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 aplicatiei

Aplicatia *Room Detector* este o aplicatie conceputa pentru detectarea spatiului. Cu ajutorul acestei aplicatii, utilizatorii vor putea sa stocheze si sa detecteze diferite spatii(camere). Conceptul de spatiu in aceeasta aplicatie se refera la o camera care poate fi identificata dupa un nume introdus de utilizator. Utilizatorul poate folosi aceasta aplicatie din doua module prima fiind incarcarea spatiilor ce se doresc a fi inregistrare pentru a fi recunoscute mai tarziu, iar cel de-al doilea modul se va ocupa de detectarea spatiilor.

Pentru a incarca informatii despre o incapere utilizatorul va selecta din meniu optiunea „Upload Images", aceasta actiune va deschide un nou ecran care va contine lista camerelor existente si optiunea de a adauga o camera noua (care va fi indentificata dupa nume). Dupa adaugarea unui nou nume de camera, utilizatorul trebuie sa selecteze o camera si apoi sa apese pe butonul care simbolizeaza o camera foto. Aceasta actiune va deschide camera foto a utilizatorului si va avea posibilitatea de a trimite cadre din diferite unghiuri pentru camera selectata. Aceste cadre sunt utilizate mai apoi pentru recunoasterea spatiului respectiv.

Cel mai doilea modul, „Detects space", ofera utilizatorului posibilitatea de a scana spatii si ai afla numele spatiului, asa cum a fost configurat in pasul anterior. Selectarea acestei optiuni va deschide camera telefonului si va incepe detectarea spatiului. Atunci cand un spatiu este detectat numele camerei va aparea in partea de jos a ecranului. De asemenea, daca in timpul detectarii utilizatorul merge in alta camera, aplicatia va detecta schimbarea spatiului si ii va afisa numele camerei curente.

Motivatie

In ultimii ani, evolutia tehnologiei a fost ampla si rapida si oameni incearca sa atribuie majoritatea sarcinilor unor masinarii. In acest sens detectarea spatiilor este o unealta foarte puternica si folositoare. Aceasta aplicatie isi propune sa puna bazele unui mecanism de detectare flexibil, care poate fi adaptat mai multor solutii. Intalnim detectarea spatiului atat in depozitele marilor producatori, cat si in diferite electrocasnice care ne fac viata mult mai usoara. In acest sens acest algoritm de detectarea spatiilor poate fi folosit de micii robotei de la Amazon care se ocupa cu organizarea coletelor in depozite dar si micul si minunatul aspirator care este capabil sa faca curat in toata casa pentru noi.

Contributii

Aplicatia *Room detector* este o aplicatie care ofera utilizatorului sansa de configurare si de recunoastere a spatiului, intr-un timp foarte scurt. Aceasta aplicatie a inceput ca un proiect de cercetare, trecand astfel prin descoperirea a mai multor metode de detectare a spatiului. Majoritatea metodelor de detectare constau in pregatirea datelor inainte, iar utilizarea aplicatiei nu putea fi realizata imediat dupa configurare. Utilizatorul isi pregateste datele, le proceseaza si dupa o vreme pot fi folosite de aplicatie. Practic pentru pregatirea datelor utilizatorul trebuie sa mearga intr-o camera si sa realizeze un numar mare de fotografii. Dupa acest pas, fotografiile treceau printr-un proces de pregatire unde dura foarte mult. Aceasta cauza se datoreaza faptului ca fotografiile treceau printr-un proces de antrenare iar de abia la final, modelul antrenat putea fi utilizat. Nici in acel moment utilizatorul nu putea fi sigur ca modelul da randament 100% si trebuie testat in prealabil, iar daca este nevoie este pus inapoi la antrenat.

Dupa o perioada indelungata de cercetare si testare, am descoperit libraria OpenCV, pe care am folosit-o in aplicatie. Cu ajutorul librariei OpenCV am reusit sa gasesc cea mai buna si rapida abordare, utilizand algoritmul Oriented FAST and rotated BRIEF (ORB). Acest algoritm m-a ajutat sa extrag din poze doar ce este esential si sa trec peste procesul de antrenare care este folosit in majoritatea metodelor de detectare. Practic in momentul cand utilizatorul doreste sa recunoasca un spatiu, poate sa foloseasca foarte usor partea de admin care se numeste „Upload images”, trimitand astfel imagini cu spatiul care il doreste. Dupa ce a terminat de configurat, poate sa deschisa partea de utilizator care se numeste „Detects space” si sa inceapa scanarea si detectarea spatiului. In momentul cand utilizatorul deschide aceasta scena, pe server sunt pregatite datele instant, netrecand 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 in timp real de la aplicatie si returneaza numele camerei cu potrivirea punctelor cea mai buna. Spre deosebire de metodele celalte de detectare, aplicatia *Room detector* ofera utilizatorului sansa de configurare a spatiului intr-un timp foarte scurt, dandui posibilitatea de folosire imediata avand o acuratete foarte mare de detectare.

Aplicatia *Room detector* este o aplicatie de baza care este capabila sa se adapteze oricand la noi module cat si la alte tipuri de detectari, astfel incat in urma cercetarilor si implementarii actuale, aplicatia se poate adapta si poate fi folosita pentru rezolvarea mai multor situatii. Una dintre ele fiind chiar ajutorarea persoanelor cu dezabilitati de vedere, inlocuind in aplicatie afisarea textului cu un mesaj auditiv.

Capitole

Aspecte tehnice

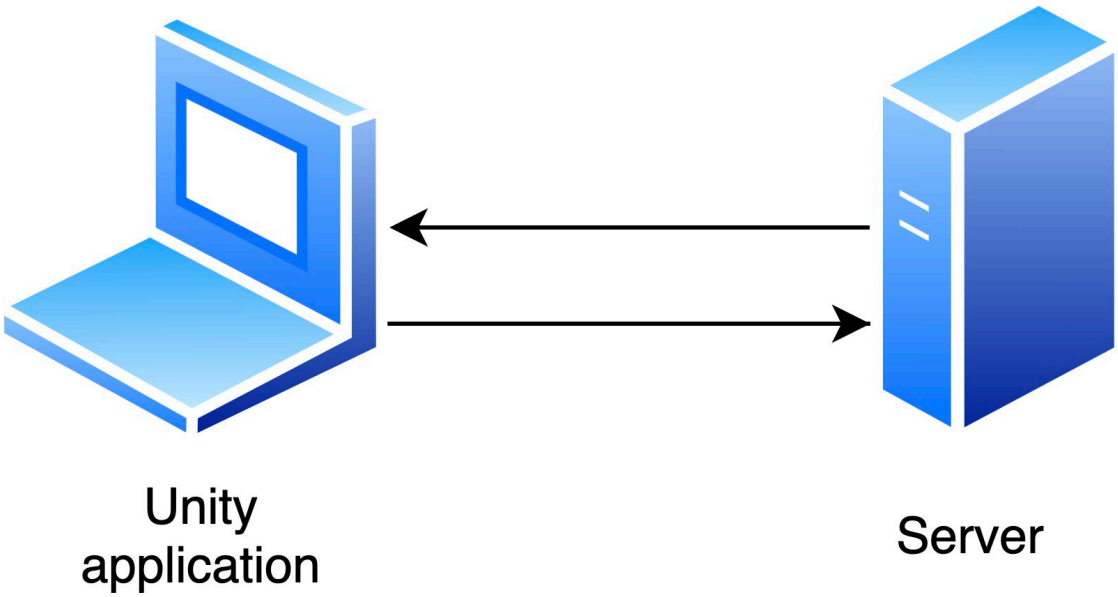

Figura 1 – Structura aplicatiei

Aplicatia Room Detector este o aplicatie creata in engine-ul Unity avand o comunicare continua cu un server unde trimite si primeste date de la acesta. Aplicatia Unity trimite ca date imagini care sunt procesate sau salvate pe server.

Platforme

Unity Engine

Unity Engine este un software care ofera creatorilor de joc setul de functii necesare pentru a construi jocuri rapid si eficient. Un motor de joc este un cadru pentru dezvoltarea jocurilor care sustin si reunesc mai multe domenii de baza. Poate importa elemente 2D cat si elemente 3D din alte software-uri ca Maya, Photoshop s.a.m.d.. Toate aceste elemente adaugate intr-o scena, avand si efecte luminoase si audio, animatii si fizica pot rezulta un joc.

Tehnologii

AR Foundation

AR Foundation este un framework ce ne permite sa lucram cu platforme de realitate augmentata cu ajutorul engine-ului Unity. Acest framework reprezinta o interfata ce este construita peste framework-urile ARKit XR sau ARCore XR, astfel incat poate fi folosit in momentul construiri unei aplicatii, pentru a realiza o aplicatie comuna atat pentru dispozitive cu iOS cat si pentru dispozitive cu Android.

AR Foundation APIs

AR Foundation foloseste un set de metode din clasa de baza MonoBehaviour, si are un set de API-uri pentru anumite dispozitive care accepta urmatoarele concepte:

- World tracking
    - Urmarirea pozitiei si orientarii dispozitivului in spatiul fizic
- Plane detection
    - Detectarea suprafetelor orizontale si verticale
- Point clouds
    - Seturi de date ce reprezinta puncte caracteristice.
- Anchor
    - Pozitia si orientarea pe care dispozitivul o recunoaste si o urmareste
- Light estimation
    - Estimari pentru nuanta de culoare medie pentru detectarea luminozitatii in spatiul fizic
- Environment probe
    - Tool pentru generarea unei harti care reprezinta o anumita zona in mediul fizic
- Face tracking
    - Detectarea si urmarirea fetelor umane
- Image tracking
    - Detectarea si urmarirea imaginilor 2D

AR Foundation Subsystems

AR Foundation este construit pe subsystems. Un subsystem este o interfata pentru afisarea diferitelor tipuri de informatii. Acestea pot fi gasite in namespace-ul UnityEngine.XR.ARSubsystems. Fiecare subsystem gestioneaza functionalitati specifice pentru fiecare furnizor in parte. Furnizorii sunt ARCore XR si ARKit XR. Fiecare furnizor individua stabileste modul de implementare pentru fiecare subsystem a lui. In general ele infasoara SDK-uri nativ al platformei respective (ARKit pentru iOS si 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 si usoare, cu posiblitatea de adaptare la aplicatii complexe. Flask ofera instrumente, biblioteci si tehnologii care permit construirea unei aplicatii 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 in care un server web comunica cu aplicatiile web si modul in care aplicatiile web pot fi inlantuite impreuna 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 faciliteaza primirea unei solicitari HTTP si dirijarea cererii HTTP catre controlul adecvat, expedierea controlerului si returnarea unui raspuns 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 interfata conceputa pentru a fi utilizata de om. Caracteristici:

- Setul de date poate fi mare, facand descarcarea prin FTP
- Utilizatorii acceseaza datele in timp real
- Datele pot fi actualizate frecvent
- Utilizatorii au nevoie de acces la date simultan
- Utilizatorii au capabilitatea sa efectueze si alte actiuni, cum ar fi contribuirea, actualizarea sau stergerea datelor.

Open-CV

Este o biblioteca software de computer vision si software de invatare automata. Open-CV a fost construit pentru a furniza o infrastructura comuna pentru aplicatiile de viziune computerizata. Biblioteca are peste 2500 de algoritmi optimizati, care include un set de algoritmi pentru computior vision dar si de invatare automata. Acesti algoritmi pot fi folositi pentru detectarea si recunoasterea fetelor, identificarea obiectelor, clasificarea actiunilor umane in videoclipuri, urmarirea miscarilor camerei, urmarirea obiectelor in miscare, extragerea de modele 3D, producerea unor point clounds 3D din camerele stereo, lipirea imaginilor pentru a produce o imagine cu rezolutie inalta s.a.m.d..

Open-CV are interfete pentru C++, Python, Java si MATLAB si accepta Windows, Linux, Android si Mac OS. Open-CV se bazeaza pe aplicatiile de viziune in timp real si profita de instructiunile MMX si SSE, atunci cand sunt sunt disponibile. In prezent sunt dezvoltate activ o inferfata cu caracteristici complete CUDA si OpenCL. Exista peste 500 de algoritmi si de aproximativ 10 ori mai multe functii care compun sau sustin acei algoritmi. Open-CV este scris nativ in C++ si are o interfata sablonata care functioneaza perfect cu containerele STL.

CUDA

CUDA este o platforma de calcul paralela si un model de programare care face ca utilizarea unui GPU pentru calcul general sa fie simpla si eleganta.

Acceleratoarele moderne ale GPU-ului au devenit puternice si sunt eficiente pentru a efectua calcule cu scop general. Este o zona cu o dezvoltare foarte rapida, care genereaza mult interes din partea oamenilor de stiinta, cercetatorilor si inginerilor care dezvolta aplicatii intensiv din punct de vedere computerizat. In ciuda dificultarilor de reimplementare a algoritmilor pe GPU, multi oameni o fac pentru a verifica cat de repede ar putea fi. Pentru a sustine astfel de eforturi, /au fost disponibile o multime de limbi si instrumente avansate, precum CUDA.

Open Computing Language (OpenCL)

Este un standard pentru scrierea codului care ruleaza pe platforme eterogene incluzand procesoare, GPU-uri, DSP-uri si etc. In special OpenCL ofera aplicatiilor un acces la GPU-uri pentru calcul non-grafic care. In computer vision multi algoritmi pot rula pe un GPU mult mai eficient decat pe un procesor. Procesarea imaginii, aritmetica matriciala, detectarea obiectelor sunt unele dintre exemple.

Modulele aplicatiei

Scena principala

Ecranul splash

Splash screen este un ecran introductiv continand imaginea de fundal si logo-ul aplicatiei si pe parcursul a 3 secunde se realizeaza o trecerea de la splash screen catre primul screen printr-o tranzitie alpha.

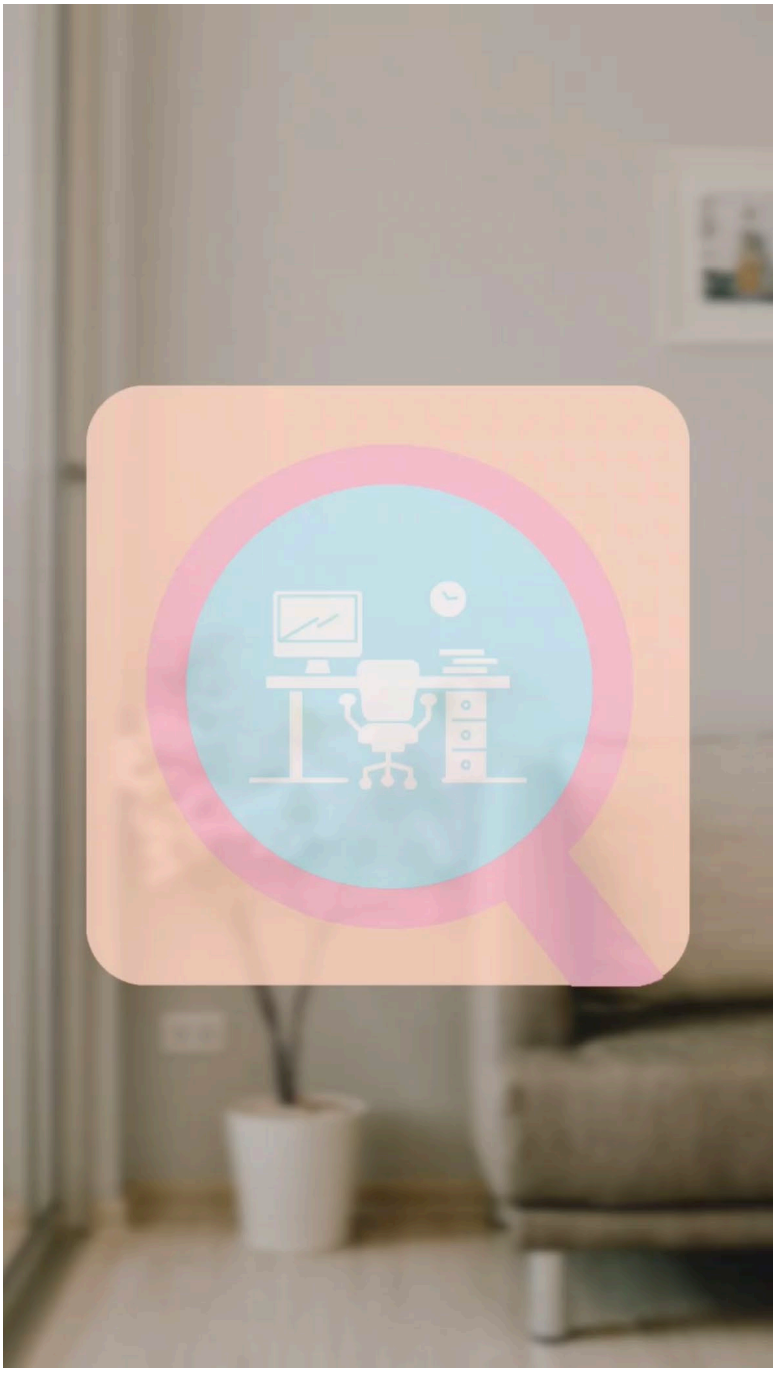

Figura 2 – Ecran introductiv cu logo-ul aplicatiei

Ecranul principal

In primul ecran utilizatorul gaseste titlul aplicatiei impreuna cu cele doua butoane unde in functie de fiecare buton, utilizatorul va ajunge pe un alt ecran. Cele 2 butoane sunt urmatoarele:

1. Upload Images
2. Detects space

Figura 3 – Primul ecran al aplicatiei

Scena „Upload Images”

Configurare camera

Imediat dupa apasarea butonului „Upload Images”, utilizatorul va fi redirectionat catre acest ecran, unde poate sa aleaga, sa adauge o camera sau sa vada camera curenta care a fost selectata. Imediat dupa selectarea unei camere poate sa apese pe butonul cu icon-ul tip camera si va fi redirectionat catre cel de-al doilea ecran.

Selectarea unei camere se realizeaza prin apasarea unui item din lista de pe pagina. Pentru adaugarea unei camere trebuie sa selectam campul din josul paginii si sa tastam numele pe care il dorim sa il asignam camerei. Ulterior putem apasa pe butonul „Add room” si camera se va adauga in lista.

Figura 4 – Configurarea camerelor

Incarcarea imaginilor pentru camere

Dupa ce am apasat pe butonul tip camera utilizatorul fi redirectionati la acest ecran, unde camera dispozitivului se va activa si va reda utilizatorului fiecare cadru. In acest ecran putem gasi numele camerei curente, un buton „Send image” pentru trimiterea imaginei catre server, dar si un buton de back daca am dori sa ne intoarcem la screen-ul anterior.

Imediat dupa ce ne-am selectat cadrul pe care-l dorim sa il trimitem la server putem apasa pe Send Image.

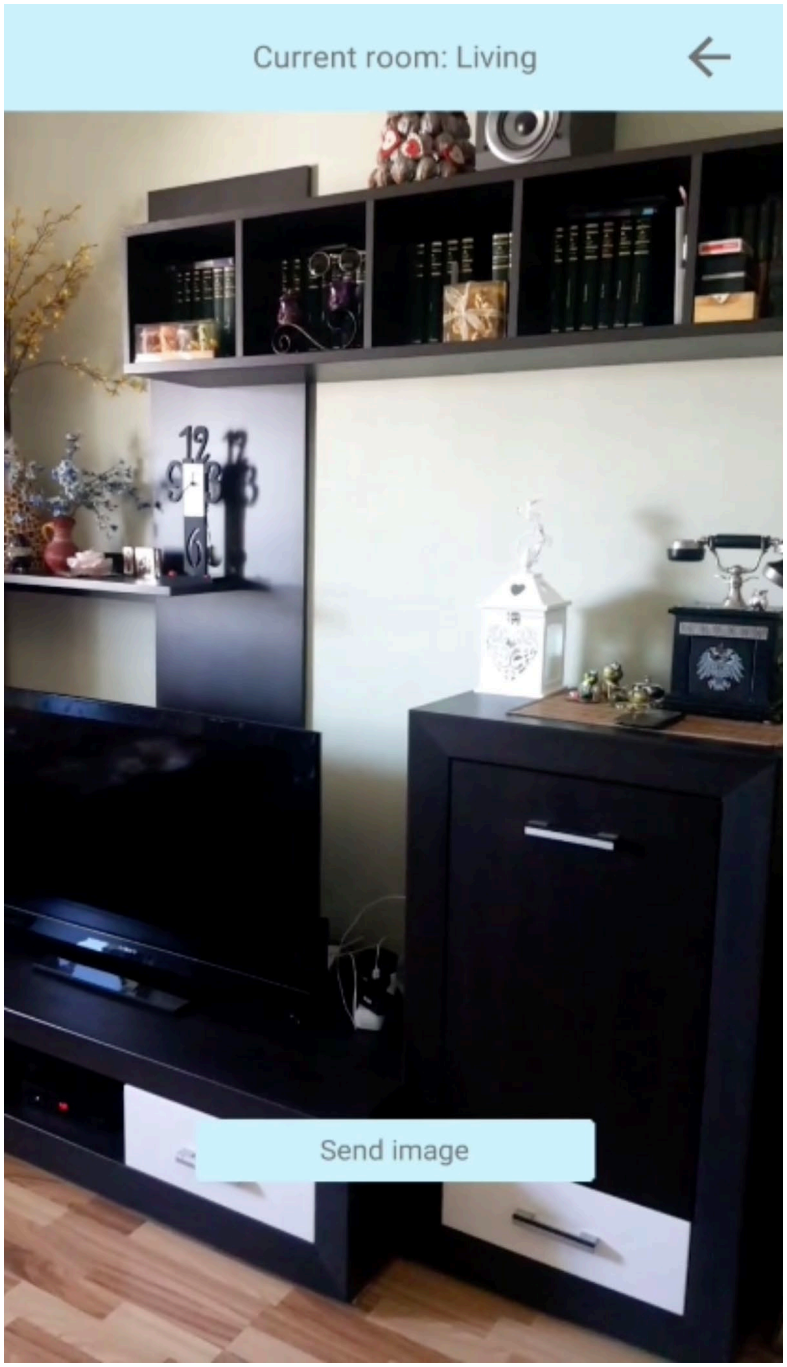

Figura 5 – Incarcarea imaginilor pe server

Pentru confirmarea trimiterii imaginei catre server avem un pop-up cu un mesaj: „Room image uploaded!”

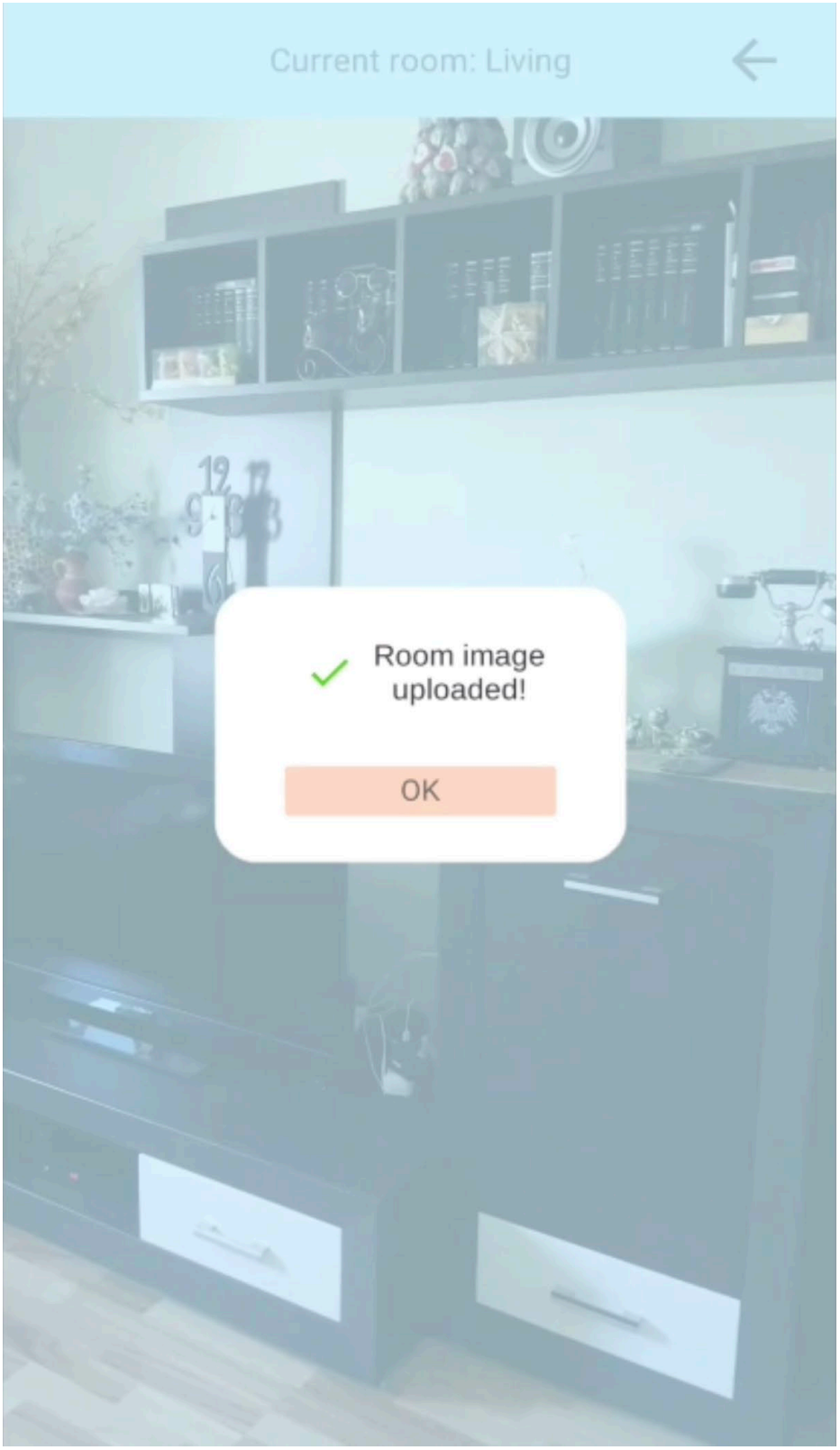

Figura 6 – Notificare incarcare cu succes

Scena „Detects space”

In acest moment, serverul pregateste datele, aplicatia trimite cadrele de la camera catre server, iar serverul trimite inapoi un raspuns daca a fost sau nu recunoscut spatiul respectiv.

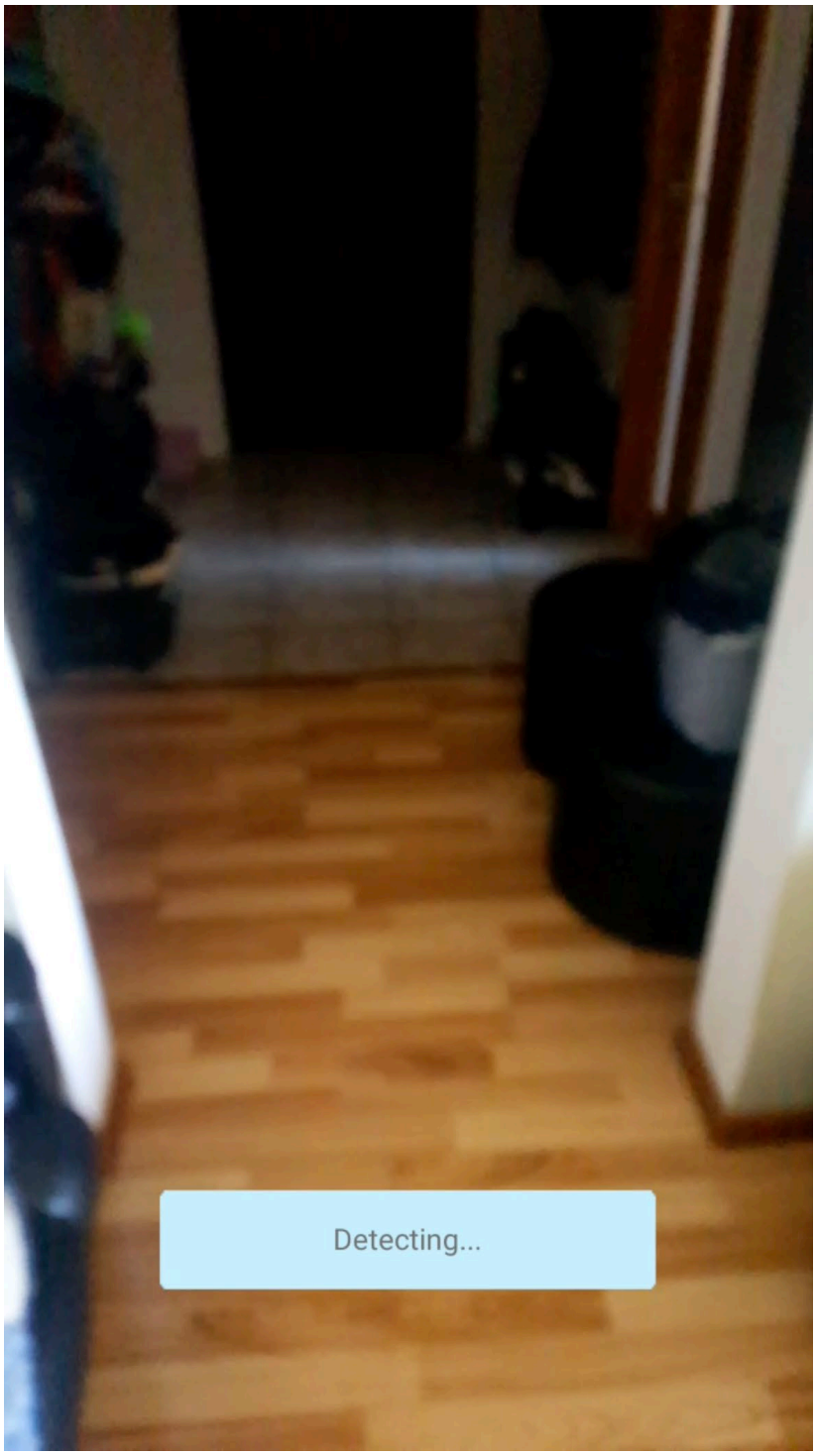

Figura 7 – Primul ecran din scena utilizatorului

In momentul cand spatiul este recunoscut, textul din josul paginiei se va modifica de la „Detecting...” sau din numele camerei curente, la ultima camera care a fost recunoscuta.

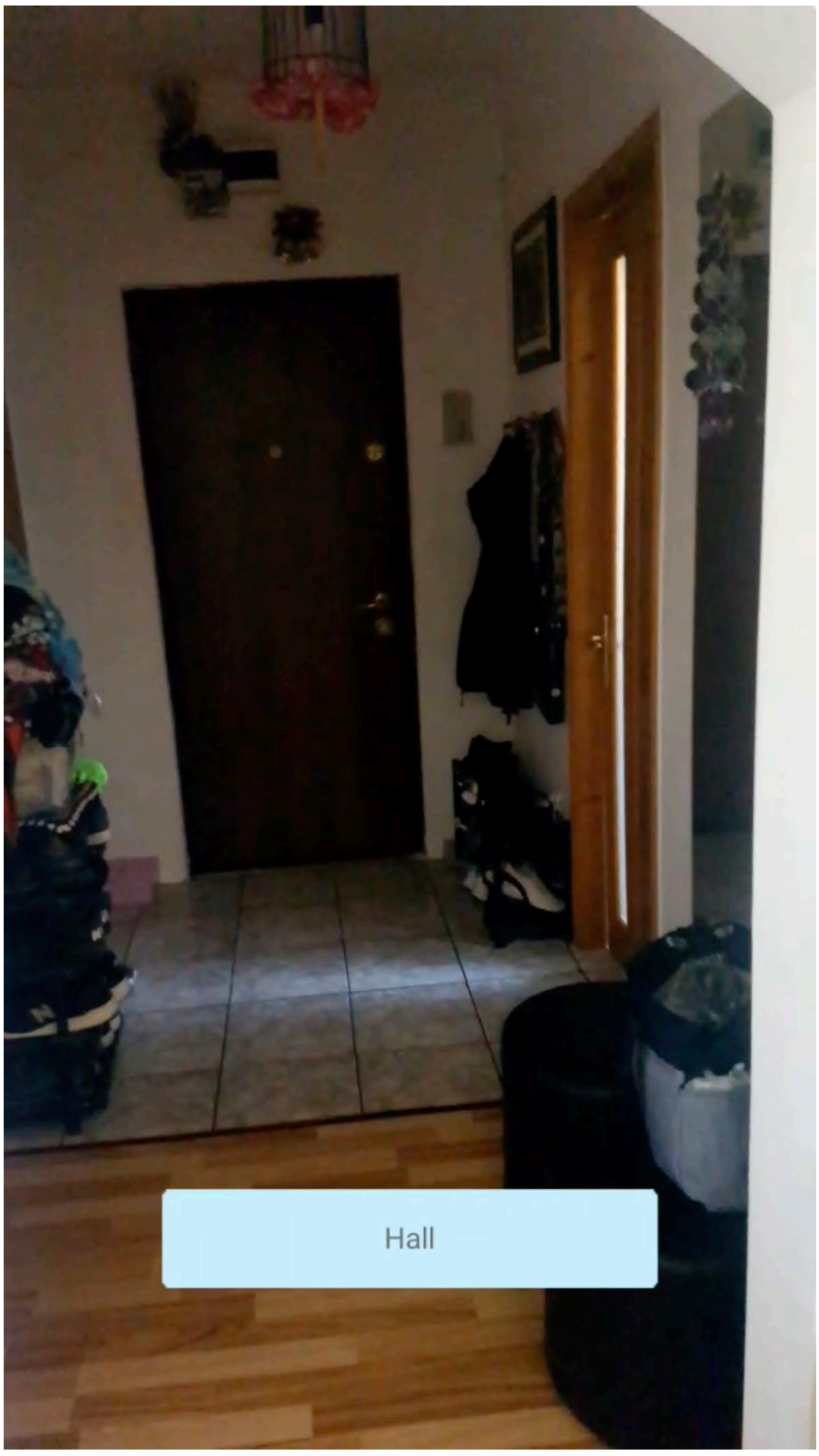

Figura 8 – Afisarea spatiului detectat

Descrierea solutiei

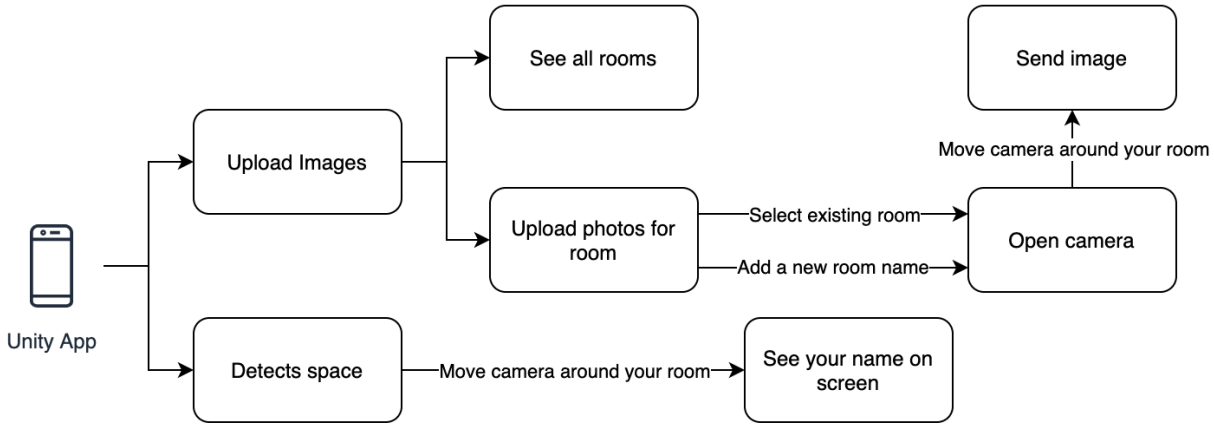

Figura 9 – Diagrama de stari

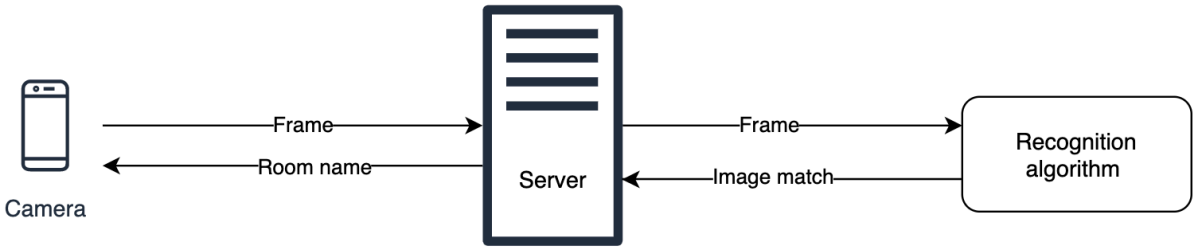

Figura 10 – Diagrama aplicatiei

Nucleul aplicatiilor Unity se bazeaza pe Entity Component System (ECS), iar fiecare termen are o anumita semnificatie:

- Entity – Entitatile sau lucrurile care populeaza aplicatia Unity (GameObject)
- Component – Datele asociate pe entitatile noastre
- System – Logica care transforma datele din starea curenta in starea urmatoare

In Unity, entitatile mai au numele de „GameObject”. Ele sunt obiecte fundamentale din Unity ce sunt adaugate in scena. Ca exemplu un GameObject poate sa fie un model 3D sau un obiect gol. Ele nu realizeaza prea multe, in schimb pot detine componente unde ele implementeaza functionalitatea reala astfel incat ulterior sa actioneze asupra GameObject-urilor. O scena detine GameObject-urile noastre, reprezentand un nivel a aplicatiei.

Scripturile Unity stau la baza MonoBehaviour-ului. MonoBehaviour este o clasa de baza de la care deriva fiecare script. Aceasta clasa ofera un numar de functii astfel incat ne va ajuta sa dezvoltam o aplicatie Unity. Script-urile au un rol foarte important astfel incat ele le spun entitatilor/GameObject-urilor cum sa se comporte. Unity ruleaza intr-o bucla mare astfel incat script-urile atasate de GameObject-uri creaza o interactiune intre ele, citind toate datele existente dintr-o scena si realizeaza comportamentul pe care il dorim.

Majoritatea script-urile pot sa detina una sau mai multe metode din MonoBehaviour. Cele mai folosite metode sunt urmatoarele:

- OnEnable()
    - Metoda OnEnable() se apeleaza atunci cand GameObject-ul se activeaza.
- Awake()
    - Aceasta metoda este apelata o singura data cand este initiat un GameObject. Daca GameObject-ul este inactiv, atunci aceasta metoda nu va fi apelata chiar daca componenta este activa sau nu. In caz ca GameObject-ul este activ, dar componenta este sau nu activa, metoda Awake() se va apela.
- Start()
    - Metoda Start() se apeleaza doar daca GameObject-ul si Componenta sunt active si mereu se va apela inaintea metodei Update().
- FixedUpdate()
    - Aceasta metoda se foloseste atunci cand in scena avem GameObject-uri care au anumita fizica, adica cele care detin componente Rigidbody.
- Update()
    - Metoda Update() se apeleaza de fiecare data cand aplicatia trece la un nou frame. In aceasta metoda se introduce logica care dorim sa fie continua, adica la fiecare cadru al aplicatiei. De exemplu: Animatii, AI, s.a.m.d.
- LateUpdate()
    - Aceasta Metoda se apeleaza la sfarsitul cadrului. In momentul cand metoda Update() termina de executat se va trece la LateUpdate(). Un exemplu de functionalitate este atunci cand o camera ar trebui sa urmareasca un personaj. Pozitia personajului este actualizata pe metoda Update(), iar pe LateUpdate() este executata camera care urmareste personajul, astfel incat nu se creaza diferenta vizuala dintre cadre.
- OnDisable()
    - Metoda OnDisable() se apeleaza atunci cand GameObject-ul se dezactiveaza.

Majoritatea claselor din Unity care au anumite responsabilitati importante sunt facute singleton. Singleton este unul dintre cele mai cunoscute pattern-uri din ingineria software. Un singleton este o clasa care permite crearea unei singure instante de sine si ofera acces simplu la acea instanta. In aplicatia „Room Detector” am folosit pentru cateva clase acest pattern. Am implementat o metoda „Singleton” unde verifica daca instanta este null, atunci aceasta poate sa fie asignata cu ea insasi. In caz ca instanta nu este null si este diferita de ea insasi, putem sa distrugem componenta respectiva astfel inca sa nu existe mai multe instante de acelasi tip. Metoda „Singleton” este apelata din metoda „Awake”, doarece aceasta metoda se apeleaza prima data in momentul cand deschidem aplicatia. In caz ca o alta clasa apeleaza instanta noastra inainte sa fie initializata, va intra pe proprietate si va cauta in scena Unity, GameObjectul care are numele identic cu numele clasei astfel incat forteaza initializarea prin preluarea componentei. Prin acesta abordare 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

In aplicatia s-au mai folosit metode Coroutine. O metoda Coroutine este o functie care are capacitatea de a intrerupe propria executie si de a returna controlul din acelasi punct dupa ce o conditie este indeplinita. Dupa cum stim ca functia simpla poate returna orice tip, functiile coroutine pot returna doar IEnumerator si trebuie utilizat cuvvantul cheie „yield” inainte de cuvantul cheie „return”. Aceasta este diferenta principala intre functiile standard C# si functiile Coroutine. Aceste metode se folosesc pentru a imparti fluxul de lucru in mai multe cadre, fiind

o solutie mult mai optima decat folosire metodei „Update” neavand nici un control asupra ei, in timp ce codul Coroutine poate fi executat atunci cand indeplineste o anumita conditie. De exemplu:

- yield return null
  - Acest randament are capacitatea de a relua executia dupa ce au fost apelate toate functiile de actualizare pe cadrul urmator.
- yield return new WaitForEndOfFrame
  - Acest randament aceaspta pana la sfarsitul cadrului dupa ce Unity a redat fiecare camera.
- yield return new WaitForFixedUpdate
  - Acest randament aceaspta pana la urmatoarea functie de actualizarea a radei de cadru fixa.
- yield return new WaitForSeconds
  - Suspenda executia metodei pentru durata data de secunde folosind timpul scalat. Timpul real suspendat este egal cu timpul dat impartit la Time.timeScale.
- yield return new WaitForSecondsRealtime
  - Aceast randament suspenda executia metodei pentru durata data de secunde folosind timpul neschimbat. In comparatie cu metoda „WaitForSeconds” nu tine cont de timpul scalat din aplicatie.
- yield return new WaitUntil
  - Aceast randament suspenda executia metodei pana cand conditia delegate-ului furnizat devine true.
- yield return new WaitWhile
  - Aceast randament suspenda executia metodei pana cand conditia delegate-ului furnizat devine false.
- yield return new WWW(URL)
  - Acest randament are capacitatea de a relua executia dupa ce resursa web de la URL a fost descarcata sau esuata.

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

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

Pentru oprirea unei Coroutine se poate apela medota StopCoroutine(), avand parametrul chiar instanta apelului StartCoroutine().

Unity are 3 tipuri de metode Coroutine:

- Asynchronous Coroutines
- Synchronous Coroutines
- Parallel Coroutines

Asynchronous Coroutines sunt coroutine in cadrul unei coroutine existente. Nu interactinoeaza direct si cel mai important nu se asteapta unul pe celalalt.

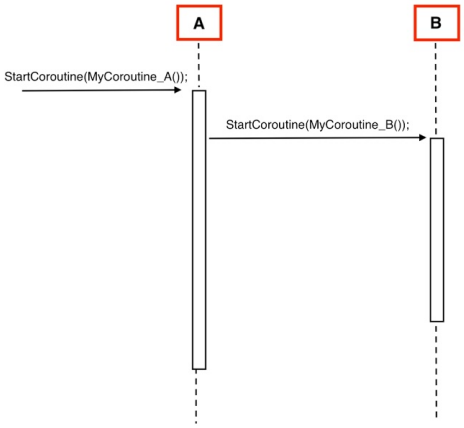

Figura 12 – Asynchronous Coroutines

Synchronous Coroutines sunt coroutine care asteapta executarea altei coroutine, print „yield return StartCoroutine(SampleCoroutine());”. In momentul cand o coroutine porneste alta coroutina blocheaza executarea pana cand acea coroutine nu este gata

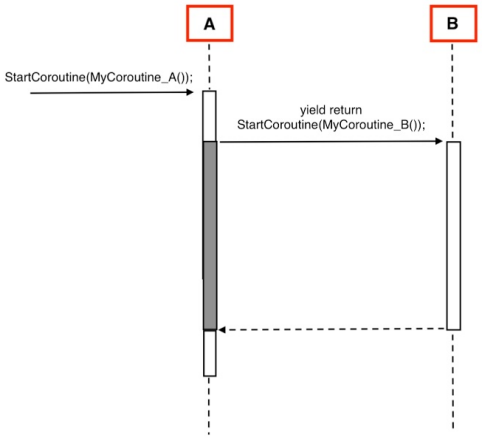

Figura 13 – Synchronous Coroutines

Parallel Coroutines sunt coroutine care au la finalul metodei interogarea instantei unei alte coroutine. In comparatie cu Synchronous Coroutines unde interogheaza direct apelul StartCoroutine().

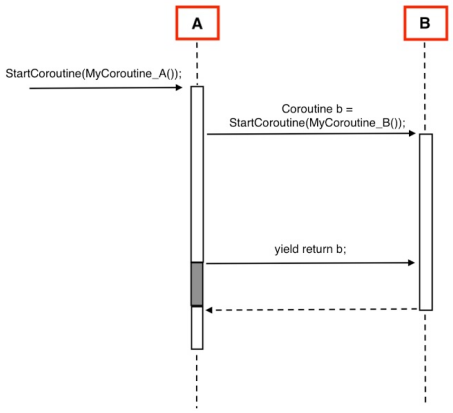

Figura 14 – Parallel Coroutines

Exemple cu manageri.

Pentru comunicarea cu server-ul am creat o clasa numita „ApiManager” avand responsabilitatea de a face anumite apel-uri catre end-point-urile de pe server. Aceasta clasa este singleton, deoarece avem nevoie sa o accesam din mai multe puncte al aplicatiei. In clasa gasim 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

```
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 in aplicatie pot fi gasite in clasa statica „Constants”, continand mai multe constante despre rute, host, port, s.a.m.d..

```
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";

    //const string HOST = "http://127.0.0.1";
    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 principala

In Main Scene vom gasi prima pagina a aplicatiei, unde putem alege unu dintre cele 2 butoane Upload Images sau Detects space. In functie de butonul ales vom fi trimisi la o anumita scena.

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

Figura 20 – Incarcarea scenei bazat pe ID

Fiecare scena are un id astfel incat sa fie usor identificata si incarcata. Metoda LoadScene este asignata de fiecare buton din MainScene impreuna cu id-ul scenei ca parametru.

Scena admin sau Upload Images

In momentul cand deschidem aceasta scena, clasa AdminSceneManager realizeaza un apel catre end-point-ul Constants.UriGetRooms = „/admin” din metoda Awake() pentru a aduce lista cu numele camerelelor.

```
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 catre server pentru aducerea camerelelor

Cand server-ul primeste apelul, se pregatesc datele despre camere pe server si returneaza catre aplicatie o lista avand numele fiecareia.

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

Figura 22 – End-point „/admin”

Dupa primirea listei, aplicatia Unity deserializeaza JSON-ul primit si populeaza lista cu camerele care sunt pe server. (Figura de mai sus) Utilizatorul poate sa selecteze una dintre camere si sa treaca la urmatoarea pagina unde este capabil sa incarce poze cu camera respectiva. Pe server ajung imaginile unde sunt salvate intr-un folder cu numele camerei curente. Daca camera selectata de abia a fost creata, se va genera un folder nou. Pentru trimiterea imaginilor se apeleaza end-point-ul „/send”, avand ca date un JSON reprezentand modelul ImageData. ImageData contine numele camerei, dar si un array de byte fiind imaginea noastra.

```
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.SetActivePopup(callback == Constants.MessageSuccessServer);
                        Debug.Log(callback);
                    }
            )
    );
}
```

Figura 22 – Apel catre server pentru trimiterea cadrului

In momentul cand utilizatorul apasa pe butonul „Send Image”, se va apela metoda din figura de mai sus. In aceasta metoda verificam daca a fost selectata o camera astfel incat daca nu a fost selectata atentionam utilizatorul sa se intoarca la ecranul precendent si sa selecteze una. In cazul in care o camera a fost selectata, pregatim trimiterea cadrului si asteptam raspuns de la server daca a fost sau nu trimisa poza cu succes.

```python
@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

Dupa ce am terminat partea de configurare a camerelelor, putem sa ne intoarcem la primul ecran pentru a testa daca datele/imaginile de pe server sunt de ajuns, pentru o recunoastere usoara si rapida.

Scena utilizatorului sau Detects Space

In comparatie cu scena anterioara, in aceasta scena se va folosi camera telefonului fara oprire trimitand cadre catre server. Pentru aceasta scena, clasa UserSceneManager realizeaza un apel catre end-point-ul UriPrepareContentServer = „/user” in metoda Awake() pregatind datele pe server.

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

Figura 25 – Trimitearea unui apel catre server pentru pregatiera datelor

Cand server-ul receptioneaza apelul de la aplicatie pregateste local intr-o lista descriptorii pentru fiecare imagine si trimite un mesaj de validare daca totul a functionat 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);
        }
    }
    // ReSharper disable once IteratorNeverReturns
}
```

Figura 27 – Pornirea unei coroutine si procesarea cadrelor

Aceasta clasa are implementata o coroutiune la Start, astfel incat la initializarea componentei nu se va apela metoda Start ca de obicei, ci se va crea o Coroutine care se va bloca pana cand conditia este indeplinita. Conditia este sa primim de la server mesajul de confirmare. Daca mesajul de confirmare a ajuns atunci putem incepe trimiterea cadrelor catre server pentru a incepe detectarea spatiului.

Trimiterea cadrelor se realizeaza printr-o bucla cu ajutorul instructiunii „while”. In momentul trimiterii imaginei catre server prin end-point-ul UriDetection = „/detection”, se asteapta raspunsul daca s-a realizat cu succes sau nu trimiterea cadrului, astfel incat la primirea raspunsului aplicatia va pregati si va trimite un nou cadru. Cadrele sunt preluate din clasa CameraManager prin metoda GetJpgTexture. Aceasta clasa este preluata de la ARFoundation, fiind folosita 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

app.run(host = '0.0.0.0', port = '5000')
```

Figura 28 – End-point „/detection”

In momentul cand se realizeaza un apel catre end-point-ul „/detection”, se vor prelua datele si se vor transforma acestea intr-un format de imagine. Imediat dupa acest proces, se va apela urmatoarea linie „value = FindRoom(GetDescriptors(img));”, unde sunt identificate 2 metode. Una dintre ele fiind „GetDescriptors()”, unde primeste ca parametru imaginea primita din aplicatie si returneaza 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 primita de la aplicatie

In metoda se mai apeleaza o alta functie „GetKeypointsAndDescriptors()”, primind imaginea care vine ca parametru. „GetKeypointsAndDescriptors()” returneaza keypoint-urile si descriptorii pentru cadrul curent primit de la aplicatie.

Cea dea doua metoda este „FindRoom()” unde primeste ca parametru descriptorii imaginei curente din aplicatie si parcurge astfel prin toti descriptorii fiecarei imagini de pe server returnand cea mai buna potrivire intre descriptori.

```
def FilterMatches(matches, matchesThreshold):
    dist = [m.distance for m in matches]
    if(len(dist) != 0):
        thres_dist = (sum(dist) / len(dist)) * matchesThreshold
    matches = [m for m in matches if m.distance < thres_dist]
    return matches

def GetMatches(des1, des):
    try:
        # Match descriptors.
        matches = bf.match(des1 ,des)
    except:
        matches = []
        print("An exception occurred")
    return FilterMatches(matches, 0.5)

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 si returnarea numelei camerei

In unele metode de mai sus s-au folosit 2 variabile „orb” si „bf”. Aceste 2 variabile sunt variabilele cheie pentru recunoasterea spatiului. Prima variabila reprezinta initierea detectorului ORB, iar cea dea doua reprezinta un obiect BFMatcher. Pentru mai multe detalii despre ORB, se pot gasi in punctul 4.

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

Figura 30 – Instantele algoritimilor ORB si BFMatcher

Algoritmul „Oriented FAST and rotated BRIEF” (ORB)

ORB a fost dezvoltat in laboratoarele OpenCV de Ethan Rublee, Vincent Rabaud, Kurt Konolige si Gary R. Bradski in 2011, ca o alternativa eficieta si viabila la algoritmi SIFT si SURF. ORB a fost conceput in principar deorece algoritmi SIFT si SURF sunt algoritmi patentati, insa algoritmul ORB este liber de utilizat.

ORB efectueaza la fel de bine ca algoritmul SIRF sarcina de detectare a caracteristicilor, fiind mai bun si rapid decat SURF de 2 ori mai mult. ORB se bazeaza pe binecunoscutul detector de puncte cheie FAST si descriptorul BRIEF. Ambele tehnici sunt atractive datorita performantei lor bune si a costurilor reduse.

Principalele contributii ale ORB sunt urmatoarele:

- Adaugarea unei componente de orientare rapida si precisa la FAST
- Calcul eficient al caracteristicilor orientate BRIEF
- Analiza variantzei si corelarea caracteristicilor orientate BRIEF
- O metoda de invatare prin functiile BRIEF sub invariantza rotativa, ceea ce duce la o performanta mai buna

Fast (Features from Accelerated and Segments Test)

Avand in vedere un pixel p dintr-o poza, se compara rapid luminozitatea pixelului p cu alti 16 pixeli, aflandu-se intr-un cerc mic in jurul lui p. Pixelii din cerc sunt apoi sortati in trei clase (mai deschise decat p, mai inchise decat p sau similare cu p). Daca mai mult de 8 pizeli sunt mai intunecati sau mai luminasi decat p, este selectat ca punct cheie. Astfel, punctele cheie gasite de algorimtul FAST ne ofera informatii despre locatie determinand marginile dintr-o imagine.

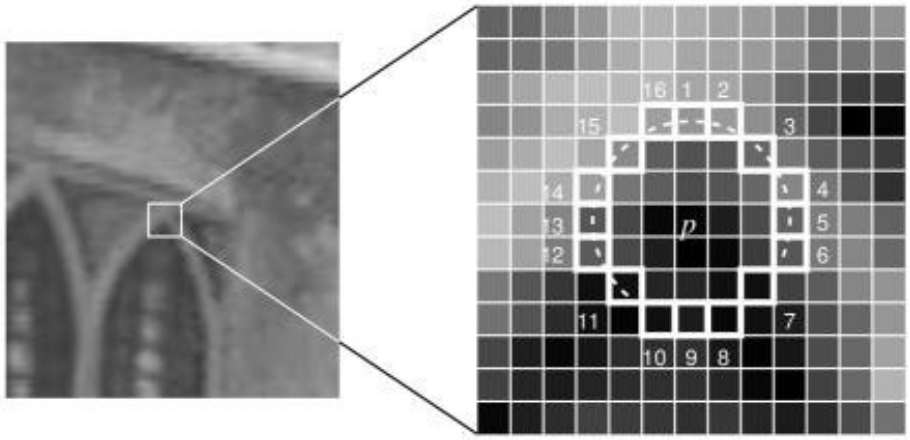

Figura 31 – Exemplu detectare punct cheie

Cu toate acestea, functiile FAST nu au o componenta de orientare si functii pe mai multe niveluri. Deci, algoritmul ORV foloseste o piramida de imagine pe mai mule niveluri. O piramida a imaginii este o reprezentare pe mai multe niveluri a unei singure imagini, care consta din secvente de imagini, toate fiind versiuni ale imaginii la diferite rezolutii. Fiecare nivel din piramida contine versiunea esantionata a imaginii fata de nivelul anterior. Odata ce algoritmul ORB a creat o piramida, acesta folosesete algoritmul FAST pentru a detecta punctele cheie din imagine. Prin detectarea punctelor cheie la fiecare nivel, ORB localizeaza in mod eficient punctele cheie la o scara diferita. In acest fel, ORB este invariabil la scara partiala.

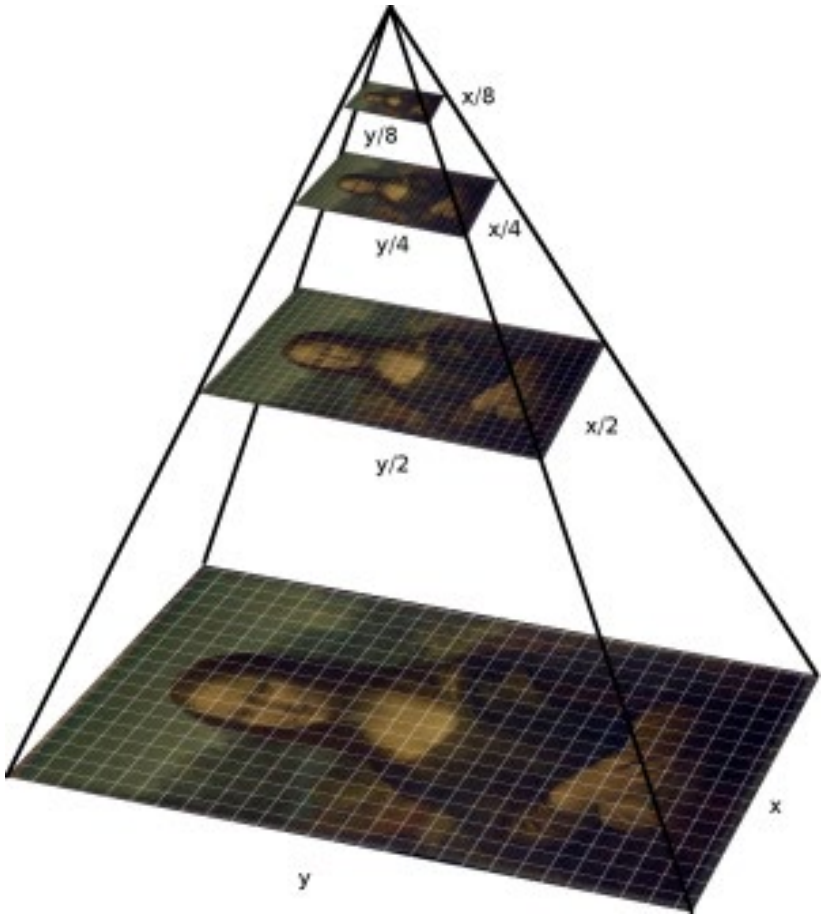

Figura 32 – Piramida imaginei

Dupa localizarea punctelor cheie, ORB le atribuie o orientare fiecarui punct, cum este orientat spre standa sau spre dreapta, in functie de cum se schimba nivelurile de intensitate in jurul acelui punct. Pentru detectarea schimbarii de intensitate orb foloseste „centroid de itensitate”. Centroidul de intensitate presupune ca intesitatea unui colt este compensata de centrul sau si acest vector poate fi utilizat pentru a impune o orientare.

Brief (Binary robust independent elementary feature)

Brief preia toate punctele cheie gasite de algoritmul FAST si le converteste intr-un vector de caractere binar, astfel incat impreuna sa poata reprezenta un obiect. Vectorul caracteristicilor binare este de asemenea, un descriptor de functii binare reprezentand un vector de caracteristici care contine doar 1 si 0. Pe scurt, fiecare punct cheie este descis de un vector de caracteristici, care este un sir de 128-521 biti.

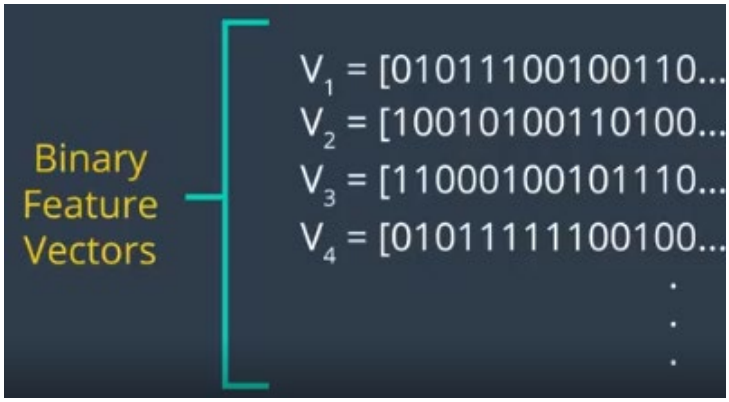

Figura 33 – Exemple de date pentru algoritmul BRIEF

Pentru a impiedica descriptorii sa fie senzibili la anumite zone cu inalta frecventa se foloseste un nucleu gaussian. Se selecteaza o pereche de pixeli aleatori intr-o zona definita in jurul punctului cheie. Zona definita in jurul pixelului este cunoscuta sub numele de „patch”, care este un patrat cu o latime si o inaltime a pixelilor. Primul pixel din perechea aleatorie este extras dintr-o distributie gaussiana centrata in jurul punctului cheie. Cel de-al doilea pixel din perechea aleatorie este extras dintr-o distributie gaussiana centrata in jurul primului pixel. Acum daca primul pixel este mai luminos decat al doilea, acesta atribuie valoarea lui 1, altfel 0. Pe scurt, se selecteaza o pereche intamplatoare si se atribuie volari. Pentru un vector de 128 de biti, se repeta acest proces de 128 de ori pentru un punct cheie. Algoritmul Brief creaza un vector ca acesta pentru fiecare punct cheie dintr-o imagine si nu este de asemenea invariabil la rotatie, astfel incat ORB foloseste rBRIEF (Rotation BRIEF). ORB inceasca sa adauge aceasta functionalitate fara a pierde din aspectul de viteza al algoritmului BRIEF.

Figura 34 – Exemplu puncte cheie

BFMatcher (Brute force matcher)

BFMatcher este un algoritm ce compara si potriveste caracteristicile fiecarei imagini. Acesta ia descriptorul unei caracterestici din primul set si este comparat cu toate celelalte caracteristici din setul doi folosind un anumit calcul al distantei, iar cel mai apropiat este returnat.

Punerea in aplicare

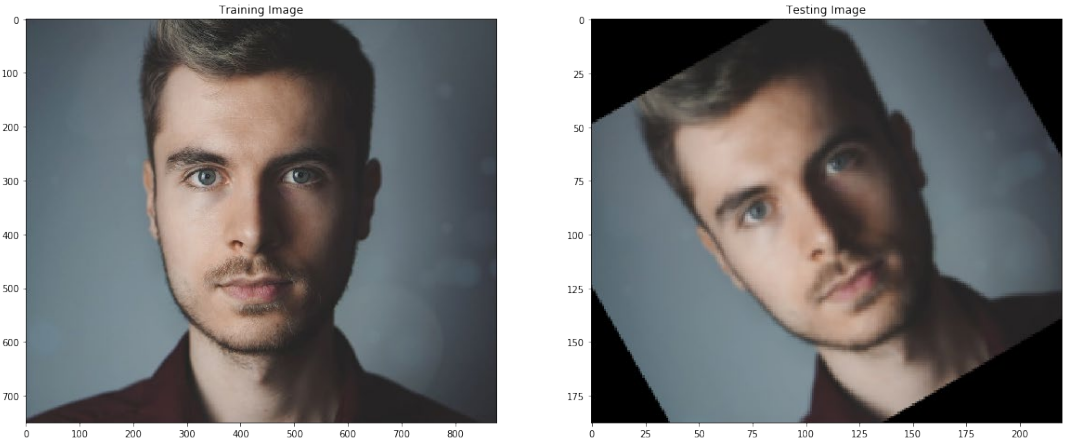

Figura 35 – Exemple de test

Primul pas este de a avea un format de imagine astfel incat sa extragem punctele cheie si descriptorii din ea.

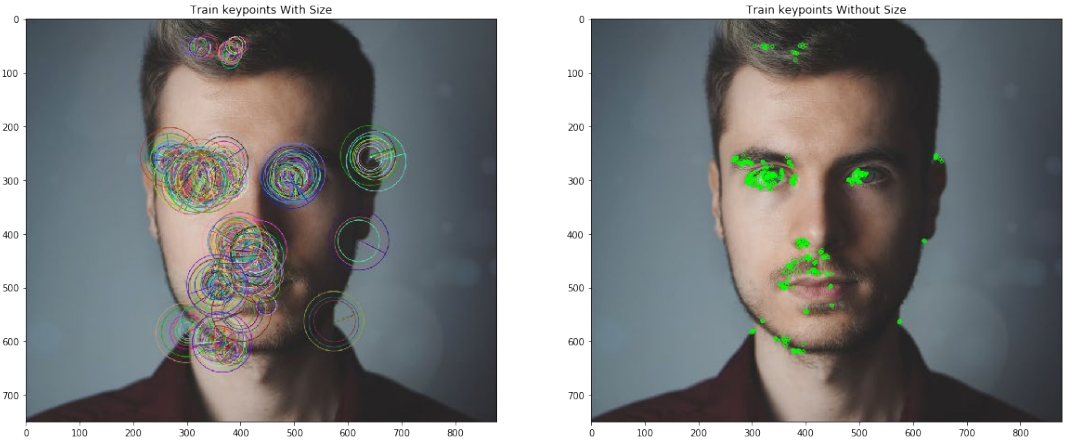

Figura 36 – Extragerea punctelor cheie

In cel de-al doilea pas este vorba de potrivirea datelor astfel incat se creaza o instanta BFMatcher si se compara datele iar rezultatul fiind urmatorul.

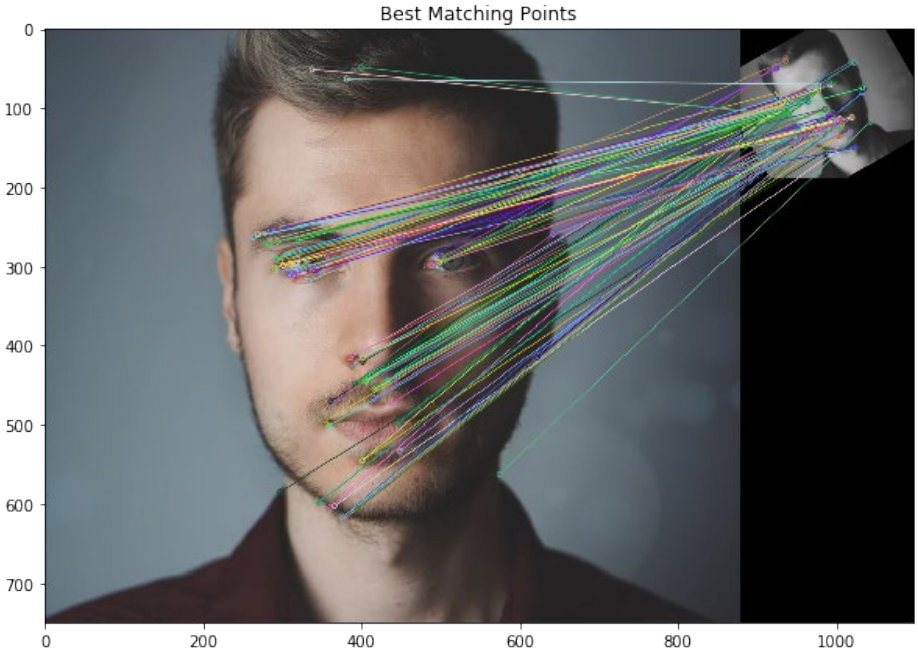

Figura 37 – Potrivirea datelor

Concluziile lucrarii

In concluzie aplicatia *Room detector* este o aplicatie de detectarea spatiilor realizata cu tehnologii precum procesarea de imagini si realitatea augmentata si poate fi vazuta ca o aplicatie de baza pentru multe alte solutii inovatoare. Aceste solutii sunt atat din domeniul electrocasnicelor, dar cel mai important in domeniul medical, pentru ajutarea oamenilor aflati in dificultate. In acest sens aplicatia *Room detector* poate fi extinsa pentru a crea aplicatii pentru oamenii orbi sau pentru oameni cu dizabilitati care folosesc un scaun cu rotile.

Am ales sa dezvolt aceasta aplicatie bazandu-ma pe cele mai noi tehnologii, astfel incat aplicatia este capabila sa se adapteze oricand la noi module cat si la alte tipuri de detectari. Aceasta aplicatie este construita cu ajutorul platformei Unity, avand astfel acces la cel mai nou, flexibil si complex framework, AR Foundation. Cu ajutorul acestui framework, avem acces la realitatea augmentata atat pe aplicatiile cu Android cat si la aplicatiile cu iOS. In aplicatia *Room detector*, framework-ul AR Foundation este folosit pentru preluarea cadrului de pe camera foto a device-ului.

Aceasta aplicatia a inceput ca un proiect de cercetare, ajungand in acest moment cu cea mai buna si rapida abordare pe care am putut sa o gasesc utilizand algoritmul *Oriented FAST and rotated BRIEF (ORB)*. Pana sa ajung la aceasta solutie am trecut printr-o perioada de cercetare si de testare.

Din punct de vedere a imbunatatirilor, as putea sa precizez ca pe viitor s-ar putea integra OpenCV-ul direct in aplicatie, astfel incat sa nu mai fim nevoiti sa avem o conexiune constanta la internet. Daca aceasta abordare nu se poate realiza sau nu se dovedeste a fi eficienta, putem optimiza partea de algoritmi de pe server astfel incat toate calculele sa fie realizate pe placa video (GPU), avand sansa astfel sa adaugam cat mai multe camere si spatii mari.

Consider ca aplicatia *Room detector* este o aplicatie care poate aduce multe beneficii asupra noilor tehnologii precum AR Cloud. AR Cloud-ul este o harta a lumii 3D, suprapusa peste lumea reala, permitand augmentarea, partajarea si legarea informatiile si experientelor la anumite locatii fizice. Misiunea AR Cloud-ului este de a conduce dezvoltarea tehnologiei de calcul spatial, a datelor si a standardizarii pentru conectarea lumilor fizice si digitale in beneficiul tuturor.

Bibliografie

1. Unity engine - 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. What is CUDA? - https://blogs.nvidia.com/blog/2012/09/10/what-is-cuda-2/
11. CUDA - https://opencv.org/platforms/cuda/
12. OpenCL - https://opencv.org/opencl/
13. Wait for end of frame - https://docs.unity3d.com/ScriptReference/WaitForEndOfFrame.html
14. Wait for fixed update - https://docs.unity3d.com/ScriptReference/WaitForFixedUpdate.html
15. Wait for seconds - https://docs.unity3d.com/ScriptReference/WaitForSeconds.html
16. Wait for seconds realtime - https://docs.unity3d.com/ScriptReference/WaitForSecondsRealtime.html
17. Wait until - https://docs.unity3d.com/ScriptReference/WaitUntil.html
18. Wait while - https://docs.unity3d.com/ScriptReference/WaitWhile.html
19. Unity Coroutines - https://www.tutorialspoint.com/unity/unity_coroutines.htm
20. How to use a coroutines? - https://vasundharavision.com/blog/Unity/how-to-use-coroutines-in-unity
21. Singleton - https://csharpindepth.com/articles/singleton
22. Introduction to ORB (Oriented fast and rotated brief) - https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf
23. Brute force matcher (BFmatcher) - https://docs.opencv.org/3.4/dc/dc3/tutorial_py_matcher.html

24. Accesing the Camera Image on the CPU https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@1.0/manual/cpu-camera-image.html?_ga=2.217778326.100917793.1592813107-1319051932.1570540714
25. Feature matching - http://sword-work-special-offer.top/2614016cgrr/feature-matching.html
26. Open AR Cloud - https://www.openarcloud.org/
27. AR Cloud Immersive Technology - https://www.foundry.com/insights/vr-ar-mr/ar-cloud-immersive-technology