UNIVERSITATEA BABEȘ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

LUCRARE DE LICENȚĂ

Sistem de navigare indoor și de detecție a obiectelor

Conducător științific:

Conf. Univ. Dr. Chira Camelia

Absolvent:
Pungă Alexandru

2021

Abstract

Time is man's most important resource, so losing it is a real problem. In everyday life we can reach environments unknown to us, making it difficult to orient ourselves and find what we need. In such situations, quite a lot of time can be lost.

While for outdoor environments GPS is a viable solution for locating and navigating, when it comes to indoor environments it has a very low accuracy due to the lack of signal. Over the years, many methods of navigating indoor environments that do not use GPS have been tried. These include methods using sensors, infrared, radio technologies and computer vision techniques. The best method in terms of the minimum equipment required and scaling the solution is the one that uses computer vision.

Abstracting the problem, it all comes down to internal navigation and object recognition. The proposed solution consists of a system that allows a user to choose the items he wants to find from a list. This project consists of three subprojects: one for identifying objects, one for reading QR codes and navigation, and one that manages the list of items. The subproject for reading QR codes and navigation is integrated with the help of the Google Vision library. This library has been adapted to manage the codes read according to the stage of the application. The QR codes read are parsed, and the information extracted from them is used to obtain data about a location from the back-end of the system or to navigate from one point to another. The subproject that helps to identify the objects is an adaptation of the TensorFlow lite library for detection. This part of the application receives a list of objects that will need to be recognized and requires user intervention to move from one activity to another. The last subproject is the one responsible for creating a list of elements, this being the basis and also the starting point of the application.

All presented subprojects create the front-end of the system which is in the form of a native android mobile application. These projects combined are capable to save a lot of valuable time in element searching.

Abstract

Timpul este cea mai importantă resursă a omului, astfel pierderea lui este o adevărată problemă. În viața de zi cu zi, un idivid poate ajunge în medii necunoscute lui, fiindu-i greu să se orienteze și să găsească cele necesare. În astfel de situații, se poate pierde destul de mult timp.

Dacă pentru mediile exterioare GPS este o soluție viabilă pentru localizare și navigare, în mediile interioare acesta are o acuratețe foarte scăzută datorită lipsei semnalului. De-a lungul anilor s-au încercat numeroase metode de navigare în medii interioare care nu utilizează GPS. Printre acestea se numară metode ce utilizează senzori, infraroșu, tehnologii radio sau tehnici de viziune computerizată. Cea mai bună metodă din punct de vedere al minimului de echipament necesar și al scalării soluției este cea care utilizează computer vision.

Abstractizând problema, totul se reduce la navigare internă și la recunoaștere de obiecte. Soluția propusă constă într-un sistem care permite unui utilizator să își aleagă dintr-o listă elementele pe care dorește să le găsească. Acest sistem este alcătuit din trei subproiecte: unul pentru identificarea de obiecte, unul pentru citire de coduri QR și navigare, și unul care gestionează lista de elemente. Subproiectul destinat citirii de coduri QR și navigării este integrat cu ajutorul librăriei Google Vision. Această bibliotecă a fost adaptată pentru a gestiona codurile citite în funcție de etapa în care se află aplicația. Codurile QR citite sunt analizate, iar informațiile extrase din ele sunt folosite pentru a obține datele despre o locație de la back-end-ul sistemului sau pentru navigarea de la un punct la altul. Subproiectul ce ajută la detectarea obiectelor este o adaptare a librăriei TensorFlow Lite pentru detecție. Această parte a aplicației primește o listă de obiecte ce trebuiesc detectate și solicită intervenția utilizatorului pentru a trece de la o activitate la alta. Ultimul subproiect este cel responsabil de crearea unei liste de elemente, acesta fiind baza și punctul de start al aplicației.

Toate aceste subproiecte creează front-end-ul sistemului care este sub forma unei aplicații mobile native de Android. Aceste subproiecte combinate sunt capabile să salveze timpul prețios pierdut in momentul când se caută anumite obiecte

Cuprins

1 Introducere ..... - 6 -

1.1 Motivația ..... - 7 -

1.2 Principalele obiective ..... - 8 -

1.3 Structura tezei ..... - 9 -

2 Definiții și concepte ..... - 10 -

2.1 Introducere în inteligența artificială ..... - 10 -

2.2 Introducere în algoritmica grafurilor ..... - 13 -

3 Metode și abordări ale navigării indoor și ale detecției de obiecte ..... - 16 -

3.1 Rutarea indoor cu ajutorul computer vision ..... - 16 -

3.1.1 Metode bazate pe markeri ..... - 17 -

3.1.2 Metode bazate pe caracteristici extrase din poze ..... - 18 -

3.1.2.1 Algoritmi ce extrag caracteristici ce ies în evidență ..... - 18 -

3.1.2.2 Algoritmi ce augmentează proprietăți ale imaginilor ..... - 19 -

3.2 Recunoașterea de obiecte dintr-o imagine ..... - 22 -

3.2.1 Sarcini de recunoaștere a obiectelor ..... - 22 -

3.2.1.1 Clasificarea de imagini ..... - 22 -

3.2.1.2 Localizarea obiectelor ..... - 23 -

3.2.1.3 Detecția de obiecte ..... - 23 -

3.2.2 Algoritmi de recunoaștere de obiecte ..... - 25 -

3.2.2.1 Familia de algoritmi R-CNN ..... - 25 -

3.2.2.2 Familia de algoritmi YOLO ..... - 27 -

3.2.2.3 MobileNet SSD ..... - 29 -

4 Abordare propusă ..... - 32 -

4.1 Navigarea în spații închise ..... - 32 -

4.2 Identificarea de obiecte ..... - 35 -

4.3 Back-end ..... - 36 -

4.4 Front-end ..... - 41 -

5 Arhitectura și testarea sistemului ..... - 48 -

5.1 Arhitectura de proiectare mobile ..... - 48 -

5.2 Arhitectura software a sistemului ..... - 49 -

5.2.1 Modelul funcțional ..... - 50 -

5.2.2 Modelul dinamic ..... - 51 -

5.2.3 Modelul obiectual ..... - 52 -

5.3 Testarea sistemului ..... - 56 -

6 Concluzii și direcții viitoare ..... - 60 -

6.1 Principalele contribuții ..... - 60 -

6.2 Direcții viitoare ..... - 61 -

Capitolul 1

Introducere

Încă de la începuturile civilizației, granițele popoarele s-au modificat datorită războaielor sau a condițiilor precare de trai. Timpului a arătat faptul că oamenii au fost interesați să călătorească, să viziteze, să cunoască. Indiferent de perioadă sau zonă, cartografierea regiunilor a fost o necesitate pentru multe domenii precum transport, comerț, turism, etc.

Odată cu primele cercetări ale spațiului cosmic au fost lansați și primii sateliți pentru a servi ca bază a serviciului Global Positioning System (GPS). În anul 1960 sistemul a fost disponibil și complet funcțional pentru armata Americană. Anul 1983 este însă unul de referință în domeniul localizării globale, serviciul GPS devine accesibil pentru prima dată în scopuri civile, având însă o eroare a acurateții de până la 100 yarzi (91.44 m). În anul 2000 această eroare a fost redusă până la 10 yarzi (9.14 m) ceea ce a încurajat dezvoltarea domeniului și crearea aplicațiilor și sistemelor de rutare așa cum le cunoaștem în ziua de astăzi [13].

Cu toate că tehnologiile bazate pe poziționarea și rutarea cu ajutorul serviciului GPS au evoluat foarte mult, acestea pot fi utilizate corespunzător doar în mediile exterioare. În mediile interioare (clădiri de birouri, mall-uri, spitale, stații de tren, etc.), GPS nu este o soluție viabila deoarece semnalul captat poate fi slab din cauza pereților și al tavanului, iar acuratețea determinării poziției poate scădea drastic [12].

Aplicațiile de rutare în aer liber necesită conexiune la internet pentru încărcarea hărții, și semnal GPS pentru localizare. În aplicațiile de navigație internă, însă nu este atât de simplu. În lipsa unui semnal GPS corespunzător s-au încercat numeroase metode de

localizare: bazate pe senzori [24], bazate pe infraroșu [5], bazate pe tehnologii radio [19] și pe numeroase tehnici de viziune computerizată (exemplu: markeri vizuali [6]).

Fiecare dintre aceste metode prezintă avantaje și dezavantaje. În general metodele care folosesc senzori, infraroșu sau tehnologii radio au marele dezavantaj că necesită aparatură adițională pentru a permite maparea și/sau navigarea, astfel fiind dificil de adaptat la alte medii. Metodele de computer vision, deși nu au neapărată nevoie de echipamente adiționale, în afara unui telefon sau a unui calculator, pot suferi în timp dacă mediul se schimbă. Pe lângă acest dezavantaj modelele rezultate de computer vision pot fi complexe, fiind greu de antrenat. Cu toate acestea, aplicațiile de navigare internă încep să ia amploare, fiind ușor de adaptat de la o zonă la alta odată ce modelul antrenat este gata.

Dintre toate abordările prezentate cea pe care o tratez în prezenta lucrare este bazată pe computer vision.

Scopul final al acestui studiu este de a ajuta la crearea unei rute în găsirea celui mai scurt traseu între anumite obiecte, fiecare dintre ele fiind situate într-o încăpere/regiune.

Rutarea are loc între camere/regiuni. Ajuns în zona indicată se va căuta pe rând câte un obiect care se știe că se află în regiunea respectivă. În momentul în care obiectul va fi găsit, va fi înconjurat de un chenar de o anumită culoare.

Acest mod de abordare al navigării interne combinat cu separarea datelor despre obiecte și localizarea lor, poate avea aplicații în diverse domenii. În comerț poate ajuta la minimizarea timpului petrecut în magazine (hypermarket-uri în general) și fluidizarea fluxului de oameni de la o anumită oră. În logistică poate fi folosit în depozite cu scopul de a ține evidența mai ușor a locațiilor produselor.

1.1 Motivația

Principala motivație a realizării proiectului a plecat de la o problemă reală întâmpinată de o mulțime de oameni, printre care pot spune că mă aflu și eu. Această problemă se manifestă prin timpul pierdut la magazine în vederea găsirii celor dorite. Se știe că adesea magazinele, în special hypermarket-urile, schimbă poziția diferitor produse în funcție de oferte, de produsele cele mai cumpărate într-o anumita perioada a anului și,

probabil, în funcție de multe alte criterii. Pentru un cumpărător acest fapt poate fi frustrant deoarece modificarea poziției unui produs poate însemna timp pierdut pentru găsirea lui. Pe lângă acest fapt se pot lua în considerare și cumpărăturile făcute în diferite spații comerciale a cărei structură cumpărătorul nu o cunoaște. Cel mai bun exemplu este atunci când o persoana merge la un magazin într-un oraș diferit. În oricare dintre situații s-ar afla, un cumpărător poate pierde de la câteva minute, la câteva ore.

1.2 Principalele obiective

Știind că timpul este poate cea mai importantă resursă din viața unei persoane am constatat că problema descrisă în subcapitolul de motivație este reală. Astfel, abstractizând situația, am ajuns la o problema de navigare internă și de recunoaștere de obiecte.

În linii mari contribuția adusă stă în atingerea obiectivului de a crea un sistem adresat dispozitivelor mobile cu sistem de operare Android, care are rolul de a găsi într-o locație cu mai multe camere, elemente selectate dintr-o listă. Acest sistem este alcătuit din două părți:

- Partea necesară navigării pâna la o anumită zonă. După o analiză a metodelor existente de navigare, am ales ca cea mai ușor de implementat, cea mai scalabilă, cea care utilizează cele mai puține resurse și are nevoie de cel mai scurt timp pentru a obține poziția la un anumit moment este metoda care folosește markeri de tip QR code. Aceasta nu este neapărat cea mai precisă metodă, dar utilizând suficient de mulți markeri așezați în locații-cheie ale zonei de interes, se îmbunătățește performanța.

- Partea necesară detecției de obiecte. Acest subproiect al sistemului utilizează un model pre-antrenat MobileNet SSD preluat și adaptat din biblioteca TensorFlow Lite. În urma analizei algoritmilor descriși în capitolul trei, am constat că cele mai bune opțiuni de algoritmi pentru recunoaștere de obiecte pe dispozitive mobile sunt YOLO si MobileNet SSD. Alegerea algoritmului MobileNet SSD s-a datorat vitezei și ratei de detecție ce sunt mai mari comparativ cu algoritmul YOLO [36].

1.3 Structura tezei

Identificarea unei soluției a fost posibilă datorită urmării unor pași: documentare în vederea găsirii unor soluții pentru rezolvarea întregii probleme, împărțirea problemei în subprobleme și documentarea fiecărei părți, crearea părții de back-end a aplicației pentru a gestiona date, aducerea datelor în aplicația de front-end, utilizarea unui algoritm de grafuri pentru a găsi cel mai scurt drum necesar identificării elemntelor, integrarea și adaptarea unei biblioteci de citire de coduri QR în vederea navigării intre zone ale aceleiași locații și utilizarea altei biblioteci pentru sarcina de recunoaștere de obiecte.

Pașii urmați în vederea identificării soluției sunt detaliați in prezenta lucrare. Acestă teză este structurată în șase capitole:

- Capitolul 1 - prezintă problema și motivația pentru care am ales să găsesc o soluție;
- Capitolul 2 - conține noțiunile de bază folosite atât în abordările deja existente cât și in propria abordare pentru rezolvarea problemei;
- Capitolul 3 - înglobează unele din metodele existente pentru navigarea în spații indoor și pentru detecția de obiecte;
- Capitolul 4 - oferă informații despre abordarea proprie în vederea găsirii unei soluții;
- Capitolul 5 - prezintă arhitectura sistemului creat si testarea lui;
- Capitolul 6 - concluzioneaza abordarea proprie si oferă sugestii pentru continuarea dezvoltării sistemului.

Capitolul 2

Definiții și concepte

În acest capitol sunt prezentate concepte de baza ale inteligenței artificiale și din algoritmica grafurilor folosite în soluția propusă sau pentru înțelegerea capitolului ce prezintă stadiul actual.

2.1 Introducere în inteligența artificială

„Inteligența artificială (AI) este o ramură largă a informaticii care se ocupă cu construirea de mașini inteligente capabile să îndeplinească sarcini care necesită de obicei inteligență umană” [3].

Acest domeniul al informaticii activează încă din anii 1950 de la publicația lui Alan Turing „Computing Machinery and Intelligence”, în care au fost descrise obiectivele fundamentale și viziunea lui asupra inteligenței artificiale.

Odată cu trecerea timpului metodele considerate inteligente au evoluat devenind tot mai numeroase, iar sarcinile au putut fi împărțite in șase mari ramuri: sisteme expert, roboți, învățare automată, deep learning, fuzzy logic si procesarea limbajului natural.

În prezenta lucrare se folosește inteligența artificială pentru a detecta obiecte și pentru a citi coduri QR. Aceste sarcini fac parte din categoria vederii sau viziunii computerizate (computer vision) a ramurii de roboți din cadrul inteligenței artificiale.

Primele aplicații ale vederii computerizate datează de la începutul anilor 1960. Tot în acea perioadă părintele viziunii computerizate, Larry Roberts, discuta în teza sa de doctorat la MIT posibilitatea extragerii formelor geometrice 3D din perspective 2D [33].

În prezent computer vision poate îndeplini cu succes sarcini precum, navigare asistată, procesarea imaginilor 2D, recunoașterea de obiecte, recunoașterea de persoane și a fețelor acestora și multe altele.

Computer vision se bazează pe metode și algoritmi de extragere a caracteristicilor dintr-o imagine. Unele dintre aceste metode sunt mașinile cu suport vectorial (SVM) și rețelele neuronale artificiale (ANN).

Rețelele neuronale reprezintă poate cea mai revoluționara metoda a inteligentei artificiale, fiind inspirată din neuronii biologici.

Creierul uman este un instrument complex de calcul, fiind capabil să proceseze în paralel informații. Fiecare neuron deține aproximativ 10000 de conexiuni.

Neuronul este unitatea de bază a sistemului nervos fiind alcătuit din corp celular (Soma), axoni si dendrite. Neuronii sunt legați intre ei prin intermediul sinapselor.

Oamenii de știință au găsit o similaritate între rețeaua neuronală biologică și cea artificială, identificând termeni ce pot fi puși în comun. Corpul celular poate fi reprezentat ca un nod, dendritele pot fi considerate ca fiind date de intrare ce urmează a fi procesate, axonii pot fi interpretați ca și date de ieșire ce au fost procesate, activarea neuronilor are corespondent cu procesarea în sine, iar sinapsa poate fi considerată o conexiune ponderată. Conexiunile dintre neuroni sunt cunoscute ca ponderi sinaptice și sunt folosite în stocarea informației.

După ce s-a procesat local informația pe baza datelor stocate în legăturile sinaptice, se multiplică cu valoarea unei ponderi stocate apoi se însumează global toate operațiile de acest fel la nivelul unui neuron. Aceste elemente compun structura generală a unui neuron așa cum arată și Figura 2.3

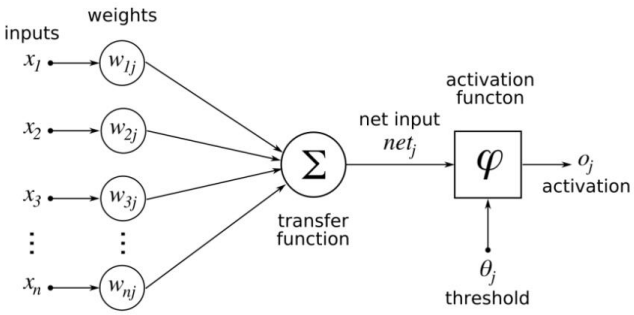

Fig. 2.1 Structura generală a unui neuron artificial

Pentru a proiecta o rețea neuronală este nevoie de cinci componente:

- n noduri de intrare
- m noduri de ieșire
- specificare straturilor ascunse și a numărului de noduri conținut de fiecare
- inițializarea ponderilor
- stabilirea a câte o funcție de activare corespunzătoare fiecărui nod

Un tip mai complex de rețele neuronale este reprezentat de rețelele neuronale convoluționale (CNN). Aceste rețele fac parte din categoria rețelelor neuronale profunde (deep neural network). Diferența dintre aceste rețele și rețelele neuronale obișnuite este aceea că datele de input sunt sub forma unei imagini. Straturile de neuroni din cadrul CNN au o structură aparte fiind pe trei dimensiuni: lățime, înălțime, adâncime.

Straturile convoluționale sau „*convolutional layer*”, receptează o parte dintr-o imagine de o anumita dimensiune, cu scopul de a extrage caracteristici ce pot ajuta la clasificare. Aceste straturi produc astfel unul sau mai multe imagini numite hărți de trăsături sau „*feature maps*”.

La finalul unei convoluții caracteristicile extrase din regiunea unde a fost aplicată se comprimă. Acest proces se numește *downsample*. După comprimare se poate aplica din nou o convoluție.

Pe lângă straturile convoluționale mai apare și denumirea de straturi de agregare sau „*pooling layer*”. Acest strat micșorează dimensiunea imaginii de input comprimând output-ul unui strat convoluțional.

Asupra stratului de convoluție și a stratului de agregare se aplică o funcție de activare pentru a asigura un comportament neliniar al rețelei. Cea mai întâlnită funcție de

activare este ReLU (Rectified Linear Unit). Funcția returnează 0 dacă primește o valoare negativă sau valoarea primită dacă aceasta este mai mare decât 0.

O componentă se poate numi complet conectată sau „*fully connected*” dacă efectuează și clasificarea. Această componentă are forma unei rețele neuronale clasice căreia i se furnizează ca date de input hărțile de trăsături. La output-ul acestei rețele se aplica funția softmax reprezentata in figura 2.2.

Funcția softmax transpune valorile unui vector cu numere reale în valori din intervalul (0,1). Acest fapt se realizează pentru a nu avea discrepanțe mari între date.

$$output\_neuro(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

Fig. 2.2 Funcția softmax

Toate componentele prezentate mai sus fac parte din arhitectura CNN [32] și se pot vedea în Figura 2.3.

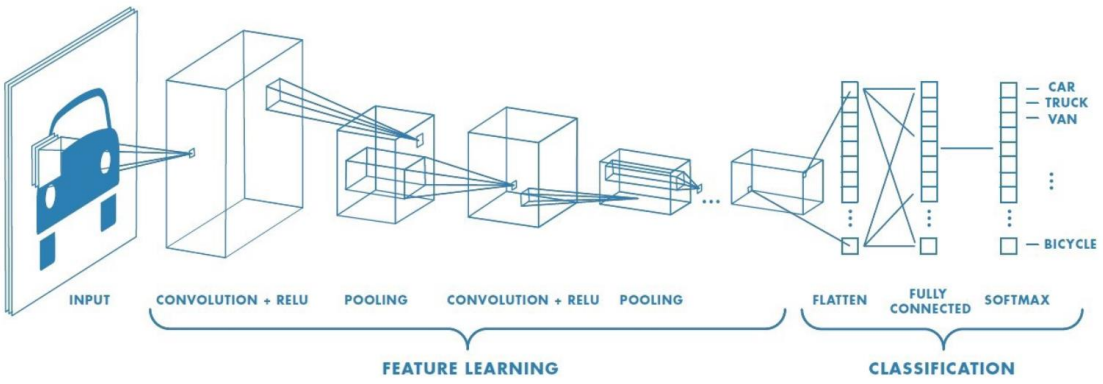

Fig. 2.3. Arhitectură CNN.

2.2 Introducere în algoritmica grafurilor

Grafurile reprezintă una dintre cele mai puternice metode de a modela date. Grafurile sunt implementate în diferite aplicații precum social media, pagini web (linkuri), chiar și pentru localizare (rute GPS).

Una dintre cele mai întâlnite probleme în grafuri este găsirea celui mai scurt drum de la un nod la altul având aplicații în mai toate domeniile. Prezenta problemă poate data încă de la începuturile timpului sub o formă sau alta. Această problemă, în viața de zi cu zi, poate fi formulată ca identificarea celui mai scurt drum pentru a ajunge în toate punctele unde este nevoie să ajungă o persoană într-o zi. Această problemă mai este cunoscută ca și problema comisului-voiajor sau Travelling salesman problem.

Există o mulțime de algoritmi care încearcă să rezolve această problemă. Pentru a obține soluția necesara am ales un algoritm cunoscut ca algoritmul lui Dijkstra.

Algoritmul Dijkstra calculează distanța dintre un nod și toate celelalte noduri dintr-un graf.

Pașii urmați de acest algoritm sunt :

- Se alege un nod de start;
- Se notează distanța minimă a fiecărui nod către nodul ales. Inițial nodul ales va avea distanța minimă 0, iar toate celelalte vor avea distanța un număr cat mai mare;
- Vecinii nodului selectat primesc pe rând minimul distanței dintre acestea și distanța deja salvată;
- Odată ce s-a verificat și după caz actualizat distanța vecinilor nodului curent, acesta se consideră vizitat;
- Se alege un alt nod de start. Acest nod nu trebuie să nu fi fost vizitat și csa aibă distanța minimă către nodul selectat inițial;
- Algoritmul se repetă până toate nodurile grafului sunt marcate ca fiind vizitate.

Acești pași sunt reprezentați in figura 2.4.

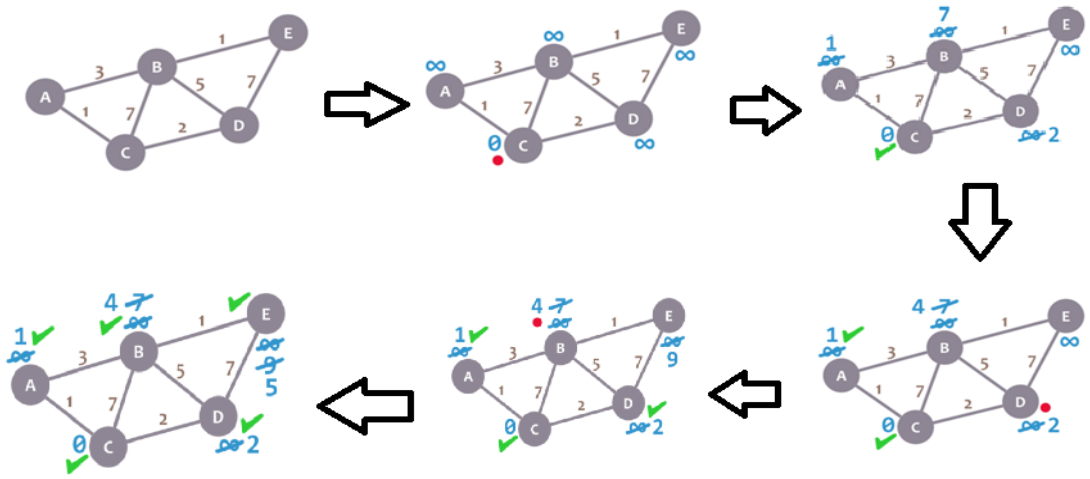

Fig. 2.4 Exemplu Dijkstra [1]

Capitolul 3

Metode și abordări ale navigării indoor și ale detecției de obiecte

Din abstractizarea problemei prezentată în capitolul de introducere se poate extrage o soluție bazată pe navigarea în spații închise și pe detectarea de obiecte din acesta.

Cele mai folosite metode apărute în literatură pentru navigarea în spații închise sunt cele bazate pe senzori [24], pe infraroșu [5], pe tehnologii radio [19] sau pe numeroase tehnici de viziune computerizată. Analizând aceste metode am concluzionat că cele mai bune modele, din punct de vedere al scalabilității și al lipsei echipamentelor adiționale necesare, sunt cele bazate pe computer vision.

Pentru sarcina de detecție am analizat câțiva algoritmi constatând că cei mai potriviți pentru dezvoltarea unui sistem ce urmează să ruleze pe un dispozitiv mobile sunt YOLO și MobileNet SSD. Dintre acestea doua, algoritmului ales a fost MobileNet SSD datorat vitezei și ratei de detecție ce sunt mai mari comparativ cu algoritmul YOLO [36].

3.1 Rutarea indoor cu ajutorul computer vision

În linii mari metodele care folosesc computer vision se pot împărți în două categorii:

- Metode care utilizează markeri pentru a identifica locația curentă
- Metode care se bazează pe caracteristici preluate din imagini

3.1.1 Metode bazate pe markeri

Markerii vizuali pot fi și ei o opțiune bună pentru localizare. Aceștia pot fi reprezentați sub diferite forme precum poze, tablouri, etc. Markerii care se recunosc cel mai bine sunt cei sub forma unor coduri QR și bar code, datorită formelor geometrice unice și al contrastului. În unele cazuri chiar și componente fizice din mediu pot fi văzute ca și markeri. Pentru ca un obiect să poată fi folosit pe post de marker trebuie să aibă suficient de multe trăsături vizuale unice. Această tehnică de poziționare impune așezarea strategică a markerilor astfel încât să ocupe întreaga suprafață. Dezavantajul acestei metode constă în faptul că markerii trebuie plasați fizic în mediu.

Abordările bazate pe markeri, mai exact markerii fiduciali, sunt folosite, în general, deoarece prezintă un cost computațional mai mic decât metodele non-vizuale și pot fi ușor de decodata chiar și de un smartphone. Conform *Klopschitz si Schmalstieg* [20], localizarea poziției unui utilizator se realizează cu succes când se pot utiliza suficient de mulți markeri. Informațiile legate de locație sunt încărcate când markerii sunt scanați. Această implementare necesită ca, înaintea scanării, markerii să fie printați și să fie poziționați pe pereți asemănător cu Figura 3.1.

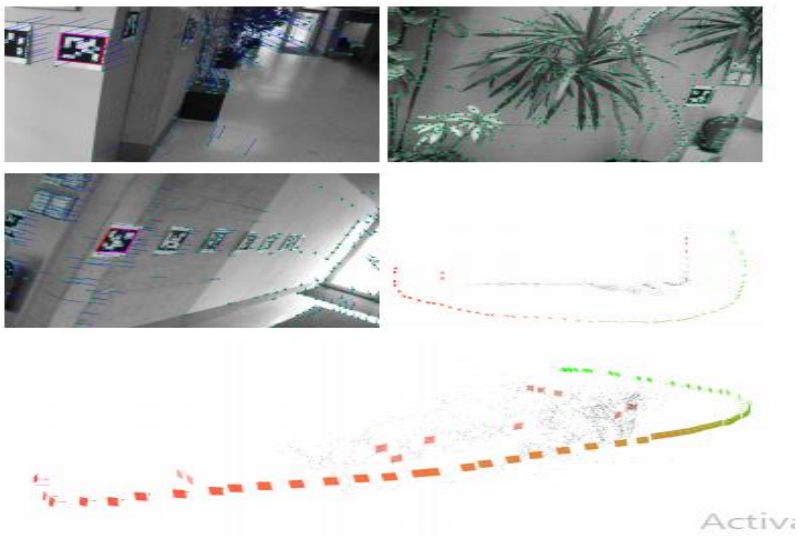

Fig. 3.1 Markeri fiduciali printați și puși pe pereți. Aceștia ajută la construirea unui model 3D [20].

3.1.2 Metode bazate pe caracteristici extrase din poze

În ultimii ani au fost dezvoltate o mulțime de tehnici de analiză a imaginilor în domeniul computer vision. În studiul lui H. Kawaji [14] aceste tehnici sunt împărțite în două grupuri. Primul este cel al algoritmilor care extrag caracteristici ce ies în evidență (diferențe mari de contrast). Acest grup prezintă o stabilitate mai mare față de rotația și scalarea imaginilor. Printre dezavantajele acestei tehnici se pot enumera: nevoia de putere mare de calcul pentru a compensa mărimea bazei de date, numărul mare de operații, creșterea timpului de așteptare al aplicației până la soluționarea analizei și recunoașterea de obiecte sau locații care au caracteristici similare. Cel de-al doilea grup prezentat în studiu este cel al algoritmilor ce augmentează proprietăți ale imaginilor (colțurile, marginile, etc. ale obiectelor din poză). Aceste tipuri de algoritmi pot avea probleme daca mediul suferă schimbări.

3.1.2.1 Algoritmi ce extrag caracteristici ce ies în evidență

În general algoritmii care fac parte din această grupă folosesc diverse metode de computer vision pentru a estima poziția cu ajutorul imaginilor dintr-un mediu bine cunoscut. Unii dintre cei mai cunoscuți algoritmi ai grupei sunt SIFT (Scale-Invariant Feature Transform) și SURF (Speeded Up Robust Features).

SIFT este un algoritm ce extrage caracteristicile unei poze folosind o metoda bazată pe neclaritatea Gaussiană și pe colectarea punctele de extrem local. Pentru fiecare punct de interes un descriptor de rotație este calculat. Distanța Euclidiană dintre descriptori oferă atât recunoașterea de caracteristici cât și a obiectelor și imaginilor.

SURF este asemănător cu SIFT având însă aproximări mai puțin precise, dar mai rapide în oferirea de puncte de extrem. Avantajul acestui algoritm se datorează vitezei de extragere a punctelor de extrem, folosind memoria foarte eficient astfel încât randamentul este unul ridicat. Această metodă poate fi folosită chiar și pe dispozitivele mobile.

În acest domeniu al poziționării interne au fost încercate multe alternative printre acestea fiind cea prezentată în studiul făcut de H. Kawaji, de K. Hatada, de T. Yamasaki, și de K. Aizawa [14] ce au folosit imagini omni-direcționale cu o variantă a algoritmului SIFT. Algoritmul a funcționat bine în spații închise cu o arie mare, dar precizia a depins foarte mult de densitatea imaginilor panoramice, deoarece nu s-au aplicat și alte metode de corectare a poziției.

3.1.2.2 Algoritmi ce augmentează proprietăți ale imaginilor

Această grupare de algoritmi se utilizează de obicei în medii necunoscute unde SLAM (Simultaneous Localization and Mapping) este efectuată. Sistemul VSLAM (subcategorie a SLAM) construiește o hartă din imaginile preluate de cameră și extrage simultan puncte de caracteristici din imagine ce ajută la localizare[23].

SLAM este una dintre cele mai folosite tehnici în navigarea indoor, fiind prima dată folosită în domeniul vehiculelor autonome. Această tehnică încearcă să obțină date cum ar fi: Received Signal Strength și 3D Point Clouds cu scopul de a efectua dinamic în tandem maparea unei suprafețe și urmărirea poziției subiectului. Multe din tehnologiile de navigație internă (Wi-FI, Bluetooth, recunoaștere de imagini) pot conlucra cu SLAM [35]. Această abordare nu are nevoie de putere mare de calcul astfel încât este potrivită pentru dispozitivele mobile.

În esența, SLAM încearcă să adune date și repere despre un mediu necunoscut de la un agent mobil (robot) și să îl cartografieze. Acest mod de mapare poate avea două abordări principale: metoda bazată pe filtrare (filtering SLAM) și metoda bazată pe optimizarea grafului vizual (VisualSLAM sau VSLAM).

Metoda SLAM bazată pe filtrare a fost propusă pentru prima dată de R. Smith în anul 1986. Această abordare a modelat mediul cu ajutorul filtrelor Extended Kalman Filters (EKF), obținând poziția relativă dintre robot și repere, filtrând zgomotul Gaussian al datelor observate. Sistemul ia în calcul orientarea robotului, datele de input ale senzorilor și datele înregistrate în momentul curent utilizării pentru a deduce poziția robotului

utilizând estimarea Bayesiană. Aceasta este doar una din metodele folosite pentru implementarea unui sistem bazat pe filtrarea SLAM[38].

VisualSLAM a apărut în jurul anului 2004. Această metodă folosește imagini preluate de la camere și de la alți senzori pentru imagine. VSLAM poate utiliza în principiu orice fel de cameră. Implementarea se realizează cu un cost redus, camerele fiind relativ ieftine. Detectarea reperelor poate fi combinată cu optimizarea grafului vizual. Construcția grafului ajută la estimarea mișcării robotului prin schimbarea imaginii. VSLAM poate fi folosit în algoritmi precum: DTAM (Dense tracking and mapping), LSD-SLAM (Large-Scale Direct SLAM), DSO (Direct Sparse Odometry) și SVO (Semi-Direct Visual Odometry) [4].

Figura 3.2 este extrasă din studiul lui Manfred Klopschitz și al lui Dieter Schmalstieg [20] ce prezintă metoda punctelor extrase (points cloud) și estimări ale poziției camerei printr-o metoda SLAM vizuala (VSLAM). Punctele u, x si y reprezintă un punct de observație, poziția camerei, respectiv structura de puncte extrase. Se extrag și se pun în potrivire punctele asemănătoare (u) din imagini diferite. Poziția camerei (x) și structura punctelor (y) sunt identificate folosind optimizarea globală, minimizând distanța dintre imaginea prezisă și punctele de observații.

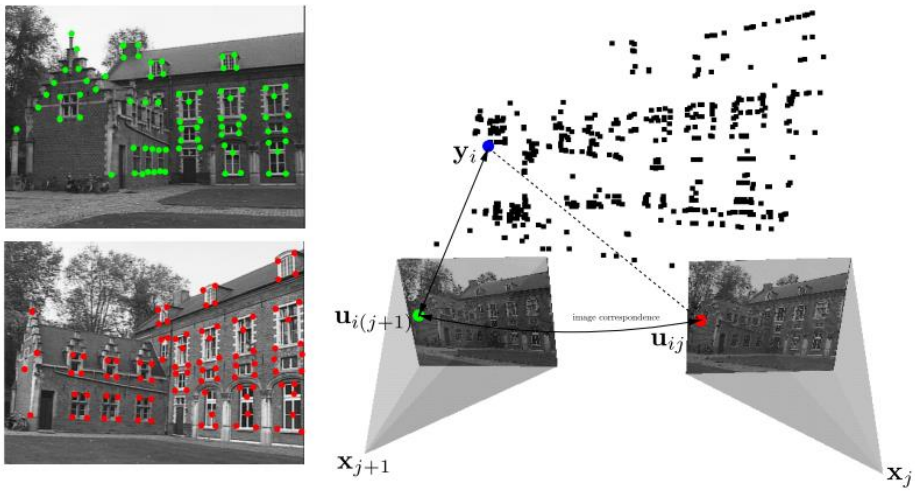

Fig. 3.2 SLAM vizual (VSLAM)

La fel ca și SLAM, metodele ce folosesc Point Cloud Pattern pot acoperi o arie mai mare și pot identifica locația mai ușor chiar dacă punctele de reper sunt mai depărtate de cameră. Identificarea punctelor distincte într-un mediu intern se realizează mai ușor decât

recunoașterea unei poze sau al unui marker. Dezvoltarea unui sistem de localizare bazat pe această metodă necesită o prescanare 3D a mediului. Pentru prescanarea suprafeței și localizarea ulterioară pot fi folosite diferite tool-uri precum Metaio SDK. Acest tool oferă posibilitatea de a suprapune elemente astfel fiind posibile construirea aplicațiilor AR în platforma android.

Studiul *„ Augmented Reality-based Indoor Navigation: A Comparative Analysis of Handheld Devices vs. Google Glass* ” [35] abordează o astfel de navigare indoor bazată pe AR. În acest studiu scanarea suprafeței s-a realizat utilizând Metaio Toolbox. Ca alegeri de design suprafața minimă de scanare a fost de 2 m pentru a nu apărea nicio diferență când informațiile de poziție bazate pe AR sunt suprapuse. Toate suprafețele au fost luate în considerare, mai utile fiind cele ce au avut o textură variată. Obiectele din aria studiului (scaune, mese, etc.) au fost scanate din diferite unghiuri. Pragul minim de caracteristici scanate dintr-o anumita zonă era de 1500. Zonele în care mediul continuă la stângă sau la dreapta erau scanate astfel încât acuratețea să fie mai mare. Procesul de scanare se încheia odată ce au fost identificate suficient de multe caracteristici pe un traseu.

Numărul de puncte scanate din fiecare locație poate afecta ușurința cu care este identificată fiecare locație. În cazul în care erau scanate prea puține puncte, algoritmul nu era capabil să furnizeze indicații. Dacă erau suficiente puncte, dar erau împrăștiate, utilizatorul trebuia să scaneze împrejurimile până când algoritmul identifica locația. În caz contrar, când sunt foarte multe puncte într-o locație timpul de identificare a poziției este foarte scurt. Chiar dacă într-o zona erau foarte multe puncte scanate, după ce se atingea pragul minim de identificare, restul nu mai erau luate în considerare. Numărul minim de puncte necesare pentru ca performanța sistemului să nu scadă semnificativ era de 500.

În figura 3.3 este reprezentată un preview al aplicației lui Rehman și Cao [34]. Informațiile sunt suprapuse folosind metoda 3D Point Clouds din diferite zone. Punctele au fost afișate numai în etapa de testare.

Fig. 3.3 Exemplu aplicație bazat pe metoda 3D Point Clouds

După ce traseele au fost scanate complet, au fost plasate într-o ordine secvențială. Următorul pas a fost adăugarea asistenței vizuale (prin săgeți) și auditive (prin instrucțiuni: „La următorul colt mergeți la dreapta”, etc.).

3.2 Recunoașterea de obiecte dintr-o imagine

Încă din anii 1950, de la primele experimente ce utilizau computer vision pentru a detecta marginile unui obiect sau pentru a-l clasifica în diferite categorii se poate observa înclinația domeniului către recunoașterea de obiecte[15].

Recunoașterea de obiecte este alcătuită din trei ramuri mari: clasificarea de imagini, localizarea obiectelor într-o imagine și detecția de obiecte[25].

3.2.1 Sarcini de recunoaștere a obiectelor

3.2.1.1 Clasificarea de imagini

Image classification sau clasificarea imaginilor ajută la repartizarea unui obiect din imagine într-o anumita categorie atribuindu-i o etichetă. Acest lucru se realizează prin

diferiți algoritmi ce compară caracteristicile extrase din poza curentă cu cele extrase dintr-un set de imagini cunoscute. Performanța unui model pentru clasificara de imagini constă în eroarea medie a clasificării între etichetele claselor prezise.

3.2.1.2 Localizarea obiectelor

Acest aspect al viziunii computerizate este folosit la localizarea diferitor obiecte dintr-o anumită categorie în imagine și încadrarea lor într-un chenar. În literatură „ *Object Localization*” are rolul de a găsi în imagine un obiect dintr-o anumită categorie. Performanța acestei tehnici constă în diferența dintre poziția chenarului așteptat și cel prezis.

3.2.1.3 Detecția de obiecte

Object detection reprezintă cea mai dificila sarcină dintre cele trei prezentate și este combinația dintre clasificarea imaginilor și localizarea obiectelor dintr-o imagine. Diferența între localizarea de obiecte și detecția obiectelor constă în faptul că localizarea de obiecte se axează pe detectarea unui singur tip de obiect în timp ce detectarea evidențiază obiecte din mai multe categorii. Performanța constă în analizarea acurateții, preciziei și a rapelului.

Figura 3.4 prezinta diferența dintre între localizarea și detecția obiectelor [25].

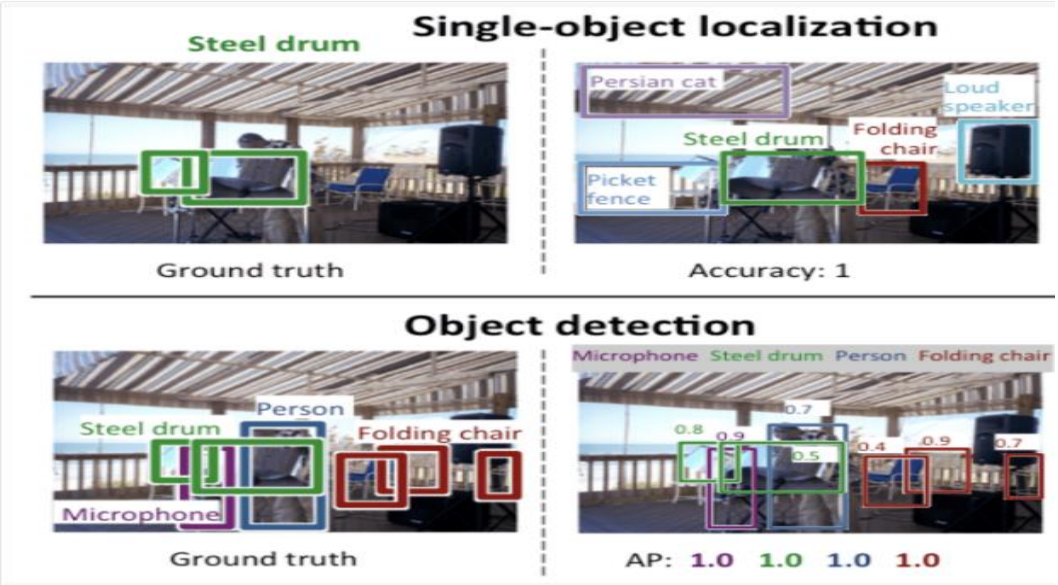

Fig. 3.4 Comparație între localizarea și detecția obiectelor[25]

Pe parcursul anilor, tehnicile de localizare, detecție și clasificare de imagini au fost abordate atât prin algoritmi de învățare supervizată cât și prin algoritmi de învățare nesupervizată. Câțiva dintre acești algoritmi de machine learning sunt ANN, SVM, K-MEANS, etc.

În ultimul deceniu, recunoașterea de obiecte a avut o îmbunătățire majoră, odată cu apariția bazei de date cu imagini și a competiției anuale create de ImageNet. Această baza de date oferă în jur de 22000 de categorii, având aproximativ 15 milioane de imagini etichetate[26]. În cadrul competiției lor se folosesc aproximativ 1000 de categorii de imagini. De la prima ediție a concursului până în prezent eroarea clasificării imaginilor a scăzut de la 28% la sub 3%, fiind mai bună chiar decât recunoașterea umană care are o eroare de 5%.

Mulți algoritmi de recunoaștere de obiecte au apărut în istoria recentă, fiecare având atât avantaje cât și dezavantaje. Printre cei mai cunoscuți algoritmi sau familii de algoritmi se enumeră: familii de algoritm R-CNN (Region-Based Convolutional Neural Network), familia de algoritmi YOLO (You Only Look Once) si algoritmul MobileNet SSD.

3.2.2 Algoritmi de recunoaștere de obiecte

3.2.2.1 Familia de algoritmi R-CNN

Algoritmul **R-CNN** reprezentat în figura 3.5 a fost propus prima dată de Ross Girshick în [30]. Modelul poate fi împărțit în trei regiuni: extragerea a aproximativ 2000 de regiuni de dimensiuni diferite din poză, colectarea de caracteristici folosind o rețea neuronală convoluțională și apoi clasificarea fiecărei regiuni utilizând SVM.

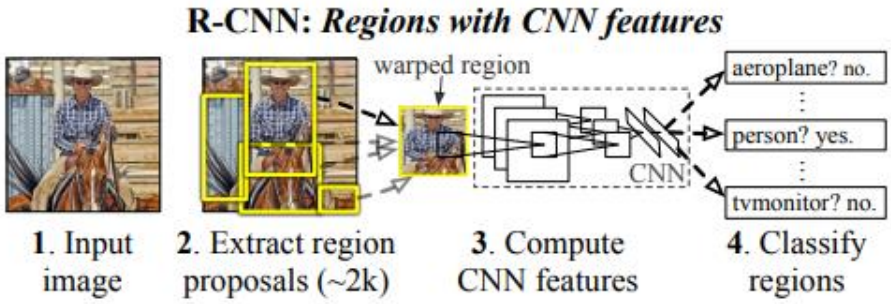

Fig. 3.5. Structura algoritmului R-CNN. [30]

Algoritmul **Fast R-CNN** a apărut în anul 2015 avându-l ca autor tot pe Ross Girshick. Acest algoritm s-a dezvoltat în cadrul unui studiu făcut de Microsoft [31]. Articolul începe prin prezentarea limitărilor algoritmului R-CNN. Autorul evidențiază trei probleme ale algoritmului anterior. Prima problemă este reprezentată însăși de împărțirea algoritmului în trei module distincte. A doua problemă constă în faptul că antrenarea este costisitoare din punct de vederea al timpului și al spațiului. Ultima problemă afirmă că extragerea caracteristicilor utilizând un CNN se realizează foarte încet datorită regiunilor numeroase.

Fast R-CNN reprezentat in figura 3.6. este introdus ca fiind alcătuit dintr-un singur modul care returnează regiunile și clasificările direct. Arhitectural în loc sa se folosească câte o rețea neuronală convoluțională pentru fiecare regiune, s-a ales ca toate regiunile să fie trimise printr-o singură rețea ca un întreg, împărțind matricea de caracteristici. Tot aceeași matrice de caracteristici se împărțea pentru clasificarea obiectelor și încadrarea lor într-un chenar. Prin urmare împărțirea puterii de calcul accelerează viteza algoritmului R-

CNN. Din matricea de caracteristici se propune o regiune de interes (RoI), din care se extrage câte un vector de caracteristici cu lungime fixă. Din acest vector vor rezulta două date de ieșire: probabilitatea softmax și informațiile legate de construirea chenarului în jurul obiectului.

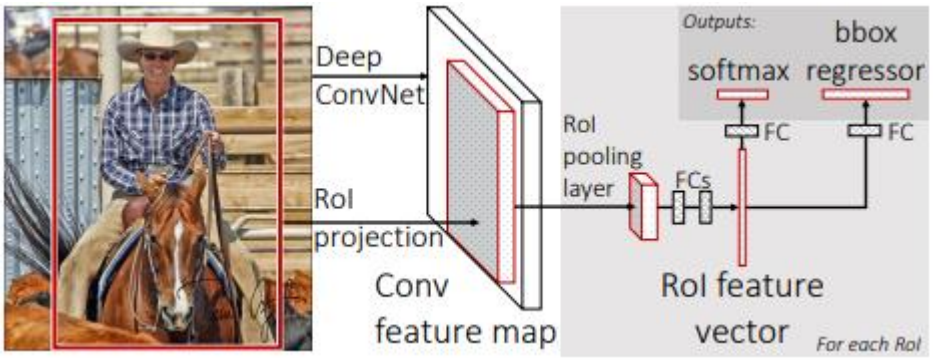

Fig. 3.6. Modul de lucru al algoritmului Fast R-CNN [31]

Fast R-CNN este mai rapid decât R-CNN atât la antrenarea datelor cât și la testarea lor. Îmbunătățirea însă nu este una semnificativă deoarece selectarea regiunilor se realizează într-o componentă separată având un cost foarte mare.

O soluție de îmbunătățire a algoritmului Fast R-CNN se regăsește în Faster R-CNN care rezolva slăbiciunea algoritmului anterior prin integrarea componentei ce alege regiunile în CNN. Această abordare a apărut prima dată în lucrarea „*Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*” [31], printre autori se regăsește și Ross Girshick.

Faster R-CNN, reprezentat în figura 3.7, combină selecția regiunii cu rețeaua neuronală convoluțională descrisă de algoritmul Fast R-CNN, făcând parte din procesul de antrenare. Rețeaua de propunere a regiunii (RPN) acționează ca un mecanism de atenționare pentru rețeaua Fast R-CNN, anunțând rețeaua de zonele unde ar trebui căutate obiectele.

RPN funcționează preluând datele de output de la CNN și trecând matricea de caracteristici printr-o rețea astfel încât se oferă propuneri de regiuni și clasificări pentru fiecare regiune. Se utilizează o procedură prin care ambele rețele sunt antrenate în același timp.

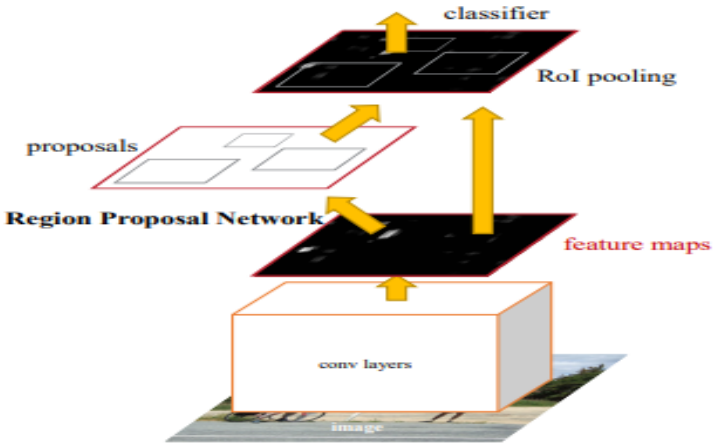

Fig. 3.7. Faster R-CNN reprezintă combinația modulului de selectare de regiuni cu cel din algoritmul Fast R-CNN[17].

3.2.2.2 Familia de algoritmi YOLO

O altă familie populară de algoritmi de recunoaștere de obiecte este YOLO dezvoltată de Joseph Redmon în [17].

Diferențele dintre algoritmii YOLO și cei R-CNN constau în viteza de recunoaștere a obiectelor și acuratețea cu care se recunosc. YOLO este mult mai rapid decât R-CNN, recunoașterea obiectelor făcându-se în timp real. Față de R-CNN, YOLO are o acuratețe mai slabă.

Abordarea **YOLO** presupune folosirea unei singure rețele neuronale care preia o fotografie, prezicând chenare și etichete pentru fiecare dintre acestea. Algoritmul are acuratețea prezicerii mai mică, dar poate analiza între 45 și 155 de cadre pe secundă. Modelul împarte imaginea în celule. În cazul în care centrul unui chenar coincide cu una din celule se prezice un chenar. Fiecare celulă poate ajuta la prezicerea a maxim 2 chenare. Celulele ce prezic un chenar oferă coordonatele x, y, reprezentând centrul chenarului alături de lungimea și lățimea chenarului. În figura 3.7 este prezentat grid-ul și modul cum se aleg chenarele pentru detecția obiectelor.

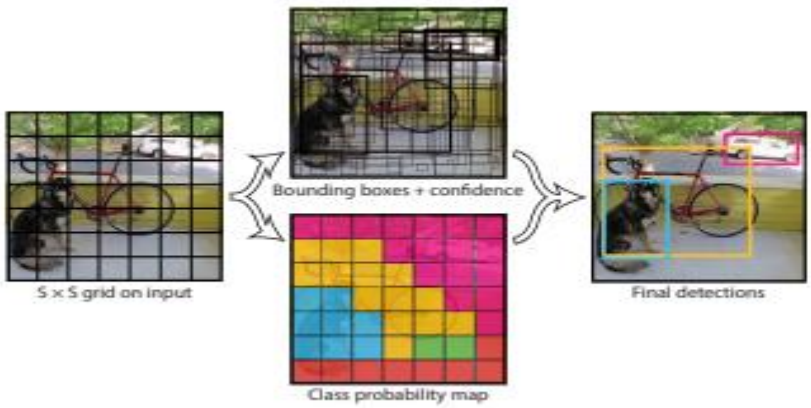

Fig. 3.7. Exemplu YOLO [16]

YOLOv2 sau **YOLO9000** sunt două denumiri pentru aceeași versiune a algoritmului. Această tehnică se bazează pe îmbunătățirea algoritmului precedent YOLO. Numele de YOLO9000 provine de la faptul că antrenarea modelului a fost făcută pe două seturi de date diferite, rezultatul fiind capabil să prezică peste 9000 de obiecte din categorii diferite fără a renunța la viteza predecesorului său.

YOLO9000 aduce o serie numeroasă de îmbunătățiri[18]:

- Antrenarea sistemului cu imagini cu rezoluție mare a permis îmbunătățirea detecției;
- Normalizarea fiecărui strat convoluțional a sporit semnificativ performanța;
- Similar cu algoritmul Faster R-CNN, se folosesc straturi convoluționale pentru a prezice mijlocul unui chenar;
- Față de R-CNN, pentru antrenarea alegerilor chenarelor se folosește un algoritm K-means;
- În etapa de antrenare se dau imagini de input cu dimensiuni variate pentru a asigura detecția de obiecte în imagini cu orice fel de dimensiune;
- Pentru a mări viteza predicției YOLOv2 folosește un nou model, DarkNet-19.

Pentru a ajunge la performanța de a identifica 9000 de obiecte din categorii diferite, YOLO9000 combină mai multe baze de date (ImageNet, COCO/Pascal). Pentru eficientizarea clasificarii, algoritmul folosește un arbore bazat pe ierarhia WordNet, prezentată în figura 3.8, unde cu cat o clasă este mai generală este mai aproape de rădăcină, iar clasele cele mai specifice sunt mai apropiate de frunze.

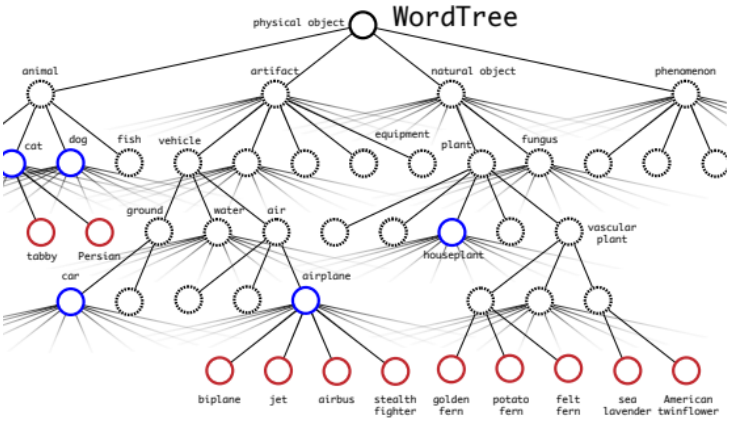

Fig. 3.8. Exemplu ierarhie WordNet[18].

Până în prezent au apărut și alte versiuni ale algoritmului care ajută la îmbunătățirea performanței (YOLOv3 [18], YOLOv4 [7], YOLOv5).

3.2.2.3 MobileNet SSD

MobileNet SSD reprezintă combinația dintre arhitectura unei rețele neuronale profunde creată pentru aplicații mobile ce au nevoie de prelucrarea imaginilor (MobileNet) cu un model de detectare de obiecte (SSD).

MobileNet a apărut în anul 2017 având o structură ușor diferită de rețelele neuronale profunde obișnuite pentru a suplini lipsa puterii de procesare. Această arhitectura folosește convoluții separate în profunzime pentru a reduce semnificativ parametri ce vor fi antrenați.

Acest model de rețele profunde nu se ocupă doar de dimensiunile spațiale ci și de adâncimea imaginii, având două tipuri de convoluții: convoluția în profunzime și convoluția punctuală ce sunt descrise în Figura 3.9. Arhitectura în întregime a modelului MobileNet apare în Figura 3.10.

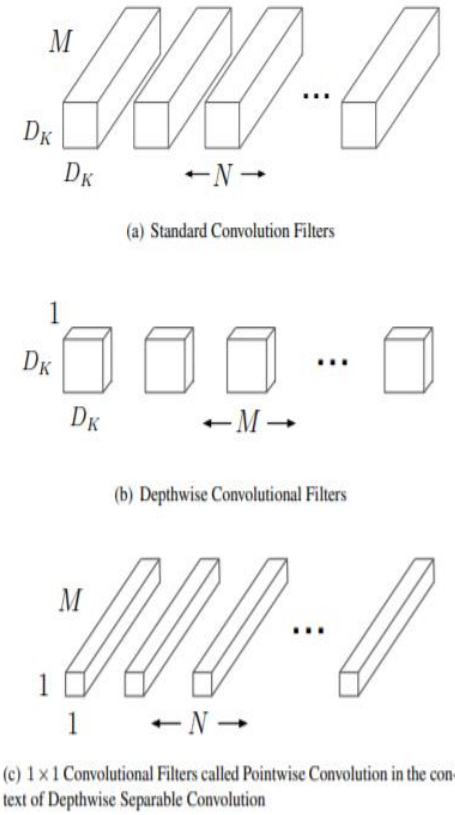

Fig. 3.9 Tipuri de convoluții folosite de rețeaua MobileNet (b si c) [8]

Type / Stride	Filter Shape	Input Size
Conv / s2	3 × 3 × 3 × 32	224 × 224 × 3
Conv dw / s1	3 × 3 × 32 dw	112 × 112 × 32
Conv / s1	1 × 1 × 32 × 64	112 × 112 × 32
Conv dw / s2	3 × 3 × 64 dw	112 × 112 × 64
Conv / s1	1 × 1 × 64 × 128	56 × 56 × 64
Conv dw / s1	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 128	56 × 56 × 128
Conv dw / s2	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 256	28 × 28 × 128
Conv dw / s1	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 256	28 × 28 × 256
Conv dw / s2	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 512	14 × 14 × 256
5× Conv dw / s1	3 × 3 × 512 dw	14 × 14 × 512
5× Conv / s1	1 × 1 × 512 × 512	14 × 14 × 512
Conv dw / s2	3 × 3 × 512 dw	14 × 14 × 512
Conv / s1	1 × 1 × 512 × 1024	7 × 7 × 512
Conv dw / s2	3 × 3 × 1024 dw	7 × 7 × 1024
Conv / s1	1 × 1 × 1024 × 1024	7 × 7 × 1024
Avg Pool / s1	Pool 7 × 7	7 × 7 × 1024
FC / s1	1024 × 1000	1 × 1 × 1024
Softmax / s1	Classifier	1 × 1 × 1000

Fig. 3.10 Arhitectura MobileNet [8]

SSD este prescurtarea de la Single Shot Detector. Arhitectura SSD are la bază una dintre cele mai cunoscute arhitecturi de rețele neuronale convoluționale, VGGnet. Numele rețelei provine de la faptul că localizarea obiectului și clasificarea se face într-un singur pas direct în rețea.

Această rețea are la baza dezvoltării rețeaua VGG-16, unde a fost adăugată o structură suplimentară ce conține straturi convoluționale la sfârșitul acesteia care permit să se detecteze obiecte variate.

Față de YOLO, SSD adaugă mai multe straturi de caracteristici la sfârșitul rețelei de bază. Acestea prezic casete pentru imagini de diferite dimensiuni. S-a constatat că SSD-ul ce primește ca dată de input o imagine de 300 × 300 depășește semnificativ performanța YOLO ce primește o imagine de 448 × 448. Aceasta comparație este evidențiată și în figura 3.11 preluata din studiul „*SSD: Single Shot MultiBox Detector*” [37].

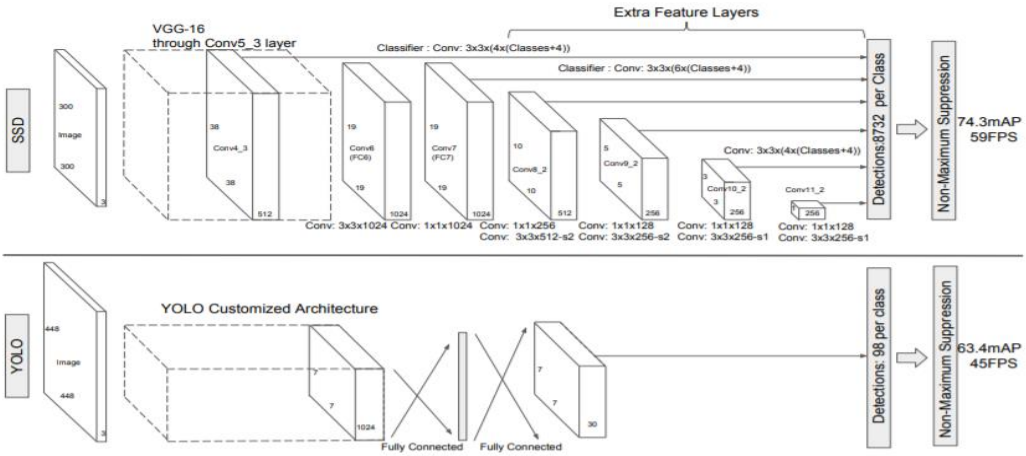

Fig 3.11 Comparația a două modele ce clasifică și localizează obiecte în aceeași rețea [37]

În scopul reducerii resurselor utilizate și a energiei consumate de calculele rețelelor obișnuite în sarcinile de detecție, arhitectura MobileNet a fost contopită cu cea a SSD. Pentru a uni cele două rețele, ultimul strat din MobileNet fiind prezentat în figura 3.12. În afară de output-ul ultimului strat se mai iau și ieșirile altor straturi și se oferă ca input straturilor convoluționale din SSD.

Straturile din rețeaua MobileNet convertesc pixelii din imaginea de intrare în caracteristici ce descriu conținutul imaginii, iar mai apoi sunt trimise straturilor din SSD. MobileNet este folosit pentru a capta caracteristici pentru a două rețea neuronală.

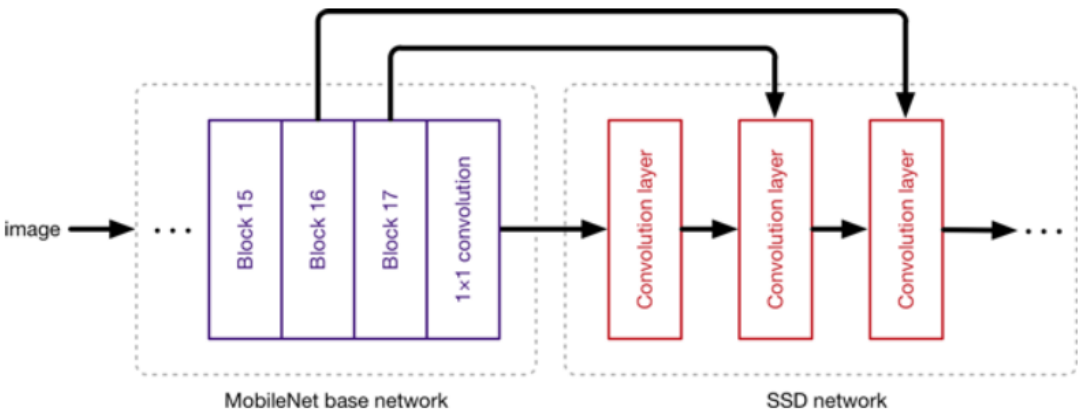

Fig. 3.12 Legătura dintre arhitectura MobileNet și arhitectura SSD [22]

Capitolul 4

Abordare propusă

In prezentul capitol se relatează propria abordare in vederea rezolvării problemei enunțată in capitolul de introducere.

Soluția aleasă pentru rezolvarea problemei constă intr-un sistem adresat aplicațiilor native de Android care sa permită localizarea și navigarea intr-un spațiu interior și recunoașterea și evidențiarea diferitelor iteme.

4.1 Navigarea în spații închise

Plecând de la ideea că GPS nu este o opțiune bună în localizarea în spațiile închise, în literatură au apărut diferite abordări ale acestui subiect. Abordarea mea constă în utilizarea unor markeri de tip QR pentru a extrage informațiile necesare rutării și direcționării.

Sarcina de detectare și scanare a codurilor a fost facută adaptând una din librării din pachetul Google Vision ce poate recunoaște, scana și returna mesajul ascuns într-un code QR. Informația preluată de librărie a fost parsată astfel încat să se poată obține date despre locație și despre poziția în cadrul locației prin intermediul unui cod mapat. Acest cod permite sa interoghez o tabelă ce cunoaște toți vecinii și distanțele pana la ei, să calculez cea mai scurtă rută și sa afișez mesaje de direcționare între două puncte de interes.

Modul ales de modelare al datelor mi-a permis cu ușurință să aplic un algoritm de grafuri pentru găsirea celui mai scurt drum. Informațiile, sub formă de coduri din trei litere

(exemplu: „AAA”, „AAB”, etc.), extrase cu ajutorul activității ce citește codurile QR, reprezintă mapări ale regiunilor/camerelor/zonelor unei locații.

Inițial, chiar înainte de a citi un cod QR, aplicația încărca datele unei locații într-o structură de graf pentru a putea aplica algoritmul Dijkstra. Odată ce se pornește activitatea ce se ocupă cu citirea și returnarea de informații din coduri QR, primul cod QR citit, care respectă formatul necesar, va fi nodul de plecare din care se calculează distanțele către toate celelalte noduri. Ulterior calculării distanței către fiecare nod, se apelează o funcție ce primește ca parametru o listă de noduri ce reprezintă regiuni unde se găsesc elemente de identificat (zone de interes). Funcția returnează o listă cu drumul către cea mai apropiată zonă de interes. După ce se primește rezultatul funcției acesta se adaugă la drumul care trebuie parcurs de un utilizator pentru a ajunge în punctele de interes, iar din lista de noduri ce reprezintă regiuni importante, se șterge nodul de start considerat pentru algoritmul Dijkstra. Se alege un nou nod de start care va fi ultimul nod din lista adusă de funcția care returnează cel mai scurt drum între două puncte. Algoritmul se tot repetă până când lista de regiuni importante devine vidă. O reprezentare vizuală a algoritmului se poate vedea in figura 4.1.

```
// The current position is stored in currentPos variable
String currentPoz = startPoint;

// The ImportantPoints list will keep the code of a region where items can be found.
// This will be used to know when a user reaches an area where he can find items to detect
List<String> importantPoints = new ArrayList<>();

// This loop will continue until all important regions will be stored in the navigation path
while (nodeRegion.size() > 0) {

    for (Node node : DijkstraAlgo.nodes) {
        node.parent = null;
    }

    // In this point the distance between the current node and all the other nodes will be calculated
    DijkstraAlgo.computePaths(DijkstraAlgo.nodes.get(DijkstraAlgo.nodes.indexOf(new Node(currentPoz))));

    // Next line will send the list  region list  as a parameter for the function which
    // will find the closest region from a  list and will compute the path to it
    List<Node> segmentNode = DijkstraAlgo.getShortestPathTo(nodeRegion);
    System.out.println("Segment: " + segmentNode);
    if (path.size() > 0) {
        path.remove( index: path.size() - 1);
    }
    path.addAll(segmentNode);

    // The current node becomes the last node form computed path
    currentPoz = segmentNode.get(segmentNode.size() - 1).value;

    // From the list of region will be removed current region
    int removePoze = nodeRegion.indexOf(new Node(currentPoz));
    Node removedNode = nodeRegion.remove(removePoze);
    importantPoints.add(removedNode.value);
}
```

Fig. 4.1 Codul pentru obținerea drumului ce trebuie sa treacă prin mai multe puncte

După obținerea rutei finale care trece prin toate punctele de interes, începe navigarea propriu-zisă. Pe baza rutei se obțin indicații de direcționare între două puncte de interes. Aceste informații sunt preluate din baza de date a telefonului. În cazul în care noul nod citit nu este și următorul din lista ce reprezintă ruta, se verifică dacă printre vecinii nodului citit de codul QR se află următorul punct al rutei ce trebuie urmate. Dacă se află, se indică locația către acesta. Dacă nu se află, se obține o nouă rută ce pleacă din punctul curent către toate celelalte puncte de interes rămase.

In momentul când se ajunge intr-un punct de interes se deschide activitatea de detecție obiecte.

4.2 Identificarea de obiecte

Identificare de obiecte și clasificarea lor în anumite categorii de obiecte reprezintă o sarcină importantă. Algoritmii și metodele care ajută la detecția de obiecte sunt în expansiune având aplicații in numeroase domenii.

In cadrul sistemului făcut, sarcina de detecție a fost îndeplinită cu ajutorul librăriei TensorFlow Lite. Din această librărie am extras partea care se ocupă de detecția de obiecte si am modificat-o pentru a îmi găsi doar obiecte care fac parte dintr-o anumită categorie (exemplu: cani, scaune, mese, etc.). Modelul utilizat pentru detecția obiectelor este MobileNet SSD. Am ales acest model deoarece este special conceput pentru dispozitive care nu dețin performanțe computaționale foarte mari și în comparație cu alte modele precizia de detecție a obiectelor este mai mare.

Activitatea de detecție primește mereu o listă de obiecte ce ar trebui identificate într-o anumită regiune. Obiectele se determină pe rând, astfel încât algoritmul știe la un anumit moment ce obiect ar trebui determinat. Când un anumit element este identificat cu succes se evidențiază prin încadrarea obiectului într-un chenar.

Pentru a trece de la un obiect la altul, utilizatorul trebuie să apese pe unul din butoanele bifă sau „X”. În momentul când se apasă pe oricare dintre aceste butoane se adaugă într-o listă specifică elementul și un status care determina dacă acesta a fost găsit sau nu. Codul responsabil cu gestionarea elementelor este prezentat in figura 4.2. În momentul în care se identifica toate elementele dintr-o anumită regiune pot avea loc două acțiuni:

- Dacă mai sunt elemente de identificat din alte regiuni, activitatea se închide și se deschide cea de navigare;
- Dacă nu mai sunt elemente de identificat din alte regiuni, activitatea se închide și se deschide o activitate care afișează lista cu elementele selectate și daca acestea au fost identificate sau nu.

```
// Getting the list of object
List<String> elements = detectionElement.getSelectItems();

// Remove the first one
String element = elements.get(0);
elements.remove( index: 0);

// Delete the element that should be detected from the object list
detectionElement.setSelectItems(elements);

// The element will be deleted form the list that contains all the elements from any  region
List<String> allElementsForSearching = searchingItem.second;
List<IconItemState> allIconElementedForSearching = searchingItem.first;
allIconElementedForSearching.remove( index: 0);
allElementsForSearching.remove(allElementsForSearching.indexOf(element));
searchingItem = new Pair<>(allIconElementedForSearching,allElementsForSearching);

// The deleted item will be added to the list of found objects, and a new FOUND status will
// be added to the status list.
List<String> allElementFound = foundItem.second;
List<IconItemState> allIconElementedFound = foundItem.first;
allElementFound.add(element);
allIconElementedFound.add(IconItemState.FOUND);
foundItem = new Pair<>(allIconElementedFound,allElementFound);
```

Fig 4.2 Codul butonului ce gestionează elementele ce sunt identificate (gestiunea elementelor neidentificate se face similar)

Cele două sarcini descrise fac parte dintr-un sistem ce se alcătuit din două aplicații. Aplicația responsabilă de stocarea datelor și gestionarea lor este considerată back-end-ul sistemului. Aplicația prin care se permite utilizatorului sa interacționeze cu sistemul este front-end-ul aplicației.

4.3 Back-end

Back-end-ul aplicației constă într-un proiect în python ce utilizează tehnologia Django.

Python este un limbaj orientat pe obiecte, de nivel înalt. Sintaxa limbajului este una simplă oferind dezvoltatorilor posibilitatea de a construi aplicații complexe mai rapid. Un mare avantaj constă în faptul că este un limbaj interpretat. În cazul apariției mai multor erori, Python afișează doar una dintre acestea, fapt care ușurează cu mult procesul de debugging.

Pentru un dezvoltator una dintre cele mai importante probleme este lizibilitatea codului. Prin sintaxa sa, Python, permite rezolvarea anumitor sarcini cu mai puține linii de cod comparativ cu alte limbaje precum C, C++, sau Java fapt care oferă un plus de lizibilitate.

Python este printre cele mai utilizate limbaje de programare fiind util pentru o gamă variată de tipuri de aplicații. Limbajul oferă suport pentru un număr imens de librării, ce pot scuti dezvoltatorii să implementeze bucăți din aplicații de la zero. Python este considerat ca fiind o alegere excelenta pentru proiecte care implică parți de inteligență artificială deoarece are suport pentru numeroase librării precum TensorFlow, Keras, etc.

Django este un framework web al limbajului python ce a fost dezvoltat inițial între anii 2003 si 2005. Acest framework poate fi folosit atât pentru partea de back-end a unei aplicații web cât și pentru partea de front-end.

Django utilizează ORM (Object–relational mapping) pentru maparea entităților și crearea tabelelor. Unul din cele mai importante atuuri ale acestei tehnologii este suportul legat de securitate aplicațiilor dezvoltate, ajutând la evitarea greșelilor comune. Django mai oferă și un mod sigur de a gestiona utilizatori si conturile acestora.

Django utilizează o arhitectură bazată pe componente ce nu se partajează. Fiecare parte a arhitecturii este independentă de celelalte, prin urmare poate fi ușor înlocuibilă. Aceasta separare semnifică ușurința de scalare și adaptare a framewok-ului la orice tip de aplicație. Printre cele mai mari site-uri care utilizează componente Django se numără Instagram si Disqus [36].

Datorita faptului ca framework-ul Djago este scris in python, putem afirma ca este portabil si poate rula pe gama foarte variata de dispozitive.

Printre motivele Django este potrivit pentru partea de back-end a sistemului creat, se numară ușurința cu care se lucrează cu această tehnologie, viteza ridicată de development și posibilitatea de a încărca proiectul pe platforma Heroku [27].

Heroku este un cloud PaaS (Platform as a Service) container. Inițial platforma a fost făcută pentru proiecte de tip Ruby on Rails, însă acum permite încărcarea, gestionarea și scalarea aplicațiilor moderne într-un mod autonom. Platforma suportă cele mai populare instrumente existente la momentul actual pentru dezvoltarea, incărcarea, testarea și gestionarea aplicațiilor end-to-end.

Baza de date utilizată este PostgreSQL deoarece se integrează foarte bine cu serviciile de hosting oferite de Heroku.

Utilizând Django am creat entitătile necesare aplicației și implicit și baza de date.

Așa cum arată și diagrama ce reprezintă relațiile dintre tabelele si clasele din figura 4.3, entitățile necesare pentru alcătuirea sistemului și memorarea informațiilor sunt:

- „Location” – ce are ca atribute un id si un nume de locație. Aceasta clasa e folosită pentru a memora locațiile pentru care poate fi folosită aplicația;
- „Code” – ce are ca atribute un id și potențiale coduri pentru regiunile unei locații;
- „CodeLocation” – ce este o tabelă de legătură având ca și chei străine id-ul unui element de tip „Code” și id-ul unui element de tip „Location”. Acestă tabelă are de asemenea un id propriu, diferit de cheile străine;
- „Route” - ce are ca atribute un id, un câmp de distantă numit între 2 puncte, un câmp de direcționare între doua puncte numit „aid_message” și două câmpuri chei străine ce conțin id-ul a două elemente „CodeLocation” diferite;
- „Element” – ce are ca atribute un id și un nume de obiect. Această tabela memorează o listă de elemente ce ar putea fi găsite în diferite regiuni ce aparțin unei locații;
- „Region” – ce are ca atribute un id, un nume de regiune (Bucătărie, Sufragerie, etc.), și un câmp ce reprezintă o cheie străină catre „CodeLocation”. Legătura cu „CodeLocation” are sens deoarece trebuie să se știe cărei locații aparține o regiune;
- „ElementRegion” – ce are ca atribute un id, și două chei străine legate de „Element” și de „Region”. Aceasta tabelă se folosește pentru a ști ce elemente se pot găsi într-o anumita regiune.

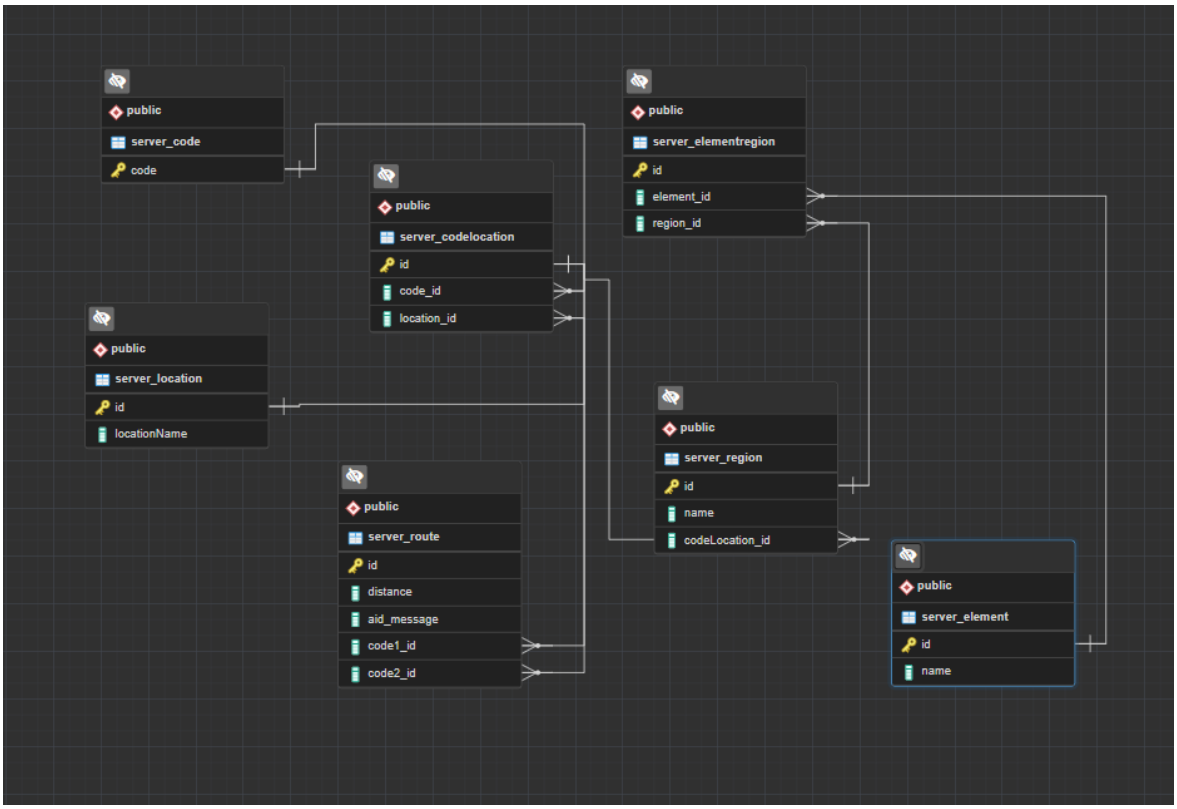

Fig. 4.3 Relațiile dintre tabele extrase din PostgreSQL

Pentru a putea trimite obiectele prin intermediul call-urilor HTTP, fiecare clasă prezentată mai sus are creată și câte o clasă de serializare in fișierul serializer.py (Exemplu in Fig. 4.4).

```
class LocationSerializer(serializers.ModelSerializer):
    # Serialize all fields from Location class
    class Meta:
        model = Location
        fields = '__all__'

class CodeSerializer(serializers.ModelSerializer):
    # Serialize all fields from Code class
    class Meta:
        model = Code
        fields = '__all__'
```

Fig. 4.4 Exemplu clase de serializare

Toate call-urile făcute pentru a obține datele de la server la aplicația mobile sunt de tip „GET”, iar codul sursă al funcțiilor apelate se află în fișierul views.py. Aceste call-uri primesc ca parametru id-ul unei locații si returnează datele locației necesare aplicației.

Există câte un astfel de call pentru fiecare din clasele „CodeLocation”, „Region”, „Element”, „ElementRegion” și „Route”.

In cazurile in care o funcție view trebuie să folosească mai multe tabele pentru a putea lua datele ce corespund unei regiuni dintr-o anumită tabelă am extras logica necesară și am creat funcții separate in fișierul utils.py pentru a respecta principiul singurei responsabilități (Exemplu in figura 4.5 si 4.6).

```python
@api_view(['GET'])
def getRegion(request):
    # The function gets from the headers the id of a location
    try:
        data = request.headers
        # Method getRegion from utils.py is called for getting all regions of a location
        list_regions = utils.getRegion(data["location"])
        return Response(list_regions)
    except Exception as er:
        return Response({'error':str(er)},status=status.HTTP_400_BAD_REQUEST)
```

Fig. 4.5 Exemplu funcție view pentru returnarea regiunilor

```python
def getRegion(id):
    # This fuction returns all regions of a location
    # Input: id - integer
    # Output: a list of region

    list_regions = []
    codesLocation = CodeLocation.objects.filter(location = id)
    for code in codesLocation:

        # For every points in a location a region should be brought from a database
        regions = Region.objects.filter(codeLocation = str(code.id))
        serializerRegion = RegionSerializer(regions, many = True)
        list_regions = list_regions + serializerRegion.data

    return list_regions
```

Fig. 4.6 Exemplu functie ce returneaza regiuni pe baza codurilor

4.4 Front-end

Proiectul de front-end a fost conceput ca o aplicație nativa de Android utilizându-se mediul oficial de dezvoltare al sistemului de operare Android, Android Studio. Această platformă de dezvoltare a fost creata de JetBrains' IntelliJ IDEA. Ca si limbaj de dezvoltare al aplicației am ales Java deoarece acesta sta la baza creării sistemului Android.

Într-o manieră simplă se poate afirma că aplicația are rolul de a găsi într-o locație cu mai multe camere, elemente selectate dintr-o listă. Aplicația poate fi împărțită după două criterii: după structură și după funcționalități.

Din punct de vedere structural, aplicația este alcătuita din patru activități sau ecrane.

1) Prima activitate este chiar cea de selectare a elementelor;
2) A doua activitate are mai multe roluri:
    - Rolul de a declanșa activitatea de citire coduri QR. Ulterior acestei acțiuni se receptează id-ul unei locații și se fac call-urile la server necesare obținerii informațiilor;
    - Rolul de a arata un preview al elementelor selectate care se găsesc în locația respectiva;
    - Rolul de a declanșa începerea navigării în vederea detectării obiectelor;
    - Rolul de a declasa închiderea aplicației, daca se dorește, după ce s-au detectat toate obiectele;
    - Rolul de a declanșa deschiderea activității de selectare elemente cu scopul de a crea o noua lista de elemente, daca se dorește.

3) A treia activitate este cea de citire coduri QR. Aceasta activitate are de asemenea mai multe roluri:

- Rolul de a citi codul QR in vederea obținerii locației și navigarea înapoi la a doua activitate;

- Rolul de a citi coduri QR, și de a ajuta la navigarea in interiorul unei locații. În cazul în care ultimul cod QR citit are in componență sa codul unei regiuni în care se găsesc elemente selectate din lista inițială, aplicația declanșează începerea activității de detectare a obiectelor.

4) A patra activitate este cea de detectare a obiectelor. În momentul când se deschide această activitate se creează o listă cu elementele selectate, corespunzătoare unei regiuni. Elementele se detectează în ordine, iar utilizatorul trebuie să apese pe un buton dacă elementul curent care trebuie identificat, a fost găsit cu succes, sau pe alt buton in cazul in care elementul detectat poate fi eronat sau nu se găsește in zona respectivă.

Din punct de vedere funcțional, aplicația poate fi împărțita in trei subproiecte

1) Primul subproiect consta în detectarea de coduri QR bazat pe librăria de detecție de coduri de la Google, Google Visionș
2) Al doilea subproiect constă in detectarea de obiecte, încadrându-le într-un chenar cu o eticheta atașata. Acest proiect este o adaptare a proiectului „object detection" din librăria TensorFlow Lite. Modelul de antrenare folosit este MobileNet SSD, iar baza de date folosită pentru antrenarea modelului este COCO dataset;
3) Al treilea subproiect consta in selectarea elementelor, aducerea datelor corespunzatoare unei locatii, gasirea celui mai mic drum dintre mai multe puncte utilizand algoritmul Dijkstra si contopirea la acesta a celorlalte doua proiecte.

In cele ce urmează se va prezenta modul cum funcționează aplicația:

- Utilizatorul deschide aplicația și își alege unul sau mai multe elemente așa cum se prezintă in figura 4.7. Dacă utilizatorul apasă butonul de start inainte de a selecta cel puțin un element apare un mesaj care îl împiedică pe utilizator sa treacă la activitatea următoare figura 4.8;

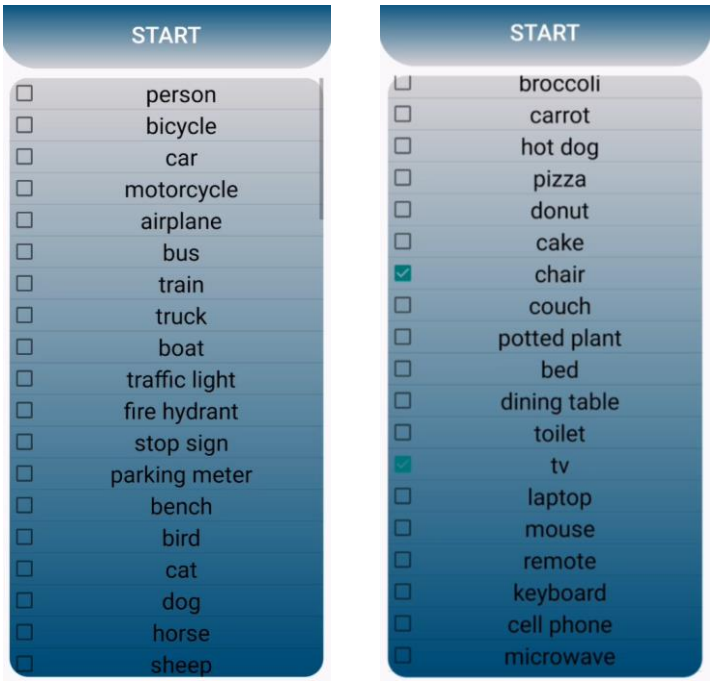

Figura 4.7 Deschiderea aplicației și selectarea elementelor

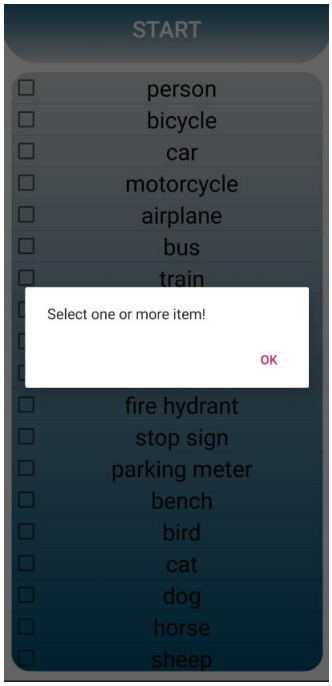

Fig. 4.8 Mesaj indicație în cazul în care utilizatorul apasă pe buton de start fără a selecta niciun element

- După ce utilizatorul selectează minim un element și apasă butonul de start se trece la a doua activitate care solicită apăsarea butonului „Get Data” pentru a obține datele unei locații figura 4.9;

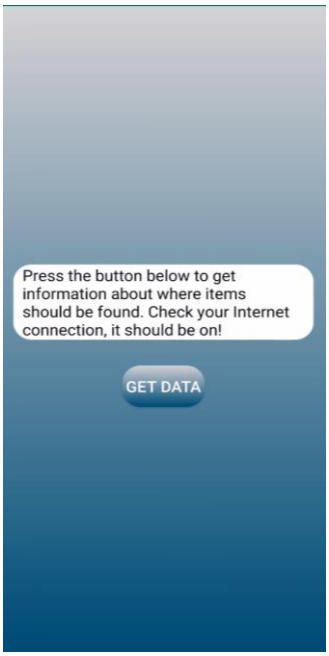

Fig. 4.9 Varianta a celei de-a doua activități care înștiințează utilizatorul de nevoia de a obține datele despre locație

- Odată apăsat butonul „GET DATA” se deschide activitatea de scanare a unui cod QR. Din acesta se extrage locația și se trimite la activitatea a doua (exemplu activitate QR pentru extragerea locației se arata in figura 4.10);

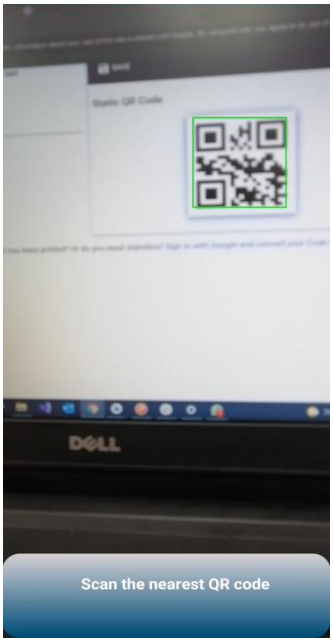

Fig. 4.10 Citirea unui cod QR

- În momentul în care s-a detectat codul QR activitatea curentă se închide și se pornește cea de-a doua activitate. Odată ce se începe a doua activitate se fac call-urile necesare pentru a obține datele locației. După ce se obțin datele se verifică dacă elementele selectate in prima activitate se pot găsi in locație. Dacă se pot găsi vor fi afișate într-o listă și va apărea butonul de „Routing” care va permite începerea navigării așa cum este arata figura 4.11. În cazul in care nu toate elementele selectate se pot identifica, comportamentul este unul asemănător, diferența constând in înștiințarea utilizatorului printr-un mesaj si marcarea elementelor in lista cu un icon corespunzător (figura 4.12);

Fig. 4.11 Afișarea elementelor după primirea datelor de la server

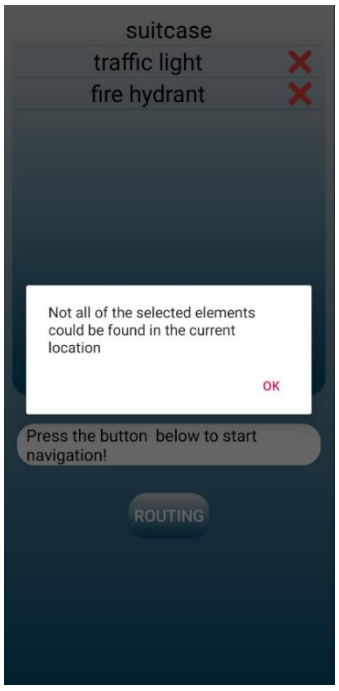

Fig. 4.12 Mesaj afișat in cazul in care nu se identifica toate elementele

- Când utilizatorul apasă butonul de routing se va închide a doua activitate și se va deschide a treia (cea de scanare coduri QR), de data aceasta pentru a arata drumul către fiecare regiune și a începe activitate de detecție elemente. Atunci când aplicația e deschisă și se scanează primul cod, cu ajutorul algoritmului Dijkstra se va găsi cel mai scurt drum pentru a ajunge în fiecare regiune care are elemente de identificat. La scanarea fiecărui cod QR va apărea indicația către următorul cod așa cum se arată în figura 4.13. Dacă se scanează un cod care corespunde unei regiuni

în care ar trebui să se detecteze unul sau mai multe elemente se deschide activitatea de detecție.

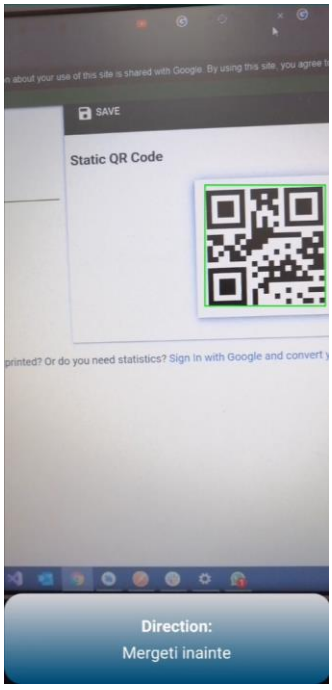

Fig. 4.13 Indicațiile ce apar în momentul când un cod QR este scanat

- Când activitatea de detecție obiecte este deschisă ea primește o listă de iteme corespunzătoare unei regiuni ce trebuiesc detectate. Din această listă se afișează mereu primele două elemente în cele două casete. Elementele se detectează pe rând. Cele două icon-uri sunt butoane prin care utilizatorul afirmă dacă a găsit sau nu un anumit obiect. Dacă se apasă pe icon-ul „bifă” obiectul se consideră găsit și se adaugă la o listă de obiecte găsite, mai apoi se încercă să se detecteze următorul obiect. Procesul este similar și pentru butonul ce are icon-ul „X” (figura 4.14). Dacă elementul ce trebuie selectat este ultimul din lista pentru regiunea curentă în momentul când se apasă unul din cele două butoane pot apărea două acțiuni:
    - In cazul în care ultimul item din lista pentru regiunea respectivă NU este ultimul din lista tuturor elementelor selectate activitatea curentă se închide și se deschide activitatea de QR code pentru a continua navigarea către celelalte regiuni.

- In cazul în care care ultimul item din lista pentru regiunea respectivă este și ultimul din lista tuturor elementelor selectate, activitatea curenta se închide și se deschide a doua activitate în care se va afișa lista elementeleor selectate cu bifa pentru cele găsite și cu „X” pentru cele ce nu au fost identificate sau nu se regaseau in locatia respectiva (figura 4.15).

Fig. 4.14 Interfața activității de detecție.

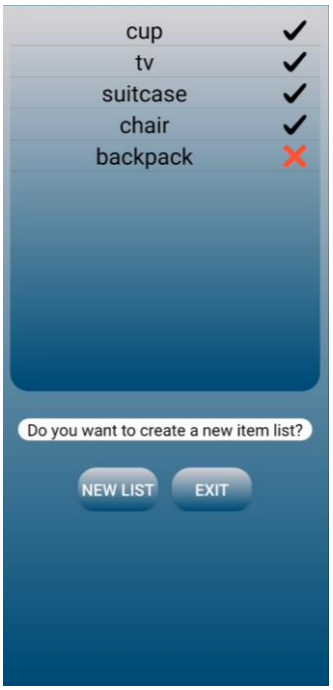

Fig. 4.15 Interfața activității a doua în momentul în care se termină de detectat toate elementele selectate inițial.

- La finalul detecției tuturor elementelor se redeschide activitate a doua, de această dată prezentând în lista de elemente statusul tuturor elementelor selectate. Apare posibilitatea de a încerca crearea unei noi liste și reluarea procesului sau de a părăsi activitatea Fig. 4.15.

Capitolul 5

Arhitectura și testarea sistemului

Capitolul curent prezintă proiectul din perspectiva ingineriei și arhitecturii sistemului. Acest sistem este prezentat atât într-o manieră teoretică cât și practică prin intermediul diferitor diagrame. Pe lângă partea arhitecturală în acest capitol se mai pot găsi atât detalii despre verificarea si validarea sistemului, prin intermediul testelor, cât și detalii despre performanțele sistemului în comparație cu alte abordări.

5.1 Arhitectura de proiectare mobile

Construcția aplicației mobile a fost făcută după un model asemenător structurii de proiectare MVVM (Model-View-ViewModel). Din denumire se observă că acest model are trei componente: Model, View si ViewModel.

Componenta Model este alcătuita din clasele ce sunt modelatoare de date prelucrate în aplicație, API-uri si clase repository.

Clasa repository destinata aducerii datelor de pe server și a interogării bazei interne de date a telefonului este FlowDataRepository. Aceasta clasa apelează interfața ce conține metodele pentru end-pointuri, WebService.

Obținerea datelor se datorează interacțiunii cu pachetul de networking, iar prin clientul REST folosit, Retrofit, call-urile necesare sunt trimise către partea de back-end. Retofit oferă o soluție ușor de implementat a unui serviciu web bazat pe arhitectura REST. Pentru a face solicitări de tip HTTP, Retrofit utilizează biblioteca OkHTTP.

Pachetul Model interacționează și cu baza locala de date a telefonului. Entitățile ce fac parte din aplicație pot fi salvate în baza de date locala a telefonului printr-o librărie de tip ORM(Object Relational Mapping).

Pachetul View se regăsește în aplicație prin intermediul elementelor de UI. Elementele de tip view ce pot fi găsite in aplicația Android fac parte din structuri de tip fragment sau activități. Activitățile stau la baza dezvoltării aplicațiilor Android. Android are o structura unica, punctul de start al aplicației fiind în sine o activitate ce conține diferite metode care corespund unor etape din „viată" unei activități. Aplicația este alcătuita din patru activități:

- SelectelementsListActivity, este punctul de start al aplicației. In aceasta activitate se selectează anumite elemente dintr-o lista;
- MainActivity, este cea mai folosita activitate din aplicației, utilizata atât pentru a face legătura cu activitatea de citirea a codurilor QR cat și pentru a îndruma utilizatorului;
- QRVisionScannerActivity, este activitate care scanează coduri QR si ajuta la navigarea in interiorul unei locații;
- DetectorActivity, activitatea care detectează obiecte.

Ultimul pachet care face parte din structura MVVM este ViewModel. În general acest pachet se ocupă cu transformare datelor din Model în elemente de tip View.

Ceea ce diferențiază modelul implementat de MVVM este faptul că nu exista clase propriu zise care să îndeplinească sarcinile unui ViewModel. Cu toate acestea, in interiorul activităților SelectelementsListActivity si MainActivity există o componenta de tip listă care este gestionată într-o clasă separată. Prin acest fapt se poate considera că cele două activități îndeplinesc atât rol de View cât si de ViewModel.

5.2 Arhitectura software a sistemului

Arhitectura unei aplicații surprinde atât modul prin care componentele acesteia sunt legate între ele cât și interacțiunea utilizatorului cu sistemul. Aceste legături se pot pune în evidența utilizând mai multe modele software. In mare aceste modele pot fi împărțite în trei categorii:

- Modelul funcțional;
- Modelul dinamic;
- Modelul obiectual.

Principalul mod prin care se pot evidenția modelele este prin intermediul unor diagrame. Cel mai folosit limbaj în crearea de diagrame este limbajul UML (Unified Modeling Language). UML este limbajul universal pentru descrierea de proiecte software [11].

5.2.1 Modelul funcțional

Acest model prezintă sistemul din punct de vedere al interacțiunii cu utilizatorul. Diagrama corespunzătoare modelului poartă numele de diagrama cazurilor de utilizare. Aceasta diagrama se face de obicei în momentul când se colectează datele unei probleme, în vederea reprezentării funcționalităților sistemului. Componentele UML ce alcătuiesc aceasta diagramă sunt: actorii, relațiile de includere, relațiile de extindere, cazurile de utilizare și relații de generalizare și asociere.

Rolul diagramei claselor de utilizare este de a defini componentele generale ale unui sistem fata a preciza structura sa. In figura 5.1. este prezentata diagrama cazurilor de utilizare corespunzătoare sistemului creat

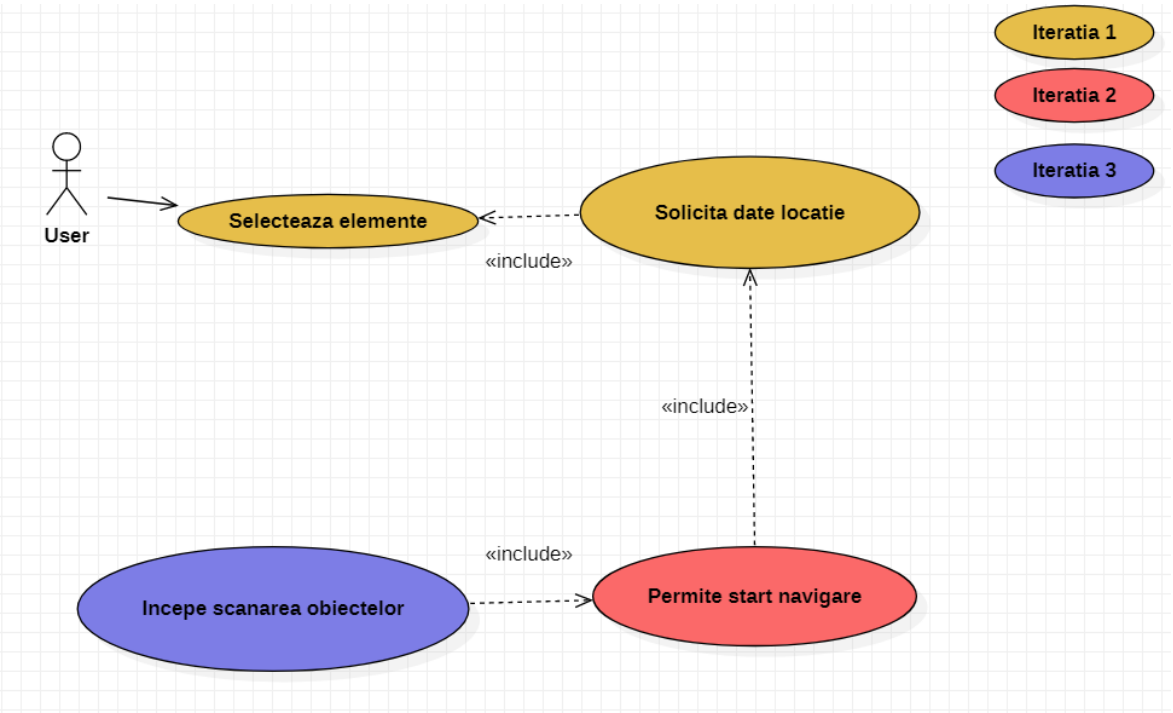

Fig. 5.1 Diagrama cazurilor de utilizare corespunzătoare sistemului

5.2.2 Modelul dinamic

Principala caracteristica a acestui model constă în descrierea implementării modului cum se modifică și când se modifică sistemul. Acestui model îi corespund două diagrame care sunt echivalente: diagrama de secvență și diagrama de comunicare. Elementele apărute pe oricare dintre aceste diagrame ajută la înțelegerea ordini în care ar trebui să aibă loc acțiunile. Pentru a prezenta sistemul din punctul de vedere al modelului dinamic, se va utiliza diagrama de secvență.

Modul de reprezentare al elementelor și acțiunilor în diagrama de secvențe se face pe baza timpului, astfel obiectele participante au forma unor dreptunghiuri reprezentate orizontal, iar axa timpului este reprezentată ca o linie pe verticală. Cea mai din stânga linie temporară corespunde utilizatorului, astfel descriindu-se și ordinea în care acesta trebuie să interacționeze cu sistemul. In figura 5.2. se descrie diagrama de secvență a cazurilor de utilizare a aplicației mobile.

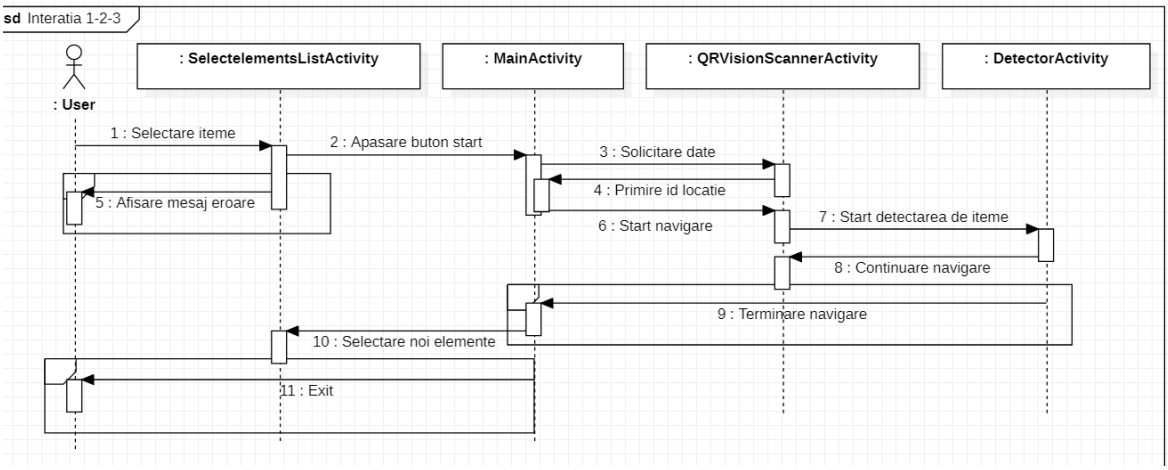

Figura 5.2. Diagrama de secvența

5.2.3 Modelul obiectual

Modelul obiectual descrie obiectele și asocierile dintre acestea. Diagrama corespunzătoare acestui model este diagrama de clase. Această diagramă are un aspect asemănător unui graf ale cărui noduri sunt reprezentată te de entitățile ce alcătuiesc sistemul și ale cărui arce reprezintă relațiile între entități. Modul de reprezentare al entităților în diagrama de clase este printr-un dreptunghi ce este împărțit în trei regiuni, pentru nume de clasă, atribute și metode. Relațiile ce pot avea loc între entități sunt de asociere, agregare, generalizare și compoziția. Fiecare dintre acestea sunt reprezentate ca diferite tipuri de linii.

Figura 5.3. prezinta diagrama de clase aferenta aplicației mobile.

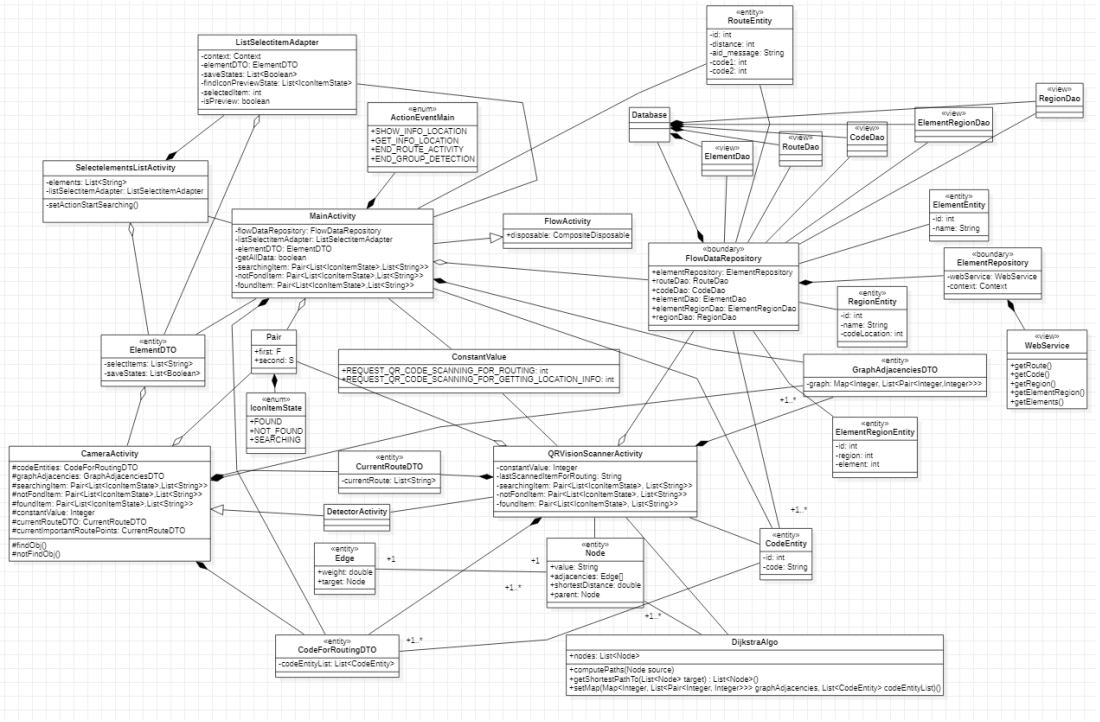

Fig. 5.3. Diagrama de clase a sistemului

Diagramele sunt utile pentru a ușura procedeul prin care se construiesc produse soft. Un alt mod utilizat pentru a ajuta la dezvoltarea mediilor soft este crearea scenariilor de utilizare. Acestea practic descriu ce poate sau ce nu poate să facă un utilizator. Tabelele 6.1 si 6.2 prezintă câte un scenariu de utilizare creat pentru activitatea de navigare respectiv pentru activitatea de detecție.

Nume	Navigarea
Participanți	Inițiat de Agent Comunica cu baza de date
Flux de evenimente	1. Agentul deschide activitatea de navigare. 2. Agentul scanează cel mai apropiat cod QR. 3. Agentul urmează indicațiile oferite de sistem. Aceste indicații sunt preluate de la baza de date 4. In momentul in care Agentul scaneze codul unei regiuni in care exista elemente de identificat, navigarea se oprește

Condiții de intrare	Selectarea cel puțin a unui element care se regăsește la locația unde se face rutarea
Condiții de ieșire	Agentul a ajuns în toate punctele in care se găsesc elemente
Cerințe de calitate	Nu se poate naviga daca niciun element nu a fost selectat

Tabelul 6.1

Nume Scenariu: Navigare

Participanți: George - Agent

Flux de evenimente:

1. Agentul deschide activitatea de navigare.

2. Agentul scanează cel mai apropiat cod QR.

3. Agentul urmează indicațiile oferite de sistem. Aceste indicații sunt preluate de la baza de date

4. In momentul în care Agentul scaneze codul unei regiuni în care există elemente de identificat, navigarea se oprește

Scenariul descris pentru navigare este scenariul în care utilizatorul folosește așa cum se presupune aplicația. In mod normal se creează numeroase astfel de scenarii pentru fiecare funcționalitate. Un alt scenariu, dar de data asta corespunzător activității ce ajută la detecția de obiecte, este prezentat in tabelul 6.2.

Nume	Detecție obiecte

Participanți	Inițiat de Agent
Flux de evenimente	1. Agentul deschide activitatea de detecție când scanează codul unei regiuni unde trebuie să se găsească elemente de identificat. 2. Agentul verifică dacă elementul curent se găsește în mediu . 3. Când găsește elementul, Agentul apasă pe butonul de „bifă” și se va încerca să se găsească un nou element sau, dacă nu mai sunt elemente ce trebuiesc detectate în lista de obiecte, activitatea se închide.
Condiții de intrare	Exista cel puțin un element care trebuie detectat în regiunea respectiva
Condiții de ieșire	Nu mai sunt elemente ce trebuiască detectate
Cerinte de calitate	Nu se poate porni activitatea fără a fi elemente ce trebuiesc detectate în zona respectiva

Nume Scenariu: Detecție obiecte

Participanți: George - Agent

Flux de evenimente:

1. George deschide activitatea de detecție când scanează codul unei regiuni unde trebuie să se găsească elemente de identificat.

2. George verifică dacă elemntul curent se găsește în mediu .

3. Când găsește elementul, George apasă pe butonul de „bifă” și se va încerca să se găsească un nou element sau dacă nu mai sunt elemente ce trebuiesc detectate în lista de obiecte, activitatea se închide.

5.3 Testarea sistemului

Atât arhitecturile de modelare cât și arhitecturile de proiectare sunt necesare în momentul construcției unui sistem soft. Acestea ajută la o înțelegere profundă a cerințelor problemei și la realizarea unui model cât mai bun ce poate fi ușor scalat și care e predispus la cât mai puține bug-uri si vulnerabilități. Pentru a avea un grad în plus de siguranță de cele mai multe ori sistemele soft prezintă și diferite tipuri de teste. Acestea sunt efectuate cu scopul de a vedea dacă aplicația funcționează corespunzător.

Domeniul testării este unul larg dezbătut în comunitatea dezvoltatorilor de soft. Testarea este un proces fără de care aplicațiile software nu ar putea exista. In unele cazuri testarea riguroasă a sistemelor soft poate salva vieți. Există numeroase tipuri de teste cele mai folosite fiind: unit test, automation test, integration test si UI test.

In general, pentru ca testele să acopere cât mai bine funcționalitățile sistemului se utilizează anumite metode de testare. Cele mai întâlnite metode sunt Black Box Testing și White Box Testing.

Black Box Testing este o abordare prin care se pot crea teste, fără a se cunoaște structura internă. In general se folosește pentru testări de sisteme, cunoștințele de programare nefiind necesare. Pentru a avea o acoperire cât mai mare asupra codului testat există două metode consacrate, ECP (Equivalence Class Partitioning) ce împarte cazurile de testare în clase valide și invalide, și BVA (Boundary Value Analysis) ce testează și granițele ce delimitează un test de a fi valid sau invalid.

White Box Testing este o abordare prin care se pot crea teste, cunoscând structura internă. In general acesta abordare este folosită pentru testări de tip unit test sau integration test ale uneia sau mai multor funcționalități. La fel ca Black Box testing și White Box testing are două abordări principale consacrate. Una dintre aceste metode de testare este prin acoperirea drumurilor. Acestă metodă vede o funcționalitate ca pe un graf al cărui noduri sunt instrucțiunile decizionale si repetitive, iar muchiile reprezintă tranzițiile prin intermediul codului de la un nod la altul. A doua abordare prin care se poate realiza testare de tip White Box este prin acoperirea codului sursa. Această abordare testează instrucțiunile de control folosind un număr minim de teste. Termenii specifici acestei abordări sunt acoperirea instrucțiunilor (*statement coverage*), acoperirea deciziilor

(*decision coverage*), acoperirea condițiilor (*condition coverage*), acoperirea deciziilor și condițiilor (*decision-condition coverage*), acoperirea condițiilor multiple (*multiple condition coverage*), acoperirea buclelor (*loop coverage*) [10].

Softul implementat cuprinde, de asemenea, anumite teste. Testele utilizate sunt de tip unit test si UI test.

Unit testele au fost făcute pentru funcționalitatea de navigare, mai exact pentru metoda ce calculează ruta dintre două puncte prin intermediul algoritmului Dijkstra. Acest test este surprins în figura 5.4.

```
@Test
public void dijkstraValidCase(){

    // Set posible final points
    List<Node> finalPoints = new ArrayList<>();
    finalPoints.add(DijkstraAlgo.nodes.get(1));
    finalPoints.add(DijkstraAlgo.nodes.get(3));

    // In this point the distance between the current node and all the other nodes will be calculated
    DijkstraAlgo.computePaths(DijkstraAlgo.nodes.get(0));

    // Next line will send the list of possible final points as a parameter for the function which
    // will find the closest final point from a the list and will compute the path to it
    List<Node> listResultPath = DijkstraAlgo.getShortestPathTo(finalPoints);

    // AAA AAF AAB will be expected
    List<Node> expectedNode = new ArrayList<>();
    expectedNode.add(DijkstraAlgo.nodes.get(0));
    expectedNode.add(DijkstraAlgo.nodes.get(5));
    expectedNode.add(DijkstraAlgo.nodes.get(1));

    // Test if compute path is the same as expected list
    assertTrue(listResultPath.equals(expectedNode));
}
```

Fig. 5.4. Test algoritm Dijkstra

Metoda de testare aleasă este de tip Black Box. Datele care au fost folosite pentru testarea fac parte din clasa de echivalența „Valid”, deoarece graful este o componentă conexă, iar algoritmul parcurge fiecare nod. Pentru a avea rezultate în clasa de echivalența „Invalid”, este necesară furnizarea unei structuri de graf cu mai multe componente conexe.

Din punct de vedere al testării White Box, faptul că structura metodelor din clasa DijkstraAlgo, clasa în care este implementat algoritmul Dijkstra, este relativ simplă, multiple condition coverage coincide cu decision coverage. Prin rularea testului cu

coverage se poate observa în figura 5.5 acoperire 100% la nivelul clasei, la nivelul metodelor, dar și la nivelul liniilor de cod a clasei DijkstraAlgo.

Element	Class, %	Method, %	Line, %
Enums	0% (0/2)	0% (0/8)	0% (0/21)
ConstantValue	0% (0/1)	100% (0/0)	100% (0/0)
DijkstraAlgo	100% (1/1)	100% (3/3)	100% (47/47)
Edge	100% (1/1)	100% (1/1)	100% (4/4)
Node	100% (1/1)	50% (3/6)	62% (10/16)
Pair	100% (1/1)	20% (1/5)	36% (4/11)
Result	0% (0/1)	0% (0/4)	0% (0/7)

Fig. 5.5 Acoperire algoritm Dijkstra

La nivelul interfeței grafice testarea se face utilizând UI testing. Acest tip de test presupune simularea interacțiunii unui user cu sistemul, prin intermediul interfeței grafice. Există mai multe framework-uri care permit crearea de UI test. Framework-ul utilizat de mine pentru a crea acest tip de teste este Espresso.

Espresso este un framework creat de Google pentru testarea automată a interfeței aplicațiilor Android. Espresso este util doar pentru aplicații Android native. Acest framework ofera API-uri pentru a simula interacțiunea utilizatorului cu aplicația.

Testele Ui au fost făcute pentru activitatea de selectare a elementelor. Pentru această activitate sunt făcute trei teste. Primul test verifică dacă se poate trece mai departe în flow-ul aplicației dacă nu se selectează nici un element. Al doilea test verifică dacă se găsește un anumit element în listă, îl selectează apoi încearcă să treacă mai departe în flow. Al treilea test este o combinație între cele două, acesta încercând mai întâi să treacă mai departe în flow fără a selecta nici un element, apoi caută un element în listă și îl selectează. Aceste teste pot fi văzute in figura 5.6.

```java
@Test
public void checkIfCouldStartFlowWithoutSelectingAnyElement (){
    // This test checks if a user could start the flow without selecting any items
    onView(withId(R.id.start)).perform(click());
}

@Test
public void checkIfCouldStartFlowSelectingMinimumOneElement (){
    // This test searches for an item in a list, scroll to it, and press it.
    // If the item can be found, try to start the flow
    onData(is( value: "backpack")).inAdapterView(withId(R.id.listElements))
            .perform(scrollTo(), click());
    onView(withId(R.id.start)).perform(click());
}

@Test
public void checkIfCouldStartFlowFirstWithoutSelectingMinimumOneElement (){
    // This test combines the tests above
    onView(withId(R.id.start)).perform(click());
    onView(withText("OK")).perform(click());
    onData(is( value: "backpack")).inAdapterView(withId(R.id.listElements))
            .perform(scrollTo(), click());

    onView(withId(R.id.start)).perform(click());
}
```

Fig. 5.6 Teste UI

Capitolul 6

Concluzii și direcții viitoare

6.1 Principalele contribuții

In prezenta lucrare au fost relatate abordări pentru navigarea în spații interioare și pentru detecția de obiecte. Problema de la care s-a plecat a constat în timpul pierdut de oameni în căutarea produselor necesare la magazine. Abstractizând problema și generalizând-o, am constatat că soluția acesteia poate fi împărțită în navigarea în spații indoor și detectarea de obiecte.

In cadrul rutării indoor, GPS nu are o adevărată importanță datorită lipsei de semnal. Există mai multe tehnici de mapare a suprafeței și navigare, dar cea care este cea mai puțin costisitoare din punct de vedere financiar și depinde cel mai puțin de mediu este cea care folosește computer vision.

In funcție de locația spațiului unde se face rutarea și mai ales de dotările mediului, se pot adăuga și tehnologii radio sau bazate pe infraroșu pentru o mai bună poziționare în hartă. La fel ca și navigarea, detecția de obiecte poate fi îmbunătățită aplicându-se noile versiuni ale algoritmilor MobileNet SSD, YOLO sau chiar R-CNN.

Abordarea proprie din această lucrare a constat în crearea unui sistem care are datele structurate și stocate într-o bază de date de tip PostgreSQL. Această bază de date este legată de sistem prin intermediul aplicației back-end. Back-end-ul gestionează call-urile primite de la front-end și trimite informațiile necesar. Front-end-ul este reprezentat de o aplicație mobile nativă pentru Android care este alcătuita din trei subproiecte. Primul se ocupă cu selectarea unor elemente și obținerea de informații de la back-end. Al doilea

subproiect constituie partea de navigare și obținere a id-ului unei locații prin intermediul scanării unor coduri QR. Al treilea subproiect și ultimul se ocupă cu detectarea obiectelor, fiind o adaptare a librăriei TensorFlow Lite pentru detecția de obiecte.

Sistemul descris e capabil de a oferi informații optime despre punctele în care trebuie să ajungă un utilizator pentru a găsi obiectele dorite, oferind indicații textuale. Pe lângă indicații, sistemul poate recunoaște obiectele dorite cu ajutorul camerei.

Concluzionând, contribuția adusa consta în crearea unui sistem care îmbina armonios doua sarcini diferite, navigarea și detecția de obiecte, cu scopul de a rezolva problema timpului pierdut în magazine sau în orice fel de spațiu în care trebuie identificate obiecte.

6.2 Direcții viitoare

Se cunoaște faptul că mereu se pot aduce îmbunătățiri unui sistem, unei aplicații sau chiar unei funcționalități. Fie că este vorba de bug-uri nedescoperite inițial sau de noi tehnologii și metode care pot spori performanța, un sistem este nevoit să evolueze pentru a ține pasul cu noi cerințe și criterii.

Sistemul prezentat poate fi îmbunătățit atât din punct de vedere al recunoașterii de obiecte cât și al rutei oferite pentru direcționare.

In cazul scalării sistemului prezentat, recunoașterea de obiecte solicită o analiză vastă a modelelor de detecție și a ușurinței cu care se antrenează acestea. Modelul utilizat în actualul sistem este MobileNet SSD, acesta fiind recunoscut ca unul dintre cele mai bune. Pentru a ști ce model se potrivește cel mai bine cu sistemul este necesara atât testarea teoretică prin analiza performanțelor generale ale modelului cât și testarea practică prin integrarea fiecărui model cu proiectul. Acest fapt este valabil și pentru partea sistemului care se ocupa de navigare.

Pentru a ști ca o aplicație este gata sa fie folosita de utilizatori este necesara o validare si o testare riguroasa atât a funcționalităților cat si a legăturilor dintre acestea. Se recomanda ca o aplicație sa fie testata pentru a preveni eventualele Bug-uri si

vulnerabilități ale sistemului. Sistemul trebuie testat pe zone care au suprafețe mult mai mari fata de dimensiunea unui apartament (unde deja a fost testat), plecând din diferite poziții ale unei locații.

Deși sistemul creat îndeplinește cerința de a naviga în spații indoor și de a recunoaște obiecte, acesta trebuie îmbunătățit și modificat astfel încât utilizarea lui să fie una foarte intuitivă. Sistemul ar trebui să poată salva anterior listele de obiecte ce se doresc a fi identificate. Această funcționalitate este una destul de importantă deoarece un utilizator ar putea să își creeze lista de obiecte pe care dorește să le găsească anterior ajungerii lui in zona unde se identifică obiectele. Pentru ca sistemul să fie folosit de utilizatori este nevoie de o analiză a cerințelor și întocmirea diagramelor specifice care să conțină toate funcționalitățile necesare.

Bibliografie

1. „Dijkstra's Algorithm”, https://www.codingame.com/playgrounds/1608/shortest-paths-with-dijkstras-algorithm/dijkstras-algorithm, ultima accesare: 15.06.2021.
2. „Django introduction”, https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction, ultima accesare: 15.06.2021
3. „What is Artificial Intelligence (AI)?”, https://builtin.com/artificial-intelligence, ultima accesare, 17.05.2021
4. „What Is SLAM?”, site: https://www.mathworks.com/discovery/slam.html, ultima accesare: 29.03.2021
5. A. Butz, J. Baus, A. Kruger, M. Lohse. „A hybrid indoor navigation system”. articol apărut în a 6-a conferință internațională IUI, paginile 25 - 32, apariție 14-17.01.2001.
6. A. Mulloni, D. Wagner, I. Barakonyi, and D. Schmalstieg. „Indoor positioning and navigation with camera phones”, aparut in “Pervasive Computing”, volumul 8, numărul 2, paginile 22-31, anul apariției: 2009.
7. Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao, „YOLOv4: Optimal Speed and Accuracy of Object Detection”, articol aparut la data: 23.04.2020.
8. Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang Tobias Weyand, Marco Andreetto, Hartwig Adam, „MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”, data aparitiei: 17.04.2017.
9. Ben Dickson, „What are artificial neural networks (ANN)?”, site: https://bdtechtalks.com/2019/08/05/what-is-artificial-neural-network-ann/, data aparitiei: 05.08.2019, ultima accesare: 15.06.2021
10. Chisalita-Cretu Camelia, curs „Verificarea și validarea sistemelor soft”, Universitatea Babeş-Bolyai, anul 2020 - 2021.
11. Cornelia Novac Ududec, cartea „Ingineria sistemelor de programare”, Editura Alma Mater Bacau 2011.
12. David Schneider, „New Indoor Navigation Technologies Work Where GPS Can’t”, site: https://spectrum.ieee.org/telecom/wireless/new-indoor-navigation-

technologies-work-where-gps-cant, Data apariție: 20.11.2013, ultima accesare 21.03.2021

13. Dong Ngo, „Celebrating 10 years of GPS for the masses”, site: https://www.cnet.com/news/celebrating-10-years-of-gps-for-the-masses/, data apariție: 01.05.2010, ultima accesare 21.03.2021
14. H. Kawaji, K. Hatada, T. Yamasaki, si K. Aizawa, “Image-Based Indoor Positioning System: Fast Image Matching Using Omnidirectional Panoramic Images,”, articol aparut in „ACM International Workshop on Multimodal Pervasive Video Analysis”, aparitie: 01.2010
15. History of computer vision, https://www.sas.com/en_us/insights/analytics/computer-vision.html#:~:text=Computer%20vision%20is%20a%20field,to%20what%20they%20%E2%80%9Csee.%E2%80%9D , ultima accesare: 08.04.2021
16. Joseph Redmon , Ali Farhadi, „YOLO9000: Better, Faster, Stronger”, articol publicat in „2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)”, paginile: 6517 – 6525, data aparitiei: 21 – 26.07.2017
17. Joseph Redmon , Santosh Divvala, Ross Girshick , Ali Farhadi, „You Only Look Once: Unified, Real-Time Object Detection”, articol aparut in „2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)”, paginile: 779 – 788, data aparitiei: 27-30.06.2016
18. Joseph Redmon, Ali Farhadi, „YOLOv3: An Incremental Improvement”, artico aparut la data: 08.04.2018
19. M. Kranz, C. Fischer, and A. Schmidt. „A comparative study of dect and wlan signals for indoor localization”, apărut în conferinta IEEE PerCom, paginile 235 - 243, anul apariției: 2010.
20. Manfred Klopschitz si Dieter Schmalstieg, „Automatic Reconstruction of Wide-Area Fiducial Marker Models”, articol aparut in „2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality”, paginile 71 – 74, data aparitiei: 13 – 16.11.2007
21. Martin Werner, Moritz Kessel, Chadly Marouane, „Indoor Positioning Using Smartphone Camera”, articol publicat in „2011 International Conference on Indoor Positioning and Indoor Navigation”, paginile 1 – 6, data aparitiei: 21 – 23.09.2011

22. Matthijs Hollemans, „MobileNet version 2”, site: https://machinethink.net/blog/mobilenet-v2/, 22.04.2018, ultima accesare: 15.06.2021
23. N. Karlsson, E. D. Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M. E. Munich, “The VSLAM Algorithm for Robust Localization and Mapping,”, articol aparut in „International Conference on Robotics and Automation”, paginile 24 – 29, anul aparitiei: 2005.
24. O. Woodman, R. Harle. „Pedestrian localisation for indoor environments”, articol apărut în a 10-a conferință internațională UbiComp , paginile 114-123, anul apariției: 2008.
25. Object recognition, https://machinelearningknowledge.ai/popular-image-classification-models-in-imagenet-challenge-ilsvrc-competition-history/, ultima accesare: 08.04.2021.
26. Olga Russakovsky, Jia Deng, Hao Su, „ImageNet Large Scale Visual Recognition Challenge”, data aparitiei: 30.01.2015.
27. Platforma hosting, https://www.heroku.com
28. Richard A. Newcombe, „Dense Visual SLAM”, aparitie: 12.2012
29. Ross Girshick, „Fast R-CNN”, articol publicat in „2015 IEEE International Conference on Computer Vision (ICCV)”, paginile 1440 – 1448, data aparitiei: 07 – 13.12.2015.
30. Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, „Rich feature hierarchies for accurate object detection and semantic segmentation”, articol publicat in „2014 IEEE Conference on Computer Vision and Pattern Recognition”, paginile 580 – 587, data aparitiei: 23 – 28.06.2014.
31. Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, „Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, articol publicat in „IEEE Transactions on Pattern Analysis and Machine Intelligence”, paginile: 1137 – 1149, data aparitie: 16.06.2016
32. Sumit Saha, „A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way”, site: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, data aparitiei: 15.12.2018, ultima accesare: 15.06.2021
33. T. S. Huang, „Computer Vision: Evolution and Promise”, articol publicat in „CERN School of Computing”, paginile 21 – 25, anul aparitiei: 1996

34. U. Rehman, S. Cao, „Augmented Reality-Based Indoor Navigation Using Google Glass as a Wearable Head-Mounted Display”, articol publicat in „IEEE International Conference on Systems, Man, and Cybernetics”, paginile 1452 – 1457, data aparitie: 9 – 12.10.2015.
35. Umair Rehman, Shi Cao, „Augmented Reality-based Indoor Navigation: A Comparative Analysis of Handheld Devices vs. Google Glass”, articol publicat in „IEEE Transactions on Human-Machine Systems”, paginile 140 – 151, data aparitiei: 14.11.2016.
36. Wattanapong Kurdthongmee, „A comparative study of the effectiveness of using popular DNN object detection algorithms for pith detection in cross-sectional images of parawood”, articol publicat in „Heliyon”, data aparitiei: 28.02.2020
37. Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg, „SSD: Single Shot MultiBox Detector”, articol publicat in: „Computer Vision – ECCV 2016”, paginile 21 – 37, data aparitiei: 17.09. 2016
38. Zhiwei Kong, Qiang Lu, „A Brief Review of Simultaneous Localization and Mapping”, articol publicat in „IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society”, paginile 5517 – 5522, data aparitiei: 29.10 – 01.11.2017