Centroidul de intensitate presupune ca intensitatea unui colț să fie compensată de centru, și acest vector poate fi utilizat pentru a impune o orientare.

https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf

Brief (Binary robust independent elementary feature)

Brief preia toate punctele cheie găsite de algoritmul FAST și le convertește într-un vector de caractere binare, astfel încât împreună să poată reprezenta un obiect. Vectorul caracteristicilor binare este de asemenea, un descriptor de funcții binare reprezentând un vector de caracteristici care conține doar 1 și 0. Pe scurt, fiecare punct cheie este deschis de un vector de caracteristici, care este un șir de 128-521 biți.

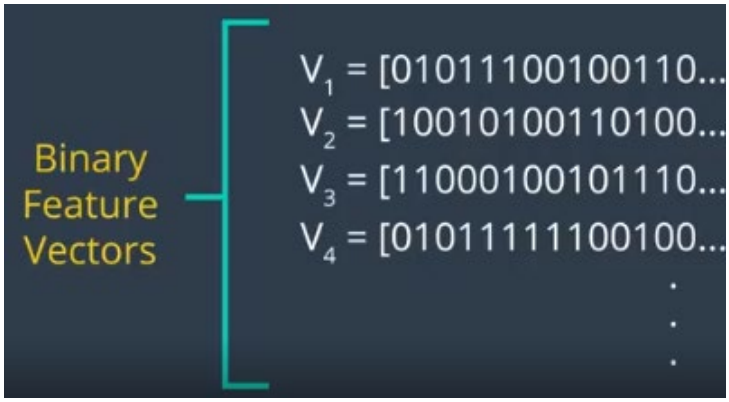

Figura 34 – Exemple de date pentru algoritmul BRIEF

Pentru a împiedica descriptorii să fie sensibili la anumite zone cu înaltă frecvență se folosește un nucleu gaussian. Se selectează o pereche de pixeli aleatorii într-o zonă definită în jurul punctului cheie. Zona definită în jurul pixelului este cunoscută sub numele de *patch*, care este un pătrat cu o lățime și o înălțime a pixelilor. Primul pixel din perechea aleatorie este extras dintr-o distribuție gaussiană în jurul punctului cheie. Cel de-al doilea pixel din perechea aleatorie este extras dintr-o distribuție gaussiană în jurul primului pixel. Acum dacă primul pixel este mai luminos decât al doilea, acesta atribuie valoarea lui 1, altfel 0. Pe scurt, se selectează o pereche întâmplătoare și se atribuie valori. Pentru un vector de 128 de biți, se repeta acest proces de 128 de ori pentru un punct cheie. Algoritmul Brief creează un vector ca acesta pentru fiecare punct cheie dintr-o imagine și nu este de asemenea invariabil la rotație, astfel încât ORB folosește rBRIEF (Rotation BRIEF). ORB încearcă să adauge această funcționalitate fără a pierde din aspectul de viteză al algoritmului BRIEF.

Figura 35 – Exemplu puncte cheie

https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf

BFMatcher (Brute force matcher)

BFMatcher este un algoritm ce compară și potrivește caracteristicile fiecărei imagini. Acesta ia descriptorii din primul set și descriptorii din al doilea set și le compară folosind un anumit calcul al distanței, iar cel mai apropiat este returnat.

Punerea în aplicare

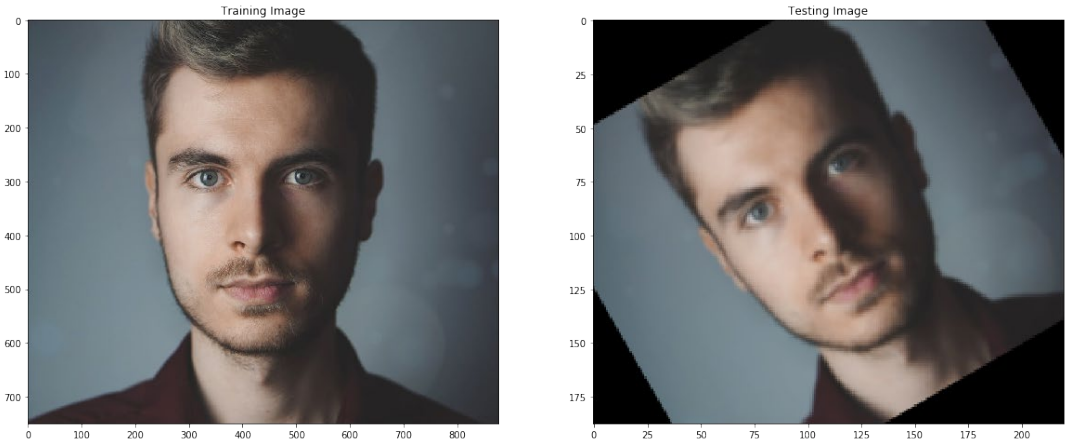

Figura 36 – Exemple de test

Primul pas este construirea unui format de imagine astfel încât să putem extrage punctele cheie și descriptorii din ea.

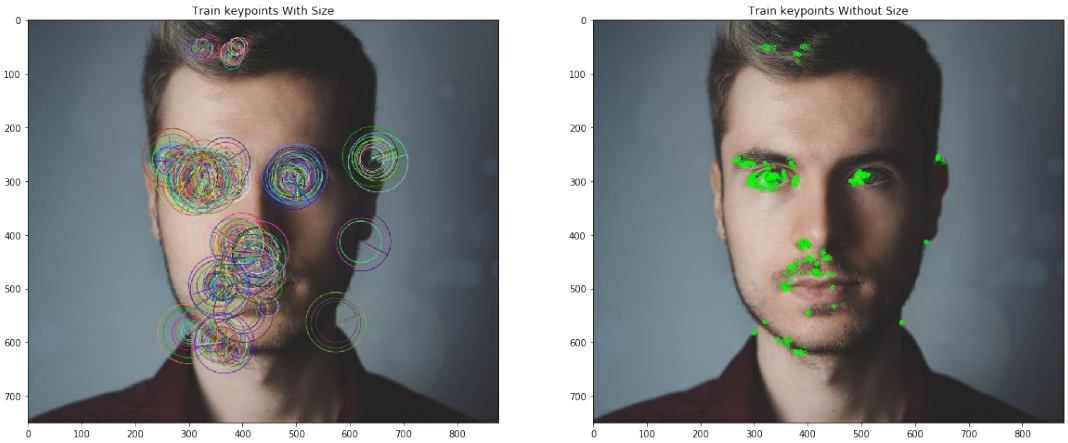

Figura 37 – Extragerea punctelor cheie

În cel de-al doilea pas este vorba de potrivirea datelor și astfel se creează o instanță BFMatcher și se compară datele, rezultatul fiind următorul.

https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf

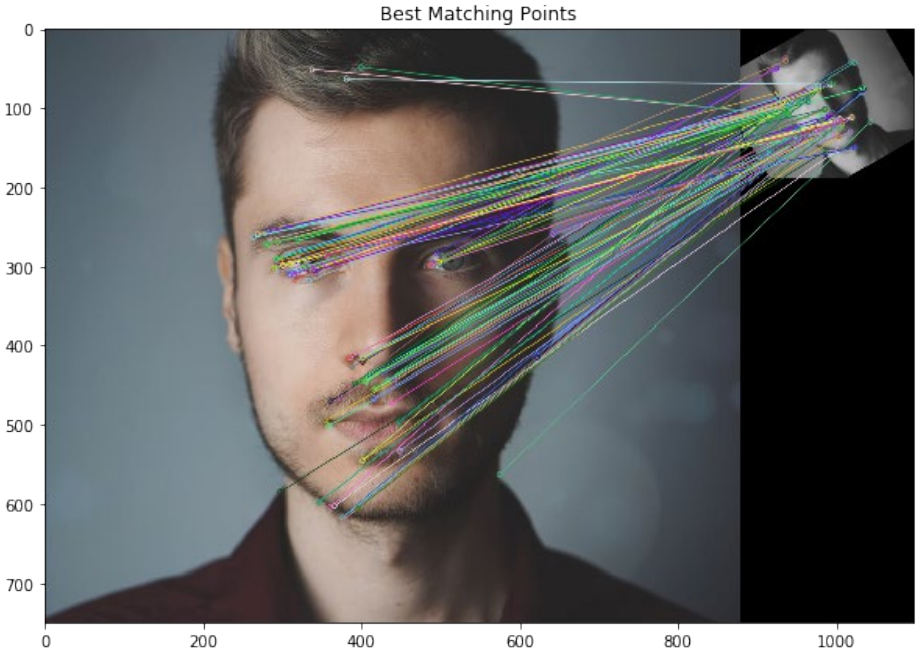

Figura 38 – Potrivirea datelor

https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf

Concluziile lucrării

În concluzie aplicația *Room detector* este o aplicație de detectare a spațiilor realizată cu tehnologii precum procesarea de imagini și realitatea augmentată și poate fi văzută ca o aplicație de bază pentru multe alte soluții inovatoare. Aceste soluții sunt atât din domeniul electrocasnicelor, dar cel mai important în domeniul medical, pentru ajutarea oamenilor aflați în dificultate. În acest sens aplicația *Room detector* poate fi extinsă pentru a crea aplicații pentru oamenii orbi sau pentru oameni cu dizabilități care folosesc un scaun cu rotile.

Am ales să dezvolt această aplicație bazându-mă pe cele mai noi tehnologii, astfel încât aplicația este capabilă să se adapteze oricând la noi module cât și la alte tipuri de detectări. Această aplicație este construită cu ajutorul platformei Unity, având astfel accesul la cel mai nou, flexibil și complex framework, AR Foundation. Cu ajutorul acestui framework, avem acces la realitatea augmentată atât de pe dispozitivele cu sistem de operare Android cât și cu sistem de operare iOS. În aplicația *Room detector*, framework-ul AR Foundation este folosit pentru preluarea cadrului de pe camera foto a device-ului.

Această aplicație a început ca un proiect de cercetare, ajungând în acest moment cu cea mai bună și rapidă abordare pe care am putut să o găsesc utilizând algoritmul *Oriented FAST and rotated BRIEF (ORB)*. Până să ajung la această soluție am trecut printr-o perioadă de cercetare și de testare.

Din punct de vedere a îmbunătățirilor, aș putea să precizez că pe viitor s-ar putea integra OpenCV direct în aplicație, pentru a nu fi nevoiți să avem o conexiune constantă la internet. Dacă această abordare nu se poate realiza sau nu se dovedește a fi eficientă, putem optimiza partea de algoritmi de pe server astfel încât toate calculele să fie realizate pe placa video (GPU), având șansa astfel să adăugăm cât mai multe camere și spații mari.

Consider că aplicația *Room detector* este o aplicație care poate aduce multe beneficii asupra noilor tehnologii precum AR Cloud. AR Cloud este o colecție de puncte cheie reprezentând o hartă a lumii 3D, suprapusă peste lumea reală, permițând augmentarea, partajarea și legarea informațiilor și experiențelor la anumite locații fizice. Misiunea AR Cloud este de a conduce dezvoltarea tehnologiei de calcul spațial, a datelor și a standardizării pentru conectarea lumilor fizice și digitale în beneficiul tuturor.

Bibliografie

1. Unity engine - https://unity3d.com/what-is-a-game-engine
2. Unity MonoBehaviour - https://docs.unity3d.com/ScriptReference/MonoBehaviour.html
3. AR Foundation – https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@3.1/manual/index.html
4. Flask - https://palletsprojects.com/p/flask/
5. Flask documentation - https://pymbook.readthedocs.io/en/latest/flask.html
6. Web server gateway interface (WSGI) - https://wsgi.readthedocs.io/en/latest/what.html
7. Microframework - https://en.wikipedia.org/wiki/Microframework
8. APIs with Flask - https://programminghistorian.org/en/lessons/creating-apis-with-python-and-flask#installing-python-and-flask
9. Open-CV - https://opencv.org/about/
10. Background Image - https://unsplash.com/photos/s0ftlzyQ8No
11. What is CUDA? - https://blogs.nvidia.com/blog/2012/09/10/what-is-cuda-2/
12. CUDA - https://opencv.org/platforms/cuda/
13. OpenCL - https://opencv.org/opencl/
14. Wait for end of frame - https://docs.unity3d.com/ScriptReference/WaitForEndOfFrame.html
15. Wait for fixed update - https://docs.unity3d.com/ScriptReference/WaitForFixedUpdate.html
16. Wait for seconds - https://docs.unity3d.com/ScriptReference/WaitForSeconds.html
17. Wait for seconds realtime - https://docs.unity3d.com/ScriptReference/WaitForSecondsRealtime.html
18. Wait until - https://docs.unity3d.com/ScriptReference/WaitUntil.html
19. Wait while - https://docs.unity3d.com/ScriptReference/WaitWhile.html
20. Unity Coroutines - https://www.tutorialspoint.com/unity/unity_coroutines.htm
21. How to use a coroutines? - https://vasundharavision.com/blog/Unity/how-to-use-coroutines-in-unity
22. Unity Web Request - Get - https://docs.unity3d.com/ScriptReference/Networking.UnityWebRequest.Get.html
23. Unity Web Request - Post - https://docs.unity3d.com/ScriptReference/Networking.UnityWebRequest.Post.html

24. Unity Web Request - Put - https://docs.unity3d.com/ScriptReference/Networking.UnityWebRequest.Put.html
25. Singleton - https://csharpindepth.com/articles/singleton
26. Introduction to ORB (Oriented fast and rotated brief) - https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf
27. Brute force matcher (BFmatcher) - https://docs.opencv.org/3.4/dc/dc3/tutorial_py_matcher.html
28. Accesing the Camera Image on the CPU https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@1.0/manual/cpu-camera-image.html
29. Feature matching - http://sword-work-special-offer.top/2614016cgrr/feature-matching.html
30. Open AR Cloud - https://www.openarcloud.org/
31. AR Cloud Immersive Technology - https://www.foundry.com/insights/vr-ar-mr/ar-cloud-immersive-technology